REPORT NO.
UCB/EERC-87/13
SEPTEMBER 1987

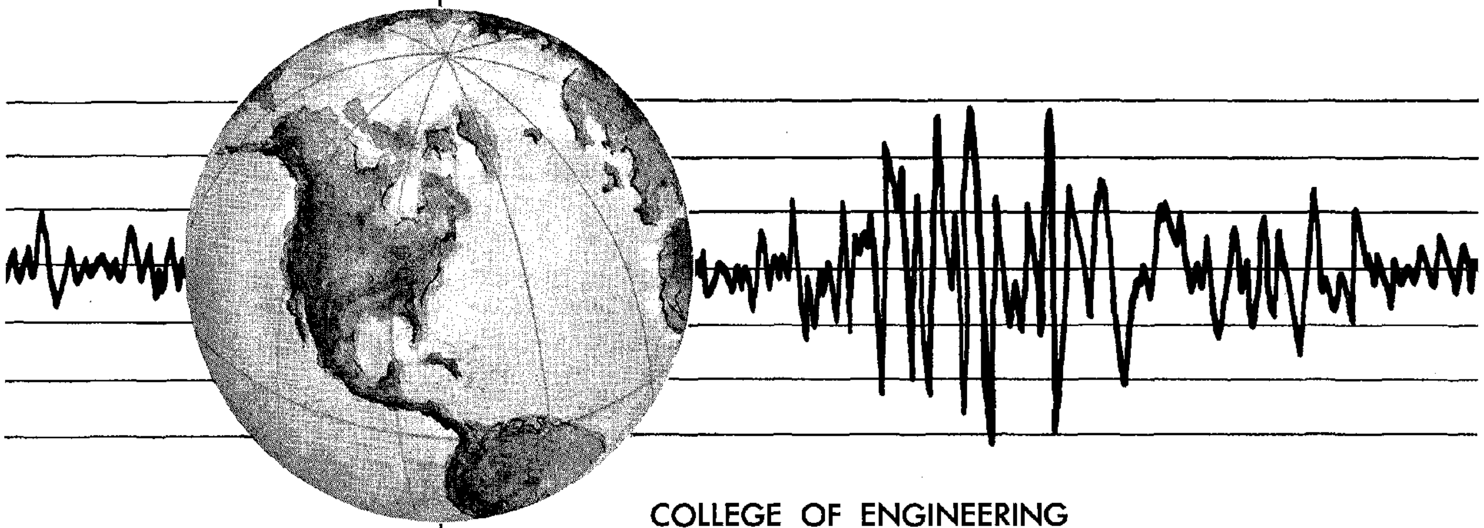
EARTHQUAKE ENGINEERING RESEARCH CENTER

CSTRUCT
AN INTERACTIVE COMPUTER ENVIRONMENT FOR THE DESIGN AND ANALYSIS OF EARTHQUAKE RESISTANT STEEL STRUCTURES

by

M. A. AUSTIN
S. A. MAHIN
K. S. PISTER

Report to the National Science Foundation



COLLEGE OF ENGINEERING

UNIVERSITY OF CALIFORNIA · Berkeley, California

50272-101

REPORT DOCUMENTATION PAGE	**1. REPORT NO.** NSF/ENG-87038	**2.**	**3. Recipient's Accession No.** PB88 174339/AS
4. Title and Subtitle CSTRUCT An Interactive Computer Environment for the Design and Analysis of Earthquake Resistant Steel Structures			**5. Report Date** September 1987
			6.
7. Author(s) M.A. Austin, S.A. Mahin and K.S. Pister			**8. Performing Organization Rept. No.** UCB/EERC - 87/13
9. Performing Organization Name and Address Earthquake Engineering Research Center, University of California, 1301 South 46th Street, Richmond, California 94804			**10. Project/Task/Work Unit No.**
			11. Contract(C) or Grant(G) No. (C) (G) PFR-7908261 ENG-7810992
12. Sponsoring Organization Name and Address National Science Foundation, 1800 G. Street, N.W., Washington, D.C. 20550			**13. Type of Report & Period Covered**
			14.

15. Supplementary Notes

16. Abstract (Limit: 200 words)

The purpose of the report is to describe progress on the development of a new computer-aided design package called CSTRUCT for the analysis and design of earthquake resistant steel frames. Its implementation is for 32-bit engineering workstations with bit-mapped high resolution graphics capable of operating under Ultrix, and the X-Window System. Software development issues such as the user interface, management of data, and the choice of a computer language are covered.

As the problem description evolves, the frame and optimization attributes are identified, and employed by the user in the ensuing design process. The use of the command language and the graphical tools to identify attributes of the optimization problem, critical frame actions, frame displacements under different loadings, and to determine the overall adequacy of a design is shown. Software implementation is summarized and suggested directions for continued software development are listed.

17. Document Analysis a. Descriptors

computer-aided design
earthquake resistant steel frames
frame displacement
optimization

b. Identifiers/Open-Ended Terms

CSTRUCT
Ultrix
X-Window system

c. COSATI Field/Group

18. Availability Statement: Release unlimited	**19. Security Class (This Report)** Unclassified	**21. No. of Pages** 103
	20. Security Class (This Page) Unclassified	**22. Price** PC# 19.95 / MF 6.95

(See ANSI-Z39.18) *See Instructions on Reverse* OPTIONAL FORM 272 (4-77) (Formerly NTIS-35) Department of Commerce

CSTRUCT: AN INTERACTIVE COMPUTER ENVIRONMENT FOR THE DESIGN AND ANALYSIS OF EARTHQUAKE RESISTANT STEEL STRUCTURES

by

M.A. Austin

S.A. Mahin

and

K.S. Pister

A report on research sponsored by the
National Science Foundation
Grants PFR-7908261 and ENG-7810992

Report No. UCB/EERC-87/13
Earthquake Engineering Research Center
College of Engineering
University of California
Berkeley, California
September 1987

ACKNOWLEDGEMENTS

Much of this work would not have been possible without the stimulating discussion and helpful suggestions from Joe Landers, Chris Thewalt and Stephen Wuu. The study reported herein was supported by the National Science Foundation under Grants PFR-7908261 and ENG-7810992. The support is gratefully acknowledged. Views are those of the writers and not necessarily those of NSF.

Table of Contents

CHAPTER 1 - INTRODUCTION

1.1 Introduction

Engineering design is the process of deriving the description of an artifact from a set of specifications. Typically, these specifications will include the artifact's functional requirements and a description of its environment, as well as guidelines on aesthetics, and the ease with which it must be built and maintained. Even though preliminary design specifications may be limited to a few key parameters, this is often enough information for experienced designers to generate design alternatives, and decide on solution strategies for eliminating alternatives. The process of verifying the design artifact's behavior in its intended environment is called simulation. For the results of simulations to be of use, not only must a designer have an understanding of what constitutes desirable behavior within the environment, but also the insight to modify intelligently the initial properties of the artifact to improve behavior. A good design is not always easy to derive because some of the preliminary designs may perform inadequately, no matter how they are changed.

The motivation for designers to use computers in such a design process becomes significant when it is believed that the design process as a whole will be speeded up. Computers are more likely to be employed when they can be used to address practical problems, produce results that are useful, and are easy to use. Improved efficiency may simply be due to the designer being relieved from repetitive design tasks, but it could also be due to a graphical display of the artifact's behavior, or perhaps a mechanism for presenting information in a manner that enables the designer to easily identify and compare the attributes of design alternatives. The difficulties in providing computational assistance in design are largely due to the diversity of areas that need to be integrated before the variety of cognitive skills required for design are properly represented. During the synthesis (brainstorming) stages of design, for example, computational assistance in the form of heuristics or rules of thumb seems appropriate because it mimics a designers often adhoc approach to generating design alternatives. Approximate analysis procedures seem appropriate for preliminary design unless the project represents a significant diversion from experience. In the latter stages of design, however, complex simulations are often required before the designer has enough information to confidently tradeoff attributes among design goals, while simultaneously ensuring that the final design specifications are adhered to. Problems of the latter type lend themselves to a step-by-step solution procedure.

Because no single person or group is likely to have the time or personnel to tackle this problem in its entirety, contributions to the development of design procedures and computer software are incremental. Changing attitudes play an important role in the focus of each contribution. For example, our perspective of what constitutes a reasonable amount of computation is rapidly changing as more powerful hardware becomes available. Objections to design methods currently perceived as being computationally intensive will no longer be relevant by the time most of the required developments have been completed. During the early 1970's it was commonly believed that development of computer-aided design tools would allow complete automation of the design process. The notion of a *black-box* approach to design being sufficient is clearly conveyed by the partial quotation "methodology of *automated* or *hands-off* design, where the *algorithm* replaces *insight*[16]." Because these designers failed to perceive the needs or advantages of an interactive computing environment, it is not surprising that their design programs provided little feedback. Some veterans of the design community probably wondered how the designer was expected to accept full responsibility for a design when he or she wasn't even expected to be part of the process!! And apart from the fact that a rigorous implementation of this type has yet to be produced, there is little doubt the issues of professional negligence have dampened some early enthusiasm. Frustrations of a similar nature are more recently reported by Bobrow et al.[14] in their critical review of knowledge engineering and expert systems, and the failure of some developments to live up to their early claims. This does not mean that knowledge-based expert systems will play a lesser role in the future, however. It is just that our expectations of automated design are being modified. Currently, a realistic requirement is that a designer and computer should be complementary as they work together to complete a design.

1.2 Motivation for this Research Program

At Berkeley, there has been a substantial effort to develop algorithms, formulate design methodologies, and design computer software that will help engineers achieve rational designs that are consistent with adopted design philosophies. The structural engineering component of this inter-disciplinary effort has concentrated on the design of dynamically loaded structures, in particular, the design of earthquake-resistant steel structures.

This design problem is complicated by the large uncertainty in predicting the spatial and temporal nature of future seismic events. Further uncertainties are introduced due to the limited ability of analytical models to properly describe the nonlinear response of structures under severe earthquake excitations. Consequently, designers have difficulty in making quantitative decisions regarding the adequacy of a design, and in choosing rationally among different design alternatives. The decision making process is further complicated by the fact that performance criteria are usually multi-tiered and related to notions of acceptable risk.

For example, the Structural Engineers' Association of California[48] recommends a three-tiered seismic design criterion for buildings that must perform satisfactorily during earthquake loadings. In addition to carrying gravity loads, structures should resist minor earthquakes without any damage, and have sufficient strength to assure protection against structural damage from moderate ground shakings. In the event of an unusually severe earthquake, extensive structural damage without collapse is accepted. These criteria have become the *accepted design philosophy* for conventional building structures. In order to make the design process tractable, most current design codes[3,53] approach the design problem indirectly by means of load and resistance factors, simplified "equivalent loads" and simplified analyses. Conventional structures are deemed to satisfy these criteria if they satisfy the basic strength and drift limitations, and if prescribed detailing requirements are followed.

Nonetheless, the relationship of the accepted design method to the accepted design philosophy is tenuous. While design analyses give the implication that the structure will respond elastically to the design loadings, the accepted design criteria rely on extensive inelastic deformations to absorb energy under severe earthquake excitations. The situations where this incompatibility is likely to cause designers difficulty include the design of complex or irregular systems, those employing new materials or design detailing, or situations where considerations of economics or post-earthquake functionality necessitate enhanced performance criteria. Designers with little experience in cases such as these may find it difficult to assess the quality of a final design and identify the changes necessary to improve the performance of a structure. Indeed, problems of this type are becoming more prevalent because structural systems requiring special performance criteria are coming into vogue. The required behavior of base isolated[5] and friction braced frames[7,47] under severe lateral loads, for example, is more stringent than for conventional structures under the *accepted design philosophy*.

Consequently, the specific goals of this research program are to employ advanced numerical analysis procedures, optimization theory, reliability theory, and techniques from computer science to develop design methodologies and computer software that help designers achieve structural designs that are consistent with specified performance criteria.

1.3 Summary of Past Work

The thrust of the work during the early stages of this research was to produce a computer-aided design environment called DELIGHT.STRUCT[12]. This environment requires the seismic design problem to be recast into a series of mathematical statements that capture its objectives and constraints. Linear and nonlinear time history analyses are used to evaluate the design objectives and constraints for each limit state. Early applications of DELIGHT.STRUCT were restricted to the deterministic design of moment-resistant[13] and friction-braced steel structures[7]. It is known, however, that the spatial variation in acceleration waveforms can be significant even for a single event measured over a localized region[15]. When one also considers that peak values of structural response are known to be sensitive to the details of an incoming ground motion[50], it immediately becomes apparent that although these early designs based on a single earthquake input were optimal for the chosen ground motion, a designer had no assurance of their ability to perform satisfactorily for other ground motions.

In an effort to mitigate these deficiences, a design methodology that includes linear and nonlinear time history analyses, and reliability-based ideas within the design process itself has been proposed[6,8]. The scatter in structural response outputs due to earthquake loads is explicitly accounted for by generating a family of ground motion records for each limit state considered, and performing dynamic analyses for each input motion. Interpretation of the statistics of relevant frame response quantities is facilitated by assuming that the significant frame response variations and uncertainties arising in design may be assigned to two groups: (a) a range for frame response values over which its ability to carry loads or deform without failure becomes significantly less certain and (b) a range of frame response levels whose probability of being exceeded is believed to bound the most desirable level of reliability. The adequacy of a design is ascertained by simply comparing the expected actions at the prescribed reliability level to the ability of the structure to carry these actions without failure. To facilitate this comparison a single design entity called *designer dissatisfaction* that quantifies the results is defined.

$$D(const_i) = \left[\begin{array}{l} 0 \textit{ for } [\, LOW\_resp - GOOD \,] < 0 : \textit{otherwise} \\ \left[\dfrac{LOW\_resp - GOOD}{(LOW\_resp - HIGH\_resp) + (BAD - GOOD)} \right] \end{array} \right. \tag{1}$$

In Eq. 1, $const_i$ is the i^{th} constraint. The GOOD and BAD frame response levels bound the frame's ability to perform. The GOOD value corresponds to a dependable level of system performance, while the BAD level of structural response represents a threshold at which undesirable performance is almost assured if exceeded. LOW_resp and HIGH_resp are structural response levels corresponding to the HIGH and LOW fractiles of probability of being exceeded. The former represents the lowest level of reliability the designer is prepared to accept when the limit state is actived, while the latter represents a level of safety which the designer considers to border on *conservative safety* against failure for the limit state.

Dissatisfaction is not a boolean variable simply describing whether or not a constraint is satisfied, but a function whose value depends on the magnitude of a constraint violation. It is zero for a conservative design, becomes slightly nonzero (ie, within the interval [0,1]) as the design becomes more economical, and increases above 1 as the design becomes increasingly unconservative. It increases monotonically with increasing frame response scatter. Ideally, a maximum dissatisfaction among all of the performance attributes of about 0.5 should be aimed at since this is roughly half way between a design that is too conservative, and one that is believed to be unreliable.

The DELIGHT.STRUCT software has been modified significantly to accommodate the new methodology. Moreover, because an effective structure balances the attributes of cost, performance and reliability in some optimal way, algorithms[45] permitting multiple design criteria to be simultaneously assessed have been added to DELIGHT.STRUCT. Results of a prototype implementation to the design of moment-resistant frames is reported in reference [9]. A comprehensive study of the behavior and optimization of concentrically braced frame systems is about to conclude. The results of these studies are contained in a doctoral thesis[36] to be completed in August 1987, and two papers[34,35].

1.4 Objectives and Scope of this Report

Now that considerable success has been achieved with the development of the design methodology and its prototype implementation, the immediate development is being directed towards improving the software implementation, and the quality of interaction and communication between the user and the computing environment. Our experience with DELIGHT.STRUCT indicates that the design process cannot be regarded as a black-box operation without designer involvement[6]. Success is most likely when the problem and solution are graphically described, and when the designer is provided with the tools to both help understand how different parts of the structure would likely behave, and interact with the design process and modify engineering specifications. Furthermore, it should enable the engineer to explicitly set special performance criteria for unique structures and assist the designer in making decisions based on likely structural performance versus design criteria and expected costs.

A current limitation of the DELIGHT.STRUCT software is that it is strongly tied to the statistical limit states design methodology. Not all designers care to use these formal optimization procedures. Moreover, DELIGHT.STRUCT only provides assistance at the final stages of design after a structural system has been selected and the initial member sizes have been determined. No mechanisms currently exist for considering alternative structural systems, applying approximate analyses, and for each mechanism concisely summarizing the key response values that control the design. If such an environment were developed, then better informed designers would be in position to make judgements with confidence that might otherwise be unobtainable.

The purpose of the report is to describe progress on the development of a new computer-aided design package called CSTRUCT for the analysis and design of earthquake resistant steel frames. Its implementation is for 32-bit engineering workstations with bit-mapped high resolution graphics capable of operating under Ultrix, and the X-Window System[23]. Software development issues such as the user interface, management of data, and the choice of a computer language(s) is covered in Chapter 2. Chapter 3 introduces the basic features of CSTRUCT, and the various styles of user interaction provided. As the problem description evolves in Chapters 4 and 5, the frame and optimization attributes are identified, and employed by the user in the ensuing design process. Chapter 6 shows how the command language and the graphical tools are used to identify attributes of the optimization problem, critical frame actions, frame displacements under different loadings, and to determine the overall adequacy of a design. A summary of software implementation, and suggested directions for continued software development are listed in Chapter 7.

CHAPTER 2 - ISSUES OF SOFTWARE IMPLEMENTATION

2.1 Introduction

No factors are currently having a greater impact on our view of computer-aided design than rapid advances in computer hardware, the introduction of advanced color display workstations to the marketplace, and a significant drop in the price/performance ratio of engineering workstations. The emerging features of this technology are substantial mass storage of data, high processing speed for computationally intensive applications, high resolution bitmapped graphics, interactive computation, and networking that allows data to be shared among multiple users. When these workstations were first introduced a significant gap existed between the capabilities of the workstation hardware and the demands of most application programs. Since then, however, the development of window systems such as X has mitigated this problem by providing designers with the tools to build application programs around the capabilities of the workstation hardware. X not only manages the visual appearance of the screen layout and the mechanical interaction of the user with the mouse and keyboard input devices, but supports the concurrent display of multiple applications on the screen, allowing a user to freely switch between applications. It also provides for both low-level and high-level interfaces such as line drawing and menus.

Now it is possible to develop design programs that provide more than a mere increase in computational speed, with the possible elimination errors due to the automation of some tedious and repetitive tasks. Improved insight into the behavior of the structure with use of high resolution graphics is achievable. Further, with the use of multiple windows mechanisms and advanced procedures for managing the design data, the most pertinent items of design information can be collected, and displayed to the user in a manner that most conveniently summarizes results or trends in behavior. The implementation of such a system, however, is unlikely to be successful unless the users needs are clearly defined, and the requirements of a data management system to support these needs are examined beforehand. Of course, an appropriate computer language(s) must also be selected. Each of these aspects is now discussed in detail.

2.2 Language Considerations

The frame and optimization *pre* and *post* processors of the current version of DELIGHT.STRUCT are written in RATTLE[44], with the design evaluations being performed in Fortran using the nonlinear structural analysis package called ANSR[41]. RATTLE is an interpretive language[1] that is based on the Ratfor[32] language, and employs C-like control structures. It was developed with the intention of being *easy to use* and providing support for incremental program development. Although this goal was achieved, programs written in RATTLE run very slowly. More recently the need to continue developing programs in RATTLE has been reduced due to the release of symbolic debuggers such as *dbx*[38], and the wide use/acceptance of UNIX utilities such as *make*[21] for controlling the processes of program compilation.

The pre/postprocessors of this environment are written in the C programming language. It is selected because its execution speed in significantly greater than the RATTLE language, and its data structures allow for considerable flexibility in the manipulation and organization of design data. C may be combined with other languages such as Fortran[2]. This means that the ANSR simulation package[41] (or comparable FORTRAN codes) can be retained to calculate the structural response, with the storage of response values handled by the C data structures. Moreover, C easily interfaces with the low-level graphics facilities provided in X, as well as higher level general purpose toolkits such as Sx[46] for building and managing the layout and selection of subwindows, pull-down menus, titlebars, scrollbars and notifiers. Another important reason for selecting C, in this application at least, is its ability to communicate with parser generators such as YACC[29]. Parser generators provide mechanisms for associating meaning to components of a grammar in such a way that interpretation and evaluation can take place. The details of how YACC works are not trivial, and no attempt is made to explain them here. Instead, the interested developer is referred to Chapter 8 of Kernighan[33] and Aho et al.[1] for discussions on YACC and how to use it.

[1] According to Bill Nye[44] RATTLE is an acronym for "RATfor Terminal Language Environment."
[2] Under Unix 4.2 and Ultrix, anyway.

2.3 Data Management

Even though data management during the 1960's was little more than a box of punched cards, this was sufficient for the solution of many problems. As designers attempted to apply these techniques to the solution of problems with much larger volumes of data, however, the inadequacies of this approach became less tolerable. Difficulties of a similar nature in the business community provided a strong motivation for the development of formal mechanisms for manipulating and storing data. As a result, a significant volume of literature now exists on data models and database managers[20,27,30,50,52], together with guidelines on the best selection of data model (relational, hierarchal, or network) for different problem structures, the rates at which data must be updated, and variations of anticipated queries[20]. Yet, database implementations based on a purely relational, network or hierarchal models have had only limited success in engineering. One reason for this is that data models customized to maximize performance in engineering applications can be very different from anything developed in the business arena. Sreekaanta et al.[52], for example, report on the development a data model to handle the storage and manipulation of sparse matrices in the solution of finite element and structural optimization problems. It appears that the needs or uses of such an organization of data with the business domain had not even been perceived. Moreover, because the derivation of an artifact is an evolutionary process of design versions and sequences of incrementally changed designs, the form and layout of data in engineering design tends to be more dynamic than for business applications. A computational environment that explicitly handles data for various versions of design is currently under development[30], and is expected to offer significant improvements in the way in which engineering data is managed.

For these reasons, an existing database manager is not used in the version of the program. Rather, an effort is made to organize the information into data models appropriate to the problem currently at hand, with the expectation that revisions to the management of data will be required in future versions of CSTRUCT. Appendix 1 summarizes some of the data models used in this development; namely, data structures used for the frame definition, frame geometry attributes, frame response storage, and frame performance assessment.

2.4 User Interface

One factor that played a minor role in determining the success of application programs prior to the advent of interactive computing environments was the quality of communication between a user and machine. For many years the important issues were program features, portability, graphics, and whether or not a computer was really needed to get the job done in the first place. But with interactive computing environments becoming indispensable for the solution of much more difficult design problems, there is growing evidence to suggest that *ease of use* is at least as important as *functionality*[24,25,28] in determining the likely success of an application program. Now it is simply unrealistic to develop new styles of design without also ensuring the users are familiar with its features, and know how to use them[3]. Developers should acknowledge the wide variety of user backgrounds by customizing the design environment to the changing skills and requirements of users as they acquire experience with the new design style. A novice must be convinced that the details of the design method will be easy to learn, while experienced designers are simultaneously made to feel that their capabilities are not limited by the tools in the interface. Currently, the most practical way of dealing with these variations is to develop tools that allow tasks to be completed via a number of interaction styles. Foley and Van Dam[22] point out that user-computer dialogues such as menus and prompts are popular with novice users because the computer takes the initiative in guiding the user through the intricacies of specifying input. Conversely, *dialogues* in which the user has control and invokes one of many alternatives, typically without being presented with an explicit set of alternatives are called user-initiated dialogues and are suitable for experienced users. Menus are appropriate for experienced users if the menus can be presented very quickly, and without affecting the visual continuity and *sense of place* within the interaction display. The main limitations of menu interaction are that the scope of operations is often restricted, and the definition of problem attributes can be very slow.

Most of the recent developments in structural analysis and design have menu based user interfaces (see references[39,40] for the work from Cornell) or simply rely on text editors to prepare input data files, and graphics for displaying analysis results (see reference [55] for SAP80). Although some of these

[3] Nowhere are the pitfalls of this observation more evident than with DELIGHT, and the amount of user knowledge required to select solution procedures from its library of algorithms.

developments claim to support design, in actuality they do little more than analysis with some checking of design constraints. Generally speaking there is no notion of satisfying behavior for a number of limit states, adjusting the modeling assumptions for consistency with expected behavior, catering for special performance requirements, or incorporating methods for automatically updating the design.

The development goal of CSTRUCT is to provide designers with a computational environment for the analysis and design of earthquake resistant steel frames. Its user interface must provide mechanisms for: (a) describing the design problem, (b) the querying of information about the design, (c) setting special performance criteria for unique structures, and (d) interpreting design performance and behavior, and comparing the attributes of alternative designs. Since these tasks are somewhat diverse, an interface that supports multiple styles of user interaction with graphics is considered essential. Accordingly, the CSTRUCT user interface employs both the mouse and keyboard as input devices, and tools from the Sx[46] window library. Sx is a collection of routines that supplement the X window system with pull-down menus, scroll bars and notifiers. Sx also provides a framework for building and managing window-based application programs. This framework consists of an *event dispatcher* for dealing with the interactions between a window and the application program, a *packer* for managing the layout of subwindows, and a *selection manager* for providing a consistent interface with information that the user has selected.

Although the X window manager supports overlapping windows, a fixed window layout is assumed for CSTRUCT, with the Sx packer maintaining a consistent layout when the windows are resized or moved. The two main components of this layout (see Figure 3.1) are a graphics window and a scrolling text window. Associated with each window is a title bar and a menu bar. Because the design process consists of many stages the manner in which some commands affect the design problem will inevitably depend on the context of the task at hand, while other *utility* commands have the same post-command action, irrespective of the design task being considered. The adopted style of development for this project is to put frequently used *utility* commands in the menu bar. Thus a user can execute these commands by selecting the appropriate menu item of give a keyboard command with an equivalent post-command action. A second use of the menu bar is to control a series of popup windows or forms containing information about the design. For instance, this could be a subset of the AISC tables, a summary of scaled earthquakes in the ground motion library, or information on some aspect of the frame performance. If information needs to be selected from a table for the design then this should be passed back to the program. The final user interface issue considered here is the type of communication between the user and machine employed for the design problem description. Unlike control system problems (for an example, see DELIGHT.MIMO[56]), the behavior and design of structural engineering applications strongly depends on the geometry of the design artifact. One possibility is to specify the structural geometry using a mouse. However, this approach tends to be slow, and some graphics windows may not have sufficient magnification to ensure adequate precision. Consequently, a keyboard style of interaction is used for this job. This style of interaction is generally faster than using a mouse, and precision may be specified up to the word capacity of the machine.

CHAPTER 3 - CSTRUCT for BEGINNERS

3.1 Introduction

This chapter introduces the user to the CSTRUCT environment, and the various styles of user interaction it supports. As outlined in Chapters 1 and 2, CSTRUCT's functional purpose is to provide engineers with analysis tools for the design of earthquake-resistant steel frames. Not only must CSTRUCT permit engineers to evaluate designs in manner that is consistent with expected behavior, but it should also be flexible enough so that designers can employ approximate analysis procedures with factored loads if desired. The selection of an appropriate modeling procedure is a decision left to the designer. By default, frame performance is evaluated in a manner that is consistent with the *accepted design philosophy*. The analyses used for each limit state are:

(a) **Limit state 1** is for static analysis. This limit state is appropriate for frames loaded with gravity loads plus statically applied point loads.

(b) **Limit state 2** is for linear time history analysis. Behavior of the frame loaded with gravity loads plus moderately scaled ground motions would be appropriate for this limit state.

(c) **Limit state 3** is for nonlinear time history analysis. Modeling assumptions of this type may be required to capture the inelastic response of frames loaded with gravity loads plus ground motions scaled to severe intensity.

If a designer wishes to model ground accelerations with a pseudo-static lateral load, and introduce load and resistance factors, then a limit state 1 modeling assumption would be appropriate. Otherwise, limit states 2 or 3 modeling assumptions with unfactored dead and live loads would be used.

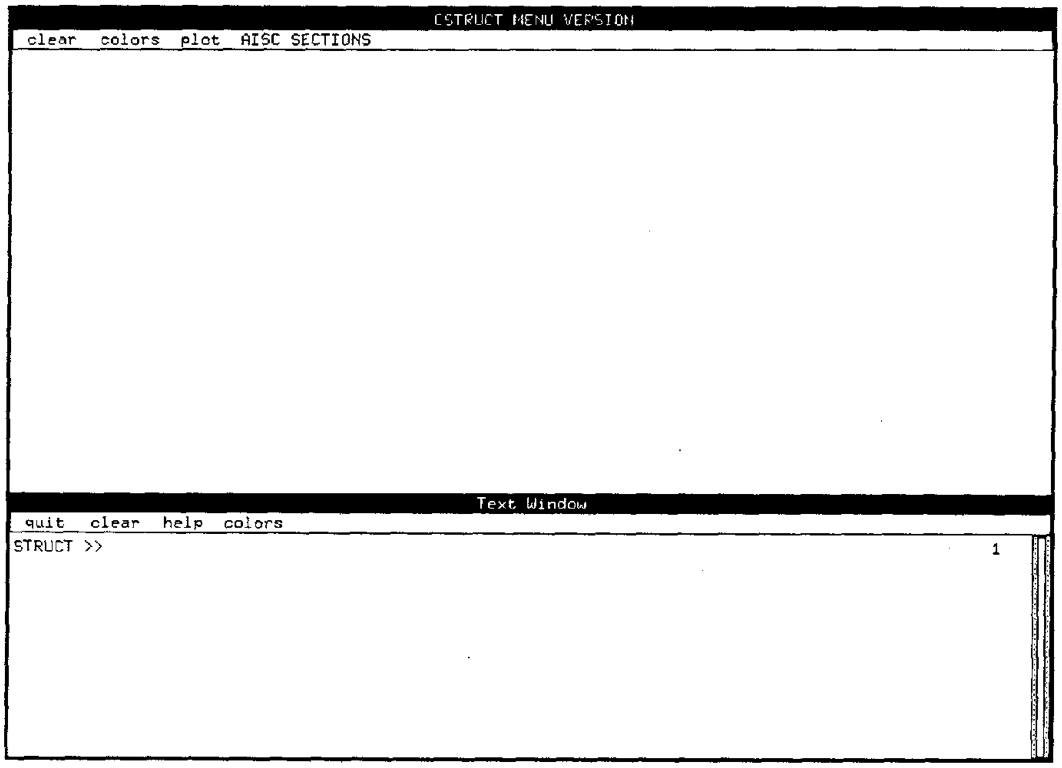
A working knowledge of UNIX is assumed; in all the scripts that follow, % is the unix prompt and commands that should be typed from the screen are shown in **boldface**. Relevant features of the the X-Window system are explained, however.

3.2 Starting CSTRUCT

To start CSTRUCT simply give the command

```
% CSTRUCT
```

from anywhere the within the unix shell. If the user is not in the CSTRUCT working directory (../CSTRUCT/work.d), then the user is automatically moved to the working directory before initialization of the program parameters begins. The most important tasks completed at this stage are: (a) setting default frame simulation parameters, (b) building a table of material properties and AISC[2] section sizes, and (c) reading and storing families of ground motions. When this is complete the default screen layout for CSTRUCT is mapped to the screen, as shown in Figure 3.1.



```
STRUCT >>                                                    1
```

FIG. 3.1 : Window Layout for CSTRUCT

The two main components of this arrangement are a graphics window for the presentation of results, and a text window (with a vertical scrollbar) for keyboard input and the echoing of output. Finally, the STRUCT

>> prompt appears indicating that CSTRUCT is ready to accept commands. An important feature of CSTRUCT is its support for multiple and mixed styles of user interaction. To select a menu item simply move the cursor the the appropriate menu item and depress one of the mouse buttons. If a pulldown menu is selected then a further set of options will be displayed. Otherwise a menu option is selected directly. Each window has been programmed to respond to window events appropriate to the context of its function or purpose. For instance, the text window responds to keyboard events when the cursor is inside the text window. In particular, the key control-H deletes or erases a character, control-U the whole line, and control-W the last word. Hitting the return key indicates that the input line is complete and that it should now be interpreted (parsed). The contents of the text window may be scrolled by moving the mouse to the scrollbar buttoning for the required level of scrolling. When the return key is hit, then the window contents are redrawn in their original state.

3.3 Evaluation of Arithmetic Expressions

Perhaps the simplest use of CSTRUCT is as a calculator. The command syntax to print the results of arithmetic expressions is

```
STRUCT >> <verb> <expr>
```

where the <> parentheses indicate that the command is essential. The command sequence

```
STRUCT >> print 3/4
STRUCT_print >>                0.75
STRUCT_print >> (3 + 4)^2.3
STRUCT_print >>                87.85
STRUCT >>
```

demonstrates its use. The token **print** is matched by the list of available verbs listed above, and <expr> is the numerical result of an arithmetic expression, evaluated according to the hierarchy of operators shown in Table 3.1.

A further point to note is that the command **print** is pushed onto the command stack after it is received, and correctly matched with available keywords in the grammar. The program is now in a **print** state. The results of further numerical expressions may be printed without retyping the command **print**. To exit the print state, type either a null return command as shown above, or move directly to a new program state beginning with

Precedence	Name	Description
1	-	Unaryminus
2	^	Exponentiation
3	*	Multiplication
3	/	Division
4	+	Addition
4	-	Subtraction

Table[3.1] : Operators for Arithmetic Expressions

the command sequence <verb> <noun> ..... examples of this capability are shown later in the report.

3.4 Operations on Lists of Numbers

Using the <expr> format described above, lists of numbers (referred to as <numlist>) may now be constructed from one of the rules

```
<numlist> : <expr>
          | <expr> to <expr>
          | <expr> to <expr> by <expr>
          | <expr> at <expr>
          | <expr> at <expr> from <expr>
```

where the symbol | separates alternative rules in the grammar. The words **to**, **by**, **and**, and **from** are called token names and are actually typed at the keyboard. The simplest applications of this rule occur for a list built from a single <numlist> entity. For example, the commands

```
STRUCT >> print 2 to 6
STRUCT_print >>      2     3     4     5     6
STRUCT_print >> 2 to 3*4/2 by (4/2)
STRUCT_print >>      2     4     6
```

demonstrate use of the first two rules; they allow the user to break a desired numerical range into intervals. The third and fourth rules are used for specifying coordinate offsets where the default offset is 0, unless otherwise specified. Numerical lists may be joined with the **and** and **except** operators, enabling the union and difference of numerical lists to be evaluated. Both operators are identified as the expression is parsed from left to right; when an **and** operator is encountered a flag is set indicating that numerical lists following should be added to the numerical list already built. Conversely, the **except** operator indicates that items in the following argument list should be removed from the numerical list already built, if they exist. For the purposes of completeness, the extended rules currently in the grammar are shown

```
<numlist> : <expr>
          | <expr> to <expr>
          | <expr> to <expr> by <expr>
          | <expr> at <expr>
          | <expr> at <expr> from <expr>
          | <numlist> , <expr>
          | <numlist> and <numlist>
          | <numlist> except <numlist>
```

The comma (,) introduced into the grammar serves the purpose of separating terms in the grammar. So, for example, the following argument lists are admissible:

```
STRUCT >>
STRUCT >> print 2 to 6 and 3.5
STRUCT_print >>          2        3       3.5       4        5        6
STRUCT_print >> 2 to 5 and 3.5, 1.2
STRUCT_print >>        1.2        2        3      3.5        4        5
STRUCT_print >> 2 to 12 except 3 to 5 by 2 and 46, 42
STRUCT_print >>          2        4        6        7        8        9
STRUCT_print >>         10       11       12       42       46
STRUCT_print >>
STRUCT >>
```

3.5 An Overview of the Command Language

A comprehensive and interactive command interpreter based on SDMS[37], the SQL[17] database query format, and the UNIX tool YACC[29] is used to control the design and optimization processes. The results of several researchers may be used as a guideline in setting up the command interpreter. Simplicity is desirable since previous work has shown that for two systems that are functionally equivalent, the one with the simpler syntax produced fewer errors, and was more quickly learned[49]. Foley and Van Dam[22] note, however, that a considerable variety of grammars is possible even for simple commands. A syntax of the form

```
<verb> <noun> [adjective] [option] [scope]
```

is assumed where the [] parentheses signify an optional feature of the command language. The <verb> and <noun> tokens symbolically represent *an action* that should be applied to *an object*. An [adjective] has the the purpose of indicating *how* the operation is to be executed, while an [option] provides additional information on the intended range of the post-command action. For example, the command

```
STRUCT >> print dconst
```

indicates that design constraints should be printed after the command line is completely read. It is simply extended to:

```
STRUCT >> print dconst all
```

if **all** design constraints at limit states 1 to 3 are to be printed. Unfortunately, this level of precision is often inadequate for design purposes. What the designer really needs is the ability to specify exactly the load conditions and regions within the frame that the post-command action is to be applied. The SQL[17] approach to solving a query of this type is to express it as

```
SELECT #name FROM  #location
              WHERE #conditions
```

where items having **#name** within **#location** will be selected only if **#conditions** are satisfied. A [scope] option is appended to the grammar, allowing the user to further qualify the range of the command. Restricted scope may be in terms of (a) a portion of the frame geometry, (b) a subset of the limit state loadings, (c) implied context associated with numerical lists given in the command line, or (d) applied ground motion records. To cover the range of these divisions the [scope] part of the syntax is subdivided further into [listgroup], [region] and [conditions] satisfying the rules:

```
[scope] : [listgrp] [region]
        | [region]  [conditions]
```

The first rule is used during the frame definition, and is discussed further in the following section. Substituting the second rule of [scope] into the grammar gives

```
<verb> <noun> [adjective] [option] [region] [conditions]
```

A region is specified by typing the literal character @ followed by a list of commands restricting the range of the post-command action. The expression

```
STRUCT >> print dconst all @ limst 1
```

has the effect of printing all the design constraints at limit state 1, while the command

```
STRUCT >> print dconst @ elmt 2 to 4 limst 1
```

restricts the range of design constraints of interest to elements 2,3 and 4 for limit state 1 alone. It is also worth noting that the command

```
STRUCT >> print dconst @ limst 1 elmt 2 to 4
```

is also accepted by the grammar, and results is the same post-command action. There are many instances where a designer may only be interested in identifying those design constraints that are close to controlling the design. As a first cut, the designer could search for all design constraints having non-zero dissatisfaction. The [conditions] part of the grammar permits restriction of the post-command action to those entities satisfying an inequality(ies); it is specified by typing the character | followed by a list of inequality expressions that must be satisfied. For example

```
STRUCT >> print dconst all @ limst 1 | dissat != 0
```

prints only those design constraints for the gravity loads alone limit state having non-zero dissatisfaction. The operators ==, >=, <=, and != may be used in the evaluation of conditional statements. Multiple conditional statements may be appended to the command by using the **and** and **or** operators, as previously described. A list of design constraints having dissatisfactions within the interval [0,0.5] at limit state 1 can be obtained with

```
STRUCT >> print dconst all @ limst 1 | dissat > 0 and dissat < 0.5
```

3.6 Combined Graphics-Command Language Interaction

Facilities also exist for combined graphics-command language interaction. The principal mechanism is a rubber banding procedure that defines a rectangular region in the graphics viewport, builds lists of elements and nodes located within the region, and then carrys out a post-command action for these list as they apply to the current program state. Depressing the mouse button for the first time within the graphics window defines one coordinates of one vertex of a rectangle. The instantaneous coordinates of the repositioned mouse defines the coordinates of the diagonal vertex. The final region is designated by moving the mouse until the temporary box covers the required region, and then depressing the mouse button for a second time. Lists of frame elements and nodes within the region are then built and the post-command action carried out using these lists. For example, a user could type the command

```
STRUCT >> print dconst @ limst 1 and 3
```

if he or she wanted to print design constraints for limit states 1 and 3, but didn't want to specify in the command line the exact geometric locations the command is to apply. Now the rubber-banding procedure may be successively applied to different parts of the frame geometry, and all design constraints located within the banded region will be printed to the screen.

3.7 Utility Commands

Utility commands are available at all program states, will not change the context of the current program state, and have the form

```
STRUCT >> <utility> [STRING]
```

where STRING is an alphanumeric string. Examples of their use are given throughout Chapter 4.

3.8 The CSTRUCT Helper Facility

CSTRUCT has a very simple helper facility to assist users with the syntax of commands, to provide a detailed selection of example commands, and to show the values of global CSTRUCT variables. The syntax for the helper is

```
STRUCT >> help [option]
```

where the available [options] are **syntax**, **example**, **all**, and **variable**. Examples of their use are given in Chapter 4. Alternatively, advice on the syntax of commands may be activated by selecting the appropriate menu item.

3.9 Setting Graphics Windows

By default, the dimensions of the graphics window coordinates are taken to equal the number of pixels inside the window when the program is started up, or the window is resized. To see what window coordinates are currently being used, simply type

```
STRUCT >> print window
   INFO >> ... window coordinates ...
   INFO >> ... MINX_WINDOW =     0.0  : MINY_WINDOW =     0.0
   INFO >> ... MAXX_WINDOW =  573.0  : MAXY_WINDOW =   573.0
STRUCT_print_window >>
```

Similarly, the viewport coordinates may be examined with the command **print viewport**. The window coordinates may now be interactively adjusted with a command of the form

```
STRUCT >> MAXX_WINDOW = 4000
STRUCT >> MAXY_WINDOW = 3000
```

This feature is of most use when the desired window coordinates exceed the default window coordinates at the initial stages of the problem definition. Alternatively, a user can move to the **set window** program state (or **set viewport** program state) and use the rubber banding procedure described in Section 3.6.

3.10 Text Labels

Graphs may be labeled with character strings by using the utility command **text**. For instance, the the label **this is a test**

```
STRUCT >> text
What is the string : this is a test
STRUCT >>
```

is specified in response to the prompt. The user is then prompted for the appropriate graphics region before the label is drawn. The utility command **clear** cleans graphics screen. This may be typed at the keyboard, or buttoned in the graphics menu bar.

3.11 Accommodating Errors

Users will inevitably attempt to carry out actions that are not recognized by the computer system. Ideally, computer systems should be tolerant to such mistakes, gracefully recovering the user from such mishaps, before providing help and feedback so that the user can easily get the task completed on the next attempt.

Perhaps the most frequent user error will be typing mistakes in long command lines. As already mentioned YACC parses the command line from left to right, looking ahead as far as necessary to uniquely match the expression with the specified rules. After the rule is identified, the parser takes action on the most recently read tokens, before returning to process the remainder of the command line. The strategy for interpreting command lines in CSTRUCT is to accept all tokens that satisfy the command syntax. If the token moves CSTRUCT into a new command state, then this is done before the remainder of the command line is parsed. For example, in the command script

```
STRUCT >>
STRUCT >> add node z 0 to 500 by 100 y 0 to 400 by 80
 ERROR_add_node >>  ... "z" : command not found
STRUCT_add_node >>
```

CSTRUCT is moved into the **add node** state before the unrecognized **z** token is encountered. The remainder of the command line is ignored, and a error message is given to highlight the unidentified token to the user. The user might now use the **help example** command to obtain examples of correct commands.

3.12 Redirection of Output

A series of utility commands is available for the redirection and/or echoing of command line input, and program output to a file. The command sequence

```
STRUCT >> echo stuff
  INFO >> Created file "stuff"
STRUCT >>
STRUCT >> # this line is a comment statement
STRUCT >>
STRUCT >> echoend
  INFO >> Echo output is is file "stuff"
STRUCT >>
```

first creates a file called "stuff". All subsequent output and command line input is echoed to the screen and to the file "stuff." The process is terminated with the command **echoend**. Table [3.3] summarizes the commands available for input/output redirection.

Command	Effect
echo	echo both input and output to the stated file
echo_output	echo output only to the stated file
echo_onto	append echo output and input to the stated file
echoend	end redirection of output

Table[3.3] : Redirection of Input/Output

3.13 Batch Mode Operation

It is possible to run a complete process in batch mode by creating a file containing an equivalent sequence of interactive commands. For example, the inputfile could contain

```
echo output_file_name
#
# Design evaluation of test problem AMY : 10th Feb. 1987
#
load AMY
write ansr @ limst 1 to 3
run ansr @ limst 1 to 3
#
# [a] Design Constraints with non-zero dissatisfaction
#
print dconst all @ limst 1 to 3 | dissat != 0.0
#
#i [b] Complete summary of design constraint dissatisfactions
#
print dconst all @ limst 1 to 3
#
# [c] Complete summary of frame actions
#
print action all @ limst 1 to 3
echoend
```

The UNIX shell is redirected to take the file instructions (rather than the screen) with the command

```
% CSTRUCT < inputfile &
```

The ampersand & tells the terminal to put the job into the background and immediately take more commands from the screen. Variations on this command are

```
% ( sleep 2000 ; CSTRUCT < inputfile ) &
and
% CSTRUCT < inputfile > /dev/null &
```

In the former example, the semicolon acts as a command terminator, and parentheses group the entire command. The background process starts, but immediately sleeps for 2000 seconds before activating CSTRUCT in batchmode. In the second example, unnecessary screen output is avoided by redirecting it to device null with the command **/dev/null**. The user can now logout and go home.

3.14 Leaving CSTRUCT

The utility commands **exit** and **quit** are used to leave CSTRUCT. The former writes the contents of the current problem to data files (see Section 4.16) before leaving CSTRUCT, whereas **quit** leaves CSTRUCT directly.

CHAPTER 4 - FRAME PREPROCESSOR

4.1 Introduction

The frame preprocessor is a computational tool for the description of 2-dimensional steel frames that must perform satisfactorily for loading conditions of the *accepted design philosophy*. Specifications are given for: (a) the frame geometry, (b) the dead and live gravity loads for each limit state, (c) earthquake loads for each limit state, (d) initial section sizes and material properties, (e) boundary conditions, and (f) the master-slave degrees of freedom for modeling each limit state. The user must begin by defining the frame geometry. Beyond this point, however, tasks (b)-(e) may be completed in any order.

Units of force and length are *kips*, and *inches*, respectively. A (x,y,r) coordinate system is assumed, where the x and y describe the horizontal and vertical coordinates respectively, and r an anticlockwise rotation about an axis pointing into the x-y plane. A 3 bay 5 story frame is now described as an example.

4.2 Description of the Frame Geometry

The frame geometry description can be divided into a three step procedure. First, a grid of nodal points is defined in the (x,y) coordinate space. Frame elements are then attached to the nodal grid. Finally, the frame description is the cleaned; all nodes not attached to any elements are removed from the list of nodes, and attribute lists of frame elements and nodes belonging to each of the frame story levels, floors, bays and column lines are built. Each of these stages is now outlined in detail.

The command syntax for the frame definition is

```
<verb> <noun> [adjective] [listgrp] [region]
```

where the [listgrp] option of the grammar assumes a form

```
[listgrp] : id1 <numlist> id2 <numlist> ...
```

The tokens **id1** and **id2** and so on are identifiers used to provide context to the following numerical lists. A grid of nodal coordinates may be specified with commands of the form

```
STRUCT >> add node [adjective] [listgrp]
```

For example, the command sequence

```
STRUCT >>
STRUCT >> add node x 0 to 100 by 100 and 250,350 y 5 at 80
   INFO_add_node >>  No of nodes generated =   24
   INFO_add_node >>  Total no of nodes     =   24
STRUCT_add_node >>
```

moves CSTRUCT into the **add node** state before generating 24 nodal coordinates. The literal characters **x** and **y** are designated as identifier tokens for numerical lists describing the nodal coordinate positions along the x and y axes, and the post command output indicates the number of nodes added with the most recent command, as well as the total number of nodes defined.

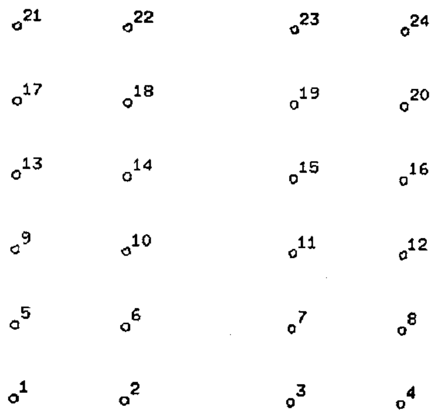


FIG. 4.1 : Grid of Nodal Coordinates

The nodal coordinates may be checked with the command

```
STRUCT >>
STRUCT >> print coord @ node 1 to 21 by 4
   INFO >> Node no          X              Y
   INFO >> ===========================
   INFO >>          1      0.00         0.00
   INFO >>          5      0.00        80.00
   INFO >>          9      0.00       160.00
   INFO >>         13      0.00       240.00
   INFO >>         17      0.00       320.00
   INFO >>         21      0.00       400.00
STRUCT_print_coord >>
```

Other possibilities may be obtained by typing **help example** from the **print coord** command state.

4.3 Drawing and Labeling the Nodal Grid

The syntax for drawing and labeling the nodes is

```
STRUCT >> draw node [option] [region]
```

and

```
STRUCT >>
STRUCT >> label node [option] [region]
```

respectively. For example, the command sequence

```
STRUCT >>
STRUCT >> draw node
STRUCT_draw_node >> all
STRUCT_draw_node >> label node all
STRUCT_label_node >>
STRUCT_label >>
```

draws and labels the nodal grid shown in Figure 4.1. A point to note is that although default colors are assigned to the nodes, elements, design constraints and so on when CSTRUCT is started up, colors may be set explicitly by first moving to the desired program state (eg; **draw node**) and then selecting the appropriate color menu item. Individual nodes may be labeled with either the **label node @ node** <numlist> command, or by moving into the **label node** program state, and using the rubber-banding procedure (outlined in Section 3.6) to define a list of appropriate nodes. Similarly, labeled nodes may be erased by following the command sequence **erase node** [option] [region].

4.4 Description of Element Layout

Three types of elements are currently supported. By default horizontal elements are assumed to be girders, vertical elements columns and diagonal elements braces. An element grid is now superimposed on the nodal grid using either the command syntax

```
STRUCT >> add elmt [adjective] [region]
```

or

```
STRUCT >> add elmt [listgrp]
```

For example, the command script

```
STRUCT >> add elmt
STRUCT_add_elmt >> series @ node 5 to 8
STRUCT_add_elmt_series >> @ node 9 to 12
STRUCT_add_elmt_series >> @ node 13 to 15
STRUCT_add_elmt_series >> @ node 17 to 19
STRUCT_add_elmt_series >> @ node 21 to 23
STRUCT_add_elmt_series >> @ node 1 to 21 by 4
STRUCT_add_elmt_series >> @ node 2 to 22 by 4
STRUCT_add_elmt_series >> @ node 3 to 23 by 4
STRUCT_add_elmt_series >> @ node 4 to 12 by 4
STRUCT_add_elmt >>
STRUCT_add >>
```

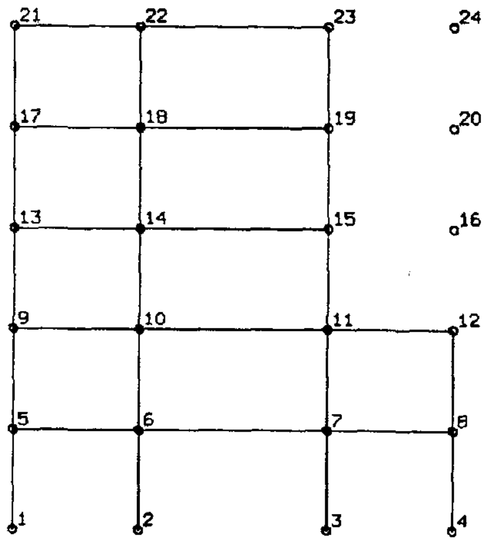


FIG. 4.2 : Frame Element Layout for the Example Problem

generates the element layout shown in Figure 4.2. When the adjective **series** is used, the **i** and **j** ends of the elements are connected to nodes assuming that the **i** end of the element belongs to the node having the lower numerical value and the **j** end of the element to the node having the greater numerical value. Alternatively, the command sequence:

```
STRUCT >> add elmt i 1 to 3 j 2 to 4
STRUCT_add_elmt >> i 5 to 7 j 6 to 8
STRUCT_add_elmt >> i 9 to 12 j 3 to 5
STRUCT_add_elmt >> i 1 to 9 by 4 j 1 to 9 by 4
STRUCT_add_elmt >>
STRUCT_add >>
```

might be used to generate an identical element layout, but with the **i** and **j** ends of the elements interchanged. Moreover, it is conceivable that someone might try to add elements by typing

```
STRUCT >> add elmt x 2 to 4 y 1 to 3
STRUCT_add_elmt >>
```

However, no action is taken in this case because numerical lists associated with the **x** and **y** coordinates have no meaning in the context of adding elements.

4.5 Geometry Attributes and Context

When the user is finished describing the desired layout of nodes and element connectivities, the command

```
STRUCT >> clean [option]
```

may be given to condense the available information. If no option is invoked, then executing the **clean** command only removes all nodes not connected to frame elements. However, if the **all** option is given as in

```
STRUCT >> clean all
   INFO >> ... build (X,Y,Z) axes list
   INFO >> ... build floor level list
   INFO >> ... build column line list
   INFO >> ... build bay contents list
   INFO >> ... build story level list
```

then all elements marked as being deleted are permanently removed from the element list, and all nodes not connected to any element are deleted from the nodal list. Nodal connectivity lists associated with the uniform loads, nodal point loads, master-slave degrees of freedom, boundary conditions, or section sizes are also updated. Finally, the overall frame dimensions as calculated and stored.

Lists of frame elements and nodes attributes belonging to each story level, floor level, column line, and bay are then built. Elements and nodes may now be referred to by their number, or with respect to their geometric location in the frame. The primary advantage of this feature is that the user does not have to adjust his perception of a problem after the elements have been renumbered because of a minor modification of the frame geometry, or perhaps due to a renumbering of the elements to reduce the bandwidth in the stiffness matrix. Instead for printing the nodal coordinates at floor 1 with

```
STRUCT >> print coord @ node 5 to 9
```

one could achieve the same effect with

```
STRUCT >> print coord @ floor 1
```

As a final note, the grammar also allows one to further restrict the range of frame geometry of interest by combining orthogonal frame attributes. The command

```
STRUCT >> print coord @ node 5 to 6
```

has the same effect as

```
STRUCT >> print coord @ floor 1 bay 1
```

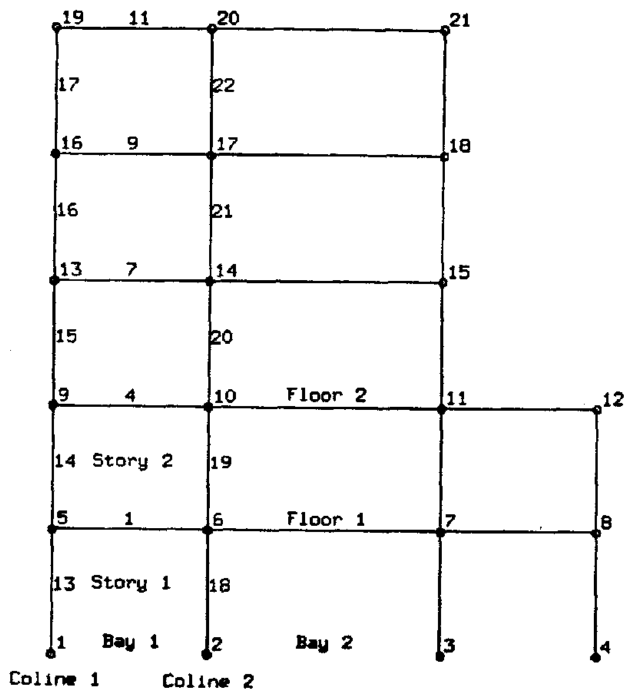


FIG. 4.3 : Cleaned Frame Geometry with Attributes

Of course, if it is decided that frame elements should be added or deleted at a later date, then these element lists must be rebuilt.

4.6 Plotting the Frame

If the command **clean all** has already been given, the graphics window coordinates may be set so that the complete frame will just fit inside the specified graphics viewport by giving the command

```
STRUCT >> set window auto
```

Otherwise the window coordinates may need to be set explicitly, as already demonstrated in Chapter 3. From this point on nodes and elements are drawn with commands of the form

```
STRUCT >> draw <noun> [option] [region]
```

Currently, the most general command available is **draw frame**, which has the effect of drawing all the frame nodes and elements. A more specific command such as

```
STRUCT >> draw elmt @ floor 1 and 2
```

might be given if only a portion of the frame geometry is currently of interest.

4.7 Labeling the Frame Elements

The syntax for labeling the frame elements is

```
STRUCT >> label <noun> [option] [region]
```

where **all** is a permissible option. The help facility may be used to obtain a list of other relevant commands.

4.8 Dead and Live Gravity Loads

The syntax for specifying uniform gravity loads is

```
STRUCT >> add uload [listgrp] [region]
```

where the identifiers for the [listgrp] option are **dead** and **live**, and [region] may be described by geometric, limit state (**limst**) and loadcase (**lcase**) descriptors. Currently, snow and wind loadings are not considered. For example, the command sequence

```
STRUCT >> add uload
STRUCT_add_uload >> dead 0.2 @ floor 1 to 5
   INFO_add_uload >>  limst  1 : 2 load cases : 12 uniform loads added
   INFO_add_uload >>  limst  2 : 3 load cases : 12 uniform loads added
   INFO_add_uload >>  limst  3 : 3 load cases : 12 uniform loads added
STRUCT_add_uload >> live 0.04 @ floor 1 to 5 limst 2 and 3
   INFO_add_uload >>  limst  2 : 3 load cases : 12 uniform loads added
   INFO_add_uload >>  limst  3 : 3 load cases : 12 uniform loads added
STRUCT_add_uload >> live 0.04 @ floor 1 to 5 limst 1 lcase 1
   INFO_add_uload >>  limst  2 : 3 load cases : 12 uniform loads added
STRUCT_add_uload >> live 0.10 @ floor 1 to 5 bay 2 limst 1 lcase 2
   INFO_add_uload >>  limst  1 : 1 load cases :  5 uniform loads added
STRUCT_add_uload >>
STRUCT_add >>
```

adds dead loads of 0.2 *kips/in* on all floors for limit states 1 to 3, and uniform live loads on all floors of 0.04 *kips/in* for limit states 2 and 3. Limit state 1 has two load cases, and limit states 2 and 3, three load cases (see Section 6.2 for a description of how to set the number of load cases for each limit state). Two patterns of live load are defined for the gravity loads alone limit state, however, including zero live loads on all floors for load case 3. Similarly, the syntax for specifying point loads is nodal point loads can be added with commands of the type

```
STRUCT >> add pload [listgrp] [region]
```

As an example, lines of horizontally oriented point loads along the coline 1 nodal list could be specified with the command series

```
STRUCT >>
STRUCT >> add pload fx 2.3 @ coline 1 limst 1 lcase 1
  INFO_add_pload >>  limst  1 :  1 load cases :   6 point loads added
STRUCT_add_pload >> fx -2.3 @ coline 1 limst 1 lcase 2
  INFO_add_pload >>  limst  1 :  1 load cases :   6 point loads added
STRUCT_add_pload >>
```

Point loads in the y-direction and rotational moments in the z-direction may be specified the identifiers **fy** and **rz**, respectively. To check that the loads have in fact been added, simply type the command

```
STRUCT >>
STRUCT >> print uload @ floor 1 limst 1 lcase 1 and 2
  INFO >> Limst Elmt Lcase      dead       live
  INFO >>    No   No    No  (kips/in)  (kips/in)
  INFO >> ====================================
  INFO >>     1    1     1       0.20       0.04
  INFO >>     1    1     2       0.20       0.04
  INFO >>     1    2     1       0.20       0.10
  INFO >>     1    2     2       0.20       0.04
  INFO >>     1    3     1       0.20       0.04
  INFO >>     1    3     2       0.20       0.04
STRUCT_print_uload >>
```

Currently, CSTRUCT does not have a **delete uload** command. Unwanted uniform loads and point loads can be removed by simply setting their numerical values to zero. The parameters **SCALEUL** and **SCALEPL** in the command script

```
STRUCT >>
STRUCT >> draw uload
STRUCT_draw_uload >> help variable
  INFO_draw_uload >> SCALEUL =    50.00
STRUCT_draw_uload >> all @ limst 1
STRUCT_draw_uload >>
STRUCT_draw >> pload
STRUCT_draw_pload >> help variable
```

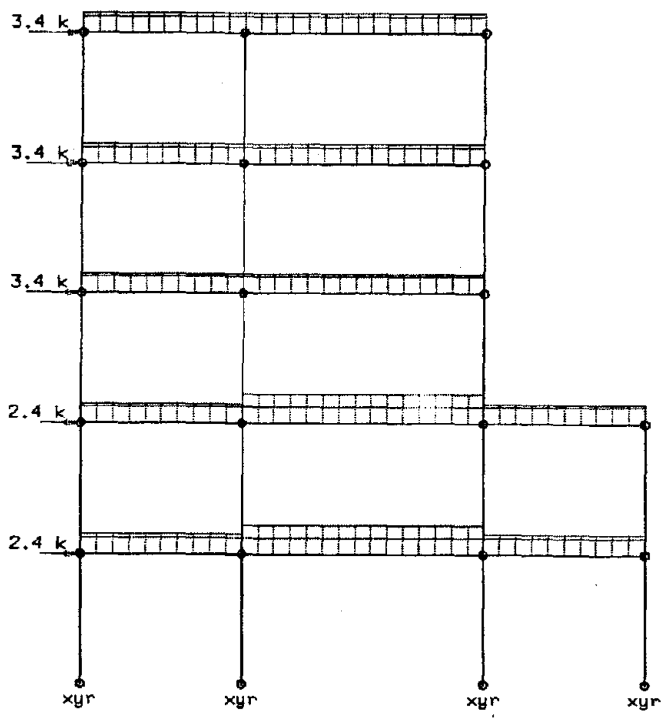


FIG. 4.4 : UniformGravity Loads, Point Loads and Boundary Conditions

```
   INFO_draw_pload >> SCALEPL =    50.00
STRUCT_draw_pload >> SCALEPL = 10
STRUCT_draw_pload >> all @ limst 1
```

are the scale factors for drawing the uniform and point loads in terms of the window coordinates. The adopted convention is to draw the scaled dead loads closest to the element center line, with the live loads on top. Their current values many be printed with the **help variable** command. The magnitude of the point loads may now be printed with the command

```
STRUCT >>
STRUCT_draw_pload >>
STRUCT_draw >> print pload @ coline 1 limst 1 lcase 1 and 2
   INFO >> Limst Node Lcase       X          Y
   INFO >>    No   No    No   (kips)     (kips)
   INFO >> ==================================
   INFO >>     1    1     1     2.30       0.00
   INFO >>     1    1     2    -2.30       0.00
   INFO >>     1    5     1     2.30       0.00
   INFO >>     1    5     2    -2.30       0.00
   INFO >>     1    9     1     2.30       0.00
   INFO >>     1    9     2    -2.30       0.00
   INFO >>     1   13     1     2.30       0.00
   INFO >>     1   13     2    -2.30       0.00
   INFO >>     1   16     1     2.30       0.00
   INFO >>     1   16     2    -2.30       0.00
   INFO >>     1   19     1     2.30       0.00
   INFO >>     1   19     2    -2.30       0.00
```

```
STRUCT_print_pload >>
STRUCT_print >>
STRUCT >>
```

An important point to keep in mind when specifying the uniform loads is that full dead plus live loads are used for assessing frame performance under gravity loads alone. However, for the severe and moderate lateral load limit states, the mass matrix used in the dynamic analyses on dead loads only.

4.9 Material Properties

A limited number of material properties are read from the file **../data.d/material.h** when CSTRUCT is started. Table 4.2 shows the section properties currently available. The symbols E, S, μ, σ_t, and σ_c are abbreviations for for Youngs Modulus, the strain hardening ratio, Poissons ratio, the tensile yield stress, and compressive yield stress.

NAME	E	S	μ	σ_t (kip/in/in)	σ_c (kip/in/in)
STEEL1	29000.0	0.01	0.3	36.0	36.0
STEEL2	29000.0	0.02	0.3	55.0	55.0
STEEL3	29000.0	0.02	0.3	36.0	00.0
STEEL4	29000.0	0.02	0.3	55.0	00.0

Table [4.2] : Material Properties

4.10 Frame Element Sizes

Typically the design process will involve an iterative followed by possible refinement of the element sizes either to make the design feasible, or improve cost. The cycle time of this process may be shortened if the designer is provided with computational tools to: (a) obtain information on section sizes that have properties close to what the designer feels will be appropriate, and (b) easily assign frame element sizes and material properties.

The list of available materials is discussed in Section 4.9. A subset of the AISC[2] section sizes are read from the file **../data.d/section.h** when CSTRUCT is started. Designer's have the choice of using either available AISC section sizes, or parametrically defined sections for wide flange steel sections[54]. The details of using parametric defined sections in an optimization formulation are left till Chapter 5. The goal of this section is to concentrate on the specification of AISC sections. The command sequence

```
STRUCT >> print section
STRUCT_print_section >> help example
  INFO >> print section table
  INFO >> print section
  INFO >> print section | Ixx >= 200
  INFO >> print section | Ixx >= 200 and area < 35
  INFO >> print section @ floor 1 | area < 30
```

shows how the **help example** utility is used to obtain a list of relevant commands and options at the **print section** command state. Now, the [qualifiers] part of the language is used to select a restricted range of sections. A list of section sizes satisfying 400 in^4 < Ixx < 600 in^4 can be obtained by simply typing

```
STRUCT >>
STRUCT >> print section table | Ixx < 600 and Ixx > 400
  INFO >>
  INFO >>    NAME    AREA  DEPTH      Ixx      Iyy   WEIGHT
  INFO >>   (char)   (in)   (in)      (in)     (in)   (lb/ft)
  INFO >> -------------------------------------------------------
  INFO >>   W18x35   10.3  17.70     510.0     15.3     35.0
  INFO >>   W16x45   13.3  16.13     586.0     32.8     45.0
  INFO >>   W16x40   11.8  16.01     518.0     28.9     40.0
  INFO >>   W16x36   10.6  15.86     448.0     24.5     36.0
  INFO >>   W14x53   15.6  13.92     541.0     57.7     53.0
  INFO >>   W14x48   14.1  13.79     485.0     51.4     48.0
  INFO >>   W14x43   12.6  13.66     428.0     45.2     43.0
  INFO >>   W12x58   17.0  12.19     475.0    107.0     58.0
  INFO >> -------------------------------------------------------
STRUCT_print_section_table >>
STRUCT_print_section >>
```

The qualifiers that may be used for printing sections are **Ixx**, **Iyy**, **area** and **depth**. Consequently, an expression of the type **print section table | Ixx < 600 and area > 10** is also allowed.

4.11 Specifying Element Sizes and Material Properties

The syntax for specifying element sizes and material properties is

```
STRUCT >> add section [listgrp] [region]
```

where **material** and **type** are identifiers for the material properties and sections, respectively. The command sequence

```
STRUCT >> add section
STRUCT_add_section >> material STEEL1 @ elmt 1 to 29
STRUCT_add_section >> type W16x36 @  floor 1 to 5
STRUCT_add_section >> type W16x45 @ coline 1 to 4
```

shows how the material STEEL1 (see Table 4.2) is allocated to all the frame elements, and appropriate AISC section types to the columns and girders. Notice that the materials and sections are referred to by name. The sections allocated to each element may now be labelled with the command **label section all**, as shown in Figure 4.5. Of course, the mouse may also be used to define the a rectangular region for labelling sections.

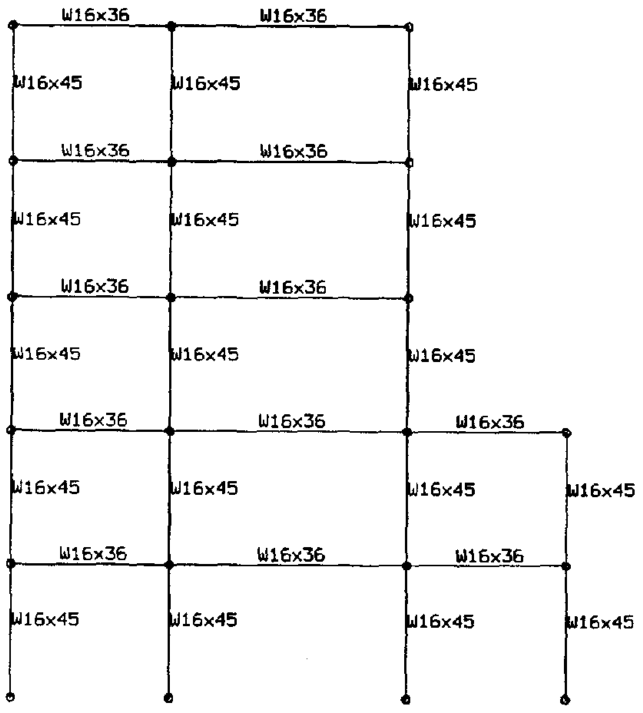


FIG. 4.5 : Labeled Frame Sections

Should a section need to be modified, then detailed description of the current frame sections may be printed by issuing a command of the form

```
STRUCT_print_section_table >>
STRUCT_print_section >> @ elmt 13 and 14
   INFO >>  Elmt  Material  Section  Inertia    Area
   INFO >>   No     Name      Name   (in**4)  (in**2)
   INFO >> =======================================
   INFO >>   13    STEEL1   W16x45   586.00    13.30
   INFO >>   14    STEEL1   W16x45   586.00    13.30
STRUCT_print_section >>
```

Alternatively, current section sizes may be labeled with the **label section** command.

4.12 Boundary Conditions

Frame nodes may be *free* or *fixed* in their translational and rotational degrees of freedom. The syntax for specifying boundary conditions is

```
STRUCT >> add bcond [listgrp] [region]
```

For example, the command

```
STRUCT >> add bcond dx dy rz @ node 1 to 4 limst 1 to 3
```

fixes the [**rz**] rotational and [**dx dy**] translational degrees of freedom at nodes 1 to 4 for limit state loadings 1 to 3. By default, nodal degrees of freedom are assumed to be *free* unless otherwise specified. Moreover, if the range of applicable limit state loadings is not explicitly specified, then a complete range of limit states [1 to 3] and load cases is assumed. The boundary conditions at limit states 1 and 2 might now be checked with the command

```
STRUCT >>
STRUCT >> print bcond @ node 1 to 5 limst 1 and 2
   INFO >> Limst Node           X           Y           R
   INFO >>    No   No      fixity      fixity      fixity
   INFO >> ==============================================
   INFO >>     1    1       FIXED       FIXED       FIXED
   INFO >>     1    2       FIXED       FIXED       FIXED
   INFO >>     1    3       FIXED       FIXED       FIXED
   INFO >>     1    4       FIXED       FIXED       FIXED
   INFO >>     1    5    NOTFIXED    NOTFIXED    NOTFIXED
   INFO >>     2    1       FIXED       FIXED       FIXED
   INFO >>     2    2       FIXED       FIXED       FIXED
   INFO >>     2    3       FIXED       FIXED       FIXED
   INFO >>     2    4       FIXED       FIXED       FIXED
   INFO >>     2    5    NOTFIXED    NOTFIXED    NOTFIXED
STRUCT_print_bcond >>
STRUCT_print >>
```

Finally, the command **label bcond** [option] [region] may be used to obtain a graphical representation of applied boundary conditions. For instance, the full fixity boundary conditions applied to column lines 1 to 4 for the gravity loads alone limit state are shown in Figure 4.4.

4.13 Master-Slave Degrees of Freedom

The syntax to specify master-slave degrees of freedom is

```
STRUCT >> add const [option] [listgrp] [region]
```

Perhaps the best way to demonstrate the range of possible commands is via the **help example** command, namely:

```
STRUCT_add_const >> help example
EXAMPL >> add const auto
EXAMPL >> add const auto @ limst 1 and 3
EXAMPL >> add const master 10 slave 11 to 15
EXAMPL >> add const master 10 slave 11 to 15 @ limst 1
```

In these examples **master** and **slave** are the identifiers for specifying a master node and a list of slave node numbers. The commands **print const** and **label const** may be used to verify that the command has had the desired action.

The **auto** option has the effect of assigning default master-slave degrees of freedom to each limit state. For the gravity loads alone limit state, each node is assumed to have 2 translational and 1 rotational degree of freedom. However, for the moderate and severe lateral load limit state loadings, axial deformations in the columns are ignored, and the translational degrees of freedom are slaved at each floor level. Hence, assuming that the column bases are fully fixed for all limit state loadings, then the example problem is modeled with 51 degrees of freedom for the gravity loads alone limit state, and 22 degrees of freedom for the moderate and severe lateral load limit states.

4.14 Ground Motions

Because statistically-based limit state design methods base performance on the statistics of structural response due to an ensemble of scaled ground motion records, their success to a large extent depends on the designer's ability to scale ground motions to moderate and severe lateral load intensities. Ideally, each record should be scaled to cause equivalent structural damage potential. Characteristics of ground motion that have been suggested as suitable parameters include peak ground acceleration, RMS acceleration, Arias Intensity and Spectral Intensity. Unfortunately none of these is entirely adequate, and extensions[19] to the work discussed in [18,42] are currently underway to mitigate this problem.

The designer should be provided with the tools to graphically compare records before and after scaling, Facilities have been developed to plot the time variation of shaking, acceleration response spectra, and Arias Intensity. They provide a graphical means of comparing the relative magnitudes of loadings for each limit state, and their consequences in terms of frame response quantities controlling the design.

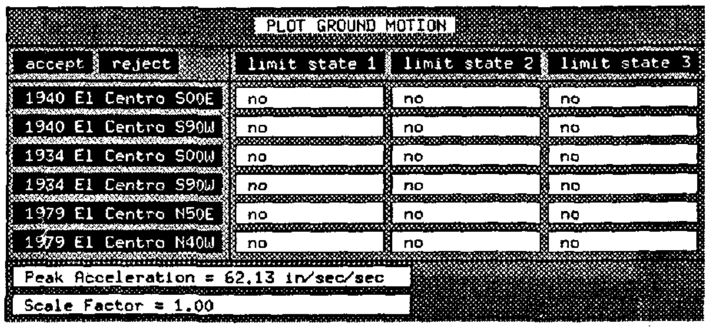
During the startup procedure of CSTRUCT families of ground motions are read from the ascii data files **../gmrecords.d/record1**, **../gmrecords.d/record2** and so on. A typical ground motion header file looks like

```
*** GROUND ACCELERATION RECORD ***

   the maximum allowable number of data points in a record is 999
   units for the accelerations are inches per seconds squared
   1940 EL CENTRO S00E RECORD
   number of data points =  500
   time increment in seconds = 0.02
   peak acceleration for this record in inches per seconds squared = 133.81
   severe quake acceleration in g's = 0.360
   moderate quake acceleration in g's = 0.108
   format for the following accelerations is (8f10.2)
    -31.60    -17.70     -1.48     13.36     30.10     45.02     62.10
     94.15    106.63    118.72    125.17    133.72    110.25     90.69
```

The user should edit the datafile before the problem definition begins to ensure desirable scaling to moderate and severe ground motion intensities. Recommended procedures for scaling ground motions can be found in references [6] and [31].

After CSTRUCT has been started it is inconvenient to leave the program just to examine the properties and compare ground motions by editing the appropriate file. Instead a summary of ground motion properties may be obtained by selecting the **ground motions** menu item, and displaying a ground motions form. Typically, this form will look like:



	limit state 1	limit state 2	limit state 3
1940 El Centro S00E	no	no	no
1940 El Centro S90W	no	no	no
1934 El Centro S00W	no	no	no
1934 El Centro S90W	no	no	no
1979 El Centro N50E	no	no	no
1979 El Centro N40W	no	no	no

Peak Acceleration = 62.13 in/sec/sec

Scale Factor = 1.00

Limit states 1,2, and 3 correspond to the unscaled records, records scaled to moderate lateral load intensity, and severe lateral loads respectively. As the user moves the mouse from window to window, the properties of the current (possibly scaled) ground motion are shown in the summary box at the bottom of the ground motions form. Although peak ground acceleration of the scaled record and the scaling factor are the only ground motion indices shown in the information box, it is relatively straight forward to add further indices of ground motions at a later date. Buttoning on the menu items switches the items from **no** to **yes** status and vise versa. The **accept** and **reject** buttons are used to exit the form: **accept** causes the appropriate items to be plotted and **reject** leaves the form with no action resulting.

4.15 Plotting Ground Motions, Ground Motion Spectra

Ground motions may be plotted may be indicated by filling in the pop-up table shown above, or by giving a keyboard command of the form

```
STRUCT >> draw accn [adjective] [region]
```

For example, the command sequence:

```
STRUCT >> print viewport
   INFO >> ... Viewport Coordinates ...
   INFO >> ... MINX_VIEWPORT =     0.000 : MINY_VIEWPORT =     0.000
   INFO >> ... MAXX_VIEWPORT =     1.000 : MAXY_VIEWPORT =     1.000
STRUCT_print_view >> MAXX_VIEWPORT = 0.9
STRUCT_print_view >> MAXY_VIEWPORT = 0.8
STRUCT_print_view >> MINY_VIEWPORT = 0.2
STRUCT_print_view >> draw accn @ record 1
STRUCT_draw_accn >> text
Type in the text string : EL CENTRO 1940 NS COMPONENT
STRUCT_draw_accn >>
```

might be followed to define a suitable viewport size before producing and labelling the plot of a 10 second segment extracted from the 1940 El Centro ground motion (see Figure 4.6). Similarly, acceleration response spectra of multiple ground motion records scaled to moderate and severe lateral loading may be graphically compared by selecting the appropriate menu item and filling out the pop-up table. Results of the command **draw accn spectra @ record 1 to 4** are shown in Figure 4.7. By default, the unscaled records spectra and Arias Intensities are automatically drawn for limit states not equal to **2** or **3**. As a final note, equivalent plots of Arias Intensity may be obtained by substituting the adjective **spectra** with the command **arias**.

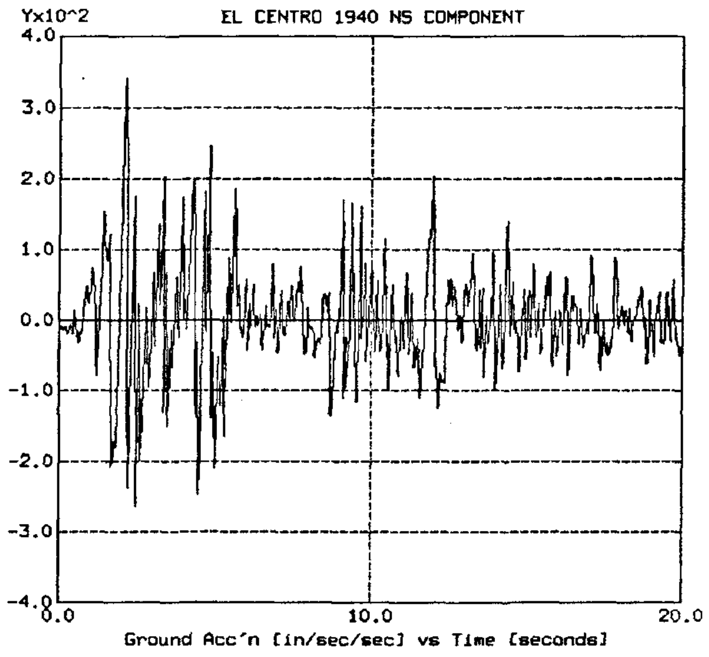


FIG. 4.6 : Ground Motion Record

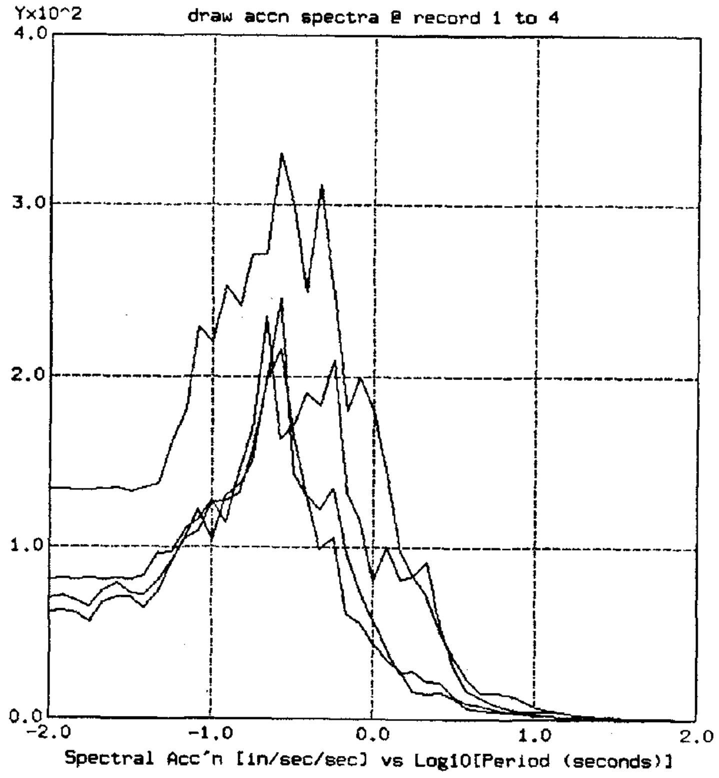


FIG. 4.7 : Acceleration Response Spectra

4.16 Save and Restore Capabilities

A useful feature of CSTRUCT is the ability to assign identities to problems. The syntax for this utility command is

```
STRUCT >> start NAME
```

where NAME is an alphanumeric string of the problem name. For instance, the current problem could be called AMY by simply typing **start AMY**. Should the user omit to specify the expected name, he or she will be prompted for an adequate name as demonstrated in the script

```
STRUCT >> start
Who is the problem ?? : AMY
STRUCT >>
```

If the user wishes to end an interactive session, The current state of the problem AMY may be written to binary datafiles by typing

```
STRUCT >> save AMY
   INFO >> ... writing <AMY.frame>
   INFO >> ... writing <AMY.nodes>
   INFO >> ... writing <AMY.elmts>
   INFO >> ... writing <AMY.nodeloads>
   INFO >> ... writing <AMY.elmtloads>
   INFO >> ... writing <AMY.bconds>
   INFO >> ... writing <AMY.resp>
   INFO >> ... writing <AMY.opt>
   INFO >> ... writing <AMY.dparam>
   INFO >> ... writing <AMY.bconst>
   INFO >> ... writing <AMY.dobjec>
   INFO >> ... writing <AMY.dconst>
```

In fact, a problem may be saved at any point after the frame nodes and elements have been cleaned (see Section 4.5). This allows the construction of bay, story, floor and column line lists to be successfully built. A previously defined problem, say for example BERT, can now be loaded with the command

```
STRUCT >>
STRUCT >> load BERT
   INFO >> ... including file <BERT.frame>
   INFO >> ... including file <BERT.nodes>
   INFO >> ... including file <BERT.elmts>
   INFO >> ... including file <BERT.nodeloads>
   INFO >> ... including file <BERT.elmtloads>
   INFO >> ... including file <BERT.bconds>
   INFO >> ... including file <BERT.resp>
   INFO >> ... including file <BERT.opt>
   INFO >> ... including file <BERT.dparam>
   INFO >> ... including file <BERT.bconst>
  ERROR >> ... can't open file <BERT.bconst>
   INFO >> ... including file <BERT.dparam>
  ERROR >> ... can't open file <BERT.dparam>
```

```
 INFO >> ... including file <BERT.dconst>
ERROR >> ... can't open file <BERT.dconst>
 INFO >> ... build (X,Y,Z) axes list
 INFO >> ... build floor level list
 INFO >> ... build column line list
 INFO >> ... build bay contents list
 INFO >> ... build story level list
```

In the above example the datafiles for the box constraints, design parameters, and design constraints have not yet been defined, and are therefore not read.

CHAPTER 5 - DESCRIPTION OF THE OPTIMIZATION PROBLEM

5.1 Introduction

A key step in this design process is the writing of the design problem in a mimimax optimization format. The basic ingredients in a formulation of this type are: (a) a set of quantifiable objectives, (b) a set of well defined constraints, and (c) a process for obtaining tradeoff information among objectives. Constituents (a) and (b) define the scope of most optimization problems. Component (c) contains a decision rule that enables the best compromise to be made among multiple criteria. A semi-infinite nonlinear programming problem of the form:

$$\min_{x} \; [\; \max_{i} w_i cost_i(x) : i = 1,2..L \;] \tag{2}$$

$$\textit{subject to} \quad g_j(x) \leq 0 : j = 1,2..M$$

$$\textit{and} \quad f_k(x,t) \leq 0 \; \forall \; t \in [T_b, T_e] : k = 1,2..N$$

$$\textit{and} \quad X_{\min} \leq x_i \leq X_{\max} : i = 1,2..P$$

is assumed. In Eq. 2, w_i is the weighting coefficient for the i^{th} goal of L objective functions, $g_j(x)$ the j^{th} entity of M conventional constraints, and $f_k(x,t)$ the k^{th} member of N functional inequality constraints. The parameters T_b and T_e bound the range of the independent parameter t. Box constraints limit the range of permissible values on each of the P design variables.

Design alternatives may be defined by decision variables, or simply as a list of alternatives. Designers should keep in mind that in this formulation the purpose of the design parameters is to describe those aspects of a structure that may be modified in order to get an improved design. The design constraints serve the purpose of discouraging the design parameters from taking values that are impractical, and from moving into a region that has an unacceptably high level of risk of unsatisfactory frame performance. Measures of economy and structural performance at the global level are quantified by the design objectives; these mathematical statements are generally a function of the design variables, and should provide the motivation and direction for moving towards a better design.

5.2 An Overview of the Design/Optimization Process.

Because the design constraints and design objectives serve different functions in this formulation, each has its requirements for evaluation. The step-by-step procedure for constraint evaluation is:

(1) Specify the [GOOD,BAD] and [HIGH,LOW] pairs for each constraint. Each constraint is given either a HARD or SOFT attribute. HARD constraints are ones that must be fully satisfied (perhaps to satisfy a physical law that cannot be violated), and once satisfied must remain satisfied, and not partake in tradeoffs among constraints and objectives. SOFT constraints are those in which a moderate constraint violation is tolerable, and can be traded off against other SOFT constraints and performance attributes.

(2) Simulate the frame response for the appropriate limit state(s).

(3) Identify the appropriate frame response quantities. Calculate the mean and standard deviation of the response quantities and plot a histogram of the results. An important point to note is that a time-history analysis is required for each ground motion input before the functional constraints can be evaluated. The histogram of peak frame response quantities is constructed by taking just the peak value from each of the time-history response(s).

(4) Assume a probability distribution type and calculate its parameters from the data provided.

(5) Calculate the characteristic values on frame response [HIGH_resp , LOW_resp] corresponding to the HIGH and LOW exceedance probabilities specified at step (1).

(6) Substitute into Eq. 1 (see Chapter 1) to get the designer dissatisfaction.

Similarly, the calculation procedure for the design objectives is:

(1) Identify the design objectives relevant to the problem at hand.

(2) Specify GOOD and BAD values for each design objective.

(3) Simulate the frame response for the appropriate limit state(s).

(4) Identify the relevant frame response parameters and calculate the appropriate statistics of frame performance.

(5) Substitute frame performance response quantities into Eq. 3

$$D(response\_value) = \begin{cases} 0 \; for \; [\; actual\_resp \text{ - } GOOD \;] < 0 \; : otherwise \\ \left[\dfrac{response\_value \; - GOOD}{BAD \; - GOOD} \right] \end{cases} \tag{3}$$

where *response_value* is the characteristic response quantity for the design objective. An important distinction between the constraints and objectives is that frame response values corresponding to the HIGH and LOW fractiles of reliability are not required for the design objectives. It is the job of the constraints to ensure that the reliability of a design is adequate. Once the design is feasible, then only GOOD and BAD design objective values are needed to provide a general direction for change to an improved design.

A convenient way of managing the overall design problem is to divide it into components, and view the design process as the solution to a sequence of sub-problems. From the discussion in Chapter 1, it is evident that design evaluations are required at each stage of the design process, with the post-evaluation action depending not only on the calculated design performance, but the current state of the design process itself. The general solution strategy used in the most recent work (ie; see references [6,8,9,10] and [36]) has been to regard the design problem as a 3 phase process. First, priority is given to satisfying all of the HARD design constraints. The design goal in phase 2 of the design procedure is to find a design having the minimax dissatisfaction among all of the SOFT design constraints and design objectives, while simultaneously ensuring that the HARD design constraints remain satisfied. A final third phase of the design process may be entered if all of the constraints are completely satisfied (ie; better than their GOOD values); further improvement in the design objectives is sought without causing a constraint violation.

During the initial stages of simulation the designer's goal is to tune the [GOOD,BAD] and [HIGH,LOW] preference pair settings until the relative design objective and frame constraint performances are correctly represented by a hierarchy of ranked dissatisfactions. For this task to be completed in an expedient manner, a designer should have a good idea of what constitutes adequate reliability and constraint performance before the design process begins. Code recommendations and technical papers, together with experience, can be used to this end. Ascertaining GOOD and BAD values for the design objectives can be more difficult because they tend to be less well defined, often assuming values that are strongly related to a structure's size and configuration. Moreover, because designer perception is is subjective, several conceptual revisions of what constitutes adequate performance may occur before a satisfactory hierarchy of dissatisfactions is obtained. Only then can an algorithm (or trial and error design procedure) be expected to produce a sequence of improved designs which also have lower dissatisfactions.

5.3 Design Parameters

Frame members may be subject to design or fixed. During the preliminary stages of design, allocating frame elements to a design parameter group and designating members as "to be designed" versus "already sized and not to be designed further" is subjective, as is the selection of the best design parameter arrangement or layout. As the design process continues (and perhaps construction starts for fast-track projects), certain frame elements become fixed; only a subset of the frame elements are left to size. This means that the specification of design parameters must be flexible. A designer should be provided with the ability to consider several parameter layouts, as well as the ability to designate regions of the frame with fixed AISC section sizes, while examining the sensitivity of overall frame performance to perturbations in the frame element sizes remaining to be fixed.

The frame elements are each modeled by a single section property parameter. Moment of inertia is the primary section property parameter used for the beam and column elements, and cross-sectional area for truss elements. Element properties of secondary importance such as radius of gyration and element depth are obtained from empirical relations derived by Walker[54] for economy wide flange steel sections. Default values for the empirical section relationships are read from the file **../data.d/assume_elmt** (see Table 5.1) during the CSTRUCT startup procedure. A further point to note is that when empirical relations are employed, the material properties also default to those shown in Table 5.1.

```
*** ASSUMED MATERIAL VALUES AND SECTION RELATIONSHIPS ***
BEAMS AND COLUMN ELEMENTS.

   Youngs modulus for steel = 29000.0
   Yield stress for steel = 36.0
   Strain hardening ratio for steel = 0.05
```

```
For columns:
  moment yield coordinate fraction = 1.0
  axial yield coordinate fraction  = 0.15
  radius of gyration = 0.39 * depth ** 1.04
  for inertia <= 429.0
  depth        = 1.47 * inertia ** 0.368
  otherwise
  depth        = 10.5 * inertia ** 0.0436

For girders:
   steel poisson ratio = 0.3
   radius of gyration =  0.52 * depth   ** 0.92
   depth              =  2.66 * inertia ** 0.287

For braces:
   Youngs  modulus for bracing   = 29000.
   Yield stress for bracing      = 30.
   Brace strain hardening ratio = 0.02
   inertia = 0.169 * area ** 3.0
```

Table 5.1 : Default Wide Flange Section Relations

5.3.1 Specifying the Design Parameters

The design parameters are specified in a two-step process. First, a design parameter name is associated with those frame elements subject to design. Element sizes are then assigned to the design parameters by name. The syntax for completing the first step is

```
STRUCT >> set dparam NAME [region]
```

where NAME is an alphanumeric name for the design parameter selected from the series **x1**, **x2**, **x3** ... onto **x20**, and [region] a portion of the frame geometry. Element sizes can now be assigned to the design parameters with

```
STRUCT >> NAME = <expr>
```

In fact, these steps can be combined into

```
STRUCT >> set dparam NAME = <expr> [region]
```

as demonstrated by the script

```
STRUCT >>
STRUCT >> add dparam x1 @ story 3 to 5 coline 1 to 3
STRUCT_add_dparam >> x2 = 300 @ floor 3 to 5
STRUCT_add_dparam >>
STRUCT_add >> x1 = 400
```

to assign names and values to the design parameter layout shown in Figure 5.1. The syntax for labeling the design parameters is

```
STRUCT >> label dparam [option] [region]
```

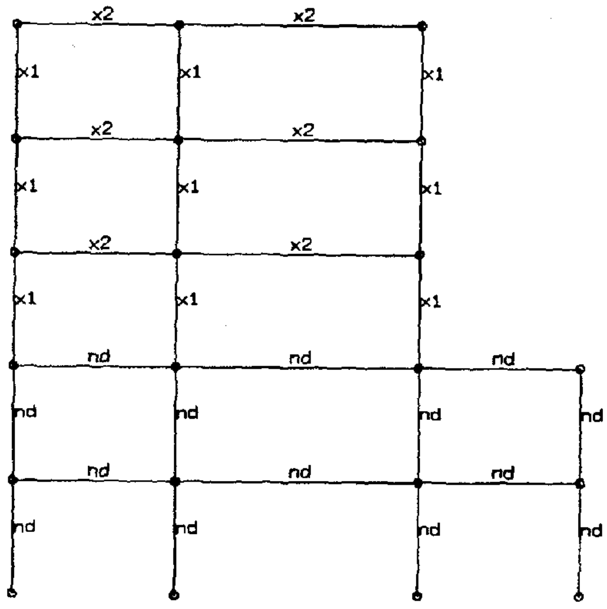


FIG. 5.1 : Design Parameter Layout

Information on the design parameters may be printed in two forms. The current value of a design parameter at a specific frame element can be obtained with the command

```
STRUCT >> print dparam [option] [region]
```

For example, the command

```
STRUCT >> print dparam @ story 5
   INFO >> Elmt No          Name          Value
   INFO >> ---------------------------------
   INFO >>       11           x1        400.000
   INFO >>       12           x1        400.000
   INFO >>       17           x1        400.000
   INFO >>       22           x2        300.000
   INFO >>       27           x2        300.000
STRUCT_print_dparam >>
```

prints the design parameters at story 5, together with their associated section sizes. Frame elements with no design status (ie labeled **nd** in Fig 5.1.) are represented as missing entries when the design parameters are printed. It may also be convenient to know the list of frame elements corresponding to each design parameter. The command

```
STRUCT >> print dvector
  INFO >> ... build design vector list
  INFO >>   X[] :      Name  : Lead  : Followers
  INFO >> --------------------------------
  INFO >> X[ 1] :       x2  :    7 :     8     9    10    11    12
  INFO >> X[ 2] :       x1  :   15 :    16    17    20    21    22    25    26
  INFO >>                               27
STRUCT_print_dvector >>
```

has the effect of building an ordered list of the frame elements associated with each item in design vector, before printing the lists. The first element in each list is called the leading design parameter, while the remaining elements are called followers.

5.4 Design Objectives

Our previous research efforts indicate that: (a) multiple design objectives are required to adequately describe structural performance, and (b) the most appropriate design objectives depend on the type of structural system being designed. Since the specification and description of objectives is still an active research problem, discussion in this section is limited to a summary of the ideas motivating the design objectives used in the most recent work[6,36].

Volume of Structural Elements: Utilizing minimum volume as a design objective reflects a typical design philosophy. Although volume is correlated to material cost, a modest material saving may be of lesser importance than other possible objective functions when considering the structure's lifetime performance and cost. Nonetheless, minimum volume is often used as the starting point for optimization inasmuch as it reflects the minimum initial material cost of the structure.

Story Drifts: Drift control generally ensures structural integrity and the control of non-structural damage.

Energy Based Design: Although it is possible to design a structure to resist severe lateral earthquake loads elastically, economic factors usually dictate that it is more feasible to design a system having the largest energy dissipation capacity that is consistent with tolerable deformations[4]. A structure frame should survive these motions with reasonable predictability, which usually implies that a frame should reach full plastic yielding before the maximum lateral frame displacements are reached. An essential ingredient in this type of formulation is the energy balance equation; it reduces a conglomerate of complex mechanical information distributed in both space and time into a time-dependent scalar equation of the form:

$$W = E_k + D + E_e + E_i \tag{4}$$

where E_k = *kinetic energy*

D = *damped energy at the element level*

E_e = *Elastic energy*

E_i = *Inelastic or dissipated energy*

W = *total input energy (or work done) by externally applied loads*

During an earthquake energy is fed into the base of the structure. It is important to know how the energy is distributed among the terms in the energy balance equation, and how each term is related to the physical characteristics of structural behavior. Input energy is the scalar product of the base shear force (plus all external loads) moving through an incremental displacement at each timestep integrated over time. This quantity is a function of the structure's properties, including its mass, damping, and stiffness. With respect to structural behavior, input energy generally decreases and becomes less sensitive to ground motion frequency content for structures that have a low yield level. Structures with high yield values primarily dissipate energy through element damping, whereas structures with low yield values dissipate energy through inelastic cycling. The contribution of the elastic and kinetic energy terms are usually of secondary importance in the balance equation.

A qualitative design objective is needed that seeks a structure which not only satisfies the constraints, but performs well under severe lateral loads. For ductile structures, safely minimizing input energy and maximizing the percentage of energy dissipated through inelastic internal energy results in good overall performance because the structure attracts smaller quantities of energy, and distributes it to as many elements as possible without violating constraints. In other words, optimizing the dual criteria:

$$\textit{minimize } W \tag{5}$$

$$\text{and} \quad \text{maximize} \left[\frac{E_i}{W} \right]$$

over the duration of the earthquake is required. Specifying design objectives for structures of limited ductility (conventionally braced frames or masonry structures) or those containing passive energy dissipating devices (base-isolated structures and friction-braced frames) is a more difficult problem. For braced frames, limited ductility is due to the inability of the structural system to develop the required displacements for energy dissipation without a localized member failure or a rapid deterioration in element strength. The limit ductility of masonry structures is simply due to the brittle nature of the construction material. Consequently, design objectives that encourage maximum hysteretic energy dissipation are inappropriate, as are design objectives which ensure a completely elastic response even for a maximum credible ground motion input.

While it is generally agreed that structural systems containing passive energy dissipating devices should have performance requirements that are more stringent than the *accepted design philosophy*, the engineering research community is still undecided how much more stringent these requirements should be. Pall and Marsh[47], for example, design the main structural systems of friction-braced frames to remain completely elastic during severe ground motions. Some researchers contend that while improved performance is assured, this philosophy is too conservative. They argue that it is possible to design safe structures that are more economical even if limited inelastic deformations in the main structural elements are permitted. Issues of a similar nature also exist for the design of base-isolated structures, and required behavior of the superstructure during severe lateral loads. The interested reader is directed to Section 4.5.3 of reference [6] for a more complete discussion on design objectives, their purpose and implementation.

5.4.1 Specifying the Design Objectives

CSTRUCT currently supports three types of design objectives. The command syntax for specifying the design objectives is

```
STRUCT >> add dobjec [option]
STRUCT >> add dobjec NAME [region]
```

where the identifier **dv** is a reserved word for the frame volume, **sd** for the story drift, and **e1**, **e2**, **e3** and **e4** are identifiers for the energy groups 1 to 4, respectively. The volume and story drift design objectives serve the purposes outlined in Section 5.4. In the latter case, the designer designates energy group number attributes for selected frame elements. The basic idea in completing this procedure is to distinguish the elements of the frame that are capable dissipating large quantities of energy from those that are less capable of dissipating excessive quantities of inelastic energy without adverse consequences to the overall integrity of a structure. Two measures of performance are used for each energy group; the first quantifies the average effectiveness of an energy group to dissipate energy, while the second measures the variation of energy dissipation among the elements in the group. To illustrate these features, the command sequence

```
STRUCT >>
STRUCT >> add dobjec
STRUCT_add_dobjec >> dv @ elmt 1 to 29
STRUCT_add_dobjec >> sd @ coline 1 floor 1 to 5
STRUCT_add_dobjec >> e1 @ floor 1 and 2
STRUCT_add_dobjec >> e2 @ floor 3 to 5
STRUCT_add_dobjec >> e3 @ coline 1 to 4
STRUCT >>
```

declares 8 design objectives for frame performance assessment. Included are all of the frame elements in the volume design objective, and the column line 1 nodes in the story drift objective calculation. Finally, 3 energy groups are declared for assessing the designs ability to dissipate hysteretic energy in a desirable manner.

Default values for the design objectives may be setup by including the **auto** option in the command. All of the frame elements are included in the design volume calculation, all of the storys in the story drift objectives, and 3 energy groups are declared; one for the girders, and a second for the columns, and a third for the braces. The interested reader is referred to Austin[6,9] for recommended parameter settings for the design objectives. A graphical summary of the specified design objectives may be obtained with a command of the type

```
STRUCT >> label dobjec [option] [region]
```

Figure 5.2 shows a schematic of the design objectives declared when the **all** option is given. The crosses draw on the nodes along column line 1 indicate which stories are included in the story drift design objective calculation.

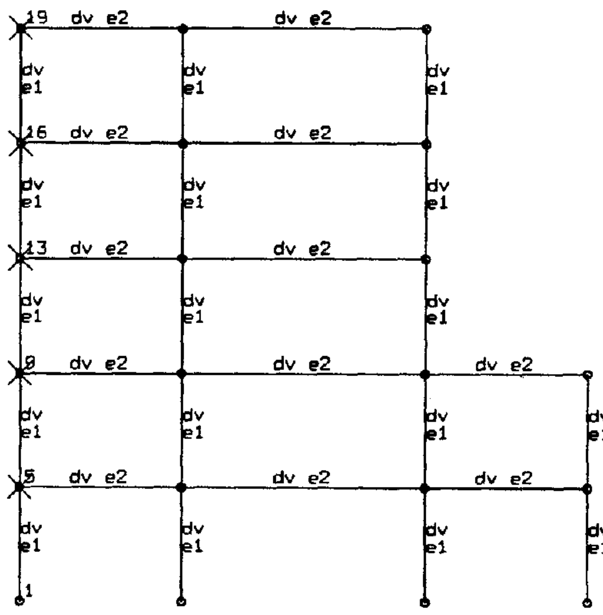


FIG. 5.2 : Frame Design Objectives

The command sequence

```
STRUCT >> print dobjec [option]
  INFO >> Design Objectives
  INFO >> Objective Name : dissat : mean value : good value : bad value
  INFO >> ---------------------------------------------------------------
  INFO >>  frame volume    0.5102    15102.33    10000.00    20000.00
  INFO >>  input energy   -0.9943       56.54        0.00      500.00
  INFO >>  mean energy 1   0.0172        0.00        0.00        0.20
  INFO >>   var energy 1   0.0000        0.00        0.00      500.00
  INFO >>  mean energy 2   0.0621        0.01        0.00        0.20
  INFO >>   var energy 2   0.0000        0.00        0.00      500.00
```

demonstrates how the dissatisfactions may be printed. The design may wish to adjust the GOOD and BAD parameter values so that the hierarchy of design objective dissatisfactions corresponds to the designers feelings about the current designs performance.

5.5 Design Constraints

The design constraints are divided into two major groups; box constraints and limit state design constraints. Only a summary of the design constraints along with the recommended parameter settings is given here, since detailed discussions are already documented in references [6-11].

5.5.1 Box Constraints

Box constraints ensure that only practicable section sizes are considered for the design. Default values for beam, column, and brace elements are read from the file **../data.d/assume_box** at the startup procedure.

```
*** DEFAULT BOX CONSTRAINT VALUES ***

   For columns:
     Constraint  State = ACTIVATED
     Distribution Type = NORMAL
     Constraint   Type = SOFT
     HIGH Reliability level = 0.2
     LOW  Reliability level = 0.1
     [  BAD ,   GOOD ] < inertia < [   GOOD ,    BAD ]
     [ 50.0 , 100.0 ] < inertia < [ 3000.0 , 3500.0 ]
```

Table 5.2 : Default Box Constraints Assumptions File

Table 5.2 shows the header file together with a general template for the design constraint information. The user indicates whether the constraint is to be ACTIVATED (vs NOTACTIVATED), the type of statistical distribution assumed for the response (currently, NORMAL, LOGNORMAL and TYPE1 for the Extreme Type 1 distribution are allowed), as well as if the constraint type is HARD or SOFT. HIGH and LOW fractiles of reliability may also be specified, but these are currently ignored in the dissatisfaction calculation because the frame element sizes are assumed to be deterministic. Finally, pairs of GOOD and BAD parameter values are specified at the upper and lower ends of the box constraint.

5.5.1.1 Specifying the Box Constraints

Because the number of design constraints for even a small structure can easily be several hundred, it is somewhat impractical to expect a designer to interactively designate all design constraints that need to be set. Instead, procedures have been written to read the default constraint values for each limit state, and automatically allocate memory for the storage of constraints. Executing the command

```
STRUCT >>
STRUCT >> add bconst auto
   INFO >> ... set default box constraint values
   INFO >> ... allocate box constraint storage
STRUCT_add_bconst >>
STRUCT_add >>
```

```
STRUCT >>
```

for the example problem generates 29 box constraints. Now the box constraints at column line 1 may be viewed by simply typing

```
STRUCT >>
STRUCT >> print bconst @ coline 1
   INFO >> Box Constraints
   INFO >> Elmt : Constr Name : dissat : section size : good value : bad value
   INFO >> -----------------------------------------------------------------
   INFO >>   13       lower box   0.0000          586.00       100.00       50.00
   INFO >>   14       lower box   0.0000          586.00       100.00       50.00
   INFO >>   15       lower box   0.0000          586.00       100.00       50.00
   INFO >>   16       lower box   0.0000          586.00       100.00       50.00
   INFO >>   17       lower box   0.0000          586.00       100.00       50.00
   INFO >>   13       upper box   0.0000          586.00      3000.00     3500.00
   INFO >>   14       upper box   0.0000          586.00      3000.00     3500.00
   INFO >>   15       upper box   0.0000          586.00      3000.00     3500.00
   INFO >>   16       upper box   0.0000          586.00      3000.00     3500.00
   INFO >>   17       upper box   0.0000          586.00      3000.00     3500.00
STRUCT_print_bconst >>
```

5.5.2 Limit State Design Constraints

The limit state design constraints are used to check the adequacy of performance for each of the limit states in the *accepted design philosophy*. Accordingly, constraints are checked for the gravity loads alone limit state, gravity loads plus moderate lateral loads, and finally, gravity loads plus severe lateral earthquake loads.

5.5.2.1 Constraints Under Gravity Loads Alone

The following conventional constraints apply to the beams and columns under gravity loading alone:

$$[\ \textit{column axial force}\ ] < \textit{Colax} \times \textit{Column axial force} \tag{6}$$

$$[\ \textit{column end moment}\ ] < \textit{Colgra} \times \textit{Column yield moment} \tag{7}$$

$$[\ \textit{girder end moment}\ ] < \textit{Girgra} \times \textit{Girder yield moment} \tag{8}$$

$$[\ \textit{girder midspan deflection under live load}\ ] < \textit{Girdef} \times \textit{Girder span} \tag{9}$$

A convention of nomenclature introduced in equations (6) to (9) is now explained. The parameters *Colax*, *Colgra* and so on should be interpreted as a shortened notation for the GOOD and BAD performance pair

PARAMETER	VALUE/TYPE	DESCRIPTION
Good_Colax	.5000	good gravity column axial force factor
Bad_Colax	.6000	bad gravity column axial force factor
Good_Colgra	.6000	good gravity column yield factor
Bad_Colgra	.8000	bad gravity column yield factor
Good_Girgra	.6000	good gravity girder yield factor
Bad_Girgra	.8000	bad gravity girder yield factor
Good_Girdef	4.170e-3 [1/240]	good girder midspan deflection
Bad_Girdef	4.570e-3 [1/219]	bad girder midspan deflection
Good_Volmax	1.000e+5	good volume maximum
Bad_Volmax	1.200e+5	bad volume maximum

Table[5.3] : Gravity Loads Alone Constraint Parameters

settings. For example, the GOOD value of the column axial force constraint is simply the dependable column axial force factor *Good_Colax* shown in Table[5.3] multiplied by axial force required for Euler buckling of the column.

5.5.2.2 Constraints Under Combined Gravity and Moderate Earthquake Loads

Damage to frame members, windows, partitions and other architectural elements is related to relative frame displacements. These are controlled by enforcing a constraint on story drifts of the form:

$$[\ \textit{story drift}\ ]_{\textit{max over time}} < \textit{Drift} \times \textit{story height} \tag{10}$$

Similarly, floor acceleration is used as a measure of damage to a structure's contents, equipment, and elements attached to the floors. The form of this constraint is:

$$[\ \textit{absolute floor acceleration}\ ]_{\textit{max over time}} < \textit{Accel} \times \textit{acc'n of gravity} \tag{11}$$

A structure should also possess sufficient strength so that under moderate lateral loads structural damage is minimal. Inelastic deformations are discouraged by ensuring that the frame response satisfies the following constraints:

$$[\ \textit{column end moments}\ ]_{\textit{max over time}} < \textit{Colyld} \times \textit{column yield moments} \tag{12}$$

$$[\ \textit{girder end moments}\ ]_{\textit{max over time}} < \textit{Giryld} \times \textit{girder yield moments} \tag{13}$$

PARAMETER	VALUE/TYPE	DESCRIPTION
Good_Drift	4.500e-3	good max moderate story drift
Bad_Drift	8.000e-3	bad max moderate story drift
Good_Accel	0.700	good max moderate floor accel in gs
Bad_Accel	1.400	bad max moderate floor accel in gs
Good_Colyld	.8500	good moderate column yield factor
Bad_Colyld	1.100	bad moderate column yield factor
Good_Giryld	.9000	good girder yield factor
Bad_Giryld	1.100	bad girder yield factor

Table[5.4] : Moderate Lateral Loads Constraint Parameters

5.5.2.3 Constraints Under Combined Gravity and Severe Earthquake Loads

Constraints are divided into two categories reflecting frame behavior at the global level and frame behavior at the element level. Global frame instability is generally attributed to enhanced bending moments due to P-delta effects when large lateral frame displacements act in conjunction with high axial forces. This type of behavior is prohibited herein by placing an upper bound on allowable peak frame displacements. For this development, large displacements at the top of the frame are used as an approximate measure of the possibility of collapse. The parameter *Sway* is defined as the maximum relative horizontal displacement at the top of the frame divided by the frame height and the constraint is described as follows:

$$[\,\textit{frame sway}\,]_{\textit{max over time}} < \textit{Sway} \times \textit{frame height} \tag{14}$$

PARAMETER	VALUE/TYPE	DESCRIPTION
Good_Sway	1.400e-2 [1.4%]	good structure sway max
Bad_Sway	2.000e-2 [2.0%]	bad structure sway max
Good_Colduc	3.000	good column ductility
Bad_Colduc	4.000	bad column ductility
Good_Girduc	4.000	good girder ductility
Bad_Girduc	6.000	bad girder ductility

Table[5.5] : Severe Lateral Loads Constraint Parameters

Structural damage at the material level is closely related to the extent of inelastic deformations. One reversed cycle at a high ductility range may cause damage equivalent to many cycles at a lower ductility range. A constraint on allowable energy dissipation is formulated by assuming that the total hysteretic energy dissipated under an arbitrarily changing deformation history may be equated to the energy dissipated by a monotonic load moving through an equivalent displacement to ultimate failure. The allowable energy dissipation in the latter mechanism is used to form the constraint, namely:

$$E_d < E_y \times f(\mu, S) \tag{15}$$

$$\textit{where} \quad f(\mu, S) = [\mu - 1]\cdot[1 - S]\cdot[2 + S\cdot[\mu - 1]]$$

Table[5.5] summarizes the assumed beam and column ductility factors. The conventional constraints represented by Eq. 15 are:

$$\textit{Column end inelastic energy dissipation} < E_y \times f(\textit{Colduc}, S) \tag{16}$$

$$\textit{Girder end inelastic energy dissipation} < E_y \times f(\textit{Girduc}, S) \tag{17}$$

Currently, no checks are made on a section's lateral and local buckling failure modes. The scatter in frame response quantities such as cumulative energy dissipation, which are somewhat intrinsic to the frame material, are modeled with the Gumbel Extreme Type 1 distribution[26]. Where information on a particular statistical distribution type is not available (ie; peak story drifts, floor accelerations, elastic bending moments, and the maximum frame sway), constraints are statistically described by the normal distribution.

5.4.3 Specifying Limit State Design Constraints

Limit state design constraints are defined at the element and nodal levels. A template for each constraint type is defined for the performance of each element, under each of the limit state loadings. For example

```
*** DEFAULT LIMIT STATE 1 DESIGN CONSTRAINT VALUES ***

FOR THE COLUMNS:
    [a] : Bending Moment Parameters
    Constraint  State = ACTIVATED
    Distribution Type = NORMAL
    Constraint   Type = SOFT
    [ GOOD, BAD ] element resistance = [ 0.6 , 0.8 ]
    [ HIGH, LOW ] reliability level  = [ 0.2 , 0.1 ]

    [b] : Axial Force
    Constraint  State = ACTIVATED
    Distribution Type = NORMAL
    Constraint   Type = SOFT
    [ GOOD, BAD ] element resistance = [ 0.4 , 0.5 ]
[ HIGH, LOW ] reliability level  = [ 0.2 , 0.1 ]

[c] : Axial Displacement
Constraint  State = NOTACTIVATED
Distribution Type = NORMAL
Constraint   Type = SOFT
[ GOOD, BAD ] element resistance = [ 0.4 , 0.5 ]
[ HIGH, LOW ] reliability level  = [ 0.2 , 0.1 ]

[d] : Energy Dissipation
Constraint  State = NOTACTIVATED
Distribution Type = TYPE1
Constraint   Type = SOFT
[ GOOD, BAD ] element resistance = [ 3.0 , 4.0 ]
[ HIGH, LOW ] reliability level  = [ 0.2 , 0.1 ]

[e] : Midspan Deflection
Constraint  State = NOTACTIVATED
Distribution Type = NORMAL
Constraint   Type = SOFT
[ GOOD, BAD ] element resistance = [ 0.003 , 0.004 ]
[ HIGH, LOW ] reliability level  = [ 0.200 , 0.100 ]
```

Table [5.5] : Default Limit States Constraints Assumptions File

Table 5.5 shows the templates for the column element design constraints under gravity loads alone (ie, limit state 1). Each beam-column element may be checked for: (a) bending moment at each end, (b) energy dissipation at each end, (c) axial force, (d) axial displacement, and (e) midspan deflection. Not all constraint types are activated for each element type. In addition, varying parameter settings may apply for different limit state loadings due to the variation in expected frame performance with each of the limit state loadings. Similarly, at the nodal level templates are defined for the story drift and floor acceleration constraints.

Specifying a rigid boundary on acceptable level of frame risk cannot be justified, especially in the absence of experience[6]. Experience requires hindsight; because this style of design is still being prototyped, relatively wide difference between the HIGH and LOW exceedance probabilities have been selected for the pilot studies[6,9,36]. Further, the same HIGH and LOW exceedance probabilities (20% and 10%, respectively) have been assumed for all constraint types within a single limit state. The command

```
STRUCT >>
STRUCT >> add dconst auto
   INFO >> ... set default constraint values for limit state 1
   INFO >> ... allocate design constraint storage for limit state  1
   INFO >> ... set default constraint values for limit state 2
   INFO >> ... allocate design constraint storage for limit state  2
   INFO >> ... set default constraint values for limit state 3
   INFO >> ... allocate design constraint storage for limit state  3
STRUCT_add_dconst >>
STRUCT_add >>
```

STRUCT >>

automatically sets default design constraints values (according to the information specified in the datafiles **../data.d/assume_limstl** and so on) for each of the limit states, and allocates the required memory for the storage of constraints. Memory is also allocated for the storage of terms in the energy balance equation calculated during the gravity loads plus severe lateral loads limit state frame response.

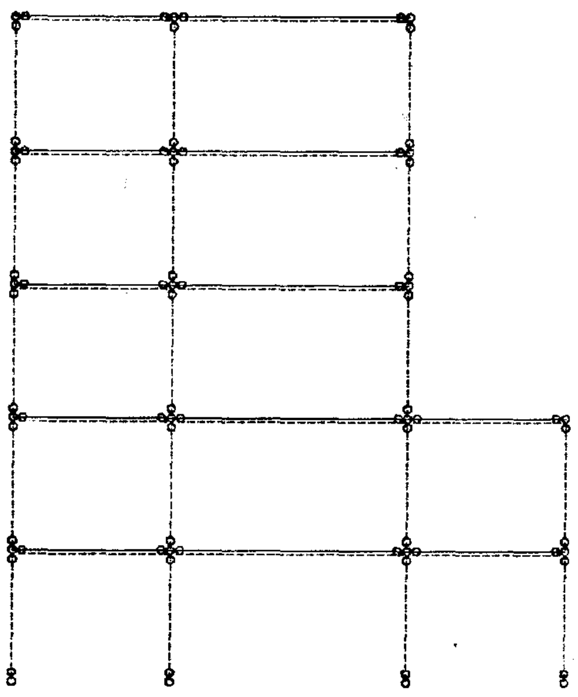


FIG. 5.3 : Design Constraints for Gravity Loads Alone

Figure 5.3 shows a schematic of the default design constraints for the gravity loads alone limit state. The circles at each end of the columns and girders indicates the locations were allowable moments are checked. The dashed line along the column axis is used to indicate which column elements are checked for allowable axial forces, and the horizontally drawn dashed lines below the floors shows which girders are checked for midspan girder deflections. Similarly, Figure 5.4 shows the design constraints checked under gravity loads plus a family of ground motions scaled to moderate intensities. Bending moments are checked at the girder and column ends, but axial forces in the columns are not; this is consistent with the modeling assumptions outlined in Section 4.13. Two types of constraints are checked at the nodal level. The crosses drawn over the nodes along column line 1 indicate that the relative horizontal displacements of these nodes is used for the story drift constraint checks. Similarly, the horizontally drawn dashed lines above the floor levels indicates that floor acceleration constraints are checked. A similar figure may also be drawn for constraints checked under gravity loads plus severe lateral loads.

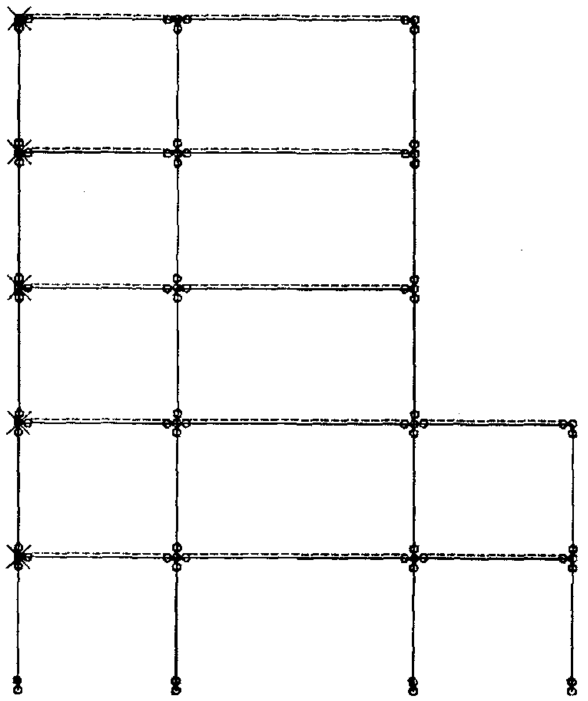


FIG. 5.4 : Design Constraints for Moderate Lateral Loads.

5.4.4 Adjusting the Design Constraint and Objective Parameters

Facilities exist for setting the design constraint and design objective parameters. The syntax for setting the parameters is

```
STRUCT >> NAME = <expr> [region]
```

where NAME is a design constraint or objective name, and [region] the geometric location in the frame, or the load cases for which the design parameter setting is to be applied. A list of design constraint parameter names can be obtained by first moving into the **set dconst** state and then issuing the **help** command, as in

```
STRUCT >> set dconst
STRUCT_set_dconst >> help
   INFO >> List of variables:
   INFO >> good_moment  bad_moment  high_moment   low_moment
   INFO >> good_axial   bad_axial   high_axial    low_axial
   INFO >> good_deflect bad_deflect high_deflect  low_deflect
```

```
    INFO >> good_energy   bad_energy   high_energy    low_energy
    INFO >> good_drift    bad_drift    high_drift     low_drift
    INFO >> good_accel    bad_accel    high_accel     low_accel
```

An example of how the design parameters might now be adjusted is

```
   STRUCT >> set good_moment = 0.60 @ limst 1 coline 1 to 4
   STRUCT_set >> bad_moment  = 0.75 @ limst 1 coline 1 to 4
   STRUCT_set >> good_moment = 0.90 @ limst 2 coline 1 to 4
   STRUCT_set >> bad_moment  = 1.00 @ limst 2 coline 1 to 4
   STRUCT_set >> good_moment = 0.90 @ limst 2 floor 1 to 5
   STRUCT_set >> bad_moment  = 1.10 @ limst 2 floor 1 to 5
```

Now, the modified design parameter settings can be verified with the **oprint dconst param** command, as in

```
STRUCT_print >> dconst
STRUCT_print_dconst >> param
STRUCT_print_dconst_param >> @ elmt 13 and 14 limst 1
   INFO >>
   INFO >> Limit State 1 Constraint Parameter Values
   INFO >> Elmt Node : Constraint Name : type : high :  low : good :  bad
   INFO >> -----------------------------------------------------------------
   INFO >>   13    1        end moment   SOFT   0.20   0.10   0.60   0.80
   INFO >>         5        end moment   SOFT   0.20   0.10   0.60   0.80
   INFO >>   14    5        end moment   SOFT   0.20   0.10   0.60   0.80
   INFO >>         9        end moment   SOFT   0.20   0.10   0.60   0.80
   INFO >>   13                  axial   SOFT   0.20   0.10   0.40   0.50
   INFO >>   14                  axial   SOFT   0.20   0.10   0.40   0.50
```

CHAPTER 6 - SIMULATION and DESIGN EVALUATION

6.1 Introduction

Now that the design problem has been written in a minimax optimization format, attention is focused on simulating the structure for the required limit state loadings, and evaluating its performance. As already outlined in Section 3.1, this implementation supports various styles of design. During the preliminary stages of design most designers will maximize computational efficiency by employing psuedo-static lateral loads, load and resistance factors, and elastic analyses. The design problem might be further simplified by focusing on a subset of limit state loadings. In the latter stages of design, however, linear and nonlinear time-history analyses with multiple ground motion inputs may be required before adequate estimates of structural reliability can be obtained. The purpose of this chapter is to outline step-by-step procedure for completing a simulation. This includes: (a) checking and adjusting the default assumptions for the simulation, (b) writing the data files for the simulator, (c) running the simulations themselves, and (d) examining the frame performance and behavior.

6.2 Frame Simulation Assumptions

The default frame simulation assumptions are contained in the datafile **../include.d/assume_sim**. Its current contents are shown in Table 6.1.

```
*** FRAME SIMULATION ASSUMPTIONS ***

Modelling assumptions.

   Number of load cases for limit state 1 =    2
   Number of load cases for limit state 2 =    3
   Number of load cases for limit state 3 =    3

   Number of time steps for limit state 1 =    1
   Number of time steps for limit state 2 = 1101
   Number of time steps for limit state 3 = 1101

   Storage increment for limit state 1 =       1
   Storage increment for limit state 2 =       3
   Storage increment for limit state 3 =       5
```

Table 6.1 : Default Simulation Assumptions

The first block of statements contains the number of load cases for each limit state. Currently, the maximum number of load cases that can be handled is NO_LOADCASES; see Appendix 1 for more details. The number of integration time steps for each limit state are specified in the second data field. In the most recent work, 1101 time steps have been required to calculate an eleven second time history response. Because the required storage for large design problems can tax the limitations of even the most recently developed workstations, a third data field is added for the increment at which frame response quantities are to be stored. Stored frame response quantities are useful from two perspectives. First, the time dependent variation in responses may be employed to observe behavior, either in the form of plots, or perhaps as structural animation. Since the resolution of many graphics displays will not be fine enough to detect minute variations in behavior, little information will be lost if only every 3rd or 5th point is stored with *real* precision. By contrast, extreme response values are used to assess design performance. Because the partial derivatives of response quantities with respect to perturbations in the design are essential ingredients for optimization (see Chapter 5 of reference [6] for examples), these quantities should be stored with *double* precision. The strategy used in this implementation is to update the extreme values of response at the end of every time step in the linear and nonlinear time history calculations, before deciding if the response should be saved for post-processing purposes.

6.3 Writing the Simulation Files

The command syntax for automatically writing data files for the ANSR simulation package is

```
STRUCT >> write ansr [region]
```

where [region] is matched by a list of limit state loadings. By default, data files are written for all three limit states unless explicitly stated. For example, in the command script

```
STRUCT >> write ansr @ limst 1 and 3
   INFO >> ... allocate frame response memory for limit state  1
   INFO >> ... allocate frame response memory for limit state  3
   INFO >> ... allocate energy balance memory for limit state  3
   INFO >> ... build design vector list
   INFO >> ... checking frame data
   INFO >> ... crunch limit state 1 simulation info
   INFO >> ... writing <limitstate1.1>
   INFO >> ... crunch limit state 3 simulation info
```

```
    INFO >> ... writing <limitstate3.1>
    INFO >> ... writing <limitstate3.2>
    INFO >> ... writing <limitstate3.3>
    INFO >> ... writing <limitstate3.4>
    INFO >> ... writing <limitstate3.5>
  STRUCT_write_ansr >>
```

preparation of ANSR data files is restricted to limit states 1 and 3. The first task is to use the information specified in Table 6.1 to allocate memory for the storage of the frame responses. Checks are then made to ensure that: (a) all frame elements have been allocated frame element sizes and material properties, (b) boundary conditions have been specified for each limit state, and (c) gravity loads have been specified at all floor levels for those limit states requiring a dynamic analyses. Immediately before the data files are written for each limit state, information described at the frame preprocessor stage is crunched into a format compatible with the simulation; the main task is to convert the uniform gravity loads into equivalent nodal point loads and moments for the ANSR analyses.

6.4 Frame Simulation

Simulations serve the purpose of calculating the frame behavior in its intended environment. As with the writing of the simulation datafiles, a performance evaluation is assumed for all limit states unless otherwise noted. The command script

```
STRUCT >> run ansr @ limst 1 and 3
  INFO >> ... set section properties
  INFO >> ... limit state  1 simulation not required
  INFO >> ... limit state  3 simulation
  INFO >> ... eigenvalue analysis
  INFO >> ... load case 1
  INFO >> ... load case 2
  INFO >> ... load case 3
  INFO >> ... check design constraints : limit state 1
  INFO >> ... check design constraints : limit state 3
  INFO >> ... check box constraints
  INFO >> ... check design objectives
```

demonstrates the simulation features by requesting performance evaluations for limit states 1 and 3. Before the simulations actually begin, the frame section properties are calculated for the current design parameter vector. The current section properties are then compared to section properties of the most recent simulation for each limit state considered. If the comparison is very close then the former response is assumed to be identical to the required behavior; a new simulation is not calculated. Otherwise, each of the load cases is simulated. For the linear and nonlinear time history analyses, the damping matrix is modeled as a linear combination of the mass and stiffness matrices. This Rayleigh damping matrix has the form:

$$\begin{bmatrix} C \end{bmatrix} = a_1 \begin{bmatrix} M \end{bmatrix} + a_2 \begin{bmatrix} K \end{bmatrix} \tag{18}$$

where : $\begin{bmatrix} C \end{bmatrix}$ = *damping matrix*

$\begin{bmatrix} M \end{bmatrix}$ = *mass matrix*

$\begin{bmatrix} K \end{bmatrix}$ = *stiffness matrix*

$$a_1 = \left[\frac{2 \lambda w_1 w_2}{w_1 + w_2} \right]$$

$$a_2 = \left[\frac{2 \lambda}{w_1 + w_2} \right]$$

w_1, w_2 = *first and second natural circular frequencies*

λ = *percentage of critical damping in the 1st and 2nd modes*

The percentage of critical damping, λ, is specified in the data file **../include.d/assume_frame**. In order to maintain constant damping values in the first and second modes the coefficients a_1 and a_2 are updated for each current design at the beginning of the calculations for this limit state. First, ANSR is employed to form the mass and stiffness matrices (including geometric stiffness effects) for the present design. A subspace iteration routine extracted from the program FEAP[57] is called to calculate the frame's natural periods of vibration corresponding to the non-zero mass degrees of freedom. After all of the limit state calculations are complete, levels of dissatisfaction are calculated for the design objectives, box constraints, and design constraints.

6.5 Frame Response Actions

The command syntax for examining the frame response actions is

```
STRUCT >> print action [option] [region]
```

where the most commonly employed [option] is **all**, and [region] restricts the scope of the post-command action to a subset of the limit state loadings, load cases and/or frame geometry. For example, the command

```
STRUCT_print >>
STRUCT >> print action @ elmt 13 limst 2
   INFO >>
   INFO >> Limit State  2 Actions   :   entity name  : min value  : max value
   INFO >> ------------------------------------------------------------------
   INFO >> elmt  13 node   1 load  1    end moment        -249.6       194.7
   INFO >>                   load  2    end moment        -101.4       43.18
   INFO >>                   load  3    end moment        -114.5       70.71
   INFO >>            node  5 load  1    end moment        -182.1       73.87
   INFO >>                   load  2    end moment        -96.95      -13.89
   INFO >>                   load  3    end moment        -105.2       1.627
STRUCT_print_action >>
```

prints the maximum and minimum bending moments at the end of element 13 for each of the three ground motion inputs at limit state 2. Element axial forces are not printed in this example because the vertical degrees of freedom for limit state 2 were eliminated (see Section 4.13).

6.6 Plotting the Bending Moment Diagram

Plots of the frame bending moment diagram, and bending moments at the element level for static analyses may be drawn. The command syntax is

```
    STRUCT >> draw bmoment  [option]  [region]
```

where **all** and **item** are appropriate [options], and [region] is a subset of load cases, limit state loadings, or the frame geometry. For instance, in the script

```
    STRUCT >>
    STRUCT >> draw bmoment
    STRUCT_draw_bmoment >> help variable
      INFO_draw_bmoment >> SCALEBM =     50.00
    STRUCT_draw_bmoment >> all @ limst 1 lcase 1 and 2
    STRUCT_draw_bmoment >>
```

the **help variable** command is used to print out the current scale factor for drawing the bending moments. The option **all** is then employed to plot the frame bending moment diagrams for load cases 1 and 2 under gravity loads alone (see Figure 6.1). While this plot is useful for verifying the spatial distribution of bending throughout a structure, and the variation of bending moments among load cases, it does not allow the magnitude of moments along an element to be easily examined. This latter problem is handled by issuing a command of the type

```
    STRUCT >> draw bmoment  item [region]
```

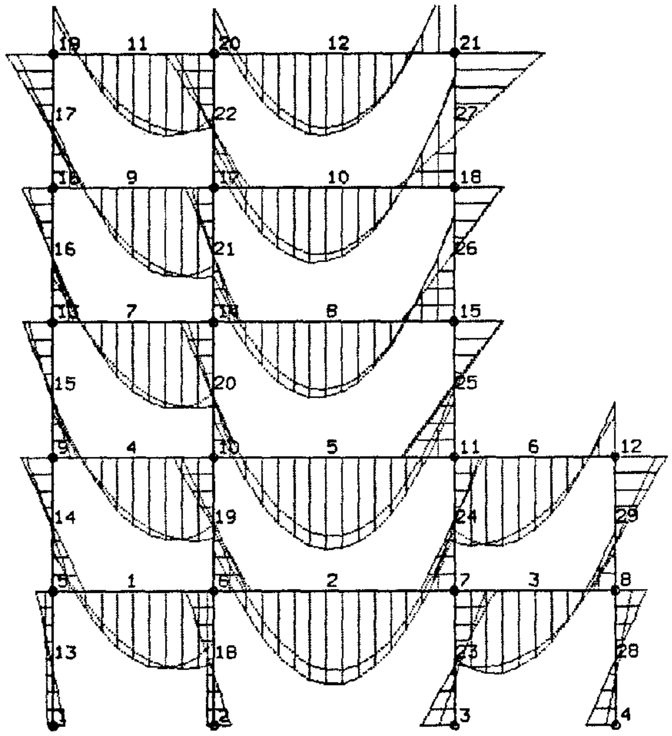


FIG. 6.1 : Bending Moment Diagram

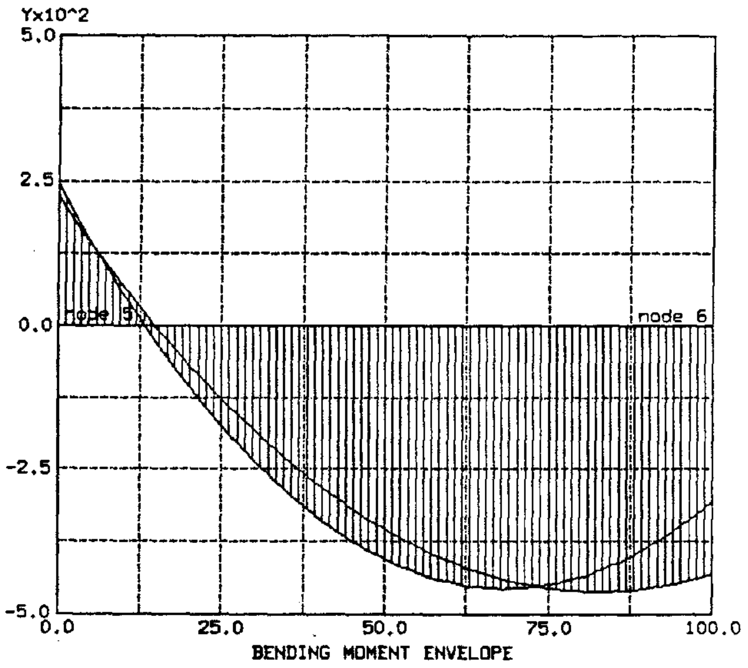


FIG. 6.2 : Bending Moment Item

where [region] includes the desired element number, as well as the range of limit states and load cases desired for the bending moment envelope. The results of the command

```
STRUCT >> draw bmoment item @ elmt 1 limst 1 lcase 1 and 2
```

are shown in Figure 6.2.

6.7 Plotting the Shear Force Diagrams

Plots of frame shear forces at the global and frame element levels may be prepared by following the command syntax

```
STRUCT >> draw shear [option] [region]
```

In all other respects the results are the same as for drawing bending moment envelope diagrams.

6.8 Frame Displacements

Summaries of extreme nodal displacements may be obtained with commands of the type

```
STRUCT >> print deflect [option] [region]
```

As an example of its implementation, the nodal displacements along column line 1 for stories 1 and 2 (ie; elements 13 and 14) for limit state 1 may be obtained by simply typing

```
STRUCT_print >>
STRUCT_print >> deflect @ story 1 and 2 coline 1 limst 1
   INFO >> Limit state 1 deflection         : min deflect : max deflect
   INFO >> ----------------------------------------------------------------
   INFO >> node    1 x_coord     0.00 lcase  1       0.00000        0.00000
   INFO >>                            lcase  2       0.00000        0.00000
   INFO >> node    1 y_coord     0.00 lcase  1       0.00000        0.00000
   INFO >>                            lcase  2       0.00000        0.00000
   INFO >> node    5 x_coord     0.00 lcase  1       0.03354        0.03354
   INFO >>                            lcase  2      -0.02766       -0.02766
   INFO >> node    5 y_coord    80.00 lcase  1      -0.01776       -0.01776
   INFO >>                            lcase  2      -0.01624       -0.01624
   INFO >> node    9 x_coord     0.00 lcase  1       0.06922        0.06922
   INFO >>                            lcase  2      -0.06102       -0.06102
   INFO >> node    9 y_coord   160.00 lcase  1      -0.03256       -0.03256
   INFO >>                            lcase  2      -0.02926       -0.02926
STRUCT_print_deflect >>
```

The main difference in responses between load cases 1 and 2 is due to the positive and negatively applied lateral loads specified in Section 4.8. The second point to note about the printed output is that the maximum and minimum nodal displacements are identical for static analyses. Clearly, this will not be the case for limit state calculations employing time-history responses.

6.9 Plotting the Deflected Frame

The syntax for plotting the deflected frame under gravity loads plus (possible) static lateral loads is

```
STRUCT >> draw deflect [option] [region]
```

The complete deflected frame may be plotted by issuing the **all** [option]. If a portion of the deflected frame frame is of interest, then this may be specified with the [region] part of the grammar. In addition, parameters exist for scaling the frame displacements and adjusting the frame colors before plotting. For example, the command sequence

```
STRUCT >>
STRUCT >> draw elmt all
STRUCT_draw_elmt >>
STRUCT_draw >> node all
STRUCT_draw_node >>
STRUCT_draw >> deflect
STRUCT_draw_deflect >> help variable
STRUCT_draw_deflect >>  SCALED =       100.00
STRUCT_draw_deflect >> SCALED = 400
STRUCT_draw_deflect >> all
STRUCT_draw_deflect >>
STRUCT_draw >>
```

draws the frame nodes and elements of the undeformed frame before superimposing the deformed frame with deflections magnified by a factor of 400 (see Figure 6.3).

6.10 Plotting Terms in the Energy Balance Equation

As outlined in Section 5.4, the energy balance equation is a useful tool for identifying the fundamental *cause and effect* mechanisms of earthquake induced damage, because it reduces a complex conglomerate of information about the structure and the ground motion(s) into a time dependent scalar equation. The command syntax for producing plots of the various terms in the energy balance equation is

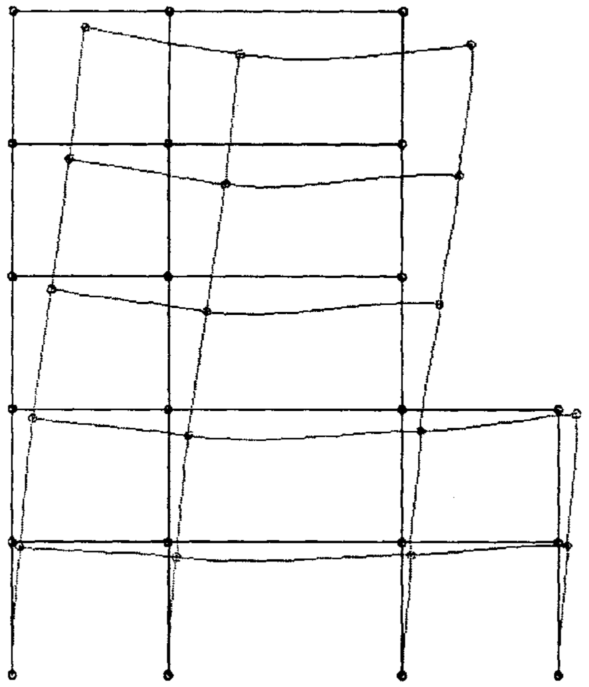


FIG. 6.3 : Deflected Frame

```
STRUCT >> draw energy [option] [region]
```

where the list of available energy options includes **input**, **hysteretic**, **damped**, and **kinetic**. For example, the command

```
STRUCT >> draw energy kinetic @ record 1 and 2
```

would be used to plot the time variation of kinetic energy for the **record1** and **record2** ground motion inputs, scaled to severe lateral load intensity. If an [option] is not specified, then the user is prompted for the terms to be included in the plots. For instance, results of the script

```
STRUCT >> draw energy @ record 1 to 3
  INFO >> Indicate energy terms to be plotted ...
  INFO >> .....input energy ( yes/no ) : yes
  INFO >> ....damped energy ( yes/no ) : no
  INFO >> ...kinetic energy ( yes/no ) : no
  INFO >> hysteretic energy ( yes/no ) : no
STRUCT >>
```

are shown in Figure 6.4.

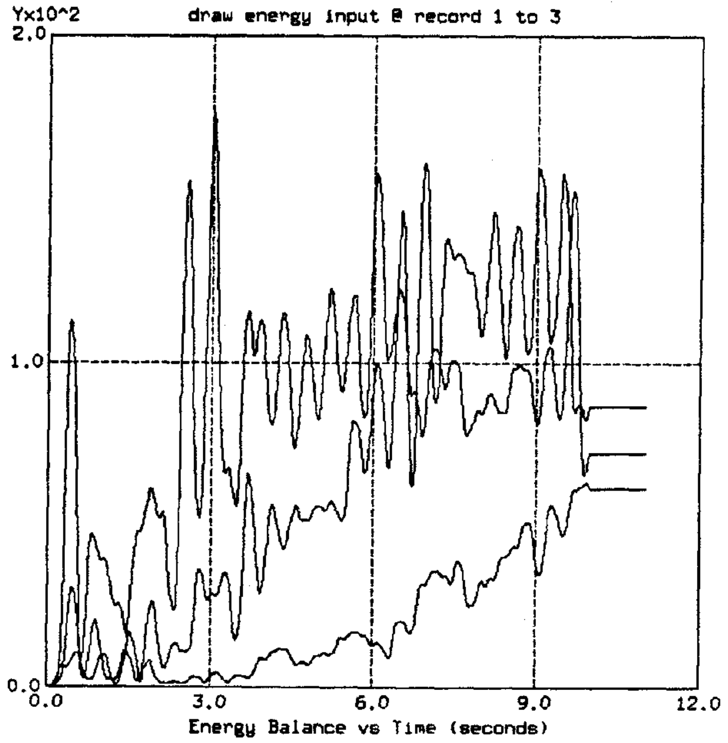


FIG. 6.4 : Input Energy

6.11 Evaluating Design Performance

The adequacy of a design is ascertained by comparing the calculated actions at the [HIGH,LOW] fractile of reliability to the ability of the structure to carry these loads without failure. The ability of the frame to carry loads and deform is described by a [GOOD,BAD] performance pair, where the GOOD level of response is a dependable value, and the BAD level of structural response a level at which undesirable performance is almost assured. A single design entity called *designer dissatisfaction* has been defined to facilitate this comparison. The command syntax for examining the terms contributing to design constraint performance is

```
    STRUCT >> draw dconst [option] [region]
```

where [option] and [region] take there usual meanings. For example, the command script

```
STRUCT >>
STRUCT >> print dconst @ limst 2 elmt 13 and 14
   INFO >>
   INFO >> Frame Responses Values for Limit State 1 Constraints
   INFO >> Elmt Node  :  Constr Name : dissat :    high :      low :      good :      bad
```

```
  INFO >> -------------------------------------------------------------------------
  INFO >>    13     1 bending moment   0.0000    211.58    241.06    2634.47   3219.91
  INFO >>           5 bending moment   0.0000    160.38    177.25    2634.47   3219.91
  INFO >>    14     5 bending moment   0.0000    116.81    121.04    2634.47   3219.91
  INFO >>           9 bending moment   0.0000    177.06    188.73    2634.47   3219.91
```

prints a concise performance summary of elements 13 and 14 under moderate lateral loads. A more graphical representation of the frame response quantities contributing to a design constraint, together with the calculated [HIGH,LOW] and [GOOD,BAD] bandwidths of frame response may be obtained with the command syntax:

```
STRUCT >> draw dconst item [region]
```

where a limit state loading and element number must be specified for the [region] part of the command. To illustrate these features, the results of the command script

```
STRUCT >> # plot elastic bending moment versus time
STRUCT >> # at element 4, limit state 2
STRUCT >>
STRUCT >> draw dconst item @ elmt 4 limst 2
   INFO >> Indicate design constraint entity to be plotted
   INFO >> moment i end          = 1  moment j end        = 2
   INFO >> axial force           = 3  axial displ         = 4
   INFO >> energy disp i end     = 5  energy disp j end = 6
   INFO >> floor accel           = 7  story drift         = 8
   INFO >> deflection            = 9
   INFO >>
   INFO >> ... type in the entity no : 1
STRUCT_draw_dconst_item >>
STRUCT_draw_dconst >>
STRUCT_draw_dconst >> # plot hysteretic energy @ elmt 4
STRUCT_draw_dconst >>
STRUCT_draw_dconst >> item @ elmt 4 limst 3
   INFO >> Indicate design constraint entity to be plotted
   INFO >> moment i end          = 1  moment j end        = 2
   INFO >> axial force           = 3  axial displ         = 4
   INFO >> energy disp i end     = 5  energy disp j end = 6
   INFO >> floor accel           = 7  story drift         = 8
   INFO >> deflection            = 9
   INFO >>
   INFO >> ... type in the entity no : 5
STRUCT_draw_dconst_item >>
```

are shown in Figures 6.5 and 6.6. The two most important features of these figures are the time variation and scatter among the frame response quantities, and the relative bandwidths bounded by frame responses at the [HIGH,LOW] fractiles of reliability, and [GOOD,BAD] levels of frame response. Figure 6.5 shows the bending moment versus time at element 4 for the elastic response of the frame due to three ground motion inputs scaled to moderate intensity. Dissatisfaction for this constraint is zero because the frame response levels at the HIGH and LOW fractiles of reliability are less than the dependable level for frame response (ie,

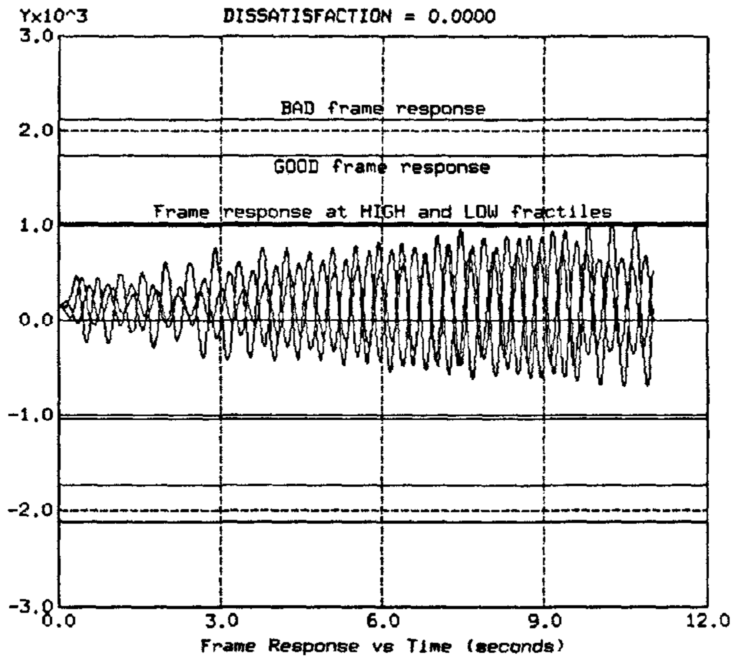


FIG. 6.5 : Design Constraint Item : Bending Moments

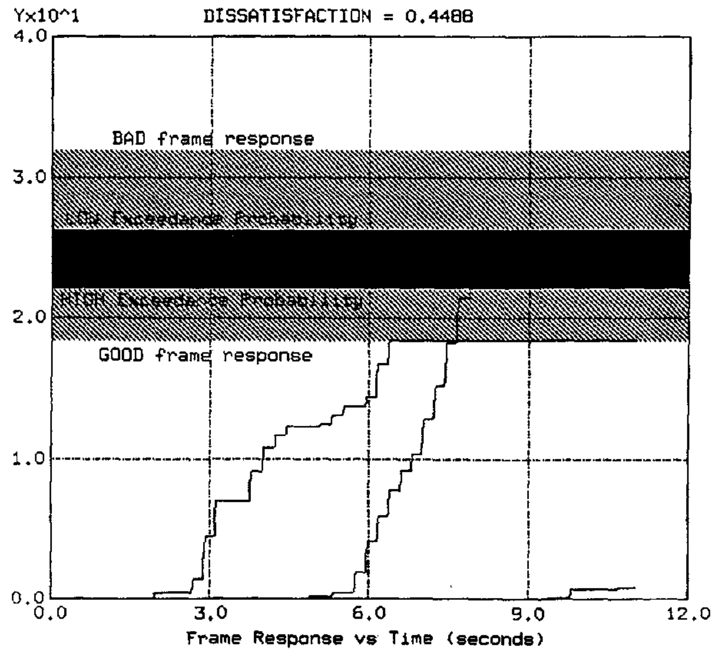


FIG. 6.6 : Design Constraint Item : Hysteretic Energy Dissipation

the GOOD level of frame response). Figure 6.6 shows the distribution of hysteretic energy dissipation for the same three ground motion inputs scaled to severe lateral load intensity. The scatter in response quantities in this case is much larger than the input energy (see Figure 6.4) and the elastic bending moment response. The main cause of the latter observation is excursions of the overall frame behavior into the inelastic range. As a result, a significant enhancement of the mean hysteretic energy dissipation response quantities is required before the HIGH and LOW fractiles of reliability are reached. A moderate level of dissatisfaction for the hysteretic energy dissipation constraint is calculated because the [HIGH,LOW] and [GOOD,BAD] bandwidths are intersecting. This case differs from Figure 6.5 in that the bandwidth between GOOD and BAD frame responses covers a major portion of the overall response (hysteretic energy dissipation). This suggests that efforts are needed to not only improve our understanding of the ground motion inputs, but also to obtain more precise estimates of required hysteretic energy dissipation to cause failure.

Frame performance attributes controlling a design may be identified by requesting a search of all the constraints having non-zero dissatisfaction. The command

```
STRUCT_print >>
STRUCT >>
STRUCT >> print dconst all @ limst 1 | dissat != 0
   INFO >>
   INFO >> Frame Responses for Limit State 1 Constraints
   INFO >> Elmt Node  : Constr Name : dissat : high :   low :   good :    bad
   INFO >> ---------------------------------------------------------------
   INFO >>   14     5    end moment    0.2430   0.00   0.00  283.64  378.19
```

demonstrates this feature by requesting a search over all the frame design constraints for the gravity loads alone limit state, printing only those constraints with nonzero dissatisfaction. Similarly, a graphical representation of the locations controlling a design may be obtained with the command format

```
STRUCT >> draw dconst all @ limst 1 | dissat != 0
```

CHAPTER 7 - CONCLUSIONS

7.1 Introduction

This report documents the ongoing development and implementation of a design methodology for the statistical limit states design of earthquake resistant structures. While the first report[6] in this series, and subsequent papers [8,9,10,11,36], focused on the description and prototype testing of the methodology, the purpose of this report has been to describe the initial stages of the design methodologies implementation in an engineering workstation environment. The development of computational tools for describing the design problem, graphically interpreting structural behavior, providing assistance in the comparison of design alternatives, and carrying out design optimization are all parts of the required implementation. As noted in Chapter 1, however, contributions to software projects of this type are incremental simply because no group has the personnel or time to complete this task in its entirety. The material presented in Chapters 3 to 6 is characteristic of this observation.

When this implementation was first started the development goal was to replicate the features of the DELIGHT.STRUCT environment, but with a significantly more flexible graphically oriented user interface. The contents of this report are a first step towards satisfying this objective. However, the recent interest in CSTRUCT shown by experimentally based research groups, and persons requiring teaching aids for earthquake design and structural analysis, indicates a much larger population of potential users than originally anticipated. In an effort to capitalize on this interest, the focus of software development has been modified to accommodate the demands of some of these special interest groups. Now there is a need to prepare a simplified version of CSTRUCT for use in structural analysis and design classes. At the time of writing (July-August 1987) the development of CSTRUCT is at the stage where prototype versions of the environment may be distributed to local research groups. It is the writers' expectation that useful feedback on the performance of CSTRUCT together with suggested enhancements will be provided in return. Therefore, the important reasons for writing this report have been to: (a) document the features of CSTRUCT, and (b) provide its users with an explanation of the ideas motivating this research project's long term goals.

The software development described herein has concentrated on the description of the design problem, the graphical interpretation of results, and implementation of the user interface. Still, a significant amount of programming is required before the capabilities of DELIGHT.STRUCT are mimicked. Continued work is needed for the presentation of design information in pop-up tables. In particular, the AISC tables should be organized into a convenient format that allows the designer to select section sizes or simply browse for information on section sizes. Another possibility is to build tables containing all of the design/modeling assumptions, and to provide users with an editor for making modifications as required. There is scope for improvement in the simulation capabilities of CSTRUCT. A useful extension would be to add Newmark-Hall Spectra[43] to CSTRUCT for preliminary design purposes. Since the time dependent responses of the structure are already stored, it should be a straight forward task to animate the linear and nonlinear structural response. A parallel extension would allow the time variation in bending moments due to dynamic loads to be drawn. Finally, the Phase I-II-III method of feasible directions algorithm[45] needs to be added to the environment, together with software for tuning the design constraint and objective parameters, monitoring algorithm performance, and graphically displaying the design changes over several iterations of optimization.

REFERENCES

1. Aho A.V., Sethi R., Ullman J.D., **Compilers : Principles, Techniques and Tools**, *Addison-Wesley*, 1985.

2. American Institute of Steel Construction, **Manual of Steel Construction**, Seventh Edition, 1973.

3. Applied Technology Council, "Working Draft of Recommended Seismic Design Provisions for Buildings," January 1976.

4. Applied Technology Council, "Tentative Provisions for the Development of Seismic Regulations for Buildings," National Bureau of Standards, June 1978.

5. Applied Technology Council, "Tentative Seismic Isolation Design Requirements," *Proceedings of a Seminar and Workshop on Base Isolation and Passive Energy Dissipation*, March 1986.

6. Austin M.A., Pister K.S., Mahin S.A., "A Methodology for the Computer-Aided Design of Seismic-Resistant Steel Structures," *Report* No UCB/EERC-85/13, Earthquake Engineering Research Center, University of California, Berkeley, December 1985.

7. Austin M.A., Pister K.S., "Design of Friction-Braced Frames under Seismic Loading," *Journal of the Structural Division*, ASCE, December 1985.

8. Austin M.A., Pister K.S., Mahin S.A., "A Methodology for the Probabilistic Limit States Design of Earthquake-Resistant Structures," *Journal of the Structural Division*, ASCE, August 1987.

9. Austin M.A., Pister K.S., Mahin S.A., "Probabilistic Limit States Design of Moment-Resistant Frames under Seismic Loading," *Journal of the Structural Division*, ASCE, August 1987.

10. Austin M.A., Pister K.S., Mahin S.A., "Probabilistic Limit States Design of Moment-Resistant Frames under Seismic Loading," *3rd U.S. National Conference on Earthquake Engineering*, Charleston, South Carolina, August 1986.

11. Austin M.A., Pister K.S., Mahin S.A., "Optimal Probabilistic Limit States Design of Earthquake Resistant Steel Structures," *Structures Congress 86*, ASCE, New Orleans, September 15-18 1986.

12. Balling R.J., Pister K.S., and Polak E., "DELIGHT.STRUCT: A Computer-Aided Design Environment for Structural Engineering," *Computer Methods in Applied Mechanics and Engineering*, (1983), pp 237-251, North-Holland Publishing Company.

13. Balling R.J., Ciampi V., Pister K.S., and Polak E., "Optimal Design of Seismic-Resistant Planar Steel Frames," *Report No* EERC 81-20, Earthquake Engineering Research Center, Univ of California, Berkeley, December 1981.

14. Bobrow D.G., Sanjay M., Stefik M.J., "Expert Systems : Perils and Promise," *Communications of the ACM.*, Sep't 1986, pp 880-894.

15. Bolt B.A., Tsai Y.B., Hsu M.K.,"Earthquake Strong Motions Recorded by a Large Near-Source Array of Digital Seismographs," *Earthquake Engineering and Structural Dynamics*, Vol 10, pp 561-573, 1982.

16. Calahan D.A., **Computer-Aided Network Design**, McGraw-Hill Book Company, New York(1972).

17. Chamberlin D.D., Boyce R.F., "SEQUEL : A Structured English Query Language," Proc. 1974 ACM SIGMOD *Workshop on Data Description, Access and Control.*

18. Chang M.K., Kwiakowski J.W., Nau R.F., Oliver R.M. and Pister K.S. "ARMA Models for Earthquake Ground Motions," *Earthquake Engineering and Structural Dynamics*, Vol 10, pp 651-662, 1982.

19. Conte J., Austin M.A., Pister K.S., Mahin S.A., "Use of ARMA Models in Earthquake-Induced Damage Mechanisms," *EERC Report*, November 1987 [in preparation].

20. Cramer R.A., "Database Primer : Things you should know about Database Systems ... but were afraid to ask," *Unix World*, February, 1986.

21. Feldman S., "Make - A Program for Maintaining Programs," *Software - Practice and Experience*, April 1979.

22. Foley J.D., Van Dam A.,**Fundamentals of Interactive Computer Graphics**, Addison-Wesley Publishing Company, 1983.

23. Gettys J., Newman R., Fera T.D., "Xlib - C language X Interface Protocol Version 10," Digital Equipment Corporation, 1986.

24. Goodwin N.C., "Functionality and Usability," *Communications of the ACM*, March, 1987, pp 229-233.

25. Gould J.D., Lewis C., "Designing for Usability : Key Principles and what Designers think," *Communications of the ACM*, Vol. 28, No. 3, March, 1985.

26. Gumbel E., **Statistics of Extremes**, *Columbia University Press*, 1958.

27. Hardwick M., Spooner D.L., "Comparison of some Data Models for Engineering Objects," *IEEE Computer Graphics and Applications*, March 1987, pp 56-66.

28. Heckel P., "The Elements of Friendly Software Design," *Warner Book Company*, 1982.

29. Johnson S.C.,"YACC - Yet another Compiler Compiler," *Computer Science Technical Report 32*, AT&T Bell Laboratories, Murray Hill, New Jersey, 1975.

30. Katz R.H. et al., "Design Version Management," *IEEE Design and Test*, February 1987.

31. Kennedy R.P., "Peak Acceleration as a Measure of Damage," Presented at the 4th International *Seminar on Extreme-Load Design of Nuclear Power Facilities*, Paris, France, 1981.

32. Kernighan B.W. "RATFOR : A Preprocessor for a Rational Fortran," *Software - Practice and Experience*, October 1975.

33. Kernighan B.W., Pike R., **The UNIX Programming Environment**, *Prentice-Hall Software Series*, 1984.

34. Khatib I.F. et al., "Dynamic Inelastic Behavior of Chevron Braced frames," *5th Canadian Conference on Earthquake Engineering*, Ottawa, July 6-8, 1987.

35. Khatib I.F., Mahin S,A., "Methods for Improving the Seismic Response of Concentrically Braced Steel Frames," *6th ASCE Structures Congress*, Orlando, Florida, August 17-20 1987

36. Khatib I.F., "Behavior and Optimization of Concentrically Braced Frames Subjected to Earthquake Loading," *In partial fullfillment of requirements for the Doctor of Engineering degree*, University of California, Berkeley, August, 1987.

37. Landers J., "SDMS : Structural Data Management System, VAX - Version 1.0," *Skidmore, Owings and Merrill*, May 1982.

38. Linton M. "dbx," *Masters Thesis*, University of California, Berkeley, 1982.

39. Mink C., "Speaking Without Words," *DEC Professional Review*, July 1987.

40. Mc Guire W., Pesquera C.L., "Interactive Computer Graphics in Steel Analysis/Design - A Progress Report," *Engineering Journal/American Institute of Steel Construction*, Third Quarter, 1983.

41. Mondkar D.P., Powell G.H., "ANSR - 1 General Purpose program for Analysis of Nonlinear Structural Response," *Report No. EERC 75-37*, Earthquake Engineering Research Center, U. C. Berkeley, December 1975.

42. Nau R.F., Oliver R.M. and Pister K.S., "Simulating and Analyzing Artificial Non-stationary Earthquake Ground Motions," University of California, *Technical Report* ORC 80-16, Operations Research Center, Berkeley, California (1980).

43. Newmark N.M., Hall W.J., **Earthquake Spectra and Design**, *Earthquake Engineering research Institute*, Berkeley, California, 1982.

44. Nye W.T., " DELIGHT : An Interactive System for Optimization-Based Engineering Design," UCB/ERL M83/33 Univ of California, Berkeley, CA, May 1983.

45. Nye W.T., Tits A.L., "An Application-Oriented, Optimization-Based Methodology for Interactive Design of Engineering Systems," *International Journal of Control* Vol. 43, No. 6, pp 1693-1721, 1986.

46. Osterhout J. "The Sx Window Library," Dep't of Computer Science, University of California, Berkeley, 1987.

47. Pall A.S., Marsh C., "Response of Friction Damped Braced Frames," *Journal of the Structural Division*, ASCE, Vol(108), pp 1313-1323, No ST6, June 1982.

48. Recommended Lateral Force Requirements and Commentary, Seismology Committee, Structural Engineers Association of California, San Francisco, Calif, 1975.

49. Reisner P.,"Formal Grammar and Human Factors Design of an Interactive Graphics System," *IEEE Trans. on Software Engineering*, SE-7(2), March 1981, pp 229-240.

50. Shen K., Fenves S.J., "Table : An Engineering, Scientific, and Management Application-Oriented Database Model," *DRC-12-16-83*, Design Research Center, Carnegie-Mellon University, Pittsburgh, PA 15213.

51. Sues R.H., "Stochastic Evaluation of Seismic Structural Performance," *Journal of the Structural Division*, Vol. 111, No. 6, June 1985.

52. Sreekanta Murthy T., Arora J.S., "Data Base Design Methodology for Structural Analysis and Design Optimization," *Engineering with Computers*, Volume 1, No. 3, 1986.

53. Uniform Building Code, International Conference of Building Officials, Whittier, CA, 1979 Edition.

54. Walker N.D.,"Automated Design of Earthquake Resistant Multistory Steel Building Frames," *Report No 77-12*, Earthquake Engineering Research Center, University of California, Berkeley, CA, May 1977.

55. Wilson E.L., Habibullah A., "SAP 80 : Structural Analysis Programs," *Users Guides*, Computers and Structures Inc, University Avenue, Berkeley, California.

56. Wuu T. ,"DELIGHT.MIMO : An Interactive System for Optimization-based Multivariable Control Systems Design," *Memorandum No. UCB/ERL M86/90*, Electronics Research Laboratory, University of California, Berkeley, December 1986.

57. Zienkiewicz O.C., Taylor R.L., **The Finite Element Method - Chapter 24**, *3rd Edition*, Mc Graw-Hill, New York, 1977, 787pp.

APPENDIX 1 - DATA STRUCTURES

A.1 Introduction

One important strength of the C programming language is its support for the logical organization and management of information. Arrays, trees and linked lists are all commonly employed data structures in the development of application programs. The issues in selecting the best model for a particular task include: (a) the ease of development, (b) the frequency at which the data is accessed, (c) the frequency at which the data is updated, and (d) the volume of data to be stored. Studies[27] indicate that while all models perform well in some aspects, no data model excels in all areas. Furthermore, memory for some data structures is more conveniently allocated at compile-time, while in other cases it is better to allocate and free memory at run-time. With this brief background in mind, the data structures used for the frame definition, frame geometry attributes, frame response storage, and frame performance assessment are now discussed.

A.1 Frame Definition

Information on the frame geometry and its material properties is described with a graph-based model that links the frame's elements and nodes with data pointers. Arrays of data structures are used in this prototype implementation; this data structures is relatively easy to implement and it allows information to be accessed very fast. For example, the script

```
typedef struct element {
        int node[ MAX_NODES_PER_ELEMENT ];
        int connectflag;
        int deleted;
        int material;       /* section material type             */
        int section_id;     /* section identification number     */
        int kind;           /* element kind : col,girder,disspator */
        double length;      /* element length                    */
      } ELEMENT_LIST, *ELEMENT_LIST_PTR;

ELEMENT_LIST elmts[ NO_ELEMENTS ];
```

makes a declaration for the frame element data type and then allocates memory for an array of of length NO_ELEMENTS containing the data structure *element*, Included in the declaration is space for the pointers to a second array of structures for the frame nodal coordinates. A three-dimensional array of structures of the form

```
typedef struct element_loads {
          float unif_dead_load;      /* uniform dead load line */
          float unif_live_load;      /* uniform live load line */
        } ELEMENT_LOADS, *ELEMENT_LOADS_PTR;

ELEMENT_LOADS elmt_loads[ NO_ELEMENTS ][ NO_LOADCASES ][ NO_LIMITSTATES ];
```

is used for the storage of the frame element dead and live loads, where the parameters NO_ELEMENTS, NO_LOADCASES, and NO_LIMITSTATES define the maximum number of frame elements, load cases for each limit state, and number of frame performance limit states, respectively.

A.2 Frame Geometry Attributes

In Section 4.5 the advantages of building frame geometry attributes on top of the lists elements and nodes was explained. The script

```
typedef struct arglist {
    double number;
        struct arglist *next;
    }  ARG_LIST,     *ARG_LIST_PTR;

typedef struct floor_conts {
        float y_coord;
    struct arglist *first_node;
    struct arglist *first_element;
        } FLOOR_CONTENTS, *FLOOR_CONTENTS_PTR,
          STORY_CONTENTS, *STORY_CONTENTS_PTR,

FLOOR_CONTENTS_PTR floor_contents[ NO_FLOORS ];
STORY_CONTENTS_PTR story_contents[ NO_STORYS ];
```

makes declares a generic linked list data structure and a second data structure containing pointers to the first member of the node and element linked lists belonging to the attribute. Finally, memory is allocated for the headers to the element and nodal lists at each story and floor level. An identical declaration process applies for the column lines and bays.

A.3 Storage of the Frame Response

Frame response storage is by far the most demanding factor on the overall requirements for program storage. For this reason, memory for a 3-dimensional array of structures is not automatically allocated. A 2-dimensional array spanning the maximum number of frame elements (NO_ELEMENTS) and the number of limit states (NO_LIMITSTATES) is declared instead. Within each element of the array is a 1-dimensional array of pointers for each of the potential response quantities to be stored. A response storage template having the layout

```
typedef struct resp_attr {
        int   activated;          /* "activated" vs "not activated"  */
        char      *name;          /* response attribute "name"       */
        float     *resp;          /* address of the storage array    */
        int resp_length;          /* length of response storage      */
        double max_value;
        double min_value;
        } RESP_ATTR, *RESP_ATTR_PTR;
```

is the first declaration for the response storage. The template includes a pointer to a character array for its name, a pointer to an array of length *resp_length* for the storage of response values, as well as maximum and minimum values of frame response. Two sets of more general frame response templates

```
typedef struct frame_elmt_resp {
        /* (a) : response flag "stored vs not-stored" */
        int store_moment_i_end;
        int store_moment_j_end;
        int store_rotation_i_end;
        int store_rotation_j_end;
        int store_axial_force;
        int store_axial_displ;
        int store_energy_disp_i_end;
        int store_energy_disp_j_end;
        int store_energy_disp_total;
        /* (b) : pointer to response storage */
        struct resp_attr *moment_i_end[ NO_LOADCASES ];
        struct resp_attr *moment_j_end[ NO_LOADCASES ];
        struct resp_attr *rotation_i_end[ NO_LOADCASES ];
        struct resp_attr *rotation_j_end[ NO_LOADCASES ];
        struct resp_attr *axial_force[ NO_LOADCASES ];
        struct resp_attr *axial_displ[ NO_LOADCASES ];
        struct resp_attr *energy_disp_i_end[ NO_LOADCASES ];
        struct resp_attr *energy_disp_j_end[ NO_LOADCASES ];
        struct resp_attr *energy_disp_total[ NO_LOADCASES ];
        } RESP_ELMT_LIST, *RESP_ELMT_LIST_PTR;

typedef struct frame_node_resp {
        /* (a) : response flag "stored vs not-stored" */
        int store_x_accel;
        int store_x_veloc;
        int store_x_displ;
        int store_y_accel;
        int store_y_veloc;
        int store_y_displ;
        int store_rotation;
        /* (b) : pointer to response storage */
        struct resp_attr  *x_accel[ NO_LOADCASES ];
        struct resp_attr  *x_veloc[ NO_LOADCASES ];
        struct resp_attr  *x_displ[ NO_LOADCASES ];
        struct resp_attr  *y_accel[ NO_LOADCASES ];
        struct resp_attr  *y_veloc[ NO_LOADCASES ];
        struct resp_attr  *y_displ[ NO_LOADCASES ];
        struct resp_attr *rotation[ NO_LOADCASES ];
        } RESP_NODE_LIST, *RESP_NODE_LIST_PTR;

RESP_ELMT_LIST elmt_resp[ NO_LIMITSTATES ][ MAX_NO_ELEMENTS ];
RESP_NODE_LIST node_resp[ NO_LIMITSTATES ][ MAX_NO_NODES ];
```

are then declared for the storage of frame responses at the element and nodal levels respectively. The main features of this declaration are a flag indicating whether or not each of the frame response attributes is to be stored, and arrays of pointers of length NO_LOADCASES to the templates containing the response storage. An important point to keep in mind is the strong connection among the modeling assumptions, the form of output obtained, and way in which the simulation results may be used to evaluate performance. Consistency among the modeling assumptions, expected behavior, and anticipated response must be maintained.

A.4 Optimization Description/Design Constraints

Discussion in this section is limited to the data structures for the design constraints so that its length is kept reasonable. Design constraint templates of the form

```
typedef struct const_attr {
        char *name;               /* constraint "name"                 */
        int type;                 /* constraint type : HARD and SOFT   */
        int distr_type;           /* statistical distribution type     */
        float good_value;         /* GOOD constraint value             */
        float bad_value;          /* BAD  constraint value             */
        float high_value;         /* HIGH exceedance probability       */
        float low_value;          /* LOW  exceedance probability       */
        float mean_resp_value;    /* mean frame response value         */
        float std_resp_value;     /* std  frame response value         */
        float good_resp_value;    /* GOOD frame response value         */
        float bad_resp_value;     /* BAD  frame response value         */
        float high_resp_value;    /* response at HIGH exceedance prob  */
        float low_resp_value;     /* response at  LOW exceedance prob  */
        float dissatisfaction;    /* constraint dissatisfaction        */
        } CONST_ATTR, *CONST_ATTR_PTR;
```

are declared in an analogous manner to the response storage. Associated with each constraint [see Equation (1)] are designer specified GOOD and BAD values for allowable frame performance, and HIGH and LOW levels of frame response reliability. After the limit state simulations are completed, the frame response values corresponding specified parameter values are calculated, and stored. The levels of designer dissatisfaction follow directly.

Design constraints are defined at the element and nodal levels. For example the declaration for the storage of constraints at the element level is

```
typedef struct elmt_constraint {
        /* (a) : constraint flag "activated" vs "not activated" */
        int act_moment_i_end;
        int act_moment_j_end;
        int act_axial_force;
        int act_axial_displ;
        int act_energy_disp_i_end;
        int act_energy_disp_j_end;
        int act_energy_disp_total;
        int act_deflection;
        /* (b) : pointer to constraint storage */
        struct const_attr *moment_i_end;
        struct const_attr *moment_j_end;
        struct const_attr *axial_force;
        struct const_attr *axial_displ;
        struct const_attr *energy_disp_i_end;
        struct const_attr *energy_disp_j_end;
        struct const_attr *energy_disp_total;
        struct const_attr *deflection;
        } CONST_ELMT_LIST, *CONST_ELMT_LIST_PTR;

CONST_ELMT_LIST elmt_const[ NO_LIMITSTATES ][ MAX_NO_ELEMENTS ];
```

where the parameters *act_moment_i_end* and so on, indicate which constraints are *activated* versus *notactivated*. A similar declaration is made for the story drift and floor acceleration constraints stored at the nodal level.

EARTHQUAKE ENGINEERING RESEARCH CENTER REPORT SERIES

EERC reports are available from the National Information Service for Earthquake Engineering(NISEE) and from the National Technical Information Service(NTIS). Numbers in parentheses are Accession Numbers assigned by the National Technical Information Service; these are followed by a price code. Contact NTIS, 5285 Port Royal Road, Springfield Virginia, 22161 for more information. Reports without Accession Numbers were not available from NTIS at the time of printing. For a current complete list of EERC reports (from EERC 67-1) and availablity information, please contact University of California, EERC, NISEE, 1301 South 46th Street, Richmond, California 94804.

UCB/EERC-80/01	"Earthquake Response of Concrete Gravity Dams Including Hydrodynamic and Foundation Interaction Effects," by Chopra, A.K., Chakrabarti, P. and Gupta, S., January 1980, (AD-A087297)A10.
UCB/EERC-80/02	"Rocking Response of Rigid Blocks to Earthquakes," by Yim, C.S., Chopra, A.K. and Penzien, J., January 1980, (PB80 166 002)A04.
UCB/EERC-80/03	"Optimum Inelastic Design of Seismic-Resistant Reinforced Concrete Frame Structures," by Zagajeski, S.W. and Bertero, V.V., January 1980, (PB80 164 635)A06.
UCB/EERC-80/04	"Effects of Amount and Arrangement of Wall-Panel Reinforcement on Hysteretic Behavior of Reinforced Concrete Walls," by Iliya, R. and Bertero, V.V., February 1980, (PB81 122 525)A09.
UCB/EERC-80/05	"Shaking Table Research on Concrete Dam Models," by Niwa, A. and Clough, R.W., September 1980, (PB81 122 368)A06.
UCB/EERC-80/06	"The Design of Steel Energy-Absorbing Restrainers and their Incorporation into Nuclear Power Plants for Enhanced Safety (Vol 1a): Piping with Energy Absorbing Restrainers: Parameter Study on Small Systems," by Powell, G.H., Oughourlian, C. and Simons, J., June 1980.
UCB/EERC-80/07	"Inelastic Torsional Response of Structures Subjected to Earthquake Ground Motions," by Yamazaki, Y., April 1980, (PB81 122 327)A08.
UCB/EERC-80/08	"Study of X-Braced Steel Frame Structures under Earthquake Simulation," by Ghanaat, Y., April 1980, (PB81 122 335)A11.
UCB/EERC-80/09	"Hybrid Modelling of Soil-Structure Interaction," by Gupta, S., Lin, T.W. and Penzien, J., May 1980, (PB81 122 319)A07.
UCB/EERC-80/10	"General Applicability of a Nonlinear Model of a One Story Steel Frame," by Sveinsson, B.I. and McNiven, H.D., May 1980, (PB81 124 877)A06.
UCB/EERC-80/11	"A Green-Function Method for Wave Interaction with a Submerged Body," by Kioka, W., April 1980, (PB81 122 269)A07.
UCB/EERC-80/12	"Hydrodynamic Pressure and Added Mass for Axisymmetric Bodies.," by Nilrat, F., May 1980, (PB81 122 343)A08.
UCB/EERC-80/13	"Treatment of Non-Linear Drag Forces Acting on Offshore Platforms," by Dao, B.V. and Penzien, J., May 1980, (PB81 153 413)A07.
UCB/EERC-80/14	"2D Plane/Axisymmetric Solid Element (Type 3-Elastic or Elastic-Perfectly Plastic) for the ANSR-II Program," by Mondkar, D.P. and Powell, G.H., July 1980, (PB81 122 350)A03.
UCB/EERC-80/15	"A Response Spectrum Method for Random Vibrations," by Der Kiureghian, A., June 1981, (PB81 122 301)A03.
UCB/EERC-80/16	"Cyclic Inelastic Buckling of Tubular Steel Braces," by Zayas, V.A., Popov, E.P. and Martin, S.A., June 1981, (PB81 124 885)A10.
UCB/EERC-80/17	"Dynamic Response of Simple Arch Dams Including Hydrodynamic Interaction," by Porter, C.S. and Chopra, A.K., July 1981, (PB81 124 000)A13.
UCB/EERC-80/18	"Experimental Testing of a Friction Damped Aseismic Base Isolation System with Fail-Safe Characteristics," by Kelly, J.M., Beucke, K.E. and Skinner, M.S., July 1980, (PB81 148 595)A04.
UCB/EERC-80/19	"The Design of Steel Emergy-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 Supported Excitations," by Lee, M.C. and Penzien, J., June 1980, (PB82 201 872)A08.
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 Dickens, J.M. and Wilson, E.L., June 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 Kelly, J.M. and Skinner, M.S., June 1980.
UCB/EERC-80/22	"3D Solid Element (Type 4-Elastic or Elastic-Perfectly-Plastic) for the ANSR-II Program," by Mondkar, D.P. and Powell, G.H., July 1980, (PB81 123 242)A03.
UCB/EERC-80/23	"Gap-Friction Element (Type 5) for the Ansr-II Program," by Mondkar, D.P. and Powell, G.H., July 1980, (PB81 122 285)A03.
UCB/EERC-80/24	"U-Bar Restraint Element (Type 11) for the ANSR-II Program," by Oughourlian, C. and Powell, G.H., July 1980, (PB81 122 293)A03.
UCB/EERC-80/25	"Testing of a Natural Rubber Base Isolation System by an Explosively Simulated Earthquake," by Kelly, J.M., August 1980, (PB81 201 360)A04.
UCB/EERC-80/26	"Input Identification from Structural Vibrational Response," by Hu, Y., August 1980, (PB81 152 308)A05.
UCB/EERC-80/27	"Cyclic Inelastic Behavior of Steel Offshore Structures," by Zayas, V.A., Mahin, S.A. and Popov, E.P., August 1980, (PB81 196 180)A15.
UCB/EERC-80/28	"Shaking Table Testing of a Reinforced Concrete Frame with Biaxial Response," by Oliva, M.G., October 1980, (PB81 154 304)A10.
UCB/EERC-80/29	"Dynamic Properties of a Twelve-Story Prefabricated Panel Building," by Bouwkamp, J.G., Kollegger, J.P. and Stephen, R.M., October 1980, (PB82 138 777)A07.
UCB/EERC-80/30	"Dynamic Properties of an Eight-Story Prefabricated Panel Building," by Bouwkamp, J.G., Kollegger, J.P. and Stephen, R.M., October 1980, (PB81 200 313)A05.
UCB/EERC-80/31	"Predictive Dynamic Response of Panel Type Structures under Earthquakes," by Kollegger, J.P. and Bouwkamp, J.G., October 1980, (PB81 152 316)A04.

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 Fatique," by Spanner, P., Parker, E.R., Jongewaard, E. and Dory, M., 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 Stiemer, S.F. and Godden, W.G., September 1980, (PB82 201 880)A05.
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 Spencer, P., 1980.
UCB/EERC-80/35	"Experimental Testing of an Energy-Absorbing Base Isolation System," by Kelly, J.M., Skinner, M.S. and Beucke, K.E., October 1980, (PB81 154 072)A04.
UCB/EERC-80/36	"Simulating and Analyzing Artificial Non-Stationary Earth Ground Motions," by Nau, R.F., Oliver, R.M. and Pister, K.S., October 1980, (PB81 153 397)A04.
UCB/EERC-80/37	"Earthquake Engineering at Berkeley - 1980," by , September 1980, (PB81 205 674)A09.
UCB/EERC-80/38	"Inelastic Seismic Analysis of Large Panel Buildings," by Schricker, V. and Powell, G.H., September 1980, (PB81 154 338)A13.
UCB/EERC-80/39	"Dynamic Response of Embankment, Concrete-Gavity and Arch Dams Including Hydrodynamic Interation," by Hall, J.F. and Chopra, A.K., October 1980, (PB81 152 324)A11.
UCB/EERC-80/40	"Inelastic Buckling of Steel Struts under Cyclic Load Reversal.," by Black, R.G. , Wenger, W.A. and Popov, E.P., October 1980, (PB81 154 312)A08.
UCB/EERC-80/41	"Influence of Site Characteristics on Buildings Damage during the October 3,1974 Lima Earthquake," by Repetto, P., Arango, I. and Seed, H.B., September 1980, (PB81 161 739)A05.
UCB/EERC-80/42	"Evaluation of a Shaking Table Test Program on Response Behavior of a Two Story Reinforced Concrete Frame," by Blondet, J.M., Clough, R.W. and Mahin, S.A., December 1980, (PB82 196 544)A11.
UCB/EERC-80/43	"Modelling of Soil-Structure Interaction by Finite and Infinite Elements," by Medina, F., December 1980, (PB81 229 270)A04.
UCB/EERC-81/01	"Control of Seismic Response of Piping Systems and Other Structures by Base Isolation," by Kelly, J.M., January 1981, (PB81 200 735)A05.
UCB/EERC-81/02	"OPTNSR- An Interactive Software System for Optimal Design of Statically and Dynamically Loaded Structures with Nonlinear Response," by Bhatti, M.A., Ciampi, V. and Pister, K.S., January 1981, (PB81 218 851)A09.
UCB/EERC-81/03	"Analysis of Local Variations in Free Field Seismic Ground Motions," by Chen, J.-C., Lysmer, J. and Seed, H.B., January 1981, (AD-A099508)A13.
UCB/EERC-81/04	"Inelastic Structural Modeling of Braced Offshore Platforms for Seismic Loading. ," by Zayas, V.A., Shing, P.-S.B., Mahin, S.A. and Popov, E.P., January 1981, (PB82 138 777)A07.
UCB/EERC-81/05	"Dynamic Response of Light Equipment in Structures," by Der Kiureghian, A., Sackman, J.L. and Nour-Omid, B., April 1981, (PB81 218 497)A04.
UCB/EERC-81/06	"Preliminary Experimental Investigation of a Broad Base Liquid Storage Tank," by Bouwkamp, J.G., Kollegger, J.P. and Stephen, R.M., May 1981, (PB82 140 385)A03.
UCB/EERC-81/07	"The Seismic Resistant Design of Reinforced Concrete Coupled Structural Walls," by Aktan, A.E. and Bertero, V.V., June 1981, (PB82 113 358)A11.
UCB/EERC-81/08	"Unassigned," by Unassigned, 1981.
UCB/EERC-81/09	"Experimental Behavior of a Spatial Piping System with Steel Energy Absorbers Subjected to a Simulated Differential Seismic Input," by Stiemer, S.F., Godden, W.G. and Kelly, J.M., July 1981, (PB82 201 898)A04.
UCB/EERC-81/10	"Evaluation of Seismic Design Provisions for Masonry in the United States," by Sveinsson, B.I., Mayes, R.L. and McNiven, H.D., August 1981, (PB82 166 075)A08.
UCB/EERC-81/11	"Two-Dimensional Hybrid Modelling of Soil-Structure Interaction," by Tzong, T.-J., Gupta, S. and Penzien, J., August 1981, (PB82 142 118)A04.
UCB/EERC-81/12	"Studies on Effects of Infills in Seismic Resistant R/C Construction," by Brokken, S. and Bertero, V.V., October 1981, (PB82 166 190)A09.
UCB/EERC-81/13	"Linear Models to Predict the Nonlinear Seismic Behavior of a One-Story Steel Frame," by Valdimarsson, H., Shah, A.H. and McNiven, H.D., September 1981, (PB82 138 793)A07.
UCB/EERC-81/14	"TLUSH: A Computer Program for the Three-Dimensional Dynamic Analysis of Earth Dams," by Kagawa, T., Mejia, L.H., Seed, H.B. and Lysmer, J., September 1981, (PB82 139 940)A06.
UCB/EERC-81/15	"Three Dimensional Dynamic Response Analysis of Earth Dams," by Mejia, L.H. and Seed, H.B., September 1981, (PB82 137 274)A12.
UCB/EERC-81/16	"Experimental Study of Lead and Elastomeric Dampers for Base Isolation Systems," by Kelly, J.M. and Hodder, S.B., October 1981, (PB82 166 182)A05.
UCB/EERC-81/17	"The Influence of Base Isolation on the Seismic Response of Light Secondary Equipment," by Kelly, J.M., April 1981, (PB82 255 266)A04.
UCB/EERC-81/18	"Studies on Evaluation of Shaking Table Response Analysis Procedures," by Blondet, J. Marcial, November 1981, (PB82 197 278)A10.
UCB/EERC-81/19	"DELIGHT.STRUCT: A Computer-Aided Design Environment for Structural Engineering. ," by Balling, R.J., Pister, K.S. and Polak, E., December 1981, (PB82 218 496)A07.
UCB/EERC-81/20	"Optimal Design of Seismic-Resistant Planar Steel Frames," by Balling, R.J., Ciampi, V. and Pister, K.S., December 1981, (PB82 220 179)A07.

UCB/EERC-82/01 "Dynamic Behavior of Ground for Seismic Analysis of Lifeline Systems," by Sato, T. and Der Kiureghian, A., January 1982, (PB82 218 926)A05.

UCB/EERC-82/02 "Shaking Table Tests of a Tubular Steel Frame Model," by Ghanaat, Y. and Clough, R.W., January 1982, (PB82 220 161)A07.

UCB/EERC-82/03 "Behavior of a Piping System under Seismic Excitation: Experimental Investigations of a Spatial Piping System supported by Mechanical Shock Arrestors," by Schneider, S., Lee, H.-M. and Godden, W. G., May 1982, (PB83 172 544)A09.

UCB/EERC-82/04 "New Approaches for the Dynamic Analysis of Large Structural Systems," by Wilson, E.L., June 1982, (PB83 148 080)A05.

UCB/EERC-82/05 "Model Study of Effects of Damage on the Vibration Properties of Steel Offshore Platforms," by Shahrivar, F. and Bouwkamp, J.G., June 1982, (PB83 148 742)A10.

UCB/EERC-82/06 "States of the Art and Pratice in the Optimum Seismic Design and Analytical Response Prediction of R/C Frame Wall Structures," by Aktan, A.E. and Bertero, V.V., July 1982, (PB83 147 736)A05.

UCB/EERC-82/07 "Further Study of the Earthquake Response of a Broad Cylindrical Liquid-Storage Tank Model," by Manos, G.C. and Clough, R.W., July 1982, (PB83 147 744)A11.

UCB/EERC-82/08 "An Evaluation of the Design and Analytical Seismic Response of a Seven Story Reinforced Concrete Frame," by Charney, F.A. and Bertero, V.V., July 1982, (PB83 157 628)A09.

UCB/EERC-82/09 "Fluid-Structure Interactions: Added Mass Computations for Incompressible Fluid. ," by Kuo, J.S.-H., August 1982, (PB83 156 281)A07.

UCB/EERC-82/10 "Joint-Opening Nonlinear Mechanism: Interface Smeared Crack Model," by Kuo, J.S.-H., August 1982, (PB83 149 195)A05.

UCB/EERC-82/11 "Dynamic Response Analysis of Techi Dam," by Clough, R.W., Stephen, R.M. and Kuo, J.S.-H., August 1982, (PB83 147 496)A06.

UCB/EERC-82/12 "Prediction of the Seismic Response of R/C Frame-Coupled Wall Structures," by Aktan, A.E., Bertero, V.V. and Piazzo, M., August 1982, (PB83 149 203)A09.

UCB/EERC-82/13 "Preliminary Report on the Smart 1 Strong Motion Array in Taiwan," by Bolt, B.A. , Loh, C.H., Penzien, J. and Tsai, Y.B., August 1982, (PB83 159 400)A10.

UCB/EERC-82/14 "Shaking-Table Studies of an Eccentrically X-Braced Steel Structure," by Yang, M.S., September 1982, (PB83 260 778)A12.

UCB/EERC-82/15 "The Performance of Stairways in Earthquakes," by Roha, C., Axley, J.W. and Bertero, V.V., September 1982, (PB83 157 693)A07.

UCB/EERC-82/16 "The Behavior of Submerged Multiple Bodies in Earthquakes," by Liao, W.-G., September 1982, (PB83 158 709)A07.

UCB/EERC-82/17 "Effects of Concrete Types and Loading Conditions on Local Bond-Slip Relationships," by Cowell, A.D., Popov, E.P. and Bertero, V.V., September 1982, (PB83 153 577)A04.

UCB/EERC-82/18 "Mechanical Behavior of Shear Wall Vertical Boundary Members: An Experimental Investigation," by Wagner, M.T. and Bertero, V.V., October 1982, (PB83 159 764)A05.

UCB/EERC-82/19 "Experimental Studies of Multi-support Seismic Loading on Piping Systems," by Kelly, J.M. and Cowell, A.D., November 1982.

UCB/EERC-82/20 "Generalized Plastic Hinge Concepts for 3D Beam-Column Elements," by Chen, P. F.-S. and Powell, G.H., November 1982, (PB83 247 981)A13.

UCB/EERC-82/21 "ANSR-II: General Computer Program for Nonlinear Structural Analysis," by Oughourlian, C.V. and Powell, G.H., November 1982, (PB83 251 330)A12.

UCB/EERC-82/22 "Solution Strategies for Statically Loaded Nonlinear Structures," by Simons, J.W. and Powell, G.H., November 1982, (PB83 197 970)A06.

UCB/EERC-82/23 "Analytical Model of Deformed Bar Anchorages under Generalized Excitations," by Ciampi, V., Eligehausen, R., Bertero, V.V. and Popov, E.P., November 1982, (PB83 169 532)A06.

UCB/EERC-82/24 "A Mathematical Model for the Response of Masonry Walls to Dynamic Excitations," by Sucuoglu, H., Mengi, Y. and McNiven, H.D., November 1982, (PB83 169 011)A07.

UCB/EERC-82/25 "Earthquake Response Considerations of Broad Liquid Storage Tanks," by Cambra, F.J., November 1982, (PB83 251 215)A09.

UCB/EERC-82/26 "Computational Models for Cyclic Plasticity, Rate Dependence and Creep," by Mosaddad, B. and Powell, G.H., November 1982, (PB83 245 829)A08.

UCB/EERC-82/27 "Inelastic Analysis of Piping and Tubular Structures," by Mahasuverachai, M. and Powell, G.H., November 1982, (PB83 249 987)A07.

UCB/EERC-83/01 "The Economic Feasibility of Seismic Rehabilitation of Buildings by Base Isolation," by Kelly, J.M., January 1983, (PB83 197 988)A05.

UCB/EERC-83/02 "Seismic Moment Connections for Moment-Resisting Steel Frames.," by Popov, E.P., January 1983, (PB83 195 412)A04.

UCB/EERC-83/03 "Design of Links and Beam-to-Column Connections for Eccentrically Braced Steel Frames," by Popov, E.P. and Malley, J.O., January 1983, (PB83 194 811)A04.

UCB/EERC-83/04 "Numerical Techniques for the Evaluation of Soil-Structure Interaction Effects in the Time Domain," by Bayo, E. and Wilson, E.L., February 1983, (PB83 245 605)A09.

UCB/EERC-83/05 "A Transducer for Measuring the Internal Forces in the Columns of a Frame-Wall Reinforced Concrete Structure," by Sause, R. and Bertero, V.V., May 1983, (PB84 119 494)A06.

UCB/EERC-83/06 "Dynamic Interactions Between Floating Ice and Offshore Structures," by Croteau, P., May 1983, (PB84 119 486)A16.

UCB/EERC-83/07 "Dynamic Analysis of Multiply Tuned and Arbitrarily Supported Secondary Systems. ," by Igusa, T. and Der Kiureghian, A., July 1983, (PB84 118 272)A11.

UCB/EERC-83/08 "A Laboratory Study of Submerged Multi-body Systems in Earthquakes," by Ansari, G.R., June 1983, (PB83 261 842)A17.

UCB/EERC-83/09 "Effects of Transient Foundation Uplift on Earthquake Response of Structures," by Yim, C.-S. and Chopra, A.K., June 1983, (PB83 261 396)A07.

UCB/EERC-83/10 "Optimal Design of Friction-Braced Frames under Seismic Loading," by Austin, M.A. and Pister, K.S., June 1983, (PB84 119 288)A06.

UCB/EERC-83/11 "Shaking Table Study of Single-Story Masonry Houses: Dynamic Performance under Three Component Seismic Input and Recommendations," by Manos, G.C., Clough, R.W. and Mayes, R.L., July 1983, (UCB/EERC-83/11)A08.

UCB/EERC-83/12 "Experimental Error Propagation in Pseudodynamic Testing," by Shiing, P.B. and Mahin, S.A., June 1983, (PB84 119 270)A09.

UCB/EERC-83/13 "Experimental and Analytical Predictions of the Mechanical Characteristics of a 1/5-scale Model of a 7-story R/C Frame-Wall Building Structure," by Aktan, A.E., Bertero, V.V., Chowdhury, A.A. and Nagashima, T., June 1983, (PB84 119 213)A07.

UCB/EERC-83/14 "Shaking Table Tests of Large-Panel Precast Concrete Building System Assemblages," by Oliva, M.G. and Clough, R.W., June 1983, (PB86 110 210/AS)A11.

UCB/EERC-83/15 "Seismic Behavior of Active Beam Links in Eccentrically Braced Frames," by Hjelmstad, K.D. and Popov, E.P., July 1983, (PB84 119 676)A09.

UCB/EERC-83/16 "System Identification of Structures with Joint Rotation," by Dimsdale, J.S., July 1983, (PB84 192 210)A06.

UCB/EERC-83/17 "Construction of Inelastic Response Spectra for Single-Degree-of-Freedom Systems," by Mahin, S. and Lin, J., June 1983, (PB84 208 834)A05.

UCB/EERC-83/18 "Interactive Computer Analysis Methods for Predicting the Inelastic Cyclic Behaviour of Structural Sections," by Kaba, S. and Mahin, S., July 1983, (PB84 192 012)A06.

UCB/EERC-83/19 "Effects of Bond Deterioration on Hysteretic Behavior of Reinforced Concrete Joints," by Filippou, F.C., Popov, E.P. and Bertero, V.V., August 1983, (PB84 192 020)A10.

UCB/EERC-83/20 "Analytical and Experimental Correlation of Large-Panel Precast Building System Performance," by Oliva, M.G., Clough, R.W., Velkov, M. and Gavrilovic, P., November 1983.

UCB/EERC-83/21 "Mechanical Characteristics of Materials Used in a 1/5 Scale Model of a 7-Story Reinforced Concrete Test Structure," by Bertero, V.V., Aktan, A.E., Harris, H.G. and Chowdhury, A.A., October 1983, (PB84 193 697)A05.

UCB/EERC-83/22 "Hybrid Modelling of Soil-Structure Interaction in Layered Media," by Tzong, T.-J. and Penzien, J., October 1983, (PB84 192 178)A08.

UCB/EERC-83/23 "Local Bond Stress-Slip Relationships of Deformed Bars under Generalized Excitations," by Eligehausen, R., Popov, E.P. and Bertero, V.V., October 1983, (PB84 192 848)A09.

UCB/EERC-83/24 "Design Considerations for Shear Links in Eccentrically Braced Frames," by Malley, J.O. and Popov, E.P., November 1983, (PB84 192 186)A07.

UCB/EERC-84/01 "Pseudodynamic Test Method for Seismic Performance Evaluation: Theory and Implementation," by Shing, P.-S. B. and Mahin, S.A., January 1984, (PB84 190 644)A08.

UCB/EERC-84/02 "Dynamic Response Behavior of Kiang Hong Dian Dam," by Clough, R.W., Chang, K.-T., Chen, H.-Q. and Stephen, R.M., April 1984, (PB84 209 402)A08.

UCB/EERC-84/03 "Refined Modelling of Reinforced Concrete Columns for Seismic Analysis," by Kaba, S.A. and Mahin, S.A., April 1984, (PB84 234 384)A06.

UCB/EERC-84/04 "A New Floor Response Spectrum Method for Seismic Analysis of Multiply Supported Secondary Systems," by Asfura, A. and Der Kiureghian, A., June 1984, (PB84 239 417)A06.

UCB/EERC-84/05 "Earthquake Simulation Tests and Associated Studies of a 1/5th-scale Model of a 7-Story R/C Frame-Wall Test Structure," by Bertero, V.V., Aktan, A.E., Charney, F.A. and Sause, R., June 1984, (PB84 239 409)A09.

UCB/EERC-84/06 "R/C Structural Walls: Seismic Design for Shear," by Aktan, A.E. and Bertero, V.V., 1984.

UCB/EERC-84/07 "Behavior of Interior and Exterior Flat-Plate Connections subjected to Inelastic Load Reversals," by Zee, H.L. and Moehle, J.P., August 1984, (PB86 117 629/AS)A07.

UCB/EERC-84/08 "Experimental Study of the Seismic Behavior of a Two-Story Flat-Plate Structure. ," by Moehle, J.P. and Diebold, J.W., August 1984, (PB86 122 553/AS)A12.

UCB/EERC-84/09 "Phenomenological Modeling of Steel Braces under Cyclic Loading," by Ikeda, K., Mahin, S.A. and Dermitzakis, S.N., May 1984, (PB86 132 198/AS)A08.

UCB/EERC-84/10 "Earthquake Analysis and Response of Concrete Gravity Dams," by Fenves, G. and Chopra, A.K., August 1984, (PB85 193 902/AS)A11.

UCB/EERC-84/11 "EAGD-84: A Computer Program for Earthquake Analysis of Concrete Gravity Dams," by Fenves, G. and Chopra, A.K., August 1984, (PB85 193 613/AS)A05.

UCB/EERC-84/12 "A Refined Physical Theory Model for Predicting the Seismic Behavior of Braced Steel Frames," by Ikeda, K. and Mahin, S.A., July 1984, (PB85 191 450/AS)A09.

UCB/EERC-84/13 "Earthquake Engineering Research at Berkeley - 1984," by , August 1984, (PB85 197 341/AS)A10.

UCB/EERC-84/14 "Moduli and Damping Factors for Dynamic Analyses of Cohesionless Soils," by Seed, H.B., Wong, R.T., Idriss, I.M. and Tokimatsu, K., September 1984, (PB85 191 468/AS)A04.

UCB/EERC-84/15 "The Influence of SPT Procedures in Soil Liquefaction Resistance Evaluations," by Seed, H.B., Tokimatsu, K., Harder, L.F. and Chung, R.M., October 1984, (PB85 191 732/AS)A04.

UCB/EERC-84/16 "Simplified Procedures for the Evaluation of Settlements in Sands Due to Earthquake Shaking," by Tokimatsu, K. and Seed, H.B., October 1984, (PB85 197 887/AS)A03.

UCB/EERC-84/17 "Evaluation of Energy Absorption Characteristics of Bridges under Seismic Conditions," by Imbsen, R.A. and Penzien, J., November 1984.

UCB/EERC-84/18 "Structure-Foundation Interactions under Dynamic Loads," by Liu, W.D. and Penzien, J., November 1984, (PB87 124 889/AS)A11.

UCB/EERC-84/19 "Seismic Modelling of Deep Foundations," by Chen, C.-H. and Penzien, J., November 1984, (PB87 124 798/AS)A07.

UCB/EERC-84/20 "Dynamic Response Behavior of Quan Shui Dam," by Clough, R.W., Chang, K.-T., Chen, H.-Q., Stephen, R.M., Ghanaat, Y. and Qi, J.-H., November 1984, (PB86 115177/AS)A07.

UCB/EERC-85/01 "Simplified Methods of Analysis for Earthquake Resistant Design of Buildings," by Cruz, E.F. and Chopra, A.K., February 1985, (PB86 112299/AS)A12.

UCB/EERC-85/02 "Estimation of Seismic Wave Coherency and Rupture Velocity using the SMART 1 Strong-Motion Array Recordings," by Abrahamson, N.A., March 1985, (PB86 214 343)A07.

UCB/EERC-85/03 "Dynamic Properties of a Thirty Story Condominium Tower Building," by Stephen, R.M., Wilson, E.L. and Stander, N., April 1985, (PB86 118965/AS)A06.

UCB/EERC-85/04 "Development of Substructuring Techniques for On-Line Computer Controlled Seismic Performance Testing," by Dermitzakis, S. and Mahin, S., February 1985, (PB86 132941/AS)A08.

UCB/EERC-85/05 "A Simple Model for Reinforcing Bar Anchorages under Cyclic Excitations," by Filippou, F.C., March 1985, (PB86 112 919/AS)A05.

UCB/EERC-85/06 "Racking Behavior of Wood-framed Gypsum Panels under Dynamic Load," by Oliva, M.G., June 1985.

UCB/EERC-85/07 "Earthquake Analysis and Response of Concrete Arch Dams," by Fok, K.-L. and Chopra, A.K., June 1985, (PB86 139672/AS)A10.

UCB/EERC-85/08 "Effect of Inelastic Behavior on the Analysis and Design of Earthquake Resistant Structures," by Lin, J.P. and Mahin, S.A., June 1985, (PB86 135340/AS)A08.

UCB/EERC-85/09 "Earthquake Simulator Testing of a Base-Isolated Bridge Deck," by Kelly, J.M., Buckle, I.G. and Tsai, H.-C., January 1986, (PB87 124 152/AS)A06.

UCB/EERC-85/10 "Simplified Analysis for Earthquake Resistant Design of Concrete Gravity Dams," by Fenves, G. and Chopra, A.K., June 1986, (PB87 124 160/AS)A08.

UCB/EERC-85/11 "Dynamic Interaction Effects in Arch Dams," by Clough, R.W., Chang, K.-T., Chen, H.-Q. and Ghanaat, Y., October 1985, (PB86 135027/AS)A05.

UCB/EERC-85/12 "Dynamic Response of Long Valley Dam in the Mammoth Lake Earthquake Series of May 25-27, 1980," by Lai, S. and Seed, H.B., November 1985, (PB86 142304/AS)A05.

UCB/EERC-85/13 "A Methodology for Computer-Aided Design of Earthquake-Resistant Steel Structures," by Austin, M.A., Pister, K.S. and Mahin, S.A., December 1985, (PB86 159480/AS)A10 .

UCB/EERC-85/14 "Response of Tension-Leg Platforms to Vertical Seismic Excitations," by Liou, G.-S., Penzien, J. and Yeung, R.W., December 1985, (PB87 124 871/AS)A08.

UCB/EERC-85/15 "Cyclic Loading Tests of Masonry Single Piers: Volume 4 - Additional Tests with Height to Width Ratio of 1," by Sveinsson, B., McNiven, H.D. and Sucuoglu, H., December 1985.

UCB/EERC-85/16 "An Experimental Program for Studying the Dynamic Response of a Steel Frame with a Variety of Infill Partitions," by Yanev, B. and McNiven, H.D., December 1985.

UCB/EERC-86/01 "A Study of Seismically Resistant Eccentrically Braced Steel Frame Systems," by Kasai, K. and Popov, E.P., January 1986, (PB87 124 178/AS)A14.

UCB/EERC-86/02 "Design Problems in Soil Liquefaction," by Seed, H.B., February 1986, (PB87 124 186/AS)A03.

UCB/EERC-86/03 "Implications of Recent Earthquakes and Research on Earthquake-Resistant Design and Construction of Buildings," by Bertero, V.V., March 1986, (PB87 124 194/AS)A05.

UCB/EERC-86/04 "The Use of Load Dependent Vectors for Dynamic and Earthquake Analyses," by Leger, P., Wilson, E.L. and Clough, R.W., March 1986, (PB87 124 202/AS)A12.

UCB/EERC-86/05 "Two Beam-To-Column Web Connections," by Tsai, K.-C. and Popov, E.P., April 1986 , (PB87 124 301/AS)A04.

UCB/EERC-86/06 "Determination of Penetration Resistance for Coarse-Grained Soils using the Becker Hammer Drill," by Harder, L.F. and Seed, H.B., May 1986, (PB87 124 210/AS)A07.

UCB/EERC-86/07 "A Mathematical Model for Predicting the Nonlinear Response of Unreinforced Masonry Walls to In-Plane Earthquake Excitations," by Mengi, Y. and McNiven, H.D., May 1986, (PB87 124 780/AS)A06.

UCB/EERC-86/08 "The 19 September 1985 Mexico Earthquake: Building Behavior," by Bertero, V.V., July 1986.

UCB/EERC-86/09 "EACD-3D: A Computer Program for Three-Dimensional Earthquake Analysis of Concrete Dams," by Fok, K.-L., Hall, J.F. and Chopra, A.K., July 1986, (PB87 124 228/AS)A08.

UCB/EERC-86/10 "Earthquake Simulation Tests and Associated Studies of a 0.3-Scale Model of a Six-Story Concentrically Braced Steel Structure," by Uang, C.-M. and Bertero, V.V., December 1986.

UCB/EERC-86/11 "Mechanical Characteristics of Base Isolation Bearings for a Bridge Deck Model Test," by Kelly, J.M., Buckle, I.G. and Koh, C.-G., 1987.

UCB/EERC-86/12 "Modelling of Dynamic Response of Elastomeric Isolation Bearings," by Koh, C.-G. and Kelly, J.M., 1987.

UCB/EERC-87/01 "FPS Earthquake Resisting System: Experimental Report," by Zayas, V.A., Low, S.S. and Mahin, S.A., June 1987.

UCB/EERC-87/02 "Earthquake Simulator Tests and Associated Studies of a 0.3-Scale Model of a Six-Story Eccentrically Braced Steel Structure," by Whittaker, A., Uang, C.-M. and Bertero, V.V., July 1987.

UCB/EERC-87/03 "A Displacement Control and Uplift Restraint Device for Base-Isolated Structures," by Kelly, J.M., Griffith, M.C. and Aiken, I.G., April 1987.

UCB/EERC-87/04	"Earthquake Simulator Testing of a Combined Sliding Bearing and Rubber Bearing Isolation System," by Kelly, J.M. and Chalhoub, M.S., 1987.
UCB/EERC-87/05	"Three-Dimensional Inelastic Analysis of Reinforced Concrete Frame-Wall Structures," by Moazzami, S. and Bertero, V.V., May 1987.
UCB/EERC-87/06	"Experiments on Eccentrically Braced Frames With Composite Floors," by Ricles, J. and Popov, E., June 1987.
UCB/EERC-87/07	"Dynamic Analysis of Seismically Resistant Eccentrically Braced Frames," by Ricles, J. and Popov, E., June 1987.
UCB/EERC-87/08	"Undrained Cyclic Triaxial Testing of Gravels - The Effect of Membrane Compliance, " by Evans, M.D. and Seed, H.B., July 1987.
UCB/EERC-87/09	"Hybrid Solution Techniques For Generalized Pseudo-Dynamic Testing," by Thewalt, C. and Mahin, S. A., July 1987.
UCB/EERC-87/10	"Investigation of Ultimate Behavior of AISC Group 4 and 5 Heavy Steel Rolled-Section Splices with Full and Partial Penetration Butt Welds," by Bruneau, M. and Mahin, S.A., July 1987.
UCB/EERC-87/11	"Residual Strength of Sand From Dam Failures in the Chilean Earthquake of March 3, 1985," by De Alba, P., Seed, H. B., Retamal, E. and Seed, R. B., September 1987.
UCB/EERC-87/12	"Inelastic Response of Structures With Mass And/Or Stiffness Eccentricities In Plan Subjected to Earthquake Excitation," by Bruneau, M., September 1987.
UCB/EERC-87/13	"CSTRUCT: An Interactive Computer Environment For the Design and Analysis of Earthquake Resistant Steel Structures," by Austin, M.A., Mahin, S.A. and Pister, K.S., September 1987.