JAVA APPLET FOR TEACHING FINITE ELEMENT ANALYSIS

by

Suryanarayana Raju Sagi Venkata Naga

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute & State University
In partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE
in
Civil Engineering

Approved:

_____
Dr. Kamal B. Rojiani, Chairman

_____ _____
Dr. Raymond H. Plaut Dr. C. L. Roberts - Wollmann

February 3, 2006
Blacksburg, Virginia

Key Words: Java Applets, Finite Element Analysis, Web-Based Education.

JAVA APPLET FOR TEACHING FINITE ELEMENT ANALYSIS

by

Suryanarayana Raju Sagi Venkata Naga

Committee Chairman: Kamal B. Rojiani,
Charles E. Via, Jr. Department of Civil Engineering
Virginia Polytechnic Institute and State University

(ABSTRACT)

A Java applet is developed to interactively and dynamically illustrate the fundamental concepts of finite element analysis. An applet is a computer program written in the object-oriented Java programming language and is embedded in web pages. Java applets are well suited for delivering interactive graphical content over the Internet since they are platform and operating system independent. The applet developed includes a wide range of elements including one-dimensional truss and beam elements, triangular and quadrilateral plane stress and plane strain elements, and two-dimensional four-node and eight-node iso-parametric elements and plate elements. Along with the applet there is a series of web pages describing the fundamental concepts of finite element analysis, example problems and instructions for use. The applet provides a novel approach for teaching basic finite element analysis concepts. It provides students a means for checking their work, reinforces fundamental concepts learned in class, and enhances students' learning experiences by allowing them to experiment by building and analyzing complex models and visualizing results as changes are made to the model. The applet can be used as supplementary material complementing classroom and textbook instruction.

Acknowledgements

I would like to extend my sincere thanks to my advisory committee chair Dr. Rojiani for his encouragement, time and kind support at every stage of this work, without whose help this task would not have been accomplished. Likewise, I would like to thank Dr. Plaut and Dr. Wollmann for kindly agreeing to serve as my committee members.

My family has been a constant source of support and encouragement throughout the studies and my life. I am grateful to my father Rama Raju Sagi, my mother Satya Sagi and my sister Sri Vidya.

I would like to express my gratitude to Vyas for his guidance in Java during this work.

I wish to thank my roommate and friend Vivek Sethi who was a constant support to me during the course of my graduate studies at Virginia Tech and who also helped me with the testing of my program. His uncomplaining nature has made my stay much more enjoyable.

I thank Ravi who helped me with the documentation of my work. I would also like to thank my friends Suresh, Sowjanya, Sravan, Mahipal, Manoj and Hasan and my cousins Varma and Pavan for their encouragement and support.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Introduction

The finite element method is the most widely used tool for computer-based numerical solution of a wide range of engineering problems. This method is used for solving problems in application areas such as structural analysis, heat transfer, fluid mechanics, vibrations, seepage, electrical and magnetic fields, and many other fields. With the application of this method, problems that were previously intractable are now solved routinely.

The finite element analysis method is an approximate method where a large number of simultaneous equations are solved and a considerable amount of computational effort is required. With the rapid advancement in programming technologies and computing hardware, the finite element method has become a popular method for analyzing structures.

The rapid increase in enrollment in engineering colleges coupled with a shortage of teachers has posed new educational challenges. These challenges have motivated educators to look for new technology such as the Internet. Homes, public libraries, and corporate learning rooms with access to the Internet have become new places for learning and have reduced the need for the traditional teacher and classroom approach to education.

Web based teaching has enhanced and pioneered new learning opportunities. It is a new medium of instruction which can reach everyone without time or distance constraints. This has led to the creation of a curriculum which is more student centric and allows them to work at their own pace. Important elements of web based learning are the incorporation of graphics and the opportunity for students to interact with the application. Java applets are an excellent means for developing interactive programs to explain fundamental concepts. The work presented here utilizes Java to create a web based learning resource for teaching finite element analysis.

1.2 Objective and Scope

Finite element analysis is generally taught at the graduate level in most universities due to the complexities associated in explaining the concepts. However, with the rapid growth and widespread use of this method, there is now a growing trend and need to teach finite element analysis at the undergraduate level. Given the importance of finite element analysis, it is essential that engineering students have a good understanding of the fundamental concepts of finite element analysis and the computational procedures and programming methodologies used in the implementation.

The primary objective of this work is to develop a web-based applet in Java for teaching finite element analysis. The applet incorporates a wide range of finite elements such as one-dimensional truss and beam elements, triangular and quadrilateral plane stress and plane strain elements, two-dimensional four-node and eight-node iso-parametric elements, and plate elements. The applet is interactive and makes extensive use of graphics. It has features for entering the finite element model of the structure such as joint coordinates, supports, connectivity information, element data, material properties, and loading, and for graphically displaying the finite element model as it is entered.

A second goal of the work is to develop a website where the applet will be hosted. The web site has detailed information on the formulation of the various elements, implementation details such as a description of the various classes used to represent the different elements and the analysis procedure, and instructions for using the applet and representative examples. To verify the accuracy of the program, results obtained from this applet are compared with those obtained from SAP 2000, a commercial finite element analysis program.

1.3 Organization

This thesis is divided into nine chapters including the Introduction. The second chapter deals with an overview of the finite element analysis method, the use of the web for teaching, and the Java language. Chapter 3 presents the development of the one-

dimensional bar element and the two-dimensional truss elements and describes the implementation of these elements in Java. The two-dimensional beam element and its implementation in a Java applet is presented in Chapter 4. In Chapter 5, the development of membrane elements such as the CST element, four-node iso-parametric quadrilateral element, eight-node iso-parametric quadrilateral element, and their implementation in Java is discussed. Chapter 6 presents the development of plate bending elements and their implementation in Java. A description of the Java applet, the program structure, and the user interface is given in Chapter 7. In Chapter 8 the results obtained for test problems are compared with SAP 2000. Finally, Chapter 9 presents the summary of work done and offers suggestions for future development.

Chapter 2: Background

2.1 Introduction

The finite element method was first introduced by Richard Courant in 1943. The application of this method gained significance in the early to late 1950's when it was used for airframe and structural analysis. This was the time during which Turner et al. derived stiffness matrices for truss, beam, and other elements (Chandrupatla et al.. 1997). The term finite element was coined and used by Clough (1960) as a finite set of discrete objects used to model a continuum (Chandrupatla et al. 1997). An element is an individual piece used in approximating a continuum.

In a finite element problem, a continuous structure is discretized with a set of finite elements. The material and physical governing relationships such as stress-strain, compatibility, and equilibrium principles are applied over these elements in terms of unknown values at nodes (Chandrupatla et al. 1997). The individual elements are then assembled to form the structure. The assembled system along with the loads and the boundary conditions reduce to a set of equations. The solution of this system gives the approximate behavior of the structure.

The accuracy of the solution depends mainly on two factors, the function used to approximate the element behavior and the number of elements into which the structure is discretized. This has led to the development of higher order elements which can predict the behavior of a structure with fewer elements. This method is also known as the p-method of refinement where the polynomial function is modified for refining the results. Another method for decreasing errors is by processing more equations (Cook 1989), that is, representing the structure with a large number of simpler elements, also known as the h-method of refinement.

Recent developments in the finite element method include the introduction of different elements to solve complex structural problems. There are five major types of elements – truss, beam, two-dimensional plane stress, shell, and solid elements.

The attractiveness of the finite element method is mainly due to its versatility since it can be applied to structures with arbitrary shape and irregular loadings. Another feature of the finite element method is its close resemblance to a real physical structure. Thus the mathematical model is not purely an abstraction (Cook 1989). Hence this method provides better insight into understanding the behavior of a structure. The division of a continuum into smaller sub-domains called elements has many other advantages. It allows inclusion of dissimilar materials and enables the accurate representation of the solution within each element to deal with local effects (Reddy 1993).

Solving the system of equations is computationally intensive and this method would not have been so popular if digital computers were not available. The widespread success of the finite element method is partly attributed to the rapid development in computers, availability of powerful programming languages, and the development of commercial software packages such as ANSYS, ABAQUS, NASTRAN, and SAP. The advances in and the ready availability of computers and software has brought this method within the reach of most engineers and students (Chandrupatla et al. 1997).

The finite element method also has some disadvantages. In general, the analysis of a finite element problem requires a large amount of input data. Also, the output obtained can be quite extensive and has to be properly sorted out. Hence engineering judgment and experience are required for making proper use of the results obtained (Cook 1989). This method cannot be taken for granted just because of its superiority as compared to classical methods of analysis. The improper use of elements may lead to inaccurate results. As with other software, providing a finite element program with inappropriate data and support conditions may lead to incorrect results. Hence it is recommended that the results obtained from finite element analysis be compared with those obtained from manual calculations made on a simplified model of the structure.

2.2 Web Based Teaching

The World Wide Web has emerged as a powerful tool for teaching. The Web has allowed people to share information unprecedented in human history (Wikipedia 2005). The Internet has created a virtual world wherein people from any place can learn about anything at any time, thereby erasing geographical boundaries and time frames. This new technology has paved the way for creating universal resources for teaching.

Web based teaching allows integrated environments of various technologies to support diverse learners' needs via the internet (Storey et al. 2002). Storey et al. (2002) emphasize the multiple roles played by the web in complementing class room work and enhancing students understanding.

In a report submitted to the President of the United States, the Web Based Education Commission (2000) headed by then senator Bob Kerry calls on educators to use the full potential of the web as a tool for learning, to expand the horizons of students of all ages. The report speaks about the advantages of the web and multiple paths such as graphics, video, and sound. The commission also suggests that the administration should embrace "e-learning" as an agenda of the nation's federal education policy.

In 2004 NASA launched a web based education program for providing resources to teachers and students about microgravity. The site has attracted students from over 100 countries with 1.5 million hits over the past two years. The success of this program led to further innovation of web imparting resources at various levels, and special features such as quizzes, puzzles, and games were added to make learning more interesting.

Web based learning has added a new dimension to computer based teaching. Traditional computer based teaching methods were created using MS-PowerPoint slides or animations generated by software packages like 3Dmax, Flash, GIF animator, etc. Although these techniques helped in understanding fundamental concepts in a better manner, they had several limitations. A significant limitation was the inability to perform real time simulations and calculations. They could only be used to demonstrate a set of predefined examples which restricted the students' understanding of the concepts under a different set of conditions. This limitation can be avoided if the teaching method is based on interactive software which makes it possible to explain concepts for a variety of

conditions and also allows the students to explore various possibilities and behavior under a variety of inputs.

Another issue of traditional computer based pedagogy is that it uses large amounts of resources. Animation file sizes are huge and a lot of bandwidth is required for transferring these files across the Internet. This becomes a serious issue when a student or tutor in a remote place cannot afford the cost associated with high bandwidth. Also, any updates made to the software require that these files be downloaded again.

The problems mentioned above can be overcome if web-based instructional units are developed in the Java programming language. With Java, it is possible to develop interactive web based computational software that can be accessed over the Internet with a Java enabled browser.

2.3 Java Programming Language

Java is an object oriented programming language developed by Sun Microsystems in the year 1991. There were five primary goals in the creation of the Java language (Wikipedia, 2005)

- It should use the object oriented programming methodology
- It should allow the same program to run on different platforms
- It should contain built-in support for using computer networks
- It should allow the code to be executed securely from remote locations
- It should be easy to use, like C++

Java is an object oriented programming language, that is, the program is composed of a collection of units called objects. In an object oriented program each object can communicate with other objects or can inherit properties from other objects. Object oriented programming provides more flexibility, and changes to a program can easily be made. This property is what makes Java an ideal tool for developing finite element analysis software.

To manage increasing complexity, object oriented programming organizes a program around its data. Classes are templates used to construct objects. The following are the basic principles of object oriented programming (Hunt 1998).

Abstraction: An abstraction denotes the essential characteristics of an object that distinguishes it from all other objects and thus provides well defined boundaries. The concept of abstraction is used to break large complex procedures into small simple procedures.

Encapsulation: This is the process of hiding details of objects from other objects. Each method or variable in a class may be marked private or public. The public interface allows the methods in a class to be accessed by other objects. However, private methods and data can only be accessed within the class itself.

Inheritance: It is the process by which one object acquires the properties of another object (Shildt 2005). In object oriented programming, classes can be divided into a subclass or a superclass. A subclass is derived from a superclass and inherits all its properties. Furthermore, the subclass can have additional properties that give it a unique identity.

Polymorphism: It is a feature which allows one interface to be used for a general class of actions. It is essentially the ability to request that the same operation be performed by different objects. How the request is processed depends on the object that receives the request. Effectively, this means that one can ask many different objects to perform the same action.

When properly applied, all the above mentioned properties combine to produce a programming environment that supports the development of far more robust and superior programs.

Many object oriented programming languages have been developed in recent years. These include C++, C#, and Java. Java is considered the most popular object oriented programming language because it has several unique features. Java is a simple language developed along the lines of C and C++ and has a similar syntax to C++. Hence, most programmers who are already versed in C++ have little trouble learning Java. Java has better memory management features. Java has a special class for garbage collection that dynamically allocates memory and prevents memory loss. In other programming languages the programmer must manually allocate and free all dynamic memory, and any mistake in allocating or de-allocating memory leads to poor

performance. There is also an error exception handling class built into Java. With proper use of this class, most run-time errors can be easily managed.

2.4 Java and Web Based Teaching

Java is platform independent. That is, the same code can be used for executing the program on various platforms like Windows, Mac, or Linux. This is a property that has made Java a popular programming language. With Java it is possible to use the same application on different platforms.

Java is a multithreading language. It can easily handle thousands of tasks simultaneously (Newman et al. 1996). Nikishkov (2004), who developed a platform independent Java application for teacher student interaction, stresses the importance of a language like Java with inherent abilities to create multithreaded applications as multiple clients (students) work with the server at the same time.

Large sets of data have to be handled when performing finite element analysis. The various input data such as coordinates, type of element, restraints conditions, and nodal loads can be represented as objects. Also, different objects can be created for handling elements. This makes it possible to add new elements to the program without having to make extensive changes to other parts of the program. Object oriented design is a mechanism which allows modules to "plug and play" (Newman et al. 1996).

Java programs can be embedded in HTML pages and can be easily transmitted over the internet. These programs are called applets. Applets can be used for delivering interactive graphical content over the internet, and can be made available to students and instructors throughout the world. This approach overcomes many, if not all, of the limitations of other computer and web based educational approaches and has significant advantages. The documents and program developed are stored as separate files on a server, thus making it easy to modify or upgrade them with ease. One further issue is that unlike some other multimedia tools that offer only limited capability to program interactive applications and computations, Java can be used to program complex applications.

Another reason for the popularity of Java is its security. When Java applets are run on a Java compatible web browser, there is little risk of infection by a virus program. Java achieves this protection by confining a Java program to the Java execution environment and not allowing it access to other parts of the computer. The ability to download applets with the confidence that no harm will be done and no security will be breached is an important feature for developing web based instructional applets.

Many successful attempts have been previously made to use Java Applets for web based teaching of structural analysis. A series of web-based instructional units for teaching structural mechanics to undergraduate students was developed by Rojiani et al. (2000). Instructional units developed included applets for shear, moment, and deflection of beams, computation of section properties of sections built up from standard geometric shapes, analysis of statically determinate and indeterminate trusses and frames, and shear center for open and closed section thin-walled tubes (Rojiani et al. 2000). These applets make extensive use of graphics for depicting structural behavior and allow for dynamic interaction with users.

Benjamin et al. (2003) discussed the use of Java to develop interactive instructive courseware. They developed a series of applets for demonstrating the assembly of the global stiffness matrix of a structure. These applets feature online help and interactive feedback along with an appealing graphical user interface.

The use of Java applets for design and analysis of L shaped beams was presented by Karthik et al. (2000). Here the authors used Java and VRML (Virtual Reality Modeling Language) to develop a finite element applet. After submitting the required input, like material type and loads acting on beam, the Java program meshes the beam automatically and calculates 3D stresses in the beam at various locations. The results obtained are color coded and displayed. This applet illustrates the potential for using Java in virtual design and 3D Visualization.

Nikishkov (2003) developed a finite element applet in Java for analyzing plane cracks in three-dimensional components. The Java library graphics classes were used to display color contours of stresses in objects modeled with four-node elements. He further demonstrated the ability of Java to plot stress contours with a sparse mesh of finite elements. Lu et al. (2001) demonstrated the application of Virtual Reality to finite

element analysis using the Java graphical library. According to him, the use of graphics and interactive Java programs gives engineering students an intuitive understanding of the finite element analysis technique.

Paul et al. (2004) recommend the use of simplified finite element analysis for undergraduate students to understand the basic fundamentals of the method. Since commercial finite element programs are excessively complex, Paul implemented a simple finite element analysis program for helping students understand the fundamentals of finite element analysis. The response from students to the software was quite encouraging.

Chapter 3: One-Dimensional Bar and Truss Elements

3.1 Introduction

Two types of elements are presented in this chapter: (1) One-dimensional Bar Element, and (2) Truss Element. The development of the stiffness matrix for these elements, and the computation of displacements, stresses and forces are presented. The implementation details of these elements in Java are also discussed. Each element is represented by a separate class. Details of these element classes are presented.

The one-dimensional bar element and the truss element are the simplest elements that can be used to explain the fundamentals of the finite element method. These elements are used to analyze structures which are subjected to pure axial forces (i.e., no bending or shear). The one-dimensional bar element is used to analyze a structure having degrees of freedom only in one direction, whereas the truss element is used for the analysis of determinate and indeterminate trusses with two degrees of freedom at each node.

An important difference between the one-dimensional bar and the truss element is the orientation of the element in relation to the structure. The longitudinal axis of the one-dimensional bar element is parallel to the corresponding axis of the structure, whereas the axis of the truss element can have any orientation in the plane of structure. Since all elemental axes are aligned in the same direction, no coordinate transformation is required for one-dimensional bar elements. The truss element can have any orientation in its plane, thus transformation matrices are required to convert displacements and stresses in the local coordinate system to the global coordinate system.

3.2 One-dimensional Bar Element

This is the easiest element to begin with for teaching the finite element method. This element has one degree of freedom at each node for a total of two degrees of

freedom per element. The stress, strain, displacements, and loading depend only on one variable, either x or y. One-dimensional bar elements are used to model shafts subjected to axial force.

The stress-strain and strain-displacement relations are

$$\sigma = E\varepsilon \tag{3.1}$$

$$\varepsilon = \frac{du}{dx} \tag{3.2}$$

where σ = stress, ε = strain, u = displacement and E = modulus of elasticity. The procedure for developing the stiffness matrix is as follows.

Consider a uniform prismatic bar element as shown in Figure 3.1

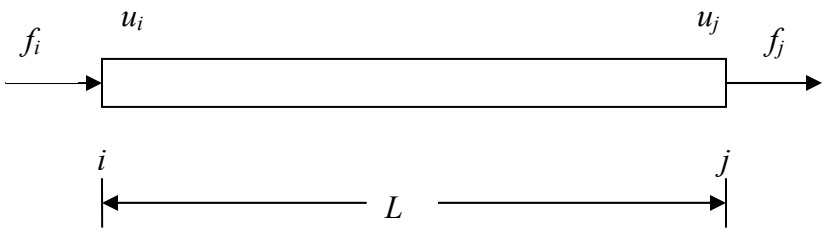

Figure 3.1 Bar element.

The two nodes are labeled i and j. The nodal displacements are u_i and u_j, and the corresponding nodal forces are f_i and f_j.

The strain is given by

$$\varepsilon = \frac{du}{dx} = \frac{u_j - u_i}{L} = \begin{bmatrix} -\frac{1}{L} & \frac{1}{L} \end{bmatrix} \begin{Bmatrix} u_i \\ u_j \end{Bmatrix} \tag{3.3}$$

where L is the length of the bar.

The stress is

$$\sigma = E\varepsilon \qquad = E\frac{\Delta u}{L} \tag{3.4}$$

where $\Delta u = (u_j - u_i)$ is the change in length of the bar.

The force F in the bar is given by

$$F = \sigma A \tag{3.5}$$

where A is the cross sectional area of the bar.

Substituting Equation (3.4) in (3.5) gives

$$F = \frac{EA\Delta u}{L} = k\Delta u \tag{3.6}$$

where $k = \frac{EA}{L}$ is the stiffness of the bar.

Also,

$$f_j = k(\Delta u) = k(u_j - u_i) \tag{3.7}$$

and

$$f_i + f_j = 0 \tag{3.8}$$

Therefore, from equilibrium of the bar, $f_i = -f_j$

(3.9)

The relationship between the forces at the ends of the bar and the nodal displacements is given by

$$\begin{Bmatrix} f_i \\ f_j \end{Bmatrix} = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \begin{Bmatrix} u_i \\ u_j \end{Bmatrix} \tag{3.10}$$

and the stress in the bar element can be obtained from

$$\sigma = E\begin{bmatrix} \frac{-1}{L} & \frac{1}{L} \end{bmatrix} \begin{Bmatrix} u_i \\ u_j \end{Bmatrix} \tag{3.11}$$

3.3 Truss Element

The one-dimensional bar element is constrained to deform only along the axis of the element. However, in the case of a plane truss, the nodes can move in both the x and y directions. Hence there are two degrees of freedom per node.

The development of the stiffness matrix of a truss element requires a transformation between the global and local coordinate systems. The truss element is

shown in Figure 3.2, along with the global and local coordinate systems which are defined as follows

Local coordinate system: This is denoted by the x and y axes. All element properties such as element forces, stresses, and distributed element loads are defined in the local coordinate system. Each element in a structure has its own set of local coordinates.

Global coordinate system: The global coordinate system for the structure is fixed and is represented by X and Y axes. Quantities such as nodal loads and nodal displacements are defined in the global coordinate system.

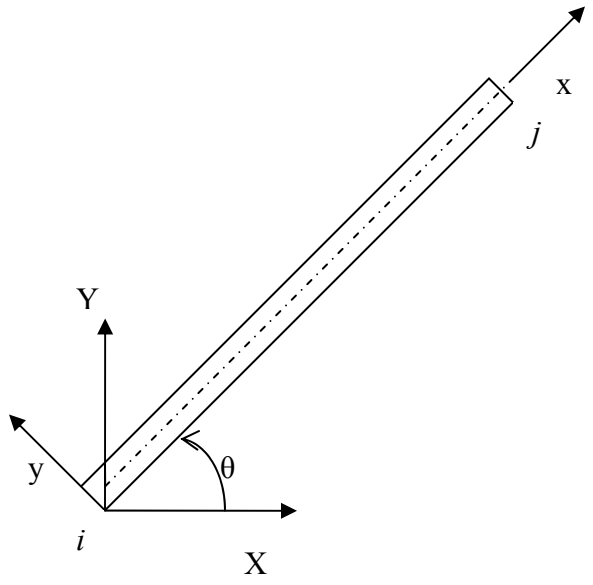

Figure 3.2 Local and global coordinates.

The nodal displacements in the local coordinate system are shown in Figure 3.3. The corresponding nodal displacements in the global coordinate system are shown in Figure 3.4.

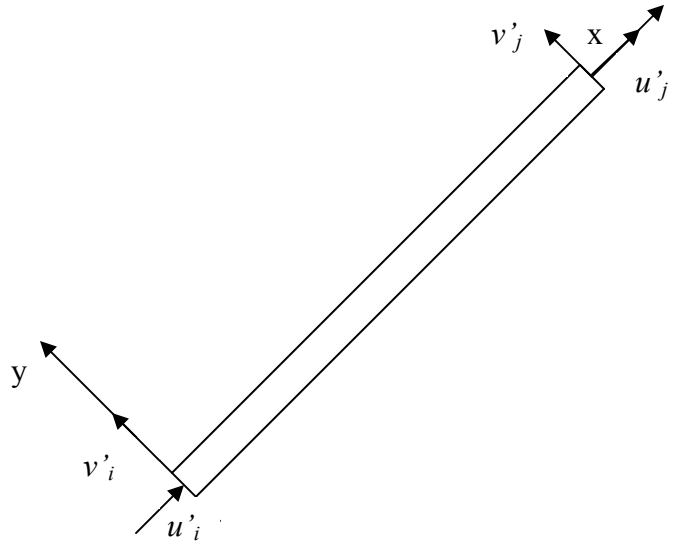

Figure 3.3 Nodal displacements in local coordinates.

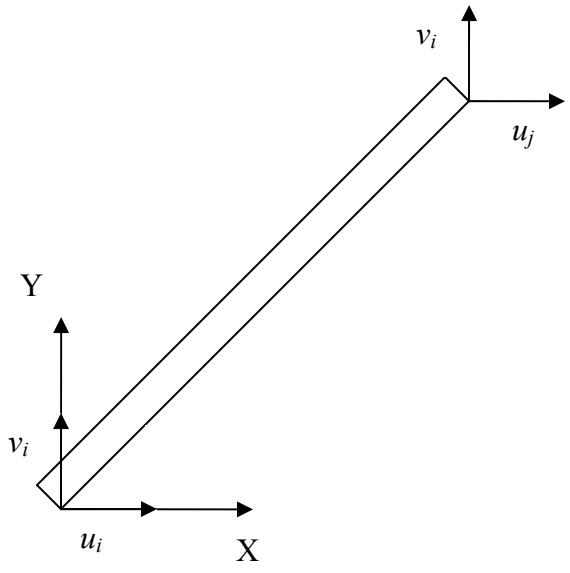

Figure 3.4 Nodal displacements in global coordinates.

The local coordinates $\{x\} = \{x, y\}^T$ and global coordinates $\{X\} = \{X, Y\}^T$ are related by the following transformation equations:

$$x = X\cos\theta + Y\sin\theta \tag{3.12}$$

$$y = -X\sin\theta + Y\cos\theta \tag{3.13}$$

This can be represented in matrix form as

$$\{x\} = [t]\{X\} \tag{3.14}$$

$$\{x\} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{Bmatrix} X \\ Y \end{Bmatrix} \tag{3.15}$$

The matrix [t] is called the transformation matrix. The same transformation matrix [t] can be used to map nodal displacements in the global coordinate system to nodal displacements in the local coordinate system:

$$\begin{Bmatrix} u' \\ v' \end{Bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{Bmatrix} u \\ v \end{Bmatrix} \tag{3.16}$$

The transformation from the global to local system is obtained by taking the inverse of [t]. Thus

$$\{X\} = [t]^{-1}\{x\} \tag{3.17}$$

It can be shown that the inverse of [t] is equal to its transform, $[t]^T$, that is.

$$[t]^{-1} = [t]^T = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{3.18}$$

A matrix whose inverse is equal to its transpose is called an orthogonal matrix.

The nodal displacements at the two nodes *i* and *j* of the two node truss element are related to the corresponding global displacements by the transformation

$$\begin{Bmatrix} u'_i \\ v'_i \\ u'_j \\ v'_j \end{Bmatrix} = \begin{bmatrix} c & s & 0 & 0 \\ -s & c & 0 & 0 \\ 0 & 0 & c & s \\ 0 & 0 & -s & c \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \end{Bmatrix} \tag{3.19}$$

where c=$\cos\theta$ and $s=\sin\theta$ represent the directional cosines. Equation (3.19) can be written as

$$\{ u' \} = [\mathrm{T}]\{u\} \tag{3.20}$$

The relationship between local and global nodal forces can be written in a similar manner:

$$\{ f' \} = [\mathrm{T}]\{f\} \tag{3.21}$$

The relationship between stiffness, the nodal displacements, and nodal forces expressed in local coordinates is

$$\frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{Bmatrix} u'_i \\ u'_j \end{Bmatrix} = \begin{Bmatrix} f_i' \\ f_j' \end{Bmatrix} \tag{3.22}$$

Augmenting the stiffness matrix to include displacements in the Y direction gives

$$\frac{EA}{L} \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} u'_i \\ v'_i \\ u'_j \\ v'_j \end{Bmatrix} = \begin{Bmatrix} f'_i \\ 0 \\ f'_j \\ 0 \end{Bmatrix} \tag{3.23}$$

The above equation can be written as

$$[k']\{u'\} = \{ f' \} \tag{3.24}$$

where k' is the stiffness matrix of the truss element in the local coordinate system and is given by

$$k' = \frac{EA}{L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{3.25}$$

To obtain the stiffness matrix in global coordinates, we use the transformations in equations (3.20).

Substituting for u' from Equation (3.20) in Equation (3.24) gives

$$[k'][\mathrm{T}]\{u\} = [\mathrm{T}]\{f\} \tag{3.26}$$

Multiplying both sides of Equation (3.26) by $[\mathrm{T}]^{\mathrm{T}}$,

$$[\mathrm{T}]^{\mathrm{T}}[k'][\mathrm{T}]\{u\} = [\mathrm{T}]^{\mathrm{T}}[\mathrm{T}]\{f\} \tag{3.27}$$

$$[\mathrm{T}]^{\mathrm{T}}[\mathrm{T}] = [\mathrm{T}]^{-1}[\mathrm{T}] = \mathrm{I} \tag{3.28}$$

where I is identity matrix

Therefore Equation (3.27) can be expressed as

$$[\mathrm{K}]\{u\} = \{f\} \tag{3.29}$$

where

$$[\mathrm{K}] = [\mathrm{T}]^{\mathrm{T}} k' \ [\mathrm{T}] \tag{3.30}$$

is the global stiffness matrix of truss element.

The global stiffness matrix can be written as

$$[\mathrm{K}] = \frac{EA}{L}\begin{bmatrix} c^2 & cs & -c^2 & -cs \\ cs & s^2 & -cs & -s^2 \\ -c^2 & -cs & c^2 & cs \\ -cs & -s^2 & cs & s^2 \end{bmatrix} \tag{3.31}$$

where c and s are directional cosines. These can be obtained from the global coordinates of the element as

$$c = cos\theta = \frac{x_j - x_i}{L} \tag{3.32}$$

$$s = sin\theta = \frac{y_j - y_i}{L} \tag{3.33}$$

where L is the length of the element and is given by

$$L = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \tag{3.34}$$

The global stiffness matrix of the structure is obtained by adding the global stiffness matrices of the individual elements:

$$\mathrm{K} = \sum_{i=1}^{N} K_i \tag{3.35}$$

where n is the number of truss elements in the structure. Once the global stiffness matrix is formed, the system is solved for nodal displacements in the global coordinates. These nodal displacements are the used to evaluate stresses and forces in each element.

The element stresses are obtained from

$$\sigma = E\varepsilon = \mathrm{E}\,\frac{u'_j - u'_i}{L} = \mathrm{E}\left[\frac{-1}{L}\;\frac{1}{L}\right]\begin{Bmatrix} u_i' \\ u_j' \end{Bmatrix} \tag{3.36}$$

Thus,

$$\sigma = \mathrm{E}\left[\frac{-1}{L}\;\frac{1}{L}\right]\begin{bmatrix} c & s & 0 & 0 \\ 0 & 0 & c & s \end{bmatrix}\begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \end{Bmatrix} \tag{3.37}$$

which gives

$$\sigma = \frac{E}{L}\begin{bmatrix} -c & -s & 0 & 0 \end{bmatrix}\begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \end{Bmatrix} \tag{3.38}$$

The element forces are evaluated from

$$\begin{Bmatrix} f_i' \\ f_j' \end{Bmatrix} = \frac{EA}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{Bmatrix} u'_i \\ u'_j \end{Bmatrix} \tag{3.39}$$

3.4 Implementation of the One-dimensional Bar Element in Java

The one-dimensional bar element is implemented in Java using a class called `OneDBarelement`.

The interface for the `OneDBarelement` class is given below:

```
public class OneDBarelement {
private double area,length;      //area and length
private double[] coodx;          //X coordinates of each element
private double[] coody;          //Y coordinates of each element
private double ym;               //Young's modulus
public double[][] onedkelem;     //element stiffness matrix
public double[] onedstress;      //stress values
public double[] onedforce;       //force values
```

The variables `area` and `length` represent the cross-sectional area and length of the element. The arrays `coodx[]` and `coody[]` are used for storing the nodal coordinates of the element. The variable `ym` represent the modulus of elasticity of the bar element. The stiffness matrix [k] is stored in the two-dimensional array `onedkelem` . Stresses and forces are stored in two one-dimensional arrays `onedstress` and `onedforce`, respectively.

The constructor of the class `OneDBarelement` is

```
public OneDBarelement(double[] x, double[] y, double moe,
                 double th) {
     ym = moe;
     area = th;
     coodx = x;
     coody = y;
 }
```

Whenever an instance of the `OneDBarelement` class is created, the properties of the element like Young's modulus, area and coordinates are passed as arguments to the constructor. These variables are later used for further operations like calculation of stiffness matrix, forces, and stresses.

The methods in class `OneDBarelement` are listed in Table 3.1.

Table 3.1 Methods in class `OneDBarelement`

Method	Description
`calcelemk()`	Calculates the stiffness matrix of bar element
`calcstress()`	Calculates the stresses and forces in the element

The element stiffness matrix is calculated by calling the function `calcelemk()`. This function needs no arguments as the variables required for calculating the stiffness matrix are initialized when the class is instantiated.

To analyze a structure consisting of one-dimensional bar elements, an instance of the `OneDBarelement` class is created for each element in the structure by calling the class constructor. The function `calcelemk()` is invoked for each element to get the stiffness matrix for the element. The stiffness matrix so obtained is used to assemble the global stiffness matrix. The functions for calculating displacements are defined in a separate class called `Analyze`. The `Analyze` class assembles the structure stiffness matrix and solves the equation to obtain nodal displacements. Details of the `Analyze` class are given in Chapter 7.

Once the displacements are calculated, the stress and forces in the bar element are evaluated by calling the function `calcstress()`. The method `calcstress()` is declared as

```
public void calcstress(double[] u,int[] elemcon,Stresses stodbe)
```

To calculate the stress in each bar element, we need the nodal displacements and connectivity information. The stiffness and other details needed for calculating stresses are obtained from the `OneDBarelement` object. The computed forces and stresses are stored in separate variables. These variables are used while displaying the results.

3.5 Implementation of Truss Element in Java Applet

The implementation of the truss element in Java is similar to that of the one-dimensional bar element with some minor changes. The class developed for representing a truss element is the `TwoDTrusselement` class.

The interface for the `TwoDTrusselement` class is:

```
public class TwoDTrusselement {
      private double area,length;      //area & length
      private double[] coodx;          //X  Coordinates  of  each
                                       element
```

```
private double[] coody;          //Y Coordinates of each
                                 element
private double ym;                //Youngs Modulus
public double[][] twodtrusskelem;//element stiffness matrix
private double[][]twodkelem;     //  Stiffness matrix in
                                 local coordinates
public double[] twodstress;       //stresses in element
public double[] twodforce;       //forces in element
```

The variables `area`, `length`, `ym` and arrays `coodx`, `coody` are similar to those used in the `OneDBarelement` class. The stiffness matrix, stresses, and forces of an element are stored in the arrays `twodtrusskelem`, `twodstress`, and `twodforce`, respectively.

The constructor of this class is

```
public TwoDTrusselement(double[] x, double[] y, double moe,
                    double th){
       ym = moe;
       area = th;
       coodx = x;
       coody = y;
      }
```

The properties of the element such as Young's modulus, area, and coordinates are passed as arguments to the constructor when an instance of the `TwoDTrusselement` class is created.

The methods used in class `TwoDTrusselement` are listed in Table 3.2.

Table 3.2 Methods in class `TwoDTrusselement`

Method	Description
`calctdelemk()`	Calculates stiffness matrix for a truss element
`calcstress()`	Calculates stresses and forces in a truss element

The function `calctdelemk()` calculates the stiffness matrix of the truss element. The transformation matrices are needed in order to calculate the stiffness matrix. These are calculated using the data members of the `TwoDTrusselement` object. For each truss element, an instance of the `TwoDTrusselement` is created.

The `Analyze` class assembles the global stiffness matrix of the structure by adding the stiffness matrix of each truss element to the corresponding nodes of the structural stiffness matrix and solves the system of equations. The displacements obtained from the `calcdisp()` method of `Analyze` class are used to calculate element stresses and forces by calling the method `calcstress()`, which is declared as

```
public void calcstress(double[] u,int[] elemcon,Stresses sttdte)
```

The stresses in each truss element are stored in the object `sttdte` which is an instance of `Stresses` class. The deformed shape of the truss is then plotted by calling an instance of the `Deform` class.

Chapter 4: Two-Dimensional Beam and Frame Element

4.1 Introduction

In this chapter the development of the two-dimensional beam element and the frame element is presented. The degrees of freedom considered for the beam element are rotation about an axis perpendicular to the plane of the beam and translations perpendicular to the plane of the beam. For the frame element, which is an extension of the beam element, axial degrees of freedom are included. Details of the implementation of the two-dimensional beam element and the frame element in Java are also presented.

Beams are members that are used for supporting transverse loading. These elements are used in a variety of structures simulating buildings, bridges, and scaffoldings. The Two-dimensional Beam element has two degrees of freedom at each node: a rotation about an axis perpendicular to the plane of beam and a translation perpendicular to the axis of the beam. Axial deformations in the beam are neglected. The frame element has axial deformation at each node in addition to the beam deformations. Hence, the frame element has three degrees of freedom at each node. The beam element provides flexural rigidity, a property that the truss element does not have. Since there is no coupling between the axial and flexural deformations, the stiffness matrix of the beam is first derived and it is then extended to the frame element by incorporating the axial stiffness.

Unlike plane trusses, which are subjected only to nodal loads, the external loads on beams may be applied both on the members as well on the nodes. Member loads are converted into equivalent nodal loads by using the concept of fixed end forces.

4.2 Two-dimensional Beam Element

The procedure to develop the stiffness matrix of a beam element is the same as that of truss elements, but in addition to axial rigidity, flexural rigidity is considered.

Consider a prismatic beam element as shown in Figure 4.1

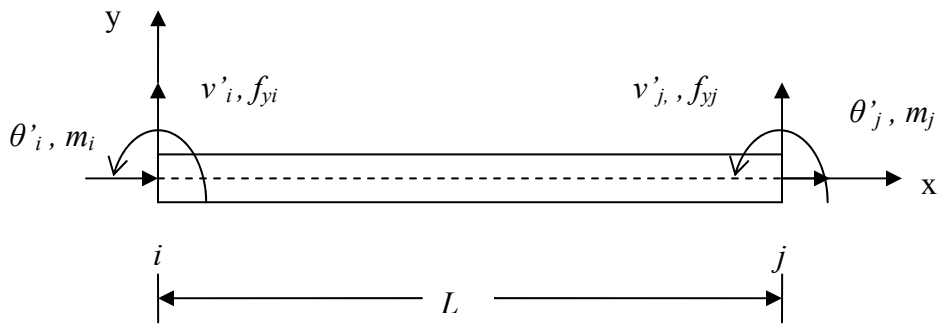

Figure 4.1 Two-dimensional beam element.

The two nodes are labeled *i* and *j*. Here v'_i, and v_j' are the vertical displacements at node i and j, and θ'_i and θ'_j. are the rotations at nodes i and j. Counter clockwise rotation is assumed to be positive. Each node is associated with a pair of shear and bending forces. The corresponding nodal forces are represented by f_{yi}, m_i at the i end and f_{yj}, m_j at the j end, respectively. The properties of the beam are:
L = length, I = moment of inertia, and E = Young's modulus

4.3 Euler-Bernoulli Beam Theory

Beam elements are based on either the Euler-Bernoulli or Mindlin's theory. Here the beam element is formulated using the Euler-Bernoulli theory and shear deformations in the beam element are neglected. The lateral displacement v can be related to the rotation θ as

$$\theta = \frac{dv}{dx} \tag{4.1}$$

Also, the bending moment and shear forces in the beam can be related to v as

$$EI\frac{d^2v}{dx^2} = M \tag{4.2}$$

$$EI\frac{d^3v}{dx^3} = V \tag{4.3}$$

where M and V are the moment and shear acting at a particular section

The bending stress is given by

$$\sigma = -\frac{My}{I} \tag{4.4}$$

4.4 Interpolation Functions

The displacements at any location in the element are expressed in terms of nodal displacements using an interpolation function. The lateral displacement of the element at any point x is represented as

$$v(x) = \alpha_o + \alpha_1 x + \alpha_2 x^2 + \alpha_3 x^3 \tag{4.5}$$

where the α_i's are undetermined constants.

Equation (4.5) can be expressed in matrix form as

$$v(x) = \begin{bmatrix} 1 & x & x^2 & x^3 \end{bmatrix} \begin{Bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{Bmatrix} \tag{4.6}$$

Using Equations (4.1) and (4.5), the rotation at any location can be expressed as

$$\theta(x) = \alpha_1 + 2\alpha_2 x + 3\alpha_3 x^2 \tag{4.7}$$

Hence, at any location x, the lateral displacement and rotation are given by

$$\begin{Bmatrix} v \\ \theta \end{Bmatrix} = \begin{bmatrix} 1 & x & x^2 & x^3 \\ 0 & 1 & 2x & 3x^2 \end{bmatrix} \begin{Bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{Bmatrix} \tag{4.8}$$

At the ends x=0 and x=*L,* the nodal displacements and rotations are

$$\begin{Bmatrix} v_i \\ \theta_i \\ v_j \\ \theta_j \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & L & L^2 & L^3 \\ 0 & 1 & 2L & 3L^2 \end{bmatrix} \begin{Bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{Bmatrix} \tag{4.9}$$

Equation (4.9) can be written as

$$\{u\} = [A]\{\alpha\} \tag{4.10}$$

where

$$\{u\} = \begin{Bmatrix} v_i \\ \theta_i \\ v_j \\ \theta_j \end{Bmatrix} \tag{4.11}$$

Thus,

$$\{\alpha\} = [A]^{-1}\{u\} \tag{4.12}$$

where $[A]^{-1}$ is given by

$$[A]^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{-3}{L^2} & \frac{-2}{L} & \frac{3}{L^2} & \frac{-1}{L} \\ \frac{2}{L^3} & \frac{1}{L^2} & \frac{-2}{L^3} & \frac{1}{L^2} \end{bmatrix} \tag{4.13}$$

Substituting Equation (4.12) in Equation (4.1),

$$\text{v(x)} = \begin{bmatrix} 1 & x & x^2 & x^3 \end{bmatrix} [\text{A}]^{-1}\{u\} \tag{4.14}$$

or

$$\text{v(x)} = [\text{N}]\ \{u\} \tag{4.15}$$

Equation (4.15) can be written as

$$\text{v(x)} = \begin{bmatrix} N_1(x) & N_2(x) & N_3(x) & N_4(x) \end{bmatrix} \begin{Bmatrix} v_i \\ \theta_i \\ v_j \\ \theta_j \end{Bmatrix} \tag{4.16}$$

where

$$N_1(x) = 1 - \frac{3x^2}{L^2} + \frac{2x^3}{L^3}$$

$$N_2(x) = x - \frac{2x^2}{L} + \frac{x^3}{L^2}$$

$$N_3(x) = \frac{3x^2}{L^2} - \frac{2x^3}{L^3}$$

$$N_4(x) = -\frac{x^2}{L} + \frac{x^3}{L^2} \tag{4.17}$$

are the shape functions $N_i(x)$ and are called Hermitian polynomials since they contain both the function itself (displacement) and its derivatives (rotation).

4.5 Element Stiffness Matrix

The procedure for developing the stiffness matrix is as follows

The internal virtual work for a beam is

$$\delta W_{int} = -\int M \delta k dx \tag{4.18}$$

where k is the curvature and M is the bending moment. The curvature k and the bending moment M are given by

$$k = \frac{d^2v}{dx^2} \tag{4.19}$$

$$M = EI\frac{d^2v}{dx^2} \tag{4.20}$$

The curvature is obtained as

$$\frac{d^2v}{dx^2} = \frac{d}{dx^2}[N]\{u\} = [B]\{u\} \tag{4.21}$$

where $[B]$ is the strain-displacement matrix and is

$$[B] = \left[\left(-\frac{6}{L^2} + \frac{12x}{L^3}\right) \quad \left(-\frac{4}{L} + \frac{6x}{L^2}\right) \quad \left(\frac{6}{L^2} - \frac{12x}{L^3}\right) \quad \left(\frac{-2}{L^2} + \frac{6x}{L^2}\right)\right] \tag{4.22}$$

The virtual curvature δk is

$$\{\delta k\} = [B]\{\delta u\} \tag{4.23}$$

and the internal virtual work is

$$\delta W_{int} = -\int_0^L \left([B]\{\delta u\}\right)\left(EI[B]\{u\}\right)dx \tag{4.24}$$

Since $[B]\{\delta u\}$ is a scalar, and it can be rewritten as $\{\delta u\}^T[B]^T$

Hence,

$$\delta \mathrm{W}_{\mathrm{int}} = -\int_0^L \{du\}^T[B]^T\left(EI\right)[B]\{u\}dx \tag{4.25}$$

$$\delta \mathrm{W}_{\mathrm{int}} = -\{du\}^T\left(\int_0^L [B]^T\left(EI\right)[B]dx\right)\{u\} \tag{4.26}$$

since $\{u\}$ and $\{\delta u\}$ are independent of x. Therefore, the stiffness matrix of the two-dimensional beam element is

$$\left[k'\right] = \int_0^L [B]^T(EI)[B]dx \tag{4.27}$$

After performing matrix operations and integrating each term, we obtain

$$[K] = \frac{EI}{L^3}\begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix} \tag{4.28}$$

Further, the element equations can be written as

$$\frac{EI}{L^3}\begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix}\begin{Bmatrix} v_i \\ \theta_i \\ v_j \\ \theta_j \end{Bmatrix} = \begin{Bmatrix} fy_i \\ m_i \\ fy_j \\ m_j \end{Bmatrix} \tag{4.29}$$

4.6 Equivalent Nodal Loads

As mentioned previously, the beam element can be subjected to member loads in addition to nodal loads. Member loads are converted to equivalent nodal loads. The equivalent nodal loads are given by

$$\{f\} = \int_0^L [N]^T q(x) dx \tag{4.30}$$

where $q(x)$ is a distributed load. For a beam element subjected to a uniformly distributed load of q, the equivalent nodal force vector can be expressed as

$$\{f\} = \begin{Bmatrix} -\frac{qL}{2} \\ -\frac{qL^2}{12} \\ \frac{-qL}{2} \\ \frac{qL^2}{12} \end{Bmatrix} \tag{4.31}$$

4.7 Stiffness Matrix for the Two-Dimensional Frame Element

Consider the frame element shown in Figure 4.2

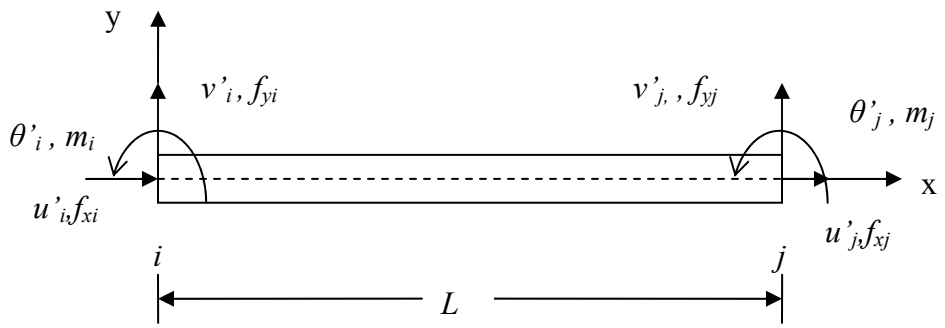

Figure 4.2 Frame element.

The stiffness matrix of the frame element is obtained by adding the axial stiffness of the element at the respective degrees of freedom by expanding the beam stiffness matrix. The stiffness matrix of the frame element is given by

$$k' = \begin{bmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{4EI}{L} \end{bmatrix} \tag{4.32}$$

The stiffness matrix presented in Equation (4.32) is in terms of the local coordinate system of the frame element. The relationship between the local element displacements and global displacements are identical to those of the truss element. Thus, the transformation matrix for a frame element is given by

$$T = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\theta & \sin\theta & 0 \\ 0 & 0 & 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.33}$$

where $cos\theta$ and $sin\theta$ are the direction cosines and are obtained from the nodal coordinates of the element,

$$cos\theta = \frac{x_j - x_i}{L} \quad \text{and} \quad sin\theta = \frac{y_j - y_i}{L} \tag{4.34}$$

and $L = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$

is the length of the element

The global stiffness matrix for the frame element is given by

$$[\mathrm{K}] = [\mathrm{T}]^{\mathrm{T}} [k'] \; [\mathrm{T}] \tag{4.35}$$

The global stiffness matrix of the structure is obtained by adding the individual global stiffness matrices of the elements:

$$K = \sum_{i=1}^{n} K_i \tag{4.36}$$

where n is the number of truss elements in the structure. Once the global stiffness matrix is formed, the system is solved for the nodal displacements in global coordinates. These nodal displacements are then used to evaluate stresses and forces in each element.

4.8 Implementation of Two-dimensional Beam Element in Java

To implement the beam element in Java, a class called `TwoDBeam` is developed. The interface for the class `TwoDBeam` is:

```
public class TwoDBeam {
    private double mi,length;    //Moment of intertia and length
     private double[] coodx;     //X Coordinates for each node
     private double[] coody;     //Y Coordinates for each node
     private double ym;                          //Young's Modulus
     public double[][] twodbeamkelem;           //element
                                                stiffness    matrix
     public double[] twodbforce,twodbeamstress;//stresses
```

The variables `mi` and `length` represent the moment of inertia and the length of the beam element, respectively. The arrays `coodx[]` , `coody[]` represent nodal coordinates. The modulus of elasticity is stored in the variable `ym`. The stiffness matrix of the beam element is stored in the two-dimensional array `twodbeamkelem`. Finally, the arrays `twodbforce` and `twodbeamstress` are used to store the calculated forces and stresses in the beam element.

The constructor for the `TwoDBeam` is given by

```
public TwoDBeam(double[] x, double[] y, double moe, double nu
                  double th){
```

```
        ym = moe;
        mi = nu;
        ar = th;
        coodx = x;
        coody = y;
    }
```

To create an instance of the `TwoDBeam` object, the element coordinates, section, and material properties are passed through the constructor.

The methods in the `TwoDBeam` class calculate the stiffness matrix of the element and evaluate the stresses and forces in the element. The methods available in the class `TwoDBeam` are listed in Table 4.1

Table 4.1 Methods in class `TwoDBeam`

Method	**Description**
`calcelemk()`	Calculates the element stiffness matrix
`calcstress()`	Calculates the stresses and forces in element
`Calcnodalloads()`	Calculates nodal loads for non nodal loading

The method `calcelemk()`calculates the stiffness matrix of a two-dimensional beam element using the variables obtained by instantiation of the `TwoDBeam` object. There is one instance of a class for each beam element `TwoDBeam` in the structure. To analyze a structure with beam elements, the beam element object is initialized in the `Analyze` class as shown in the statement below:

```
TwoDBeam tdb = new TwoDBeam(x,y,moe,nu,th);
```

The methods in the class `TwoDBeam` are then used to calculate the stiffness matrix and stresses for each beam element.

The method `calcnodalload()`converts the member loads acting on the beam to equivalent nodal loads. The method `calcstress()`evaluates the stresses, nodal forces, and reactions at the restraints. The header for this method is

```
public void calcstress(double[] u,int[] elemcon,Stresses stbe)
```

The stresses are written to an object `stdbe.` The method `calcstress()` also calculates the reactions at the restrained nodes of the structure.

Chapter 5: Plane Stress Elements

5.1 Introduction

Three types of membrane elements are developed for this applet: (1) constant strain triangle, (2) four-node iso-parametric quadrilateral element, and (3) eight-node iso-parametric quadrilateral element. In this chapter the development of the stiffness matrices of these elements along with their implementation in Java is discussed.

5.2 Two-dimensional Stresses and Strains

Two-dimensional problems are sometimes modeled as plane stress or plane strain. These problems generally involve structures whose thickness is small compared to the other two dimensions. When the structure is subjected to inplane forces, the displacements at any point {x, y} in the element are given by {u, v} where, u and v are displacements in the x and y directions, respectively:

$$\mathrm{U} = \begin{Bmatrix} u \\ v \end{Bmatrix} \tag{5.1}$$

The stresses are strains are given by

$$\sigma = \{\sigma_x, \sigma_y, \tau_{xy}\}^T$$
$$\varepsilon = \{\varepsilon_x, \varepsilon_y, \gamma_{xy}\}^T \tag{5.2}$$

5.2.1 Plane Stress

Plane stress is a condition that prevails in a flat plate in the *xy* plane, loaded only in its own plane and without any restraints in the z-direction, so that $\sigma_z = 0$, $\tau_{yz} = 0$, and

$\tau_{zx} = 0$ (Cook 1989). For an isotropic material, the stress-strain relationship for plane stress is given by

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = [D] \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} \tag{5.3}$$

where [*D*] is the material matrix and is expressed as

$$[D] = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \dfrac{1-\nu}{2} \end{bmatrix} \tag{5.4}$$

E = Young's modulus of elasticity, and *v* = Poisson's ratio.

5.2.2 Plane Strain

If a long body of uniform cross section is subjected to transverse loading along its length, a small thickness in the loaded area can be treated as being subjected to plane strain (Chandrupatla et al. 1997). In this case, $\varepsilon_z = 0$, $\gamma_{yz} = 0$, and $\gamma_{zx} = 0$. The material matrix [*D*] for the plane strain condition for an isotropic material is given by

$$[D] = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & 0 \\ \nu & 1-\nu & 0 \\ 0 & 0 & \dfrac{1-2\nu}{2} \end{bmatrix} \tag{5.5}$$

5.3 Constant Strain Triangle

This is the simplest element for analyzing two-dimensional problems. It has three nodes, and each node has two inplane degrees of freedom, making a total of six degrees of freedom. The triangular element is a widely used element. Triangular elements are used for structures with irregular boundary conditions, unlike rectangular elements. The CST element is shown in Figure 5.1. The nodes are numbered in a counter-clockwise direction for consistency.

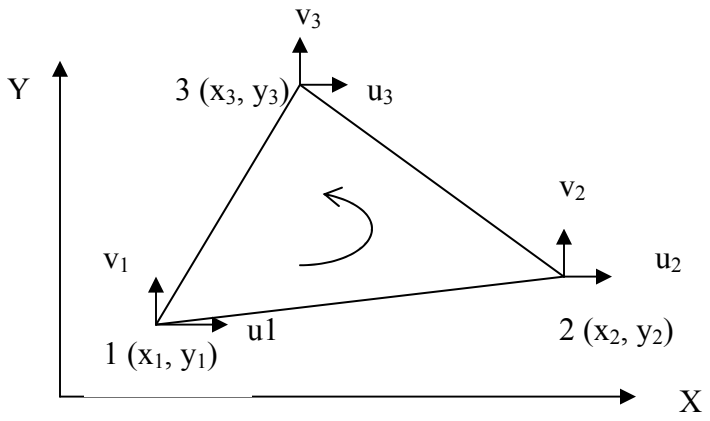

Figure 5.1 Constant Strain Triangle.

The procedure for developing the stiffness matrix of the CST element is as follows.
The displacements $u(x, y)$ and $v(x, y)$ are assumed to vary linearly with x and y, and hence the displacement function can be represented by

$$u(x, y) = \alpha_o + \alpha_1 x + \alpha_2 y$$

$$v(x, y) = \alpha_3 + \alpha_4 x + \alpha_5 y \qquad (5.6)$$

where $u(x, y)$ and $v(x, y)$ represent displacements in the x and y directions, respectively. The above equations are rewritten in matrix form as

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} 1 & x & y & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x & y \end{bmatrix} \begin{Bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{Bmatrix} \tag{5.7}$$

or

$$u(x, y) = [X]\ \{\alpha\} \tag{5.8}$$

Given the displacements, the strains are computed as

$$\begin{aligned} \varepsilon_x(x, y) &= \frac{\partial u}{\partial x} = \alpha_1 \\ \varepsilon_y(x, y) &= \frac{\partial u}{\partial y} = \alpha_5 \\ \gamma_{xy}(x, y) &= \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = \alpha_2 + \alpha_4 \end{aligned} \tag{5.9}$$

It can be observed that the strains are not functions of (x, y), hence the name constant strain triangle.

Substitution of nodal coordinates in Equation (5.7) results in

$$\begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_1 & y_1 \\ 1 & x_2 & y_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_2 & y_2 \\ 1 & x_3 & y_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_3 & y_3 \end{bmatrix} \begin{Bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{Bmatrix} \tag{5.10}$$

or

$$\{u\} = [A]\{\alpha\} \tag{5.11}$$

The coefficients $\{\alpha\}$ are obtained by inverting the matrix $[A]$ and multiplying it with $\{u\}$:

$$\{\alpha\} = [A]^{-1}\{u\} \tag{5.12}$$

Substituting Equation (5.12) in Equation (5.8),

$$u(x, y) = [X]\,[A]^{-1}\{u\} \tag{5.13}$$

where $[X][A]^{-1} = [N]$ represents the shape function.

Inverting matrix [A] and multiplying with [X] results in

$$\begin{Bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{Bmatrix} = \frac{1}{2A} \begin{bmatrix} x_2y_3 - x_3y_2 & 0 & x_3y_1 - x_1y_3 & 0 & x_1y_2 - x_2y_1 & 0 \\ y_2 - y_3 & 0 & y_3 - y_1 & 0 & y_1 - y_2 & 0 \\ x_3 - x_2 & 0 & x_1 - x_3 & 0 & x_2 - x_1 & 0 \\ 0 & x_2y_3 - x_3y_2 & 0 & x_3y_1 - x_1y_3 & 0 & x_1y_2 - x_2y_1 \\ 0 & y_2 - y_3 & 0 & y_3 - y_1 & 0 & y_1 - y_2 \\ 0 & x_3 - x_2 & 0 & x_1 - x_3 & 0 & x_2 - x_1 \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} \tag{5.14}$$

where A is the area of the triangle, and is given by

$$A = \frac{1}{2}\{x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)\} \tag{5.15}$$

The shape functions are obtained by substituting Equation (5.14) in Equation (5.7).and rearranging the terms:

$$\begin{bmatrix} N_1 \\ N_2 \\ N_3 \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} (x_2y_3 - x_3y_2) + (y_2 - y_3)x + (x_3 - x_2)y \\ (x_3y_1 - x_1y_3) + (y_3 - y_1)x + (x_1 - x_3)y \\ (x_1y_2 - x_2y_1) + (y_1 - y_2)x + (x_2 - x_1)y \end{bmatrix} \tag{5.16}$$

These functions are used to interpolate the displacement at any point (*x, y*) using the nodal displacements.

Displacements can be obtained from the shape functions using the following relationship:

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} \tag{5.17}$$

or

$$\{u\} = [\mathrm{N}]\{u_i\} \tag{5.18}$$

Strains are obtained from the strain-displacement relationships

$$\begin{aligned} \varepsilon_x &= \frac{\partial u}{\partial x} = \frac{\partial}{\partial x}\sum_{i=1}^{3} N_i u_i \\ \varepsilon_y &= \frac{\partial v}{\partial y} = \frac{\partial}{\partial y}\sum_{i=1}^{3} N_i v_i \\ \gamma_{xy} &= \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = \frac{\partial}{\partial y}\sum_{i=1}^{3} N_i u_i + \frac{\partial}{\partial x}\sum_{i=1}^{3} N_i v_i \end{aligned} \tag{5.19}$$

The above equations can be expressed in matrix form as

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} \tag{5.20}$$

or

$$\{\varepsilon(\mathrm{x}, \mathrm{y})\}_{3\mathrm{x}1} = [B(\mathrm{x}, \mathrm{y})]_{3\mathrm{x}6}\{u_i\}_{6\mathrm{x}1} \tag{5.21}$$

The derivatives are evaluated with respect to x and y to obtain the strain-displacement matrix [B]:

$$[B(x,y)] = \frac{1}{2A}\begin{bmatrix} y_2 - y_3 & 0 & y_3 - y_1 & 0 & y_1 - y_2 & 0 \\ 0 & x_2 - x_3 & 0 & x_1 - x_3 & 0 & x_2 - x_1 \\ x_3 - x_2 & y_2 - y_3 & x_1 - x_3 & y_3 - y_1 & x_2 - x_1 & y_1 - y_2 \end{bmatrix} \tag{5.22}$$

The element stiffness matrix can now be obtained from

$$[k] = \int_v [B]^T [E][B] dv \tag{5.23}$$

For a constant thickness, the volume integral can be reduced to an area integral

$$[k] = t\int_A [B]^T [E][B] dA \tag{5.24}$$

where t is the thickness of the element. Since [B] is independent of (x, y), the stiffness matrix can be written as

$$[\mathrm{k}] = t[B]^T [E][B]\int_A dA \tag{5.25}$$

or

$$[\mathrm{k}] = tA[B]^T [E][B] \tag{5.26}$$

5.3.1 Implementation of CST Element in Java

To implement the CST element in Java, the `CSTelement` class is developed. The interface of this class is shown below:

```
public class CSTelement {
 private double thick;           //thickness of the element
 private double[] coodx;         //x coordinates of each node
```

```
private double[] coody;          //y coordinates of each node
private double[][] d;            //material Matrix
private double a;                //area of the element
private double[][] b;            //strain-displacement matrix (3x6)
public double[][] cstkelem;      //Element   stiffness   matrix   6x6
public double[] cststress;       //Stresses in element
```

The variables `thick` and the arrays `coodx[]`, `coody[]` represent the thickness of the element and the nodal coordinates, respectively. The two-dimensional arrays `b[]` and `cstkelem[]` contain the strain-displacement and the element stiffness matrices. The array `cststress` is used to store calculated stresses. The area of the CST element is stored in the variable `a`.

The constructor for the `CSTelement` is given as

```
public CSTelement(double[] x, double[] y, double[][] matb,
                    double th) {
         d = matb;
         thick = th;
         coodx = x;
         coody = y;

  }
```

To create an object of the CST element class, the constructor of the `CSTelement` class is called by passing the coordinates, thickness, and the material property matrix of the element as arguments. The methods in the CST element class use these variables to calculate the strain-displacement matrix, shape functions, stiffness matrix, and stresses.

The methods in the `CSTelement` class are summarized in Table 5.1.

Table 5.1 Methods in class `CSTelement`

Method	Description
`area()`	Calculates area of CST element
`cstBMatrix()`	Calculates strain-displacement matrix
`calcelemkcst()`	Calculates element stiffness matrix
`calcstress()`	Calculates stresses in element

The method `calcelemkcst()` defines the stiffness matrix of the element. To generate the stiffness matrix, we need both the area and strain-displacement matrix. These are obtained by calling the methods `area()` and `cstBMatrix()` from the `calcelemkcst()`method. The nodal coordinates required to calculate area and strain-displacement matrix are obtained from the instantiation of the object. An instance of the CST element is created by calling the constructor of the `CSTelement` class:

```
CSTelement cse= new CSTelement(x,y,matb,th);
```

where `cse` is an instance of the `CSTelement` class. The required values such as x and y coordinates and material matrix, are passed while creating the object. These values are used by the various methods to calculate the stiffness matrix, which is returned to the calling method in the `Analyze` class. The `Analyze` class further assembles the structure stiffness matrix by obtaining the element stiffness matrix for each CST element in the structure.

The method `calcstress()` mentioned above is used to calculate the stresses in the element and store them in a new `Stresses` object for further retrieval during the display of results. The `calcstress()` method is declared as

```
public void calcstress(double[] u,int[] elemcon,Stresses stsc)
```

This method uses the nodal displacements of the element obtained after analysis from the `Analyze` class for calculating the stresses. The stresses obtained are stored in a new

`Stresses` object `stsc` along with element connectivity information in order to associate them with nodal coordinates.

5.4 Four-node Iso-Parametric Quadrilateral Element

This is the simplest 2-D quadrilateral element. It has four nodes and two degrees of freedom at each node for a total of eight degrees of freedom. The four nodes are numbered in a counterclockwise direction as shown in Figure 5.2. The nodal displacement vector is given by

$$\{u\}= \{u_1, v_1, u_2, v_2, u_3, v_3, u_4, v_4\}^T \tag{5.27}$$

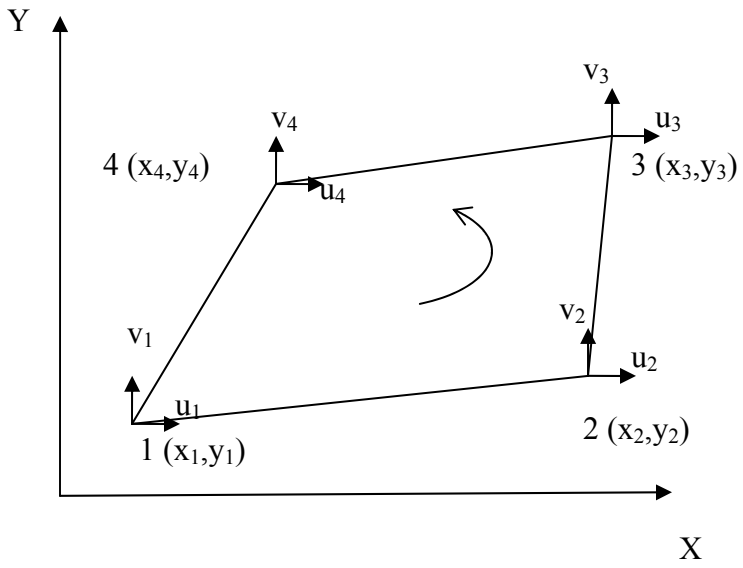

Figure 5.2 Four-node iso-parametric quadrilateral element.

The formulation of the stiffness matrix for the quadrilateral element is as described below (Cook 1989). For developing the stiffness matrix, the master (or parent) element is defined in the natural (ξ,η) coordinates as shown in Figure 5.3.

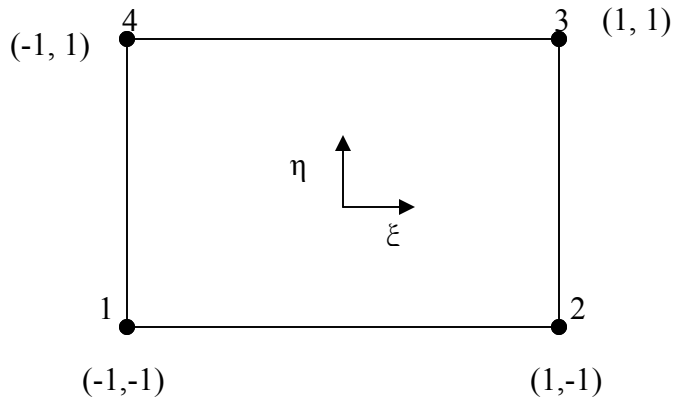

Figure 5.3 Four-node iso-parametric quadrilateral in natural coordinates.

5.4.1 Interpolation Functions

The displacements u and v are obtained from nodal coordinates by means of Lagrange interpolation functions as

$$u(\xi,\eta) = \sum_{i=1}^{4} N_i u_i$$

$$v(\xi,\eta) = \sum_{i=1}^{4} N_i v_i \qquad (5.28)$$

For an iso-parametric element, the same shape function can be used to determine the geometry and the displacements, which is why this element is known as iso-parametric.
The geometry of the element can be expressed as

$$x(\xi,\eta) = \sum_{i=1}^{4} N_i x_i$$

$$y(\xi,\eta) = \sum_{i=1}^{4} N_i y_i \qquad (5.29)$$

where N_1, N_2, N_3 and N_4 are shape functions in the natural coordinate system. These shape functions can be written as

$$N_1(\xi,\eta)=\frac{1}{4}(1-\xi)(1-\eta)$$

$$N_2(\xi,\eta)=\frac{1}{4}(1+\xi)(1-\eta)$$

$$N_3(\xi,\eta)=\frac{1}{4}(1+\xi)(1+\eta)$$

$$N_4(\xi,\eta)=\frac{1}{4}(1-\xi)(1+\eta) \tag{5.30}$$

The displacements $u(\xi,\eta)$ and $v(\xi,\eta)$ are rewritten in matrix form as

$$\begin{Bmatrix} u(\xi,\eta) \\ v(\xi,\eta) \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{Bmatrix} \tag{5.31}$$

5.4.2 Stiffness Matrix

To obtain the element stiffness matrix of an iso-parametric quadrilateral element, the strain-displacement is converted from Cartesian space to natural space. Hence the derivatives of function *f* in x, y coordinates need to be expressed in terms of ξ,η . For a function *f(x, y),* the derivatives with respect to ξ and η are obtained from the chain rule of differentiation:

$$\frac{\partial f}{\partial \xi}=\frac{\partial f}{\partial x}\frac{\partial x}{\partial \xi}+\frac{\partial f}{\partial y}\frac{\partial y}{\partial \xi}$$
$$\frac{\partial f}{\partial \eta}=\frac{\partial f}{\partial x}\frac{\partial x}{\partial \eta}+\frac{\partial f}{\partial y}\frac{\partial y}{\partial \eta} \tag{5.32}$$

or

$$\begin{Bmatrix} \frac{\partial f}{\partial \xi} \\ \frac{\partial f}{\partial \eta} \end{Bmatrix} = J \begin{Bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{Bmatrix} \tag{5.33}$$

The transformation matrix used to map the strain-displacement matrix from Cartesian space to natural space is known as the Jacobian matrix and is given by

$$[J] = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \tag{5.34}$$

The determinant of the Jacobian is given by

$$|J| = J_{11}J_{22} - J_{21}J_{12} \tag{5.35}$$

and the inverse $[J]^{-1}$ is

$$[J]^{-1} = \frac{1}{|J|} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} \tag{5.36}$$

Substituting the shape functions from Equation (5.29), the terms in the Jacobian matrix are obtained as

$$\begin{aligned} J_{11} &= \frac{\partial x}{\partial \xi} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial \xi} x_i \\ J_{12} &= \frac{\partial y}{\partial \xi} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial \xi} y_i \\ J_{21} &= \frac{\partial x}{\partial \eta} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial \eta} x_i \\ J_{22} &= \frac{\partial y}{\partial \eta} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial \eta} y_i \end{aligned} \tag{5.37}$$

The strain-displacement relations are expressed as

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \end{Bmatrix} \tag{5.38}$$

The derivatives in Cartesian space can be expressed in terms of derivatives in natural space as

$$\begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial u}{\partial y} \end{Bmatrix} = \frac{1}{|J|} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} \begin{Bmatrix} \dfrac{\partial u}{\partial \xi} \\ \dfrac{\partial u}{\partial \eta} \end{Bmatrix} \tag{5.39}$$

$$\begin{Bmatrix} \dfrac{\partial v}{\partial x} \\ \dfrac{\partial v}{\partial y} \end{Bmatrix} = \frac{1}{|J|} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} \begin{Bmatrix} \dfrac{\partial v}{\partial \xi} \\ \dfrac{\partial v}{\partial \eta} \end{Bmatrix} \tag{5.40}$$

Combining Equations (5.38), (5.39) and (5.40), the strains are expressed in term of local coordinates *(ξ,η)* as

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = A \begin{Bmatrix} \dfrac{\partial u}{\partial \xi} \\ \dfrac{\partial u}{\partial \eta} \\ \dfrac{\partial v}{\partial \xi} \\ \dfrac{\partial v}{\partial \eta} \end{Bmatrix} \tag{5.41}$$

where

$$A = \frac{1}{|J|} \begin{bmatrix} J_{22} & -J_{12} & 0 & 0 \\ 0 & 0 & -J_{21} & J_{11} \\ -J_{21} & J_{11} & J_{22} & -J_{12} \end{bmatrix} \tag{5.42}$$

Further, the derivatives of the displacements with respect to natural coordinates are expressed as

$$\frac{\partial u}{\partial \xi} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial \xi} u_i$$

$$\frac{\partial u}{\partial \eta} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial \eta} u_i$$

$$\frac{\partial v}{\partial \xi} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial \xi} v_i$$

$$\frac{\partial v}{\partial \eta} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial \eta} v_i \tag{5.43}$$

The above equations can be rewritten in matrix form as

$$\begin{Bmatrix} \frac{\partial u}{\partial \xi} \\ \frac{\partial u}{\partial \eta} \\ \frac{\partial v}{\partial \xi} \\ \frac{\partial v}{\partial \eta} \end{Bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & 0 & \frac{\partial N_2}{\partial \xi} & 0 & \frac{\partial N_3}{\partial \xi} & 0 & \frac{\partial N_4}{\partial \xi} & 0 \\ \frac{\partial N_1}{\partial \eta} & 0 & \frac{\partial N_2}{\partial \eta} & 0 & \frac{\partial N_3}{\partial \eta} & 0 & \frac{\partial N_4}{\partial \eta} & 0 \\ 0 & \frac{\partial N_1}{\partial \xi} & 0 & \frac{\partial N_2}{\partial \xi} & 0 & \frac{\partial N_3}{\partial \xi} & 0 & \frac{\partial N_4}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & 0 & \frac{\partial N_2}{\partial \eta} & 0 & \frac{\partial N_3}{\partial \eta} & 0 & \frac{\partial N_4}{\partial \eta} & 0 \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{Bmatrix} \tag{5.44}$$

Equation (5.44) can be written as

$$\{\varepsilon\} = [A][G]\{u\} \tag{5.45}$$

where

$$[G] = \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & 0 & \frac{\partial N_2}{\partial \xi} & 0 & \frac{\partial N_3}{\partial \xi} & 0 & \frac{\partial N_4}{\partial \xi} & 0 \\ \frac{\partial N_1}{\partial \eta} & 0 & \frac{\partial N_2}{\partial \eta} & 0 & \frac{\partial N_3}{\partial \eta} & 0 & \frac{\partial N_4}{\partial \eta} & 0 \\ 0 & \frac{\partial N_1}{\partial \xi} & 0 & \frac{\partial N_2}{\partial \xi} & 0 & \frac{\partial N_3}{\partial \xi} & 0 & \frac{\partial N_4}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & 0 & \frac{\partial N_2}{\partial \eta} & 0 & \frac{\partial N_3}{\partial \eta} & 0 & \frac{\partial N_4}{\partial \eta} & 0 \end{bmatrix} \tag{5.46}$$

Therefore,

$$\{\varepsilon\} = [B]\{u\} \tag{5.47}$$

where $[B] = [A][G]$ is the strain-displacement matrix

The stresses in the element are given by

$$\{\sigma\} = [D][B]\{u\} \tag{5.48}$$

where [D] is the material matrix.

The element stiffness matrix is given by

$$[k] = t\int_A [B]^T [D][B]dA \tag{5.49}$$

Since [B] is in terms of natural coordinates, Equation (5.49) is integrated with respect to natural coordinates. Substituting

$$dA = dxdy = \left|J(\xi,\eta)\right| d\xi d\eta \tag{5.50}$$

we obtain the element stiffness matrix as

$$[k] = t\int_{-1}^{1}\int_{-1}^{1} [B(\xi,\eta)]^T [D][B(\xi,\eta)]\left|J(\xi,\eta)\right| d\xi d\eta \tag{5.51}$$

where t is the thickness of the element. The element stiffness matrix so obtained is an 8x8 matrix and each term is a function of ξ and η. Gauss quadrature is used to integrate these terms. If a 2x2 Gauss quadrature rule is used, then the stiffness matrix is

$$[k] = t\sum_{j=1}^{2}\sum_{i=1}^{2} w_j w_i [B(\xi,\eta)]^T [D][B(\xi,\eta)]\left|J(\xi,\eta)\right| d\xi d\eta \tag{5.52}$$

The roots and weights for 2x2 Gauss quadrature are given in Table 5.2 (Chandrupatla et al. 1997).

Table 5.2 Roots and weights for 2x2 Gauss quadrature.

Roots	Weight Functions (w)
±0.7745966692	0.55555555555556

5.4.3 Element Force Vectors

Uniformly distributed forces acting on the elements are converted to equivalent nodal loads. This is determined by using the potential energy expression (Chandrupatla et al. 1997)

$$\int_V u^T(\xi,\eta) f dV \tag{5.53}$$

Using Equation (5.28) and considering the body force $f = [f_x, f_y]^T$, we get

$$\int_V u^T(\xi,\eta) f dV = \sum_{i=1}^{N} u^T f_i \tag{5.54}$$

where N is the number of elements in the structure. The force vector of each element is given by

$$f = t\int_{-1}^{1}\int_{-1}^{1} [N(\xi,\eta)]^T \left|J(\xi,\eta)\right| d\xi d\eta \begin{Bmatrix} f_x \\ f_y \end{Bmatrix} \tag{5.55}$$

The force vector is evaluated using Gauss quadrature.

5.4.4 Implementation of Four-node Quadrilateral Element in Java Applet

For implementing the four-node iso-parametric quadrilateral element in Java, the class `Qelement` is developed. The following is a brief description of variables and functions in the `Qelement` class.

```
public class Qelement {
   private double thick;   //thickness of the element
   private double[] coodx; //x coordinates of each element
   private double[] coody; //y coordinates of each element
   private double[][] d;   // material matrix
   private double[] dxi;   //derivatives of shape functions w.r.t si
   private double[] deta; //derivaitives of shape functions w.r.t eta
   private double[][] jacb;      // Jacobain martrix
   private double djac;          //determinant of Jacobian matrix
   private double[][] b;         //strain-displacement matrix
   public double[][] q4kelem;    //element Stiffness matrix
   public double[] stressmat;    //stress matrix
```

The variables `thick` and the arrays `coodx[]`, `coody[]` contain the coordinates of the element. Arrays `dxi` and `deta` contain the derivates of the shape functions with respect to ξ and η, respectively. The two-dimensional arrays `d`, `b`, `jacb` and `q4kelem` store the material

matrix [*D*], calculated strain-displacement matrix [*B*], the Jacobian matrix, and the element stiffness matrix, respectively.

The constructor for the `Qelement` class is

```
public Qelement(double[] x, double[] y, double[][] matb,
                  double th) {

       d = matb;
       thick = th;
       coodx = x;
       coody = y;
   }
```

To create an instance of the four-node quadrilateral element object, the constructor of the `Qelement` class is called and the coordinates, thickness, and material property matrix are passed as arguments. The methods in the `Qelement` class use this data to calculate the strain-displacement matrix, the shape functions, the stiffness matrix, and stresses.

The methods in the `Qelement` class are listed in Table 5.3

Table 5.3 Methods in class `Qelement`

Method	Description
`q4dshapefx()`	Calculates shape functions
`jacobian()`	Calculates Jacobian matrix
`q4Bmatrix()`	Calculates the strain-displacement matrix
`calcelemk()`	Calculates the element stiffness matrix
`calcstress()`	Calculates element stresses

Method `calcelemk()` calculates the stiffness matrix by using 2x2 Gauss quadrature. For each Gauss point the methods `q4dshapefx()`, `jacobian()` and `q4Bmatrix()` are invoked to calculate the shape functions, Jacobian matrix, and strain-displacement matrix, respectively.

Then the contributions to the element stiffness matrices at each Gauss point are calculated and these are added to form the element stiffness matrix.

The approach used to analyze a structure consisting of four-node quadrilateral elements is as follows. For each element an instance of the `Qelement` class is created by calling the constructor from the `Analyze` class.

```
Qelement q= new Qelement(x,y,matb,th);
```

where `q` is an instance of the `Qelement` class. The required values such as the x and y coordinates, and the material matrix, are passed in the constructor. These values are then used by the methods in the `Qelement` class to calculate the element stiffness matrix which is returned to the calling method in the `Analyze` class. The `Analyze` class further assembles the structure stiffness matrix by using the element stiffness matrix for each `Qelement` object .

The method `calcstress()` method mentioned calculates the stresses in the element and stores the results in a new `Stresses` object.
The `calcstress()` method is declared as

```
public void calcstress(double[] u,int[] elemcon,Stresses sts)
```

This method uses the nodal displacements of the element obtained after analysis from the `Analyze` class for calculating the stresses. The stresses at each node are calculated using Gauss 2x2 quadrature. The methods `q4dshapefx()`, `jacobian()` and `q4Bmatrix()` are called at each Gauss point and stresses are calculated using the stress-strain relationships mentioned in section 5.4.2. The stresses obtained are stored in a new `Stresses` object `sts` along with element connectivity information to associate them with nodal coordinates.

5.5 Eight-node Iso-Parametric Quadrilateral

The eight-node isoparametric quadrilateral element is an extension of the four-node element. Four additional nodes are considered along the mid-points of the element boundaries.

The nodes of the quadrilateral element are numbered as shown in Figure 5.4

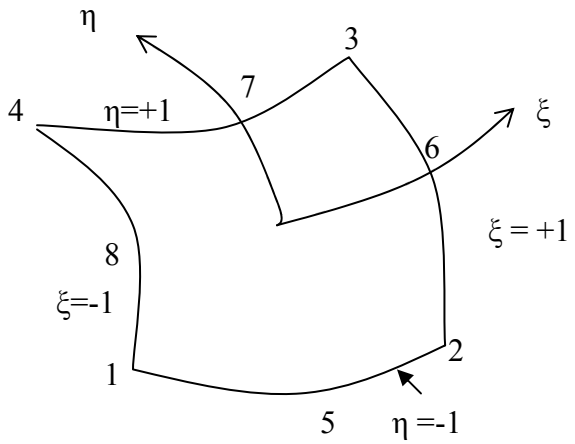

Figure 5.4 Eight-node quadrilateral element in x, y space.

The eight-node quadrilateral element in natural coordinates is shown in Figure 5.5

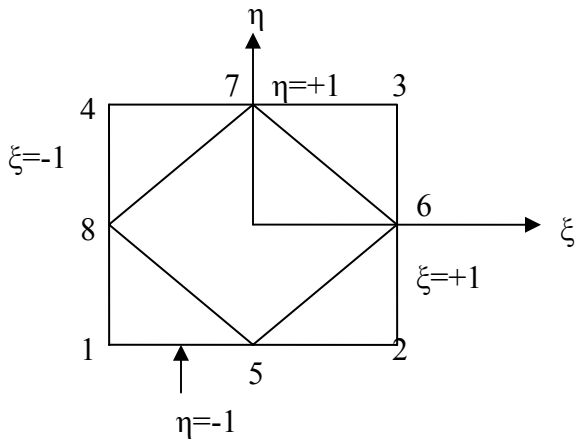

Figure 5.5 Eight-node quadrilateral element in ξ, η space.

Each node has two degrees of freedom, an x and y translation, for a total of sixteen degrees of freedom for the element.

As for the four-node quadrilateral element, the displacements and coordinates at any point in the element are expressed in terms of the nodal displacements by

$$u(\xi,\eta) = \sum_{i=1}^{8} N_i u_i$$

$$v(\xi,\eta) = \sum_{i=1}^{8} N_i v_i \qquad (5.56)$$

The x and y coordinates of any point are obtained from

$$x(\xi,\eta) = \sum_{i=1}^{8} N_i x_i$$

$$y(\xi,\eta) = \sum_{i=1}^{8} N_i y_i \qquad (5.57)$$

where, N_1, N_2, N_3 ,… N_8 are the shape functions in the natural coordinate system. These shape functions can be written as

$$N_1(\xi,\eta) = -\frac{1}{4}(1-\xi)(1-\eta)(1+\xi+\eta)$$

$$N_2(\xi,\eta) = -\frac{1}{4}(1+\xi)(1-\eta)(1-\xi+\eta)$$

$$N_3(\xi,\eta) = -\frac{1}{4}(1+\xi)(1+\eta)(1-\xi-\eta)$$

$$N_4(\xi,\eta) = -\frac{1}{4}(1-\xi)(1+\eta)(1+\xi-\eta)$$

$$N_5(\xi,\eta) = \frac{1}{2}(1-\xi^2)(1-\eta)$$

$$N_6(\xi,\eta) = \frac{1}{2}(1+\xi\ )(1-\eta^2)$$

$$N_7(\xi,\eta) = \frac{1}{2}(1-\xi^2)(1+\eta)$$

$$N_8(\xi,\eta) = \frac{1}{2}(1-\xi\ )(1-\eta^2) \qquad (5.58)$$

The formulation of the stiffness matrix for this element is similar to that for the four-node quadrilateral element. Also, numerical integration and the calculation of the load vector follow the same approach as before. Although Gauss quadrature using a 3x3 grid is required to integrate all the terms of the stiffness matrix, it has been suggested that a Gauss two point

(2x2) quadrature be used to prevent the element from becoming too stiff and to yield conservative estimates of deflections and stresses (Liang-Wu Cai 2004).

5.5.1 Implementation of the Eight-node Quadrilateral Element in Java

The implementation of the eight-node quadrilateral element in Java is similar to that of the four-node quadrilateral element with few modifications such as matrix declarations and the shape functions. The `EightNelement` class is developed for implementing the eight-node iso-parametric quadrilateral element in Java. The following is a brief description of the variables and functions in the `EightNelement` class.

```
public class EightNelement {
    private double thick;  //thickness of the Element
   private double[] coodx; //x coordinates of each element
   private double[] coody; //y coordinates of each element
   private double[][] d;   // material matrix
   private double[] dxi;   //derivative of shape functions w.r.t si
   private double[] deta;  //derivative of shape functions w.r.t eta
   private double[][] jacb;      // Jacobain matrix
   private double djac;          //determinant of the Jacobian matrix
   private double[][] b;         //strain-displacement matrix
   public double[][] q8kelem;    //element Stiffness matrix
   public double[] stressmat;    //stress matrix
```

The variable `thick` and the arrays `coodx[]`, `coody[]` are used to store the thickness and the coordinates of the element. Arrays `dxi` and `deta` represent the derivatives of the shape functions with respect to ξ and η respectively. The material matrix [*D*], strain-displacement matrix [*B*], Jacobian matrix and the element stiffness matrix are stored in the two-dimensional arrays `d[]`, `b[]`, `jacb[]` and `q8kelem[],` respectively.

The constructor for the `EightNelement` class is

```
public EightNelement(double[] x, double[] y, double[][] matb,
                     double th) {

        d = matb;
        thick = th;
        coodx = x;
        coody = y;
    }
```

To create an eight-node quadrilateral element object, the constructor of the `EightNelement` class is called and the coordinates, thickness, and material property matrix of the element are passed as arguments. The methods in the `EightNelement` class use these variables as data to calculate the strain-displacement matrix, shape functions, stiffness matrix, and stresses.

The methods in the `EightNelement` class are summarized in Table 5.4

Table 5.4 Methods in class `EightNelement`.

Method	Description
`q8dshapefx()`	Calculates shape functions
`jacobian()`	Calculates the Jacobian matrix
`q8Bmatrix()`	Calculates the strain-displacement matrix
`calcelemk()`	Calculates the element stiffness matrix
`calcstress()`	Calculates stresses in the element

Method `calcelemk()` calculates the stiffness matrix by using Gauss 2x2 quadrature. For each Gauss point the methods `q8dshapefx()`, `jacobian()` and `q8Bmatrix()` are invoked to calculate the shape functions, Jacobian matrix, and the strain-displacement matrix. Then the element stiffness matrix at each Gauss point is computed and is added to form the element stiffness matrix.

The approach used to analyze a structure containing eight-node quadrilateral elements is as follows. For each element an instance of the `EightNelement` class is created by calling the constructor from the `Analyze` class:

```
EightNelement q8 = new EightNelement(x,y,matb,th);
```

where `q8` is an instance of the `EightNelement` class. The required values such as x and y coordinates, and material matrix, are passed in the constructor. These values are used by the various methods to calculate the stiffness matrix which is returned to the calling method in the `Analyze` class. The `Analyze` class further assembles the structure stiffness matrix by obtaining the element stiffness matrix for each `EightNelement` object.

The method `calcstress()`mentioned calculates the stresses in the element and stores them in a new `Stresses` object for later retrieval. The `calcstress()` method is declared as

```
public void calcstress(double[] u,int[] elemcon,Stresses sts)
```

This method uses the nodal displacements of the element obtained after analysis from the `Analyze` class for calculating the stresses. The stresses at each node are calculated using Gauss 2x2 quadrature. The methods `q8dshapefx()`, `jacobian()` and `q8Bmatrix()` are called at each Gauss point and stresses are calculated using the stress-strain relationships mentioned in Section 5.4.2. The stresses obtained are stored in a new `Stresses` object `sts` along with element connectivity information to associate them with nodal coordinates.

Chapter 6: Plate Bending Elements

6.1 Introduction

Two types of plate bending elements are discussed in this chapter: (1) Discrete Kirchhoff Triangle (DKT) element developed by Batoz et al. (1980) and (2) Discrete Kirchhoff Quadrilateral (DKQ) element developed by Batoz and Tahar (1982). The development of the element stiffness matrix for each of these elements and the implementation in Java are presented in this chapter. The classes for implementing both of these elements were developed in a master's thesis by KaushalKumar Kansara(Virginia Tech 2004) under the supervision of Professor Rojiani and are included in the Java applet with due permission.

6.2 Bending of Flat Elastic Plates

A plate can be defined as the two-dimensional equivalent of a beam with bending in two principal directions. Plates are subjected to transverse loads perpendicular to the plane of the plate. The behavior of the plate can be described by classical plate theory which is an extension of the Euler – Bernoulli beam theory to plates. This is also known as Kirchhoff plate theory.

The assumptions for bending of thin plates are as follows:

1. Plate is of uniform thickness
2. A line normal to the middle surface of the plate before deformation remains normal after deformation. This assumption is known as Kirchhoff's assumption.
3. The middle surface of the plate remains unchanged after deformation.
4. Transverse shear stresses are neglected as they are small compared to normal stresses.

For an isotropic elastic plate with uniform thickness h in the XY plane, the moments are given by the following relationships:

$$M_x = \int_{-h/2}^{h/2} \sigma_x z dz$$

$$M_y = \int_{-h/2}^{h/2} \sigma_y z dz$$

$$M_{xy} = \int_{-h/2}^{h/2} \tau_{xy} z dz \qquad (6.1)$$

where M_x and M_y are moments per unit length in the x and y directions, respectively, and M_{xy} is the twisting moment per unit length.

6.3 Strain-Displacement Relations

Consider a section through the plate before and after deformation as shown in Figure 6.1. Point A at a distance z from the middle surface before deformation has moved to point A' after deformation.

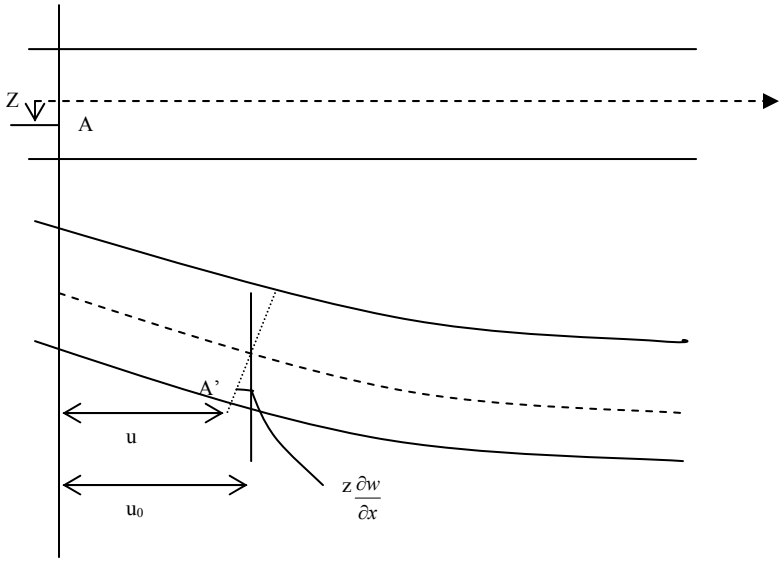

Figure 6.1 Bending of plate.

The displacement of point A in the x-direction represented by u is given by

$$u = u_o - z\frac{\partial w}{\partial x} \tag{6.2}$$

where u_o is the middle surface displacement and w is displacement of the plate in the Z direction.

Similarly the displacement in the y direction is

$$v = v_o - z\frac{\partial w}{\partial y} \tag{6.3}$$

The terms u_o and v_o remain constant based on the assumption made previously that there is no change in length of the middle surface of the plate on deformation. Thus, the strains are obtained by differentiation:

$$\begin{aligned}
\varepsilon_x &= \frac{\partial u}{\partial x} = -z\frac{\partial^2 w}{\partial x^2} \\
\varepsilon_y &= \frac{\partial u}{\partial y} = -z\frac{\partial^2 w}{\partial y^2} \\
\gamma_{xy} &= \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = -2z\frac{\partial^2 w}{\partial x \partial y}
\end{aligned} \tag{6.4}$$

The relationship between stresses and strains is given by

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = [D] \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} \tag{6.5}$$

where [D] is the material matrix and is expressed as

$$[D] = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \dfrac{1-\nu}{2} \end{bmatrix} \tag{6.6}$$

Here, E = Young's modulus of elasticity and ν = Poisson's ratio.

The stresses in the plate are given by substituting Equation (6.6) in Equation (6.5):

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \dfrac{1-\nu}{2} \end{bmatrix}\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} \tag{6.7}$$

Hence the stresses can be expressed as

$$\sigma_x = \frac{-Ez}{1-\nu^2}\left[\frac{\partial^2 w}{\partial x^2} + \nu\frac{\partial^2 w}{\partial y^2}\right]$$

$$\sigma_y = \frac{-Ez}{1-\nu^2}\left[\frac{\partial^2 w}{\partial y^2} + \nu\frac{\partial^2 w}{\partial x^2}\right]$$

$$\sigma_z = \frac{-2Ez}{1-\nu^2}\frac{(1-\nu)}{2}\frac{\partial^2 w}{\partial x \partial y} \tag{6.8}$$

The moments in the plate are obtained by substituting Equations (6.8) in Equation (6.1) and integrating over the thickness h:

$$M_x = \frac{-Eh^3}{12(1-\nu^2)}\left[\frac{\partial^2 w}{\partial x^2} + \nu\frac{\partial^2 w}{\partial y^2}\right]$$

$$M_y = \frac{-Eh^3}{12(1-\nu^2)}\left[\frac{\partial^2 w}{\partial y^2} + \nu\frac{\partial^2 w}{\partial x^2}\right]$$

$$M_{xy} = \frac{-Eh^3}{12(1-\nu^2)}(1-\nu)\left[\frac{\partial^2 w}{\partial x \partial y}\right] \tag{6.9}$$

The term $\frac{Eh^3}{12(1-\nu^2)}$ is known as the bending stiffness of the plate and is represented by D.

6.4 Discrete Kirchhoff Triangle (DKT) Element

The DKT element was developed by Batoz et al. (1980). This element is widely used for the analysis of irregular shaped plates. The stiffness matrix for the DKT element developed by Batoz et al. is discussed in this section. Figure 6.2 shows the DKT element with three degrees of freedom at each node.

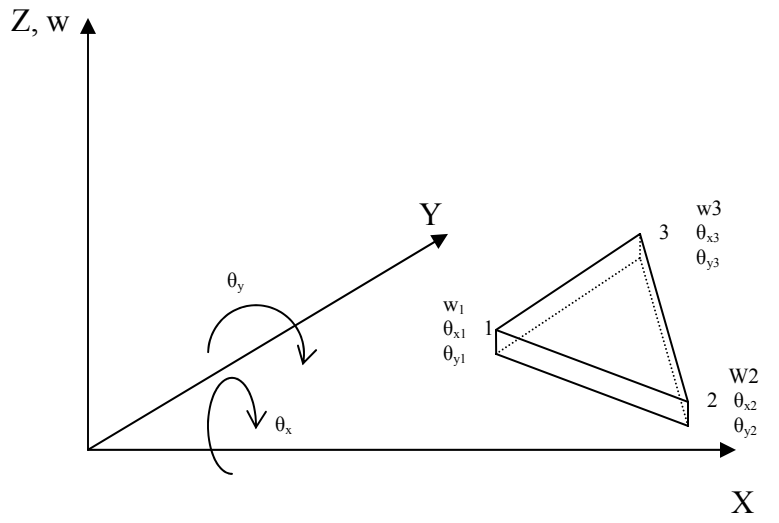

Figure 6.2 Discrete Kirchhoff Triangle.

From the assumptions mentioned earlier, the displacements u, v, and w of a point with coordinates x, y and z are given by

$$u = z\beta_x(x,y) \qquad v = z\beta_y(x,y) \qquad w = w(x,y) \tag{6.10}$$

where w is the transverse displacement in the z direction and β_x, β_y are the rotations of the normal to the undeformed middle surface in the xz and yz planes, respectively, and are given by

$$\begin{Bmatrix} \beta_x \\ \beta_y \end{Bmatrix} = \begin{Bmatrix} -\dfrac{\partial u}{\partial x} \\ -\dfrac{\partial u}{\partial y} \end{Bmatrix} \tag{6.11}$$

The curvatures are given by

$$\kappa = z \begin{bmatrix} \beta_{x,x} \\ \beta_{y,y} \\ \beta_{x,y} + \beta_{y,x} \end{bmatrix} = \begin{bmatrix} -z\dfrac{\partial^2 w}{\partial x^2} \\ -z\dfrac{\partial^2 w}{\partial y^2} \\ -2z\dfrac{\partial^2 w}{\partial x \partial y} \end{bmatrix} \tag{6.12}$$

and the transverse shear strains are given by

$$\gamma = \begin{bmatrix} w_{,x} + \beta_x \\ w_{,y} + \beta_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial w}{\partial x} - \dfrac{\partial u}{\partial x} \\ \dfrac{\partial w}{\partial y} - \dfrac{\partial u}{\partial y} \end{bmatrix} \tag{6.13}$$

The total strain energy is

$$U = U_b + U_s \tag{6.14}$$

where

$$U_b = \frac{1}{2}\int_A \kappa^T D_b \kappa \, dx dy \tag{6.15}$$

$$U_s = \frac{1}{2}\int_A \gamma^T D_s \gamma \, dx dy \tag{6.16}$$

The variables U_b and U_s represent the bending and transverse shear contributions. For thin plates the transverse shear strains and the transverse shear strain energy U_s are negligible compared to the bending energy U_b. Hence the stiffness matrix is derived based on the bending energy expression

$$U = \frac{1}{2}\int_A \kappa^T D_b \kappa dxdy \tag{6.17}$$

where

$$D_b = \frac{Eh^3}{12(1-\nu^2)}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \tag{6.18}$$

The following observations were made by Batoz et al. to relate the rotations of the normal to the middle surface β_x, β_y to the transverse displacement w (which does not appear in Equation (6.17)):

1. The triangular element must have only 9 degrees of freedom, that is, the displacement w and the rotations θ_x and θ_y at the three corner nodes.
2. Using Kirchhoff theory, the nodal point rotations should be $\theta_x = \frac{\partial u}{\partial x}$ and $\theta_y = \frac{\partial u}{\partial y}$.
3. The Kirchhoff plate theory assumptions can be imposed at any discrete point.
4. Compatibility of the rotations β_x and β_y should not be lost.

The formulation of the DKT element is based on the following assumptions made by Batoz et al.:

1. β_x and β_y at any point over the element are given by

$$\beta_x = \sum_{i=1}^{6} N_i \beta_{xi} \;;\; \beta_y = \sum_{i=1}^{6} N_i \beta_{yi} \tag{6.19}$$

where β_{xi} and β_{xi} are the nodal rotations at the corners and at the mid-nodes.

The shape functions for the DKT element in area coordinates are given by

$$N_1 = 2(1-\xi-\eta)\left(\frac{1}{2}-\xi-\eta\right)$$

$$N_2 = \xi(2\xi-1)$$

$$N_3 = \eta(2\eta-1)$$

$$N_4 = 4\xi\eta$$

$$N_5 = 4\eta(1-\xi-\eta)$$

$$N_6 = 4\xi(1-\xi-\eta) \tag{6.20}$$

2. The Kirchhoff hypothesis is imposed to remove transverse strains.
   a. At the corner nodes 1,2, and 3

$$\gamma = \begin{bmatrix} w_{,x} + \beta_x \\ w_{,y} + \beta_y \end{bmatrix} = 0 \tag{6.21}$$

   b. At the mid-nodes defined in a counter-clockwise sense around the element

$$w_{,sk} + \beta_{sk} = 0 \text{ where k = 4, 5, and 6} \tag{6.22}$$

3. The variation of the transverse displacement w is cubic and is given by

$$w_{,sk} = \frac{-3}{2l_{ij}} w_i - \frac{1}{4} w_{si} + \frac{3}{2l_{ij}} w_j - \frac{1}{4} w_{sj} \tag{6.23}$$

   where k denotes the mid-node of side ij and l_{ij} represents the length of side ij.

4. The variation of rotations along the sides are given by

$$\beta_{nk} = \frac{1}{2}\left(\beta_{ni} + \beta_{nj}\right) \tag{6.24}$$

   where k = 4, 5, and 6 represent the mid-nodes of sides 23, 31, and 12, respectively.

Since w varies cubically along the sides, $w_{,s}$ and β_s vary quadratically. Hence $w_{,s}$ matches β_s at the three points along each side, thereby satisfying the Kirchhoff hypothesis along the entire boundary.

The nodal degrees of freedom of this element are represented by

$$U^T = \{w_1, \theta_{x1}, \theta_{y1}, w_2, \theta_{x2}, \theta_{y2}, w_3, \theta_{x3}, \theta_{y3}\} \tag{6.25}$$

β_x and β_y are expressed in terms of the nodal degrees of freedom using the following expressions:

$$\beta_x = H_x^T(\xi,\eta)U$$
$$\beta_y = H_y^T(\xi,\eta)U \tag{6.26}$$

where H_x^T and H_y^T are the component vectors of the shape functions and are represented by the following expressions:

$$H_x = \begin{bmatrix} 1.5(a_6N_6 - a_5N_5) \\ b_5N_5 + b_6N_6 \\ N_1 - c_5N_5 - c_6N_6 \\ 1.5(a_4N_4 - a_6N_6) \\ b_6N_6 + b_4N_4 \\ N_2 - c_6N_6 - c_4N_4 \\ 1.5(a_5N_5 - a_4N_4) \\ b_4N_4 + b_5N_5 \\ N_3 - c_4N_4 - c_5N_5 \end{bmatrix} \tag{6.27}$$

$$H_y = \begin{bmatrix} 1.5(d_6N_6 - d_5N_5) \\ -N_1 + e_5N_5 + e_6N_6 \\ -b_5N_5 - b_6N_6 \\ 1.5(d_4N_4 - d_6N_6) \\ -N_2 + e_6N_6 + e_4N_4 \\ -b_6N_6 - b_4N_4 \\ 1.5(d_5N_5 - d_4N_4) \\ -N_3 + e_4N_4 + e_5N_5 \\ -b_4N_4 - b_5N_5 \end{bmatrix} \tag{6.28}$$

Also,

$$a_k = -x_{ij} / l_{ij}^2$$
$$b_k = \frac{3}{4} x_{ij} y_{ij} / l_{ij}^2$$
$$c_k = (\frac{1}{4} x_{ij}^2 - \frac{1}{2} y_{ij}^2) / l_{ij}^2$$

$$
\begin{aligned}
d_k &= -y_{ij} / l_{ij}^2 \\
e_k &= (\frac{1}{4} y_{ij}^2 - \frac{1}{2} x_{ij}^2) / l_{ij}^2 \\
x_{ij} &= x_i - x_j \\
y_{ij} &= y_i - y_j \\
l_{ij}^2 &= (x_{ij}^2 + y_{ij}^2)
\end{aligned}
\tag{6.29}
$$

where k = 4, 5, and 6 for sides ij = 23, 31, and 12, respectively.

The stiffness matrix of the DKT element is evaluated using the displacement method and is given by

$$
K_{DKT} = 2A \int_0^1 \int_0^{1-\eta} B^T D_b B d\xi d\eta \tag{6.30}
$$

where B is the strain-displacement matrix given by

$$
B(\xi, \eta) = \frac{1}{2A} \begin{bmatrix} y_{31} H_{x,\xi}^T + y_{12} H_{x,\eta}^T \\ -x_{31} H_{y,\xi}^T - x_{12} H_{y,\eta}^T \\ -x_{31} H_{x,\xi}^T - x_{12} H_{x,\eta}^T + y_{31} H_{y,\xi}^T + y_{12} H_{y,\eta}^T \end{bmatrix} \tag{6.31}
$$

and A is the area given by $A = 0.5(x_{31} y_{12} - x_{12} y_{31})$

Assuming, the element is of constant thickness, the stiffness matrix of the DKT element is evaluated using three point Gauss quadrature.

Once the nodal displacements are determined, the bending moments at any point in the element can be obtained using the following equations (Batoz et al. 1980)

$$
M(x, y) = D_b B(x, y) U \tag{6.32}
$$

where

$$
\begin{aligned}
x &= x_1 + \xi x_{21} + \eta x_{31} \\
y &= y_1 + \xi y_{21} + \eta y_{31}
\end{aligned}
\tag{6.33}
$$

6.5 Implementation of DKT Element in Java

The class `DKTElement` was developed by KaushalKumar Kansara and requires the x and y coordinates, material matrix [D], and thickness of the element. When analyzing a structure using the `DKTElement`, an instance of this class is created by the `Analyze` class. The `DKTElement` class has several methods for calculating the shape functions, the strain-displacement matrix, and the element stiffness matrix of the element. Table 6.1 lists the methods available in `DKTElement` along with their use.

Table 6.1 Methods in class `DKTelement`

Method	**Description**
`Geometry()`	Calculates the geometric rotation components for an element
`ShFnHXxi()`	Calculates the derivatives of the component vector of the shape functions in the x direction with respect to ξ .
`ShFnHXeta()`	Calculates the derivatives of the component vector of the shape functions in the x direction with respect to η
`ShFnHYxi()`	Calculates the derivatives of the component vector of the shape functions in the y direction with respect to ξ
`ShFnHYeta()`	Calculates the derivatives of the component vector of the shape functions in the y direction with respect to η
`DKTElementKMatrix()`	Calculates the stiffness matrix of the DKT element
`DKTBMatrix()`	Calculates the strain-displacement matrix of the DKT element
`CalcStresses()`	Calculates stresses in the DKT element

The stresses are calculated using the stress-strain relationships mentioned previously and the values at each node are passed to the `Stresses` class which stores them.

6.6 Discrete Kirchhoff Quadrilateral (DKQ) Element

The Kirchhoff Quadrilateral plate bending element (DKQ) is essentially developed along the same lines as the DKT element using the Kirchhoff plate bending assumptions. The DKQ element has four nodes with three degrees of freedom at each node for a total of 12 degrees of freedom for the element. The Discrete Kirchhoff Quadrilateral (DKQ) element was developed by Batoz and Tahar (1982) for analyzing plate structures such as flat slabs and footings. Figure 6.3 shows the DKQ element with w, θ_x, and θ_y, representing the transverse displacement and rotation in the x and y directions.

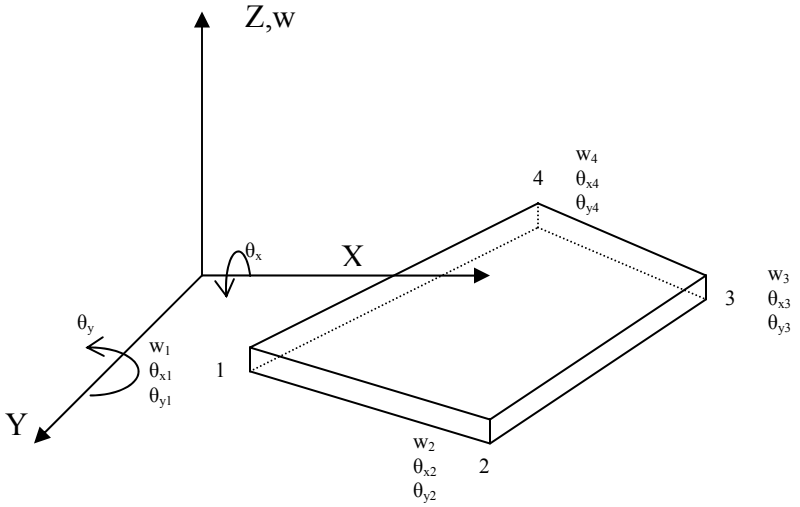

Figure 6.3 Discrete Kirchhoff Quadrilateral.

The displacements at any point *(x, y)* are given by

$$w = w(x, y) \qquad \theta_x = \frac{\partial w}{\partial y} = w_{,y} \qquad \theta_y = \frac{\partial w}{\partial x} = w_{,x} \tag{6.34}$$

This section discusses the development of the DKQ element by Batoz and Tahar. The basic arguments and formulation of DKQ element remain the same as that of the DKT element. Neglecting the transverse shear strain energy, the total strain energy is given by

$$U = \sum_e U_b^e \tag{6.35}$$

where U_b^e is the element strain energy due to bending and is given by

$$U_b^e = \frac{1}{2}\int_{A^e} \langle \chi \rangle [D_b] \{\chi\} \, dxdy \tag{6.36}$$

and A^e is the area of element

The curvatures for a homogenous isotropic plate are given by

$$\{\chi\} = \begin{Bmatrix} \partial\beta_x / \partial x \\ \partial\beta_y / \partial y \\ \partial\beta_x / \partial y + \partial\beta_y / \partial x \end{Bmatrix} \tag{6.37}$$

where β_x and β_y are the rotations of the normal to the undeformed middle surface in the xz and yz planes, respectively.

The material matrix is given by

$$D_b = \frac{Eh^3}{12(1-\nu^2)} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \dfrac{1-\nu}{2} \end{bmatrix} \tag{6.38}$$

E, ν, and h are Young's modulus, Poisson's ratio, and thickness of the plate, respectively.

Batoz and Tahar made the following observations while formulating the DKQ element:

1. β_x and β_y are defined by incomplete cubic polynomials

$$\beta_x = \sum_{i=1}^{8} N_i \beta_{xi} \qquad \beta_y = \sum_{i=1}^{8} N_i \beta_{yi} \tag{6.39}$$

Here N_i represent the shape functions of the eight-node quadrilateral element and are given by

$$N_1(\xi,\eta) = -\frac{1}{4}(1-\xi)(1-\eta)(1+\xi+\eta)$$

$$N_2(\xi,\eta) = -\frac{1}{4}(1+\xi)(1-\eta)(1-\xi+\eta)$$

$$N_3(\xi,\eta) = -\frac{1}{4}(1+\xi)(1+\eta)(1-\xi-\eta)$$

$$N_4(\xi,\eta) = -\frac{1}{4}(1-\xi)(1+\eta)(1+\xi-\eta)$$

$$N_5(\xi,\eta) = \frac{1}{2}(1-\xi^2)(1-\eta)$$

$$N_6(\xi,\eta) = \frac{1}{2}(1+\xi\ )(1-\eta^2)$$

$$N_7(\xi,\eta) = \frac{1}{2}(1-\xi^2)(1+\eta)$$

$$N_8(\xi,\eta) = \frac{1}{2}(1-\xi\ )(1-\eta^2) \tag{6.40}$$

2. The Kirchhoff hypothesis is imposed to remove transverse strains.

   a. At the corner nodes 1,2, 3 and 4

$$\gamma = \begin{bmatrix} w_{,xi} + \beta_{xi} \\ w_{,yi} + \beta_{yi} \end{bmatrix} = 0 \tag{6.41}$$

   b. At the mid-point nodes defined in a counter-clockwise sense around the element

$$w_{,sk} + \beta_{sk} = 0 \text{ where k =5, 6, 7, and 8} \tag{6.42}$$

   where *s* represents coordinates along the element boundary.

3. The transverse displacement w at any node is defined by a cubic expression and its derivative with respect to s at the mid-node k is given by

$$w_{,sk} = \frac{-3}{2l_{ij}}(w_i - w_j) - \frac{1}{4}(w_{,si} + w_{,sj}) \tag{6.43}$$

where k = 5, 6, 7, and 8 is the mid-node of the sides k = 12, 23, 34, 41, respectively, and l_{ij} represents the length of side ij.

4. The rotations at mid-nodes are average values of the corresponding end nodes:

$$\beta_{nk} = \frac{1}{2}\left(\beta_{ni} + \beta_{nj}\right) = \frac{1}{2}\left(w_{,ni} + w_{,nj}\right) \tag{6.44}$$

where k = 5, 6, 7, and 8 corresponds to the mid node of sides 12, 23, 34, 41, respectively.

For formulating the stiffness matrix of the DKQ element, the rotations β_x and β_y are to be explicitly expressed in terms of nodal variables. The nodal variables for the DKQ element are given by

$$U_n = \{w_1, \theta_{x1}, \theta_{y1}, w_2, \theta_{x2}, \theta_{y2}, w_3, \theta_{x3}, \theta_{y3}, w_4, \theta_{x4}, \theta_{y4}\} \tag{6.45}$$

Hence, the rotations are given by the following expressions.

$$\beta_x = H^x(\xi, \eta)U_n$$
$$\beta_y = H^y(\xi, \eta)U_n \tag{6.46}$$

where H^x and H^y are component vectors of shape functions represented by the following expressions:

$$H^x = \begin{bmatrix} 1.5(a_5N_5 - a_8N_8) \\ b_5N_5 + b_8N_8 \\ N_1 - c_5N_5 - c_8N_8 \\ 1.5(a_6N_6 - a_5N_5) \\ b_6N_6 + b_5N_5 \\ N_2 - c_6N_6 - c_5N_5 \\ 1.5(a_7N_7 - a_6N_6) \\ b_7N_7 + b_6N_6 \\ N_3 - c_7N_7 - c_6N_6 \\ 1.5(a_8N_8 - a_7N_7) \\ b_8N_8 + b_7N_7 \\ N_4 - c_8N_8 - c_7N_7 \end{bmatrix} \tag{6.47}$$

$$H_y = \begin{bmatrix} 1.5(d_5N_5 - d_8N_8) \\ -N_1 + e_5N_5 + e_8N_8 \\ -b_5N_5 - b_8N_8 \\ 1.5(d_6N_6 - d_5N_5) \\ -N_2 + e_6N_6 + e_5N_5 \\ -b_6N_6 - b_5N_5 \\ 1.5(d_7N_7 - d_6N_6) \\ -N_3 + e_7N_7 + e_6N_6 \\ -b_7N_7 - b_6N_6 \\ 1.5(d_8N_8 - d_7N_7) \\ -N_4 + e_8N_8 + e_7N_7 \\ -b_8N_8 - b_7N_7 \end{bmatrix} \tag{6.48}$$

Also,

$$a_k = -x_{ij} / l_{ij}^2$$

$$b_k = \frac{3}{4} x_{ij} y_{ij} / l_{ij}^2$$

$$c_k = (\frac{1}{4} x_{ij}^2 - \frac{1}{2} y_{ij}^2) / l_{ij}^2$$

$$
\begin{aligned}
d_k &= -y_{ij} / l_{ij}^2 \\
e_k &= (\frac{1}{4} y_{ij}^2 - \frac{1}{2} x_{ij}^2) / l_{ij}^2 \\
x_{ij} &= x_i - x_j \\
y_{ij} &= y_i - y_j \\
l_{ij}^2 &= (x_{ij}^2 + y_{ij}^2)
\end{aligned}
\tag{6.49}
$$

where k = 5, 6, 7, and 8 for sides ij = 12, 23, 34, and 41, respectively.

The stiffness matrix of the DKQ element is evaluated using the standard procedure of the displacement method and is given by

$$
K^e = \int_{-1}^{1} \int_{-1}^{1} B^T D_b B \det[J] d\xi d\eta \tag{6.50}
$$

where B is the strain-displacement matrix given by

$$
B(\xi, \eta) = \begin{bmatrix} j_{11} H^x_{,\xi} + j_{12} H^x_{,\eta} \\ -j_{21} H^y_{,\xi} - j_{22} H^x_{,\eta} \\ j_{11} H^y_{x,\xi} + j_{12} H^y_{,\eta} + j_{21} H^x_{,\xi} + j_{22} H^x_{,\eta} \end{bmatrix} \tag{6.51}
$$

In Equation (6.50), j_{11}, j_{12}, j_{21}, and j_{22} are the components obtained by inverting the Jacobian matrix $[J]$ of the transformation between the parent and actual element. The Jacobian matrix is given by

$$
[J] = \begin{bmatrix} x_{21} + x_{34} + \eta(x_{12} + x_{34}) & y_{21} + y_{34} + \eta(y_{12} + y_{34}) \\ x_{32} + x_{41} + \xi(x_{12} + x_{34}) & y_{32} + y_{41} + \xi(y_{12} + y_{34}) \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \tag{6.52}
$$

Thus, the components are given by

$$
\begin{aligned}
j_{11} &= \frac{1}{\det[J]} J_{22} \qquad & j_{12} &= \frac{-1}{\det[J]} J_{12} \\
j_{21} &= \frac{-1}{\det[J]} J_{21} \qquad & j_{22} &= \frac{1}{\det[J]} J_{22}
\end{aligned}
\tag{6.53}
$$

Also, the determinant of the Jacobian matrix is

$$|J| = \frac{1}{8}(y_{42}x_{31} - y_{31}x_{42}) + \frac{\xi}{8}(y_{34}x_{21} - y_{21}x_{34}) + \frac{\eta}{8}(y_{41}x_{32} - y_{32}x_{41}) \quad (6.54)$$

Assuming the element is of constant thickness, the stiffness matrix of the DKQ element is evaluated using the two-point Gauss quadrature.

6.7 Implementation of the DKQ Element in Java

The class `DKQElement` was developed by KaushalKumar Kansara and the inputs to the constructor for the class are x and y coordinates, material matrix [D], and thickness of the element. When analyzing a structure using `DKQElement,` an object of this class is created in the `Analyze` class for each plate in the structure. There are several methods in the `DKQElement` class for calculating shape functions, the strain-displacement matrix, and the element stiffness matrix of the element.

Table 6.2 lists the methods available in the `DKQElement` along with a brief description.

Table 6.2 Methods in class `DKQelement`

Method	Description
`Geometry()`	Calculates the geometric rotation components for an element
`ShFnHXxi()`	Calculates the derivatives of the component vector of the shape functions in x direction with respect to ξ .
`ShFnHXeta()`	Calculates the derivatives of the component vector of the shape functions in the x direction with respect to η
`ShFnHYxi()`	Calculates the derivatives of the component vector of the shape functions in the y direction with respect to ξ
`ShFnHYeta()`	Calculates the derivatives of the component vector of the shape functions in the y direction with respect to η
`DKQElementKMatrix()`	Calculates the stiffness matrix of the DKQ element
`DKQBMatrix()`	Calculates the strain-displacement matrix of the DKQ element
`CalcStresses()`	Calculates stresses in the DKQ element
`Jacobian()`	Calculates the Jacobian matrix

The stiffness matrix of the DKQ Element is computed using the `DKQElementKMatrix()` method. At each Gauss point, the methods `ShFnHXxi()`, `ShFnHXeta()`, `ShFnHYxi()`, `ShFnHYeta()`, and `DKQBMatrix()` are called to evaluate the shape functions and the strain-displacement matrix. Then the contribution to the element stiffness matrix at each Gauss point is calculated and added to form the final stiffness matrix.

The DKQ Element also has a `calcstresses()` method. This method evaluates the stresses at each node and stores the results in a new `Stresses` object for later retrieval during printing of results.

Chapter 7: Applet Structure and Interface

7.1 Introduction

A Java Applet is developed to perform finite element analysis using different elements. This applet is embedded in a HTML page and can be accessed from any place over the Internet. All that is required is a Java enabled web browser. In this chapter, the details regarding the structure of the program and the various classes are presented. The program interface and required input are also described.

7.2 The Applet Structure

The classes developed for the finite element analysis program are divided into three different types: (1) Input and Output classes, (2) Processing classes, and (3) Interface classes. Figure 7.1 shows the structure of the applet along with all of the classes. A description of these classes follows.

Dividing a large complex program into various classes is the essential idea behind object oriented programming. `FemApplet` is the main class of the applet. This class has methods for reading the input data and the `FemApplet` class creates an instance of the `Analyze` class. The `Analyze` class collects the input data and creates the model of the structure with the `Model` class. Data from the `Model` class is used by the `Analyze` class object to perform the analysis of the structure. The `Analyze` class also performs post processing.

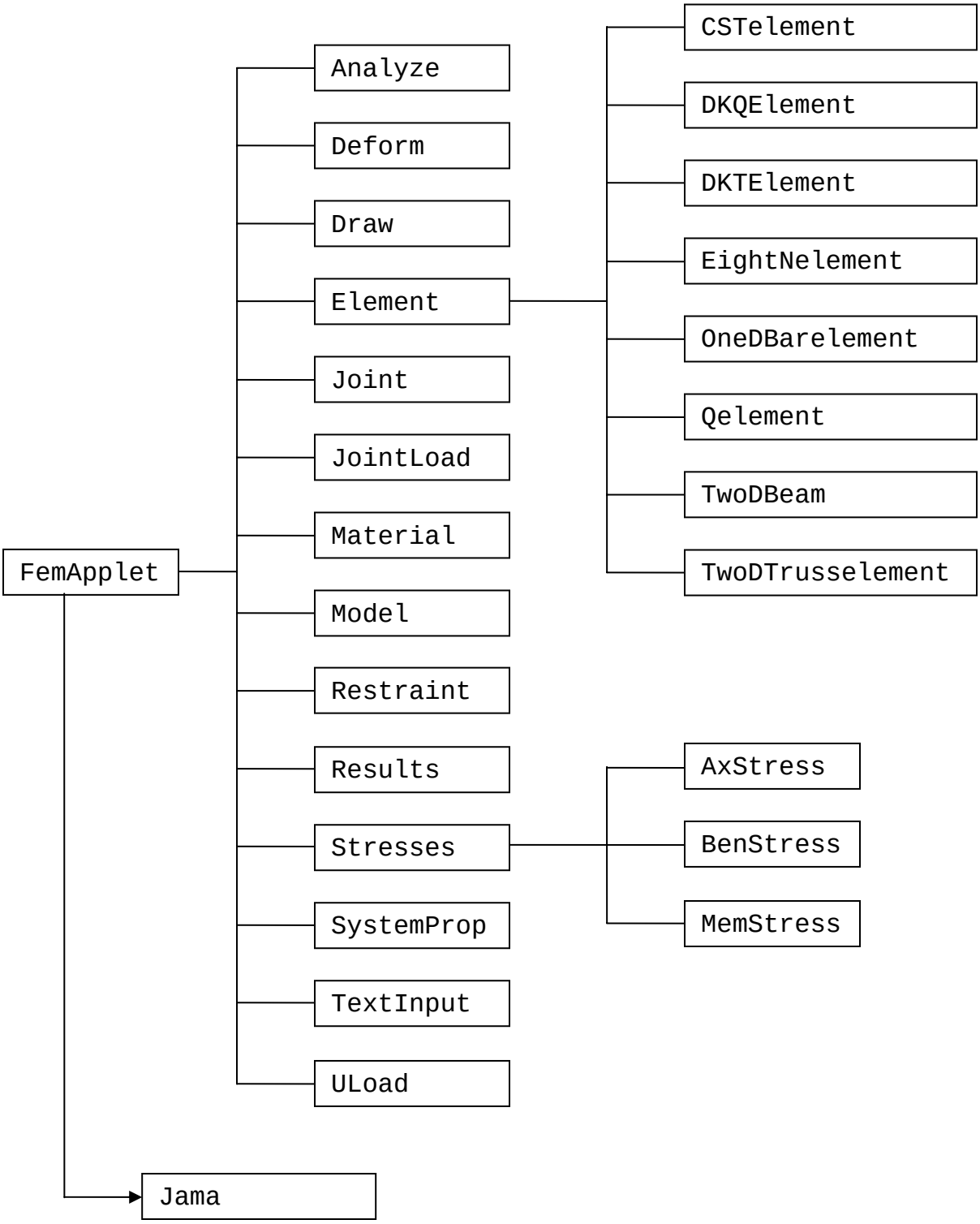

Figure 7.1 Applet structure diagram.

The `Element` class is an important class of this program. Since different types of elements are used for analysis, the `Element` class is the super class and it contains sub classes for representing the different elements such as `OneDBarelement`, `TwoDTrusselement`, `TwoDBeam`, `CSTelement` classes. These classes are derived from the `Element` class by inheritance. The `Element` class also contains methods to determine the type of element and the nodes to which the element is connected. The nodal data of the structure such as the node number and joint coordinates is stored in the `Joint` class. There is an instance of the `Joint` class for each node in the structure. Restraints, nodal loads, and uniform or pressure loads are represented by separate classes.

`Stresses` is a container class which contains classes defined to store different types of stresses. `AxStress`, `BenStress` and `MemStress` are classes defined to represent axial, bending, and membranes stresses, respectively. The `TextInput` class reads the input and creates a model of the structure when text input is used in the program. The `Results` class acquires the data such as displacement and stresses and displays them in the applet window. The `Draw` and `Deform` classes are called by the `FemApplet` class to plot the structure and the deformed shape of the structure.

Finally, a third party Java package named *Jama* developed by the National Institute of Standards and Technology (NIST) is used for matrix operations. The classes in the *Jama* package are called by the `FemApplet` class while performing matrix inversion or multiplication.

Details of these classes are provided in the following paragraphs.

7.3 Input and Output Classes

The required input for the finite element model is specified either by a text file or by entering data in the various input fields provided in the graphical user interface of the program. The text input option essentially creates a finite element model using the format provided by typical commercial finite element programs. This input includes nodal data, connectivity information, material properties, restraints, and joint loads acting on the structure specified in an orderly format as described in Appendix A.

`TextInput` Class

The `TextInput` class is initialized when the input for the program is provided in the form of text. The information required for generating a finite element model is extracted and stored as an instance of the `Model` class. The data acquired by reading the text file is stored into different vectors in the `Model` class. These vectors store information such as joint numbers, joint coordinates, material properties, and connectivity information.

The constructor of the `TextInput` class has `JTextArea` and `FemApplet` as arguments. When the constructor is called, the text from the text area is read in and assigned to a string. All the necessary code required for interpreting and storing the data is provided in this class.

`FemApplet` Class

This is an interface class. It reads in the input entered from the graphical user program. The interface part of this class is discussed in a later section. Several nested classes are contained in this class to provide the functionality for processing the buttons of the input tabs. Action Listeners are added to all interface components like buttons, check boxes, combo boxes, and radio buttons. Whenever an **Add** button is pressed an action event is triggered which in turn calls the corresponding nested class associated with that event. This inner class has a set of instructions for adding the input data to the various vectors in the `Model` object along with displaying the input in the list boxes. The same procedure is applied for the addition of components like nodal coordinates, connectivity information, and restraints. Similarly, **Remove** buttons are provided to remove the selected data from the `Model` object.

Result Class

The `Result` class represents the results frame. Large amounts of data such as displacements at nodes, and stresses in elements, are obtained after performing the analysis. These data are stored in different vectors as `MemStress` or `AxStress` and `BenStress` objects depending on the type of stress.

The `Result` class contains only one method named `AddText()` which adds the displacements and different types of stresses obtained at each node to the `JTextArea` component of the Result frame which is passed as an argument to the constructor of this class. `AddText()` is a method for obtaining the calculated values from the respective vectors in which they are stored and tabulating them in an orderly fashion before printing them. The output format of the results displayed is similar to that of SAP 2000.

7.4 Processing Classes

This is a set of classes which store the input data, perform analysis, and store the results of the structure as objects. Once the structure to be analyzed is defined, the input data obtained from the `TextInput` class or the `FemApplet` class is stored in the `Model` class. The data required to create a finite element model such as degrees of freedom, nodal coordinates, material properties, restraints, element nodal connectivity, and joint loads are represented as objects of class `SystemProp`, `Joint`, `Material`, `Restraint`, `Element`, and `JointLoad`, respectively.

Once the `Model` object is created, an instance of `Analyze` class is generated. This class performs the necessary operations to evaluate displacements and stresses at each node. Instances of the element classes such as `QElement` and `OneDBarElement` are called to generate the stiffness matrix of the respective element.

After performing the analysis, instances of the classes `AxStress`, `BenStress`, and `MemStress` are created to store axial, bending, and membrane stresses. These objects are further stored in a vector using the methods of the `Stresses` class.

`SystemProp` Class

The type of element to be used for analysis is determined by the degrees of freedom possible at each node. The `SystemProp` class defines the behavior of the structure, that is, the possible directions in which the structure can deform on loading. This class is composed of three methods. Method `getdof()` returns an array which is used by the `Analyze` class along with the number of nodes per element to determine the type of element for carrying out analysis based on the permitted degrees of freedom at each node. For example, two types of elements are possible with three nodes: either the CST Element or the DKT plate element. The type of element used is determined by considering the degrees of freedom at each node. Method `getsdf()` returns the total degrees of freedom at each node. Method `getoption()` contains information which is used by the `Analyze` class to determine whether the problem is a plane stress or a plane strain condition.

`Joint` Class

This class represents a joint in the structure. An instance of this class is created for each node in the structure by calling the class constructor. The input to the constructor are node number and x and y coordinates of the joint. These objects are used by the `Analyze` class for computing the stiffness matrix of each element. Methods `getx(), gety(),` and `getnodes()` of this class return the x coordinate, y coordinate, and node number, respectively.

`Material` Class

Material properties such as Young's modulus and Poisson's ratio are required for evaluating the material matrix [D] for either plane stress or plane strain condition. The `Material` class is used to create objects of this class representing material properties of

each element and these are stored as a vector in `Model` class. The methods `getym()` and `getpr()` return Young's modulus and Poisson's ratio of the material. The method `getAth()` returns the section properties such as the thickness and area of cross-section of the elements.

`Restraint` Class

Finite element analysis essentially requires the solution of a large number of equations. The number of equations to be solved depends on the total number of restraints and the number of degrees of freedom for the entire structure. The number of restraints at each node depends on the degrees of freedom allowed at each node. The `Restraint` class represents restraints in the structure and is used to create objects which are used by the `Analyze` class. Further, these objects are also used by the `Result` class for displaying results. The methods `getrestloc()`, `getrestx()`, and `getresty()` can be called to determine whether there is a restraint in the x or y direction at any given node.

`Element` Class

To assemble the stiffness matrix, connectivity details of each element are required. The `Element` class contains details such as connectivity, element number and element type. Methods `getconnect()`, `getelemno()`, and `getelemtype()` return the array of connectivity, element number, and element type, respectively. The `Analyze` class calls an instance of the `Element` class to get the node connectivity array for the element. The coordinates of each node are obtained by calling the methods defined in the `Joint` class for each node. Once the details of the element are obtained, the stiffness matrix of the element is generated by calling the corresponding element class such as `Qelement` and `DKTElement` by passing the coordinates and material

properties of the element as arguments to the constructor. The class hierarchy structure of the `Element` class is given in Figure 7.2

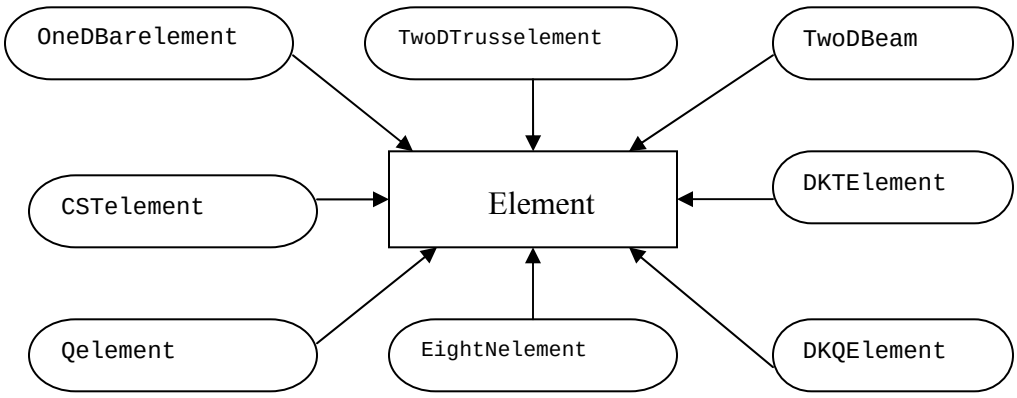

Figure 7.2 `Element` class hierarchy diagram.

`JointLoad` Class

The global load vector is formed by calling the methods of the `JointLoad` class to compute the applied force or moment acting at each node. The `getload()` and `getloadpos()` methods provide the magnitude of the applied force and the node number, respectively. The load vector assembled is used for calculating nodal displacements.

`ULoad` Class

Uniform or pressure loads acting on the elements are converted into equivalent nodal forces. The `ULoad` class represents uniform or pressure loads. The methods `getelno()`, `getstartpos()`, and `getendpos()` are used to obtain the element number and the location of the loads on the element. The uniform loads are stored in a vector `utloaddata` of `Model` class. The `Analyze` class calls the corresponding element

class such as `OneDBarelement`, `CSTelement` and passes information regarding load magnitudes and location needed to convert uniform loads into equivalent nodal loads. The loads evaluated are added to existing loads for performing analysis.

`Model` Class

As mentioned previously, this class serves as a container for storage of the finite element model. Objects of all structural component classes previously defined such as degrees of freedom, nodal coordinates, material properties, restraints, element nodal connectivity, and joint loads are stored as instances of the `Vector` class.

The `Vector` class belongs to the Java class library. A `Vector` is a dynamic array of objects. The components of a Vector can be accessed by using an integer index. Moreover, a `Vector` has a unique property which allows it to add or remove items after the `Vector` has been created. There are a set of well defined methods to manipulate a `Vector`. The dynamic resizing property of `Vector` allows the programmer to create storage containers for the objects without any limitations regarding capacity. Method `addElement()`adds a specified object to the end of a `Vector.` To remove an object from a specified location in a `Vector`, the `removeElementAt()` method is called by passing an index of the location of the object as an argument.

The `Model` class consists of various instances of the `Vector` class. Vectors such as `elementdata`, `nodedata`, `jointloaddata`, `restraintdata`, `materialprop`, and `systemdata` are used to store objects of classes `Element`, `Joint`, `JointLoad`, `Restraint`, `Material`, and `SystemProp`, respectively. Hence all of the components required for performing finite element analysis of a structure are stored in `Vectors` instantiated by `Model` class.

The methods for manipulating the data stored in `Vectors` are also defined in the `Model` class. The addition and removal of any object from `Vector` is done by the `addElementAt()` or `removeElementAt()` methods defined previously. A summary of methods in the `Model` class is given in Table 7.1. A class diagram depicting the relationship among the various classes and the `Model` class is given in Figure 7.3.

Table 7.1 Methods in `Model` class .

Method	Description
`addelem()`	Creates an instance of the `Element` class and adds it to `elementdata Vector`
`removeelem()`	Removes a object of the `Element` class from `elementdata Vector`
`addnode()`	Creates an instance of the `Joint` class and adds it to `jointdata Vector`
`removenode()`	Removes an object of the `Joint` class from `jointdata Vector`
`addload()`	Creates an instance of the `JointLoad` class and adds it to `jointloaddata Vector`
`removeload()`	Removes an object of `JointLoad` class from `jointloaddata Vector`
`addrestraint()`	Creates an instance of the `Restraint` class and adds it to `restraintdata Vector`
`removerestraint()`	Removes an object of `Restraint` class from `restraintdata Vector`
`addmaterial()`	Creates an instance of the `Material` class and adds it to `materialdata Vector`
`removematerial()`	Removes an object of `Material` class from `materialdata Vector`
`addsystem()`	Creates an instance of the `SystemProp` class and adds this to `systemdata Vector`

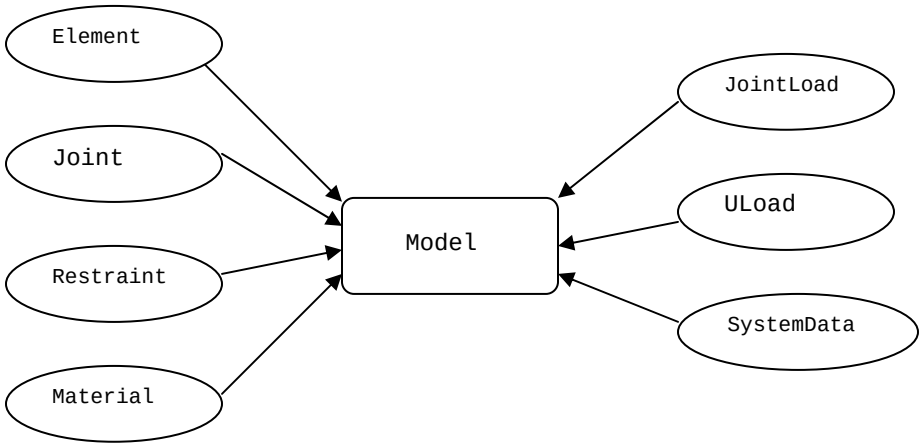

Figure 7.3 `Model` class diagram.

`Analyze` Class

This class utilizes all of the previously defined classes and performs the analysis of the structure. An instance of this class is called by the `FemApplet` class. Once this class is instantiated it uses the data stored in the `Model` object for calculating stiffness matrices, displacements, and stresses at each node. Each of the above mentioned task is carried out by methods defined in this class, which are listed in Table 7.2.

Table 7.2 Methods in `Analyze` class

Method	Description
`analysisdata()`	Extracts data required for analysis from the `Model` object
`stiffnessmatrix()`	Calculates stiffness matrix based on element type
`calcdisp()`	Calculates displacement at each node
`stress()`	Calculates stresses in each element
`Calckpstress()`	Calculates material matrix for plane stress condition
`Calckpstrain()`	Calculates material matrix for plane strain condition

The method `analysisdata()` is used to obtain analysis data stored in the form of vectors in the `Model` object. This method further calculates the number of restraints and total number of degrees of freedom for the structure, and assigns equation numbers for assembling the stiffness matrix. The `stiffnessmatrix()` method uses the number of elements obtained from `analysisdata()` for assembling the global stiffness matrix by performing the following the operations for each element of the structure.

- Connectivity details of the element are obtained.
- Nodal coordinates are obtained by using the methods of the `Joint` class for each node of the element.
- The stiffness matrix of the element is generated by calling an instance of the particular element such as `CSTelement` and `DKTElement` by passing the nodal coordinates and material properties of the element as arguments.
- The global stiffness matrix is obtained by adding the element stiffness matrix using the equation numbers generated earlier.

Deflections at each node are then calculated in the `calcdisp()` method by inverting the structural stiffness matrix and multiplying it by the global load vector. The matrix operations are performed using the *Jama* Library class library developed by NIST (National Institute of Standards and Technology). The stresses in the element are evaluated by calling the `stress()` method which instantiates the specific element class and then calls the `calcstress()` method of that class to calculate stress by passing the nodal displacements and the connectivity array as arguments. The stresses are stored in vectors defined in the `Stresses` class. Methods `calcpkstress()` and `calcpkstrain()` are used to calculate the [D] matrix based on the analysis option chosen. Figure 7.4 shows the class diagram for the `Analyze` class.

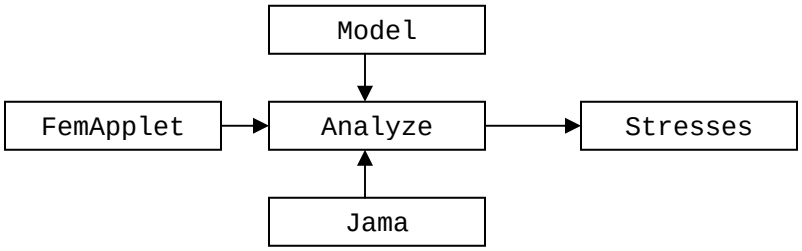

Figure 7.4 `Analyze` class diagram.

`AxStress` Class

This class contains methods for storing calculated axial stresses in the `OneDBarelement` and `TwoDTrusselement` classes. Methods `gets1()` and `getf()` are used to get the axial stress and force in each element while printing results. The `AxStress` class objects are stored in the `axstress` vector of `Stresses` class.

`MemStress` Class

Membrane stresses are obtained when the structure is subjected to inplane loading. Elements such as `CSTelement`, `Qelement`, and `EightNelement` classes yield membrane stresses on loading. The `MemStress` class stores normal stresses in the x and y directions, shear stress, maximum and minimum principal stresses and the angle of the principal axis at each node. The methods of the `MemStress` class are listed in Table 7.3.

Table 7.3 Methods in class `MemStress`

Method	Description
`getnodenum()`	Returns node number at which stresses are evaluated
`gets1()`	Returns normal stress in x direction at the given node
`gets2()`	Returns normal stress in y direction at the given node
`gets12()`	Returns shear stress at the given node
`getsmax()`	Returns maximum principal stress at the node
`getsmin()`	Returns minimum principal stress at the node
`getsang()`	Returns angle of principal axis at the node

BenStress class

This class is used for storing stresses when plate bending elements are used. This class stores the normal stresses in the x and y directions, shear stress, maximum and minimum principal stresses, and angle of the principal axis at each node for plate bending problems. The methods defined in this class are listed in Table 7.4

Table 7.4 Methods in class `BenStress`

Method	Description
`getnodenum()`	Returns node number at which stresses are evaluated
`gets1()`	Returns normal stress in x direction at the given node
`gets2()`	Returns normal stress in y direction at the given node
`gets12()`	Returns shear stress at the given node
`getsmax()`	Returns maximum principal stress at the node
`getsmin()`	Returns minimum principal stress at the node
`getsang()`	Returns angle of principal axis at the node

`Stresses` Class

Objects of classes `AxStress`, `MemStress`, and `BenStress` are organized and stored in `axstress`, `memstress`, and `benstress` vectors, respectively, by this class. The `Analyze` class calls an instance of each element class such as `CSTelement` and `DKTElement` for calculating stresses. The `Calcstress()` method defined for each element takes an instance of the `Stresses` class as an argument along with the connectivity and nodal displacements arrays of the element. The evaluated stresses at each node are then stored in vectors as objects of `AxStress`, `MemStress` or `BenStress` based on the type of element used for analysis. Table 7.5 lists the methods defined in the

`Stresses` class. The relationship between the `Stresses` class and other classes is shown in Figure 7.5.

Table 7.5 Methods in `Stresses` class

Method	Description
`addAxialStress()`	Creates an instance of the `AxStress` class and adds this to the `axstress` vector
`addMemStress()`	Creates an instance of `MemStress` class and adds this to the `memstress` vector
`addBendingStress()`	Creates an instance of `BenStress` class and adds this to the `benstress` vector

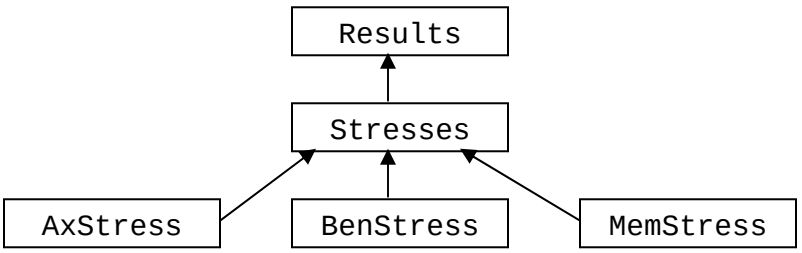

Figure 7.5 `Stresses` class diagram.

7.5 Applet Interface

The main window of the applet which is derived from the `JApplet` class is shown in Figure 7.6. A menu bar is provided with the **File**, **Model**, **Analysis**, and **Help** menus. The **Model** menu contains the **Input** and **Draw** commands. The **Input** command is discussed in section 7.7. When **Run** is selected from the **Analysis** menu, an instance of

the `Analyze` class is created for performing the analysis of structure. The **Results** command instantiates the `Result` class which displays the results on the Results frame once the analysis is completed.

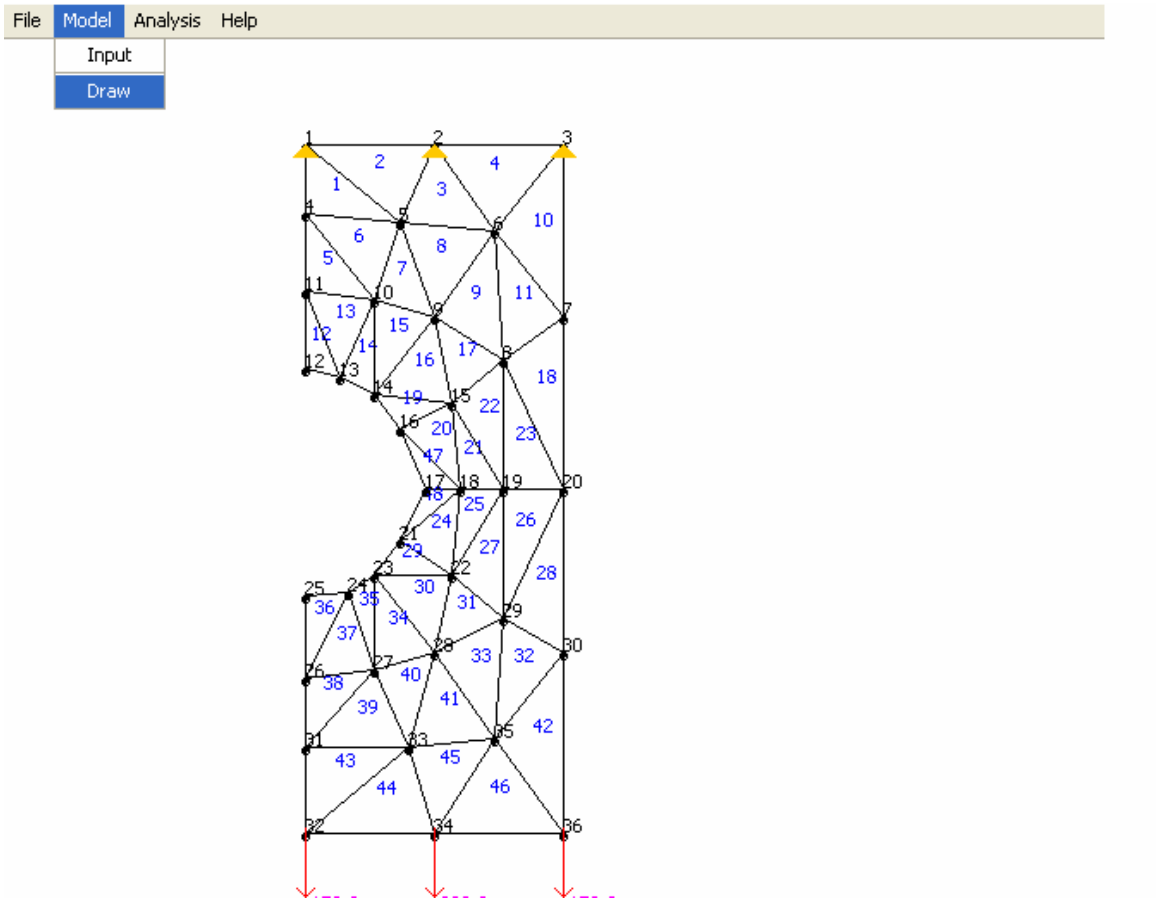

Figure 7.6 Finite element analysis applet view.

7.6 Interface Classes

The graphical user interface of the applet is mainly provided by the `FemApplet` class. The `Draw` class displays the structure. The deformed shape of the structure under loading is plotted by the `Deform` class.

`FemApplet` Class

`FemApplet` class is created by extending the `JApplet` class of the Java class library. The size of applet is defined as 640 by 520 pixels for compatibility with monitors of any resolution. This class contains all the required code for controlling the behavior of the components defined in it. Various components such as `JTextBox`, `JLabel`, `JList`, `JScrollPane`, `JTextArea`, `JButton`, `JCheckBox`, and `JRadioButton` are used for developing the user interface for manual input. Several inner classes are provided for event handling operations of the above mentioned components.

`Draw` Class

This class draws the structure based on the input data. The user can make changes to the input and modify the structure until the desired model of the structure is created. The `Draw` class scales the diagram so as to fit in the applet. This class contains methods for displaying the structure, concentrated loads, uniform loads, and restraints. Each element is numbered along with the nodes as given in the input. In short, it creates the graphical representation of the structure being analyzed.

`Deform` class

The `Deform` class generates the deformed shape of the structure. The deformed shape of the structure helps in visualizing the response of the structure to loading. The addition of the `Deform` class is to make the applet more interactive.

7.7 Input

The input for this applet is divided into three steps. The first step is similar to the pre-processing step of commercial finite element programs where input for the structure is entered. Two types of input are possible for this program: a) manual input and b) text

input. The user has the choice of making changes to the structure input. Once the user is done with input, the structure is analyzed. To view results such as displacements at various nodes and stresses in each element, the Results frame is activated. The results can also be saved by copying them to a text file.

7.7.1 Manual Input

The manual input frame essentially consists of five tabs. The input frame of the applet is activated by selecting the **Input** command from the **Model** menu. This leads to the input area shown in Figure 7.7 where the element type, coordinates of nodes, connectivity information, restraints, and loads acting at various nodes are entered. Each input tab is discussed in detail in the following paragraphs.

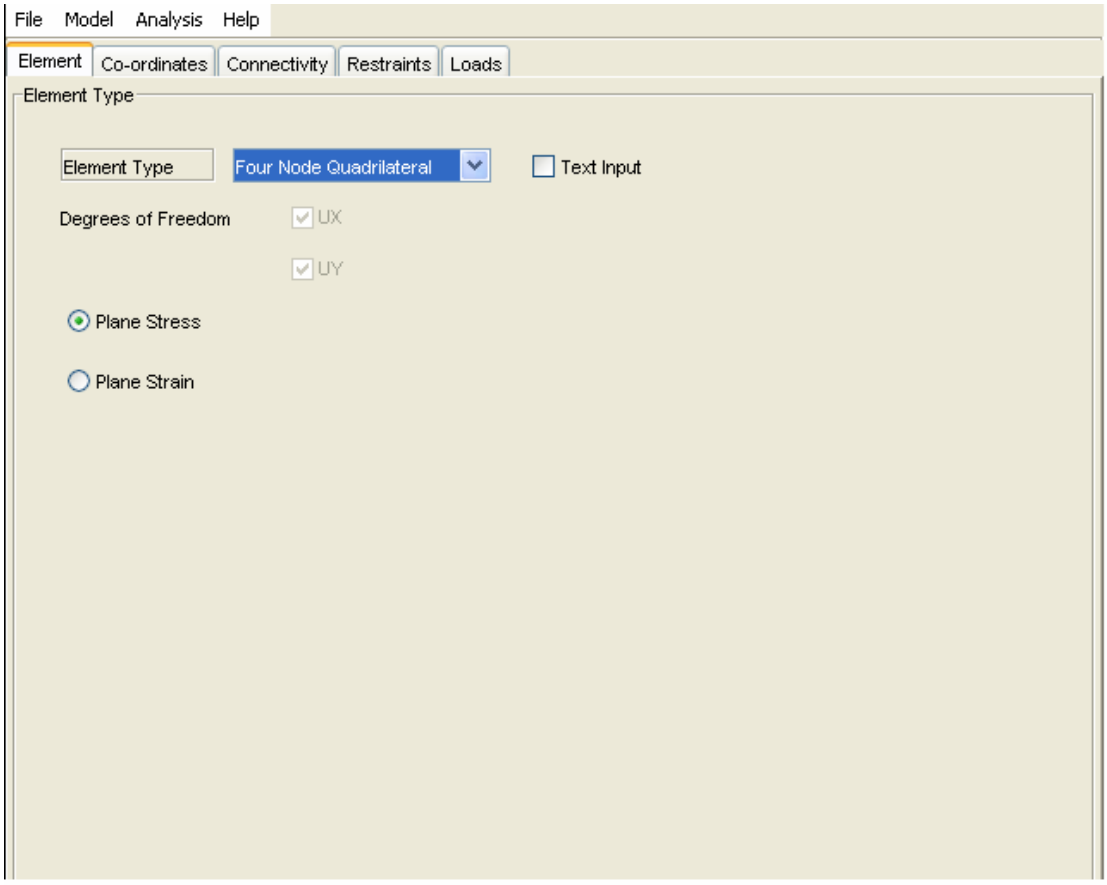

Figure 7.7 Element tab.

Element Type tab: Input regarding the type of element to be used for modeling the structure, degrees of freedom possible, and the method of input are entered in this tab. The options for plane stress or plane strain are activated when the selected element type is a membrane or plate element. The Element Type combo list gives the user the option of selecting from one-dimensional bar element, two-dimensional truss element, two-dimensional beam element, CST element, four-node iso-parametric quadrilateral element, eight-node iso-parametric quadrilateral element, Discrete Kirchhoff triangular plate element, and Discrete Kirchhoff quadrilateral plate element. The Element Type tab is shown in Figure 7.7

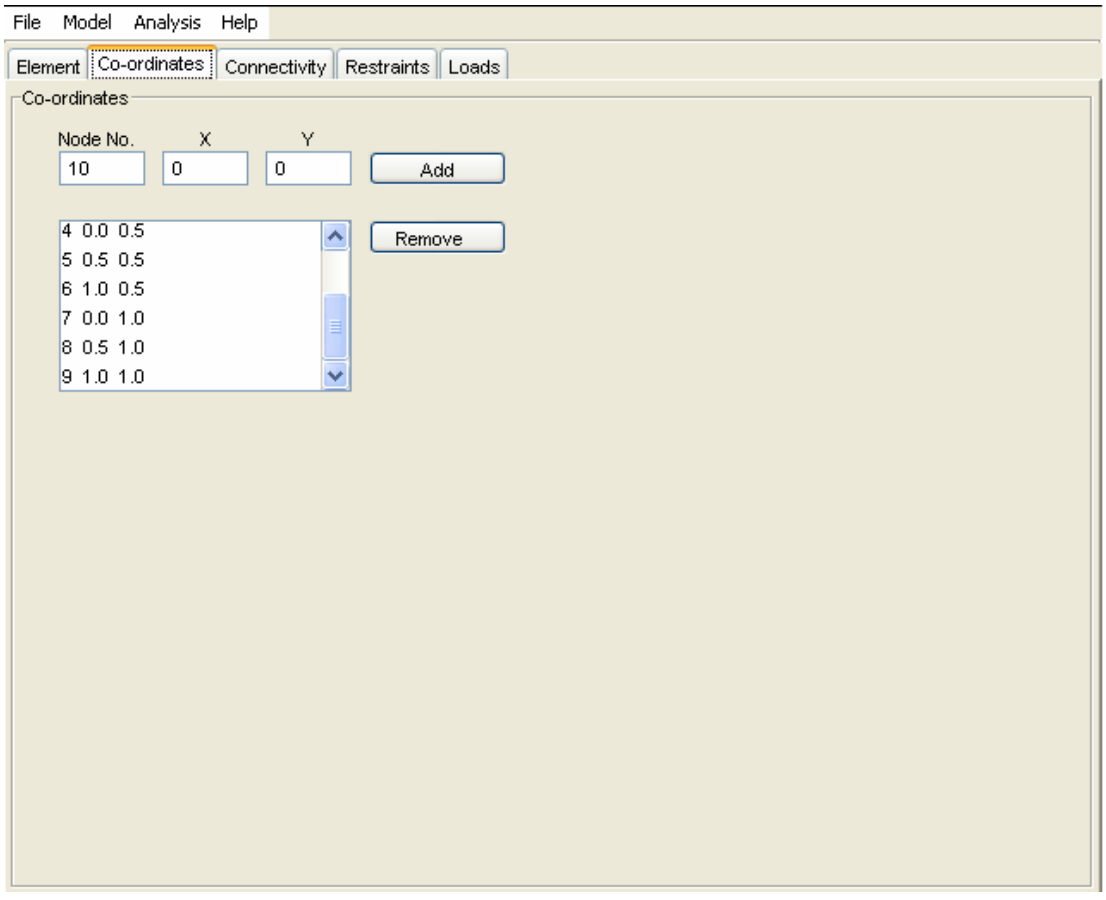

Figure 7.8 Co-ordinates tab.

Co-ordinates Tab: Each node in the structure is defined by a set of X and Y coordinates. All coordinates entered are to be specified in the global system. To add a node, the node number and X and Y text fields are filled with respective values and the **Add** command button is pressed. This action displays the previously entered values in a list box as shown in Figure 7.8. To remove a node, the corresponding node is selected and the **Remove** button is pressed. There are no limitations on the number of nodes that can be contained in a structure.

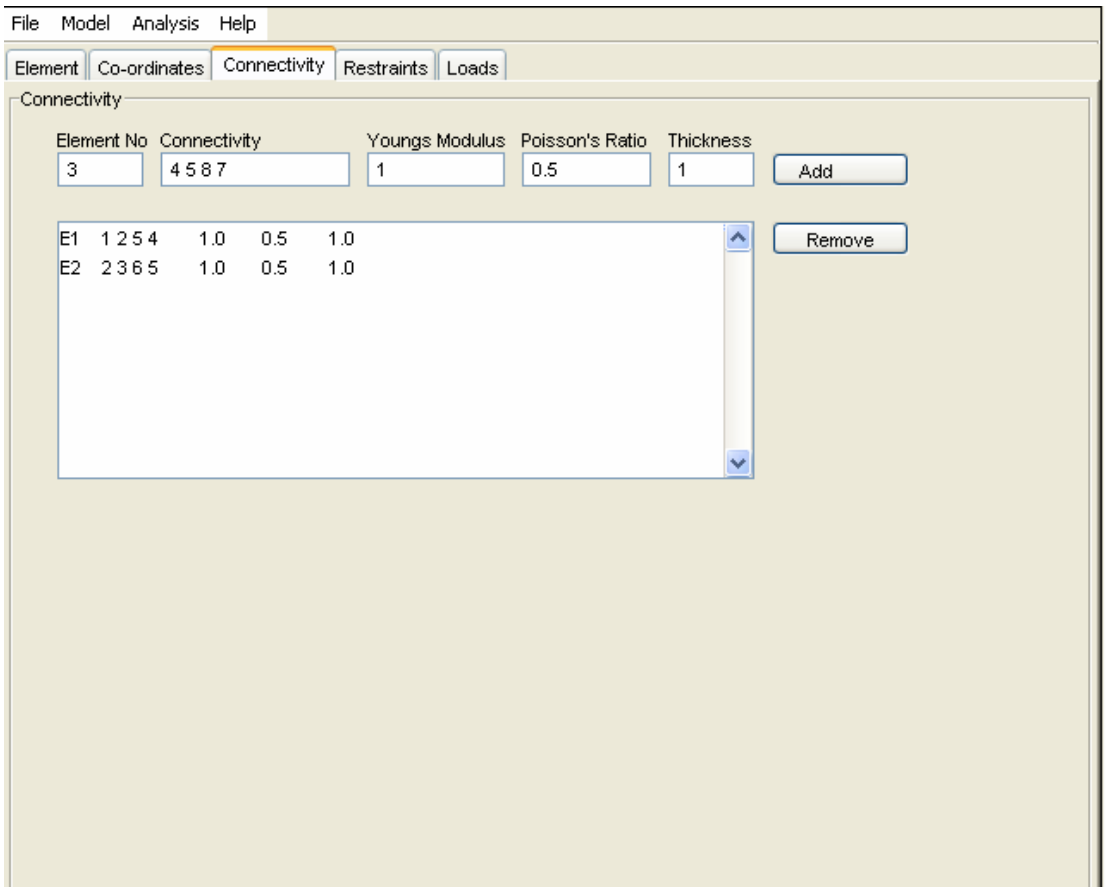

Figure 7.9 Connectivity tab.

Connectivity Tab: Each element is connected to a set of nodes. The number of nodes to which each element is connected depends on the type of element. Connectivity information is required for calculating the stiffness matrix and stresses at each node. Here, along with the connectivity information, the material properties of element such as

Young's modulus, Poisson's ratio, and section properties like area or moment of inertia are entered. The **Add** and **Remove** buttons function similarly to those mentioned in the Co-ordinates tab. Figure 7.9 illustrates the Connectivity tab.

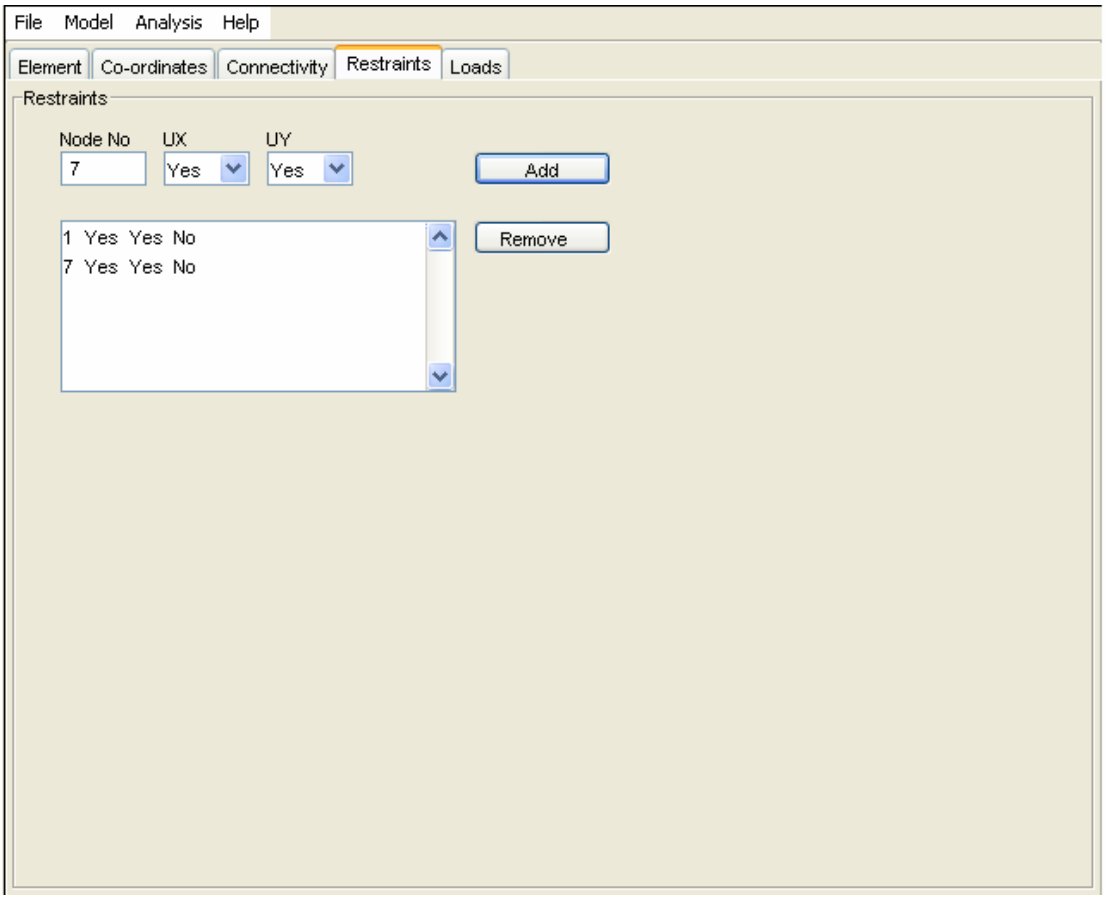

Figure 7.10 Restraint tab.

Restraint Tab: The restraints are entered in this tab. The node number of the restraint, along with the direction in which it is restrained, are entered. Figure 7.10 shows the Restraint tab.

Load Tab: The functioning of *Load Tab* is similar to the Restraint Tab where, instead of restraints in the X and Y directions, loads acting in the X and Y direction are specified for nodal loads. Details of the input for uniform loads are given in Appendix A.

7.7.2 Text Input

When a structure with a large number of nodes is analyzed, it is more convenient to provide input in text format. Generally, an input text file is created and is read by the program. But in Java an applet cannot read input from a file due to security reasons. Hence, the input text is copied to the clipboard and then pasted to the text input field which is activated when the Text input option is selected on the Element Type tab. The format of input text is specified in Appendix A. Figure 7.11 illustrates the use of the input text field.

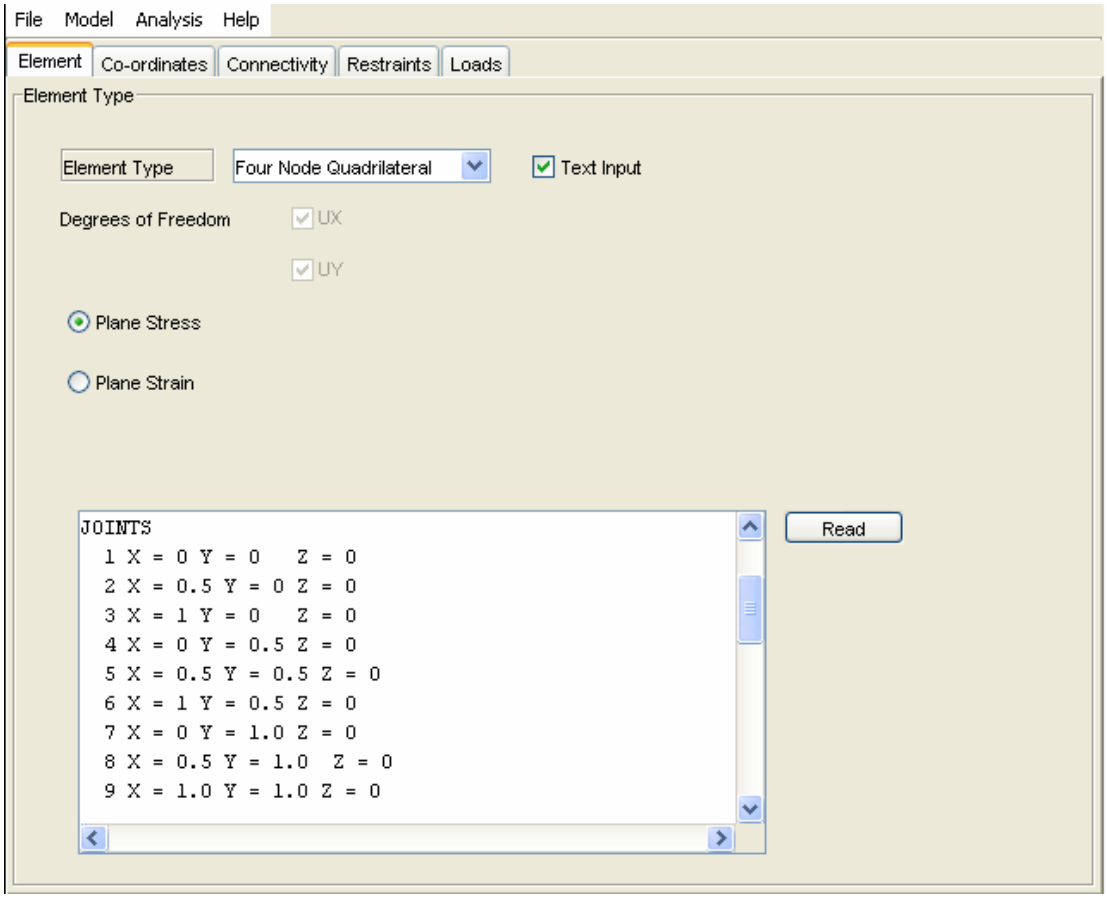

Figure 7.11 Text input field.

Chapter 8: Test Problems and Verification of Results

8.1 Introduction

In this chapter the deflections and stresses obtained at selected nodes from the applet for test problems are compared to those obtained from SAP 2000, a commercial finite element analysis package developed by Computers and Structures Inc. (CSI), Berkeley. All units involve pounds and inches.

8.2 Test Problems for One-dimensional Bars and Truss

Two test examples were considered and the results obtained from the applet are compared to those obtained from SAP 2000. Deflections at critical nodes, forces in elements, and reactions are compared.

Problem 1: The first problem is a fixed beam of length 30 in. with three equal segments (Sennett 2000). The structure consists of four nodes and three elements. The edges are fixed at nodes 1 and 4 and loads are applied at nodes 2 and 3. Figure 8.1 shows the finite element model of the beam.

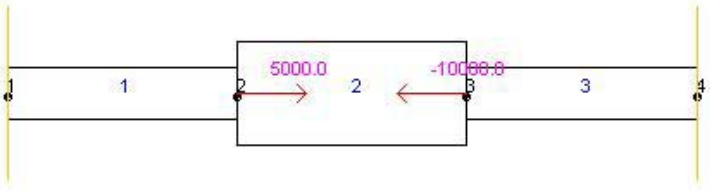

Figure 8.1 Model of Problem 1 - Fixed beam

Geometric Data

Span 1: Length = 10 in., area = 1 in^2.

Span 2: Length = 10 in., area = 2 in^2.

Span 3: Length = 10 in., area = 1 in^2.

Material Properties

E = 10 x 10^6 psi

Loading

A concentrated load of 5000 lb at node 2

A concentrated load of -10000 lb at node 3

Restraints

Nodes 1 and 4 are restrained in the X direction

Comparison of Results

The results obtained from the applet and SAP 2000 are shown in Table 8.1. The results compared are displacements at nodes 2 and 3 and reactions at nodes 1 and 4. It is observed that the results obtained are identical to those obtained from SAP 2000. Table 8.1 also shows the forces in elements 1 and 3.

Table 8.1 Displacements, reactions and forces for Problem 1

Location	Parameter	Result from Applet	Result from SAP 2000	Percent error
Node 2	**UX**	-0.0010	-0.0010	**0.00**
Node 3	**UX**	-0.0040	-0.0040	**0.00**
Node 1	**RX**	1000.0000	1000.0000	**0.00**
Node 4	**RX**	4000.0000	4000.0000	**0.00**
Element 1	**FX**	-1000.0000	-1000.0000	**0.00**
Element 3	**FX**	4000.0000	4000.0000	**0.00**

Problem 2: The second test problem considered is a thin tapered steel plate of 1 in. uniform thickness subjected to its self weight in addition to a concentrated load at the midpoint (Chandrupatla et al. 1997). The width of the plate at the top and bottom are 6 in. and 3 in., respectively, and the total length of the plate is 24 in. The plate is modeled with two one-dimensional bar elements of 12 in. length. The finite element model is shown in Figure 8.2

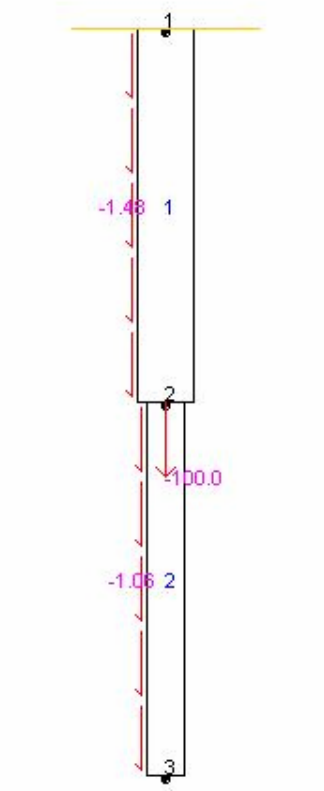

Figure 8.2 Plate modeled with two one-dimensional elements.

Geometric Data

Element 1: Length = 12 in., area = 5.25 in^2.

Element 2: Length = 12 in., area = 3.75 in^2.

Material Properties

E = 30 x 10^6 psi

Loading

A concentrated load of 100 lb at node 2

Self-weight per unit length of magnitude -1.48 lb/in. on element 1

Self-weight per unit length of magnitude -1.06 lb/in. on element 2

Restraints

Node 1 is restrained in the Y direction

Comparison of Results

Displacements at nodes 1, 2, and 3, and forces in elements 1 and 2 obtained from SAP 2000 and from the applet are the same. Table 8.2 illustrates the comparison of results.

Table 8.2 Displacements, reactions and forces for Problem 2

Location	Parameter	Result from Applet	Result from SAP 2000	Percent Error
Node 2	**UY**	-9.26476E-6	-9.26E-06	**0.00**
Node 3	**UY**	-9.94316E-6	-9.94E-06	**0.00**
Node 1	**RY**	130.480	130.480	**0.00**
Element 1	**FY**	121.600	121.600	**0.00**
Element 2	**FY**	6.36000	6.3600	**0.00**

The deformed shape of the plate generated by the applet is shown in Figure 8.3.

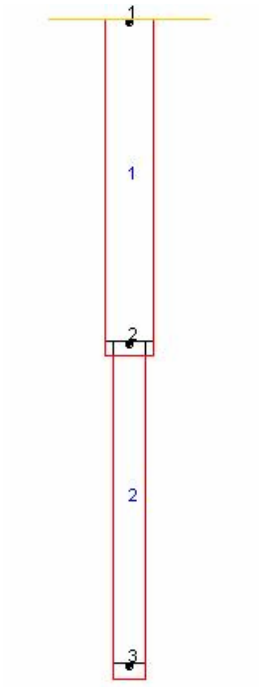

Figure 8.3 Deformed shape of structure of Problem 2.

8.3 Test Problems for Two-dimensional Truss

Two test examples were considered to verify the accuracy of the results obtained from the trusses. The results from the applet are compared to those obtained from SAP 2000. Deflections at selected nodes and reactions are compared.

Problem 3: A four bar determinate truss is analyzed (Chandrupatla et al. 1997). The cross section area of all truss members is 1 in.^{2}. The finite element model of the truss is shown in Figure 8.4.

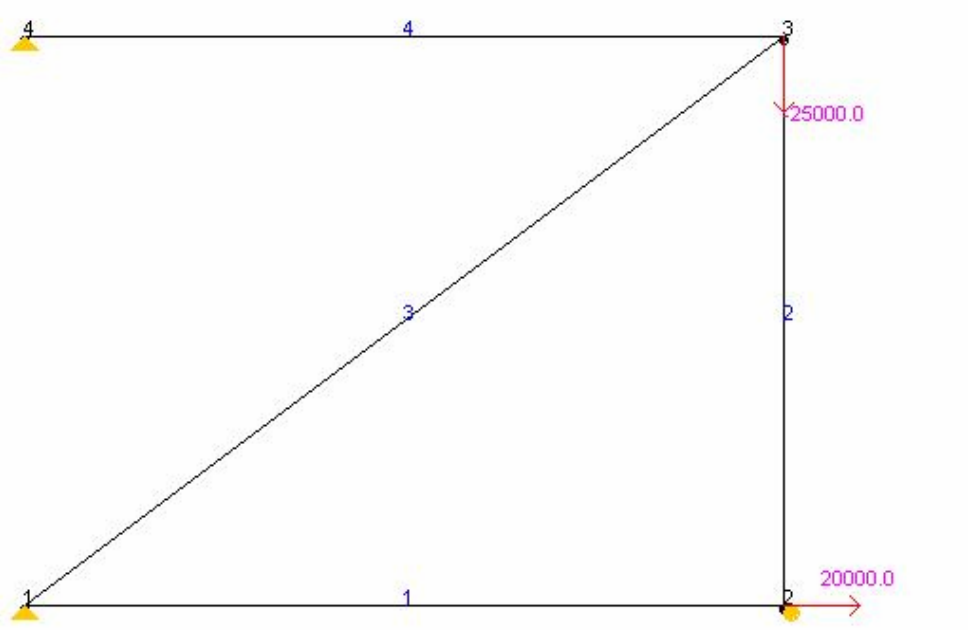

Figure 8.4 Truss model for Problem 3.

Geometric Data

Element 1: Length = 40 in., area = 1 in^2.

Element 2: Length = 30 in., area = 1 in^2.

Element 3: Length = 50 in., area = 1 in^2.

Element 4: Length = 40 in., area = 1 in^2.

Material Properties

E = 29.5 x 10^6 psi

Loading

A concentrated load of 20000 lb is applied at node 2 in the X direction.

A concentrated load of 25000 lb is applied at node 3 in the Y direction.

Restraints

Nodes 1 and 4 are restrained in both the X and Y directions

Node 2 is restrained in the Y direction

Comparison of Results

The displacements at nodes 2 and 3, and the reactions at nodes 1 and 2, are compared. Table 8.3 shows the comparison. It can be seen that the results obtained are the same as those from SAP 2000. The deformed shape of the truss from the applet is shown in Figure 8.5.

Table 8.3 Displacements and reactions for Problem 3

Location	Parameter	Result from Applet	Result from SAP 2000	Percent error
Node 2	**UX**	0.0271	0.0271	**0.00**
Node 3	**UX**	0.0056	0.0056	**0.00**
	UY	-0.0222	-0.0222	**0.00**
Node 1	**RX**	-15833.3000	-15833.3000	**0.00**
	RY	3125.0000	3125.0000	**0.00**
Node 2	**RY**	21875.0000	21875.0000	**0.00**

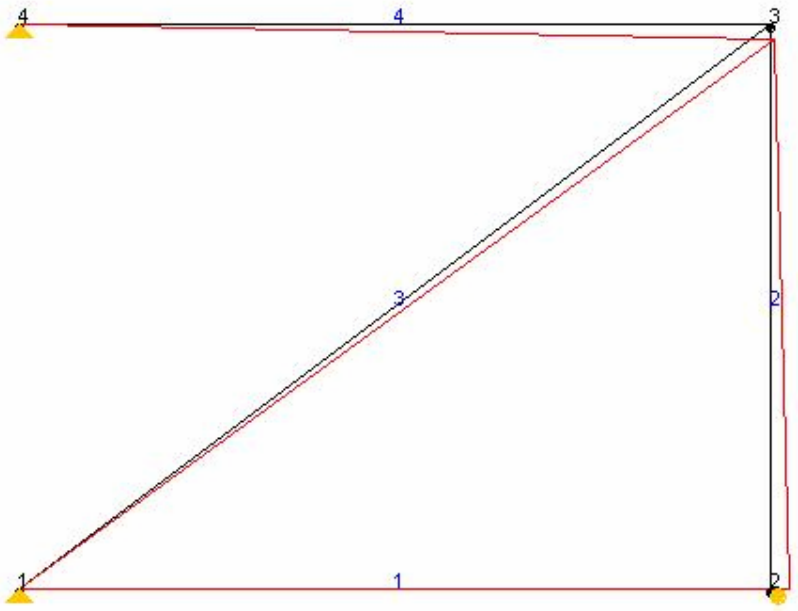

Figure 8.5 Deformed shape of truss of Problem 3.

Problem 4: An indeterminate truss problem (Kassimali 1999) as illustrated in Figure 8.6 is analyzed. The area of all truss members is 6 in.^{2}

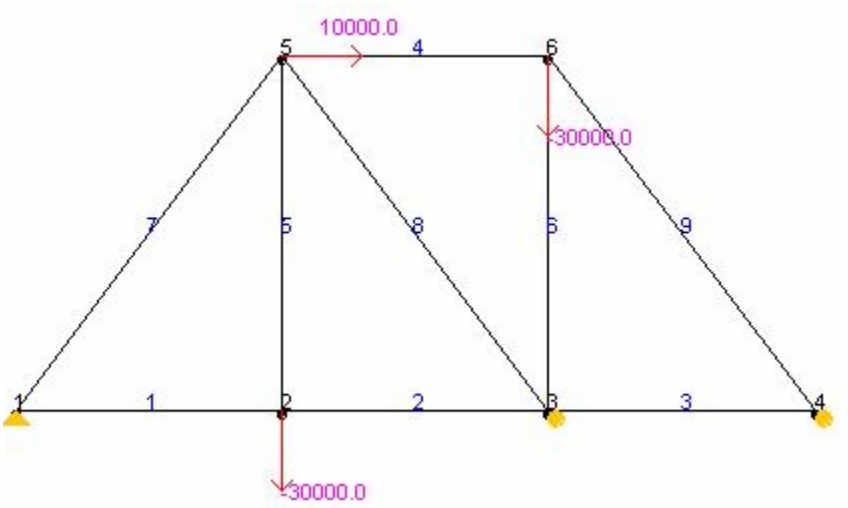

Figure 8.6 Truss model for Problem 4.

Geometric Data

The truss geometry is shown in Figure 8.6. The truss spans 45 ft and is 20 ft in height.

Material Properties

E = 29 x 10^6 psi

Loading

A concentrated load of 10000 lb is applied at node 5 in the X direction.

A concentrated load of 30000 lb is applied at nodes 2 and 6 in the Y direction.

Restraints

Node 1 is restrained in both the X and Y directions.

Nodes 3 and 4 are restrained in the Y direction.

Comparison of Results

The displacements at nodes 2, 5, and 6 and reactions at nodes 1 and 4 are presented in Table 8.4. The results obtained from the applet are the same as those obtained from SAP 2000. The deformed shape of the structure is shown in Figure 8.7.

Table 8.4 Displacements and reactions for Problem 4

Location	Parameter	Result from Applet	Result from SAP 2000	Percent error
Node 2	**UX**	0.0194	0.0194	**0.00**
	UY	-0.0964	-0.0964	**0.00**
Node 5	**UX**	0.0312	0.0312	**0.00**
	UY	-0.0560	-0.0560	**0.00**
Node 6	**UX**	0.0259	0.0259	**0.00**
	UY	-0.0320	-0.0320	**0.00**
Node 1	**RX**	-10000.000000	-10000.000000	**0.00**
	RY	12032.0000	12032.0000	**0.00**
Node 4	**RY**	6997.2800	6997.2800	**0.00**

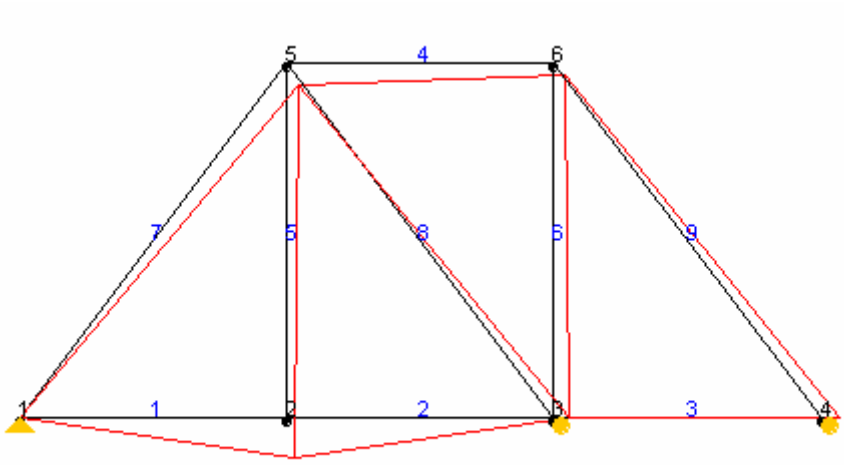

Figure 8.7 Deformed shape of truss in Problem 4.

8.4 Test Problems for Two-dimensional Beams and Frames

Two test examples were considered to verify the accuracy of the applet for two-dimensional beams and frames. The results obtained from the applet are compared to those obtained from SAP 2000. The first example is an indeterminate beam and the second example is a frame structure. Deflections at selected nodes and reactions are compared.

Problem 5: The two span continuous beam as illustrated in Figure 8.8 is analyzed (Sennett 2000). The beam model consists of four nodes. The supports at nodes 1 and 4 are fixed and node 2 is a hinged support.

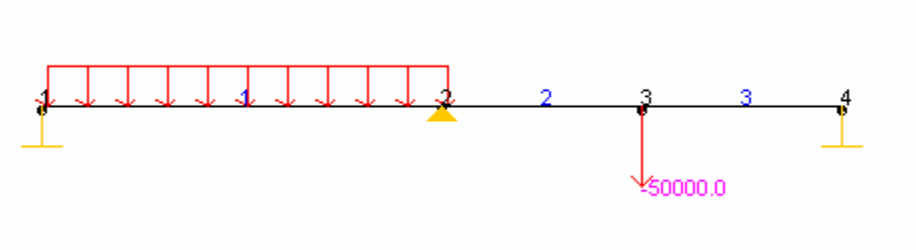

Figure 8.8 Model of beam for Problem 5.

Geometric Data

Element 1: Length = 360 in., I = 233 in^4.

Element 2: Length = 540 in., I = 233 in^4.

Element 3: Length = 720 in., I = 233 in^4.

Material Properties

E = 29 x 10^6 psi

Loading

A concentrated load of 50000 lb is applied at node 3 in the Y direction.

A uniform load of 300 lb/in. is applied on span 1 in the Y direction.

Restraints

Nodes 1 and 4 are restrained against translation in the X, Y directions and against rotation in the Z direction.

Node 2 is restrained in the Y direction.

Comparison of Results

The rotation at node 2 and the reaction at nodes 1, 2, and 4 are compared to those obtained from SAP 2000. Table 8.5 shows the results of the comparison. It is clear that the results obtained from the applet are the same as those from SAP 2000. Figure 8.9 shows the deformed shape of beam under loading.

Table 8.5 Displacements and reactions for Problem 5

Location	Parameter	Result from Applet	Result from SAP 2000	Percent error
Node 2	**UR**	0.0066	0.0066	**0.00**
Node 1	**RY**	56062.5000	56062.5000	**0.00**
	MZ	3487500.0000	3487500.0000	**0.00**
Node 2	**RY**	79000.0000	79000.0000	**0.00**
Node 4	**RY**	22937.5000	22937.5000	**0.00**
	MZ	-2002500.0000	-2002500.0000	**0.00**

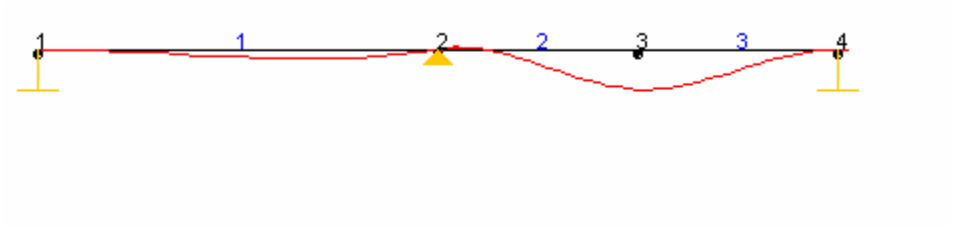

Figure 8.9 Deformed shape of Beam in Problem 5.

Problem 6: A portal frame is analyzed using two-dimensional frame elements. The structure is illustrated in Figure 8.10 (Chandrupatla et al. 1997).

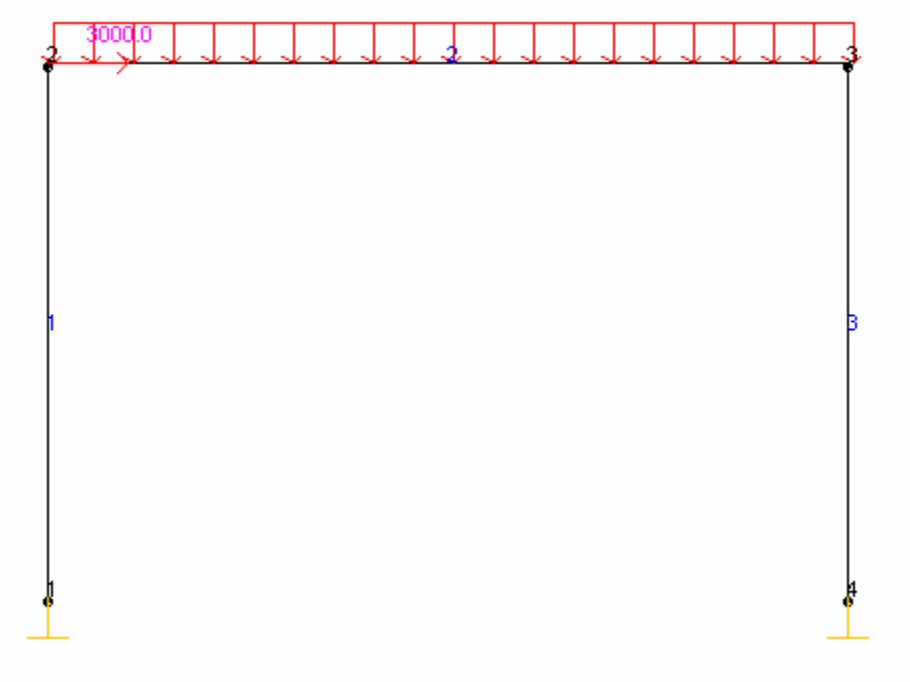

Figure 8.10 Model of portal frame of Problem 6.

Geometric Data

Element 1: Length = 96 in., I = 65 in.^{4}, area = 6.8 in.^{2}

Element 2: Length = 144 in., I = 65 in.^{4}, area = 6.8 in.^{2}

Element 3: Length = 96 in., A = 65 in.^{4}, area = 6.8 in.^{2}

Material Properties

E = 30 x 10^6 psi

Loading

A concentrated load of 3000 lb is applied at node 2 in the X direction.

A uniform load of 42 lb/in. is applied on Element 2 from node 2 to node 3 in the Y direction.

Restraints

Nodes 1 and 4 are restrained against translation in the X, Y directions and against rotation in the Z direction.

Comparison of Results

Displacements at nodes 2 and 3 and support reactions at node 1 are compared. Table 8.6 shows the comparison. It can be seen that results obtained from the applet and from SAP 2000 are in perfect agreement. The deformed shape of the frame is shown in Figure 8.11.

Table 8.6 Displacements and reactions for Problem 6

Location	Parameter	Result from Applet	Result from SAP 2000	Percent error
Node 2	**UX**	0.091766	0.091766	**0.00**
	UY	-0.001036	-0.001036	**0.00**
	UR	-0.001387	-0.001387	**0.00**
Node 3	**UX**	0.090119	0.090119	**0.00**
	UY	-0.001788	-0.001788	**0.00**
	UR	-0.000039	-0.000039	**0.00**
Node 1	**RX**	-665.776000	-665.776150	**0.00**
	RY	2201.200000	2201.2020	**0.00**
	MZ	60138.300000	60138.3110	**0.00**

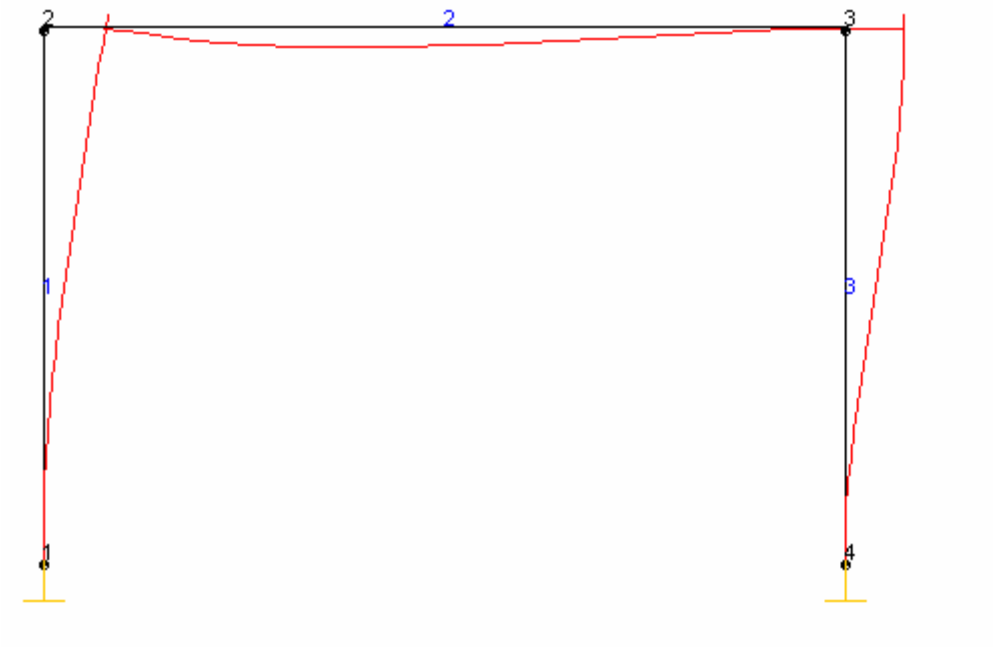

Figure 8.11 Deformed shape of portal frame of Problem 6.

8.4 Test Problems for Plane Stress Analysis

The accuracy of the plane stress elements developed in this applet is verified by considering five test problems. Standard test problems consisting of CST, four-node quadrilateral, and eight-node quadrilateral elements are analyzed using the applet and SAP 2000. Deflections and stresses at critical nodes are compared. The following section describes the finite element models for these test problems and presents a comparison of results.

Problem 7: A cantilever of length 50 in. and height 10 in. is modeled with 10 CST elements as shown in Figure 8.12. The thickness of the cantilever is 1 in. It is subjected to two concentrated loads of 20 kips each at nodes 6 and 12.

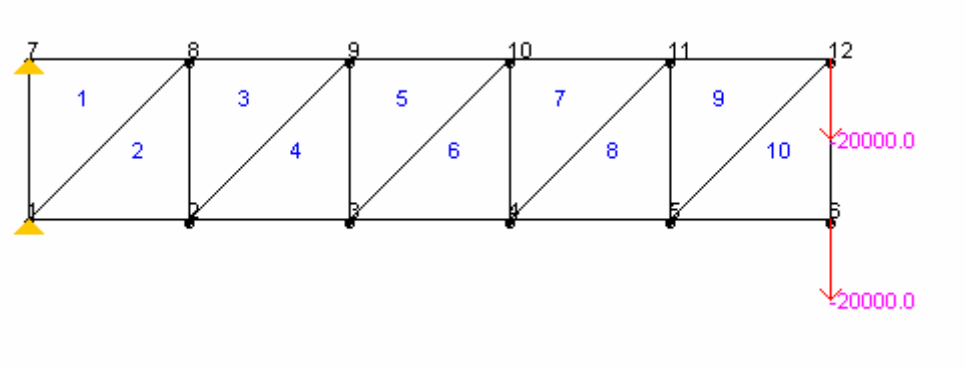

Figure 8.12 FE model for Problem 7.

Geometric Data

Length L = 50 in.

Height h = 10 in.

Thickness t = 1 in.

Material Properties

E = 30 x 10^6 psi.

Poisson's ratio ν = 0.25

Loading

A concentrated load of 20000 lb at node 6 in the Y direction

A concentrated load of 20000 lb is applied at node 12 in the Y direction

Restraints

Nodes 1 and 7 are restrained in both the X and Y directions.

Comparison of Results

The results obtained from the applet are compared to those obtained in SAP 2000 and are shown in Table 8.7. Displacements at node 6 and 12 and stresses at node 4 are compared. It is observed that the results obtained from the applet are identical to those obtained from SAP 2000.

Table 8.7 Displacements and stresses for Problem 7

Location	Parameter	Result from Applet	Result from SAP 2000	Percent error
Node 6	**UX**	-0.023541	-0.023541	**0.00**
	UY	-0.168199	-0.168199	**0.00**
Node 12	**UX**	0.022189	0.022189	**0.00**
	UY	-0.167592	-0.167592	**0.00**
Node 4	**S11**	-14545.700000	-14545.700000	**0.00**
	S22	-1818.350000	-1818.350000	**0.00**
	S12	1454.270000	1454.270000	**0.00**

The deformed shape of the structure is shown in Figure 8.12.

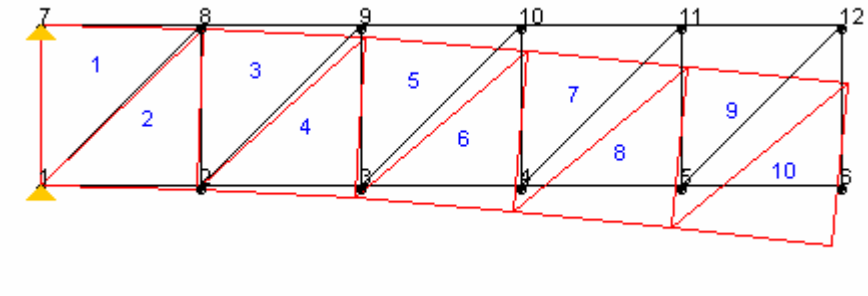

Figure 8.12 Deformed shape of the cantilever for Problem 7.

Problem 8: The same cantilever beam of Problem 7 is analyzed using four-node quadrilateral elements. The length of the beam is 50 in., depth is 10 in., and thickness is 1 in. A vertical load totaling 40000 lb is applied at the free end of the beam. The finite element mode has 33 nodes and 20 four-node quadrilateral elements as shown in Figure 8.13.

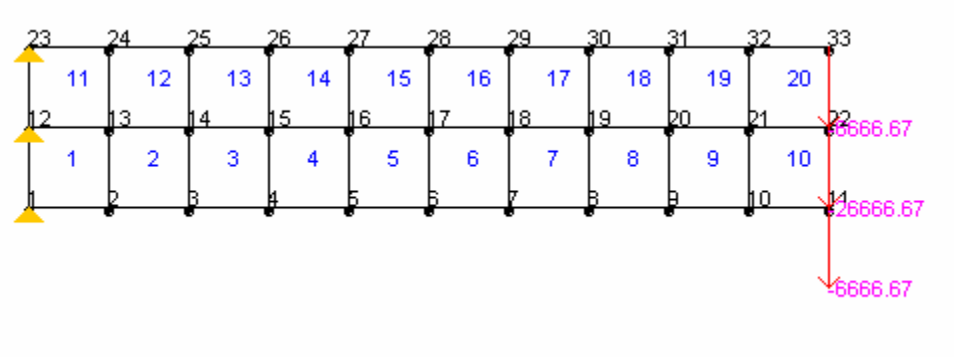

Figure 8.13 FE Model of cantilever with four-node quadrilateral elements

Geometric Data

Length L = 50 in.

Height h = 10 in.

Thickness t = 1 in.

Material Properties

E = 30 x 10^6 psi

Poisson's ratio ν = 0.25

Loading

A concentrated load of 6666.67 lb is applied at nodes 11 and 33 in the Y direction

A concentrated load of 26666.67 lb is applied at node 22 in the Y direction

Restraints

Nodes 1, 12, and 23 are restrained in the X and Y directions.

Comparison of Results

The displacements at nodes 11 and 33 and stresses at node 6 are shown in Table 8.9. The difference in results obtained from the applet and SAP 2000 is about 0.01 percent.

Table 8.8 Displacements and stresses for Problem 8

Location	Parameter	Result from Applet	Result from SAP 2000	Percent error
Node 11	UX	-0.089254	-0.089259	**0.01**
	UY	-0.609463	-0.609497	**0.01**
Node 33	UX	0.089254	0.089259	**0.01**
	UY	-0.609463	-0.609497	**0.01**
Node 6	S11	-61247.3000	-61250.791000	**0.01**
	S22	-8592.8300	-8593.309000	**0.01**
	S12	8084.910000	8085.594000	**0.01**
	S-MAX	-7379.3800	-7379.729	**0.00**
	S-MIN	-62460.8000	-62464.371	**0.01**

The deformed shape of the cantilever is shown in Figure 8.14

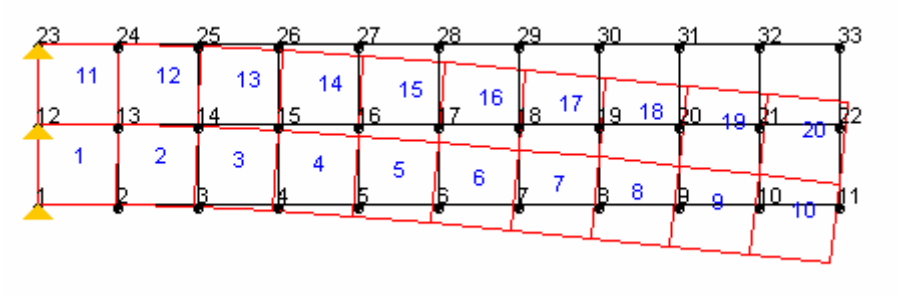

Figure 8.14 Deformed shape of cantilever for Problem 8

Problem 9: The cantilever beam shown in Figure 8.15 is analyzed using five eight-node quadrilateral elements. The length of the cantilever is 50 in., depth is 10 in., and thickness is 1 in. A concentrated load totaling 40000 lb acts at the end as shown in Figure 8.15.

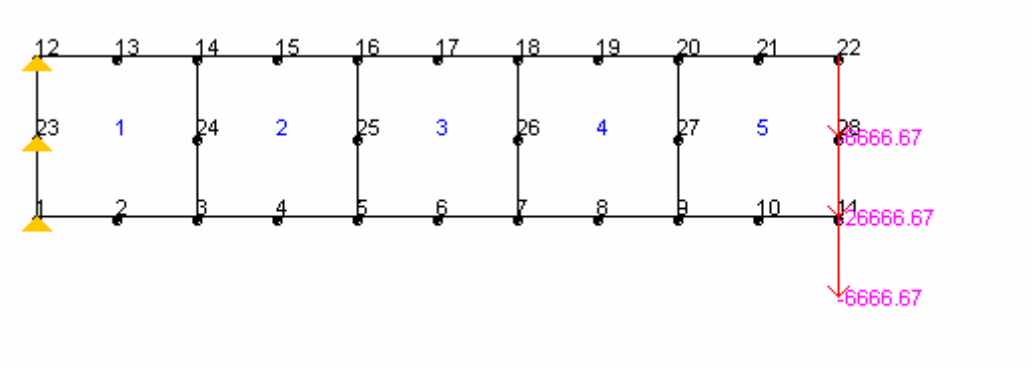

Figure 8.15 Cantilever modeled with eight-node quadrilateral elements

Geometric Data

Length L = 50 in.

Height h = 10 in.

Thickness t = 1 in.

Material Properties

E = 30 x 10^6 psi

Poisson's ratio ν = 0.25

Loading

A concentrated load of 6666.67 lb is applied at nodes 11 and 22 in the Y direction

A concentrated load of 26666.67 lb is applied at node 28 in the Y direction

Restraints

Nodes 1, 12, and 23 are restrained in the X and Y directions.

Comparison of Results

The displacements at nodes 11 and 22 and stresses at nodes 4 and 12 are shown in Table 8.9. The difference in results is less than 1% except for the stresses at node 5. The

reason for this difference is that SAP 2000 employs an error estimation method and is therefore more accurate. The deformed shape of the structure is shown in Figure 8.16.

Table 8.9 Displacements and stresses for Problem 9

Location	Parameter	Result from Applet	Result from SAP 2000	Percent error
Node 11	**UX**	-0.099261	-0.099255	**-0.01**
	UY	-0.676005	-0.675435	**-0.08**
Node 22	**UX**	0.099261	0.099255	**-0.01**
	UY	-0.676005	-0.675435	**-0.08**
Node 4	**S11**	-83985.600000	-84000.021000	**0.02**
	S22	2941.210000	2803.249000	**-4.92**
Node 12	**S11**	119976.000000	119455.572	**-0.44**
	S22	29994.100000	2.99E+04	**-0.44**

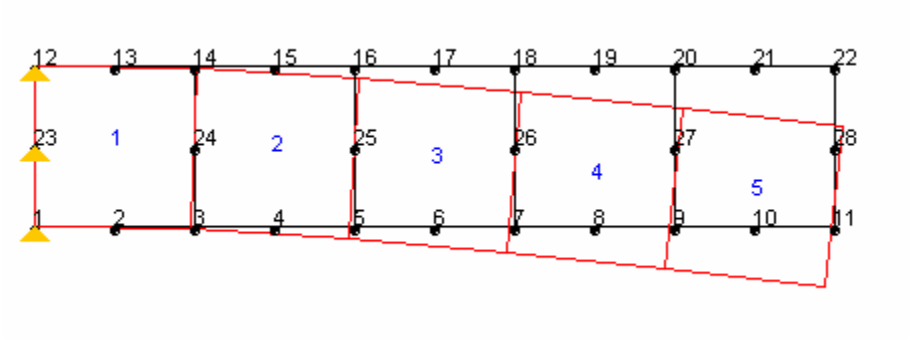

Figure 8.16 Deformed shape of cantilever for Problem 9.

Problem 10: A plate with a semi-circular hole of radius 2.5 in. at its center is analyzed using the applet and SAP 2000. The plate is subjected to a force of 100 lb/in. at its bottom end and is fixed at the top. The length of the plate is 16 in., width is 6 in. and the thickness is 1 in. The plate is modeled with 48 CST elements as shown in Figure 8.17.

Geometric Data

Width $b = 6$ in.

Height $h = 12$ in.

Thickness $t = 1$ in.

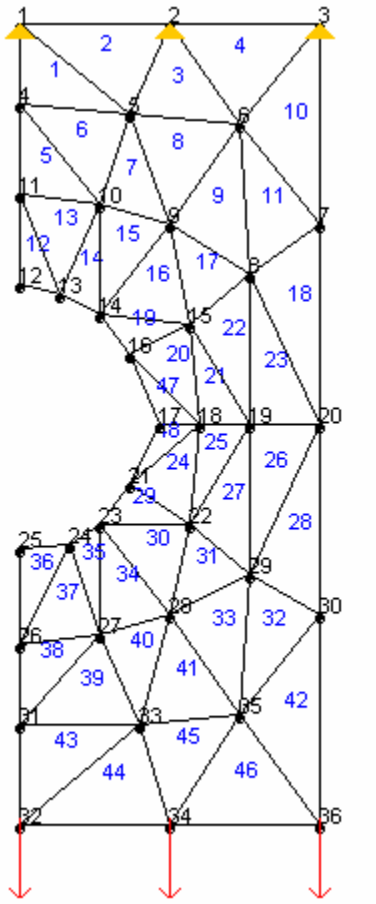

Figure 8.17 Plate with semi-circular hole modeled with CST Elements.

Material Properties

E = 30 x 10^6 psi.

Poisson's ratio ν = 0.3

Loading

A uniform load of 100 lb/in^2 is applied at the bottom end.

Restraints

Nodes 1, 2, and 3 are restrained in the X and Y directions.

Comparison of Results

Displacements at nodes 25 and 32 and stresses at nodes 1 and 17 obtained from the applet are compared to those obtained from SAP 2000 in Table 8.10. The difference in displacements at nodes 25 and 32 is less than 0.4%. The stresses at node 17 differ by

0.43 percent. The reason for this difference is that SAP 2000 averages the stresses over adjacent elements and uses an error estimation technique to arrive at more accurate values. Figure 8.18 shows the deformed shape of the plate.

Table 8.10 Displacements and stresses for Problem 10

Location	Parameter	Result from Applet	Result from SAP 2000	Percent error
Node 25	**UX**	5.75E-05	0.000058	**0.05**
	UY	-1.39E-04	-0.000139	**0.34**
Node 32	**UX**	1.73E-04	0.000173	**0.20**
	UY	-1.53E-04	-0.000153	**-0.19**
Node 1	**S11**	3.12E+01	31.212398	**0.00**
	S22	1.04E+02	104.041326	**0.00**
Node 17	**S11**	3.46E+00	3.466700	**0.29**
	S22	5.24E+02	5.26E+02	**0.43**

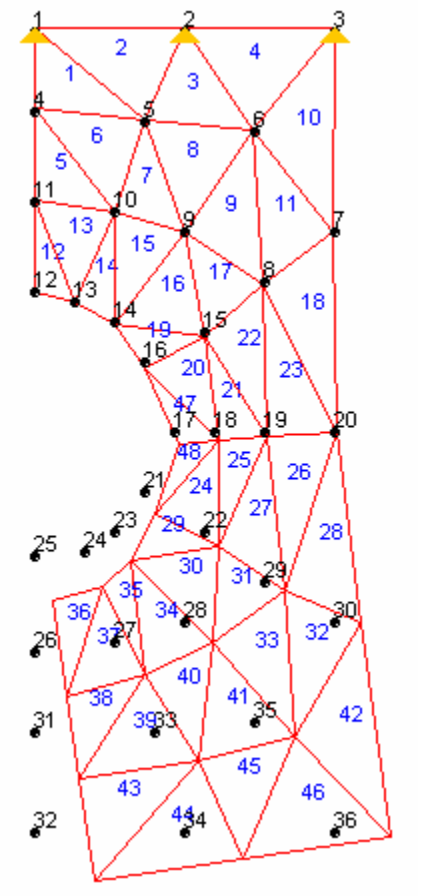

Figure 8.18 Deformed shape of plate with semi-circular hole.

Problem 11: For this test case a rectangular plate of size 4 ft x 3 ft with a square hole of size 1ft x 1ft is analyzed. The thickness of the plate is 1 in. The plate is fixed on one end and a uniform load of 100 lb/in. acts on the other end. The plate is modeled with 62 nodes and 44 four-node quadrilateral elements. A finite element discretization of this problem is shown in Figure 8.19

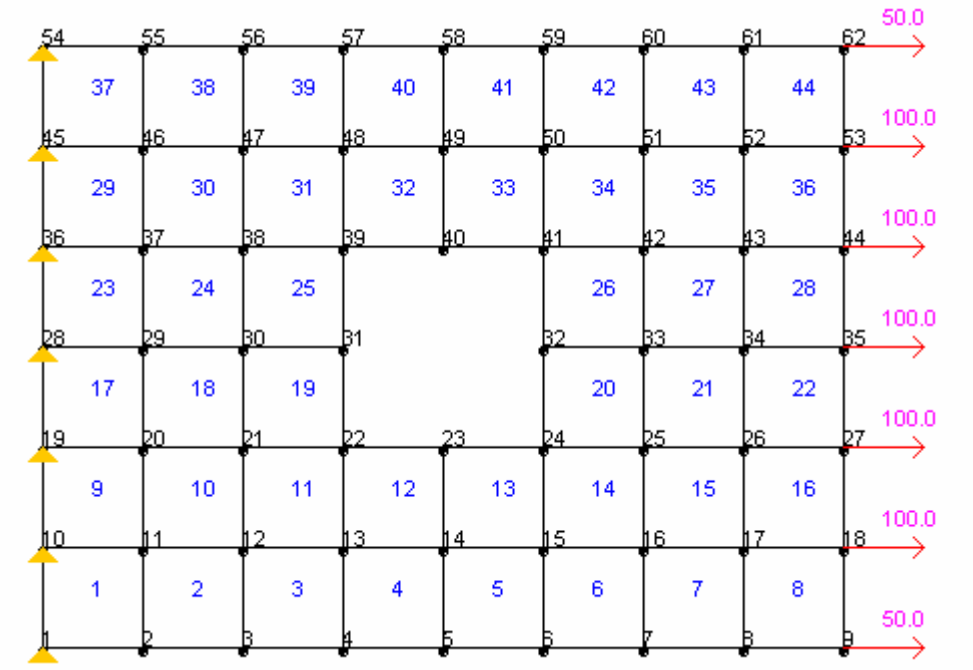

Figure 8.19 Plate with a square hole at center.

Geometric Data

Length L=48 in.

Height h=36 in.

Thickness t = 1 in.

Material Properties

E = 29 x 10^6 psi.

Poisson's ratio ν = 0.3

Loading

A uniform load of 100 lb/in. acts along the right edge as shown in Figure 8.19

Restraints

The left end of the plate is restrained against displacements in the X and Y directions. (i.e. nodes 1,10, 19, 28, 36, 45, 54 are restrained)

Comparison of Results

Table 8.11 shows the comparison of displacements and stresses obtained from the applet and SAP 2000 for this problem. The displacements at nodes 32 and 35 and stresses at nodes 23 and 40 are compared. Results obtained from the applet match with those from SAP 2000 with a minor deviation of 0.08% for the displacement at node 32 which might be due to truncation. The deformed shape of the plate is illustrated in Figure 8.20.

Table 8.11 Displacements and stresses in plate

Location	Parameter	Result from Applet	Result from SAP 2000	Percent error
Node 32	**UX**	0.000030	0.000031	**0.08**
	UY	0.000000	0.000000	**0.00**
Node 35	**UX**	0.000036	0.000036	**0.04**
	UY	0.000000	0.000000	**0.00**
Node 23	**S11**	232.044000	232.046372	**0.00**
	S22	20.222400	20.223189	**0.00**
	S12	14.281200	14.281677	**0.00**
Node 40	**S11**	232.044000	232.046372	**0.00**
	S22	20.222400	20.223189	**0.00**
	S12	-14.281200	-14.281677	**0.00**

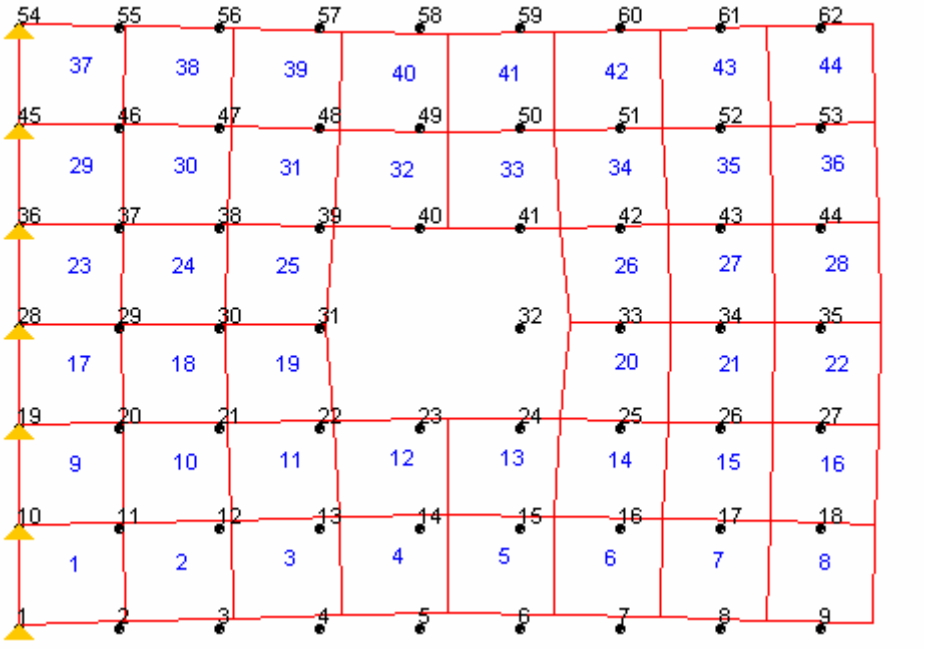

Figure 8.20 Deformed shape of plate with square hole.

8.5 Test Problems for Plate Bending Elements

The comparison of displacements and stresses obtained from the applet with those obtained from SAP 2000 for plate bending elements is presented in this section. Three problems were selected for verification of DKT and DKQ elements. A description of each problem and discussion of results is presented in this section.

Problem 12: The first problem is a square cantilever plate is of size 3 ft x 3ft and a thickness of 2 in. The plate is fixed on two adjacent sides as shown in plan (Figure 8.21). The opposite adjacent edges of plate are free. A concentrated load of 1000 lb acts at the corner (node 1).

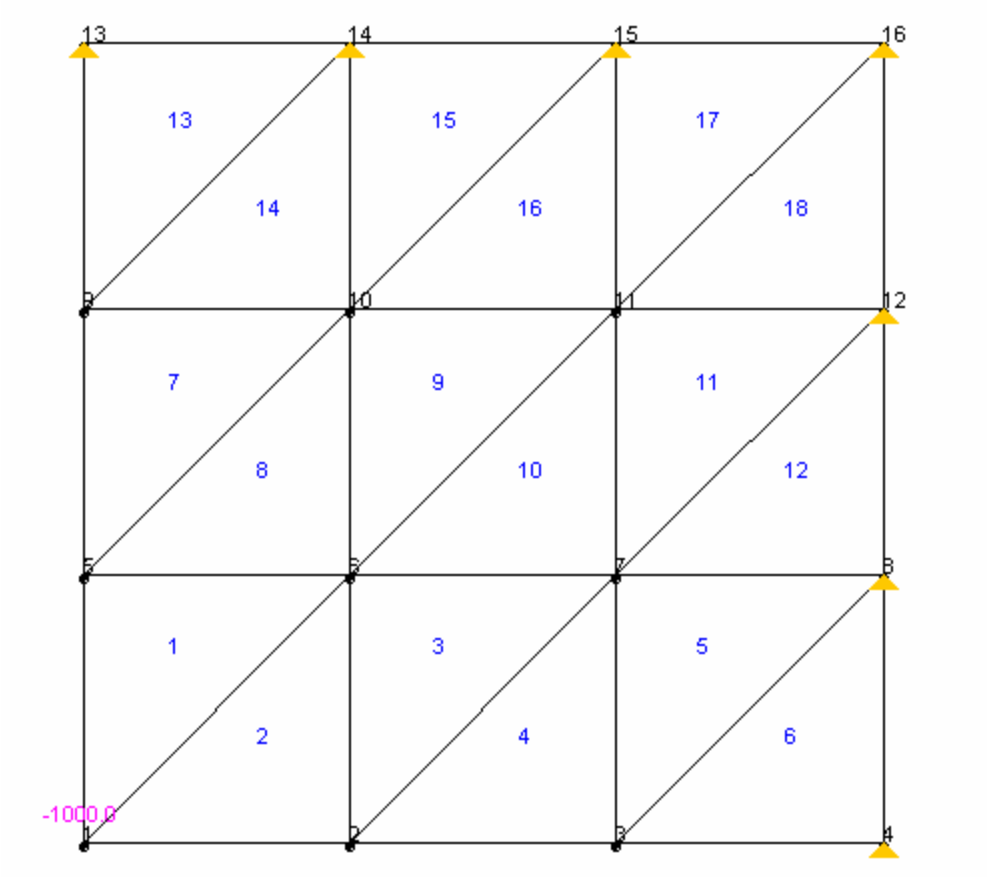

Figure 8.21 Cantilever plate modeled with DKT elements.

Geometric Data

Length L=36 in.

Height h=36 in.

Thickness t = 2 in.

Material Properties

E = 29 x 10^6 psi

Poisson's ratio ν = 0.3

Loading

A concentrated load of 1000 lb acts at corner node 1 as shown in Figure 8.21.

Restraints

The adjacent edges of the plate are fixed and the remaining two edges are free.

Comparison of Results

Table 8.12 shows the comparison of displacements at nodes 1 and 9, and stresses at nodes 15. The vertical displacement at the free end deviates from that obtained in SAP 2000 by 1.5 percent. The variation in stress at node 15 is 2.5 percent. The difference in the results is due to the fact that SAP 2000 uses a different shell element.

Table 8.12 Displacements and stresses for Problem 12

Location	Parameter	Result from Applet	Result from SAP 2000	Percent error
Node 1	UZ	-0.004382	-0.004446	**1.44**
	RX	0.000343	0.000351	**0.00**
	RY	-0.000353	-0.000351	**0.00**
Node 9	UZ	-0.000771	-0.000773	**0.29**
	RX	0.000210	0.000211	**0.37**
	RY	-0.000057	-0.000054	**0.00**
Node 15	S11	82.459700	80.520000	**-2.41**
	S22	274.866000	280.353620	**1.96**

Problem 13: A square plate of size 3 ft x 3ft and thickness of 2 in. The plate is fixed on all edges and is subjected to a uniform surface load of 100 lb/in.^{2}. The plate is modeled with 36 DKQ elements and the finite element model is shown in Figure 8.22.

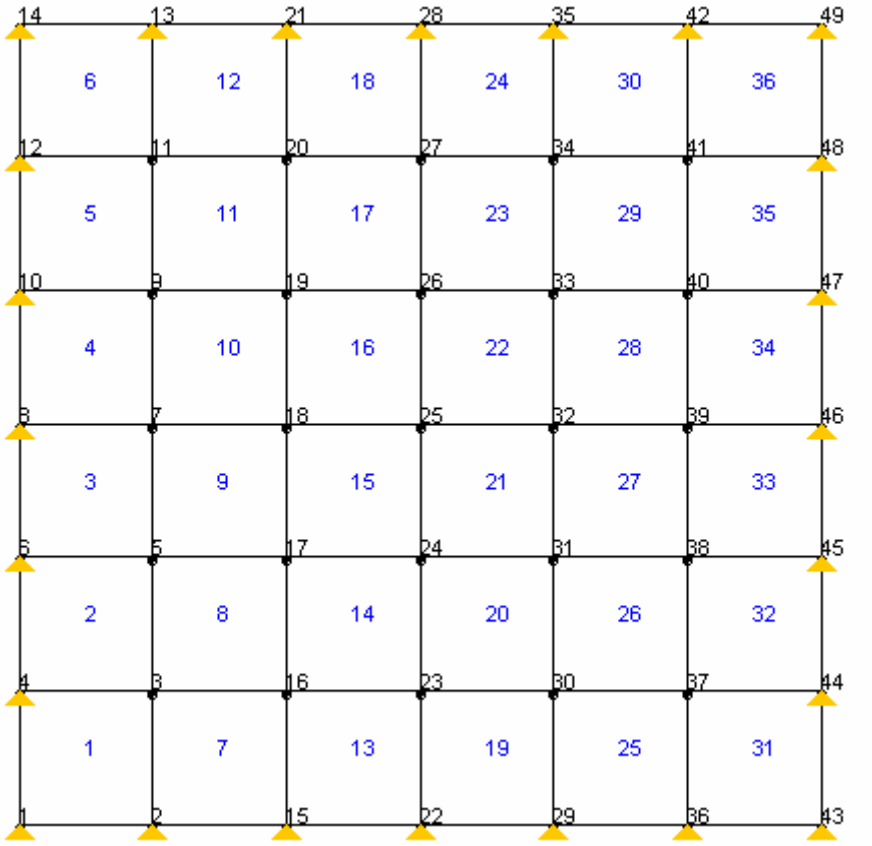

Figure 8.22 Plan view of square plate fixed on all edges.

Geometric Data

Length L=36 in.

Height h=36 in.

Thickness t = 2 in.

Material Properties

E = 29 x 10^6 psi.

Poisson's ratio ν = 0.3

Loading

A uniform surface load of 1000 lb is applied on the plate.

Restraints

All four edges of the plate are fixed.

Comparison of Results

The displacements and stresses obtained from the applet and SAP 2000 are tabulated in Table 8.13. Displacements are compared at the mid-point of the plate that is at node 25. The stresses are compared at nodes 8 and 25. The results obtained from the applet are similar to those obtained in SAP 2000.

Table 8.13 Stresses and displacements for Problem 13

Location	Parameter	Result from Applet	Result from SAP 2000	Percent error
Node 25	**UZ**	0.010747	0.010746	**0.00**
	RX	0.000000	0.000000	**0.00**
	RY	0.000000	0.000000	**0.00**
Node 8	**S11**	-9753.320000	-9753.317000	**0.00**
	S22	-2925.990000	-2925.995000	**0.00**
Node 25	**S11**	4947.330000	4947.333000	**0.00**
	S22	4947.330000	4947.333000	**0.00**

The results of the verification indicate that for the test problems considered the results obtained from the applet are essentially the same as those obtained form SAP 2000. The difference in results for the few cases where the results do not match are due to the more accurate stress computation procedure used in SAP 2000 and due to a different plate element being used in SAP 2000.

Chapter 9: Summary and Conclusions

9.1 Summary

The objective of this study was to develop a web based tool for teaching finite element analysis to engineering students. An applet and a set of web pages were developed to illustrate the basic concepts of finite element analysis. The applet was written in the object-oriented Java programming language and is platform and operating system independent. The web pages and applet are hosted on a web server, which makes them accessible to students and instructors over the Internet. Eight different elements are developed to analyze a structure using this applet. One-dimensional bar elements, truss elements, and beam elements are provided for explaining the basics of finite element analysis. The membrane elements developed in this applet included a three node triangle element (CST element), a four-node quadrilateral element, and an eight-node quadrilateral element. Plate bending elements implemented in this applet can be used to explain advanced topics in finite element analysis. The two plate bending elements provided are the Discrete Kirchhoff Triangle (DKT) and the Discrete Kirchhoff Quadrilateral (DKQ) element. The Java classes for implementing plate bending elements were developed by Kansara (2004). The applet has a graphical user interface which makes it very easy to learn. The applet also has many interactive features such as plotting the structure and its deformed shape under loading which makes it more attractive to use.

Input to the applet can be provided either directly into the program or in text format. The format of the text input is similar to SAP 2000 with a few modifications. The applet has several tabbed input forms for entering nodal coordinates, connectivity, material properties, restraints, and loading. Details of input are given in Appendix A. The purpose of providing text input is to allow students to save the finite element analysis model of the structure for future modifications. Text input is also useful when a structure with a large number of nodes and elements is analyzed.

The applet computes displacements and stresses at each node of the structure. The results obtained can be copied to the clipboard and can be saved for later reference. The applet also displays the structure and its deformed shape.

The accuracy of the applet is verified by analyzing a series of test structures and comparing the results obtained from the applet with those obtained from commercial finite element analysis software (SAP 2000). The results obtained are compared at critical locations of the structure.

9.2 Conclusions

The applet was developed using object oriented programming principles. Several classes were developed representing various structural components like nodes, elements, restraints, and loads as discussed in Chapter 7. The benefit of such an approach is that it is possible to include additional elements such as shell elements without modifying the rest of the code.

The results obtained from the applet are similar to those obtained from SAP 2000. It was found that for most cases the difference in results was less than one percent. For the few cases where the difference was larger, the difference can be attributed to a) a more accurate approach used in SAP 2000 for computing stresses and b) a different plate element used in SAP 2000. However, there is a slight deviation in stresses as no error correction procedure was incorporated in the applet.

The one-dimensional bar element, the truss element, and the two-dimensional frame elements all yielded accurate results with no deviation at all. The displacements and stresses obtained for structures consisting of the membrane elements are the same as those obtained from SAP 2000. The deviation in stresses ranged from 1 percent to 5 percent. The reason for this deviation is that SAP 2000 uses a stress averaging technique for computing stress.

Displacements calculated for plate structures using the DKT and DKQ plate bending elements are close to those obtained from SAP 2000. The difference in results was less than 2 percent. The stresses computed differed by 3 percent. The difference in stresses can be attributed to the fact that SAP 2000 uses a different plate element than the one used in the applet. Nevertheless, the variation in stresses is acceptable.

In conclusion, the applet developed has all the basic elements required for teaching finite element analysis. It has a number of features that make it very attractive in

teaching. The graphical user interface combined with the various input forms make it easy to enter data. The ability to plot the finite element model promises interactive and immediate feedback and makes it easy to detect errors in the finite element model. The applet coupled with web pages can be a useful tool for teaching engineering students. There is no limitation on the number of nodes and elements. Also, the deformed shape generated by the applet makes it easy to understand the behavior of the structure under different types of loading.

9.3 Future Development

The objective of this work is to develop an applet for teaching finite element analysis. The applet developed can be improved to make it more interactive. The functionality of the applet can be enhanced by providing additional tools for meshing the structure and providing a graphical interface for the input. Shell and solid elements can also be added to make it a complete tool for teaching finite element analysis of structures.

References

Batoz J.-L., Bathe K. J. and Ho L. W., A Study of Three-Noded Triangular Plate Bending Elements, *International Journal for Numerical Methods in Engineering*, Vol.15, 1771-1812 (1980).

Batoz J.-L. and Tahar M. B., Evaluation of a New Quadrilateral Thin Plate Bending Element, *International Journal for Numerical Methods in Engineering*, Vol. 18, 1655-1677 (1982).

Benjamin C.V., Rencis J.J., Alam J., and Hartley T. G, Jr., "Using Java to develop interactive learning material for the World-Wide Web." Worchester Polytechnic Institute, 1996.

Chandrupatla R.T., and Belegundu A.D., *Introduction to Finite Elements in Engineering*, 2nd ed., Prentice Hall, 1997.

Cook R. D., Malkus D. S., and Plesha M. E., *Concepts and Applications of Finite Element Analysis*, 3rd ed., John Wiley & Sons, 1989.

Hunt J., *Java and object orientation: an introduction*, 1st ed., Springer-Verlag, 1998.

Kansara.K., Development of Membrane, Plate and Flat Shell Elements in Java, M.S Thesis, Virginia Polytechnic Institute and State University, 2004.

Karthik R., and Kurt G., "3-D Finite Element Analysis on the Internet using Java and VRML." Proceedings, ASEE Annual Conference, June, St. Louis, MO, 2000.

Kassimali A., *Matrix Analysis of Structures*., 1st ed., Brooks/Cole Publishing Company, 1999.

Liang-Wu C., Lecture Notes for Finite Element Analysis., Kansas State University, 2004.

Ling L., Connell M., Tullberg O., "The Use of Virtual Reality in Interactive Finite Element Analysis: State of the Art Report". Department of Structural Mechanics Chalmers University of Technology, Gothenburg, Sweden, 2001

Storey M.-A., Phillips B., Maczewski M., and Wang M., Evaluating the usability of Web-based learning tools, *Education Technology & Society*, Vol. 5, 2002.

"NASA – NASA expands web-based education program"., Marshall center space news release , January 2004.

Newman A., *Using Java*, 1st ed., Que Corporation, 1996.

Nikishkov G.P., Tsuchimoto T., and Mirenkov N.N., "Web-Based Teacher-Student Interaction in a Traditional Course", University of Aizu, Aizu-Wakamatsu, Fukushima 965-8580, Japan.

Nikishkov G.P., "Generating contours on FEM/BEM higher-order surfaces using Java 3D textures", Advances in Engineering Software, 2003, 34(8), pp. 469-476.

Paul S. S, Gallagher E., "Use of Simplified FEA to Enhance Visualization in Mechanics". Proceedings, ASEE Annual Conference, 2004.

Reddy J.N., *An Introduction to the Finite Element Method*, 2nd ed., McGraw Hill, 1993.

Report of the Web-Based Education Commission to the President and the Congress of the United States, "The power of Internet for learning: Moving from promise to practice", December 2000.

Rojiani K. B., Kim Y.Y., and Kapania R.K., "Web-Based Java Applets for Teaching Engineering Mechanics." Proceedings, ASEE Annual Conference, June, St. Louis, MO, 2000.

SAP-2000 - Integrated Finite Element Analysis and Design of Structures, Analysis Reference, Computers and Structures, Inc., Berkeley, California, 1997.

SAP-2000 - Integrated Finite Element Analysis and Design of Structures, Input File Format, Computers and Structures, Inc., Berkeley, California, 1997.

Schildt H., *Java: the complete reference* , 5th ed., McGraw Hill/Osborne, 2005.

Sennett R.E., *Matrix Analysis of Structures*., 1st ed., Waveland Press, Inc. 2000.

Wikipedia – Wikimedia Foundation Inc., http://www.wikipedia.org

Appendix – A: Input for the program

As mentioned in previous chapters, the input to the program can be provided either manually or from a text file. Manual input is useful for explaining the pre-processing step in finite element analysis where information on the finite element model such as nodal coordinates, elements, and material data is entered. Manual input can be cumbersome for large models. Text input is useful when a model with a large number of nodes and elements is to be analyzed. For this case, text input is a more efficient way to enter model data. All input data is given in inches and pounds or in any consistent system of units. This section discusses both input methods.

A-1 Manual Input

Manual input is divided into several steps, from selecting the joint type through defining the loads. Individual tabs are provided for entering coordinates, connectivity, restraints, and loading data.

Step 1: Select element type.

1. From the **Model** menu, select the **Input** menu item. This displays the **Element** tab.
2. Select the type of element to be used for analyzing the structure from the **Element Type** drop down box provided in the Element tab.
3. For a one-dimensional bar element, select the degree of freedom (X or Y direction) by selecting the appropriate check box. For all other elements, both X and Y are active.
4. For membrane elements, select plane stress or plane strain by clicking on the appropriate check box.

Step 2: Enter joint coordinates.

1. Select the **Co-ordinates** tab.
2. Enter the node number and X and Y coordinates in the text fields provided.
3. Press the **Add** button to add the node. The added node is displayed in the List Box. Repeat this until all the nodes are entered.
4. To remove a node or change its coordinates, select the node from the List Box and press the Remove button.

Step 3: Enter connectivity data.

1. Select the **Connectivity** tab.
2. Enter the element number and the node numbers at which the element is connected. The connecting nodes are specified in a counterclockwise direction and are separated by a space.
3. Enter material properties of the element in the respective fields. Press the **Add** button to add the element connectivity and material data.
4. Repeat the above procedure (Steps 1-3) to enter connectivity data for all elements in the structure.

Step 4: Enter joint restraints.

1. Select the **Restraints** tab.
2. Enter the node number and select the direction of the restraint by clicking the drop down box corresponding to the restraint direction.
3. Press the **Add** button to enter restraint data.
4. To remove a restraint, select the restraint from List Box and press the **Remove** button.

Step 5: Enter loads.

1. Select the **Loads** tab.
2. Enter the node number and loads in x and y directions.
3. Press the **Add** button to add the loads.
4. To remove a load, select the list box and press the **Remove** button.
5. To enter a linear or uniform load on a element, specify the start and end nodes and the magnitude of the load at these nodes.

Step 6: View the model.

1. Select **Draw** form **Model** menu to view the model. Follow steps 1 through 5 to make modifications to the model.

Step 7: Analyze the structure.

1. Once the desired model is created, press **Run** from the **Analysis** menu to perform the analysis.

Step 8: View results and deformed shape.

1. To view the output of displacements and stresses, click **Results** from the **Analysis** menu.
2. The deformed view of the structure can be obtained by clicking the **Deformed Shape** from the **Analysis** menu.

To perform a new analysis, select **New** from the **File** menu.

A-2 Text Input

The input for the program can also be provided by creating a text file and copying it into the text area of the applet. The format of the text file essentially follows that used in SAP 2000. However, a few modifications were made to this format. Due to security reasons, a Java applet cannot directly read a text file and hence the input text is copied to the clipboard and then transferred to the applet.

To provide input in text format, the **Text Input** checkbox is selected, which then displays the text panel. The text from the clipboard is then copied to this text form. Once the text is pasted, the **Read** button is pressed. The input is then read and is interpreted and the model is generated.

The data is provided in input blocks and each data block is separated by a title which defines the block and separates it from the other data blocks. The data blocks are tabulated in Table A-1 and must be in the same order as listed in Table A-1.

Table A -1 Data blocks for input file.

Data Block	Function
SYSTEM	Defines system properties
JOINTS	Defines joint coordinates
RESTRAINTS	Defines restraints at nodes
MATERIAL	Defines material properties and thickness
CONNECTIVITY	Defines joint connectivity for each element
LOADS	Defines applied loads on nodes and elements
END	Marks the end of input text

SYSTEM Data Block

The SYSTEM data block defines the degrees of freedom for the entire structure. The possible restraints are derived from this data block. The format of the system data is shown in Table A-2.

Table A - 2 SYSTEM Data Block

SYSTEM	
DOF = UX OR DOF = UY	One-dimensional Bar Element
DOF = UX UY	Truss Element and Plane Stress Elements
DOF = UX UY RZ	Two-dimensional Beam Element
DOF = UZ RX RY	Plate Bending Elements

JOINTS Data Block

The JOINTS data block specifies the coordinates of the joints. For each joint there is a separate line for each node. The input on each line consists of the joint number and the X and Y coordinates of the joint.

Table A - 3 JOINTS Data Block

JOINTS		
1	X = 1	Y = 1
Node Number	X Coordinate	Y Coordinate

RESTRAINTS Data Block

The RESTRAINTS data block specifies the restraints at different nodes of the structure. The notations UX, UY represents translational restraints in the global X and Y directions and RX, RY, and RZ represent rotational restraints in the global X, Y, and Z directions respectively.

Table A- 4 RESTRAINTS Data Block

RESTRAINTS	
ADD = 1	DOF = UX UY RX RY RZ
Adds restraint at node	Defines the direction of restraints.

MATERIAL Data Block

The MATERIAL data block defines material properties for each element such as the modulus of elasticity and Poisson's ratio. This section also defines section properties of the element such as the thickness and area of the element.

Table A- 5 MATERIAL Data Block

MATERIAL			
STEEL	E = 29000	U = 0.3	TH = 1 or AR = 1
Name of section	Modulus of elasticity	Poisson's ratio	thickness or area of element

CONNECTIVITY Data Block

The CONNECTIVITY data block provides the connectivity data and material type. For each element the nodes are specified in a counterclockwise direction. The material type specifies the name of the material corresponding to the material label specified in the material data block.

Table A-6 CONNECTIVITY Data Block

CONNECTIVITY	
1	J = 1 2 STEEL
Element number	Defines nodes j1 and j2 (in a counterclockwise direction) for one-dimensional bar, truss, and beam elements. Label STEEL defines the material type.
1	J = 1 2 3 STEEL
Element number	Defines nodes j1, j2, and j3 (in a counterclockwise direction) for triangular elements (both plane stress and plate elements)
1	J = 1 2 3 4 STEEL
Element number	Defines nodes j1, j2, j3, and j4 (in a counterclockwise direction) for four-node quadrilateral elements (both plane stress and plate elements)
1	J = 1 2 3 4 5 6 7 8 STEEL
Element number	Defines nodes j1,j2……j8 (in a counterclockwise direction) for eight-node quadrilateral elements

LOADS Data Block

The LOADS data block defines two types of loads. For concentrated loads applied at a node, the direction of the load and the magnitude of the load are specified. A concentrated load can be applied at any node. In the case of uniform loads, additional details such as the start and end nodes and magnitude of the load at nodes are specified.

Table A- 7 LOADS Data Block

LOADS	
ADD = 1	UX = 100 UY = -50
Adds concentrated load at specified node	Magnitude of load in X and Y directions
ADDU = 1	S = 1 S = 2 UM = 2
Adds uniform load on specified element	Specifies start node and end node of load and magnitude of load.

END Data Block

The END data block consists of a single line with the keyword END and specifies the end of input.

A-3 Examples of the Input File.

Example 1: The input file for the truss of Problem 4 in Chapter 8 is shown in below.

```
SYSTEM
  DOF = UX UY

JOINTS
  1 X = 0 Y = 0
  2 X = 180 Y = 0
  3 X = 360 Y = 0
  4 X = 540 Y = 0
  5 X = 180 Y = 240
  6 X = 360 Y = 240

RESTRAINTS
  ADD = 1 DOF = UX UY
  ADD = 3 DOF = UY
```

```
  ADD = 4 DOF = UY

MATERIAL
  MAT1 E = 29E6 U = 0 TH = 6.0


CONNECTIVITY
  1 J = 1 2 MAT1
  2 J = 2 3 MAT1
  3 J = 3 4 MAT1
  4 J = 5 6 MAT1
  5 J = 2 5 MAT1
  6 J = 3 6 MAT1
  7 J = 1 5 MAT1
  8 J = 5 3 MAT1
  9 J = 6 4 MAT1
LOADS
   ADD = 2 UX = 0 UY = -30000
  ADD = 5 UX = 10000 UY = 0
  ADD = 6 UX = 0 UY = -30000

END
```

Example 2: The input file for the plate of Problem 12 in Chapter 8 is shown in below.

```
JOINTS
  1 X = -9 Y = -9
  2 X = -3 Y = -9
  3 X = 3 Y = -9
  4 X = 9 Y = -9
  5 X = -9 Y = -3
```

```
  6 X = -3 Y = -3
  7 X = 3 Y = -3
  8 X = 9 Y = -3
  9 X = -9 Y = 3
  10 X = -3 Y = 3
  11 X = 3 Y = 3
  12 X = 9 Y = 3
  13 X = -9 Y = 9
  14 X = -3 Y = 9
  15 X = 3 Y = 9
  16 X = 9 Y = 9

RESTRAINTS
  ADD = 4 DOF = UX UY RZ
  ADD = 8 DOF = UX UY RZ
  ADD = 12 DOF = UX UY RZ
  ADD = 13 DOF = UX UY RZ
  ADD = 14 DOF = UX UY RZ
  ADD = 15 DOF = UX UY RZ
  ADD = 16 DOF = UX UY RZ

MATERIAL
  MAT1 E = 2.9E7 U = 0.3 TH = 2


CONNECTIVITY
  1 J = 6 5 1 MAT1
  2 J = 2 6 1 MAT1
  3 J = 7 6 2 MAT1
  4 J = 3 7 2 MAT1
  5 J = 8 7 3 MAT1
  6 J = 4 8 3 MAT1
```

```
  7 J = 10 9 5 MAT1
  8 J = 6 10 5 MAT1
  9 J = 11 10 6 MAT1
  10 J = 7 11 6 MAT1
  11 J = 12 11 7 MAT1
  12 J = 12 7 8 MAT1
  13 J = 14 13 9 MAT1
  14 J = 10 14 9 MAT1
  15 J = 15 14 10 MAT1
  16 J = 11 15 10 MAT1
  17 J = 16 15 11 MAT1
  18 J = 12 16 11 MAT1

LOADS

      ADD = 1 UX = -1000 UY = 0


END
```

Vita

Suryanarayana Raju Sagi Venkata Naga was born on December 4, 1983 in Visakhapatnam, India. He did his schooling from Kendriya Vidyalaya- Srivijaynagar. He received his Bachelor of Engineering degree in Civil Engineering with a gold medal for outstanding academic performance from Andhra University, Visakhapatnam in March 2004. He joined Virginia Tech to pursue his Master of Science degree in Civil Engineering in January 2005.