A SYSTEMS APPROACH TO NONLINEAR FINITE ELEMENT ANALYSIS OF SHELL STRUCTURES

Xiaoguang Chen and Mark Austin

Institute for Systems Research
University of Maryland
College Park, MD 20742

December 14, 1995

Abstract

This report describes a systems approach to the nonlinear finite element analysis of shell structures. The research objective is to understand the structure a small language and computational environment should take so that matrix and nonlinear finite element computations that can interact in a seamless manner.

One four node thick shell finite element and one eight node thick shell finite element is formulated and implemented in ALADDIN [1]. The finite elements are based on a three-dimensional continuum formulation, and are simplified by assuming a flat element geometry. Numerical experiments are presented for in-plane displacements of a flat plate, and out-of-plane bending of a cantilever structure. In each case, material nonlinearities are modeled with bi-linear and Ramberg-Osgood stress-strain curves. The report concludes with recommendations for further work in the areas of nonlinear finite element solution procedures, and enhancements to ALADDIN's problem solving infrastructure.

Contents

Chapter 1

Introduction

1.1 Problem Statement

Now that high-speed personal computers and engineering workstations with Internet connectivity are readily available, many engineering companies are eager to finds new ways of enhancing business productivity with computer-based support for engineering and business processes. A key characteristic of this development is expectations of computing that are much higher than just one or two decades ago. Whereas an engineer in the late 1970's might have expected a computer program to provide numerical solutions to an application-specific problem, the same engineer today might require engineering analyses, plus computational support for design code checking, optimization, and interactive computer graphics for visualization. In the near future, engineers will also expect connectivity to a wide-array of on-line information services – electronic contract negotiations, management of project requirements, and availability of materials and construction services are among the likely services. For many companies, the long-term objective of this development pathway is seamless integration of distributed engineering and business activities throughout the entire life cycle of a facility. The participating computer programs and networking infrastructure will need to be fast and accurate, flexible, reliable, and of course, easy to use.

Before this long-term objective can become a practical reality, however, some vexing systems integration issues must be resolved. A crucial problem is that in the 1970's and early 1980's, engineering software was written for problem solving on mainframe computers. For the most part, the software development process did not take into account the benefits of networking technology, and it did not consider how parts of a computer program might be designed for reuse at a later date. Many of these so-called "legacy computer programs" are written in dated languages, have poorly designed program architectures, and have functionality that has been blurred by maintenance operations [15]. They are not surprisingly, difficult to extend in functionality, and difficult to integrate with other applications. We have found, for example, that the integration of optimization and finite element packages can be a difficult and time consuming process, with the resulting software having a short life-cycle [2, 3, 4]. And yet, when finite element and optimization procedures are viewed simply as specialized matrix computations that could be coded as software libraries, it is difficult to see why these disciplines should not mesh together in a seamless way. In our opinion, the cause of the software integration barrier is an ad-hoc approach to software tool development in the first place.

The dilemma faced by companies with large economic investments in "legacy computer software" is that the shortcomings in software performance and lack of flexibility for problem solving must be pitted against the high cost of software replacement. Naturally, management would like the benefits of improved software communications without having to reinvest in the basic application-specific software. In practice, however, making decisions on a pathway for systems integration development is a difficult problem because the integration process lacks a theoretical foundation. This means that problems of methods development and systems integration must be solved on a case-by-case basis using empirical procedures and case-study problems.

There is a strong need for a framework of empirical guidelines that will assist engineers and management in the systems integration of computer programs. Such a framework should help engineers and management recognize functional similarities among software packages, and opportunities for software reuse when it exists. The framework should help company management decide when integration of commercial software systems is plausible, and when new applications software must be developed in order to achieve a desired level of functionality in computational assistance. A framework for two areas of computer program integration is needed:

Problem Area [1] : Integration of Custom-Built Code : The easier of the two problem areas is integration of code that can be designed and written from scratch (i.e. custom-built code). In deciding what structure this code should take, the basic questions to ask are:

- "What are the engineering/business processes that we are trying to automate ?"
- "What are the common features among these disciplines ?"
- "How do these disciplines communicate with one another ?"
- "What languages and models are needed to support these processes and their communication ?"

With the answers to these questions in hand, prototype computer programs can be designed, written, and tested. The success of these prototypes will guide the formulation of the systems integration recommendations, and the potential need for reformulation of disciplines to improve their communication with other disciplines. The results of research in this problem area should provide a benchmark for what can be achieved.

Problem Area [2] : Integration of Commercial Codes : The second, and much more difficult problem area, is integration of software packages when real world constraints (e.g. financial, personnel resources, time) preclude the development of code from scratch.

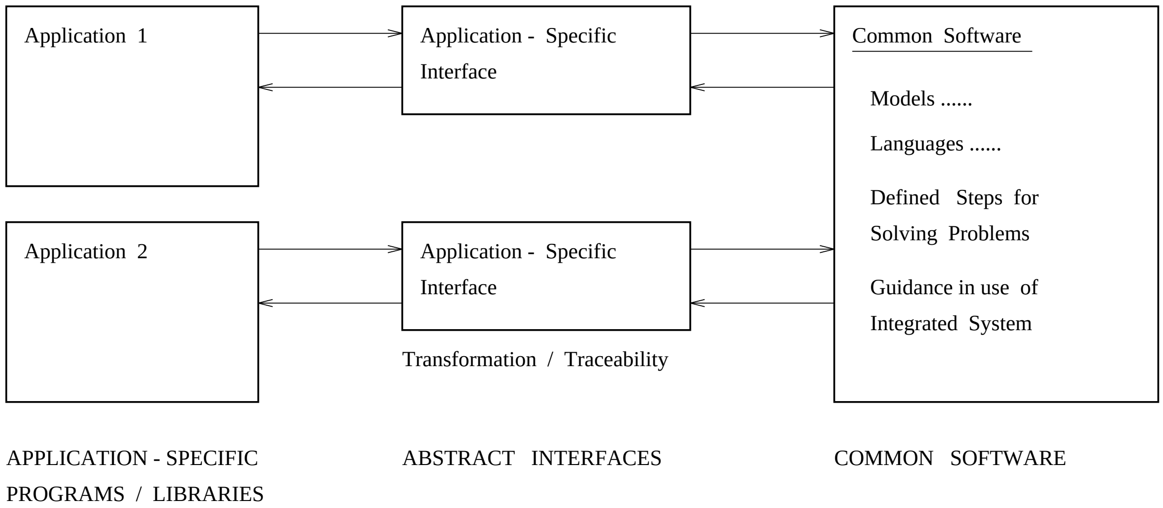

Figure 1.1: Architecture of Integrated Software Packages

Figure 1.1 shows the architecture for a family of integrated engineering/business software packages. The heart of the integrated system is the "common software module" shown on the right-hand side of Figure 1.1. Application specific software packages are shown along the left-hand side. When the languages used by the common software and the application package do not match, an abstract interface (or abstract interface object) is inserted to take care of the language and model transformations/mappings, and the definition of message passing protocols from one software package to another. The abstract interface should also trace "features required by the common software" onto "features in the applications software," thereby providing a mechanism for the identification of mismatches between the common software requirements and capabilities of the applications software. Ideally, the software component interfaces should be descriptive enough so that individual software modules can be replaced by anyone, and without loosing compatibility with earlier versions.

A first step in this direction is the Object Management Group's CORBA (an acronym for Common Object Request Broker Architecture), a standard for integrating object applications running in heterogeneous, distributed computing environments [10]. CORBA's Interface Definition Language (IDL) is a language-neutral way of defining or specifying how programming languages should access interfaces, invoke services, and handle exceptions. The IDL contains specifications for module, constant, and interface definitions. Client programs written in different languages communicate via mappings of the IDL interfaces to the native programming language(s).

A similar concept can also be found in Microsoft's OLE (originally an acronym for Object Linking and Embedding) technology [11]. OLE allows for the integration of program language-independent components through the use of an external binary standard – the standard allows legacy code components to be integrated alongside the newer object-oriented code. OLE allows objects to communicate via levels of abstraction that are higher than simply function interfaces, and for a client to ask an object whether or not it supports a particular feature before that client attempts to use that feature. This is traceability.

1.2 Objectives and Scope

The long-term objective of this work is formulation of a framework for guiding the systems integration of interactive computer programs and information services, as described in Section 1.1. The framework should allow for the integration of matrix and finite element analysis with optimization and other engineering fields, and for an engineer to install new code and functional capabilities into the system on the fly. With this said, it is clearly unrealistic to strive for a framework that is equally suited to all possible applications. One general purpose direction for development is to compile an all-encompassing collection of useful concepts from diverse domains. The main shortcoming of this approach is that the resulting software can easily become an unwieldy collection of concepts that lack harmony. This is precisely what we are trying to avoid ! At the other end of the development spectrum, a small number of primitive abstractions are written, out of which it is possible to construct powerful domain-specific concepts. This strategy of development has the disadvantage of requiring substantial up-front effort and expertise to build up the necessary application-specific programs [28].

Our research objective is to design and implement a small language that strikes a balance in these approaches. The language will support matrix and finite element analyses, and optimization, and allow an engineer to dynamically link user-defined application-specific functions into the computational system. The immediate research problem is to understand the structure the computational environment should take so that the various disciplines can interact in a seamless manner.

1.2.1 Preliminary Work

The preliminary result of this work is ALADDIN (Version 1.0), a computational toolkit for interactive engineering matrix and finite element analysis [1]. We have selected the integration of matrix and finite element computations as a study starting point because of the many features these disciplines share. 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 are viewed as dimensionless physical quantities. A detailed description of the representation for physical quantities, matrices of physical quantities, and the finite element library, may be found in Part IV of Austin, Chen, and Lin [1].

The development of ALADDIN has been inspired in part by the systems integration methods developed for the European ESPRIT Project [18]. A key result of the ESPRIT work is the recommendation that system specifications contain four components:

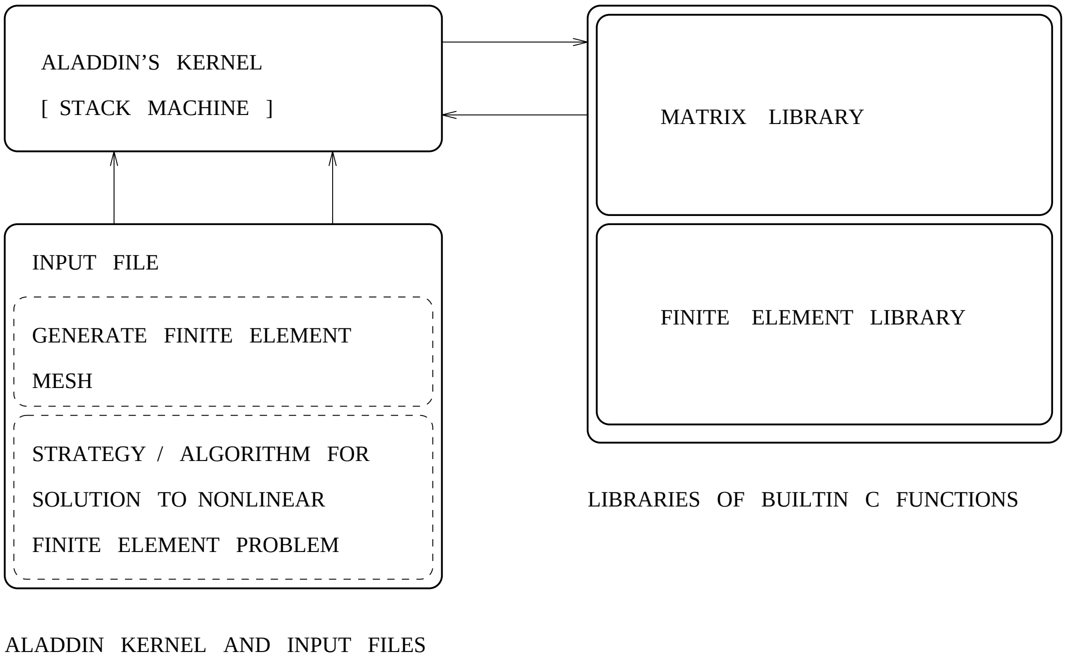

Figure 1.2: High Level Components in ALADDIN (Version 1.0)

[1] `A Model` : The model will include data structures for the information to be stored and manipulated.

[2] `A Language` : These language should be a composition of data, control structures, and functions [27]. Two important aspects of a language are its syntax and semantics:

`Syntax` : The syntax of a language defines its form. The form a language is composed of the terminal symbols (i.e. the keywords that make up the language) and the production rules (i.e how phrases are composed from terminals and subphrases).

`Semantics` : The semantics of a language assign meaning to its expressions and symbols.

Together the syntax and semantics of the language provide a means for describing, storing, and manipulating information within a problem domain.

[3] `Defined Steps and Ordering of the Steps` : The steps will define the transformations that can be carried out on system components (e.g. nearly all engineering processes will require iteration and branching).

[4] `Guidance for Applying the Specification` : Guidance includes factors such as descriptive problem description files and computer program documentation.

These four steps constitute the systems approach (or method) to software systems integration, and they are seen as a prerequisite to successful tool integration, and an extension of the software life-cycle. These benefits are not without precedence – for example, the DMAP (Direct Matrix Abstraction Program) for Nastran, and MAPOL (Matrix Algebra Problem Oriented Language) for ASTROS, have been developed along these lines. Both developments claim among their successes, an extended software life cycle [17].

Figure 1.2 shows the three main components of ALADDIN's architecture, and Figure 1.3 the relationship between the engineer and language, and components of the language and the underlying model in ALADDIN. In working out the details of the language/model interaction, and the mapping of components in Figure 1.2 onto Figure 1.3, the main question to ask is:

- "What components of a problem description should be handled by features of the language, and what components of a problem should be handled by ALADDIN's kernel and the matrix/finite element libraries ?"

This is a resource allocation problem that is complicated by trade-offs between the speed of interpreted ALADDIN language code, and the (relatively fast) speed of compiled C code, and ALADDIN language flexibility versus ease of program use. We do not want the ALADDIN's language to be so complicated/detailed that problem description files look just like C code. By guiding advances in the language, model, and engineering-application software libraries within the structure of Figure 1.3, and by using the language syntax and semantics to control the interaction between the matrix and finite element disciplines, the hope is that ALADDIN will evolve into an easy to use, extensible, integrated system that supports seamless communication among disciplines.

It is important to keep in mind that as the speed of Central Processing Units (CPU's) 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, as is maximum reuse of ALADDIN programming constructs from one problem domain to the next. To this end, we have emphasized in the language design: (1) liberal use of comment statements, (2) use of physical units in the problem description, (3) consistent use of function names and function arguments, (4) consistent use of physical quantities and matrices, and (5) consistent use of branching and looping programming constructs.

Matrix and engineering problems are solved using components of ALADDIN that are part interpreter-based, and part compiled C code. The matrix library has functions for matrix arithmetic, the solution of linear matrix equations, and the computation of the generalized eigenproblem for symmetric matrices. The matrix functions and corresponding language features have been used to compute the buckling loads in a slender elastic rod, and to simulate the stiffness method using matrix computations – for details, see Chapter 2 of Austin, Chen, and Lin [1]. Similarly, the finite element library has been used for the linear static analysis of two- and three-dimensional building and bridge structures, the linear dynamic analysis of two dimensional building structures using Newmark and Modal Analysis procedures, and the linear static analysis of various shell structures [1, 16].

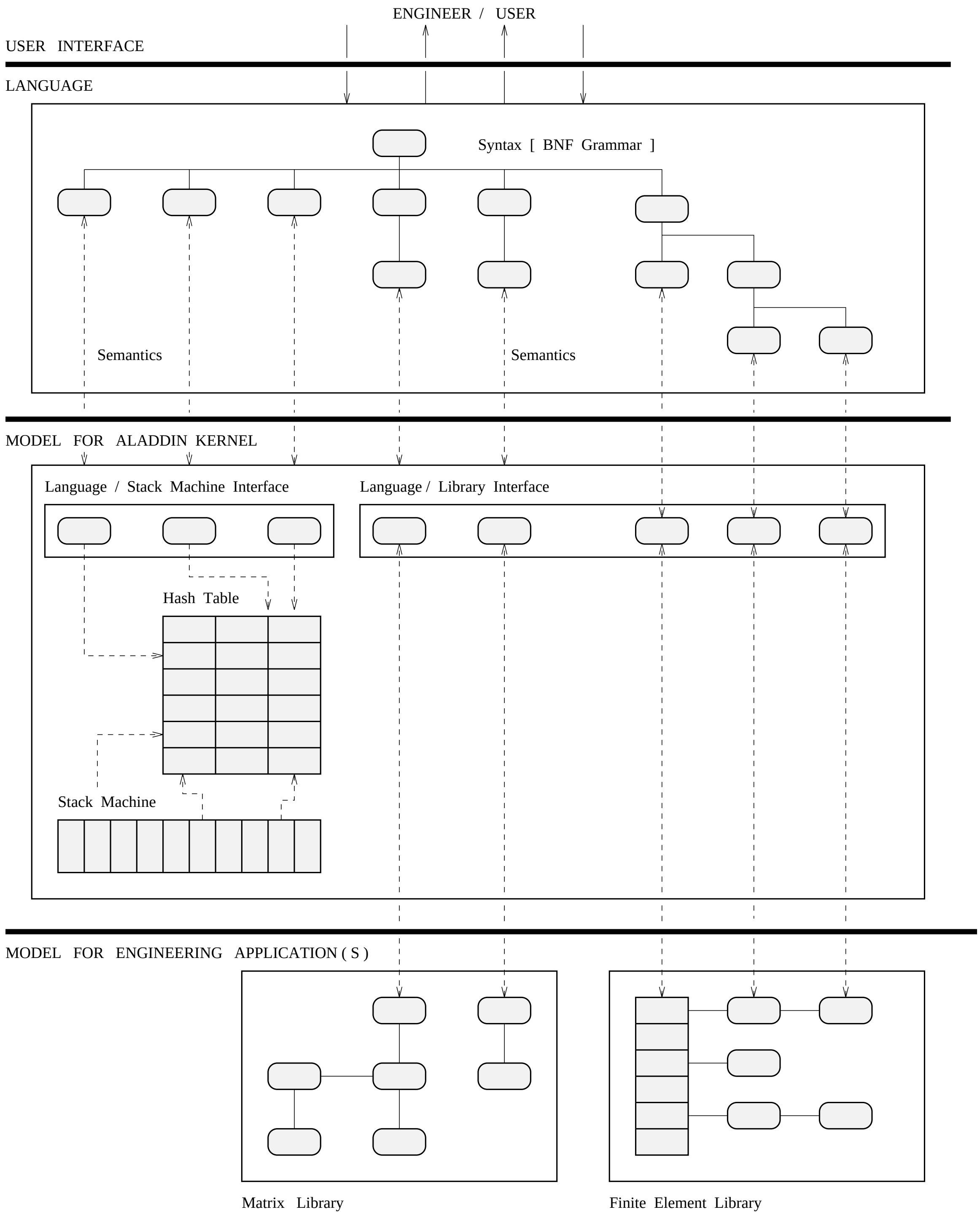

Figure 1.3: Interaction of Language and Underlying Model

1.2.2 Objectives and Scope of this Study

In this study we will extend ALADDIN's functionality to the solution of finite element problems containing material nonlinearities. We will validate ALADDIN's performance via the formulation and testing of one four node thick shell finite element, and one eight node thick shell finite element. In an effort to keep the formulation relatively simple, the shell finite elements will be based on a degenerated three-dimensional continuum formulation, and will have a flat geometry.

A key objective of this study will be to solve the nonlinear finite element problem in ways that maximize the reuse of ALADDIN's framework already in place. The *defined steps* will be specified in ALADDIN's input file, and will include numerical procedures capable of solving coupled nonlinear finite element equations. The preliminary work for this report is contained in Chapter 4 of Austin et al. [1], where BFGS-type algorithms are described and tested on ensembles of nonlinear numerical equations. The same algorithms will be applied in this report to the solution of nonlinear equations generated by the shell finite elements. Readers should note that when a tentative solution has failed, BFGS algorithms have the ability to back-step solutions, and to start forward again with a new set of algorithm control parameters and stress/strain data saved from a previous equilibrium state.

There are four chapters in this report. Chapter two describes the theory of the four and eight node shell finite elements, and numerical techniques for the plasticity computation. Results of the numerical experiments are given in Chapter three. Chapter 4 contains conclusions and suggestions for future work.

Chapter 2

General Shell Element

2.1 Introduction to Shell Element

Modern shell technology may be traced back to the fundamental work of Kirchhoff and Love in the second half of the nineteenth century [21]. Since then, considerable work has been done on the theory, analysis, modeling, and construction of shells. Nowadays, shell elements are load carrying components in a wide variety of modern engineering systems. Their expanded use is due in part to increased knowledge about the behavior and efficient load-carrying capabilities of shells, and in part, to the availability of new materials for construction [24].

Recent advances in shell technology include: (a) the development of computational models for the mechanical, thermal, and electromagnetic behavior of shells; (b) efficient discretization techniques, computational strategies and numerical algorithms, and (c) versatile and powerful software systems for the solution of large sets of nonlinear equations generated by complicated shell structures [24]. A comprehensive review of shell technology may be found in reference [22].

Many types of plate and shell finite elements have been proposed for the linear and nonlinear analyses of the plate, specific shell, and general shell structures [23]. Of all the possible approaches, the isoparametric formulation of the plate and shell elements for nonlinear analysis is particularly appealing because the formulation is both consistent and general, and the elements can be effectively employed in a variety of the plates and shells. To be more precise, because the shell element formulation does not depend on a specific shell theory, the isoparametric approach can, in principle, be used in the analysis of any plate/shell structure [6, 7].

The scope of this report will be restricted to the three-dimensional behavior of general shell elements that follow the Reissner-Mindlin assumptions for the theory of plates. The Reissner-Mindlin assumptions are [26, 20]:

[1] Particles of the plate originally on a line that is normal to the undeformed middle surface remain on a straight line during deformation. However, this line need not

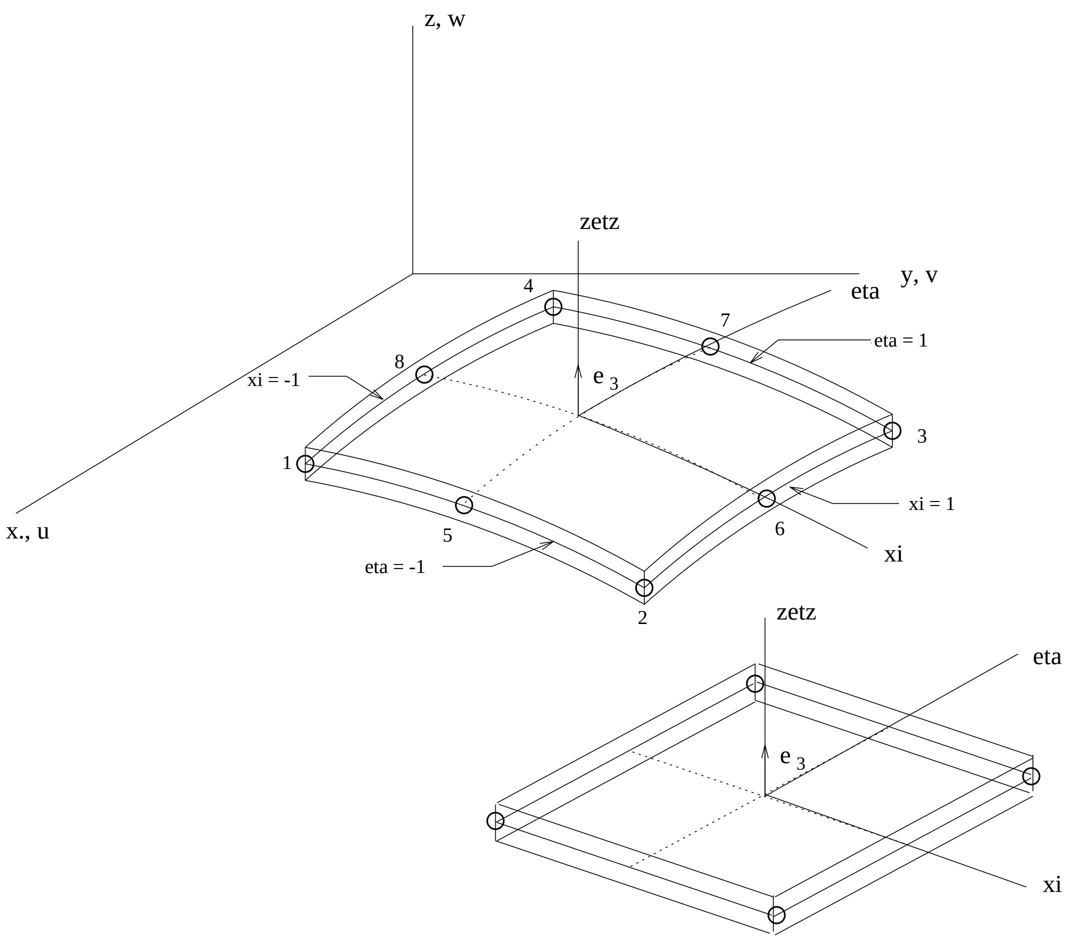

Figure 2.1: Geometry of 4 and 8 Node Shell Elements

remain normal to the deformed middle surface.

[2] Direct stresses in the z direction (normal) are small, and hence, direct strains in the z direction can be neglected.

These assumptions apply to thick plates and shells. The inconsistency of approximation in the second assumption can be compensated for by assuming plane stress conditions in each lamina. The omission of the constraint associated with the thin plate theory (i.e. normals remaining normal to the middle plate after deformation) permits the shell to experience shear deformation. This is an important feature in the thick shell situation [34]. To deal with the thin shell elements, the reduced integration technique is used.

2.2 Geometry of Shell Element

The geometry of general shell element is shown in Figure 2.1, where (x,y,z) are the global coordinates. The finite element formulation is general in the sense that finite elements can have four, eight, nine or even sixteen nodes. In each case, the formulation of the shell finite element is basically the same, except for details on the shape functions associated with the element nodes. The scope of this study will be restricted to four- and eight-node finite elements, as implemented in the formulations of Weaver and Johnston [32], Zienkiewicz and Taylor [34], and Bathe [6].

In the formulation of a typical shell element, the external surface of the shell may be flat or curved, sections across the thickness are generated by straight lines, and pairs of points on top and bottom surfaces are prescribed by the shape of element.

Let the ξ, η be two curvelinear coordinates in the middle plane of the shell and ζ a linear coordinate in the thickness direction. The ξ, η and ζ parameter vary between -1 and +1 on the respective surfaces of the element. The cartesian coordinates of any point in the shell element can be interpolated using curvelinear coordinates in the form:

$$\left\{ \begin{array}{c} x \\ y \\ z \end{array} \right\} = \sum N_i(\xi,\eta) \frac{1+\zeta}{2} \left\{ \begin{array}{c} x_i \\ y_i \\ z_i \end{array} \right\}_{top} + \sum N_i(\xi,\eta) \frac{1-\zeta}{2} \left\{ \begin{array}{c} x_i \\ y_i \\ z_i \end{array} \right\}_{bottom} \tag{2.1}$$

where $N_i(\xi, \eta)$ is a shape function at the nodes i. Equation 2.1 may be conveniently rewritten in the form specified by the vector connecting the upper and lower points and the mid-surface coordinates, namely:

$$\left\{ \begin{array}{c} x \\ y \\ z \end{array} \right\} = \sum N_i(\xi,\eta) \left\{ \begin{array}{c} x_i \\ y_i \\ z_i \end{array} \right\}_{mid} + \sum N_i(\xi,\eta)\, \zeta \frac{h_i}{2} \mathbf{v_{3i}} \tag{2.2}$$

with

$$\mathbf{v_{3i}} = \frac{1}{h_i} \left\{ \left\{ \begin{array}{c} x_i \\ y_i \\ z_i \end{array} \right\}_{top} - \left\{ \begin{array}{c} x_i \\ y_i \\ z_i \end{array} \right\}_{bottom} \right\} \tag{2.3}$$

where $\mathbf{v_{3i}}$ is the unit vector normal to the middle surface.

2.3 Displacement Field

In this formulation, displacements of the shell element at any point are described with respect to a global coordinate system, and stresses and strains, with respect to a local coordinate system. We use the notation $\mathbf{u}$ = {u,v,w} for displacements in the global coordinate system. Nodal displacements are given by three translational components (in global directions), and two rotations θ_{xi} and θ_{yi} about the local tangential axes x' and y,' respectively. Hence, overall nodal displacements are given by $\mathbf{q_i} = \{u_i, v_i, w_i, \theta_{xi}, \theta_{yi}\}$.

A consequence of the Reissner-Mindlin assumption is negligible strains in the direction normal to the mid-surface of the shell. Moreover, displacements throughout the element will be uniquely defined by the three translation components of the mid-surface nodal displacement and two rotations of the nodal vector $\mathbf{v_{3i}}$ about orthogonal directions normal to it. Let $\mathbf{v_{1i}}$ and $\mathbf{v_{2i}}$ be two orthogonal directions tangential to the plane of the flat shell element. the displacement field can be written in terms of nodal displacements is

$$\left\{ \begin{array}{c} u \\ v \\ w \end{array} \right\} = \sum N_i \left\{ \begin{array}{c} u_i \\ v_i \\ w_i \end{array} \right\} + \sum N_i \, \zeta \frac{h_i}{2} [-\mathbf{v_{2i}}, \mathbf{v_{1i}}] \left\{ \begin{array}{c} \theta_{xi} \\ \theta_{yi} \end{array} \right\} \tag{2.4}$$

Here, we simplify notation by dropping the suffix *mid*, and employ the method described by Belytschko et al. [9] to estimate the orientation of orthogonal base vectors $\mathbf{v_{1i}}, \mathbf{v_{2i}}, \mathbf{v_{3i}}$. $\mathbf{v_{3i}}$ is computed first – in the case of the eight node shell element, $\mathbf{v_3}$ is assumed to be perpendicular to the vectors $\mathbf{r_{75}} = \mathbf{r_7} - \mathbf{r_5}$ and $\mathbf{r_{86}} = \mathbf{r_8} - \mathbf{r_6}$. $\mathbf{r_i}$ is the vector from the origin of the global coordinate frame to node i of the shell element. It follows that vector $\mathbf{v_3}$ is given by

$$\mathbf{v_3} = \frac{\mathbf{r_{75}} \times \mathbf{r_{86}}}{||\mathbf{r_{75}} \times \mathbf{r_{86}}||} \tag{2.5}$$

For four node shell elements, the $\mathbf{r_{75}}$ and $\mathbf{r_{86}}$ vectors are estimated by

$$\mathbf{r_{75}} = \frac{1}{2}(\mathbf{r_4} + \mathbf{r_3}) - \frac{1}{2}(\mathbf{r_2} + \mathbf{r_1}) \tag{2.6}$$

$$\mathbf{r_{86}} = \frac{1}{2}(\mathbf{r_4} + \mathbf{r_1}) - \frac{1}{2}(\mathbf{r_2} + \mathbf{r_3}) \tag{2.7}$$

Once the $\mathbf{v_{3i}}$ is determined the vectors $\mathbf{v_{1i}}$ and $\mathbf{v_{2i}}$ are determined from the following procedures. The $\mathbf{v_{1i}}$ is computed by

$$\mathbf{v_{1i}} = \mathbf{e_y} \times \mathbf{v_{3i}} \tag{2.8}$$

then

$$\mathbf{v_{2i}} = \mathbf{v_{3i}} \times \mathbf{v_{2i}} \tag{2.9}$$

If $\mathbf{e_y}$ is parallel to $\mathbf{v_{3i}}$ then $\mathbf{e_y}$ is replaced by $\mathbf{e_z}$. where $\mathbf{e_x}, \mathbf{e_y}, \mathbf{e_z}$ are the unit vectors of global coordinates. To simplify the following derivation, we let $\mathbf{e_{1i}} = \{l_{1i}, m_{1i}, n_{1i}\}$ and so on. This formulation may be easily extended to curved shell elements.

2.4 Shape Functions

The displacement shape functions in Eq. (2.4) may be cast into the matrix form.

$$\mathbf{N_i} = \mathbf{N_{Ai}} + \zeta \mathbf{N_{Bi}} \tag{2.10}$$

where

$$\mathbf{N_{Ai}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} N_i \tag{2.11}$$

and

$$\mathbf{N_{Bi}} = \begin{bmatrix} 1 & 0 & 0 & -l_{2i} & l_{1i} \\ 0 & 1 & 0 & -m_{2i} & m_{1i} \\ 0 & 0 & 1 & -n_{2i} & n_{1i} \end{bmatrix} \frac{h_i}{2} N_i \tag{2.12}$$

The shape functions for the four-node shell element are

$$N_i = \frac{1}{4}(1+\xi_i\xi)(1+\eta_i\eta) \quad (i = 1,2,3,4) \tag{2.13}$$

where for nodes i = 1 to 4, the values of ξ_i are (-1, 1, 1, -1), and the values of η_i are (-1, -1, 1, 1). Similarly, for the eight node shell element the shape functions are

$$N_i = \frac{1}{4}(1+\xi_i\xi)(1+\eta_i\eta)(\xi_i\xi+\eta_i\eta-1) \quad (i = 1,2,3,4) \tag{2.14}$$

$$N_i = \frac{1}{2}(1-\xi^2)(1+\eta_i\eta) \quad (i = 5,7) \tag{2.15}$$

$$N_i = \frac{1}{2}(1+\xi_i\xi)(1-\eta^2) \quad (i = 6,8) \tag{2.16}$$

where for nodes i = 1 to 8, the values of ξ_i are (-1, 1, 1, -1, 0, 1, 0, -1), and the values of η_i are (-1, -1, 1, 1, -1, 0, 1, 0).

2.5 Stresses and Strains

The following non-zero stresses in the directions of primed axes(Figure 2.2) will be considered:

$$\sigma' = \{\sigma_{x'}, \sigma_{y'}, \tau_{x'y'}, \tau_{y'z'}, \tau_{z'x'}\} \tag{2.17}$$

and the corresponding strains are

$$\varepsilon' = \{\varepsilon_{x'}, \varepsilon_{y'}, \gamma_{x'y'}, \gamma_{y'z'}, \gamma_{z'x'}\}. \tag{2.18}$$

The relationship between strains and displacements is given as:

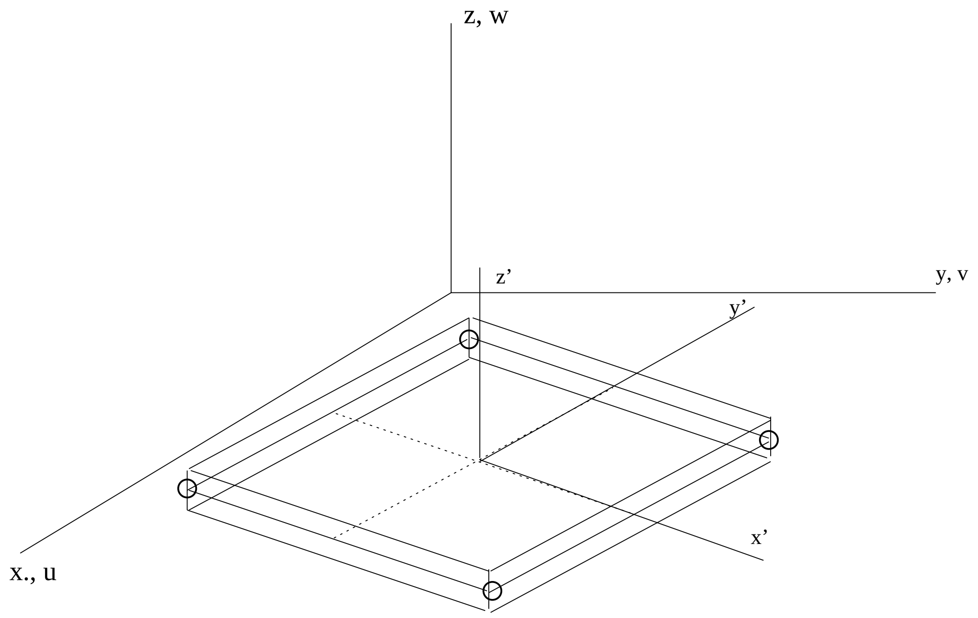

Figure 2.2: The primed axes of shell element

$$\varepsilon' = \begin{Bmatrix} \varepsilon_{x'} \\ \varepsilon_{y'} \\ \gamma_{x'y'} \\ \gamma_{y'z'} \\ \gamma_{z'x'} \end{Bmatrix} = \begin{Bmatrix} u_{,x'} \\ u_{,y'} \\ u_{,y'} + u_{,x'} \\ u_{,z'} + u_{,y'} \\ u_{,x'} + u_{,z'} \end{Bmatrix} \tag{2.19}$$

For linear materials, the relation between strains and stresses is expressed by the usual elastic matrix $\mathbf{E}$:

$$\mathbf{E} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 & 0 & 0 \\ 1 & \nu & 0 & 0 & 0 \\ 0 & 0 & \frac{1-\nu}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{1-\nu}{2k} & 0 \\ 0 & 0 & 0 & 0 & \frac{1-\nu}{2k} \end{bmatrix} \tag{2.20}$$

where k = 1.2 is the ratio of relevant strain energies (k accounts for transverse shearing stresses producing too little strain energy [30, 32]). For elastic-plastic materials, the relation between strains and stresses is expressed by

$$\sigma = \mathbf{E}(\varepsilon - \varepsilon^{\mathbf{P}}) \tag{2.21}$$

or elastic-plastic material matrix $\mathbf{E_{ep}}$ for the incremental stresses and strains.

$$\boldsymbol{\Delta}\sigma = \mathbf{E_{ep}}\boldsymbol{\Delta}\varepsilon \tag{2.22}$$

where $\varepsilon^{\mathbf{p}}$ is plastic strain, and ε and σ are stress and strain. $\mathbf{\Delta}\sigma$ and $\mathbf{\Delta}\varepsilon$ are incremental stress and strain.

2.6 Formulation of Finite Element Analysis

The mass and stiffness matrices for the flat shell finite element are derived via the principle of virtual work. Element strains are computed from the discretization and shape functions given in Eq. (2.4). In matrix form we have

$$\varepsilon = [\mathbf{B}]\mathbf{q^e} \tag{2.23}$$

The strain-displacement matrix $\mathbf{B}$ is given by $[B_1, B_2, \cdots, B_n]$ where B_i is [32, 6]:

$$\mathbf{B_i} = \begin{bmatrix} a_i & 0 & 0 & -d_i l_{2i} & d_i l_{1i} \\ 0 & b_i & 0 & -e_i m_{2i} & e_i m_{1i} \\ 0 & 0 & c_i & -g_i n_{2i} & g_i n_{1i} \\ b_i & a_i & 0 & -e_i l_{2i} - d_i m_{2i} & e_i l_{1i} + d_i m_{1i} \\ 0 & c_i & b_i & -g_i m_{2i} - e_i n_{2i} & g_i m_{1i} + e_i n_{1i} \\ c_i & 0 & a_i & -d_i n_{2i} - g_i l_{2i} & d_i n_{1i} + g_i l_{1i} \end{bmatrix} \tag{2.24}$$

and i = 1 to 8 for the eight node shell element, and i = 1 to 4 for the four node element. As detailed in Weaver and Johnston, and Bathe, the matrix coefficients are given by:

$$\begin{aligned} a_i &= J^*_{11} N_{i,\xi} + J^*_{12} N_{i,\eta} & d_i &= \frac{h_i}{2}(a_i \zeta + J^*_{13} N_i) \\ b_i &= J^*_{21} N_{i,\xi} + J^*_{22} N_{i,\eta} & e_i &= \frac{h_i}{2}(b_i \zeta + J^*_{23} N_i) \\ c_i &= J^*_{31} N_{i,\xi} + J^*_{32} N_{i,\eta} & g_i &= \frac{h_i}{2}(c_i \zeta + J^*_{33} N_i) \end{aligned} \tag{2.25}$$

where J^* is the inverse of Jacobian matrix. J and J^* are given as:

$$\mathbf{J} = \begin{bmatrix} x_{,\xi} & y_{,\xi} & z_{,\xi} \\ x_{,\eta} & y_{,\eta} & z_{,\eta} \\ x_{,\zeta} & y_{,\zeta} & z_{,\zeta} \end{bmatrix} \qquad \mathbf{J^{-1}} = \mathbf{J}^* = \begin{bmatrix} \xi_{,x} & \eta_{,x} & \zeta_{,x} \\ \xi_{,y} & \eta_{,y} & \zeta_{,y} \\ \xi_{,z} & \eta_{,z} & \zeta_{,z} \end{bmatrix} \tag{2.26}$$

The local strains in the vector ε' are related to the global strains in the vector ε by using a (6x6) strain transformation matrix $\mathbf{T}_\varepsilon$ as follows:

$$\varepsilon' = \mathbf{T}_\varepsilon \varepsilon \tag{2.27}$$

Here:

$$\mathbf{T}_\varepsilon = \begin{bmatrix} l_1^2 & m_1^2 & n_1^2 & l_1m_1 & m_1n_1 & n_1l_1 \\ l_2^2 & m_2^2 & n_2^2 & l_2m_2 & m_2n_2 & n_2l_2 \\ l_3^2 & m_3^2 & n_3^2 & l_3m_3 & m_3n_3 & n_3l_3 \\ 2l_1l_2 & 2m_1m_2 & 2n_1n_2 & l_1m_2+l_2m_1 & m_1n_2+m_2n_1 & n_1l_2+n_2l_1 \\ 2l_2l_3 & 2m_2m_3 & 2n_2n_3 & l_2m_3+l_3m_2 & m_2n_3+m_3n_2 & n_2l_3+n_3l_2 \\ 2l_3l_1 & 2m_3m_1 & 2n_3n_1 & l_3m_1+l_1m_3 & m_3n_1+m_1n_3 & n_3l_1+n_1l_3 \end{bmatrix} \quad (2.28)$$

We now observe that ε_z' will be zero – a consequence of the second Reissner-Mindlin assumption – and that the third row of matrix T_ε must be deleted. For a general point in the shell element, the direction cosines given by:

$$\mathbf{e_1} = \mathbf{J_1}/\|\mathbf{J_1}\|, \quad \mathbf{e_3} = (\mathbf{J_1}\times\mathbf{J_2})/\|\mathbf{J_1}\times\mathbf{J_2}\|, \quad \mathbf{e_2} = \mathbf{e_3}\times\mathbf{e_1} \quad (2.29)$$

where $\mathbf{J_i}$ is the i^{th} row of the Jacobian matrix. The corresponding transformation between **B** matrix is:

$$\mathbf{B}' = \mathbf{T}_\varepsilon\mathbf{B} \quad (2.30)$$

Let n be the number of nodes per element. **B'** matrix has the size of (5 x 5n), and the **B** matrix, size of (6 x 5n). The stiffness matrix is given by:

$$\mathbf{K_{ep}} = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1}\mathbf{B'^TE_{ep}B'}\|\mathbf{J}\|d\xi d\eta d\zeta \qquad \mathbf{K} = \int_{-1}^{1}\int_{-1}^{1}\int_{-s1}^{1}\mathbf{B'^TEB'}\|\mathbf{J}\|d\xi d\eta d\zeta. \quad (2.31)$$

The consistent mass matrix for shell elements is

$$\mathbf{M} = \rho\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1}\mathbf{N^TN}\|\mathbf{J}\|d\xi d\eta d\zeta. \quad (2.32)$$

For linear dynamic analysis, the final form of finite element formulation of equilibrium is expressed as:

$$\mathbf{M\ddot{q}} + \mathbf{Kq} = \mathbf{R} \quad (2.33)$$

where $\mathbf{q}$ is the displacement vector, $\mathbf{R}$ is the external force, $\mathbf{M}$ and $\mathbf{K}$ are the global mass and stiffness matrix. For nonlinear dynamic analysis, the solution is solved incrementally. Therefore the finite formulation of the equilibrium of is

$$\mathbf{M\Delta\ddot{q}} + \mathbf{K_{ep}\Delta q} = \mathbf{R} - \mathbf{F^{int,i}} - \mathbf{M\ddot{q}_e^{(i)}} \quad (2.34)$$

where $\mathbf{\Delta q} = \mathbf{q} - \mathbf{q^{(i)}}$, $\mathbf{q^{(i)}}$ is the displacement at time t = i, and $\mathbf{F^{int,i}}$ is the internal load due to the stresses at t = i. The interested reader is referred to Bathe [8] for details on solution procedures to nonlinear equations.

2.7 Material Property

This study will employ three material models: linear elastic, elastic-perfectly plastic, and elastic-plastic. However, for the purposes of brevity, discussion in this section will be restricted to nonlinear material models.

2.7.1 Elastic-perfectly plastic and elastic-plastic materials

Within the framework of theory of plasticity, the stress strain relation of a homogeneous, isotropic materials may be described by the incremental theory of plasticity. The following subsections summarize the criteria and relations used in the implementation of this theory.

Yield Criterion: The yield criterion of ductile materials can usually be characterized through von Mises criterion

$$f(\sigma) = \mathbf{S}^T\mathbf{S} - R^2 = 0. \tag{2.35}$$

Here $\mathbf{S}$ is the deviatoric part of shifted stress $\sigma - \sigma_\alpha$, σ is the total stress, and σ_α is the back-stress (i.e. location of the yield surface). R $= \sqrt{\frac{2}{3}}F_y$ is the radius of the yield surface in the π-plane. For elastic perfectly-plastic materials, R is a constant. For elastic-plastic materials, R increases as a function of plastic strains, which are load history dependent. In mathematical terms, if $f(\sigma) < 0$, then the material is in the elastic region. The material will yield when $f(\sigma) = 0$.

Loading/Unloading Criterion: The total strain incremental is the summation of the elastic and plastic parts of the strain incremental. The loading and unloading condition is determined by the plastic strain incremental and incremental stress, namely:

$$\begin{aligned} &\text{if } f = 0, \quad d\varepsilon^{pT} \cdot \mathbf{d}\sigma < 0 \quad \text{elastic} \\ &\text{if } f = 0, \quad d\varepsilon^{pT} \cdot d\sigma = 0 \quad \text{plastic} \end{aligned} \tag{2.36}$$

Consistency Condition : Because the yield surface and load vector move together in the post-elastic range, the consistency condition for a plastic processes is

$$f = 0 \tag{2.37}$$
$$d\,f = 0 \tag{2.38}$$

Flow Rule : The associated flow rule states that

$$d\varepsilon^p = \Lambda \frac{\frac{\partial f}{\partial \mathbf{S}}}{\|\frac{\partial f}{\partial \mathbf{S}}\|} = \Lambda \mathbf{N} \tag{2.39}$$

where Λ is a constant of proportionality, and $\mathbf{N}$ the direction vector of plastic strain normal to the yield surface.

Stress-Strain Relation : The stress and strain relation can be written as

$$\mathbf{d}\sigma - \mathbf{d}\sigma_\alpha = \mathbf{E}(\mathbf{d}\varepsilon - \mathbf{d}\varepsilon^{\mathbf{P}}) \tag{2.40}$$

$$\sigma - \sigma_\alpha = \mathbf{E}(\varepsilon - \varepsilon^{\mathbf{P}}) \tag{2.41}$$

or in terms of deviatoric stress

$$\mathbf{dS} = 2G(\mathbf{de} - \mathbf{d}\varepsilon^{\mathbf{P}}) \tag{2.42}$$

$$\mathbf{S} = 2G(\mathbf{e} - \varepsilon^{\mathbf{P}}) \tag{2.43}$$

where G is shear modulus and $\mathbf{e}$ is the deviatoric part of the total strain.

Strain Hardening Rule : The strain hardening rule for the strain hardening materials is

$$R = \beta\sqrt{\frac{2}{3}} \int H'(\overline{\varepsilon}^p) d\overline{\varepsilon}^p \tag{2.44}$$

$$dR = \beta\sqrt{\frac{2}{3}}\, H'(\overline{\varepsilon}^p) d\overline{\varepsilon}^p \tag{2.45}$$

$$H' = \frac{d\overline{\sigma}}{d\overline{\varepsilon}^p} \tag{2.46}$$

$$d\sigma_\alpha = \frac{2}{3}(1-\beta)H' d\varepsilon^{\mathbf{P}} \tag{2.47}$$

where $\beta = 1$ for isotropic strain hardening and $\beta = 0$ for kinematic hardening. H' is a plastic modulus obtained from the quotient uniaxial stress/plastic strain. The parameters $\overline{\sigma}$ and $d\overline{\varepsilon^p}$ are effective stress and effective plastic strain increment, and are defined as:

$$\begin{gathered} \overline{\sigma} = \sqrt{\tfrac{3}{2}\textstyle\sum_{i,j} S_{ij}S_{ij}} \\ d\overline{\varepsilon^p} = \sqrt{\tfrac{2}{3}\textstyle\sum_{i,j} d\varepsilon^p_{ij} d\varepsilon^p_{ij}} \end{gathered} \tag{2.48}$$

2.7.2 Material Load Curves

The load curve of materials is the one dimensional stress strain curve. When loadings are multi-dimensional, the material load curve represents the relation between the effective stress and effective strain.

Two mathematical models are commonly employed for the simulation of behavior in engineering materials. The first is the so-called "Ramberg-Osgood" model, and it is written:

$$\frac{\varepsilon}{\varepsilon_s} = \frac{\sigma}{F_y} + \alpha \left(\frac{\sigma}{F_y}\right)^n \tag{2.49}$$

Here n is the strain-hardening exponent and $\varepsilon_s = F_y/E$. The corresponding plastic modulus H' is:

$$H' = \frac{E}{\alpha n} \left(\frac{\sigma}{F_y}\right)^{1-n}. \tag{2.50}$$

The second model is the so-called **bi-linear** material model, and it assumes that strain hardening will be linear. The stress strain relation for bi-linear model is:

$$\sigma = \begin{cases} E\varepsilon & \|\varepsilon\| \leq \varepsilon_s \\ F_y + E_t(\varepsilon - \varepsilon_s) & \varepsilon > \varepsilon_s \\ -F_y - E_t(\varepsilon + \varepsilon_s) & \varepsilon < -\varepsilon_s \end{cases} \tag{2.51}$$

where E_t is the slope of the curve in plastic range. The corresponding plastic modulus H' is

$$H' = \frac{EE_t}{E - E_t} \tag{2.52}$$

2.8 Stress Update and Integration Algorithm

The basic principle of stress update is to use the previous state (i) to estimate next state (i+1). For static analysis, time is interpreted as a loading parameter. Over the increment Δt a load step is applied to the structure. The constitutive routine integrates the plasticity rate equations over the time increment to determine the updated stresses and plastic state variables.

2.8.1 Radial Return Algorithm

The radial return algorithm uses the consistency condition and hardening rule to compute the effective plastic strain increment, the incremental plastic strain vector Λ, and the stress increment [33]. Several variants of this algorithm are now in use, and the following algorithm is adapted from the text of Dodds [13], and the class notes of Lee [19].

First, the trial stress σ^* is computed by using the elastic stress and strain relations, namely:

$$\sigma^* = \sigma^{(\mathbf{i})} + \mathbf{E}\boldsymbol{\Delta}\varepsilon \tag{2.53}$$

Then, calculate the shift stress, the corresponding deviatoric parts $\mathbf{S}^*$, and test if the material has yielded

$$A^2 \quad = \quad \mathbf{S}^{*T}\mathbf{S}^*. \tag{2.54}$$

There are two cases for the stress update:

Case 1 : The material is in an elastic state if $A < R$. In such cases, $\sigma^{(i+1)} = \sigma^*$.

Case 2 : The material is in a plastic state if $A > R$. Updates are needed for the stress and plastic strain, and for kinematic hardening, updates are also required in the back-stress. The update equations are as follows:

$$\sigma^{\mathbf{i+1}} = \sigma^{(\mathbf{i})} - \mathbf{2G}\boldsymbol{\Delta}\varepsilon^{\mathbf{p}} + \boldsymbol{\Delta}\sigma_\alpha \tag{2.55}$$

$$\sigma_\alpha{}^{\mathbf{i+1}} = \sigma_\alpha{}^{(\mathbf{i})} - \frac{\mathbf{2}}{\mathbf{3}}(\mathbf{1} - \beta)\mathbf{H}'\boldsymbol{\Delta}\varepsilon^{\mathbf{p}} \tag{2.56}$$

$$\varepsilon^{\mathbf{pi+1}} = \varepsilon^{(\mathbf{i})} + \boldsymbol{\Delta}\varepsilon^{\mathbf{p}} \tag{2.57}$$

In the following derivation we use the the trial stress σ^* to estimate the plastic strain incremental vector. A prerequisite for the plastic process is satisfaction of the consistency condition. The normal vector and the plastic strain increment are expressed with trial stress as

$$\mathbf{N} \quad = \quad \frac{\frac{\partial f}{\partial S}}{\|\frac{\partial f}{\partial \mathbf{S}}\|} = \frac{S^*}{A} \tag{2.58}$$

$$\boldsymbol{\Delta}\varepsilon^{\mathbf{p}} \quad = \quad \Lambda\mathbf{N} \tag{2.59}$$

The normal direction of plastic strain increment $\mathbf{N}$ is given by the following equation:

Based on the consistency condition, the at step i+1, $f(\sigma^{(i+1)}, \mathbf{R}^{(i+1)}) = 0$. Therefore,

$$\mathbf{S}^{(i+1)} = 2G(\mathbf{e} - \varepsilon^p) = \mathbf{S}^* - 2G\varepsilon^{\mathbf{p}} = \left(1 - 2G\frac{\Lambda}{A}\right)\mathbf{S}^* \tag{2.60}$$

$$\mathbf{S}^{(i+1)^T}\mathbf{S}^{(\mathbf{i+1})} = \left(1 - 2G\frac{\Lambda}{A}\right)^2 \mathbf{S}^{*T}\mathbf{S}^* = \left(\mathbf{R}^{(i+1)}\right)^2 \tag{2.61}$$

or in terms of effective stress, one has

$$\overline{\sigma}^{(i+1)} = \left(1 - 2G\frac{\Lambda}{A}\right)\overline{\sigma}^* = \sqrt{\frac{3}{2}}R^{(i+1)} \tag{2.62}$$

To eliminate the Λ, the effective plastic strain increment is used.

$$(\Delta\overline{\varepsilon^p})^2 = \left(\frac{2}{3}\right)^2 \Lambda^2 \left(\frac{\overline{\sigma}^*}{A}\right)^2 \tag{2.63}$$

$$\Lambda = \frac{3A\Delta\overline{\varepsilon^p}}{2\overline{\sigma}^*} = \sqrt{\frac{3}{2}}\Delta\overline{\varepsilon^p} \tag{2.64}$$

Using the result for Λ in equation 2.62 yields

$$\sqrt{\frac{3}{2}}R^{(i+1)} = \overline{\sigma}^* - 3G\Delta\overline{\varepsilon^p} \tag{2.65}$$

where $R^{(i+1)}$ is calculated from the strain hardening rule, for sufficient small incremental steps, $H'^{(i)}$ can be considered as a constant during the step, therefore,

$$R^{(i+1)} = R^{(i)} + \sqrt{\frac{2}{3}}H'^{(i)}\Delta\overline{\varepsilon^p} \tag{2.66}$$

Solving for effective plastic strain increment, we have

$$\Delta\overline{\varepsilon^p} = \frac{\overline{\sigma}^* - \sqrt{\frac{3}{2}}R^{(i)}}{H'^{(i)} + 3G} = \sqrt{\frac{3}{2}}\frac{A - R^{(i)}}{H'^{(i)} + 3G} \tag{2.67}$$

Once the effective plastic strain increment is calculated, the plastic strain increment vector, the back-stress and the stress are all calculated and updated. Then the H' can be updated with the trial effective stress $\overline{\sigma}^*$ using equation 2.50.

The algorithm is computationally efficient and accurate for single step and sub-incremental schemes, and unconditionally stable. Moreover, it does not requires computation of the contact stress or ad hoc scaling of the updated stress vector. Mixed isotropic-kinematic hardening is easily included [13].

Direct Calculation of Elastic-Plastic Stiffness : The elastic-plastic stiffness may be computed directly by evaluating equation 2.31. The heart of the computation is the elastic-plastic material matrix $\mathbf{E_{ep}}$. Consider following product, we have

$$
\begin{aligned}
\mathbf{N}^T\mathbf{dS} &= \frac{\mathbf{S}}{\sqrt{\mathbf{S^TS}}}\,\mathbf{dS} = \sqrt{\frac{2}{3}}d\bar{\sigma} \\
&= \sqrt{\frac{2}{3}}H' d\bar{\varepsilon^p} = \frac{2}{3}H'\Lambda
\end{aligned} \tag{2.68}
$$

On the other hand, the above product can also be written as:

$$
\begin{aligned}
\mathbf{N}^T\mathbf{dS} &= \mathbf{N}^T\ 2G(\mathbf{de}-\mathbf{de}^p) = \mathbf{N}^T\ 2G(\mathbf{d}\varepsilon-\mathbf{d}\varepsilon^p) \\
&= 2G\mathbf{N}^T\mathbf{d}\varepsilon - 2G\Lambda
\end{aligned} \tag{2.69}
$$

Equating these two equations, one can express the Λ with following expression:

$$
\Lambda = \frac{2G\mathbf{N}^T\mathbf{d}\varepsilon}{(2G+\frac{2}{3}H')} \tag{2.70}
$$

Substituting this Λ expression into incremental stress strain relation:

$$
\begin{aligned}
\mathbf{d}\sigma &= \mathbf{E}(\mathbf{d}\varepsilon - \mathbf{d}\varepsilon^{\mathbf{p}}) \\
&= \left(\mathbf{E} - \frac{2G}{(2G+\frac{2}{3}H')}\mathbf{ENN}^T\right)\mathbf{d}\varepsilon
\end{aligned} \tag{2.71}
$$

One can obtain the elastic-plastic material matrix as :

$$
\mathbf{E_{ep}} = \mathbf{E} - \frac{2G}{(2G+\frac{2}{3}H')}\mathbf{ENN}^T \tag{2.72}
$$

2.8.2 Sub-Incrementation Scheme

A crucial factor affecting accuracy of radial return algorithm is selection of an appropriate step length for return to the yield surface. Crisfield [12] discusses several algorithms for reducing errors with the method of sub-incrementation. With this technique, the incremental strain $\Delta\varepsilon$ is divided into m sub-steps each of $(\Delta\varepsilon)/m$. Nyssen [25] estimates that the number of required substeps to give a tolerance of β_1 in yield condition f = 0 is

$$m = \frac{2\overline{\Delta\sigma}}{\beta_1 F_y}. \tag{2.73}$$

The recommended value for β_1 is 0.05. Crisfield [12] argues that

$$m \propto \left(\frac{\overline{\Delta\sigma}}{F_y}\right)^{1/2} \tag{2.74}$$

is an appropriate number of substeps. In this study, we have used a modified version of Nyssen's equation, namely:

$$m = \frac{2\overline{\Delta\sigma}}{\beta_1 F_y} + 1 \tag{2.75}$$

with the term 1 being added to make sure m has at least one sub-step.

2.8.3 Reduced Integration

For quadratic shape functions, three-point Gaussian Integration is needed for accurate results. Unfortunately, direct use of three-point integration in finite element computations leads to overly-stiff stiffness matrices. The problem is mitigated by so-called reduced integration techniques to relax the stiffness matrix. In this study, we employ by default, 2x2 integration in the surface direction, as shown in Figure 2.3, and two points of integration through the thickness direction of the shell element. For elastic analyses and some elastic-plastic analyses, the default points of integration are good enough to produce adequate finite element results. Belytschko et al. [9] point out, however, that more integration points may be needed in some nonlinear cases. Indeed, Belytschko and co-workers [9] report on nonlinear experiments where three and five points of integration are used through the thickness of a simply supported plate under uniform pressure.

In an effort to overcome the abovementioned variability in performance from problem-to-problem, we have designed the ALADDIN language so that the the engineer can adjust the number of integration points used to compute the stiffness of shell elements. For elastic analyses, there is no need to specify the number of integration points in the input file – the default number is used. As we will see in the following chapter, the

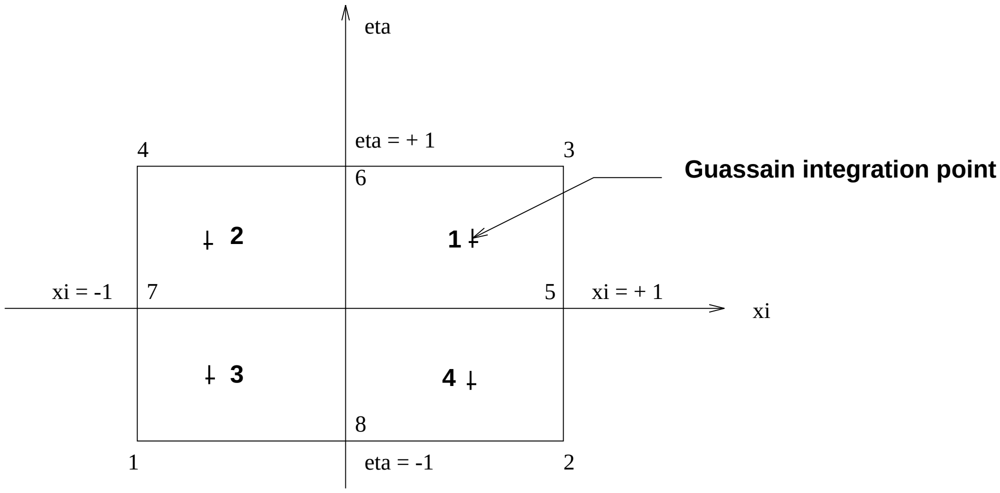

Figure 2.3: Gaussian Integration points in Surface of Element

nonlinear analysis of the unaxial loaded plate is calculated with two integration points through the thickness of the plate. This is because that there are no dramatic deformation through the thickness for plane stress problem. In the cantilever problem, at least three integration points are needed to capture the a parabolic distribution of plasticity through the thickness.

2.9 Solving Nonlinear Equations with BFGS algorithm

Let $x = (x_1, x_2, \cdots, x_n)^T$ be a coordinate point in n-dimensional space, and $f = (f_1, f_2, \cdots, f_n)^T$ be a n-dimensional (nonlinear) vector. The roots of nonlinear equations f are given by solutions to

$$f(x) = 0. \tag{2.76}$$

The methods of "Newton-Raphson" and "Secant Approximation" are among the most popular for computing the roots of nonlinear equations. Let vector x_o be an initial guess at the solution to equation (2.76). The method of Newton-Raphson corresponds to a sequence of first order Taylor series expansion about $x_{(k)}$, namely

$$f(x_{(k+1)}) = f(x_{(k)}) + \nabla f(x_{(k)}) \cdot (x_{(k+1)} - x_{(k)}) \approx 0$$

where k = 0, 1, 2, $\cdots$, n, and $(\nabla f(x))_{ij} = (\partial f_i)/(\partial x_j)$. In matrix form, the gradient approximation is

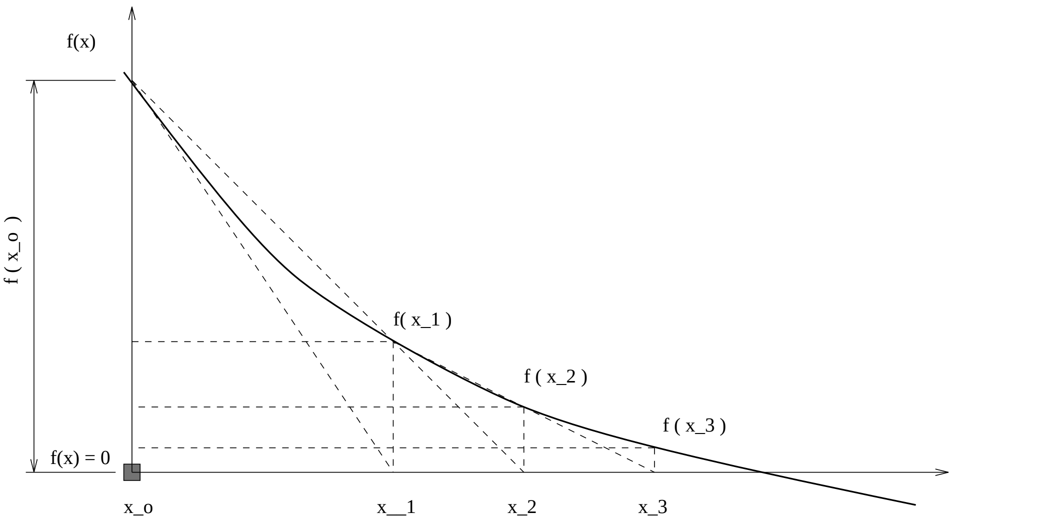

Figure 2.4: Secant Approximation of Quasi-Newton Method

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_2(x)}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial f_n(x)}{\partial x_1} & \frac{\partial f_n(x)}{\partial x_2} & \cdots & \frac{\partial f_n(x)}{\partial x_n} \end{bmatrix}. \tag{2.77}$$

If $\nabla f(x_{(k)})$ has an inverse, then the (full) Newton-Raphson update is:

$$x_{(k+1)} = x_{(k)} - \left[f(x_{(k)})\right]^{-1} \cdot \nabla f(x_{(k)}). \tag{2.78}$$

The procedure is repeated until convergence criteria are satisfied. While the method of full Newton-Raphson can be efficient in some specific nonlinear analyses, Bathe [5, 8] reports that in general, full Newton-Raphson is not a competitive computational method for computing the roots of a wide-range of nonlinear equations. A major limitation of full Newton-Raphson is the need for updating and factorizing the coefficient matrix -$\nabla f(x_{(k)})$ at each iteration. One strategy for avoiding these computationally expensive steps is to replace -$\nabla f(x_{(k)})$ in equation (2.78) with -$\nabla f(x_{(0)})$, thereby eliminating the need to recalculate and factorize -$\nabla f(x_{(k)})$ at each iteration. From a mathematical point of view, this simplification corresponds to a linearization of the gradient $f(x_o)$. For problems with significant nonlinearities – in particular when the system stiffens during the response – this linearization can lead to a very slow convergence in the iteration. Even worse, the iteration may diverge.

The class of quasi-Newton methods are a second alternative to full Newton Raphson. Quasi-Newton methods update the coefficient matrix, or it's inverse, to provide a secant approximation to the matrix from iteration `(k)` to `(k+1)`. Figure 2.4 shows, for example, a sequence of secant approximations one might apply during the computation of roots in a one-dimension nonlinear equation. If a displacement increment is defined

$$\delta_{(k+1)} = x_{(k+1)} - x_{(k)} \tag{2.79}$$

an increment in the residuals defined as

$$\gamma_{(k+1)} = f(x_{(k)}) - f(x_{(k+1)}), \tag{2.80}$$

then the updated secant stiffness matrix, $K_{(k+1)}$, will satisfy the quasi-Newton equation

$$K_{(k+1)}\delta_{(k+1)} = \gamma_{(k+1)}$$

.

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) Algorithm : Among Quasi-Newton methods, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method appears to be the most effective. Within iteration `(k+1)`, the BFGS method employs the following procedure to evaluate $x_{(k+1)}$ and $K_{(k+1)}$ [8]:

Step 1 : Initialization. Provide an initial value of vector $x = x_0$, and calculate the corresponding K_o matrix at $x = x_o$.

Step 2 : Evaluated the x vector increment

$$\Delta x = -\Big[f(x_{(k)})\Big]^{-1} \cdot \nabla f(x_{(k)}). \tag{2.81}$$

This vector increment defines a *direction* for the actual increment.

Step 3 : Perform a line-search along direction Δx to satisfy "equilibrium." Details of the line-search are as follows. First, set $\beta = 1.0$ and let the x vector

$$x_{(k+1)} = x_{(k)} + \beta\Delta x \tag{2.82}$$

where β is a scalar multipler. The product $\beta \mid \Delta x \mid$ represents the distance between points $x_{(k)}$ and $x_{(k+1)}$. The value of β is changed until the component of the residual in direction Δx, as defined by the inner product of $\Delta x^T f(x_{(k+1)})$, is approximately zero. The latter condition is satisfied when

$$\Delta x^T \cdot f(x_{(k+1)}) \leq \ STOL \cdot \Delta x^T \cdot f(x_{(k)}). \tag{2.83}$$

STOL is a convergence tolerance. The distance of the line search is automatically halved after each failure of equation (2.83) by halving β; $x_{(i)}$ is then recalculated with the new β. If after five or ten line search attempts, equation (2.83) still isn't satisfied, then the direction given by the Δx is probablely wrong. A new direction calculation is needed. This procedure is summarized in four-substeps:

Step 3.1 : Use equation (2.81) to update `x`.

Step 3.2 : `Begin while loop` : While equation (2.83) is not satisfied, work through Substeps 3.3 to 3.5.

Step 3.3 : If the line search number is less than or equal to `MaxLineSearchCount`, then set $\beta = \beta/2.0$, and use equation (2.81) to update `x`.

Step 3.4 : If the line search number is greater than `MaxLineSearchCount`, then break the while loop. Recalculate matrix `K` at $x = x_{(k+1)}$. Go to Step 2.

Step 3.5 : `End while loop` : Go to Step 4 for BFGS update.

Step 4 : Use equations (2.79) and (2.80) to calculate $\delta_{(k+1)}$ and $\gamma_{(k+1)}$.

Step 5 : Use the BFGS update [8] to revise the inverse of coefficient matrix `K`. The update of `K` is given by

$$K^{-1}{}_{(k+1)} = A_{(k+1)}{}^T K^{-1}{}_{(k)} A_{(k+1)} \tag{2.84}$$

where the matrix $A_{(k+1)}$ is a $(n \times n)$ matrix

$$A_{(k+1)} = I + v^{(k+1)} \cdot w^{(k+1)^T}. \tag{2.85}$$

Vectors $v^{(k+1)}$ and $w^{(k+1)}$ are given by

$$v^{(k+1)} = -\left[\frac{\delta_{(k+1)}{}^T \gamma_{(k+1)}}{\delta_{(k+1)}{}^T \beta f(x_{(k)})} \right] \cdot \beta \cdot f(x_{(k)}) - \gamma_{(k+1)} \tag{2.86}$$

and

$$w^{(k+1)} = \left[\frac{\delta_{(k+1)}}{\delta_{(k+1)}{}^T \gamma_{(k+1)}} \right]. \tag{2.87}$$

Step 6 : To avoid numerically dangerous updates, the conditional number

$$c_{(k+1)} = \left[\frac{\delta_{(k+1)}{}^T \gamma_{(k+1)}}{\delta_{(k+1)}{}^T \beta f(x_{(k)})} \right]^{1/2} \tag{2.88}$$

of the updating matrix $A_{(k+1)}$ must be compared to some predetermined tolerance. A large condition number implies that the updated inverse matrix will be nearly singular. Numerical updates are not performed if the condition number exceeds this tolerance – in this project, we follow the recommendation of Bathe [8], and set the tolerance at 10^5.

Step 7 : Check convergence of *force* and *energy* equilibriums. The force convergence criterion requires that the norm of the out-of-balance residual or *force* to be within a pre-set tolerance ϵ_F of the first residual.

$$\|f(x_{(k+1)})\|_2 \leq \epsilon_F \cdot \|f(x_{(o)})\| \tag{2.89}$$

A force criterion is not sufficient to ensure convergence. Consider the case of function f(x) with small or closing to zero gradients, the out-of-balance residual or *force* may be very small while the variable x or displacement may be grossly in error. Therefore, the "energy equilibrium" condition is necessary to provide the indication that both $\delta x^{(k+1)}$, and residual are approaching zeros. It requires computation of the work done by the force residual moving through displacement increment Δx.

$$\Delta x_{(k+1)} \cdot f(x_{(k)}) \leq \epsilon_E \cdot \Delta x_{(o)} \cdot f(x_{(0)}) \tag{2.90}$$

where the ϵ_E is a preset energy tolerance.

In the following examples we will use $\epsilon_F = 10^{-4}$ and $\epsilon_E = 10^{-5}$.

Chapter 3

Numerical Examples

Several examples have been used to demonstrate the performance and accuracy of the general shell element. In our preliminary numerical examples, we found that the bilinear four node shell element gives poor behavior. Therefore that only the eight-node element is discussed further. Four examples are discussed here including two linear static problem, one linear dynamic analysis and one nonlinear problem.

3.1 Linear Static Analysis

3.1.1 Simply supported square plate subjected to a concentrated load

Consider a simply supported square plate subjected a concentrated load at the center of the plate as shown in Figure 3.1. The analytical solution of the transverse displacement at the center, w_c, is given by Timoshenko [30] as:

$$w_c = 0.01160\frac{PL^2}{D} \tag{3.1}$$

$$D = \frac{Eh^3}{12(1-\nu^2)} \tag{3.2}$$

where P is the load, L and h are the length and thickness of the plate. For P = 7500 lbf, h = 0.5 in, the $w_c = 0.304$ inch, and P = 75 lbf, h = 0.05 in, $w_c = 3.04$ inch, according to the thin shell/plate theory [30]. For finite element meshing, only a quarter of the plate is considered due to the symmetry. The number of elements on the quarter of plate are N = 2x2, 4x4. The numerical results is given in the Table 3.1 and 3.2.

For elements of finite size it is found that pure bending deformation modes are always accompanied by some shear stresses, which in fact, do not exist in the conventional thin plate or shell bending theory. Thus large elements deforming mainly under bending

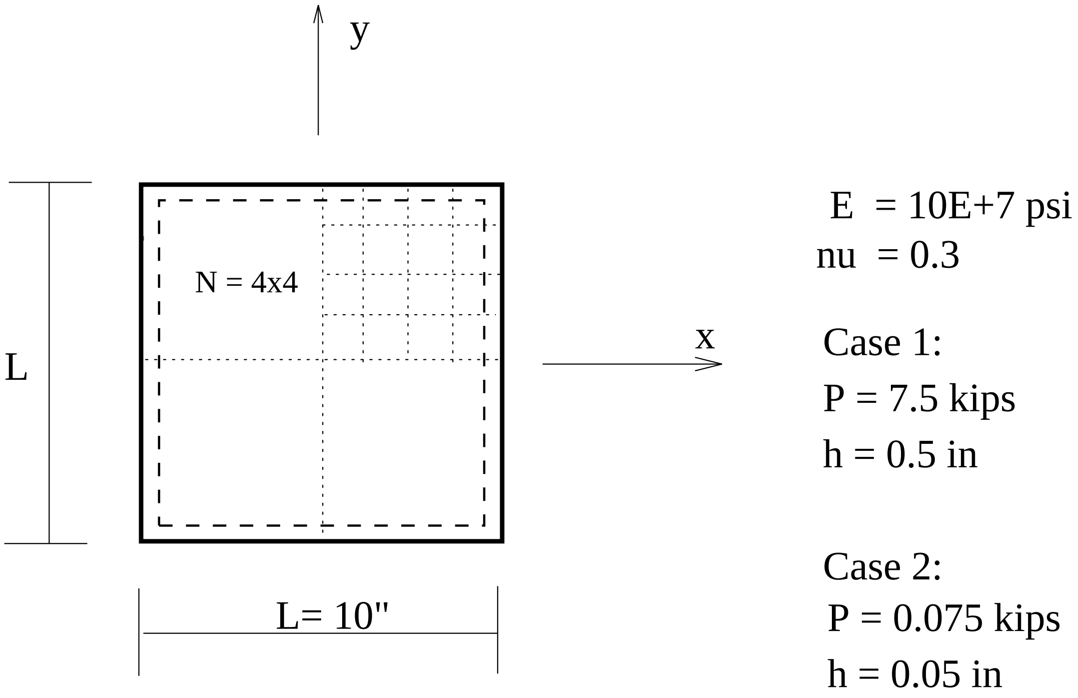

Figure 3.1: Simple-supported square plate subjected to a concentrated load

action (as would be the case of the shell element degenerated to a flat plate) tend to be appreciably too stiff. In such cases, certain limits of the ratio of side of element to its thickness have to be imposed. However, it will be found that such restrictions are relaxed by the simple expedient of reducing the integration order [34]. In this study a 2x2 integration points are used in the surface instead of 3x3 points. Due to such relaxation, the displacements calculated from finite element method tend to be larger than predicted by the analytical solution of thin plate/shell theory as reported in [34].

The results in above tables also show the same trends. For the ratio of h/L = 0.05, the convergence is quickly achieved. Yet with increase of the elements, the results move away the solution given by thin plate/shell theory. For the ratio of h/L = 0.005, the finite element results quickly converge to the exact solutions of the thin plate/shell theory.

3.1.2 Cantilever beam Subject to a Concentrated Load

Consider a cantilever beam with rectangular cross section subjected a concentrated load at the tip of cantilever as shown in Figure 3.2. The analytical solution of the tip displacement, Δ, is given by beam theory [31]as:

$$\Delta = \frac{PL^3}{3EI} \tag{3.3}$$

Load p = 7.5 (kips); Plate thickness h = 0.5 (in)			
Mesh Size	N = 2x2	N = 4x4	N = 8x8
w_c (FEM)	3.15739e-01 (in)	3.18724e-01 (in)	3.20491e-01 (in)
w_c (Theoretical)	3.04e-01 (in)	3.04e-01 (in)	3.04e-01 (in)
Error(%)	3.86	4.84	5.42

Table 3.1: Results for a simply supported square plate subjected a concentrated load: P = 7.5 kips, h = 0.5 in

Load p = 0.075 (kips); Plate thickness h = 0.05 (in)			
Meshes	N = 2x2	N = 4x4	N = 8x8
w_c (FEM)	2.86100 (in)	3.02731 (in)	3.04025 (in)
w_c (Analytical)	3.04 (in)	3.04 (in)	3.04 (in)
Error(%)	5.92	0.42	8.22e-3

Table 3.2: Results for a simply supported square plate subjected a concentrated load: P = 0.075 kips, h = 0.05 in

$$\theta = \frac{PL^2}{2EI} \qquad (3.4)$$

where P is the load, L = 96 inch is the length and I is the moment of inertia about the y axis, b = 12 in is the width and h = 1 in is the thickness of the plate. The cross section is rectangular. For P = 40 lbf, the Δ = 0.3932 inch, θ = 6.14400e-03 rad. Two set of meshings are used. N = 1x4 and N = 1x 8. The numerical results is given in the Table 3.3. These two examples indicate that the 8-node general shell elements give good performance for the flat plate/shells.

3.2 Nonlinear Static Analysis

The nonlinear feature of the element is demonstrated by the following examples. Unlike linear analysis, the nonlinear analysis needs many times trail-and-error processes to adequately model a problem. This is not only due to the complexity of the problem, but also due to the numerical algorithms used for the problem. Since most of the algorithms used for the finite element analysis, are either Newton or Quasi-Newton based algorithms, which are sensitive to the initial guessing value, the incremental steps chosen in the analysis can be crucial for the convergence of the iteration. As the Bathe and Cimento

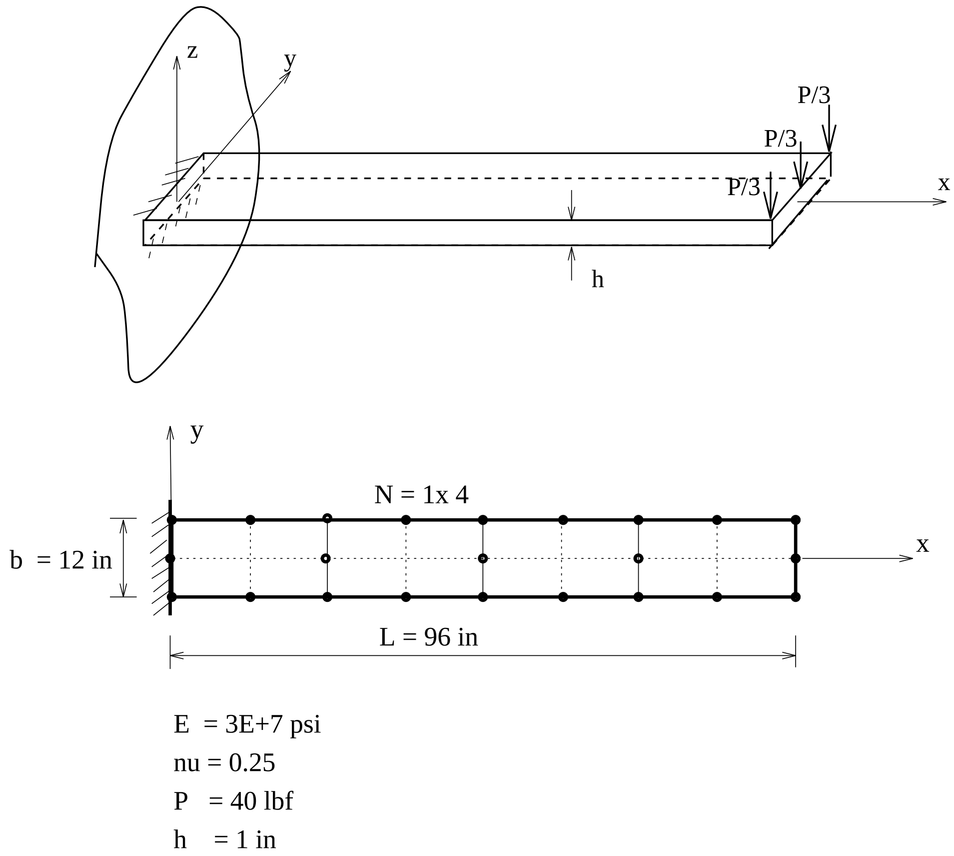

Figure 3.2: A cantilever beam subjected to a concentrated load

putted " the determination of the most effective approach to a general nonlinear analysis is at present a largely a matter of experience on the part of the analyst" [8].

In this study, only the force and energy criteria (Eqs. 2.89 and 2.90) are used for the convergence check. We use following parameters for line-search, force and energy criteria: STOL = 0.5, $\varepsilon_f = 1E-1$, and $\varepsilon_e = 1E-3$, the same parameters were used in Bathe and Cimento paper [8].

3.2.1 Rectangular Plate subjected to Uniaxial Loading

As shown in the Figure 3.3, a rectangular plate is subjected to an uniaxial. The initial load is P_0 = 1E+4*L*t. The total load is P = Factor*P_0. The integration points in the surface are 2x2 points and in the thickness are points. Two cases are examined for the isotropic strain hardening model. The Young's modulus is E = 1E+7 and the yield stress is 1E+4 psi, Possion ratio is 0.3.

Case 1: The first case is for the bi-linear stress strain curve. The tangent modulu E_t = 0.5 E. The load factors for the first case is Factor = [1.5, 2, 1.255, 0.01, -1.255, -2.5, -3.0, -3.5, -4.0, -2.0, -1.0,0]. The calculated the load vs. displacement in load direction is

Meshes	N = 1x4	N = 1x8
Δ (FEM)	3.87927e-01 (in)	3.90385e-01 (in)
Δ (Analytical)	3.93216e-01 (in)	3.93216e-01 (in)
Error(%)	1.34	0.72
θ (FEM)	6.07834e-03 (rad)	6.10308e-03 (rad)
θ (Analytical)	6.14400e-03 (rad)	6.14400e-03 (rad)
Error(%)	1.07	0.67

Table 3.3: Results for cantilever beam subjected a concentrated load

shown in Figure 3.4.

Case 2: The second case is for the Ramberg-Osgood stress-strain model. The strain hardening exponent, n, and the coefficient, α, of the Ramberg-Osgood model are : 4 and 3/7, respectively. The load factor is given as: Factor = [1.0, 1.5, 2, 1.255, 0.01, -1.255]. Between every load factor and next load factors, the load steps are subdivided into eleven sub-steps. And the load load steps are stopped at No. 63. The load steps are shown at Figure 3.5, and the results is shown by Figure 3.6.

3.2.2 Cantilever Beam subjected to Tip Load

A cantilever beam with rectangular cross section subjected a concentrated load at the tip of cantilever. The length of the cantilever L is 10 in, width is 1 in and thickness is 1 in. The initial load in z direction is P = -300 lbf. Again two cases are stuied. Four through-thickness integration points are used in both cases.

Case 1: In the first case, the material model is a Bi-Linear stress-strain curve with isotropic hardening. The young's modulus is E = 2.9E+7 psi, Possion ratio is 0.2, E_t = 0.1 E and the yield stress is 36 ksi. The load steps are illustrated in Figure 3.7, and the results for the hysteresis loop is given in Figure 3.8.

Case 2: In the second case, the material model is a Ramberg-Osgood stress-strain curve with isotropic hardening. The young's modulus is E = 2.9E+7 psi, Possion ratio is 0.2, the Ramberg-Osgood coefficient is $\alpha = 3/7$, the strain hardening exponent is n = 6. The yield stress is 36 ksi.

The load steps are illustrated in Figure 3.9, and the results for the hysteresis loop is given in Figure 3.10.

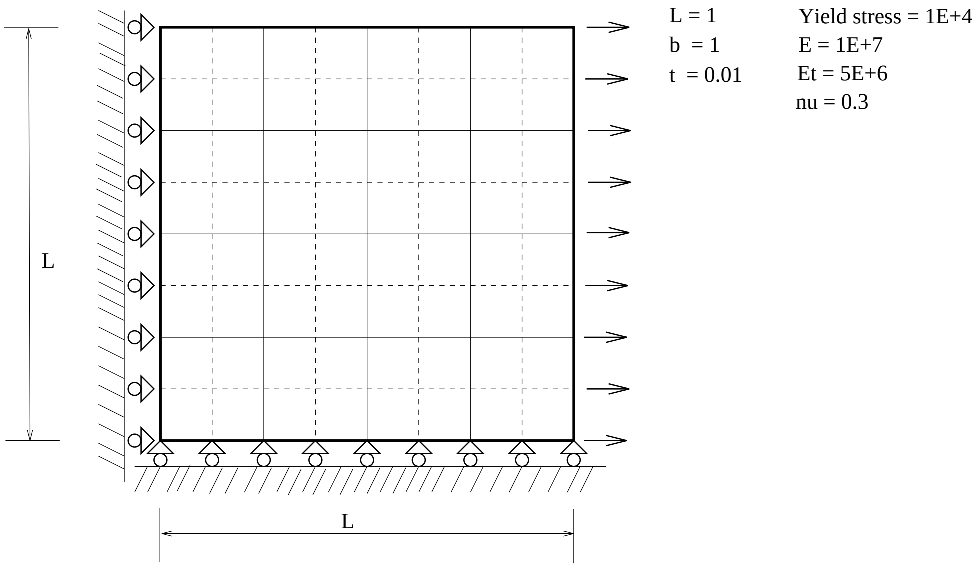

Figure 3.3: A Rectangular Plate Subjected to An Uniaxial Load

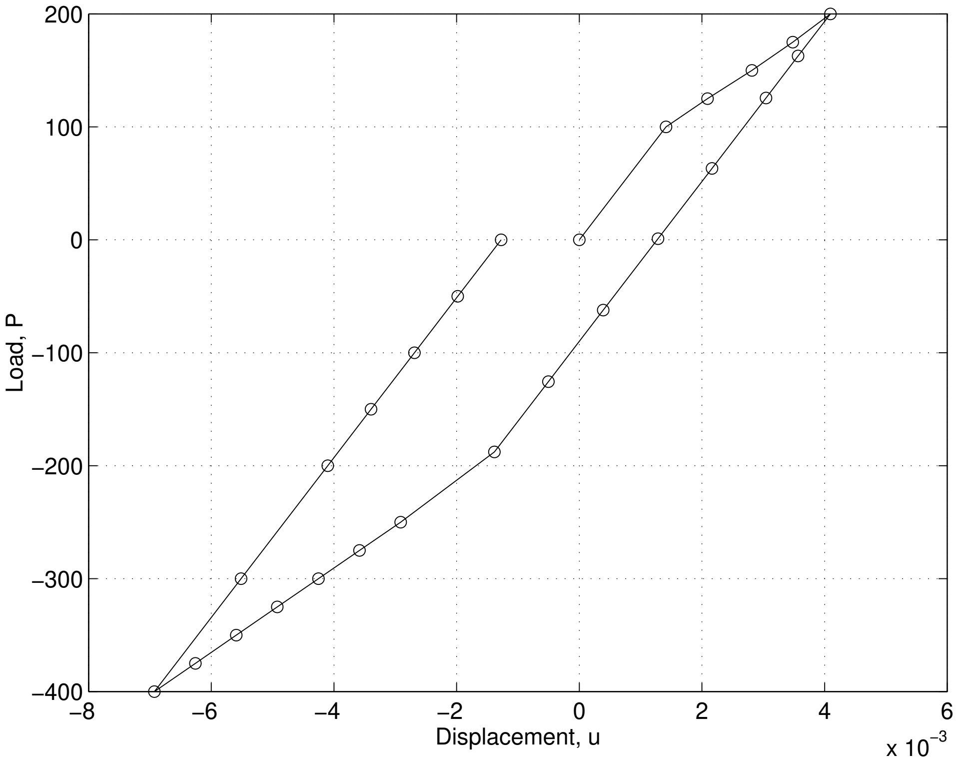

Figure 3.4: Hysteresis Loop of a Rectangular Plate under Unixaxial Load, Case 1: Bi-Linear Load Curve,Isotropic Hardening

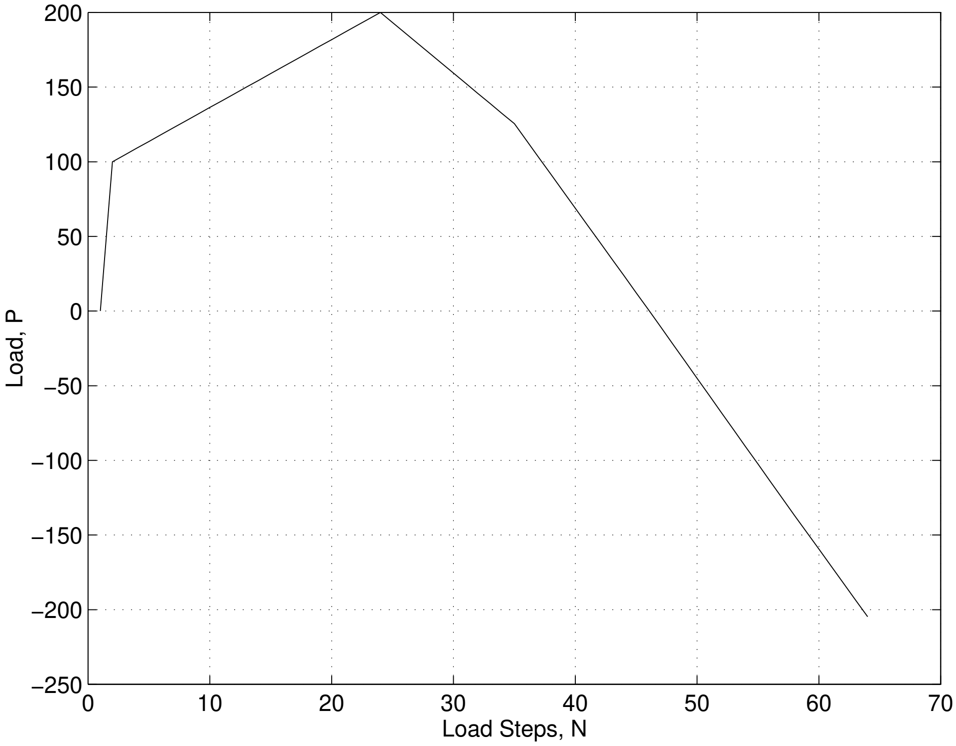

Figure 3.5: Load Steps for Case 2

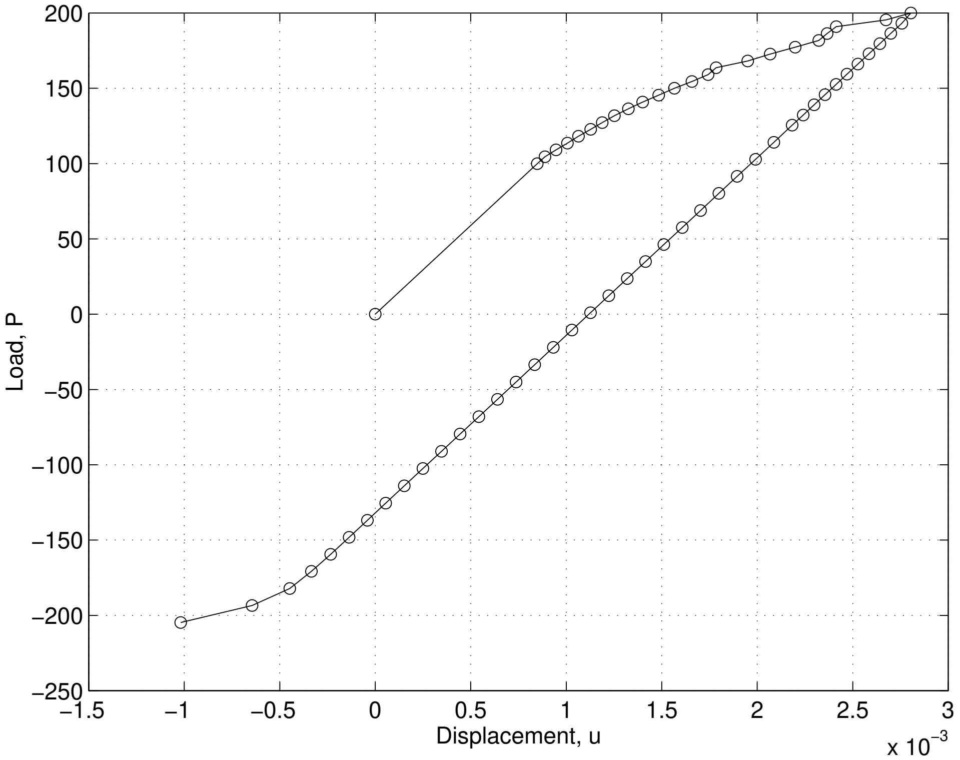

Figure 3.6: Load Displacement Curve of a Rectangular Plate under Unixaxial Load, Case 2: Ramberg_Osgood Load Curve, Isotropic Hardening

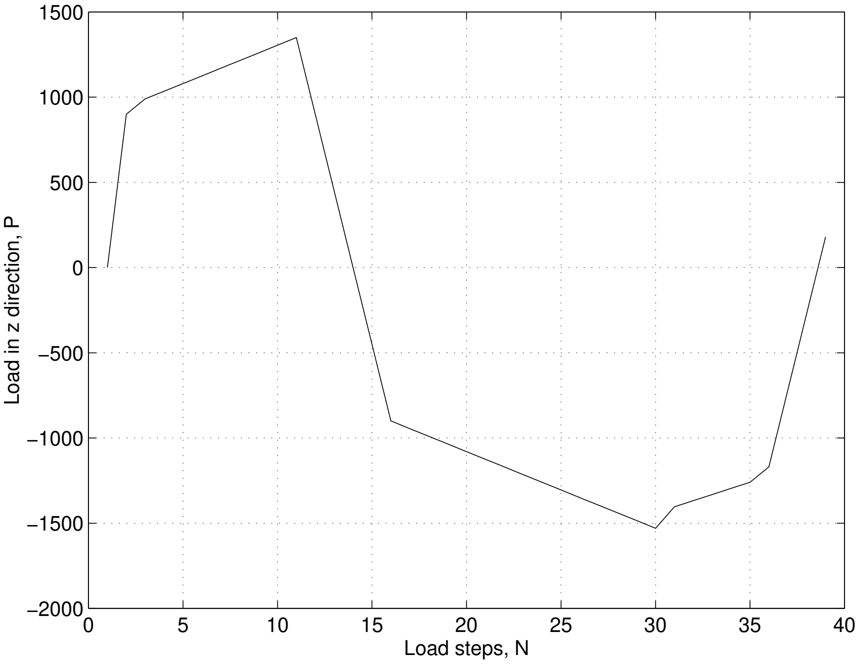

Figure 3.7: Load Steps for Cantilever Subjected To Tip Load

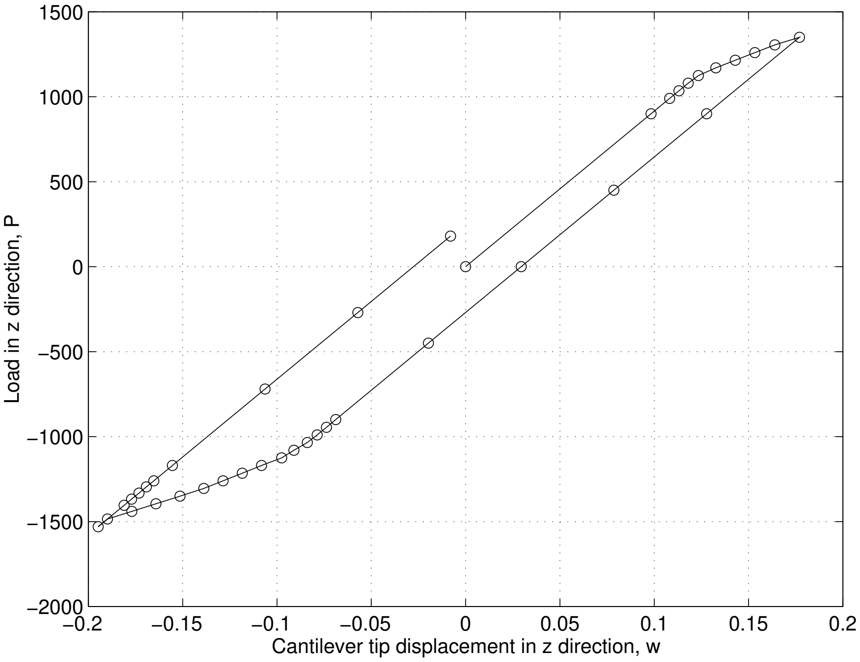

Figure 3.8: Hysteresis Loop of a Cantilever beam under Tip Load

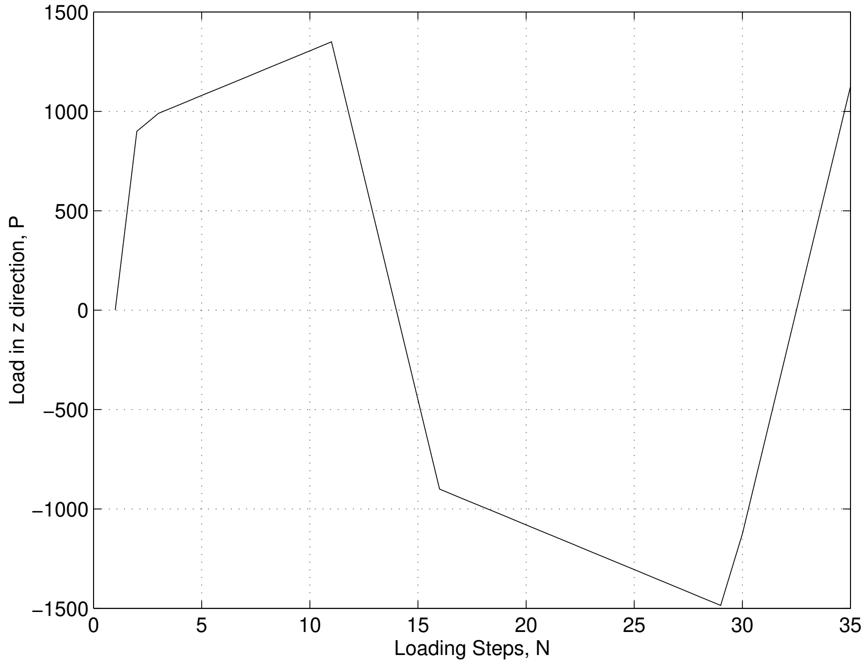

Figure 3.9: Load Steps for Cantilever Subjected To Tip Load

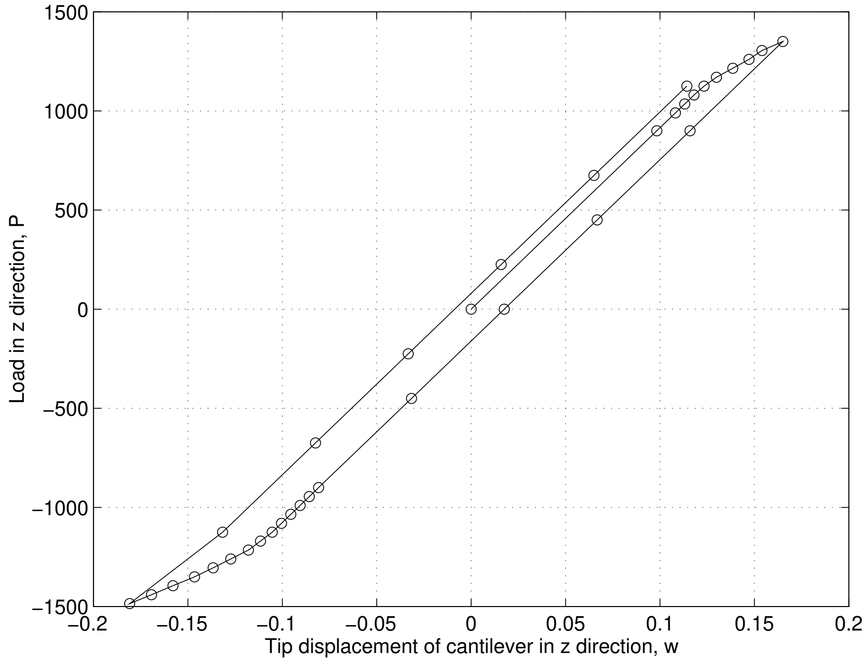

Figure 3.10: Hysteresis Loop of a Cantilever beam under Tip Load

3.2.3 Solution Strategy for Nonlinear Finite Element Problem

The purposes of this section are two-fold. First, we provide details on the use of BFGS in nonlinear finite element problems. We then illustrate the procedure by working through the details of the ALADDIN [1] input file needed to setup, and solve, the quasistatic nonlinear analysis of the cantilever beam structure.

Using BFGS in Nonlinear Finite Element Problem

The procedure for solving the nonlinear finite element problem is the same as for computing the roots of nonlinear equations, as detailed in Chapter 2. Figure 3.11 is a flow chart of the BFGS Algorithm embedded within the finite element mesh generation and solution procedure.

Input File for Cantilever Beam Problem

The input file is composed of ten sections:

Step 1 : Define Global Parameters for the analysis:

```
STOL  = 0.5;        /* Convergence tolerance in line search      */
EPS   = 1.e-5;      /* Delta for convergence in line search      */
IMAX  = 20;         /* Maximum number of iteration for BFGS      */
LMAX  = 50;         /* Maximum number of load steps              */
Ef    = 1E-1;       /* Convergence tolerance for force  balance */
Ee    = 1E-3;       /* Convergence tolerance for energy balance */
```

Step 2 : Initializse database for finite element mesh and input nodal coordinates and finite element connectivity information. The cantilever has a rectangular cross section (width = 1, height = 1, and length = 10). It is modeled with a (5×2) mesh of eight-node shell finite elements.

Step 3 : Define element properties and material properties. The cantilever is modeled with the eight-node shell finite element described in Chapter 2. The material properties are as follows – Yield Stress = 36000, Young's Modulus E = 29 $\times 10^6$, and Tangent Modulus $E_t = 0.1 \cdot E = 2.9 \times 10^6$. The material density equals 1, and Poisson's ratio is 0.2.

The stress-strain curve is the Ramberg-Osgood relationship, as defined in equation 2.49, with $\alpha = 3/7$, $n = 6$, $\epsilon_o = \sigma_o/E$, where σ_o is the Yield Stress. The plastic deformation is assumed to follow isotropic strain hardening – this is indicated by setting $\beta = 1$.

Step 4 : Setup boundary conditions – nodes at the base of the cantilever have full fixity.

Step 5 : Compile Finite Element Mesh.

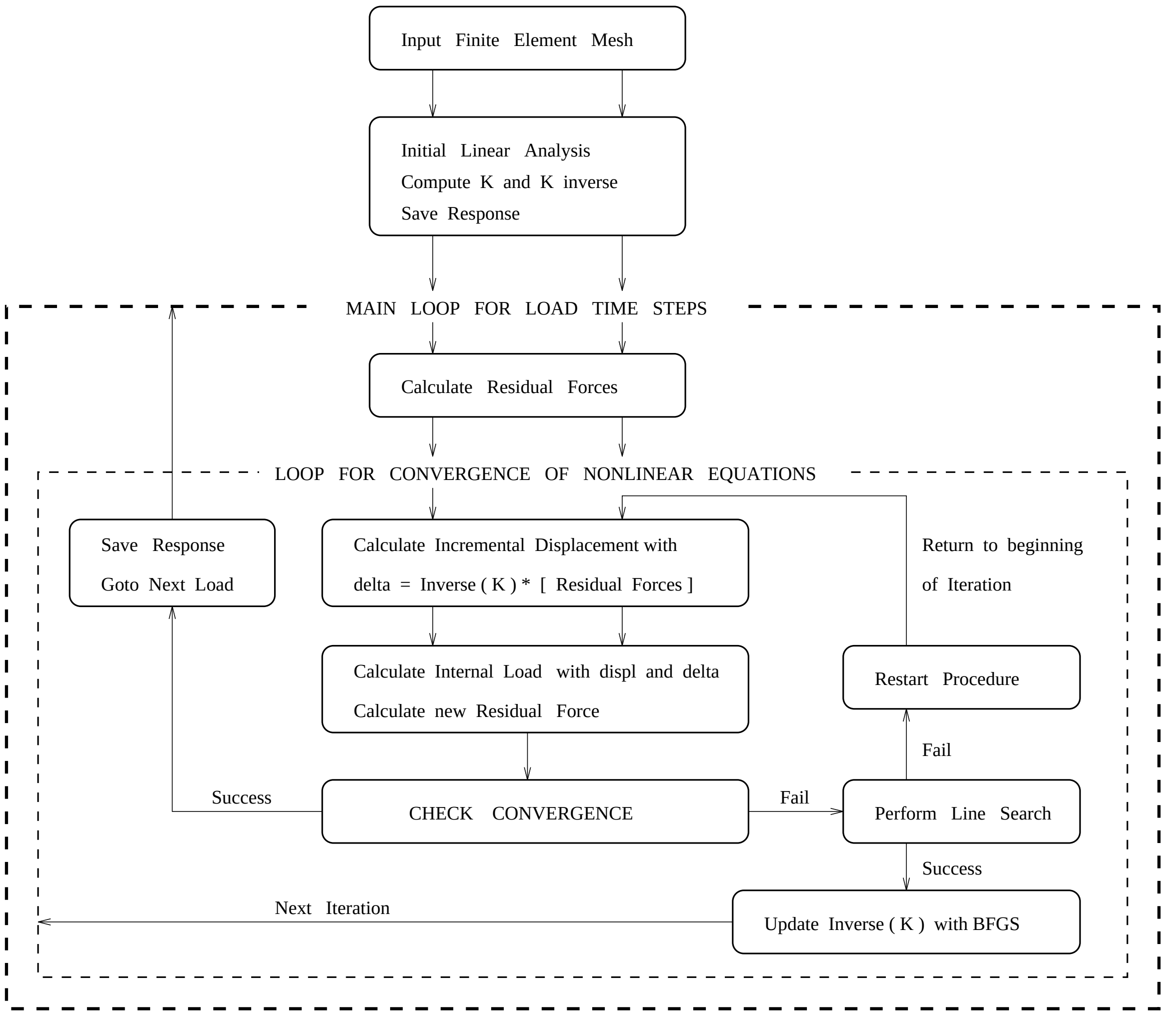

Figure 3.11: Solution Strategy for Nonlinear Finite Element Analysis

Step 6 : Define externally applied load versus time t.

$$\text{External Load}(t) = \text{Load Factor}(t) \cdot P \tag{3.5}$$

The unfactored load at the end of the cantilever, P, is $3 \cdot \texttt{Fz} = \texttt{900}$.

Step 7 : Allocate matrix memory to store a summary of the load-displacement response history. The first column of the matrix will store the step number, the second column, the factored external load, and the third column, the computed vertical displacement at the tip of the cantilever.

Step 8 : Compute mass matrix and initial stiffness matrix.

Step 9 : Use BFGS method to compute nonlinear response.

Step 10 : Print a summary of the computed response.

START OF INPUT FILE

```
/*
 *  ================================================================
 *  Nonlinear Analysis of Cantilever with Shell Finite Element
 *
 *   -- Illustrate application of BFGS algorithm to solution of
 *      nonlinear equations.
 *
 *  Written By : X.G. Chen                           July-August 1994
 *  ================================================================
 */

print "*** DEFINE PROBLEM SPECIFIC PARAMETERS \n\n";

NDimension         = 3;
NDofPerNode        = 5;
MaxNodesPerElement = 8;
ThicknessIntegPts  = 4;

STOL               = 0.5;
EPS                = 1.e-5;
IMAX               = 20;         /* Maximum number of iteration for BFGS */
LMAX               = 34;         /* Maximum number of load steps         */
Ef                 = 1E-1;
Ee                 = 1E-3;
COUNTER            = 7;

restart_counter    = 0;
SetUnitsOff;

    StartMesh();

/*  [a] : Generate Finite Element Mesh for Cantilever Beam */

    L    = 10.0;    /* Length of Beam    */
    b    =  1.0;    /* Width of Beam     */
    t    =  1.0;    /* Thickness of Beam */

    z    = 0 ;
    dL   = 2;
    n2   = L/dL + 1;
    NN   = n2;
    n1   = 3*n2 + 2*(n2-1);

    for ( i = 1; i <= n2; i = i + 1)  {
      for (j = 1; j <= 3; j = j + 1) {
           node = j + 5*(i-1);
           x    = dL*(i-1);
           y    = (b/2)*(j-2);
           AddNode(node, [x, y, z]);
      }
    }
```

```
    for ( i = 1; i < n2; i = i + 1)  {
       node = 3 + 1 + 5*(i-1);
       x    = (i-1)*dL + dL/2;
       y    = -b/2;
       AddNode(node, [x, y, z]);
       node = 3 + 2 + 5*(i-1);
       x    = (i-1)*dL + dL/2;
       y    =  b/2;
       AddNode(node, [x, y, z]);
    }

    /* Attach Elements to Grid of Nodes */

    for ( i = 1; i < n2; i = i + 1) {
       elmtno = i;
       a = 1 + 5*(i-1);
       b = 6 + 5*(i-1);
       c = 8 + 5*(i-1);
       d = 3 + 5*(i-1);

       e  = 4 + 5*(i-1);
       f  = 7 + 5*(i-1);
       gg = 5 + 5*(i-1);
       h  = 2 + 5*(i-1);
       node_connec = [a, b, c, d, e, f, gg, h];
       AddElmt(elmtno, node_connec, "name_of_elmt_attr");
    }

/*
 *  [b] : Define Element, Section and Material Properties
 *
 *        Note : beta = 1 for Isotropic strain hardening
 */

    ElementAttr("name_of_elmt_attr") { type     = "SHELL_8N";
                                        section  = "rectangular";
                                        material = "ELASTIC_PLASTIC";
                                     }

    MaterialAttr("ELASTIC_PLASTIC") { density = 1.0;
                                      poisson = 0.2;
                                      yield   = 36000;
                                      E       = 2.9E+7;
                                      Et      = 2.9E+6;
                                      beta    = 1.0;
                                      type    = "Ramberg-Osgood";
                                      n       = 6;
                                      alpha   = 3/7;
                                    }

    SectionAttr("rectangular") { width     = 1;
                                 thickness = 1;
                               }
```

```
/*  [c] : Setup boundary conditions */

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

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

    FixNode(1, bc);
    FixNode(2, bc);
    FixNode(3, bc);

/*  [d] : Add Point Nodal Loads */

    Fx = 0;   Fy = 0 ;   Fz = 300;
    Mx = 0;   My = 0 ;

    NodeLoad( n1-2, [ Fx, Fy, Fz, Mx, My]);
    NodeLoad( n1-1, [ Fx, Fy, Fz, Mx, My]);
    NodeLoad( n1,   [ Fx, Fy, Fz, Mx, My]);

/*  [e] : Compile and Print Finite Element Mesh */

    EndMesh();
    PrintMesh();

/*  [f] : Generate Load Factor Matrix for Cyclic Loading */

    Load_Factor = Matrix( [LMAX,1] );

    Load_Factor[1][1] = 1.0;
    for (k = 2; k <= LMAX; k = k +1) {
        if(k <= 10) then {
           Load_Factor[k][1] = 1.0 + 0.05*k;
        } else {
           if(k > 10 && k <= 15) then {
              Load_Factor[k][1] = 1.5 - 0.5*(k-10);
           } else {
              if(k <= 28) then {
                 Load_Factor[k][1] = -1.0 - 0.05*(k-15);
              } else {
                 Load_Factor[k][1] = -1.65 + 0.5*(k-28);
              }
           }
        }
    }

/*
 *  [g] : Allocate Matrix to store force-diplacement curve --
 *        Col 1 = step;
 *        Col 2 = load;
 *        Col 3 = displacement;
 */

    response = Matrix( [ LMAX , 3] );
```

```
/*  [h] : Compute Initial Displacement */

    print "*** Start Load Step 1 \n";
    print "*** Compute Initial Displacement \n";

    stiff  = Stiff();
    Dimen  = Dimension(stiff);
    size   = Dimen[1][1];
    lu     = Decompose(stiff);
    eload0 = ExternalLoad();
    eload  = eload0;
    displ  = Substitution(lu, eload);
    iload  = InternalLoad(displ);

    print "*** Initial Displacement = ", displ[size-2][1], "\n";

    response[1][1] = 1;
    response[1][2] = Load_Factor[1][1]*3.0*Fz;
    response[1][3] = displ[size-2][1];

    UpdateResponse();

/*
 *  [i] : User BFGS Method to compute solution to next load step
 *
 *             K(x)*delta_displ = R-F = R1
 *
 */

    I       = Diag([size, 1]);
    K_inver = Inverse(stiff);

    for (jj = 2; jj <= LMAX; jj = jj +  1) {
        print "*** Start Load Step ", jj , "\n";

        eload   = Load_Factor[jj][1]*eload0;
        R2      = eload - iload;
        R       = R2;
        R1      = R2;
        Iter_no = 0;

     /* [i.1] : Compute Solution to next load step K(x)*delta_displ = R-F = R1 */

        for (ii = 1; ii <= IMAX; ii = ii + 1) {
           print "***       Load Step ", jj ," : Iteration = ", ii, "\n";
           Iter_no     = Iter_no + 1;
           beta        = 1.0;
           delta_displ = K_inver*R2;
           iload       = InternalLoad(displ, delta_displ); /* beta = 1 */
           R2          = eload - iload;

        /* [i.2] : Line Search */
```

```
displ_temp = displ + delta_displ;
force_crt1 = L2Norm(R2);                         /* Force Criterion */
force_crt2 = L2Norm(R)*Ef;

temp2  = QuanCast(Trans(delta_displ)*R2);
energy_crt1 = abs(temp2);                        /* Energy Criterion  */
if(ii == 1) {
   temp1  = QuanCast(Trans(delta_displ)*R);
   energy_crt2 = abs(temp1*Ee);
}

if( (force_crt1 <= force_crt2) && (energy_crt1 < energy_crt2)) then {
    displ = displ_temp;
    UpdateResponse();
    break;
} else {
    temp1  = QuanCast(Trans(delta_displ)*R1);
    temp2  = QuanCast(Trans(delta_displ)*R2);

    counter = 0;
    while (temp2 > temp1*STOL + EPS) {
        counter = counter + 1;
        if(counter > COUNTER) then {
           print "***       ERROR : Too many iterations in line search \n";
           break;
        } else {
           beta = beta/2.0;
           delta_displ_temp =  beta*delta_displ;
           displ_temp       = displ + delta_displ_temp;
           iload            = InternalLoad(displ, delta_displ_temp);
           R2               = eload - iload;
           temp2            = QuanCast(Trans(delta_displ)*R2);
        }
    }

    displ = displ_temp;

 /* [i.3] : Restart for Failed Line Search */

    if( counter > COUNTER ) then {
        restart_counter = restart_counter + 1;
        if( restart_counter > 3) {
            print "ERROR >> *** Too many restarts \n";
            break;
        }

        print "***  Restart at new Initial Value \n";
        PrintMatrix(eload, iload);
        ii      = 1;
        K       = Stiff();
        K_inver = Inverse(K);
        R1      = R2;
    } else {
```

```
            /* [i.4] : BFGS Algorithm */

               gamma   = R1-R2;
               tem1    = QuanCast(Trans(delta_displ)*R1)*beta;
               tem2    = QuanCast(Trans(delta_displ)*gamma);
               COND_NO = sqrt(tem2/tem1);                    /* condition number */
               V       = COND_NO*beta*R1 - gamma;
               W       = delta_displ/tem2;
               A       = V*Trans(W);
               A       =  I + A;
               K_inver = Trans(A)*K_inver*A;

            /* [i.5] : Test Convergence Criteria */

               if (COND_NO > 1E+5) {
                   print" *** Condition number  = ", COND_NO, " \n";
                   print" *** ERROR .. Large condition number implies that    \n";
                   print"              Jocobian matrix inverse nearly singular\n";
                   break;
               }
               R1 = R2;
          }
      }
      UpdateResponse();
   }

  /* [i.6] : Save response */

     response[jj][1] = jj;
     response[jj][2] = Load_Factor[jj][1]*3.0*Fz;
     response[jj][3] = displ[size-2][1];
}

/* [j] : Print summary of force-displacement response */

   print "*** Summary of Force-Displacement Response   \n";
   print "*** ====================================== \n\n";

   PrintMatrix ( response );

   quit;
```

Points to note are:

[1] In part `[a]` we specify that this will be a two-dimensional analysis. The maximum number of degrees of freedom per node will be three, and the maximum number of nodes per element will be two. The parameters `NDimension`, `NDofPerNode`, and `MaxNodesPerElement` are used by ALADDIN to assess memory requirements for the problem storage and solution.

[2] We use a nested `for()` loop and a single `if()` statement to generate the planar layout of 24 finite element nodes. Before the boundary conditions are applied, the structure has 72 degrees of freedom. In Section `[f]` we apply full-fixity to each column at the foundation level – this reduces degrees of freedom from 72 to 60.

[3] The nonlinear analysis is more complicated than the linear static/dynamic analyses presented in Austin, Chen and Lin [1] because the elastic-plastic behavior of the material is irreversible. The stresses and strains in the cantilever are not uniquely determined by its displacements. Instead, stresses and strains are a function of the loading path/history, and then are calculated accumulately using the plastic flow rule.

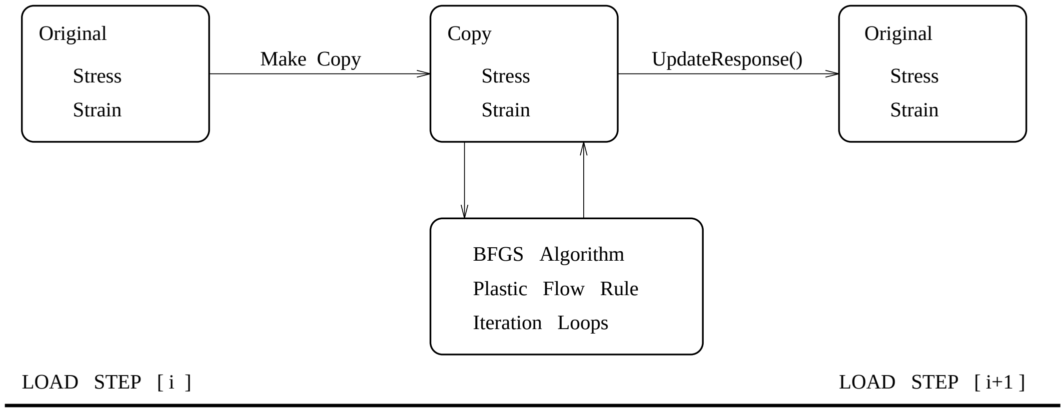

Figure 3.12: Stress/Strain Values in BFGS Algorithm and Plastic Flow Rule

Because the BFGS algorithms requires several iterations to satisfy the plastic flow rule, and may in fact, need several trial runs to achieve convergence, a copy of the stress and strain is used in the BFGS algorithm. Once covergence is achieved, the copy is used to update the stress, strain, and displacement. This update process is triggered by the `UpdateResponse()` function in ALADDIN.

Abbreviated Output File : The output file contains summaries of the mass and stiffness matrices for the shear building, and abbreviated details of the external loading, `"myload"`, and the response matrix `"response"`.

START OF ABBREVIATED OUTPUT FILE

```
===========================================
Title : DESCRIPTION OF FINITE ELEMENT MESH
===========================================

=======================
Profile of Problem Size
=======================

   Dimension of Problem        =      3
```

```
   Number Nodes                =     28
   Degrees of Freedom per node =      5
   Max No Nodes Per Element    =      8

   Number Elements             =      5
   Number Element Attributes   =      1
   Number Loaded Nodes         =      3
   Number Loaded Elements      =      0

------------------------------------------------------------------------------------
Node#       X_coord      Y_coord      Z_coord     Dx     Dy     Dz     Rx     Ry     Rz
------------------------------------------------------------------------------------

    1    0.0000e+00  -5.0000e-01   0.0000e+00     -1     -2     -3     -4     -5
    2    0.0000e+00   0.0000e+00   0.0000e+00     -6     -7     -8     -9    -10
    3    0.0000e+00   5.0000e-01   0.0000e+00    -11    -12    -13    -14    -15
    4    1.0000e+00  -5.0000e-01   0.0000e+00      1      2      3      4      5
    5    1.0000e+00   5.0000e-01   0.0000e+00      6      7      8      9     10

    ...... details of nodal coordinates removed .....

   24    9.0000e+00  -5.0000e-01   0.0000e+00    101    102    103    104    105
   25    9.0000e+00   5.0000e-01   0.0000e+00    106    107    108    109    110
   26    1.0000e+01  -5.0000e-01   0.0000e+00    111    112    113    114    115
   27    1.0000e+01   0.0000e+00   0.0000e+00    116    117    118    119    120
   28    1.0000e+01   5.0000e-01   0.0000e+00    121    122    123    124    125

--------------------------------------------------------------------------------------
Element#   Type  node[1]   [2]   [3]   [4]   [5]   [6]   [7]   [8]  Element_Attr_Name
--------------------------------------------------------------------------------------

       1 SHELL_8N      1     6     8     3     4     7     5     2  name_of_elmt_attr
       2 SHELL_8N      6    11    13     8     9    12    10     7  name_of_elmt_attr
       3 SHELL_8N     11    16    18    13    14    17    15    12  name_of_elmt_attr
       4 SHELL_8N     16    21    23    18    19    22    20    17  name_of_elmt_attr
       5 SHELL_8N     21    26    28    23    24    27    25    22  name_of_elmt_attr

------------------------
Element Attribute Data :
------------------------

ELEMENT_ATTR No.   1  : name = "name_of_elmt_attr"
                      : section = "rectangular"
                      : material = "ELASTIC_PLASTIC"
                      : type = SHELL_8N
                      : Young's Modulus =  E =        2.900e+07
                      : Yielding Stress = fy =        3.600e+04
                      : Poisson's ratio = nu =        2.000e-01
                      : Density         =        1.000e+00
                      : Inertia Izz     =        0.000e+00
                      : Area            =        0.000e+00

EXTERNAL NODAL LOADINGS
```

```
--------------------------------------------
Node#      Fx      Fy      Fz     Mx     My
   26    0.000   0.000 300.000  0.000  0.000
   27    0.000   0.000 300.000  0.000  0.000
   28    0.000   0.000 300.000  0.000  0.000

============= End of Finite Element Mesh Description ==============

*** Start Load Step 1
*** Compute Initial Displacement
*** Initial Displacement =    0.09833
*** Start Load Step         2
***       Load Step         2  : Iteration =          1
*** Start Load Step         3
***       Load Step         3  : Iteration =          1
*** Start Load Step         4
***       Load Step         4  : Iteration =          1
*** Start Load Step         5
***       Load Step         5  : Iteration =          1
***       Load Step         5  : Iteration =          2
***       Load Step         5  : Iteration =          3
*** Start Load Step         6
***       Load Step         6  : Iteration =          1
***       Load Step         6  : Iteration =          2
***       Load Step         6  : Iteration =          3
***       Load Step         6  : Iteration =          4
***       Load Step         6  : Iteration =          5
*** Start Load Step         7

    ....... details of load step and BFGS Algorithm Information removed .....

*** Start Load Step        33
***       Load Step        33  : Iteration =          1
*** Start Load Step        34
***       Load Step        34  : Iteration =          1

*** Summary of Force-Displacement Response
*** ====================================

MATRIX : "response"

row/col                  1            2            3
   1            1.00000e+00  9.00000e+02  9.83343e-02
   2            2.00000e+00  9.90000e+02  1.08168e-01
   3            3.00000e+00  1.03500e+03  1.13084e-01
   4            4.00000e+00  1.08000e+03  1.18001e-01

  ...... details of response matrix removed .....

  32            3.20000e+01  3.15000e+02  1.58910e-02
  33            3.30000e+01  7.65000e+02  6.50581e-02
  34            3.40000e+01  1.21500e+03  1.14225e-01
```

Summary's of the applied load versus step number, and applied load versus vertical tip diplacement of the cantilever are shown in Figures 3.9 and 3.10, respectively.

Chapter 4

Conclusions and Future Work

4.1 Conclusions

The purpose of this study has been to extend ALADDIN's functionality to the solution of finite element problems containing material nonlinearities. The conclusions and findings of this study are as follows:

[1] We have investigated the features needed by ALADDIN's language and kernel for the solution of problems with material nonlinearities. The direction of research has been motivated in part by the formulation of four- and eight-node shell finite elements, and validated via numerical experiments on problems having linear and nonlinear material behavior. In our preliminary experiments on linear problems, we observed that the four node bi-linear shell element performs poorly. Consequently, it the was removed from further consideration in the study. We also observed that when an assembly of interconnected shell finite elements has reasonable fidelity, the eight node element is capable of accurately predicting behavior. A summary of numerical results is presented in Chapter 3 – we have shown that ALADDIN can compute nonlinear in-plane displacements of a flat plate, and out-of-plane bending of a cantilever structure. In both applications, material nonlinearities are modeled with bi-linear and Ramberg-Osgood stress-strain curves.

[2] The numerical experiments have shown that the elastic-plastic rate integration algorithm and the BFGS update algorithm can reproduce the hysteresis loops of load-displacement. The basic BFGS algorithm was first implemented and tested on suites of nonlinear equations by Austin and co-workers [1]. Surprisingly few modifications to the basic BFGS algorithm were needed in the transformation from numerical to finite element equations. ALADDIN's kernel was extended so that it could copy and save displacements, stresses and strains, and a function called `UpdateResponse()` was added to ALADDIN's vocabulary. As the name suggests, `UpdateResponse()` updates the system's state once equilibrium is achieved. We note that because the source of the nonlinear equations – nonlinear constituitive relations in the shell material – is completely decoupled from ALADDIN's kernel and the `UpdateResponse()` response function, there is reason to believe that these

program extensions are general. We would not be surprised to learn, for example, that with suitable adjustments to the finite element library, ALADDIN is now also capable of solving other types of nonlinear problems (e.g. nonlinearities due to large geometric displacements and/or contact/impact).

[3] The shell finite elements have been formulated under the assumption that displacements within an element can be represented by a linear combination of shape functions and nodal displacements. The result of this formulation is an element level stiffness matrix. A fundamental problem with the stiffness-based approach to modeling is the difficulty in computing nonlinear behavior near an element's ultimate resistance – this is where the element's stiffness matrix may suddenly become singular, thereby causing a force-displacement relationship that is no longer unique. If the global stiffness matrix also becomes singular (or severely ill-conditioned), then an engineer may have no choice but to terminate the computations. A second problem with stiffness-based methods is the difficulty in computing solutions to nonlinear finite element equations. Taucer et al. [29] point out that when displacements within an element are modeled with cubic interpolation functions, the distribution of curvature within the element will be linear. The latter may provide a poor approximation to the behavior of real systems, especially within regions of a structure that are highly nonlinear. And as such, computing solutions to the associated equations of equilibrium may be very difficult.

4.2 Future Work

The future work can be divided into two areas: finite element applications, and ALADDIN infrastructure.

Finite Element Applications : The near-term application area for ALADDIN is design and analysis of earthquake resistant buildings and highway bridge structures. The results of this study are an intermediate step in this direction in the sense that rational modeling of seismically resistant structures during severe ground motions often requires nonlinear time-history analyses. There is considerable evidence, however, that engineers will not incorporate these modeling procedures into their daily practice unless they are clear, reliable, and robust [14, 29]. Unfortunately, we are not at that point yet. Our recommendations for future work are:

[1] `Flexibility-Based Elements` : Further work is needed to mitigate the problems listed in Item [3] of Section 4.1 since most practicing engineers will have neither the time, nor numerical background to deal with them.

We are now implementing a three-dimensional flexibility-based beam-column element, formulated by Filippou and co-workers [29]. Our objective is to use this finite element – or a modified version of it – for the modeling of nonlinear hysteretic behavior in base-isolation pads. Unlike stiffness-based finite elements, which employ shape functions of displacement, flexibility-based finite elements use distribution functions of bending moment and axial force that are capable of maintaining equilibrium and compatibility within an element, and converging to a state that satisfies the section constituitive relations within a specified tolerance. These characteristics apply to both the linear and plastic regimes of an element.

The flexibility-based elements must be accompanied by a special algorithm for state determination for the computation of resisting forces in the elements. We are currently examining these details to see what, if any, extensions to ALADDIN will be required.

ALADDIN Infrastructure : We have shown that even though its language is small, ALADDIN can solve a wide range of matrix and finite element problems in almost a seamless manner. The future work on ALADDIN's infrastructure should focus on:

[1] `User-Defined Functions` : One problem with the BFGS algorithm and associated definition of the finite element problem is that the input files are now becoming quite long and detailed. There is a need to extend ALADDIN's language and kernel so that user-defined functions are supported. With such a feature in place, BFGS-type algorithms could be packaged as file modules, and simply loaded into the finite element problem description files. This feature would enhance the nonlinear solution procedure(s) reliability by reducing the demands on an engineer's experience.

[2] `CORBA/OLE Interfaces` : In this project and our previous work [1, 16], ALADDIN modules have been written in C, and have followed the model of custom-built development described in Chapter 1. We are confident that the same pathway of implementation would work for the extension of ALADDIN's functionality to the solution of optimization problems.

There is, nonetheless, a strong need to explore alternative pathways of module implementation, such as the feasibility of extending the the language/library interface (see Figure 1.3) so that it can handle CORBA Interface Definition Language specifications. Such a feature would allow engineers to program new finite element routines and their IDL mappings in their language of choice (e.g. Fortran, C, C++), and then link the compiled object code into ALADDIN. IDL mappings could be added to legacy finite element code, and then linked into ALADDIN. In either case, the stub routines required for the IDL/finite element interface need to be worked out, as do details of "modifications to the legacy finite element code" for compatibility with IDL. The implementation of optimization routines would follow a similar procedure.

Future work should explore the viability of user-defined functions having IDL-compliant interfaces. Such a combination would allow engineers to define an application specific function within ALADDIN, and leave the details of the function evaluation to an external module.

Bibliography

[1] Austin M. A., Chen, X. G. and Lin, W-J. ALADDIN: A Computational Toolkit For Interactive Engineering Matrix And Finite Element Analysis. Technical Research Report TR 95-74, Institute for Systems Research, College Park, MD 20742, August 1995.

[2] 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.

[3] 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.

[4] 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.

[5] Bathe K. J. *Numerical Methods in Finite Element Analysis.* Prentice-Hall, Englewood Cliffs, N.J., 1976.

[6] Bathe K. J. *Finite Element Procedures in Engineering Analysis.* Prentice-Hall, Englewood Cliffs, N.J., 1982.

[7] Bathe K. J., Bolourchi, S. A Geometric and Material Nonlinear Plate and Shell Element. *Journal of Computers and Structures*, 11:23–48, 1980.

[8] Bathe K. J., Cimento, A. P. Some Practical Procedures for the Solution of Nonlinear Finite Element Equations. *Computer Methods in Applied Mechanics and Engineering*, 22:59–85, 1980.

[9] Belytschko, Ted, Lin, Jerry. I. and Tsay, Chen-Shyh. Explicit Algorithms for the Nonlinear Dynamics of Steels. *Computer Methods in Applied Mechanics and Engineering*, 42:225–251, 1984.

[10] Ben-Natan R. *CORBA : A Guide to Common Object Request Broker Architecture.* Computing : McGraw-Hill, 1995.

[11] Brockschmidt K. *Inside OLE.* Microsoft Press, Redmond, Washington, 2nd edition, 1995.

[12] Crisfield, M. A. *Non-Linear Finite Element Analysis of Solid and Structures.* John Wiley & Sons, 1991.

[13] Dodds R. H. Numerical Techniques for Plasticity Computations in Finite Element Analysis. *Computers & Structures*, 26(5):767–779, 1987.

[14] Engelmann R.E., Whirley R. G. ISLAND Interactive Solution Language for an Adaptive NIKE Driver – User Manual. Report UCRL-MA-108721, Lawerence Livermore National Laboratory, Livermore, CA, October 1991.

[15] Gall H., Jazayeri M., Klosh R. Research Directions in Software Reuse : Where to go from here ? In M. Samadzadeh and M. Zand, editor, *Software Engineering Notes*, pages 225–228. ACM Press, Auguest 1995.

[16] Jin Lanheng. Analysis and Evaluation of a Shell Finite Element with Drilling Degree of Freedom. Technical report, University of Maryland, College Park, Maryland 20742, 1994.

[17] Johnson E.H., Neill D.J., Herendeen D.L., and Canfield R.A. Automated Structural Optimization System (ASTROS) User Training Workshop. Research Report AFWAL-TR-88-3101, Flight Dynamics Laboratory, Wright-Patterson AFB, Ohio 45433-6523, March 1989.

[18] Kronlof K. *Method Integration : Concepts and Case Studies.* John-Wiley and Sons, 1993.

[19] Lee, S. Notes for Computational Plasticity. Lecture Notes, Aerospace Department, University of Maryland, College Park, MD 20742, 1991.

[20] Mindlin R. D. Influence of rotatory inertia and shear in flexural motions of isotropic elastic plates. *Journal of Applied Mechanics*, 18:31–38, 1951.

[21] Naghdi, P. M. *The theory of Shells and Plates, in Handbuch der Physik*, volume VI a/2. Springer Verlag, Berlin, 1972.

[22] Noor, A. K. List of Books, Monographs, Conference Proceedings and Survey Papers on Shells. In A. K. Noor,T. Belytschko, and J. Simo, editor, *Analytical and Computational Models of Shells*, volume 3. Americal Society of Mechanical Engineering, 1989.

[23] Noor, A. K., Belytschko, T. and Simo, J. In A. K. Noor,T. Belytschko, and J. Simo, editor, *Analytical and Computational Models of Shells*, volume 3. Americal Society of Mechanical Engineering, 1989.

[24] Noor, A. K., Belytschko, T. and Simo, J. Praface. In A. K. Noor,T. Belytschko, and J. Simo, editor, *Analytical and Computational Models of Shells*, volume 3. Americal Society of Mechanical Engineering, 1989.

[25] Nyssen, C. An efficient and accurate iteration iterative method allowing large incremental steps to solve elasto-plastic problems. *Computer & Structures*, 13:63–71, 1981.

[26] Reissner, E. The effect of transverse shear deformation on the bending of elastic plate. *Journal of Applied Mechanics*, 12, March 1945.

[27] Salter K.G. A Methodology for Decomposing System Requirements into Data Processing Requirements. *Proc. 2nd Int. Conf. on Software Engineering*, 1976.

[28] Selic B., Gullekson G., and Ward P.T. *Real-Time Object-Oriented Modeling*. John Wiley Professional Computing, 1994.

[29] Taucer F., Spacone E., Filippou F.C. A Fiber Beam-Column Element for Seismic Response Analysis of Reinforced Concrete Structures. Technical Report Report No UCB/EERC-91-17, Earthquake Engineering Research Center, University of California, Berkeley, December 1991.

[30] Timoshenko, S. P. and Woinowsky-Kriger, S. *Theory of Plates and Shells*. McGraw-Hill Book Company, New York, 2 edition, 1959.

[31] Weaver, W. Jr. and Gere, J. M. *Matrix Analysis of Framed Structures*. D. Van Nostrand Company, 135 West 50th Street, New York, NY 10020, 2 edition, 1980.

[32] Weaver, W. Jr. and Johnston, P. R. *Structural Dynamics by Finite Elements*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1987.

[33] Wilkins, M. L. Calculation of elastic-plastic flow. In Alder, B. et.al, editor, *Methods of Computational Physics*, volume 3, pages 253–272. Academic Press, New York, September 1964.

[34] Zienkiewicz, O. C. and Taylor, R. L. *The Finite Element Method*, volume 2. McGraw-Hill Book Company, New York, 2 edition, 1991. Solid and Fluid Mechanics, Dynamics and Non-Linearity.