ABSTRACT

Title of Dissertation: MODERN COMPUTATIONAL ENVIRONMENTS FOR SEISMIC ANALYSIS OF HIGHWAY BRIDGE STRUCTURES

Wane-Jang Lin, Doctor of Philosophy, 1998

Dissertation directed by: Associate Professor Mark A. Austin
Department of Civil Engineering

This dissertation is about the architecture, design, and implementation of ALADDIN, a new high-level scripting language and toolkit for the interactive matrix and finite element analysis of engineering structures. In ALADDIN finite element computations are viewed as a specialized form of matrix computations, matrices are viewed as rectangular arrays of physical quantities, and numbers viewed as dimensionless physical quantities. The ALADDIN programming language is C-like in the sense that it uses only a small number of keywords, supports a variety of familiar looping and branching constructs, and links to libraries of matrix and finite element functions. It has been carefully designed so that the input files are human readable and consist of a friendly, intuitive syntax rather than a table of numbers. Engineers are provided with the flexibility to modify problem parameters and problem solving algorithms without having to recompile source codes.

The capabilities of ALADDIN are demonstrated by solving linear static and dynamic analyses for simple structural systems and highway bridge structures, the principal application area for this research. With appropriate structural response quantities in hand (e.g., distributions of stress, displacement, and hysteretic energy dissipation), ALADDIN can evaluate selected design rules from the AASHTO design code. This dissertation presents the formulation of a special flexibility-based fiber beam-column element that includes both flexural and shear effects. Shear effects are an important displacement component in base-isolated highway bridge structures. By using the fiber elements to model the lead-rubber isolators, we are able to study the behavior of base isolated highway bridge structures. The bilinear force-displacement hysteresis loops for lead-rubber isolators are obtained by ALADDIN and compared with the experimental data to show the accuracy of modeling. The essential benefits of base isolation are illustrated by computing the dynamic mode shapes for isolated and non-isolated versions of an example highway bridge. Finally, an energy-balance calculation is performed for a full nonlinear time-history analysis of a three-dimensional base-isolated bridge. The important contribution of this work is that now all of the energy balance computations can be specified from the ALADDIN input file (and not hard coded into the program source code), thereby enabling energy-based calculations to be directly linked with checking of design rule/code regulations.

MODERN COMPUTATIONAL ENVIRONMENTS FOR SEISMIC ANALYSIS OF HIGHWAY BRIDGE STRUCTURES

by

Wane-Jang Lin

Dissertation submitted to the Faculty of the Graduate School
of the University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
1998

Advisory Committee:

Associate Professor Mark A. Austin, Chairman/Advisor
Professor Pedro A. Albrecht
Professor David R. Schelling
Professor Amde M. Amde
Assistant Professor Raymond A. Adomaitis

ACKNOWLEDGEMENTS

My special thanks go to my advisor, Mark Austin, for his guidance and advice in my academic years in University of Maryland. Special thanks go also to John O'Fallon of Federal Highway Administration for his encouragement and support. Thanks are also due to my committee members, Professors Pedro Albrecht, David Schelling, Amde Amde, and Raymond Adomaitis, for their constructive comments and criticisms. The financial support for this dissertation is provided by the fellowship grants from Federal Highway Administration, I am grateful for that. In addition, I thank the co-writer of our program, Xiaoguang Chen, for his contribution in developing the first version of the program. Finally, I would like to thank my parents, family and friends for their support, especially my husband, Min-Ching Chen, who is always there providing understanding, support, and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

MODERN COMPUTATIONAL ENVIRONMENTS
FOR SEISMIC ANALYSIS OF HIGHWAY
BRIDGE STRUCTURES

Wane-Jang Lin

January 23, 1998

This comment page is not part of the dissertation.

CHAPTER 1

Introduction

1.1 Computational Technology for Structural Engineering

The primary function of any building or highway bridge structure is support and transfer of externally applied loads to the reaction points in a safe and reliable manner. In present-day structural design new structures are often designed initially on the basis of experience with similar types of structures, perhaps using some simple analytical calculation procedures. Subsequent versions of the design are progressively detailed, and are analyzed using numerical methods having a sophistication consistent with the fidelity of required performance assessment. The structural design process continues in an iterative manner until a satisfactory cost-effective solution is obtained [11].

Structural analysis procedures are concerned with the quantitative assessment of structural performance under prescribed loading and displacement conditions. External design loadings and displacement conditions can be static or dynamic – we say that a load is dynamic when its time-varying characteristics have a significant affect on the structural response. The dynamic load of greatest importance in this study will be that produced by moderate and severe earthquake ground motions. Commonly assessed response quantities include distributions of stress, displacement, structural strength and ductility demand ratios. Distributions of stress and displacement give the designer a good sense of how the structure will behave under the service load conditions. Distributions of strength and ductility demand ratios give the designer a good sense of how individual structural components and the overall structural system will behave under extreme loading conditions. For each of these loading cases, the computed response quantities provide a basis for assessing how well the likely structural actions can be supported by the structure [11, 52].

During the early 1960s, soon after high-level programming languages were introduced, engineers envisioned the need for computational problem-solving environments that would be powerful enough to solve a target class of applications problems and interact with human users [18]. When this vision is interpreted in the context of design and analysis of building and highway bridge structures, computational support is needed for:

1. The construction of mathematical models of the phenomenon under study.

2. The selection of relevant physics (e.g., constitutive models) and structural geometry.

3. The manipulation of equations and associated conditions, thereby allowing suitable simplified solution methods.

4. The automated construction of test problems and data sets.

5. The specification of appropriate programming languages and problem solving methods (perhaps from scratch or by modifying existing materials).

6. The application of the program to the test data and validation of results.

7. The assessment of structural performance, including the collection and manipulation of data generated by the structural analyses, and comparison of this information with design code regulations.

8. The communication of results to the scientific community.

Where appropriate, models should account for natural variations in material properties, uncertainties in loading conditions, and inaccuracies in simplified modeling techniques.

Of course new methods are more likely to be accepted if they complement and extend time-tested traditional procedures. In this respect we note that in traditional approaches to problem solving, engineers write the details of a problem and its solution procedure on paper. They use physical units to add clarity to the problem description, and may specify step-by-step details for a numerical solution to the problem. Matrices and linear matrix algebra are an essential part present-day structural analysis because they enable problems and their solution procedures to be specified at a relatively high level of abstraction, and because linear matrix operations are ideally suited for automatic computation on computers. Similarly, the finite element method is an integral part of modern structural analysis. This method is based on the concept that a complex system can be modeled by an assembly discrete "elements" whose behavior is readily known. A complete solution is obtained by combining the element-level displacement or stress distributions in a manner that satisfies the force-equilibrium and displacement compatibility conditions for each "node" or connecting point of the elements [52]. By harnessing computer power for step-by-step solutions to sophisticated matrix and finite element problems, engineers can approach these calculations without recourse to simplifying assumptions. Units, physical quantities, matrices, and finite element analysis are techniques fundamental to good engineering practice, and should be incorporated into new computational environments.

Even though opportunity for these advances was identified in the 1960s, the primitive state of computing at that time (measured by today's standards) made the implementation of a suitable environment formidable. During the past twenty years, however, remarkable advances in computer hardware and software have enabled the development of engineering software tools to mature to the point where importance is placed on ease of use, and a wide-array of practical services being made available (i.e., the best computer programs can solve a wide variety of problems) to the engineering profession as a whole. Computer programs written for engineering computations are expected to be fast and accurate, flexible, reliable, checkable, and of course, easy to use [21]. Whereas an engineer in the 1970s might have been satisfied by a computer program that provided numerical solutions to a very specific engineering problem, the same engineer today might require the engineering analysis, plus computational support for design code checking, optimization, interactive computer graphics, network connectivity, and so forth. Many of the latter features are not a bottleneck for getting the job done. Rather, features such as interactive computer graphics simply make the job of describing a problem and interpreting results easier. The pathway from ease-of-use to productivity gains is well defined [6].

1.2 Modeling and Analysis of Highway Bridge Structures

During the past three decades most of the advances in computer-aided modeling and analysis of building and highway bridge structures can be traced back in part to advances in digital computing [11, 49, 36]. A model is a tool that facilitates the mathematical formulation of the geometry and behavior characteristics of a prototype system. Structural modeling techniques typically assume that the "behavior of the real-world structure" can be captured by suitable assemblies of discrete mathematical elements. Various levels of discretization are possible, ranging from very simple spring-mass systems through to three-dimensional finite element models that capture nonlinear geometric and material behavior.

To facilitate the design of complex seismic resistant structural systems recent design codes prescribe a series of standard performance levels for seismically resistant structures. A performance level is a limiting state of acceptable damage for the main structural system, the structure's contents, non-structural components and utilities. Performance levels are selected based on the safety, economic and societal impacts of damage. The goal of performance-based engineering is to define performance objectives for structures of various uses, and to control the risk associated with each limit-state to a pre-defined level of acceptability [45].

Analytical procedures are needed for the assessment of structural behavior in a manner consistent with expected behavior. Under moderate earthquake loadings, for example, a structure should suffer no structural damage. Limited non-structural damage is permitted, however. Since a structure is expected to remain essentially elastic, a linear time-history analysis or modal analysis is appropriate. Severe ground shaking is assumed to correspond to the maximum credible ground motion for the site. The important performance criterion in events of this type is assuring that loss of life does not occur. Extensive structural damage in the form of large plastic deformations is acceptable. While the level of damage may be beyond repair, collapse is nonetheless prohibited. Analytical procedures should capture the nonlinear behavior of systems under these loading

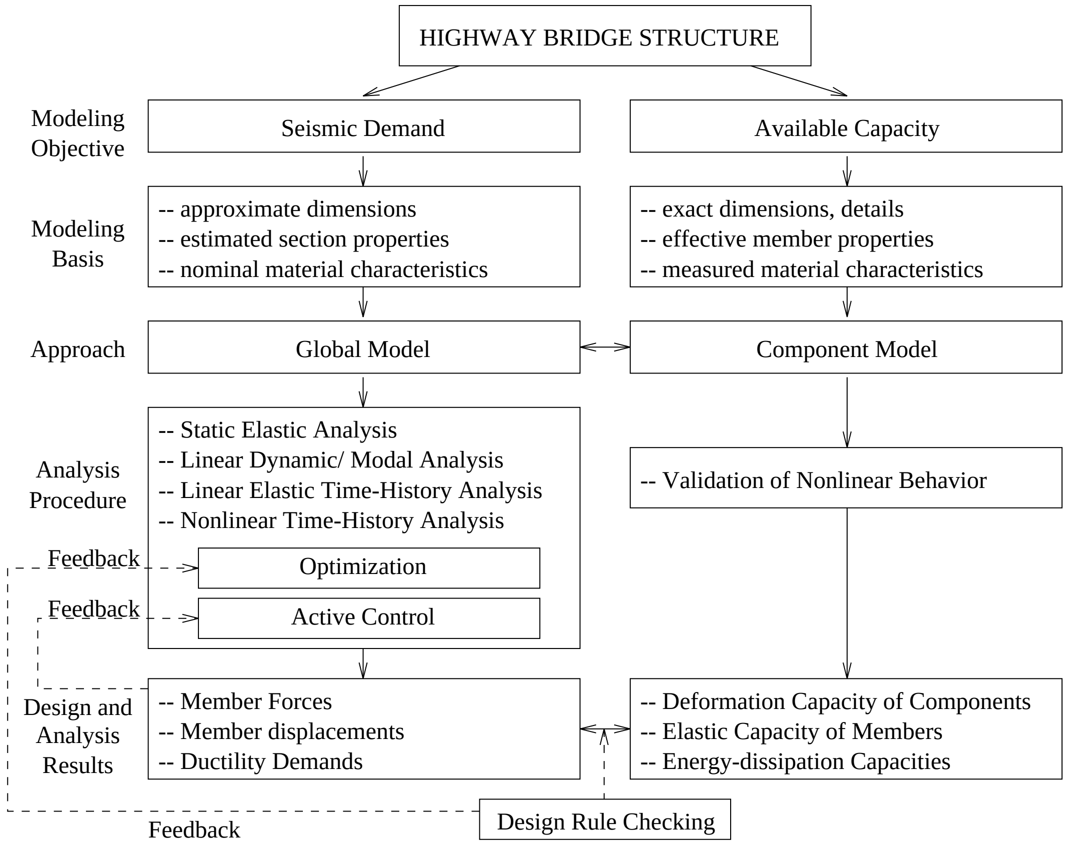


Figure 1.1: Modeling, Analysis, and Design of Highway Bridge Structures

conditions.

Figure 1.1 is adapted from the text of Priestley et al. [35], and summarizes the objectives, approaches and analytical procedures commonly in use for the modeling and analysis of seismically resistant highway bridge structures. A prerequisite to the formulation of appropriate models and analysis techniques is an understanding of structural dynamics. The latter includes equations of motion (i.e., governing differential equations), bridge dynamic response characteristics (i.e., eigenvalue and eigenvector calculations) and numerical solution procedures for the governing differential equations [12, 14]. A good analysis tool will provide information on: (1) the overall seismic bridge design process, (2) the dynamic response of bridge structures under earthquake loads, (3) the consequences of inaccuracies in modeling assumptions, and (4) the available modeling and analysis techniques [35].

1.3 Literature Review

Computer-aided structural analysis tools are now an accepted part of standard structural engineering practice because they help engineers unravel the basic phenomena of real world structural behavior. They contain many of the characteristics of general purpose problem-solving environments, as well as application-specific techniques that cannot easily be used in other applications (e.g., Mathematica and MATLAB [48]) [18]. Early versions of structural analysis and finite element computer programs such as ABAQUS [3], ANSR [30], and FEAP [53] were written in the FORTRAN computer language, and were developed with the goal of optimizing numerical and/or instructional considerations alone. These programs offered a restricted, but well implemented, set of numerical procedures for static structural analyses, and linear/nonlinear time-history response calculations. Table 1.1 contains a summary of software packages for the seismic analysis of highway bridge structures. Most of them can be found on the National Information Service for Earthquake Engineering (NISEE) web server [15].

Although present-day software packages have increased their capacity to solve engineering problems, and handle input/output in a graphical manner, some problems still remain. For example, physical units are not part of the program infrastructure. Most engineering analysis packages simply assume that an engineer will check that units are applied in a consistent manner. The few computer programs that do incorporate physical units handle them at either the program

Software	Developer	Description
ANSR	D.P. Mondkar and G.H. Powell [30]	ANSR is a general purpose computer program for static and dynamic analysis of nonlinear structures.
MicroSARB	D. Orie, M. Saiidi and B. Douglas [33]	This program implements procedure 1 of the Applied Technology Council (ATC-6) seismic design guidelines for straight regular highway bridges. The ATC-6 procedure 1 employs the Single Mode Spectral Analysis Method (SMSM) for seismic analysis of "regular" highway bridges.
SEISAB	Imbsen & Associates, Inc.	SEISAB is a computer program developed specifically for use in designing new structures or for evaluating existing structures to determine retrofit requirements. The program's capabilities include the analysis procedures specified in the current AASHTO Standard Specifications for Highway Bridges and the AASHTO Standard Specifications for Seismic Design of Highway Bridges.
NEABS	J. Penzien, R. Imbsen, and W.D. Liu [34]	NEABS performs nonlinear dynamic analysis of long, multiple span bridge systems.
ISADAB	M. Saiidi, R. Lawver, and J. Hart [38]	ISADAB was developed for the transverse inelastic analysis of reinforced concrete highway bridges.
CALANSR	Pmb Engineering, Inc.	CALANSR is a general-purpose structural analysis program specifically developed for fully-coupled, nonlinear seismic analysis. The program can assess ultimate system capacity through "pushover" analyses using specified loading or deformation patterns.
ASPIDEA	R. Giannini, G. Monti, G. Nuti and T. Pagnoni [20]	ASPIDEA (A Program for Nonlinear Analysis of Isolated Bridges Under Non-synchronous Seismic Action) is a nonlinear dynamic program specifically developed for the analysis of isolated bridges, using a number of generated accelerograms.

Table 1.1: Seismic Analysis Softwares of Highway Bridges

input and output stage (i.e., the input and output will be presented and displayed in a certain set of units) or at the physical quantity level (e.g., x = 2 in). In practical engineering analysis, however, units can be as important as the numerical quantity itself. We have not been able to find any computer programs that systematically integrate units into the definition of physical quantities, matrices, and finite element analysis.

Readability of input files is another problem for many matrix and finite element software packages. Frequently the format specifications for input files in engineering packages are so cryptic that it is impossible to understand the purpose of the engineering problem – the latter requires a careful reading of the program user manual.

While these limitations may have been acceptable twenty years ago, the advent of modern software engineering tools means that today we can do a much better job! For example, many of these problems can be mitigated by moving the balance of software development from "compiled programs" to programs that are both "compiled" and "interpreted." Ease-of-use and increased flexibility in problem solving and program portability are the main reasons for moving towards interpreted languages – MATLAB and Java are two good examples of languages that have these features. While programming languages like MATLAB, MathCad, and Mathematica support matrix operations, physical units are not incorporated into all computations, and the extension of these environments to finite element computations is non trivial.

Economics and Difficulty of Software Development : The difficulty in following-up on the above mentioned hardware advances with appropriate software developments is clearly reflected in the economic costs of project develop-

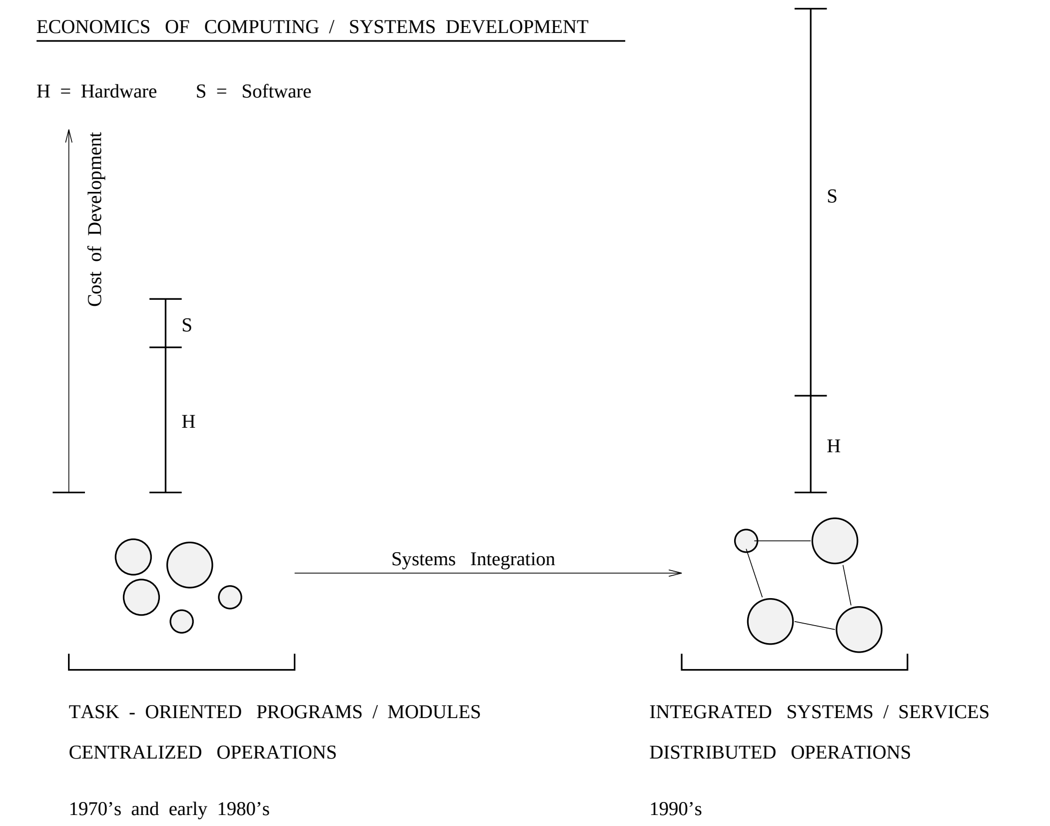


Figure 1.2: Economics of Software Development and Integration

ment. In the early 1970s software consumed approximately 25% of total costs, and hardware 75% of total costs for development of data intensive systems. Nowadays, development and maintenance of software typically consumes more than 80% of the total project costs. This change in economics is the combined result of falling hardware costs, and increased software development budgets needed to implement systems that are much more complex than they used to be. See Figure 1.2.

Whereas one or two programmers might have written a complete program twenty years ago, today's problems are so complex that teams of programmers are needed to understand a problem and fill-in the details of required development.

When a computer program has a poorly designed architecture, its integration with another package can be very difficult, with the result often falling short of users' expectations. Let us suppose, for example, that we wanted to interface the finite element package FEAP [53] with the interactive optimization-based design environment called DELIGHT [9, 32, 51]. Since FEAP was not written with interfaces to external environments as a design criterion, a programmer(s) faced with this task would first need to figure out how FEAP and DELIGHT work (not an easy task), and then devise a mapping from DELIGHT's external interface routines to FEAP's subroutines. The programmer(s) would need the computer skills and tenacity to stick-with the lengthy period of code development that would ensue. And what about the result? In our experience [7, 8, 9], the integrated DELIGHT-FEAP tool would most likely do a very good job of solving a narrow range of problems. Simply extending the DELIGHT-FEAP environment to account for a new set of design rules might require an indepth knowledge of the program architecture and computer programming. Because of these difficulties, the DELIGHT-FEAP program would most likely have a short life cycle. These barriers to software integration are frustrating because finite element and optimization procedures are essentially specialized matrix computations – the disciplines should fit together in almost a seamless way. is an ad-hoc approach to software tool development in the first place.

1.4 Research Objectives and Design Criteria for ALADDIN

This research project takes the position that the main barrier to software integration. Rather than simply repeat the above-mentioned "scenario procedure" for yet another set of packages, this research project attempts to understand the structure matrix and finite element (and optimization and control) packages should take so that they can be integrated in a natural way. The problem will be investigated by designing and implementing a system specification for how a matrix and finite element system ought to work. To verify that these ideas will in fact work we will use the system to study the seismic analysis and performance assessment of highway bridge structures.

Our research direction is inspired in part by the systems integration methods developed for the European ESPRIT Project, and by the success of C. Although the C programming language has only 32 keywords, and a handful of control structures, its judicious combination with external libraries has resulted in the language being used in a very wide range of applications. The system specification will include: [6, 25]:

1. **A Model :** The model will include data structures for the information to be stored, and a stack machine for the execution of the matrix and finite element algorithms.

2. **A Language :** The language acts as an interface between the engineer and the underlying computational model. It is a means for describing matrices, finite element meshes and numerical solution procedures.

3. **Defined Steps and Ordering of the Steps :** The steps will define the transformations that can be applied to the system components (e.g., nearly all engineering processes will require iteration and branching).

4. **Guidance for Applying the Specification :** Guidance includes factors such as easy-to-understand problem description files, documentation, and

example problems. The last two components have been implemented on the ALADDIN web site.

Using grammatical rules and compiler construction tools based on the work of linguist Niam Chomsky, design and analysis concepts are translated into mathematical and computational models. Working out the details of language translation requires a combination of design and artistic skills – we would like the new environment's language to be textually descriptive and strike a balance between simplicity and extensibility. It should use a small number of data types and control structures, incorporate physical units, and yet, be descriptive enough so that the *pencil and paper* and *problem description* files are almost the same. A key advantage in designing a software environment around a well defined language is that underlying software components must be modular for the system to work.

The ALADDIN language has a grammar, a syntax, and semantics meaningful to the structural analysis and design problem solving domain. ALADDIN symbols can represent physical quantities, matrices of physical quantities, components of finite element meshes, and components of numerical solution procedures. The fundamental branching constructs are `if` and `if..then..else`, and the looping control constructs are `while()` and `for()`.

Blocks of design and analysis statements are parsed and converted into low-level stack machine instructions. As we will soon see this step is far from trivial – the linguistic constructs and generation tools must be powerful enough to handle both the geometrical and functional aspects of structural analysis/design, and yet, precise so that high-level ALADDIN statements can be efficiently mapped to stack machine instructions without ambiguity.

1.5 Classes and Needs of ALADDIN Users

ALADDIN is designed for three groups of people:

1. **Graduate Students in Structural Engineering :** A graduate student might use ALADDIN to solve a structural analysis or finite element problem as part of his or her classwork. Some graduate students will use ALADDIN as a framework for implementing new finite elements, perhaps as part of an advanced class in finite element analysis.

2. **Researchers in Structural Analysis :** ALADDIN provides researchers with an environment to test out their new algorithms and finite elements, and to freely collect the results (all without writing a whole new program or modifying and compiling the source codes). When researchers are developing a new numerical algorithm or solution procedure, one problem they face is validation. As part of the validation procedure a researcher may need to collect, for example, the time variation in plastic energy dissipation computed during an earthquake time-history analysis. Existing analysis programs provide neither this information nor flexibility. Yet with ALADDIN, the required information can be collected and processed by simply writing the appropriate statements in the input file.

3. **Engineering Practitioners :** Practicing engineers can use ALADDIN simply as a units-based calculator, or for the solution of small-scaled analysis problems involving matrix arithmetics, linear matrix equations, design rule checking, and so forth.

We expect that the people in each of these categories will already have a good understanding of engineering analysis procedures. While they may also be familiar with programming logic and flow charts, few of them will have the skills of professional software developers. That is fine! They are, after all, primarily interested in using the software as a problem-solving tool.

From a software point of view ALADDIN users want a computational environment that:

1. **Is Easy to Learn :** A good way of making a new programming language easy to learn is to deliberately make its syntax close to that of languages engineers will already be familiar. Most engineers are familiar with one or more high-level programming languages (e.g., FORTRAN, BASIC, C, PASCAL). High level scripting languages such as MATLAB are also coming into vogue in many areas of engineering. It therefore makes good sense to design the ALADDIN language so that its syntax will look like one or more of the languages engineers are already familiar.

2. **Supports Documentation :** Engineers should be provided with the mechanisms to document and explain problem descriptions in a flexible manner. This strategy of development improves communication among members of a design team.

3. **Incorporates Physical Units :** Engineers should be provided with the freedom to use a variety of units in their engineering computations. In this respect, the ALADDIN environment should verify the units are consistent before proceeding with an arithmetic or matrix operation.

4. **Supports Matrix Operations :** Since matrix operations are the basic operations for all structural engineering problems, their operation should

work automatically in a program. For example, if A and B are previously defined matrices, matrix addition should proceed by simply writing $A + B$.

5. **Supports Finite Element Analysis :** As already stated, finite element analysis is the main analysis method used in structural analysis. It should include basic elements for framed structures, and some special elements for special modeling. The finite element program should be capable of dealing with both static and dynamic analysis, in a linear or nonlinear regime.

6. **Supports Custom Problem-Solving Procedures :** Engineers should be provided with the tools to write custom problem-solving procedures in the ALADDIN language alone. A knowledge of ALADDIN's inner working should not be required.

And from an engineering point of view ALADDIN should provide its users with support for:

1. **Modeling of bridge systems at multiple levels of fidelity :** For example, a engineer may need to model the nonlinear hysteretic properties of a base isolation component using a detailed nonlinear finite element analysis. The same engineer may also need to model an entire highway bridge system – in this second case, some components of the bridge system will be assumed to remain linear elastic, with nonlinear behavior being confined to specific elements in the bridge. This problem solving strategy provides reasonably realistic estimates of nonlinear system behavior without having to work with matrix equations that are unnecessarily large.

2. **Different types of prescribed load :** Including static dead and live loads, moving live loads, earthquake ground motion accelerograms and re-

sponse spectra analysis.

3. **Numerical algorithms :** And they are reliable for structural analyses (i.e., they won't fail to converge midway through a lengthy nonlinear analysis.)

4. **Validation of the structural analysis results :** This can take the form of design code checking, or perhaps energy evaluation in a static/dynamic nonlinear history analysis.

5. **Different numerical solution procedures for engineering analysis :** Such as Newmark method, Wilson-θ method, Mode-Superposition method, Newton-Raphson method, etc..

All of these feature should work within a single integrated computational environment.

1.6 Dissertation Outline

Chapter 2 contains a detailed description of the architecture and design for ALADDIN, with examples explaining what the input looks like and how the implementation works. Chapter 3 shows how static linear finite element problems can be solved with ALADDIN – it also walks through a number of examples that incorporate structural analysis solution procedures. Chapter 4 introduces a special flexibility based fiber beam-column element originally proposed by Filippou et al. [46]. Its implementation in ALADDIN is also described. Chapter 5 shows solution procedures for dynamic nonlinear finite element analysis with ALADDIN, and introduces the study of strain energy calculations. Chapter 6 contains an analytical study for the base isolation of bridges, and covers the lead-rubber isolator and modeling and analysis of isolated bridges. Finally, Chapter 7 contains the conclusions of this work and suggestions for future research.

CHAPTER 2

Architecture and Design of ALADDIN

2.1 Introduction to ALADDIN

ALADDIN is a computational toolkit for the interactive matrix and finite element analysis of large engineering structures. In ALADDIN finite element computations are viewed as a specialized form of matrix computations, matrices are viewed as rectangular arrays of physical quantities, and numbers viewed as dimensionless physical quantities. ALADDIN provides engineers with:

1. Mechanisms to define physical quantities with units, and matrices of physical quantities.

2. Facilities for physical quantity and matrix arithmetic.

3. A SI and US units package. Conversion of units may be applied to physical quantity constants, physical quantity variables, and matrices of physical quantities.

4. A matrix package. Its capabilities include matrix arithmetic, solution of linear matrix equations, and the general symmetric eigenvalue problem.

5. Programming constructs to control the solution procedure (i.e., branching and looping) in matrix and finite element problems.

6. A finite element mesh generation package. Two- and three-dimensional finite element meshes can be created.

7. A library of finite elements. Currently, the finite element library includes elements for plane stress/plane strain analysis, two-dimensional and three-dimensional beam/column analysis, three-dimensional truss analysis, DKQ plate analysis, and a variety of shell finite elements.

The target application area for ALADDIN is static and dynamic finite element analysis of multi-story buildings and highway bridge structures. The seismic analysis of structures due to earthquake loads is of particular interest.

2.2 Architecture of ALADDIN Environment

Figure 2.1 is a schematic of the ALADDIN architecture, and shows its three main parts: (1) the input file(s), (2) the kernel, and (3) libraries of matrix and finite element functions.

The main challenge in designing the kernel and input file is in finding a language syntax and semantics that will enable a number of disciplines involved in a problem solving procedure to be integrated in a seamless manner. To be useful to engineers, the problem description language should be reasonably high level and yet precise, perhaps resulting in hundreds/thousands of low-level native machine instructions per ALADDIN statement. The second part of the program architecture design is concerned with the system components – on this side of the problem, the main challenge lies in the design of data structures

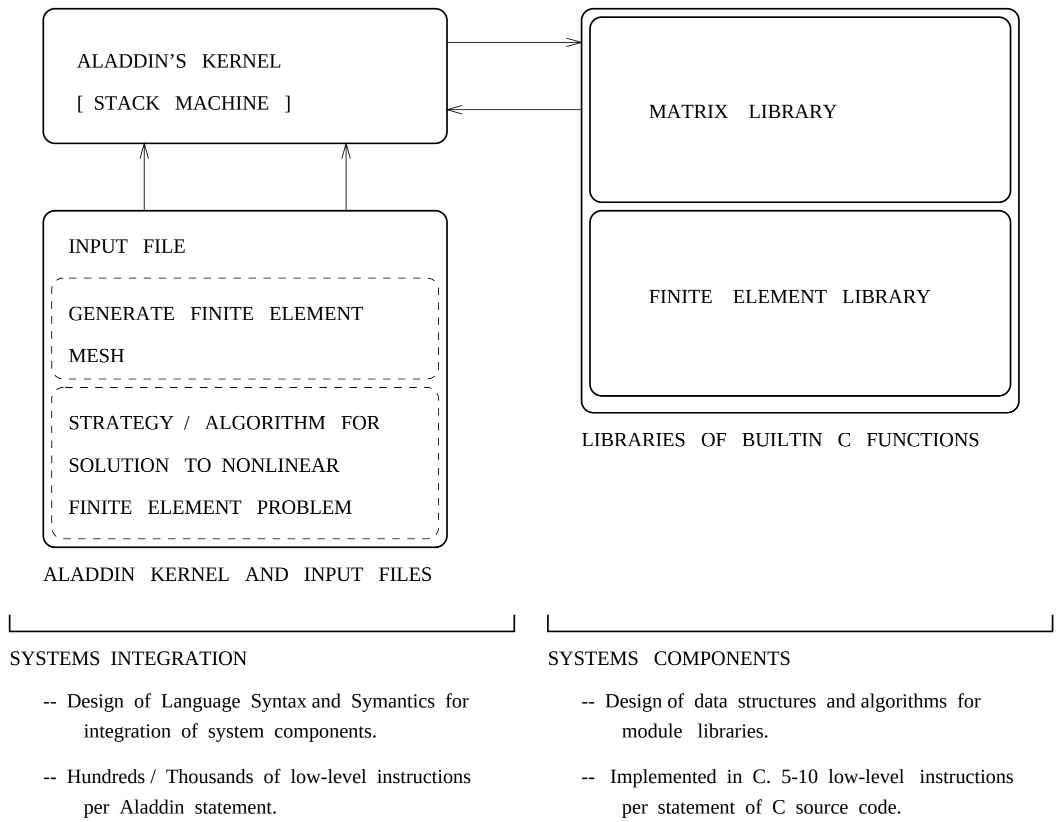


Figure 2.1: High Level Architecture for ALADDIN

and algorithms for the underlying matrix and finite element operations that are computationally efficient. As already pointed out in Chapter 1, a successful implementation requires a seamless integration of library components with the kernel and language designs. This necessitates a component-language-test software development cycle. New component modules associated language features must be throughly tested before they are included in an ALADDIN release (verifying that new program components work properly can be an extremely time consuming process).

Figure 2.2 is a second high-level view of the ALADDIN system that emphasizes the relationship between an engineer/user and the ALADDIN language, components of the ALADDIN kernel, and their communication with functions

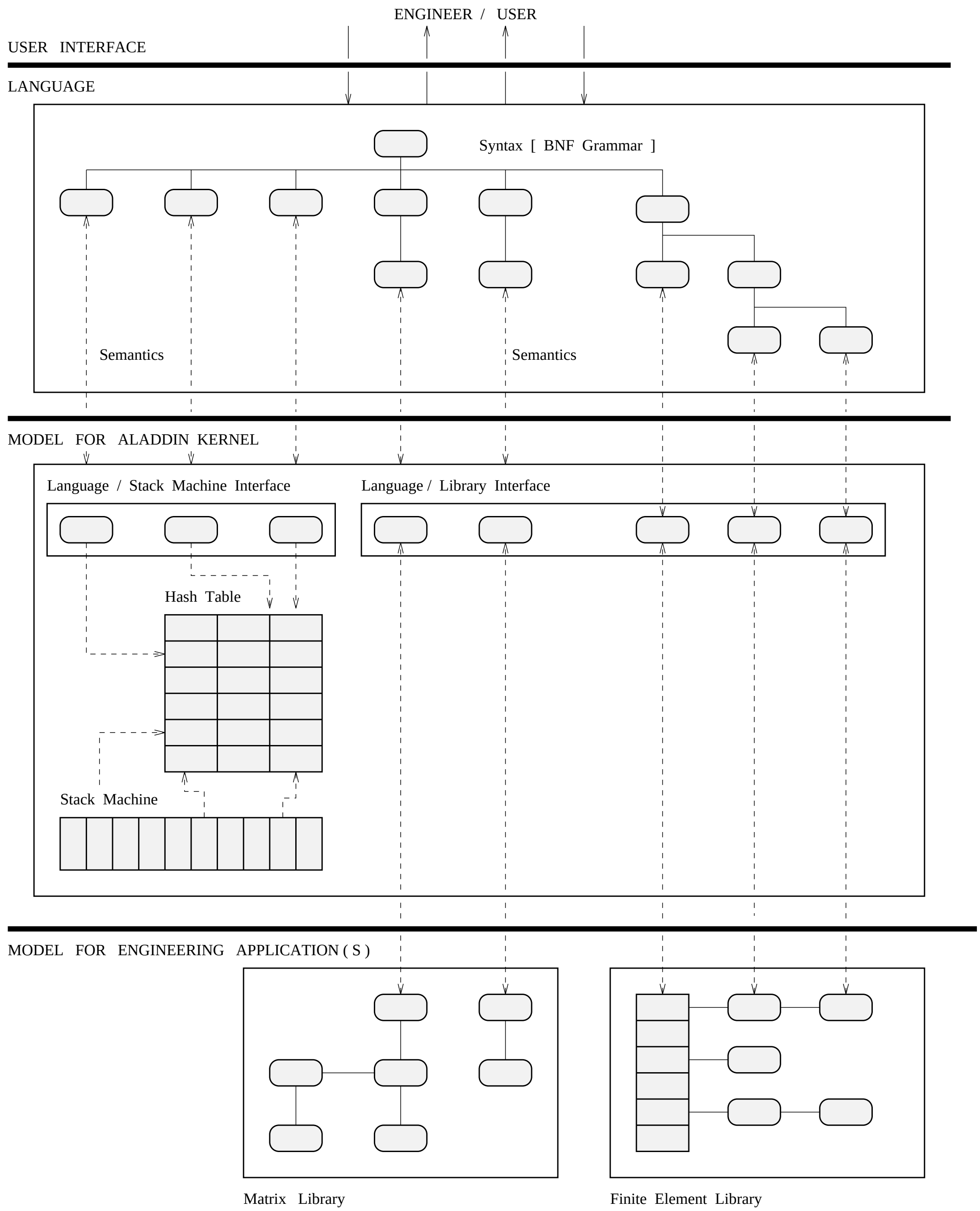


Figure 2.2: Interaction of Language and Underlying Model

in the matrix and finite element libraries. In a typical problem solving session, an engineer/user will write blocks of ALADDIN language statements in an input file. The ALADDIN kernel parses the description statements, identifying the names and types of input tokens and then builds up the parser tree for further stack operations.

To ensure that the tokens are available for further processing, the language parser stores them in a hash (a symbol) table. Stack machines are suitable for modeling systems that have a finite number of internal configurations or states, and whose behavior evolves as a linear sequence of discrete points in time. It also constructs an ensemble of low-level stack machine operations (we will illustrate these details in a moment). The stack machine then calls and executes functions in the stack instructions. The functions include matrix and finite element libraries in lowest level of program. Results are then passed back to user after the function calls are complete.

2.2.1 Problem Description Statements in Input File

Specific engineering problems are defined in ALADDIN problem description files, and solved using components of ALADDIN that are part interpreter-based and part compiled C code. It is important to keep in mind that as the speed of CPU processors increases, the time needed to prepare a problem description increases relative to the total time needed to work through an engineering analysis. Hence, clarity of an input file's contents is of paramount importance. In the design of the ALADDIN language we attempt to achieve these goals with:

1. Liberal use of comment statements (as with the C programming language, comments are inserted between `/* .... */`),

2. Consistent use of function names and function arguments,

3. Use of physical units in the problem description, and

4. Consistent use of variables, matrices, and structures to control the flow of program logic.

ALADDIN problem description statements and their solution algorithms are a composition of three elements: (1) data, (2) control, and (3) functions [39].

1. **Data :** ALADDIN supports three data types, "character string" for variable names, physical quantities, and matrices of physical quantities for engineering data. For example, the statement

   ```
   x = 3 cm/sec;
   ```

   defines a variable called "`x`" to represent a velocity of 3 cm per second. Notice how `3` juxtaposed with `cm/sec` implies multiplication; we have hard-coded this interpretation into the ALADDIN language because `3 cm/sec` is more customary and easier to read than `3 * cm/sec`. There are no integer data types in ALADDIN. Floating point numbers are stored with double precision accuracy, and are viewed as physical quantities without units. Matrices of physical quantities are a direct extension of this feature. For example, the statement

   ```
   y = [0 m, 1 m, 2 rad];
   ```

   defines a (1×3) matrix of displacements. Matrix elements `y[1][1]` and `y[1][2]` have units of length, and `y[1][3]` has a planer angle rotation.

   Of course physical quantities can be used in arithmetic calculations, logical and relational operations. For example, let

```
x = 3 cm;
y = 5 cm;
z = 7 cm;
```

Table 2.1 serves two purposes. First, it summarizes the logical and relational operators that can be applied to physical quantities. It also demonstrates the application of those operators and their computed results.

Operator	Description	Expression	Result
`<`	less than	`x < y`	true
`>`	greater than	`x > y`	false
`<=`	less than or equal to	`x <= y`	true
`>=`	greater than or equal to	`x >= y`	false
`==`	identically equal to	`x == y`	false
`!=`	not equal to	`x != y`	true
`&&`	logical and	`(x<y) && (x<z)`	true
`||`	logical or	`(y>x) || (y>z)`	true

Table 2.1: Summary of Logical and Relational Operators

2. **Control :** Control is the basic mechanism in a programming language for using the outcome of logical and relational expressions to guide the pathway of a program execution. ALADDIN supports the branching constructs of `if` and `if-then-else`, and the `while` and `for` looping constructs, with logical and relational operations being computed on physical quantities.

   A schematic of branching and looping constructs is shown in Figure 2.3. Branching and looping constructs have one entry and one exit. For example, the for-loop has the following syntax:

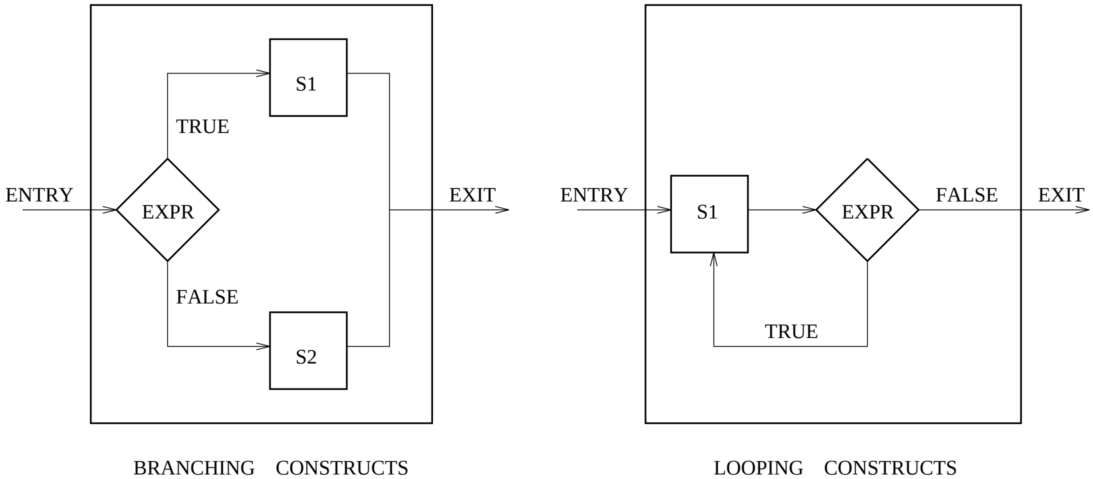


Figure 2.3: Branching and Looping Constructs in ALADDIN

Syntax	Example

`for(initializer;condition;increment){`
`    statements;`
`}`
	`for( x=1m; x<=5m; x=x+2m ){`
	`     print "x =", x, "\n";`
	`}`

The example generates the output:

Loop No.	`x`	`x <= 5 m`	Output
1	`1 m`	true	`x = 1 m`
2	`3 m`	true	`x = 3 m`
3	`5 m`	true	`x = 5 m`
4	`7 m`	false	

The `initializer`, `condition`, and `increment` statements can be zero or more statements separated by a comma. Similarly, zero or more statements may be located in the for-loop body. You should notice that all of the elements in the for-loop statement are dimensionally consistent. Execution of the for-loop begins with the `initializer` statement (i.e., we assign `1 m` to variable `x`). The `condition` statement is then evaluated to see if it is true or false. In this particular example, `x <= 5m` evaluates to true and so statements inside the body of the loop are executed (i.e., we print out the value of `x`). When all of the statements inside the `for`-loop body have finished executing, the `increment` statement is executed (i.e., we increase the value of `x` by `2m`) and `condition` is re-evaluated to see if another loop of computations is required. For this example, the loop cycles continue for 4 iterations, until the `condition` evaluates to false. At the end of the for-loop execution, the value of `x` will be 7 m.

3. **Functions :** The functional components of ALADDIN provide hierarchy to the solution of our matrix and finite element solution procedures, and are located in libraries of compiled C code, as shown on the right-hand side of Figure 2.1.

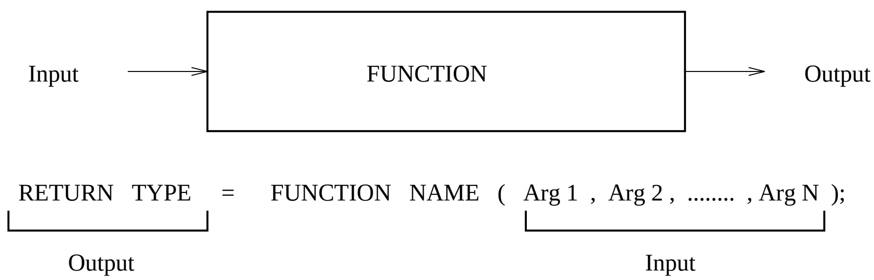


Figure 2.4: Schematic of Functions in ALADDIN

Figure 2.4 shows the general components of a function call, including the input argument list, the function name, and return type. Version 1.0 of ALADDIN has functional support for basic matrix and finite element operations. For example, the function call

```
spectra = Matrix ([20,2]);
```

calls the function `Matrix` to dynamically allocate memory for a 20 by 2 matrix of physical quantities, and assigns the result to a variable `spectra`.

The ALADDIN function `Matrix` accepts one matrix argument as its input, and returns one matrix as the result.

A key strategy we have followed in ALADDIN's development is to keep the number and type of arguments employed in library function calls small. Whenever possible, the function's return type and arguments should be of the same data type, thereby allowing the output from one or more functions to act as the input to following function call. More precisely, we would like to write input code that takes the form

```
eload = ExternalLoad();
stiff = Stiff();

displacement = Solve( stiff, eload );
```

If `ExternalLoad()` and `Stiff()` belong to application area 1 (e.g., finite element analysis), and `Solve()` belongs to application area 2 (e.g., matrix analysis), then this language structure allows application areas 1 and 2 to be combined in a natural way.

These three components are the basic ingredients of ALADDIN problem solving procedures involving matrix and finite element computations.

Finite Element Input Files : Finite element problems require the development of an input file having the following five-part format:

START OF INPUT FILE

```
/*  ========================================================= *
 *  A description of the finite element problem goes here... *
 *  ========================================================= */

  ... Part [1] : Problem specification parameters.
```

```
StartMesh();

  ... Part [2] : Generate finite element mesh. Specify section and material
                 properties, external loads, and boundary conditions.

EndMesh();

  ... Part [3] : Describe solution procedure for finite element problem.

  ... Part [4] : If applicable, check performance of structure against
                 design rules.

  ... Part [5] : If applicable, generate arrays of output that are suitable
                 for plotting with MATLAB.

quit;
```

The information supplied in each part of the input file is as follows:

1. The problem specification parameters allow an engineer to state whether a finite element problem will be two- or three- dimensional, the maximum number of degree of freedom per node, and the maximum number of nodes per element.

2. ALADDIN statements are written for the finite element mesh generation, the definition of section and material properties, the specification of external loads, boundary conditions, and for linking finite element degrees of freedom. Mesh generation begins and ends with the function calls:

   `StartMesh()` : Allocates the working memory for the finite element data structures.

   `EndMesh()` : Loads the information provided in Part [2] into the finite element database.

3. The problem solving procedure usually begins with the assembly of the global stiffness matrix, external load vectors, and if applicable, assembly of a global mass matrix. Specific linear/nonlinear static/dynamic finite element problems are solved by inserting the details of a numerical algorithm (e.g., Newmark integration, modal analysis, optimization procedures) at this point.

4. If applicable, the performance of structure is checked against rules from a design code.

5. If applicable, arrays of formatted program output are generated for specialized manipulations (see, for example, Section 5.2 on energy evaluation) and plotting with MATLAB.

2.2.2 Architecture of Program Modules

In order to develop a modular software, the program architecture is carefully partitioned into six modules. Figure 2.5 shows the interfaces and relationships among the six modules. They are (1) main module, (2) pre-processor module, (3) central control module, (4) matrix module, (5) finite element module, and (6) engineering units module. A summary of each module is as follows:

1. **Main Module :** The main module acts as the entry point for program execution. It loads information specified in ALADDIN's header files into the symbol table, directs the source of expected input (keyboard or files), and details of command options. Then the main module calls the pre-processor module, and executes the program. Also during initializing the environment, the program loads keywords, constants, finite element ana-

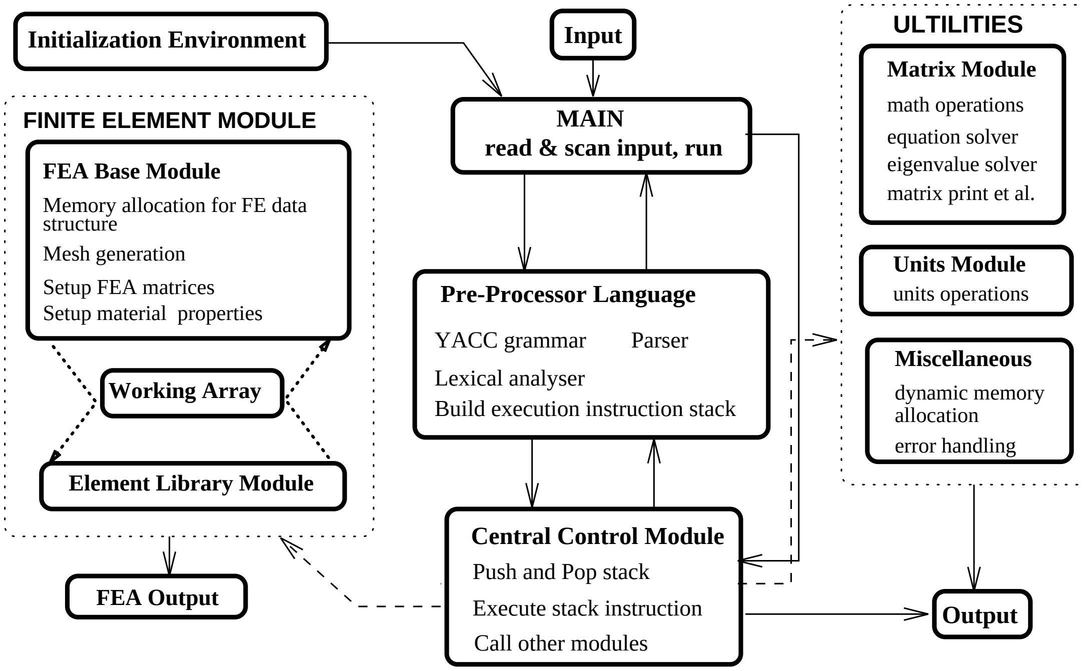


Figure 2.5: Architecture of Program Modules

lysis information, and built-in functions needed for the command language into ALADDIN's symbol table.

2. **Preprocessor Language Module :** ALADDIN's preprocessor module parses input, and prepares an array of machine instructions that will be executed by ALADDIN's stack machine. The design and implementation details for the language will be described in Section 2.3.

3. **Central Control Module :** The central control module is composed of a stack machine, and is the heart of ALADDIN. The design and implementation details for the stack machine will be described in later Section 2.4.

4. **Engineering Units Module :** The engineering units module provides the units operations of engineering quantities and matrices for both US and

SI systems of units. Operations for units conversion are provided, as are facilities to turn units operations on/off. The design and implementation details for the physical units will be described in Section 2.5.

5. **Matrix Module :** The matrix module contains functions to allocate memory for matrices, to compute basic matrix operations and solutions to families of linear equations, and to solve the symmetric eigenvalue problem.

6. **Finite Element Module :** The finite element module contains ALADDIN's data structures for finite element analysis, functions for finite element analysis, and a library of finite elements. All of these details are contained in a base module and an element library module. The information needed for the finite element analysis is generated and prepared by the base module. The base module also collects and assembles results generated by the element library module. The element library module has codes to compute element stiffness matrix, mass matrix, internal force, and nonlinear response if applicable. These two sub-modules transfer the data to each other through the use of working arrays.

2.2.3 Matrix and Finite Element Libraries

ALADDIN's built-in libraries widen the program's problem solving ability and are a big part of making the ALADDIN language extensible. These libraries are implemented as compiled C code.

The matrix library includes functions for dynamic allocating and printing matrices, transposing and inverting a matrix, substitution and extraction of sub-matrices, linear equation solver, eigenvalue/eigenvector problem solver, and other miscellaneous functions. Functions are provided for addition, subtraction, and multiplication of matrices, with or without units. For more details, see reference [6].

The built-in finite element library includes functions for: (a) the generation of finite element meshes, (b) the definition of external loads, (c) the specification of boundary conditions, (d) the specification of section and material properties, and (e) linking finite element degrees of freedom. To assist the engineer with basic finite element computations, built-in functions are provided for the assembly of global stiffness matrices, global mass matrices, and external load vectors. The finite element library also provides facilities for querying information on the finite element mesh, section and material properties, and the computed displacements and stresses.

Of course the finite element library also includes a collection of finite elements. The ALADDIN Version 2.0 finite elements include: two dimensional 4-node plane stress/plane strain element, two dimensional 2-node frame element, three dimensional 2-node frame element, three dimensional 4 and 8-node five degree of freedom shell elements (linear/nonlinear elastic-plastic materials), 4-node flat shell element with six degrees of freedom per node, 4-node discrete Kirchoff quadrilateral (DKQ) plate element, two dimensional and three dimensional 2-node fiber beam/column element (linear/bilinear elastic-plastic materials). Each finite element has source code to compute element stiffness and mass matrices, plus for a given displacement vector, a vector of internal forces acting on the nodal degrees of freedom.

2.3 Language Design and Implementation

A key design objective for ALADDIN is the development of a program structure that is very modular. We have captured this principle by designing the program architecture, and supporting software modules, around a "language grammar" and a compiler construction tool called YACC (an acronym for Yet Another Compiler Compiler) [22]. YACC takes a language description/grammar and automatically generates C code for a parser that will match streams of input against the rules of the language using a lookahead left recursive (LALR) strategy. In ALADDIN, the details of the YACC grammar and associated C code are located in a four-part file called grammar.y. The four parts are:

```
%{
  Part 1 : Optional C statement, declaration;
%}
  Part 2 : YACC declarations, lexical tokens, grammar variable,
           precedence and associativity information
%%
  Part 3 : Grammar rules and semantic actions
%%
  Part 4 : Lexical analysis with a C function called yylex().
```

In Part 4 of the YACC specification, we write a C function called `yylex()` to scan streams of input, and identify the name and types of tokens. Black spaces, tabs, and all input between comment statements is automatically removed from the input. Numbers must begin with either a digit of a decimal point – they are temporarily stored in ALADDIN's symbol table. Character strings are enclosed within quotes (i.e., "...."). Variables and built-in function names are alphanumeric strings that must begin with a character – the details of keywords in ALADDIN's programming language are stored in the symbol table (see Figure 2.2).

YACC takes the specifications in Parts 1 to 3 of grammar.y, and generates a C code function called `yyparse()` for: (a) the matching of tokens and their types against the grammatical rules of the language, and (b) the handling of semantic actions.

The ALADDIN Version 2.0 language employs 134 grammatical rules for the identification of dimensions, physical quantities, matrices, program control of flow statements, and so forth. Exact details of the grammar can be examined by downloading the grammar.y file from the ALADDIN Web site. To see how this process works in practice, consider the problem of parsing the input statements:

```
x = 2 in;
y = 5 in + x;
```

For the `x = 2 in` statement, the sequence of parsing operations is as follows:

1. The YACC parser recognizes that the whole input statement is a type "stmt" with the final result of type "quantity".

2. The parser recognizes that "quantity" (from step 1) is a combination of a variable `x` with token name VAR, an assign operator `=`, and a quantity `2 in` of type "quantity". The grammatical rule for assignment (i.e., `"="`) has right associativity, meaning that tokens to the right of the `=` will be handled before the result is assigned to `x`.

3. The parser recognizes that `2 in` is a quantity composed of a number `2` with token name NUMBER and a units dimension `in` of type "dimensions".

4. The parser recognizes that `in` is a dimension. It has the token name DIMENSION.

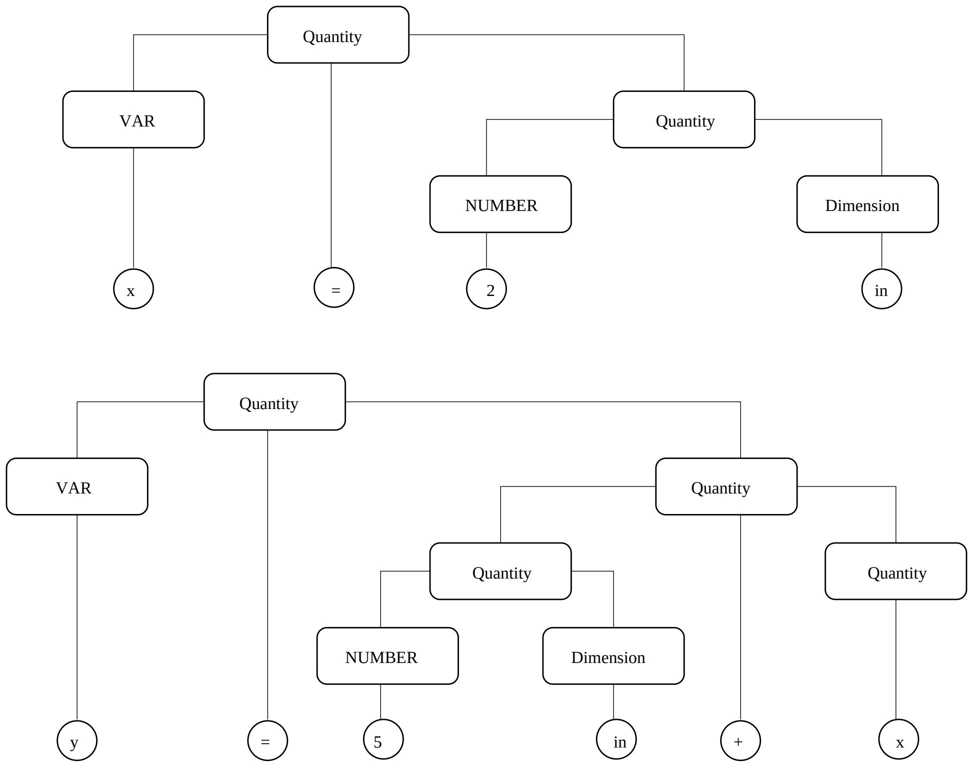


Figure 2.6: Parser Trees for `x = 2 in` and `y = 5 in + x`

The relationship among these components and steps is summarized by the parser tree in the upper half of Figure 2.6.

For the `y = 5 in + x` statement the parsing sequences are basically the same as in the preceding paragraph. The resulting parser tree is shown in the lower half of Figure 2.6. Because YACC rules are recursively defined, `5 in + x` is matched by:

$$\underbrace{\texttt{quantity : quantity '+' quantity}}_{\textit{Syntactic rule}} \qquad \{\ \underbrace{\texttt{Code(Quantity\_Add);}}_{\textit{Semantic action}}\ \}$$

This syntactic rule basically says that if the input stream includes a quantity, an arithmetical operator "+", and another quantity, then the result is recognized as a `quantity`. The appropriate semantic actions are specified inside the braces {}

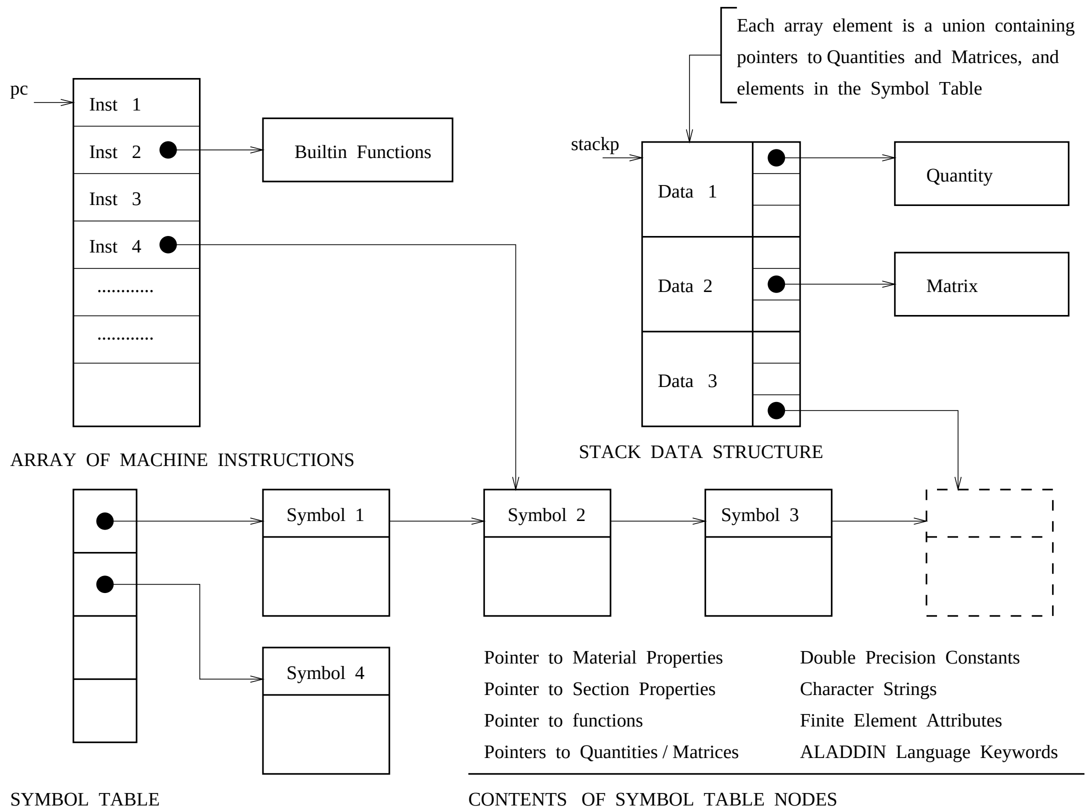


Figure 2.7: Data Structures in ALADDIN's Stack Machine

following the rule. In this particular case, the C code located inside the function `Quantity_Add()` generates the appropriate operations for the stack machine.

2.4 Stack Machine for Central Control

The data and control components of ALADDIN are implemented as a finite-state stack machine model, which follows in the spirit of work presented by Kernighan and Pike [23].

The purpose of Figure 2.7 is to show how the stack machine is constructed from three connected data structures; an array of machine instructions, a program stack and a symbol table. Each rule in the ALADDIN grammar has a semantic action that generates zero or more low-level stack machine instructions. Commonly used stack machine instructions include pushing and popping data to/from the program stack, retrieving data from the ALADDIN hash table, calling a function in the matrix/finite element libraries, and so forth. Each ALADDIN statement (or block of statements) generates a sequence of semantic actions that results in the construction of an array of stack machine instructions (see the upper left-hand corner of Figure 2.7. Note that because the underlying parsing algorithm is LALR, the stack machine array contains instructions beginning with the last rule parsed, and finishing with the first rule identified.

In phase two of the statement processing, the stack machine walks along the array of machine instructions and executes the functions pointed to by the machine instructions. These functions will retrieve matrices and physical quantities from the symbol table, and push copies onto the program stack (see the upper right-hand side of Figure 2.7). The pop and push operands of the program stack follow a last-in-first-out rule. When the stack machine has finished executing the array of instructions, the program stack will be empty.

Example of Machine Stack Execution : We now demonstrate use of the stack machine by working step-by-step through the details of processing the assignment `x = 2 in;`. At the conclusion of the statement parsing phase, the variable `x` (of undetermined data type) will have been added to the symbol table, and the array of machine instructions will contain the items shown on the top left-hand side of Figure 2.8. The step-by-step procedure for execution of the stack machine is:

1. Push symbol table pointer onto stack for the variable `in`.

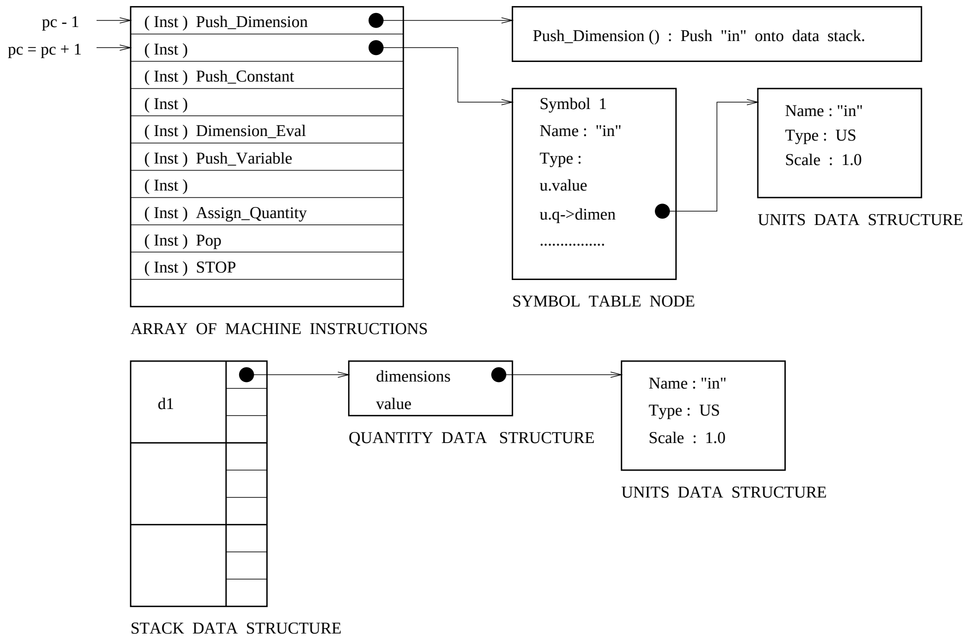


Figure 2.8: Step 1 – Push Unit onto Stack

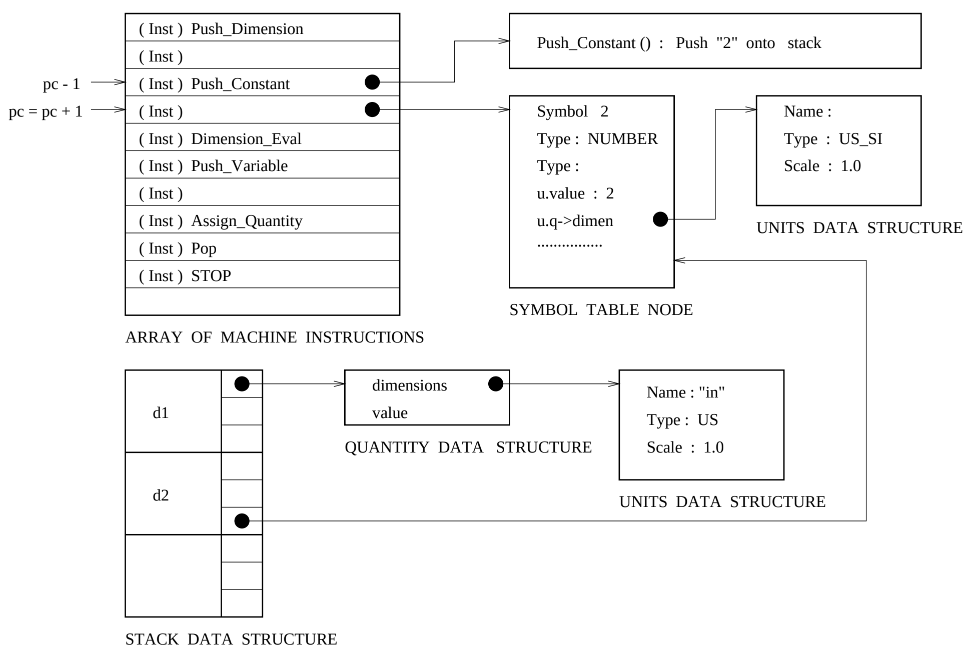


Figure 2.9: Step 2 – Push Number onto Stack

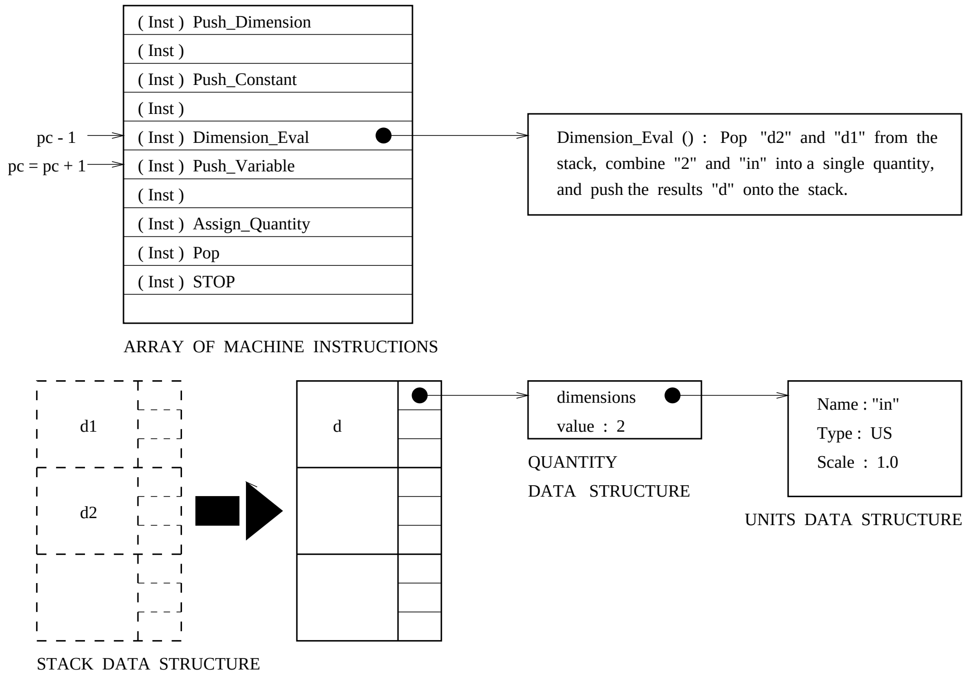


Figure 2.10: Step 3 – Combine Number and Unit into Quantity

2. Push a constant `2` onto the stack.

3. Pop both `2` and `in` off the stack, and combine them into a single quantity `2 in`. The quantity is pushed back onto the stack.

4. Push onto the stack, the symbol table pointer to variable `x`.

5. Pop `x` and `2 in` from the stack. Assign `2 in` to `x` and push `x` back onto the stack.

6. Data `x` is popped and cleared from the stack.

Figures 2.8 to 2.10 show the relevant details of the machine array, symbol table, and program stack for Steps 1 to 3 of the execution procedure. The step-by-step procedure is handled by the C function Execute();

```
int Execute( Inst *p ) {
    for(pc = p; *pc != STOP; ) {
        pc = pc+1;
        if( Check_Break() ) break;
        (*(*(pc-1)))();
    }
}
```

`pc` is a program counter that points to elements in the array of machine instructions, as shown in Figures 2.8 to 2.10. The elements of the machine array are pointers to C functions that implement the stack operation tasks. The arrows represent the two stages of `pc` positions. As shown by the upper arrow in Figure 2.8, the program counter `pc` initially points to `(Inst) Push_Dimension()`. You should notice that the program counter `pc` is incremented to the lower arrow position (`pc = pc+1`) before `Push_Dimension` is called. Now (`pc-1`) points to `(Inst) Push_Dimension()`, and the new value of `pc` (lower arrow position) points to a quantity stored in the symbol table (i.e., the units dimension `in` of the input statement). `Push_Dimension()` allocates memory for a new quantity and assigns `pc` to the base address. After the unit of the quantity has been copied to the newly allocated block, the new quantity is pushed onto the program stack. After the function `Push_Dimension()` has completed its execution, the program counter `pc` is increased to point to the next function `Push_Constant()`.

The `Push_Constant()` function extracts the value of quantity `2` from the symbol table, stores it in a new quantity, and pushes the copy onto the program stack (see `d1` and `d2` on the stack shown in Figure 2.9). The program counter `pc` is then incremented to point at `Dimension_Eval()`.

`Dimension_Eval()` pops `d1` and `d2` from the stack, assigns the units dimension `in` in `d1` to the quantity `d2` The result, `2 in`, is pushed onto the program stack (i.e., `d`), as shown in Figure 2.10.

Next, the symbol table pointer for variable `x` is pushed onto the program stack by the C function `Push_Variable()`. The C function `Assign_Quantity()` assigns the quantity `2 in` to variable `x`, and the result is stored in the symbol table. The new `x` is pushed onto the program stack. The second-to-last machine instruction pops the last item (i.e., the variable `x` having value `2` and units dimension `in`) from the program stack. Finally, `(Inst) STOP`, halts the looping mechanism in C function `Execute()`.

2.5 Physical Quantities in ALADDIN

A physical quantity is a measure of some quantifiable aspect of the modeled world. In structural engineering circles, basic engineering quantities such as length, mass, and force are defined by a numerical value plus physical units. Systems of physical units enable quantities to be expressed in a number of ways. For example, a certain length can be measured in terms of meters, inches, and feet. All three sets of units have the same dimension, with the numerical value of the physical quantity differing only by a scale factor. While a conversion between different dimensions is not possible, a conversion of units of the same dimension only requires a proper scaling.

2.5.1 Physical Units

Figure 2.11 is a schematic of the primary base units, supplementary units, and derived units that occur in structural analysis. This diagram is a subset of units presented in the Unit Conversion Guide [4]. Derived units are expressed algebraically in terms of base and supplementary units by means of multiplication and division. Some of the derived units have special names that may themselves

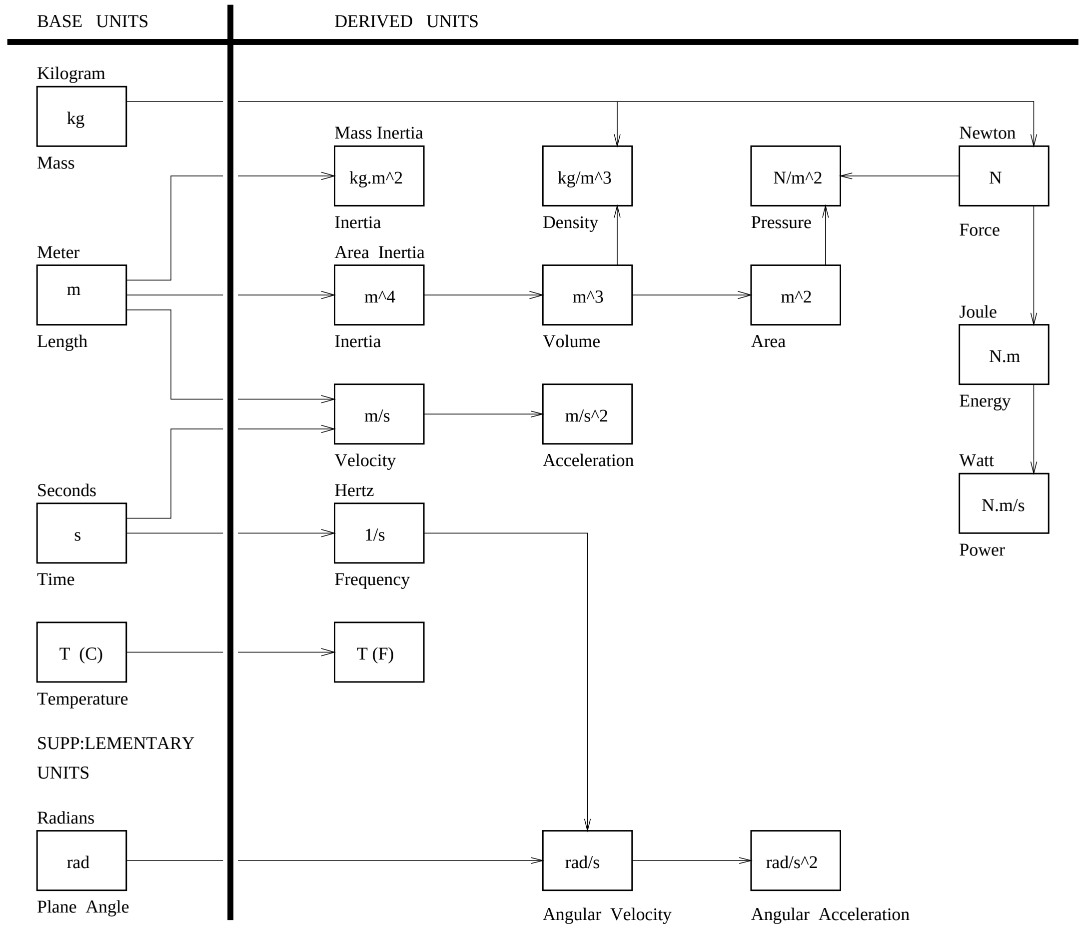


Figure 2.11: Primary Base and Derived Units in Structural Analysis

be used to express other derived units in a simpler way than in base units.

ALADDIN's engineering units module supports operations on engineering quantities and matrices for both US and SI systems of units. A physical quantity as an "unscaled value" with respect to a set of reference units, and by default, all quantities are stored internally in the SI units system. Basic engineering quantities such as length, mass, and force, are defined by a numerical value plus physical units. The four basic units needed for engineering analysis are the length unit L, the mass unit M, the time unit t, and the temperature unit T. In the SI system of units meter "m" is the reference for length, kilogram "kg" for mass, second "sec" for time and "deg_C" for temperature. Planar angles are represented by the supplementary base unit *rad*. Any engineering unit can be obtained with the following combinations:

$$\text{unit} = kL^{\alpha}M^{\beta}t^{\gamma}T^{\delta} \cdot \text{rad}^{\epsilon} \tag{2.1}$$

where $\alpha, \beta, \gamma, \delta$ and ϵ are exponents, and k is the scale factor. For units of length and mass, the family of exponent settings $[\alpha, \beta, \gamma, \delta, \epsilon]$ are $[1, 0, 0, 0, 0]$ and $[0, 1, 0, 0, 0]$, respectively. Numbers are non-dimensional quantities, and given by the family of zero exponents (i.e., $[\alpha, \beta, \gamma, \delta, \epsilon] = [0, 0, 0, 0, 0]$).

Arithmetic Operation with Units : In addition to providing clarity for problem input and output for engineering applications, the integration of units into arithmetic operations on physical quantities provides a powerful check for the dimensional consistency of formulas. Put another way, ALADDIN enables units to be carried along its computations. They act like variables obeying associativity and commutativity, and laws of exponents. Units are compatible if they represent equivalent physical quantities, which means that the values of all units exponents are the same. In fact, many improperly formed expressions can be identified without an in-depth knowledge of the problem background.

Conversions between different but equivalent sets of units is performed automatically. Units of the same dimension are chosen freely, and differ only by their scale. Scale factors are needed to convert units in US to SI, and vice versa. All quantities in US units must be converted into SI before they can be used in calculations.

The main advantage in defining the unit's data structure by equation 2.1 is that we can easily compute the arithmetic operation involving units. To see how this works in practice, let q_1 be a physical quantity with unit scale factor k_1, and unit exponents $[\alpha_1, \beta_1, \gamma_1, \delta_1, \epsilon_1]$. And let q_2 be a physical quantity with unit scale factor k_2, and unit exponents $[\alpha_2, \beta_2, \gamma_2, \delta_2, \epsilon_2]$. The logical and relational comparison operations of two physical quantities q_1 and q_2 (i.e., $q_1 \equiv q_2$, $q_1 \neq q_2$, $q_1 < q_2$, $q_1 > q_2$, $q_1 \leq q_2$, $q_1 \geq q_2$)can proceed only if the units of q_1 and q_2 are equivalent. A units compatibility check is made before the operation proceeds. The same units checking is also required for addition and subtraction operations.

For addition and subtraction of physical quantities the unit of the result depends on the unit types of the operands. If the operands all have the same units type (e.g., let us say all are SI or US type), then the result units will be set according to the first operand. If the operands have different units type, then the result units will be set according to the operand which has the same units type as the environmental units type. When there are more than one operands which have the same units type as the environmental units type, the first one will be the basis for the result units. The default environmental units type is SI. It can be switch to US type in the input using function `SetUnitsType()`.

Description	Expression	Scale Factor	Unit Exponents
Multiplication	$q_1 * q_2$	$k_1 \cdot k_2$	$[\alpha_1+\alpha_2, \beta_1+\beta_2, \gamma_1+\gamma_2, \delta_1+\delta_2, \epsilon_1+\epsilon_2]$
Division	q_1/q_2	k_1/k_2	$[\alpha_1-\alpha_2, \beta_1-\beta_2, \gamma_1-\gamma_2, \delta_1-\delta_2, \epsilon_1-\epsilon_2]$
Modulus	q_1/q_2	k_1/k_2	$[\alpha_1-\alpha_2, \beta_1-\beta_2, \gamma_1-\gamma_2, \delta_1-\delta_2, \epsilon_1-\epsilon_2]$
Exponential	$q_1 \hat{} q_2$	$k_1^{N\dagger}$	$[N\alpha_1, N\beta_1, N\gamma_1, N\delta_1, N\epsilon_1]^\dagger$

† N is the value of q_2.

1. The expression of modulus only makes sense when the values of both operands are integers.

2. The expression of exponential only makes sense when q_2 is dimensionless.

Table 2.2: Physical Units in Arithmetic Operation

Table 2.2 shows the result units in other arithmetic operations, including multiplication, division, modulus, and exponential.

2.5.2 Matrices of Physical Quantities

The units for elements in a matrix are stored in two one-dimensional arrays of data type DIMENSIONS. One array stores column units, and the second array row units. The units for matrix element at row `i` and column `j` is simply the product of the $\mathtt{i}^{th}$ element of the row units buffer and the $\mathtt{j}^{th}$ element of column units buffer. For example, a 4×4 matrix "stiff" having the row and column buffers is shown in Figure 2.12.

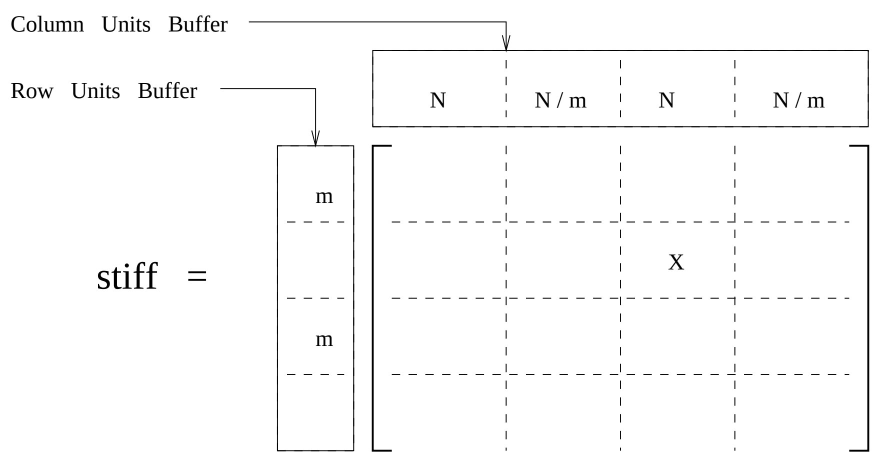


Figure 2.12: Matrix with Units Buffers

The units for matrix element `stiff[i][j]` are defined by the product of units at the $\mathtt{i}^{th}$ and $\mathtt{j}^{th}$ locations of the row and column units buffers. This strategy for storing units not only requires much less memory than complete element-by-element storage of units, but it reflects the reality that most engineering matrices are in fact, convenient representations of equations of motion and equilibrium. The units of individual terms in these equations must be consistent.

Matrix Addition/Subtraction : To illustrate the application of matrix operations with units, let A be a (2×2) stiffness matrix with row units buffer [m,   ] and column units buffer $[N, N/m]$. And let B be a (2×2) matrix with row units buffer [cm,   ] and a column units buffer $[kN, kN/cm]$. A missing item in the row/column units buffers means the corresponding units component is dimensionless. The units for matrix elements $(A)_{11}$ and $(B)_{11}$ are $m * N$ and $cm * kN$, respectively.

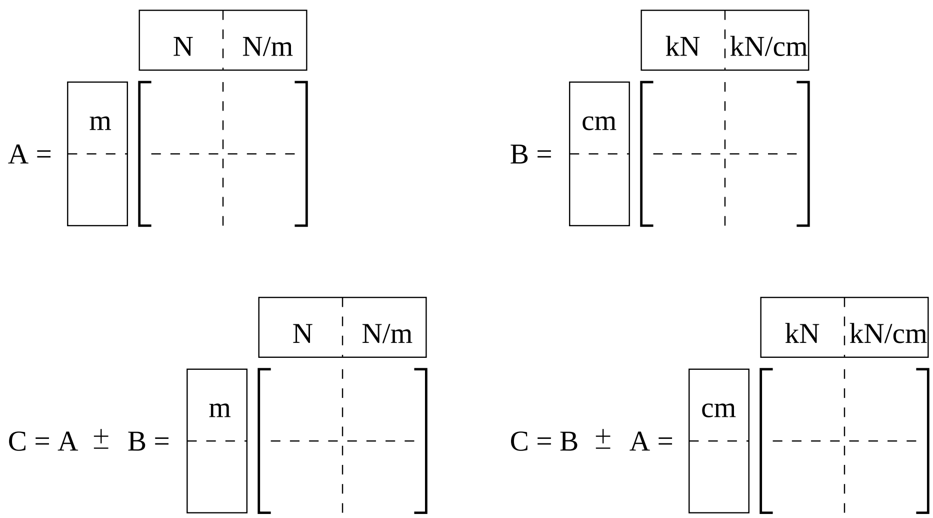


Figure 2.13: Units Buffer Addition/Subtraction of Two Matrices

Now let matrix C be the result of A $\pm$ B. The units of the output matrix C will be the same as A or B, depending on which one is the first operand (see Figure 2.13). However, before the matrix operation can proceed, the units of A and B must be checked for their compatibility. The matrix dimensions must be checked for consistency.

Matrix Multiplication : The handling of units in the multiplication of two dimensional matrices needs special attention. Let A be a $(p \times q)$ matrix with row units buffer [$a_1, a_2, \cdots, a_p$] and column units buffer [$b_1, b_2, \cdots, b_r$]. And let B be a $(q \times r)$ matrix with row units buffer [$c_1, c_2, \cdots, c_q$] and a column units buffer $[d_1, d_2, \cdots, d_q]$. The units for elements $(A)_{ik}$ and $(B)_{kj}$ are $a_i * b_k$ and $c_k * d_j$, respectively. Moreover, let C be the product of A and B. From basic linear algebra we know that $(C)_{ij} = A_{ik} * B_{kj}$, with summation implied on indices k. The units accompanying $(C)_{ij}$ are $a_i b_k * c_k d_j$ for k = 1, 2, ..., q.

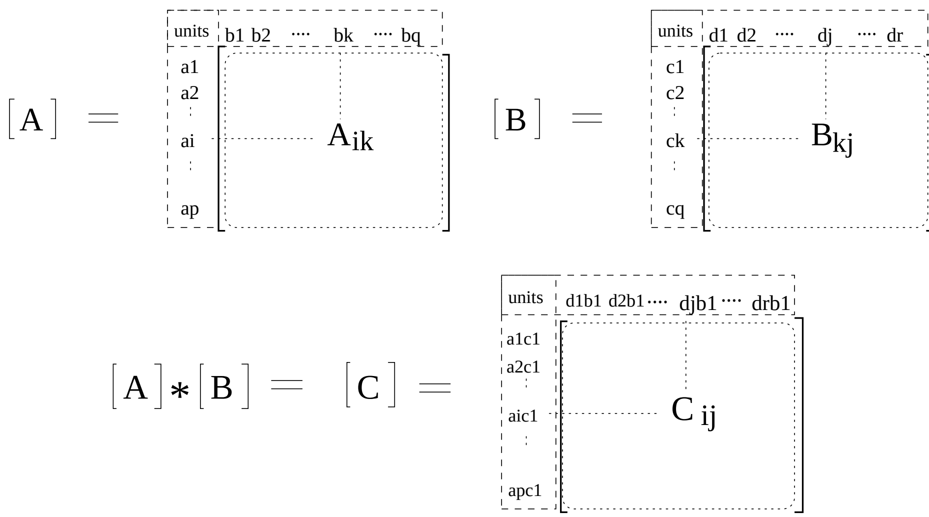


Figure 2.14: Units Buffer Multiplication of Two Matrices

Due to the consistency condition, all of the terms in $\sum_{k=1}^{q} A_{ik} * B_{kj}$ must have same units. This check is made for every element in the matrix (i.e., $a_i b_1 c_1 d_j = a_i b_2 c_2 d_j = \cdots = a_i b_q c_q d_j$). The units for C_{ij} are $a_i b_1 c_1 d_j$. The units buffers for matrix C are written as a row units buffer [$a_1 c_1, a_2 c_1, \cdots, a_p c_1$], and a column buffer is [$d_1 b_1, d_2 b_1, \cdots, d_r b_1$]. This arrangement of units exponents is graphically displayed in Figure 2.14. It is important to notice that although the units for matrix C are unique, the solution for the units buffers is not.

CHAPTER 3

Procedures for Linear Static Structural Analysis

3.1 Specifications of Finite Element Mesh

This section briefly describes ALADDIN's capabilities for finite element mesh generation, including addition of nodes and elements, specification of section and material properties, external loads, and boundary conditions.

3.1.1 Problem Specification Parameters

ALADDIN's specification parameters are used for the allocation of memory for the finite element mesh.

Short Example : In this short example, we initialize the problem specification parameters for a three-dimensional structural analysis that uses an eight-node shell finite element.

```
NDimension        = 3;
NDofPerNode       = 5;
MaxNodesPerElement = 8;   /* .... etc ..... */
```

3.1.2 Adding Nodes and Finite Elements

Two functions, `AddNode()` and `AddElement()`, are employed for the generation of finite element nodal coordinates, and the attachment of finite elements to the nodes.

Short Example : Figure 3.1 shows a two-dimensional coordinate system, and a line of six finite element nodes connected by 2-node beam finite elements. The nodes are located at y-coordinate = 1 m, and are spaced along the x-axis at 1 m centers, beginning at x = 1 m and finishing at x = 6 m.

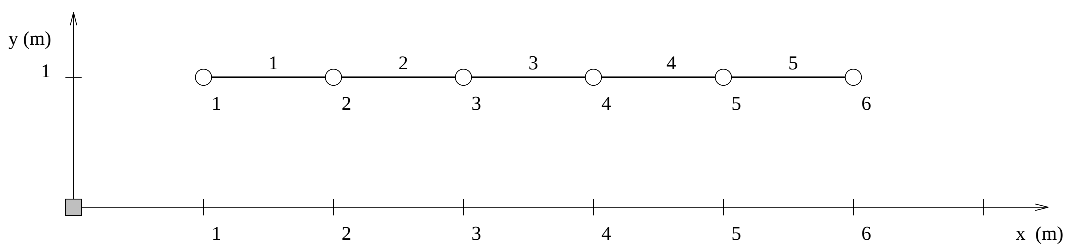


Figure 3.1: Line of Nodal Coordinates and Beam Finite Elements

ALADDIN's looping constructs are ideally suited for specification of the finite element nodes in a compact manner, and for the attachment of the two-node finite elements. For example, the fragment of code:

```
print "*** Generate grid of nodes for finite element model \n\n";

nodeno = 0;
x = 1 m; y = 1 m;
while(x <= 6 m) {
    nodeno = nodeno + 1;
    AddNode(nodeno, [x, y]);
    x = x + 1 m;
}

print "*** Attach finite elements to nodes \n\n";
```

```
elmtno = 0; nodeno = 0;
while(elmtno < 5) {
    elmtno = elmtno + 1; nodeno = nodeno + 1;
    AddElmt( elmtno, [nodeno, nodeno + 1], "name_of_elmt_attr");
}
```

generates the one-dimensional mesh shown in Figure 3.1. In the first half of the script, 6 nodal coordinates are added to ALADDIN's database. The second block of code attaches 5 elements to the nodes. You should notice how we have used the notation `[nodeno, nodeno + 1]` to generate (1×2) matrices containing the node numbers to which that `elmtno` will be attached.

3.1.3 Material and Section Properties

The element, section, and material type attributes are specified with three functions, `ElementAttr()`, `SectionAttr()` and `MaterialAttr()`, followed by parameters inserted between braces $\{\cdots\}$. A list of the section and material property parameters can be found in Reference [6].

Short Example : The following script loads a finite element attribute called "floorelmts" into ALADDIN's database.

```
ElementAttr("floorelmts") { type     = "FRAME_2D";
                            section  = "floorsection";
                            material = "floormaterial";
                          }

SectionAttr("floorsection") { Ixy     = 1 m^4;
                              Iyy     = 2 m^4;
                              Ixx     = 3 m^4;
                              Izz     = 0.66666667 m^4;
                              depth   =   2 m;
                              width   =   1.5 m;
                            }

MaterialAttr("floormaterial") {  E    = 1E+7 kN/m^2;
                                 density = 0.1024E-5 kg/m^3;
```

```
     poisson = 1.0/3.0;
     yield   = 36000 psi;
}
```

The "floorelmts" attribute has three components – the finite element type is set to FRAME_2D, for two-dimensional beam column finite elements. The element's section and material properties are defined via links to the section attribute "floorsection" and the material attribute "floormaterial."

3.1.4 Boundary Conditions

Boundary conditions are applied to a structure using the `FixNode()` function. `FixNode()` has one matrix argument containing one row, and a number of columns equal to the number of degrees of freedom at the node. A matrix element value of 1 means that the corresponding degree of freedom is fully fixed. A matrix element value of 0 means that the corresponding degree of freedom is free to move.

Short Example : This script fixes the boundary conditions of a finite element mesh at nodes 1 through 5.

```
dx = 1; dy = 1 ; dz = 1;
rx = 0; ry = 0 ; rz = 0;

bcond = [ dx, dy, dz, rx, ry, rz ];

for( iNode = 1; iNode <= 5; iNode = iNode + 1 ) {
     FixNode ( iNode , bcond );
}
```

We have used the variables `dx`, `dy`, and `dz` to represent translational displacements in the x, y, and z directions, respectively, and the variables `rx`, `ry`, and `rz` for rotational displacement about the x, y, and z axes. Now nodes 1 through 5 are fixed in their translational degrees of freedom, and pinned in the three rotational degrees of freedom.

3.1.5 External Nodal Loads

Externally applied nodal loads are specified with the function `NodeLoad(nodeno, load_vector);`.

Short Example : The following script adds two translational forces, and one moment to nodes 1 through 5 in a two-dimensional finite element mesh.

```
FxMax = 1000.0 lbf; Fy = -1000.0 lbf; Mz = 0.0 lb*in;

for( iNode = 1; iNode <= 5; iNode = iNode + 1 ) {
     Fx = (iNode/5)*FxMax;
     NodeLoad( iNode , [ Fx, Fy, Mz ]);
}
```

Externally applied loads in the x-direction are `200 lbf` at node 1, and increase linearly to `1000 lbf` at node 5. A gravity load of `Fy = -1000.0 lbf` is applied to each of the nodes 1 through 5.

3.2 Generation of Mass and Stiffness Matrices

After the details of the finite element mesh have been fully specified, the next step is to calculate the finite element properties and assemble them into an equilibrium system. For structural finite element analysis, this step involves calculation of the stiffness and mass matrices, and an external load vector.

Short Example : In the following script of code, `mass` is the global mass matrix, `stiff` is the global stiffness matrix, and `eload` is a vector of external nodal loads applied to the finite element global degrees of freedom.

```
mass  = Mass([1]);    /* [1] : lumped mass, [-1] : consistent mass */
stiff = Stiff();
eload = ExternalLoad();
```

Note that the three functions `Mass()`, `Stiff()`, and `ExternalLoad()` should be called only after the function call to `EndMesh()`.

3.3 Solution of Linear Matrix Equations

After the finite element mesh has been generated, and the mass, stiffness , and external load matrices have been assembled, the next step in the structural analysis is solution of the linear matrix equations

$$[A]\{x\} = \{b\}$$

where $[A]$ is a $(n \times n)$ square matrix, and $\{x\}$ and $\{b\}$ are $(n \times 1)$ column vectors. For a linear structural analysis $[A]$ will correspond to the stiffness matrix, $\{b\}$ will be the vector of externally applied nodal loads, and $\{x\}$ will be the vector of nodal displacements that needs to be computed.

Generally speaking, the computational work required to solve one or more families of linear equations is affected by:

1. The size and structure of matrix `A`,
2. The computational algorithm used to compute the numerical solution, and
3. The number of separate families of equations for which solutions are required.

Figure 3.2 summarizes four pathways of computation for the solution of linear equations $[A]\{x\} = \{b\}$. When matrix `A` is is either lower or upper triangular form, solutions to $[L]\{x\} = \{b\}$ can be computed with forward substitution,

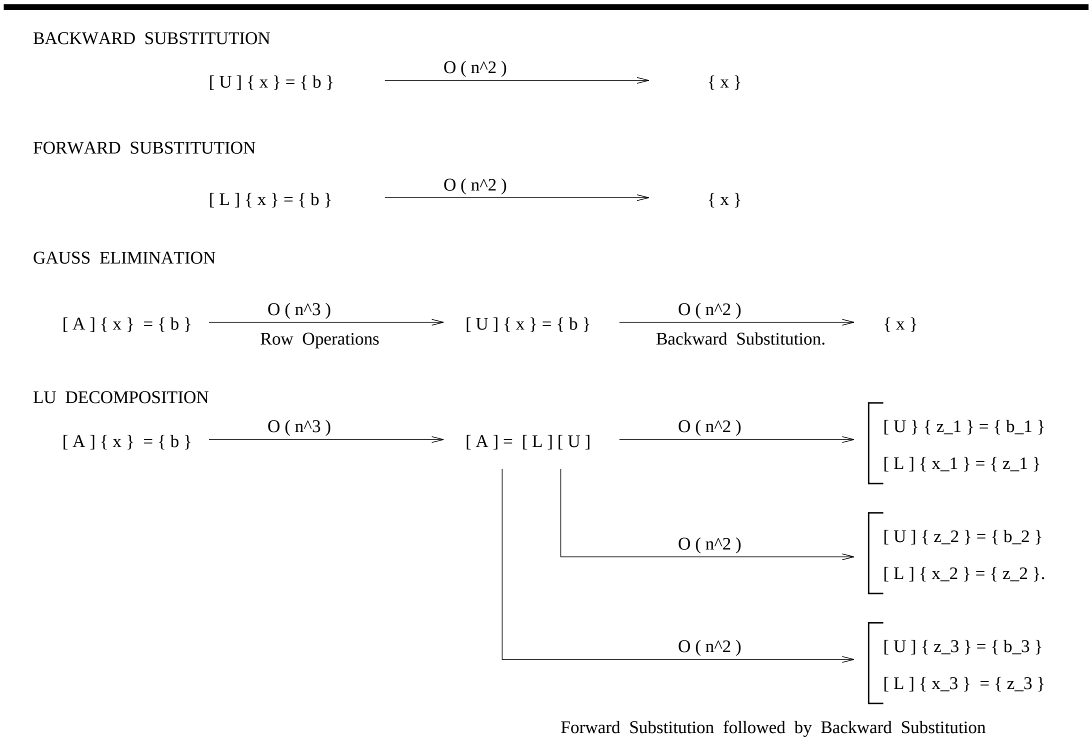


Figure 3.2: Strategies for Solving $[A]\{x\} = \{b\}$

and solutions to $[U]\{x\} = \{b\}$ via backward substitution. Algorithms for forward/backward substitution require $O(n^2)$ computational work. The method of Gauss Elimination is perhaps the most widely known method for solving systems of linear equations. In the first stage of Gauss Elimination, a set of well defined row operations transforms $[A]\{x\} = \{b\}$ into $[U]\{x\} = \{b^*\}$. In the second stage of Gauss Elimination, the solution matrix `x` is computed via back substitution. The first and second stages of Gauss Elimination require $O(n^3)$ and $O(n^2)$ computational work, respectively.

When solving the families of equations [A]{x}= {b} many times with different right-hand side vectors `b`, for example, in Figure 3.2, $\{b_1\}$, $\{b_2\}$, and $\{b_3\}$ represent three distinct right-hand sides to [A]{x}= {b}, the optimal solution procedure is to first decompose A into a product of lower and upper triangular matrices (requiring $O(n^3)$ computational work), and then use forward substitution to solve $[L]\{z\} = \{b\}$, followed by backward substitution for $[U]\{x\} = \{z\}$ (this step requires $O(n^2)$ computational work). While solutions to the first set of equations requires $O(n^3)$ computational work, solutions to all subsequent families of [A]{x} = {b} requires only $O(n^2)$ computational work.

The solutions to linear equations can be computed with the single command

```
x = Solve(A, b).
```

The ALADDIN function `Solve()` computes the solution to a single family of equations via the method of LU decomposition. The numerical procedure is identical to the two-command sequence

```
LU = Decompose(A);
x  = Substitution(LU, b).
```

LU decomposition can be used to solve a family of equations with different right-hand side vectors `b`. For example:

```
LU = Decompose(A);
x1  = Substitution(LU, b1).   /* Solve [A].x1 = b1 */
x2  = Substitution(LU, b2).   /* Solve [A].x2 = b2 */
x3  = Substitution(LU, b2).   /* Solve [A].x3 = b3 */
```

The following section contains two numerical examples that demonstrate how the solution of matrix equations applies to the analysis of highway bridge structures.

3.3.1 Three-Dimensional Analysis of Highway Bridge

This example illustrates the linear elastic three-dimensional analysis of a two-span highway bridge using a four node shell element. The highway bridge will

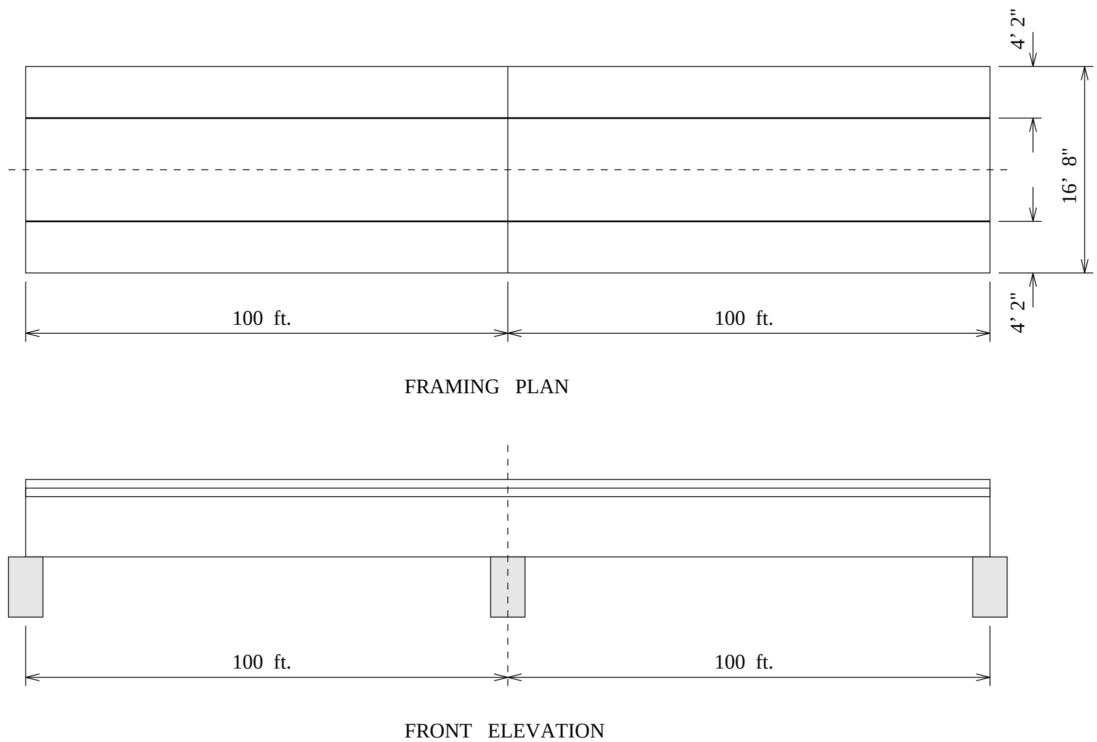


Figure 3.3: Plan and Front Elevation of Bridge

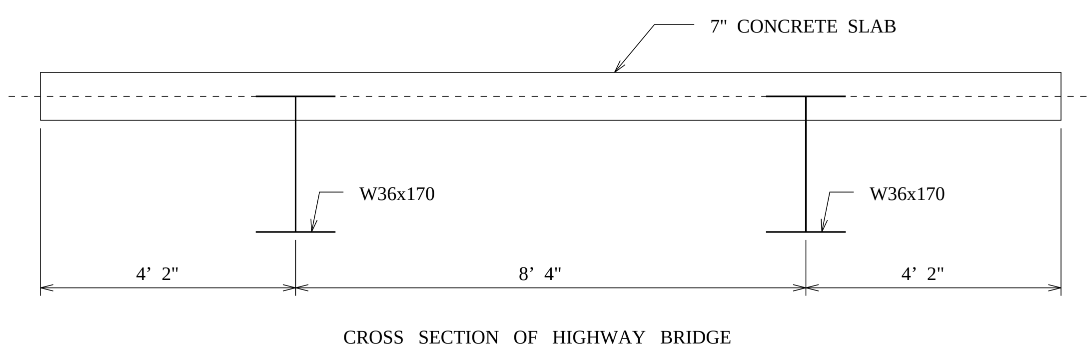


Figure 3.4: Cross Section of Bridge

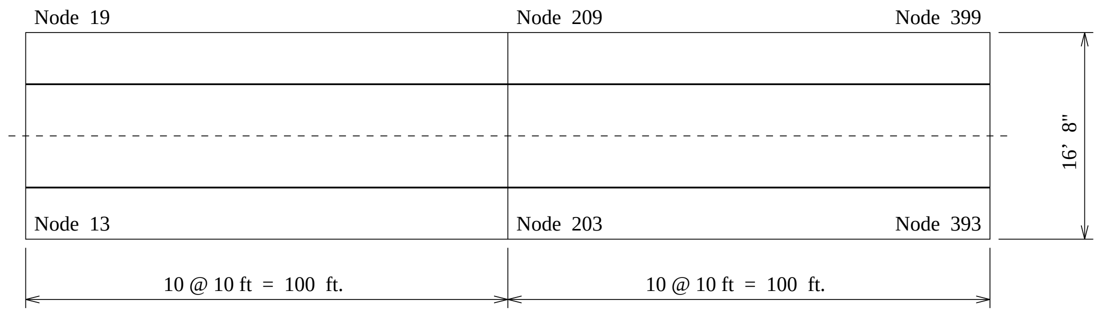


Figure 3.5: Plan of Finite Element Mesh for Bridge

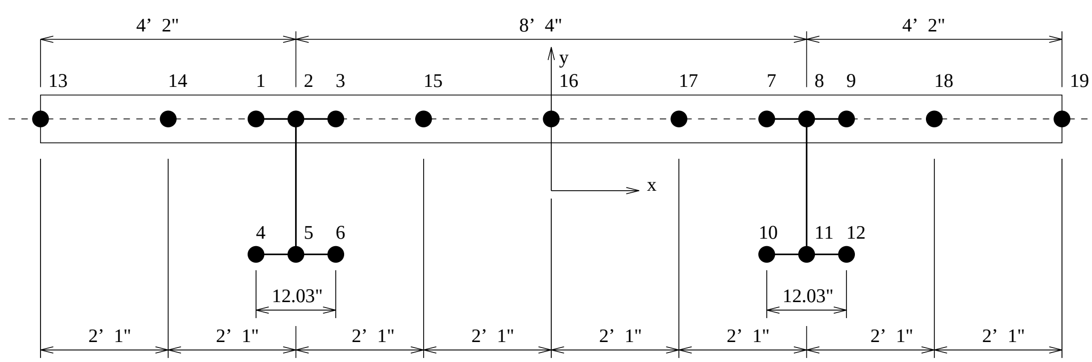


Figure 3.6: Cross Section of Finite Element Mesh for Bridge

be analyzed for two loading conditions. First, we compute the deflections of the bridge due to gravity loads alone, and in part two, the moving live load diagram generated by a truck moving across the bridge.

A plan and front elevation view of the highway bridge is shown in Figure 3.3. The bridge has two spans, each 100 ft long. The width of the bridge is 16 ft - 8 inches. A cross section view of the bridge is shown in Figure 3.4.

A detailed description of the material properties can be found in Austin et al. [6]. The left-hand side of the bridge has a hinged support boundary condition. The right-hand side of the bridge is supported on a roller. The finite element model has 399 nodes and 440 shell elements. After the boundary conditions are applied, the model has 2374 d.o.f.

Static Dead Load Analysis : In part one of analysis, we conduct a static analysis with dead load alone. The abbreviated input file is:

ABBREVIATED INPUT FILE

```
   ...... details of parameter definition ......

   StartMesh();

   ...... details of mesh generation ......

   EndMesh();

/* Compute stiffness matrix and external load vector */

   eload = ExternalLoad();
   stiff = Stiff();

/* Compute and print static displacements */

   displ  = Solve( stiff, eload);

/* Print analysis results and quit program */

   PrintDispl(displ);
   PrintStress(displ);
```

```
quit;
```

Figures 3.7 and 3.8 are a contour plot and three-dimensional view of the bridge deck deflections, respectively. (the vertical axis of the deflection of Figure 3.8 has the units of inches). The bridge deflections are due to dead loads alone, and since the bridge geometry and section properties are symmetric, we expect that the deflections will also exhibit symmetry. They do.

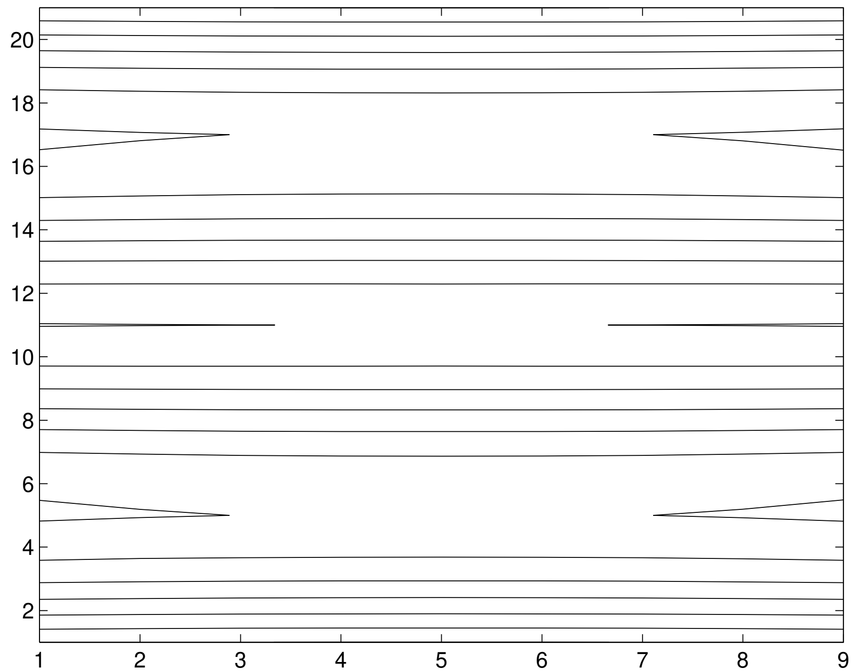


Figure 3.7: Contour Plot of Bridge Deck Deflections

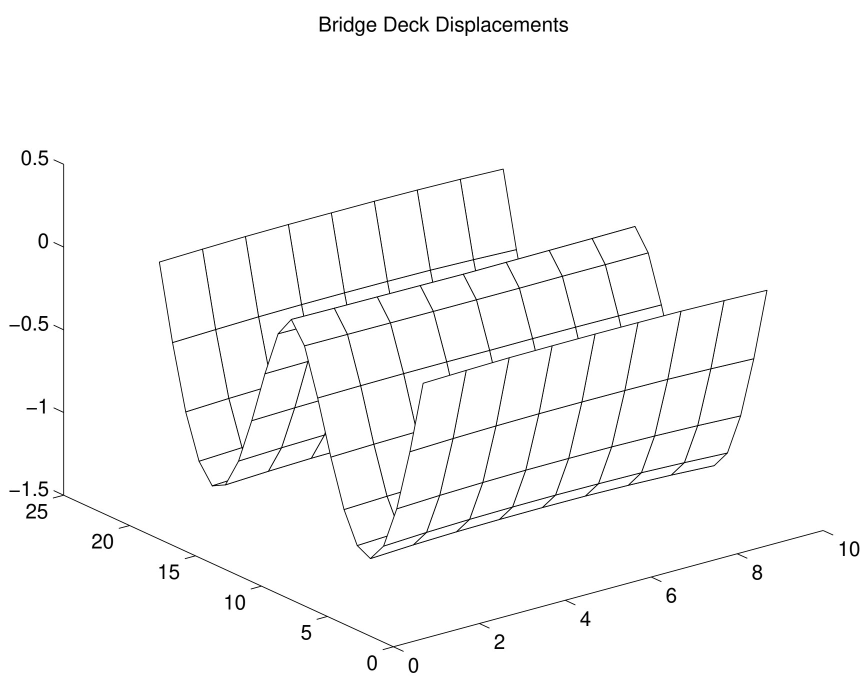


Figure 3.8: Three-Dimensional Mesh of Bridge Deck Deflections

Moving Truck Load Analysis : In part two of the bridge analysis, a 1000 kips concentrated live load moves along one of the outer bridge girders. We compute and plot the influence line for the moving load. The latter sections of the input file are extended so that response envelopes are computed for a point load moving along the bridge.

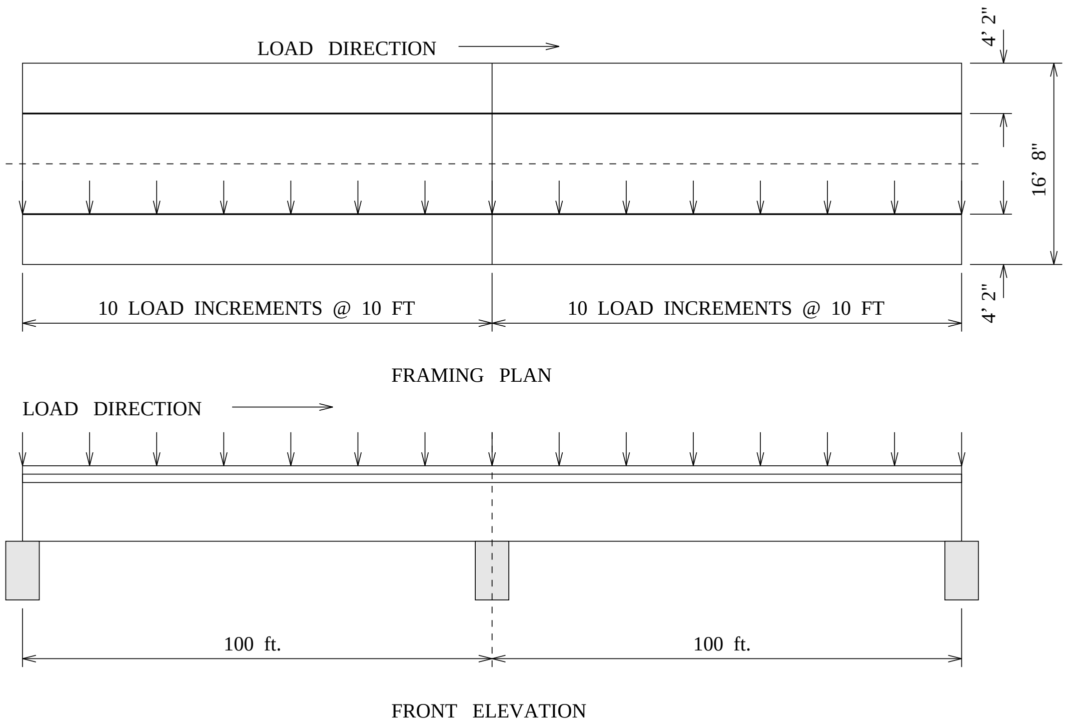


Figure 3.9: Plan and Front Elevation of Bridge with Moving Live Load

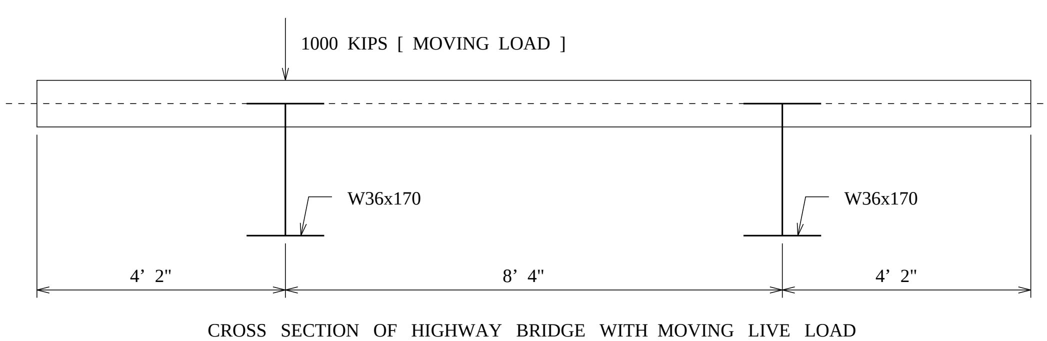


Figure 3.10: Cross Section of Bridge with Moving Live Load

Figure 3.9 shows plan and elevation views of the bridge and moving point load. A cross sectional view of the bridge and the moving point load is shown in Figure 3.10. The relevant details of the abbreviated input file are as follows:

ABBREVIATED INPUT FILE

```
/* Compute stiffness matrix and LU decomposition */

   stiff = Stiff();
   lu = Decompose (stiff);

/* Adding moving truck load

   for( i=1 ; i<=step ; i=i+1 ) {
        NodeLoad( load_node, [Fx,Fy,Fz,Mx,My,Mz] );

        eload = ExternalLoad();
        displ = Substitution( lu, eload );

        node_displ_1 = GetDispl( [nodeno1], displ );
        node_displ_2 = GetDispl( [nodeno2], displ );

        influ_line1[i][1] = node_displ_1[1][3];
        influ_line2[i][1] = node_displ_2[1][3];

        NodeLoad( load_node, [-Fx,-Fy,-Fz,-Mx,-My,-Mz] );
        load_node = load_node + nodes_per_section;
   }
```

Figure 3.11 shows the influence line of vertical displacement in the middle of one span (i.e., at node no 97) for the first girder subjected to 1000 kips moving live load. Similarly, Figure 3.12 shows the influence line of displacement in the middle of one span (i.e., node no 2 = 103) due to the 1000 kips moving live load.

The moving load analysis is one situation where a family of linear equations is solved with multiple right-hand sides. With this observation in mind, notice how we have called the function `Decompose()` once to decompose `stiff` into a product of upper and lower triangular matrices, and then called `Substitution()` to compute the forward and backward substitution for each analysis. This strategy of equation solving reduces the overall solution time by approximately 70%.

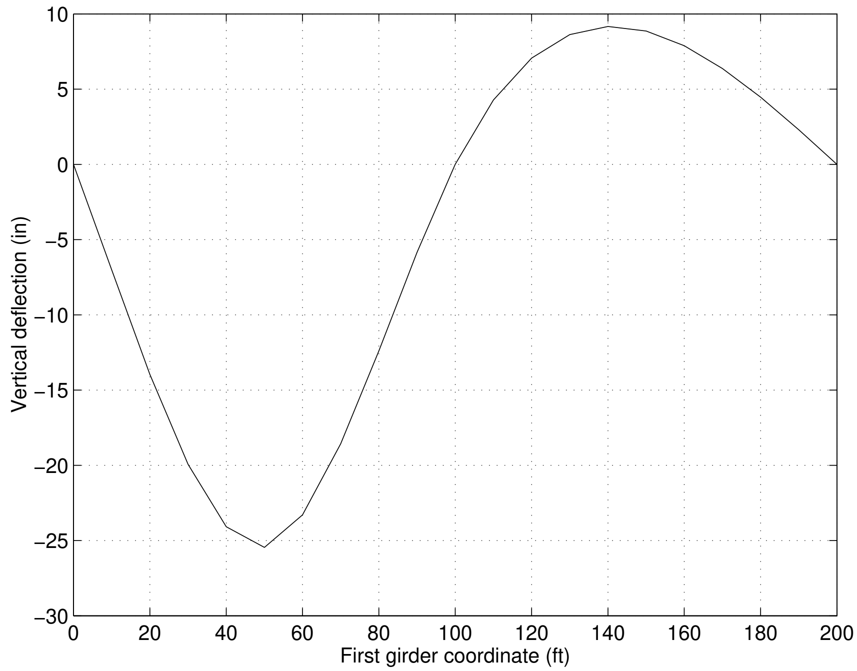


Figure 3.11: Influence Line of Displacement in the Middle of One Span for the First Girder Subjected to 1000 kips Moving Live Load

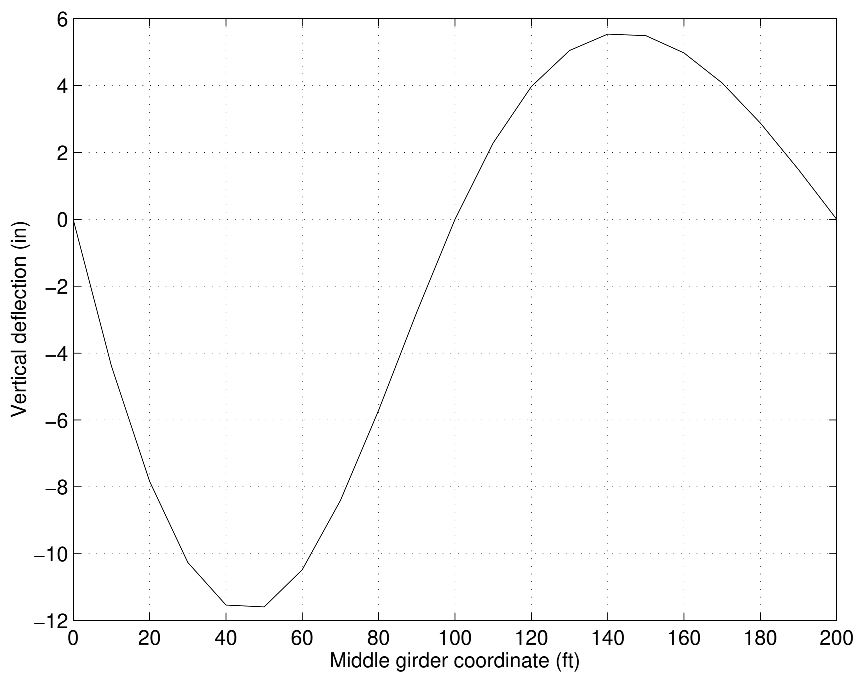


Figure 3.12: Influence Line of Displacement in the Middle of One Span for the Second Girder

3.3.2 Working Stress Design (WSD) Checking of Simplified Bridge

This example illustrates the application of ALADDIN to AASHTO Working Stress Design (WSD) code checking. It is adapted from one of the examples for MERLIN DASH presented in the ENCE 751 class notes, University of Maryland [40].

We will conduct a finite element analysis of a one span simply supported composite bridge that has a cover plate under the girders. The girder response will then be checked against a small family of WSD design rules. The analysis will be simplified by considering only an internal girder of the bridge system. A plan and cross sectional view of a typical bridge system is shown in Figures 3.13 and 3.14.

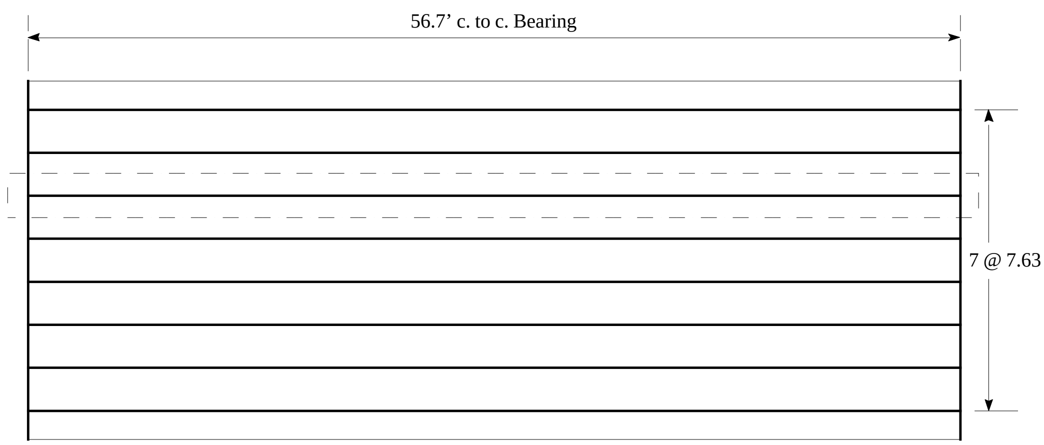


Figure 3.13: Plan of Highway Bridge

The bridge girders are made of rolled beam W33x130 with a 14" $\times$ 3/4" steel cover plate. An elevation view of the bridge and the position of the steel cover plate is shown in Figure 3.15. The material properties are F_y = 50 ksi and E_s = 29000 ksi. The effective cross sectional properties of the composite

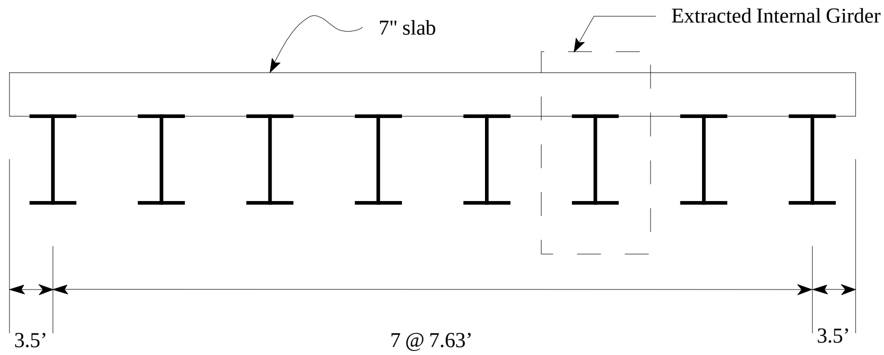


Figure 3.14: Typical Cross Section

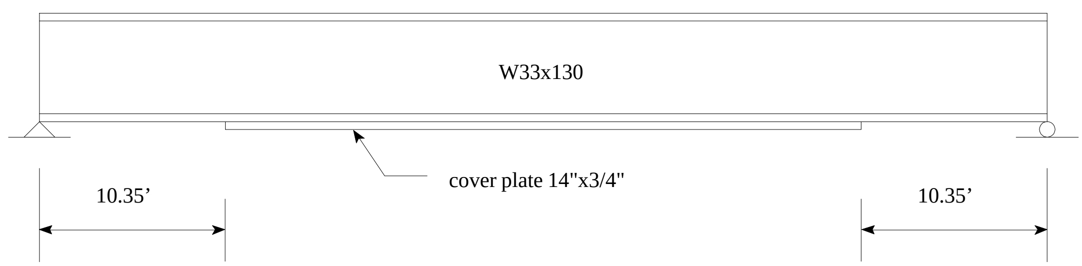


Figure 3.15: Elevation of Beam

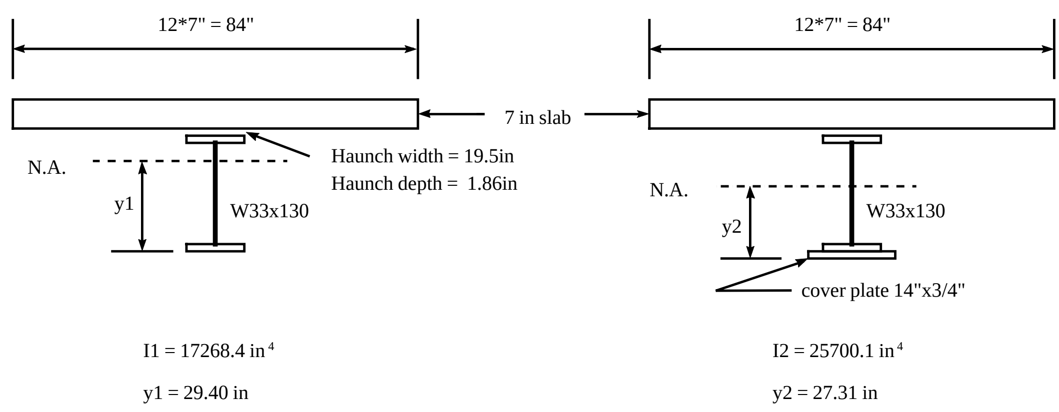


Figure 3.16: Section Properties (n = 10)

steel/concrete girder (with and without the cover plate) are computed with $n = E_s/E_c = 10$. The section properties are shown in Figure 3.16.

The bridge is subjected to dead and live external loading. The design dead load is due to a concrete slab that is 7 inches thick, a steel girder, and superimposed load. The design live load consists of a 72 kips HS-20 truck, modeled as a single concentrated load moving along the girder nodes. The abbreviated input file below shows details of the WSD rule checking.

ABBREVIATED INPUT FILE

```
/* WSD code checking for deflections and stress requirements */
/* [1] Deflection checking */

   impact = 1 + 50/(length+125);
   if( -impact*max_displ_live[1][2] > (1/800)*length ) then {
       print "\n\tWarning: (LL+I) deflection exceeds 1/800 span\n";
   } else {
       print "\n\tOK : (LL+I) deflection less than 1/800 span\n";
   }

/* [2] Moment stress checking */

   if( stress1 > 0.55*my_material[3][1] ) then{
       print "\n\tWarning : moment stress without cover plate larger
                          than 0.55*Fy\n";
   } else {
       if( stress2 > 0.55*my_material[3][1] ) then{
           print "\n\tWarning : moment stress with cover plate larger
                              than 0.55*Fy\n";
       } else {
           print "\n\tOK : moment stress less than 0.55*Fy\n";
       }
   }

/* [3] Shear stress checking */

   if( shear > 0.33*my_material[3][1] ) then{
       print "\n\tWarning : shear stress larger than 0.33*Fy\n";
   } else {
       print "\n\tOK : shear stress less than 0.33*Fy\n";
   }
```

Points to note in the input are:

1. In this example, we check the analysis result with AASHTO WSD specification. The impact factor for live load is based on the formula AASHTO Eq. (3-1)

$$I = \frac{50}{L + 125}$$

in which

I = impact fraction (maximum 30 percent);

L = length in feet of the portion of the span;

2. The deflection checking is based on AASHTO Art.10.6.2, the deflection due to service live load plus impact shall not exceed 1/800 of the span.

3. The allowable stress is $0.55 \times F_y$ for tension and compression member, $0.33 \times F_y$ for shear in web.

4. The follow array elements are used in the generation of program output:

```
max_displ[1][2] = the maximum displacement of the beam.
max_mom[2][3]   = the maximum moment.
max_sh[1][2]    = the maximum shear force.
cover_mom[2][3] = the moment at where the bridge section changed.
```

A summary of the bridge response is contained in Figures 3.17 to 3.19. The following points are noted:

1. Since this is a simple-supported bridge, the maximum displacement and maximum bending moment will occur at the middle of the span. The maximum shear force will occur at the end support.

2. The final results of moment, shear and displacement are calculated according to AASHTO WSD request: Total = DL + impact * LL. The output message about the deflection checking is

```
OK : (LL+I) deflection less than 1/800 span
```

3. The stress due to bending is given by:

$$\text{Moment stress} = \frac{M \cdot y}{I}.$$

Because there are two different section properties, we have to calculate not only the maximum moment stress in the middle of the span (stress2), but also the moment stress of where the section changed (stress1). Shear stresses are given by:

$$\text{Shear stress} = \frac{V}{tw \cdot d}.$$

We assume that the shear force is carried by the girder web alone, and therefore, we only check for maximum shear at the end support. The output messages about the moment and shear stress checking are

```
OK : moment stress less than 0.55*Fy
OK : shear stress less than 0.33*Fy
```

4. The influence line diagram for the bending moment at the middle of the span is shown in Figure: 3.18. It is obtained by iteratively positioning one truck load at a finite element node, then solving for the reaction forces. A similar procedure is employed to compute the influence line of shear force at the end support – see Figure 3.19.

5. Figure 3.17 shows the distribution of bending moments due to truck loading (we note in passing that the bending moment diagram corresponds to an envelope of the moment influence lines of the truck load).

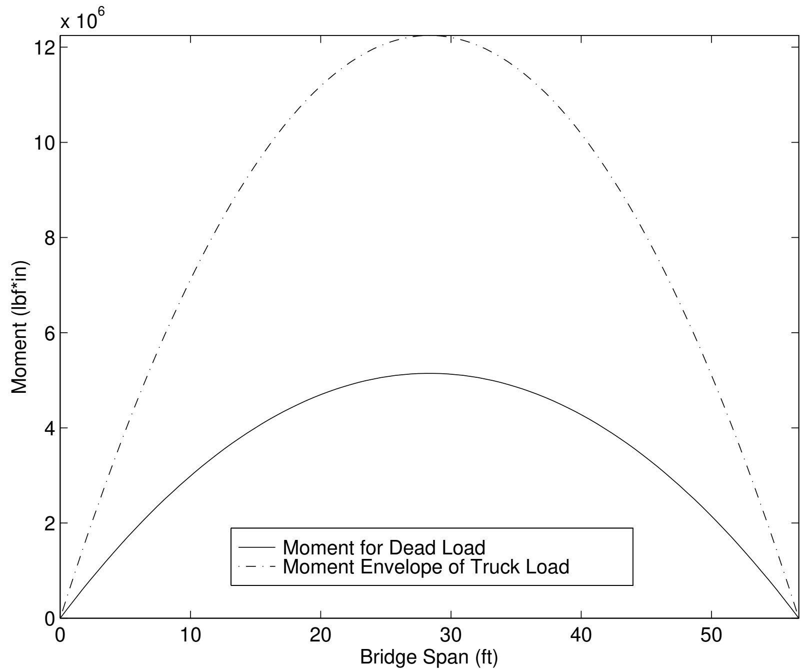


Figure 3.17: Moment Diagram of Dead Load and Truck Load

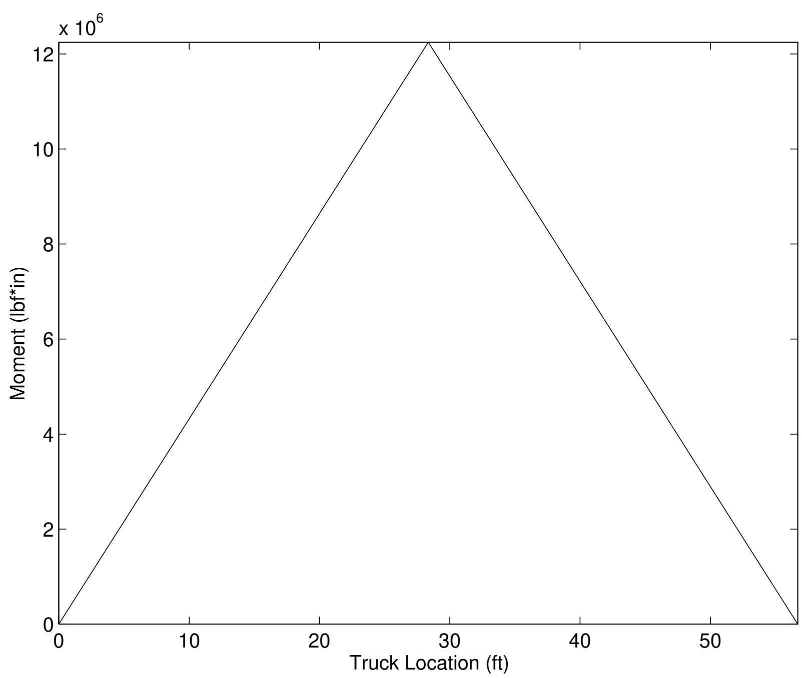


Figure 3.18: Influence Line of Moment at Mid-Span

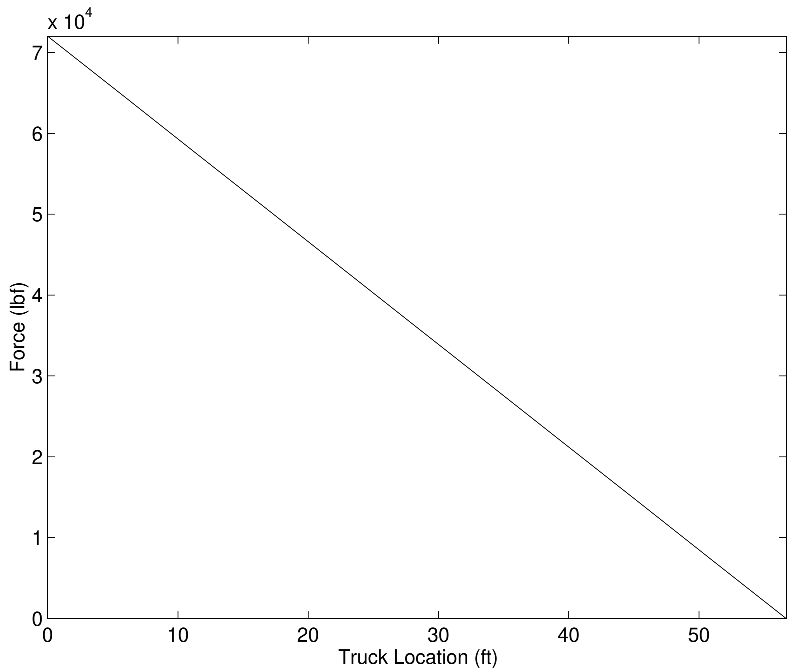


Figure 3.19: Influence Line of Shear at End Support

CHAPTER 4

Fiber Beam-Column Element

4.1 Introduction

A fundamental tenet of performance-based seismic design is the expectation of nonlinear structural behavior caused by severe earthquake ground motions. Events of this type necessitate nonlinear time-history analysis of structures. However, due to the complex interactions between the various components of real structures, the nonlinear response of a structure is not always easy to capture. State-of-the-art solution procedures use structural models that are an assembly of interconnected elements capable of modeling nonlinear material and geometric behavior.

The earliest beam-column elements incorporating nonlinear behavior assumed that plastic deformations would only occur at the beam end-points. For example, the fiber hinge model proposed by Lai et al. [26] assumes that nonlinear behavior will only occur at the element end-points. The nonlinear components of the element consist of a bunch of springs having nonlinear force-displacement relations, as shown in Figure 4.1. More accurate descriptions of the inelastic member behavior are possible with distributed nonlinearity models. Material nonlinearities can take place at any element section. The element behavior is de-

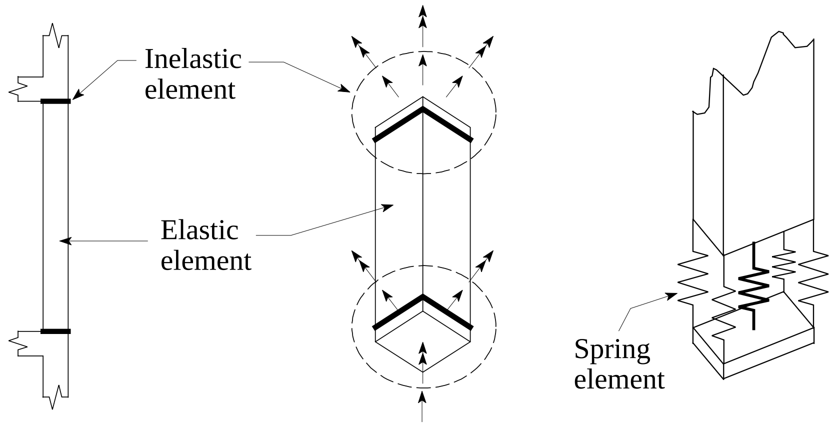


Figure 4.1: Lai's Model for Inelastic Element

rived by weighted integration of the section response. In practice, however, since the element integrals are evaluated numerically, only the behavior of selected sections at the integration points is monitored.

Filippou and Issa [16] have proposed an analysis method where elements are broken into sub elements. Each sub element is capable of describing a single effect. The interaction between these effects is achieved via a judicious combination of sub elements. The advantages of this modeling approach include the use of relatively simple nonlinear hysteretic laws at the sub element level, while not compromising the ability of the element system to mimic complex hysteretic behavior through the interaction of the different sub elements. The first elements with distributed nonlinearity were formulated with the classical stiffness method. However, as these investigators soon discovered, the main shortcoming of stiffness-based elements is their inability to describe the behavior of the member near its ultimate resistance. Many implementations are plagued with problems of numerical instability.

Mahasuverachai [27] was the first investigator to propose the use of flexibility dependent shape functions that are continuously updated during the analysis as inelastic deformations spread into the member. The flexibility approach is based on force interpolation functions within the element, and it has the benefit of permitting a more accurate description of the force distribution within the element. In fact, for those cases where no element loads are applied, the force interpolation functions satisfy element equilibrium in a strict sense. This condition holds even when material nonlinearities occur at the section level.

For structural members constructed from more than one material, (e.g., reinforced concrete members and lead-rubber base isolator elements), the most promising models for the nonlinear analysis of such members are flexibility-based fiber elements, as proposed by Filippou et al. [46]. Filippou and co-workers also present a nonlinear iterative element state determination procedure that always maintains equilibrium and compatibility within the element, and eventually converges to a state satisfying the section constitutive relations within a specified tolerance.

The purposes of this chapter are two fold. First, the formulation of a flexibility-based fiber element that incorporates both flexural and shear effects is presented. We then demonstrate the effectiveness of the element state determination procedures by computing the load-displacement relationship for a material softening bar subject to a range of axial loads.

4.2 Fiber Beam-Column Element

4.2.1 Fiber Model

Figure 4.2 shows the elevation and cross-section views of a typical fiber beam-column element. In its longitudinal direction, the element is subdivided into

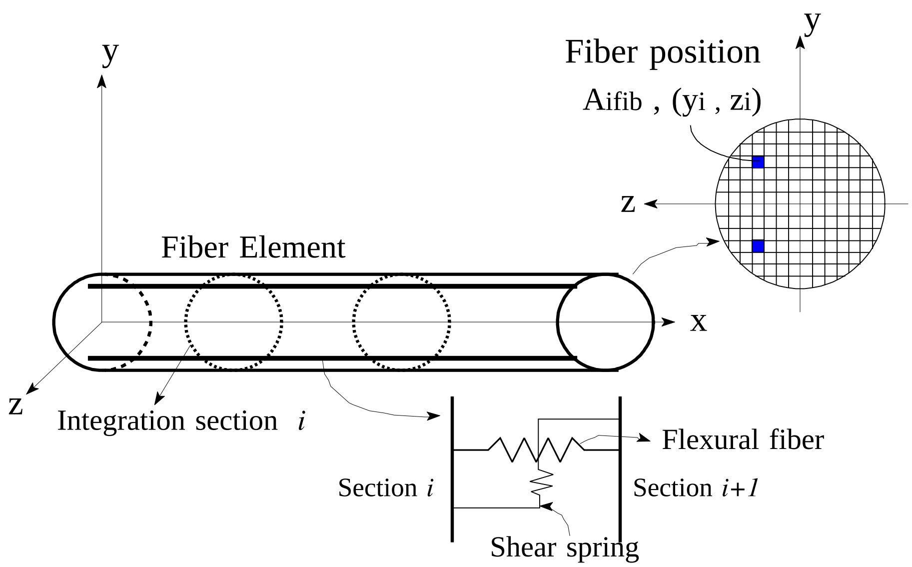


Figure 4.2: Fiber Element Model

a discrete number of fiber sections located at control points of the numerical integration scheme used in the element formulation. The cross-sectional view indicates how individual fibers of area A_{ifib} are positioned in the $y - z$ reference system.

The nonlinear constitutive relations of the overall element cross-section are derived by integration of the nonlinear stress-strain relations of the individual fibers. Each fiber follows a uniaxial stress-strain relation for a particular material.

The fiber beam-column element formulation is based on the assumption of linear geometry. Plane sections remain plane and normal to the longitudinal axis during the element deformation history. From the assumption that plane sections remain plane and normal to the longitudinal axis, we conclude that all fiber strains and stresses act parallel to this axis. Since the reference axis is fixed, the geometric centroid of the sections form a straight line that coincides with the reference axis. If an element does not comply with this hypothesis, then it should be divided into sub elements that connect the centroids of the selected sections.

4.2.2 Material Nonlinearity

In a fiber element, the constitutive relation for each integration section corresponds to the integration of individual fiber relations at that integration point. Complex nonlinear section behavior can occur even if the individual fibers are modeled with nonlinear stress-strain relations in the uniaxial direction alone. The fiber element that we have incorporated into ALADDIN has a bi-linear stress-strain relationship in uniaxial stress, and follows the kinematic hardening rule. See Figure 4.3.

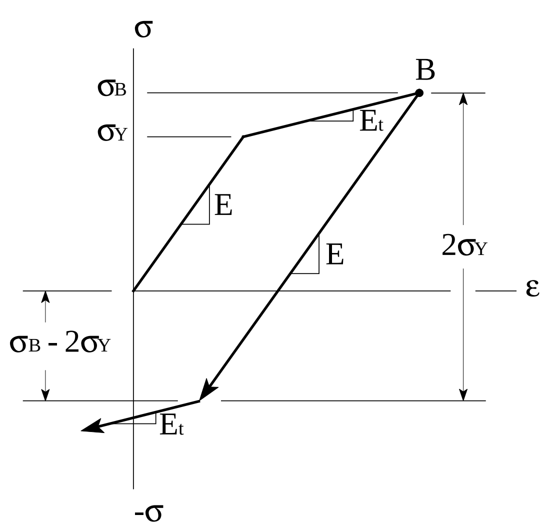


Figure 4.3: Stress-Strain Relationship for Fiber Element

The precise stress-strain relations are defined by three criteria:

1. **Yield criterion :** Yielding begins when $|\sigma|$ reaches σ_Y, either in tension or compression.

2. **Flow rule :** if the material has yielded, $d\sigma = E_t\ d\epsilon$; it the material has yet to yield or is unloading, then $d\sigma = E\ d\epsilon$.

3. **Kinematic hardening rule :** While reloading, the response will be elastic until it reaches the previous unloading point, which is point B in Figure 4.3. While loading is reversed, the yielding reappears at $\sigma_B - 2\sigma_Y$, which means that there is a total $2\sigma_Y$ elastic stress range.

4.3 Formulation of Fiber Beam-Column Element

Formulation of the fiber beam-column element requires two steps. First the generalized forces and deformations must be defined. The distribution of forces along the element and the section material and cross-sectional properties are then used to complete the tangent stiffness matrix.

4.3.1 Definition of Generalized Forces and Deformations

The generalized element forces and deformations, and the corresponding section forces and deformations are shown in Figure 4.4. Rigid body modes are not included in Figure 4.4. Since the present formulation is based on linear geometry, rigid body modes can be incorporated with a simple geometric transformation. To simplify the definition and manipulation of the relevant equations, the forces and deformations in the element and section states, stresses and strains of fibers at each section, are grouped into the following vectors:

$$\text{Element force vector} \quad \mathbf{Q} = \{\, Q_1 \quad Q_2 \quad Q_3 \quad Q_4 \quad Q_5 \,\}^T$$

$$\text{Element deformation vector} \quad \mathbf{q} = \{\, q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5 \,\}^T$$

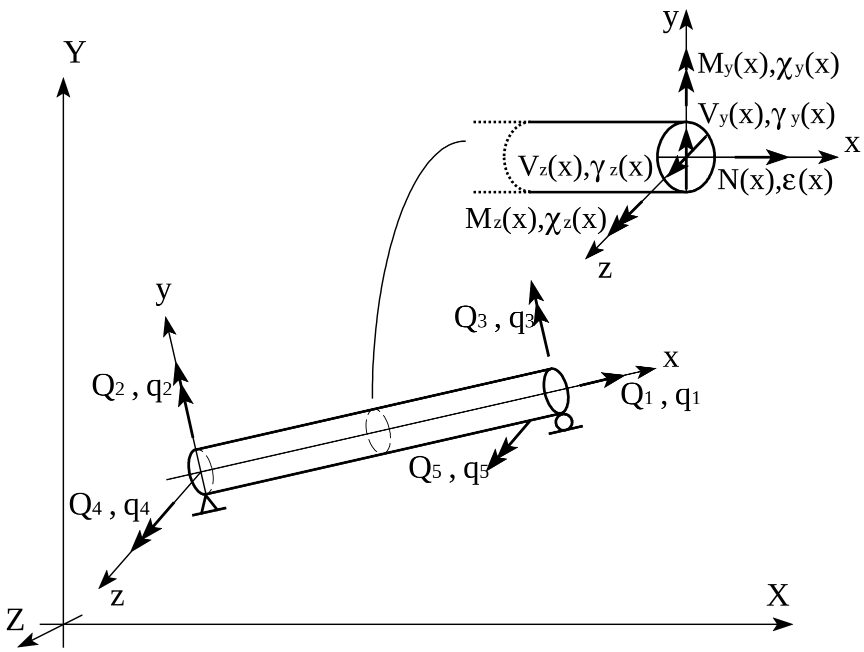


Figure 4.4: Generalized Forces and Deformations at Element and Section Level

$$\text{Section force vector} \quad \mathbf{D}(x) = \begin{Bmatrix} N_x(x) \\ M_y(x) \\ M_z(x) \\ V_y(x) \\ V_z(x) \end{Bmatrix}$$

$$\text{Section deformation vector} \quad \mathbf{d}(x) = \begin{Bmatrix} \epsilon_x(x) \\ \chi_y(x) \\ \chi_z(x) \\ \gamma_y(x) \\ \gamma_z(x) \end{Bmatrix}$$

$$\text{Fiber strain vector} \quad \boldsymbol{\epsilon}(x) = \left\{ \begin{array}{c} \epsilon_1(x, y_1, z_1) \\ \vdots \\ \epsilon_{ifib}(x, y_{ifib}, z_{ifib}) \\ \vdots \\ \epsilon_n(x, y_n, z_n) \\ \gamma_y \\ \gamma_z \end{array} \right\}$$

$$\text{Fiber stress vector} \quad \boldsymbol{\sigma}(x) = \left\{ \begin{array}{c} \sigma_1(x, y_1, z_1) \\ \vdots \\ \sigma_{ifib}(x, y_{ifib}, z_{ifib}) \\ \vdots \\ \sigma_n(x, y_n, z_n) \\ \tau_y \\ \tau_z \end{array} \right\}$$

In the fiber state, vector x describes the position of the section along the longitudinal reference axis, and vectors y_{ifib} and z_{ifib} refer to the fiber position in the cross section, as shown in Figure 4.2. The variable n refers to number of fibers in the cross section.

4.3.2 Fiber Beam-Column Element Formulation

The force distribution $\mathbf{D}(x)$ along the element is related to the element generalized force vector $\mathbf{Q}$ by the force interpolation matrix $\mathbf{b}(x)$:

$$\mathbf{D}(x) = \mathbf{b}(x) \cdot \mathbf{Q}.$$

Now lets assume that the axial force field $\mathbf{N}_x(x)$ shown in Figure 4.4 is constant. It follows that the bending moment fields $\mathbf{M}_y(x)$ and $\mathbf{M}_z(x)$ will be linear, and the shear force fields $\mathbf{V}_y(x)$ and $\mathbf{V}_z(x)$ will be constant. The force interpolation matrix is selected as:

$$\mathbf{b}(x) = \left\{ \begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{x}{L} - 1 & \frac{x}{L} & 0 & 0 \\ 0 & 0 & 0 & \frac{x}{L} - 1 & \frac{x}{L} \\ 0 & 0 & 0 & \frac{1}{L} & \frac{1}{L} \\ 0 & -\frac{1}{L} & -\frac{1}{L} & 0 & 0 \end{array} \right\}.$$

Following the hypothesis that plane sections remain plane and normal to the longitudinal axis, the fiber strain vector and the section deformation vector are related by the section compatibility matrix $\mathbf{l}(x)$ by:

$$\boldsymbol{\epsilon}(x) = \mathbf{l}(x) \cdot \mathbf{d}(x),$$

where $\mathbf{l}$(x) is a linear geometric matrix as follows

$$\mathbf{l}(x) = \left\{ \begin{array}{ccccc} 1 & z_1 & -y_1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & z_{ifib} & -y_{ifib} & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & z_n & -y_n & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array} \right\}.$$

The tangent modulus E_{ifib} of the fibers is determined from the appropriate fiber stress-strain relations, computed when the fiber strains are updated for a new given deformation increment $\Delta\mathbf{d}(x)$. The tangent modulus and the areas of all fibers are written in diagonal matrices:

$$
\mathbf{E}_{tan} = \begin{Bmatrix} E_1 & & & & & & \\ & \ddots & & & & & \\ & & E_{ifib} & & & & \\ & & & \ddots & & & \\ & & & & E_n & & \\ & & & & & G & \\ & & & & & & G \end{Bmatrix} \quad \mathbf{A} = \begin{Bmatrix} A_1 & & & & & & \\ & \ddots & & & & & \\ & & A_{ifib} & & & & \\ & & & \ddots & & & \\ & & & & A_n & & \\ & & & & & A_z^* & \\ & & & & & & A_y^* \end{Bmatrix}
$$

where G is element shear modulus, A_z^* and A_y^* are element shear area in **z** and **y** directions. The new section tangent stiffness matrix can be calculated from

$$\mathbf{k}(x) = \mathbf{l}^T(x) \cdot (\mathbf{E}_{tan} \cdot \mathbf{A}) \cdot \mathbf{l}(x).$$

The section tangent stiffness matrix $\mathbf{k}(x)$ is then inverted to obtain the new section tangent flexibility matrix $\mathbf{f}(x)$. The section resisting forces $\mathbf{D}_R(x)$ are computed by summing all of the new fiber stresses. In mathematical terms, we compute:

$$\mathbf{D}_R(x) = \mathbf{l}^T(x) \cdot \mathbf{A} \cdot \boldsymbol{\sigma}(x).$$

It is important to point out that all section matrices and vectors are computed with respect to a fixed section reference system which coincides with element **y** and **z** axes.

4.3.3 Newton-Raphson Method

The analysis of structures having nonlinear material behavior typically requires solutions to families of nonlinear equations. One of the most commonly used procedures is Newton-Raphson Iteration [13].

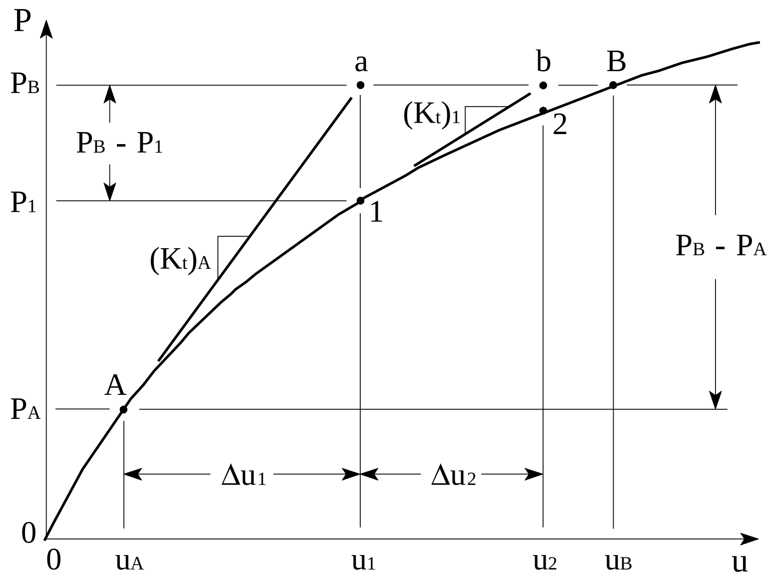


Figure 4.5: Newton-Raphson Solution Procedure

Figure 4.5 illustrates the Newton-Raphson solution procedure for a displacement u_B caused by an external force P_B. The iterative procedure begins at point A, (i.e., at coordinate $(u_A,\ P_A)$) and involves repeated solutions of $[\mathbf{K}_s]^{i-1}\{\Delta\mathbf{p}\}^i = \{\Delta\mathbf{P}_E\}^i$, where the tangent-stiffness matrix $[\mathbf{K}_s]$ is updated after each iteration. The resisting force $\{\mathbf{P}_R\}^i$ is calculated and is used to compute the unbalanced load $\{\mathbf{P}_U\}^i = \{\mathbf{P}\} - \{\mathbf{P}_R\}^i$. If the unbalanced load $\{\mathbf{P}_U\}^i$ is not within the specified tolerance, i is incremented to $i+1$ and the next iteration begins with $\{\Delta\mathbf{P}_E\}^{i+1} = \{\mathbf{P}_U\}^i$. The solution process seeks to reduce the unbalanced load $\{\mathbf{P}_U\}$, and consequently $\{\Delta\mathbf{p}\}$, to zero. Iterations continue until the unbalance loads reduce to less than a pre-specified level.

4.3.4 Summary of the Fiber Beam-Column Element State Determination

During the structure state determination, each iteration of Newton-Raphson iteration i is organized as follows [46]:

1. Solve the global system of equations and update the structure displacements.

$$\mathbf{K}_s^{i-1} \cdot \Delta\mathbf{p}^i = \Delta\mathbf{P}_E^i \tag{4.1}$$

$$\mathbf{p}^i = \mathbf{p}^{i-1} + \Delta\mathbf{p}^i \tag{4.2}$$

2. Compute the element deformation increments and update the element deformations.

$$\Delta\mathbf{q}^i = \mathbf{L}_{ele} \cdot \Delta\mathbf{p}^i \tag{4.3}$$

$$\mathbf{q}^i = \mathbf{q}^{i-1} + \Delta\mathbf{q}^i \tag{4.4}$$

   The matrix $\mathbf{L}_{ele}$ relates structural displacements with element deformations and is the combination of two transformations: let $\bar{\mathbf{q}}$ be the element displacements with rigid-body modes in the local reference system, in the first transformation the element displacements in the global reference system $\mathbf{p}$ are transformed to the displacements $\bar{\mathbf{q}}$ in the element local reference system. In the second transformation the element displacements $\bar{\mathbf{q}}$ are transformed to element deformations $\mathbf{q}$ by elimination of the rigid-body modes.

3. Start the fiber beam-column element state determination. The state determination of each element is performed in a loop j that surrounds all elements in the structure. The index of the first iteration is $j = 1$.

When $j = 1$, $\{\}^{j-1} = \{\}^0 = \{\}^{i-1}$ unless indicated otherwise, where $i-1$ corresponds to the state of the element at the end of the last $i-1$ iteration.

4. Compute the element force increments.

$$\Delta \mathbf{Q}^j = \mathbf{K}^{j-1} \cdot \Delta \mathbf{q}^j \tag{4.5}$$

When $j = 1$, $\Delta \mathbf{q}^1 = \Delta \mathbf{q}^i$.

5. Update the element forces.

$$\mathbf{Q}^j = \mathbf{Q}^{j-1} + \Delta \mathbf{Q}^j \tag{4.6}$$

6. Compute the section force increments.

$$\Delta \mathbf{D}^j(x) = \mathbf{b}(x) \cdot \Delta \mathbf{Q}^j \tag{4.7}$$

$$\mathbf{D}^j(x) = \mathbf{D}^{j-1}(x) + \Delta \mathbf{D}^j(x) \tag{4.8}$$

Steps 6 through 13 are performed at all of the control sections (i.e., integration points) in a fiber element.

7. Compute the section deformation increments.

$$\Delta \mathbf{d}^j(x) = \mathbf{f}^{j-1}(x) \cdot \Delta \mathbf{D}^j(x) + \mathbf{r}^{j-1}(x) \tag{4.9}$$

$$\mathbf{d}^j(x) = \mathbf{d}^{j-1}(x) + \Delta \mathbf{d}^j(x) \tag{4.10}$$

where $\mathbf{r}(x)$ is the residual section deformations from the previous iteration. Note that when $j = 1$, $\mathbf{r}^0(x) = 0$.

8. Compute the fiber deformation increments.

$$\Delta \boldsymbol{\epsilon}^j(x) = \mathbf{l}(x) \cdot \Delta \mathbf{d}^j(x) \tag{4.11}$$

$$\boldsymbol{\epsilon}^j(x) = \boldsymbol{\epsilon}^{j-1}(x) + \Delta \boldsymbol{\epsilon}^j(x) \tag{4.12}$$

9. Compute fiber stresses and tangent modules.

   According to the material properties and the stress-strain relationship of the fibers, the stresses $\sigma^j_{ifib}(x, y_{ifib}, z_{ifib})$ and tangent modules $E^j_{ifib}(x)$ of all fibers are computed from the stresses $\sigma^{j-1}_{ifib}(x, y_{ifib}, z_{ifib})$ and strains $\epsilon^j_{ifib}(x, y_{ifib}, z_{ifib})$ at the previous step $j-1$, and the current fiber deformation increments $\Delta\epsilon^j_{ifib}$.

10. Compute the section tangent stiffness and flexibility matrices.

$$\mathbf{k}(x) = \mathbf{l}^T(x) \cdot (\mathbf{E}_{tan} \cdot \mathbf{A}) \cdot \mathbf{l}(x)$$

   The result of the section tangent stiffness is:

$$\mathbf{k}^j(x) =$$

$$\left\{\begin{array}{ccccc} \sum\limits_{ifib=1}^{n} E_{ifib}A_{ifib} & \sum\limits_{ifib=1}^{n} E_{ifib}A_{ifib}z_{ifib} & -\sum\limits_{ifib=1}^{n} E_{ifib}A_{ifib}y_{ifib} & 0 & 0 \\ \sum\limits_{ifib=1}^{n} E_{ifib}A_{ifib}z_{ifib} & \sum\limits_{ifib=1}^{n} E_{ifib}A_{ifib}z^2_{ifib} & -\sum\limits_{ifib=1}^{n} E_{ifib}A_{ifib}y_{ifib}z_{ifib} & 0 & 0 \\ -\sum\limits_{ifib=1}^{n} E_{ifib}A_{ifib}y_{ifib} & -\sum\limits_{ifib=1}^{n} E_{ifib}A_{ifib}y_{ifib}z_{ifib} & \sum\limits_{ifib=1}^{n} E_{ifib}A_{ifib}y^2_{ifib} & 0 & 0 \\ 0 & 0 & 0 & GA^*_z & 0 \\ 0 & 0 & 0 & 0 & GA^*_y \end{array}\right\} \tag{4.13}$$

   where n is the total numbers of fibers in the cross-section, A_{ifib} is the cross-section area, and E_{ifib} is the tangent Young's modulus of the $ifib^{th}$ fiber element at section x. The latter is computed at Step 9.

   The stiffness matrix is then inverted to obtain the new section flexibility matrix.

$$\mathbf{f}^j(x) = [\mathbf{k}^j(x)]^{-1} \tag{4.14}$$

11. Compute the section resisting forces.

$$\mathbf{D}_R^j(x) = \mathbf{l}^T(x) \cdot \mathbf{A} \cdot \boldsymbol{\sigma}(x)$$

After carrying out the multiplication, the result of the section resisting forces corresponds to the summation of fiber element axial force, bending moment, and shear force contributions is:

$$\mathbf{D}_R^j(x) = \left\{ \begin{array}{c} \sum_{ifib=1}^{n} \sigma_{ifib}^j A_{ifib} \\ \sum_{ifib=1}^{n} \sigma_{ifib}^j A_{ifib} z_{ifib} \\ -\sum_{ifib=1}^{n} \sigma_{ifib}^j A_{ifib} y_{ifib} \\ \tau_y^j A_z^* \\ \tau_z^j A_y^* \end{array} \right\}. \tag{4.15}$$

where σ_{ifib}^j is the stress of the $ifib^{th}$ fiber element at section x that is computed from Step 9.

12. Compute the section unbalanced forces.

$$\mathbf{D}_U^j(x) = \mathbf{D}^j(x) - \mathbf{D}_R^j(x) \tag{4.16}$$

13. Compute the residual section deformations.

$$\mathbf{r}^j(x) = \mathbf{f}^j(x) \cdot \mathbf{D}_U^j(x) \tag{4.17}$$

14. Compute the element flexibility and stiffness matrices.

$$\begin{aligned} \mathbf{F}^j &= \int_0^L \mathbf{b}^T(x) \cdot \mathbf{f}^j(x) \cdot \mathbf{b}(x) \cdot dx \\ &= \sum_{nsec=1}^{m} w_{nsec} \cdot \mathbf{b}^T(x_{nsec}) \cdot \mathbf{f}^j(x_{nsec}) \cdot \mathbf{b}(x_{nsec}) \end{aligned} \tag{4.18}$$

$$\mathbf{K}^j = [\mathbf{F}^j]^{-1} \tag{4.19}$$

The numerical integration is carried out with the Gauss-Lobatto [44] integration scheme. m is the number of monitored sections in the beam-column element, x_{nsec} is the $\mathbf{x}$ coordinate of the section in the local reference system, and w_{nsec} is the corresponding weight factor.

15. Check for convergence of resisting forces from the element deformations.

    (a) If the element has converged, set element forces $\mathbf{Q}^i = \mathbf{Q}^j$ and stiffness $\mathbf{K}^i = \mathbf{K}^j$. Go to Step 16;

    (b) If the element forces have not converged, compute the residual element deformations

$$\begin{aligned} \mathbf{s}^j &= \int_0^L \mathbf{b}^T(x) \cdot \mathbf{r}^j(x) \cdot dx \\ &= \sum_{nsec=1}^{m} w_{nsec} \cdot \mathbf{b}^T(x_{nsec}) \cdot \mathbf{r}^j(x_{nsec}) \end{aligned} \tag{4.20}$$

    then increment j to $j+1$ and set $\Delta\mathbf{q}^{j+1} = -\mathbf{s}^j$. Repeat Steps 4 through 15 with $\Delta\mathbf{q}^{j+1}$ until element convergence is reached.

16. Compute the new structure resisting forces and structure stiffness matrix. Iteration i is complete when all of element forces have converged.

$$\mathbf{P}_R^i = \sum_{ele=1}^{n} \mathbf{L}_{ele}^T \cdot (\mathbf{Q}^i)_{ele} \tag{4.21}$$

$$\mathbf{K}_s^i = \sum_{ele=1}^{n} \mathbf{L}_{ele}^T \cdot (\mathbf{K}^i)_{ele} \cdot \mathbf{L}_{ele} \tag{4.22}$$

If convergence at the structural level is achieved, apply a new load increment, otherwise continue the Newton-Raphson iteration process.

4.4 Numerical Examples

4.4.1 Material Softening Composite Bar

Before implementing the fiber element in ALADDIN, we want to test the algorithm defined by equations 4.1 to 4.22 with the force-displacement computation for a material softening composite bar subject to axial loads. This example is inspired from Appendix C of Filippou et al. [46]. It is important to bear in mind that while this problem looks trivial, standard stiffness-based elements experience difficulty with this problem because assumed distributions of force-displacement deviate significantly from actual displacements, especially in the post-yielding region. Hence, standard stiffness-based approaches to this problem have difficulty computing a displacement that satisfies equilibrium of the section forces.

The composite bar is constructed from two materials – one is always elastic, the second has softening material behavior after yielding. The section dimensions and material properties are shown in Figure 4.6. The axial load versus time step is shown in Figure 4.7. The following abbreviated script of code shows the essential details of the solution algorithm for the two-element softening bar problem:

ABBREVIATED INPUT FILE

```
   L = [-1 , 1];    /* transformation matrix Lele */
   bx = 1;          /* force interpolation matrices b(x) */

/* assemble initial structure tangent stiffness matrix BigK */
   BigK = [ Ks1+Ks2, -Ks2;  -Ks2, Ks2 ];

/* increase displacement, structure determination */

   for ( step=1; step<=total_step ; step=step+1 ) {

      p = p + d_p;
```

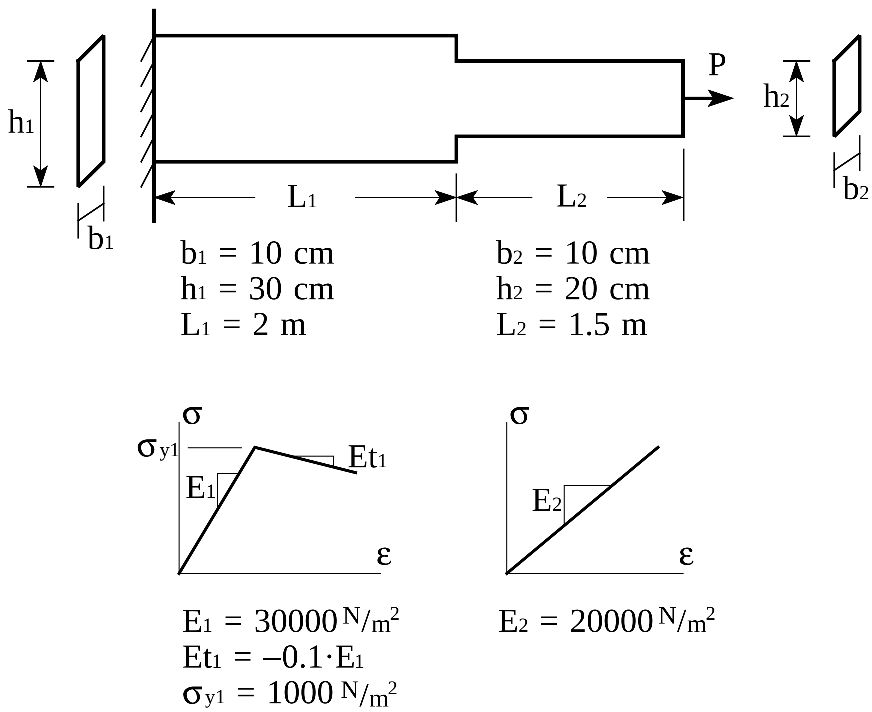


Figure 4.6: Composite Bar under Axial Load

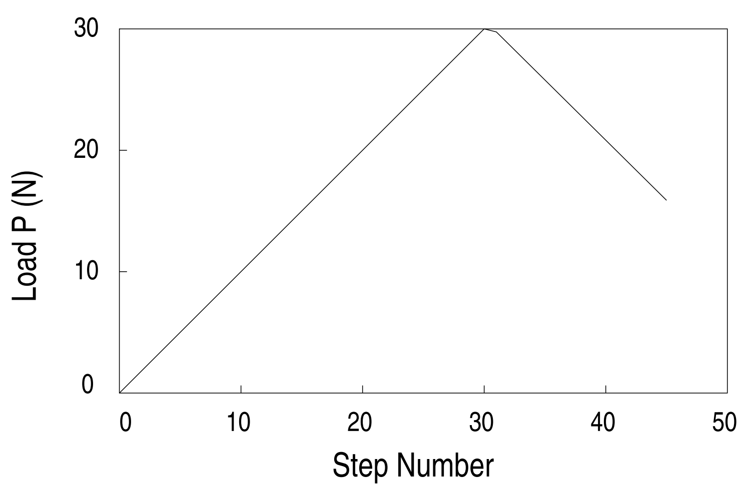


Figure 4.7: Axial Load

```
/* state determination for each element */

   for( ele=1;ele<=2;ele=ele+1 ) {

   ...... details about retrieving data from (j-1) ......
      q = q + d_q;

   /* element converge, j */

      while( abs(DUx) > 0.00001 N ){

         d_Q = K*d_q;         /* element force increment       */
         Q = Q + d_Q;         /* update element force          */
         d_Dx = bx*d_Q;       /* section force increment       */
         d_dx = rx + fx*d_Dx; /* section deformation increment */
         Dx = Dx + d_Dx;      /* update section force          */
         dx = dx + d_dx;      /* update section deformation    */

         /* get new section tangent flexibility f(x) */
         /* and section resisting force DR(x)        */
         ...... details removed ......

         DUx = Dx - DRx;  /* section unbalanced force */
         rx  = fx*DUx;    /* section residual deformation */
         F = bx*fx*bx;    /* new element flexibility */
         K = 1/F;         /* new element stiffness */
         s = bx*rx;       /* element residual deformation */
         d_q = -s;
      }                   /* end of while loop, j */

      ...... details of updating data at loop j ......

      PRe[ele][1] = Q;    /* element resisting force */
   }

/* assemble structure resistant force */

   PR[1][1]=PRe[1][1]-PRe[2][1];
   PR[2][1]=PRe[2][1];

...... details of storing response removed ......
...... details of adjusting after yielding removed ......

}  /* end of for loop step */
```

Some points to note in the input file are:

1. The body of the `while()` loop is a step-by-step implementation of equa-

tions 4.1 through 4.22. In fact, a user can write down the whole algorithm formulation in the input file, and test it even before a finite element is coded for ALADDIN.

2. This input file contains matrices defined with units, and a well defined sequence of matrix operations. Physical units are carried through every step of the calculation procedure. While the addition of units results in some computational overhead, and slow down the speed, the verification of consistent units provides a helpful check in the identification of errors.

The final results are plotted in Figure 4.8, 4.9 and 4.10. We can see that after the softening material starts to yield, the resistant force drops with further increases in the bar elongation.

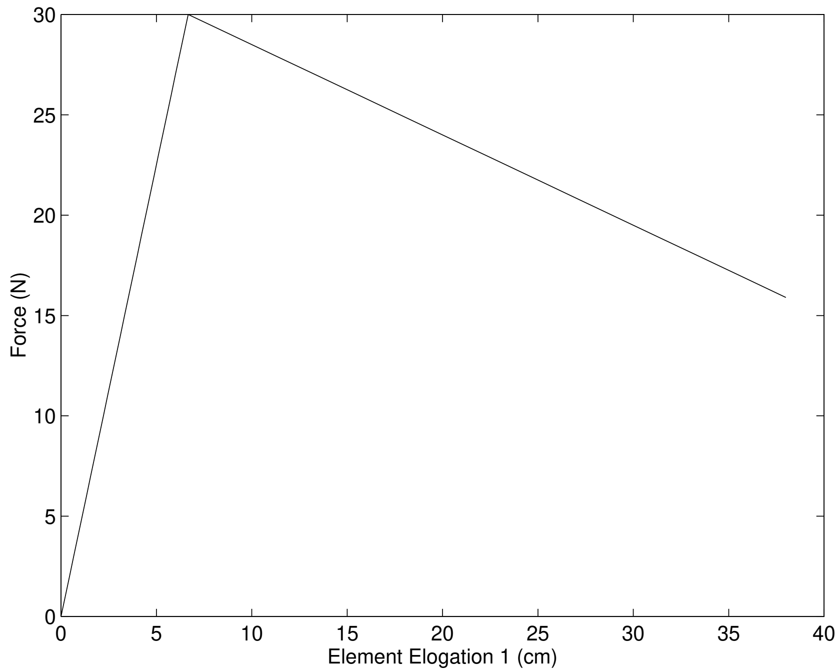


Figure 4.8: Force-Deformation History of Material Softening Element

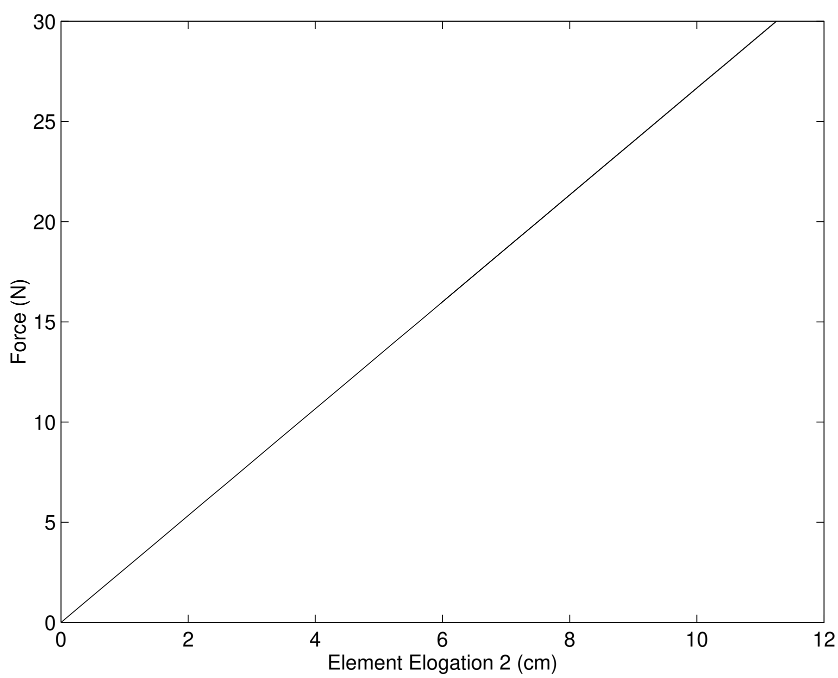


Figure 4.9: Force-Deformation History of Elastic Element

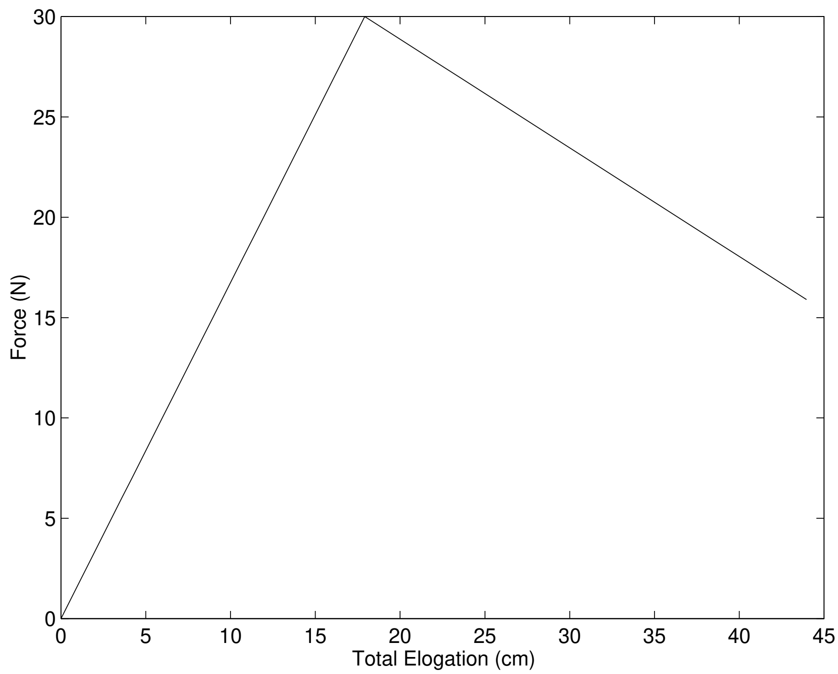


Figure 4.10: Force-deformation history of composite bar

Finite Element Implementation : Now that we know the algorithm works well on a computationally difficult problem, the next step is to implement the fiber element and the iterative solution procedure in ALADDIN's finite element library. The input statements for the whole solution procedure will be reduced to:

```
for(step=1 ; step <= total_step ; step=step+1){

   displ = displ + dp;        /* get structure displacement    */
   ElmtStateDet(dp);          /* element state determination   */
   PR = InternalLoad(displ);  /* get structure resistant force */
   UpdateResponse();          /* update element information    */

}
```

The function `ElmtStateDet()` takes care of the Step 2 to 15 in the algorithm summary, and it has one matrix argument containing the structure incremental displacements in global reference system. The function `InternalLoad()` calculate the structure resistant force in Step 16. The function `UpdateResponse()` saves information on the stress, strain, and material properties for all elements at the current step.

4.4.2 Cantilever Beam with Material Nonlinearity

Our second example in this chapter illustrates the behavior of a two dimensional cantilever beam constructed from a material having bi-linear behavior. The beam is subject to a monotonically increasing point load at its free end. The section dimension and material properties are shown in Figure 4.11.

We use the fiber element from the ALADDIN's element library to solve this problem (see APPENDIX A for details about the fiber element). The cantilever is modeled with 10 `FIBER_2D` elements, each element contains 40 fibers and is cut into 5 integration sections. The mesh is shown in Figure 4.12 The following abbreviated input file shows the for-loop of the loading process.

ABBREVIATED INPUT FILE

```
/* Setup the response matrices */

   total_step = 60;
   tip_response=ColumnUnits(Zero([total_step+1,3]),[lbf,in,rad]);
   displacement=ColumnUnits(Zero([total_node,6]),[in]);
   rotation    =ColumnUnits(Zero([total_node,6]),[rad]);
   curvature   =ColumnUnits(Zero([total_elmt,6]),[rad/in]);

   flag = 0;  index = 1;

   for( step=1 ; step <= total_step ; step=step+1 ) {

   /* Add incremental nodal load at each step */

      dPk   = [ 0 lbf, 1 lbf, 0 lbf*in ];
      NodeLoad( total_node, dPk );
      P_new = ExternalLoad();
      dP    = P_new - P_old;
      P_old = P_new;

   /* Newton-Raphson Iteration */

      while( L2Norm(dP) > 0.001 ) {

         dp  = Solve( Ks, dP );
         displ = displ + dp;

         ElmtStateDet( dp );
```

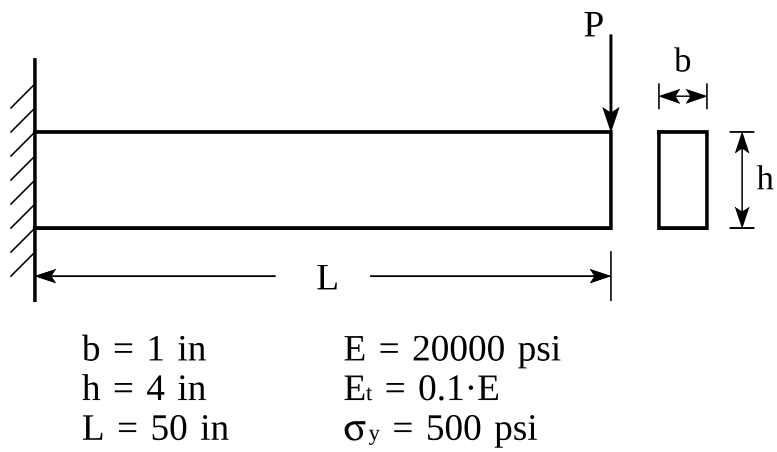


Figure 4.11: Cantilever Beam Subjected to Incremental Tip Load

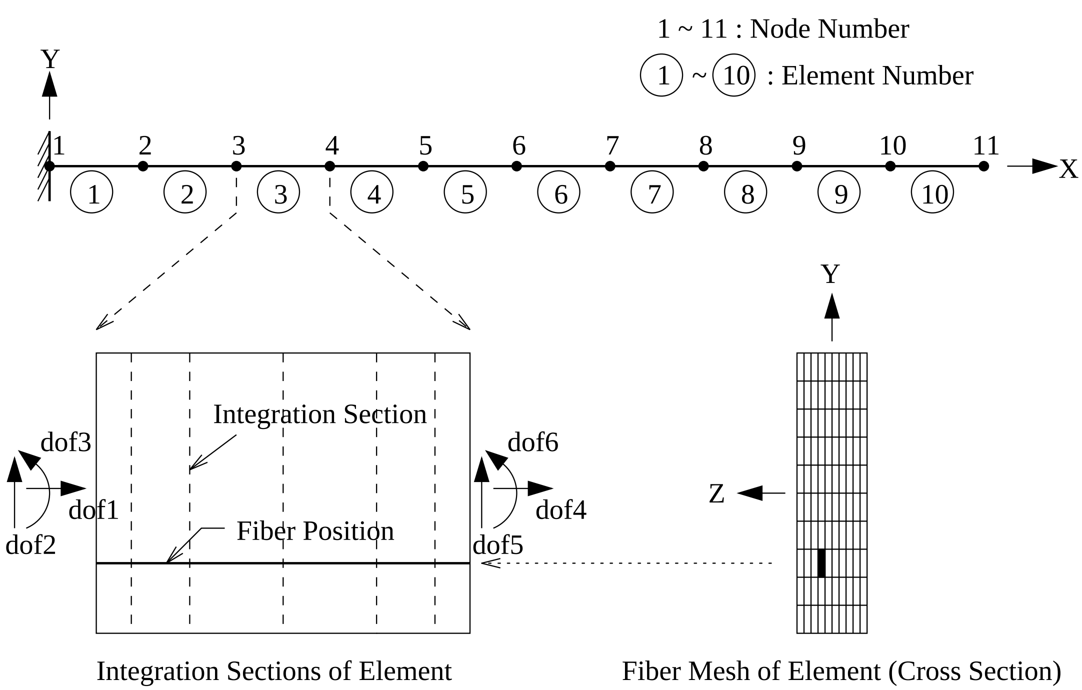


Figure 4.12: Element Mesh of Cantilever Beam

```
      Ks = Stiff();               /*Compute new stiffness   */
      PR = InternalLoad( displ ); /*Compute internal load   */
      dP = P_new - PR;            /*Compute unbalanced force*/

   }  /* end of while loop, dP converges */

/* Update element history data and save tip response */

   UpdateResponse();

/* store the tip response */

   tip_response[step+1][1] = P_new[max_dof][1];
   tip_response[step+1][2] = displ[max_dof][1];
   tip_response[step+1][3] = displ[max_dof+1][1];

/* store all element results every 10 steps */

   flag = flag+1;
   if( flag == 10 ) {
      for( node=2 ; node<=total_node ; node=node+1 ) {
         node_displacement = GetDispl([node],displ);
         displacement[node][index] = node_displacement[1][2];
         rotation[node][index] = node_displacement[1][3];
      }
      for(elmt_no=1 ; elmt_no<=total_elmt ; elmt_no=elmt_no+1){
         curvature[elmt_no][index]=(rotation[elmt_no+1][index]
                             - rotation[elmt_no][index])/(5 in);
      }
      flag = 0;  index = index+1;
   }

} /* end of load step */
```

Points to note in input are:

1. A matrix `tip_response` stores the applied external load, tip displacement, and tip rotation for each step, including the initial non-loaded status. A matrix `displacement` stores the nodal displacement every 10 steps. A matrix `rotation` stores the nodal rotation every 10 steps.

2. We also allocate memory for a matrix `curvature` which holds the element curvatures every 10 steps. We can easily calculate the curvature for the

element i by

$$\kappa_i = \frac{\theta_{i+1} - \theta_i}{L_i}.$$

because curvature along an element is constant. Here L_i is the length of element i, θ_i and θ_{i+1} are the end rotations, and κ_i is the curvature for the fiber element.

3. The nonlinear analysis method we are using here is Newton-Raphson solution process as presented in Section 4.3.3. We call the built-in function `ElmtStateDet()` to perform all element state determinations, as formulated in the previous section. Finally, the built-in function `UpdateResponse()` updates the element stress, strain, and material information after each load step finished.

The final results are plotted in Figures 4.13 through 4.16. Our observations and conclusions of this analysis are:

1. According to the equations:

$$\sigma = \frac{M \cdot y}{I} \quad \text{and} \quad M = P \cdot L$$

first yielding of the fibers occurs in the outer layers of the beam at the fixed end when the tip load reaches $26.7lbf$. The coordinates of first yield are $x = 0in$ and $y = \pm 2in$. However, the element is meshed so that the y coordinate of the outer fiber is $1.8in$, therefore the extreme fiber first yields at the tip load of $29.3lbf$.

2. Figures 4.13 and 4.14 show the cantilever tip deflection and rotation versus applied tip load. Both curves are smooth even though the material behavior

is bi-linear. This behavior can be attributed to the gradual spread of fiber yielding and from the exterior fibers towards the beam centroid.

3. Figure 4.15 shows that the beam deflection grows rapidly after yielding.

4. In Figure 4.16 we easily see which elements are elastic, which elements are partially plastic, and which elements are totally plastic based on the curvature change.

5. This fiber element includes effects of shear deformation. When $P = 20lbf$, for example, the calculated tip deflection is $7.85489in$. The tip load considerations indicate that the theoretical exact solution for the tip loaded cantilever beam [19] is

$$v_b = \frac{PL^3}{3EI} = 7.8125in, \quad v_s = \frac{f_s PL}{GA} = 0.0375in, \quad v_t = v_b + v_s = 7.85in$$

where f_s is the shear factor. If the relative shear deformation effects are not considered (like in the FRAME_2D element), the relative numerical error will be

$$err = \frac{v_s}{v_t} = 0.478\%.$$

The relative numerical error for this example is

$$err = \frac{|\delta - v_t|}{v_t} = 0.0623\%$$

which is much less than the traditional plane frame element. In this example the numerical error of FRAME_2D is small because the beam is slender (i.e., $h/L < 1/10$). Shear deformation would make a much larger contribution to the overall displacement for deep beams with large h/L ratios.

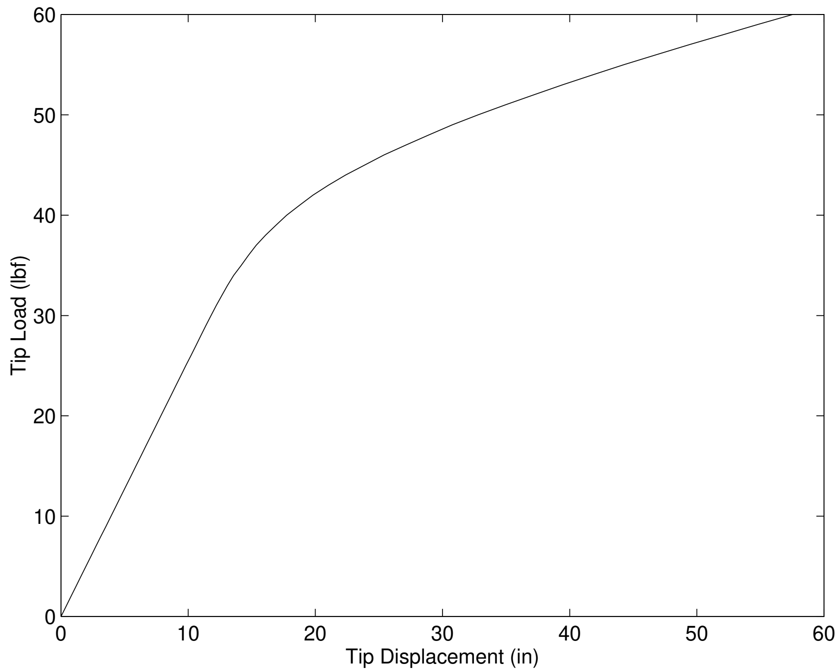


Figure 4.13: Tip Deflection Response

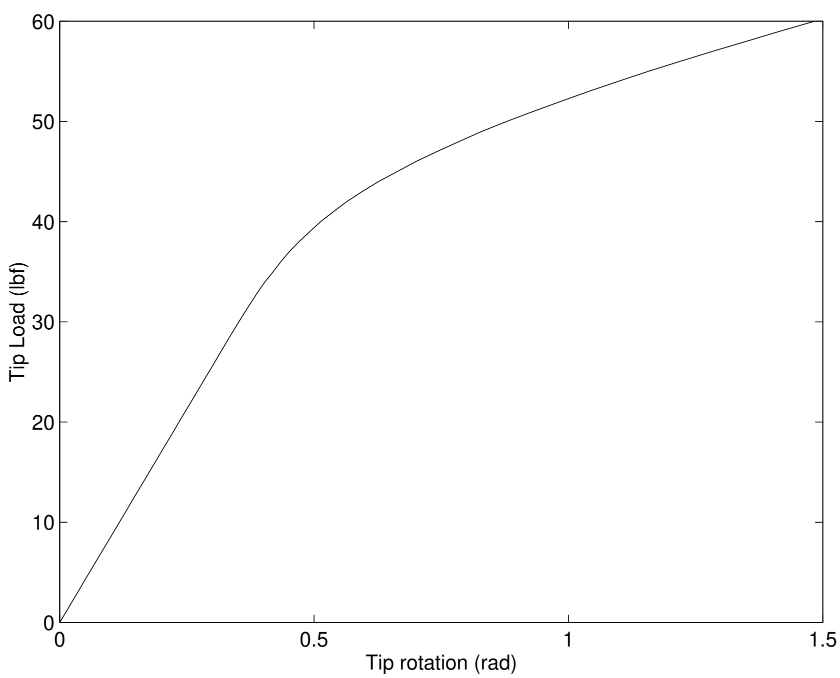


Figure 4.14: Tip Rotation Response

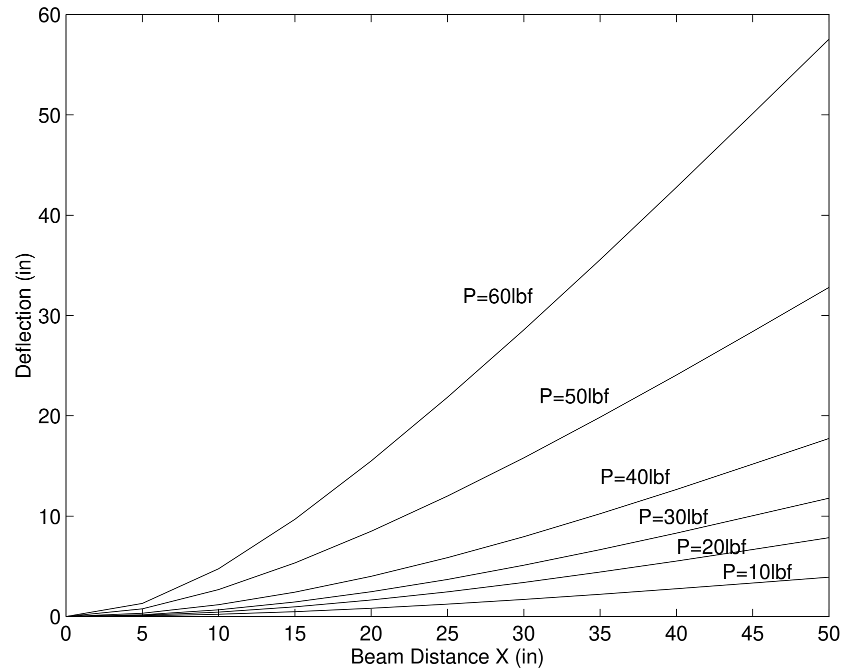


Figure 4.15: Nodal Deflection Along the Cantilever Beam

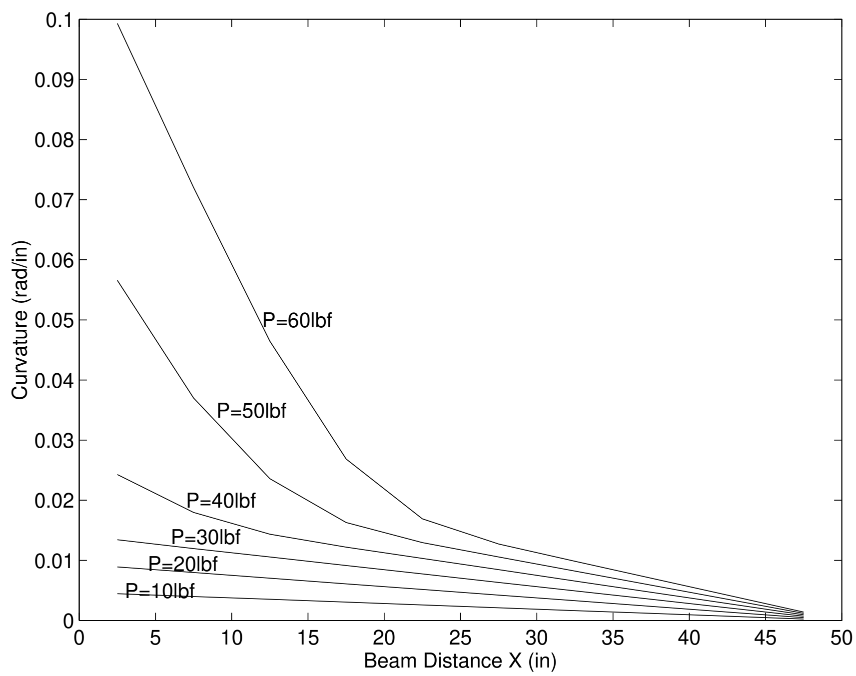


Figure 4.16: Element Curvature Along the Cantilever Beam

CHAPTER 5

Procedures for Dynamic Analysis of Structures

5.1 Solution Methods for Dynamic Analysis of Structures

The purpose of this chapter is to show how ALADDIN and the fiber element procedures developed in Chapter 4 can be applied to the dynamic time-history analysis of multi-degree of freedom structural systems. We will assume that the behavior of these systems is governed by solutions to the family of matrix equations:

$$\mathbf{M}\ddot{X}(t) + \mathbf{C}\dot{X}(t) + \mathbf{K}(X(t))X(t) = \mathbf{P}(t). \tag{5.1}$$

Here $\mathbf{M}$, $\mathbf{C}$, and $\mathbf{K}$ are $(n \times n)$ mass, damping, and stiffness matrices, respectively. $\mathbf{P}(t)$, $\mathtt{X}(t)$, $\dot{\mathtt{X}}(t)$, and $\ddot{X}(t)$ are $(n \times 1)$ external load, displacement, velocity, and acceleration vectors at time `t`. The notation $\mathbf{K}(X(t))$ indicates that the stiffness matrix will be a function of the system displacements.

Because analytical solutions to equation 5.1 are intractable for all but the simplest systems and loading conditions, in practice, numerical solution procedures must be relied upon for time-history computations. Three of these methods are presented in the following subsections.

5.1.1 Modal Analysis

Let $\mathbf{\Phi}$ be a $(n \times p)$ nonsingular matrix $(p \leq n)$, and Y(t) be a $(p \times 1)$ matrix of time-varying generalized displacements. The objective of the method of modal analysis is to find a transformation

$$X(t) = \mathbf{\Phi} Y(t) \tag{5.2}$$

that will simplify the direct integration of equations (5.1). The modal equations are obtained by substituting equations (5.2) into (5.1), and then pre-multiplying (5.1) by Φ^T. The result is:

$$\mathbf{\Phi^T M \Phi} \ddot{Y}(t) + \mathbf{\Phi^T C \Phi} \dot{Y}(t) + \mathbf{\Phi^T K \Phi} Y(t) = \mathbf{\Phi^T} P(t). \tag{5.3}$$

The notation in (5.3) may be simplified by defining the generalized mass matrix as $\mathbf{M}^* = \mathbf{\Phi^T M \Phi}$, the generalized damping matrix as $\mathbf{C}^* = \mathbf{\Phi^T C \Phi}$, the generalized stiffness matrix as $\mathbf{K}^* = \mathbf{\Phi^T K \Phi}$, and the generalized load matrix as $\mathbf{P}^*(t) = \mathbf{\Phi^T P}(t)$. Substituting these definitions into equation (5.3) gives

$$\mathbf{M}^* \ddot{Y}(t) + \mathbf{C}^* \dot{Y}(t) + \mathbf{K}^* Y(t) = \mathbf{P}^*(t). \tag{5.4}$$

The transformation matrix $\mathbf{\Phi}$ is deemed effective when the bandwidth of matrices in (5.4) is much smaller than in equations (5.1). From a theoretical viewpoint, there may be many transformation matrices $\mathbf{\Phi}$ which will achieve this objective – a judicious choice of transformation matrix will work much better than many other transformation matrices, however.

Example : To see how the method of modal analysis works in practice, consider the free vibration response of an undamped system

$$\mathbf{M} \ddot{X}(t) + \mathbf{K} X(t) = 0, \tag{5.5}$$

where $\mathbf{M}$ and $\mathbf{K}$ are $(n \times n)$ mass and stiffness matrices. We postulate that the time-history response of (5.5) may be approximated by a linear sum of p harmonic solutions

$$X(t) = \sum_{i=1}^{p} \phi_i \cdot y_i(t) = \sum_{i=1}^{p} \phi_i \cdot A_i \cdot \sin(w_i t + \beta_i) = \mathbf{\Phi} \cdot Y(t), \tag{5.6}$$

where $y_i(t)$ is the i^{th} component of $Y(t)$, and ϕ_i is the i^{th} column of $\mathbf{\Phi}$. The amplitude and phase angle for the i^{th} mode are given by A_i and β_i, respectively – both quantities may be determined from the initial conditions of the motion. We solve the symmetric eigenvalue problem

$$\mathbf{K\Phi} = \mathbf{M\Phi}\Lambda \tag{5.7}$$

for $\mathbf{\Phi}$ and Λ is a $(p \times p)$ diagonal matrix of eigenvalues

$$\Lambda = diag(\lambda_1, \lambda_2, \cdots, \lambda_p) = diag(w_1^2, w_2^2, \cdots, w_p^2). \tag{5.8}$$

It is well known that the eigenvectors of problem (5.6) will be orthogonal to both the mass and stiffness matrices. This means that the generalized mass and stiffness matrices will have zero terms except for diagonal terms. The generalized mass matrix takes the form

$$\mathbf{M}^* = \begin{bmatrix} {m_1}^* & 0 & \ldots & 0 \\ 0 & {m_2}^* & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & {m_p}^* \end{bmatrix}, \tag{5.9}$$

and the generalized stiffness looks like

$$\mathbf{K}^* = \begin{bmatrix} {w_1}^2 {m_1}^* & 0 & \ldots & 0 \\ 0 & {w_2}^2 {m_2}^* & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & {w_n}^2 {m_p}^* \end{bmatrix}. \tag{5.10}$$

If the damping matrix, $\mathbf{C}$, is a linear combination of the mass and stiffness matrices, then the generalized damping matrix $\mathbf{C}^*$ will also be diagonal. A format that is very convenient for computation is

$$\mathbf{C}^* = \begin{bmatrix} 2\xi_1 w_1 m_1{}^* & 0 & \dots & 0 \\ 0 & 2\xi_2 w_2 m_2{}^* & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 2\xi_n w_n m_p{}^* \end{bmatrix}, \tag{5.11}$$

where ξ_i is the ratio of critical damping for the i^{th} mode of vibration.

For the undamped vibration of a linear multi-degree of freedom system the eigenvalue/vector transformation is ideal because it reduces the bandwidth of $\mathbf{M}^*$, $\mathbf{C}^*$, and $\mathbf{K}^*$ to 1. In other words, the eigenvalue/vectors transform equation (5.1) from n coupled equations into p ($p \leq n$) uncoupled single degree-of-freedom systems. The required computation is simplified because the total time-history response may now be evaluated in two (relatively simple) steps:

1. Computation of the time-history responses for each of the p single degree-of-freedom systems, followed by

2. Combination of the SDOF's responses into the time-history response of the complete structure.

A number of computational methods can be used to compute the time variation of displacements in each of the single degree of freedom systems. From a theoretical viewpoint, it can be shown that the total solution (or general solution) for a damped system is given by

$$y_i(t) = B_i(t)\sin(w_{id}t) + C_i(t)\cos(w_{id}t) \tag{5.12}$$

where $w_{id} = w_i\sqrt{1 - \xi_i^2}$ is the damped circular frequency of vibration for the i^{th} mode. The time variation in coefficients $B_i(t)$ and $C_i(t)$ is given by

$$B_i(t) = e^{-\xi w_i t}\left[\frac{\dot{y}(0) + \xi w_i y(0)}{w_d} + \frac{1}{m_i^* w_{id}}\int_0^t P(\tau)e^{\xi_i w_i \tau}\cos(w_{id}\tau)d\tau\right] \quad (5.13)$$

$$C_i(t) = e^{-\xi_i w_i t}\left[y(0) - \frac{1}{m_i^* w_{id}}\int_0^t P(\tau)e^{\xi_i w_i \tau}\sin(w_{id}\tau)d\tau\right] \quad (5.14)$$

where $y_i(0)$ and $\dot{y}_i(0)$ are the initial displacement and velocity for the i^{th} mode.

If the details of $P(\tau)$ are simple enough, then analytic solutions to (5.12) may be possible. For most practical problems (e.g., earthquake ground motions), however, numerical solutions to $B_i(t)$ and $C_i(t)$ must be relied upon.

Numerical Example : We demonstrate the method of modal analysis by computing the time-history response of a four story building structure subject to a time-varying external load applied at the roof level. Details of the shear building and external loading are shown in Figures 5.1 and 5.2.

We obtain a simplified model of the building by assuming that all of the building mass is lumped at the floor levels, that the floor beams are rigid, and that the columns are axially rigid. Together these assumptions generate a model that is commonly known as a shear-type building, where displacements at each floor level may be described by one degree-of-freedom alone. Only four degrees of freedom are needed to describe total displacements of the structure.

Details of the mass and stiffness matrices are shown on the right-hand side of Figure 5.1. From a physical point of view, element `(i,j)` of the stiffness matrix corresponds to the nodal force that must be applied to degree of freedom `j` in order to produce a unit displacement at degree of freedom `i`, and zero displacements at all other degrees of freedom. Structural damping in the shear building is ignored.

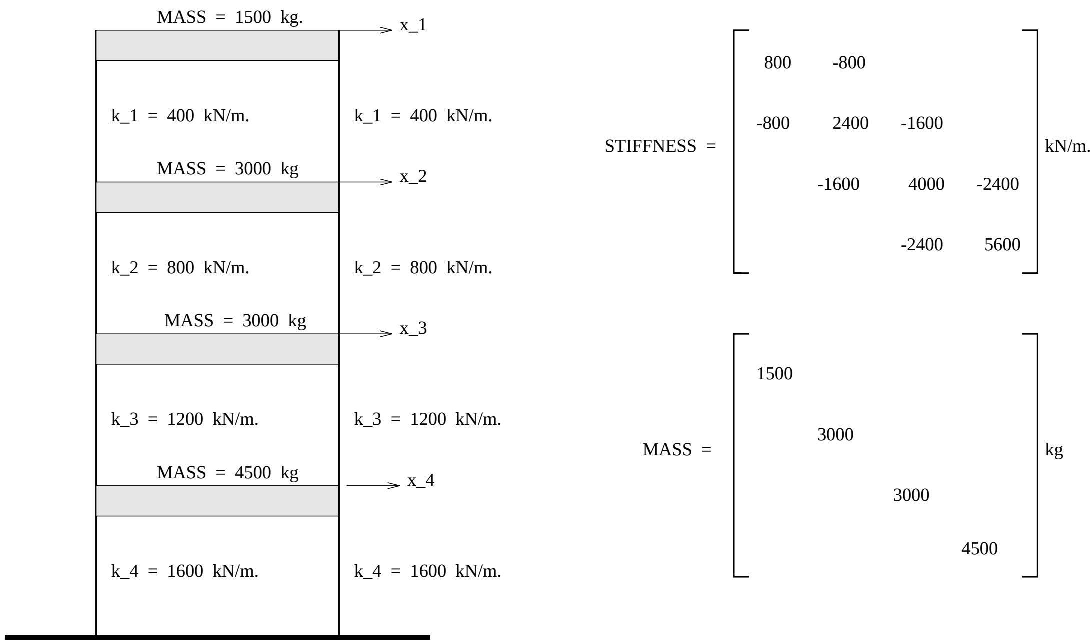


Figure 5.1: Schematic of Shear Building

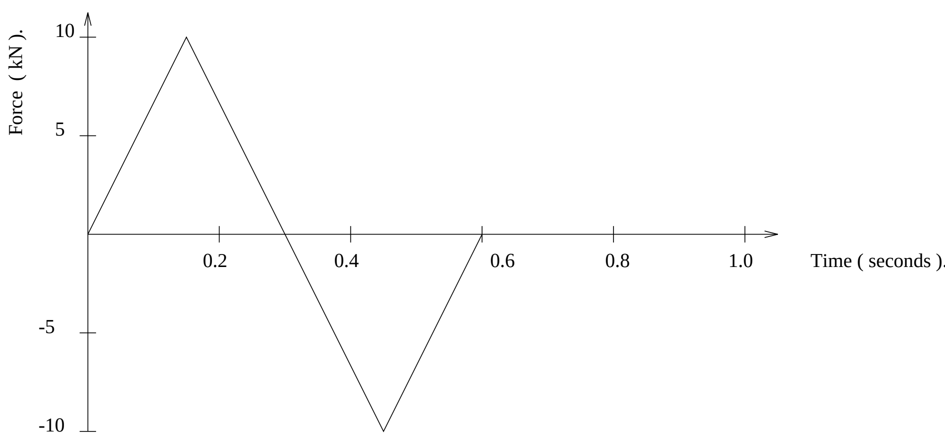


Figure 5.2: Externally Applied Force (kN) versus Time (sec)

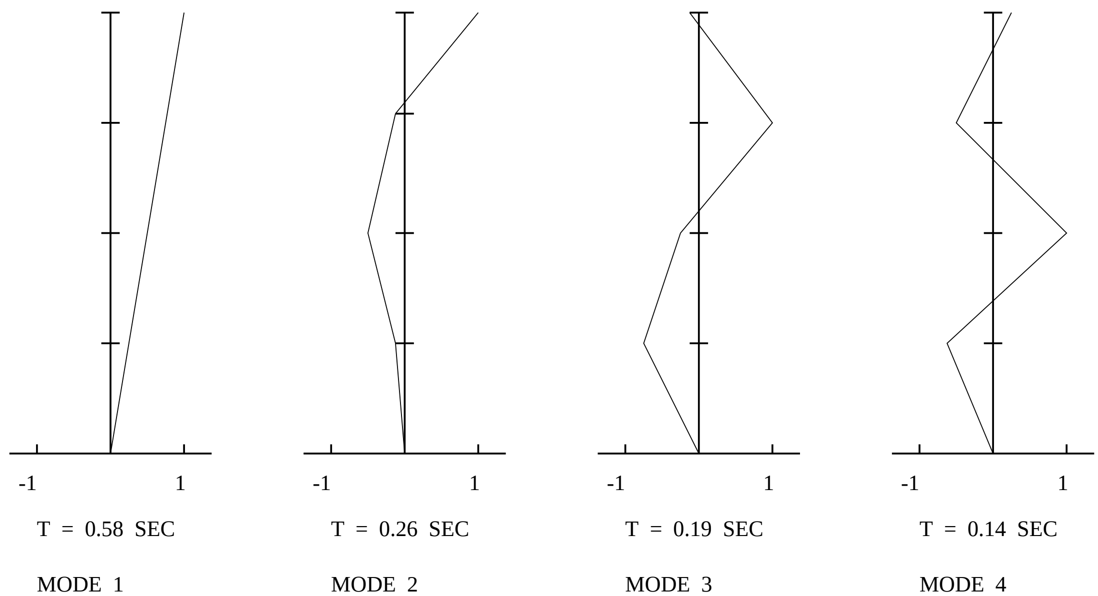


Figure 5.3: Mode Shapes and Natural Periods for Shear Building

Figure 5.3 summarizes the mode shapes and natural periods of vibration for each of the modes in the shear building. The shear building has a fundamental period of 0.5789 seconds.

Dynamic behavior of the shear building is generated by a horizontal time-varying force (see Figure 5.2 for the details) applied at the roof level degree of freedom. We have deliberately selected the time-scale of the applied force so that it has a period close to the first natural period of the structure (i.e., 0.5789 seconds versus 0.6 seconds period for the applied load).

In the input file that follows, (2×2) generalized mass and stiffness matrices are generated by first computing the (4×2) transformation matrix $\mathbf{\Phi}$ corresponding to the first two eigenvectors in the shear building. The elements of the generalized mass and stiffness matrices are zero, except for diagonal terms. Each of the decoupled equations is then solved as single degree-of-freedom system.

ABBREVIATED INPUT FILE

```
/* Compute first two eigenvalues, periods, and eigenvectors */

   no_eigen    = 2;
   eigen       = Eigen(stiff, mass, [no_eigen]);
   eigenvalue  = Eigenvalue(eigen);
   eigenvector = Eigenvector(eigen);

/* Compute generalized mass, stiffness and load matrices */

   EigenTrans = Trans(eigenvector);
   Eigenmass  = EigenTrans*mass;

   Mstar   = Eigenmass*eigenvector;
   Kstar   = EigenTrans*stiff*eigenvector;
   Pstar   = EigenTrans*eload;

/* Mode-Displacement Solution for Response of Undamped MDOF System  */

   for(i = 1; i <= nsteps; i = i + 1) {

   /* [1] : Update external load */

      time = time + dt;
      if(time <= 0.6 sec) then {
              eload[1][1] = myload[i+1][2];
      } else {
              eload[1][1] = 0.0 kN;
      }

      Pstar_new = EigenTrans*eload;
      D_Pstar   = Pstar_new - Pstar;

   /* [2] : Retrieve modal initial conditions */

      Mdisp = Eigenmass*disp;
      Mvel  = Eigenmass*vel;
      for( r=1 ; r<=no_eigen ; r=r+1 ) {
         Mdisp[r][1] = Mdisp[r][1]/Mstar[r][r];
         Mvel[r][1]  = Mvel[r][1]/Mstar[r][r];
      }


   /* [3] : Compute new generalized displacement for each SDOF system */
   /*     : Using piecewise-linear interpolation of excitation        */

      for( r=1 ; r<=no_eigen ; r=r+1 ) {
         w  = sqrt(eigenvalue[r][1]);
         wt = w*dt;

         Mdisp_new[r][1] = Mdisp[r][1]*cos(wt) + Mvel[r][1]*sin(wt)/w
```

```
                        + Pstar[r][1]/Kstar[r][r]*( 1-cos(wt) )
                        + D_Pstar[r][1]/Kstar[r][r]*( wt-sin(wt) )/wt;

        Mvel_new[r][1]  = w*( -Mdisp[r][1]*sin(wt) + Mvel[r][1]*cos(wt)/w
                        + Pstar[r][1]/Kstar[r][r]*sin(wt)
                        + D_Pstar/Kstar[r][r]*( 1-cos(wt) )/wt );
     }

  /* [4] : Update new response */

     Pstar = Pstar_new;
     disp  = eigenvector*Mdisp_new;
     vel   = eigenvector*Mvel_new;
  }
```

Some points to note in the input file are:

1. Overall behavior of the system is represented by (4×4) global mass and stiffness matrices. A (4×2) transformation matrix, Φ, is computed by solving equation (5.7) for the first two eigenvectors. It follows that the generalized mass and stiffness will be (2×2) diagonal matrices.

2. The main loop of our modal analysis computes the time-history response via piecewise-linear interpolation of the excitation for the two decoupled equations. However, as we will see in the next section it is computationally simpler to use the method of Newmark integration to solve both sets of decoupled equations together.

The building system response is summarized in Figures 5.4 through 5.6. Points to note are:

1. Figures 5.4 and 5.5 show the time-history response for the first and second modes, respectively. Notice that the amplitude of vibration for the first mode is an order of magnitude larger than for the second mode. You should also observe that after the external load finishes at time = 0.6 seconds, the

amplitude of vibration is constant within each mode, with the natural periods of vibration closely matching the eigenvalues/periods shown in Figure 5.3.

2. The combined first + second modal response is shown in Figure 5.6.

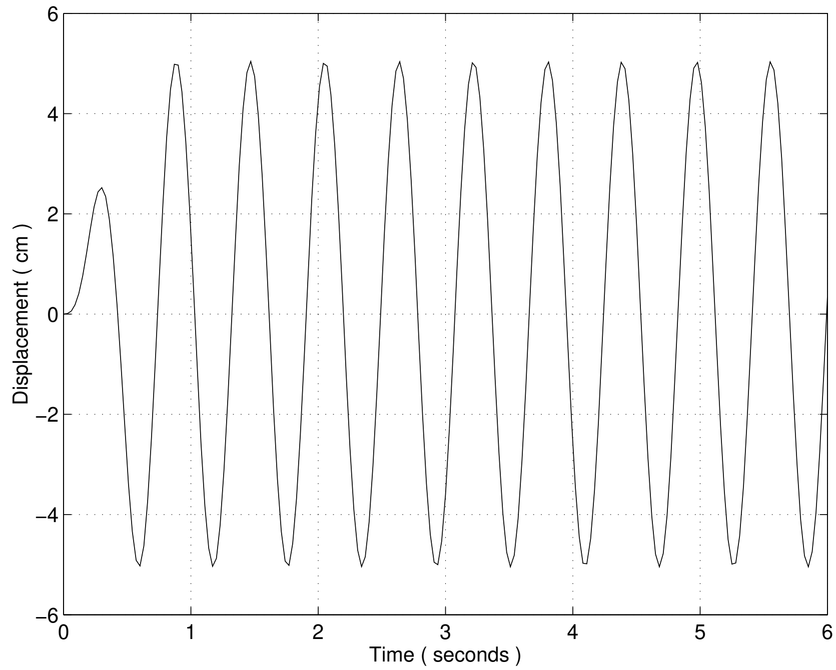


Figure 5.4: Modal Analysis : First Mode Displacement of Roof (cm) versus Time (sec)

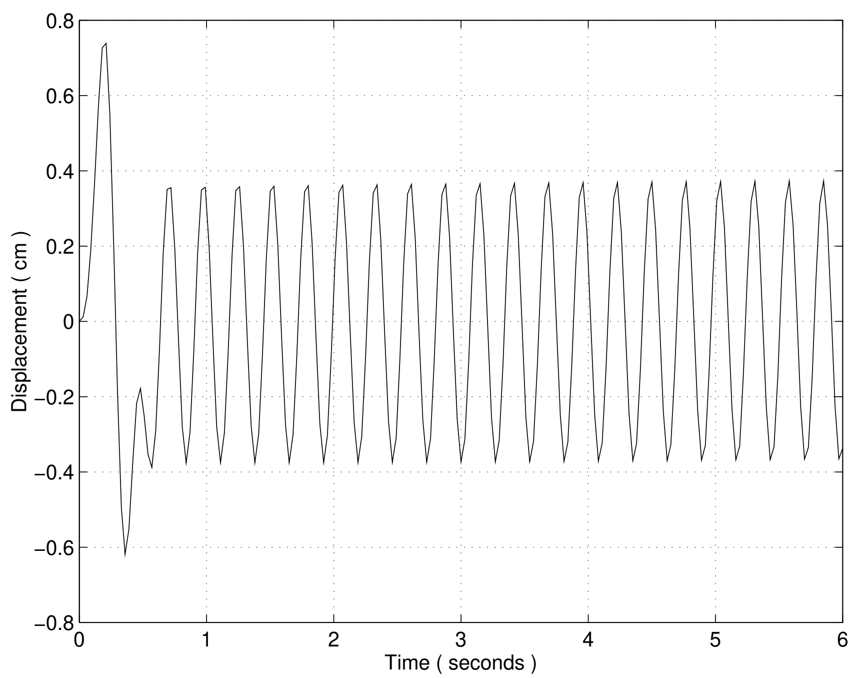


Figure 5.5: Modal Analysis : Second Mode Displacement of Roof (cm) versus Time (sec)

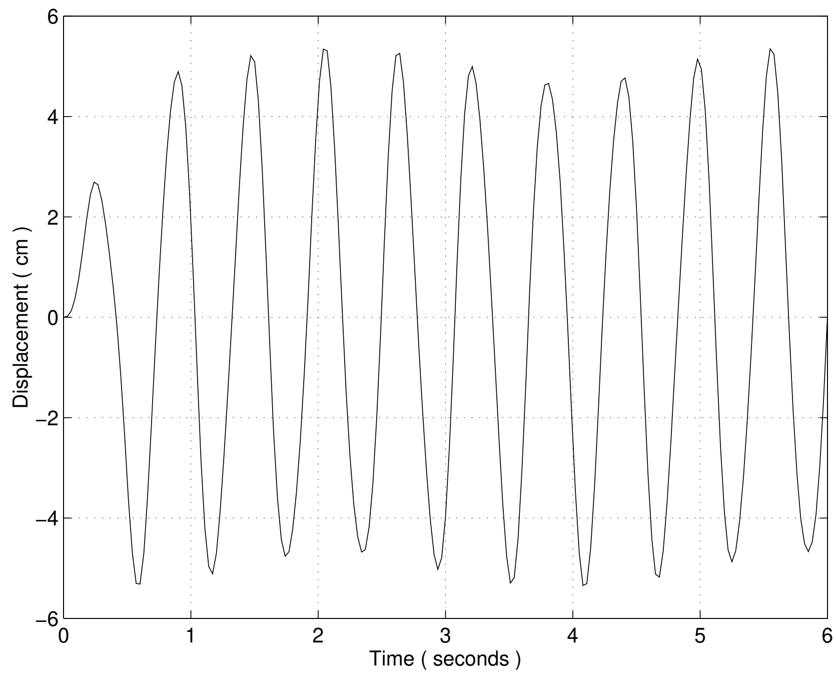


Figure 5.6: Modal Analysis : First + Second Mode Displacement of Roof (cm) versus Time (sec)

5.1.2 Newmark Algorithm Method

An important limitation of mode-superposition methods (e.g., see the calculation for the response of linear MDOF systems with proportional damping) is their inability to compute the nonlinear time-history response of structural systems. Step-by-step numerical integration procedures must be used instead.

Newmark integration methods [10] approximate the time-dependent response of linear and nonlinear second-order equations by insisting that equilibrium be satisfied only at a discrete number of points (or time steps).

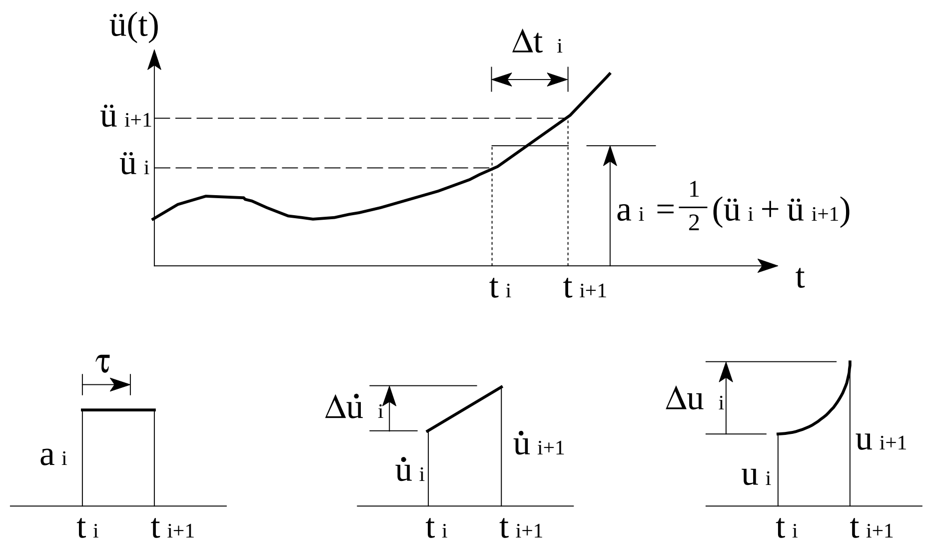


Figure 5.7: Numerical Integration Using Average Acceleration Method

If (t) and $(t + \triangle t)$ are successive time steps in the integration procedure, the two equations of equilibrium that must be satisfied are

$$\mathbf{M}\ddot{X}(t) + \mathbf{C}\dot{X}(t) + \mathbf{K}X(t) = \mathbf{P}(t), \tag{5.15}$$

$$\mathbf{M}\ddot{X}(t + \triangle t) + \mathbf{C}\dot{X}(t + \triangle t) + \mathbf{K}X(t + \triangle t) = \mathbf{P}(t + \triangle t). \tag{5.16}$$

Now let us assume that solutions to equation (5.15) are known and (5.16) needs to be solved. At each time step there are $3n$ unknowns corresponding to the displacement, velocity, and acceleration of each component of X. Since we only have n equations, the natural relationship existing between the acceleration and velocity, and velocity and displacement

$$\begin{aligned}
\dot{X}(t+\triangle t) &= \dot{X}(t) + \int_{(t)}^{(t+\triangle t)} \ddot{X}(\tau) d\tau, && (5.17)\\
X(t+\triangle t) &= X(t) + \int_{(t)}^{(t+\triangle t)} \dot{X}(\tau) d\tau, && (5.18)
\end{aligned}$$

must be enforced to reduce the number of unknowns to n. $\ddot{X}(\tau)$ is an unknown function for the acceleration across the time step. The Newmark family of integration methods assume that:

1. Acceleration within the time step behaves in a prescribed manner, and
2. The integral of acceleration across the time step may be expressed as a linear combination of accelerations at the endpoints.

Discrete counterparts to the continuous update in velocity and displacement are:

$$\begin{aligned}
\dot{X}(t+\triangle t) &= \dot{X}(t) + \triangle t\Big[(1-\gamma)\ddot{X}(t) + \gamma \ddot{X}(t+\triangle t)\Big] && (5.19)\\
X(t+\triangle t) &= X(t) + \triangle t \dot{X}(t) + \frac{\triangle t^2}{2}\Big[(1-2\beta)\ddot{X}(t) + 2\beta \ddot{X}(t+\triangle t)\Big] && (5.20)
\end{aligned}$$

with the parameters γ and β determining the accuracy and stability of the method under consideration. The equations for discrete update in velocity and displacement are substituted into equation (5.16) and rearranged to give:

$$\hat{\mathbf{M}}\ddot{X}(t+\triangle t) = \hat{\mathbf{P}}(t+\triangle t) \quad (5.21)$$

where

$$\hat{\mathbf{M}} = \mathbf{M} + \gamma \triangle t \mathbf{C} + \beta \triangle t^2 \mathbf{K} \quad (5.22)$$

and

$$
\begin{aligned}
\hat{\mathbf{P}}(t+\triangle t) \;&= \mathbf{P}(t+\triangle t) - \mathbf{C}\dot{X}(t) - \mathbf{K}X(t) - \triangle t\,[\mathbf{K}]\,\dot{X}(t) - \\
&\triangle t\Big[(1-\gamma)\mathbf{C} + \tfrac{\triangle t}{2}(1-2\beta)\mathbf{K}\Big]\ddot{\mathbf{X}}(t).
\end{aligned} \tag{5.23}
$$

It is well known that when $\gamma = 1/2$ and $\beta = 1/4$, acceleration is constant within the time step $t \in [t, t + \triangle t]$, and equal to the average of the endpoint accelerations. In such cases, approximations to the velocity and displacement will be linear and parabolic, respectively, as shown in Figure 5.7. Moreover, this discrete approximation is second order accurate and unconditionally stable [14]. When $\gamma = 1/2$ and $\beta = 1/6$, acceleration is linear within the time step $t \in [t, t + \triangle t]$, and passes through the endpoint accelerations. While the second discrete approximation is more accurate than the former method, it is only conditionally stable, and will diverge if it is applied to modal response components having periods of vibration less than 1.8 times the integration interval [12].

Numerical Example : We demonstrate the Newmark algorithm method by repeating the linear time-history computation defined in the previous example. Details of the shear building and external loading are shown in Figures 5.1 and 5.2.

An eight-part input file is needed to define the mass and stiffness matrices, external loading, and solution procedure via the method of Newmark Integration. The step-by-step details of our Newmark Algorithm are:

1. Form the stiffness matrix $\mathbf{K}$, the mass matrix $\mathbf{M}$, and the damping matrix $\mathbf{C}$. Compute the effective mass matrix $\hat{\mathbf{M}}$.

2. Initialize the displacement $X(0)$ and velocity $\dot{X}(0)$ at time 0. Backsubsti-

tute $X(0)$ and $\dot{X}(0)$ into equation (5.15), and solve for $\ddot{X}(0)$.

3. Select an integration time step $\triangle t$, and Newmark parameters γ and β.

4. Enter Main Loop of Newmark Integration.

5. Compute the effective load vector $\hat{\mathbf{P}}(t + \triangle t)$.

6. Solve equation (5.21) for acceleration $\ddot{\mathbf{X}}(t + \triangle t)$.

7. Compute $\dot{\mathbf{X}}(t + \triangle t)$ and $\mathbf{X}(t + \triangle t)$ by backsubstituting $\ddot{\mathbf{X}}(t + \triangle t)$ into the equations for discrete update in velocity and displacement.

8. Go to Step 4.

The following abbreviated input file illustrates Step 1, and the main loop of the Newmark integration, which is Steps 4 through 8.

ABBREVIATED INPUT FILE

```
/* Compute (and compute LU decomposition) effective mass */

   MASS  = mass + stiff*beta*dt*dt;
   lu    = Decompose(MASS);

/* Newmark Iteration Loop */

   for(i = 1; i <= nsteps; i = i + 1) {

    /* [1] : Update external load, and compute effective load */

       time = time + dt;
       if( time <= 0.6 sec ) then {
           eload[1][1] = myload[i+1][2];
       } else {
           eload[1][1] = 0.0 kN;
       }

       R = eload - stiff*(displ + vel*dt + accel*(dt*dt/2.0)*(1-2*beta));

    /* [2] : Compute new acceleration, velocity and displacement  */
```

```
    accel_new = Substitution(lu,R);
    vel_new   = vel   + dt*(accel*(1.0-gamma) + gamma*accel_new);
    displ_new = displ + dt*vel
              + ((1 - 2*beta)*accel + 2*beta*accel_new)*dt*dt/2;

 /* [3] : Update new response */

    accel = accel_new;
    vel   = vel_new;
    displ = displ_new;
}
```

Figure 5.8 is time-history plot of the roof level displacement. The curve is virtually identical to that computed with the modal analysis method.

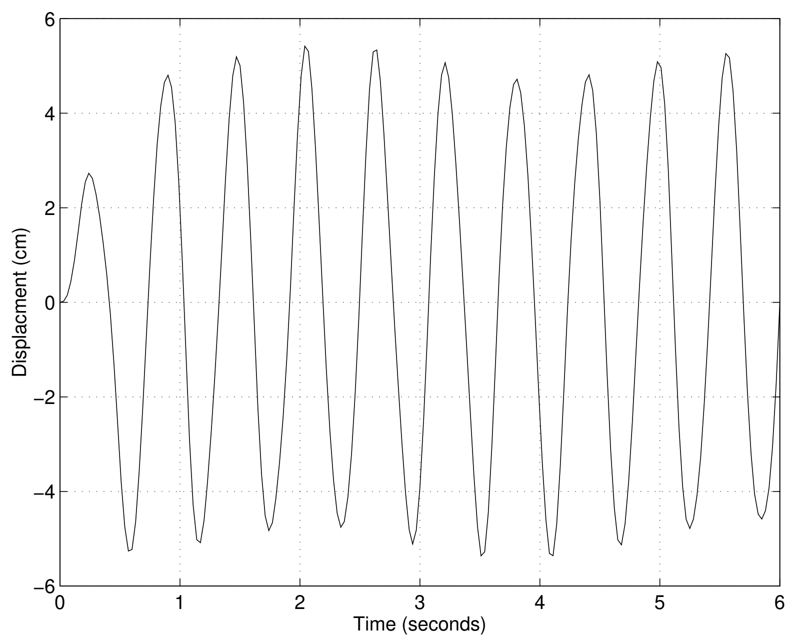


Figure 5.8: Newmark Integration : Lateral Displacement of Roof (cm) versus Time (sec)

5.1.3 Wilson-θ Method

For certain types of multi-degree-of-freedom structures, such as models of multi-story buildings idealized to have only one degree of freedom per story, the Newmark method with linear acceleration across the time steps is an effective way of computing linear and nonlinear time-history responses. For finite element idealizations of structures having more complex geometries, this method is sometimes unsatisfactory because of the very short time increment required to avoid numerical instability. Unconditionally stable methods are required instead.

A number of unconditionally stable step-by-step methods have been developed for dynamic response analysis (see references [14, 12]). One of the simplest and best of these is the Wilson-θ method.

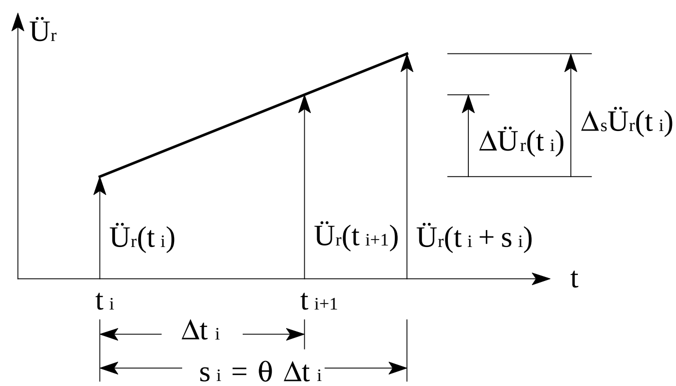


Figure 5.9: Linear Acceleration Assumption of Wilson-θ Method

The basic assumption of the Wilson-θ method is that each component $\ddot{\mathbf{U}}_r$ of the acceleration vector $\ddot{\mathbf{U}}$ varies linearly with the time over an extended time step $s_i = \theta\Delta t_i$ as indicated in Figure 5.9. The Wilson-θ method is unconditionally stable only for $\theta > 1.37$ [50, 12, 14]. The optimum value of θ is 1.420815.

Numerical Example : In this example we compute the nonlinear time-history

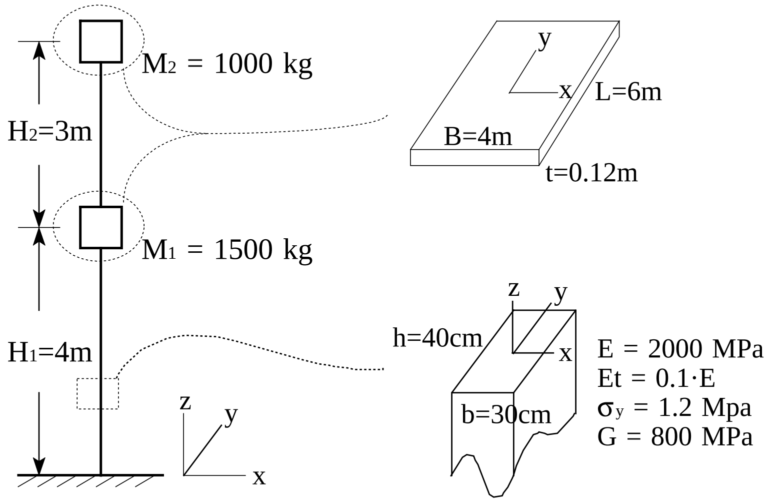
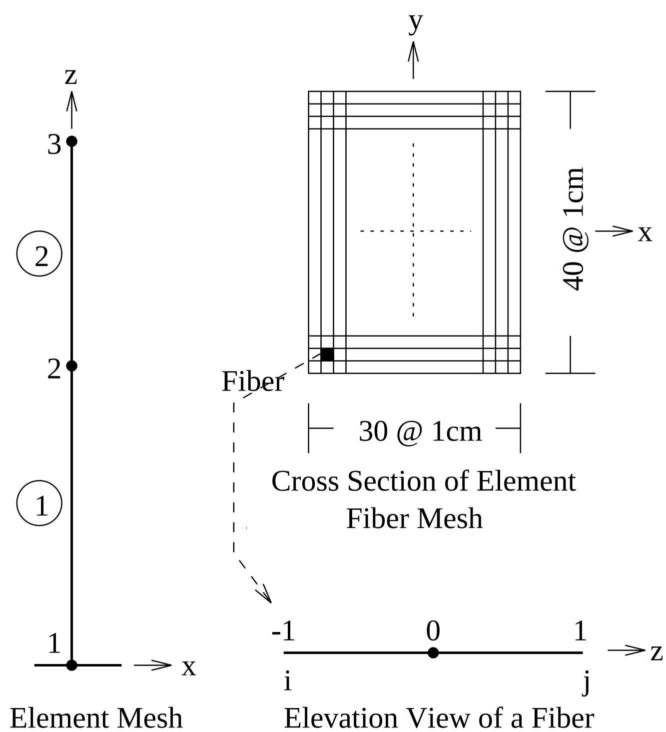


Figure 5.10: Three Dimensional Two-Story Lumped Mass Pier



i	-1.0
1	-0.94
2	-0.83
3	-0.63
4	-0.40
5	-0.14
6	0.14
7	0.40
8	0.63
9	0.82
10	0.94
j	1.0

Location of Integration Points

Figure 5.11: Element Mesh of Two-Story Lumped Mass Pier

response of a two story lumped mass pier subject to earthquake ground motions simultaneously applied in the x and y directions. The Wilson-θ algorithm is used for numerical integration of the underlying equations of equilibrium.

The pier is modeled with two FIBER_3D elements with $30 \times 40 = 1200$ fibers and 10 Gauss-Lobatto integration points for each element. The element mesh is shown in Figure 5.11. The section dimension and material properties are shown in Figure 5.10. The finite element model has 3 nodes and 2 fiber elements. The boundary conditions are full fixity at the base. Axial deformations (u_z) and torsional rotations (θ_z) in each element are likewise assumed to be zero. After the boundary conditions are applied, the model has only 8 degrees of freedom (d.o.f.). And this is in spite of the 2400 fibers used to model the pier deformations.

The total mass of the first floor is 1500 kg. The second floor has mass 1000 kg. We assume for the purposes of this analysis that each floor has the dimensions shown in the top right-hand corner of Figure 5.10. This effect includes the rotational inertias of both floors, namely:

$$
\begin{aligned}
J_x &= L^2/12, & \qquad J_y &= B^2/12, \\
M_{jx} &= M \cdot J_x, & \qquad M_{jy} &= M \cdot J_y,
\end{aligned}
$$

The lumped mass matrix of each floor in local coordinates is:

$$
mass_i = \begin{bmatrix} M_i & & & \\ & M_i & & \\ & & M_{jxi} & \\ & & & M_{jyi} \end{bmatrix}.
$$

The time-history response of the two-story lumped mass pier structure is computed for ground motion accelerograms simultaneously applied in the x and y directions. The ground motion accelerograms are ten second samples extracted

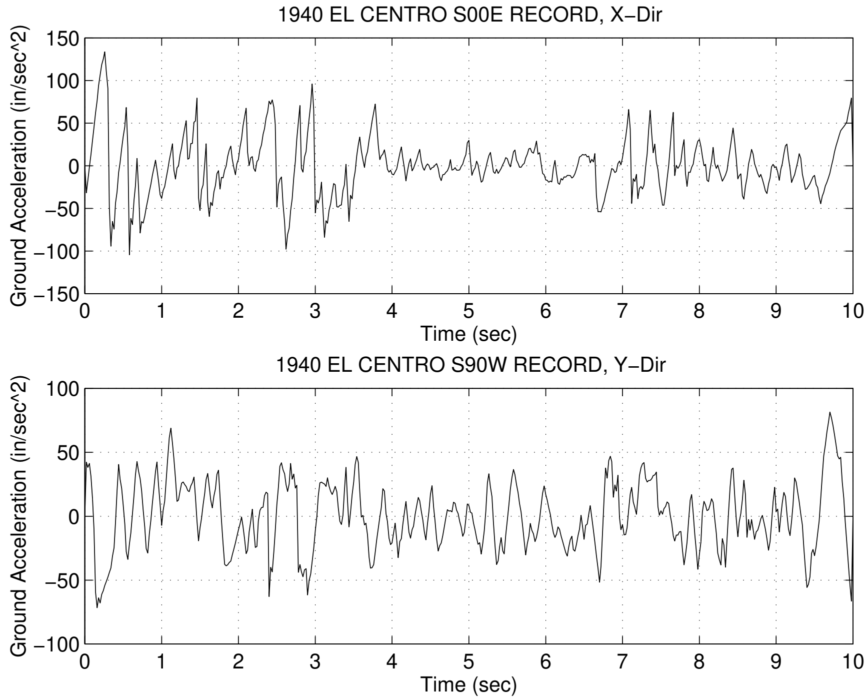


Figure 5.12: 1940 El Centro Earthquake Record

from the 1940 El Centro record by Balling et al. [9]. See Figure 5.12. The standard way of solving this problem is to note that the total displacement v^t may be expressed as the sum of the relative displacement v and the pseudo-static displacements v^s that would result from a static-support displacement. That is

$$v^t = v + v^s.$$

The pseudo-static displacements may be conveniently described by an influence vector $\mathbf{r}$ representing the displacements resulting from a unit support displacement. For a planar structure that only has horizontal degrees of freedom, $v^s = \mathbf{r}v_g$, where v_g is the ground displacement and

$$v^t = v + \mathbf{r}v_g,$$

the influence vector $\mathbf{r}$ simply contains a column of ones [12]. A slightly modified

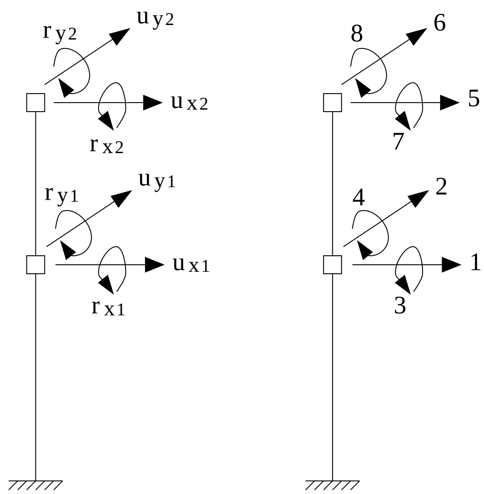


Figure 5.13: Global DOF for Two-Story Lumped Pier

version of this procedure is needed for the multi-component time-history analysis. Using $\mathbf{r}_x$ to represent the influence coefficient vector $\mathbf{r}$ in the x direction, and $\mathbf{r}_y$ in the y direction, the effective-force vector generated by the earthquake ground motion components is

$$\{\mathbf{P}(t)\} = -[\mathbf{M}]\{\mathbf{r}\}\ddot{v}_g(t) = -[\mathbf{M}](\{\mathbf{r}_x\}\ddot{v}_{gx}(t) + \{\mathbf{r}_y\}\ddot{v}_{gy}(t)).$$

If the structural degrees of freedom are as shown in Figure 5.13, then a unit displacement in the x direction affects only d.o.f. 1 and 5, and a unit displacement in the y direction affects only d.o.f. 2 and 6. Hence

$$r_x = \begin{Bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{Bmatrix} \quad \text{and} \quad r_y = \begin{Bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{Bmatrix}.$$

The following abbreviated input file shows the essential details of computing the multi-component time-history analysis with the Wilson-θ method.

```
/* Compute initial stiffness and mass matrices */
   stiff = Stiff();
   mass  = Mass([1]);

/* Manually add lumped mass and rotational inertial to mass matrix. */
   dof = GetDof([2]);
   mass[dof[1][1]][dof[1][1]] = M1;
   mass[dof[1][2]][dof[1][2]] = M1;
   mass[dof[1][4]][dof[1][4]] = Mjx1;
   mass[dof[1][5]][dof[1][5]] = Mjy1;
   dof = GetDof([3]);
   mass[dof[1][1]][dof[1][1]] = M2;
   mass[dof[1][2]][dof[1][2]] = M2;
   mass[dof[1][4]][dof[1][4]] = Mjx2;
   mass[dof[1][5]][dof[1][5]] = Mjy2;

   mass_inv = Inverse(mass);

/* Setup Rayleigh damping and damping matrix */
   rdamping = 0.05;
   A0 = 2*rdamping*w1*w2/(w1+w2);
   A1 = 2*rdamping/(w1+w2);
   damp = A0*mass + A1*stiff;

/* Setup initial displacement, velocity and acceleration */
   NodeLoad( 1, [ 0 kN, 0 kN, 0 kN, 0 kN*m, 0 kN*m, 0 kN*m] );
   P_ext = ExternalLoad();
   displ = Solve( stiff, P_ext );
   velocity = displ/(1 sec);
   accel = velocity/(1 sec);

/* Setup the influence vector in both dir-X and dir-Y */
   rx = displ/(1 m);  ry = displ/(1 m);
   accel_dir_x = 1;   accel_dir_y = 2;
   for( i=1 ; i<=total_node ; i=i+1 ) {
      dof = GetDof([i]);
      if(dof[1][accel_dir_x]>0) { rx[dof[1][accel_dir_x]][1]=1; }
      if(dof[1][accel_dir_y]>0) { ry[dof[1][accel_dir_y]][1]=1; }
   }

/* Define theta value for Wilson-theta method */
   theta = 1.420815;
   ds    = theta*dt;

/* Wilson-theta time-history analysis */

   for( stepno=1 ; stepno <= total_stepno ; stepno=stepno+1 ) {

   /* [1] : Compute effective incremental loading */
```

```
   time = time + dt;
   if( time <= quake_time ) then {
      P_ext = -mass*( rx*ground_accel_x[stepno][1]
                    + ry*ground_accel_y[stepno][1]);
   } else {
      P_ext = -mass*(rx*(0.0 m/sec/sec));
   }

   dPeff = theta*(P_ext-P_old) + mass*((6/ds)*velocity+3*accel)
         + damp*(3*velocity+(ds/2)*accel);

/* [2] : Compute effective stiffness from tangent stiffness */
   Keff = stiff + (3/ds)*damp + (6/ds/ds)*mass;

/* [3] : Solve for estimated delta_displacement */
   dps = Solve( Keff, dPeff );

/* [4] : Compute estimated displacement, velocity and acceleration */
   ds_accel = (6/ds/ds)*dps - (6/ds)*velocity - 3*accel;
   new_velocity = velocity + dt*accel + (dt/2/theta)*ds_accel;
   new_displ = displ + dt*velocity + (dt*dt/2)*accel
               + (dt*dt/6/theta)*ds_accel;

/* [5] : Check material yielding and compute new stiffness */
   dp = new_displ - displ;
   ElmtStateDet( dp );
   stiff = Stiff();

/* [6] : Compute new internal load, damping force, acceleration */
   Fs = InternalLoad( new_displ );
   Fd = damp*new_velocity;
   new_accel = mass_inv*( P_ext-Fs-Fd );

...... details of energy balance calculation explained later ......

/* [7] : Update histories for this time step */

   UpdateResponse();

   P_old    = P_ext;
   displ    = new_displ;
   velocity = new_velocity;
   accel    = new_accel;
}
```

The first and second floor displacements in the x and y directions are plotted in Figure 5.14 and Figure 5.15, respectively.

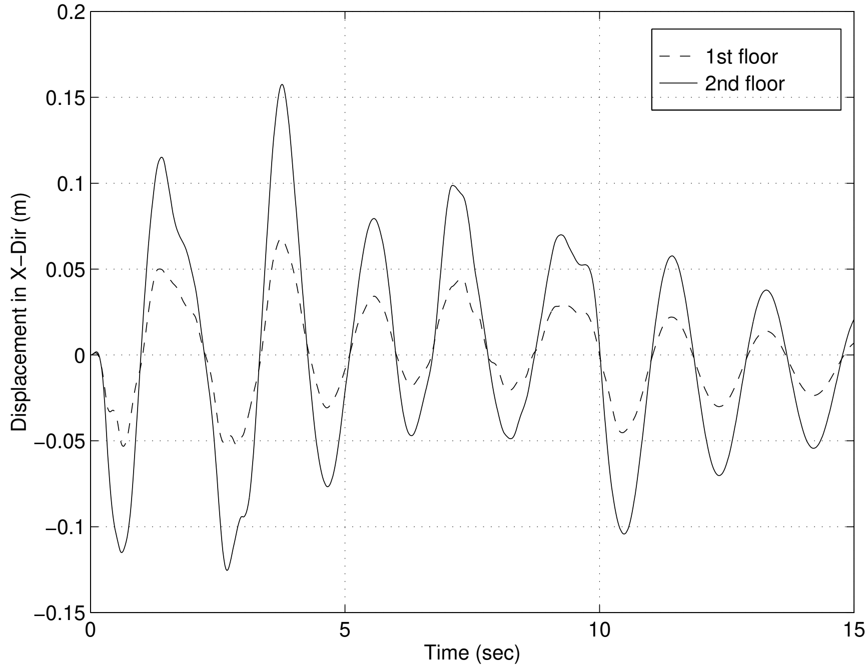


Figure 5.14: Earthquake Response for Two-Story Pier in x Direction

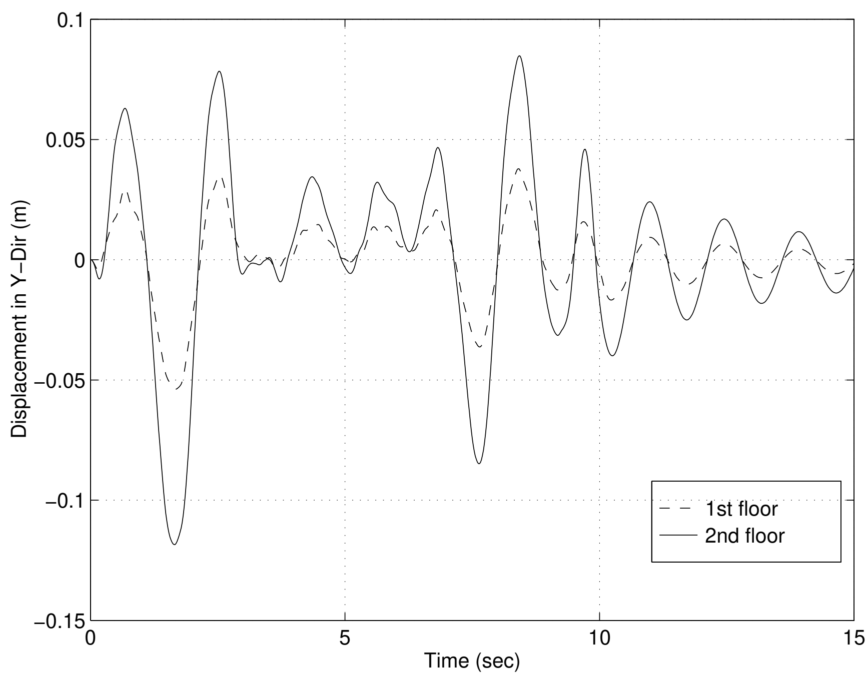


Figure 5.15: Earthquake Response for Two-Story Pier in y Direction

5.2 Energy Evaluation

5.2.1 Strain Energy

The concept of strain energy is of fundamental importance in applied mechanics [19]. To illustrate the basic ideas we apply a gradually increasing static load on a prismatic bar. See Figure 5.16.

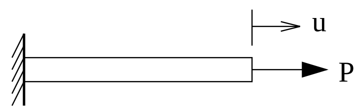


Figure 5.16: Prismatic Bar Subjected to a Statically Applied Load

Application of the load produces strains in the bar, and the effect of these strains is to increase the energy level of the bar itself. Strain energy is defined as the energy absorbed by the bar during the loading process. It is equal to the work done by an external load moving through the displacement u, provided no energy is added or subtracted in the form of heat. Therefore,

$$U_n = \int_0^{u_n} P(u)du, \tag{5.24}$$

where U_n is the strain energy at load step number n.

Sometimes strain energy is referred to as internal work to distinguish it from external work. When the force P is slowly removed from the bar, the bar will shorten and either partially or fully return to its original length depending upon whether the elastic limit has been exceeded. Thus, during the unloading procedure, some or all of the strain energy of the bar may be recovered in the form of work.

The load-deflection relationship for a typical nonlinear system is shown in Figure 5.17. During loading, the work done is equal to the area under the curve

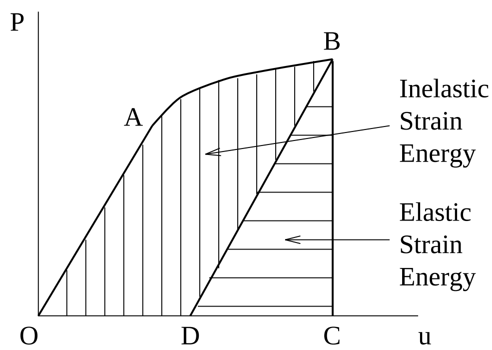


Figure 5.17: Elastic and Inelastic Strain Energy

(i.e., area OABCDO). When the load is removed the load-deflection diagram follows line BD. A permanent elongation OD remains when point B is beyond the elastic limit. Thus, the elastic strain energy recovered during unloading is represented by the shaded triangle BCD. Area OABDO represents energy that is lost in the process of permanently deforming the bar. This energy is known as the inelastic (or plastic) strain energy [19].

Now let us define ks = elastic tangent stiffness. The elastic strain energy that can be recovered in the unloading process is:

$$U_{elastic,n} = Area_{BCD} = \frac{1}{2} \cdot \frac{P_n^2}{ks}.$$

Approximating equation 5.24 by the trapezoidal rule gives:

$$U_n = Area_{OABCDO} = U_{n-1} + \frac{1}{2}(P_{n-1} + P_n)du_n.$$

From Figure 5.17 and the principle of conservation of energy, it follows that

$$U_n = U_{elastic,n} + U_{plastic,n}.$$

The energy that is lost in the process of permanently deforming the bar is the inelastic (plastic) strain energy, and it can be obtained by

$$U_{plastic,n} = Area_{OABDO} = U_n - U_{elastic,n}.$$

Numerical Example : To illustrate how ALADDIN can be used to calculate the strain energy in a nonlinear system response, a nonlinear SDOF spring system is presented in Figure 5.18. The lumped masses are $m1 = 1.0kg$ and $m2 = 1.0kg$ at degrees of freedom 1 and 2 respectively. $c1 = 1.5N.sec/m$ and $c2 = 1.0N.sec/m$ represent the coefficient of viscous damping at degrees of freedom 1 and 2 respectively. Springs 1 and 2 both have a bi-linear force-displacement relationships which follow the kinematic hardening rule. The undeformed springs start out with an initial stiffness ks that lasts until the load exceeds the yield force fy. The tangent stiffness then changes to kt. When the loading is reversed, the tangent stiffness switches back to the initial stiffness ks until the yielding reappears. The elastic force-displacement range is $2 \cdot fy$.

This system is subjected to the time-varying load shown in Figure 5.19. Our analysis will be divided into two parts. First we assume that each time step is very long and the load is slowly applied during the load step – in other words, the externally applied loads are static, and the underlying equations of equilibrium are not influenced by damping or inertial effects. The second part of our analysis will account for dynamic effects.

In the static analysis, a Newton-Raphson procedure is used to calculate the load-deflection response of this spring system. We also calculate the elastic and plastic components of energy of the spring elements. Except for the spring properties and strain energy calculation, the input file is almost the same as for the composite bar example presented in the preceding chapter. The abbreviated input file is:

ABBREVIATED INPUT FILE

```
/* allocate the matrices for storing energy calculations */
```

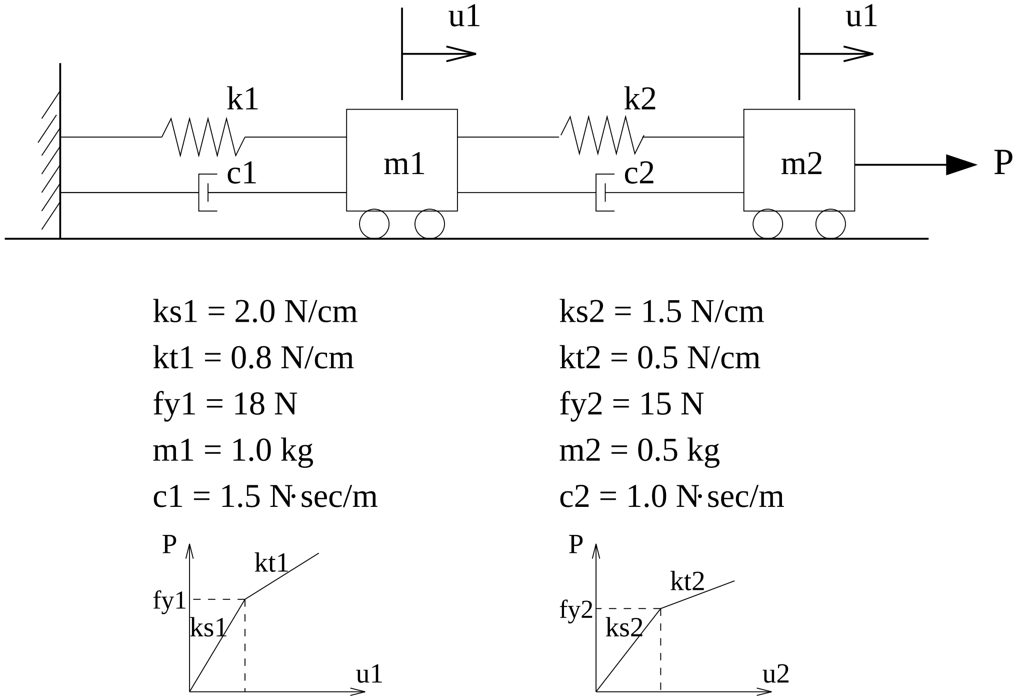


Figure 5.18: Nonlinear SDOF Spring System

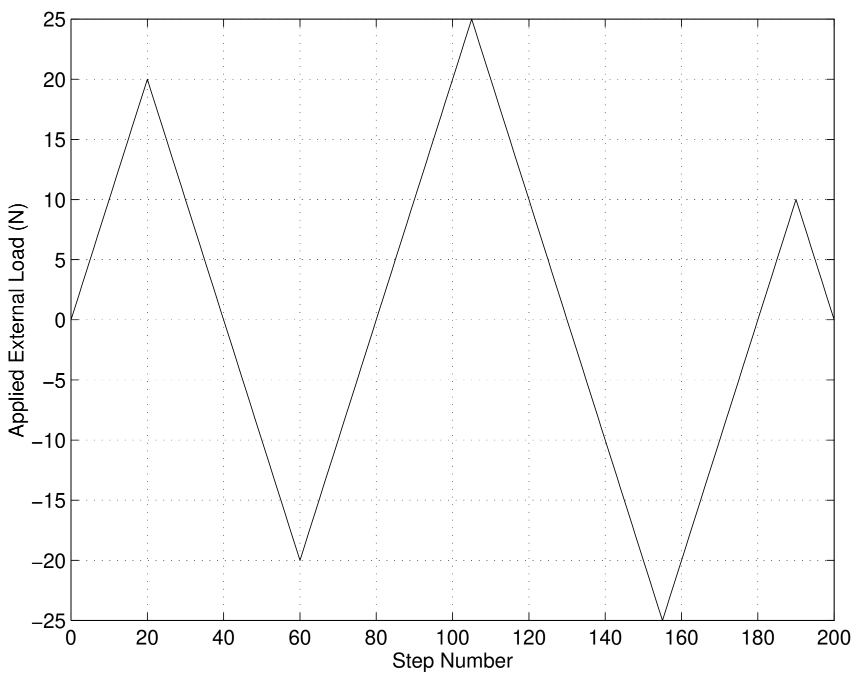


Figure 5.19: Varying Load Applied on Spring System

```
   system_energy  = ColumnUnits( Matrix([total_step+1,4]), [Jou] );
   element_energy = ColumnUnits( Matrix([total_step+1,4]), [Jou] );

/* Increase External Load */
   for ( k=1; k<=total_step ; k=k+1 )
   {

   /* define force increment for each step */

      if( k<=20 )           { d_P =  [0 N; 1 N]; }
      if( k>20  && k<=60  ) { d_P = -[0 N; 1 N]; }
      if( k>60  && k<=105 ) { d_P =  [0 N; 1 N]; }
      if( k>105 && k<=155 ) { d_P = -[0 N; 1 N]; }
      if( k>155 && k<=190 ) { d_P =  [0 N; 1 N]; }
      if( k>190 && k<=200 ) { d_P = -[0 N; 1 N]; }
      if( k>200 )           { d_P =  [0 N; 0 N]; }
      P = P + d_P;

      element_energy[k+1][1] = element_energy[k][1];
      element_energy[k+1][3] = element_energy[k][3];

   /* i-th Newton-Raphson Iteration */
      index[1][1] = 1;   index[2][1] = 1;
      err = tol+1;
      while( err > tol )
      {
         d_p = Solve(BigK,d_P);
         p = p + d_p;

      /* state determination for each element */

         for( ele=1;ele<=2;ele=ele+1 )
         {
         ...... details about retrieving data from (j-1) ......
            q = q + d_q;

         /* element converge, j */

            while( abs(DUx) > 0.00001 N )
            { ...... details of checking element convergence  ...... }

            ...... details of updating data at loop j ......

         /* energy calculations for each element ele */

            element_energy[k+1][2*ele-1] = element_energy[k+1][2*ele-1]
                           + 0.5*(Q_saved[ele][1]+Q)*(q-q_saved[ele][1]);
            element_energy[k+1][2*ele]   = 0.5*Q*Q/Ks[ele][1];
         }
```

```
   /* assemble structure resistant force */
      ...... details of assembling structure resistant force ......

   /* assemble new structure stiffness */
      ...... details of assembling new structure stiffness ......

   d_P = P - PR;
   err = L2Norm(d_P);

}  /* i-th iteration in Newton-Raphson while loop */

...... details of storing response removed ......
...... details of updating element history ......

/* Reassemble the System Energy */

   system_energy[k+1][1] = system_energy[k][1]
      + 0.5*(result[k+1][1]+result[k][1])*(result[k+1][2]-result[k][2]);
   system_energy[k+1][2] = element_energy[k+1][1]+element_energy[k+1][3];
   system_energy[k+1][3] = element_energy[k+1][2]+element_energy[k+1][4];
   system_energy[k+1][4] = system_energy[k+1][2] - system_energy[k+1][3];
}  /* end of for loop step */
```

In the input file, the matrix `element_energy` stores the element energy for all loading steps. The first column stores the internal work of element 1. The second column stores the element elastic strain energy of element 1. The third column stores the internal work of element 2. The fourth column stores the element elastic strain energy of element 2. A second matrix `system_energy` stores the system energy for all of the loading steps. The first column stores the total external work done by the system. The second column stores the total internal work of the system (it is the sum of internal work in the elements). The third column stores the total elastic strain energy of the system (it is the sum of elastic strain energy in the elements). The fourth column stores the total plastic strain energy of the system (again, this quantity is the sum of plastic strain energies in the elements).

The load-displacement and energy results are plotted in Figures 5.20

through 5.25. In the energy plots, you should observe that the plastic energy dissipation remains constant during periods of load until another yield point is reached.

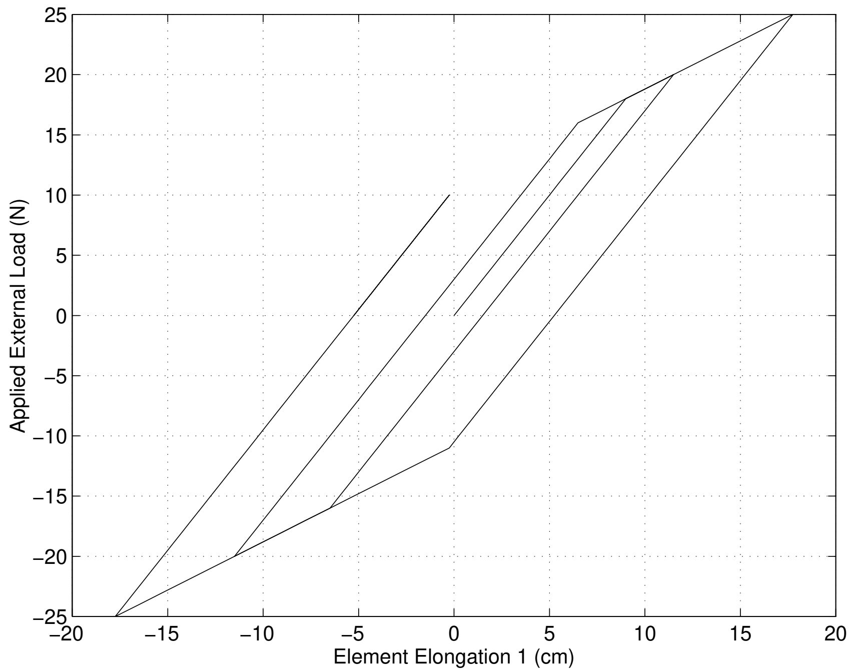


Figure 5.20: Load-Deflection Diagram of Spring 1

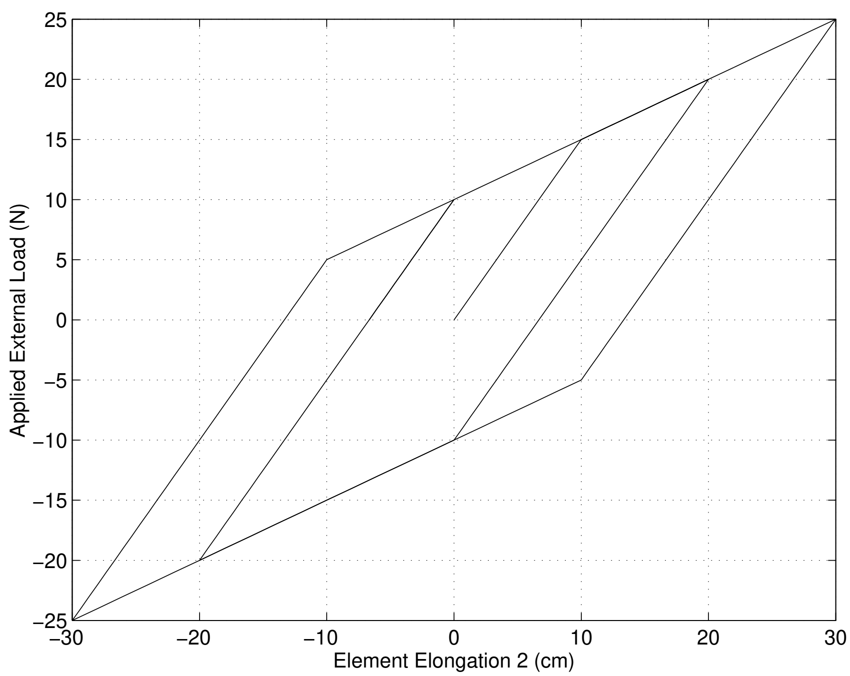


Figure 5.21: Load-Deflection Diagram of Spring 2

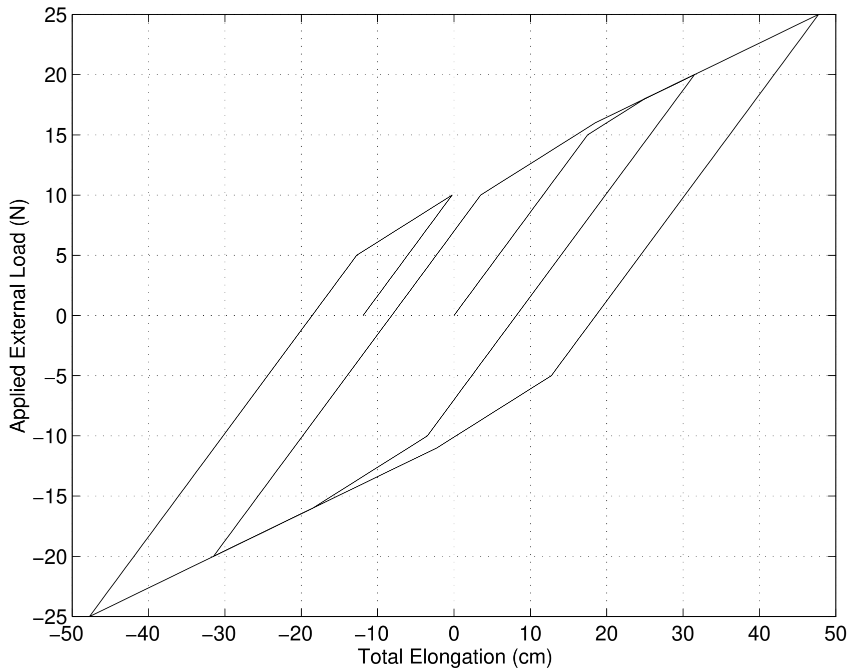


Figure 5.22: Load-Deflection Diagram of System

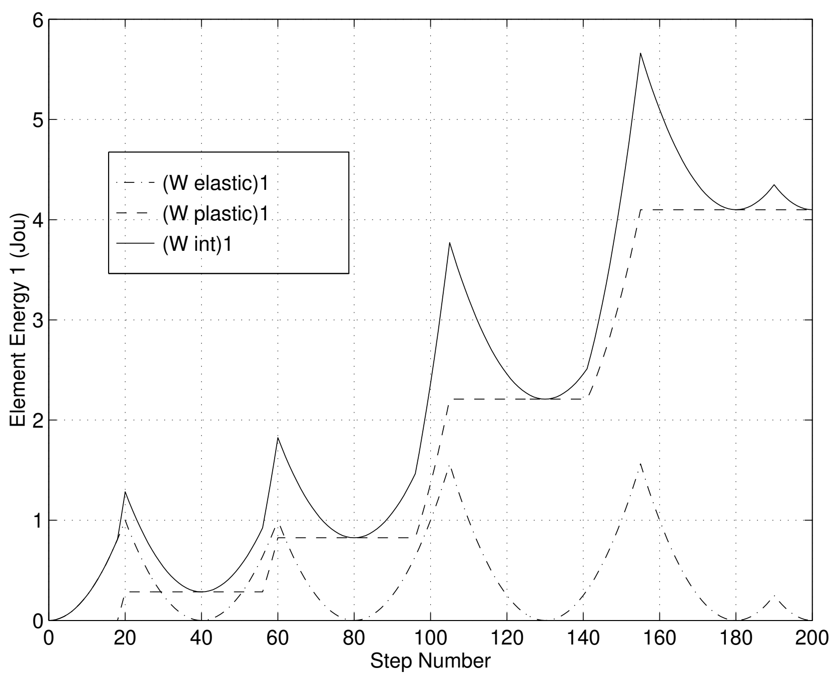


Figure 5.23: Strain Energy Plot of Spring 1

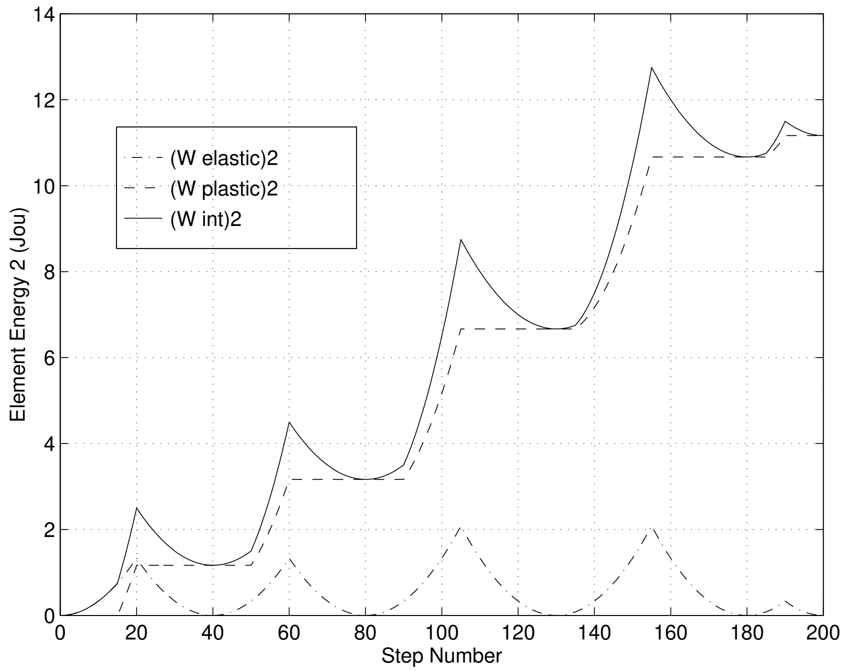


Figure 5.24: Strain Energy Plot of Spring 2

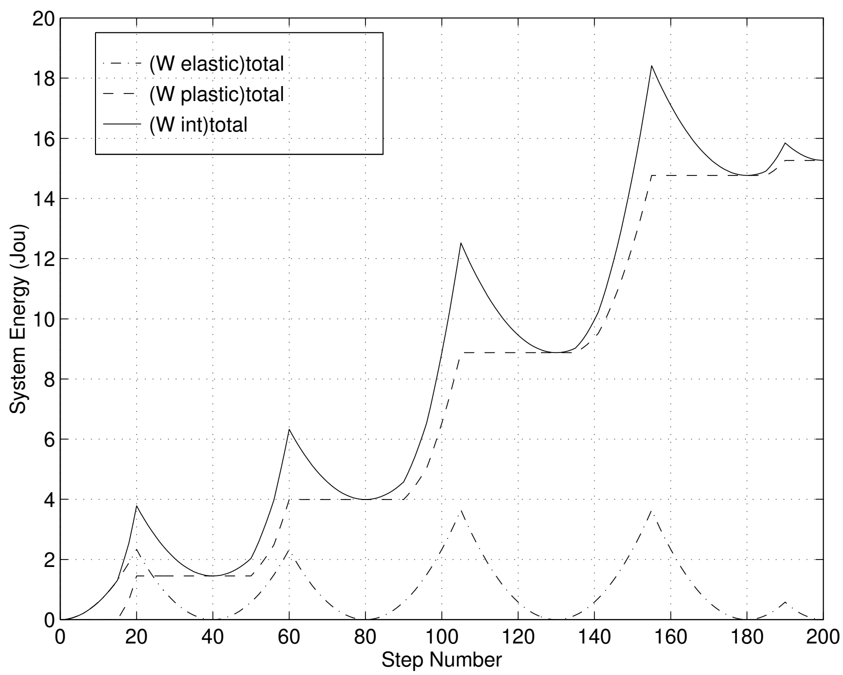


Figure 5.25: Strain Energy Plot of System (Static)

Dynamic Analysis of Nonlinear SDOF System : We now repeat the system analysis assuming that the load is incremented in time steps of only 0.01 second. The total loading time is 2 seconds. The system response is computed for a total of 4 seconds, however, using the Newmark algorithm with average acceleration across the time steps. We also calculate the strain energy, elastic and plastic energies of the spring elements, and the natural periods of the system throughout the dynamic analysis. The abbreviated input file is shown below:

ABBREVIATED INPUT FILE

```
/* assemble initial structure tangent stiffness matrix BigK */
   BigK = [ ks1+ks2, -ks2;  -ks2, ks2 ];

/* assemble mass matrix and damping matrix */
   BigM = [ m1, 0 kg;  0 kg, m2 ];
   BigC = [ c1+c2, -c2;  -c2, c2 ];

   M_inv = Inverse(BigM);

/* initial time = 0sec conditions */
   total_step = 400;
   dt = 0.01 sec;
   time = 0 sec;

/* Setup initial velocity and acceleration */
   p = [ 0 cm; 0 cm ];
   vel = p/(1 sec);
   accel = vel/(1 sec);

/* allocate the matrices for storing output history */

   result = ColumnUnits(Matrix([total_step+1,8]), [N,cm,cm,cm,N,N,sec,sec]);
   element_energy = ColumnUnits( Matrix([total_step+1,4]), [Jou] );

/* compute dynamic periods */
   eigen = Eigen(BigK, BigM, [2]);
   eigenvalue  = Eigenvalue(eigen);

/* increase external load, structure determination */
   for ( k=1; k<=total_step ; k=k+1 )
   {
      time = time + dt;

   /* define force increment for each step */
```

```
   ...... details of defining dP ......

/* Compute effective incremental loading */
   dPeff = d_P + ((4/dt)*BigM + 2*BigC)*vel + 2*BigM*accel;

/* Compute effective stiffness from tangent stiffness */
   Keff = BigK + (2/dt)*BigC + (4/dt/dt)*BigM;

/* Solve for d_displacement, d_velocity */
   d_p = Solve( Keff, dPeff );
   d_v = (2/dt)*d_p - 2*vel;

/* Compute displacement, velocity */
   new_p   = p + d_p;
   new_vel = vel + d_v;

/* Check material yielding and compute new stiffness */
   for( ele=1 ; ele<=2 ; ele=ele+1 )
   {
      ...... details of element state determination ......

   /* energy calculations for each element ele */

      element_energy[k+1][2*ele-1] = element_energy[k+1][2*ele-1]
                     + 0.5*(Q_saved[ele][1]+Q)*(q-q_saved[ele][1]);

      if( abs(Q) > 2*Fy[ele][1] ) then {
         if( kx == Ks[ele][1] ) then {
            delta_Q = abs(sr[ele][1]) - 2*Fy[ele][1];
         } else {
            delta_Q = abs(Q) - 2*Fy[ele][1];
         }
         delta_x = delta_Q*(1/Kt[ele][1]-1/Ks[ele][1]);
         element_energy[k+1][2*ele] =
            0.5*Q*Q/Ks[ele][1] + 0.5*delta_x*delta_Q;
      } else {
         if( abs(sr[ele][1]) > 2*Fy[ele][1] ) then {
            if( kx == Ks[ele][1] ) then {
               delta_Q = abs(sr[ele][1]) - 2*Fy[ele][1];
               delta_x = delta_Q*(1/Kt[ele][1]-1/Ks[ele][1]);
               element_energy[k+1][2*ele] =
                  0.5*Q*Q/Ks[ele][1] + 0.5*delta_x*delta_Q;
            } else {
               if(((Q>0N)&&(sr[ele][1]>0N))||((Q<0N)&&(sr[ele][1]<0N)))
               then {
                  element_energy[k+1][2*ele] = 0.5*Q*Q/Kt[ele][1];
               } else {
                  element_energy[k+1][2*ele] = 0.5*Q*Q/Ks[ele][1];
               }
            }
```

```
         } else {
            element_energy[k+1][2*ele] = 0.5*Q*Q/Ks[ele][1];
         }
      }

    }  /* ele */

 /* assemble new structure stiffness */
    BigK[1][1] = tangent[1][1] + tangent[2][1];
    BigK[1][2] = -tangent[2][1];
    BigK[2][1] = -tangent[2][1];
    BigK[2][2] = tangent[2][1];

 /* assemble new internal load Fs, damping load Fd, and acceleration */
    Fs[1][1] = PRe[1][1] - PRe[2][1];
    Fs[2][1] = PRe[2][1];

    Fd = BigC*new_vel;
    new_accel = M_inv*( P-Fs-Fd );

 /* store analysis results */
    ...... details of storing analysis results ......

 /* compute eigen problem */
    eigen = Eigen(BigK, BigM, [2]);
    eigenvalue  = Eigenvalue(eigen);

    result[k+1][7] = 2*PI/sqrt( eigenvalue[1][1] );
    result[k+1][8] = 2*PI/sqrt( eigenvalue[2][1] );

 /* updating history for each load step k */
    for( ele=1 ; ele<=2 ; ele=ele+1 )
    { ...... details of updating element history ......  }

 /* update results for this step */
    p      = new_p;
    vel    = new_vel;
    accel  = new_accel;
 }  /* k in for() loop, increase to next time step */
```

Points to note in input are:

1. The matrix `result` stores the computation results. Each column stores:

```
/* column[1] : external applied load at end node (2)      */
/* column[2] : total elongation at end node (2) = [3]+[4]  */
/* column[3] : element elongation 1, node 1                */
/* column[4] : element elongation 2, node 2                */
```

```
/* column[5] : element force 1, node 1                    */
/* column[6] : element force 2, node 2                    */
/* column[7] : dynamic natural period 1 (T1) for each step */
/* column[8] : dynamic natural period 2 (T2) for each step */
```

2. The matrix `element_energy` stores results of the element energy calculation. Each column stores:

```
/* column[1] : internal strain energy for element 1 */
/* column[2] : elastic strain energy for element 1  */
/* column[3] : internal strain energy for element 2 */
/* column[4] : elastic strain energy for element 2  */
```

Figures 5.26 and 5.27 show the first and second modal periods versus time for our nonlinear analysis. We note that:

1. At any time in the time-history response, the mass-spring system will assume one of four possible states of stiffness. These states are due to various stages of spring stiffness yielding of course. The combinations are: first, both springs are elastic; spring 1 yields and spring 2 is elastic; spring 1 is elastic and spring 2 yields, and finally, both springs have yielded. In its unloaded state, the mass-spring system has a first mode period of 0.595 seconds and a second mode period of 0.271 seconds. The first mode period elongates to 0.961 seconds and the second mode period elongates to 4.591 seconds when both springs are in a post-yielding state.

2. The average natural period of the system during the loading period is shown to indicate the period shift of the system ($(T_1)_{average} = 0.827 sec$ and $(T_2)_{average} = 0.374 sec$).

3. Figures 5.26 and 5.27 are important because they provide a rational basis for assessing the validity and limitations of simplified procedures for the

> design of nonlinear systems. Rather than work directly with a time-varying nonlinear system, a number of simplified analysis procedures are based on the behavior of an equivalent elastic system. The equivalent system will have a damping ratio and effective natural period that is a function (most likely an empirical function) of the system ductility.

The strain energy plot is shown in Figure 5.28. The elastic energy is gradually damped to zero. The plastic energy is still increasing after the external loading ceases at time 2 seconds (this is because of the inertial effects remaining on the system). Finally, total elongations of the static and dynamic systems are compared in Figure 5.29. The elongation of the static system remains constant after the loading ceases, while the elongation of the dynamic system gradually vibrates back to zero residual displacement.

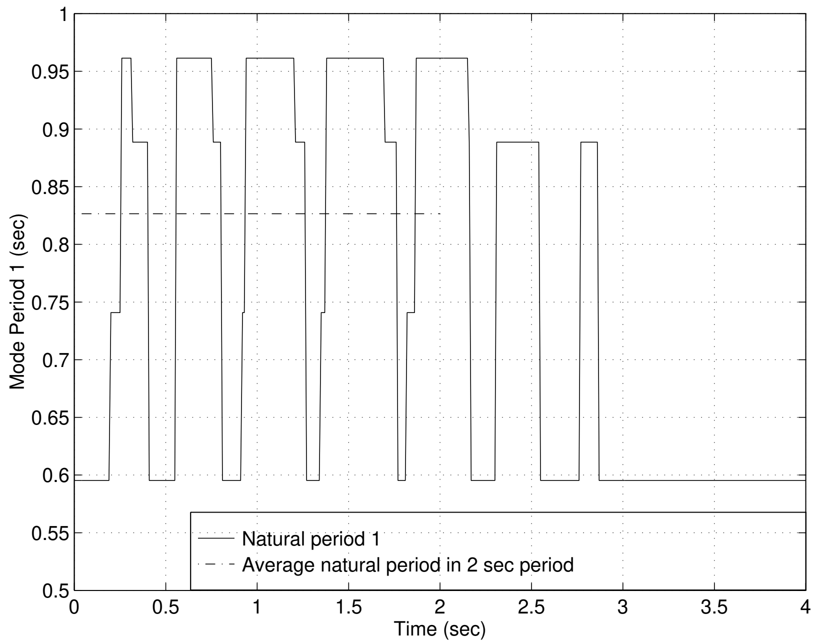


Figure 5.26: Change of Dynamic Natural Period 1

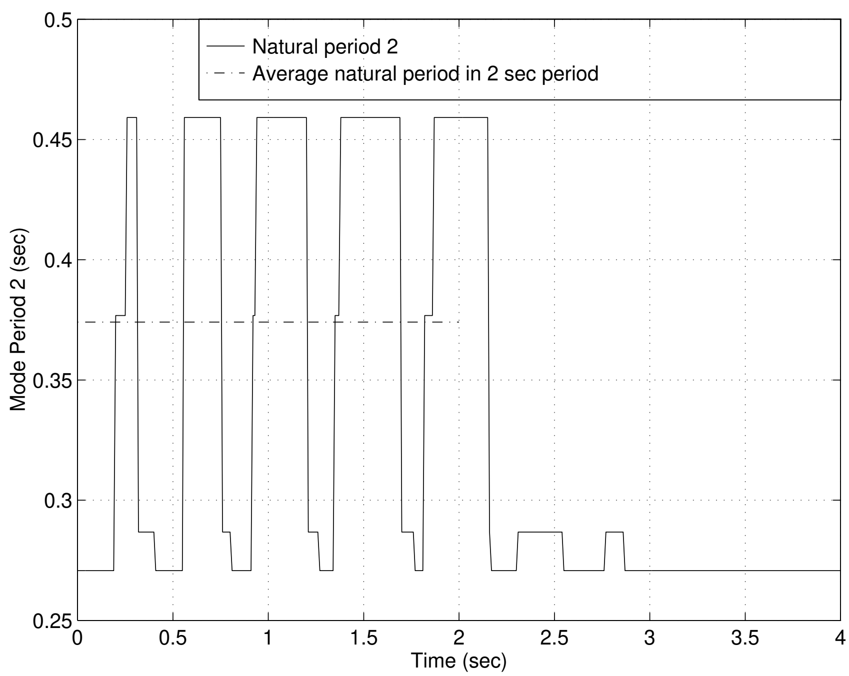


Figure 5.27: Change of Dynamic Natural Period 2

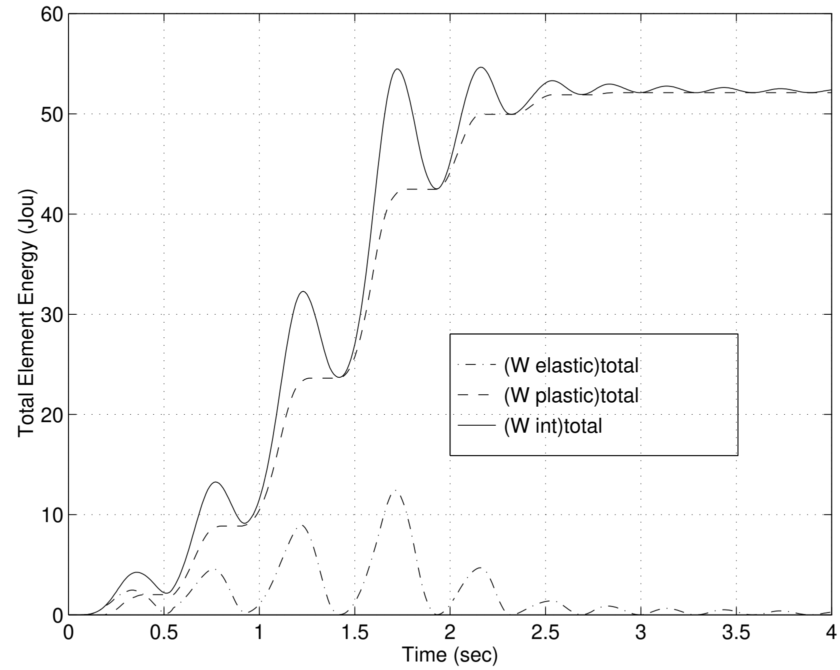


Figure 5.28: Strain Energy Plot of System (Dynamic)

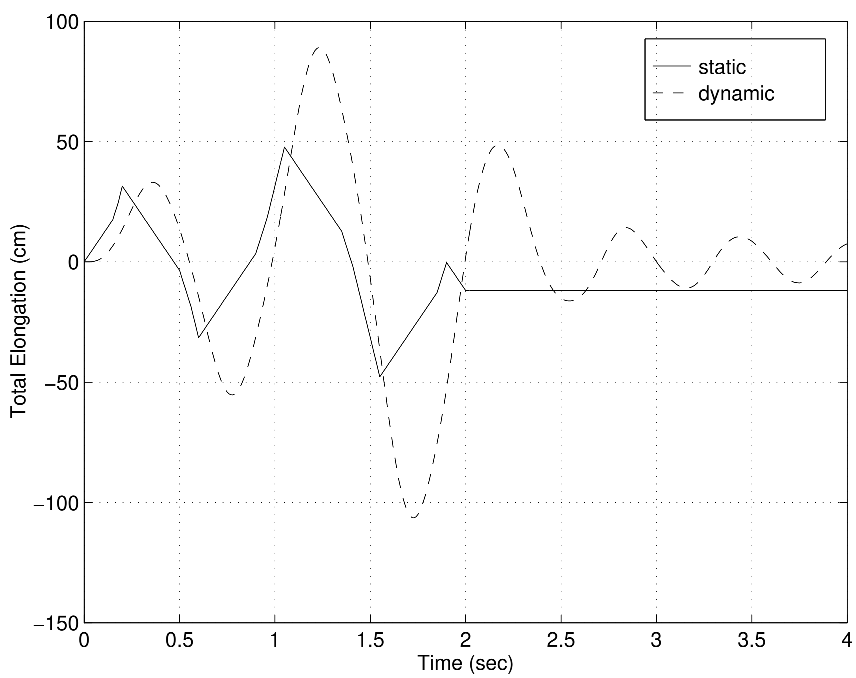


Figure 5.29: Comparison of Total Elongations of Static System and Dynamic System

5.2.2 Energy Balance Calculations

In the analysis of nonlinear dynamic problems, it is usually advisable to perform an energy balance check as a means of validating the computations are stable and accurate. At the highest level of abstraction we can say that the system energy at time $(n+1)\Delta t$ should satisfy the equation:

$$W_{n+1}^{int} + T_{n+1} = W_{n+1}^{ext} \tag{5.25}$$

where W (without subscripts and superscripts) represents work done and T represents kinetic energy. In physical term, equation 5.25 states that the work done by external loads is converted to kinetic energy, and to energy stored either elastically or dissipated by plastic deformations [13].

The internal work, W_{n+1}^{int}, represents the work done by nodal loads that are developed from the straining of materials. It is given by

$$W_{n+1}^{int} = W_n^{int} + \int_{n\Delta t}^{(n+1)\Delta t} \dot{W}^{int} dt. \tag{5.26}$$

A discrete approximation to equation 5.26 may be obtained by noticing that:

$$\dot{W}_n^{int} = \{\dot{\mathbf{D}}\}_n^T \{\mathbf{R}^{int}\}_n.$$

Here $\{\dot{\mathbf{D}}\}_n$ represents the velocity vector at time step n, and $\{\mathbf{R}^{int}\}_n$ represents the resisting nodal loads at time step n. Approximating the integral in equation 5.26 by the trapezoidal rule gives

$$W_{n+1}^{int} = W_n^{int} + \frac{\Delta t}{2}(\{\dot{\mathbf{D}}\}_n^T \{\mathbf{R}^{int}\}_n + \{\dot{\mathbf{D}}\}_{n+1}^T \{\mathbf{R}^{int}\}_{n+1}). \tag{5.27}$$

The external work, W_{n+1}^{ext}, represents the work done by changes in externally applied loads moving through displacement of the system degrees of freedom.

It is given by

$$W_{n+1}^{ext} = W_n^{ext} + \int_{n\Delta t}^{(n+1)\Delta t} \{\dot{\mathbf{D}}\}^T\{\mathbf{R}^{ext}\}dt. \tag{5.28}$$

A suitable discrete approximation for W_{n+1}^{ext} can be obtained from equation 5.27 by replacing superscript "int" by "ext"; i.e.,

$$W_{n+1}^{ext} = W_n^{ext} + \frac{\Delta t}{2}(\{\dot{\mathbf{D}}\}_n^T\{\mathbf{R}^{ext}\}_n + \{\dot{\mathbf{D}}\}_{n+1}^T\{\mathbf{R}^{ext}\}_{n+1}). \tag{5.29}$$

For systems that are damped, the resisting nodal load $\{\mathbf{R}^{int}\}$ includes at time step n is a combination of straining force and damping force components. The damping force at time step n is given by

$$\{\mathbf{R}_{damp}\}_n = [\mathbf{C}]_n\{\dot{\mathbf{D}}\}_n.$$

The straining force $\{\mathbf{R}_{stiff}\}_n$ is obtained directly from the stress-strain curve. Therefore

$$\{\mathbf{R}^{int}\}_n = \{\mathbf{R}_{damp}\}_n + \{\mathbf{R}_{stiff}\}_n. \tag{5.30}$$

Finally, the kinetic energy T_{n+1} is given by

$$T_{n+1} = \frac{1}{2}\{\dot{\mathbf{D}}\}_{n+1}^T[\mathbf{M}]\{\dot{\mathbf{D}}\}_{n+1}. \tag{5.31}$$

To construct an energy balance, we note that in general equation 5.25 is not satisfied exactly. The quality of a numerical solution can be measured with the convergence criterion:

$$W_n^{int} + T_n - |W_n^{ext}| \leq e(W_n^{int} + T_n - |W_n^{ext}|) \tag{5.32}$$

where e is a tolerance factor and the absolute magnitude bars are a precaution against small negative values of W^{ext} caused by spurious numerical errors. Terms within parentheses on the right-hand side of equation 5.32 represent the total energy in the system. The left-hand side is the energy error.

Numerical Example : We now extend the previous two-story lumped mass pier example with the energy balance calculations. The following abbreviated input file is positioned after the end of each load step calculation, but before the updating of the element history response.

ABBREVIATED INPUT FILE

```
/* calculate the energy balance, Wint(n+1) + T(n+1) = Wext(n+1) */

      trans_vel     = Trans(velocity);
      trans_vel_new = Trans(new_velocity);
      trans_dis_new = Trans(new_displ);

      if( stepno == 1 ) then {

         Wint_fs = dt/2*(trans_vel_new*Fs);
         Wint_fd = dt/2*(trans_vel_new*Fd);
         T    = (trans_vel_new*mass*new_velocity)/2;
         Wext = dt/2*(trans_vel_new*P_ext);

         energy[1][1] = Wint_fs[1][1];
         energy[1][2] = Wint_fd[1][1];
         energy[1][3] = T[1][1];
         energy[1][4] = Wext[1][1];

      } else {

         Wint_fs = dt/2*(trans_vel*Ps_int + trans_vel_new*Fs);
         Wint_fd = dt/2*(trans_vel*Pd_int + trans_vel_new*Fd);
         T    = (trans_vel_new*mass*new_velocity)/2;
         Wext = dt/2*(trans_vel*P_old + trans_vel_new*P_ext);

         energy[stepno][1] = energy[stepno-1][1] + Wint_fs[1][1];
         energy[stepno][2] = energy[stepno-1][2] + Wint_fd[1][1];
         energy[stepno][3] = T[1][1];
         energy[stepno][4] = energy[stepno-1][4] + Wext[1][1];

      }
```

The matrix `energy` stores the results for each time step. The first column contains the internal energy done by straining force, the second column contains the internal energy done by damping force, the third column stores the kinetic energy, and the fourth column stores the external energy.

The overall results of our energy balance calculation are plotted in Figure 5.30 and Figure 5.31. Figure 5.31 shows a great match of the internal energy and the external energy, therefore proving that the analysis of this nonlinear dynamic problem has stable and accurate computation. Also note from Figure 5.31 that after the earthquake ground motions have ceased, there is no more external energy input into the system. Hence, the curve W_{ext} versus time is constant over the response interval $t \in [10, 15]$ seconds.

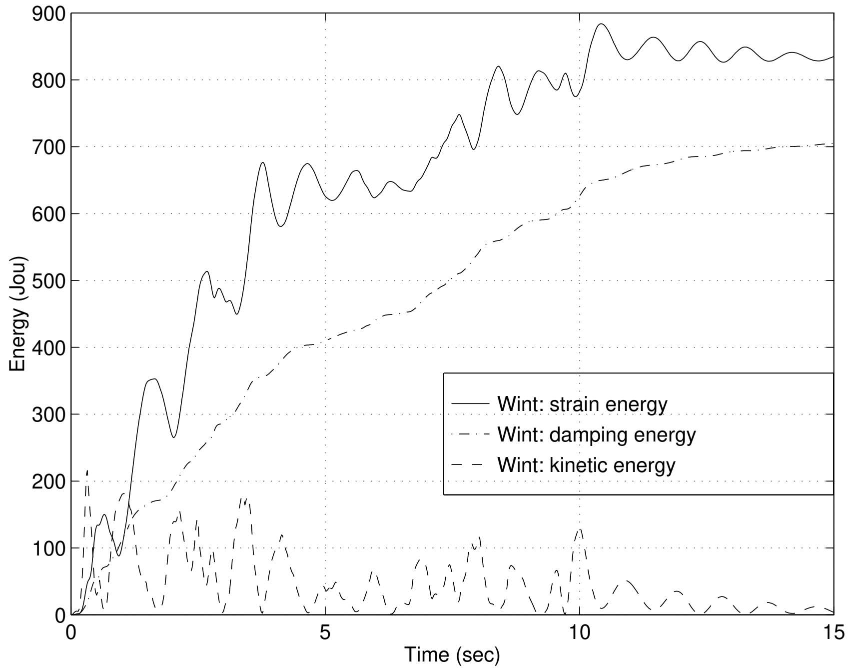


Figure 5.30: Internal Energy Time History for Two-Story Pier

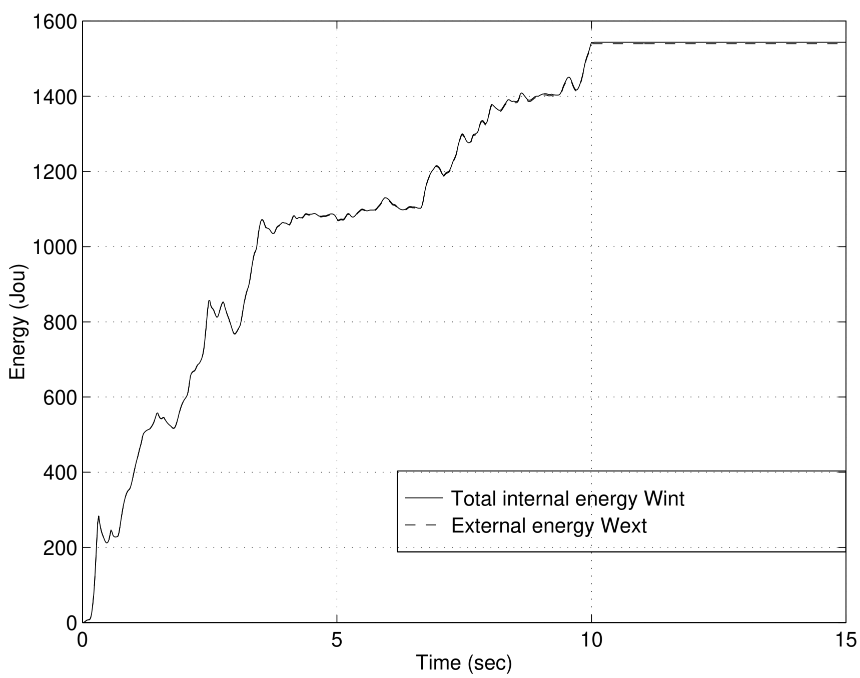


Figure 5.31: Total Energy Time History for Two-Story Pier

CHAPTER 6

Analytical Studies for Base Isolation of Bridges

6.1 Introduction to Base Isolation

Base isolators are artificial elements that protect highway bridge structures from the full intensity of seismic attack. They are usually positioned between the bridge piers and abutments, and the deck, and are designed for high energy dissipation without a loss in strength occurring. In principle at least, this capability enables the remainder of the structure to respond elastically, and suffer no structural damage.

A number of mechanisms contribute to base isolation protection, including high levels of viscous damping and energy dissipation, and movement of the bridge system's natural periods of vibration to regions of low dynamic response. High levels of viscous damping and energy dissipation are desirable because they lower the forces and displacements a structure must resist. Studies of earthquake response spectra indicate that similar improvements in performance will occur when the natural period of a structure is moved to a region having low spectral accelerations. Collectively these mechanisms lower the variation in bridge responses caused by a wide range of ground motion inputs, and reduce the likelihood of undesirable concentrations of ductility demand. Unfortunately, reductions in lateral forces are sometimes accompanied by increases in structural displacements. A balance in these criteria is therefore required [35, 12, 1].

Most of the current isolation systems fall into two categories – elastomeric isolation systems and sliding isolation systems. Elastomeric bearings are constructed from laminated rubber bearings reinforced with steel plates. A lead plug provides the isolator with stiffness to withstand moderate lateral loads without yielding, and a capacity to dissipate large quantities of energy during high lateral loads. Sliding isolation systems (e.g., Teflon-slider sliding on a stainless steel plate) protect a superstructure by decoupling it from the ground. They dissipate energy by means of frictional behavior. Restoring force and re-centering capabilities are provided by helical springs (or by springs) in the form of rubber cylinders [31].

The purposes of this chapter are two-fold. First, we formulate a nonlinear fiber finite element for the modeling of elastomeric isolators. Since fiber elements cannot model sliding isolation elements accurately, this aspect of isolation protection will not be discussed further in this chapter. The second objective for this chapter is application of the element to the nonlinear time-history analysis of a four-span bridge structure.

6.2 Lead-Rubber Bearings

The lead-rubber bearing is a laminated rubber bearing containing a lead plug insertion, as shown in Figure 6.1. The lead plug provides energy dissipation for seismic response and stiffness for static loads and small lateral loads (e.g., wind loadings). Lead-rubber bearings have been used extensively in bridge structures that must resist severe seismic attack, and are an economical and effective solution for bridge isolation, incorporating period shifting and increased damping in a single device [35].

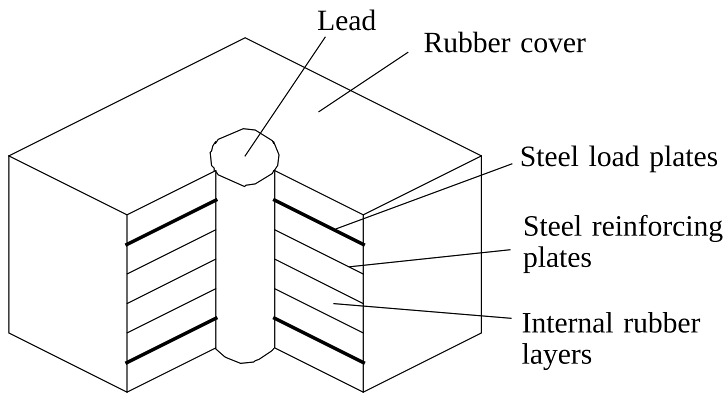


Figure 6.1: Lead-Rubber Laminated Bearing

6.2.1 Material Properties

There are several good reasons for constructing the bearing center from lead. The material properties of lead include: (a) a low yield shear strength [about 10 MPa (1.45 ksi)], (b) sufficiently high initial shear stiffness [the shear modulus G is approximately equal to 130 MPa (18.8 ksi)], (c) post yielding behavior is essentially elastic-plastic, and (d) good fatigue properties for plastic cycles. Experimental studies indicate that lead responds essentially with elastic-perfectly plastic loops. Hence, for practical purposes the post-yielding isolator stiffness is equal to the stiffness of the rubber bearing alone. The global hysteresis loop is a bi-linear solid with an initial elastic stiffness $k_1 = 10k_r$ where k_r is the stiffness of rubber, then followed by a post yield stiffness $k_2 = k_r$ [37, 43]. The size of the lead plug is proportional to the yield strength of the isolator. The post-yielding stiffness is proportional to the rubber bearing stiffness, and increases with the plan size of the rubber bearing and reductions in the isolator height. These trends

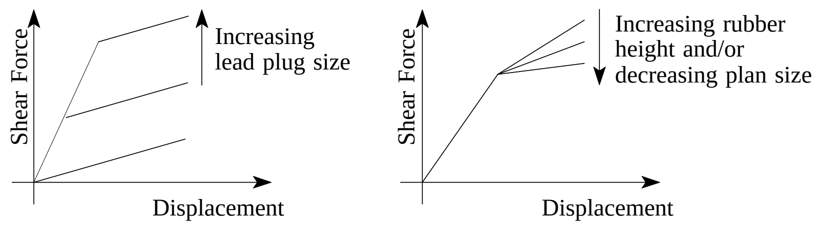


Figure 6.2: Effects of Geometrical Variations of the Lead Plug and Rubber Bearing on the Overall Response

are shown in Figure 6.2.

6.2.2 Lead-Rubber Isolator Modeling in ALADDIN

We now consider the problem of modeling the lead-rubber isolator with the fiber element in ALADDIN. Because the lead-rubber isolator is composed of two different materials, each with its own shear modulus, the isolator behavior is modeled with a `FIBER_3DS` element (see APPENDIX A for the definition and usage of `FIBER_3DS` element).

Figure 6.3 is a plan view of a lead-rubber isolator, showing the section dimensions and positions of the fiber elements. Each quadrant of the isolator is modeled with two fibers, one for the lead, and a second for the rubber. The fibers are positioned at the centroid of the material quadrant. The material properties are as mentioned in the previous section.

Figure 6.4 compares the force-displacement relationship for the `FIBER_3DS` element with the experimental data generated by Robinson [37]. Since there is a good match between these curves, we conclude that the bi-linear fiber element will provide sufficient modeling accuracy for this study.

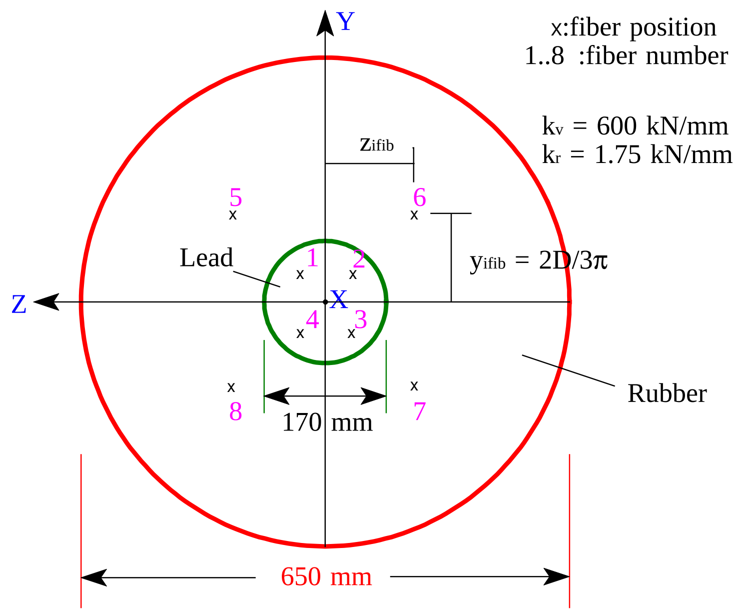


Figure 6.3: Modeling of Lead-Rubber Isolator

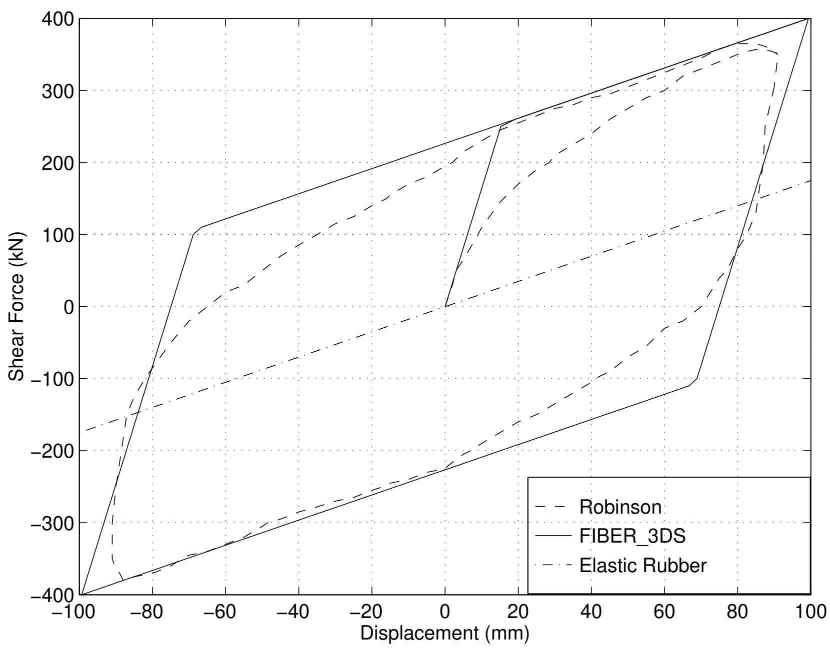


Figure 6.4: Force-Displacement Hysteresis Loops for Lead-Rubber Isolator

6.3 Modeling of Isolated Bridges

The modeling of an isolated structure can take on varying levels of complexity. The simplest case is a single-degree-of-freedom model with a linear-elastic isolation system and a rigid superstructure. The most complex case is a full three-dimensional model of both the isolation system and the superstructure, with a nonlinear isolation system and nonlinear frame type superstructure.

Shenton and Taylor [42] have provided a spectrum of analytical models that can be used in the analysis of base isolated structures. Maragakis and Saiidi [28] present some simple analytical models of lead-rubber base isolated bridges. A detail discussion on techniques for modeling and analyzing and designing base isolated highway bridges can be found in Priestley's book [35].

6.4 Analysis Methods

Generally speaking, the analysis of a base isolated structure is more complicated than the analysis of a conventional structure which may not need to withstand seismic loadings. This situation is not helped by the lack of guidelines for the analysis of base isolated structures [42]. By resorting to first principles, however, a number of avenues exist for simplifying the analysis. First, there is no need to take into account nonlinear response of the pier elements. The pier masses and their own modes of vibration should be part of an analysis, however, since isolation of the lower-frequency modes involving the deck mass may increase the importance of higher-frequency modes. Because the bridge superstructure is expected to remain essentially elastic, even for the largest seismic events, its behavior can be modeled with linear elastic elements. During the preliminary stages of design, effects of soil-structure interaction and the coupling effects of the deck may be neglected. Refined analyses are required for detailed bridge design. For example, by modeling the deck and pier with linear beam elements having mass, a MDOF model can be used to account for coupling effects of the deck. The isolation system will be modeled with linear equivalent highly damped elements, or, more appropriately, with bilinear elements [35]. In practice a nonlinear time-history analysis is almost always conducted.

The AASHTO Guide Specifications [1] contain two analysis procedures for bridges having a variety of spans, geometric complexity, and the Seismic Performance Category. Procedure 1 is the single-mode method of analysis. Procedure 2 is the multi-mode method of analysis. Both the single and multi-mode methods of analysis for seismic isolation design assume that energy dissipation of the isolation system can be expressed in terms of equivalent viscous damping, and the stiffness of the isolation system can be expressed as an effective linear stiffness. These assumptions simplify the required complexity of analysis enormously. These guidelines stipulate that for isolation systems without self-centering capabilities, or for isolation systems where the effective damping (% of critical) exceeds 30 percent, a three-dimensional nonlinear time-history analysis shall be performed.

1. **Single Mode Spectral Analysis :** The single mode method of analysis given in the AASHTO Standard Seismic Specifications [2] is appropriate for seismic isolation design. The only difference is that the statically equivalent force is determined associated with the displacement across the isolation bearings, and the effective linear stiffness of the isolators used in the analysis shall be calculated at the design displacement.

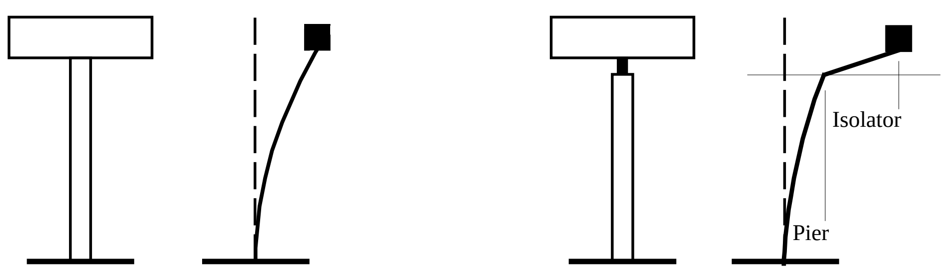
2. **Multimode Mode Spectral Analysis :** The guidelines given in Section 5.4 of the AASHTO Standard Seismic Specifications are also appropriate for the equivalent linear response spectrum analysis of an isolated structure with the two modifications. First, the isolation bearings are modeled by use of their effective stiffness properties determined at the design displacement. In the second modification, the ground response spectrum is modified to incorporate the damping of the isolation system.

3. **Time History Analysis :** A nonlinear time-history analysis is generally considered to be the most complete and time-consuming structural analysis method. When applied correctly, however, it can provide the most reliable results for the variation of forces and displacements during earthquake ground motion. The frequency content of ground motion accelerations should be scaled so they closely match the appropriate ground response spectra for a particular site. In addition, the analytical model should incorporate the nonlinear deformational characteristics of the isolation system.

A comparison of AASHTO recommended analytical methods applied to a 4-span continuous bridge is presented by Mayes [29]. The California Department of Transportation (CALTRANS) has also proposed two methods for the analysis of bridge isolated with bearings whose hysteresis behavior can be appropriately represented by a bi-linear model. In the first CALTRANS proposed method, the hysteresis behavior of a base-isolated regular bridge is idealized by a bi-linear model in the direction of consideration. Then with the hysteresis loop of the entire base-isolated bridge in hand, an empirical model is used to determine the effective period and equivalent viscous damping ratio of the bridge. In the second CALTRANS proposed method, an empirical model is used to determine the effective stiffness and equivalent damping ratio of isolation bearings rather than the base-isolated bridge itself. A modal strain energy method combined with the concept of component energy ratio was utilized to formulate the "composite damping ratio" of the entire base-isolated bridge [41].

6.5 Combination of Isolator and Support Properties

When a seismic base isolator is installed between the top of a bridge column and beneath the deck structure, the parameters of the column combine with those of the isolator. The overall characteristics of the superstructure and isolation system reflect a combination of individual composite dynamic parameters [43, 5].



(a)Non-isolated (b)Isolated

Figure 6.5: Lateral Direction Mode Shapes

For most practical situations, the superstructure mass is much greater than the column mass, and hence the first mode of vibration is dominated by displacements of the superstructure. The second mode, in comparison, involves lateral motion of the top of the column with little movement of the superstructure (see, for example, Figure 6.5).

6.6 Example of Nonlinear Time History Bridge Analysis

In this section we use ALADDIN for a three dimensional nonlinear time-history analysis of a 4 span base isolated highway bridge. The particulars of this example are taken from the text of Priestley et al. [35]. The longitudinal view of the bridge geometry is shown in Figure 6.7. The pier and deck section properties are shown in Figure 6.9. Capacities of the piers and isolators are listed in Table 6.1. The isolators are assumed to have elastic-perfectly plastic behavior, which means no strain hardening after yielding. The deck and piers are assumed to remain elastic during the dynamic responses. We choose the 1940 El Centro S00E record (see Figure 6.6) as the input earthquake with the peak ground acceleration scaled to $0.5g$.

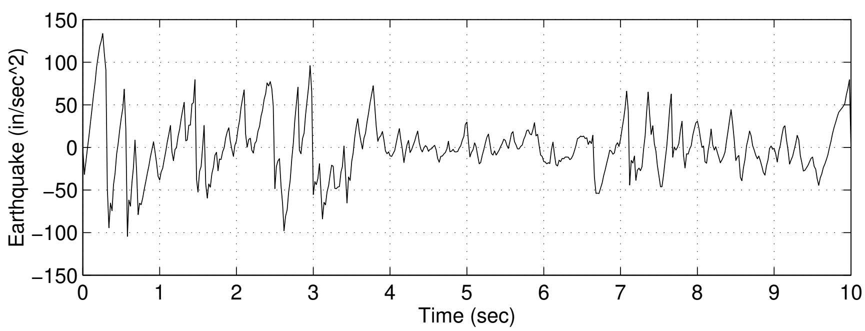


Figure 6.6: 1940 El Centro S00E Record

The finite element mesh is shown in Figure 6.8. The deck and columns are modeled with elastic 3-D frame elements (i.e., element type `FRAME_3D`). Only the isolators are modeled as 3-D nonlinear fiber elements with bi-linear shear-displacement response (i.e., element type `FIBER_3D`). The lower ends of the piers and abutments are fully fixed. We also assume that there are no longitudinal deformations and twisting at the both ends of the bridge deck. In total there

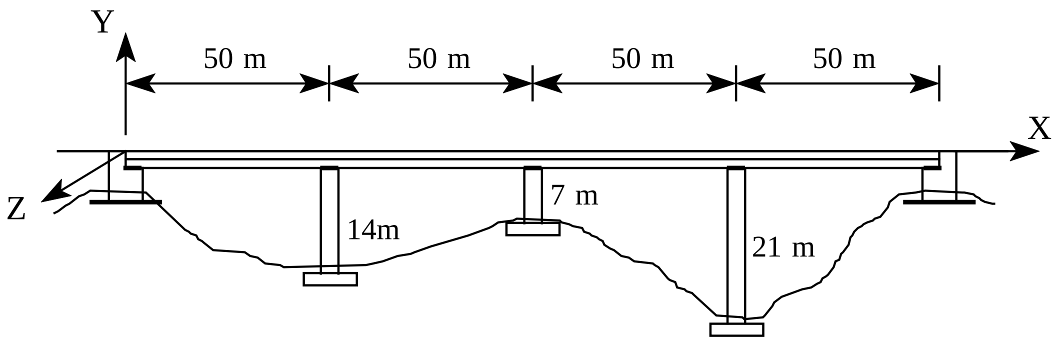


Figure 6.7: Elevation Plan of Isolated Bridge

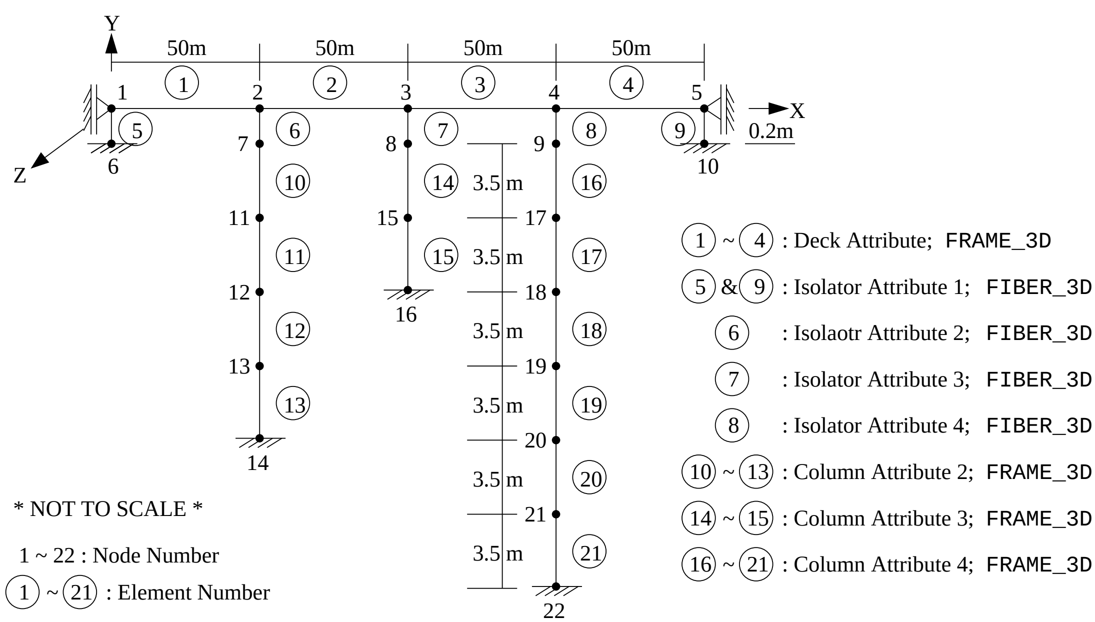
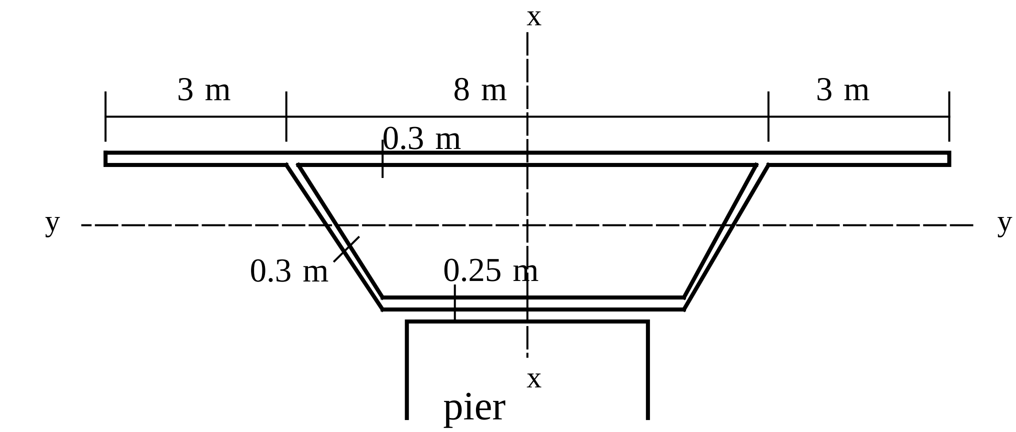
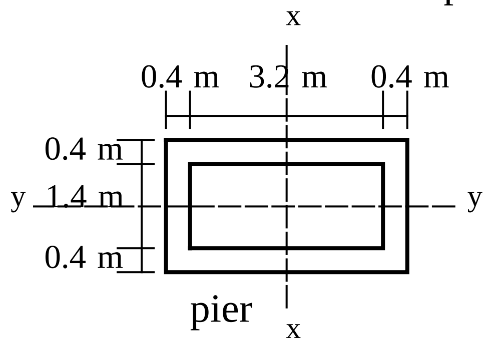


Figure 6.8: Finite Element Mesh of Isolated Bridge





	deck	pier
area(m^2)	6.88	4.16
Ix(m^2)	87.24	7.39
Iy(m^2)	5.26	0.67
w(kN/m)	200	104

Figure 6.9: Pier and Deck Section Properties

	secant stiffness (kN/m)	yield strength (kN)
isolator 1,5 at abutments	5300	2100
isolator 2	16100	2100
isolator 3	11500	2100
isolator 4	47000	2100
pier 2	30700	
pier 3	124400	
pier 4	13650	

Table 6.1: Pier and Isolator Capacities

are 21 elements, 8 different element attributes and 98 degrees of freedom in this model.

Eigenvalue Analysis : The following fragment of code shows the essential details of code needed to compute the first two modes of vibration, and their natural periods.

ABBREVIATED INPUT FILE

```
/* Form stiffness matrix and mass matrix */
   stiff = Stiff();
   mass  = Mass([1]);

/* Compute first two eigenvalues, periods, and eigenvectors */

   no_eigen    = 2;
   eigen       = Eigen(stiff, mass, [no_eigen]);
   eigenvalue  = Eigenvalue(eigen);
   eigenvector = Eigenvector(eigen);

   T1 = 2*PI/sqrt( eigenvalue[1][1] );
   T2 = 2*PI/sqrt( eigenvalue[2][1] );
```

The first two modes of vibration for both the non-isolated and isolated bridges are shown in the three dimensional plots of Figures 6.10, 6.11, 6.12 and 6.13. We can see from these plots that the non-isolated bridge deck deforms in *cosine* and *sine* curves. The columns deform elastically. For the isolated bridge, the deck undergoes a rigid body displacement. Most of the deformations in the piers occur at isolators. Also note that the natural periods for the first two modes have been significantly shifted, from (0.71 sec, 0.57 sec) to (2.02 sec, 1.97 sec).

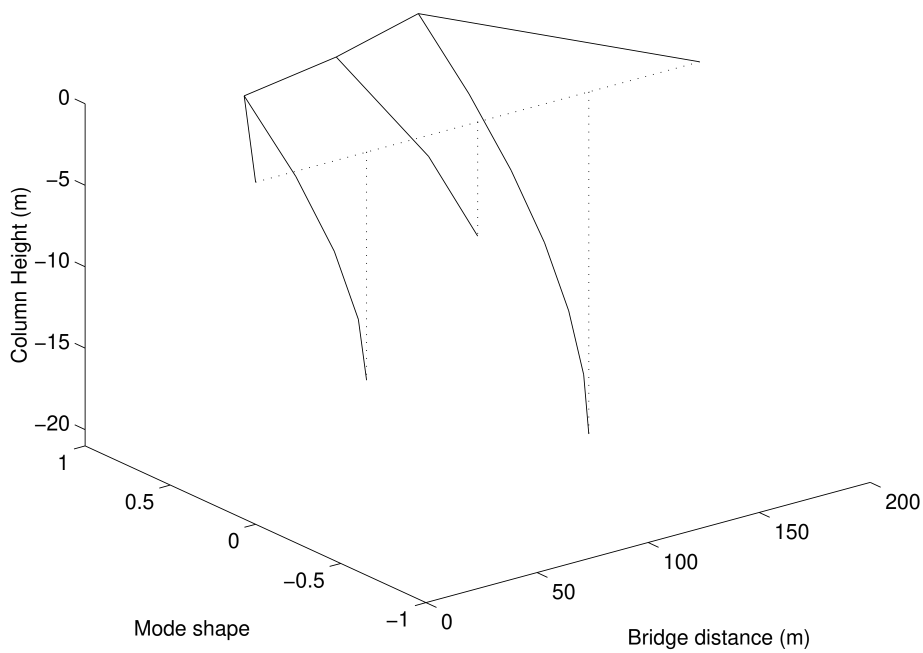


Figure 6.10: First Mode of Non-Isolated Bridge, T1 = 0.71 sec

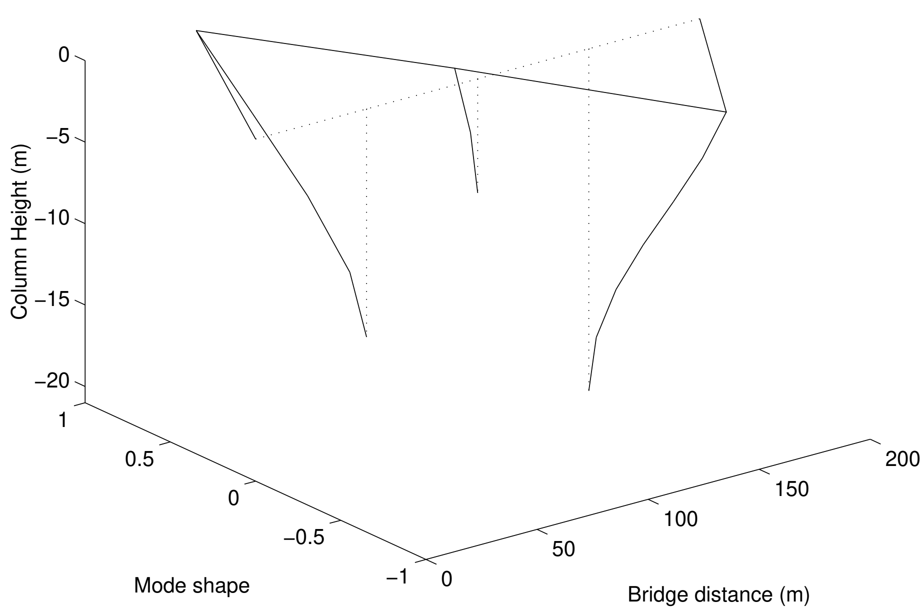


Figure 6.11: Second Mode of Non-Isolated Bridge, T2 = 0.57 sec

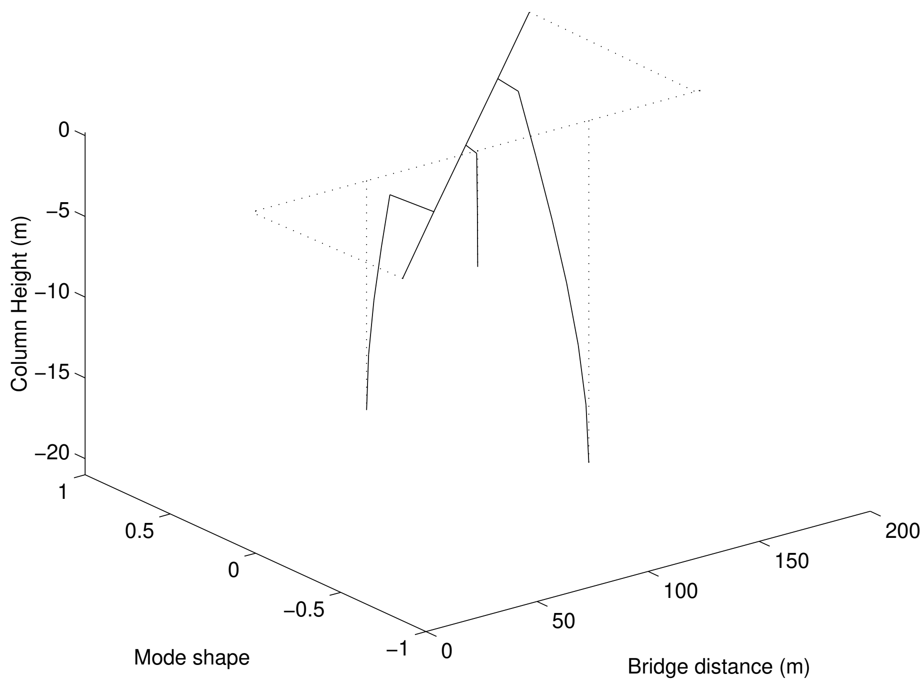


Figure 6.12: First Mode of Isolated Bridge, T1 = 2.02 sec

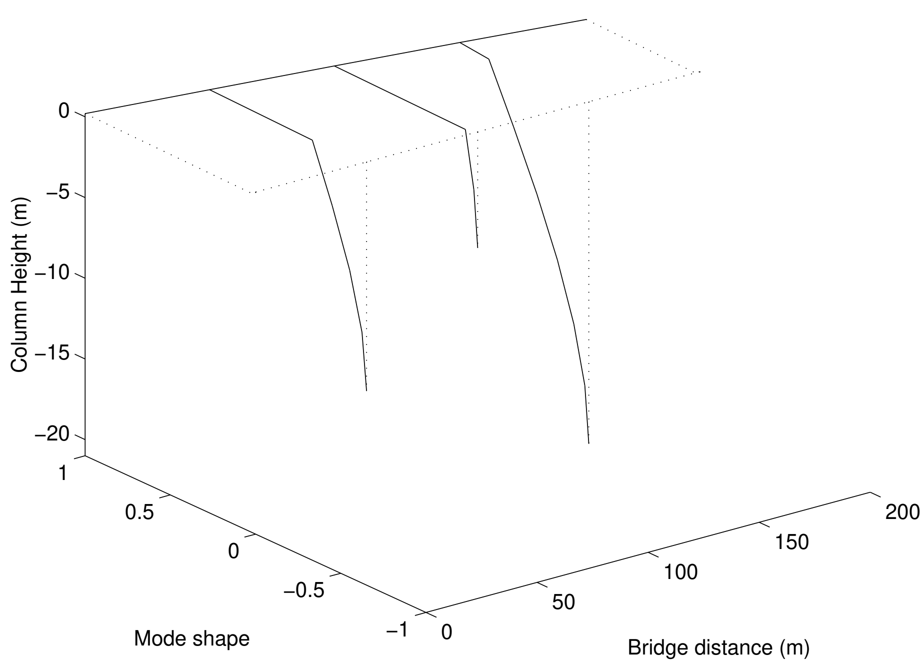


Figure 6.13: Second Mode of Isolated Bridge, T2 = 1.97 sec

Nonlinear Time History Analysis : We now extend the previous analysis to time-history analysis. The time-history analysis assumes equivalent viscous damping of 5% in the form of Rayleigh damping. A second time-history analysis is computed for a non-isolated bridge, thereby enabling comparison of the isolated and non-isolated responses, and assessment of the benefits of base isolation. The following script of code shows the essential details of the non-linear Newmark analysis:

ABBREVIATED INPUT FILE

```
while(stepno=1; stepno <= total_stepno; stepno=stepno+1) {

/* Update time and step no */
   time   = time + dt;

/* Compute effective incremental loading */
   if( stepno <= dimen[1][1] ) then {
      P_ext = -mass*r*Elcentro[stepno][1];
   } else {
      P_ext = -mass*r*(0.0 m/sec/sec); }

   dPeff = P_ext - P_old + ((4/dt)*mass+2*damp)*velocity + 2*mass*accel;

/* Compute effective stiffness from tangent stiffness */
   Keff = stiff + (2/dt)*damp + (4/dt/dt)*mass;

/* Solve for d_displacement, d_velocity */
   dp = Solve( Keff, dPeff);
   dv = (2/dt)*dp - 2*velocity;

/* Compute displacement, velocity */
   new_displ    = displ + dp;
   new_velocity = velocity + dv;

/* Check material yielding and compute new stiffness */
   ElmtStateDet( dp );
   stiff = Stiff();

/* Compute new internal load, damping force, and acceleration */
   Fs = InternalLoad( new_displ );
   Fd = damp*new_velocity;
   new_accel = mass_inv*( P_ext-Fs-Fd );

/* tolerance is satisfied, update histories for this time step */
```

```
    UpdateResponse();
    P_old    = P_ext;
    Fs_old   = Fs;
    Fd_old   = Fd;
    displ    = new_displ;
    velocity = new_velocity;
    accel    = new_accel;
}
```

Figure 6.15 through 6.30 summarize a variety of displacement, shear force, and energy computations versus time for the isolated and non-isolated bridge structures. Points to note are:

1. The deck displacements at the pier top are shown in Figure 6.15 and 6.16 for both the non-isolated and isolated bridges. As expected, the isolated bridge undergoes larger displacements than the non-isolated bridge. For example, at Pier 4 the maximum displacement of the isolated bridge is about 3.2 times larger than the maximum displacement of the non-isolated bridge (see Figure 6.19). Also, because the isolators exhibit nonlinear behavior, the isolated bridge has a $0.02m$ residual displacement after the earthquake response has ceased.

2. The shear forces at the bottom of the piers are plotted in Figure 6.17 and 6.18. Let us consider shear forces in the non-isolated bridge first. Pier 3 has the shortest height – it has the highest stiffness, and attracts the highest concentrations of shear force accordingly. For the isolated bridge, the isolators have been designed so that the effective stiffness and displacements of the piers are similar. Hence, the structural response is quite regularized. The isolators also reduce the maximum bottom shear force in pier 3 by a factor of 6 (see Figure 6.20). Another thing we want to point out is that

once the isolators have yielded, displacements of the bridge deck can occur without a corresponding increase in isolator shear forces. This observation is consistent with our assumption of elastic-perfectly plastic behavior.

3. The force-displacement responses for the isolators are shown in Figure 6.21 through 6.24. Noting that the isolator ductility μ_D may be defined as

$$\mu_D = \frac{\Delta_P}{\Delta_{DE}} + 1,$$

and the effective global ductility μ_G as

$$\mu_G = \frac{\Delta_P}{\Delta_S + \Delta_{DE}} + 1,$$

where Δ_P, Δ_S and Δ_{DE} are defined in Figure 6.14, we can calculate the ductilities of each pier-isolator system.

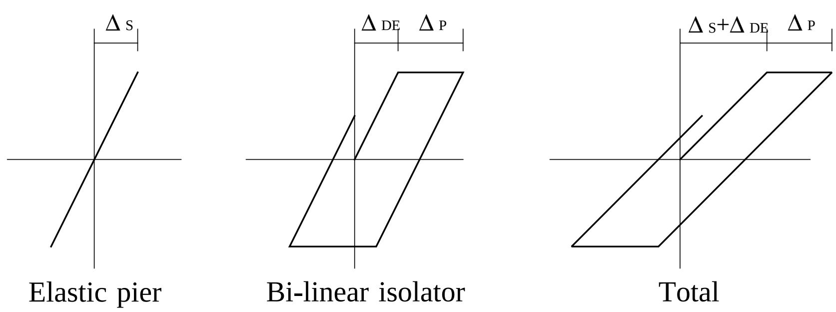


Figure 6.14: Definition of Structure Ductility

	abutments 1,5	pier 2	pier 3	pier 4
μ_D	1.41	1.64	1.45	2.78
μ_G	1.41	1.42	1.41	1.40

Table 6.2: Ductility of Pier-Isolator System

The results are summarized in Table 6.2. It is evident that by designing the individual isolators for a different ductility demand, the overall bridge system can be designed so that the ductility demand on the pier systems (i.e., pier + isolator) is about the same. This strategy of design regularizes the structural response.

4. Figures 6.25 and 6.26 plot components of the energy balance analysis for the non-isolated and isolated bridges, respectively. From Figure 6.27 we can see that the energy history of the isolated bridge is much higher than the non-isolated bridge. This is because displacements in the isolated bridge response are much larger than the non-isolated bridge, and because of the plastic strain energy (hysteresis energy) of the isolators.

5. The shear force-displacement response diagrams for the isolators (see Figure 6.21 through 6.24) can be used to compute contours of elastic and plastic shear strain energy history. The results of these computations are plotted in Figure 6.28. Because the non-isolated bridge response is assumed to be elastic, the internal strain energy caused by stiffness forces will gradually return to zero after the earthquake ends. But for the isolated bridge, the internal strain energy (also caused by stiffness forces) has a residual energy after the earthquake ends. This residual energy is the plastic shear-strain energy (or hysteretic energy) dissipated by the isolators. The latter results are shown in Figure 6.29.

6. The bridge strain energy is equal to the sum of strain energies in the isolators, piers and deck. Therefore the strain energy of the deck and piers is equal to bridge strain energy minus total isolator strain energy. The time-history of these strain energy components is shown in Figure 6.30. Because the strain energy components are roughly proportional to the square of the

bridge displacements, higher strain levels of energy in the bridge components means higher possibility of damage. Figure 6.30 clearly shows that the peak values of strain energy in the isolated bridge deck are lower than those in the conventionally supported bridge deck. Hence, we conclude that the isolation system protects the bridge superstructure, as required.

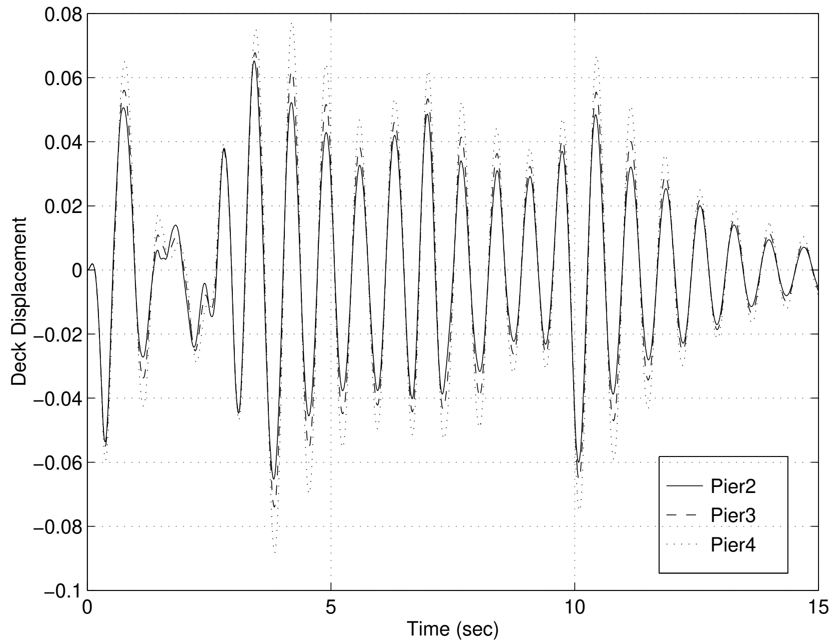


Figure 6.15: Deck Displacement of Non-Isolated Bridge

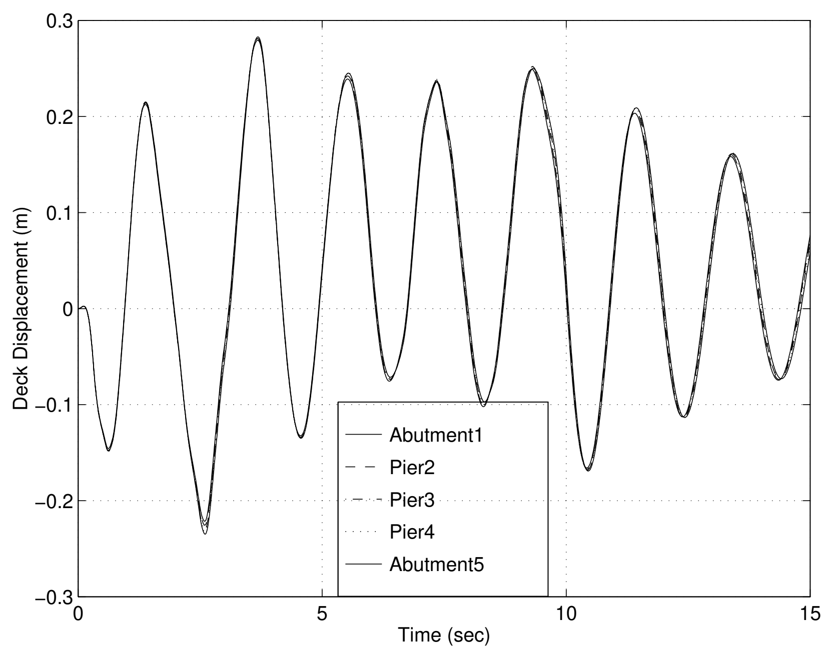


Figure 6.16: Deck Displacement of Isolated Bridge

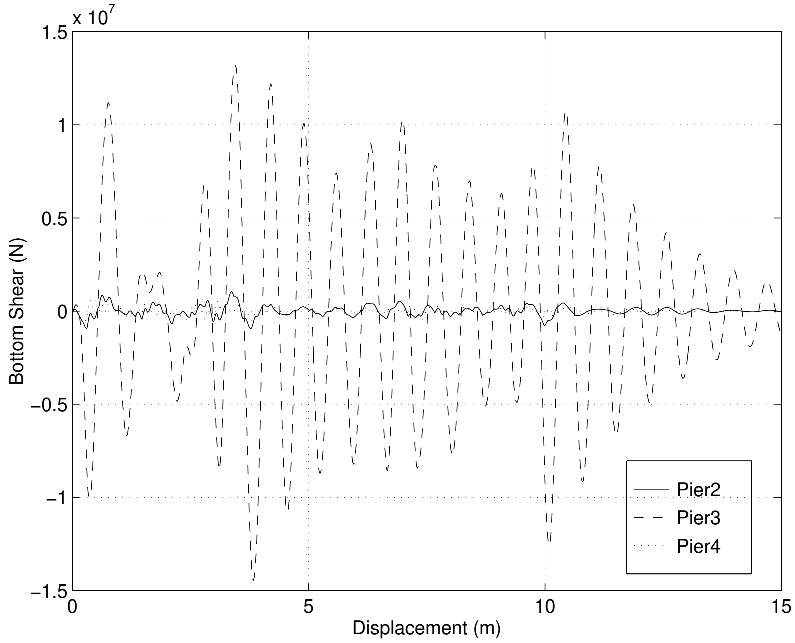


Figure 6.17: Bottom Shear of Non-Isolated Bridge

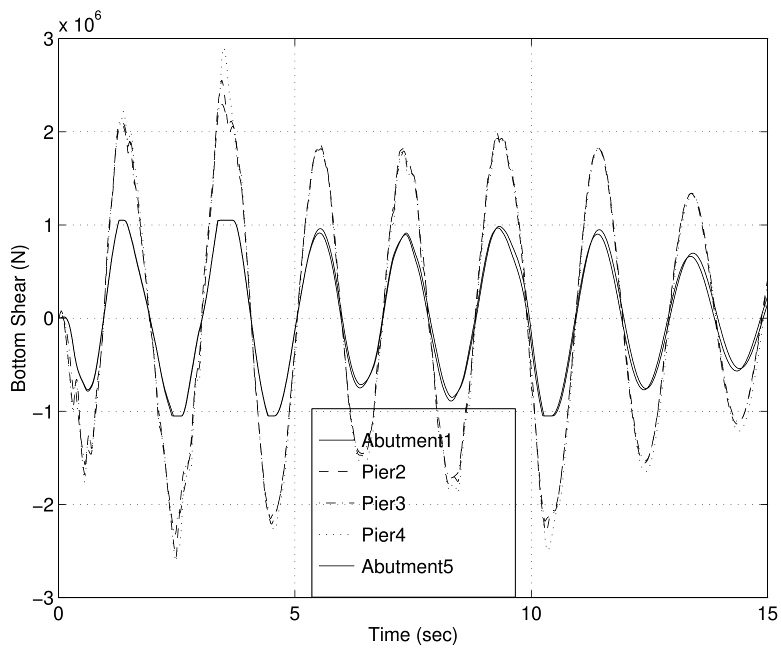


Figure 6.18: Bottom Shear of Isolated Bridge

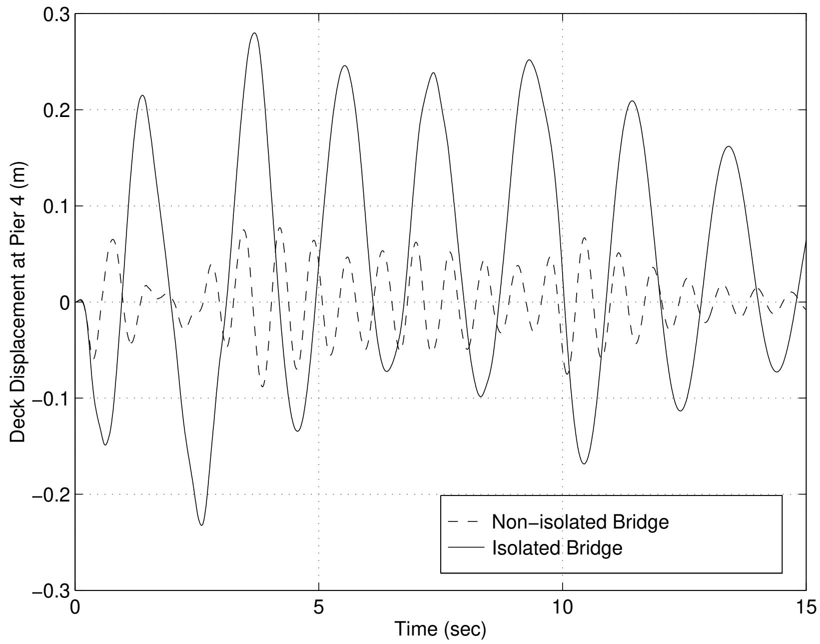


Figure 6.19: Deck Displacement Comparison at Pier 4 of Bridges

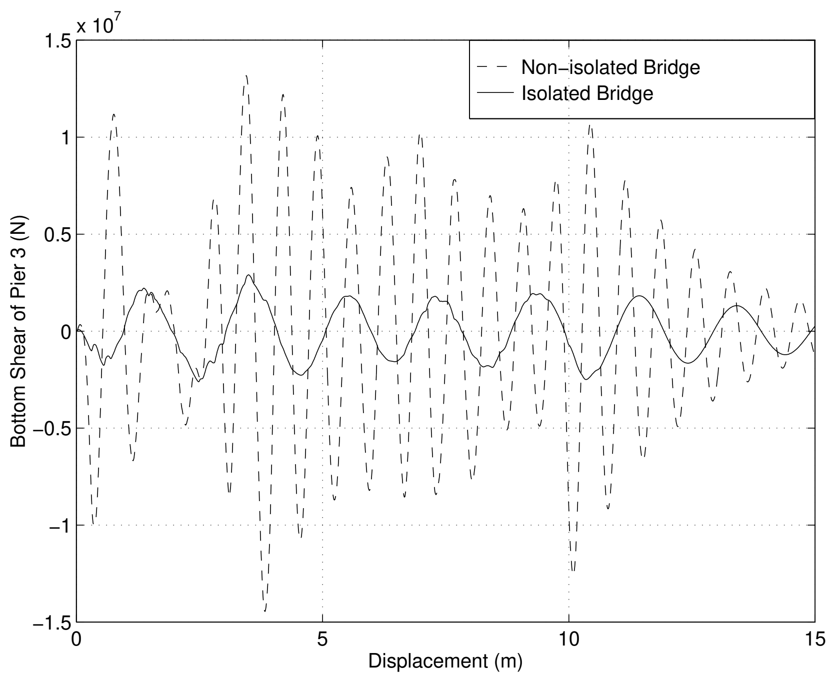


Figure 6.20: Bottom Shear Force Comparison at Pier 3 of Bridges

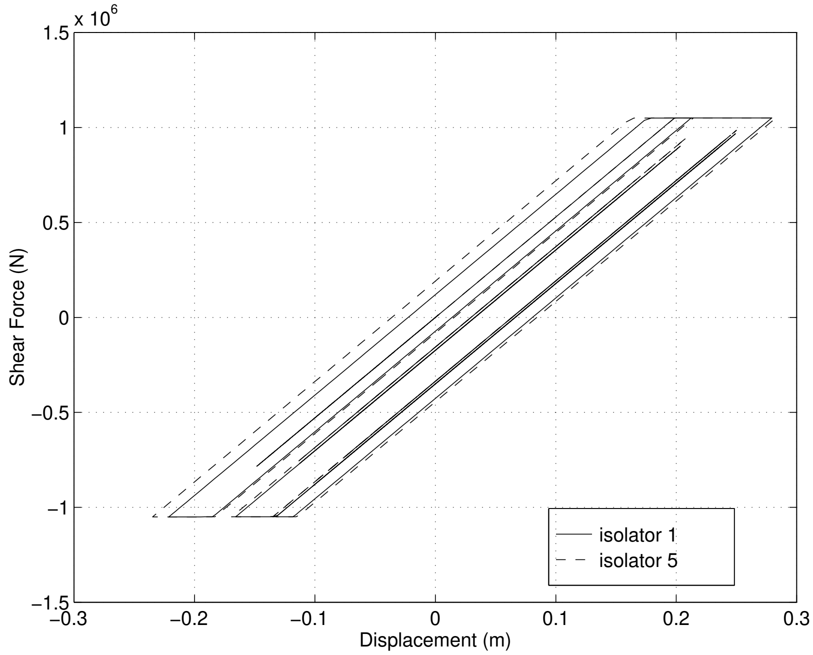


Figure 6.21: Force-Displacement Response of Isolators at Abutment 1,5

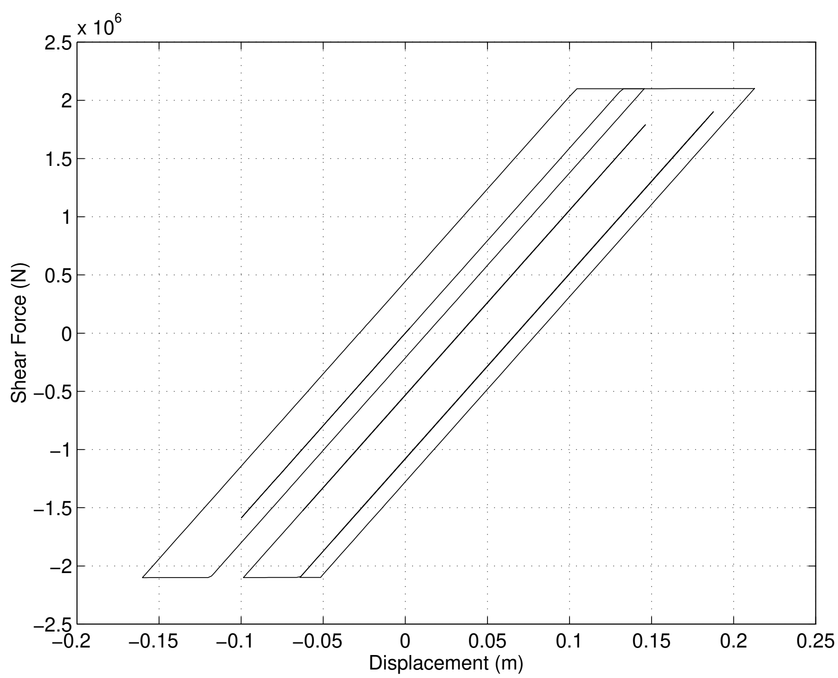


Figure 6.22: Force-Displacement Response of Isolator at Pier 2

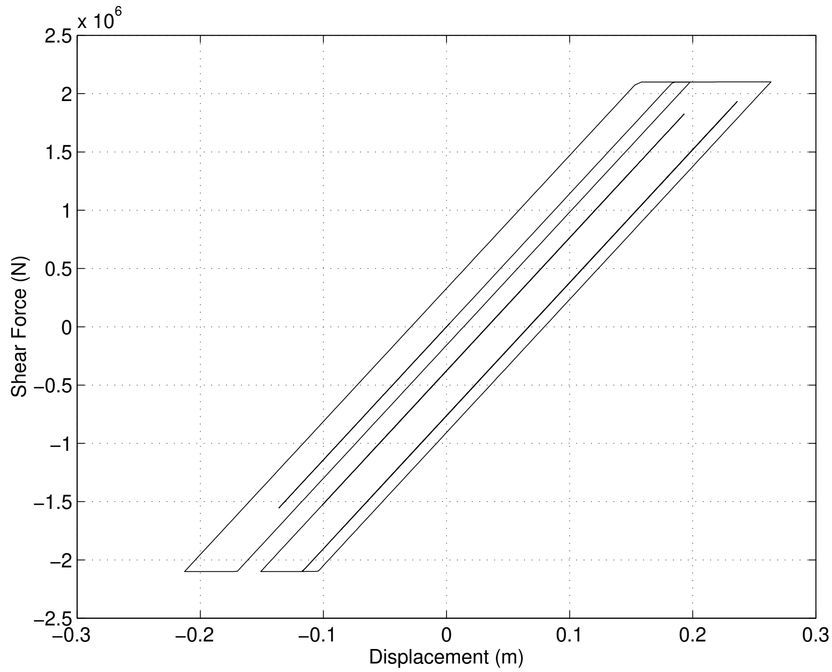


Figure 6.23: Force-Displacement Response of Isolator at Pier 3

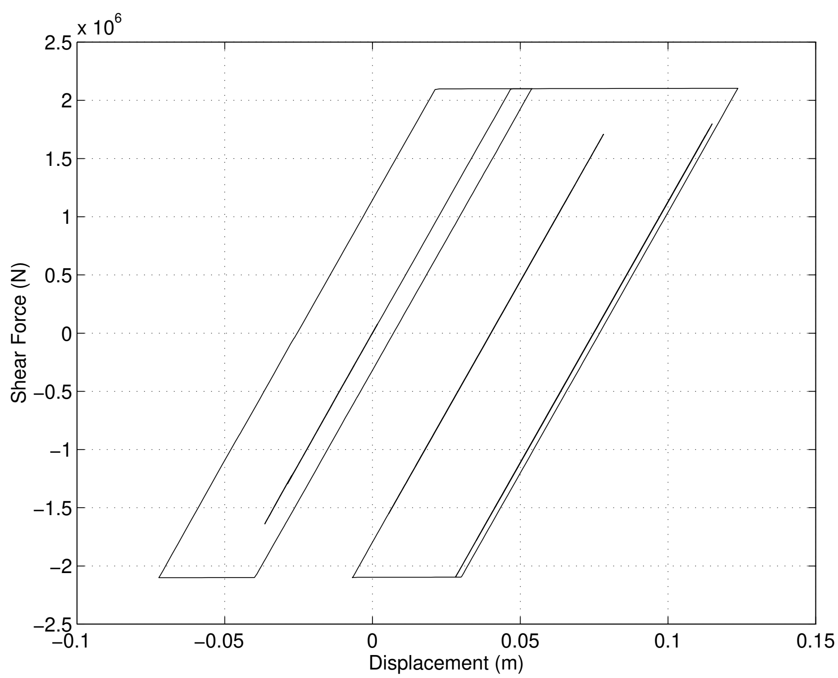


Figure 6.24: Force-Displacement Response of Isolator at Pier 4

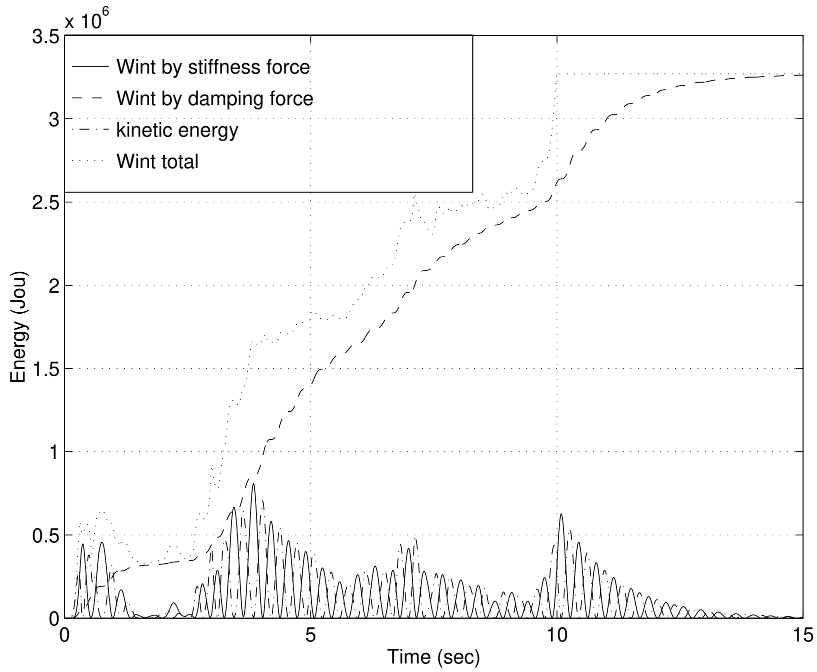


Figure 6.25: Energy History of Non-Isolated Bridge

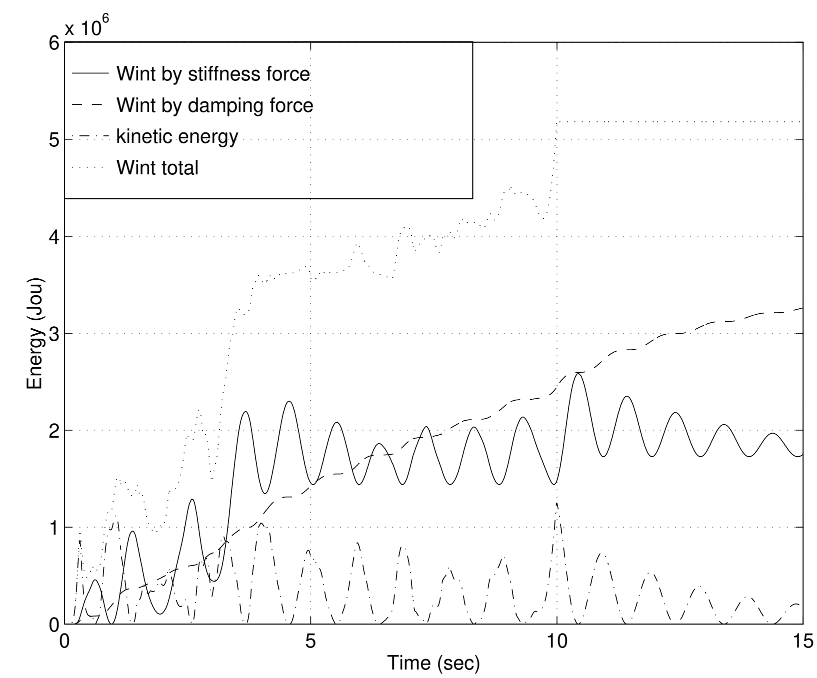


Figure 6.26: Energy History of Isolated Bridge

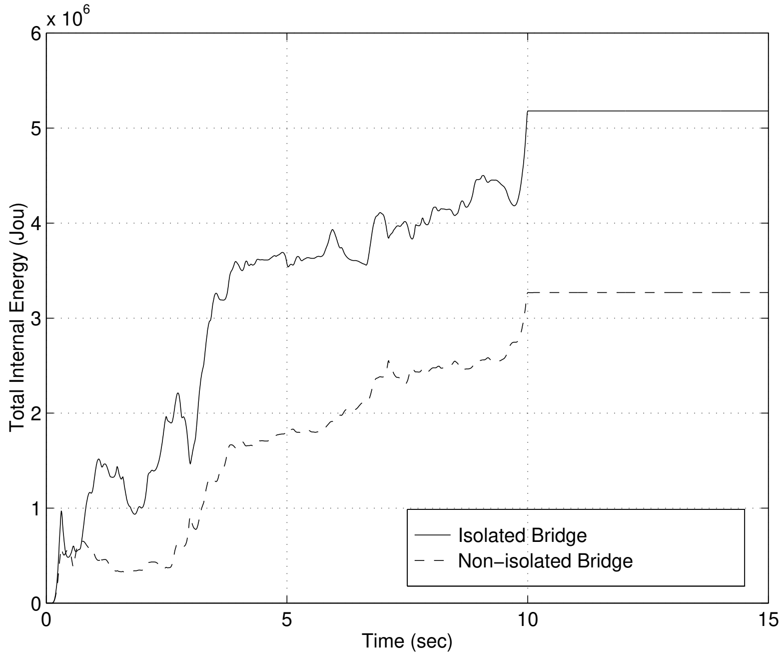


Figure 6.27: Energy History Comparison of Non-Isolated Bridge and Isolated Bridge

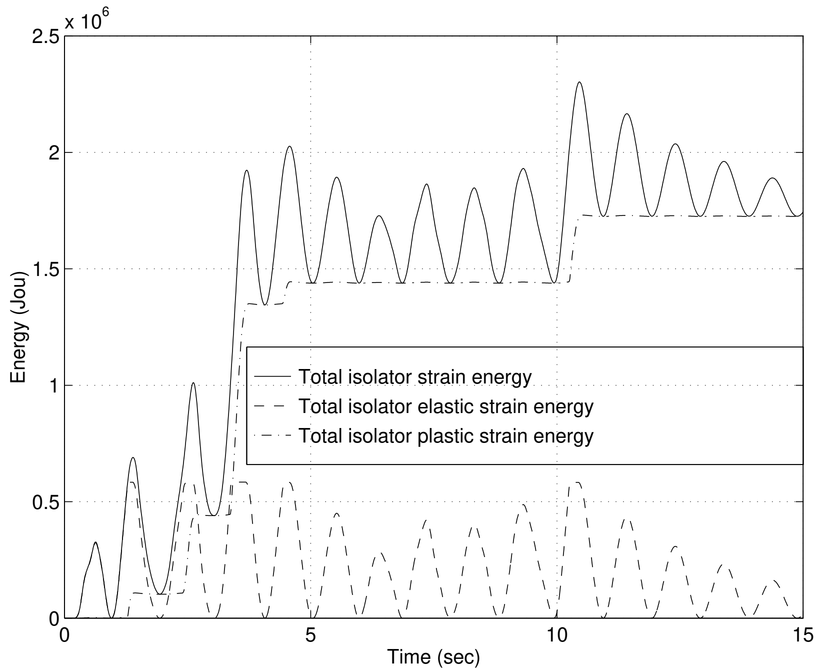


Figure 6.28: Elastic and Plastic Strain Energy of Isolators

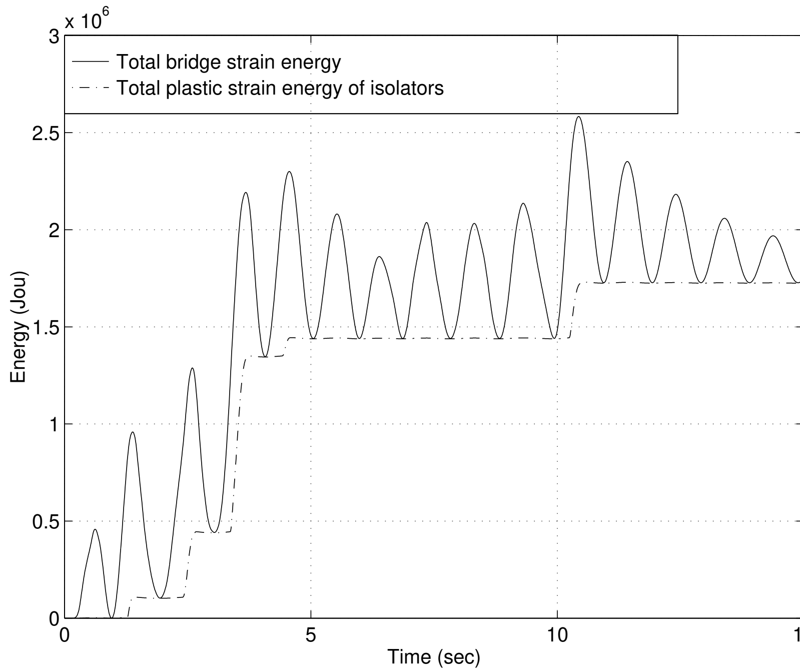


Figure 6.29: Strain Energy History of Isolated Bridge

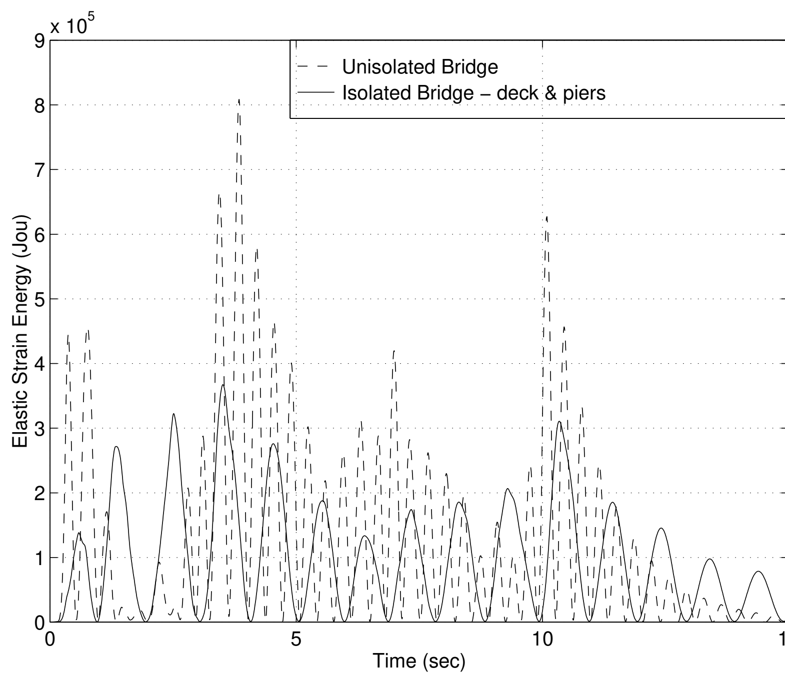


Figure 6.30: Strain Energy History Comparison of Deck + Piers

CHAPTER 7

Conclusions and Future Work

7.1 Conclusions

In the general area of computational technology for engineering, the 1990s are shaping up to be the decade of global networking and multimedia. These advances will allow for the prototyping of interactive computing environments tailored to the life cycle development of complex engineering systems. For these environments to be commercially successful, they will need to support development of engineering systems from multiple viewpoints, and at multiple levels of abstraction. Engineers should be provided with the tools to integrate once disparate disciplines. While a strong need for the merging of matrix computations with finite elements (and optimization, control and graphics) is already evident, many other possibilities exist. For instance, now that many companies are engaging in business practices that are geographically dispersed, engineers will soon expect connectivity to a wide range of information services – electronic contract negotiations, management of project requirements, design rule checking over the Internet, and availability of materials and construction services are likely candidates. The participating computer programs and networking infrastructure will need to be fast and accurate, flexible, reliable, and of course, easy to use [17, 47].

With these observations in mind, our long-term goal for ALADDIN is to provide engineers with such a computational problem-solving environment. Version 1.0 of ALADDIN is simply a first step in this direction. For Version 1.0 the emphasis in development has focused on the basic system specification, and its application to matrix and finite element problems. This work involved the determination of data types, control structures, and functions that would allow engineers to solve a wide variety of problems, without the language becoming too large and complicated, and without extensibility of the system being compromised.

This dissertation contains material that will become ALADDIN Version 2. A new fiber element that includes flexural and shear deformations, and is capable of modeling bi-linear material behavior has been added to the finite element library for Version 2. Behavior and accuracy of this new element has been verified through studies of the hysteretic force-displacement relations in a lead-rubber isolator. Furthermore, a nonlinear time-history analysis of a base isolated bridge has been computed by modeling all of the isolators with the fiber elements. The energy-balance calculations and plots of displacement versus time indicate that during a severe earthquake ground motion, the isolators protect the bridge superstructure and pier columns by reducing shear forces in the pier columns and by reducing the strain energy absorbed by the deck and columns.

From the beginning of this project the capabilities of the ALADDIN computational environment have been growing steadily. The suite of ALADDIN test problems has been expanded for Version 2.0, and now includes applications of design rule checking, nonlinear numerical algorithms, linear/nonlinear time-history analyses of structural systems involving multi-component earthquake ground motions, and energy balance calculations. The implementation of basic design rule checking required only very minor extensions to the language. Now nonlinear numerical analysis can be computed with a variety of algorithms. The application problems involving multiple components of ground motion input, and energy balance calculations are simply two cases of how structural analyses can be customized from the ALADDIN input file. There is no need to change the ALADDIN program source code.

In projects of this type, program user feedback is necessary component of ensuring the software works as advertised. Since the Internet is now the most powerful resource for information dissemination, in April 1996 a World Wide Web (WWW) home page for ALADDIN was created (the home page address is *http://www.isr.umd.edu/~austin/aladdin.html*). The home page contains an assortment of example problems and the source code to ALADDIN Version 1.0. So far more than 1070 copies of the program have been downloaded, and comments have been sent back to authors. According to those comments and our observations, our goals for the future work are as follows.

7.2 Future Work

Because ALADDIN is a problem-solving environment for engineers, the future work will be guided by the needs of engineering analysis. Extensions of the software to handle new types of engineering problems are required, as are enhancements to the ALADDIN software.

7.2.1 Engineering Side

The application areas we are currently interested in include:

1. **Hybrid Control of Base Isolated Highway Bridge :** This dissertation has been concerned with the construction of models for base isolated highway bridge structures. The next logical step in this research is hybrid control of isolated highway bridges. Studies of the active hybrid control require models of the bridge response due to earthquake loads, and the collection of response data for adjust of the response via feedback control. Our current thinking is that this extension can be done by adding just a few extra functions to the program, and perhaps by linking ALADDIN to the MATLAB Control and Optimization Toolboxes [48].

2. **Dynamics of Tethered Airship Systems :** This application area is concerned with the formulation of mathematical models for airships anchored to the ground by a Kevlar tether. The airships will fly at an altitude of 15000 ft, and will carry a surveillance payload. Before a variety of these systems can become operational, however, detailed mathematical models of the aerostat system that include interaction of the structural and mechanical systems, aerodynamics, and meteorology (e.g., wind, downdrafts, electrical storms) are needed.

3. **Structural Optimization :** We would like to extend the language specification so that engineers can describe design objectives, design constraints, and design parameters, for general engineering optimization problems in a compact manner. The issue of "compactness" is particularly relevant for optimization problems that involve finite element analysis because they often contain hundreds, and sometimes thousands, of constraints.

Further work is needed to determine how groups of similar constraints can be bundled into groups, and how ALADDIN's control structures can be exploited to write down these constraints in a compact manner. Once the issue of constraints is sorted out, the pathway for describing the design parameters and objectives should (hopefully) become clearer. Again, rather than implement our own home-grown optimization algorithms, links to the MATLAB Optimization Toolbox will be provided.

4. **Performance Based Design :** As pointed out in Chapter 1 the goal of performance-based engineering is to define performance objectives for various types of structures, and to control the risk associated with each limit-state to a pre-defined level of acceptability. ALADDIN's scripting language gives users the ability to collect and manipulate the data generated by the analyses, to choose performance objectives, and to define levels of acceptability. The work presented in this dissertation is one step in this direction, and there is a need to extend the energy calculations so that they are coupled to estimates of structural damage caused by nonlinear hysteretic deformations. Performance based studies should also account for system reliability, perhaps by computing statistics of extreme response caused by a small ensemble of ground motion inputs.

7.2.2 Software Side

Advances in the engineering capabilities of ALADDIN need to be accompanied by appropriate improvements in software support. These enhancements include:

1. **New Library of Functions :** We have found that when a new element is added to the program, some related functions may be needed in

the matrix, finite element or kernel libraries. For example, the function `ElmtStateDet()` was added to the finite element library for state determination of the fiber element deformations. The function `GetStress()` was added for design rule checking. There is a need to improve the instructions/guidelines for adding new library functions (perhaps by putting them on the web site).

2. **Internet User Interface for ALADDIN :** Future versions of ALADDIN will most likely operate in a client-server mode over the Internet, as shown in Figure 7.1.

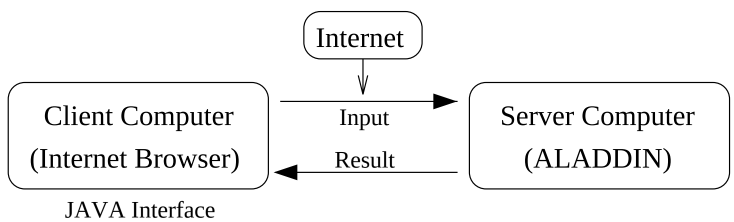


Figure 7.1: Relationship of ALADDIN and JAVA interface

In this arrangement of computing, analysis problems will be run on a high-speed ALADDIN server and displayed on an inexpensive client side computer. We are particularly interested in using JAVA technology to build the networking software and client-side platform independent user interfaces. We want to setup such a JAVA interface on our ALADDIN Web page and then let people from all over the world try out our program before downloading it. The main technical challenge is determining the protocol of communication between ALADDIN and the JAVA-based interface.

3. **User-defined Functions :** The current version of ALADDIN does not permit the use of user-defined functions and procedures in the input file.

When we first began working on ALADDIN, our feeling was that the input file language should be relatively simple – otherwise, why not simply code the whole problem in C? An easy way of controlling language complexity is to disallow user-defined functions in the input file, and this is what we have done for ALADDIN Version 1.0. Now that the matrix and finite element libraries are working, this early decision needs to be revised. We would like to combine the finite element analysis with discrete simulated annealing optimization algorithms, along the lines proposed by Kirkpatrick [24]. The latter could be succinctly implemented if user-defined functions were available. The current version of ALADDIN assumes that all user-defined variables are global. When user-defined functions are added to ALADDIN, notions of scope should also be added to variables (this could be implemented via an array of symbol tables). We also need to create a suite of example problems to show how these user-defined functions work.

4. **Assembling Input from Component Input Files :** To promote reuse of problem specifications, we need to create mechanisms for assembling an input file from component input files, as shown in Figure 7.2.

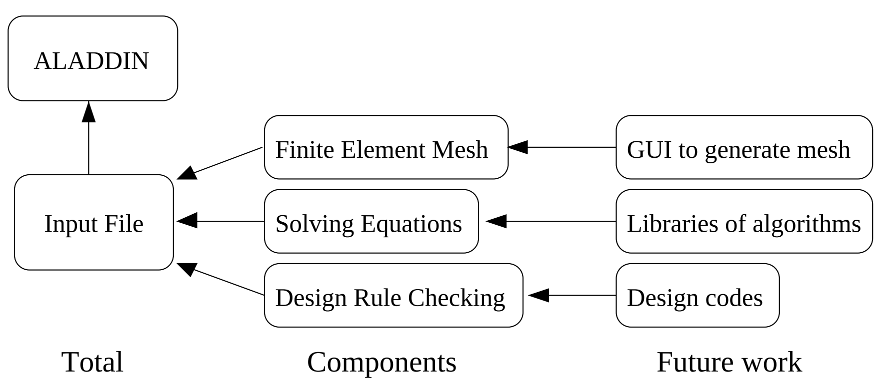


Figure 7.2: Future Input File for ALADDIN

Designing a graphical interface that would allow a user to incrementally assemble problem components is a very challenging research problem.

APPENDIX A

Fiber Elements in ALADDIN

Fiber elements can be used for modeling of bi-linear materials with flexural and shear effects. There are four types of fiber elements in ALADDIN; `FIBER_2D`, `FIBER_2DS`, `FIBER_3D`, and `FIBER_3DS`. The numbers `2` and `3` indicate two- or three- dimensional elements, respectively. The difference between the element types (`FIBER_2D`, `FIBER_3D`) and (`FIBER_2DS`, `FIBER_3DS`) lies in the shear modeling. `FIBER_2DS` and `FIBER_3DS` can be used to model elements having more than one shear property in the sub elements. If the element shear properties are homogeneous, use of `FIBER_2D` and `FIBER_3D` will reduce the memory allocation requirements and the required calculation time.

A number of new language features have been added to ALADDIN for finite element computation using fiber elements. First, a new problem specification parameter `GaussIntegPts` has been created for fiber elements. It defines how many Gauss-Lobatto integration sections (besides the two end sections) you want for each element. Any integer number between 2 to 10 is valid (the default number is 2).

A fiber element attribute is defined with the `ElementAttr()`$\{\cdots\}$ function. Four character string arguments should be specified inside the braces:

`type`, `section`, `material`, and `fiber`. The definitions of `type`, `section`, and `material` are the same as for the other finite elements. The character string for the `fiber` attribute is defined with `FiberAttr(a,"b")`$\{\cdots\}$ function, where `a` is number of fibers in the element, `b` is a character string for the name of the fiber attribute. The $\cdots$ part inside the braces includes definitions of four matrices : `FiberCoordinate`, `FiberArea`, `FiberMaterialAttr`, and `FiberMaterialMap`. `FiberCoordinate` defines the coordinate of each fiber in the element cross section. `FiberArea` defines the cross section area of each fiber in the element cross section. `FiberMaterialAttr` defines different material types of fibers. `FiberMaterialMap` defines the mapping number of each fiber's material type.

A.1 Two Dimensional Fiber Elements

The fiber element has the same degrees of freedom as the traditional plane frame element.

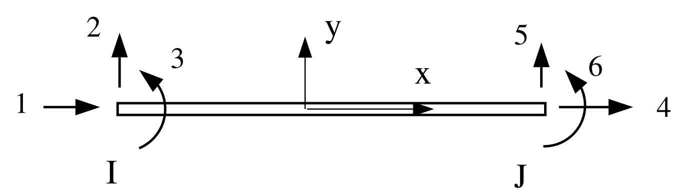


Figure A.1: Schematic of Two Dimensional Fiber Beam/Column Element

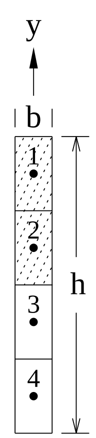
A.1.1 2D Fiber Element with Homogeneous Shear Properties

`FIBER_2D` is the element type name for a two dimensional 2-node fiber element with homogeneous shear properties.

Section Properties : The element section properties are defined for the whole element. The section properties include the element's total cross-section area, `area` (`area` may be computed from the section `depth` times its `width` or `bf`). The `shear_factor` for shear effect in the element is 1.2 by default. In order to calculate a mass matrix for dynamic analysis, either `unit_weight` in the section properties, or `density` in the material properties must be supplied. The fiber element supports only lumped mass matrix.

Material Properties : The element material properties are defined for the whole element. The material properties include Poisson's ratio ν (`poisson`, 0.3 by default), and `density`. The material properties in element type `FIBER_2D` also include the shear modulus (`G`), tangent shear modulus (`Gt`), and shear yield stress (`shear_yield`). Note that `G`, `Gt` and `shear_yield` represent the shear properties for the whole element. Young's modulus (`E`), tangent Young's modulus (`Et`) and yield stress fy (`yield`) are not defined here; they will be defined in fiber attribute.

Fiber Attribute : `FiberCoordinate` includes y-coordinate of each fiber in the element cross section. `FiberMaterialAttr` includes the Young's modulus, tangent Young's modulus, and yield stress.



	Young's Modulous	Tangent Young's Modulus	Yield Stress
	E1	Et1	fy1
	E2	Et2	fy2

Figure A.2: Example of Two-Dimensional Fiber Element Modeling

Example : The following script contains the ALADDIN statements needed to model the 4 fiber elements shown in Figure A.2.

ABBREVIATED INPUT FILE

```
AddElmt( 1, [1,2], "elmt_attr" );

/* Define Element Attribute */

ElementAttr("elmt_attr"){ type     = "FIBER_2D";
                          section  = "sec_name";
                          material = "mat_name";
                          fiber    = "fib_name";  }

/* Define Section Properties */

SectionAttr("sec_name") { area = b*h;
                          width = b; depth = h;
                          unit_weight = 1 N/m;  }

/* Define Material Properties */

MaterialAttr("mat_name"){ poisson = nu;  G = G;  shear_yield = fv; }

/* Element is modeled with 4 fibers and equally meshed. */

no_of_fiber = 4;

fiber_coord = [ 3/8*h, 1/8*h, -1/8*h, -3/8*h ];
fiber_area  = [ b*h/4, b*h/4,  b*h/4,  b*h/4 ];

/* Element is composed of two materials. */

fiber_attr = [ E1,   E2;
               Et1,  Et2;
               fy1,  fy2 ];

/* Fiber No.1 and No.2 are material 1 */
/* Fiber No.3 and No.4 are material 2 */

fiber_map = [ 1, 1, 2, 2 ];

/* Define Fiber Attribute */

FiberAttr( no_of_fiber, "fib_name" ) { FiberMaterialAttr = fiber_attr;
                                       FiberCoordinate   = fiber_coord;
                                       FiberArea         = fiber_area;
                                       FiberMaterialMap  = fiber_map;  }
```

A.1.2 2D Fiber Element with Different Shear Properties

`FIBER_2DS` is the element type name for a two dimensional 2-node fiber element with different shear properties.

The section properties and material properties are the same as `FIBER_2D`. One exception is that there is now no need to define shear properties in material properties, since they will be defined in `FiberMaterialAttr` matrix.

Fiber Attribute : `FiberMaterialAttr` of `FIBER_2DS` includes the Young's modulus, tangent Young's modulus, yield stress, shear modulus, tangent shear modulus, and shear yield stress.

ABBREVIATED INPUT FILE

```
/* Element is composed of two different shear properties materials. */

fiber_attr = [ E1,   E2;
               Et1,  Et2;
               fy1,  fy2;
               G1,   G2;
               Gt1,  Gt2;
               fv1,  fv2 ];
```

A.2 Three Dimensional Fiber Elements

The three-dimensional fiber element has the same degrees of freedom as the traditional space frame element.

A.2.1 3D Fiber Element with homogeneous Shear Properties

`FIBER_3D` is the element type name for a three dimensional 2-node fiber element with homogeneous shear properties.

Section Properties :

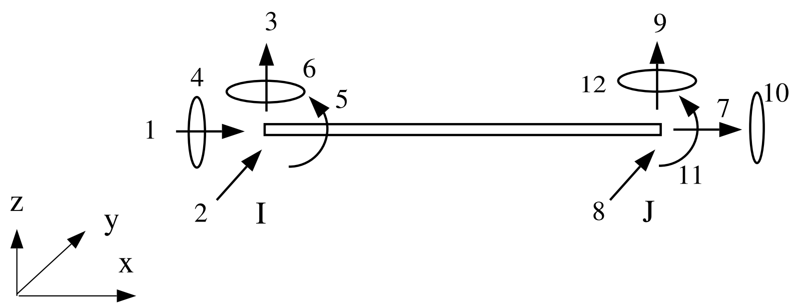


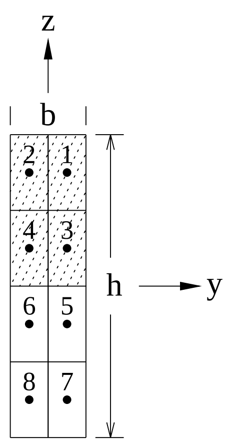
Figure A.3: Schematic of Three Dimensional Fiber Beam/Column Element

The element section properties defined here are for the whole element. The section properties include element's total cross-section `area` (`area` may be computed from the section `depth` times its `width` or `bf`). The `shear_factor` for shear effect in the element is 1.2 by default. The torsional constant (`J`) has to be given in section properties. For dynamic analysis, in order to calculate mass matrix, `unit_weight` in section properties or `density` in material properties must be given. The fiber element supports only a lumped mass matrix.

Material Properties :

The element material properties are defined for the whole element. The material properties include mass `density` and Poisson's ratio ν (`poisson`, 0.3 by default). The material properties in `FIBER_3D` also include the shear modulus (`G`), tangent shear modulus (`Gt`), and shear yield stress (`shear_yield`). Note that `G`, `Gt` and `shear_yield` represent the shear properties for the whole element. Young's modulus (`E`), tangent Young's modulus (`Et`) and yield stress fy (`yield`) are not defined here, they will be defined in fiber attribute.

Fiber Attribute : `FiberCoordinate` defines both y-coordinate and z-coordinate of each fiber in the element cross section. `FiberMaterialAttr` defines different material types of fibers, it includes the Young's modulus, tangent Young's modulus, and yield stress.



	Young's Modulous	Tangent Young's Modulus	Yield Stress
	E1	Et1	fy1
	E2	Et2	fy2

Figure A.4: Example of Three-Dimensional Fiber Element Modeling

Example : An element is modeled with 8 fiber elements as shown in Figure A.4. The ALADDIN code needed to construct this section is:

ABBREVIATED INPUT FILE

```
AddElmt( 1, [1,2], "elmt_attr" );

/* Define Element Attribute */

ElementAttr("elmt_attr"){ type     = "FIBER_3D";
                          section  = "sec_name";
                          material = "mat_name";
                          fiber    = "fib_name";  }

/* Define Section Properties */

SectionAttr("sec_name") { area = b*h;
                          width = b; depth = h;
                          J = 0.2 m^4;
                          unit_weight = 1 N/m;  }

/* Define Material Properties */

MaterialAttr("mat_name"){ poisson = nu;  G = G;  shear_yield = fv; }

/* Element is modeled with 8 fibers and equally meshed. */
```

```
/*  (4 rows by 2 columns) of fibers per cross section   */

no_of_fiber = 2*4;

fiber_coord=[  b/4,  -b/4,   b/4,  -b/4,    b/4,   -b/4,    b/4,   -b/4];
             3/8*h, 3/8*h, 1/8*h, 1/8*h, -1/8*h, -1/8*h, -3/8*h, -3/8*h];
fiber_area =[b*h/8, b*h/8, b*h/8, b*h/8,  b*h/8,  b*h/8,  b*h/8,  b*h/8];

/* Element is composed of two materials. */

fiber_attr = [ E1,   E2;
               Et1,  Et2;
               fy1,  fy2 ];

/* Fiber No.1 to No.4 are material 1 */
/* Fiber No.5 to No.8 are material 2 */

fiber_map = [ 1, 1, 1, 1, 2, 2, 2, 2 ];

/* Define Fiber Attribute */

FiberAttr( no_of_fiber, "fib_name" ) { FiberMaterialAttr = fiber_attr;
                                       FiberCoordinate   = fiber_coord;
                                       FiberArea         = fiber_area;
                                       FiberMaterialMap  = fiber_map;  }
```

A.2.2 3D Fiber Element with Different Shear Properties

`FIBER_3DS` is the element type name for a three dimensional 2-node fiber element with different shear properties.

The section properties and material properties are the same as `FIBER_3D`, except that there is no need to define shear properties in material properties; they will be defined in `FiberMaterialAttr` matrix.

Fiber Attribute : `FiberMaterialAttr` of `FIBER_3DS` includes the Young's modulus, tangent Young's modulus, yield stress, shear modulus, tangent shear modulus, and shear yield stress.

ABBREVIATED INPUT FILE

```
/* Element is composed of two different shear properties materials. */

fiber_attr = [ E1,   E2;
               Et1,  Et2;
               fy1,  fy2;
               G1,   G2;
               Gt1,  Gt2;
               fv1,  fv2 ];
```

BIBLIOGRAPHY

[1] AASHTO. *Guide Specifications for Seismic Isolation Design*. 1991.

[2] AASHTO. *Standard Specifications for Highway Bridges*. 1991.

[3] ABAQUS. *Example Problems Manual, version 5.2*. Hibbitt, Karlsson & Sorensen, Inc., Hibbitt, Karlsson & Sorensen, Inc., 1080 Main Str., Pawtuchet, RI 02860, 1992.

[4] *Unit Conversion Guide*. Fuels and Petrochemical Division of AICHE, 1990.

[5] Allred B. A., Shepherd R. Ultimate Restraint Considerations in Base-Isolated Bridges. In *Proceeding of the Third US-Japan Workshop on Earthquake Protective Systems for Bridges*, 1994. Session 4: Design Issues and Application 1, U9.

[6] Austin M. A., Chen X. G., Lin W-J. ALADDIN: A Computational Toolkit for Interactive Engineering Matrix and Finite Element Analysis. Technical Report TR95-74, Institute for Systems Research, University of Maryland, College Park, December 1995.

[7] 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*, 113(8):1642–1659, August 1987.

[8] 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*, 113(8):1660–1677, August 1987.

[9] Balling R. J., Pister K. S., Polak E. DELIGHT.STRUCT: A Computer-Aided Design Environment for Structural Engineering. *Computer Methods in Applied Mechanics and Engineering*, pages 237–251, January 1983.

[10] Bathe K. J. *Finite Element Procedures in Engineering Analysis*. Prentice-Hall, Englewood Cliffs, N.J., 1982.

[11] Chapra S. C., Canale R. P. *Numerical Method for Engineers*. McGraw-Hill, Inc., 1985.

[12] Clough R. W., Penzien J. *Dynamics of Structures*. McGraw-Hill, Inc., 1984.

[13] Cook R. D., Malkus D. S., Plesha M. E. *Concepts and Applications of Finite Element Analysis*. John Wiley & Sons, Inc., New York, 1989.

[14] Craig R. R. Jr. *Structural Dynamics – An Introduction to Computer Methods*. John Wiley & Sons, Inc., New York, 1981.

[15] Earthquake Engineering Research Center, UC Berkeley. National Information Service for Earthquake Engineering. *http://nisee.ce.berkeley.edu*.

[16] Filippou F. C., Issa A. Nonlinear Analysis of Reinforced Concrete Frames Under Cyclic Load Reversals. Technical Report EERC 88-12, Earthquake Engineering Research Center, Berkeley, 1988.

[17] Gallopoulos E., Houstis E. Computer as a Thinker/Doer : Problem-Solving Enviroments for Computational Science. *IEEE Computational Science and Engineering*, 1(2):11–23, 1994.

[18] Gallopoulos E., Houstis E., Rice J. R. Computer as Thinker/Doer: Problem-Solving Environments for Computational Science. *IEEE Computational Science & Engineering*, pages 11–23, summer 1982.

[19] Gere J. M., Timoshenko S. P. *Mechanics of Materials*. Wadsworth, Inc., 1984.

[20] Giannini R., Monti G., Nuti G., Pagnoni T. ASPIDEA: A Program for Nonlinear Analysis of Isolated Bridges Under Non-synchronous Seismic Action. Technical Report 5/92, Dipartimento di Ingegneria Civile Délle Acque e del Terréno, Universitá del' Aquila, 1992.

[21] Goudreau G. L. Computational Structural Mechanics : From National Defense to National Resource. *IEEE Computational Science and Engineering*, 1(1):33–42, 1994.

[22] Johnson S. C. YACC - Yet another Compiler Compiler. Technical report, AT&T Bell Laboratories, Murray Hill, New Jersey, 1975.

[23] Kernighan B. W., Pike R. *The UNIX Programming Environment*. Prentice-Hall Software Series, 1983.

[24] Kirkpatrick S., Gelatt C. D., Vecchi M. P. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, May 1983.

[25] Kronlof K. *Method Integration : Concepts and Case Studies.* John-Wiley and Sons, 1993.

[26] Lai S., Will G., Otani S. Model for Inelastic Biaxial Bending of Concrete Members. *Journal of Structural Engineering, ASCE*, 110(ST11):2563–2584, 1984.

[27] Mahasuverachai M. Inelastic Analysis of Piping and Tubular Stuctures. Technical Report EERC 82-27, Earthquake Engineering. Research Center, Berkeley, 1982.

[28] Maragakis E. M., Saiidi M. Development and Application of Simple Analytical Models Of Lead-Rubber Base Isolated Bridges. In *Proceeding of the Second US-Japan Workshop on Earthquake Protective Systems for Bridges*, pages 275–282, 1992. Technical Memorandum of PWRI No. 3196.

[29] Mayes R. L. Application of AASHTO Seismic Isolation Design and Analysis Requirements. In *Proceeding of the Second US-Japan Workshop on Earthquake Protective Systems for Bridges*, pages 221–237, 1992. Technical Memorandum of PWRI No. 3196.

[30] Mondkar D. P., Powell G. H. ANSR - 1 General Purpose program for Analysis of Nonlinear Structural Response. Technical Report EERC 75-37, Earthquake Engineering Research Center, U.C. Berkeley, December 1975.

[31] Nagarajaiah S., Reinhorn A. M., Constantinou M. C. Nonlinear Dynamic Analysis of 3-D Base-Isolated Structures. *Journal of Structural Engineering*, 117(7):2035–2054, 1991.

[32] Nye W. T., Riley D. C., Sangiovanni-Vincentelli A. L., Tits A. L. DELIGHT.SPICE: An Optimization-Based System for the Design of Integrated Circuits. *IEEE Trans. CAD Integrated Circuits and Systems, CAD-7*, pages 501–520, 1987.

[33] Orie D., Saiidi M., Douglas B. A Micro-CAD System for Seismic Design of Regular Highway Bridges. Technical Report CCEER 88-2, Civil Engineering Department, University of Nevada, Reno, June 1988.

[34] Penzien J., Imbsen R., Liu W. D. NEABS - Nonlinear Earthquake Analysis of Bridge Systems. Technical report, Earthquake Engineering Research Center, Berkeley, June 1981.

[35] Priestley M. J. N., Seible F., Calvi G. M. *Seismic Design and Retrofit of Bridges*. John Wiley & Sons, Inc., 1996.

[36] Przemieniecki J. S. *Theory of Matrix Structural Analysis*. McGraw-Hill, Inc., 1993.

[37] Robinson W. H. Lead-Rubber Hysteretic Bearings Suitable for Protecting Structures During Earthquakes. *Earthquake Engineering and Structural Dynamics*, 10:593–604, 1982.

[38] Saiidi M., Lawver R., Hart J. User's Manual for ISADAB and SIBA, Computer Programs for Nonlinear Transverse Analysis of Highway Bridges Subjected to Static and Dynamic Lateral Loads. Technical Report CCEER 86-2, Civil Engineering Department, University of Nevada, Reno, September 1986.

[39] Salter K. G. A Methodology for Decomposing System Requirements into Data Processing Requirements. *Proc. 2nd Int. Conf. on Software Engineering*, 1976.

[40] Schelling D. ENCE751: Advanced Probelms in Structural Behavior, Department of Civil Engineering, University of Maryland, College Park, 1993. *Class Notes.*

[41] Sheng L. H., Hwang J. S., Gates J. H. Current CALTRANS Analysis Methods of Bridges Isolated with Bi-Linear Hysteresis Bearings. In *Proceeding of the Third US-Japan Workshop on Earthquake Protective Systems for Bridges*, 1994. Session 4: Design Issues and Application 1, U8.

[42] Shenton H. W. III, Taylor A. W. Guidelines and Benchmarks for Analysis of Isolated Buildings. In *Analysis and Computation, Proceedings of the twelfth Conference held in conjunction with Structures Congress XIV*, pages 236–245, 1996.

[43] Skinner R. I., Robinson W. H., McVerry G. H. *An Introduction to Seismic Isolation.* John Wiley & Sons, Inc., 1993.

[44] Stroud A. H., Secrest D. *Gaussian Quadrature Formulas.* Prentice-Hall, Inc., 1966.

[45] Structural Engineers Association of California. Performance Based Seismic Engineering of Buildings. 1, April 1995. Part 1 : Interim Recommendations, Part 2 : Conceptual Framework.

[46] Taucer F., Spacone E., Filippou F. C. A Fiber Beam-Column Element for Seismic Response Analysis Of Reinforced Concrete Structures. Technical

Report UCB/EERC-91/17, Earthquake Engineering Research Center, UC Berkeley, December 1991.

[47] Tesler L.G. Networked Computing in the 1990's. *Scientific American*, 265(3):86–93, September 1991.

[48] The MathWorks, Inc. *The Student Edition of MATLAB: Version 4 User's Guide*. Prentice-Hall, Inc., 1995.

[49] Weaver, W. Jr., Gere, J. M. *Matrix Analysis of Framed Structures*. D. Van Nostrand Company, 135 West 50th Street, New York, NY 10020, 2 edition, 1980.

[50] Wilson E. L., Farhoomand I., Bathe K. J. Nonlinear Dynamic Analysis of Complex Structures. *Earthquake Engineering and Structural Dynamics*, Vol. 1:241–252, 1973.

[51] Wu T. L. DELIGHT.MIMO : An Interactive System for Optimization-Based Multivariable Control System Design. Technical Report UCB/ERL-M86/90, Department of Electrical Engineering, U.C. Berkeley, December 1986.

[52] Zienkiewicz O. C. *The Finite Element Method*. McGraw-Hill, Inc, 3 edition, 1986.

[53] Zienkiewitz O. C., Taylor R. L. *The Finite Element Method*. McGraw-Hill, Inc, 1989. For details on FEAP, see Chapter 15.