PB81-218851

REPORT NO.
UCB/EERC-81/02
JANUARY 1981

EARTHQUAKE ENGINEERING RESEARCH CENTER

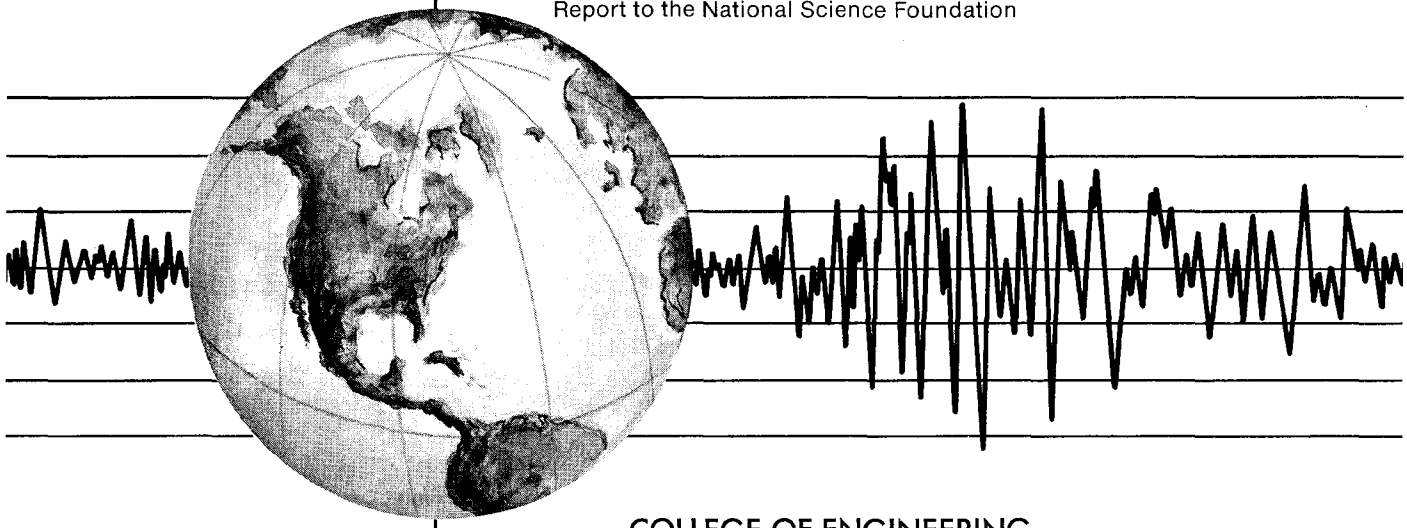
OPTNSR

AN INTERACTIVE SOFTWARE SYSTEM FOR OPTIMAL DESIGN OF STATICALLY AND DYNAMICALLY LOADED STRUCTURES WITH NONLINEAR RESPONSE

by

M.A. BHATTI
V. CIAMPI
K.S. PISTER
E. POLAK

Report to the National Science Foundation



COLLEGE OF ENGINEERING

UNIVERSITY OF CALIFORNIA · Berkeley, California

For sale by the National Technical Information Service, U.S. Department of Commerce, Springfield, Virginia 22161.

See back of report for up to date listing of EERC reports.

OPTNSR - AN INTERACTIVE SOFTWARE SYSTEM FOR OPTIMAL DESIGN OF STATICALLY AND DYNAMICALLY LOADED STRUCTURES WITH NONLINEAR RESPONSE

by

M. A. Bhatti

V. Ciampi

K. S. Pister

and

E. Polak

Prepared under the sponsorship of
the National Science Foundation
Grant PFR-7908261

Report No. UCB/EERC-81/02
Earthquake Engineering Research Center
College of Engineering
University of California
Berkeley, California

January 1981

ABSTRACT

This report describes a software system for optimization-based, interactive computer-aided design of statically and dynamically loaded structures with nonlinear response. The system combines two programs, INTEROPTDYN and MINI-ANSR. The program INTEROPTDYN is based on a feasible directions algorithm for solving a constrained optimization problem, where both non-parametric and parametric (time-dependent) constraints are allowed. Program MINI-ANSR is a modification of an existing general purpose structural analysis program, ANSR-1. Details of these programs are described, together with an interfacing package connecting the analysis and optimization phases of the design process.

The following features are available to the user: Stop and restart capability as well as user-supplied changes in both design variables as well as parameters in the optimization algorithm. Graphical display of key information is available at all stages of the design process.

Two example problems - one an elastic, statically loaded truss and the second an impulsively loaded nonlinear braced frame - are included to illustrate use of the system.

ACKNOWLEDGMENTS

This research was supported by the National Science Foundation under Grant PFR-7908261 with the University of California, Berkeley. Computing facilities were provided in part by equipment Grant ENG-7810442 from the National Science Foundation. The authors also wish to acknowledge the developers of parts of the computer software that have been modified for use in the OPTNSR System. Names of these researchers appear in the references cited at points throughout the report.

The support of Professor Ciampi by the Italian National Research Council during the course of the research is gratefully acknowledged.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 PRELIMINARY REMARKS

The common practice in design of structures is to use a trial and error design procedure. First, an initial design is chosen, which may then be analyzed using a computer program which simulates the behavior of the physical system. By looking at the results of computer simulation, the designer adjusts the initial design in an attempt to satisfy a set of given specifications which are usually not met by the initial design or to obtain a better design in terms of performance criteria. After the adjustment, a new simulation is performed and the process is repeated until a satisfactory design is obtained. The success of this procedure depends critically on the experience of the designer and may involve a considerable amount of professional-level effort.

Since the early 1950's, research in computer simulation of structural systems has made considerable progress, resulting in a number of excellent general purpose structural analysis programs [1-2]. At the same time, several attempts have been made to automate the above design process using optimization techniques. A summary of this work is contained in the survey papers [3-6]. Despite this considerable research activity, optimization techniques are not as widely used as might be expected. In the authors opinion, the main reasons for this lack of interest are:

(i) Lack of a proper definition of design problems in terms of an optimization problem.

(ii) Lack of robust optimization algorithms applicable to general design problems involving dynamic constraints.

(iii) Lack of familiarity with optimization techniques.

The definition of a design problem in terms of an optimization problem involves identifying an objective function and suitable constraint functions. Historically, since optimization techniques were first used in the aerospace industry, weight of the structure has been considered as the objective function. For design of structures subjected to dynamic loads, such as earthquake excitation, other objective functions such as life-time cost better reflect appropriate performance objectives, [7]. For some special types of structures, such as braced frames, maximizing energy absorption by the bracing system could be an objective. Thus, depending upon a particular application, any function of design parameters and/or structural response functions is a candidate for consideration as an objective. Obviously, along with different objective functions, one must define appropriate constraint functions in order that the problem is well-posed. The computer programs developed for optimal structural design, so far, have been specialized either for a particular objective function, such as minimum weight, [8], or for particular structures, e.g., trusses or shear frames. Hence, their application has been very limited. Thus, in order to look at different problem formulations, a more flexible programming structure is needed, in which users can define their own objective and constraint functions in order to widen the range of applicability to practical problems.

The optimization algorithms used up to now to solve the design problem have been too primitive for the task at hand. For example, they have not been capable of solving non-convex problems and problems with dynamic constraints. Even in simple cases, the cost-benefit ratio

has frequently been unfavorable because the algorithms failed to converge to a solution in a reasonable amount of computer time. This situation may arise because of several factors, such as: ill-conditioning of the mathematical programming problem into which the design problem is translated; weak convergence properties of the algorithms used (e.g., penalty function method with conjugate gradient method for line search); poor choice of internal parameters of algorithm; or poor initial design. Since optimization algorithms may require several structural analyses per iteration, it is clear that very slow convergence or worse, no convergence at all, may be considered as a very expensive accident!

Recently, new algorithms have been developed, for general non-convex problems involving dynamic constraints [9-10], which have better convergence properties. At the same time, methods for early detection of ill-conditioning in mathematical programming problems are emerging. Since, in general, the transcription of a design problem into a mathematical programming problem is not unique, heuristics are currently being developed which suggest ways of changing the transcription to eliminate the ill-conditioning. However, these algorithms are still sensitive to the choice of internal parameters as well as initial values of design parameters.

In order to deal with these difficulties, an interactive software system for optimal design is indispensable. Interactive computing permits one to stop, restart or modify any of the parameters as the computation progresses. This results in substantial savings, not only in computing time, but also in overall time needed to carry out a design.

An additional advantage of an interactive system using computer graphics is that it can be used as a tool to familiarize designers with optimization techniques. They can change parameters of the algorithm

and execute a few iterations while monitoring the computation closely through graphical information displays. This will give them a "feel" for these parameters and the algorithms itself, removing some of the "black box" character of the process.

In the following sections there is described an interactive software system, OPTNSR, for optimal design of structures, in which the above attributes are incorporated.

1.2 THE OPTNSR SYSTEM

The program OPTNSR is an interactive software package for OPTimal design of statically and dynamically loaded structures with Nonlinear Structural Response. The system is currently operating on a DEC VAX 11/780 computer obtained through a grant from the National Science Foundation. The operating system is a virtual memory version of UNIX (a Bell System trade mark) developed, at the University of California, Berkeley. The system consists of the following:

(i) The INTEROPTDYN program, which is a general purpose interactive optimization program capable of solving problems with or without dynamic constraints.

(ii) The MINI-ANSR program, which is a modified version of the ANSR-1 program [2]. It is capable of analyzing linear and nonlinear structural systems subjected to static and dynamic loads.

(iii) Interfacing routines between INTEROPTDYN and MINI-ANSR. These routines define a design problem in terms of an optimization problem and call for analysis of the structure as needed.

Section 2 describes some of the main features of the INTEROPTDYN system. A short description of MINI-ANSR and instructions for adding

new elements to it is contained in Section 3. Section 4 describes the interfacing routines. Since some of these routines are problem - dependent, two typical examples are discussed in detail in Section 5 to clarify the structure of these routines. Instructions for preparing input data for INTEROPTDYN and MINI-ANSR are included in the appendices.

2. THE INTEROPTDYN SYSTEM

2.1 INTRODUCTION

The program INTEROPTDYN is a general purpose, interactive optimization program capable of solving design problems which can be expressed as:

$$\begin{aligned} &\min_{\underline{z}} \quad f(\underline{z}) \\ \text{Subject to} \quad &\max_{\omega} \quad \phi^j(z, \omega) \leq 0 \qquad j = 1, \ldots, m \quad \omega \in [\omega_o, \omega_c] \\ &\qquad\quad g^j(\underline{z}) \leq 0 \qquad j = 1, \ldots, \ell \end{aligned} \tag{2.1.1}$$

where

$z \in \mathbb{R}^p$ = vector of design variables

f = objective or cost function

ϕ^j = functional or dynamic constraints

g^j = conventional inequality constraints which depend on design variables only.

The system is based on a batch program, called OPTDYN [11] which utilizes a feasible directions type algorithm to solve (2.1.1). It has been made interactive by combining it with an interpreter of an interactive language INTRAC-C, evolved from INTRAC, originally developed at the Department of Automatic Control, Lund Institute of Technology, Sweden [12].

There is some "on-line" documentation about the system. Bare essentials will be covered in the following sections to familiarize a user with the system.

2.2 A FEASIBLE DIRECTIONS ALGORITHM

A short description of the optimization algorithm is given below in order to facilitate later discussion. Details of convergence proofs are given in [10], while implementation details are given in [11].

The feasible domain F, is defined by:

$$F = \left\{ \underline{z} \in \mathbb{R}^p \;\middle|\; \begin{array}{lll} \max\limits_{\omega} \; \phi^j(\underline{z}, \omega) & \leq 0, & j = 1,\ldots, m; \\ g^j(\underline{z}) & \leq 0 & j = 1,\ldots, \ell \end{array} \right\} \tag{2.2.1}$$

Define a function ψ as follows:

$$\psi(\underline{z}) = \max \{0; \max_{\omega} \phi^j(\underline{z}, \omega), \; j = 1,\ldots, m; \; g^j(\underline{z}), \; j = 1,\ldots,\ell\} \tag{2.2.2}$$

Note that if $\underline{z} \in F$, then $\psi(\underline{z}) = 0$

Define the "ε-active constraint index" set for functional and conventional constraints as follows:

$$J_\phi = \left\{ (j, \omega) \;\middle|\; \begin{array}{l} \phi^j(\underline{z}, \omega) - \psi(\underline{z}) \geq -\varepsilon, \quad j = 1,\ldots, m \\ \text{and } \omega \text{ is a left local maximizer} \end{array} \right\} \tag{2.2.3}$$

$$J_g = \{j \mid g^i(\underline{z}) - \psi(\underline{z}) \geq -\varepsilon, \quad j = 1,\ldots, \ell\} \tag{2.2.4}$$

ALGORITHM:

DATA:

$\underline{z}_o \in \mathbb{R}^p$	:	initial design
$\varepsilon_o > 0$	:	initial value of ε for ε-active constraints
$\alpha, \beta, \delta \in (0,1)$	:	Armijo parameters for step length computation
$s_{max} > 0$	:	Parameter controlling max. step length
$\gamma \geq 1$	:	Parameter influencing search direction when infeasible
$q_o, q_{max} \geq q_o$	:	No. of points into which the interval $[\omega_o, \omega_c]$ is discretized.

$\mu_1, \mu_2 > 0$: Convergence parameters.

STEP 0: Set $i = 0$, $q = q_o$

STEP 1: Set $\varepsilon = \varepsilon_o$

STEP 2: Compute functions $f(\underline{z}_i)$, $g^j(\underline{z}_i)$ and $\phi^j(\underline{z}_i, \omega)$ for all j.

STEP 3: Direction Finding

a. Find ε-active constraints

b. Evaluate gradients of cost function and ε-active constraints.

c. Compute the optimality function $\theta(\underline{z})$, where

$$\theta(\underline{z}_i) = \max_{\mu \geq 0} \left\{ -\frac{1}{2} \left\| \mu_f \nabla \underline{f}(\underline{z}_i) + \sum_{j \in J_g} \mu_g^j \nabla \underline{g}^j(z_i) + \sum_{(j,\omega) \in J_\phi} \mu_\phi^{j,\omega} \nabla_{\underline{z}} \phi^j(\underline{z}_i, \omega) \right\|_2^2 \right.$$
$$\left. - \gamma \mu_f \psi(\underline{z}_i) \mid \mu_f + \sum_{j \in J_g} \mu_g^j + \sum_{(j,\omega) \in J_\phi} \mu_\phi^{j,\omega} = 1 \right\} \tag{2.2.5}$$

d. Using the values of μ's obtained by solving (2.2.5) compute the search direction $\underline{h}(\underline{z})$ where

$$\underline{h}(\underline{z}_i) = \mu_f \nabla \underline{f}(\underline{z}_i) + \sum_{j \in J_g} \mu_g^j \nabla \underline{g}^j(\underline{z}_i) + \sum_{(j,\omega) \in J_\phi} \mu_\phi^{j,\omega} \nabla_{\underline{z}} \phi^j(\underline{z}_i, \omega) \tag{2.2.6}$$

STEP 4. ε-Reduction

If $\theta(z_i) \leq -2\varepsilon\delta$ go to step 6.

Else set $\varepsilon = \varepsilon/2$ and proceed.

STEP 5: Mesh Refinement and Termination Criteria

If $\varepsilon > \varepsilon_o \frac{\mu_1}{q}$ or $\psi(\underline{z}_i) > \frac{\mu_2}{q}$, go to step 2.

Otherwise, set $q = 2q$ and if $q > q_{max}$ STOP,

else go to step 1.

STEP 6: Step Length Computation by Armijo Rule

Compute the largest step size $s = \beta^k \in (0, M)$

where $M = \max\left\{1, \frac{s_{max}}{||\underline{h}(\underline{z}_i)||_\infty}\right\}$ and k is an integer, such that

(i) if $z_i \notin F$, then

$$\psi\,[\underline{z}_i + s\,\underline{h}(\underline{z}_i)] - \psi(\underline{z}_i) \leq -\alpha\, s\, \delta\, \varepsilon$$

(ii) if $\underline{z}_i \in F$, then

$$f\,[\underline{z}_i + s\,\underline{h}\,(\underline{z}_i)] - f(\underline{z}_i) \leq -\alpha\, s\, \delta\, \varepsilon$$

$$g^j[\underline{z}_i + s\,\underline{h}\,(\underline{z}_i)] \leq 0 \quad j = 1,\ldots,\ell$$

$$\phi^j[\underline{z}_i + s\,\underline{h}\,(\underline{z}_i)\,,\,\omega] \leq 0 \quad j = 1,\ldots,m$$

$\omega \in [\omega_o, \omega_c]$ discretized into q points.

STEP 7: Set $\underline{z}_{i+1} = z_i + s\,\underline{h}\,(z_i)$

Set $i = i+1$ and go to step 2

REMARKS

The algorithm as presented above does not require a feasible initial design. If $\underline{z}_o \notin F$, then $\psi(\underline{z}_o) \neq 0$ and the algorithm constructs a sequence of designs with monotonically decreasing $\psi(\underline{z})$ until it becomes zero. This aspect of the algorithm is very advantageous in case of a complicated problem where choice of an initial feasible design is not obvious.

According to step 3, the search direction calculation problem turns out to be the negative of the nearest vector to the origin in the convex hull of the gradients of the cost and of the ε-active constraints. Figure 1 shows the geometry of the problem when only one

g constraint is active. From the figure it is clear that if the norms of the ε-active constraint gradients are much larger than that of the cost gradient, then the direction obtained is not very good because it will be almost perpendicular to the constraint gradient. The best way to safeguard against such a problem is to formulate the mathematical problem in such a way that the constraints and their gradients have similar magnitudes. In most cases a simple scaling of the problem is sufficient. In order to deal with cases where this is not possible, "pushfactors" are introduced in the direction-finding problem, which effectively scale the gradients. Expressions for these pushfactors, used in the present version, are given below.

For cost function: $$\rho_f = \xi_f \left(\frac{1}{||\nabla f(\underline{z})||_\infty} - 1 \right)$$

For 'g' constraints: $$\rho_g^j = \xi_g^j + \eta \left[1 + \frac{g^j(\underline{z}) - \psi(\underline{z})}{\varepsilon} \right] \quad \forall\ j$$

For 'ϕ' constraints: $$\rho_\phi^{j,\omega} = \xi_\phi^j + \eta \left[1 + \frac{\phi^j(\underline{z},\omega) - \psi(\underline{z})}{\varepsilon} \right] \quad \forall\ j$$

where ξ_f, ξ_g^j, ξ_ϕ^j and η are input parameters. For more details see [11].

2.3 INTERACTIVE IMPLEMENTATION OF THE ALGORITHM

Computational experience with batch use of the program OPTDYN revealed the following difficulties:

(i) The choice of parameters best suited for the problem at hand was not obvious and required several adjustments before reaching a set of parameters which gave good computational behavior.

(ii) Sometimes the problem was badly scaled with respect to the algorithm, requiring several adjustments before a solvable problem was obtained.

In order to cope with these difficulties the program was made interactive through a general purpose interactive language interpreter INTRAC. The interaction allows the designer:

(i) To interrupt the computing process, change parameter values and restart the process;

(ii) to control the flow of the algorithm by single-stepping through its loops (This feature is most useful in diagnosing reasons for poor computational behavior.);

(iii) to display quantities computed by the optimization and simulation algorithms;

(iv) to use the computer as a "scratch pad" for side computations on variables, vectors and matrices used in the algorithm. This feature is useful to perform tests not originally foreseen in the program and to check, for example, condition of key matrices, their eigenvalues, etc.

The first step in implementing interaction is to decide where the interaction should take place and what quantities need to be changed and/or otherwise manipulated. According to the above considerations, interaction should be implemented at each step of the main loop of the algorithm as well as at each step of every internal loop. Thus breakpoints have been inserted after the corresponding statement of OPTDYN. At each breakpoint a subroutine INTCAL is called, which checks the condition associated with the breakpoint. The

condition may be NEVER, ALWAYS or an IF clause. If it is NEVER, no action is taken and the control is returned to OPTDYN. If it is ALWAYS, INTRAC-C is called and an interaction phase takes place. The quantities which need to be changed or displayed are declared in a symbol table (data base). During the interaction phase (marked by a prompt '>'), the user has access to all these quantities and can modify them using the SET command of INTRAC-C. A list of quantities included in the symbol table, along with their FORTRAN names is given in Table 1.

In the following subsections, some of the more commonly used commands will be described. A command has the generic form:

< command identifier > < argument list >

The arguments must not contain spaces but they may be separated by an arbitrary number of spaces. If no doubts can arise, the arguments need not be separated by any space (for example, when one of the arguments is a delimiter). The following notation is used when describing the structure of the commands:

| | or (separates terms in a list from which one and only one must be chosen);

{ } groups terms together;

[ ] groups terms together and denotes that the group is optional;

< > denotes that the enclosed term is not used literally but is replaced by its appropriate value.

The commands are divided into the following categories:

(i) commands for flow control

(ii) commands to handle the symbol table

(iii) commands for graphics

(iv) commands for scratch pad

(v) miscellaneous commands

(vi) original INTRAC commands, summarized in Appendix A.

In addition to these commands, there are a number of "macros" written to make the interaction easier. A macro is a text file stored on mass storage containing a sequence of commands. The macro can be used as a new command and the sequence of the commands will be executed. Some of the more useful macros are also described in later subsections.

2.4 COMMANDS FOR FLOW CONTROL

2.4.1 BREAKS - Displays a list of break points in the algorithm.

Syntax: BREAKS

The break point names are made up of the first few letters of the subroutine followed by the statement number after which the break point occurs. It also displays the halt condition of a break point. The halt condition is either ALWAYS, NEVER or an if-clause. If the halt condition is ALWAYS the program will always stop when that break-point is reached and a prompt (>) will be given to signify its readiness for further action. If the halt condition is NEVER, the execution will go on normally and no break will occur at that break point. In the case of if-clause the condition is NEVER if the if-clause is not satisfied and ALWAYS otherwise.

A list of breakpoints names along with their initial condition and location within the algorithm is given in Table 2.

2.4.2 WHERE - Displays the name of the current breakpoint.

Syntax: WHERE

Name of the current breakpoint is displayed at which the program has stopped.

2.4.3 HALT - Sets up halt condition at a specified break point

Syntax: HALT [< breakpoint >] [< condition >]

< breakpoint >: = name of the break point at which the halt condition is to be set. Use the command BREAKS to get a list of legal names for the breakpoints. The default is the current breakpoint.

< condition >: = {ALWAYS | NEVER | < if-clause >}

< if-clause >: = IF < variable > < operator > {< variable > | < value >}

< operator >: = {< | > | =}

ALWAYS is used if a break is desired at this breakpoint, NEVER if no break is desired and the < if-clause > is used for a conditional break.

Examples:

HALT QP90 IF ITER > 5 break is set at QP90 after ITER > 5.

HALT NEVER sets current breakpoint condition to NEVER

HALT COPFE90 sets COPFE90 to ALWAYS.

2.4.4 GO - Transfers control from one breakpoint to another

Syntax: GO [< breakpoint >]

The program starts execution at the first statement following the named breakpoint. Current breakpoint is default.

Example:

GO QP90

start execution from the first statement after breakpoint QP90.

2.4.5 STOP - This command stops the execution of the program.

2.5 COMMANDS TO HANDLE SYMBOL TABLE

The variables which need to be changed during execution are stored in a symbol table (data base). The following commands can be used to manipulate these variables:

2.5.1 SYMBOL - Displays the symbol table

Syntax: SYMBOL [< variable >]

Displays the name, type and value of the specified variable in the symbol table. The default is to display all the variables in the table. If a variable is an array its dimension and the value of the first element are displayed.

Examples:

SYMBOL	displays the whole symbol table.
SYMBOL ALPHA	displays type and current value of variable ALPHA.

2.5.2 PRINT - Displays a variable from the symbol table

Syntax: PRINT < arg >

< arg >: = {< variable > | < number >}

If a variable is a 1-dimensional array, it will be displayed as a column and if it is a 2-dimensional array it will be displayed with xx columns on each line, where xx depends on the type of the variable:

integer	xx = 8
real	xx = 5
complex	xx = 2

For long arrays only the first 100 columns are printed.

Example:

PRINT Z prints the array Z as a column vector

2.5.3 SET - Changes the value of a variable in the symbol table

Syntax: SET < variable > = < arg >

< variable >: = any variable or an array element in the symbol table.

< arg >: = {< variable > | < number >}

Examples:

SET OLDSTP = 2 sets variable OLDSTP = 2.0

SET N = 3 sets variable N = 3

SET ALPHA = BETA sets variable ALPHA = BETA.

2.5.4 CHECK - Checks if a variable has been changed by SET

Syntax: CHECK {< variable > | - ANY}

Checks if a variable in the symbol table has been changed by using SET command. The result of CHECK is returned in the global variable FLAG.SET (= 0 means variable not changed, = 1 means variable has changed). Changes are measured from the last CLEAR command or the start of the program. If the argument -ANY is used the program checks for changes in all the variables and FLAG.SET is set equal to 1 if any of the variables has been changed.

Examples:

CHECK -ANY checks for all the variables for any changes.

CHECK ALPHA checks if ALPHA has been changed.

2.5.5 CLEAR - Clears flags used for CHECK

Syntax: CLEAR {< variable > | -ALL}

Clears flag used in command CHECK for the specified variable. The argument -ALL clears flags for all the variables.

Example:

CLEAR ALPHA sets FLAG.SET = 0 corresponding to variable ALPHA.

2.5.6 SETDIM - Changes actual dimension of a variable in the symbol table

Syntax: SETDIM {NCOL | NROW} (< variable >) = < arg >

NCOL changes the column dimension of the variable and NROW changes the row dimension of the variable. Note that it is only the dimension information in the symbol table that is changed and of course not the real dimensions of the FORTRAN declared array. The change will only affect commands using the dimension information, such as PRINT or LINE.

Examples:

SETDIM NCOL(Z) = 2 sets the column dimension of array Z to 2

SETDIM NROW(AQP) = N sets the row dimension of array AQP to the value of variable N

2.5.7 TRANS - Transfers value of symbol table variable to INTRAC

Syntax: TRANS < variable >

Transfers the value of a variable from the symbol table to the INTRAC. TRANS will create a global variable with the same name as the specified variable but prefixed with a '.' (dot). Note that several array elements from the same array will only create one global variable.

Example:

TRANS ALPHA transfers symbol table variable ALPHA to an INTRAC global variable .ALPHA

2.6 COMMANDS FOR GRAPHICS

To display information graphically, a number of low-level graphics commands are included in the package. These commands are used

to build commonly used macros, described later. The graphics commands work on:

1. Tektronix 4027 Color Graphics Terminal
2. Ramtek 6000 Series Color Graphics Terminal
3. HP 2648 Black & White Graphics Terminal.

2.6.1 GRINIT - Graphics initialization

Syntax: GRINIT

This command must be given before doing any graphics. The first time this command is given, the terminal type is requested.

2.6.2 DEFINE - Defines rectangular windows on the screen by a user-specified name.

Syntax: DEFINE [<xorig> <yorig> <xsize> <ysize> <name>]

The origins and sizes are given as real numbers in a coordinate system in which (0,0) is in the lower left corner and (1,1) is in the upper right corner of the largest square which can be placed, lower-left justified, on the terminal screen.

Examples:

DEFINE 0.0 0.9 0.1 0.1 WU
defines a tiny square window, called 'WU', in the upper left corner of the screen.

'DEFINE' alone prints a list of all the defined windows with their origins and sizes.

2.6.3 WINDOW - Enters a specified window

Syntax: WINDOW [< name >]

Enters a specified window so that 0.0 to 1.0 coordinates appear only in the previously defined window. 'WINDOW' alone prints out the

name of the present window. The name of the whole screen is 'SCREEN' and is the default starting window.

2.6.4 ERASE - Erases a specified window

Syntax: ERASE [< name >]

Erases a window specified by its name. 'ERASE' alone, erases the whole screen. Note that this command will erase only the contents of the graphics memory.

2.6.5 COLOR - Sets color for subsequent graphics output.

Syntax: COLOR [< color >]

< color >: = red|orange|yellow|green|blue|violet|brown|black

On the HP2648, these colors are translated into distinct dotted and dashed lines. 'COLOR' alone prints the present color.

2.6.6 VECTOR - Draws a vector between specified starting and ending coordinates.

Syntax: VECTOR <x1coor> <y1coor> <x2coor> <y2coor>

Draws a vector from (x1coor, y1coor) to (x2coor, y2coor) in the current window.

2.6.7 MOVE - Moves cursor to specified coordinate

Syntax: MOVE <xcoor> <ycoor>

Moves cursor to (xcoor, ycoor) coordinate in preparation for a DRAW.

2.6.8 DRAW - Draws a vector

Syntax: DRAW < xcoor > <ycoor >

Draws vector from previous cursor position ('MOVE' command) to (xcoor, ycoor) coordinate.

2.6.9 CURSOR - Moves cursor in preparation for text output.

Syntax: CURSOR < xcoor > < ycoor >

Moves cursor to (xcoor, ycoor) coordinate in the current window in preparation for text output using 'TEXT' command.

2.6.10 CURSOREL - Positions cursor a specified number of character size units away from (x,y) coordinate

Syntax: CURSOREL < xcoor > < ycoor > < ncx > < ncy >

< ncx >: number of character positions relative to < xcoor >

< ncy >: number of character positions relative to < ycoor >

Example:

```
CURSOREL  0.5  0.5  -3  0
```

Moves cursor to 3 characters to the left of the center of the window.

2.6.11 TEXT - Outputs text at the position of the graphics cursor.

Syntax: TEXT {[<quoted string>][<constant>][<scalar variable>]}

Outputs strings or numeric values at the position of the graphics cursor. A 'CURSOR' or 'CURSOREL' command must precede a text command.

Example:

```
TEXT  'The value of f(1:2) =' f(1:2)
```

2.7 COMMANDS FOR SCRATCHPAD

One of the most useful features of the package is that it allows the user to employ the computer as a "scratch pad" to do side calculations. In addition to the main symbol table, a separate symbol table is created for the scratch pad. The scratch pad commands can access both symbol tables but can only alter values in the scratch pad symbol table. The commands in the scratch pad are given below.

2.7.1 GETDIM - Returns actual array dimension from the symbol table.

Syntax: GETDIM <variable> = {NCOL|NROW}(<variable>)

This command is used to get actual array dimensions from the symbol table. The left hand side variable will be created in the scratch pad as an integer variable containing the requested dimension value. The variable on the right hand side can be an array in the symbol table or in the scratch pad.

Examples:

GETDIM ICOL = NCOL(AQP): Variable ICOL is created with value equal to actual number of columns in AQP.

GETDIM NN = NROW(Z): Variable NN is created with value equal to number of rows in array Z.

2.7.2 PDIM - Creates a variable in scratch pad (external symbol table)

Syntax: PD[IM] <name> [(<nrow>[: <ncol>])] <type>

< name >: = name of variable or array to be created.

< nrow >: = No. of rows

< ncol >: = No. of columns

< type >: = {I|R|D|C}; type of variable
I - integer, R - real, D - double, C - complex.

PD or PDIM is the only command that creates an array in the scratch pad. All entries in the array are initialized to zero.

Examples:

PDIM X(3) R Creates a real vector 'X' with dimension 3.

PD AR(ITER: 5) I: Creates an integer matrix 'AR' with 'ITER' rows and 5 columns. 'ITER' must be a variable in the scratch pad or symbol table.

2.7.3 PREM - Removes a variable from the scratch pad

Syntax: PR[EM] < variable >

The specified variable is removed from the scratch pad.

Example:

PR X : removes X from the scratch pad.

2.7.4 PTAB - Displays external symbol table.

Syntax: PT[AB] [< variable >]

Displays the name, type and value of the specified variable in the scratch pad. The default is to display all the variables in the table. If a variable is an array, its dimension and the value of the first element are displayed also.

Examples:

PT : displays the whole scratch pad table

PT X : displays type and current value of variable X.

2.7.5 PSCAL - Scalar operations in the scratch pad

Syntax: PS[CAL] < variable > = < expression >

< expression >: = {[[<arg1>] <op>] <arg2>|<func>(<arg1> [<arg2>])}

< OP > : = { + | - | * | / }
addition sub. mult. division

< func > : = {MAX|MIN|SIN|COS|TAN|ARCSIN|ARCCOS|ATAN|
ATAN2|PWR|AINT|CMPLX|REAL|AIMAG|
ANGLE|CONJ|ABS|SQRT|EXP|ALOG|ALOG10}

If < variable > is a name without indices, a new variable will be created in the scratch pad, but if it is an array element, the array must exist in the scratch pad (created by PDIM command). All computations are performed in double precision or complex arithmetic.

Examples = -

```
PS  X     =  Z(1) + 3.5

PSCAL  FF =  ALOG10(B)

PS  PP(I) =  Z(J) * Z(K)

PS  CC    =  CMPLX (1.5  -2.0)

PS  ASQR  =  PWR (A  2)
```

2.7.6 PMAT - Matrix operations in the scratch pad.

Syntax: PM[AT]<variable> = {<variable>|<number>}<OP><variable>

or

PM[AT]<variable>[<variable2>] = <func>(<variable>)

< OP > : = { * | ^ | + | - }
mult. scalar mult. addition sub.

< func >: = { INV | TRANS | EIGEN | TRACE | DET }

Examples:

PM AT = TRANS(A): Defines 'AT' as transpose of 'A'

PM C = B - A : Substracts matrix 'A' from 'B' and stores results in 'C'

All the arrays must be created by using a PDIM command, before using PMAT command.

2.8 MISCELLANEOUS COMMANDS

The following general utility commands are included in the package.

2.8.1 ALGO - Displays the solution algorithm

Syntax: ALGO [<step number>]

This command without the argument displays, in a condensed form, the solution algorithm. If more information is desired for a particular step, then that step number should be given as the argument.

Example:

ALGO STEP 6 Prints details of step 6 of the algorithm.

2.8.2 HELP - Explains usage of the commands

Syntax: HELP [<command>]

This command without any argument lists all the available commands with a short description of their functions. If more information on a specific command is desired, that command can be used as an argument for the HELP command.

Example:

HELP HALT - Gives syntax of HALT command.

2.8.3 ED - Calls a text editor

Syntax: ED < macro name >

In order to write and modify macros during execution, a text editor can be called using the command 'ED'. This editor is a subset of UNIX editor 'EX'. A summary of commands is included in Appendix B.

2.8.4 LIST - Lists a macro file

Syntax: LIST [LP] < macro name >

Lists a macro file on a terminal. If LP is specified, then the file is written on a file 'FORT.8' which can be sent to the line printer using the CSH command, described later.

Example:

LIST FILE1 - Lists 'FILE1' on the terminal

2.8.5 COPY - Copies a macro file

Syntax: COPY < file 1 > < file 2 >

This command creates 'file 2' with the same contents as 'file 1'.

2.8.6 DELETE - Deletes a macro file

Syntax: DELETE < macro file >

This command deletes a macro file from the mass storage.

2.8.7 CSH - Calls shell to execute a UNIX command

Syntax: CSH 'any UNIX command within quotes'

or

CSH

Command 1

Command 2

.

.

.

XX

This command makes it possible to call the shell and execute any unix command from the package.

2.9 MACROS FOR ROUTINE USAGE

The commands given in Sections 2.3 - 2.8, in combination with INTRAC commands, are used as basic building blocks to write macros that perform specified tasks. This section presents some of the macros which are very useful for the routine usage of the optimization algorithm. These macros provide the following features:

1. Simple problems, or problems with which considerable experience has been acquired, require very little interaction since most of the parameters can be preset. In this case a macro, called RUN, can be used to perform a specified number of iterations just as in batch mode.

2. Complicated problems sometimes require that the computational behavior be monitored in more detail. A series of macros is

written so that a user can essentially single step through the algorithm and change any of the parameters as desired.

3. Macros which make the use of graphics and scratch pad easier.

In addition to these ready-made macros, users can write their own macros to perform specified tasks. Some of the more commonly used macros will be described here.

2.9.1 RUN - Performs a specified number of iterations of the overall algorithm.

Syntax: RUN < nitn > [< option >]

< nitn > : = number of iterations of the algorithm to be performed. The program will stop for further action, if the number of iterations exceeds 'nitn' and the optimum has not yet been achieved. The program can be restarted by using RUN macro again, if desired.

< option >: = {STORE | PRTALL}

If PRTALL option is specified, then the program prints iteration number, cost function, θ, ε and ψ on the terminal as the computation is progressing. With the 'STORE' option, cost function, f, ψ and design variables are stored in scratch pad in arrays 'FG', 'PSIG' and 'ZG'. These arrays can be later used to plot, for example, the decrease in the cost versus iteration number, history of a particular variable over several iteration, etc. If no option is specified, only the iteration number is printed.

2.9.2 STEP2 - Computes objective and constraint functions

Syntax: STEP2

This macro performs Step 2 of the algorithms, i.e., it computes the objective function f, simple inequality constraints g and functional constraints ϕ.

2.9.3 STEP3 - Computes a usable feasible direction

Syntax: STEP3

This macro performs calculations in the step 3 of the algorithm to find a usable feasible direction. Angles between the direction vector and function gradients can be displayed by using macro PRTANG. If these angles are not satisfactory, the push factors can be changed and a new direction computed. This macro also performs tests in step 4, and step 5 of the algorithm to see where the program is going to branch next.

2.9.4 ARMIJO - Performs step length calculations using Armijo's rule.

Syntax: ARMIJO < nitn > [< display >]

< nitn > : = maximum number of iterations to be performed

< display >: = display option.

This macro performs iterations within step 6, until either, the Armijo rule is satisfied or the number of iterations exceeds the maximum specified. For the display option, macros 'GRAPHO' or 'GRAPHOS' can be used. Both of these macros plot Armijo step and simple constraints as bar charts and functional constraints at each iteration within the loop. The only difference between the two is that GRAPHOS stores values of f, ψ and z in arrays FG, PSIG and ZG created by using RUN macro with STORE option. A typical plot generated by GRAPHO is shown in Figure 2. The graphics screen is divided into three windows. In the top window, a line corresponding to the current step length being tried, is drawn. The line is below the diagonal line

if the cost reduction is satisfactory (i.e. $\Delta f \leq -\alpha s \delta \varepsilon$), but is above the diagonal otherwise. In the middle window bars are drawn corresponding to g constraints and the maximum value of ϕ constraints. Bars at successive interations are drawn a little to the right of the previous bars. The ε line is also shown. The bottom window is divided equally into several portions to accomodate all the functional constraints. The functional constraints are plotted at each iteration in their respective portions of the window.

These graphs gives a clear picture of what is going on within the Armijo loop. It is easy to identify a particular constraint that is causing difficulties in satisfying the Armijo rule. To correct this situation, a new direction can be computed with that particular constraint in the ε-active set or the problem may be rescaled.

2.9.5 RARMIJO - Performs iterations of the overall algorithm with Armijo display

Syntax: RARMIJO < nitn >

This macro combines RUN macro with the Armijo display GRAPHO. One iteration of the overall algorithm will be performed with the GRAPHO display in the step length loop. RESUME command is given to start the next iteration, as long as the number of iterations is less than < nitn >.

Note: In parallel with this macro, there is another macro called 'RARMIJOS' which combines RUN with GRAPHOS for storing values in global arrays, as explained in 'ARMIJO'.

2.9.6 GRAPHF - Plots cost function versus iteration number

Syntax: GRAPHF < yesno >

< yesno >: = { Y | N }

Plots cost function versus iteration number from the values of F stored in array FG. If < yesno > is 'Y', the curve will be marked to signify a new iteration.

A plot created by GRAPHF is shown in Figure 3(a).

2.9.7 GRAPHPSI - Plots ψ function versus iteration number.

Syntax: GRAPHPSI < yesno >

< yesno >: = { Y | N }

Plots ψ function versus iteration number from the values stored in PSIG. < yesno > has the same meaning as in GRAPHF. A plot created by this macro is shown in Figure 3(b).

2.9.8 GRAPHZ - Plots history of design variables

Syntax: GRAPHZ < number > [< yesno >]

Plots a particular design variable, specified by < number > versus the iteration number from the array ZG. < yesno > has the usual meaning.

2.9.9 HELP MACROS

There are many other macros which are written to make the use of the commands easier. They are grouped into macros for graphics and macros for scratch pad. A list of these macros and their syntax can be obtained by using the following help macros

HLPGR - Gives a list and syntax of macros for graphics

HLPPAD - Gives a list and syntax of macros which facilitate use of scratch pad.

TABLE 1

SYMBOL TABLE FOR INTEROPTDYN

Generic Name	FORTRAN Variable	Type	Description
f	F	double precision	Cost or objective function.
g	G	double precision	Array containing conventional inequality constraints.
ϕ	PHI	double precision	Vector containing functional constraints. The ith row contains the ith functional constraint at specified intervals.
Z	Z	double precision	Vector of design variables.
ψ	PSI	double precision	Function ψ.
	N	Integer	Number of design variables.
	JP	Integer	Number of simple inequality constraints.
	JQ	Integer	Number of functional inequality constraints.
ω_o	WO	double precision	Initial value of the interval $[\omega_o,\omega_c]$ for which ϕ constraints are defined.
ω_c	WC	double precision	Final value of the interval $[\omega_o,\omega_c]$.
	Q	Integer	Number of steps into which the interval $[\omega_o,\omega_c]$ has been divided.
q_{max}	QMAX	Integer	Maximum number of steps into which the interval $[\omega_o,\omega_c]$ is to be divided.
	MAXITN	Integer	Maximum number of iterations of the overall algorithm allowed.
	ITER	Integer	Current iteration number.

Generic Name	FORTRAN Variable	Type	Description
	NCUT	Integer	Maximum number of simplex iterations allowed in solving the quadratic programming problem for direction finding.
	ITRSTP	Integer	Maximum number of iterations allowed in step length calculations.
μ_1	MU1	double precision	Convergence parameter.
μ_2	MU2	double precision	Convergence parameter.
γ	GAMMA	double precision	Parameter influencing search direction when infeasible.
ε	E	double precision	Smear parameter ε.
δ	DELTA	double precision	Parameter δ used in convergence check and step length calculations.
α	ALPHA	double precision	Parameter α in step length calculation rule.
β	BETA	double precision	Parameter β in step length calculation rule.
s_{max}	SMAX	double precision	Parameter controlling maximum step length.
η	SCALE	double precision	Parameter for computing push factors.
ξ_f	PUSHF	double precision	Push factor parameter for cost function.
ξ_g	PUSHG	double precision	Vector of push factor parameter for 'g' constraints.
ξ_ϕ	PUSHPH	double precision	Vector of push factor parameters for 'ϕ' constraints.
h	H	double precision	Search direction vector.
	AGRAD	double precision	Matrix containing gradients of cost function and ε-active constraints. First row always contains cost gradient.

Generic Name	FORTRAN Variable	Type	Description
	AQP	double precision	Matrix of scaled gradients used for direction finding.
	ENORM	double precision	Vector containing row norm of AGRAD matrix.
	ATHETA	double precision	Vector containing angles between the direction vector and cost gradient and ε-active contraint gradients.
μ	MUBAR	double precision	Optimal values of μ's in direction finding process.
θ	THETA	double precision	Optimality function, θ.
	NEPTG	Integer	Vector containing 1 or 0 at the ith location depending upon whether ith 'g' constraint is active or not.
	NEPTF	integer	Matrix containing mesh point numbers of ε-active local maxima for functional constraints.
	S	double precision	Current step length being tried in Armijo.
	OLDSTP	double precision	Step length at the last iteration.
	ZNEW	double precision	Vector of new design variables corresponding to the current step length being tried.
	FNEW	double precision	New cost function, corresponding to ZNEW.
	TOL	double precision	Tolerance on minimum step length.

TABLE 2

EXPLANATION OF BREAKPOINTS IN INTEROPTDYN

Name	Initial Condition	Description
COPFE90	never	This breakpoint is at the beginning of step 1 of the algorithm. The program has read all the data from the input file and initialization has been completed.
COPFE110	never	This breakpoint is at the beginning of step 2.
COPFE150	never	This breakpoint is located just after calls to FUNCF, FUNG and FUNCPH, for evaluating f, g, ϕ and ψ functions, i.e., at the end of step 2.
QP70	never	This breakpoint is located just before step 3a of the algorithm.
QP90	always	This breakpoint occurs at the end of step 3a, after determining ε-active constraints. The program always stops at this breakpoint when it is first started.
EACTI250	always	The dimensions of ε-active arrays are controlled by a variable NACTIV. In the present version, this variable, as well as dimensions of arrays, is set to 10. If there are more constraints active, control is transferred to this breakpoint. Two possible ways of proceeding then occur. The first is to stop execution and increase the dimensions in the FORTRAN program, as explained in the listing of the program in [11]. A second possibility is to reduce ε, so that some of the constraints are dropped, thereby reducing the dimensionality. After changing ε- the control should be transferred to QP70.
QP205	never	This breakpoint occurs after the program has computed a new search direction and corresponding optimality function. It has also computed the angles between the direction vector and cost and ε-active constraint gradients by this time. At this point these angles can be examined and a decision made as to whether to compute a new direction by changing some parameters or to go ahead and compute a step length in the computed search direction.

Name	Initial Condition	Description
QP220	always	If the quadratic programming problem for direction finding is not solved properly, the control is transferred to this breakpoint. The user can then examine different variables and may use the scratch pad facility to determine the cause of this phenomenon.
COPFE155	never	This breakpoint is at the end of step 3 after returning from the direction finding routine QP.
COPFE165	always	This occurs at the end of step 5. The optimal solution has been achieved.
ARMJJ100	never	This breakpoint is at the beginning of step length calculations (step 6).
ARMIJ155	never	At the beginning of main loop of Armijo.
ARMIJ180	always	If the number of iterations within Armijo exceeds ITRSTP, the control is transferred to this breakpoint. ITRSTP should be increased and control transferred to ARMIJ100.
ARMIJ190	always	If the step length is smaller than a certain tolerance TOL, the program stops at this breakpoint. This is to warn the user that the computation might be jamming up. A closer inspection of the computation should be made if the process is to be continued.
COPFE180	never	At the end of the iteration of the overall algorithm (step 7).
COPFE190	always	If the number of iterations exceeds MAXITN, the control is transferred to this breakpoint. MAXITN should be increased to continue the process.

3. THE MINI-ANSR SYSTEM

3.1 INTRODUCTION

Since the optimization algorithm requires many simulations of response of the structure, an efficient structural analysis program is indispensable. Moreover, it should be based on general structural analysis concepts employing the finite element method in order that it may be applied to a wide class of problems. A general purpose structural analysis program - ANSR-1 developed by Mondkar and Powell [2] was selected as the best available program, combining broad scope and large capacity with computational efficiency. The program structure has been designed to satisfy the following requirements:

(a) Modularity The program should be modular so that new program capabilities, such as new elements, new constitutive laws, etc. can be added by developing a few subroutines, without changes to the existing program. This has been achieved by structuring the program as a base program to which a number of auxiliary programs, defining particular finite elements, can be added. Storage allocation and computations common to all finite elements are performed within the base program, while computations associated with specific elements are carried out within the auxiliary programs.

(b) Computational Efficiency The program should incorporate efficient computational algorithms, including efficient equation solvers, stress computation algorithms etc.

(c) Solution Strategy The program should include a flexible solution strategy so that a wide range of nonlinear structural systems can be analyzed. Flexibility has been achieved by implementing a strategy

defined in terms of a number of solution parameters. By assigning different values to these parameters, a wide variety of solution schemes can be implemented.

The program MINI-ANSR is a version of ANSR-1, modified for mini computers with virtual memory operating systems. The major modifications are in storage allocation and in use of core. Two separate common blocks for real and integer data are created, to which the storage is allocated dynamically. Advantage is taken of the virtual memory operating system to perform disc operations more efficiently. In the program it is assumed that there is enough storage available for the analysis and the operating system calls for writing and reading of blocks of information to and from the disc.

In the following sections some of the main features of the program are described and instructions for writing new elements are given.

3.2 PROGRAM FEATURES AND LIMITATIONS

3.2.1 Structural Idealization

(a) The structure is idealized as an assemblage of discrete finite elements connected at nodes. Each node may possess up to six displacement degrees of freedom. Provision is made for degrees of freedom to be deleted or combined. This feature provides the user with ample flexibility in the idealization of the structure, and may permit the size of the problem to be substantially reduced.

(b) The mass of the structure is assumed to be lumped at the nodes, so that the mass matrix is diagonal.

(c) Viscous damping effects may be included, if desired. Damping effects proportional to mass, initial elastic stiffness and/or tangent stiffness can be specified.

3.2.2 Static and Dynamic Loadings

(a) Loads are assumed to be applied only at the nodes. Static and/or dynamic loads may be specified; however, static loads, if any, must be applied prior to the dynamic loads.

(b) For static analysis, a number of static force patterns must be specified. Static loads are then applied in a series of load increments, each load increment being specified as a linear combination of the static force patterns. This feature permits nonproportional loads to be applied. Each load increment can be specified to be applied in a number of equal steps.

(c) The dynamic loading may consist of earthquake ground accelerations, time dependent nodal loads, and prescribed initial values of the nodal velocities and accelerations. These dynamic loadings can be specified to act singly or in combination.

3.2.3 Solution Procedure

(a) The program incorporates a solution strategy defined in terms of a number of control parameters. By assigning appropriate values to these parameters, a wide variety of solution schemes including step-by-step, iterative and mixed schemes, may be implemented.

(b) For static analysis, a different solution scheme may be employed for each load increment. The use of this feature can reduce the solution time for structures in which the response must be computed more precisely for certain ranges of loading than for others. In such

cases, a sophisticated solution scheme with equilibrium iteration might be used for the critical ranges of loading, whereas a simpler step-by-step scheme without iteration might suffice for other loading ranges.

(c) The dynamic response is computed by step-by-step integration of incremental equations of motion using Newmark's method. A variety of integration operators may be obtained by assigning appropriate values to the parameters β and γ.

3.2.4 Other Features

(a) The stiffness matrix of the structure is stored column - wise in a compacted form omitting most zero elements to save storage.

(b) The stiffness matrix is modified, rather than completely reformed, as the tangent stiffness changes. During solution, the decomposition is carried out only on that part of the updated stiffness matrix which follows the first modified coefficient. Significant savings in solution time can sometimes be obtained by numbering the nodes connecting nonlinear elements to be last, so that the decomposition operations are limited to the end of the matrix.

(c) Data checking runs may be made prior to execution runs. During data checking, the program reads and prints all input data, but performs no substantial analysis.

(d) Nonlinearities are introduced at the element level only, and may be due to large displacements, large strains and/or nonlinear materials. The programmer adding a new element may include any type or degree of nonlinearity in the behavior of the element.

3.3 FINITE ELEMENT LIBRARY

At present, the following finite elements are included in the program. New finite elements may be added to the library with relative ease by following the instructions given in the next section.

3.3.1 Three-Dimensional Elastic Truss Element

This element can be located arbitrarily in an X, Y, Z cartesian coordinate system. It can transmit axial forces only.

3.3.2 Three-Dimensional Nonlinear Truss Element

This element may yield in tension and yield or buckle elastically in compression. Large displacement effects may be included. See [2] for theoretical details of this element.

3.3.3 Two-Dimensional Elastic Beam Element

Two-dimensional elastic beam elements can be located arbitrarily in an X, Y cartesian coordinate system. Shear deformations are ignored.

3.3.4 Two-Dimensional Nonlinear Beam Element

This element may be arbitrarily oriented in the global X Y Z reference frame. Each element must be assigned an axial stiffness plus a major axis flexural stiffness. Torsional and minor axis flexural stiffnesses may also be specified if necessary. Flexural shear deformations and the effects of eccentric end connections can be taken into account. Yielding may take place only at concentrated plastic hinges at the element ends. Hinge formation is affected by the axial force and major axis bending moment only. Strain hardening and large displacement effects can be approximated. See [13] for theoretical details of this element.

3.3.5 Three-Dimensional Nonlinear Beam Element

This element may be arbitrarily oriented in the global X Y Z reference frame. Each element must be assigned flexural stiffness and axial stiffness. Plastic hinges can form at the element ends. Interaction among the bending moments, torsional moment and axial force is taken into consideration. Displacements are assumed to be small, although the P-delta effect may be considered. Theoretical details are given in [13].

3.4 ADDITION OF ELEMENTS TO PROGRAM

The computer program is organized so as to facilitate addition of new elements to the existing element library of the program. For this purpose, the program is divided into two parts, namely, (1) the base program consisting of a series of subroutines performing specific tasks required for static and dynamic analysis, and (2) a number of auxiliary programs, each program consisting of a package of subroutines required for a specific type of finite element in the element library. The user wishing to add a new element to the library is mainly concerned with the structure and organization of the auxiliary program, which will be described in the subsequent sections. The organization of the base program will not be described in this report; however, sufficient details will be given to provide an understanding of the linkage and information transmittal between the base program and the auxiliary program.

3.4.1 Transmittal of Information

During input, the elements are arranged into groups, such that all elements in any group are of the same type. Depending on the type of element, the base program refers to the package of subroutines of

the auxiliary program, at various phases of the computation. Information is transmitted to or returned from the subroutines of the auxiliary program through the argument lists and through labelled COMMON blocks.

For each element, two blocks of information are created, one for real variables and the other for integer variables. These are continuously updated during execution. All information to be retained for any element must be contained within these blocks.

The base program transfers the element information to a subroutine in the auxiliary program through the arrays COMS and ICOMS. The addresses assigned to these arrays in the base program correspond to the first words of information, in real and integer blocks, for the corresponding element. To transfer the data from the arrays COMS and ICOMS to the element information blocks, the following FORTRAN statements must appear at the beginning of each auxiliary subroutine.

```
      COMMON/INFELI/IMEM,...
      COMMON/INFELR/RDATA(1)
      DIMENSION COMS(1), ICOMS(1), COM(1), ICOM(1)
      EQUIVALENCE (IMEM,ICOMS(1)), (RDATA(1),COMS(1)).
      DO  100  J = 1, NINFCI
100   ICOM(J)  =  ICOMS(J)
      DO  110  J  =  1, NINFCR
110   COM(J)  =  COMS(J)
```

in which NINFCI = Number of words in the common block INFELI for that element

NINFCR = Number of words in the common block INFELR for that element.

The contents of common blocks INFELI and INFELR will be described subsequently.

The data within the blocks INFELI and INFELR will usually be updated during computations in the subroutine, so that it is necessary to transmit the updated data back to the arrays COMS and ICOMS at the end of the subroutine. This is achieved through the following FORTRAN statements.

```
      DO  200  J = 1, NINFCI
200   ICOMS(J)  =  ICOM(J)
      DO  210  J = 1, NINFCR
210   COMS(J)  =  COM(J)
```

It may be noted that in most cases only a part of the data is updated. Hence, it may be more efficient to transfer the modified data selectively. However, it can be expected that the computer time required to transfer data from arrays COMS and ICOMS to the blocks INFELR, and INFELI and vice versa, will be a small proportion of the total execution time.

3.4.2 Labelled Common Blocks

(a) COMMON Blocks

The labelled COMMON blocks used in subroutines of the auxiliary program are as follows.

(a) COMMON/TAPES/NIU, NOU, NT1, NT2, NT3, NT4, NT5, NTEMP

(b) COMMON/INFELI/IMEM, KST, LM(...),...

(c) COMMON/INFELR/RDAT(1)

(d) COMMON/WORK/WORK (2000)

(b) Input/Output Unit Block (/TAPES/)

This block contains disc file units assigned by the base program. These should not be changed in any of the subroutines of the auxiliary program. NIU is the input unit to read data and NOU is the output unit to print data. Other units are not used in the present version.

(c) Element Information Block (/INFELI/)

This block contains all integer data to be retained for any element. The data can be arranged by the programmer in any desired order except for the following restrictions:

(1) The first word of the block must be the element number. The variable name IMEM is suggested.

(2) The second word must be the stiffness update code, as explained subsequently. Variable name KST is suggested.

(3) The third word must be the first word of the element location matrix. The suggested variable name is LM. The length of the vector LM equals the number of degrees of freedom of the element.

The remaining data of the block can be arranged in any order. These data will typically consist of the output history code, code for including geometric effects, etc.

(d) Element Information Block (/INFELR/)

This block contains all real or double precision data to be retained for any element. These data can be arranged by the programmer in any desired order. Such data will typically consist of element material properties, nodal coordinates, strain-displacement transformation matrices, current stiffness matrix, strains and stresses at integration points, envelope values of stresses and strains, plastic strains, etc.

(e) Work Block (/WORK/)

This block provides a core area for use by the programmer. The work area provided by this block can be used for storage and manipulation of data during execution of any subroutine in the auxiliary program. Because this area is also used for temporary data storage by subroutines in the base program, it must not be used to transfer data between auxiliary subroutines.

3.4.3 Auxiliary Program

(a) General

Each auxiliary program consists of a package of subroutines required for a specific type of finite element. Each program consists of four main subroutines, as follows:

(a) INEL: Input and initialization of element information.

(b) STIF: Formation of element tangent stiffness in static analysis, or of element effective stiffness in dynamic analysis.

(c) RESP: Computation of element deformations (strains) and actions (stresses); determination of yield status; updating of element information; computation of equivalent nodal loads in equilibrium with the current state of stress; computation of equivalent damping loads; and printing of strain and stress results. As will be explained subsequently, control is exercised by the base program to perform selectively any one or a combination of the above operations.

(d) OUT: Output of envelope values of element deformations (strains) and actions (stresses) at specified load increments in static analysis or at specified time intervals in dynamic analysis.

Each of these four routines must be identified by a number designating the element type, suffixed to the subroutine name. For example, the names of subroutines for the element type 1 must be INEL1, STIF1, RESP1 and OUT1. The programmer can also write, if needed,

additional secondary subroutines which are referenced by any one of the four main subroutines. At the end of such a subroutine control will be returned to a main subroutine, whereas at the end of a main subroutine control will be returned to the base program. Information may be transferred to and from secondary subroutines through argument lists, through the WORK common block, or through other labelled COMMON blocks created specifically for such information transfer.

Explanations of the tasks performed by each of the main subroutines, and the meanings of the variables of the argument lists, are given in the following sections.

(b) Subroutine INEL

This subroutine is referenced by the base program once for each group of elements of the corresponding element type. For example, subroutine INEL1 will be called once for each group of elements containing elements of type 1.

The purpose of the subroutine is to read the input data for all elements in the group, and to initialize the variables in the element information blocks INFELI and INFELR.

The subroutine requires labelled COMMON blocks TAPES, INFELI and INFELR. The labelled COMMON block WORK may be used if desired. The argument list is as follows.

LPAR: A vector of dimension 10, which upon entry contains up to 10 control parameters for each element group.

FLPAR: A vector of dimension 6 which upon entry contains up to 6 control parameters for each element group.

NDOF: Number of element degrees of freedom.

NINFCR: Number of real words of information stored for each element in the element group. This number equals the length of the labelled COMMON block INFELR for elements of the type being considered.

NINFCI: Number of integer words of information stored for each element in the element group. This number equals the length of the labelled COMMON block INFELI for elements of the type being considered.

NJT: Total number of nodes in the structure. This value is assigned by the base program.

NDKOD: An array of dimension (NJT x 6), which upon entry contains the numbers of the structure degrees of freedom. That is, NDKOD (I,1) thru NKDOD (I,6) contain the numbers of the structure degrees of freedom corresponding to the X displacement, Y displacement, Z displacement, X rotation, Y rotation and Z rotation, respectively, at node I. These values are generated by the base program, and must not be changed in the auxiliary program.

X,Y,Z: Vectors of dimension NJT each, which upon entry contain nodal coordinates. That is X(I), Y(I) and Z(I) contain the X, Y and Z coordinates, respectively of node I. These values are generated by the base program, and must not be changed in the auxiliary program.

The title of the subroutine, for example for element type 1, must be as follows:

```
SUBROUTINE INEL1 (LPAR,FLPAR,NDOF,NINFCR,NINFCI,NDKOD,X,Y,Z,NJT)
```

The values of the control parameters in vectors LPAR and FLPAR are established within the base program by reading the first data card of each element group using a (10I5, 6F5.0) format. The first three control parameters in LPAR and the first two control parameters in FLPAR are stored by the base program as control parameters for the element group, and are used subsequently. These parameters must be as follows:

LPAR(1): A number identifying the type of element in the group. For example, if 4 is entered, the subroutines called for this group will be INEL4, STIF4, RESP4, and OUT4. Presently, this parameter can be assigned values 1 through 10.

LPAR(2): Number of elements in the group.

LPAR(3): Element number of the first element in the group.

FLPAR(1): Initial stiffness damping factor β_o.

FLPAR(2): Current tangent stiffness damping factor β_T.

All other words in LPAR and FLPAR can be assigned values, as needed, by the programmer.

All subsequent data for the elements are read within the subroutine INEL, with the sequence and input formats to be decided by the programmer.

The following steps must be performed within the subroutine:

(a) Set the values of the variables NDOF, NINFCR and NINFCI.

(b) If desired, establish reference tables of material properties, fixed end forces, initial stresses etc. for later use in specifying properties for each element. The WORK block may be used to store these tables temporarily.

(c) Specify properties of each element in the group. This data will typically consist of node numbers, material properties, the initial state of stress, an indicator for inclusion of large displacement effects, etc. Any reference tables established in (b) may be used. Generation options may be incorporated, provided the elements are generated in element number sequence and information for only one element at a time is stored in the COMMON blocks INFELR and INFELI, as appropriate.

(d) For each element, the following initialization operations must be performed.
(1) Set up the element location matrix, LM, within the COMMON block INFELI. This can be done with reference to the numbers of the structure degrees of freedom contained in the array NDKOD, and the element node numbers.
(2) Set IMEM to the element number within the group. Set the stiffness update code KST to one (KST = 1).
(3) Set any status indicators established within the COMMON block INFELI to appropriate values. Such indicators will typically be used to indicate whether or not large displacement effects are to be considered; to monitor yield status; to control printing of stress-strain history results; etc.
(4) Compute and save, within the block INFELR, strain-displacement transformation matrices for formation of element stiffness terms and for state determination calculations to be carried out in the auxiliary routines STIF and RESP, respectively. It should be noted that the nodal coordinates X, Y, Z are not transferred by the base program to the auxiliary routines STIF and RESP. However, the programmer may retain the nodal

coordinates for the nodes to which the elements connects, as part of the INFELR block, if desired.

(5) Call subroutine BAND with the statement

CALL BAND (LM, NDOF)

This permits the base program to establish information on the profile of the structure stiffness matrix. This call must be made subsequent to the setting up of the element location matrix LM.

(6) Call subroutine COMPCT with the statement

CALL COMPCT

This transfers NINFCI words from INFELI block to the main array containing integer information, and NINFCR words from INFELR block to the blank COMMON containing real data. This call must be made after the element information in the blocks INFELI and INFELR has been fully initialized.

(c) Subroutine STIF

This subroutine is referenced by the base program each time a change in element stiffness is to be calculated, unless the solution control parameters are such that the structure stiffness from a previous step is to be retained. The subroutine is referenced in the following situations:

(a) For the first step in either a static analysis or a dynamic analysis, the subroutine is referenced by the base program once for each element. For static analysis, the load steps are numbered sequentially in decreasing order by the base program (ISTEP = 0, -1, -2,..., etc.) whereas for dynamic analysis the time steps are numbered sequentially in increasing order (ISTEP = 1, 2, 3,..., etc.). Thus, when ISTEP = 0, the subroutine is called once for each element to form the initial elastic stiffness; whereas when ISTEP = 1, it is called once for each element to form the effective stiffness matrix, which includes contributions due to the inertial and/or damping matrix terms.

(b) The static solution control parameters or the dynamic solution control parameters determine the frequency with which the subroutine will be referenced. Situations will arise when the solution control parameters specify no reference to the subroutine even when a stiffness change is indicated for one or more elements. However, these situations are dealt with in the base program.

As with the subroutine INEL1, the subroutine STIF1 will be called for elements of type 1. The purpose of the routine is to compute a change in element stiffness, and transfer this change to the base

program for subsequent assembly into the structure stiffness matrix. Because the structure stiffness matrix is not necessarily updated at every load step, time step, or iteration, the change in the element stiffness must reflect the change since the last update.

The subroutine requires the labelled COMMON blocks INFELI and INFELR. The labelled COMMON block WORK may be used if desired. The argument list is as follows:

ISTEP: Load step number, or time step number. This value is assigned by the base program.

NDOF: See INEL routine. This value is now assigned by the base program.

NINFCI: See INEL routine. This value is now assigned by the base program.

NINFCR: See INEL routine. This value is now assigned by the base program.

CDKO: Value of constant $a_4\beta_o$ to be used in computing the contribution of the damping terms to the effective stiffness matrix in dynamic analysis. This value is assigned by the base program.

CDKT: Value of constant $a_4\beta_T$ to be used in computing the contribution of the damping terms to the effective stiffness matrix in dynamic analysis. This value is assigned by the base program.

ICOMS: A vector of dimension NINFCI, which upon entry contains the integer element information. The address assigned to ICOMS in the base program corresponds to the first word of integer information for the element.

COMS: A vector of dimension NINFCR, which upon entry contains the real element information. The address assigned to COMS in the base program correspond to the first word of real information for the element.

FK: An array of dimension of at most (NDOF x NDOF), into which is to be placed the change in the element stiffness matrix since the last update. See explanation below.

INDFK: Indicator to specify the storage arrangement of the element stiffness matrix in the array FK. The programmer is required to assign a value of zero or one to INDFK in this subroutine as explained in the following:

The element stiffness matrix can be stored in the array FK either (1) as a square symmetric matrix of dimension (NDOF x NDOF) or (2) as a vector in which the columns of the lower part of the symmetric stiffness matrix are stacked together compactly. The number of words in the vector of form (2) will be NDOF x (NDOF + 1)/2. The programmer is required to assign, to INDFK, a value of zero if the element stiffness is stored as in (1), or a value of one if the element stiffness is stored as in (2). The base program uses INDFK in the assembly of the element stiffness matrix into the structure stiffness matrix.

The title of the subroutine, for example for element type 1, must be as follows.

```
SUBROUTINE STIF1 (ISTEP, NDOF, NINFCI, NINFCR, CDKO,
                  CDKT, ICOMS, COMS, FK, INDFK)
```

The following steps must be performed within the subroutine:

(a) Transfer the data from the arrays ICOMS and COMS to the element information block INFELI and INFELR. The procedure explained in Section 3.4.1 must be used.

(b) Set INDFK to zero or one, as appropriate.

(c) For static analysis (ISTEP $\leq$ 0), compute the change in the element tangent stiffness matrix. When ISTEP = 0, this change equals the initial elastic stiffness matrix. For dynamic analysis (ISTEP $\geq$ 1), compute the change in the element effective stiffness matrix. Store the change in array FK, the storage scheme depending on the value assigned to INDFK.

(d) Set the stiffness update code (KST) to zero. Update any other data in the COMMON blocks INFELI and INFELR.

(e) Transfer the information in the blocks INFELI and INFELR to the arrays ICOMS and COMS. The procedure explained in Section 3.4.1 must be used.

(d) Subroutine RESP

This subroutine is referenced by the base program for each element at each iteration within a load step in static analysis, and at each iteration within a time step in dynamic analysis.

As with the subroutine INEL1, the subroutine RESP1 will be called for elements of type 1.

The tasks to be performed in this subroutine are: (T1) compute the element deformations (strains) and actions (stresses); (T2) determine the change of status if any; (T3) compute equivalent nodal loads in equilibrium with the current state of stress; (T4) compute equivalent damping loads; (T5) accumulate envelope values of element deformations (strains) and actions (stresses); (T6) update the element information; and (T7) print the strain and stress results. As explained subsequently, the base program specifies, through the indicator KUPD, which of the above tasks should be performed at any iteration in a load step or time step.

The subroutine requires the labelled COMMON blocks TAPES, INFELI and INFELR. The labelled COMMON block WORK may be used if desired. The argument list for this routine is as follows.

NDOF: See INEL routine. This value is assigned by the base program.

NINFCI: See INEL routine. This value is assigned by the base program.

NINFCR: See INEL routine. This value is now assigned by the base program.

MFST: Element number of the first element in the group. This value is assigned by the base program, and equals the control parameter LPAR(3). See INEL routine.

KPR: Print indicator for element stress and strain results. This value is assigned by the base program. KPR is set equal to zero if the results are not to be printed, otherwise it is set equal to the element group number.

ICOMS: A vector of dimension NINFCI, which upon entry contains the integer element information. The address assigned to ICOMS in the base program corresponds to the first word of information (integer) for the element.

COMS: A vector of dimension NINFCR, which upon entry contains the real element information. The address assigned to COMS in the base program corresponds to the first word of information (real) for the element.

Q: A vector of dimension NDOF, which upon entry contains the increments in the element nodal displacements.

VEL: A vector of dimension NDOF, which upon entry contains the element nodal velocities.

ACC: A vector of dimension NDOF, which upon entry contains the element nodal accelerations.

FE: A vector of dimension NDOF, in which the nodal loads in equilibrium with the current state of stress must be returned.

FD: A vector of dimension NDOF, in which the damping loads at the element nodes must be returned.

TIME: Time, in seconds, at the current time step. This value is assigned by the base program. In static analysis, TIME = 0.0.

DKO: Initial stiffness damping factor, β_o. This value is assigned by the base program.

DKT: Tangent stiffness damping factor, β_T. This value is assigned by the base program.

C7: Value of a constant to be used in computing the contribution of damping to the effective load vector in dynamic analysis. This value is assigned by the base program.

C8: Value of constant a_6 to be used in computing the contribution of damping to the effective load vector in dynamic analysis. This value is assigned by the base program.

KUPD: An indicator controlling which task or combination of tasks is to be performed in this routine, as explained subsequently. The base program sets KUPD to a value of 1 through 4.

KITRN: An indicator specifying the form of the effective load vector in dynamic analysis. This value is assigned by the base program. See [2] for more details.

The values of MFST and KPR should be used by the programmer to print the element group number and an appropriate heading when the element stress and strain results are printed. Additionally, the programmer can print selectively the results for certain elements within the group, with the aid of appropriate indicator stored as part of the element information.

The indicator KUPD is required to be used as follows, in performing the tasks (T1) through (T7) specified earlier.

(1) KUPD = 1: Perform tasks (T1) through (T7)

(2) KUPD = 2: Perform tasks (T1) through (T4) and (T7)

(3) KUPD = 3: Perform task (T7) only

(4) KUPD = 4: Perform tasks (T3), (T4) and (T7)

The computation of damping stresses and equivalent nodal loads due to damping is to be performed in dynamic analysis only (i.e. when TIME > 0.0).

The title of the subroutine, for example for element type 1, must be as follows.

```
SUBROUTINE RESP1 (NDOF, NINFCI, NINFCR, MFST, KPR, ICOMS,
                  COMS, Q, VEL, ACC, FE, FD, TIME, DKO,
                  DKT, C7, C8, KUPD, KITRN)
```

The following steps must be performed within the subroutine

(a) Transfer the data from the arrays COMS and ICOMS to the element information blocks INFELR and INFELI. The procedure explained in Section 3.4.1 must be used.

(b) Perform the task (T1) through (T7), depending on the value of the indicator KUPD. If the element changes its status because of material yielding or unloading, set the stiffness update code (KST) to one. If large displacement effects are included for the element, KST must always be set to 1, because there will be a continuous change in the element geometry and hence in its stiffness. KST must be set prior to updating the element information in the block INFELI (i.e. prior to performing task (T6)).

(c) Transfer the information in the blocks INFELI and INFELR to the arrays ICOMS and COMS. The procedure explained in Section 3.4.1 must be used. The transfer of this information must be carried out only if KUPD = 1.

(e) Subroutine OUT

This subroutine is referenced by the base program for each element at selected static load increments and at specified time step intervals.

As with the subroutine INELl, the subroutine OUTl will be called for elements of type 1.

The purpose of this routine is to print the envelope values of stresses, strains and the corresponding times at which these maxima have occurred. The sequence and formats for printing these results are to be decided by the programmer. If the programmer decides to omit storing envelope values and corresponding times in the block INFELR, a dummy OUT subroutine must be supplied.

The subroutine requires the labelled COMMON blocks TAPES, INFELI and INFELR. The labelled COMMON block WORK may be used if desired. The argument list is as follows:

ICOMS: A vector of dimension NINFCI, which upon entry contains the integer element information. The address assigned to ICOMS in the base program corresponds to the first word of integer information for the element.

COMS: A vector of dimension NINFCR, which upon entry contains real element information. The address assigned to COMS in the base program corresponds to the first word of real information for the element.

NINFCI: See INEL routine. This value is assigned by the base program.

NINFCR: See INEL routine. This value is assigned by the base program

MFST: See INEL routine. This value is assigned by the base program.

The title of the subroutine, for example for element type 1, must be as follows.

```
SUBROUTINE OUT1 (ICOMS, COMS, NINFCI, NINFCR, MFST)
```

The following steps must be performed within the subroutine.

(a) Transfer the data from the arrays COMS and ICOMS to the element information blocks INFELR and INFELI. The procedure explained in Section 3.4.1 must be used.

(b) Print an appropriate heading for the results if IMEM equals MFST.

(c) Print the envelope results.

4. INTERFACE BETWEEN ANALYSIS AND OPTIMIZATION PACKAGES

4.1 INTRODUCTION

The INTEROPTDYN program, described in Section 2, is a general purpose optimization program which can be used to solve a variety of design problems. To define a particular problem, the user needs to supply the following routines:

(i) PARSYM: Called once at the beginning of the program to specify fixed system parameters.

(ii) FUNCF: To evaluate cost function.

(iii) GRADF: To evaluate cost gradient.

(iv) FUNCG: To evaluate simple inequality constraints.

(v) GRADG: To evaluate gradients of simple inequality constraints.

(vi) FUNCPH: To evaluate functional inequality constraints.

(vii) GRADPH: To evaluate gradients of functional inequality constraints.

The structural analysis program, MINI-ANSR, is called from these subroutines. This structure allows the user maximum flexibility in terms of computing constraint functions and their gradients. Moreover, it preserves the modular structure of the package. For example, a new structural analysis program could be added to replace MINI-ANSR without any difficulty.

In the following sections the calling sequence and functions of the above subroutines are described. Two examples of optimal design are discussed in Section 5, in detail, to clarify the interface between these subroutines and the MINI-ANSR routines.

4.2 CALLING SEQUENCE AND TASKS TO BE PERFORMED BY FUNCTION EVALUATION ROUTINES

The calling sequence and tasks to be performed by function evaluation routines are given below. Note that all the variables identified as input (I) are set in the base program, INTEROPTDYN, and should not be changed in the function evaluation routines.

1. PARSYM:

This subroutine is called once only at the beginning of the program and is used to specify the fixed system parameters. It is called from the base program as follows:

```
CALL PARSYM (N, Z)
```

where the arguments have the following meaning:

N: Number of optimization variables (input).

Z: Vector containing current values of optimization (design) variables (I).

This subroutine is used to perform the following tasks:

(i) Initialize COMMON blocks needed in function evaluation and analysis program.

(ii) Read problem related input data.

(iii) Declare variables into the INTEROPTDYN symbol table which need to be changed interactively, as explained below.

(iv) Define variables which remain constant during optimization.

(v) Print a short description of the problem on the screen, if desired.

The variables which need to be accessed during execution must be declared into the INTEROPTDYN symbol table. This can be achieved by calling a subroutine 'DECLAR' as follows:

```
CALL DECLAR ('VNAME','TYPE',IDIM,VAR.NROW,NCOL)
```

where the variables in the argument have the following meaning:

VNAME: Variable name to be used during interaction. Usually same as FORTRAN variable name,but could be different.

TPYE: Type of variable, first character determines type:

I - integer

L - logical

R - real

D - double precision

C - complex

S - character string

IDIM: Dimension parameter of the variable.

0 - scalar variable

1 - one dimensional array

>1 - declared row dimension for a two-dimensional array

VAR: Variable to be declared.

NROW: Number of rows, (0 for scalar).

NCOL: Number of columns (0 for scalar).

Any number of variables can be declared in this fashion. If a variable with the same name already exists in the symbol table, the program will give an error message.

2. FUNCF:

This subroutine evaluates the cost function f. It is called from the base program as follows:

```
CALL FUNCF (N,Z,F,NFUNCF)
```

where the arguments have the following meaning:

N: Number of optimization variables, (input).

Z: Vector containing current values of optimization variables, (input).

F: Value of the objective function f, (output).

NFUNCF: A counter, which counts the number of times this subroutine is called, (input).

3. GRADF:

This subroutine evaluates the gradients of the objective function. The calling sequence for this subroutine is:

CALL GRADF (N,Z,GRAD)

where the arguments have the following meaning:

N: Number of optimization variables, (input).

Z: Vector containing current values of optimization variables, (input).

GRAD: Vector containing gradients of objective function, (output). The ith entry in this vector should contain the partial derivative of the objective function with respect to the ith optimization variable.

4. FUNCG:

This subroutine evaluates conventional inequality constraint functions (functions "g"). It is called from the base program as follows:

CALL FUNCG (N,JP,Z,G,PSI,NFUNCG)

where the arguments have the following meaning:

N: Number of optimization variables, (input).

JP: Number of constraints of this type, (input).

Z: Vector containing current values of optimization variables, (input).

G: Vector of functions "g", having dimension "JP", (output). These functions could be arranged in any order, but the corresponding gradients must follow the same order in subroutine GRADG.

PSI: Function ψ. At input it is initialized to its proper value by the main program. The maximum of functions g is computed and PSI is set equal to the greater of its input value or the maximum g function value at output. This should be achieved by adding the following FORTRAN statements, just before RETURN.

```
      DO 100 I = 1, JP
  100 IF (G(I).GT.PSI)PSI = G(I)
```

NFUNCG: A counter which is set equal to the number of the current call to this subroutine, (input).

5. GRADG:

This subroutine evaluates the gradients of conventional inequality constraints (functions g). The calling sequence for this subroutine is:

```
CALL GRADG (N,J,Z,GRAD)
```

where the arguments have the following meaning:

N: Number of optimization variables, (input).

J: Serial number of the constraint function for which the gradient is to be evaluated. A separate call is made for evaluation of gradient of each function, (input).

Z: Vector containing current values of optimization variables, (input).

GRAD: Vector containing gradient of jth g constraint with respect to the optimization variables. The dimension of this vector is "N". The ith entry in this vector should contain the partial derivative of the jth conventional constraint function with respect to the ith optimization variable, (output).

6. FUNCPH:

This subroutine evaluates dynamic inequality constraint functions (functions ϕ). It is called from the base program as follows:

```
CALL FUNCPH (N,NJQ,JQ,Z,WO,WC,DELTAW,NQ,PHI,PSI,NFUNCP)
```

where the arguments have the following meaning:

N: Number of optimization variables, (input).

NJQ: Row dimension of matrix PHI in the main program, (input).

JQ: Number of constraints of this type, (input).

Z: Vector containing current values of optimization variables, (input).

WO: Initial value of the interval over which the functional constraint is to be evaluated, (input).

WC: Final value of the interval over which the functional constraint is to be evaluated, (input).

NQ: Number of discretization points, (input).

DELTAW: Discretization interval, defined as.

$$DELTAW = (WC - WO)/NQ$$

PHI: Matrix containing values of functions ϕ. The ith row of this matrix contains values of ith functional constraint at specified intervals, (output).

PSI: Function ψ. At input it is initialized to its proper value by the main program. The maximum of functions ψ is computed and PSI is set equal to the greater of its input value or the maximum ϕ function value at output. This should be achieved by adding the following FORTRAN statements, just before RETURN.

```
      DO 100 L = 1, JQ
      DO 100 K = 1, NQ
      IF(PHI(L,K).GT.PSI)PSI = PHI(L,K)
  100 CONTINUE
```

NFUNCP: A counter which is set equal to the number of the current call to this subroutine, (input).

7. GRADPH:

This subroutine evaluates gradients of dynamic inequality constraint functions (functions ϕ). It is called from the base program as follows:

```
CALL GRADPH (N,NJQ,NACTIV,JQ,WO,WC,DELTAW,NQ,NEPTF,L,Z,K,GRAD,IGRAD)
```

where the arguments have the following meaning:

N: Number of optimization variables, (input).

NJQ: Row dimension of matrix NEPTF, (input).

NACTIV: Column dimension of matrix NEPTF, (input).

JQ: Number of functional constraints, (input).

WO: Initial value of the interval over which the functional constraint is to be evaluated, (input).

WC: Final value of the interval over which the functional constraint is to be evaluated, (input)

NQ: Number of discretization points, (input).

DELTAW: Discretization interval, defined as

$$DELTAW = (WC - WO)/NQ$$

NEPTF: Matrix of points at which the ε-active intervals have local maxima. The ith row of this matrix corresponds to the ith functional constraint. It contains the mesh point number at which the constraint is active. The entries start from the first column and are in ascending order. The remainder of the entries is filled with zeros. This matrix could be used to store gradients of functional constraints only at the points included in this matrix. See example of the optimal design of a braced frame, where it is used for this purpose, (input).

L: Serial number of the current functional constraint. A separate call is made for evaluation of gradient of each ε-active point, (input).

Z: Vector containing current values of optimization variables, (input).

K: Current discretization point at which the gradient is desired, (input).

GRAD: Vector containing gradient of ϕ (L,K). The ith entry in this vector should contain the partial derivative of the lth functional constraint at the kth discretization point with respect to the ith optimization variable, (output).

IGRAD: A counter, which is equal to the number of calls to this subroutine in the current iteration. At the beginning of every iteration, this is set equal to one, (input).

4.3 MODIFICATION AND EXTRACTION OF ELEMENT INFORMATION

During the optimization phase, design variables are stored in a vector z. In structures, these variables are generally geometric or material properties of the elements. Before performing a new structural analysis, these quantities, therefore, need to be modified in the MINI-ANSR COMMON blocks. The constraints generally require element stresses and deformations, etc. which must be extracted and passed on to the function evaluation subroutines at the end of the analysis.

In order to perform this modification and/or extraction of element information from MINI-ANSR COMMON blocks, a user needs to supply the following routines for each element, in addition to the four subroutines described in Section 3.

(i) MDFL: Modify data in element COMMON blocks corresponding to optimization variables.

(ii) STOR: Store element information, relevant to the optimization package, in separate arrays accessible to the function evaluation routines.

Each of these subroutines must be identified by a number designating the element type, suffixed to the subroutine name. For example, the

names of subroutines for the element type 1, must be, MDFL1 and STOR1. Explanations of the tasks performed by each of the main subroutines, and the meanings of the variables of the argument lists, are given below.

4.3.1 Subroutine MDFL

The purpose of this subroutine is to modify data in COMMON blocks INFELR and INFELI , corresponding to the optimization variables. The argument list is as follows (in order):

ICOMS: A vector of dimension NINFCI, which upon entry contains the 'integer' element information. The address assigned to ICOMS corresponds to the first word of integer information for the element.

COMS: A vector of dimension NINFCR, which upon entry contains the 'real' element information. The address assigned to COMS corresponds to the first word of information for the element.

NINFCI: Number of words of integer information for each element in the group. This number equals the length of the COMMON INFELI for elements of the type being considered.

NINFCR: Number of words of real information for each element in the group. This number equals the length of the COMMON INFELR for elements of the type being considered.

Z: Vector containing current values of optimization variables.

The title of the subroutine, for example for element type 1, must be as follows.

```
SUBROUTINE MDFL1 (ICOMS,COMS,NINFCI,NINFCR,Z)
```

The following steps must be performed within the subroutine:

(a) Transfer the data from arrays ICOMS and COMS to the element information blocks INFELI and INFELR , as explained in Section 3.4.1.

(b) Set KST = 1.

(c) Set response values corresponding to previous variables equal to zero. This is easier to do through arrays ICOMS and COMS, if the quantities in these blocks are arranged such that the fixed quantities are placed before the response values.

(d) Modify the proper quantities.

(e) Transfer information in the blocks INFELI and INFELR to the arrays ICOMS and COMS. The procedure is explained in Section 3.4.1.

4.3.2 Subroutine STOR

This subroutine is used to store element information into arrays needed for function evaluation. It performs two tasks depending upon the value of the parameter IFLAG in the argument list.

If IFLAG = 1: Stores results of analysis in two-dimensional arrays.

IFLAG = 2: Stores element information which remains fixed during analysis e.g. lengths of elements and their connectivity for the structure geometry plot.

The argument list is as follows:

ICOMS: A vector of dimension NINFCI containing integer element information.

COMS: A vector of dimension NINFCR containing real element information.

NINFCI: Number of words of integer information for each element in the group.

NINFCR: Number of words of real information for each element in the group.

IFLAG: Flag which assigns different tasks, set by the base program.

The title of the subroutine, for example for element type 1, must be as follows.

```
SUBROUTINE STOR1 (ICOMS,COMS,NINFCI,NINFCR,IFLAG)
```

The following steps must be performed within the subroutine:

(a) Transfer the data from arrays ICOMS and COMS to the element information blocks INFELI and INFELR , as explained in Section 3.4.1.

(b) Perform two tasks depending upon the value of IFLAG.

4.4 GENERAL INTERFACING SUBROUTINES BETWEEN FUNCTION EVALUATION SUBROUTINES AND MINI-ANSR

In order to minimize coding for a new problem, a number of interfacing subroutines between function evaluation subroutines and MINI-ANSR has been devised. These subroutines are general and can be used for any problem.

1. Subroutine INANSR

This subroutine initializes MINI-ANSR COMMON blocks and calls the input module of MINI-ANSR to read input data for analysis. It can be called as follows.

```
CALL INANSR
```

Typically, this subroutine will be called from PARSYM.

2. Logical Function BIGDIF

This function finds the maximum difference between the current design variables and the one's for which the analysis was performed last. If this difference is greater than a certain tolerance value, the function returns .TRUE. otherwise .FALSE. . Its title card is:

```
LOGICAL FUNCTION BIGDIF (N,Z,ZSTR,DIFF,TOL)
```

where the arguments have the following meaning.

N: Number of optimization variables, (input).

Z: Vector of current design variables, (input).

ZSTR: Vector containing design variables values for which the structural analysis was done last, (input).

DIFF: Vector containing the difference between Z and ZSTR, (output).

TOL: Tolerance on the maximum difference, (input).

When the maximum absolute difference is greater than TOL, the function is returned as .TRUE. and ZSTR is set equal to Z. This function can be used to implement some approximation concepts. For example, if the difference is small, the cost and constraint functions can be approximated by using first order Taylor series expansion.

3. Subroutine ANAL

This subroutine calls appropriate subroutines of MINI-ANSR for static and/or dynamic analysis. The call to this subroutine is made as:

```
CALL ANAL (ZSTR)
```

where ZSTR contains the values of design variables for which the analysis is to be performed. This subroutine automatically calls for subroutine MODIFY, to modify the element data before it calls the static or dynamic analysis routines.

4. Subroutine SET

This subroutines extracts data from element information arrays which remain fixed during optimization. It calls problem dependent routines STOR1, STOR2, ... etc. with IFLAG = 2. It is called as:

```
CALL SET
```

Typically, it is called from PARSYM (after the call to INANSR) to store element geometry in arrays used for plotting.

5. Subroutine MODIFY

This subroutine is the driving routine for problem-dependent element information modification routines MDFL1, MDFL2, ... etc. It is called as:

CALL MODIFY (ZSTR)

where ZSTR is a vector containing design variables which is passed on to element routines. It is automatically called from subroutine ANAL before performing structural analysis.

6. Subroutine STORSP

This subroutine is not entirely problem-dependent but needs very little modification, if any, for a new problem. For an analysis, it is called at the end of every step. In this case, it calls STOR1, STOR2, ... etc. with IFLAG = 1. It is possible to skip several steps between storing results by passing on appropriate values of TSTART, TEND and NSKIP through COMMON block DYNPAR . Nodal responses, for both static and dynamic analysis, are saved only at the nodes which are specified for output (Section F, MINI-ANSR data preparation manual). Similarly, results for only those elements are stored for which response history is requested (see time history output code in Section G, element specification, of Appendix D).

5. EXAMPLE PROBLEMS

In order to further clarify the structure of the function evaluation subroutines and their interface with the MINI-ANSR subroutines, the following design problems are discussed here:

1. Minimum weight design of a ten-bar elastic truss subjected to static loading.
2. Minimum weight design of a two-story braced frame subjected to an impulsive base motion.

The design problems are formulated and listings of the function evaluation subroutines are subsequently given in Appendix E. Numerical results are presented and interaction is illustrated by giving typical dialogues between the user and the computer.

5.1 OPTIMAL DESIGN OF AN ELASTIC TRUSS SUBJECTED TO STATIC LOADING

A ten-member cantilever truss, shown in Figure 4, is designed for minimum weight. This truss has been used extensively in the literature for evaluating algorithms. For example, see [14] where different results are compared. Constraints are placed on the nodal displacements, member stresses and minimum member sizes. The objective and constraint functions can be expressed as follows:

OBJECTIVE FUNCTION:

$$f(\underline{z}) = \text{Weight of the structure} = \rho \sum_{i=1}^{10} L_i Z_i \qquad (5.1.1)$$

where ρ = Material density.

L_i = Length of ith element.

Z_i = ith design variable, which is the area of the ith element.

CONSTRAINT FUNCTIONS:

(a) displacement constraints

$$(u_{ix})^2 \leq (u_{ax})^2 \quad i = 1,\ldots,4 \tag{5.1.2}$$

$$(u_{iy})^2 \leq (u_{ay})^2 \quad i = 1,\ldots,4 \tag{5.1.3}$$

where

u_{ix} = displacement at ith node in X-direction

u_{iy} = displacement at ith node in Y-direction

u_{ax} = allowable displacement in X-direction

u_{ay} = allowable displacement in Y-direction

(b) stress constraints

$$(\sigma_i)^2 \leq (\sigma_a)^2 \quad i = 1,\ldots,10 \tag{5.1.4}$$

where

σ_i = stress in the ith member

σ_a = allowable stress

(c) minimum member size

$$Z_i \geq Z_i^L \quad i = 1,\ldots,10$$

where

Z_i^L = lower limit on the member area.

These constraints can be expressed as:

$$
\begin{aligned}
g^i(\underline{z}) &= -z_i + z_i^L && i = 1,\ldots,10 \\
g^i(\underline{z}) &= \left(\frac{u_{ix}}{u_{ax}}\right)^2 - 1.0 && i = 11,\ldots,14 \\
g^i(\underline{z}) &= \left(\frac{u_{iy}}{u_{ay}}\right)^2 - 1.0 && i = 15,\ldots,18 \\
g^i(\underline{z}) &= \left(\frac{\sigma_i}{\sigma_a}\right)^2 - 1.0 && i = 19,\ldots,28
\end{aligned}
\qquad (5.1.5)
$$

GRADIENTS:

The gradients of the objective and the constraint functions can be expressed as:

$$\nabla f(\underline{z}) = \rho\,[L_1, L_2, \ldots, L_{10}]^T \qquad (5.1.6)$$

$$\nabla g^1(\underline{z}) = [-1, 0, 0, \ldots, 0]^T$$

$$\nabla g^2(\underline{z}) = [0, -1, 0, \ldots, 0]^T$$

$$\nabla g^{10}(\underline{z}) = [0, 0, \ldots, 0, -1]^T$$

$$\nabla g^{11}(\underline{z}) = \frac{2u_{1x}}{u_{ax}^2}\left[\frac{\partial u_{1x}}{\partial z_1}, \ldots, \frac{\partial u_{1x}}{\partial z_{10}}\right]^T$$

$$\nabla g^{14}(\underline{z}) = \frac{2u_{4x}}{u_{ax}^2}\left[\frac{\partial u_{4x}}{\partial z_1}, \ldots, \frac{\partial u_{4x}}{\partial z_{10}}\right]^T$$

$$\nabla g^{15}(\underline{z}) = \frac{2u_{1y}}{u_{ay}^2}\left[\frac{\partial u_{1y}}{\partial z_1}, \ldots, \frac{\partial u_{1y}}{\partial z_{10}}\right]^T$$

$$\nabla g^{18}(\underline{z}) = \frac{2u_{4y}}{u_{ay}^2}\left[\frac{\partial u_{4y}}{\partial z_1}, \ldots, \frac{\partial u_{4y}}{\partial z_{10}}\right]^T$$

$$\nabla g^{19}(\underline{z}) = \frac{2\sigma_1}{\sigma_a^2}\left[\frac{\partial \sigma_1}{\partial z_1}, \ldots, \frac{\partial \sigma_1}{\partial z_{10}}\right]^T$$

$$\nabla g^{28}(\underline{z}) = \frac{2\sigma_{10}}{\sigma_a^2}\left[\frac{\partial\sigma_{10}}{\partial z_1}, \ldots, \frac{\partial\sigma_{10}}{\partial z_{10}}\right]^T \qquad (5.1.7)$$

NUMERICAL DATA

Material density, $\rho = 0.1$ lb/in^3

Young's Modulus, $E = 10000.$ ksi

Displacement limits, $u_{ax} = u_{ay} = \pm 2$ in.

Stress limit, $\sigma_a = \pm 25$ ksi

Lower limit on member area, $z^L = 0.1$ in^2

Load data: Vertical loads of 100 k at nodes 2 and 4.

A typical dialogue between the user and the computer is presented in the following pages to illustrate a simple level of interaction in the solution of this problem. The dialogue has been obtained using an HP graphics terminal connected with an hard-copy printer, which can copy on paper both the alphanumeric and the graphic parts of the screen.

The name of the executable file for this particular problem is 'trussint'. After typing in this name, some headings appear on the screen, followed by the request to specify the name of the input data file. 'Truss. data' is the name of the input file in this case; it contains values of the optimization algorithm parameters as suggested at the end of Appendix C.

The 'go' command moves the program to break point QP90, at the end of step 3a. At this point, before starting the direction finding process for the first iteration, command 'grinit', which initializes graphics, is given and, after specification of the terminal type, a

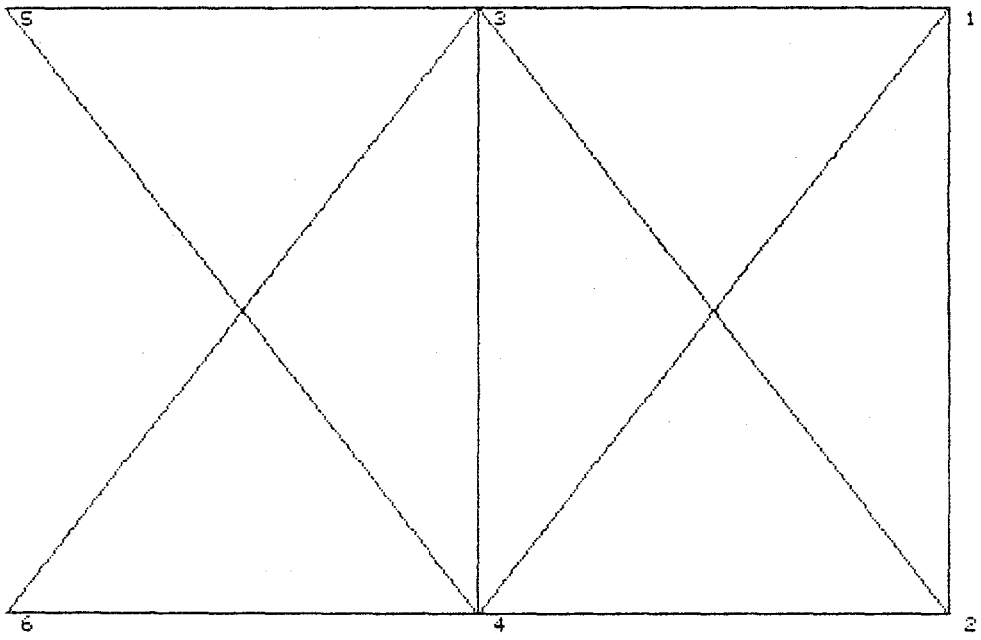
plot of the structure is requested using the macro 'gstruct'. The initial design vector, z, is also printed, using command 'print'.

We are ready now to start the design iterations and we use macro 'run' and option 'store' to perform 10 iterations and print results. The initial design is feasible, in this case, as can be verified by checking the value of function ψ at the first iteration, (PSI = 0). After 10 iterations the value of the cost function is more than halved and the design vector has been considerably modified. Four components in particular, z(2), z(5), z(6), z(10) have been reduced substantially. This suggests that one set these four values to their lowest limit $z^L = 0.1$, before continuing with 15 more iterations. As a consequence of the modification, iteration 10 is repeated, but the new z vector, which corresponds to a lower cost, is still feasible.

At the end of iteration 25 the values of the cost function and of the vector z are not very far away from the ones reported in [14], but the convergence rate has considerably slowed down for the last ten iterations. This fact is particularly manifest in the graph of cost function f versus number of iterations, obtained using macro 'graphf'.

Execution is stopped after using macro 'graphz' to plot the values of three components of the design vector, z(1), z(5), z(8) versus number of iterations.

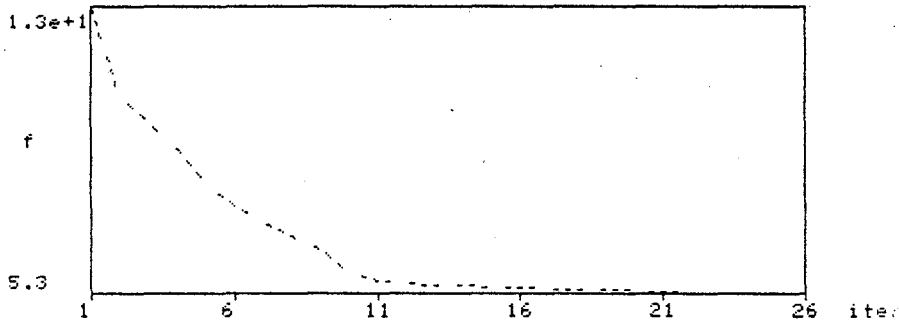
```
% trussint

          Optimization Based Computer-Aided Design Group
                University of California
                  Berkeley, California
                       U. S. A.


                    INTRAC-OPTDYN
          An Interactive Optimization Program for
          Design Problems Which can be Expressed as

                          Minimize  f (z)
                             z
          subject to
                           max  phi(z,t) <= 0
                            t
                               g(z)     <= 0
 Name of input data file:
( Default is "/usr/optcad/ciampi/optnsr.d/data" )
>truss.data
>go
>where
 Breakpoint:  QP90
>grinit
enter terminal type (2=4027 3=RAMTEK 4=HP):
4
>gstruct
>
```



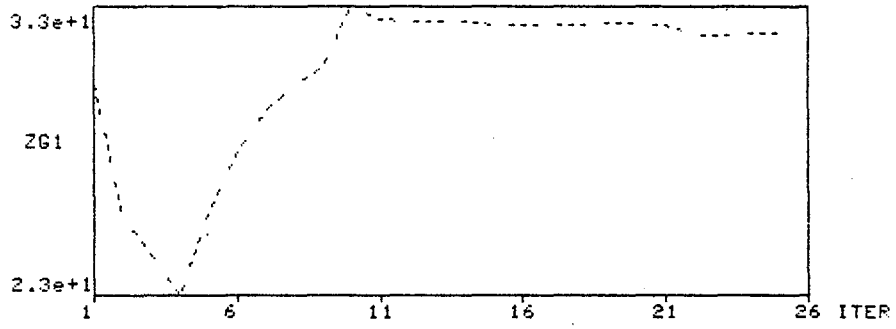
```
>print z
     30.0000
     30.0000
     30.0000
     30.0000
     30.0000
     30.0000
     30.0000
     30.0000
     30.0000
     30.0000
>run 10 store
The results of the entire computation will be stored
in the arrays FG PSIG and ZG(N:K).
Please state the total number of iterations you intend to
carry out: type in K = ?
     #50
I = 1   F = 12.5894   PSI = 0.
        THETA = 0.   E = 0.2
I = 2   F = 10.3494   PSI = 0.
        THETA = -0.009072   E = 0.2
I = 3   F = 9.6774   PSI = 0.
        THETA = -0.009072   E = 0.2
I = 4   F = 9.0054   PSI = 0.
        THETA = -0.009072   E = 0.2
I = 5   F = 8.13893   PSI = 0.
        THETA = -0.00350922   E = 0.2
I = 6   F = 7.55013   PSI = 0.
        THETA = -0.00238465   E = 0.2
I = 7   F = 7.1146   PSI = 0.
        THETA = -0.00176387   E = 0.2
I = 8   F = 6.75702   PSI = 0.
        THETA = -0.00144821   E = 0.2
I = 9   F = 6.41118   PSI = 0.
        THETA = -0.00140065   E = 0.2
I = 10   F = 6.0751   PSI = 0.
        THETA = -0.000408343   E = 0.2
Execution suspended at the end of STEP2
You may want to modify
 1. the current design vector Z
 2. the smear parameter E
>print z
     32.5313
    0.642208
     32.4006
     14.7785
    0.621857
    0.642208
     15.8748
     20.8112
     21.5719
     3.35659
>
>set z(2:1)=0.10
>set z(5:1)=0.10
>set z(6:1)=0.10
>set z(10:1)=0.10
```

```
>print z
     32.5313
    0.100000
     32.4006
     14.7785
    0.100000
    0.100000
     15.8748
     20.8112
     21.5719
    0.100000
>run 16 store
RESTART STEP2
I = 10  F = 5.85147  PSI = 0.
        THETA = -0.000408343  E = 0.2
I = 11  F = 5.60315  PSI = 0.
        THETA = -0.00100569  E = 0.2
I = 12  F = 5.55423  PSI = 0.
        THETA = -0.0073379  E = 0.2
I = 13  F = 5.53341  PSI = 0.
        THETA = -0.000937207  E = 0.2
I = 14  F = 5.51795  PSI = 0.
        THETA = -0.00722742  E = 0.1
I = 15  F = 5.47403  PSI = 0.
        THETA = -0.00658822  E = 0.1
I = 16  F = 5.46926  PSI = 0.
        THETA = -0.000715536  E = 0.1
I = 17  F = 5.43548  PSI = 0.
        THETA = -0.000456053  E = 0.1
I = 18  F = 5.42404  PSI = 0.
        THETA = -0.00571833  E = 0.1
I = 19  F = 5.39775  PSI = 0.
        THETA = -0.000354949  E = 0.1
I = 20  F = 5.39583  PSI = 0.
        THETA = -0.000959081  E = 0.1
I = 21  F = 5.3867  PSI = 0.
        THETA = -0.00456347  E = 0.1
I = 22  F = 5.33929  PSI = 0.
        THETA = -0.000640052  E = 0.1
I = 23  F = 5.33397  PSI = 0.
        THETA = -0.000799126  E = 0.1
I = 24  F = 5.32246  PSI = 0.
        THETA = -0.000517934  E = 0.1
I = 25  F = 5.31746  PSI = 0.
        THETA = -0.000748912  E = 0.1
Execution suspended at the end of STEP2
You may want to modify
 1. the current design vector Z
 2. the smear parameter E
>print z
     31.6438
    0.178467
     28.7129
     14.8891
    0.199684
    0.465926
     8.24085
     21.0192
     21.2512
    0.129929
```

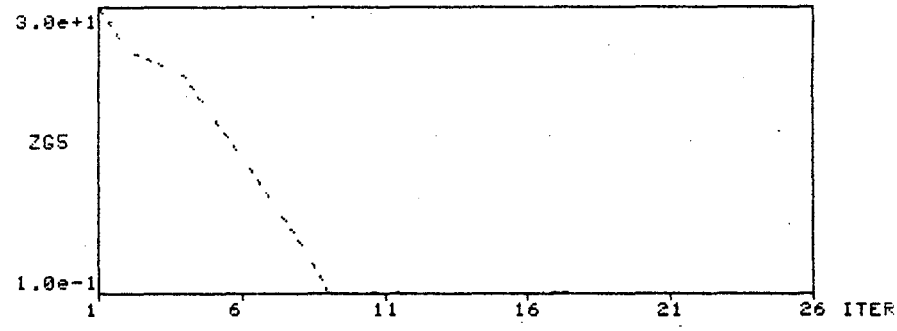
```
>graphf n
>
```



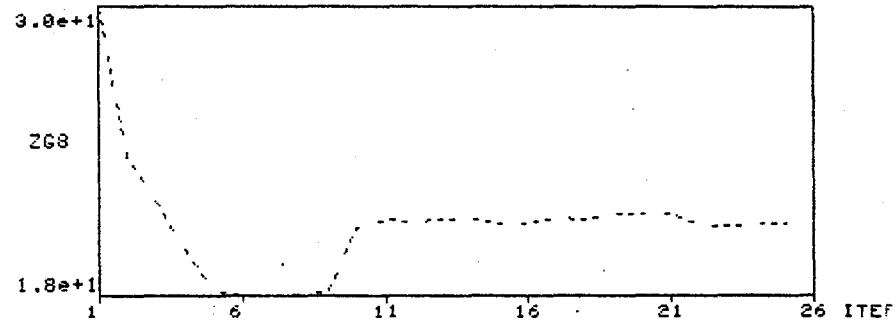

```
>graphz 1
>
```




```
>graphz 5
>
```




```
>graphz 8
>
```




```
>stop
%
```

5.2 OPTIMAL DESIGN OF AN INELASTIC BRACED FRAME SUBJECTED TO AN IMPULSIVE BASE MOTION

A two story shear-type braced frame (Fig. 5), subjected to an impulsive base motion, is designed for minimum weight, under both conventional and functional constraints. Both dynamic loads and nonlinearities are present in the example. Material nonlinearity is allowed in the diagonal bracing, which is modeled using the nonlinear truss element described in 3.3.3.

The ability of MINI-ANSR to accept specifications of both zero displacements and equal displacement components for different nodes, has been used to model the shear-type structure. Four design parameters appear naturally, the two areas of the diagonal bracing and the two moments of inertia of the columns, at the first and second floors, respectively. Area of cross section, A, and elastic section modulus, S, of columns are assumed to be related to moment of inertia I by the empirical relationships:

$$A = 0.8 \quad I^{1/2} \quad \text{(in inch units)} \tag{5.2.1}$$

$$S = 0.78 \quad I^{3/4} \tag{5.2.2}$$

For convenience of formulation of the problem, variables I_1 and I_2, having the dimensions of moments of inertia, are used as design variables instead of areas. For the bracing the same relationship 5.2.1 is assumed to hold. The four design variables are then I_1, I_2 for the bracings, I_3 and I_4 for the columns. Constraints considered refer to story drifts, stresses in columns, minimum member sizes and ratio between weight of the bracing and total weight of the structure. The objective and constraint functions and their gradients are expressed as follows:

OBJECTIVE FUNCTION:

$$f(\underline{z}) = W_t = W_b + W_c = \text{total weight of the structure}$$
$$= \rho \ell_b (A_1 + A_2) + 2\rho h_c (A_3 + A_4) =$$
$$= 0.8\ \rho \ell_b (I_1^{1/2} + I_2^{1/2}) + 1.6\ \rho h_c (I_3^{1/2} + I_4^{1/2})$$

GRADIENT OF f

$$\nabla \underline{f} = 0.4\ \rho \begin{Bmatrix} \ell_b & I_1^{-1/2} \\ \ell_b & I_2^{-1/2} \\ 2h_c & I_3^{-1/2} \\ 2h_c & I_4^{-1/2} \end{Bmatrix}$$

CONVENTIONAL CONSTRAINTS:

$$\left.\begin{array}{ll} 1) & -I_1 + I_{min}^b \leq 0 \\ 2) & -I_2 + I_{min}^b \leq 0 \\ 3) & -I_3 + I_{min}^c \leq 0 \\ 4) & -I_4 + I_{min}^c \leq 0 \end{array}\right\} \quad \text{Positiveness of the design variables } I_{min}^b,\ I_{min}^c > 0$$

5) The weight of the bracing is desired to be less than a fixed fraction α of the total weight of the frame:

$$\frac{W_b}{\alpha\ (W_b + W_c)} - 1 \leq 0$$

GRADIENTS OF CONVENTIONAL CONSTRAINTS:

$$\nabla g^1 = [-1, 0, 0, 0]^T \quad , \quad \nabla g^2 = [0, -1, 0, 0]^T \ ,$$
$$\nabla g^3 = [0, 0, -1, 0]^T \quad , \quad \nabla g^4 = [0, 0, 0, -1]^T \ ,$$
$$\nabla g^5 = \frac{1}{\alpha\ (W_b + W_c)} \begin{Bmatrix} (1-\beta) & 0.4\ \ell_b & I_1^{-1/2} \\ (1-\beta) & 0.4\ \ell_b & I_2^{-1/2} \\ -\beta & 0.8\ h_c & I_3^{-1/2} \\ -\beta & 0.8\ h_c & I_4^{-1/2} \end{Bmatrix} \ , \text{ where } \beta = \frac{W_b}{W_b + W_c}$$

FUNCTIONAL CONSTRAINTS:

Maximum allowable story drift, a

1) $|u_1| \leq a$ or $\left(\frac{u_1}{a}\right)^2 - 1 \leq 0$

2) $|u_2 - u_1| \leq a$ or $\left(\frac{u_2 - u_1}{a}\right)^2 - 1 \leq 0$

Maximum allowable stress in columns, σ_a

3) $\left|\frac{M_{1c}}{S_{1c}}\right| \leq \sigma_a$ or $\frac{M_{1c}^2}{0.78^2 \, I_3^{3/2} \, \sigma_a^2} - 1 \leq 0$

4) $\left|\frac{M_{2c}}{S_{2c}}\right| \leq \sigma_a$ or $\frac{M_{2c}^2}{0.78^2 \, I_4^{3/2} \, \sigma_a^2} - 1 \leq 0$

where

$$u_1 = \hat{u}_1 (\underline{z}, t)$$

$$u_2 = \hat{u}_2 (\underline{z}, t)$$

$$M_{1c} = \hat{M}_{1c} (\underline{z}, t)$$

$$M_{2c} = \hat{M}_{2c} (\underline{z}, t)$$

GRADIENTS OF FUNCTIONAL CONSTRAINTS:

$$\nabla\phi^1 = \frac{2u_1}{a^2} \begin{Bmatrix} \frac{\partial u_1}{\partial I_1} \\ \frac{\partial u_1}{\partial I_2} \\ \frac{\partial u_1}{\partial I_3} \\ \frac{\partial u_1}{\partial I_4} \end{Bmatrix}, \quad \nabla\phi^2 = \frac{2\,(u_2 - u_1)}{a^2} \begin{Bmatrix} \frac{\partial u_2}{\partial I_1} - \frac{\partial u_1}{\partial I_1} \\ \frac{\partial u_2}{\partial I_2} - \frac{\partial u_1}{\partial I_2} \\ \frac{\partial u_2}{\partial I_3} - \frac{\partial u_1}{\partial I_3} \\ \frac{\partial u_2}{\partial I_4} - \frac{\partial u_1}{\partial I_4} \end{Bmatrix}$$

$$\nabla\phi^3 = \frac{2M_{1c}}{0.78^2\ \sigma_a^2\ I_3^{3/2}} \begin{Bmatrix} \frac{\partial M_{1c}}{\partial I_1} \\ \frac{\partial M_{1c}}{\partial I_2} \\ \frac{\partial M_{1c}}{\partial I_3} - \frac{3}{4}\frac{M_{1c}}{I_3} \\ \frac{\partial M_{1c}}{\partial I_4} \end{Bmatrix}, \quad \nabla\phi^4 = \frac{2M_{2c}}{0.78^2\ \sigma_a^2\ I_4^{3/2}} \begin{Bmatrix} \frac{\partial M_{2c}}{\partial I_1} \\ \frac{\partial M_{2c}}{\partial I_2} \\ \frac{\partial M_{2c}}{\partial I_3} \\ \frac{\partial M_{2c}}{\partial I_4} - \frac{3}{4}\frac{M_{2c}}{I_4} \end{Bmatrix}$$

NUMERICAL DATA

Material density, $\rho = 0.1$ lb/in^3

Young's Modulus, $E = 30000.$ ksi

Maximum story drift, $a = \pm 0.45$ in.

Maximum stress in columns, $\sigma_a = \pm 24$ ksi

Minimum value of the design variables,

$I_{min} = 10.$ for the columns,

$I_{min} = 0.1$ for the bracing

Yield stress in the bracing, $\sigma_y = \pm 18$ ksi

Masses at each floor, $m_1 = m_2 = 208$ lb x sec^2/inch

Base acceleration, a rectangular pulse of 140 in/sec^2, acting for 0.5 sec.

Duration of analysis, 1 sec in 100 steps.

Numerical results for this example are presented in the form of an interactive dialogue with the computer, as in the previous problem. The name of the data file is 'brace. data' and the initial values of the optimization algorithm parameters are again the starting values suggested at the end of Appendix C. The structure geometry is

displayed* using the macro 'gstruct', and the results of the analysis corresponding to the initial design are plotted, using two new macros, specifically prepared for the problem, 'gdisp' and 'gmom'. These macros display horizontal displacements of the two floors and end moments in the columns at the first and second level, as functions of the number of time steps.

Ten iterations of the optimization procedure are then performed using the macro 'run' and the option 'store'. In this case the initial design is infeasible and seven iterations are needed to reach the feasible region.

At the end of the ten iterations cost function f and function ψ are plotted versus number of iterations, using macros 'graphf' and 'graphpsi', described in 2.6, and results of analysis corresponding to the new values of the design variables,now feasible, are displayed, again using commands 'gdisp' and 'gmom'.

Four more iterations are then requested, after which the decision is made to start monitoring very carefully what happens in the various stages of the procedure in order to make a possible rational adjustment of the parameters of the algorithm. Starting from iteration 15 macro 'step3' is used, which stops the execution at the end of step 3, that is after the calculation of a direction has been completed. Command 'prtang' gives at this point the angle between the direction vector and the cost function gradient and the angles between the direction vector and the ε-active constraint gradients. The first information that we have from 'prtang' is that there is no active

---

*The horizontal rigid girders are not drawn in view of the fact that we have chosen to model the shear frame by imposing constraints at joints, not by describing the horizontal girders as members.

constraint at the start of this iteration. We can now use the macro 'Armijo' in connection with the macro 'graphos', as explained in 2.9.4, to monitor what happens during the step length calculations up to the completion of iteration 15.

The information, which comes through the graphic representation, obtained using 'graphos', is very rich and can be fully appreciated only if the forming of the lines on the screen rather than only the final picture is observed. For iteration 15 the information can be expressed in this way: the step length is reduced in Armijo and the constraint which causes this reduction is the constraint $\phi(2)$, which was not even active at the start of the iteration. As a consequence the iteration is a bad one, as can be verified looking at the very small reduction in the cost function f from the previous step ('prtall' command has been used at the end of the iteration to print iteration number, cost function etc.).

In the subsequent iteration, as can be seen using 'prtang' after 'step3', constraint $\phi(2)$ is active and influences the choice of a direction. In 'Armijo' the step length is increased until constraint g(5) is violated; at the same time constraint $\phi(2)$ ceases to be active.

Iteration 16 has been a good one, but the next is not. In fact in the direction finding stage constraint $\phi(2)$ is not active, while, during the Armijo phase, it is still $\phi(2)$ which gives trouble and forces reduction of the step length.

Finally, in iteration 18 both g(5) and $\phi(2)$ are active and influence the choice of a direction, as a consequence the iteration proves to be a good one.

Monitoring closely the algorithm's behavior in iterations 15 through 18 has given sufficient indication for an adjustment of the

parameters. The slowing down of the solution process, connected with the alternation of a good step and a bad one, can be corrected by increasing the value of ε and forcing, consequently, both the constraints which are important at this stage, namely g(5) and $\phi(2)$, to be active at each iteration. However, increasing ε may not be sufficient, because ε may be automatically reset to the previously used smaller value in step 4 and execution sent back to step 3. It is also important to reduce parameter δ at the same time. This is actually done in this example and in particular ε is set = 0.4 and $\delta = 10^{-7}$.

The solution process is then advanced for 5 more iterations, during which the effectiveness of the adjustment of parameters is observed.

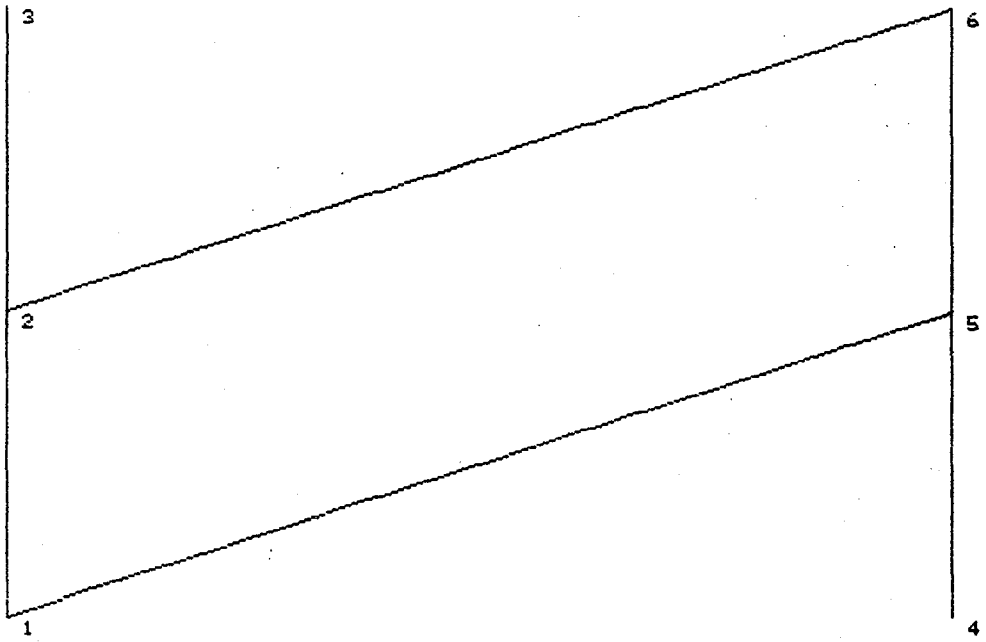
Ten more iterations are performed, after which the cost function is plotted. Again the beneficial effect of the adjustment of parameters is clearly visible in the graph.

After 18 more iterations the termination criterion is satisfied and a message of congratulations appears on the screen.

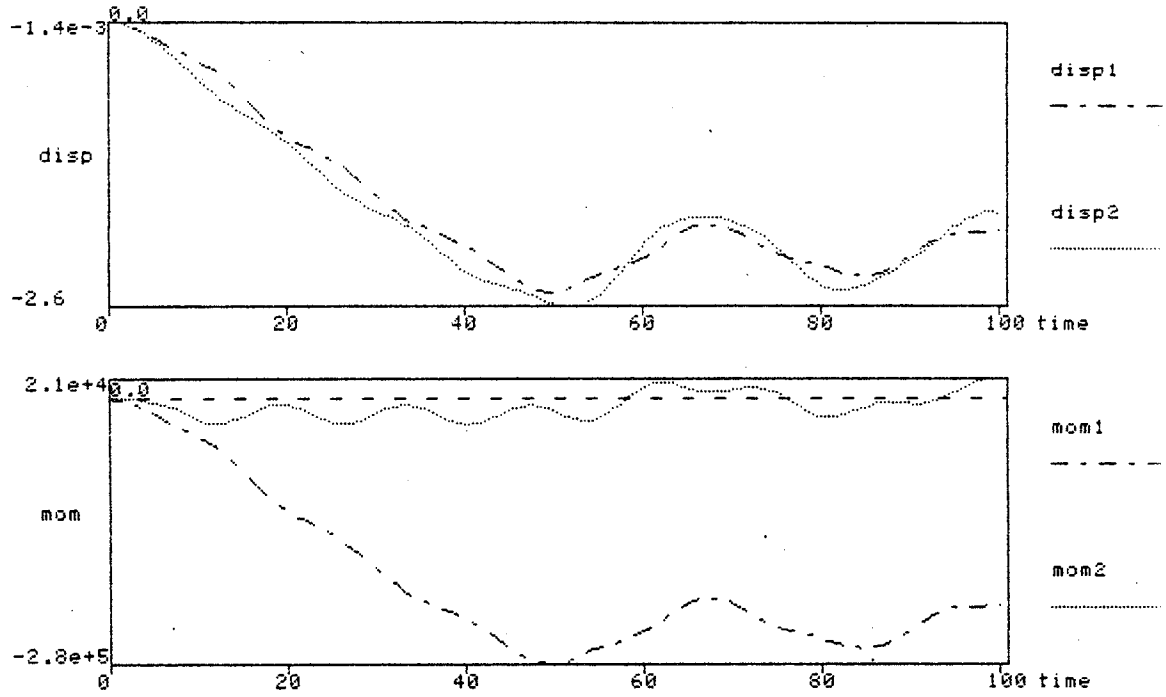
The constraints active at the optimum are g(4) and g(5), $\phi(1)$ and $\phi(2)$, as can be easily found by printing vector neptg and matrix neptf.

Results of the analysis corresponding to the optimal values of the design variables are also plotted before stopping.

```
          Optimization Based Computer-Aided Design Group
                University of California
                  Berkeley, California
                       U. S. A.


                    INTRAC-OPTDYN
          An Interactive Optimization Program for
          Design Problems Which can be Expressed as

                        Minimize  f (z)
                           z
          subject to
                         max  phi(z,t) (= 0
                          t
                              g(z)     (= 0
 Name of input data file:
( Default is "/usr/optcad/ciampi/optnsr.d/data" )
>brace.data
>print z
    20.0000
    20.0000
    20.0000
    20.0000
>go
>where
 Breakpoint:  QP90
>grinit
enter terminal type (2=4027 3=RAMTEK 4=HP 5=4025):
4
>gstruct
>
```

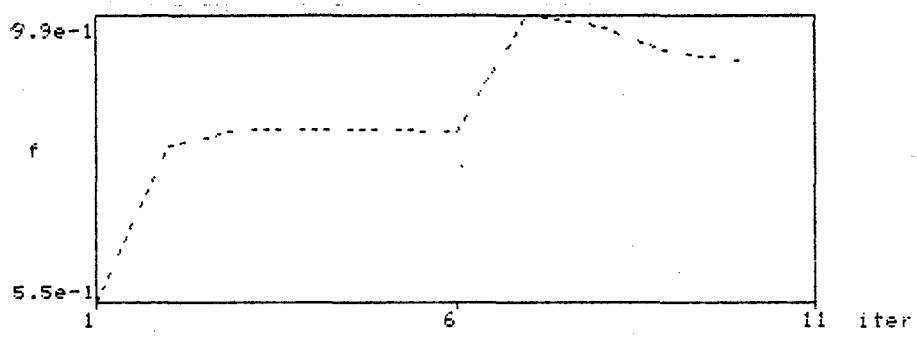


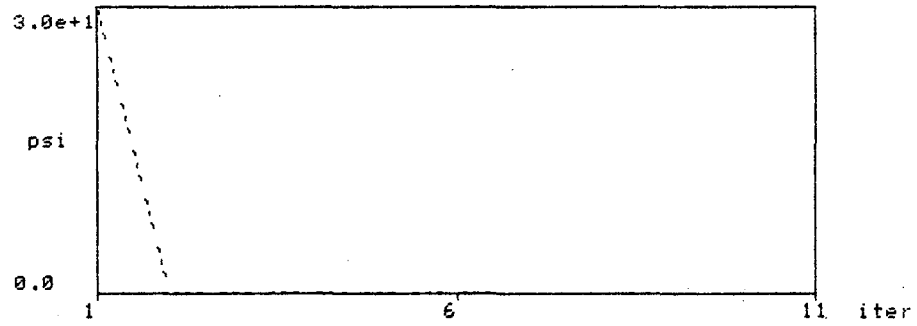
```
>gdisp
>gmom
>
```



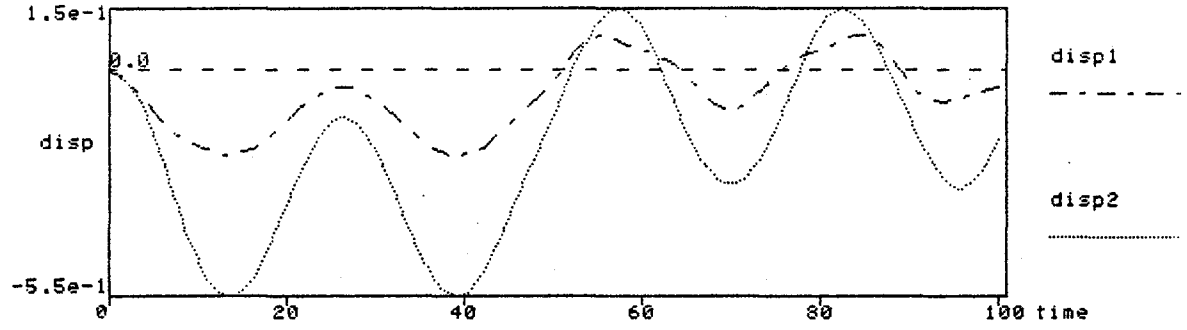

```
>run 10 store
The results of the entire computation will be stored
in the arrays FG PSIG and ZG(N:K).
Please state the total number of iterations you intend to
carry out: type in K = ?
    #100
I = 1  F = 0.545595  PSI = 29.7801
       THETA = 0.  E = 0.2
I = 2  F = 0.792752  PSI = 0.282556
       THETA = -0.504697  E = 0.2
I = 3  F = 0.815954  PSI = 0.19247
       THETA = -1.04588  E = 0.2
I = 4  F = 0.816696  PSI = 0.171042
       THETA = -1.30981  E = 0.2
I = 5  F = 0.81582  PSI = 0.152171
       THETA = -1.16204  E = 0.2
I = 6  F = 0.815235  PSI = 0.147019
       THETA = -0.995485  E = 0.2
I = 7  F = 0.994551  PSI = 0.0600003
       THETA = -0.121528  E = 0.2
I = 8  F = 0.979406  PSI = 0.
       THETA = -0.752006  E = 0.2
I = 9  F = 0.939237  PSI = 0.
       THETA = -3.69285E-4  E = 0.1
I = 10  F = 0.927019  PSI = 0.
       THETA = -3.32648E-4  E = 0.1
Execution suspended at the end of STEP2
You may want to modify
 1. the current design vector Z
 2. the smear parameter E
```

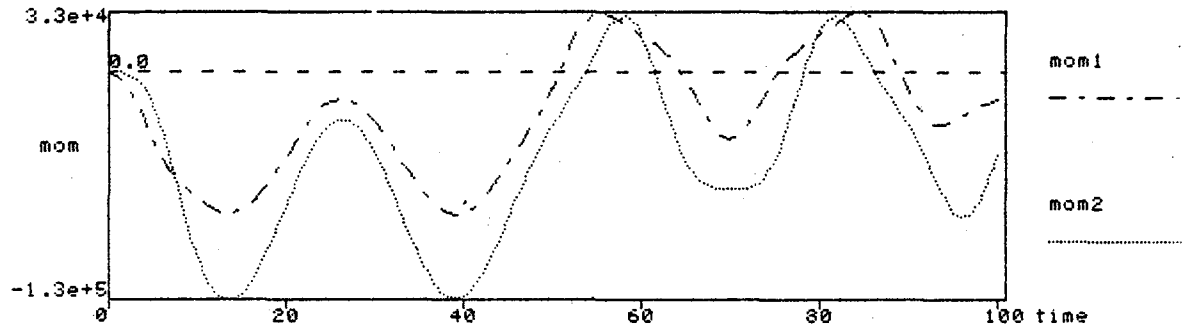
```
>print z
     93.3724
     17.8780
     70.7374
     68.0096
>graphf n
>graphpsi n
>
```



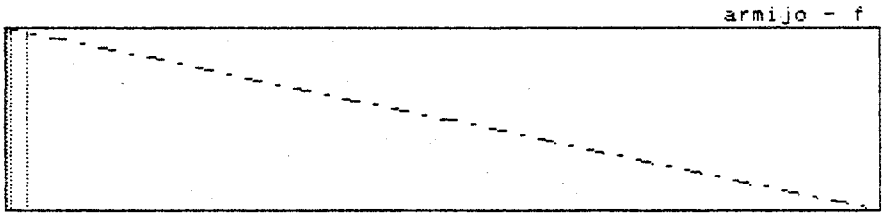


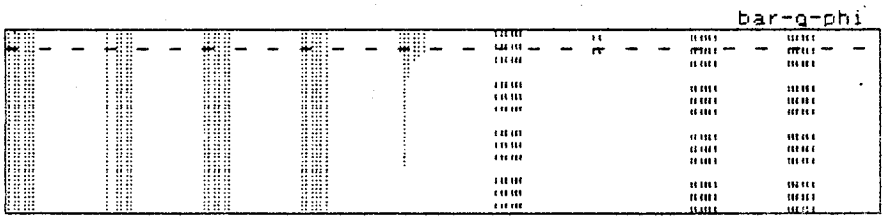

```
>gdisp
>gmom
>
```

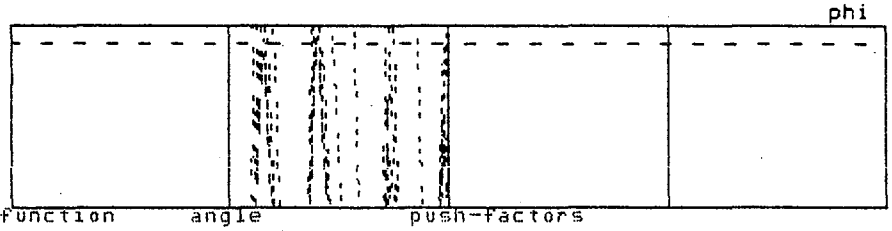




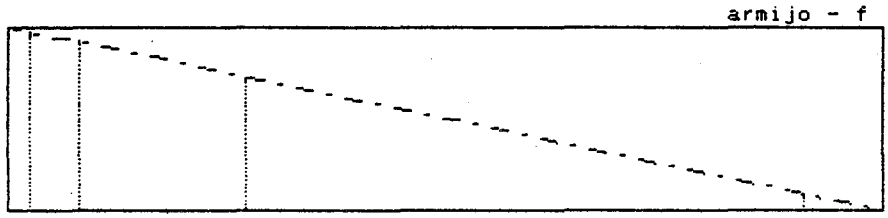
```
>run 4 store
I = 11  F = 0.914564  PSI = 0.
        THETA = -3.20792E-4  E = 0.1
I = 12  F = 0.901812  PSI = 0.
        THETA = -3.08574E-4  E = 0.1
I = 13  F = 0.900658  PSI = 0.
        THETA = -2.95997E-4  E = 0.025
I = 14  F = 0.859285  PSI = 0.
        THETA = -8.28777E-5  E = 0.025
Execution suspended at the end of STEP2
You may want to modify
 1. the current design vector Z
 2. the smear parameter E
>print z
    83.8015
    14.3257
    60.2608
    57.8666
>step3
Execution suspended at the end of STEP3
You may want to modify
 1. THETA parameters: PUSHF, PUSHG, PUSHPH, SCALE, GAMMA
 2. smear parameter: E
 3. test parameters: DELTA, MU1, MU2
Precomputation of the tests in STEP4 and STEP5
indicates that the program will branch to STEP6
>prtang
angles between search direction and cost
and e-active constraints gradients


function     angle          push-factors

F              180.            PUSHF =1.
>armijo 20 graphos
rmijo test satisfied after 6. iterations
Execution suspended at the end of STEP2
You may want to modify
 1. the current design vector Z
 2. the smear parameter E
Z>
```

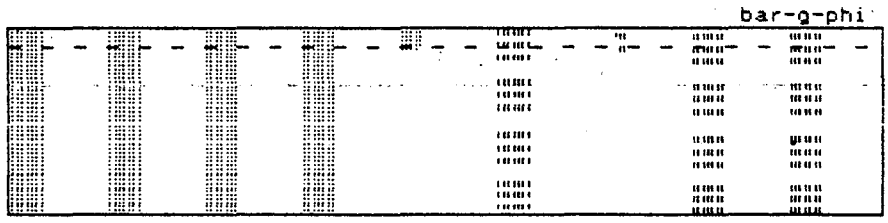


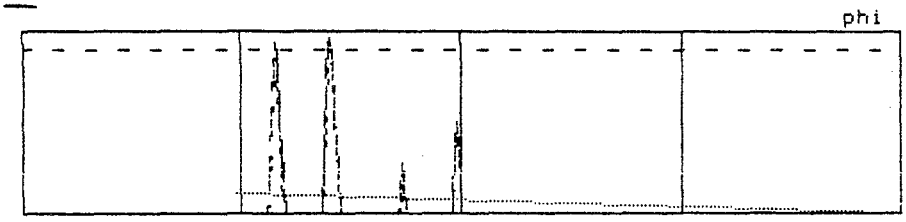





```
>prtall 0
I = 15  F = 0.858089  PSI = 0.
       THETA = -3.11836E-4  E = 0.025
>step3
Execution suspended at the end of STEP3
You may want to modify
 1. THETA parameters: PUSHF, PUSHG, PUSHPH, SCALE, GAMMA
 2. smear parameter: E
 3. test parameters: DELTA, MU1, MU2
Precomputation of the tests in STEP4 and STEP5
indicates that the program will branch to STEP6
>prtang
angles between search direction and cost
and e-active constraints gradients


function      angle          push-factors

F               123.722            PUSHF =1.
PHI(2,17)     91.3297          PUSHPH(2) = 1.
PHI(2,42)     90.7919          PUSHPH(2) = 1.
>armijo 20 graphos
annot plot Armijo: out of range
Zannot plot Armijo: out of range
Armijo test satisfied after 7. iterations
Execution suspended at the end of STEP2
You may want to modify
 1. the current design vector Z
 2. the smear parameter E
Z>
```



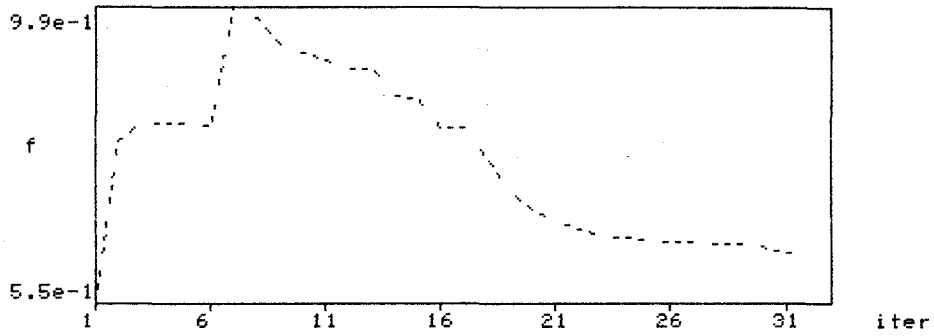





```
>prtall 0
I = 16  F = 0.810688  PSI = 0.
       THETA = -9.57191E-5  E = 0.025
>step3
Execution suspended at the end of STEP3
You may want to modify
 1. THETA parameters: PUSHF, PUSHG, PUSHPH, SCALE, GAMMA
 2. smear parameter: E
 3. test parameters: DELTA, MU1, MU2
Precomputation of the tests in STEP4 and STEP5
indicates that the program will branch to STEP6
>prtang
angles between search direction and cost
and e-active constraints gradients


function      angle          push-factors

F               180.             PUSHF =1.
G(5)            115.447            PUSHG(5) = 1.
>armijo 20 graphos
rmijo test satisfied after 6. iterations
Execution suspended at the end of STEP2
You may want to modify
 1. the current design vector Z
 2. the smear parameter E
Z>
>prtall 0
I = 17  F = 0.809427  PSI = 0.
       THETA = -3.35419E-4  E = 0.025
>step3
Execution suspended at the end of STEP3
You may want to modify
 1. THETA parameters: PUSHF, PUSHG, PUSHPH, SCALE, GAMMA
 2. smear parameter: E
 3. test parameters: DELTA, MU1, MU2
Precomputation of the tests in STEP4 and STEP5
indicates that the program will branch to STEP6
>prtang
angles between search direction and cost
and e-active constraints gradients


function      angle          push-factors

F               120.734            PUSHF =1.
G(5)            90.5866          PUSHG(5) = 1.
PHI(2,42)     90.7561          PUSHPH(2) = 1.
>armijo 20 graphos
annot plot Armijo: out of range
Zannot plot Armijo: out of range
Armijo test satisfied after 7. iterations
Execution suspended at the end of STEP2
```

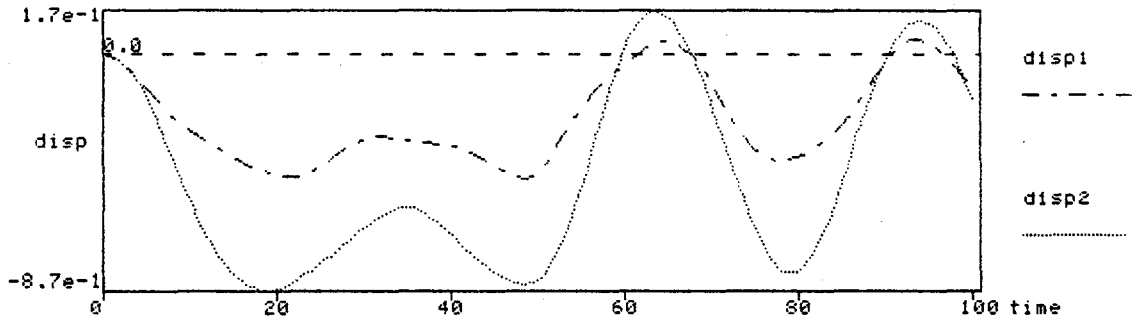
```
You may want to modify
 1. the current design vector Z
 2. the smear parameter E
Z>
>prtall 0
I = 18  F = 0.767099  PSI = 0.
       THETA = -8.72311E-5  E = 0.025
>
>

>set e=0.4
>set delta=1.e-7
>run 5 store
RESTART STEP2
I = 18  F = 0.767099  PSI = 0.
       THETA = -4.168E-4  E = 0.4
I = 19  F = 0.714417  PSI = 0.
       THETA = -1.09864E-4  E = 0.4
I = 20  F = 0.686152  PSI = 0.
       THETA = -2.08688E-4  E = 0.4
I = 21  F = 0.669059  PSI = 0.
       THETA = -4.168E-4  E = 0.2
I = 22  F = 0.659096  PSI = 0.
       THETA = -2.30062E-4  E = 0.2
Execution suspended at the end of STEP2
You may want to modify
 1. the current design vector Z
 2. the smear parameter E
>run 10 store
I = 23  F = 0.648823  PSI = 0.
       THETA = -2.35104E-4  E = 0.2
I = 24  F = 0.645509  PSI = 0.
       THETA = -2.51892E-4  E = 0.1
I = 25  F = 0.642181  PSI = 0.
       THETA = -2.5219E-4  E = 0.1
I = 26  F = 0.640577  PSI = 0.
       THETA = -4.03809E-4  E = 0.05
I = 27  F = 0.638971  PSI = 0.
       THETA = -4.01688E-4  E = 0.05
I = 28  F = 0.635625  PSI = 0.
       THETA = -2.49178E-4  E = 0.05
I = 29  F = 0.634614  PSI = 0.
       THETA = -2.50443E-4  E = 0.05
I = 30  F = 0.633601  PSI = 0.
       THETA = -2.50826E-4  E = 0.025
I = 31  F = 0.625075  PSI = 0.
       THETA = -9.67802E-9  E = 0.025
I = 32  F = 0.624293  PSI = 0.
       THETA = -6.25456E-8  E = 0.025
Execution suspended at the end of STEP2
You may want to modify
 1. the current design vector Z
 2. the smear parameter E
>print z
     35.9056
     13.1914
     48.7632
     15.4738
```

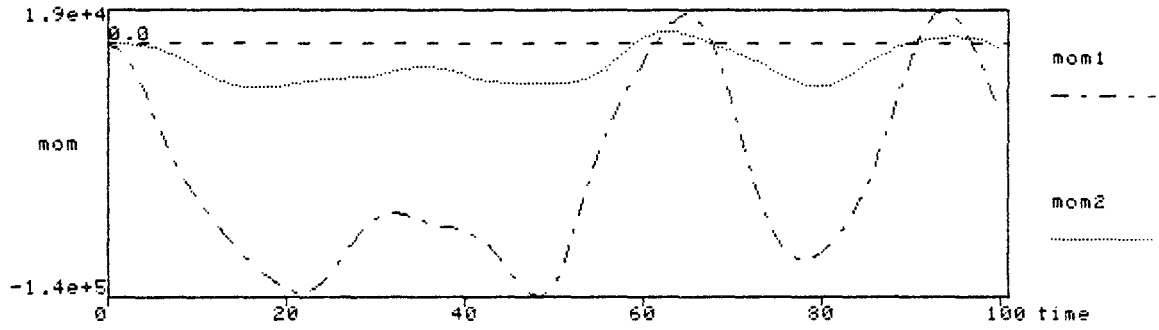
```
>graphf.n
>
```



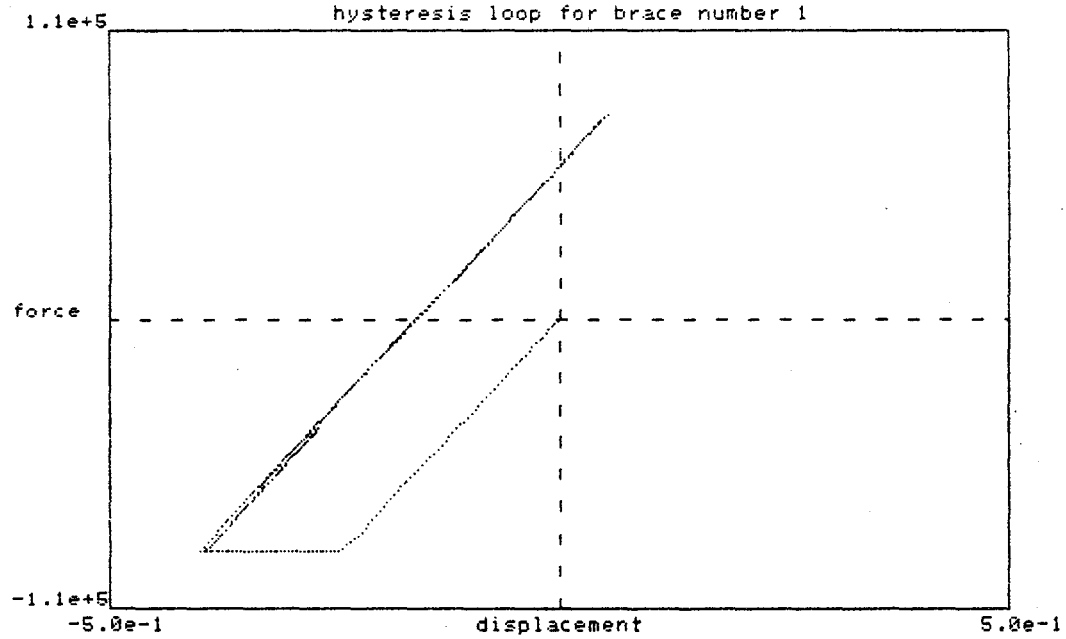

```
>run 30 prtall
I = 33   F = 0.623093   PSI = 0.
         THETA = -7.97265E-8   E = 0.025

I = 34   F = 0.622455   PSI = 0.
         THETA = -4.80294E-4   E = 0.025
I = 35   F = 0.621077   PSI = 0.
         THETA = -3.10373E-4   E = 0.025
I = 36   F = 0.6208   PSI = 0.
         THETA = -1.05315E-7   E = 0.025
I = 37   F = 0.620499   PSI = 0.
         THETA = -1.13584E-7   E = 0.025
I = 38   F = 0.620174   PSI = 0.
         THETA = -1.22273E-7   E = 0.025
I = 39   F = 0.620168   PSI = 0.
         THETA = -1.2572E-7   E = 0.025
I = 40   F = 0.619781   PSI = 0.
         THETA = -2.82831E-4   E = 0.0125
I = 41   F = 0.619569   PSI = 0.
         THETA = -1.55934E-4   E = 0.0125
I = 42   F = 0.619526   PSI = 0.
         THETA = -3.4879E-4   E = 0.00625
I = 43   F = 0.619483   PSI = 0.
         THETA = -3.48728E-4   E = 0.00625
I = 44   F = 0.61927   PSI = 0.
         THETA = -1.5601E-4   E = 0.00625
I = 45   F = 0.618031   PSI = 0.
         THETA = -2.43822E-5   E = 0.00625
I = 46   F = 0.618029   PSI = 0.
         THETA = -1.24928E-7   E = 0.00625
I = 47   F = 0.617913   PSI = 0.
         THETA = -2.8418E-4   E = 0.003125
I = 48   F = 0.617699   PSI = 0.
         THETA = -1.56505E-4   E = 0.003125
I = 49   F = 0.616444   PSI = 0.
         THETA = -2.47142E-5   E = 0.003125
I = 50   F = 0.616442   PSI = 0.
         THETA = -1.2364E-7   E = 0.003125

**********************************************************************************
**********
    congratulations, here is the optimal solution

    objective function value=  0.616442d+00
>
```

```
>print z
    35.1873
    13.1253
    57.1415
    10.0012
>print neptg
         0
         0
         0
         1
         1
         0
         0
         0
         0
         0
>print neptf
        49         0         0         0         0         0         0         0
         0         0
        18         0         0         0         0         0         0         0
         0         0
         0         0         0         0         0         0         0         0
         0         0
         0         0         0         0         0         0         0         0
         0         0
>print g(4:1)
   -1.21752d-03
>print g(5:1)
   -1.82749d-03
>print phi(1:49)
   -2.94798d-03
>print phi(2:18)
   -1.87089d-03
>gdisp
>gmom
>
```




1.9e+4
0.0
mom
-1.4e+5
0
20
40
60
80
100 time
mom1
mom2

```
>gloop 1
>stop
%
```


hysteresis loop for brace number 1
1.1e+5
force
-1.1e+5
-5.0e-1
displacement
5.0e-1

REFERENCES

1. Bathe, K.J., Wilson, E.L. and Peterson, F.E., "SAP IV - A Structural Analysis Program for Static and Dynamic Response of Linear Systems," Report No. EERC 73-11, Earthquake Engineering Research Center, University of California, Berkeley, June 1973.

2. Mondkar, D.P. and Powell, G.H., "ANSR-I A General Purpose Program for Analysis of Nonlinear Structural Response," Report No. EERC 75-37, Earthquake Engineering Research Center, University of California, Berkeley, December 1975.

3. Prager, W. and Sheu, C., "Recent Developments in Optimal Structural Design," Applied Mechanics Reviews, October 1968.

4. Venkayya, V.B., "Structural Optimization: A Review and some Recommendations," Intl. J. Numer. Methods Engr. Vol. 13, pp. 203-228 (1978).

5. Pierson, B.L., "A Survey of Optimal Structural Design Under Dynamic Constraints," Intl. J. Numer. Methods Engr. Vol. 4, pp. 491-499 (1972).

6. Rangacharyulu, M.A.V. and Done, G.T.S., "A Survey of Structural Optimization Under Dynamic Constraints," Shock Vib. Dig., Vol. 11, No. 12, December 1979.

7. Pister, K.S., "Optimal Design of Structures Under Dynamic Loading," NATO - NSF Advanced Study Institute on Optimization of Distributed Parameter Structures, University of Iowa, Iowa City, Iowa, May 21 - June 4, 1980.

8. Miura, H. and Schmit, L.A., Jr., "ACCESS-1, Approximation Concepts for Efficient Structural Synthesis - Program Documentation and User's Guide," NASA CR-144905, National Aeronautics and Space Administration, May 1976.

9. Polak, E., "Algorithms for Optimal Design," NATO-NSF Advanced Study Institute on Optimization of Distributed Parameter Structures, The University of Iowa, Iowa City, Iowa, May 21 - June 4, 1980.

10. Gonzaga, C., Polak, E. and Trahan, R., "An Improved Algorithm for a Class of Optimization Problems with Functional Inequality Constraints," Electronics Research Laboratory, Memo No. UCB/ERL - M78/56, University of California, Berkeley, 1978.

11. Bhatti, M.A., Polak, E. and Pister, K.S., "OPTDYN - A General Purpose Optimization Program for Problems With or Without Dynamic Constraints," Report No. UCB/EERC-79/16, Earthquake Engineering Research Center, University of California, Berkeley, July 1979.

12. Weslander, J. and Elmqvist, H., "INTRAC - A Communication Module for Interactive Programs," Department of Automatic Control, memo LUTFD2/(TFRT-3149)/1-060/(1978), Lund Institute of Technology, Sweden, August 1978.

13. Riahi, A., Row, D.G., and Powell, G.H., "Three Dimensional Inelastic Frame Elements for the ANSR-I Program," Report No. UCB/EERC-78/06, Earthquake Engineering Research Center, University of California, Berkeley, CA., August 1978.

14. Haug, E.J. and Arora, J.S., "Applied Optimal Design: Mechanical and Structural Systems," John Wiley and Sons, Inc. New York, 1979.

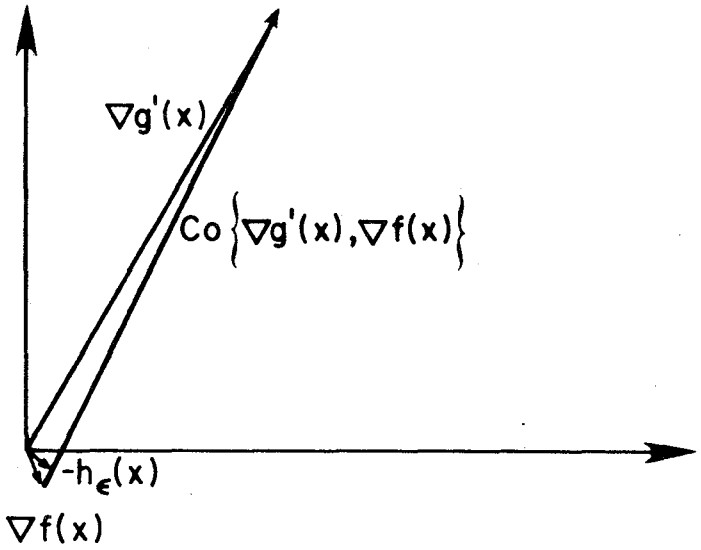


FIGURE 1 AN EXAMPLE OF THE INFLUENCE OF BAD SCALING ON THE SEARCH DIRECTION CALCULATION

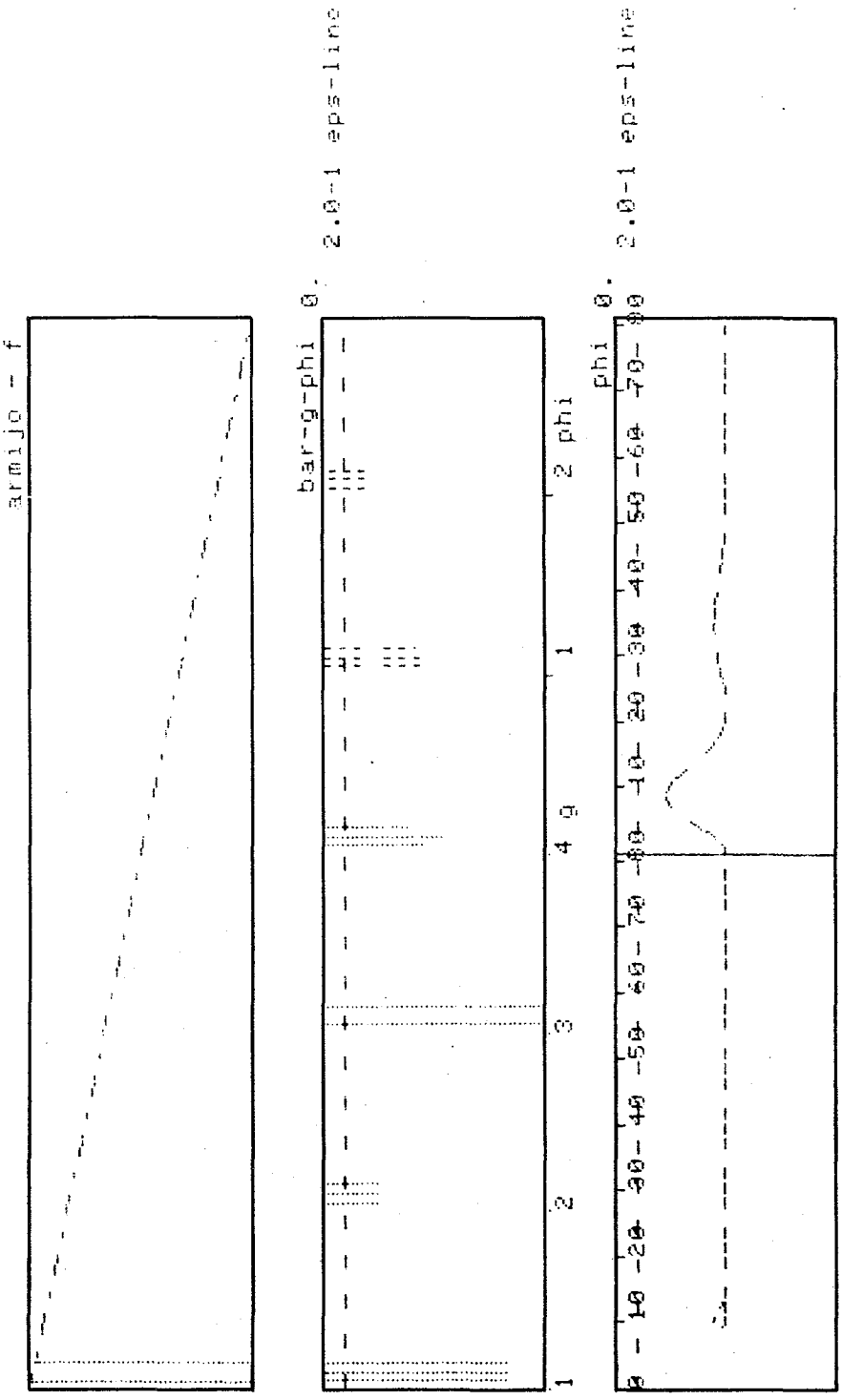


FIGURE 2 A TYPICAL PLOT OBTAINED BY USING THE MACRO GRAPHO

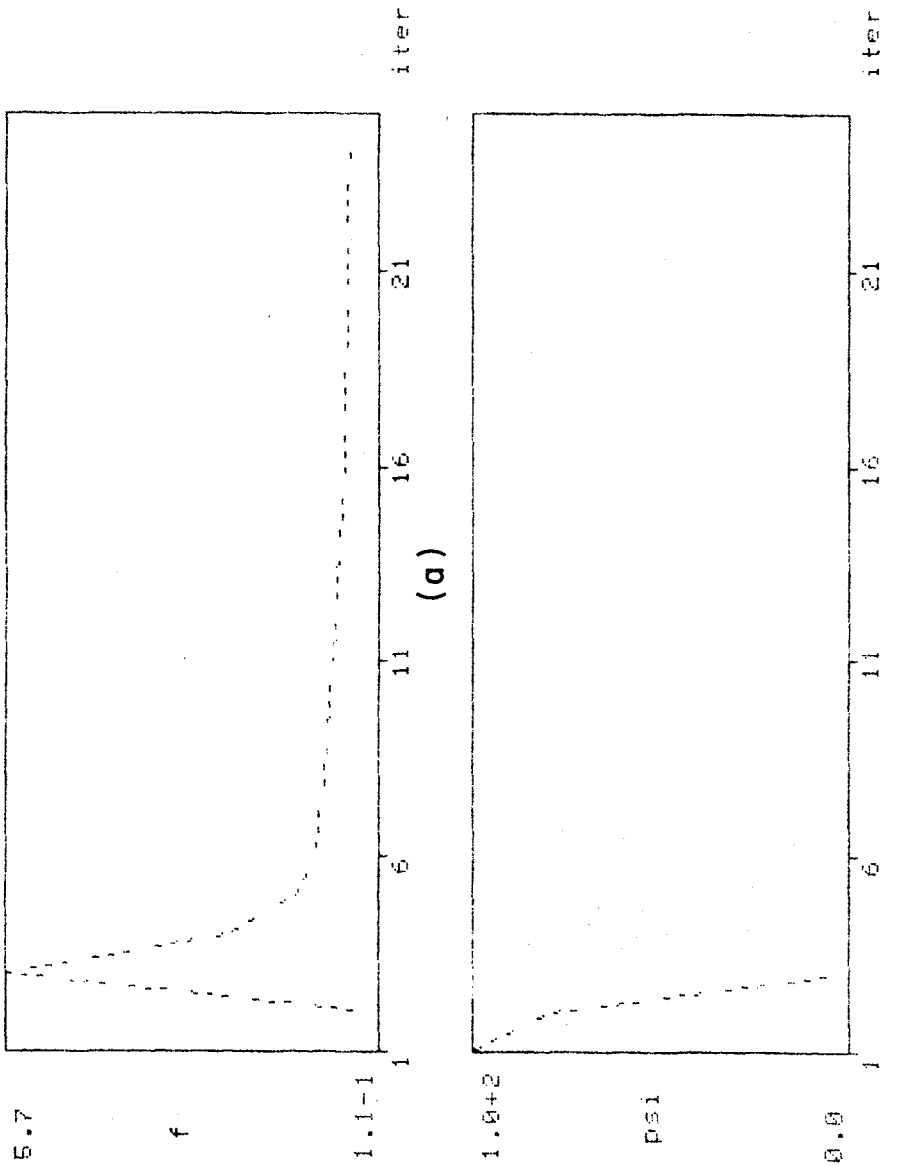


FIGURE 3 A TYPICAL PLOT CREATED BY MACROS GRAPHF AND GRAPHPSI

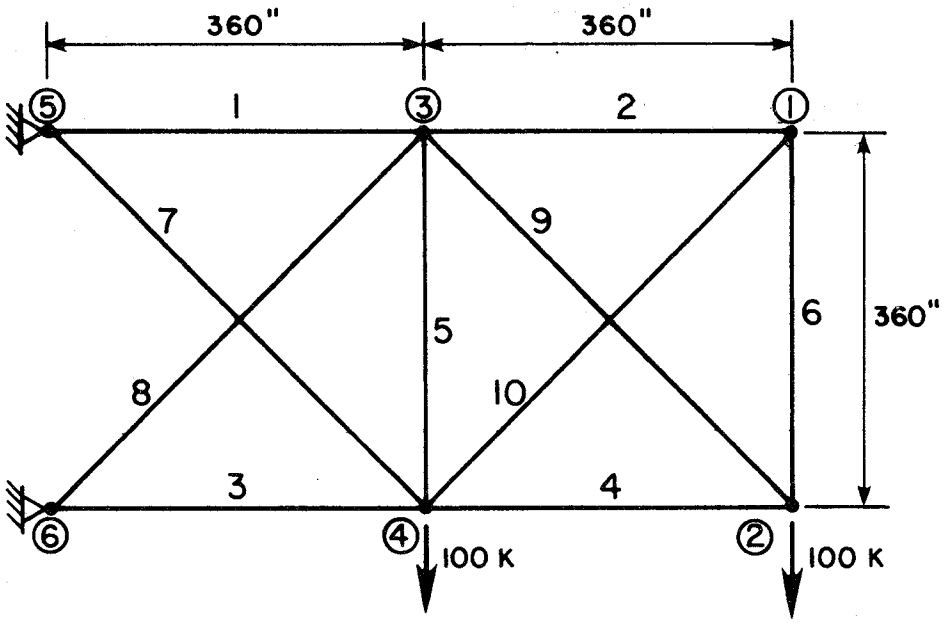


FIGURE 4 ELASTIC TRUSS FOR EXAMPLE 5.1

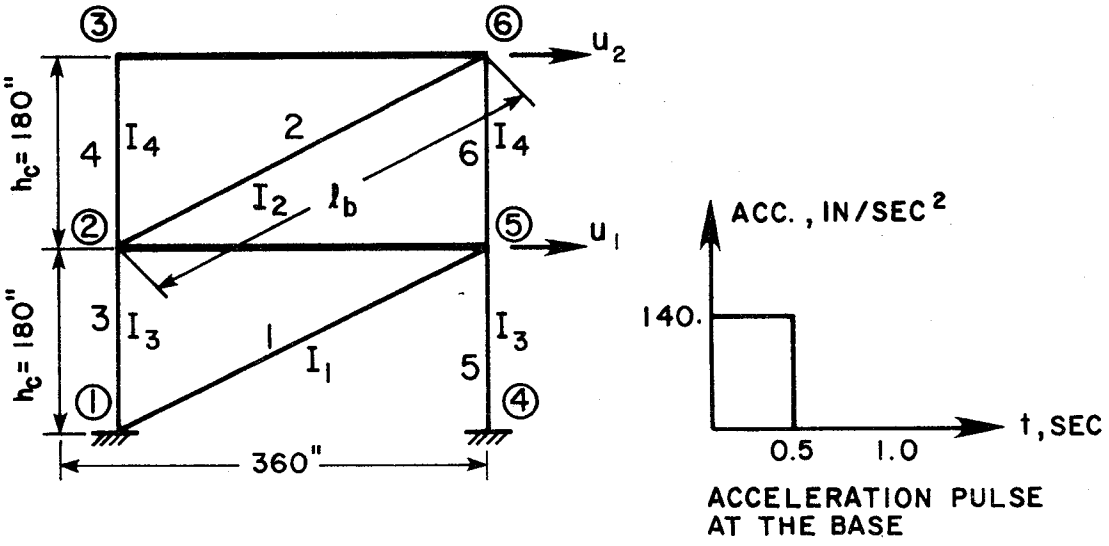


FIGURE 5 INELASTIC BRACED FRAME FOR EXAMPLE 5.2

APPENDIX A - SUMMARY OF ORIGINAL INTRAC COMMANDS

Original INTRAC commands are summarized here. For more details, see [12].

1. MACRO <macro identifier>[<formal argument>|<delimiter>|
   <termination marker>]

   Begins a macro definition and creates a macro.

2. FORMAL {<formal argument>|<delimiter>|
   <termination marker>}

   Declares formal arguments in a macro definition and when creating a macro. It can be used to extend the list of formal arguments anywhere in a macro.

3. END

   Ends a macro and ends macro creation mode. Deactivates suspended macros.

4. LET {<variable>=} {<number>[{+|-|*|/} <number>]
   |{+|-} <number>
   |<identifier>[+<integer>]
   |<delimiter>
   |<unassigned variable>

   Assigns (allocates) variables.

   Examples: LET A = B = 0 results A = B = 0 (integer)

   LET P = 3*5.5 results P = 16.5

5. DEFAULT {<variable>=} <argument>

   Assigns a variable if it is unassigned or does not exist previously.

6. LABEL <label identifier>

   Defines a label.

   Examples: LABEL SKIP

             LABEL 3

7. GOTO <label identifier>

   Makes unconditional jump.

   Example: GOTO SKIP

8. IF <argument> {EQ|NE|GE|LE|GT|LT} <argument>
   GOTO <label identifier>

   Makes conditional jump.

   Example: IF A GT 2.5 GOTO SKIP

9. FOR <variable> = <number> TO <number> [STEP <number>]

   Starts a loop.

   Example: FOR I = 1 TO FINISH STEP INCR

10. NEXT <variable>

    Ends a loop.

    Example: NEXT I

11. WRITE [([DIS|TP|LP] [FF|LF])] [<variable>|<string>]

    Writes variables and text strings or displays currently available variables. Default output is DIS (display)

    TP = Terminal Printer, LP = Line Printer, FF = Form Feed, LF = Line Feed

12. READ {{<variable> {INT|REAL|NUM|NAME|DELIM|YESNO}
    <termination marker>}

    Reads values for variables from the terminal.

13. SUSPEND

Suspends the execution of a macro.

14. RESUME

Resumes the execution of a macro.

15. SWITCH {EXEC|ECHO|LOG|TRACE} {ON|OFF}

Modifies switches in Intrac.

The switches have the following meaning.

EXEC: Determines whether the commands entered in generation mode should be executed or not.

ECHO: If ECHO is ON, the commands in a macro are echoed on the terminal as they are executed.

LOG: Determines whether the executed commands should be logged on the line printer or not.

TRACE: If TRACE is OFF only application commands are echoed and logged. Also macro calls and Intrac statements are output if TRACE is ON.

All switches have the default value OFF.

16. FREE {{<global variable>} .*}

Deallocates global variables.

17. STOP

Stops the execution of the program.

APPENDIX B - SUMMARY OF EDITOR COMMANDS

In order to write and modify macros during execution, a text editor is included in the package. It is called by using the command 'ED'. A summary of the more useful commands is given here.

Name	Abbr	Description	Examples
(.)append	a	Begins text input mode, adding lines to the buffer after the line specified. Appending continues until "." is typed alone at the beginning of a new line, followed by a carriage return. *0a* places lines at the beginning of the buffer.	:a Three lines of text are added to the buffer after the current line. . :
(.,.)change	c	Deletes indicated line(s) and initiates text input mode to replace them with new text which follows. New text is terminated the same way as with *append*.	:5,6c Lines 5 and 6 are deleted and replaced by these three lines. . :
(.,.)delete	d	Removes lines from the buffer and prints the current line after the deletion.	:13,15d New current line is printed :
ed *file*	e	Clears the editor buffer and then copies into it the named *file*, which becomes the current file. This is a way of shifting to a different file without leaving the editor. The editor issues a warning message if this command is used before saving changes made to the file already in the buffer; repeating the command overrides this protective mechanism.	:e ch10 No write since last change :e ch10 "ch10" 3 lines, 62 characters :

Name	Abbr	Description	Examples
file *name*	f	If followed by a *name*, renames the current file to *name*. If used without *name*, prints the name of the current file.	:f ch9 "ch9" [Modified] 3 lines ... :f "ch9" [Modified] 3 lines ... :
(1,$)global (1,$)global!	g g! or v	global/*pattern*/*commands* Searches the entire buffer (unless a smaller range is specified by line-number prefixes) and executes *commands* on every line with an expression matching *pattern*. The second form, abbreviated either g! or v, executes *commands* on lines that *do not* contain the expression *pattern*.	:g/nonsense/d :
(.)insert	i	Inserts new lines of text immediately before the specified line. Differs from *append* only in that text is placed before, rather than after, the indicated line. In other words, 1i has the same effect as 0a.	:1i These lines of text will be added prior to line 1. . :
(.,.+1)join	j	Joins lines together, adjusting white space (spaces and tabs) as necessary.	:2,5j Resulting line is printed :
(.,.)move*addr*	m	Moves the specified lines to a position after the line indicated by *addr*	:12,15m 25 New current lines is printed :
(.,.)print	p	Prints the text of line(s).	:+2,+3p The second and third lines after the current line :

Name	Abbr	Description	Examples
quit	q	Ends the editing session. You will receive a warning if you have changed the buffer since last writing its contents to the file. In this event you must either type w to write, or type q again to exit from the editor without writing to the file.	:q No write since last change :q %
(.)read *file*	r	Places a copy of *file* in the buffer after the specified line. Address 0 is permissible and causes the copy of *file* to be placed at the beginning of the buffer. The *read* command does not erase any text already in the buffer. If no line number is specified, *file* is placed after the current line.	:Or newfile "newfile" 5 lines, 86 characters :
(.,.)substitute	s	substitute/*pattern*/*replacement* substitute/*pattern*/*replacement*/gc Replaces the first ocurrence of *pattern* on a line with *replacement*. Including a g after the command changes all occurrence of *pattern* on the line. The c option allows the user to confirm each substitution before it is made; see the manual for details.	:3p Line 3 contains a misstake :s/misstake/mistake/ Line 3 contains a mistake :
undo	u	Reverses the changes made in the buffer by the last buffer-editing command. Note that this example contains a notification about the number of lines affected.	:1,15d 15 lines deleted new line number 1 is printed :u 15 more lines in file ... old line number 1 is printed :

Name	Abbr	Description	Examples
(1,$)write *file* (1,$)write! *file*	w w!	Copies data from the buffer onto a permanent file. if no *file* is named, the current file-name is used. The file is automatically created if it does not yet exist. A response containing the number of lines and characters in the file indicates that the write has been completed successfully. The editor's built-in protections against over-writing existing files will in some circumstances inhibit a write. The form w! forces the write, confirming that an existing file is to be overwritten.	:w "file7" 64 lines, 1122 characters :w file8 "file8" File exists ... :w! file8 "file8" 64 lines, 1122 characters
(.)z *count*	z	Prints a screen full of text starting with the line indicated; or, if *count* is specified, prints that number of lines. Variants of the z command are described in the manual.	
/*pattern*/		Searches for the next line in which *pattern* occurs and prints it.	:/This pattern/ This pattern next occurs here. :
//		Repeats the most recent search.	:// This pattern also occurs here. :
?*pattern*?		Searches in the reverse direction for *pattern*.	
??		Repeats the most recent search, moving in the reverse direction through the buffer.	

APPENDIX C - INPUT DATA FOR INTEROPTDYN

1. PROBLEM HEADING (20 A4) - one card*

COLUMNS	NOTE	VARIABLE	DESCRIPTION OF DATA ENTRY
1 - 80		HED	Problem heading to be printed with output.

2. CONTROL INFORMATION (4I5) - one card

COLUMNS	NOTE	VARIABLE	DESCRIPTION OF DATA ENTRY
1 - 5	(1)	MAXITN	Maximum number of iterations allowed.
6 - 10	(2)	ITER	Iteration number at start of this run. Leave blank if this is the first run.
11 - 15		NCUT	Maximum number of simplex iterations in solving the quadratic programming problem for direction finding.
16 - 20		ITRSTP	Maximum number of iterations allowed in step length calculations.

3. CONVERGENCE TOLERANCE PARAMETERS (8F 10.0) - one card

COLUMNS	NOTE	VARIABLE	DESCRIPTION OF DATA ENTRY
1 - 10		MU1	Parameter μ_1 used in tolerance test on ε.
11 - 20		MU2	Parameter μ_2 used in step 4 of the algorithm.
21 - 30		DELTA	Parameter δ used in step 2 (convergence check) and step 6 (step length calculations).
31 - 40		EO	ε_0, initial value of ε.
41 - 50		GAMMA	Parameter γ, used in QP.

*Here and in the sequel "card" is understood to mean a line of an input file.

4. PROBLEM SIZE (3I5) - one card

COLUMNS	NOTE	VARIABLE	DESCRIPTION OF DATA ENTRY
1 - 5		JP	Number of conventional inequality constraints (functions 'g').
6 - 10		JQ	Number of dynamic constraints (functions ϕ).
11 - 15		N	Number of optimization variables.

5. ARMIJO PARAMETERS (8F 10.0) - one card

COLUMNS	NOTE	VARIABLE	DESCRIPTION OF DATA ENTRY
1 - 10		STPMAX	Parameter controlling maximum value of step length at any iteration.
11 - 20		ALPHA	Parameter α.
21 - 30		BETA	Parameter β.
31 - 40	(3)	OLDSTP	Initial value for the step length

6. FUNCTIONAL CONSTRAINT PARAMETERS (2I5, 2F 10.0) - one card
(Skip this section if JQ is zero).

COLUMNS	NOTE	VARIABLE	DESCRIPTION OF DATA ENTRY
1 - 5		NQ	Initial number of discretization points.
6 - 10		NQMAX	Maximum number of discretization points.
11 - 20		WO	t_0 defining the interval of interest, $[t_0, t_f]$.
21 - 30		WC	t_f defining the interval of interest, $[t_0, t_f]$.

7. SCALING FACTORS (2F 10.0) - one card

COLUMNS	NOTE	VARIABLE	DESCRIPTION OF DATA ENTRY
1 - 10	(4)	SCALE	Scale factor, η, used in scaling QP.
11 - 20		PUSHF	Scale factor for cost function.

8. PUSH-OFF FACTORS FOR CONVENTIONAL CONSTRAINTS (8F 10.0)
(Skip this section if JP is zero)

As many cards as needed to specify push-off factors for all conventional inequality constraint functions.

9. PUSH-OFF FACTORS FOR DYNAMIC CONSTRAINTS (8F 10.0)
(Skip this section if JQ is zero)

As many cards as needed to specify push-off factors for all dynamic constraints.

10. INITIAL VALUES OF VARIABLES (8F 10.0)

As many cards as needed to specify initial values for N optimization variables.

NOTES

(1) The program will stop normally if either the number of iterations reaches MAXITN or the optimal solution is achieved.

(2) ITER is used only to label the output. In a number of practical situations it is not possible to let the program run for too many iterations. The process can be restarted with the latest values of the optimization variables, ε and q with ITER equal to the number of the next iteration. The output will then be labeled starting from ITER and incrementing it by one, after each subsequent iteration.

(3) The step length calculations start by assuming an initial trial value equal to OLDSTP. If a good estimate is available, it will accelerate the step length computation process.

(4) The "push-off" factors are used to force the direction vector away from or toward a constraint. Some experience is needed

before arriving at suitable values. The angles between the direction vector and objective function gradient and active constraint gradients should be used as guidelines.

STARTING PARAMETER VALUES

The following parameter values have been found to give fairly efficient behavior. Users with no prior experience can start the program with these values.

NCUT = 20 ITRSTP = 10

μ_1 = 1.0 μ_2 = 0.01 δ = 0.001

ε_0 = 0.2 γ = 2.0 STPMAX = 100.0

α = 0.2 β = 0.3 OLDSTP = 1.0

η = 0.0

PUSHF = 1.0

PUSHG = 1.0, 1.0 . . . (JP values)

PUSHPH = 1.0, 1.0 . . . (JQ values)

APPENDIX D - INPUT DATA FOR MINI-ANSR

A. PROBLEM INITIATION AND TITLE (A5, 18A4)

Columns 1 - 5: Punch the word START

6 - 77: Problem title, to be printed with output.

B. NODE INFORMATION

B1. CONTROL INFORMATION (8I5) - One card

Columns 1 - 5: Total number of nodes.

6 - 10: Number of "control" nodes, for which coordinates are specified directly (NCNOD). See Section B2.

11 - 15: Number of coordinate generation commands (NODGC). See Section B3.

16 - 20: Number of commands specifying nodes with zero displacements (NDCON). See Section B4.

21 - 25: Number of commands specifying nodes with equal displacements (NIDDOF). See Section B5.

26 - 30: Number of commands specifying nodal masses (NMSGC). See Section B6.

31 - 35: Number of element groups (NELGR, max. 20). See Section G.

40: Execution code (KEXEC) as follows.
(a) zero or blank: ful execution.
(b) 1: data checking only.

B2. CONTROL NODE COORDINATES (I5, 3F10.0) - NCNOD cards

Columns 1 - 5: Node number, in any sequence.

6 - 15: X coordinate.

16 - 25: Y coordinate.

26 - 35: Z coordinate.

B3. COORDINATE GENERATION (4I5, F10.0, 10I5) - NODGC cards

Columns 1 - 5: Node number at beginning of generation line. This must either be a control node, or must have been generated by a previous generation command.

Columns 6 - 10: Node number at end of generation line. This node must also have been specified previously.

11 - 15: Number of nodes to be generated along line. If the nodes to be generated are listed in Columns 31 - 80, this number may not exceed 10.

16 - 20: Node number difference between successive generated nodes, and between first generated node and node at beginning of generation line. May be negative. Leave blank if generated nodes are listed in Columns 31 - 80.

21 - 30: Spacing between nodes, as follows.
(a) Zero or blank: generated nodes are spaced uniformly along the generation line.
(b) Less than 1.0: spacing between nodes is this proportion of the length of the generation line.
(c) 1.0 or larger: spacing between nodes is equal to this distance.

31 - 80: Up to 10 fields, each I5. List nodes to be generated, in sequence along generation line. Required only if Columns 16 - 20 are blank.

Note: It is not necessary to provide coordinate generation commands for nodes which are sequentially numbered between the beginning and end nodes of any straight line, and which are equally spaced along that line. After all generation commands have been executed, the coordinates for each group of unspecified nodes are automatically generated assuming sequential numbering and equal spacing along a line joining the specified nodes immediately preceding and following the group. That is, any generation command with a node number difference of one and equal spacing is superfluous.

B4. NODES WITH ZERO DISPLACEMENTS (16I5) - NDCON cards

Columns 1 - 5: Node number, or number of first node in a series of nodes covered by this command. See Note following for repetition of nodes.

10: Constraint code for X displacement, as follows.
(a) Zero or blank: displacement, not constrained to be zero.
(b) 1: displacement constrained to be zero.

15: Code for Y displacement

20: Code for Z displacement

25: Code for XX rotation.

30: Code for YY rotation.

35: Code for ZZ rotation.

36 - 40: Number of last node in series of nodes covered by this command. Leave blank or punch zero for a single code, or if the nodes in the series are listed in Columns 51 - 80.

41 - 45: Node number difference between successive nodes in series. Leave blank for a single node, or if the nodes in the series are listed in Columns 51 - 80.

46 - 50: Number of nodes listed in Columns 51 - 80, following. This list is considered only if Columns 36 - 40 are blank or zero. Leave blank for a single node.

51 - 80: Up to 6 fields, each I5. List second, etc. nodes of series.

Note: If constraint codes are specified more than once for any node, the last specified value is assumed. For plane or axisymmetric problems, the first command should cover all nodes and should constrain all except the relevant displacements. Additional cards to modify the constraint codes at particular nodes should then be added.

B5. NODES WITH EQUAL DISPLACEMENTS (16I5) - NIDDOF cards

Columns 5: Equal displacement code for X displacement, as follows.
(a) Zero or blank: displacement not contrained to be identical.
(b) 1: displacement constrained to be identical for all nodes in group.

10: Code for Y displacement.

15: Code for Z displacement.

20: Code for XX rotation.

25: Code for YY rotation.

30: Code for ZZ rotation.

31 - 35: Number of nodes in group.

36 - 80: Up to 9 fields, each I5. List nodes in group. The first node must be the smallest numbered node in the group. See Note following.

Note: If the group has more than thirteen nodes, specify the remaining nodes on additional equal displacement commands. The smallest numbered node in the group must be the first node in the list for all commands defining the group. Greater computational efficiency may be obtained by constraining nodes to have equal displacements. However, the effect of specifying equal displacements may be to increase the band width of the structure stiffness matrix. This may result in an increase in the required stiffness matrix storage and/or the computational effort required to solve the equations of motion. Equal displacements specifications should therefore be used with caution. It should be noted that the equation solver used in the program is less sensitive to local increases in the stiffness matrix band width than a conventional equation solver based on a banded storage scheme.

B6. NODAL MASSES (I5, 6F10.0, 2I5) - NMSGC cards

Columns 1 - 5: Node number, or number of first node in a series of nodes covered by this command.

6 - 15: Mass associated with X-displacement degree of freedom.

16 - 25: Mass associated with Y-displacement degree of freedom.

26 - 35: Mass associated with Z-displacement degree of freedom.

36 - 45: Mass associated with X-rotation degree of freedom.

46 - 55: Mass associated with Y-rotation degree of freedom.

56 - 65: Mass associated with Z-rotation degree of freedom.

66 - 70: Number of last node in series of nodes covered by this command. Leave blank for a single node.

71 - 75: Node number difference between successive nodes in series. Leave blank for a single node.

Note: The specification commands for lumped masses will generally permit the user to input the nodal masses with only a few data cards. Any node may, if desired, appear in more than one specification command. In such cases the mass associated with any degree of freedom will be the sum of the masses specified in separate commands. If certain nodes are constrained to have an equal

displacement, the mass associated with this displacement will be the sum of the masses specified for the individual nodes. If a mass is specified for any degree of freedom that is constrained to be zero, it is ignored.

C. LOAD SPECIFICATION

C1. CONTROL CARD (8I5, 3F10.0) - One card

Columns 1 - 5: Code for static and/or dynamic analysis, (KSTAT).
(a) Zero or blank: dynamic analysis only.
(b) 1: static analysis followed by dynamic analysis.
(c) -1: static analysis only.

6 - 10: Number of static force patterns to be specified (NSPAT). See Section C2. If blank or zero, no static loads will be applied.

11 - 15: Number of static force application commands (NSLGC). See Section D.

20: Code for ground motion records (IGM), as follows.
(a) Zero or blank: no ground motion records.
(b) 1: ground motion records will be specified. See Section C3.

21 - 25: Number of dynamic force records to be specified (NDLR). See Section C4.

26 - 30: Largest number of points on any dynamic force record. This number is used for storage allocation.

31 - 35: Number of commands defining points of application of dynamic force records (NDLGC). See Section C5.

36 - 40: Number of integration time steps to be considered in dynamic analysis.

41 - 50: Integration time step, Δt.

51 - 60: Integration method parameter, δ, in Newmark's $\beta - \gamma - \delta$ method.

61 - 70: Integration method parameter, β, in Newmark's $\beta - \gamma - \delta$ method. If zero or blank, β is assumed to be equal to $0.25\ (1 + \delta)^2$.

C2. STATIC LOAD PATTERNS - NSPAT sets of cards as follows.

Each set consists of a control card followed by as many cards as needed to define the nodal loads. Load patterns are assumed to be input in numerical sequence.

C2(a) CONTROL CARD (I5, 18A4)

Columns 1 - 5: Number of nodal load commands for this pattern (NSLC).

6 - 77: Load pattern title, to be printed with output.

C2(b) NODAL LOADS (I5, 6F10.0, 2I5) - NSLC cards

Columns 1 - 5: Node number, or number of first node in a series of nodes covered by this command.

6 - 15: Load in X-direction, positive in positive direction of X-axis.

16 - 25: Load in Y-direction, positive in positive direction of Y-axis.

26 - 35: Load in Z-direction, positive in positive direction of Z-axis.

36 - 45: Moment about X-axis, positive by right hand screw rule.

46 - 55: Moment about Y-axis, positive by right hand screw rule.

56 - 65: Moment about Z-axis, positive by right hand screw rule.

66 - 70: Number of last node in series. Leave blank for a single node.

71 - 75: Node number difference between successive nodes in series. Leave blank for a single node, or if node number difference equals one.

C3. GROUND MOTION (ACCELERATION) RECORDS.

Omit if IGM, Section C1, is zero or blank. Accelerations are assumed to be in acceleration units, not as multiples of the acceleration due to gravity.

C3(a) CONTROL CARD (4I5, 6F10.0) - One card

Columns 1 - 5: Number of time points defining ground motion record in X-direction (NIPX). Leave blank or punch zero for no ground motion in this direction.

6 - 10: Number of time points defining ground motion record in Y-direction (NIPY). Leave blank or punch zero for no ground motion in this direction.

11 - 15: Number of time points defining ground motion record in Z-direction (NIPZ). Leave blank or punch zero for no ground motion in this direction.

16 - 20: Print code, as follows
(a) Zero or blank: records are not printed.
(b) 1: records are printed as input and scaled.
(c) -1: records are printed as input, scaled and interpolated at time step intervals.

21 - 30: Input time interval for X-ground motion. If blank or zero, both time and acceleration values must be input; otherwise only acceleration values must be input, the time being automatically determined. See Section C3(b).

31 - 40: Input time interval for Y-ground motion. If blank or zero, both time and acceleration values must be input; otherwise only acceleration values must be input. See Section C3(c).

41 - 50: Input time interval for Z-ground motion. If blank or zero, both time and acceleration values must be input; otherwise only acceleration values must be input. See Section C3(d).

51 - 60: Scale factor by which X-ground accelerations are to be multiplied.

61 - 70: Scale factor by which Y-ground accelerations are to be multiplied.

71 - 80: Scale factor by which Z-ground accelerations are to be multiplied.

C3(b) X RECORD - One card followed by as many cards as needed.

Omit if NIPX is blank or zero.

(i) FIRST CARD (15A4, 5A4)

Columns 1 - 60: Record title, to be printed with output.

61 - 80: Input format to read NIPX points defining the record. For example, if the format is 12F6.0, punch (12F6.0).

(ii) FOLLOWING CARDS

As many cards as needed to specify NIPX input points, with the format defined in Columns 61 - 80 of the first card. If both time and acceleration values are input, the time must immediately precede the corresponding acceleration.

C3(c) Y RECORD - One card followed by as many cards as needed.

Omit if NIPY is blank or zero.

(i) FIRST CARD (15A4, 5A4)

Columns 1 - 60: Record title, to be printed with output.

61 - 80: Input format to read NIPY points defining the record.

(ii) FOLLOWING CARDS

As many cards as needed to specify NIPY input points, with the format defined in Columns 61 - 80 of the first card.

C3(d) Z RECORD - One card followed by as many cards as needed.

Omit if NIPZ is blank or zero.

(i) FIRST CARD (15A4, 5A4)

Columns 1 - 60: Record title, to be printed with output.

61 - 80: Input format to read NIPZ points defining the record.

(ii) FOLLOWING CARDS

As many cards as needed to specify NIPZ input points, with the format defined in Columns 61 - 80 of the first card.

Note: The acceleration scale factor may be used to increase or decrease the accelerations, or to convert from multiples of the acceleration due to gravity to acceleration units.

C4. DYNAMIC FORCE RECORDS - NDLR sets of cards, as follows.

Each set consists of one card followed by as many cards as needed to define the record. Records are assumed to be numbered in sequence as input.

C4(a) FIRST CARD (2I5, 2F10.0, 8A4, 4A4)

Columns 1 - 5: Number of time points defining record (NIPT).

6 - 10: Print code, as follows.
(a) Zero or blank: record is not printed.
(b) 1: record is printed as input and scaled.
(c) -1: record is printed as input and scaled and as interpolated at time step intervals.

11 - 20: Input time interval. If blank or zero, both time and force values must be input; otherwise only force values.

21 - 30: Scale factor by which force values are to be multiplied.

31 - 62: Record title, to be printed with output.

63 - 80: Input format to read points defining the record.

C4(b) FOLLOWING CARDS

As many cards as needed to specify NIPT input points, with the format defined in Columns 63 - 80 of the first card. If both time and force values are input, the time must immediately precede the corresponding force.

C5. DYNAMIC FORCE APPLICATION (16I5) - NDLGC Cards (See Section C1)

Acceleration records, if specified, are applied automatically, assuming all support points to move in phase. Force records are applied as defined by the cards of this section.

Columns 1 - 5: Dynamic force record number.

10: Direction code, as follows.
(a) 1: X translation.
(b) 2: Y translation.
(c) 3: Z translation.
(d) 4: X rotation.
(e) 5: Y rotation.
(f) 6: Z rotation.

11 - 80: Up to 14 fields, each I5. List the nodes at which the record is to be applied. Each node in the list is subjected to the scaled force record.

Note: The dynamic forces as specified by the dynamic force record number are applied in the positive direction defined by the direction code. To apply forces in the negative direction, the scale factor by which the force values are multiplied (Section C4) should be negative.

C6. DAMPING SPECIFICATION (3F10.0) - One card

Omit if code for static and/or dynamic analysis, KSTAT (Section C1) equals -1.

Columns 1 - 10: Mass proportional damping factor, β_M.

11 - 20: Tangent stiffness proportional damping factor, β_T. See Note following.

21 - 30: Initial stiffness proportional damping factor, β_0. See Note following.

Note: If desired, it is possible to specify different values of the factors β_T and β_0 for each element group. See Section G for explanation of this option.

D. STATIC ANALYSIS SPECIFICATION - NSLGC sets of cards (See Section C1).

Each set consists of a solution procedure card followed by one or more cards defining a linear combination of static force patterns. Each set defines an increment of static load.

D1. SOLUTION PROCEDURE CARD (8I5, 4F10.0) - One card

Columns 1 - 5: Number of equal steps in which load increment is to be applied, positive if results envelopes are not to be printed at the end of the increment, otherwise negative.

6 - 10: Iteration type, as follows.
(a) Zero or blank: Newton-Raphson iteration
(b) n: Constant stiffness iteration with alpha-constant over-relaxation, the alpha matrix being reinitialized every n iterations.

15: Type of state determination calculation to be used for constant stiffness iteration as follows:
(a) Zero or blank: path independent.
(b) 1: path dependent
Path dependent state determination is always used for Newton-Raphson iteration.

16 - 20: Stiffness reformation code, as follows.
(a) Zero or blank: stiffness used in preceding step is retained.
(b) n: stiffness is reformed every n load steps.

25: Termination code, as follows.
(a) Zero or blank: If the solution does not converge within the maximum number of iterations for any load step, the next load step will be applied.
(b) 1: If the solution does not converge, the execution will terminate.

26 - 30: Print code, as follows.
(a) -1: results are not printed for this increment.
(b) Zero or blank: results are printed at the end of the increment only.
(c) 1: results are printed after each load step.
(d) 2: results are printed every iteration. This option should be used for debugging purposes only.

31 - 35: Maximum number of cycles of iteration within any load step.

36 - 40: Maximum number of iterations within any cycle.

41 - 50: Nodal force convergence to tolerance to be used in last step of load increment.

Columns 51 - 60: Nodal force convergence tolerance to be used in all except last step of load increment.

61 - 70: Nodal force tolerance for change of stiffness in Newton-Raphson iteration. If the unbalanced force reduces below this tolerance, the stiffness will not be reformed for the next iteration.

71 - 80: Maximum nodal displacement (translation or rotation) increment permitted in any iteration step. Leave blank for unlimited displacement. Displacement limits should be specified only with Newton-Raphson iteration.

D2. FOLLOWING CARDS (8F10.0) - As many cards as needed

Columns 1 - 80: Up to eight fields, each F10.0. For each static force pattern in turn, specify a scale factor by which the pattern is to be multiplied. The scaled patterns are added together to produce the load increment.

Scale factors may be positive or negative. Leave the corresponding field blank or punch zero to ignore any force pattern.

E. DYNAMIC ANALYSIS SPECIFICATION

E1. DYNAMIC SOLUTION PROCEDURE CARD (7I5, 4F10.0, I5) - One card

Omit if KSTAT (Section C1) equals -1.

Columns 1 - 5: Iteration type, as follows.
(a) Zero or blank: Newton-Raphson iteration.
(b) n > 0: Constant stiffness iteration with alpha-constant over-relaxation, the alpha matrix being reinitialized every n iterations.

10: Type of state determination calculation to be used for constant stiffness iteration, as follows.
(a) Zero or blank: path independent.
(b) 1: path dependent.
Path dependent state determination is always used for Newton-Raphson iteration.

15: Stiffness reformation code, as follows.
(a) Zero or blank: stiffness used in preceding time step is retained.
(b) n: stiffness is reformed every n time steps.

Columns 20: Termination code, as follows.
(a) Zero or blank: if the solution does not converge within the maximum number of iterations for any time step, the next time step will be applied.
(b) 1: if the solution does not converge, the execution will terminate.

21 - 25: Maximum number of cycles of iteration within any time step.

26 - 30: Maximum number of iterations within any cycle.

31 - 35: Number of time steps between application of "fine" convergence tolerence. The "coarse" tolerance is used at intermediate steps.

36 - 45: "Fine" nodal force convergence tolerance.

46 - 55: "Coarse" nodal force convergence tolerance.

56 - 65: Nodal force tolerance for change of stiffness in Newton-Raphson iteration. If the unbalanced force reduces below this tolerance, the stiffness will not be reformed for the next iteration.

66 - 75: Maximum nodal displacement (translation or rotation) increment permitted in any iteration step. Leave blank for unlimited displacement. Displacement limits should be specified only with Newton-Raphson iteration.

76 - 80: Number of initial condition generation commands (NICGC). See Section E2.

E2. INITIAL CONDITION SPECIFICATION (I5, 2F10.0, 11I5) - NICGC cards (See Section E1).

Columns 1 - 5: Direction code, as follows.
(a) 1: X translation
(b) 2: Y translation
(c) 3: Z translation
(d) 4: X rotation
(e) 5: Y rotation
(f) 6: Z rotation

6 - 15: Initial velocity.

16 - 25: Initial acceleration.

26 - 80: Up to 11 fields, each I5. List up to 11 nodes having the same initial conditions.

F. OUTPUT SPECIFICATION

This set of cards consists of a control card followed by as many cards as needed to specify node numbers for output. See Note following.

F1. CONTROL CARD (10I5, 7A4) - One card

Columns 1 - 5: Time interval for printout of nodal displacement, velocity and acceleration time histories, expressed as a multiple of the integration time step. Leave blank or punch zero for no time history output or if there is not dynamic analysis.

6 - 10: Time interval for printout of element action time histories (stresses, forces, etc.) expressed as a multiple of the integration time step. Leave blank or punch zero for no time history output or if there is no dynamic analysis.

11 - 15: Time interval for printout of intermediate envelopes of nodal displacements and element actions, expressed as a multiple of the integration time step. Leave blank or punch zero for no intermediate envelope output or if there is no dynamic analysis. Envelopes are automatically output at the end of the dynamic analysis.

16 - 20: Number of nodes for X-displacement, velocity and acceleration output (NODSX). For output at all nodes, punch -1.

21 - 25: Number of nodes for Y-displacement, velocity and acceleration output (NODSY). For output at all nodes, punch -1.

26 - 30: Number of nodes for Z-displacement, velocity and acceleration output (NODSZ). For output at all nodes, punch -1.

F2. FOLLOWING CARDS - THREE SETS OF CARDS, AS FOLLOWS.

(1) List of nodes for X response printout (16I5) - As many cards as needed to specify NODSX number of nodes, sixteen to a card. Omit if NODSX equals zero or -1.

(2) List of nodes for Y response printout (16I5) - As many cards as needed to specify NODSY number of nodes, sixteen to a card. Omit if NODSY equals zero, or -1.

(3) List of nodes for Z response printout (16I5) - As many cards as needed to specify NODSZ number of nodes, sixteen to a card. Omit if NODSZ equals zero, or -1.

Note: Results for the same nodes and elements are printed for both static and dynamic analyses, except that velocities and accelerations are not printed for static analyses.

Envelope values are printed for the dynamic analysis, and may be printed at the end of each static load increment if so specified on Card D1.

G. ELEMENT SPECIFICATION

Element must be divided into "groups". All elements in any group must be of the same type. However, elements of the same type may be divided into separate groups if desired.

Element groups may be input in any sequence. The total number of element groups may not exceed 20. The elements in any group must be numbered sequentially, the number of the first element in the group being any convenient number.

G1. THREE DIMENSIONAL ELASTIC TRUSS ELEMENT

G1(a) CONTROL INFORMATION (10I5, 6F5.0) - One card

Columns 5: Element group indicator. Punch 1 (to indicate that the group consists of three dimensional truss elements)

6 - 10: Number of elements in this group.

11 - 15: Element number of the first element in this group. If blank or zero, assumed to be equal to 1.

16 - 20: Number of material types. If blank or zero, assumed to be equal to 1.

21 - 50: Blank (not used for this element type).

51 - 55: Initial stiffness damping factor β_0. If blank or zero, β_0 is assumed to be equal to the system β_0 value input in Card C6.

56 - 60: Current tangent stiffness damping factor, β_T. If blank or zero, β_T is assumed to be equal to the system β_T value input in Card C6.

G1(b) MATERIAL PROPERTY INFORMATION (I5, F10.0) - One card for each different material type.

Columns 1 - 5: Material number, in sequence starting with 1.

6 - 15: Young's modulus of elasticity, E.

G1(c) ELEMENT GENERATION COMMANDS (4I5, F10.0, 2I5) - As many cards as needed to generate all elements in this group.

Cards must be entered in order of increasing element number. Cards for the first and last element must be included. See Note G123 for explanation of generation procedure.

Columns 1 - 5: Element number, or number of first element in a sequentially numbered series of elements to be generated by this card.

6 - 10: Node number at element end i.

11 - 15: Node number at element end j.

16 - 20: Material number. If blank or zero, assumed to be equal to 1.

21 - 30: Cross sectional area.

31 - 35: Node number increment for element generation. If blank or zero assumed to be equal to 1.

36 - 40: Time history output code. Leave blank or punch zero for no time history output. Punch 1 if time history output is required.

G2. THREE DIMENSIONAL NONLINEAR TRUSS ELEMENT.

See [2] for description of element.

G2(a) CONTROL INFORMATION (10I5, 6F5.0) - One card

Columns 5: Element group indicator. Punch 2 (to indicate that the group consists of three dimensional truss elements).

Columns 6 - 10: Number of elements in this group.

11 - 15: Element number of the first element in this group. If blank or zero, assumed to be equal to 1.

16 - 20: Number of material types. If blank or zero, assumed to be equal to 1.

21 - 50: Blank (not used for this element type).

51 - 55: Initial stiffness damping factor β_0. If blank or zero, β_0 is assumed to be equal to the system β_0 value input in Card C6.

56 - 60: Current tangent stiffness damping factor, β_T. If blank or zero, β_T is assumed to be equal to the system β_T value input in Card C6.

G2(b) MATERIAL PROPERTY INFORMATION (I5, 4F10.0) - One card for each different material type.

Columns 1 - 5: Material number, in sequence starting with 1.

6 - 15: Young's modulus of elasticity, E.

16 - 25: Strain hardening modulus as a proportion of Young's modulus (i.e. the ratio E_h/E).

26 - 35: Yield stress in tension.

36 - 45: Yield stress in compression, or elastic buckling stress in compression (input as a positive value)

G2(c) ELEMENT GENERATION COMMANDS (4I5, 2F10.0, 4I5) - As many cards as needed to generate all elements in this group.

Cards must be entered in order of increasing element number. Cards for the first and last element must be included. See Note G123 for explanation of generation procedure.

Columns 1 - 5: Element number, or number of first element in a sequentially numbered series of elements to be generated by this card.

6 - 10: Node number at element end i.

11 - 15: Node number at element end j.

Columns 16 - 20: Material number. If blank or zero, assumed to be equal to 1.

21 - 30: Cross sectional area.

31 - 40: Initial axial force on the element.

41 - 45: Node number increment for element generation. If blank or zero assumed to be equal to 1.

50: Code for large displacement effects. Leave blank or punch zero, for small displacement effects. Punch 1 for large displacement effects.

55: Time history output code. Leave blank or punch zero for no time history output. Punch 1 if time history output is required.

60: Buckling code. Leave blank or punch zero if element yields in compression without buckling. Punch 1 if element buckles elastically in compression.

G3. TWO DIMENSIONAL ELASTIC BEAM ELEMENT

G3(a) CONTROL INFORMATION (10I5, 6F5.0) - One card

Columns 5: Element group indicator. Punch 3 (to indicate that the group consists of two-dimensional elastic elements).

6 - 10: Number of elements in this group.

11 - 15: Element number of the first element in this group. If blank or zero, assumed to be 1.

16 - 20: Number of different element stiffness types (max 35).

21 - 25: Number of different end eccentricity types (max 15)*.

26 - 50: Blank

55 - 55: Initial stiffness damping factor, β_0. If blank or zero, assumed to be equal to the system β_0 value input in Card C6.

*The use of the end eccentricity option is the same as in the next element, G4. See ref. [13].

Column 56 - 60: Current tangent stiffness damping factor, β_T. If blank or zero, assumed to be equal to the system β_T value input in Card C6.

G3(b) STIFFNESS TYPES (I5, 3F10.0) - One card for each different stiffness type.

Columns 5: Stiffness type number, in sequence beginning with 1.

6 - 15: Young's modulus of elasticity.

16 - 25: Average cross sectional area.

26 - 35: Reference moment of inertia.

G3(c) END ECCENTRICITIES (I5, 4F10.0) - One card for each end eccentricity type.

Omit if there are no end eccentricities. See Fig. 2.6 in ref. [13] for explanation. All eccentricities are measured <u>from</u> the node <u>to</u> the element end, in global coordinates.

Columns 1 - 5: End eccentricity type number, in sequence beginning with 1.

6 - 15: X_i = X eccentricity at end i.

16 - 25: X_j = X eccentricity at end j.

26 - 35: Y_i = Y eccentricity at end i.

36 - 45: Y_j = Y eccentricity at end j.

G3(d) ELEMENT GENERATION COMMANDS (7I4) - As many cards as needed to generate all elements in this group.

Cards must be in order of increasing element number. Cards for the first and last elements must be included. See Note G123 for explanation of generation procedure.

Columns 1 - 4: Element number, or number of first element in a sequentially numbered series of elements to be generated by this command.

5 - 8: Node number at element end i, NODI.

Columns 9 - 12: Node number at element end j, NODJ.

13 - 16: Node number increment for element generation. If zero or blank, assumed to be 1.

17 - 24: Stiffness type number.

25 - 28: End eccentricity type number. Leave blank or punch zero if there is no end eccentricity.

29 - 32: Time history output code. If a time history of element results is not required for the element covered by this command, punch zero or leave blank. If a time history printout is required, punch 1.

NOTE G123: ELEMENT GENERATION FOR ELEMENTS 1, 2, 3

In the element generation commands, the elements must be specified in increasing numerical order. Cards may be provided for sequentially numbered elements, in which case each card specifies one element and the generation option is not used. Alternatively, the cards for a group of elements may be omitted, in which case the data for the missing group is generated as follows:

(1) All elements are assigned the same node k, strength type, etc. as for the element preceding the missing group of elements.

(2) The node numbers for each missing element are obtained by adding the specified node number increment to the node numbers of each preceding element. The node number increment is that specified for the element preceding the missing set of elements.

In the printout of the element data, generated data are prefixed by an asterisk.

G4. TWO DIMENSIONAL NONLINEAR BEAM ELEMENT

G5. THREE DIMENSIONAL NONLINEAR BEAM ELEMENT

These elements are described in detail in [13]. The input for MINI-ANSR is the same as that given in [13] for ANSR-1, with the only exception that the element group indicators are respectively 4 and 5.

H. NEW PROBLEM

Data for a new problem may follow immediately starting with Section A. Any number of structures may be analyzed in a single computer run.

I. TERMINATION CARD (A4) - One card to terminate the complete data deck.

Columns 1 - 4: Punch the word STOP.

APPENDIX E - LISTING OF FUNCTION EVALUATION SUBROUTINES FOR EXAMPLE PROBLEMS 1 AND 2

PROBLEM-INDEPENDENT SUBROUTINES

```
c     ------------------------------------------------------------------
c
      subroutine inansr
c     ------------------------------------------------------------------
c
c
c
c     common blocks for mini-ansr program
c
      implicit double precision (a-h,o-z)
      common /contrl/ nodes,ncnod,nodgc,ndcon,niddof,nmaxd,nmsgc,
     *                nelgr,ntels,kexec,kstat,kprint
      common /infelr / rdat(1000)
      common /infeli / idat(500)
      common /work   / work(2000)
      common /indat / ia(5000)
      common a(10000)
c
c     set storage
c
      nrstor = 10000
      nistor = 5000
      nrdat  = 1000
      nidat  = 500
      nwork  = 2000
c
c
c     initialize
c
      do 100 i=1,nrstor
100   a(i) = 0.
      do 110 i=1,nistor
110   ia(i) = 0
      do 120 i=1,nrdat
120   rdat(i) = 0.
      do 130 i=1,nidat
130   idat(i) = 0
      do 140 i=1,nwork
140   work(i) = 0.
c
      call input (a,nrstor,ia,nistor)
c
      kprint = 1
      call declar ('KPRINT','int',0,kprint,0,0)
      call declar ('NTELS','int',0,ntels,0,0)
      call declar ('NODES','int',0,nodes,0,0)
      return
      end
```

```
c     ------------------------------------------------------------------
c
      logical function bigdif (n, z, zstr, diff, tol)
c     ------------------------------------------------------------------
c
c     returned true if the maximum difference between z(i)
c     zstr(i) is greater than tol.  It also sets zstr(i) equal to z(i)
c     when this is true.
c
      implicit double precision (a-h, o-z)
      dimension z(1), zstr(1), diff(1)
c
      bigdif = .false.
      difmax = 0.
      do 100 i = 1, n
      diff(i) = z(i) - zstr(i)
      absdif = dabs(diff(i))
      if(z(i).ne.0.0d0) absdif = absdif/z(i)
      if (absdif .gt. difmax) difmax = absdif
100   continue
c
      if (difmax .lt. tol) return
c
      do 110 i=1, n
      zstr(i) =. z(i)
110   continue
c
      bigdif = .true.
c
      return
      end
```

```
c     ------------------------------------------------------------------
c
      subroutine anal (zresp)
c     ------------------------------------------------------------------
c
c     structural analysis interface routine
c
      implicit double precision (a-h,o-z)
      dimension zresp(1)
c
c     mini-ansr common blocks
c
      common /contrl/ nodes,ncnod,nodgc,ndcon,niddof,nmaxd,nmsgc,
     *                nelgr,ntels,kexec,kstat,kprint
      common /storag/ mrstor,mistor,neq,mband,nsto,jcol
      common /lodcon/ nspat,nslgc,nipx,nipy,nipz,kprec,dtx,dty,dtz,
     *                facgx,facgy,facgz,ndlr,mndip,ndlgc
      common /pass  / igr,naddr,naddi,kna,kedatr,kedati,kevar,
     *                istep,ipath,kupd,kitrn,ielasp,ielas,nstref
      common /addres/ kndk,kxo,kyo,kzo,kfms,kimug,ksld,ktug,kug,kgx,
     *                kgy,kgz,kpt,kipt,kxdh,kydh,kzdh,knaa,krvec,kdri,
     *                kri,kddds,kddis,kdis,kvel,kacc,kdenp,ktimp,
     *                kdenn,ktimn,kdup,kalf,kstf,kstfd,kdelk
      common /inadrs/ kvelin,kaccin
      common /tapes/ nin,nou
      common /one   / jp,jq,numvar
      common /indat / ia(1)
      common a(1)
c
      if (kexec.ge.1) go to 300
c
      call modify (zresp)
      if (kprint.eq.0) write(nou,2000) (zresp(i),i=1,numvar)
2000  format (/5x,'zresp in anal'//(5x,5(e12.5,2x)))
c
c     initialize
c
      do 100 i=krvec,mrstor
100   a(i) = 0.0
      ielasp = 1
      ielas  = 0
      nstref = 0
c
c     response analysis for static loads
c
      if (kstat.eq.0) go to 110
      nspat1 = nspat + 1
      call static (a(ksld),a(krvec),a(kri),a(kddds),a(kddis),a(kdis),
     *             a(kdenp),a(ktimp),a(kdenn),a(ktimn),ia(kxdh),
     *             ia(kydh),ia(kzdh),ia(kndk),a(kstf),a(kstfd),ia(knaa),
     *             a(kdelk),a(kdup),a(kalf),nodes,nspat1)
      if (kstat.eq.-1) go to 200
c
c     response analysis for dynamic loads
c
110   call dynmic (a(kfms),ia(kimug),a(kgx),a(kgy),a(kgz),ia(kipt),
     *             a(kpt),a(krvec),a(kri),a(kddds),a(kddis),a(kdis),
     *             a(kvel),a(kacc),a(kdenp),a(ktimp),a(kdenn),a(ktimn),
     *             ia(kxdh),ia(kydh),ia(kzdh),ia(kndk),a(kstf),a(kstfd),
     *             ia(knaa),a(kdelk),a(kdup),a(kalf),
     *             nodes,a(kvelin),a(kaccin))
c
200   return
c
300   stop
      end
```

```
c     ------------------------------------------------------------------
c
      subroutine set
c     ------------------------------------------------------------------
c
c     this subroutine sets data in common blocks to be used in the
c     interactive optimization program. As an example data are set
c     here to be used for displaying the structure geometry
c
c
      implicit double precision (a-h,o-z)
      common /contrl/ nodes,ncnod,nodgc,ndcon,niddof,nmaxd,nmsgc,
     *                nelgr,ntels,kexec,kstat,kprint
      common /storag/ mrstor,mistor,neq,mband,nsto,jcol
      common /elpar / lpar(10),flpar(6),indgr(20),nmsgr(20),mfgr(20),
     *                infgrr(20),infgri(20),ndfgr(20),dkogr(20),
     *                dktgr(20)
      common /pass  / igr,naddr,naddi,kna,kedatr,kedati,kevar,
     *                istep,ipath,kupd,kitrn,ielasp,ielas,nstref
      common /indat  / ieldat(1)
      common eldat(1)
c
      naddi = kedati
      naddr = kedatr
c
      do 260 igr=1,nelgr
c
c
      ngr  = indgr(igr)
      nels = nmsgr(igr)
c
      do 260 iel=1,nels
c
      ninfi = ieldat(naddi)
      ninfr = eldat(naddr)
c
120   go to (130,140,150,160,170,180,190,200,210,220), ngr
c
130   call stor1 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,2)
      go to 250
140   call stor2 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,2)
      go to 250
150   call stor3 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,2)
      go to 250
160   call stor4 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,2)
      go to 250
170   call stor5 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,2)
      go to 250
180   call stor6 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,2)
      go to 250
190   call stor7 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,2)
      go to 250
200   call stor8 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,2)
      go to 250
210   call stor9 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,2)
      go to 250
220   call stor10 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,2)
      go to 250
c
250   naddi = naddi + ninfi + 1
      naddr = naddr + ninfr + 1
c
260   continue
c
      return
      end
```

```
c     ------------------------------------------------------------------
c
      subroutine modify (zz)
c     ------------------------------------------------------------------
c
c     this subroutine modifies data in common blocks corresponding to the
c     optimization variables.
c
      implicit double precision (a-h,o-z)
      dimension zz(1)
      common /contrl/ nodes,ncnod,nodgc,ndcon,niddof,nmaxd,nmsgc,
     *                nelgr,ntels,kexec,kstat,kprint
      common /storag/ mrstor,mistor,neq,mband,nsto,jcol
      common /elpar / lpar(10),flpar(6),indgr(20),nmsgr(20),mfgr(20),
     *                infgrr(20),infgri(20),ndfgr(20),dkogr(20),
     *                dktgr(20)
      common /pass  / igr,naddr,naddi,kna,kedatr,kedati,kevar,
     *                istep,ipath,kupd,kitrn,ielasp,ielas,nstref
      common /indat  / ieldat(1)
      common eldat(1)
c
      kevar = ntels
      naddi = kedati
      naddr = kedatr
c
      do 260 igr=1,nelgr
c
c
      ngr  = indgr(igr)
      nels = nmsgr(igr)
c
      do 260 iel=1,nels
c
      ninfi = ieldat(naddi)
      ninfr = eldat(naddr)
c
120   go to (130,140,150,160,170,180,190,200,210,220), ngr
c
130   call mdfl1 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,zz)
      go to 250
140   call mdfl2 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,zz)
      go to 250
150   call mdfl3 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,zz)
      go to 250
160   call mdfl4 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,zz)
      go to 250
170   call mdfl5 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,zz)
      go to 250
180   call mdfl6 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,zz)
      go to 250
190   call mdfl7 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,zz)
      go to 250
200   call mdfl8 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,zz)
      go to 250
210   call mdfl9 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,zz)
      go to 250
220   call mdfl10 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,zz)
      go to 250
c
250   naddi = naddi + ninfi + 1
      naddr = naddr + ninfr + 1
c
260   continue
c
      return
      end
```

```
c     ------------------------------------------------------------------
c
      subroutine sendif (ntime,nepact,yn,y,nrow,delz,dydz,jvar,numvar)
c     ------------------------------------------------------------------
c
c
c     dimension : dydz (idim,jdim,kkdim), y (idim,kdim), yn (idim,kdim)
c
c     dydz (i,j,kk) = (yn (i,k) - y (i,k)) / delzj
c     or
c     dydz (ijkk) = (yn (ik) - y (ik)) / delzj
c     where :
c             ijkk = i + (j-1)*idim + (kk-1)*idim*jdim
c             ik = i + (k-1)*idim
c
c     here :
c             idim = nrow
c             jdim = numvar
c
      implicit double precision (a-h,o-z)
      dimension nepact(1),yn(1),y(1),dydz(1)
c
      nm = nrow*numvar
      jvarow = (jvar-1)*nrow
      do 100 i=1,nrow
      ijkk = i+jvarow
      ik = i
      do 100 k=1,ntime
      if (nepact(k).eq.0) go to 90
      dydz(ijkk) = (yn(ik)-y(ik))/delz
      ijkk = ijkk+nm
90    ik = ik+nrow
100   continue
c
      return
      end
```

PROBLEM-DEPENDENT SUBROUTINES

TRUSS PROBLEM

```
c     ****************************************************************
c     Function evaluation subroutines  for 10 bar truss problem
c     ****************************************************************
c
c     ----------------------------------------------------------------
c
      subroutine parsym (n,z)
c     ----------------------------------------------------------------
c
c
      implicit double precision (a-h,o-z)
      dimension z(1)
c
c     common blocks for 10 bar truss problem
c
      common/strpar/rleng(10),ro,adisp2,astrs2,armin,tolz,deltaz
      common /resp/ xdisp(4,1),ydisp(4,1),stress(10,1),nepact(1)
      common /dresp/ dxdz(4,10,1),dydz(4,10,1),dstrdz(10,10,1)
      common /strgom/ elndx1(10),elndx2(10),elndy1(10),elndy2(10),
     1                nod1(10),nod2(10)
      common /zansr / zresp(10), zsens(10), diff(10)
      common /nstran/ nanal,nsens
c
      nanal = 0
      nsens = 0
c
      call inansr
c
      call set
c
c
      do 100 i = 1 , n
      zresp(i) = 0.0
      zsens(i) = 0.0
100   continue
c
      ro = 0.0001
      adisp2 = 4.
      astrs2 = 625.
      armin = 0.1
      tolz = 1.0d-5
      deltaz = 1.0d-5
c
c
c     declare problem variables into intrac data base
c
      call declar ('TOLZ','double',0,tolz,0,0)
      call declar ('DELTAZ','double',0,deltaz,0,0)
      call declar ('XDISP','double',4,xdisp,4,1)
      call declar ('YDISP','double',4,ydisp,4,1)
      call declar ('STRESS','double',10,stress,10,1)
      call declar ('RO','double',0,ro,0,0)
      call declar ('ADISP2','double',0,adisp2,0,0)
      call declar ('ASTRS2','double',0,astrs2,0,0)
      call declar ('ARMIN','double',0,armin,0,0)
      call declar ('NANAL','int',0,nanal,0,0)
      call declar ('NSENS','int',0,nsens,0,0)
      call declar ('ELNDX1','double',1,elndx1,10,1)
      call declar ('ELNDX2','double',1,elndx2,10,1)
      call declar ('ELNDY1','double',1,elndy1,10,1)
      call declar ('ELNDY2','double',1,elndy2,10,1)
      call declar ('NOD1','int',1,nod1,10,1)
      call declar ('NOD2','int',1,nod2,10,1)
```

```
c
      return
      end
c
c     ------------------------------------------------------------------
c
      subroutine funcf (n,z,f,nfuncf)
c     ------------------------------------------------------------------
c
      implicit double precision (a-h,o-z)
      dimension z(1)
      common/strpar/rleng(10),ro,adisp2,astrs2,armin,tolz,deltaz
c
      f = 0.0d0
      do 100 i=1,n
      f = f + rleng(i)*z(i)
100   continue
      f = f * ro
c
      return
      end
c
c     ------------------------------------------------------------------
c
      subroutine gradf (n,z,grad)
c     ------------------------------------------------------------------
c
      implicit double precision (a-h,o-z)
      dimension z(1), grad(1)
      common/strpar/rleng(10),ro,adisp2,astrs2,armin,tolz,deltaz
c
      do 100 i=1,n
      grad(i) = ro*rleng(i)
100   continue
c
      return
      end
c
c     ------------------------------------------------------------------
c
      subroutine funcg (n,jp,z,g,psi,nfuncg)
c     ------------------------------------------------------------------
c
      implicit double precision (a-h,o-z)
      dimension z(1),g(1)
      logical bigdif
      common /zansr/ zresp(10),zsens(10),diff(10)
      common /resp/ xdisp(4,1),ydisp(4,1),stress(10,1),nepact(1)
      common /dresp/ dxdz(4,10,1),dydz(4,10,1),dstrdz(10,10,1)
      common /nstran/ nanal,nsens
      common/strpar/rleng(10),ro,adisp2,astrs2,armin,tolz,deltaz
c
      do 100 i=1,10
      g(i) = -z(i) + armin
      if (g(i) .le. psi) go to 100
      psi = g(i)
      return
100   continue
c
      if (bigdif(n,z,zresp,diff,tolz)) go to 120
      if (nsens .le. 0) go to 120
```

```
c
      do 110 j=1,n
      delz = diff(j)
      do 105 l=1,4
      xdisp(l,1) = xdisp(l,1)+dxdz(l,j,1)*delz
      ydisp(l,1) = ydisp(l,1)+dydz(l,j,1)*delz
105   continue
      do 108 l=1,10
108   stress(l,1) = stress(l,1)+dstrdz(l,j,1)*delz
110   continue
c
      go to 200
c
120   continue
      call anal(zresp)
      nanal = nanal+1
c
200   continue
      j = 0
      do 210 i=11,14
      j=j+1
      g(i) = xdisp(j,1)*xdisp(j,1)/adisp2 -1.
210   continue
c
      j = 0
      do 220 i=15,18
      j = j+1
      g(i) = ydisp(j,1)*ydisp(j,1)/adisp2 -1.
220   continue
c
      j = 0
      do 230 i=19,28
      j = j+1
      g(i) = stress(j,1)*stress(j,1)/astrs2 -1.
230   continue
c
      do 240 i=11,28
      if(g(i).gt.psi) psi=g(i)
240   continue
c
      return
      end
c
c     ------------------------------------------------------------------
c
      subroutine gradg (n,j,z,grad)
c     ------------------------------------------------------------------
c
      implicit double precision (a-h,o-z)
      dimension z(1), grad(1)
      logical bigdif
      common /zansr/ zresp(10),zsens(10),diff(10)
      common /resp/ xdisp(4,1),ydisp(4,1),stress(10,1),nepact(1)
      common /dresp/ dxdz(4,10,1),dydz(4,10,1),dstrdz(10,10,1)
      common /nstran/ nanal,nsens
      common/strpar/rleng(10),ro,adisp2,astrs2,armin,tolz,deltaz
c
c
      if (j.gt.10) go to 110
c
      do 100 i=1,n
100   grad(i) = 0.0
      grad(j) = -1.0
      return
```

```
c
110   continue
      if(.not. bigdif(n,z,zsens,diff,tolz)) go to 120
c
      nepact(1) = 1
c
      call sens(zsens,n,1)
      nsens = nsens+1
c
120   continue
      if (j.gt.18) go to 160
c
      if (j.gt.14) go to 140
      jj = j-10
      factor = 2.*xdisp(jj,1)/adisp2
      do 130 i=1,n
130   grad(i) = factor*dxdz(jj,i,1)
      return
c
140   jj = j-14
      factor = 2.*ydisp(jj,1)/adisp2
      do 150 i=1,n
150   grad(i) = factor*dydz(jj,i,1)
      return
c
160   jj = j-18
      factor = 2.*stress(jj,1)/astrs2
      do 170 i=1,n
170   grad(i) = factor*dstrdz(jj,i,1)
c
      return
c
      end
c
c     ------------------------------------------------------------------
c
      subroutine funcph (n,njq,jq,z,w0,wc,deltaw,nq,phi,psi,nfuncp)
c     ------------------------------------------------------------------
c
      return
      end
c
c     ------------------------------------------------------------------
c
      subroutine gradph
c     ------------------------------------------------------------------
c
      return
      end
c
c     ------------------------------------------------------------------
c
      subroutine sens (zz,numvar,ntime)
c     ------------------------------------------------------------------
c
      implicit double precision (a-h,o-z)
      dimension z1(10),zz(1),xdisp1(4,1),ydisp1(4,1),stres1(10,1)
      common /zansr/ zresp(10),zsens(10),diff(10)
      common /resp/ xdisp(4,1),ydisp(4,1),stress(10,1),nepact(1)
      common /dresp/ dxdz(4,10,1),dydz(4,10,1),dstrdz(10,10,1)
      common /nstran/ nanal,nsens
      common/strpar/rleng(10),ro,adisp2,astrs2,armin,tolz,deltaz
```

```
c
c
      do 110 k=1,ntime
      do 100 j=1,4
      xdisp1(j,k) = xdisp(j,k)
100   ydisp1(j,k) = ydisp(j,k)
      do 105 j=1,10
105   stres1(j,k) = stress(j,k)
110   continue
      do 115 i=1,numvar
115   z1(i) = zz(i)
c
      do 120 i=1,numvar
      delz = zz(i)*deltaz
      zz(i) = zz(i)+delz
c
      call anal(zz)
c
      call sendif (ntime,nepact,xdisp,xdisp1,4,delz,dxdz,i,numvar)
      call sendif (ntime,nepact,ydisp,ydisp1,4,delz,dydz,i,numvar)
      call sendif (ntime,nepact,stress,stres1,10,delz,dstrdz,i,
     1             numvar)
c
      zz(i) = z1(i)
120   continue
c
      do 140 k=1,ntime
      do 130 j=1,4
      xdisp(j,k) = xdisp1(j,k)
130   ydisp(j,k) = ydisp1(j,k)
      do 135 j=1,10
135   stress(j,k) = stres1(j,k)
140   continue
c
      return
      end
c
c     ----------------------------------------------------------------
      subroutine storsp (nodes,ndkod,dt,time,dis,vel,acc,lnxdh,
     1                   lnydh,lnzdh,nodsx,nodsy,nodsz,kstat,ndchk)
c     ----------------------------------------------------------------
c
c
c     subroutine to store dynamic response results into two dimensional
c     arrays. The response is stored starting from initial time
c     'tstart' to the final time 'tend' with 'nskip' time steps being
c     skipped. Values for 'tstart', 'tend' and 'nskip' are set in the
c     driving routine (main program in case of just analysis or one of
c                      the user's subroutines in case of optimization)
c     Response quantities at nodes which are specified for output
c     (section D in ANSR data preparation manual) are saved only.
c
c
      implicit double precision (a-h,o-z)
      common /tapes / nin, nou
      common /contrl/ njts,ncnod,nodgc,ndcon,niddof,nmaxd,nmsgc,
     1                nelgr,ntels,kexec,kdummy,kprint
      common /elpar / lpar(10),flpar(6),indgr(20),nmsgr(20),mfgr(20),
     *                infgrr(20),infgri(20),ndfgr(20),dkogr(20),
     *                dktgr(20)
      common /pass  / igr,naddr,naddi,kna,kedatr,kedati,kevar,
     *                istep,ipath,kupd,kitrn,ielasp,ielas,nstref
      common /indat  / ieldat(1)
      common eldat(1)
      common /dynpar/ tstart,tend,nskip,jj,kaddel
      common /resp/ xdisp(4,1),ydisp(4,1),stress(10,1),nepact(1)
```

```
c
      dimension ndkod(nodes,1),dis(1),vel(1),acc(1),lnxdh(1),lnydh(1),
     1          lnzdh(1)
c
      data zero /0.0d0/
c
c     initialize and check dimensions
c
      if (ndchk .gt. 0) go to 100
c
      ndchk  = 1
      nsk    = nskip
      jj     = 0
c
      nmxrow = 4
      nmxcol = 1
      maxrow = max0 (nodsx,nodsy,nodsz)
      if (maxrow .lt. 0) maxrow = nodes
      maxcol = 1
      if (maxrow.le.nmxrow .and. maxcol.le.nmxcol) go to 100
c
      write (nou,2000) maxrow,maxcol
2000  format (/5x,'dimension of arrays for storing dynamic response'/
     1         5x,'is too short--- dimensions needed:'/
     2         5x,'         row dimension = ' i5/
     3         5x,'      column dimension = ' i5)
      stop
c
100   if (kstat.eq.-1) go to 110
      if (time.lt.tstart .or. time.gt.tend) go to 500
c
      if (nsk .eq. nskip) go to 110
c
      nsk = nsk + 1
      go to 500
c
110   jj=jj+1
      nsk = 1
      if (nodsx) 120,160,140
c
120   do 130 i = 1,nodes
      k = ndkod(i,1)
      xdisp(i,jj) = dis(k)
      if (time.eq.zero) go to 130
c     xvel(i,jj)  = vel(k)
c     xacc(i,jj)  = acc(k)
c
c     for rotational displacements about x-axis add
c     k = ndkod(i,4)
c     xdrot(i,jj) = dis(k)
c     xvrot(i,jj) = vel(k)
c     xarot(i,jj) = acc(k)
c
130   continue
      go to 160
c
140   do 150 i=1,nodsx
      l = lnxdh(i)
      k = ndkod(l,1)
      xdisp(i,jj) = dis(k)
      if (time.eq.zero) go to 150
c     xvel(i,jj)  = vel(k)
c     xacc(i,jj)  = acc(k)
150   continue
```

```
c
160   if (nodsy) 170,210,190
c
170   do 180 i = 1,nodes
      k = ndkod(i,2)
      ydisp(i,jj) = dis(k)
      if (time.eq.zero) go to 180
c     yvel(i,jj)  = vel(k)
c     yacc(i,jj)  = acc(k)
180   continue
      go to 210
c
190   do 200 i=1,nodsy
      l = lnydh(i)
      k = ndkod(l,2)
      ydisp(i,jj) = dis(k)
      if (time.eq.zero) go to 200
c     yvel(i,jj)  = vel(k)
c     yacc(i,jj)  = acc(k)
200   continue
c
210   if (nodsz) 220,260,240
c
220   do 230 i = 1,nodes
      k = ndkod(i,3)
c     zdisp(i,jj) = dis(k)
      if (time.eq.zero) go to 230
c     zvel(i,jj)  = vel(k)
c     zacc(i,jj)  = acc(k)
230   continue
      go to 260
c
240   do 250 i=1,nodsz
      l = lnzdh(i)
      k = ndkod(l,3)
c     zdisp(i,jj) = dis(k)
      if (time.eq.zero) go to 250
c     zvel(i,jj)  = vel(k)
c     zacc(i,jj)  = acc(k)
250   continue
c
260   continue
c
c
c     storing element response
c
      naddi = kedati
      naddr = kedatr
c
      do 460 igr=1,nelgr
c
c
      kaddel = 0
      ngr  = indgr(igr)
      nels = nmsgr(igr)
c
      do 460 iel=1,nels
c
      ninfi = ieldat(naddi)
      ninfr = eldat(naddr)
```

```
c
320   go to (330,340,350,360,370,380,390,400,410,420), ngr
c
330   call stor1 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
340   call stor2 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
350   call stor3 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
360   call stor4 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
370   call stor5 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
380   call stor6 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
390   call stor7 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
400   call stor8 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
410   call stor9 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
420   call stor10 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
c
450   naddi = naddi + ninfi + 1
      naddr = naddr + ninfr + 1
c
460   continue
500   return
      end
c
c     ------------------------------------------------------------------
c
      subroutine mdfl1 (icoms,coms,ninfci,ninfcr,z)
c     ------------------------------------------------------------------
c
c     modification routine for element 1
c
      implicit double precision (a-h,o-z)
      dimension z(1), icoms(1), coms(1), icom(1), com(1)
      common /infeli/ imem,kst,lm(6),node(2),ktho
      common /infelr/ emod,xyz(3,2),area,sl,b(6),q1(6),vtot,stot,senp,
     1                senn,tsenp,tsenn,sdamp
      equivalence (imem,icom(1))
      equivalence (emod,com(1))
c
      do 100 i=1,ninfci
100   icom(i) = icoms(i)
      do 110 i=1,ninfcr
110   com(i) = coms(i)
c
      area = z(imem)
c
      kst = 1
      do 120 i=16,ninfcr
120   com(i) = 0.0
c
      do 130 i=1,ninfci
130   icoms(i) = icom(i)
      do 140 i=1,ninfcr
140   coms(i) = com(i)
c
      return
      end
```

```
c
c     ------------------------------------------------------------------
c
      subroutine stor1 (icoms,coms,ninfci,ninfcr,iflag)
c     ------------------------------------------------------------------
c
c     this subroutine has two tasks:
c     -iflag=1                          stores element response values
c                                       in dynamic or static analysis
c     -iflag=2                          stores element quantitities which are
c                                       fixed in an optimization problem, in
c                                       particular data useful for displaying
c                                       the structure
c
      implicit double precision (a-h,o-z)
      dimension icoms(1),icom(1)
      dimension coms(1),com(1)
      common /tapes / niu,nou,nt1,nt2,nt3,nt4,nt5,ntemp
      common /infeli/ imem,kst,lm(6),node(2),ktho
      common /infelr/ emod,xyz(3,2),area,sl,b(6),q1(6),vtot,stot,senp,
     1                senn,tsenp,tsenn,sdamp
      equivalence (imem,icom(1))
      equivalence (emod,com(1))
      common/dynpar/ tstart,tend,nskip,jjt,kaddel
      common /resp/ xdisp(4,1),ydisp(4,1),stress(10,1),nepact(1)
      common/strgom/ elndx1(10),elndx2(10),elndy1(10),elndy2(10),
     1               nod1(10),nod2(10)
      common/strpar/rleng(10),ro,adisp2,astrs2,armin,tolz,deltaz
c
      do 10 j=1,ninfci
10    icom(j) = icoms(j)
      do 15 j=1,ninfcr
15    com(j) = coms(j)
c
      go to (100,200) iflag
c
100   continue
      if(ktho.le.0) return
      kaddel = kaddel+1
c
c     set control on kaddel
c
      stress(kaddel,jjt) = stot / area
      return
c
c
200   continue
      rleng(imem) = sl
      elndx1(imem) = xyz(1,1)
      elndx2(imem) = xyz(1,2)
      elndy1(imem) = xyz(2,1)
      elndy2(imem) = xyz(2,2)
      nod1(imem) = node(1)
      nod2(imem) = node(2)
c
      return
      end
```

PROBLEM-DEPENDENT SUBROUTINES

BRACED FRAME PROBLEM

```
c     ****************************************************************
c     Function evaluation subroutines for 2 story braced plane frame.
c     ****************************************************************
c
c     ----------------------------------------------------------------
c
      subroutine parsym (n,z)
c     ----------------------------------------------------------------
c
c
      implicit double precision (a-h,o-z)
      dimension z(1)
c
c     common blocks for 2 story braced frame problem
c
      common/strpar/rleng(10),ro,adisp2,astrs2,rbmin,rcmin,tolz,deltaz
      common /fcomm/ glb4,glb8,glc8,glc16,wb,wt,frac
      common /resp/ xdisp(2,100),rn(2,100),del(2,100),rmom(2,100),
     *               nepact(100)
      common /dresp/ dxdz(2,4,10),dmomdz(2,4,10)
      common /strgom/ elndx1(10),elndx2(10),elndy1(10),elndy2(10),
     1                nod1(10),nod2(10)
      common /zansr / zresp(10), zsens(10), diff(10)
      common /nstran/ nanal,nsens
c
      nanal = 0
      nsens = 0
c
      call inansr
c
      call set
c
c
      do 100 i = 1 , n
      zresp(i) = 0.0
      zsens(i) = 0.0
100   continue
c
      ro = 0.0001
      adisp2 = 0.2025
      astrs2 = 576.d+6
      rbmin = 0.1
      rcmin = 10.
      tolz = 1.0d-5
      deltaz = 1.0d-5
      glb4 = ro * rleng(1) * 0.4
      glb8 = glb4 * 2.0
      glc8 = ro * rleng(3) * 0.8
      glc16= glc8 * 2.0
      frac = 0.5
c
c
c     declare problem variables into intrac data base
c
      call declar ('TOLZ','double',0,tolz,0,0)
      call declar ('DELTAZ','double',0,deltaz,0,0)
      call declar ('XDISP','double',2,xdisp,2,100)
      call declar ('RN','double',2,rn,2,100)
      call declar ('DEL','double',2,del,2,100)
      call declar ('RMOM','double',2,rmom,2,100)
      call declar ('RO','double',0,ro,0,0)
      call declar ('FRAC','double',0,frac,0,0)
      call declar ('ADISP2','double',0,adisp2,0,0)
```

```
      call declar ('ASTRS2', 'double', 0, astrs2, 0, 0)
      call declar ('RBMIN', 'double', 0, rbmin, 0, 0)
      call declar ('RCMIN', 'double', 0, rcmin, 0, 0)
      call declar ('NANAL', 'int', 0, nanal, 0, 0)
      call declar ('NSENS', 'int', 0, nsens, 0, 0)
      call declar ('ELNDX1', 'double', 1, elndx1, 10, 1)
      call declar ('ELNDX2', 'double', 1, elndx2, 10, 1)
      call declar ('ELNDY1', 'double', 1, elndy1, 10, 1)
      call declar ('ELNDY2', 'double', 1, elndy2, 10, 1)
      call declar ('NOD1', 'int', 1, nod1, 10, 1)
      call declar ('NOD2', 'int', 1, nod2, 10, 1)
c
      return
      end
c
c     ------------------------------------------------------------------
c
      subroutine funcf (n, z, f, nfuncf)
c     ------------------------------------------------------------------
c
      implicit double precision (a-h, o-z)
      dimension z(1)
      common/strpar/rleng(10), ro, adisp2, astrs2, rbmin, rcmin, tolz, deltaz
      common /fcomm/ glb4, glb8, glc8, glc16, wb, wt, frac
c
      f = 0.0
      do 100 i=1, 2
      f = f + glb8*dsqrt(z(i))
      f = f + glc16*dsqrt(z(i+2))
100   continue
c
      return
      end
c
c     ------------------------------------------------------------------
c
      subroutine gradf (n, z, grad)
c     ------------------------------------------------------------------
c
      implicit double precision (a-h, o-z)
      dimension z(1), grad(1)
      common/strpar/rleng(10), ro, adisp2, astrs2, rbmin, rcmin, tolz, deltaz
      common /fcomm/ glb4, glb8, glc8, glc16, wb, wt, frac
c
      grad(1) = glb4 / dsqrt(z(1))
      grad(2) = glb4 / dsqrt(z(2))
      grad(3) = glc8 / dsqrt(z(3))
      grad(4) = glc8 / dsqrt(z(4))
c
      return
      end
c
c     ------------------------------------------------------------------
c
      subroutine funcg (n, jp, z, g, psi, nfuncg)
c     ------------------------------------------------------------------
c
      implicit double precision (a-h, o-z)
      dimension z(1), g(1)
      common/strpar/rleng(10), ro, adisp2, astrs2, rbmin, rcmin, tolz, deltaz
      common /fcomm/ glb4, glb8, glc8, glc16, wb, wt, frac
c
```

```
      do 100 i=1,2
      g(i) = -z(i)+rbmin
100   g(i+2) = -z(i+2) + rcmin
      wb = glb8*(dsqrt(z(1))+dsqrt(z(2)))
      wc = glc16*(dsqrt(z(3))+dsqrt(z(4)))
      wt = wb+wc
      g(5) = wb/(frac*wt)-1.
c
      do 150 i=1,jp
150   if(g(i) .gt. psi) psi=g(i)
      return
      end
c
c     ------------------------------------------------------------------
c
      subroutine gradg (n,j,z,grad)
c     ------------------------------------------------------------------
c
      implicit double precision (a-h,o-z)
      dimension z(1),grad(1)
      common/strpar/rleng(10),ro,adisp2,astrs2,rbmin,rcmin,tolz,deltaz
      common /fcomm/ glb4,glb8,glc8,glc16,wb,wt,frac
c
c
      if (j .gt. 4) go to 110
c
      do 100 i=1,n
100   grad(i) = 0.0
      grad(j) = -1.0
      return
c
110   continue
      bet = wb/wt
      fac = (1.-bet)*glb4/(frac*wt)
      grad(1) = fac/dsqrt(z(1))
      grad(2) = fac/dsqrt(z(2))
      fac = -bet*glc8/(frac*wt)
      grad(3) = fac/dsqrt(z(3))
      grad(4) = fac/dsqrt(z(4))
c
      return
      end
c
c     ------------------------------------------------------------------
c
      subroutine funcph (n,njq,jq,z,w0,wc,deltaw,nq,phi,psi,nfuncp)
c     ------------------------------------------------------------------
c
      implicit double precision (a-h,o-z)
      dimension z(1),phi(njq,1)
      logical bigdif
      common/integr/nsteps,dt,dto,dampm,dampkt,dampko
      common/phigra/a2,b2,b23,b24,iw0dt
      common/strpar/rleng(10),ro,adisp2,astrs2,rbmin,rcmin,tolz,deltaz
      common /resp/ xdisp(2,100),rn(2,100),del(2,100),rmom(2,100),
     *              nepact(100)
      common /strgom/ elndx1(10),elndx2(10),elndy1(10),elndy2(10),
     1                nod1(10),nod2(10)
      common /zansr / zresp(10), zsens(10), diff(10)
      common /nstran/ nanal,nsens
      common/dynpar/ tstart,tend,nskip,jj,kaddel
      common /dresp/ dxdz(2,4,10),dmomdz(2,4,10)
      common /tapes/nin,nou
```

```
c
c
      write (nou,2010)
2010  format (5x,'entering funcph')
      if(bigdif(n,z,zresp,diff,tolz)) go to 120
      if (nsens .le. 0)  go to 300
c
      ii = 1
      do 110 i=1,nq
      if(nepact(i).eq.0) go to 110
      do 100 j=1,n
      delz = diff(j)
      xdisp(1,i) = xdisp(1,i)+dxdz(1,j,ii)*delz
      xdisp(2,i) = xdisp(2,i)+dxdz(2,j,ii)*delz
      rmom(1,i) = rmom(1,i)+dmomdz(1,j,ii)*delz
      rmom(2,i) = rmom(2,i)+dmomdz(2,j,ii)*delz
100   continue
      ii = ii+1
110   continue
c
      go to 200
c
c     set variables in common/dynpar/
c
120     nskip = (wc-w0)/(dt*nq) + 1.e-6
        iw0dt=w0/dt+1
        ns=iw0dt+(nq-1)*nskip
        ishift=(wc/dt+0.5-ns)/2
        iw0dt=iw0dt+ishift
        nstpv = nsteps
        nsteps = min0(nsteps,(ns+ishift))
        tstart=iw0dt*dt
        tend=nsteps*dt
c
c
      call anal(zresp)
      nanal = nanal+1
      write (nou,2000) nanal
2000  format (/5x,'nanal in funcph',i5)
      call mprint (z,1,n,40hz vector in funcph
     1                   )
c
      nsteps = nstpv
c
c
c
200   continue
      a2 = adisp2
      b2 = 0.6804*astrs2
      b23 = b2*z(3)**1.5
      b24 = b2*z(4)**1.5
      do 210 i=1,nq
      phi(1,i) = xdisp(1,i)*xdisp(1,i)/a2-1.
      phi(2,i) =(xdisp(2,i)-xdisp(1,i))*(xdisp(2,i)-xdisp(1,i))/a2-1.
      phi(3,i) = rmom(1,i)*rmom(1,i)/b23-1.
      phi(4,i) = rmom(2,i)*rmom(2,i)/b24-1.
210   continue
c
c     set up function psi
c
300   do 310 l=1,jq
      do 310 k=1,nq
      if(phi(l,k).gt.psi) psi=phi(l,k)
```

```
310   continue
      return
      end
c
c     ------------------------------------------------------------------
c
      subroutine gradph (n,njq,nactiv, jq, w0,wc,deltaw,nq,neptf,l,z,k,
     *                   grad,igrad)
c     ------------------------------------------------------------------
      implicit double precision (a-h,o-z)
      dimension z(1),grad(1)
      dimension neptf(njq,1)
      logical bigdif
      common/integr/nsteps,dt,dto,dampm,dampkt,dampko
      common/phigra/a2,b2,b23,b24,iw0dt
      common/strpar/rleng(10),ro,adisp2,astrs2,rbmin,rcmin,tolz,deltaz
      common /resp/ xdisp(2,100),rn(2,100),del(2,100),rmom(2,100),
     *              nepact(100)
      common /strgom/ elndx1(10),elndx2(10),elndy1(10),elndy2(10),
     1                nod1(10),nod2(10)
      common /zansr / zresp(10), zsens(10), diff(10)
      common /nstran/ nanal,nsens
      common/dynpar/ tstart,tend,nskip,jj,kaddel
      common /dresp/ dxdz(2,4,10),dmomdz(2,4,10)
c
      if(igrad.gt.1) go to 50
      do 10 i=1,nq
10    nepact(i) = 0
c
      maxsen = 1
c
      do 30 i=1,jq
      do 20 j=1,nactiv
      nj = neptf(i,j)
      if(nj.eq.0) go to 30
      nepact(nj)=1
      if(nj.gt.maxsen) maxsen = nj
20    continue
30    continue
c
      if(.not.bigdif(n,z,zsens,diff,tolz)) go to 50
      call sens (zsens,n,maxsen)
      nsens = nsens+1
50    continue
      nc=0
      do 60 i=1,k
      if(nepact(i).eq.0) go to 60
      nc = nc+1
60    continue
c
      go to (100,200,300,400) l
c
100   continue
      fact = 2.*xdisp(1,k)/a2
      do 110 i=1,n
110   grad(i) = dxdz(1,i,nc)*fact
      return
c
200   continue
      fact = 2.*(xdisp(2,k)-xdisp(1,k))/a2
      do 210 i=1,n
210   grad(i) = (dxdz(2,i,nc)-dxdz(1,i,nc))*fact
      return
```

```
c
300   continue
      fact = 2.*rmom(1,k)/b23
      do 310 i=1,n
310   grad(i) = dmomdz(1,i,nc)*fact
      grad(3) = grad(3)-0.75*fact*rmom(1,k)/z(3)
      return
c
400   continue
      fact = 2.*rmom(2,k)/b24
      do 410 i=1,n
410   grad(i) = dmomdz(2,i,nc)*fact
      grad(4) = grad(4)-0.75*fact*rmom(2,k)/z(4)
      return
c
      end
c
c     ------------------------------------------------------------------
c
      subroutine sens (zz,numvar,maxsen)
c     ------------------------------------------------------------------
c
      implicit double precision (a-h,o-z)
      dimension z1(10),zz(1),xdisp1(2,100),rmom1(2,100)
      common/integr/nsteps,dt,dto,dampm,dampkt,dampko
      common/phigra/a2,b2,b23,b24,iw0dt
      common/strpar/rleng(10),ro,adisp2,astrs2,rbmin,rcmin,tolz,deltaz
      common /resp/ xdisp(2,100),rn(2,100),del(2,100),rmom(2,100),
     *              nepact(100)
      common /nstran/ nanal,nsens
      common/dynpar/ tstart,tend,nskip,jj,kaddel
      common /dresp/ dxdz(2,4,10),dmomdz(2,4,10)
c
      do 110 j=1,2
      do 110 i=1,maxsen
      xdisp1(j,i) = xdisp(j,i)
      rmom1(j,i) = rmom(j,i)
110   continue
      do 115 i=1,numvar
115   z1(i) = zz(i)
      nstpv = nsteps
      nsteps = iw0dt+(maxsen-1)*nskip
c
      do 120 i=1,numvar
      delz = zz(i)*deltaz
      zz(i) = z1(i)+delz
c
      call anal(zz)
c
      call sendif (maxsen,nepact,xdisp,xdisp1,2,delz,dxdz,i,numvar)
      call sendif (maxsen,nepact,rmom,rmom1,2,delz,dmomdz,i,numvar)
c
      zz(i) = z1(i)
120   continue
c
      do 140 j=1,2
      do 140 i=1,maxsen
      xdisp(j,i) = xdisp1(j,i)
      rmom(j,i) = rmom1(j,i)
140   continue
      nsteps = nstpv
c
      return
      end
```

```
c
c     ------------------------------------------------------------------
c
      subroutine storsp (nodes,ndkod,dt,time,dis,vel,acc,lnxdh,
     1                   lnydh,lnzdh,nodsx,nodsy,nodsz,kstat,ndchk)
c     ------------------------------------------------------------------
c
c     subroutine to store dynamic response results into two dimensional
c     arrays. The response is stored starting from initial time
c     'tstart' to the final time 'tend' with 'nskip' time steps being
c     skipped. Values for 'tstart', 'tend' and 'nskip' are set in the
c     driving routine (main program in case of just analysis or one of
c                      the user's subroutines in case of optimization)
c     Response quantities at nodes which are specified for output
c     (section D in ANSR data preparation manual) are saved only.
c
c
      implicit double precision (a-h,o-z)
      common /tapes / nin, nou
      common /contrl/ njts,ncnod,nodgc,ndcon,niddof,nmaxd,nmsgc,
     1                nelgr,ntels,kexec,kdummy,kprint
      common /elpar / lpar(10),flpar(6),indgr(20),nmsgr(20),mfgr(20),
     *                infgrr(20),infgri(20),ndfgr(20),dkogr(20),
     *                dktgr(20)
      common /pass  / igr,naddr,naddi,kna,kedatr,kedati,kevar,
     *                istep,ipath,kupd,kitrn,ielasp,ielas,nstref
      common /indat  / ieldat(1)
      common eldat(1)
      common /dynpar/ tstart,tend,nskip,jj,kaddel
      common /resp/ xdisp(2,100),rn(2,100),del(2,100),rmom(2,100),
     *              nepact(100)
c
      dimension ndkod(nodes,1),dis(1),vel(1),acc(1),lnxdh(1),lnydh(1),
     1          lnzdh(1)
      data zero/0.0d0/
c
c     initialize and check dimensions
c
      if (ndchk .gt. 0) go to 100
c
      ndchk  = 1
      nsk    = nskip
      JJ     = 0
c
      nmxrow = 2
      nmxcol = 100
      maxrow = max0 (nodsx,nodsy,nodsz)
      if (maxrow .lt. 0) maxrow = nodes
      maxcol = ifix((tend - tstart) / (float(nskip)*dt)) + 1
      if (maxrow.le.nmxrow .and. maxcol.le.nmxcol) go to 100
c
      write (nou,2000) maxrow,maxcol
2000  format (/5x,'dimension of arrays for storing dynamic response'/
     1         5x,'is too short--- dimensions needed: '/
     2         5x,'         row dimension = ' i5/
     3         5x,'      column dimension = ' i5)
      stop
c
100   if (kstat.eq.-1) go to 110
      if (time.lt.tstart .or. time.gt.tend) go to 500
c
      if (nsk .eq. nskip) go to 110
```

```
c
      nsk = nsk + 1
      go to 500
c
110   jj=jj+1
      nsk = 1
      if (nodsx) 120,160,140
c
120   do 130 i = 1,nodes
      k = ndkod(i,1)
      xdisp(i,jj) = dis(k)
      if (time.eq.zero) go to 130
c     xvel(i,jj)  = vel(k)
c     xacc(i,jj)  = acc(k)
c
c     for rotational displacements about x-axis add
c     k = ndkod(i,4)
c     xdrot(i,jj) = dis(k)
c     xvrot(i,jj) = vel(k)
c     xarot(i,jj) = acc(k)
c
130   continue
      go to 160
c
140   do 150 i=1,nodsx
      l = lnxdh(i)
      k = ndkod(l,1)
      xdisp(i,jj) = dis(k)
      if (time.eq.zero) go to 150
c     xvel(i,jj)  = vel(k)
c     xacc(i,jj)  = acc(k)
150   continue
c
160   if (nodsy) 170,210,190
c
170   do 180 i = 1,nodes
      k = ndkod(i,2)
c     ydisp(i,jj) = dis(k)
      if (time.eq.zero) go to 180
c     yvel(i,jj)  = vel(k)
c     yacc(i,jj)  = acc(k)
180   continue
      go to 210
c
190   do 200 i=1,nodsy
      l = lnydh(i)
      k = ndkod(l,2)
c     ydisp(i,jj) = dis(k)
      if (time.eq.zero) go to 200
c     yvel(i,jj)  = vel(k)
c     yacc(i,jj)  = acc(k)
200   continue
c
210   if (nodsz) 220,260,240
c
220   do 230 i = 1,nodes
      k = ndkod(i,3)
c     zdisp(i,jj) = dis(k)
      if (time.eq.zero) go to 230
c     zvel(i,jj)  = vel(k)
c     zacc(i,jj)  = acc(k)
230   continue
      go to 260
```

```
c
240   do 250 i=1,nodsz
      l = lnzdh(i)
      k = ndkod(l,3)
c     zdisp(i,jj) = dis(k)
      if (time.eq.zero) go to 250
c     zvel(i,jj)  = vel(k)
c     zacc(i,jj)  = acc(k)
250   continue
c
260   continue
c
c
c     storing element response
c
      naddi = kedati
      naddr = kedatr
c
      do 460 igr=1,nelgr
c
c
      kaddel = 0
      ngr  = indgr(igr)
      nels = nmsgr(igr)
c
      do 460 iel=1,nels
c
      ninfi = ieldat(naddi)
      ninfr = eldat(naddr)
c
320   go to (330,340,350,360,370,380,390,400,410,420), ngr
c
330   call stor1 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
340   call stor2 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
350   call stor3 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
360   call stor4 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
370   call stor5 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
380   call stor6 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
390   call stor7 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
400   call stor8 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
410   call stor9 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
420   call stor10 (ieldat(naddi+1),eldat(naddr+1),ninfi,ninfr,1)
      go to 450
c
450   naddi = naddi + ninfi + 1
      naddr = naddr + ninfr + 1
c
460   continue
500   return
      end
```

```
c     ------------------------------------------------------------------
c
      subroutine mdfl2 (icoms,coms,ninfci,ninfcr,z)
c     ------------------------------------------------------------------
c
c     modification routine for element 2
c
      implicit double precision (a-h,o-z)
      dimension icoms(1),icom(1)
      dimension coms(1),com(1)
      dimension z(1)
      common /tapes / niu,nou,nt1,nt2,nt3,nt4,nt5,ntemp
      common /infeli / imem,kst,lm(6),node(2),kgeom,ktho,kbuck,
     *                 kod,kodp,irest(1)
      common /infelr / eprop(4),area,dumpro(4),
     *                 xyz(3,2),sl,t(3,3),du1x,du1y,du1z,q1(6),
     *                 skp(6,6),vtot,sep,sel,venp,venn,vpacp,
     *                 vpacn,vbuck,senp,senn,tvenp,tvenn,tsenp,tsenn,
     *                 sdamp,rest(1)
      equivalence (imem,icom(1))
      equivalence (eprop(1),com(1))
c
      do 10 j=1,ninfci
10    icom(j) = icoms(j)
      do 15 j=1,ninfcr
15    com(j) = coms(j)
c
      area = 0.8 * dsqrt(z(imem))
      do 20 j=1,4
20    eprop(j) = dumpro(j)*area
c
      kst = 1
      do 100 i=26,ninfcr
100   com(i) = 0.
      do 105 i=14,ninfci
105   icom(i) = 0
c
      do 110 i=1,ninfci
110   icoms(i) = icom(i)
      do 120 i=1,ninfcr
120   coms(i) = com(i)
c
      return
      end
c
c     ------------------------------------------------------------------
c
      subroutine mdfl3 (icoms,coms,ninfci,ninfcr,z)
c     ------------------------------------------------------------------
c
c     modification routine for element 3
c
      implicit double precision (a-h,o-z)
      common/tapes/niu,nou,nt1,nt2,nt3,nt4,nt5,nt6
      common/infeli/ imem,kst,lm(6),nodi,nodj,koutdt
      common/infelr/ fl,flij,ymod,area,rin,xy(2,2),af(2,6),ax(6),
     1               accums(3),senp(6),senn(6),tenp(6),tenn(6)
      dimension icom(1),com(1),icoms(1),coms(1)
      dimension z(1)
      equivalence (imem,icom(1)),(fl,com(1))
      do 10 j=1,ninfci
10    icom(j)=icoms(j)
      do 15 j=1,ninfcr
15    com(j)=coms(j)
```

```
c
c
      if(imem.gt.6) return
      if(imem.eq.3.or.imem.eq.4) rin=z(3)
      if(imem.eq.5.or.imem.eq.6) rin=z(4)
      area = 0.8*dsqrt(rin)
c
      kst=1
      do 100 i=28,ninfcr
100   com(i) = 0.0
c
      do 110 i=1,ninfci
110   icoms(i) = icom(i)
      do 120 i=1,ninfcr
120   coms(i) = com(i)
c
      return
      end
c
c     ------------------------------------------------------------------
c
      subroutine stor2 (icoms,coms,ninfci,ninfcr,iflag)
c     ------------------------------------------------------------------
c
c     this subroutine has two tasks:
c     -iflag=1                          stores element response values
c                                       in dynamic or static analysis
c     -iflag=2                          stores element quantitities which are
c                                       fixed in an optimization problem,in
c                                       particular data useful for displaying
c                                       the structure
c
      implicit double precision (a-h,o-z)
      dimension icoms(1),icom(1)
      dimension coms(1),com(1)
      common /tapes / niu,nou,nt1,nt2,nt3,nt4,nt5,ntemp
      common /infeli / imem,kst,lm(6),node(2),kgeom,ktho,kbuck,
     *                 kod,kodp,irest(1)
      common /infelr / eprop(4),area,dumpro(4),
     *                 xyz(3,2),sl,t(3,3),du1x,du1y,du1z,q1(6),
     *                 skp(6,6),vtot,sep,sel,venp,venn,vpacp,
     *                 vpacn,vbuck,senp,senn,tvenp,tvenn,tsenp,tsenn,
     *                 sdamp,rest(1)
      common/dynpar/ tstart,tend,nskip,jjt,kaddel
      common/resp/ xdisp(2,100),rn(2,100),del(2,100),rmom(2,100),
     *             nepact(100)
      common/strgom/ elndx1(10),elndx2(10),elndy1(10),elndy2(10),
     1              nod1(10),nod2(10)
      common/strpar/rleng(10),ro,adisp2,astrs2,rbmin,rcmin,tolz,deltaz
      equivalence (imem,icom(1))
      equivalence (eprop(1),com(1))
c
      do 10 j=1,ninfci
10    icom(j) = icoms(j)
      do 15 j=1,ninfcr
15    com(j) = coms(j)
c
      go to (100,200) iflag
c
100   continue
      if(ktho.le.0) return
      kaddel = kaddel+1
```

```
c
c     set control on kaddel
c
      rn(kaddel,jjt) = sel+sep
      del(kaddel,jjt) = vtot
      return
c
c
200   continue
      rleng(imem) = sl
      elndx1(imem) = xyz(1,1)
      elndx2(imem) = xyz(1,2)
      elndy1(imem) = xyz(2,1)
      elndy2(imem) = xyz(2,2)
      nod1(imem) = node(1)
      nod2(imem) = node(2)
      do 220 j=1,4
220   dumpro(j) = eprop(j)/area
      do 240  j=1,ninfcr
240   coms(j) = com(j)
c
      return
      end
c
c     ------------------------------------------------------------------
c
      subroutine stor3 (icoms,coms,ninfci,ninfcr,iflag)
c     ------------------------------------------------------------------
c
c
      implicit double precision (a-h,o-z)
      common/tapes/niu,nou,nt1,nt2,nt3,nt4,nt5,nt6
      common/infeli/ imem,kst,lm(6),nodi,nodj,koutdt
      common/infelr/ fl,flij,ymod,area,rin,xy(2,2),af(2,6),ax(6),
     1               accums(3),senp(6),senn(6),tenp(6),tenn(6)
      common/dynpar/ tstart,tend,nskip,jjt,kaddel
      common/resp/ xdisp(2,100),rn(2,100),del(2,100),rmom(2,100),
     *             nepact(100)
      common/strgom/ elndx1(10),elndx2(10),elndy1(10),elndy2(10),
     1               nod1(10),nod2(10)
      common/strpar/rleng(10),ro,adisp2,astrs2,rbmin,rcmin,tolz,deltaz
      dimension icom(1),com(1),icoms(1),coms(1)
      equivalence (imem,icom(1)),(fl,com(1))
      do 10 j=1,ninfci
10    icom(j)=icoms(j)
      do 15 j=1,ninfcr
15    com(j)=coms(j)
c
      go to (100,200) iflag
c
100   continue
      if(koutdt.le.0) return
      kaddel = kaddel+1
c     set control on kaddel
c
      rmom(kaddel,jjt) = accums(1)
      return
```

```
c
c
200   continue
      rleng(imem) = fl
      elndx1(imem) = xy(1,1)
      elndx2(imem) = xy(1,2)
      elndy1(imem) = xy(2,1)
      elndy2(imem) = xy(2,2)
      nod1(imem) = nodi
      nod2(imem) = nodj
c
      return
      end
```

DATA FILES

FOR EXAMPLE PROBLEMS 1 AND 2

```
******* brace frame *******
100,1,20,20
1.,.01,.001,.2,2.
5,4,4
100.,.2,.3,30.
100,100,0.,1.
0.,1.
1.,1.,1.,1.,1.
1.,1.,1.,1.
20.,20.,20.,20.
start    two story braced frame for design problem 2
6,6,0,2,2,2,2
1
2,0.,180.
3,0.,360.
4,360.
5,360.,180.
6,360.,360.
1,1,1,1,1,1,1,  ,  ,1,4
2,0,1,1,1,1,1,  ,  ,3,3,5,6
1,1,1,1,1,1,2,2,5
1,1,1,1,1,1,2,3,6
2,104.,0.,0.,0.,0.,0.,3,1
5,104.,0.,0.,0.,0.,0.,6,1
0,0,0,1,0,5,0,100,.01,0.
5,0,0,1,  ,  ,  ,1.
constant acceleration pulse                    (10f8.0)
0.,0.,.025,140.,.475,140.,.5,0.,1.,0.
0.,0.,0.
0,  ,1,1,1,20,1,.01,.01,.01
5,5,0,2
5,6
1,2,1,1
1,30000000.,0.,18000.,18000.
1,1,5,1,100.,0.,  ,0,1
2,2,6,1,100.,0.,  ,0,1
2,4,3,2,0
1,30000000.,13.9,300.
2,30000000.,11.3,200.
3,1,2,  ,1,  ,1
4,4,5,  ,1
5,2,3,  ,2,  ,1
6,5,6,  ,2
stop
```

```
********* 10 bar plane truss problem ***********
100,1,20,10
1.,.01,.001,.2,2.
28,0,10
100.,.2,.3,1.
0.,1.
1.,1.,1.,1.,1.,1.,1.,1.
1.,1.,1.,1.,1.,1.,1.,1.
1.,1.,1.,1.,1.,1.,1.,1.
1.,1.,1.,1.
30.0,30.0,30.0,30.0,30.0,30.0,30.0,30.0
30.0,30.0
start     10 bar plane truss problem
6,6,0,2,0,0,1,0
1,720.0,360.0,0.
2,720.0,0.,0.
3,360.0,360.0,0.
4,360.0,0.,0.
5,0.,360.0,0.
6,0.,0.,0.
1,0,0,1,1,1,1,4,1
5,1,1,1,1,1,1,6,1
-1,1,1,
2                concentrated loads of 1.0 at nodes 2 and 4
2,0.,-1.0,
4,0.,-1.0,
1,1,1,0,1,0,1,1,.001,.001,.001,
100.0
0,0,0,4,4,0
1,2,3,4
1,2,3,4
1,10,1,1,
1,10000.0
1,5,3,1,30.0,0,1
2,3,1,1,30.0,0,1
3,6,4,1,30.0,0,1
4,4,2,1,30.0,0,1
5,4,3,1,30.0,0,1
6,2,1,1,30.0,0,1
7,5,4,1,30.0,0,1
8,6,3,1,30.0,0,1
9,3,2,1,30.0,0,1
10,4,1,1,30.0,0,1
stop
```

EARTHQUAKE ENGINEERING RESEARCH CENTER REPORTS

NOTE: Numbers in parenthesis are Accession Numbers assigned by the National Technical Information Service; these are followed by a price code. Copies of the reports may be ordered from the National Technical Information Service, 5285 Port Royal Road, Springfield, Virginia, 22161. Accession Numbers should be quoted on orders for reports (PB --- ---) and remittance must accompany each order. Reports without this information were not available at time of printing. Upon request, EERC will mail inquirers this information when it becomes available.

EERC 67-1 "Feasibility Study Large-Scale Earthquake Simulator Facility," by J. Penzien, J.G. Bouwkamp, R.W. Clough and D. Rea - 1967 (PB 187 905)A07

EERC 68-1 Unassigned

EERC 68-2 "Inelastic Behavior of Beam-to-Column Subassemblages Under Repeated Loading," by V.V. Bertero - 1968 (PB 184 888)A05

EERC 68-3 "A Graphical Method for Solving the Wave Reflection-Refraction Problem," by H.D. McNiven and Y. Mengi - 1968 (PB 187 943)A03

EERC 68-4 "Dynamic Properties of McKinley School Buildings," by D. Rea, J.G. Bouwkamp and R.W. Clough - 1968 (PB 187 902)A07

EERC 68-5 "Characteristics of Rock Motions During Earthquakes," by H.B. Seed, I.M. Idriss and F.W. Kiefer - 1968 (PB 188 338)A03

EERC 69-1 "Earthquake Engineering Research at Berkeley," - 1969 (PB 187 906)A11

EERC 69-2 "Nonlinear Seismic Response of Earth Structures," by M. Dibaj and J. Penzien - 1969 (PB 187 904)A08

EERC 69-3 "Probabilistic Study of the Behavior of Structures During Earthquakes," by R. Ruiz and J. Penzien - 1969 (PB 187 886)A06

EERC 69-4 "Numerical Solution of Boundary Value Problems in Structural Mechanics by Reduction to an Initial Value Formulation," by N. Distefano and J. Schujman - 1969 (PB 187 942)A02

EERC 69-5 "Dynamic Programming and the Solution of the Biharmonic Equation," by N. Distefano - 1969 (PB 187 941)A03

EERC 69-6 "Stochastic Analysis of Offshore Tower Structures," by A.K. Malhotra and J. Penzien - 1969 (PB 187 903)A09

EERC 69-7 "Rock Motion Accelerograms for High Magnitude Earthquakes," by H.B. Seed and I.M. Idriss - 1969 (PB 187 940)A02

EERC 69-8 "Structural Dynamics Testing Facilities at the University of California, Berkeley," by R.M. Stephen, J.G. Bouwkamp, R.W. Clough and J. Penzien - 1969 (PB 189 111)A04

EERC 69-9 "Seismic Response of Soil Deposits Underlain by Sloping Rock Boundaries," by H. Dezfulian and H.B. Seed 1969 (PB 189 114)A03

EERC 69-10 "Dynamic Stress Analysis of Axisymmetric Structures Under Arbitrary Loading," by S. Ghosh and E.L. Wilson 1969 (PB 189 026)A10

EERC 69-11 "Seismic Behavior of Multistory Frames Designed by Different Philosophies," by J.C. Anderson and V. V. Bertero - 1969 (PB 190 662)A10

EERC 69-12 "Stiffness Degradation of Reinforcing Concrete Members Subjected to Cyclic Flexural Moments," by V.V. Bertero, B. Bresler and H. Ming Liao - 1969 (PB 202 942)A07

EERC 69-13 "Response of Non-Uniform Soil Deposits to Travelling Seismic Waves," by H. Dezfulian and H.B. Seed - 1969 (PB 191 023)A03

EERC 69-14 "Damping Capacity of a Model Steel Structure," by D. Rea, R.W. Clough and J.G. Bouwkamp - 1969 (PB 190 663)A06

EERC 69-15 "Influence of Local Soil Conditions on Building Damage Potential during Earthquakes," by H.B. Seed and I.M. Idriss - 1969 (PB 191 036)A03

EERC 69-16 "The Behavior of Sands Under Seismic Loading Conditions," by M.L. Silver and H.B. Seed - 1969 (AD 714 982)A07

EERC 70-1 "Earthquake Response of Gravity Dams," by A.K. Chopra - 1970 (AD 709 640)A03

EERC 70-2 "Relationships between Soil Conditions and Building Damage in the Caracas Earthquake of July 29, 1967," by H.B. Seed, I.M. Idriss and H. Dezfulian - 1970 (PB 195 762)A05

EERC 70-3 "Cyclic Loading of Full Size Steel Connections," by E.P. Popov and R.M. Stephen - 1970 (PB 213 545)A04

EERC 70-4 "Seismic Analysis of the Charaima Building, Caraballeda, Venezuela," by Subcommittee of the SEAONC Research Committee: V.V. Bertero, P.F. Fratessa, S.A. Mahin, J.H. Sexton, A.C. Scordelis, E.L. Wilson, L.A. Wyllie, H.B. Seed and J. Penzien, Chairman - 1970 (PB 201 455)A06

EERC 70-5 "A Computer Program for Earthquake Analysis of Dams," by A.K. Chopra and P. Chakrabarti - 1970 (AD 723 994)A05

EERC 70-6 "The Propagation of Love Waves Across Non-Horizontally Layered Structures," by J. Lysmer and L.A. Drake 1970 (PB 197 896)A03

EERC 70-7 "Influence of Base Rock Characteristics on Ground Response," by J. Lysmer, H.B. Seed and P.B. Schnabel 1970 (PB 197 897)A03

EERC 70-8 "Applicability of Laboratory Test Procedures for Measuring Soil Liquefaction Characteristics under Cyclic Loading," by H.B. Seed and W.H. Peacock - 1970 (PB 198 016)A03

EERC 70-9 "A Simplified Procedure for Evaluating Soil Liquefaction Potential," by H.B. Seed and I.M. Idriss - 1970 (PB 198 009)A03

EERC 70-10 "Soil Moduli and Damping Factors for Dynamic Response Analysis," by H.B. Seed and I.M. Idriss - 1970 (PB 197 869)A03

EERC 71-1 "Koyna Earthquake of December 11, 1967 and the Performance of Koyna Dam," by A.K. Chopra and P. Chakrabarti 1971 (AD 731 496)A06

EERC 71-2 "Preliminary In-Situ Measurements of Anelastic Absorption in Soils Using a Prototype Earthquake Simulator," by R.D. Borcherdt and P.W. Rodgers - 1971 (PB 201 454)A03

EERC 71-3 "Static and Dynamic Analysis of Inelastic Frame Structures," by F.L. Porter and G.H. Powell - 1971 (PB 210 135)A06

EERC 71-4 "Research Needs in Limit Design of Reinforced Concrete Structures," by V.V. Bertero - 1971 (PB 202 943)A04

EERC 71-5 "Dynamic Behavior of a High-Rise Diagonally Braced Steel Building," by D. Rea, A.A. Shah and J.G. Bouwkamp 1971 (PB 203 584)A06

EERC 71-6 "Dynamic Stress Analysis of Porous Elastic Solids Saturated with Compressible Fluids," by J. Ghaboussi and E. L. Wilson - 1971 (PB 211 396)A06

EERC 71-7 "Inelastic Behavior of Steel Beam-to-Column Subassemblages," by H. Krawinkler, V.V. Bertero and E.P. Popov 1971 (PB 211 335)A14

EERC 71-8 "Modification of Seismograph Records for Effects of Local Soil Conditions," by P. Schnabel, H.B. Seed and J. Lysmer - 1971 (PB 214 450)A03

EERC 72-1 "Static and Earthquake Analysis of Three Dimensional Frame and Shear Wall Buildings," by E.L. Wilson and H.H. Dovey - 1972 (PB 212 904)A05

EERC 72-2 "Accelerations in Rock for Earthquakes in the Western United States," by P.B. Schnabel and H.B. Seed - 1972 (PB 213 100)A03

EERC 72-3 "Elastic-Plastic Earthquake Response of Soil-Building Systems," by T. Minami - 1972 (PB 214 868)A08

EERC 72-4 "Stochastic Inelastic Response of Offshore Towers to Strong Motion Earthquakes," by M.K. Kaul - 1972 (PB 215 713)A05

EERC 72-5 "Cyclic Behavior of Three Reinforced Concrete Flexural Members with High Shear," by E.P. Popov, V.V. Bertero and H. Krawinkler - 1972 (PB 214 555)A05

EERC 72-6 "Earthquake Response of Gravity Dams Including Reservoir Interaction Effects," by P. Chakrabarti and A.K. Chopra - 1972 (AD 762 330)A08

EERC 72-7 "Dynamic Properties of Pine Flat Dam," by D. Rea, C.Y. Liaw and A.K. Chopra - 1972 (AD 763 928)A05

EERC 72-8 "Three Dimensional Analysis of Building Systems," by E.L. Wilson and H.H. Dovey - 1972 (PB 222 438)A06

EERC 72-9 "Rate of Loading Effects on Uncracked and Repaired Reinforced Concrete Members," by S. Mahin, V.V. Bertero, D. Rea and M. Atalay - 1972 (PB 224 520)A08

EERC 72-10 "Computer Program for Static and Dynamic Analysis of Linear Structural Systems," by E.L. Wilson, K.-J. Bathe, J.E. Peterson and H.H.Dovey - 1972 (PB 220 437)A04

EERC 72-11 "Literature Survey - Seismic Effects on Highway Bridges," by T. Iwasaki, J. Penzien and R.W. Clough - 1972 (PB 215 613)A19

EERC 72-12 "SHAKE-A Computer Program for Earthquake Response Analysis of Horizontally Layered Sites," by P.B. Schnabel and J. Lysmer - 1972 (PB 220 207)A06

EERC 73-1 "Optimal Seismic Design of Multistory Frames," by V.V. Bertero and H. Kamil - 1973

EERC 73-2 "Analysis of the Slides in the San Fernando Dams During the Earthquake of February 9, 1971," by H.B. Seed, K.L. Lee, I.M. Idriss and F. Makdisi - 1973 (PB 223 402)A14

EERC 73-3 "Computer Aided Ultimate Load Design of Unbraced Multistory Steel Frames," by M.B. El-Hafez and G.H. Powell 1973 (PB 248 315)A09

EERC 73-4 "Experimental Investigation into the Seismic Behavior of Critical Regions of Reinforced Concrete Components as Influenced by Moment and Shear," by M. Celebi and J. Penzien - 1973 (PB 215 884)A09

EERC 73-5 "Hysteretic Behavior of Epoxy-Repaired Reinforced Concrete Beams," by M. Celebi and J. Penzien - 1973 (PB 239 568)A03

EERC 73-6 "General Purpose Computer Program for Inelastic Dynamic Response of Plane Structures," by A. Kanaan and G.H. Powell - 1973 (PB 221 260)A08

EERC 73-7 "A Computer Program for Earthquake Analysis of Gravity Dams Including Reservoir Interaction," by P. Chakrabarti and A.K. Chopra - 1973 (AD 766 271)A04

EERC 73-8 "Behavior of Reinforced Concrete Deep Beam-Column Subassemblages Under Cyclic Loads," by O. Küstü and J.G. Bouwkamp - 1973 (PB 246 117)A12

EERC 73-9 "Earthquake Analysis of Structure-Foundation Systems," by A.K. Vaish and A.K. Chopra - 1973 (AD 766 272)A07

EERC 73-10 "Deconvolution of Seismic Response for Linear Systems," by R.B. Reimer - 1973 (PB 227 179)A08

EERC 73-11 "SAP IV: A Structural Analysis Program for Static and Dynamic Response of Linear Systems," by K.-J. Bathe, E.L. Wilson and F.E. Peterson - 1973 (PB 221 967)A09

EERC 73-12 "Analytical Investigations of the Seismic Response of Long, Multiple Span Highway Bridges," by W.S. Tseng and J. Penzien - 1973 (PB 227 816)A10

EERC 73-13 "Earthquake Analysis of Multi-Story Buildings Including Foundation Interaction," by A.K. Chopra and J.A. Gutierrez - 1973 (PB 222 970)A03

EERC 73-14 "ADAP: A Computer Program for Static and Dynamic Analysis of Arch Dams," by R.W. Clough, J.M. Raphael and S. Mojtahedi - 1973 (PB 223 763)A09

EERC 73-15 "Cyclic Plastic Analysis of Structural Steel Joints," by R.B. Pinkney and R.W. Clough - 1973 (PB 226 843)A08

EERC 73-16 "QUAD-4: A Computer Program for Evaluating the Seismic Response of Soil Structures by Variable Damping Finite Element Procedures," by I.M. Idriss, J. Lysmer, R. Hwang and H.B. Seed - 1973 (PB 229 424)A05

EERC 73-17 "Dynamic Behavior of a Multi-Story Pyramid Shaped Building," by R.M. Stephen, J.P. Hollings and J.G. Bouwkamp - 1973 (PB 240 718)A06

EERC 73-18 "Effect of Different Types of Reinforcing on Seismic Behavior of Short Concrete Columns," by V.V. Bertero, J. Hollings, O. Küstü, R.M. Stephen and J.G. Bouwkamp - 1973

EERC 73-19 "Olive View Medical Center Materials Studies, Phase I," by B. Bresler and V.V. Bertero - 1973 (PB 235 986)A06

EERC 73-20 "Linear and Nonlinear Seismic Analysis Computer Programs for Long Multiple-Span Highway Bridges," by W.S. Tseng and J. Penzien - 1973

EERC 73-21 "Constitutive Models for Cyclic Plastic Deformation of Engineering Materials," by J.M. Kelly and P.P. Gillis 1973 (PB 226 024)A03

EERC 73-22 "DRAIN - 2D User's Guide," by G.H. Powell - 1973 (PB 227 016)A05

EERC 73-23 "Earthquake Engineering at Berkeley - 1973," (PB 226 033)A11

EERC 73-24 Unassigned

EERC 73-25 "Earthquake Response of Axisymmetric Tower Structures Surrounded by Water," by C.Y. Liaw and A.K. Chopra 1973 (AD 773 052)A09

EERC 73-26 "Investigation of the Failures of the Olive View Stairtowers During the San Fernando Earthquake and Their Implications on Seismic Design," by V.V. Bertero and R.G. Collins - 1973 (PB 235 106)A13

EERC 73-27 "Further Studies on Seismic Behavior of Steel Beam-Column Subassemblages," by V.V. Bertero, H. Krawinkler and E.P. Popov - 1973 (PB 234 172)A06

EERC 74-1 "Seismic Risk Analysis," by C.S. Oliveira - 1974 (PB 235 920)A06

EERC 74-2 "Settlement and Liquefaction of Sands Under Multi-Directional Shaking," by R. Pyke, C.K. Chan and H.B. Seed 1974

EERC 74-3 "Optimum Design of Earthquake Resistant Shear Buildings," by D. Ray, K.S. Pister and A.K. Chopra - 1974 (PB 231 172)A06

EERC 74-4 "LUSH - A Computer Program for Complex Response Analysis of Soil-Structure Systems," by J. Lysmer, T. Udaka, H.B. Seed and R. Hwang - 1974 (PB 236 796)A05

EERC 74-5 "Sensitivity Analysis for Hysteretic Dynamic Systems: Applications to Earthquake Engineering," by D. Ray 1974 (PB 233 213)A06

EERC 74-6 "Soil Structure Interaction Analyses for Evaluating Seismic Response," by H.B. Seed, J. Lysmer and R. Hwang 1974 (PB 236 519)A04

EERC 74-7 Unassigned

EERC 74-8 "Shaking Table Tests of a Steel Frame - A Progress Report," by R.W. Clough and D. Tang - 1974 (PB 240 869)A03

EERC 74-9 "Hysteretic Behavior of Reinforced Concrete Flexural Members with Special Web Reinforcement," by V.V. Bertero, E.P. Popov and T.Y. Wang - 1974 (PB 236 797)A07

EERC 74-10 "Applications of Reliability-Based, Global Cost Optimization to Design of Earthquake Resistant Structures," by E. Vitiello and K.S. Pister - 1974 (PB 237 231)A06

EERC 74-11 "Liquefaction of Gravelly Soils Under Cyclic Loading Conditions," by R.T. Wong, H.B. Seed and C.K. Chan 1974 (PB 242 042)A03

EERC 74-12 "Site-Dependent Spectra for Earthquake-Resistant Design," by H.B. Seed, C. Ugas and J. Lysmer - 1974 (PB 240 953)A03

EERC 74-13 "Earthquake Simulator Study of a Reinforced Concrete Frame," by P. Hidalgo and R.W. Clough - 1974 (PB 241 944)A13

EERC 74-14 "Nonlinear Earthquake Response of Concrete Gravity Dams," by N. Pal - 1974 (AD/A 006 583)A06

EERC 74-15 "Modeling and Identification in Nonlinear Structural Dynamics - I. One Degree of Freedom Models," by N. Distefano and A. Rath - 1974 (PB 241 548)A06

EERC 75-1 "Determination of Seismic Design Criteria for the Dumbarton Bridge Replacement Structure, Vol. I: Description, Theory and Analytical Modeling of Bridge and Parameters," by F. Baron and S.-H. Pang - 1975 (PB 259 407)A15

EERC 75-2 "Determination of Seismic Design Criteria for the Dumbarton Bridge Replacement Structure, Vol. II: Numerical Studies and Establishment of Seismic Design Criteria," by F. Baron and S.-H. Pang - 1975 (PB 259 408)A11 (For set of EERC 75-1 and 75-2 (PB 259 406))

EERC 75-3 "Seismic Risk Analysis for a Site and a Metropolitan Area," by C.S. Oliveira - 1975 (PB 248 134)A09

EERC 75-4 "Analytical Investigations of Seismic Response of Short, Single or Multiple-Span Highway Bridges," by M.-C. Chen and J. Penzien - 1975 (PB 241 454)A09

EERC 75-5 "An Evaluation of Some Methods for Predicting Seismic Behavior of Reinforced Concrete Buildings," by S.A. Mahin and V.V. Bertero - 1975 (PB 246 306)A16

EERC 75-6 "Earthquake Simulator Study of a Steel Frame Structure, Vol. I: Experimental Results," by R.W. Clough and D.T. Tang - 1975 (PB 243 981)A13

EERC 75-7 "Dynamic Properties of San Bernardino Intake Tower," by D. Rea, C.-Y. Liaw and A.K. Chopra - 1975 (AD/A008 406) A05

EERC 75-8 "Seismic Studies of the Articulation for the Dumbarton Bridge Replacement Structure, Vol. I: Description, Theory and Analytical Modeling of Bridge Components," by F. Baron and R.E. Hamati - 1975 (PB 251 539)A07

EERC 75-9 "Seismic Studies of the Articulation for the Dumbarton Bridge Replacement Structure, Vol. 2: Numerical Studies of Steel and Concrete Girder Alternates," by F. Baron and R.E. Hamati - 1975 (PB 251 540)A10

EERC 75-10 "Static and Dynamic Analysis of Nonlinear Structures," by D.P. Mondkar and G.H. Powell - 1975 (PB 242 434)A08

EERC 75-11 "Hysteretic Behavior of Steel Columns," by E.P. Popov, V.V. Bertero and S. Chandramouli - 1975 (PB 252 365)A11

EERC 75-12 "Earthquake Engineering Research Center Library Printed Catalog," - 1975 (PB 243 711)A26

EERC 75-13 "Three Dimensional Analysis of Building Systems (Extended Version)," by E.L. Wilson, J.P. Hollings and H.H. Dovey - 1975 (PB 243 989)A07

EERC 75-14 "Determination of Soil Liquefaction Characteristics by Large-Scale Laboratory Tests," by P. De Alba, C.K. Chan and H.B. Seed - 1975 (NUREG 0027)A08

EERC 75-15 "A Literature Survey - Compressive, Tensile, Bond and Shear Strength of Masonry," by R.L. Mayes and R.W. Clough - 1975 (PB 246 292)A10

EERC 75-16 "Hysteretic Behavior of Ductile Moment Resisting Reinforced Concrete Frame Components," by V.V. Bertero and E.P. Popov - 1975 (PB 246 388)A05

EERC 75-17 "Relationships Between Maximum Acceleration, Maximum Velocity, Distance from Source, Local Site Conditions for Moderately Strong Earthquakes," by H.B. Seed, R. Murarka, J. Lysmer and I.M. Idriss - 1975 (PB 248 172)A03

EERC 75-18 "The Effects of Method of Sample Preparation on the Cyclic Stress-Strain Behavior of Sands," by J. Mulilis, C.K. Chan and H.B. Seed - 1975 (Summarized in EERC 75-28).

EERC 75-19 "The Seismic Behavior of Critical Regions of Reinforced Concrete Components as Influenced by Moment, Shear and Axial Force," by M.B. Atalay and J. Penzien - 1975 (PB 258 842)A11

EERC 75-20 "Dynamic Properties of an Eleven Story Masonry Building," by R.M. Stephen, J.P. Hollings, J.G. Bouwkamp and D. Jurukovski - 1975 (PB 246 945)A04

EERC 75-21 "State-of-the-Art in Seismic Strength of Masonry - An Evaluation and Review," by R.L. Mayes and R.W. Clough 1975 (PB 249 040)A07

EERC 75-22 "Frequency Dependent Stiffness Matrices for Viscoelastic Half-Plane Foundations," by A.K. Chopra, P. Chakrabarti and G. Dasgupta - 1975 (PB 248 121)A07

EERC 75-23 "Hysteretic Behavior of Reinforced Concrete Framed Walls," by T.Y. Wong, V.V. Bertero and E.P. Popov - 1975

EERC 75-24 "Testing Facility for Subassemblages of Frame-Wall Structural Systems," by V.V. Bertero, E.P. Popov and T. Endo - 1975

EERC 75-25 "Influence of Seismic History on the Liquefaction Characteristics of Sands," by H.B. Seed, K. Mori and C.K. Chan - 1975 (Summarized in EERC 75-28)

EERC 75-26 "The Generation and Dissipation of Pore Water Pressures during Soil Liquefaction," by H.B. Seed, P.P. Martin and J. Lysmer - 1975 (PB 252 648)A03

EERC 75-27 "Identification of Research Needs for Improving Aseismic Design of Building Structures," by V.V. Bertero 1975 (PB 248 136)A05

EERC 75-28 "Evaluation of Soil Liquefaction Potential during Earthquakes," by H.B. Seed, I. Arango and C.K. Chan - 1975 (NUREG 0026)A13

EERC 75-29 "Representation of Irregular Stress Time Histories by Equivalent Uniform Stress Series in Liquefaction Analyses," by H.B. Seed, I.M. Idriss, F. Makdisi and N. Banerjee - 1975 (PB 252 635)A03

EERC 75-30 "FLUSH - A Computer Program for Approximate 3-D Analysis of Soil-Structure Interaction Problems," by J. Lysmer, T. Udaka, C.-F. Tsai and H.B. Seed - 1975 (PB 259 332)A07

EERC 75-31 "ALUSH - A Computer Program for Seismic Response Analysis of Axisymmetric Soil-Structure Systems," by E. Berger, J. Lysmer and H.B. Seed - 1975

EERC 75-32 "TRIP and TRAVEL - Computer Programs for Soil-Structure Interaction Analysis with Horizontally Travelling Waves," by T. Udaka, J. Lysmer and H.B. Seed - 1975

EERC 75-33 "Predicting the Performance of Structures in Regions of High Seismicity," by J. Penzien - 1975 (PB 248 130)A03

EERC 75-34 "Efficient Finite Element Analysis of Seismic Structure - Soil - Direction," by J. Lysmer, H.B. Seed, T. Udaka, R.N. Hwang and C.-F. Tsai - 1975 (PB 253 570)A03

EERC 75-35 "The Dynamic Behavior of a First Story Girder of a Three-Story Steel Frame Subjected to Earthquake Loading," by R.W. Clough and L.-Y. Li - 1975 (PB 248 841)A05

EERC 75-36 "Earthquake Simulator Study of a Steel Frame Structure, Volume II - Analytical Results," by D.T. Tang - 1975 (PB 252 926)A10

EERC 75-37 "ANSR-I General Purpose Computer Program for Analysis of Non-Linear Structural Response," by D.P. Mondkar and G.H. Powell - 1975 (PB 252 386)A08

EERC 75-38 "Nonlinear Response Spectra for Probabilistic Seismic Design and Damage Assessment of Reinforced Concrete Structures," by M. Murakami and J. Penzien - 1975 (PB 259 530)A05

EERC 75-39 "Study of a Method of Feasible Directions for Optimal Elastic Design of Frame Structures Subjected to Earthquake Loading," by N.D. Walker and K.S. Pister - 1975 (PB 257 781)A06

EERC 75-40 "An Alternative Representation of the Elastic-Viscoelastic Analogy," by G. Dasgupta and J.L. Sackman - 1975 (PB 252 173)A03

EERC 75-41 "Effect of Multi-Directional Shaking on Liquefaction of Sands," by H.B. Seed, R. Pyke and G.R. Martin - 1975 (PB 258 781)A03

EERC 76-1 "Strength and Ductility Evaluation of Existing Low-Rise Reinforced Concrete Buildings - Screening Method," by T. Okada and B. Bresler - 1976 (PB 257 906)A11

EERC 76-2 "Experimental and Analytical Studies on the Hysteretic Behavior of Reinforced Concrete Rectangular and T-Beams," by S.-Y.M. Ma, E.P. Popov and V.V. Bertero - 1976 (PB 260 843)A12

EERC 76-3 "Dynamic Behavior of a Multistory Triangular-Shaped Building," by J. Petrovski, R.M. Stephen, E. Gartenbaum and J.G. Bouwkamp - 1976 (PB 273 279)A07

EERC 76-4 "Earthquake Induced Deformations of Earth Dams," by N. Serff, H.B. Seed, F.I. Makdisi & C.-Y. Chang - 1976 (PB 292 065)A08

EERC 76-5 "Analysis and Design of Tube-Type Tall Building Structures," by H. de Clercq and G.H. Powell - 1976 (PB 252 220) A10

EERC 76-6 "Time and Frequency Domain Analysis of Three-Dimensional Ground Motions, San Fernando Earthquake," by T. Kubo and J. Penzien (PB 260 556)A11

EERC 76-7 "Expected Performance of Uniform Building Code Design Masonry Structures," by R.L. Mayes, Y. Omote, S.W. Chen and R.W. Clough - 1976 (PB 270 098)A05

EERC 76-8 "Cyclic Shear Tests of Masonry Piers, Volume 1 - Test Results," by R.L. Mayes, Y. Omote, R.W. Clough - 1976 (PB 264 424)A06

EERC 76-9 "A Substructure Method for Earthquake Analysis of Structure - Soil Interaction," by J.A. Gutierrez and A.K. Chopra - 1976 (PB 257 783)A08

EERC 76-10 "Stabilization of Potentially Liquefiable Sand Deposits using Gravel Drain Systems," by H.B. Seed and J.R. Booker - 1976 (PB 258 820)A04

EERC 76-11 "Influence of Design and Analysis Assumptions on Computed Inelastic Response of Moderately Tall Frames," by G.H. Powell and D.G. Row - 1976 (PB 271 409)A06

EERC 76-12 "Sensitivity Analysis for Hysteretic Dynamic Systems: Theory and Applications," by D. Ray, K.S. Pister and E. Polak - 1976 (PB 262 859)A04

EERC 76-13 "Coupled Lateral Torsional Response of Buildings to Ground Shaking," by C.L. Kan and A.K. Chopra - 1976 (PB 257 907)A09

EERC 76-14 "Seismic Analyses of the Banco de America," by V.V. Bertero, S.A. Mahin and J.A. Hollings - 1976

EERC 76-15 "Reinforced Concrete Frame 2: Seismic Testing and Analytical Correlation," by R.W. Clough and J. Gidwani - 1976 (PB 261 323)A08

EERC 76-16 "Cyclic Shear Tests of Masonry Piers, Volume 2 - Analysis of Test Results," by R.L. Mayes, Y. Omote and R.W. Clough - 1976

EERC 76-17 "Structural Steel Bracing Systems: Behavior Under Cyclic Loading," by E.P. Popov, K. Takanashi and C.W. Roeder - 1976 (PB 260 715)A05

EERC 76-18 "Experimental Model Studies on Seismic Response of High Curved Overcrossings," by D. Williams and W.G. Godden - 1976 (PB 269 548)A08

EERC 76-19 "Effects of Non-Uniform Seismic Disturbances on the Dumbarton Bridge Replacement Structure," by F. Baron and R.E. Hamati - 1976 (PB 282 981)A16

EERC 76-20 "Investigation of the Inelastic Characteristics of a Single Story Steel Structure Using System Identification and Shaking Table Experiments," by V.C. Matzen and H.D. McNiven - 1976 (PB 258 453)A07

EERC 76-21 "Capacity of Columns with Splice Imperfections," by E.P. Popov, R.M. Stephen and R. Philbrick - 1976 (PB 260 378)A04

EERC 76-22 "Response of the Olive View Hospital Main Building during the San Fernando Earthquake," by S. A. Mahin, V.V. Bertero, A.K. Chopra and R. Collins - 1976 (PB 271 425)A14

EERC 76-23 "A Study on the Major Factors Influencing the Strength of Masonry Prisms," by N.M. Mostaghel, R.L. Mayes, R. W. Clough and S.W. Chen - 1976 (Not published)

EERC 76-24 "GADFLEA - A Computer Program for the Analysis of Pore Pressure Generation and Dissipation during Cyclic or Earthquake Loading," by J.R. Booker, M.S. Rahman and H.B. Seed - 1976 (PB 263 947)A04

EERC 76-25 "Seismic Safety Evaluation of a R/C School Building," by B. Bresler and J. Axley - 1976

EERC 76-26 "Correlative Investigations on Theoretical and Experimental Dynamic Behavior of a Model Bridge Structure," by K. Kawashima and J. Penzien - 1976 (PB 263 388)A11

EERC 76-27 "Earthquake Response of Coupled Shear Wall Buildings," by T. Srichatrapimuk - 1976 (PB 265 157)A07

EERC 76-28 "Tensile Capacity of Partial Penetration Welds," by E.P. Popov and R.M. Stephen - 1976 (PB 262 899)A03

EERC 76-29 "Analysis and Design of Numerical Integration Methods in Structural Dynamics," by H.M. Hilber - 1976 (PB 264 410)A06

EERC 76-30 "Contribution of a Floor System to the Dynamic Characteristics of Reinforced Concrete Buildings," by L.E. Malik and V.V. Bertero - 1976 (PB 272 247)A13

EERC 76-31 "The Effects of Seismic Disturbances on the Golden Gate Bridge," by F. Baron, M. Arikan and R.E. Hamati - 1976 (PB 272 279)A09

EERC 76-32 "Infilled Frames in Earthquake Resistant Construction," by R.E. Klingner and V.V. Bertero - 1976 (PB 265 892)A13

UCB/EERC-77/01 "PLUSH - A Computer Program for Probabilistic Finite Element Analysis of Seismic Soil-Structure Interaction," by M.P. Romo Organista, J. Lysmer and H.B. Seed - 1977

UCB/EERC-77/02 "Soil-Structure Interaction Effects at the Humboldt Bay Power Plant in the Ferndale Earthquake of June 7, 1975," by J.E. Valera, H.B. Seed, C.F. Tsai and J. Lysmer - 1977 (PB 265 795)A04

UCB/EERC-77/03 "Influence of Sample Disturbance on Sand Response to Cyclic Loading," by K. Mori, H.B. Seed and C.K. Chan - 1977 (PB 267 352)A04

UCB/EERC-77/04 "Seismological Studies of Strong Motion Records," by J. Shoja-Taheri - 1977 (PB 269 655)A10

UCB/EERC-77/05 "Testing Facility for Coupled-Shear Walls," by L. Li-Hyung, V.V. Bertero and E.P. Popov - 1977

UCB/EERC-77/06 "Developing Methodologies for Evaluating the Earthquake Safety of Existing Buildings," by No. 1 - B. Bresler; No. 2 - B. Bresler, T. Okada and D. Zisling; No. 3 - T. Okada and B. Bresler; No. 4 - V.V. Bertero and B. Bresler - 1977 (PB 267 354)A08

UCB/EERC-77/07 "A Literature Survey - Transverse Strength of Masonry Walls," by Y. Omote, R.L. Mayes, S.W. Chen and R.W. Clough - 1977 (PB 277 933)A07

UCB/EERC-77/08 "DRAIN-TABS: A Computer Program for Inelastic Earthquake Response of Three Dimensional Buildings," by R. Guendelman-Israel and G.H. Powell - 1977 (PB 270 693)A07

UCB/EERC-77/09 "SUBWALL: A Special Purpose Finite Element Computer Program for Practical Elastic Analysis and Design of Structural Walls with Substructure Option," by D.Q. Le, H. Peterson and E.P. Popov - 1977 (PB 270 567)A05

UCB/EERC-77/10 "Experimental Evaluation of Seismic Design Methods for Broad Cylindrical Tanks," by D.P. Clough (PB 272 280)A13

UCB/EERC-77/11 "Earthquake Engineering Research at Berkeley - 1976," - 1977 (PB 273 507)A09

UCB/EERC-77/12 "Automated Design of Earthquake Resistant Multistory Steel Building Frames," by N.D. Walker, Jr. - 1977 (PB 276 526)A09

UCB/EERC-77/13 "Concrete Confined by Rectangular Hoops Subjected to Axial Loads," by J. Vallenas, V.V. Bertero and E.P. Popov - 1977 (PB 275 165)A06

UCB/EERC-77/14 "Seismic Strain Induced in the Ground During Earthquakes," by Y. Sugimura - 1977 (PB 284 201)A04

UCB/EERC-77/15 "Bond Deterioration under Generalized Loading," by V.V. Bertero, E.P. Popov and S. Viwathanatepa - 1977

UCB/EERC-77/16 "Computer Aided Optimum Design of Ductile Reinforced Concrete Moment Resisting Frames," by S.W. Zagajeski and V.V. Bertero - 1977 (PB 280 137)A07

UCB/EERC-77/17 "Earthquake Simulation Testing of a Stepping Frame with Energy-Absorbing Devices," by J.M. Kelly and D.F. Tsztoo - 1977 (PB 273 506)A04

UCB/EERC-77/18 "Inelastic Behavior of Eccentrically Braced Steel Frames under Cyclic Loadings," by C.W. Roeder and E.P. Popov - 1977 (PB 275 526)A15

UCB/EERC-77/19 "A Simplified Procedure for Estimating Earthquake-Induced Deformations in Dams and Embankments," by F.I. Makdisi and H.B. Seed - 1977 (PB 276 820)A04

UCB/EERC-77/20 "The Performance of Earth Dams during Earthquakes," by H.B. Seed, F.I. Makdisi and P. de Alba - 1977 (PB 276 821)A04

UCB/EERC-77/21 "Dynamic Plastic Analysis Using Stress Resultant Finite Element Formulation," by P. Lukkunapvasit and J.M. Kelly - 1977 (PB 275 453)A04

UCB/EERC-77/22 "Preliminary Experimental Study of Seismic Uplift of a Steel Frame," by R.W. Clough and A.A. Huckelbridge 1977 (PB 278 769)A08

UCB/EERC-77/23 "Earthquake Simulator Tests of a Nine-Story Steel Frame with Columns Allowed to Uplift," by A.A. Huckelbridge - 1977 (PB 277 944)A09

UCB/EERC-77/24 "Nonlinear Soil-Structure Interaction of Skew Highway Bridges," by M.-C. Chen and J. Penzien - 1977 (PB 276 176)A07

UCB/EERC-77/25 "Seismic Analysis of an Offshore Structure Supported on Pile Foundations," by D.D.-N. Liou and J. Penzien 1977 (PB 283 180)A06

UCB/EERC-77/26 "Dynamic Stiffness Matrices for Homogeneous Viscoelastic Half-Planes," by G. Dasgupta and A.K. Chopra - 1977 (PB 279 654)A06

UCB/EERC-77/27 "A Practical Soft Story Earthquake Isolation System," by J.M. Kelly, J.M. Eidinger and C.J. Derham - 1977 (PB 276 814)A07

UCB/EERC-77/28 "Seismic Safety of Existing Buildings and Incentives for Hazard Mitigation in San Francisco: An Exploratory Study," by A.J. Meltsner - 1977 (PB 281 970)A05

UCB/EERC-77/29 "Dynamic Analysis of Electrohydraulic Shaking Tables," by D. Rea, S. Abedi-Hayati and Y. Takahashi 1977 (PB 282 569)A04

UCB/EERC-77/30 "An Approach for Improving Seismic - Resistant Behavior of Reinforced Concrete Interior Joints," by B. Galunic, V.V. Bertero and E.P. Popov - 1977 (PB 290 870)A06

UCB/EERC-78/01 "The Development of Energy-Absorbing Devices for Aseismic Base Isolation Systems," by J.M. Kelly and D.F. Tsztoo - 1978 (PB 284 978)A04

UCB/EERC-78/02 "Effect of Tensile Prestrain on the Cyclic Response of Structural Steel Connections, by J.G. Bouwkamp and A. Mukhopadhyay - 1978

UCB/EERC-78/03 "Experimental Results of an Earthquake Isolation System using Natural Rubber Bearings," by J.M. Eidinger and J.M. Kelly - 1978 (PB 281 686)A04

UCB/EERC-78/04 "Seismic Behavior of Tall Liquid Storage Tanks," by A. Niwa - 1978 (PB 284 017)A14

UCB/EERC-78/05 "Hysteretic Behavior of Reinforced Concrete Columns Subjected to High Axial and Cyclic Shear Forces," by S.W. Zagajeski, V.V. Bertero and J.G. Bouwkamp - 1978 (PB 283 858)A13

UCB/EERC-78/06 "Inelastic Beam-Column Elements for the ANSR-I Program," by A. Riahi, D.G. Row and G.H. Powell - 1978

UCB/EERC-78/07 "Studies of Structural Response to Earthquake Ground Motion," by O.A. Lopez and A.K. Chopra - 1978 (PB 282 790)A05

UCB/EERC-78/08 "A Laboratory Study of the Fluid-Structure Interaction of Submerged Tanks and Caissons in Earthquakes," by R.C. Byrd - 1978 (PB 284 957)A08

UCB/EERC-78/09 "Model for Evaluating Damageability of Structures," by I. Sakamoto and B. Bresler - 1978

UCB/EERC-78/10 "Seismic Performance of Nonstructural and Secondary Structural Elements," by I. Sakamoto - 1978

UCB/EERC-78/11 "Mathematical Modelling of Hysteresis Loops for Reinforced Concrete Columns," by S. Nakata, T. Sproul and J. Penzien - 1978

UCB/EERC-78/12 "Damageability in Existing Buildings," by T. Blejwas and B. Bresler - 1978

UCB/EERC-78/13 "Dynamic Behavior of a Pedestal Base Multistory Building," by R.M. Stephen, E.L. Wilson, J.G. Bouwkamp and M. Button - 1978 (PB 286 650)A08

UCB/EERC-78/14 "Seismic Response of Bridges - Case Studies," by R.A. Imbsen, V. Nutt and J. Penzien - 1978 (PB 286 503)A10

UCB/EERC-78/15 "A Substructure Technique for Nonlinear Static and Dynamic Analysis," by D.G. Row and G.H. Powell - 1978 (PB 288 077)A10

UCB/EERC-78/16 "Seismic Risk Studies for San Francisco and for the Greater San Francisco Bay Area," by C.S. Oliveira - 1978

UCB/EERC-78/17 "Strength of Timber Roof Connections Subjected to Cyclic Loads," by P. Gülkan, R.L. Mayes and R.W. Clough - 1978

UCB/EERC-78/18 "Response of K-Braced Steel Frame Models to Lateral Loads," by J.G. Bouwkamp, R.M. Stephen and E.P. Popov - 1978

UCB/EERC-78/19 "Rational Design Methods for Light Equipment in Structures Subjected to Ground Motion," by J.L. Sackman and J.M. Kelly - 1978 (PB 292 357)A04

UCB/EERC-78/20 "Testing of a Wind Restraint for Aseismic Base Isolation," by J.M. Kelly and D.E. Chitty - 1978 (PB 292 833)A03

UCB/EERC-78/21 "APOLLO - A Computer Program for the Analysis of Pore Pressure Generation and Dissipation in Horizontal Sand Layers During Cyclic or Earthquake Loading," by P.P. Martin and H.B. Seed - 1978 (PB 292 835)A04

UCB/EERC-78/22 "Optimal Design of an Earthquake Isolation System," by M.A. Bhatti, K.S. Pister and E. Polak - 1978 (PB 294 735)A06

UCB/EERC-78/23 "MASH - A Computer Program for the Non-Linear Analysis of Vertically Propagating Shear Waves in Horizontally Layered Deposits," by P.P. Martin and H.B. Seed - 1978 (PB 293 101)A05

UCB/EERC-78/24 "Investigation of the Elastic Characteristics of a Three Story Steel Frame Using System Identification," by I. Kaya and H.D. McNiven - 1978

UCB/EERC-78/25 "Investigation of the Nonlinear Characteristics of a Three-Story Steel Frame Using System Identification," by I. Kaya and H.D. McNiven - 1978

UCB/EERC-78/26 "Studies of Strong Ground Motion in Taiwan," by Y.M. Hsiung, B.A. Bolt and J. Penzien - 1978

UCB/EERC-78/27 "Cyclic Loading Tests of Masonry Single Piers: Volume 1 - Height to Width Ratio of 2," by P.A. Hidalgo, R.L. Mayes, H.D. McNiven and R.W. Clough - 1978

UCB/EERC-78/28 "Cyclic Loading Tests of Masonry Single Piers: Volume 2 - Height to Width Ratio of 1," by S.-W.J. Chen, P.A. Hidalgo, R.L. Mayes, R.W. Clough and H.D. McNiven - 1978

UCB/EERC-78/29 "Analytical Procedures in Soil Dynamics," by J. Lysmer - 1978

UCB/EERC-79/01 "Hysteretic Behavior of Lightweight Reinforced Concrete Beam-Column Subassemblages," by B. Forzani, E.P. Popov and V.V. Bertero - April 1979(PB 298 267)A06

UCB/EERC-79/02 "The Development of a Mathematical Model to Predict the Flexural Response of Reinforced Concrete Beams to Cyclic Loads, Using System Identification," by J. Stanton & H. McNiven - Jan. 1979(PB 295 875)A10

UCB/EERC-79/03 "Linear and Nonlinear Earthquake Response of Simple Torsionally Coupled Systems," by C.L. Kan and A.K. Chopra - Feb. 1979(PB 298 262)A06

UCB/EERC-79/04 "A Mathematical Model of Masonry for Predicting its Linear Seismic Response Characteristics," by Y. Mengi and H.D. McNiven - Feb. 1979(PB 298 266)A06

UCB/EERC-79/05 "Mechanical Behavior of Lightweight Concrete Confined by Different Types of Lateral Reinforcement," by M.A. Manrique, V.V. Bertero and E.P. Popov - May 1979(PB 301 114)A06

UCB/EERC-79/06 "Static Tilt Tests of a Tall Cylindrical Liquid Storage Tank," by R.W. Clough and A. Niwa - Feb. 1979 (PB 301 167)A06

UCB/EERC-79/07 "The Design of Steel Energy Absorbing Restrainers and Their Incorporation into Nuclear Power Plants for Enhanced Safety: Volume 1 - Summary Report," by P.N. Spencer, V.F. Zackay, and E.R. Parker - Feb. 1979(UCB/EERC-79/07)A09

UCB/EERC-79/08 "The Design of Steel Energy Absorbing Restrainers and Their Incorporation into Nuclear Power Plants for Enhanced Safety: Volume 2 - The Development of Analyses for Reactor System Piping,""Simple Systems" by M.C. Lee, J. Penzien, A.K. Chopra and K, Suzuki "Complex Systems" by G.H. Powell, E.L. Wilson, R.W. Clough and D.G. Row - Feb. 1979(UCB/EERC-79/08)A10

UCB/EERC-79/09 "The Design of Steel Energy Absorbing Restrainers and Their Incorporation into Nuclear Power Plants for Enhanced Safety: Volume 3 - Evaluation of Commercial Steels," by W.S. Owen, R.M.N. Pelloux, R.O. Ritchie, M. Faral, T. Ohhashi, J. Toplosky, S.J. Hartman, V.F. Zackay and E.R. Parker - Feb. 1979(UCB/EERC-79/09)A04

UCB/EERC-79/10 "The Design of Steel Energy Absorbing Restrainers and Their Incorporation into Nuclear Power Plants for Enhanced Safety: Volume 4 - A Review of Energy-Absorbing Devices," by J.M. Kelly and M.S. Skinner - Feb. 1979(UCB/EERC-79/10)A04

UCB/EERC-79/11 "Conservatism In Summation Rules for Closely Spaced Modes," by J.M. Kelly and J.L. Sackman - May 1979(PB 301 328)A03

UCB/EERC-79/12 "Cyclic Loading Tests of Masonry Single Piers; Volume 3 - Height to Width Ratio of 0.5," by P.A. Hidalgo, R.L. Mayes, H.D. McNiven and R.W. Clough - May 1979(PB 301 321)A08

UCB/EERC-79/13 "Cyclic Behavior of Dense Course-Grained Materials in Relation to the Seismic Stability of Dams," by N.G. Banerjee, H.B. Seed and C.K. Chan - June 1979(PB 301 373)A13

UCB/EERC-79/14 "Seismic Behavior of Reinforced Concrete Interior Beam-Column Subassemblages," by S. Viwathanatepa, E.P. Popov and V.V. Bertero - June 1979(PB 301 326)A10

UCB/EERC-79/15 "Optimal Design of Localized Nonlinear Systems with Dual Performance Criteria Under Earthquake Excitations," by M.A. Bhatti - July 1979(PB 80 167 109)A06

UCB/EERC-79/16 "OPTDYN - A General Purpose Optimization Program for Problems with or without Dynamic Constraints," by M.A. Bhatti, E. Polak and K.S. Pister - July 1979(PB 80 167 091)A05

UCB/EERC-79/17 "ANSR-II, Analysis of Nonlinear Structural Response, Users Manual," by D.P. Mondkar and G.H. Powell - July 1979(PB 80 113 301)A05

UCB/EERC-79/18 "Soil Structure Interaction in Different Seismic Environments," A. Gomez-Masso, J. Lysmer, J.-C. Chen and H.B. Seed - August 1979(PB 80 101 520)A04

UCB/EERC-79/19 "ARMA Models for Earthquake Ground Motions," by M.K. Chang, J.W. Kwiatkowski, R.F. Nau, R.M. Oliver and K.S. Pister - July 1979(PB 301 166)A05

UCB/EERC-79/20 "Hysteretic Behavior of Reinforced Concrete Structural Walls," by J.M. Vallenas, V.V. Bertero and E.P. Popov - August 1979(PB 80 165 905)A12

UCB/EERC-79/21 "Studies on High-Frequency Vibrations of Buildings - 1: The Column Effect," by J. Lubliner - August 1979 (PB 80 158 553)A03

UCB/EERC-79/22 "Effects of Generalized Loadings on Bond Reinforcing Bars Embedded in Confined Concrete Blocks," by S. Viwathanatepa, E.P. Popov and V.V. Bertero - August 1979

UCB/EERC-79/23 "Shaking Table Study of Single-Story Masonry Houses, Volume 1: Test Structures 1 and 2," by P. Gülkan, R.L. Mayes and R.W. Clough - Sept. 1979

UCB/EERC-79/24 "Shaking Table Study of Single-Story Masonry Houses, Volume 2: Test Structures 3 and 4," by P. Gülkan, R.L. Mayes and R.W. Clough - Sept. 1979

UCB/EERC-79/25 "Shaking Table Study of Single-Story Masonry Houses, Volume 3: Summary, Conclusions and Recommendations," by R.W. Clough, R.L. Mayes and P. Gülkan - Sept. 1979

UCB/EERC-79/26 "Recommendations for a U.S.-Japan Cooperative Research Program Utilizing Large-Scale Testing Facilities," by U.S.-Japan Planning Group - Sept. 1979(PB 301 407)A06

UCB/EERC-79/27 "Earthquake-Induced Liquefaction Near Lake Amatitlan, Guatemala," by H.B. Seed, I. Arango, C.K. Chan, A. Gomez-Masso and R. Grant de Ascoli - Sept. 1979(NUREG-CR1341)A03

UCB/EERC-79/28 "Infill Panels: Their Influence on Seismic Response of Buildings," by J.W. Axley and V.V. Bertero - Sept. 1979(PB 80 163 371)A10

UCB/EERC-79/29 "3D Truss Bar Element (Type 1) for the ANSR-II Program," by D.P. Mondkar and G.H. Powell - Nov. 1979 (PB 80 169 709)A02

UCB/EERC-79/30 "2D Beam-Column Element (Type 5 - Parallel Element Theory) for the ANSR-II Program," by D.G. Row, G.H. Powell and D.P. Mondkar - Dec. 1979(PB 80 167 224)A03

UCB/EERC-79/31 "3D Beam-Column Element (Type 2 - Parallel Element Theory) for the ANSR-II Program," by A. Riahi, G.H. Powell and D.P. Mondkar - Dec. 1979(PB 80 167 216)A03

UCB/EERC-79/32 "On Response of Structures to Stationary Excitation," by A. Der Kiureghian - Dec. 1979(PB 80166 929)A03

UCB/EERC-79/33 "Undisturbed Sampling and Cyclic Load Testing of Sands," by S. Singh, H.B. Seed and C.K. Chan - Dec. 1979(

UCB/EERC-79/34 "Interaction Effects of Simultaneous Torsional and Compressional Cyclic Loading of Sand," by P.M. Griffin and W.N. Houston - Dec. 1979

UCB/EERC-80/01 "Earthquake Response of Concrete Gravity Dams Including Hydrodynamic and Foundation Interaction Effects," by A.K. Chopra, P. Chakrabarti and S. Gupta - 1980

UCB/EERC-80/02 "Rocking Response of Rigid Blocks to Earthquakes," by C.S. Yim, A.K. Chopra and J. Penzien - 1980

UCB/EERC-80/03 "Optimum Inelastic Design of Seismic-Resistant Reinforced Concrete Frame Structures," by S.W. Zagajeski and V.V. Bertero - 1980

UCB/EERC-80/04 "Effects of Amount and Arrangement of Wall-Panel Reinforcement on Hysteretic Behavior of Reinforced Concrete Walls," by R. Iliya and V.V. Bertero - 1980

UCB/EERC-80/05 "Shaking Table Research on Concrete Dam Models," by A. Niwa and R.W. Clough - 1980

UCB/EERC-80/06 "Piping With Energy Absorbing Restrainers: Parameter Study on Small Systems," by G.H. Powell, C. Oughourlian and J. Simons - 1980

UCB/EERC-80/07 "Inelastic Torsional Response of Structures Subjected to Earthquake Ground Motions," by Y. Yamazaki - 1980

UCB/EERC-80/08 "Study of X-Braced Steel Frame Structures Under Earthquake Simulation," by Y. Ghanaat - 1980

UCB/EERC-80/09 "Hybrid Modelling of Soil-Structure Interaction," by S. Gupta, T.W. Lin, J. Penzien and C.S. Yeh - 1980

UCB/EERC-80/10 "General Applicability of a Nonlinear Model of a One Story Steel Frame," by B.I. Sveinsson and H. McNiven - 1980

UCB/EERC-80/11 "A Green-Function Method for Wave Interaction with a Submerged Body," by W. Kioka - 1980

UCB/EERC-80/12 "Hydrodynamic Pressure and Added Mass for Axisymmetric Bodies," by F. Nilrat - 1980

UCB/EERC-80/13 "Treatment of Non-Linear Drag Forces Acting on Offshore Platforms," by B.V. Dao and J. Penzien - 1980

UCB/EERC-80/14 "2D Plane/Axisymmetric Solid Element (Type 3 - Elastic or Elastic-Perfectly Plastic) for the ANSR-II Program," by D.P. Mondkar and G.H. Powell - 1980

UCB/EERC-80/15 "A Response Spectrum Method for Random Vibrations," by A. Der Kiureghian - 1980

UCB/EERC-80/16 "Cyclic Inelastic Buckling of Tubular Steel Braces," by V.A. Zayas, E.P. Popov and S.A. Mahin - 1980

UCB/EERC-80/17 "Dynamic Response of Simple Arch Dams Including Hydrodynamic Interaction," by C.S. Porter and A.K. Chopra - 1980

UCB/EERC-80/18 "Experimental Testing of a Friction Damped Aseismic Base Isolation System with Fail-Safe Characteristics," by J.M. Kelly, K.E. Beucke and M.S. Skinner - 1980

UCB/EERC-80/19 "The Design of Steel Energy-Absorbing Restrainers and their Incorporation into Nuclear Power Plants for Enhanced Safety (Vol 1B): Stochastic Seismic Analyses of Nuclear Power Plant Structures and Piping Systems Subjected to Multiple Support Excitations," by M.C. Lee and J. Penzien - 1980

UCB/EERC-80/20 "The Design of Steel Energy-Absorbing Restrainers and their Incorporation into Nuclear Power Plants for Enhanced Safety (Vol 1C): Numerical Method for Dynamic Substructure Analysis," by J.M. Dickens and E.L. Wilson 1980

UCB/EERC-80/21 "The Design of Steel Energy-Absorbing Restrainers and their Incorporation into Nuclear Power Plants for Enhanced Safety (Vol 2): Development and Testing of Restraints for Nuclear Piping Systems," by J.M. Kelly and M.S. Skinner - 1980

UCB/EERC-80/22 "3D Solid Element (Type 4-Elastic or Elastic-Perfectly-Plastic) for the ANSR-II Program," by D.P. Mondkar and G.H. Powell - 1980

UCB/EERC-80/23 "Gap-Friction Element (Type 5) for the ANSR-II Program," by D.P. Mondkar and G.H. Powell - 1980

UCB/EERC-80/24 "U-Bar Restraint Element (Type 11) for the ANSR-II Program," by C. Oughourlian and G.H. Powell - 1980

UCB/EERC-80/25 "Testing of a Natural Rubber Base Isolation System by an Explosively Simulated Earthquake," by J.M. Kelly 1980

UCB/EERC-80/26 "Input Identification from Structural Vibrational Response," by Y. Hu - 1980

UCB/EERC-80/27 "Cyclic Inelastic Behavior of Steel Offshore Structures," by V.A. Zayas, S.A. Mahin and E.P. Popov - 1980

UCB/EERC-80/28 "Shaking Table Testing of a Reinforced Concrete Frame with Biaxial Response," by M.G. Oliva and R.W. Clough 1980

UCB/EERC-80/29 "Dynamic Properties of a Twelve-Story Prefabricated Panel Building," by J.G. Bouwkamp, J.P. Kollegger and R.M. Stephen - 1980

UCB/EERC-80/30 "Dynamic Properties of a Eight-Story Prefabricated Panel Building," by J.G. Bouwkamp, J.P. Kollegger and R.M. Stephen - 1980

UCB/EERC-80/31 "Predictive Dynamic Response of Panel Type Structures Under Earthquakes," by J.P. Kollegger and J.G. Bouwkamp 1980

UCB/EERC-80/32 "The Design of Steel Energy-Absorbing Restrainers and their Incorporation into Nuclear Power Plants for Enhanced Safety: Vol 3, Testing of Commercial Steels in Low-Cycle Torsional Fatigue," by P. Spencer, E.R. Parker, E. Jongewaard and M. Drory - 1980

UCB/EERC-80/33 "The Design of Steel Energy-Absorbing Restrainers and their Incorporation into Nuclear Power Plants for Enhanced Safety: Vol 4, Shaking Table Tests of Piping Systems with Energy-Absorbing Restrainers," by S.F. Stiemer and W.G. Godden - 1980

UCB/EERC-80/34 "The Design of Steel Energy-Absorbing Restrainers and their Incorporation into Nuclear Power Plants for Enhanced Safety: Vol 5, Summary Report," by P. Spencer 1980

UCB/EERC-80/35 "Experimental Testing of an Energy Absorbing Base Isolation System," by J. Kelly, M.S. Skinner and K.E. Beucke - 1980

UCB/EERC-80/36 "Simulating and Analyzing Artificial Non-Stationary Earthquake Ground Motions," by R.F. Nau, R.M. Oliver and K.S. Pister - 1980

UCB/EERC-80/37 "Earthquake Engineering at Berkeley," - 1980

UCB/EERC-80/38 "Inelastic Seismic Analysis of Large Panel Buildings," by V. Schricker and G.H. Powell - 1980

UCB/EERC-80/39 "Dynamic Response of Embankment, Concrete-Gravity and Arch Dams Including Hydrodynamic Interaction," by J.F. Hall and A.K. Chopra - 1980

UCB/EERC-80/40 "Inelastic Buckling of Steel Strut Under Cyclic Load Reversal," by R.G. Black, W.A. Wenger and E.P. Popov 1980

UCB/EERC-80/41 "Influence of Site Characteristics on Building Damage During the October 3, 1974 Lima Earthquake," by P. Repetto, I. Arango and H.B. Seed - 1980

UCB/EERC-80/42 "Evaluation of a Shaking Table Test Program on Response Behavior of a Two Story Reinforced Concrete Frame," by J.M. Blondet, R.W. Clough and S.A. Mahin - 1980.

UCB/EERC-80/43 "Modelling of Soil-Structure Interaction by Finite and Infinite Element," by F. Medina - 1980

UCB/EERC-81/01 "Control of Seismic Response of Piping Systems and Other Structures by Base Isolation," edited by J.M. Kelly - 1981

UCB/EERC-81/02 "OPTNSR - An Interactive Software System for Optimal Design of Statically and Dynamically Loaded Structures with Nonlinear Response," by M.A. Bhatti, V. Ciampi and K.S. Pister - 1981