Development of Membrane, Plate and Flat Shell Elements in Java

by

Kaushalkumar Kansara

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. W. S. Easterling

Dr. C. L. Roberts - Wollmann

May, 2004
Blacksburg, Virginia

Key Words: Finite Element Analysis, Membrane, Plate, Flat Shell, Java.

Development of Membrane, Plate and Flat Shell Elements in Java

by

Kaushalkumar M. Kansara

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

(ABSTRACT)

The development of triangular and quadrilateral membrane, plate and shell elements in Java using the object oriented programming technique is presented. The membrane elements developed are the constant strain triangle (CST) element and the four node isoparametric quadrilateral membrane element. The plate bending elements developed are the discrete Kirchoff triangular (DKT) element and discrete Kirchoff quadrilateral (DKQ) element. The flat shell elements are developed by super imposing the stiffness of the membrane element and plate bending element. A finite element analysis program is also developed to check the accuracy of the developed elements. The program is developed using the object oriented programming approach as an alternative to traditional procedural programming. Several test structures are analyzed using the developed program for each developed element and the results are compared with those obtained from the commercial finite element analysis program SAP 2000. The results indicate that all elements give accurate displacements. However, there were significant differences in stresses for the shell elements, which can be attributable to the approximate approach in these elements to model the drilling degree of freedom.

Acknowledgements

I would like to first thank 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. I would like to thank Dr. Easterling and Dr. Wollmann to serve as my committee members. I would also like to thank my uncle for giving me financial support for my entire study and special thanks to my parents for their constant moral support.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Introduction

The finite element method is an approximate numerical procedure for analyzing large structures and continua (Cook *et al.*, 1989). The finite element method became popular with the advancements in digital computers since they allow engineers to solve large systems of equations quickly and efficiently. The finite element method is a very useful tool for the solution of many types of engineering problems such as the analysis of the structures, heat transfer and fluid flow. The method is also used in the design of air frames, ships, electric motors, heat engines and spacecraft. The finite element method is also used for analyzing the behavior of components of biological systems.

In most structural analysis applications it is necessary to compute displacements and stresses at various points of interest. The finite element method is a very valuable tool for studying the behavior of structures. In the finite element method, the finite element model is created by dividing the structure in to a number of finite elements. Each element is interconnected by nodes. The selection of elements for modeling the structure depends upon the behavior and geometry of the structure being analyzed. The modeling pattern, which is generally called mesh for the finite element method, is a very important part of the modeling process. The results obtained from the analysis depend upon the selection of the finite elements and the mesh size. Although the finite element model does not behave exactly like the actual structure, it is possible to obtain sufficiently accurate results for most practical applications. Once the finite element model has been created, the equilibrium equations can easily be solved using digital computers without having to solve a large number of partial differential equations by hand. The deflections at each node of the finite element model are obtained by solving the equilibrium equations. The stresses and strains then can be obtained from the stress-strain and strain-displacement relations.

The finite element method is ideally suited for implementation on a computer. With the advancements in digital computers, the finite element method is becoming the method of choice for solving many engineering problems, and is extensively used for structural analysis. The structure can be discretized using frame elements, plane elements, plate elements, or shell elements according to the behavior of the structure. The structure can also be modeled by combining different types of elements to approximate different aspects of structural behavior.

1.2 Objective and Scope

The focus of this thesis is to develop triangular and quadrilateral flat shell elements for the finite element analysis of thin shell structures. The flat shell elements are developed by combining membrane elements with plate bending elements. An important aspect of the work is to implement these elements on the computer using an object oriented approach and the Java programming language. Java was chosen as the programming language for this work; since (1) Java is an object oriented programming language, and (2) Java is platform independent.

Four elements were considered in this study. The membrane elements considered were the CST (constant strain triangle) element, and the four node isoparametric quadrilateral plane element (QUAD4). The plate bending elements used were the discrete Kirchoff plate bending elements, the DKT element (discrete Kirchoff triangular element) (Batoz *et al.*, 1980) and the DKQ element (discrete Kirchoff quadrilateral element) (Batoz and Tahar, 1982). The flat shell elements were obtained by combining the above mentioned elements.

A finite element analysis program that calculates deflections and stresses was also developed in order to verify the accuracy of these elements. This program was written in Java using the object oriented approach. The results obtained for a series of test problems were compared with those from a commercial finite element analysis program.

1.3 Organization

The thesis is divided in to eight chapters including the Introduction. An overview of the finite element analysis method and the Java programming language is presented in Chapter 2. This chapter also contains a discussion of membrane and plate bending elements and techniques for developing flat shell elements. Chapter 3 describes the development of the triangular and quadrilateral membrane elements and their implementation in Java. Chapter 4 describes the development of the triangular and quadrilateral plate bending elements based on the discrete Kirchoff theory and their implementation in Java. The development of flat shell elements by combining the membrane elements and plate bending elements and the implementation of these elements is discussed in Chapter 5. Chapter 6 describes the object oriented finite element analysis program developed to test the accuracy of the flat shell elements. The test problems used to verify the accuracy of the results obtained are described in Chapter 7. Also, the results obtained from the Java program are compared with those obtained from the SAP 2000 commercial finite element analysis program. Chapter 8 presents a summary of the important results and a discussion of the results. Suggestions for future development are also given in this chapter.

Chapter 2
Literature Review

2.1 Introduction

Since the evolution of the term finite element by Clough in 1951, there have been significant developments in finite element method. A large number of different finite elements have been developed; the finite element method has been used for solving problems in different fields of engineering. The finite element method became even more popular with the advancement of microcomputers and development of various efficient programming languages. In this chapter, the development of membrane, plate bending and flat shell finite elements is discussed. An overview of the object oriented Java programming language is also presented.

2.2 Membrane Elements

Membrane elements are among the simplest elements to develop. These elements are used for analyzing structures subjected to inplane forces. Assuming that the structure is in the xy plane, the displacements at any point of the structure are u, the translation in the x direction and v, the translation in the y direction. The stresses of interest are the normal stresses s_x and s_y and the shearing stress t_{xy}. The normal stress in the direction perpendicular to the plane of structure is considered to be zero. Membrane elements are used to model the behavior of shear wall, stiffened sheet construction, and membrane action in shells.

The membrane element used in this study for the formulation of triangular flat shell elements is the constant strain triangle (CST). The CST element is so named because the strains within the element are independent of the coordinates and hence are constant over the element. Triangular elements are useful for modeling arbitrary shaped geometry and hence are used quite extensively for the analysis of planar structures.

Isoparametric elements are useful for modeling structures with irregular boundaries. The word isoparametric derived from ‘iso’ (“same”) and ‘parametric’ (“parameter”) indicates that the same functions are used to define the shape and displacements of the element. It is often difficult to model the geometry of a structure with just the regular shaped triangular or rectangular elements. Isoparametric elements are useful for modeling structures; since the isoparametric elements can have curved sides. Such elements are formulated using higher order interpolation functions. Isoparametric elements are formulated in the natural coordinate system that maps the element geometry in terms of natural coordinates regardless of the orientation of an element in the global coordinate system; however, the relationship between the two systems must be used in the element formulation (Cook, 1974).

Irons (1966), introduced the concept of isoparametric elements in stiffness methods. The four node isoparametric quadrilateral element is the simplest element in the family of isoparametric plane elements. Ergatoudis *et al.* (1968) developed shape functions to formulate the element stiffness matrix for four node isoparametric quadrilateral element. The four node isoparametric quadrilateral plane element is used to develop the quadrilateral flat shell element in this study.

2.3 Plate Bending Elements

There has been considerable interest in the development of plate bending elements ever since their use became popular for representing the bending behavior of the shell elements. Many plate bending elements have been developed. Hrabok and Hrudey (1984) presented a review of all plate bending elements as a part of the study on the effectiveness of plate bending elements. Clough and Tocher (1965) developed the triangular plate bending element by dividing the main triangle in to three subtriangles. Bazeley *et al.* (1966) developed confirming and nonconfirming plate bending elements. They developed a triangular plate bending element by using shape functions based on the area coordinates. The nonconforming plate bending element does not pass the patch test for some mesh patterns, and the confirming element is costly to use because of the high order numerical integration scheme required to determine the stiffness matrix of the element.

Batoz *et al.* (1980) developed several effective triangular plate bending elements for the analysis of plates and shells. These elements had two rotational degrees of freedom and one translational degree of freedom at each node for a total of 9 degrees of freedom. They developed three types of plate bending elements: (1) the DKT element based on Discrete Kirchoff Theory assumptions, (2) the HSM element based on the Hybrid Stress Method, to overcome the problems in development of pure displacement based models, and (3) the SRI element based on Selective Reduced Integration scheme that includes transverse shear deformation. Batoz et al (1980) compared the results obtained for these elements. They found that the DKT and HSM elements are more effective than the SRI element. They also found that the DKT element gives better results than the HSM element because the DKT element requires less storage compared to the HSM element.

Quadrilateral plate bending elements are popular in analyzing slab structures and are used in formulating shell elements for the analysis of regular shaped shell structures. Earlier attempts to develop quadrilateral plate bending elements involved combining four triangular plate bending elements (Batoz and Tahar, 1982). However their formulation was very complicated. McNeal (1978) developed a four node quadrilateral shell element using isoparametric shape functions. This element gives very good results for plate bending. Robinson and Haggenmacher (1979) developed the quadrilateral plate bending element, LORA based on stress parameters rather than displacement fields. This element also gives very good results for plate bending.

Batoz and Tahar (1982) reviewed the earlier attempts to develop plate bending elements and concluded that these elements were useful for thick plates, but when applied to the thin plates they do not give very good results. Batoz and Tahar (1982) developed a four node quadrilateral element based on the Discrete Kirchoff theory. The basis of the formulation of this element was the Discrete Kirchoff Triangular (DKT) element developed earlier (Batoz *et al.*, 1980). The quadrilateral plate bending element (DKQ) formulated by Batoz and Tahar (1982) and the triangular plate bending element (DKT) formulated by Batoz *et al.* (1980), are based on the discrete Kirchoff assumptions in which the transverse shear strain is neglected. They considered transverse shear strain to be present in the element in the initial development and then removed the transverse shear strain terms by applying discrete Kirchoff constraints. Batoz and Tahar (1982) conducted several tests on these elements. Based on their study, they suggested that the convergence rates in displacements and stresses for DKQ element is not good as for the QUAD4 element by McNeal (1978) and LORA by Robinson and Haggenmacher (1979).

2.4 Flat Shell Elements

Shell elements are widely used to model the curved geometry of a structure. Shell elements based on classical shell theory are very difficult to develop. Many simplifying approximations are involved in the development, which leads to less accurate results. These types of elements are very efficient in modeling the curved geometry of the structure. However, because of the complexities involved, the alternative approach of modeling the structure with series of flat elements, which is simpler and easier to implement, became more popular for the analysis of shell structures.

In 1961, Green *et al.* first developed the concept of using triangular flat shell elements to model arbitrary shaped shell structures (Zienkiewicz, 1971). Shells with cylindrical shapes or regular curved surfaces can be modeled using rectangular or quadrilateral flat shell elements. Zienkiewicz (1971) recommended modeling curved surface by a series of flat shell elements, rather than using the more complex curved shell elements. He suggested developing a built up element by combining membrane and plate bending elements to develop a flat shell elements.

Bathe and Ho (1981) studied two approaches for the development of shell elements. The first approach is to use the higher order isoparametric elements, which are formulated on the basis of three dimensional stress conditions and using the higher order shape functions and integration scheme. The second approach is to use lower order shell elements, which are developed by superimposing previously available membrane and plate bending elements and hence obtaining the membrane and bending properties of the shell element. They found that the second approach is more cost effective than the first because of the simplicity of development. The lower order terms used in the formulation require less computation effort and time. They also concluded that higher order elements give far superior results than the lower order elements, but they are costly to implement on the computer because of the large size of the stiffness matrix.

McNeal and Harder (1988) suggested in their study that, the higher order elements take three times more solution effort than the lower order elements. Another drawback of higher order elements is the use of high order numerical integration schemes to avoid spurious zero energy modes. Lower order elements require a large number of elements to model the structure but they require less computational effort and hence are still cheaper as compared to the higher order elements. However, the effectiveness of the element and accuracy of results of the lower order elements largely depends on the type of the element selected for the formulation of the shell element.

Flat shell elements are developed by combining membrane elements containing two inplane translational degrees of freedom and plate bending elements containing two rotational degrees of freedom and one out of plane translational degree of freedom. Since the inplane rotational degrees of freedom are not included, that leaves null or zero values in the stiffness matrix. The null values for the inplane rotational degrees of freedom, generally called drilling degrees of freedom gives singularity in structure stiffness matrix if all the elements are co-planar. Chen (1992) suggested that problems occur in solving in-filled frames, folded plate structures and other complex structural systems when the inplane rotational stiffness is not included in the stiffness matrix of the shell element.

Several methods have been suggested by various authors for removing the singularity in the stiffness matrix. The normal approach to deal with the stiffness of the drilling degrees of freedom is to approximate the stiffness for the drilling degrees of freedom. Knight (1997) suggested that a very small value be specified for the stiffness of the drilling degrees of freedom so that the contribution to the strain energy equation from this term will be zero. Zienkiewicz (1971) developed the matrix for the stiffness of the drilling degrees of freedom for triangular flat shell elements. Bathe and Ho (1981) approximated the stiffness for drilling degrees of freedom by using a small approximate value.

Batoz and Dhatt (1972) presented the formulation of a triangular shell element named KLI element with 15 degrees of freedom and a quadrilateral shell element named KQT element with 20 degrees of freedom using the discrete Kirchoff formulation of plate bending element. The KQT element was developed by combining four triangular elements with the mid-nodes on the sides. The KQT element was found effective among the two.

Bathe and Ho (1981) developed a flat shell triangular element by combining the CST element for membrane stiffness and the plate bending element using the Mindlin theory of plates for the bending stiffness. They introduced a fictitious stiffness for the drilling degrees of freedom in the development of the element stiffness matrix for the triangular flat shell element. The element developed by Bathe and Ho (1981) was found to be very effective for the analysis of shell structures.

McNeal (1978) developed the quadrilateral shell element QUAD4, by considering two inplane displacements that represent membrane properties and one out-of-plane displacement and two rotations, which represents the bending properties. McNeal (1978) included modifications in terms of a reduced order integration scheme for shear terms. He also included curvature and transverse shear flexibility to deal with the deficiency in the bending strain energy.

The simplest method adopted to remove the rotational singularity is to add a fictitious rotational stiffness. However, Yang *et al.* (2000) suggested that, although the method solves the problem of singularity it creates a convergence problem that sometimes leads to poor results. However, the majority of the flat shell elements are developed by inducing the fictitious rotational stiffness to remove the singularity. Recent developments include using membrane elements with rotational degrees of freedom to develop an efficient flat shell element.

2.5 Java Programming Language

Computer programming languages are built around two approaches; (1) procedural programming and (2) object oriented programming. In procedural programming, the program is prepared by a series of steps or routines that follow the data provided. The programming languages FORTRAN, C, BASIC are procedural programming languages. The main drawback of the procedural programming languages is that they are not structured and the flow of the program largely depends on conditional statements that induce more chances of errors. These languages are good for small programs, but procedural program are difficult to maintain when they become larger.

In the object oriented programming approach, the program is organized around its data in the form of objects (Schildt, 2001). The object oriented programming languages are built on the concept of abstraction. Large complex procedures can be subdivided in to small procedures by abstraction. Each of these sub procedures represents different objects with their own separate identity. The series of process steps can be achieved by passing information to the objects without being affected by the complexity of the whole procedure. The three unique aspects of the object oriented programming languages are: (1) Encapsulation (2) Inheritance and (3) Polymorphism. Each of these concepts is discussed separately in following paragraphs.

Encapsulation is the most important aspect of the object oriented programming. In object oriented programming languages, classes perform the task of encapsulation. Class defines the structure and behavior of the process that will be shared by a set of objects (Schildt, 2001) such as variables and methods. The variables or methods are declared by access specifiers such as public, private or protected. A variable or method declared as public can be accessed from outside the class in the program. Variables or methods that are defined as private cannot be accessed from outside the class and hence the privacy of the data is maintained. Variables or methods declared protected are only accessible to the superclass and the subclass where the properties of the superclass are inherited.

Many programs contain objects that are dependent on each other and inherit certain properties from one object to another. In object oriented programming, the classes are divided in the superclass and subclass. The subclass inherits all of the properties of the superclass except those declared as private. Any subclass that inherits properties from its subclass may have additional properties that give it an individual identity, which is not common to the other objects or subclasses that inherit the same properties from the super class.

Polymorphism is another valuable feature of object oriented programming. Polymorphism allows the programmer to use the same interface to perform multiple tasks. The same class may contain multiple methods that are related to different activities and each method will perform a different task when it is called by the object. The call to one method will not affect the contents or activity of another method in the same class.

Several object oriented programming languages have been developed in recent years. These include C++, C#, and Java. Java is one of the more popular object oriented programming languages because it has several unique features.

Java is an object oriented programming language developed by Sun Microsystems in 1991. Java is based on the popular programming language C++. It has many of the same features of C++. Over the years, Java has become very popular because of some of its unique features. Java is platform independent, which means code developed in Java can be used on a variety of different computers without making any changes. Another advantage is that, Java programs can be embedded within HTML pages (where they are called applets) and can be easily transmitted over the internet. All this needs is a Java compatible web browser to run these applets.

Another reason for the popularity of Java is its robustness. Java provides automatic protections for memory loss and run-time errors that occur during the program execution. Java has a special garbage collection class that dynamically allocates memory and hence prevents memory loss. In other programming languages, this is done manually by the programmer and any mistake in allocating or deallocating memory may result in failure of the program. There is also an exception handling class for handling runtime errors in Java. With the use of this class it is possible to catch many common runtime errors which would otherwise result in program failure.

Until recently most finite element analysis program were written in FORTRAN. Although FORTRAN is an efficient language for developing scientific applications, it is not well suited for writing large complex programs. With the advancements in finite element analysis, many different types of elements are being developed and elements are constantly being modified to improve their behavior. It is very difficult to maintain the codes for the finite element analysis that were developed using procedural programming languages because of its complexity. An object oriented programming language such as Java is better suited for the development of large complex programs for finite element analysis, because of the many advantages discussed above. These advantages make it very attractive to implement object oriented programming techniques for the development of the finite element analysis codes.

Chapter 3
Membrane Elements

3.1 Overview

Two types of membrane elements are used for development of flat shell elements in this study: (1) the constant strain triangle (CST) element, and (2) the four node isoparametric quadrilateral element. In this chapter the development of the element stiffness matrix for these elements is presented. The implementation of these elements in Java is also discussed.

3.2 Two Dimensional Stresses and Strains

Two dimensional elasticity problems typically involve structures that are very thin and the loads are applied in the direction in the plane of the structure. Consider a structure in the xy plane with thickness t along the z direction. When inplane forces are applied to the structure, the displacements at any discrete point of the structure located by the coordiantes $\left(x, y\right)$ are,

$$\boldsymbol{U} = \left\{u, v\right\}^T \tag{3.1}$$

where, u and v are the x and y components of the displacement. The stresses and strains are given by,

$$\begin{aligned} s &= \left\{s_x, s_y, t_{xy}\right\}^T \\ e &= \left\{e_x, e_y, g_{xy}\right\}^T \end{aligned} \tag{3.2}$$

There are two classes of plane elasticity problems: (1) plane stress, and (2) plane strain. Each of these conditions is described in the following sections.

3.3 Plane Stress Condition

When the structure is subjected to forces in its own plane, the state of deformations and stresses is called plane stress condition (Weaver *et al.*, 1984). If a plate is very thin and is only subjected to inplane forces, then the displacements and stresses normal to the plane of the plate are negligible. Assuming the thin plate is in the xy plane, the stresses $s_z = 0, t_{yz} = 0$, $t_{xz} = 0$ and, $e_z \neq 0$ (Cook *et al.*, 1989). For isotropic material properties the stress-strain relationship for the plane stress condition is,

$$\begin{Bmatrix} s_x \\ s_y \\ t_{xy} \end{Bmatrix} = [\boldsymbol{E}] \begin{Bmatrix} e_x \\ e_y \\ e_z \end{Bmatrix} \qquad (3.3)$$

where, $[\boldsymbol{E}]$ is the material matrix and can be expressed as,

$$[\boldsymbol{E}] = \frac{E}{1-n^2} \begin{pmatrix} 1 & n & 0 \\ n & 1 & 0 \\ 0 & 0 & \frac{1-n}{2} \end{pmatrix} \qquad (3.4)$$

3.4 Plane Strain Condition

When a prismatic solid is subjected to a uniform load normal to its axis and the solid is divided into thin plates then each plate will have inplane forces, i.e., the forces will be in the direction of the plane of the plate (Weaver *et al.*, 1984). This condition is called the plane strain condition. For the plane strain condition $e_z = 0, e_{yz} = 0, g_{zx} = 0$, and $s_z \neq 0$ (Cook *et al.*, 1989). The material matrix $[\boldsymbol{E}]$, for the plane strain condition for an isotropic material is given by,

$$[\boldsymbol{E}] = \frac{E}{(1+n)(1-2n)} \begin{pmatrix} 1-n & 0 & 0 \\ n & 1-n & 0 \\ 0 & 0 & \dfrac{1-2n}{2} \end{pmatrix} \tag{3.5}$$

where,

E = modulus of elasticity of the material and,

n = Poisson’s ratio.

3.5 Constant Strain Triangle

The simplest triangular plane stress element is the constant strain triangle. This element has two inplane degrees of freedom at each node for a total of six degrees of freedom per element. The constant strain triangle is widely used for various analysis purposes. The nodes of the CST element are numbered in a counterclockwise direction as shown in Fig. 3.1.

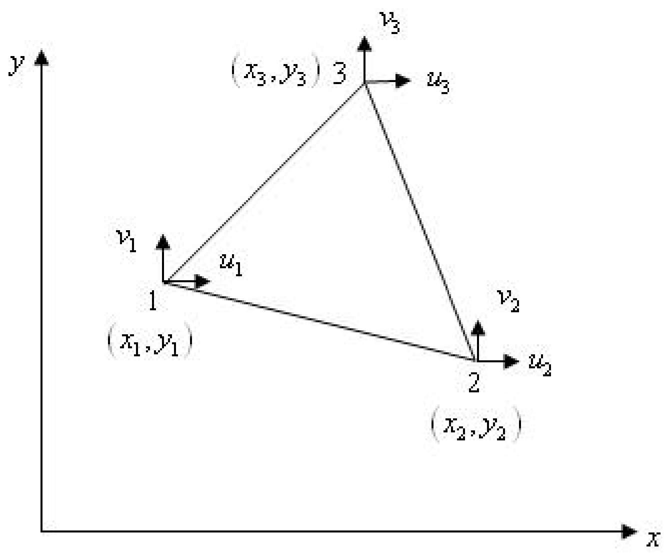


Fig. 3.1 Constant Strain Triangle (CST).

The procedure for developing the stiffness matrix of the CST element is as follows (Cook, 1974),

The assumed displacement field can be defined by,

$$u(x,y) = a_0 + a_1 x + a_2 y$$
$$v(x,y) = a_3 + a_4 x + a_5 y \qquad (3.6)$$

where $u(x,y)$ is the displacement in the x direction, and $v(x,y)$ is the displacement in the y direction.

The above equations can be written in matrix form as,

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} 1 & x & y & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x & y \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{Bmatrix}$$

or (3.7)

$$\{U(x,y)\} = [X]\{a\}$$

From Equation (3.3) and the strain-displacement relationships the following results can be obtained,

$$e_x(x,y) = \frac{\partial u}{\partial x} = a_1 \qquad (3.8)$$

$$e_y(x,y) = \frac{\partial v}{\partial y} = a_5 \qquad (3.9)$$

$$g_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = a_2 + a_4 \tag{3.10}$$

It is seen from the above equations that, terms a_0 and a_3 represents rigid body translations of the system while, the term a_1 represents a constant strain in the x direction, and the term a_5 represents a constant strain in the y direction. From Equation (3.10) it can be determined that the term $a_2 + a_4$ represents a uniform shear strain. Also from Equations (3.8), (3.9) and (3.10) it is observed that the strains are independent of x and y and are constant over the element. This is why this element is called the constant strain triangle.

As shown in Fig. 3.1, the coordinates of nodes 1, 2 and 3 are (x_1, y_1), (x_2, y_2) and (x_3, y_3) respectively. The corresponding displacements at each node are (u_1, v_1), (u_2, v_2) and, (u_3, v_3) respectively. Substituting the values of nodal coordinates, Equation (3.7) results in the following:

$$\begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_1 & y_1 \\ 1 & x_2 & y_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_2 & y_2 \\ 1 & x_3 & y_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x_3 & y_3 \end{bmatrix} \begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{Bmatrix} \tag{3.11}$$

or

$$\{u\} = [A]\{a\} \tag{3.12}$$

The coefficients $\{a\}$ are obtained by inverting Equation (3.12),

$$\{a\} = [A]^{-1}\{u\} \tag{3.13}$$

From Equation (3.7),

$$\{U(x,y)\} = [X]\{a\}$$

Therefore,

$$\{U(x,y)\} = [X][A]^{-1}\{u\} \tag{3.14}$$

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

$$[N] = [X][A]^{-1} \tag{3.15}$$

Inverting the $[A]$ matrix in Equation (3.11) and solving for $\{a\}$ gives,

$$\begin{Bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{Bmatrix} = \frac{1}{2A}\begin{bmatrix} x_2y_3 - x_3y_2 & 0 & x_3y_1 - x_1y_3 & 0 & x_1y_2 - x_2y_1 & 0 \\ y_2 - y_3 & 0 & y_3 - y_1 & 0 & y_1 - y_2 & 0 \\ x_3 - x_2 & 0 & x_1 - x_3 & 0 & x_2 - x_1 & 0 \\ 0 & x_2y_3 - x_3y_2 & 0 & x_3y_1 - x_1y_3 & 0 & x_1y_2 - x_2y_1 \\ 0 & y_2 - y_3 & 0 & y_3 - y_1 & 0 & y_1 - y_2 \\ 0 & x_3 - x_2 & 0 & x_1 - x_3 & 0 & x_2 - x_1 \end{bmatrix}\begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} \tag{3.16}$$

where, A is the area of the triangle and can be expressed as,

$$A = \frac{1}{2}\left[x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)\right] \tag{3.17}$$

The shape functions are obtained by combining Equations (3.15) and (3.16).

$$\begin{bmatrix} N_1 \\ N_2 \\ N_3 \end{bmatrix} = \frac{1}{2A} \begin{bmatrix} (x_2 y_3 - x_3 y_2) + (y_2 - y_3)x + (x_3 - x_2)y \\ (x_3 y_1 - x_1 y_3) + (y_3 - y_2)x + (x_1 - x_3)y \\ (x_1 y_2 - x_2 y_1) + (y_1 - y_2)x + (x_2 - x_1)y \end{bmatrix} \tag{3.18}$$

The displacements can now be rewritten in terms of the shape functions as,

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{Bmatrix} u(x,y) \\ v(x,y) \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} \tag{3.19}$$

The strains are obtained from the strain-displacement relationships.

$$e_x = \frac{\partial u}{\partial x} = \frac{\partial}{\partial x} \sum_{i=1}^{3} \left(N_i(x,y) u_i \right) \tag{3.20}$$

$$e_y = \frac{\partial v}{\partial y} = \frac{\partial}{\partial y} \sum_{i=1}^{3} \left(N_i(x,y) v_i \right) \tag{3.21}$$

$$g_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = \frac{\partial}{\partial y} \left(\sum_{i=1}^{3} (N_i u_i) \right) + \frac{\partial}{\partial x} \left(\sum_{i=1}^{3} (N_i v_i) \right) \tag{3.22}$$

The above equations can be written in matrix form as,

$$\begin{Bmatrix} e_x \\ e_y \\ g_{xy} \end{Bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} \tag{3.23}$$

or

$$\{e(x,y)\}_{3\times 1} = [B(x,y)]_{3\times 6}\{u_i\}_{6\times 1}$$

Taking the derivatives of the shape functions (Equation 3.18) with respect to x and y, gives the strain-displacement matrix $[B]$,

$$[B(x,y)] = \frac{1}{2A}\begin{bmatrix} y_2 - y_3 & 0 & y_3 - y_1 & 0 & y_1 - y_2 & 0 \\ 0 & x_3 - x_2 & 0 & x_1 - x_3 & 0 & x_2 - x_1 \\ x_3 - x_2 & y_2 - y_3 & x_1 - x_3 & y_3 - y_1 & x_2 - x_1 & y_1 - y_2 \end{bmatrix} \quad (3.24)$$

The element stiffness matrix can now be obtained using the strain-displacement matrix $[B]$ and the material matrix $[E]$.

$$[k]_{6\times 6} = \int_v [B]^T_{6\times 3}[E]_{3\times 3}[B]_{3\times 6}\, dv \quad (3.25)$$

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

$$[k]_{6\times 6} = t\int_A [B]^T_{6\times 3}[E]_{3\times 3}[B]_{3\times 6}\, dA \quad (3.26)$$

where,

t = thickness of the element.

Since all the terms in the strain-displacement matrix $[B]$ and the material matrix $[E]$ are constant, Equation (3.26) can be rewritten as,

$$[k]_{6\times 6} = t[B]^T_{6\times 3}[E]_{3\times 3}[B]_{3\times 6}\int_A dA$$

or

$$[k]_{6\times 6} = tA[B]^T_{6\times 3}[E]_{3\times 3}[B]_{3\times 6} \quad (3.27)$$

3.6 Implementation of the CST Element in Java

To implement the CST element in Java, a class called `CSTElement` was developed. The `CSTElement` class is derived from the base class `Element`. The base class `Element` contains the data variables and functions necessary for managing all of the different elements in the program.

The class definition and the variables declared within the `CSTElement` class are given below,

```
public class CSTElement {

  private double th; //Thickness of the element.
  private double[] EX; //x Coordinates at each node of the element.
  private double[] EY; //y Coordinates at each node of the element.
  private double[][] D; //Material Matrix.
  private double A; //Area of the element.
  private double[][] B; //Strain-Displacement Matrix.
  public double[][] CSTKelem; //Element stiffness matrix.
```

The variable `th` represents the thickness of the element, `[]EX` and `[]EY` are arrays containing the x and y coordinates of the nodes. `A` is the area of the element. The two dimensional arrays `D`, `B` and `CSTKelem` contains the material matrix, the strain-displacement matrix and the element stiffness matrix respectively. All of these instance variables except the two dimensional array `CSTKelem` representing the element stiffness matrix, are declared as private.

The constructor of the `CSTElement` is,

```
public  CSTElement(double[]  cx,  double[]  cy,  double[][]   Mat,  double
                thelem) {
   th = thelem;
   EX = cx;
   EY = cy;
   D = Mat;
}
```

The coordinates of the nodes, the material matrix, and the thickness of the element are all initialized through the constructor when an instance of the class `CSTElement` is created.

The methods of the class `CSTElement` are given in Table 3.1. All methods are declared private except for the `CalcElemK ()` method, which is declared as public since it is called from outside the class. The method `Area ()` computes the area of triangle using Equation (3.17). The `CSTBMatrix ()` method computes the B matrix for the element using Equation (3.24). The element stiffness matrix is computed in the method `ElementKMatrix ()` using the relationship from Equation (3.27).

Table 3.1 Methods in the class `CSTElement`

Method	Description
`Area ()`	Calculates the area of triangle.
`CSTBMatrix ()`	Calculates the strain-displacement matrix.
`ElementKMatrix ()`	Calculate the element stiffness matrix for CST element.

The structure of the `CalcElemK ()` method is shown below. This method takes `CSTElement` as an argument and calls the `Area ()`, `CSTBMatrix ()` and `ElementKMatrix ()` methods.

```
public void CalcElemK(CSTElement c) {

  //Calculates Area of the Triangular element.
  c.Area();

  //Method for calculating Strain-Displacement Matrix.
  c.CSTBMatrix();
  //Method for calculating Element Stiffness Matrix.
  c.ElementKMatrix();

}
```

The approach used for analyzing a structure consisting of CST elements is as follows. For each CST element an instance of the `CSTElement` class is created by calling the constructor.

```
CSTElement cst = new CSTElement (X, Y, D, TH);
```

where, `cst` is the instance of the class `CSTElement`. The x and y coordinates of the nodes, the material matrix and the thickness of the element are passed in the above constructor. Once an object of the `CSTElement` class has been created the element stiffness matrix is computed by calling the method `CalcElemK ()` of the main class by passing the `cst` object as an argument to the function.

The `CSTElement` class also contains the method `CalcStresses ()`. This method calculates stresses for the CST element. These stresses are obtained from the nodal displacements which are computed during the analysis. The `CalcStresses ()` method is declared as,

```
public void CalcStresses(CSTElement cst, double[] U, StressResults S) {
```

The parameter `cst` is an instance of the `CSTElement` class, `U` is the array of nodal displacements and `S` is an instance of the `StressResults` class which stores element stresses.

3.7 Four Node Quadrilateral Plane Stress Element

The quadrilateral plane stress element used in this study is the four node isoparametric quadrilateral element. The four node quadrilateral element (see Fig. 3.2) has two degrees of freedom per node for a total of eight degrees of freedom per element. The formulation of the element stiffness matrix (Cook, 1974) for the four node quadrilateral plane stress element (QUAD4 Element) is described below.

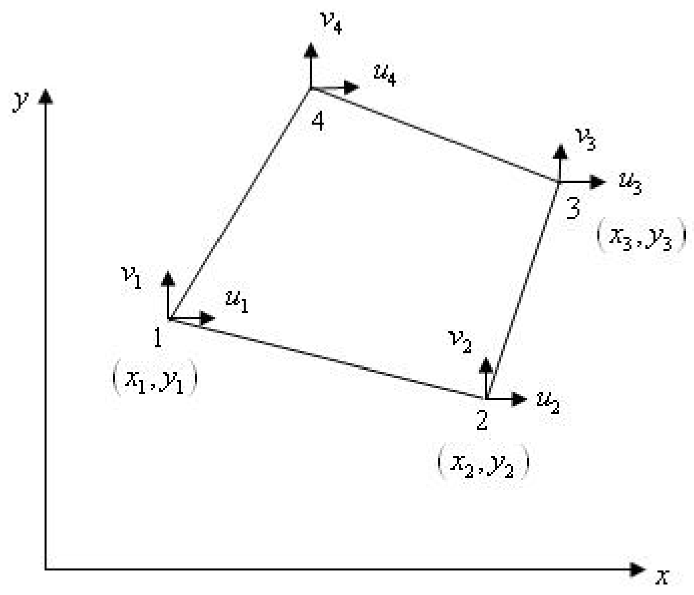


Fig. 3.2 Four Node Quadrilateral Plane Stress Element.

To develop the isoparamatric quadrilateral plane stress element, the master or parent element must be defined in the natural coordinate system (x,h) as shown in Fig. (3.3).

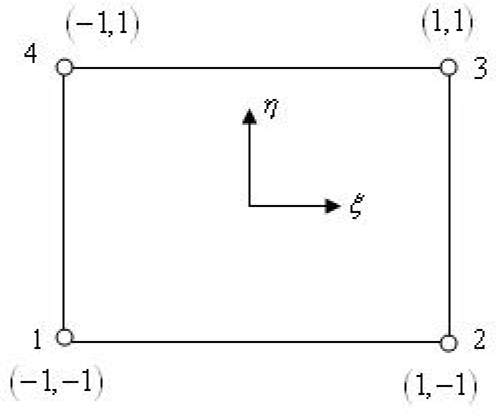


Fig. 3.3 Four Node Quadrilateral Element in Natural Coordinate System.

The relationship between the natural coordinate system and the global coordinate system can be defined using Lagrange interpolating functions.

$$x(x,h) = \sum_{i=1}^{4} (N_i x_i) \tag{3.28}$$

and,

$$y(x,h) = \sum_{i=1}^{4} (N_i y_i) \tag{3.29}$$

Similarly, the relationship between displacements in the natural coordinate system and the nodal displacements can be written in the following manner,

$$u(x,h) = \sum_{i=1}^{4} (N_i u_i) \tag{3.30}$$

$$v(x,h) = \sum_{i=1}^{4} (N_i v_i) \tag{3.31}$$

where N_1, N_2, N_3, N_4 are the shape functions for the four node quadrilateral element in the natural coordinate system. The shape functions are,

$$N_1 = \frac{1}{4}(1 - x)(1 - h).$$

$$N_2 = \frac{1}{4}(1 + x)(1 - h). \tag{3.32}$$

$$N_3 = \frac{1}{4}(1 + x)(1 + h).$$

$$N_4 = \frac{1}{4}(1 - x)(1 + h).$$

To obtain the element stiffness matrix, the strain-displacement matrix must be determined. When using isoparametric elements the element geometry is defined in the natural coordinate system and hence the strain displacement matrix must be transformed to natural coordinates. The transformation matrix used to convert the strain-displacement matrix from the element local coordinate system to the natural coordinate system is called the Jacobian matrix. The Jacobian matrix can be defined as,

$$[J] = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \tag{3.33}$$

The determinant of the Jacobian is,

$$|J| = J_{11}J_{22} - J_{12}J_{21} \tag{3.34}$$

and the inverse of the Jacobian matrix is,

$$[J]^{-1} = \frac{1}{|J|}\begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} \tag{3.35}$$

From Equations (3.28) and (3.29) the terms in the Jacobian matrix can be obtained as,

$$J_{11} = \frac{\partial x}{\partial x} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial x} x_i$$

$$J_{12} = \frac{\partial y}{\partial x} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial x} y_i$$

$$J_{21} = \frac{\partial x}{\partial h} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial h} x_i \tag{3.36}$$

$$J_{22} = \frac{\partial y}{\partial h} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial h} y_i$$

Thus, the strain-displacement relationships for the four node isoparamatric quadrilateral element are,

$$\begin{Bmatrix} e_x \\ e_y \\ g_{xy} \end{Bmatrix} = \begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial v}{\partial x} \end{Bmatrix} \tag{3.37}$$

The derivatives of the horizontal displacement with respect to x and y in terms of Jacobian matrix are,

$$\begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial u}{\partial y} \end{Bmatrix} = \frac{1}{|J|} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} \begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial u}{\partial h} \end{Bmatrix} \tag{3.38}$$

Similarly, the derivatives of the vertical displacement with respect to x and y are,

$$\begin{Bmatrix} \dfrac{\partial v}{\partial x} \\ \dfrac{\partial v}{\partial y} \end{Bmatrix} = \frac{1}{|J|} \begin{bmatrix} J_{22} & -J_{12} \\ -J_{21} & J_{11} \end{bmatrix} \begin{Bmatrix} \dfrac{\partial v}{\partial x} \\ \dfrac{\partial v}{\partial h} \end{Bmatrix} \tag{3.39}$$

From Equations (3.37), (3.38), and (3.39), the strain-displacement relationships can be obtained in terms of natural coordinates x and h as,

$$\begin{Bmatrix} e_x \\ e_y \\ g_{xy} \end{Bmatrix} = [A] \begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial u}{\partial h} \\ \dfrac{\partial v}{\partial x} \\ \dfrac{\partial v}{\partial h} \end{Bmatrix} \tag{3.40}$$

where,

$$[A] = \frac{1}{|J|} \begin{bmatrix} J_{22} & -J_{12} & 0 & 0 \\ 0 & 0 & -J_{21} & J_{11} \\ -J_{21} & J_{11} & J_{22} & -J_{12} \end{bmatrix} \tag{3.41}$$

Now, from Equations (3.30) and (3.31) the following relationships are obtained,

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

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

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

$$\frac{\partial v}{\partial h} = \sum_{i=1}^{4} \frac{\partial N_i}{\partial h} v_i \tag{3.42}$$

Equation (3.42), can be rewritten in matrix form as,

$$\begin{Bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial h} \\ \frac{\partial v}{\partial x} \\ \frac{\partial v}{\partial h} \end{Bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_4}{\partial x} & 0 \\ \frac{\partial N_1}{\partial h} & 0 & \frac{\partial N_2}{\partial h} & 0 & \frac{\partial N_3}{\partial h} & 0 & \frac{\partial N_4}{\partial h} & 0 \\ 0 & \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_4}{\partial x} \\ 0 & \frac{\partial N_1}{\partial h} & 0 & \frac{\partial N_2}{\partial h} & 0 & \frac{\partial N_3}{\partial h} & 0 & \frac{\partial N_4}{\partial h} \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{Bmatrix} \tag{3.43}$$

Now, using Equations (3.40) to (3.43) the following relationship are obtained,

$$\{e\} = [A][G]\{u\} \tag{3.44}$$

where,

$$[G] = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_4}{\partial x} & 0 \\ \frac{\partial N_1}{\partial h} & 0 & \frac{\partial N_2}{\partial h} & 0 & \frac{\partial N_3}{\partial h} & 0 & \frac{\partial N_4}{\partial h} & 0 \\ 0 & \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_4}{\partial x} \\ 0 & \frac{\partial N_1}{\partial h} & 0 & \frac{\partial N_2}{\partial h} & 0 & \frac{\partial N_3}{\partial h} & 0 & \frac{\partial N_4}{\partial h} \end{bmatrix} \tag{3.45}$$

Thus,

$$\{e\} = [B]\{u\} \tag{3.46}$$

where,

$$[B] = [A][G] \tag{3.47}$$

Equation (3.47) represents the strain-displacement matrix for the four node isoparamatric quadrilateral element.

The element stiffness matrix for the four node isoparamatric quadrilateral element for the plane stress condition is given by,

$$[k] = t\iint_A [B]^T [E][B]\,dA \tag{3.48}$$

Since the strain displacement matrix is in terms of the natural coordinates, therefore Equation (3.48) must be integrated with respect to the natural coordinates.

Substituting for dA,

$$dA = dx \cdot dy = |J|\,dx \cdot dh \tag{3.49}$$

Therefore, Equation (3.49) can be rewritten as,

$$[k]_{8\times8} = t \cdot \int_{-1}^{1}\int_{-1}^{1} \left[B(x,h)\right]_{8\times3}^{T} [E]_{3\times3} \left[B(x,h)\right]_{3\times8} \left|J(x,h)\right| dx\,dh \tag{3.50}$$

where, t = thickness of the element.

The element stiffness matrix can be obtained using 2×2 Gauss quadrature,

$$[k]_{8\times8} = t\sum_{j=1}^{2}\sum_{i=1}^{2} w_j w_i \left[B(x_i,h_j)\right]_{8\times3}^{T} [E]_{3\times3} \left[B(x_i,h_j)\right]_{3\times8} \left|J(x_i,h_j)\right| dx\,dh \tag{3.51}$$

The roots and weight functions for 2×2 Gauss quadrature are given in Table 3.2

Table 3.2 Roots and weight functions for 2×2 Gauss quadrature

Roots	**Weight Functions** w
±0.577350269189626	1.0

3.8 Implementation of the Four Node Quadrilateral Plane Element in Java

To implement the four node quadrilateral plane element in Java, a class called `QUAD4Element` was developed. The `QUAD4Element` class is derived from the base class `Element`.

The class definition and the variables declared within the `QUAD4Element` class are given below,

```
public class QUAD4Element {

  private double th; //Thickness of the Element.
  private double[] EX; //x Coordinates for each node.
  private double[] EY; //y Coordinates for each node.
  private double[][] D; //3x3 Material matrix.
  private double[] NXi; //Derivatives of shape functions w.r.t Xi.
  private double[] NEta; //Derivaitives of  the shape  functions w.r.t
                           Eta.
  private double[][] Jac; //2x2 Jacobain martrix.
  private double DJac; //Determinant of Jacobian matrix.
  private double[][] B; //Strain-displacement Matrix.
  public double[][] Q4Kelem; //Element Stiffness matrix.
```

The variable `th` represents the thickness of the element, `[]EX` and `[]EY` are arrays containing the x and y coordinates of the nodes. Arrays `[] NXi` and [] `NEta` represents the derivatives of the shape functions with respect to x and h respectively. The two dimensional arrays `Jac`, `D`, `B` and `Q4Kelem` contains the Jacobian matrix, the material matrix, the strain-displacement matrix and the element stiffness matrix respectively. All of these instance variables except the two dimensional array `Q4Kelem` representing the element stiffness matrix, are declared as private.

The constructor of the `Quad4Element` class is,

```
public QUAD4Element(double[] cx, double[] cy, double[][] MatD, double
                          thelem) {

    D = MatD;
    th = thelem;
    EX = cx;
    EY = cy;
  }
```

The coordinates of the nodes, the material matrix, and the thickness of the element are all initialized through the constructor when an instance of the class `QUAD4Element` is created.

The methods of the class `QUAD4Element` are given in Table 3.3. All methods are declared private except for the `CalcElemK ()` method, which is declared as public since it is called from outside the class. The method `QUAD4ShapeFn ()` computes the derivatives of the shape functions with respect to x and h. The `Jacobian ()` method computes the Jacobian matrix and the determinant of the Jacobian matrix for the element using Equation (3.36). The `QUAD4BMatrix ()` method computes the B matrix for the four node quadrilateral element.

Table 3.3 Methods in the class `QUAD4Element`

Method	Description
`QUAD4ShapeFn ()`	Calculates the derivatives of shape functions for four node quadrilateral element.
`Jacobian ()`	Calculates the Jacobian matrix.
`QUAD4BMatrix ()`	Calculates the strain-displacement matrix.
`QUAD4ElemMatrix ()`	Calculates the element stiffness matrix for four node quadrilateral plane element.

The element stiffness matrix is computed in the method `QUAD4ElemMatrix ()` using the 2×2 Gauss quadrature. For each Gauss point the methods `QUAD4ShapeFn ()`, `Jacobian ()`, and `QUAD4BMatrix ()` are called. The elements of stiffness matrices are computed at each Gauss point using the values obtained from these methods and are then added to obtain the final element stiffness.

The structure of the `CalcElemK ()` method is shown below. This method takes `QUAD4Element` as an argument and calls the `QUAD4ElemMatrix ()` method.

```
public void CalcElemK(QUAD4Element q) {

  //Calculates Element Stiffness Matrix.
  q.QUAD4ElemMatrix(q);

}
```

The approach used for analyzing a structure consisting of four node quadrilateral plane elements is as follows. For each element an instance of the `QUAD4Element` class is created by calling the constructor.

```
QUAD4Element q4 = new QUAD4Element (X, Y, D, TH);
```

where, `q4` is the instance of the class `QUAD4Element`. The x and y coordinates, material matrix and the thickness of the element are passed in the above constructor. Once an object of the `QUAD4Element` class has been created, the element stiffness matrix is computed by calling `CalcElemK ()` of the main class and passing the `q4` object as an argument to the function.

The `QUAD4Element` class also contains the method `CalcStresses ()`. This method calculates stresses for the four node quadrilateral plane element. These stresses are obtained from the nodal displacements which are computed during the analysis. The `CalcStresses ()` method is declared as,

```
public void CalcStresses(QUAD4Element q4, double[] U, StressResults S)
```

The parameter `q4` is an instance of the `QUAD4Element` class, `U` is the array of nodal displacements and `S` is an instance of the `StressResults` class which stores element stresses.

The stresses at each node of the element are calculated using 2×2 Gauss quadrature. The methods `QUAD4ShapeFn ()`, `Jacobian ()` and `QUAD4BMatrix ()` methods are called at each Gauss point and the stresses are calculated using the stress-strain and strain-displacement relationships.

Chapter 4
Plate Bending Elements

4.1 Overview

Two plate bending elements are chosen for representing the bending behavior of flat shell elements in this study: (1) Discrete Kirchoff Triangular (DKT) element (Batoz *et al.*, 1980), and (2) Discrete Kirchoff Quadrilateral (DKQ) element (Batoz and Tahar, 1982). In this chapter, the development of element stiffness matrix of these elements and the implementation in Java is discussed.

4.2 Bending of Flat Plates

Bending of flat plates is similar to bending of beams; the former is more complicated because plate bending is two dimensional while the bending of beam is one dimensional. The behavior of plates mainly depends on the plate thickness. Plates can be classified in to three categories depending on thickness and deformation (Timoshenko and Krieger, 1959).

1. Thin plates with small deformations.
2. Thin plates with large deformations.
3. Thick plates.

In this study we consider thin plates with small deformations. The bending properties of this plate can be used in the development of flat shell elements. There are three basic assumptions in the theory of bending for thin plates (Timoshenko and Krieger, 1959).

1. The mid-surface of the plate remains unstretched during deformations.
2. Points straight and normal to the mid-surface of the plate before bending remain straight and normal to the mid surface after bending.

3. Transverse shear stresses are small compared to normal stresses and hence can be neglected.

These assumptions are known as Kirchoff's hypothesis and are applicable to the bending of the thin plates with small deflections.

Consider an isotropic plate of uniform thickness t with the XY plane as the principal plane. According to the theory of bending for a thin plate, the plate is in the plane stress condition and hence all stresses vary linearly over the thickness of the plate.

The moments can be represented as,

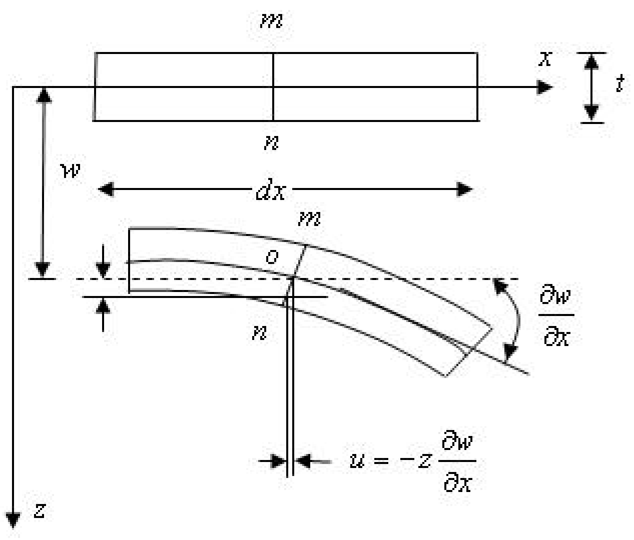
$$\begin{aligned} M_x &= \int_{-t/2}^{t/2} s_x z \cdot dz \\ M_y &= \int_{-t/2}^{t/2} s_y z \cdot dz \qquad (4.1) \\ M_{xy} &= \int_{-t/2}^{t/2} t_{xy} z \cdot dz \end{aligned}$$

where M_x and M_y are the moments in the x and y direction respectively, and M_{xy}, is the twisting moment. All moments are per length. If w is the transverse displacement of the plate, the displacement-curvature relationships for the thin plate can be written as,

$$\begin{aligned} k_x &= -\frac{\partial^2 w}{\partial x^2} \\ k_y &= -\frac{\partial^2 w}{\partial y^2} \qquad (4.2) \\ k_{xy} &= -2\frac{\partial^2 w}{\partial x \partial y} \end{aligned}$$

4.3 Basic Relationships for Bending of Thin Plates

Consider a small section of the plate of length dx in the x direction. When a load is applied in the z direction, the point O on the mid-surface of the plate moves in z direction as the plate deforms due to bending, as shown in the Fig.4.1.



m
x
t
n
w
dx
m
o
n
$\frac{\partial w}{\partial x}$
$u = -z\frac{\partial w}{\partial x}$
z

Fig. 4.1 Bending of Plate.

According to the Kirchoff assumption, a line that is straight and normal to the mid-surface before bending remains straight and normal to the mid-surface after bending (Cook *et al.*, 1989). The displacements can be written as,

$$u = -z\frac{\partial w}{\partial x}$$
$$v = -z\frac{\partial w}{\partial y} \qquad (4.3)$$

From Equation (4.3), and the strain-displacement relationship, the strains can be written as,

$$e_x = \frac{\partial u}{\partial x} = -z\frac{\partial^2 w}{\partial x^2}$$

$$e_y = \frac{\partial v}{\partial y} = -z\frac{\partial^2 w}{\partial y^2} \tag{4.4}$$

$$g_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} = -2z\frac{\partial^2 w}{\partial x \partial y}$$

For the plane-stress condition the stress-strain relationship is,

$$\begin{Bmatrix} s_x \\ s_y \\ t_{xy} \end{Bmatrix} = \frac{E}{1-n^2}\begin{bmatrix} 1 & n & 0 \\ n & 1 & 0 \\ 0 & 0 & \frac{1-n}{2} \end{bmatrix}\begin{Bmatrix} e_x \\ e_y \\ g_{xy} \end{Bmatrix} \tag{4.5}$$

Substituting for the strains from Equation (4.4), Equation (4.5) can be rewritten as,

$$\begin{Bmatrix} s_x \\ s_y \\ t_{xy} \end{Bmatrix} = \frac{E}{1-n^2}\begin{bmatrix} 1 & n & 0 \\ n & 1 & 0 \\ 0 & 0 & \frac{1-n}{2} \end{bmatrix}\begin{Bmatrix} -z\frac{\partial^2 w}{\partial x^2} \\ -z\frac{\partial^2 w}{\partial y^2} \\ -2z\frac{\partial^2 w}{\partial x \partial y} \end{Bmatrix} \tag{4.6}$$

From Equation (4.6) the stresses in the plate can be represented as,

$$s_x = \frac{-z \cdot E}{1-n^2}\left[\frac{\partial^2 w}{\partial x^2} + n\frac{\partial^2 w}{\partial y^2}\right]$$

$$s_y = \frac{-z \cdot E}{1-n^2}\left[n\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2}\right] \tag{4.7}$$

$$t_{xy} = \frac{-2z \cdot E}{1-n^2}\left(\frac{1-n}{2}\right)\left(\frac{\partial^2 w}{\partial x \partial y}\right)$$

Substituting the stresses from Equation (4.7) to Equation (4.1) and integrating over the thickness of the plate, we obtain following relationships for the moments,

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

$$M_y = \frac{-Et^3}{12\left(1-n^2\right)}\left[n\,\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2}\right] \qquad (4.8)$$

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

where, $\frac{Et^3}{12\left(1-n^2\right)} = D$ is the flexural rigidity of the plate.

Equation (4.8) can be represented in matrix form as,

$$\begin{Bmatrix} M_x \\ M_y \\ M_{xy} \end{Bmatrix} = \begin{bmatrix} D & D\cdot n & 0 \\ D\cdot n & D & 0 \\ 0 & 0 & D\left(\frac{1-n}{2}\right) \end{bmatrix} \begin{Bmatrix} -\frac{\partial^2 w}{\partial x^2} \\ -\frac{\partial^2 w}{\partial y^2} \\ -2\frac{\partial^2 w}{\partial x \partial y} \end{Bmatrix} \qquad (4.9)$$

From Equations (4.9) and (4.2), the moment-curvature relationship is given by,

$$\begin{Bmatrix} M_x \\ M_y \\ M_{xy} \end{Bmatrix} = \begin{bmatrix} D & D\cdot n & 0 \\ D\cdot n & D & 0 \\ 0 & 0 & D\left(\frac{1-n}{2}\right) \end{bmatrix} \begin{Bmatrix} k_x \\ k_y \\ k_{xy} \end{Bmatrix} \qquad (4.10)$$

4.4 Triangular Plate Bending Element Based on Discrete Kirchoff Theory

Batoz *et al.* (1980) developed a triangular plate bending element (DKT element) based on the discrete Kirchoff theory. According to the Kirchoff assumptions, the bending energy present in the element is much higher compared to the shear strain energy, and thus the transverse shear energy term can be neglected fom the energy equation. The DKT element is a widely used triangular plate bending element in finite element analysis programs. In this section the element stiffness matrix for the DKT element as given by Batoz *et al.* (1980) is presented. The DKT element is shown in Fig. 4.2,

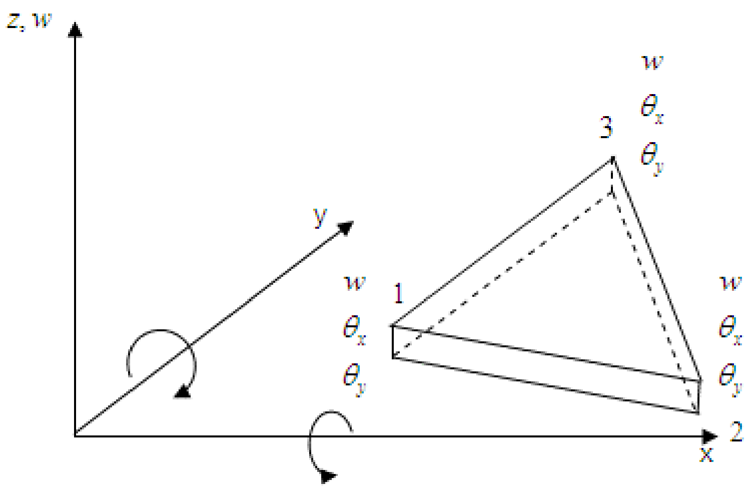


Fig. 4.2 DKT Element.

The bending energy can be represented in the following form,

$$U_b = \frac{1}{2}\int_A k^T D_b k \cdot dx \cdot dy \tag{4.11}$$

where,

$$D_b = \frac{Et^3}{12\left(1-n^2\right)}\begin{bmatrix} 1 & n & 0 \\ n & 1 & 0 \\ 0 & 0 & \frac{1-n}{2} \end{bmatrix}$$

where,

t = thickness of the plate, and

The curvatures are given by,

$$k = \begin{bmatrix} b_{x,x} \\ b_{y,y} \\ b_{x,y} + b_{y,x} \end{bmatrix} \tag{4.12}$$

According to the assumptions made in the bending theory of thin plates with small displacements, the displacement components u, v and w at any point can be represented as,

$$u = z b_x\left(x, y\right) \qquad v = z b_y\left(x, y\right) \qquad w = w\left(x, y\right) \tag{4.13}$$

where, w is the transverse displacement, and b_x and b_y are the rotations in the direction normal to the xz and yz planes respectively.

Batoz *et al.* (1980) made following observations to correlate rotations normal to the mid surface to the transverse displacement w.

(1) The triangular element should have only nine degrees of freedom; that is, the transverse displacement w and the rotations q_x and q_y at each node of the element.

(2) According to Kirchoff theory, the rotations can be defined as,

$$q_x = \frac{\partial w}{\partial x} \qquad q_y = \frac{\partial w}{\partial y} \tag{4.14}$$

(3) The Kirchoff theory can be imposed at any discrete point in the element.

(4) The compatibility of rotations b_x and b_y cannot be lost.

Batoz *et al.* (1980) made the following assumptions,

(1) The relationship between the rotations at six nodal points including mid-surface nodes and the shape functions at each six nodes is in the form of a quadratic,

$$b_x = \sum_{i=1}^{6} N_i b_{xi}$$

$$b_y = \sum_{i=1}^{6} N_i b_{yi} \tag{4.15}$$

where, b_{xi} and b_{yi} are the rotations at each node as shown in the Fig. 4.3.

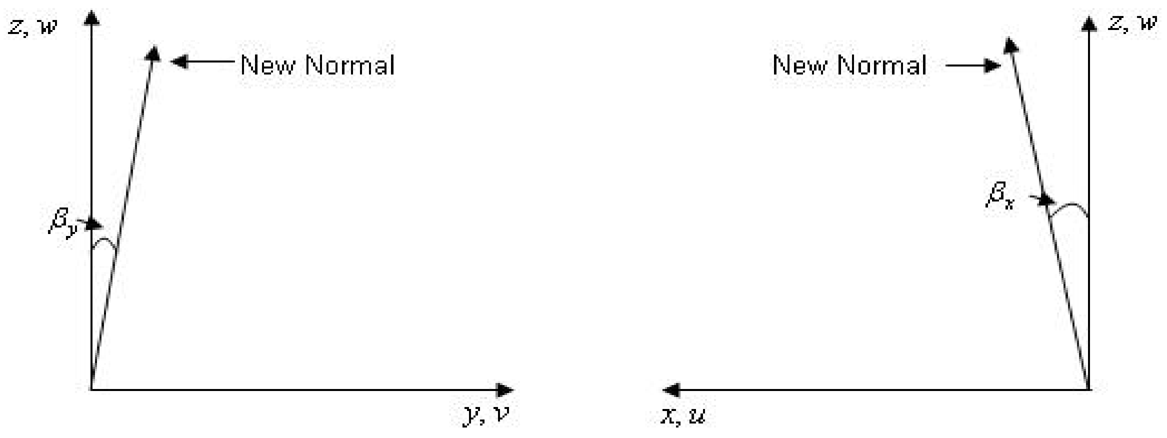


Fig. 4.3 Positive Directions of b_x and b_y.

For $i = 1$ to 6, N_i are the shape functions for the DKT element in area coordinates x and h , and can be expressed as,

$$
\begin{aligned}
N_1 &= 2(1 - x - h)(\tfrac{1}{2} - x - h).\\
N_2 &= x(2x - 1).\\
N_3 &= h(2h - 1).\\
N_4 &= 4xh.\\
N_5 &= 4h(1 - x - h).\\
N_6 &= 4x(1 - x - h).
\end{aligned}
\tag{4.16}
$$

(2) Applying Kirchoff hypothesis to remove transverse shear strain given the following equations,

At the corner nodes,

$$g = \begin{bmatrix} b_x + w,_x \\ b_y + w,_y \end{bmatrix} = 0 \qquad k = 1,2,3 \tag{4.17}$$

where, g = transverse shear strain.

At the mid nodes, $b_{sk} + w,_{sk} = 0 \qquad k = 4,5,6 \qquad (4.18)$

where k is the node number.

(3) The variation of the transverse displacements is represented by a cubic expression as,

$$w,_{sk} = -\frac{3}{2l_{ij}} w_i - \frac{1}{4} w,_{si} + \frac{3}{2l_{ij}} w_j + \frac{1}{4} w,_{sj} \tag{4.19}$$

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

(4) The variation of the rotations along the sides of the triangle is represented by linear equation,

$$b_{nk} = \tfrac{1}{2}\left(b_{ni} + b_{nj}\right) \tag{4.20}$$

where, $k = 4,5,6$ represents the mid-nodes of the sides 2-3, 3-1, and 2-1 respectively.

As a consequence of the above four assumptions, the condition that the transverse shear strain along the sides of the triangle, $g_s = b_s + w,_s = 0$ is satisfied.

The displacements at each node can be written as,

$$U^T = \left\{ w_1 \quad q_{x1} \quad q_{y1} \quad w_1 \quad q_{x\,2} \quad q_{y\,2} \quad w_2 \quad q_{x\,2} \quad q_{y\,2} \right\} \tag{4.21}$$

The relationship between the nodal displacements and b_x and b_y is given by,

$$\begin{aligned} b_x &= H_x^T\left(x,h\right)U \\ b_y &= H_y^T\left(x,h\right)U \end{aligned} \tag{4.22}$$

where H_x and H_y are the components vectors of the shape functions. Batoz *et al.* (1980) represented the component vectors of the shape functions in the form,

$$H_x(x,h) = \begin{bmatrix} 1.5(a_6 N_6 - a_5 N_5) \\ (b_5 N_5 + b_6 N_6) \\ N_1 - c_5 N_5 - c_6 N_6 \\ 1.5(a_4 N_4 - a_6 N_6) \\ (b_6 N_6 + b_4 N_4) \\ N_1 - c_6 N_6 - c_4 N_4 \\ 1.5(a_5 N_5 - a_4 N_4) \\ (b_4 N_4 + b_5 N_5) \\ N_3 - c_4 N_4 - c_5 N_5 \end{bmatrix} \tag{4.23}$$

$$H_y(x,h) = \begin{bmatrix} 1.5(d_6 N_6 - d_5 N_5) \\ -N_1 + e_5 N_5 + e_6 N_6 \\ -b_5 N_5 - b_6 N_6 \\ 1.5(d_4 N_4 - d_6 N_6) \\ -N_2 + e_6 N_6 + e_4 N_4 \\ -b_6 N_6 - b_4 N_4 \\ 1.5(d_5 N_5 - d_4 N_4) \\ -N_3 + e_4 N_4 + e_5 N_5 \\ -b_4 N_4 - b_5 N_5 \end{bmatrix} \tag{4.24}$$

where,

$$\begin{aligned} a_k &= -\frac{x_{ij}}{l_{ij}^2} \\ b_k &= \frac{3}{4}\frac{x_{ij} y_{ij}}{l_{ij}^2} \\ c_k &= \left(\tfrac{1}{4} x_{ij}^2 - \tfrac{1}{2} y_{ij}^2\right) \Big/ l_{ij}^2 \\ d_k &= -y_{ij}^2 \Big/ l_{ij}^2 \\ e_k &= \left(\tfrac{1}{4} y_{ij}^2 - \tfrac{1}{2} x_{ij}^2\right) \Big/ l_{ij}^2 \\ x_{ij} &= x_i - x_j \\ y_{ij} &= y_i - y_j \\ l_{ij}^2 &= \left(x_{ij}^2 + y_{ij}^2\right) \end{aligned} \tag{4.25}$$

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

The strain-displacement matrix for the DKT element can be represented in the following form,

$$B(x,h) = \frac{1}{2A}\begin{bmatrix} y_{31}H_{x,x}^{T} + y_{12}H_{x,h}^{T} \\ -x_{31}H_{y,x}^{T} - x_{12}H_{y,h}^{T} \\ -x_{31}H_{x,x}^{T} - x_{12}H_{x,h}^{T} + y_{31}H_{y,x}^{T} + y_{12}H_{y,h}^{T} \end{bmatrix} \tag{4.26}$$

where, $2A = x_{31}y_{12} - x_{12}y_{31}$

The derivatives of the component vectors of the shape functions with respect to x and h can be represented by the following equations.

The derivatives of the component vectors with respect to x are,

$$H_{x,x} = \begin{bmatrix} P_6(1-2x) + (P_5 - P_6)h \\ q_6(1-2x) - (q_5 + q_6)h \\ -4 + 6(x+h) + r_6(1-2x) - h(r_5 + r_6) \\ -P_6(1-2x) + h(P_4 + P_6) \\ q_6(1-2x) - h(q_6 - q_4) \\ -2 + 6x + r_6(1-2x) + h(r_4 - r_6) \\ -h(P_5 + P_4) \\ h(q_4 - q_5) \\ -h(r_5 - r_4) \end{bmatrix} \tag{4.27}$$

$$H_{y,x} = \begin{bmatrix} t_6(1-2x)+(t_5-t_6)h \\ 1+r_6(1-2x)-(r_5+r_6)h \\ -q_6(1-2x)-h(q_5+q_6) \\ -t_6(1-2x)+h(t_4+t_6) \\ -1+r_6(1-2x)-h(r_4-r_6) \\ q_6(1-2x)+h(q_4-q_6) \\ -h(t_4+t_5) \\ h(r_4-r_5) \\ -h(q_4-q_5) \end{bmatrix} \tag{4.28}$$

The derivatives of the component vectors with respect to h are,

$$H_{x,h} = \begin{bmatrix} -P_5(1-2h)-x(P_6-P_5) \\ q_5(1-2h)-x(q_5+q_6) \\ -4+6(x+h)+r_5(1-2h)-x(r_5+r_6) \\ x(P_4+P_6) \\ x(q_4-q_6) \\ -x(r_6-r_4) \\ P_5(1-2h)-x(P_4+P_5) \\ q_5(1-2h)+x(q_4-q_5) \\ -2+6h+r_5(1-2h)+x(r_4-r_5) \end{bmatrix} \tag{4.29}$$

$$H_{h,h} = \begin{bmatrix} -t_5(1-2h)-x(t_6-t_5) \\ 1+r_5(1-2h)-x(r_5+r_6) \\ -q_5(1-2h)+x(q_5+q_6) \\ x(t_4+t_6) \\ x(r_4-r_6) \\ -x(q_4-q_6) \\ t_5(1-2h)-x(t_4+t_5) \\ 1-r_5(1-2h)+x(r_4-r_5) \\ -q_5(1-2h)+x(q_4-q_5) \end{bmatrix} \tag{4.30}$$

where,

$$P_k = -6x_{ij} \Big/ l_{ij}^2$$

$$q_k = 3x_{ij} y_{ij} \Big/ l_{ij}^2$$

$$r_k = 3y_{ij}^2 \Big/ l_{ij}^2$$

$$t_k = -6y_{ij} \Big/ l_{ij}^2$$

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

The strain-displacement matrix can be calculated using Equations (4.26) through (4.30). Substituting the strain-displacement matrix in Equation (4.31), the element stiffness matrix for the DKT element can be obtained.

$$K_{DKT} = 2A\int_0^1 \int_0^{1-h} B^T D_b\, B \cdot dx\, dh \tag{4.31}$$

where D_b is the material matrix for plate bending.

It is assumed that the element has constant thickness. The element stiffness matrix for the DKT element can be obtained using a three point Gauss quadrature scheme. The three numerical integration points are located at the mid points of the sides of the triangle (Batoz *et al.*, 1980). Since the equation for the element stiffness matrix has quadratic terms a three point Gauss quadrature scheme is sufficient. The coordinates and the weight functions for the 3-point numerical integration scheme are given in Table 4.1.

Table 4.1 Coordinates and weight functions for Gauss quadrature

Integration point	Coordinates	Weight functions
1	$\left(\frac{1}{2},0\right)$	$\frac{1}{3}$
2	$\left(\frac{1}{2},\frac{1}{2}\right)$	$\frac{1}{3}$
3	$\left(0,\frac{1}{2}\right)$	$\frac{1}{3}$

The element stiffness matrix using Gauss quadrature is obtained from,

$$[k]_{9\times 9} = 2A\sum_{j=1}^{3}\sum_{i=1}^{3} w_j w_i \left[B(x_i, h_j)\right]_{9\times 3}^{T} [D_b]_{3\times 3} \left[B(x_i, h_j)\right]_{3\times 9} dx\,dh \tag{4.32}$$

4.5 Implementation of DKT Element in Java.

To implement the triangular plate bending (DKT) element in Java, a class called `DKTElement` was developed. The `DKTElement` class is derived from the base class `Element`. The base class `Element` contains all the data variables and functions necessary for managing all of the different elements in the program.

The class definition and the variables declared within the class are given below,

```
public class DKTElement {
  private double th; //Thickness of the element.
  private double A; //Area of the element.
  private double[] EX; //x Coordinates of the element.
  private double[] EY; //y Coordinates of the element.
  private double[][] Db; //Material matrix.
  private double[][] X; //x geometric components.
  private double[][] Y; //y geometric components.
  private double[][] L; //L geometric components.
  private double[] HXxi; //Component vector for x w.r.t Xi.
  private double[] HXeta; //Component vector for x w.r.t Eta.
  private double[] HYxi; //Component vector for y w.r.t Xi.
  private double[] HYeta; //Component vector for y w.r.t Eta.
  private double[] P; //Geometric components P at each node.
  private double[] t; //Geometric components t at each node.
  private double[] q; //Geometric components q at each node.
  private double[] r; //Geometric components r at each node.
  private double[][] B; //Strain-Displacement matrix.
  public double[][] DKTKelem; //Element Stiffness matrix.
```

The variable `th` and `A` represent the thickness and the area of the element respectively. `[]EX` and `[]EY` are arrays containing the x and y coordinates of the nodes. `[] X` and `[] Y` are arrays containing the geometric components representing the sides of the triangle. The array `[] L` contains lengths of the sides of triangle. `[] p`, `[] q`, `[] r` and `[] t` are arrays containing the geometric components for calculating the component vectors of shape functions. `[] HXxi`, `[] HXeta`, `[] HYxi`, and `[] HYeta` are arrays representing the derivatives of the component vectors for the shape functions with respect to x and h respectively. The two dimensional arrays `B` and `DKTKelem` contains the strain-displacement matrix and the element stiffness matrix respectively. All of these instance variables except the two dimensional array `DKTKelem` representing the element stiffness matrix, are declared as `private`.

The constructor of the `DKTElement` class is,

```
public DKTElement(double[] cx, double[] cy, double[][] Mat,
                        double thelem) {
  Db = Mat;
  th = thelem;
  EX = cx;
  EY = cy;
}
```

The coordinates of the nodes, the material matrix, and the thickness of the element are all initialized through the constructor when an instance of the class `DKTElement` is created.

The methods of the class `DKTElement` are given in Table 4.2, All methods are declared private except for the method `CalcElemK ()`, which is declared as public since it is called from outside the class. The method `Geometry()` computes the geometric components required to calculate the element stiffness matrix using Equation (4.25). The method `ShFnHXxi ()` computes derivatives of the component vectors for shape functions for x with respect to x using Equation (4.27). The method `ShFnHXeta ()` computes derivatives of the component vectors for the shape functions for x with respect to h using Equation (4.29). The method `ShFnHYxi ()` computes derivatives of the component vectors for shape functions for y with respect to x using Equation (4.28). The method `ShFnHYeta ()` computes derivatives of the component vectors for shape functions for y with respect to h using Equation (4.30). The method `DKTBMatrix ()` computes the B matrix for the DKT element using Equation (4.26).

Table 4.2 Methods in the class `DKTElement`

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

The element stiffness matrix is computed in the method `DKTElementKMatrix ()` using three point Gauss quadrature. For each Gauss point the `ShFnHXxi ()`, `ShFnHXeta ()`, `ShFnHYxi ()`, `ShFnHYeta ()`, and `DKTBMatrix ()` methods are called. The elements of the stiffness matrices are computed at each Gauss point using the values obtained from these methods and are then added to calculate final element stiffness matrix for the DKT element.

The structure of the `CalcElemK ()` method is shown below. This method takes `DKTElement` as an argument and calls the `Geometry ()` and `DKTElementKMatrix ()` methods.

```
 public void CalcElemK(DKTElement d) {

    //Calculates geometrical properties for DKT Element.
    d.Geometry();

    //Calculates Element Stiffness Matrix for DKT Element.
    d.DKTElementKMatrix(d);
  }
```

The approach used for analyzing a structure consisting of DKT elements is as follows. For each element an instance of the `DKTElement` class is created by calling the constructor.

```
DKTElement dkt = new DKTElement (X, Y, D, TH);
```

where, `dkt` is the instance of the class `DKTElement`. The x and y coordinates, material matrix and the thickness of the element are passed in the above constructor. Once an object of the `DKTElement` class has been created, the element stiffness matrix is computed by calling the method `CalcElemK ()` of the main class by passing the `dkt` object as an argument to the function.

The `DKTElement` class also contains the method `CalcStresses ()`. This method calculates stresses for a DKT element. These stresses are obtained from the nodal displacements which are computed during the analysis. The `CalcStresses ()` method is declared as,

```
public void CalcStresses(DKTElement dkt, double[] U, StressResults S)
```

The parameter `dkt` is an instance of the `DKTElement` class, `U` is the array of nodal displacements and `S` is an instance of the `StressResults` class which stores element stresses.

The stresses at each node of the element are calculated using three point Gauss quadrature. The methods `ShFnHXxi ()`, `ShFnHXeta ()`, `ShFnHYxi ()`, `ShFnHYeta ()`, and `DKTBMatrix ()` are called at each Gauss point and the stresses are calculated using the stress-strain and strain-displacement relationships.

4.6 Quadrilateral Plate Bending Element Based on Discrete Kirchoff Theory

The quadrilateral thin plate bending element is efficient and useful for representing the bending part of flat shell elements. The quadrilateral plate bending element can also be used for the analysis of plate structures such as slabs. Batoz and Tahar (1982) developed the Discrete Kirchoff Quadrilateral (DKQ) plate bending element by considering the Kirchoff assumptions for thin plates. The DKQ element has 12 degrees of freedom. Considering the element is in the xy plane (see Fig. 4.4), the degrees of freedom at each node of the element can be described as the transverse displacement w in the direction normal to the xy plane, and the inplane rotations q_x and q_y in the x and y directions respectively.

$$w = w(x,y) \qquad q_x = w,_y \qquad q_y = w,_x \tag{4.33}$$

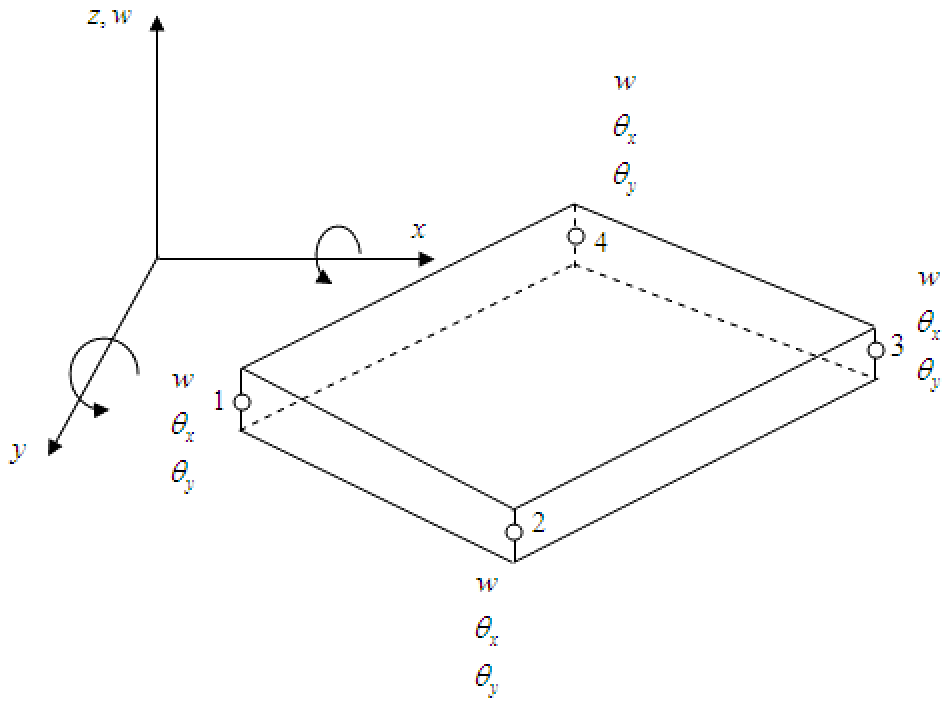


Fig. 4.4 Quadrilateral Plate Bending Element (After Batoz and Tahar, 1982).

The development of the DKQ element by Batoz and Tahar (1982) is described in this section. The formulation of the DKQ element is based on the Kirchoff assumptions which were discussed in Section 4.3. According to these assumptions the shear strain energy is neglected. The strain energy of the element is,

$$U = \sum_e U_e^b \tag{4.34}$$

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

$$U_e^b = \tfrac{1}{2} \int_{A^e} \langle c \rangle [D_b] \{c\} \, dxdy$$

and A^e is the area of an element.

For homogeneous isotropic material properties, the curvatures are given by,

$$\{c\} = \begin{Bmatrix} \partial b_x / \partial x \\ \partial b_y / \partial y \\ \partial b_x / \partial y + \partial b_y / \partial x \end{Bmatrix} \tag{4.35}$$

here,

b_x = rotation normal to the middle surface of the plate in the xz direction.

b_y = rotation normal to the middle surface of the plate in the yz direction.

The material matrix is,

$$D_b = \frac{Et^3}{12(1-n)^2} \begin{bmatrix} 1 & n & 0 \\ n & 1 & 0 \\ 0 & 0 & \frac{1-n}{2} \end{bmatrix} \tag{4.36}$$

where, n = Poisson’s ratio, t = thickness of the plate, and E = modulus of elasticity.

Batoz and Tahar (1982) developed the relationship between the transverse displacement w and the rotations b_x and b_y as follows,

1. They defined b_x and b_y by incomplete cubic polynomials:

$$b_x = \sum_{i=1}^{8} N_i b_{xi} \qquad b_y = \sum_{i=1}^{8} N_i b_{yi} \tag{4.37}$$

The shape functions for the eight node quadrilateral element are represented in the following form.

$N_1 = -\tfrac{1}{4}\left[(1-x)(1-h)(1+x+h)\right]$

$N_2 = -\tfrac{1}{4}\left[(1+x)(1-h)(1-x+h)\right]$

$$N_3 = -\tfrac{1}{4}\left[(1+x)(1+h)(1-x-h)\right]$$

$$N_4 = -\tfrac{1}{4}\left[(1-x)(1+h)(1+x-h)\right]$$

$$N_5 = \tfrac{1}{2}\left(1-x^2\right)(1-h)$$

$$N_6 = \tfrac{1}{2}(1+x)\left(1-h^2\right)$$

$$N_7 = \tfrac{1}{2}\left(1-x^2\right)(1+h)$$

$$N_5 = \tfrac{1}{2}\left(1-x^2\right)\left(1-h^2\right) \qquad (4.38)$$

2. They applied Kirchoff assumptions at the corner nodes and the midpoint of the sides,

At the corner nodes,

$$\begin{Bmatrix} b_{xi} + w,_{xi} \\ b_{yi} + w,_{yi} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} \qquad i = 1,2,3,4 \qquad (4.39)$$

At the midpoints:

$$b_{sk} + w,_{sk} = 0 \qquad k = 5,6,7,8 \qquad (4.40)$$

3. A cubic function was used to represent the transverse displacement w. Hence the derivative of the transverse displacement w with respect to s at the mid nodes of the element sides is a quadratic and is represented as,

$$w,_{sk} = \frac{-3}{2l_{ij}}\left(w_i - w_j\right) - \frac{1}{4}\left(w,_{si} + w,_{sj}\right) \qquad (4.41)$$

where,

$k = 5,6,7,8$ for $ij = 12,23,34,41$ and l_{ij} = length of the line for side connecting nodes ij

4. The rotation normal to the sides at the mid nodes varies linearly.

$$b_{nk} = \tfrac{1}{2}\left(b_{ni} + b_{nj}\right) = -\tfrac{1}{2}\left(w,_{ni} + w,_{nj}\right) \tag{4.42}$$

The nodal displacement vector for the Discrete Kirchoff Quadrilateral element is,

$$\left\langle U_n \right\rangle = \left\langle w_1 \quad q_{x1} \quad q_{y1} \quad w_2 \quad q_{x2} \quad q_{y2} \quad w_3 \quad q_{x3} \quad q_{y3} \quad w_4 \quad q_{x4} \quad q_{y4} \right\rangle \tag{4.43}$$

where,

$$q_{xi} = w,_{yi}$$
$$q_{yi} = w,_{xi} \qquad \text{for } i = 1,2,3,4$$

The quantities b_x and b_y are expressed in terms of the nodal displacements using the component vectors of the shape functions as,

$$b_x = \left\langle H^x\left(x,h\right)\right\rangle \left\{U_n\right\}$$
$$b_y = \left\langle H^y\left(x,h\right)\right\rangle \left\{U_n\right\} \tag{4.44}$$

where, $H^x\left(x,h\right)$ and $H^y\left(x,h\right)$ are the component vectors of the shape functions and are given in Equations (4.45) and (4.46),

$$\left\langle H^{x}(x,h)\right\rangle = \begin{bmatrix} \frac{3}{2}\left(a_5 N_5 - a_8 N_8\right) \\ b_5 N_5 + b_8 N_8 \\ N_1 - c_5 N_5 - c_8 N_8 \\ \frac{3}{2}\left(a_6 N_6 - a_5 N_5\right) \\ b_6 N_6 + b_5 N_5 \\ N_2 - c_6 N_6 - c_5 N_5 \\ \frac{3}{2}\left(a_7 N_7 - a_6 N_6\right) \\ b_7 N_7 + b_6 N_6 \\ N_3 - c_7 N_7 - c_6 N_6 \\ \frac{3}{2}\left(a_8 N_8 - a_7 N_7\right) \\ b_8 N_8 + b_7 N_7 \\ N_4 - c_8 N_8 - c_7 N_7 \end{bmatrix} \tag{4.45}$$

$$\left\langle H^{y}(x,h)\right\rangle = \begin{bmatrix} \frac{3}{2}\left(d_5 N_5 - d_8 N_8\right) \\ -N_1 - e_5 N_5 - e_8 N_8 \\ -b_5 N_5 - b_8 N_8 \\ \frac{3}{2}\left(d_6 N_6 - d_5 N_5\right) \\ -N_2 - e_6 N_6 - e_5 N_5 \\ -b_6 N_6 - b_5 N_5 \\ \frac{3}{2}\left(d_7 N_7 - d_6 N_6\right) \\ -N_3 - e_7 N_7 - e_6 N_6 \\ -b_7 N_7 - b_6 N_6 \\ \frac{3}{2}\left(d_8 N_8 - d_7 N_7\right) \\ -N_4 - e_8 N_8 - e_7 N_7 \\ -b_8 N_8 - b_7 N_7 \end{bmatrix} \tag{4.46}$$

with,

$$a_k = -\frac{x_{ij}}{l_{ij}^2}$$

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

$$
\begin{aligned}
c_k &= \left(\tfrac{1}{4}x_{ij}^2 - \tfrac{1}{2}y_{ij}^2\right)\Big/ l_{ij}^2 \\
d_k &= -y_{ij}^2 \Big/ l_{ij}^2 \\
e_k &= \left(\tfrac{1}{4}y_{ij}^2 - \tfrac{1}{2}x_{ij}^2\right)\Big/ l_{ij}^2 \\
x_{ij} &= x_i - x_j \\
y_{ij} &= y_i - y_j \\
l_{ij}^2 &= \left(x_{ij}^2 + y_{ij}^2\right)
\end{aligned}
\tag{4.47}
$$

$$k = 5,6,7,8 \quad \text{when} \quad ij = 12,23,34,41$$

The strain-displacement matrix is obtained from the component vectors of the shape functions as,

$$
[B] = \begin{bmatrix} \left\langle H^x,_x \right\rangle \\ \left\langle H^y,_y \right\rangle \\ \left\langle H^x,_y + H^y,_x \right\rangle \end{bmatrix} = \begin{bmatrix} j_{11}\left\langle H^x,_x \right\rangle + j_{12}\left\langle H^x,_h \right\rangle \\ j_{21}\left\langle H^y,_x \right\rangle + j_{22}\left\langle H^y,_h \right\rangle \\ j_{11}\left\langle H^y,_x \right\rangle + j_{12}\left\langle H^y,_h \right\rangle + j_{21}\left\langle H^x,_x \right\rangle + j_{22}\left\langle H^x,_h \right\rangle \end{bmatrix}
\tag{4.48}
$$

The Jacobian matrix is given by,

$$
[J] = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} = \frac{1}{4}\begin{bmatrix} x_{21} + x_{34} + h\left(x_{12} + x_{34}\right) & y_{21} + y_{34} + h\left(y_{12} + y_{34}\right) \\ x_{32} + x_{41} + x\left(x_{12} + x_{34}\right) & y_{32} + y_{41} + x\left(y_{12} + y_{34}\right) \end{bmatrix}
\tag{4.49}
$$

In the equation for the strain-displacement matrix (Equation 4.48), the terms $j_{11}, j_{12}, j_{21}, j_{22}$ are components of the inverse of the Jacobian matrix represented in Equation (4.49). These terms are obtained as follows,

$$j_{11} = \frac{1}{\det[J]} J_{22} \qquad j_{12} = \frac{-1}{\det[J]} J_{12}$$

$$j_{21} = \frac{-1}{\det[J]} J_{21} \qquad j_{22} = \frac{1}{\det[J]} J_{22} \tag{4.50}$$

The determinant of the Jacobian is,

$$\det[J] = \frac{1}{8}\left(y_{42}x_{31} - y_{31}x_{42}\right) + \frac{x}{8}\left(y_{34}x_{21} - y_{21}x_{34}\right) + \frac{h}{8}\left(y_{41}x_{32} - y_{32}x_{41}\right)$$

The derivatives for the component vectors $\left\langle H^{x},_{x} \right\rangle, \left\langle H^{x},_{h} \right\rangle, \left\langle H^{y},_{x} \right\rangle$, and $\left\langle H^{y},_{h} \right\rangle$ can be obtained by substituting the derivatives of the shape functions $N_{i,x}$ and $N_{i,h}$ respectively in place of the shape functions N_i.

The derivatives of the shape functions are,

$$N,_{x} = \begin{bmatrix} \frac{1}{4}(2x+h)(1-h) \\ \frac{1}{4}(2x-h)(1-h) \\ \frac{1}{4}(2x+h)(1+h) \\ \frac{1}{4}(2x-h)(1+h) \\ -x(1-h) \\ \frac{1}{2}(1-h^2) \\ -x(1+h) \\ -\frac{1}{2}(1-h^2) \end{bmatrix} \qquad N,_{h} = \begin{bmatrix} \frac{1}{4}(2h+x)(1-x) \\ \frac{1}{4}(2h-x)(1+x) \\ \frac{1}{4}(2h+x)(1+x) \\ \frac{1}{4}(2h-x)(1-x) \\ -\frac{1}{2}(1-x^2) \\ -h(1+x) \\ \frac{1}{2}(1-x^2) \\ -h(1-x) \end{bmatrix} \tag{4.51}$$

The element stiffness matrix is given by,

$$\left[k^e\right] = \int_{A^e} [B]^T [D_b][B]\, dxdy \tag{4.52}$$

Equation (4.52) can be written in terms of natural coordinates as,

$$\left[k^e\right] = \int_{-1}^{+1}\int_{-1}^{+1}[B]^T[D_b][B]\det[J]\cdot dx\,dh \tag{4.53}$$

According to Batoz and Tahar (1982), the double integral in Equation (4.53) can be computed using a standard 2×2 numerical integration scheme. This is found to be sufficient for the solution although theoretically 3×3 numerical integration scheme is required to integrate the quadratic functions. The weight functions and roots for two point Gauss quadrature are given in Table 4.3

Table 4.3 Weight functions and roots for 2×2 Gauss quadrature

Roots	**Weight Functions** w
±0.577350269189626	1.0

Equation (4.53) can be rewritten as,

$$[k]_{12\times12} = \sum_{j=1}^{2}\sum_{i=1}^{2} w_j w_i \left[B\left(x_i,h_j\right)\right]_{12\times3}^{T}[D_b]_{3\times3}\left[B\left(x_i,h_j\right)\right]_{3\times12}\left|J\left(x_i,h_j\right)\right|dx\,dh \tag{4.54}$$

4.7 Implementation of DKQ Element in Java.

To implement the quadrilateral plate bending (DKQ) element in Java, a class called `DKQElement` was developed. The `DKQElement` class is derived from the base class `Element`.

The class definition and the variables declared within the class are given below,

```
public class DKQElement {

  private double th; //Thickness of the element.

  private double detJ; //Determinant of the Jacobian matrix.
  private double[] EX; //array of x Coordinate.

  private double[] EY; //Array of y Coordinate.

  private double[][] Db; //Material Matrix.

  private double[][] X; //x component for geometric function.

  private double[][] Y; //y component for geometric function.

  private double[][] L; //Length of the side of the element.

  private double[] a; //Geometric function a.

  private double[] b; //Geometric function b.

  private double[] c; //Geometric function c.

  private double[] d; //Geometric function d.

  private double[] e; //Geometric function e.

  private double[] NXi; //Derivative of the shape functions w.r.t Xi.

  private double[] NEta; //Derivative of the shape functions w.r.t Eta.

  private double[][] J; //Jacobian matrix.

  private double[] HXxi; //derivative component vector x of shape
                                    functions w.r.t Xi.

  private double[] HXeta; //derivative component vector x of shape
                                    functions w.r.t Eta.

  private double[] HYxi; //derivative component vector y of shape
                                    functions w.r.t Xi.

  private double[] HYeta; //derivative component vector y of shape
                                    functions w.r.t Eta.

  private double[][] B; //Strain-Displacement matrix.

  public double[][] DKQKelem; //Element stiffness matrix for DKQ
                                    Element.
```

The variables `th`, and `detJ` represent the thickness of the element and determinant of the Jacobian respectively. `[]EX` and `[]EY` are arrays containing the x and y coordinates of the nodes. `[] X` and `[] Y` are arrays containing the geometric components representing the sides of the triangle. Array `[] L` contains lengths of the sides of triangle. The arrays `[] a`, `[] b`, `[] c`, `[] d` and `[] e` contain geometric components for calculating the component vectors of the shape functions. `[] NXi` and `[] NEta` are arrays containing the derivatives of the shape functions with respect to x and h respectively. `[] J` is the array containing the Jacobian matrix. `[] HXxi`, `[] HXeta`, `[] HYxi`, and `[] HYeta` are arrays representing derivatives of the component vectors of the shape functions with respect to x and h respectively. The two dimensional arrays `B` and `DKQKelem` contain the strain-displacement matrix and the element stiffness matrix respectively. All of these instance variables except the two dimensional array `DKQKelem` representing the element stiffness matrix, are declared as `private`.

The constructor of the `DKQElement` class is,

```
public DKQElement(double[] cx, double[] cy, double[][] MatDb,
                    double thelem) {
  Db = MatDb;
  th = thelem;
  EX = cx;
  EY = cy;
}
```

The coordinates of the nodes, the material matrix, and the thickness of the element are initialized through the constructor when an instance of the class `DKQElement` is created.

The methods of the class `DKQElement` are given in Table 4.4. All methods are declared private except for the `CalcElemK ()` method, which is declared as public. The method `Geometry()` computes the geometric components required to calculate element stiffness matrix using Equation (4.25). The method `ShFn ()` calculates the derivatives of the shape functions with respect to to x and h using Equation (4.51). The `ShFnHXxi ()` method computes derivatives of the component vectors of the shape functions for x with respect to x. The `ShFnHXeta ()` method computes derivatives of the component vectors of the shape functions for x with respect to h. The `ShFnHYxi ()` method computes derivatives of the component vectors of the shape functions for y with respect to x. The `ShFnHYeta ()` method computes derivatives of the component vectors of the shape functions for y with respect to h. The `DKQBMatrix ()` method computes the B matrix for the DKQ element using Equation (4.48).

Table 4.4 Methods in the `DKQElement` class

Method	Description
`Geometry ()`	Calculates geometric components for the element.
`ShFn ()`	Calculates derivatives of the shape functions with respect to x and h.
`ShFnHXxi ()`	Calculates derivatives of the component vector for the shape functions for x with respect to x.
`ShFnHXeta ()`	Calculates derivatives of the component vector for the shape functions for x with respect to h.
`ShFnHYxi()`	Calculates derivatives of the component vector for the shape functions for y with respect to x.
`ShFnHYeta ()`	Calculates derivatives of the component vector for the shape functions for y with respect to h.
`Jacobian ()`	Calculates the Jacobian matrix and the determinant of Jacobian matrix.
`DKQBMatrix ()`	Calculates the strain-displacement matrix.
`DKQElementKMatrix ()`	Calculates the element stiffness matrix for DKQ element.

The element stiffness matrix is computed in `DKQElementKMatrix ()` method using 2×2 Gauss quadrature. For each Gauss point, the methods `ShFn ()`, `ShFnHXxi ()`, `ShFnHXeta ()`, `ShFnHYxi ()`, `ShFnHYeta ()`, and `DKQBMatrix ()` are called. The elements of the stiffness matrices are computed at each Gauss point using values obtained from these methods are then added to obtain the final element stiffness matrix for the DKQ element.

The structure of the `CalcElemK ()` method is shown below. This method takes `DKQElement` as an argument and calls the `Geometry ()` and `DKQElementKMatrix ()` methods.

```
 public void CalcElemK(DKQElement d) {

    //Calculates geometrical properties for DKQ Element.
    d.Geometry();

    //Calculates Element Stiffness Matrix for DKQ Element.
    d.DKQElementKMatrix(d);

}
```

The approach used for analyzing a structure consisting of DKQ elements is as follows. For each element an instance of the `DKQElement` class is created by calling the constructor.

```
DKQElement dkq = new DKQElement (X, Y, D, TH);
```

where, `dkq` is the instance of the class `DKQElement`. The x and y coordinates, the material matrix and the thickness of element are passed in the above constructor. Once an object of the `DKQElement` class has been created, the element stiffness matrix is computed by calling the method `CalcElemK ()` of the main class by passing the `dkq` object as an argument to the function.

The `DKQElement` class also contains the method `CalcStresses ()`. This method calculates stresses for DKQ element. These stresses are obtained from the nodal displacements which are computed during the analysis. The `CalcStresses ()` method is declared as,

```
public void CalcStresses(DKQElement dkq, double[] U, StressResults S)
```

The parameter `dkq` is an instance of the `DKQElement` class, `U` is the array of nodal displacements and `S` is an instance of the `StressResults` class which stores element stresses.

The stresses at each node of the element are calculated using 2×2 Gauss quadrature. The methods `ShFn ()`, `ShFnHXxi ()`, `ShFnHXeta ()`, `ShFnHYxi ()`, `ShFnHYeta ()`, and `DKTBMatrix ()` are called at each Gauss point and the stresses are calculated using the stress-strain and strain-displacement relationships.

Chapter 5
Flat Shell Elements

5.1 General Shell Elements

Shell elements are very efficient for modeling the behavior of curved structures (Cook *et al.*, 1989). There are four types of general shell elements: flat shell elements, curved shell elements, axisymmetric shell elements and, Mindlin type degenerated solid elements (Yang et. al, 1990). Shell elements can also be classified according to the thickness of the shell and the curvature of the midsurface. Depending on the thickness, shell elements can be separated into thin shell elements and thick shell elements. Thin shell elements are based on the discrete Kirchoff theory in which transverse shear deformations are neglected. Thick shell elements are based on the Mindlin theory which includes transverse shear deformations.

Shell elements can also be classified according to curvature as deep shell elements and shallow shell elements. Shallow elements based on the classical shell theory and can be developed by combining the membrane and bending strain in the energy equation. Flat shell elements are developed by superimposing the stiffness of membrane and bending elements. The membrane and bending forces are totally independent of each other in the flat shell element and hence there is no membrane-bending coupling present in the element. This is a major disadvantage of the flat shell elements.

The development of the shell elements from the classical shell theory is more complex, and many approximations are required to simplify the solution. Flat shell elements are easier to formulate using previously available theories of membrane and plate bending elements.

5.2 Flat Shell Elements

The shell element is subjected to both membrane forces and bending forces and hence the development of shell elements should include a consideration of both these actions. One approach for development of flat shell elements is to include the membrane and bending properties by combining a membrane element and a plate bending element.

Two types of thin flat shell elements were considered to implement in Java: (1) triangular flat shell element and, (2) quadrilateral flat shell element. The triangular flat shell element developed by combining the CST element described in Chapter 3, which represents the membrane part of the element, and the DKT element (Batoz *et al.*, 1980) described in Chapter 4, which represents the bending part of the element. Consider the element is in the xy plane, assembly of the triangular flat shell element can be represented as shown in Fig. 5.1.

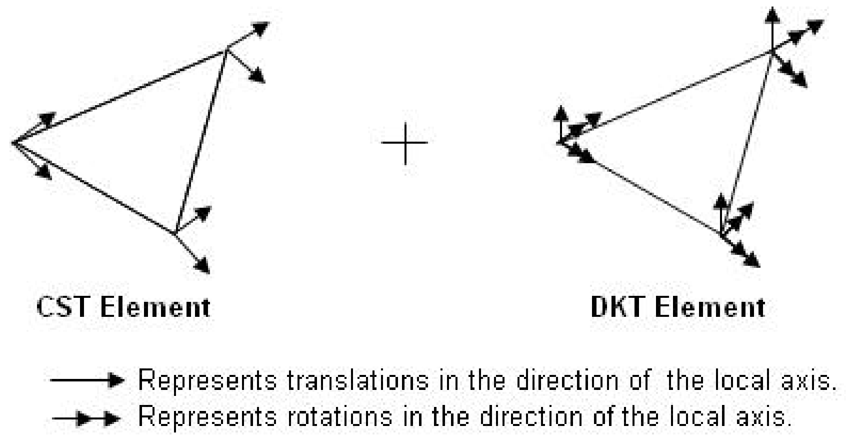

CST Element
+
DKT Element
Represents translations in the direction of the local axis.
Represents rotations in the direction of the local axis.

Fig. 5.1 Combination of CST and DKT Element.

The quadrilateral flat shell element is developed by the assembly of the four node quadrilateral plane stress element presented in Chapter 3 and the DKQ element (Batoz and Tahar, 1982) presented in Chapter 4. Considering that the element is in the xy plane, the assembly of the quadrilateral flat shell element can be represented as shown in Fig. 5.2.

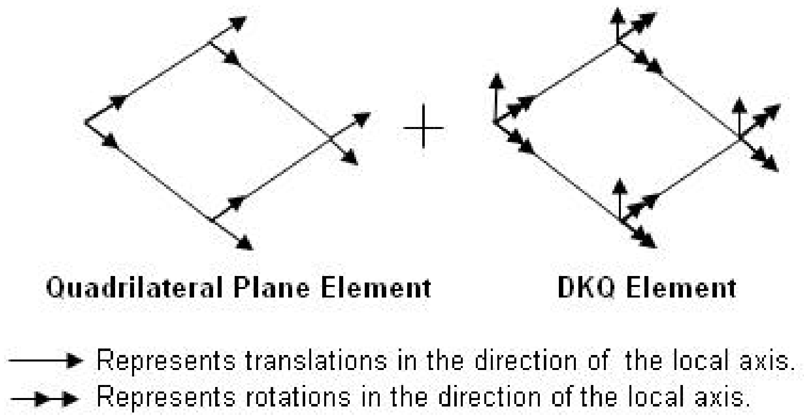

Quadrilateral Plane Element
DKQ Element
Represents translations in the direction of the local axis.
Represents rotations in the direction of the local axis.

Fig. 5.2 Combination of Quadrilateral Plane Element and DKQ Element.

5.3 Development of Stiffness Matrix for Flat Shell Elements.

In this section we will discuss the general formulation of stiffness matrix for flat shell elements. The development of the stiffness matrix for the triangular membrane element and quadrilateral membrane element was presented in Chapter 3. The development of the stiffness matrix for the triangular plate bending element and quadrilateral plate bending element was presented in Chapter 4.

The general shell has six degrees of freedom at each node. The nodal displacements of the shell element are,

$$\{U_i\} = \{u_i \quad v_i \quad w_i \quad q_{xi} \quad q_{yi} \quad q_{zi}\} \tag{5.1}$$

for, $i = 1\text{K}\ n$ $\qquad$ n = number of nodes per element.

The membrane stiffness matrix for each node is of size 2×2 , and is represented as, $[k_m]_{2\times2}$. The bending stiffness matrix for each node is of size 3×3 and is represented as, $[k_b]_{3\times3}$. The stiffness matrix at each node of the shell element is of size 6×6 and is represented as, $[k_s]_{6\times6}$. The assembly of the stiffness matrices of membrane and bending components at each node will result in a zero value on the diagonal corresponding to the rotational degree of freedom q_z since this displacement is not considered in the membrane or bending element. Fig. 5.3 represents the inplane rotation which is sometimes called drilling degrees of freedom.

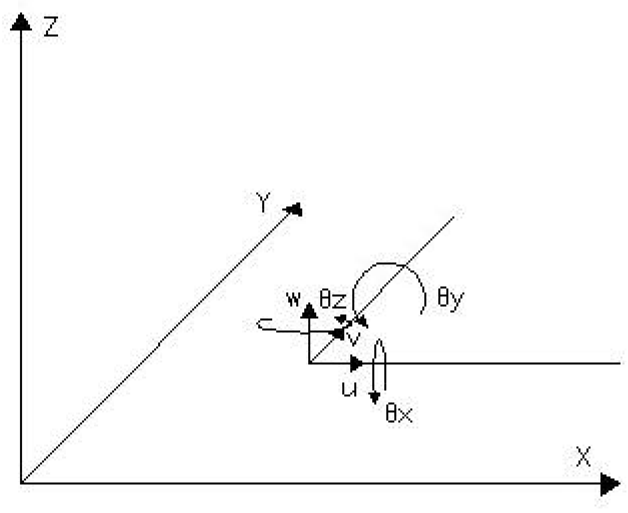


Fig. 5.3 Drilling Degrees of Freedom (Local coordinate system).

This zero stiffness for the drilling degree of freedom causes singularity in structure stiffness matrix when all the elements are coplanar and there is no coupling between the membrane and bending stiffness of the element. There are several ways to deal with this singularity.

The first approach for removing the singularity in the structure stiffness matrix is to substitute an approximate value for the diagonal value of the stiffness of drilling degree of freedom. Although, this solves the problem of singularity from the structure stiffness matrix, it sometimes does not represent actual behavior of the element because of the fact that a fictitious stiffness has been added.

The second approach is to develop a higher order membrane element that includes the drilling degree of freedom. This approach is less efficient since higher order displacement functions are needed for the membrane stiffness matrix and hence a higher order numerical integration scheme is required.

In this study the first approach is used since it is easier to implement and is more efficient. This approach of including the fictitious stiffness for the drilling degree of freedom closely approximates the behavior of the shell but sometimes it results in a stiffer structure due to the constraints present at the corner nodes.

The element stiffness matrix for the shell element is first assembled by super imposing the membrane stiffness and bending stiffness at each node. The null values of the stiffness corresponding to the drilling degree of freedom are then replaced by approximate values. This approximate value is taken to be equal to 10^{-3} times the maximum diagonal value in the element stiffness matrix. The stiffness matrix at each node of the shell element $[k_s]_i$ can thus be represented as,

$$[k_s]_i = \begin{bmatrix} [k_m]_{2\times 2} & [0]_{2\times 3} & 0 \\ [0]_{3\times 2} & [k_b]_{3\times 3} & 0 \\ 0 & 0 & \dfrac{\max\left((K_s)_{i,i}\right)}{1000} \end{bmatrix} \tag{5.2}$$

where,

$[k_s]_i$ = stiffness at each node of the shell element.

$[k_m]_{2\times 2}$ = membrane stiffness at each node of the shell element.

$[k_b]_{3\times 3}$ = bending stiffness at each node of the shell element.

K_s = element stiffness matrix for the shell element.

5.4 Coordinate Transformation

A shell is a three dimensional structure and it is often convenient to define the geometry of shell structure in the global coordinate system. However, to generate the element stiffness matrix for the membrane and plate bending elements, the elements have to be defined in the element local plane, and element local coordinates are required to calculate the stiffness of these elements. Since the flat shell elements considered in this study are based on a combination of membrane and plate bending elements it is thus necessary to use local coordinates for computing the element stiffness matrix of the flat shell elements. The transformation between global coordinates and local coordinates is required to generate the element local stiffness matrix in the local coordinate system. Also the stiffness matrix must then be transformed to the global coordinate system. This can be done using vector algebra. Direction cosines are required to transform the coordinates from the global coordinate system to the local coordinate system. The formulation of the transformation matrix for triangular and quadrilateral element is described below (SAP 2000 Analysis Reference, Computers and Structures Inc., 1997).

Triangular Element

Fig. 5.4 shows the global axis X, Y and Z and the transformed local axis x, y and z. In this study the global Z axis is in the upward direction and local z axis is normal to the plane of the element. In Fig. 5.4, the node numbers are written in the counterclockwise direction and i, j, and k represent the mid points of the sides 1-2, 2-3 and 3-1 respectively; and are used to define direction cosines for the plane of the triangle

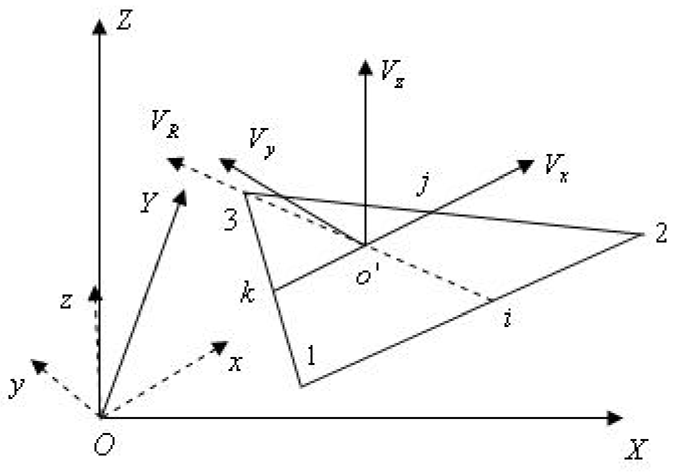


Fig. 5.4 Coordinate Transformation for Triangular Element.

Assuming the local x axis is parallel to the vector passing through nodes k and node j, the vector representing the local x direction is given by,

$$V_x = V_{jk} = \begin{Bmatrix} x_j - x_k \\ y_j - y_k \\ z_j - z_k \end{Bmatrix} = \begin{Bmatrix} x_{jk} \\ y_{jk} \\ z_{jk} \end{Bmatrix} \tag{5.3}$$

where x_k, y_k, z_k etc. represents the values of global coordinates for node k. The direction cosine l_x for the local x direction is obtained by normalizing the vector with respect to its length of the side. The direction cosine for the local x axis is given by,

$$l_x = \frac{1}{l_{kj}} \begin{Bmatrix} x_{jk} \\ y_{jk} \\ z_{jk} \end{Bmatrix} \tag{5.4}$$

where,

$l_{kj} = \sqrt{\left(x_{jk} \right)^2 + \left(y_{jk} \right)^2 + \left(z_{jk} \right)^2}$ is the length of the vector.

A reference vector defining the plane of the element is obtained by creating a vector that passes through the vector V_x which defines the local x direction. The reference vector V_R, is obtained by creating a vector that passes from nodes i and 3 by the same procedure described in Equation (5.3).

$$V_R = V_{i3} \tag{5.5}$$

The normal to the plane that represents the element local z direction is obtained by the cross product of vectors V_x and V_R,

$$V_z = V_x \times V_R \tag{5.6}$$

The direction cosine l_z for the local z direction is obtained by normalizing the vector V_z as shown in Equation (5.4). The local y axis is obtained from cross product of the vector in the local x direction and the vector in the local z direction. The cross product of these two vectors gives the vector V_y normal to the xz plane.

$$V_y = V_z \times V_x \tag{5.7}$$

The direction cosine l_y for local y direction is obtained by normalizing the vector V_y as given in Equation (5.4).

Quadrilateral Element

Fig. 5.5 shows the global and local coordinate axis for the quadrilateral element. The node numbers for the quadrilateral elements are written in counterclockwise order. The mid points of sides 1-2, 2-3, 3-4, and 4-1 are represented by i, j, k and l respectively. The element local plane is defined by creating two vectors intersecting each other and passing through the mid points of the sides 2-3 and 3-4 of the quadrilateral as shown in Fig. 5.5.

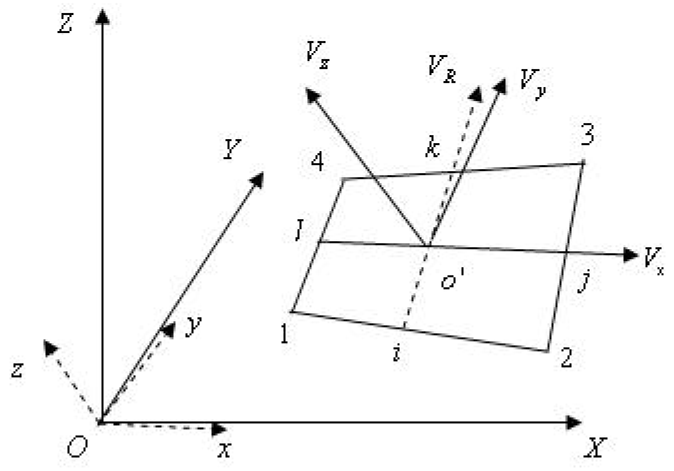


Fig. 5.5 Coordinate Transformation for Quadrilateral Element.

Assuming the local x axis of the quadrilateral is parallel to the vector passing through nodes l and j, the vector passing through these nodes is given by,

$$V_x = V_{lj} = \begin{Bmatrix} x_j - x_l \\ y_j - y_l \\ z_j - z_l \end{Bmatrix} = \begin{Bmatrix} x_{jl} \\ y_{jl} \\ z_{jl} \end{Bmatrix} \tag{5.8}$$

where, x_i, y_i, z_i etc. represents the global coordinates of node i. The direction cosine l_x for the local x direction is obtained by normalizing the vector with respect to its length.

$$l_x = \frac{1}{l_{jl}} \begin{Bmatrix} x_{jl} \\ y_{jl} \\ z_{jl} \end{Bmatrix} \tag{5.9}$$

where,

$l_{jl} = \sqrt{\left(x_{jl}\right)^2 + \left(y_{jl}\right)^2 + \left(z_{jl}\right)^2}$ is the length of the vector.

A reference vector V_R, defining the plane of the element is obtained by creating a vector passing through nodes i and k of the element as shown in Fig. 5.5.

$$V_R = V_{ik} \tag{5.10}$$

The normal to the plane is obtained by the vector cross product of V_x and V_R,

$$V_z = V_x \times V_R \tag{5.11}$$

The direction cosine for the local z direction l_z is obtained by normalizing vector V_z with respect to its length, as given in Equation (5.4). The local y axis is obtained by the vector cross product of the vector in the local x direction and vector in local z direction. The cross product of these two vectors will give the vector V_y normal to the xz plane.

$$V_y = V_z \times V_x \tag{5.12}$$

The direction cosine l_y for the local y direction is obtained by normalizing the vector V_y with respect to its length, as given in Equation (5.4).

Transformation of Coordinates and Stiffness Matrix

The 3×3 transformation matrix for the transformation of coordinates from the global to the local axis can be written as,

$$[l]_{3\times3} = \left\{ (l_x)_{3\times1} \quad (l_y)_{3\times1} \quad (l_z)_{3\times1} \right\} \tag{5.13}$$

where l_x, l_y, and l_z are the direction cosines for local x, y and z directions.

The local coordinates at each node needed to derive the element local stiffness matrix is obtained as,

$$[xyz]_{3\times1} = [l]_{3\times3}^{T}[XYZ]_{3\times1} \tag{5.14}$$

where,

$[xyz]_{3\times1}$ = local x, y and z coordinates at each node.

$[XYZ]_{3\times1}$ = global X, Y and Z coordinates at each node.

The element local stiffness matrix for membrane and plate bending element is calculated using the local coordinates obtained. The stiffness for drilling degrees of freedom is approximated as described in the previous section. The element stiffness matrix $[k_s]_{6n\times6n}$ for the shell element, where n is the number of nodes per element, is then defined in the local coordinate system.

To calculate the structure stiffness matrix, the element stiffness matrix must be transformed to the global coordinate system. The transformation of the element stiffness matrix from the local to the global coordinate system is given by,

$$[K_s]_{6n\times6n} = [T]_{6n\times6n}^{T}[k_s]_{6n\times6n}[T]_{6n\times6n} \tag{5.15}$$

where,

n = number of nodes per element.

K_s = element stiffness matrix in the global coordinate system.

k_s = element stiffness matrix in the local coordinate system.

$[T]$ = transformation matrix.

The transformation matrix for triangular shell elements is,

$$[T] = \begin{bmatrix} [l]^T & 0 & 0 \\ 0 & [l]^T & 0 \\ 0 & 0 & [l]^T \end{bmatrix} \tag{5.16}$$

The transformation matrix for quadrilateral shell element is,

$$[T] = \begin{bmatrix} [l]^T & 0 & 0 & 0 \\ 0 & [l]^T & 0 & 0 \\ 0 & 0 & [l]^T & 0 \\ 0 & 0 & 0 & [l]^T \end{bmatrix} \tag{5.17}$$

where, $[l]$ is calculated from Equation (5.13) for triangular or quadrilateral element respectively.

5.5 Implementation of Triangular Flat Shell Element in Java

To implement the triangular flat shell element in Java, a class called `TriShellElement` was developed. The `TriShellElement` class is derived from the base class `Element`. The base class `Element` contains all the data variables and functions necessary for managing all of the different elements in the program.

The class definition and the variables declared within the `TriShellElement` class are given below,

```
public class TriShellElement {

  private double th; //Thickness of the element.
  private double[] EX; //array of x coordinates.
  private double[] EY; //array of y coordinates.
  private double[] EZ; //array of z coordinates.
  private double[] X; //array of local x coordinates.
  private double[] Y; //array of local y coordinates.
  private double[][] MatD; //Material matrix for plane stress.
  private double[][] MatDb; //Material matrix for bending condition.
  private double[][] Kcst; //Element stiffness matrix for CST Element.
  private double[][] Kdkt; //Element stiffness matrix for DKT Element.
  private double[][] Ttrans; //Transformation matrix.
  private double[][] ST; //Transformation matrix for coordinates.
  private double[][] TSKelem; //Element local stiffness matrix.
  public double[][] GTSKelem; //Element Global stiffness matrix.
```

The variable `th` represents the thickness of the element. `[]EX`, `[]EY`, and `[] EZ` are arrays containing the x, y, and z coordinates of the nodes in the global coordinate system. The arrays `[] X` and `[] Y` containing the x and y coordinates of the element local coordinate system. The two dimensional arrays `[][]MatD` and `[][]MatDb` contain the material matrix for membrane and bending properties. The two dimensional arrays `[][]ST` and `[][]Ttrans` contain the transformation matrix for coordinates and element stiffness matrix respectively. The two dimensional arrays `[][]Kcst` and `[][]Kdkt` contain the stiffness matrices for the CST element and the DKT element respectively. `[][]TSKelem` and `[][]GTSKelem` are two dimensional arrays that contain the element stiffness matrix for triangular flat shell element in the local and global coordinate system. All of these instance variables except the two dimensional array `GTSKelem` representing the element stiffness matrix of triangular flat shell element in the global coordinate system, are declared as private.

The constructor of the `TriShellElement` class is declared as follows,

```
public   TriShellElement(double[]   cx,   double[]   cy,   double[]   cz,
                        double[][] D, double[][] Db, double thelem) {
    MatD = D;
    MatDb = Db;
    th = thelem;
    EX = cx;
    EY = cy;
    EZ = cz;
```

The coordinates of the nodes, the material matrices for membrane and bending properties, and the thickness of the element are all initialized through the constructor when an instance of the class `TriShellElement` is created.

The methods of the class `TriShellElement` are given in Table 5.1. All methods are declared private except for the `CalcElemK()` method, which is declared as public since it is called from outside the class. The method `CalcLocalcoord()` computes the element local coordinates by performing a coordinate transformation. The `CaclcTransformationMatrix()` method computes the transformation matrix needed to transform the element stiffness matrix from the local to the global coordinate system. The `CalcElemLocalStiffMatrix()` method computes the element stiffness matrix in the local coordinate system by superimposing the stiffness matrices of the CST element and the DKT element. The `CalcElemGlobalStiffMatrix()` method computes the element stiffness matrix in the global coordinate system.

Table 5.1 Methods in the `TriShellElement` class

Method	Description
`CalcLocalcoord()`	Calculates element local coordinates.
`CalcTransformationMatrix()`	Calculates transformation matrix.
`CalcElemLocalStiffMatrix()`	Calculates element stiffness matrix of triangular flat shell element in the local coordinate system.
`CalcElemGlobalStiffMatrix()`	Calculates element stiffness matrix of triangular flat shell element in the global coordinate system.

The structure of the `CalcElemK()` method is shown below. This method takes an object of type `TriShellElement` as an argument and calls the methods `CalcLocalcoord()`, `CalcTransformation()`, `CalcElemLocalStiffMatrix()`, and `CalcElemGlobalStiffMatrix()`.

```
public void CalcElemK(TriShellElement tri) {

  //Calculates element coordinates in local plane.
  tri.CalcLocalcoord();

  //Calculates transformation matrix.
  tri.CalcTransformationMatrix();

  //Calculates element stiffness matrix in the local coordinate system.
  tri.CalcElemLocalStiffMatrix();

  //Calculates   element   stiffness   matrix   in   the   global   coordinate
     system.
  tri.CalcElemGlobalStiffMatrix();
}
```

The approach used for analyzing a structure consisting of triangular flat shell elements is as follows. For each element an instance of the `TriShellElement` class is created by calling the constructor.

```
TriShellElement tshell = new TriShellElement(x, y, z, MatD, MatDb, TH);
```

where, `tshell` is the instance of the class `TriShellElement`. The x, y and z coordinates, material matrix for membrane and bending properties and the thickness of the element are passed in the above constructor. Once an object of the `TriShellElement` class has been created, the element stiffness matrix is computed by calling the method `CalcElemK ()` of the main class by passing the `tshell` object as an argument to the function.

The `TriShellElement` class also contains the method `CalcStresses ()`. This method calculates stresses for triangular shell element. These stresses are obtained from the nodal displacements which are computed during the analysis. The `CalcStresses ()` method is declared as,

```
public void CalcStresses(TriShellElement tshell, int elem, double[] U,
                          StressResults SR)
```

The parameter `tshell` is an instance of the `TriShellElement` class, `elem` is the element number, `U` is the array of nodal displacements, and `SR` is an instance of the `StressResults` class which stores element stresses.

The stresses at each node of the element are calculated by first calculating membrane and bending stresses for membrane element and plate bending element respectively. The summation of membrane and bending stresses gives the stresses for triangular flat shell element.

5.6 Implementation of Quadrilateral Flat Shell Element in Java

To implement the quadrilateral flat shell element in Java, a class called `QuadShellElement` was developed. The `QuadShellElement` class is derived from the base class `Element`.

The class definition and the variables declared within the `QuadShellElement` class are given below,

```
public class QuadShellElement {

  private double th; //Thickness of the element.
  private double[] EX; //Array of x Coordinates.
  private double[] EY; //Array of y Coordinates.
  private double[] EZ; //Array of z Coordinates.
  private double[] X; //Array of local x Coordinates.
  private double[] Y; //Array of local y Coordinates.
  private double[][] MatD; //Material matrix for plane stress.
  private double[][] MatDb; //Material matrix for bending condition.
  private double[][] Kquad4; //Element stiffness matrix for QUAD4
                                          Element.
  private double[][] Kdkq; //Element stiffness matrix for DKQ Element.
  private double[][] Ttrans; //Transformation matrix.
  private double[][] ST; //transformation matrix for coordinates.
  private double[][] QSKelem; //Element local stiffness matrix.
  public double[][] GQSKelem; //Element Global stiffness matrix.
```

The variable `th` represents the thickness of the element. `[]EX`, `[]EY` and `[]EZ` are arrays containing the x, y, and z coordinates of the nodes in the global coordinate system. The arrays `[]X` and `[]Y` contain the x and y coordinates of the element local coordinate system. The two dimensional arrays `[][]MatD`, and `[][]MatDb` contain the material matrix for membrane and bending properties. The two dimensional arrays `[][]ST` and `[][]Ttrans` contain the transformation matrix for coordinates and element stiffness matrix respectively. The two dimensional arrays `[][]Kquad4`, and `[][]Kdkq` contain the stiffness matrices for the four node quadrilateral plane element and the DKQ element respectively. `[][] QSKelem`, and `[][]GQSKelem` are two dimensional arrays that contain the element stiffness matrix for quadrilateral flat shell element in the local and the global coordinate system. All of these instance variables except the two dimensional array `GQSKelem` representing the element stiffness matrix, are declared as private.

The constructor of the `QuadShellElement` class is declared as follows,

```
public   QuadShellElement(double[]   cx,   double[]   cy,   double[]   cz,
                        double[][] D, double[][] Db, double thelem) {
    MatD = D;
    MatDb = Db;
    th = thelem;
    EX = cx;
    EY = cy;
    EZ = cz;
  }
```

The coordinates of the nodes, the material matrices for membrane and bending properties, and the thickness of the element are all initialized through the constructor when an instance of the class `QuadShellElement` is created.

The methods of the class `QuadShellElement` are given in Table 5.2. All methods are declared private except for the `CalcElemK()` method, which is declared as public since it is called from outside the class. The method `CalcLocalcoord()` computes the element local coordinates by performing a coordinate transformation. The `CaclcTransformationMatrix()` method computes the transformation matrix needed to transform the element stiffness matrix from the local to the global coordinate system. The `CalcElemLocalStiffMatrix()` method computes the element stiffness matrix in the local coordinate system by superimposing the stiffness matrices of the four node quadrilateral plane element and the DKQ element. The `CalcElemGlobalStiffMatrix()` method computes the element stiffness matrix in the global coordinate system.

Table 5.2 Methods in the `QuadShellElement` class

Method	Description
`CalcLocalcoord()`	Calculates element local coordinates.
`CalcTransformationMatrix()`	Calculates transformation matrix.
`CalcElemLocalStiffMatrix()`	Calculates element stiffness matrix of quadrilateral flat shell element in the local coordinate system.
`CalcElemGlobalStiffMatrix()`	Calculates element stiffness matrix of quadrilateral flat shell element in the global coordinate system.

The structure of the `CalcElemK ()` method is shown below. This method takes an object of type `QuadShellElement` as an argument and calls the methods `CalcLocalcoord()`, `CalcTransformation()`, `CalcElemLocalStiffMatrix()`, and `CalcElemGlobalStiffMatrix`.

```
public void CalcElemK(QuadShellElement quad) {

  //Calcualtes element coordinates in local plane.
  quad.CalcLocalcoord();

  //Calculates transformation matrix.
  quad.CalcTransformationMatrix();

  //Calculates element stiffness matrix in the local coordinate system.
  quad.CalcElemLocalStiffMatrix();

  //Calculates element stiffness matrix in the global coordinate
    system.
  quad.CalcElemGlobalStiffMatrix();
}
```

The approach used for analyzing a structure consisting of quadrilateral flat shell elements is as follows. For each element an instance of the `QuadShellElement` class is created by calling the constructor.

```
QuadShellElement qshell = new QuadShellElement(x, y, z, MatD, MatDb,
                                                   TH);
```

where, `qshell` is the instance of the class `QuadShellElement`. The x, y, and z coordinates, material matrix for membrane and bending properties and the thickness of the element are passed in the above constructor. Once an object of the `QuadShellElement` class has been created, the element stiffness matrix is computed by calling the method `CalcElemK ()` of the main class by passing the `qshell` object as an argument to the function.

The `QuadShellElement` class also contains the method `CalcStresses ()`. This method calculates stresses for quadrilateral shell element. These stresses are obtained from the nodal displacements which are computed during the analysis. The `CalcStresses ()` method is declared as,

```
public void CalcStresses(QuadShellElement qshell, int elem, double[] U,
                         StressResults SR)
```

The parameter `qshell` is an instance of the `QuadShellElement` class, `elem` is the element number, `U` is the array of nodal displacements and `SR` is an instance of the `StressResults` class which stores element stresses.

The stresses at each node of the element are calculated by first calculating the membrane and bending stresses for membrane element and plate bending element respectively. The summation of membrane and bending stresses gives the stresses for quadrilateral flat shell element.

Chapter 6
Program Development

6.1 Introduction

The main objective of this study is to develop triangular and quadrilateral plane stress, plate bending, and flat shell elements in Java. A second objective was to write a finite element analysis program to verify the results obtained from these elements. The program computes displacements and stresses for the structure modeled using the plane stress, plate bending, or flat shell elements. This chapter presents the development of the finite element analysis program in Java. The required input to the program is in the form of a text file. The results from the program are saved in an output file in text format. The format of the input file essentially follows that of the SAP 2000 commercial finite element analysis program with some minor modifications. The SAP 2000 program was used to generate the finite element models for testing and verification.

6.2 Program Structure

A finite element analysis program for the analysis of membrane, plate and shell structures was developed in Java. The program was developed using the object oriented approach. The classes are divided in to three categories: (1) structural classes (2) input-output classes and (3) interface classes. The description of various classes for each category in the program is presented in the following paragraphs.

6.2.1 Structural Classes

The structural classes include all the classes that represent the structural model and the classes that are used to analyze the structure.

For any finite element analysis problem, the finite element model is first created from different structural components such as system properties (representing the behavior of whole structure), nodal coordinates, restraints, material properties, element section properties, element joint connectivity, and loads. The classes that represent the objects of these real structural components are `SystemProp`, `Joint`, `Restraint`, `Material`, `SectionProp`, `Element`, `JtLoad`, and `UniLoad` respectively. The `FEModel` class is the storage class where the objects representing the finite element model are stored.

After all the required structural data is obtained, an instance of the `Analysis` class is created and displacements and stresses are calculated. The instances of the `Coordtransformation` and `EqNodalLoads` classes are used to define the transformation matrices and calculate equivalent nodal loads respectively.

The classes `MembraneStr`, `BendinStr`, and `ShellStr` represent membrane, bending and shell stresses respectively for each element of the structure. The `StressResults` class is a container class where objects containing stresses for each element are stored. The description of each structural class is given in the following paragraphs.

`SystemProp` Class

The system properties represent the behavior of the entire structure. The system properties such as the number of system degrees of freedom in each direction are defined in the `SystemProp` class. The number of system degrees of freedom defines the behavior of structure, i.e. whether it is a plane, plate or shell structure. The class `SystemProp` contains the methods `GetSysdof()` and `GetiflagSysdof()`. These methods return the values of system degrees of freedom and an array of values for system degrees of freedom.

`Joint` Class

The nodal coordinates for each node of an element is required to calculate the element stiffness matrix. The `Joint` class is the base class of all nodes in the structure. Each node has its own node number and values of x, y and z coordinates. The node number and x, y, z coordinates for a node are defined in the `Joint` class. Several methods such as `GetJtNodeNum()`, `GetX()`, `GetY()`, and `GetZ()` are defined within the `Joint` class. These methods provide access to the various data members of the.

`Restraint` Class

In order to analyze the structure it is important to know how many equations are to be solved. The number of equations required to be solved are obtained from the number of restraints provided to the structure and are derived from the class `Restraint`. The node number and an array of equation numbers for that node are defined in this class. The methods `GetRestrNodeNum()` and `GetEQNID()` are also defined in the `Restraint` class. These methods return the values of the node number and an array of equation IDs for that node when called.

`Material` Class

The `Material` class stores information regarding linear elastic material properties. The information contained in this class includes the modulus of elasticity and Poisson's ratio of the material. The methods defined in this class are `GetMoE()` and `GetNU()`. These methods are called to access the stored material properties.

`SectionProp` Class

The `SectionProp` class represents the element section properties such as thickness of the elements and the plane strain or plane stress condition. The `SectionProp` class also contains the methods `GetThickness()` and `GetFlagMat()`. These methods return the values of thickness of elements and values indicating the stress condition (i.e. plane stress or plane strain).

`Element` Class

The `Element` class is the base class for all elements. The data members in this class include element number, number of nodes for the element and a joint connectivity array. The methods `Getelemnum()`, `GetNumnodes()`, and `GetConnectivity()` are also defined in the `Element` class. These methods return the element number, number of nodes for that element and the joint connectivity array. Once the node numbers for each node of the element are obtained, the coordinates for that node can be obtained using the methods defined in the `Joint` class. The class hierarchy for the `Element` class is represented in Fig. 6.1.

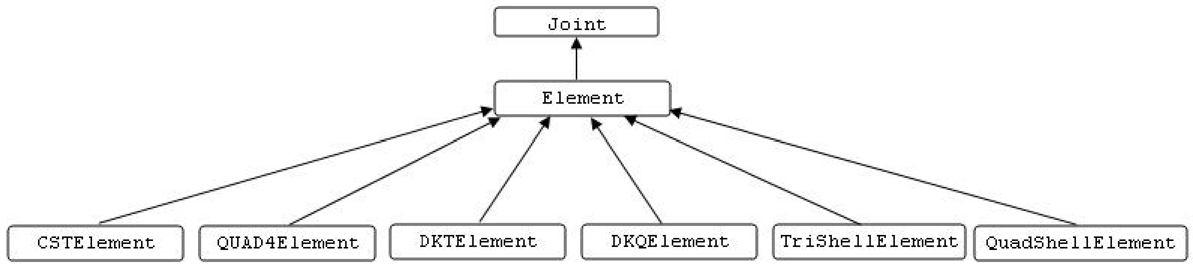


Fig. 6.1 `Element` Class Diagram.

`JointLoad` Class

The `JointLoad` class represents a force or moment applied at a joint. The data members of this class include the node number and an array of nodal loads applied at that node. The methods defined in the `JointLoad` class are `GetNodeNumber()` and `GetLoadValue()`. These methods return the node number and an array of nodal loads in each direction for that node.

`UniLoad` Class

The `UniLoad` class represents a uniformly distributed surface load applied in the direction normal to the element local plane. The `UniLoad` class contains the element number and the magnitude of the uniformly distributed surface load. The methods in the `UniLoad` class include `GetNodeNumber()` and `GetLoadValue()`. The element number and the magnitude of uniformly distributed load applied on the element can be accessed by calling these methods.

`FEModel` Class

The `FEModel` class is a container class where all of the components of a structural model are stored as objects. The Java API class `Vector` is used to store objects of each structural component such as system properties, nodes, restraints, material properties, section properties, elements, and loads.

Java provides the library of classes that dynamically allocate space for each object to be stored. These classes are modified from the collection classes in the Java class library `java.util`. The legacy classes have the additional advantage over that they are synchronized. In this program, the API class `Vector` of Java class library `java.util` is used to store the data dynamically. `Vector` is a dynamic array, which is similar to the `ArrayList` class of the Java collection classes. The `Vector` class in Java has an initial default capacity of 10. The `Vector` dynamically allocates the space for the object to be stored after the initial capacity is reached. The elements stored in the `Vector` can be easily manipulated using the methods provided in Java. Once an instance of a `Vector` is created, the element can be added to a `Vector` by calling the `addElement()` method. The value of an element from any specific location of the `Vector` can be obtained by calling the method `elementAt()`. Elements stored in the `Vector` can be removed by calling the `removeElement()` or `removeElementAt()` methods.

Several instances of the `Vector` class are declared in the `FEModel` class. These are `SystemPropList`, `NodeList`, `RestraintList`, `MaterialList`, `SectionPropList`, `ElementList`, `JtLoadList` and `UniLoadList.` They store objects of the classes `SystemProp`, `Joint`, `Material`, `SectionProp`, `Element`, `JointLoad`, and `UniLoad` that represents system properties, joint data, material properties, element section properties, element data, joint loads, and uniformly distributed surface loads respectively. Thus, each component of the finite element model is stored as an instance of the `Vector` class. The diagram representing the relationship between `FEModel` class and other structural classes is shown in Fig. 6.2.

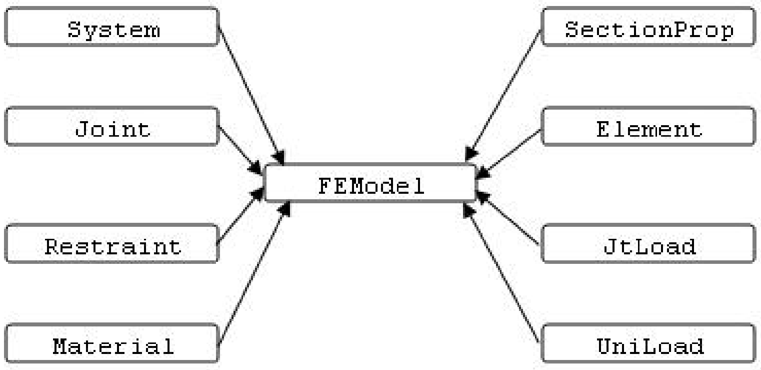


Fig. 6.2 `FEModel` Class Diagram.

The methods for storing and retrieving the objects of each structural component are defined in the `FEModel` class. Each method creates an object of different structural component and stores it to the `vector` using the `AddElement()` method from `java.util` class library. All methods are declared public. Each method of the `FEModel` class and its function is described in Table 6.1.

Table 6.1 Methods in the `FEModel` class

Method	Description
`AddSystem ()`	Creates an instance of the `System` class and adds it to the `vector SystemPropList`.
`AddNode ()`	Creates an instance of `Joint` class and adds it to the `vector NodeList`.
`AddRestraint ()`	Creates an instance of the `Restraint` class and adds it to the `vector RestraintList`.
`AddMaterial ()`	Creates an instance of the `Material` class and adds it to the `vector MaterialList`.
`AddSectionProp ()`	Creates an instance of the `SectionProp` class and adds it to the `vector MaterialList`.
`AddElement ()`	Creates an object of the `Element` class and adds it to the `vector ElementList`.
`AddJointLoad ()`	Creates an instance of the `JointLoad` class and adds it to the `vector JtLoadList`.
`AddUniLoad ()`	Creates an instance of the `UniLoad` class and adds it to the `vector UniLoadList`.
`PutNumofRestr ()`	Stores the number of restraints provided for entire structure.
`PutUDLflag ()`	Stores a flag to specify if uniformly distributed loads are applied to the structure.

`CoordTransformation` Class

The flat shell element developed in this study is a combination of a plane stress element and a plate-bending element. A shell element is three dimensional in nature and it is convenient to represent a shell element in the global coordinate system. However, plane stress element and plate bending element are two dimensional elements and hence a transformation is required to obtain the element stiffness matrix for the she ll element when combining these two elements.

The `CoordTransformation` class represents the coordinate transformation from the global to the local coordinate system. A transformation matrix can also be obtained that transforms the element stiffness matrix from the local coordinate system to the global coordinate system.

The `CoordTransformation` class contains various methods such as `CalcCoord()`, `DCOS()` and `CROSS()`. All methods are declared private except `CalcCoord()` method. The method `CalcCoord()` is called in the element classes to calculate the element stiffness matrix for shell elements. The method `DCOS()` returns an array representing the direction cosines for a `vector`. The method `CROSS()` computes the cross product of two `vectors` and returns an array containing the resulting `vector`. The `CalcCoord()` method performs coordinate transformation and calculates the transformation matrix using the procedure described in Chapter 5.

`EqNodalLoads` Class

When uniformly distributed surface loads are applied to the elements in the direction normal to their local plane, these loads have to be transformed to equivalent nodal loads applied at each node of the element.

The `EqNodalLoads` class contains methods for computing the equivalent nodal loads from the uniformly distributed surface load. The method `CalcEqNodalLoads()` is called once an instance of the `EqNodalLoads` is created. This method calculates equivalent nodal loads at the each node of the structure and creates an array of loads. The 2×2 Gauss quadrature is used to compute these nodal loads at the Gauss points. The nodal loads computed at the Gauss points are then extrapolated to the element nodes. The methods defined within the `EqNodalLoads` class are given in Table 6.2. All methods except `CalcEqNodalLoads()` are declared private.

Table 6.2 Methods in the `EqNodalLoad` class

Method	Description
`QUADShapeFn ()`	Calculates shape functions for the quadrilateral element.
`QuadJacobian ()`	Calculates the determinant of Jacobian for the quadrilateral element.
`CalcQuad4EqNodalLds ()`	Calculates equivalent nodal loads at each node of the quadrilateral element.
`CalcTriEqNodalLds ()`	Calculates equivalent nodal loads at each node of the triangular element.
`CalcEqNodalLoads ()`	Calculates equivalent nodal loads at each node.

`Analysis` Class

The analysis of the structure is implemented in the `Analysis` class. Once the finite element model is generated, all of the components of the model are stored as objects in the `FEModel` class. The analysis is then performed using these stored objects. Several methods are developed to perform the different steps in the analysis such as, compute the structure stiffness matrix, compute displacements at each node of the structure, and compute element stresses for each element. All methods are declared as private except the `RunAnalysis()` method. The methods defined in the `Analysis` class are given in Table 6.3.

Table 6.3 Methods in the `Analysis` class.

Method	**Description**
`CalcRestrDOF ()`	Calculates restraint degrees of freedom.
`CalcMaterialMatrixPlStress()`	Calculates material matrix for plane stress condition.
`CalcMaterialMatrixPlStrn ()`	Calculates material matrix for plane strain condition.
`CalcMaterialMatrixBending()`	Calculates material matrix for bending properties.
`CalcAnalysisData ()`	Calculates analysis data such as degrees of freedom, material properties, element section properties, system properties material matrix etc.
`CalcStructureKMatrix ()`	Calculates structure stiffness matrix.
`CalcDeflections ()`	Calculates displacements at each node.
`CalcStresses ()`	Calculates stresses for each element.

Once the required data for analysis is processed by the method `CalcAnalysisData()`, the analysis is performed using the methods `CalcStructureKMatrix()`, `CalcDelection()` and `CalcStresses()`. Joint connectivity data is obtained for each element from the `Element` class. Nodal coordinates for each node of an element are then obtained by calling methods from the `Joint` class. Instances of classes representing CST, four node quadrilateral, DKT, DKQ, triangular shell and quadrilateral shell elements are declared as needed in the `CalcStructureKMatrix()`. Element stiffness matrices are then calculated by calling the corresponding method for the specific element. The element stiffness matrices for each element of the structure are then superimposed to generate the structure stiffness matrix. Deflections are then calculated using the structure stiffness matrix and the load values obtained from the `JointLoad` and `EqNodalLoads` classes respectively. The Gauss-Jordan method is used to solve the system of equations and compute deflections at each node of the structure. Element stresses are then computed in the `CalcStresses()` method. Instance of classes representing each element in the structure are created and the corresponding method for computing element stresses is called to compute stresses for that element. The stresses for each element are then stored in the vectors of the `StressResult` class. The class diagram for Analysis class is shown in Fig. 6.3.

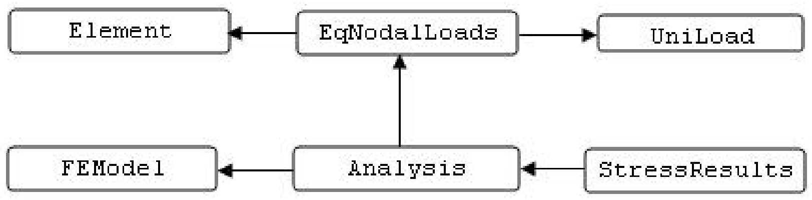


Fig. 6.3 `Analysis` Class Diagram.

`MembraneStr` Class

The `MembraneStr` class represents the stresses in a membrane element (either triangular or quadrilateral). The data members defined within this class are normal stress in the x and y directions, shearing stresses, maximum and minimum principal stresses and angle of the principal plane at each node of the membrane element. This class also contains the accessor methods that are called to obtain the membrane stresses. The methods defined within the `MembraneStr` class are given in the Table 6.5.

Table 6.4 Methods in the `MembraneStr` class

Method	Description
`GetS1 ()`	Returns an array of normal stresses in the x direction at each node of the element.
`GetS2 ()`	Returns an array of normal stresses in the y direction at each node of the element.
`GetS12 ()`	Returns an array of shearing stresses at each node of the element.
`GetSmax ()`	Returns an array of maximum principal stresses at each node of the element.
`GetSmin ()`	Returns an array of minimum principal stresses at each node of the element.
`GetANG ()`	Returns the array of angle of principal plane at each node of the element.

`BendingStr` Class

The bending stresses for an element are stored in the `BendingStr` class. This class stores the stresses normal to the x and y direction, shearing stresses, maximum and minimum principal stresses and the angle of principal plane at each node of the element. Several methods defined within the class are used to access the stored bending stresses for each eleme nt. The methods defined within `BendingStr` and their description is given in Table 6.6.

Table 6.5 Methods in the `BendingStr` class

Method	Description
`GetS1 ()`	Returns an array of normal stresses in the x direction at each node of the element.
`GetS2 ()`	Returns an array of normal stresses in the y direction at each node of the element.
`GetS12 ()`	Returns an array of shearing stresses at each node of the element.
`GetSmax ()`	Returns an array of maximum principal stresses at each node of the element.
`GetSmin ()`	Returns an array of minimum principal stresses at each node of the element.
`GetANG ()`	Returns an array of angle of principal plane at each node of the element.

`ShellStr` Class

The `ShellStr` class represents the shell stresses at the top and the bottom of a shell element. The arrays of stresses normal to the x and y direction, shearing stresses, maximum and minimum principal stresses and an array of angle of the principal plane are defined within this class. The `ShellStr` class also contains various methods that return the normal, shearing and principal stresses and angle of principal plane for each node of the element. The methods are described in Table 6.6.

Table 6.6 Methods in the `ShellStr` class

Method	Description
`GetS1 ()`	Returns an array of normal stresses in the x direction at each node of the element.
`GetS2 ()`	Returns an array of normal stresses in the y direction at each node of the element.
`GetS12 ()`	Returns an array of shearing stresses at each node of the element.
`GetSmax ()`	Returns an array of maximum principal stresses at each node of the element.
`GetSmin ()`	Returns an array of minimum principal stresses at each node of the element.
`GetANG ()`	Returns an array of angle of principal plane at each node of the element.

`StressResults` Class

The stresses for each element are computed separately in the analysis process. Therefore, it is necessary to store the computed stresses for each element dynamically. The `StressResults` stores computed stresses for each element.

In the `StressResult` class, the Java class `vector` is used to store the computed stresses dynamically. The stresses for each element are stored in the instances of the `MembreaneStr`, `BendingStr` or `ShellStr` classes depending upon the type of element. The objects of these classes are stored in the vectors defined in the `StressResult` class. The vectors defined within the `StressResults` class are `MembraneStresses`, `BendingStresses`, and `ShellStresses`. Each of these vectors stores the objects containing values of membrane stresses, bending stresses, and shell stresses respectively depend on what element is being analyzed. The methods defined in the `StressResults` class are described in Table 6.7.

Table 6.7 Methods in the `StressResults` class

Method	Description
`AddMembraneStresses ()`	Creates an instance of the `MembransStr` class and adds it to the `vector MembraneStresses`.
`AddBendingStresses ()`	Creates an instance of the `BendingStr` class and adds it to the `vector BendingStresses`.
`AddShellStresses ()`	Creates an instance of the `ShellStr` class and adds it to the `vector ShellStresses`.

An instance of the `StressResults` class is created in the `CalcStresses()` method of the `Analysis` class. This instance is then passed to the `CalcStresses()` method of the `CSTElement`, `QUAD4Element`, `DKTElement`, `DKQElement`, `TriShellElement` or `QuadShellElement` classes as an argument.. The methods `AddMembraneStresses()`, `AddBendingStresses()` or `AddShellStresses()` are then called to store the computed stresses from that element. The stresses for each element are stored to the vectors dynamically within these methods using the method `AddElement()` of Java class library. Fig. 6.4 represents the diagram for the `StressResults` class.

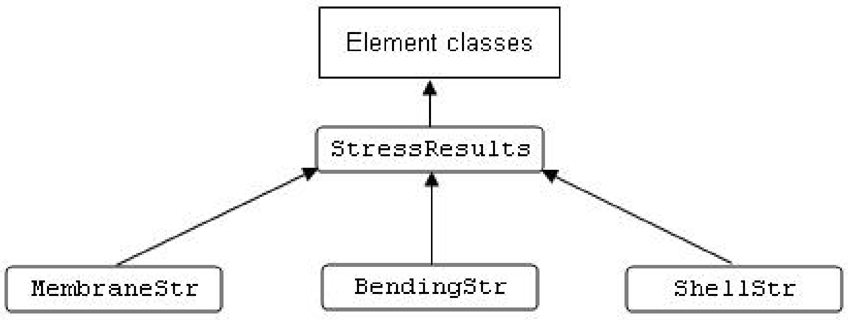


Fig. 6.4 `StressResults` Class Diagram.

6.2.2 Input and Output Classes

The geometry and structural details of the finite element model are provided to a program in the form of an input. In this study, the finite element model is generated using the interface provided by SAP 2000 and the input text file is created from this model. This input text file is used (with minor modifications). The results obtained from the analysis are saved to a text file. The `ReadInput` class reads the input file and saves the structure data. The `WriteOutput` class writes writes the displacements and stresses obtained from the `Analysis` class.

`ReadInput` Class

The input for the program is provided in the form of text file. Hence, it is necessary to read the provided input from the text file and store the input data in an instance of the `FEModel` class from where it can be used for analysis. The `ReadInput` class performs the task of reading the provided data and transfers this information to the `FEModel` class where the data is stored in different vectors. The input for the program consists of the system properties, joint coordinates, joint restraints, material properties, section properties, element joint connectivity and loads applied to the structure. Several methods provided in the `ReadInput` class to read this data from a file and to store this in an instance of the `FEModel` class. The methods defined within the class and its description is given in Table 6.8.

Table 6.8 Methods in the `ReadInput` class

Method	Description
`ReadTextFile ()`	Reads the input file.
`ReadSystem ()`	Reads system data from the SYSTEM block of the input file.
`ReadJoint ()`	Reads joint coordinates from the JOINT block of the input file.
`ReadRestraints ()`	Reads restraint data from the RESTRAINT block of the input file.
`ReadMaterial ()`	Reads material data such as modulus of elasticity and Poisson’s ratio from the MATERIAL block of the input file.
`ReadShellSection()`	Reads thickness of the element from the SHELL SECTION block of the input file.
`ReadConnectivity()`	Reads joint connectivity data from the SHELL block of the input file.
`ReadLoads ()`	Reads data of applied load such as joint loads, uniformly distributed surface load from the LOAD block of input file.
`InputReader ()`	Calls all methods listed above.

An instance of the `FEModel` class is created in the method `InputReader`. This instance is then passed to all the other methods in the `ReadInput` class. The methods in the `ReadInput` class are described below.

The method `ReadTextFile()` reads the input text file and stores it as a string. The method `ReadSystem()` reads the number of system degrees of freedom and the flag values for the system degrees of freedom in each direction. The method `AddSystemProperties()` is called in the `ReadSystem()` method from the `FEModel` class that stores the system properties. The method `ReadJoints()` reads the x, y, and z coordinates of each node. The method `AddJoint()` of `FEModel` class is called for each node in the structure. It saves the node number and x, y, and z coordinates for that node. The method `ReadRestraints()` reads the restraints provided at each node and assigns equation numbers corresponding to degrees of freedom in each direction for that node. The node number and an array representing the equation number for that node is then stored in an object of the `FEModel` class by calling the method `AddRestraint()` of `FEModel` class.

The method `ReadMaterial()` reads material properties for the structure such as modulus of elasticity and Poisson's ratio. The method `AddMaterial()` of the `FEModel` class is called to store these material properties. The method `ReadShellSection()` reads the shell section properties such as thickness of the elements. The shell thickness is then stored in the `FEModel` class by calling the method `AddSectionProp()` of the `FEModel` class. The method `ReadConnectivity()` reads joint connectivity data for each element. The method `AddElement()` is called for each element and the element number, number of nodes for that element and the array representing the joint connectivity for that element is stored in the `FEModel` object. The method `ReadLoads()` reads the type of load, load cases, values of loads and multiplication factor for a particular load case. The loads are then stored in the `FEModel` object by calling the methods `AddJtLoad()` and `AddUniLoad()` for storing the joint loads and uniformly distributed surface loads applied to the structure.

`WriteOutput` Class

The results obtained from the analysis must be stored for later use during the analysis of the structure. Due to the large amount of output generated by the program, it is convenient to store the results in the text file. The path of the text file where the results are to be stored is passed through the constructor of the `WriteOutput` class. The results from the analysis are then printed to this saved text file in the `WrteOutput` class.

The `WriteOutput` class contains methods that print the stored results to the text file. The method `WriteResults()` defined in this class is declared public. This method tabulates and prints the displacements at each node and the stresses for each element. The vectors `MembraneStresses`, `BendingStresses`, and `ShellStresses` contain objects of the `MembraneStr`, `BendingStr` and `ShellStr` classes for each element in which the values of the membrane, bending and shell stresses are stored for that element. These stored stresses are then written to the output file by calling the accessor methods defined within the `MembraneStr`, `BendingStr` and `ShellStr` classes.

6.2.3 Interface Classes

The program contains a very simple user interface since all the input and output operations are done using text files. There are two steps for analyzing the structure using the program. The first is to read the input data from the input text file and second is to run an analysis. The interface of the program contains a menu bar that has the Import, Run, and Exit and Help commands for performing the tasks of importing the input text file, running an analysis, exiting the program and displaying description of the program. The classes used for the user interface and application window are the `VTFEA`, `MainFrame` and `MainFame_AboutBox` classes. Each class is described in the following paragraphs.

`VTFEA` Class

The purpose of this class is to create an instance of the main application window and to begin program execution. The only variable declared in this class is `packframe`, which is initialized to false. Setting `packframe` to false results in a call to the `pack()` method of the Java class library `UIManager` which sets the size and location of the main application window on the screen.

An instance of the class `MainFrame` is declared in the constructor to `VTFEA`. This results in the creation of main application window. Once the main application frame is constructed, various methods such as `pack()`, `validate()`, `setsize()`, and `setvisible()` of the Java class library `UIManager` are called. These methods set the size of the main application window and display it on the screen.

The `VTFEA` class also contains the `main` method. The execution of the program begins with the main method. An instance of the `VTFEA` class is created inside the main.

`MainFrame` class

The `MainFrame` class represents the main application window of the `VTFEA` application. The purpose of this class is to create the application window and assign various user interface objects such as title, border, minimize and maximize button, and a menu bar. All of these user interface components are derived from the Java Foundation Classes (JFC). The main application window is shown in Fig. 6.2.

The `MainFrame` class includes logic for implementing various means such as reading the input file, performing the analysis, storing analysis results to a text file, and terminating program execution. It also contains logic for displaying a dialog box that contains information regarding the program. The main actions are implemented using the event handling features provided in the Java event handling classes.

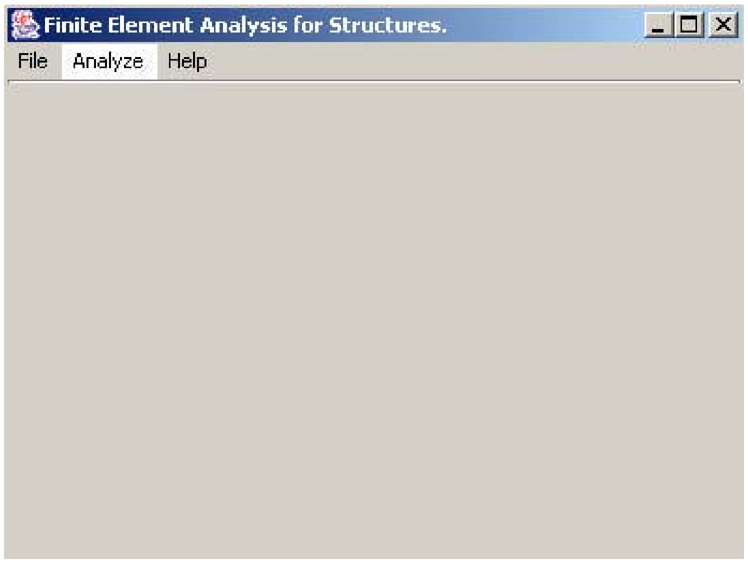


Fig. 6.5 Main Application Window for the Program.

An instance of `ReadInput` is created when the “Input” menu item is selected. The methods `Inputreader` is called that reads and stores the input data when the “Import” command is pressed from the “File” menu. Instances of the `Analysis` class and

WriteOutput class are created when the “Run” menu item from the “Analyze” menu is selected. The methods RunAnalysis() and the WriteResults() from the Analysis and WriteOutput classes respectively are called to perform the analysis and to print the displacements and stresses obtained from the analysis to the text file.

MainFrame_AboutBox Class

The “About” dialog box provides information about the program. The MainFrame_AboutBox class represents the window for the “About” dialog box. The Java Foundation Classes (JFC) are used to design the various interface components of the dialog box. The instance of this class is created when the “About” menu item is selected from the “Help” menu of the main application window and a dialog box appears on the screen that provides the details of the program.

Chapter 7
Test Examples and Verification of Results

In this chapter, a comparison of results for several test examples analyzed using the developed program for this research with those obtained from a commercial finite element analysis program SAP 2000, is presented. The results compared include maximum displacements and maximum average stresses.

7.1 Example Problems for Verification of Three Node Triangular (CST) Element

The accuracy of the CST element developed in this study is verified by analyzing three test examples using the developed program and the commercial finite element analysis program SAP 2000. The results are compared at different nodes.

<u>Test Example 1</u>

This test example consists of a cantilever beam of length 48 in., depth 12 in. and thickness of 1 in. and is taken from Chen (1992). The beam is modeled using 8 three node triangular (CST) elements. Vertical loads of 20 kips are applied at the free end of the cantilever (nodes 5 and 10). Fig. 7.1 shows the finite element model of the cantilever beam.

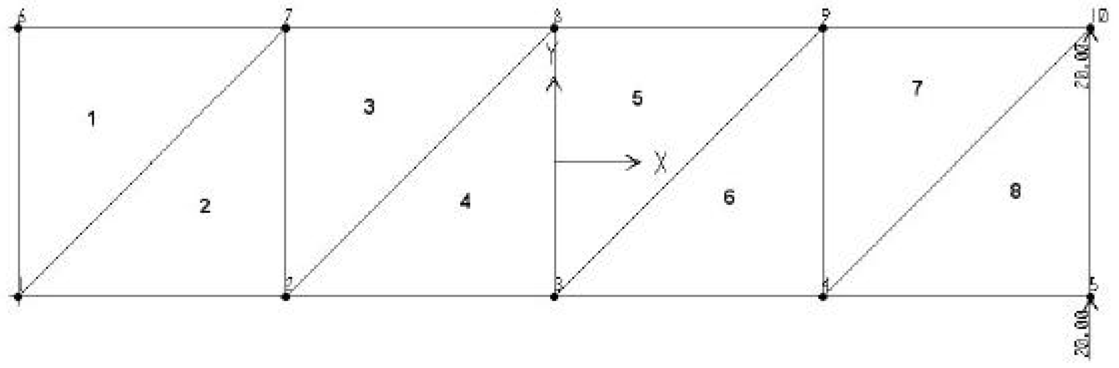


Fig. 7.1 FE Model for Test Example 1 – Cantilever Beam.

Geometric Data:

Length L = 48.0 in.

Depth h = 12 in.

Thickness t = 1.0 in.

Material Properties:

Modulus of elasticity E = 30000 ksi.

Poisson's ratio n = 0.25

Boundary Conditions:

Restraints are provided in the x and y directions at the left end of the cantilever (nodes 1 and 6).

Loading:

A concentrated load of 20 kips is applied to nodes 5 and 10.

Comparison of Results:

The results from the analysis obtained from the program and from SAP 2000 are shown in Table 7.1. The results shown are the displacements at nodes 10 and 8, and stresses at node 6. From Table 7.1, it is seen that the results given by the program are identical to those given by SAP 2000. Similar results were obtained for the other nodes.

Table 7.1 Displacements and Stresses for Test Example 1

Location		Result from Program	Result from SAP-2000	% Difference
Node – 10	**UX**	-0.014159	-0.014159	**0.00%**
	UY	0.090347	0.090347	**0.00%**
Node – 8	**UX**	-0.010825	-0.010825	**0.00%**
	UY	0.030403	0.030403	**0.00%**
Node - 6	**S11**	-17.128727	-17.128727	**0.00%**
	S22	-4.282182	-4.282182	**0.00%**
	S12	9.537940	9.537940	**0.00%**

Test Example 2

The second test example consists of the same cantilever beam of Example 1. The length of the beam is 48 in., depth is 12 in. and the thickness is 1 in. The finite element model now consists of 32 three node triangular (CST) elements. Vertical loads totaling 40 kips is applied at the free end of the cantilever beam. The finite element model of the cantilever beam as shown in Fig. 7.2.

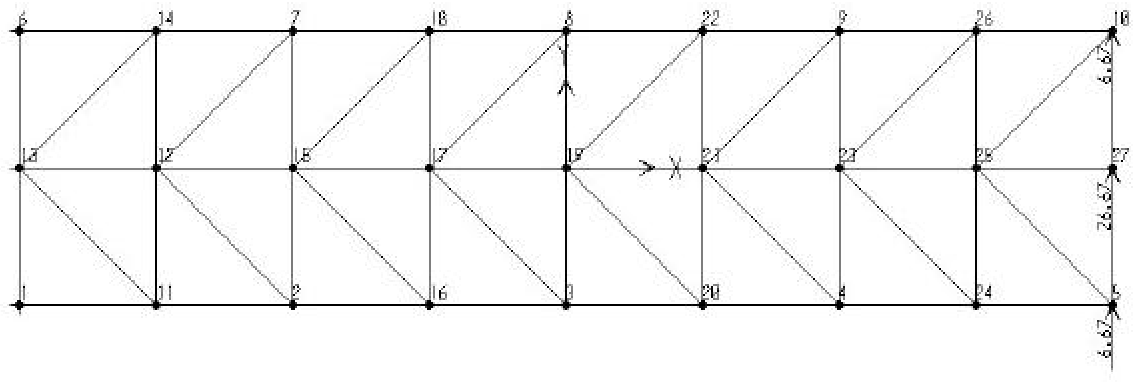


Fig. 7.2 FE Model for Test Example 2 – Cantilever Beam.

Geometric Data:

Length L = 48.0 in.

Depth h = 12 in.

Thickness t = 1.0 in.

Material Properties:

Modulus of elasticity E = 30000 ksi.

Poisson's ratio n = 0.25

Boundary Conditions:

Restraints are provided in the x and y directions at the left end of the cantilever.

Loading:

A concentrated load P_1 = 6.67 kips is applied at node 10.

A concentrated load P_2 = 26.67 kips is applied at node 27.

A concentrated load P_3 = 6.67 kips is applied at node 5.

Comparisons of Results:

Table 7.2 represents the results obtained for the second test example from the developed program and SAP 2000. The displacements at nodes 10 and 8 and stresses at node 6 are shown. The results indicate that the percentage difference in each case is less than 0.02%.

Table 7.2 Displacements and Stresses for Test Example 2

Location		**Result from Program**	**Result from SAP-2000**	**% Difference**
Node – 10	**UX**	-0.034263	-0.034271	**-0.023%**
	UY	0.194412	0.194456	**-0.023%**
Node – 8	**UX**	-0.025599	-0.025605	**-0.023%**
	UY	0.062956	0.062971	**-0.024%**
Node - 6	**S11**	-41.493731	-41.503067	**-0.022%**
	S22	-10.373433	-10.375767	**-0.022%**
	S12	11.840936	11.843600	**-0.022%**

Test Example 3

The third verification example is a plate with semi circular hole of radius 3 in. at the center. The plate is fixed at the top. A downward vertical load of 0.67 kips/in is applied at the free edge. The length of the plate is 16 in., the width is 6 in. and the thickness is 0.45 in. A plate is modeled using 110 three node triangular (CST) elements. The FE model is shown in Fig 7.3.

Geometric Data:

Width L = 16.0 in.

Width b = 6.0 in.

Thickness t = 0.45 in.

Material Properties:

Modulus of elasticity E = 30000 ksi.

Poisson’s ratio n = 0.3

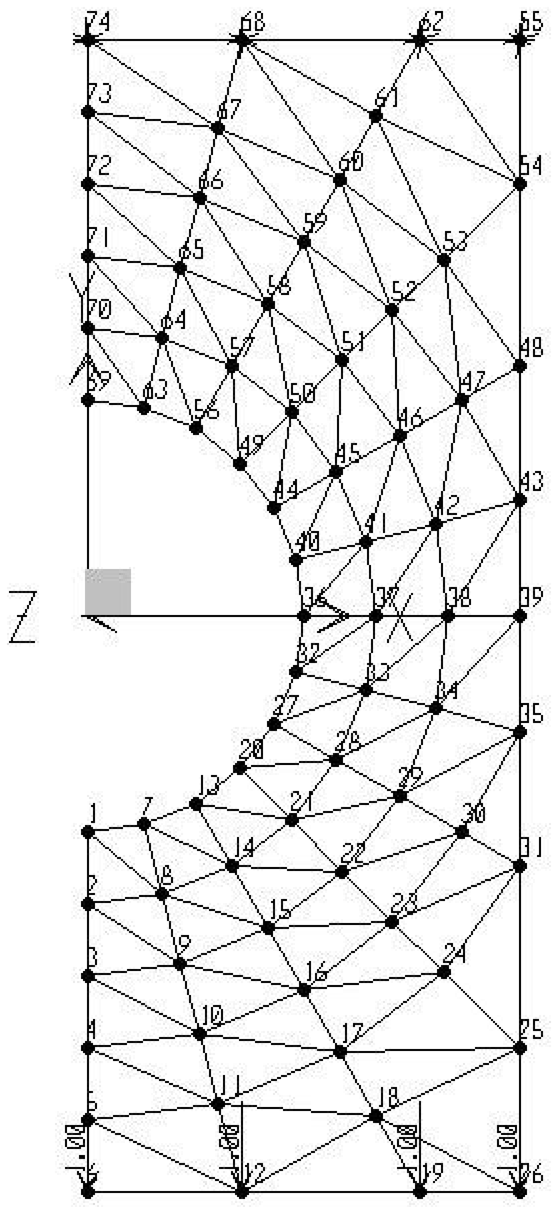


Fig. 7.3 FE Model for Test Example 3 – Plate with Semi Circular Hole.

Boundary Conditions:

The plate is restrained at the top in both the x and y directions.

Loading:

Concentrated loads of 1.0 kips are applied at each node at the bottom of the plate. i.e. nodes 6, 12, 19, and 26.

Comparison of Results:

The displacements at nodes 1 and 6 and stresses at nodes 32 and 36 are given in Table 7.3. As can be seen from the table, results obtained from the developed program and those from SAP 2000 are same. Similar results are obtained for the other nodes.

Table 7.3 Displacements and Stresses for Test Example 3

Location		Result from Program	Result from SAP-2000	% Difference
Node – 1	**UX**	0.001545	0.001545	**0.00%**
	UY	-0.003132	-0.003132	**0.00%**
Node - 6	**UX**	0.004213	0.004213	**0.00%**
	UY	-0.003355	-0.003355	**0.00%**
Node - 32	**S11**	1.250061	1.250061	**0.00%**
	S22	7.050394	7.050394	**0.00%**
	S12	0.480698	0.480698	**0.00%**
Node - 36	**S11**	1.156083	1.156083	**0.00%**
	S22	7.547362	7.547362	**0.00%**

7.2 Example Problems for Verification of Four Node Quadrilateral Plane Element

Three example problems were selected to verify the accuracy of the four node quadrilateral plane element. The structures for the second and third example problems are the same as those analyzed using the three node triangular (CST) elements.

Test Example 4

The first example consists of a cantilever beam of length 6 in., depth 0.8 in. and thickness 0.2 in. A concentrated vertical load of 10 kips is applied at the free end of the cantilever in the downward direction. The finite element model of the cantilever beam is generated using three four node quadrilateral plane elements as shown in Fig. 7.4.

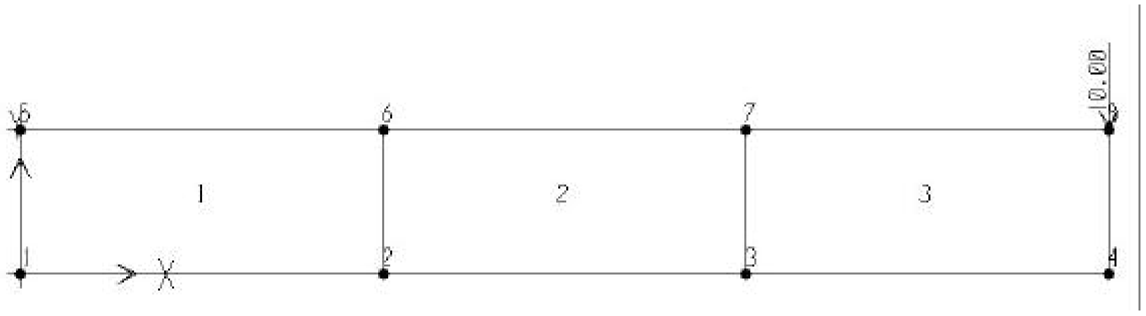


Fig. 7.4 FE Model for Test Example 4 – Tip Loaded Cantilever Beam.

Geometric Data:

Length L = 6.0 in.

Depth h = 0.8 in.

Thickness t = 0.2 in.

Material Properties:

Modulus of elasticity E = 30000 ksi.

Poisson's ratio n = 0.3

Boundary Conditions:

Restraints are provided in the x and y directions at the left end of the cantilever.

Loading:

A concentrated load of 10 kips is applied to the free end (node 8) of the cantilever.

Comparison of Results:

Table 7.4 represents the results for displacements at nodes 8 and 3, and stresses at node 1 obtained from the program and SAP 2000. The comparisons shown in the table suggest that the displacements and stresses obtained from the program are in good agreement with those obtained from SAP 2000.

Table 7.4 Displacements and Stresses for Test Example 4

Location		Result from Program	Result from SAP-2000	% Difference
Node – 8	**UX**	0.016107	0.016110	**-0.018%**
	UY	-0.162708	-0.162735	**-0.016 %**
Node – 3	**UX**	-0.014282	-0.014285	**-0.015%**
	UY	-0.084639	-0.084652	**-0.021%**
Node - 1	**S11**	-146.96544	-146.99074	**-0.017%**
	S22	-44.089634	-44.097225	**-0.017%**
	S12	-141.122551	-141.144703	**-0.015%**

Test Example 5

The second example problem to verify the four node quadrilateral plane element is similar to the cantilever beam of Example 2. The length of the cantilever beam is 48 in., depth is 12 in. and thickness is 1 in. A vertical loads totaling 40 kips are applied at the free end of the cantilever (Chen, 1992). The beam is modeled using 20 four node quadrilateral plane elements. The finite element model for the cantilever beam is shown in Fig. 7.5.

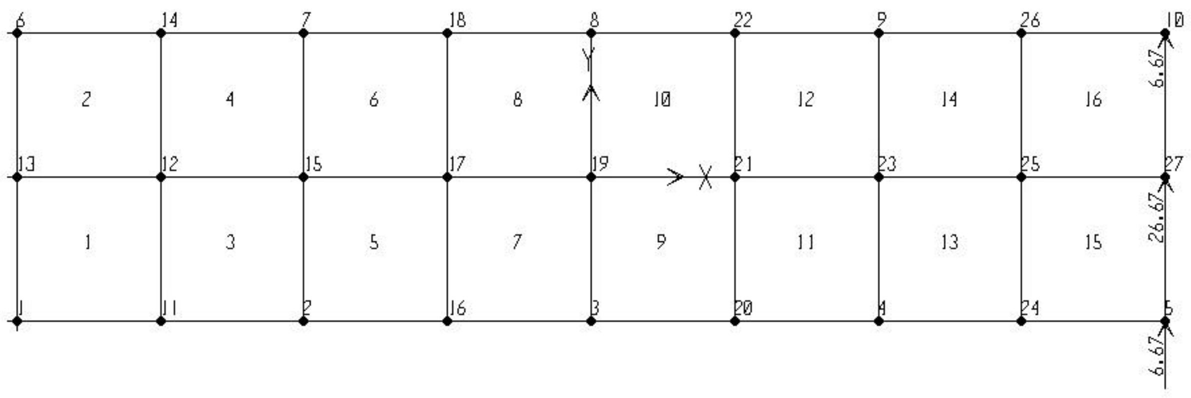


Fig. 7.5 FE Model for Test Example 5 - Cantilever Beam.

Geometric Data:

Length L = 48.0 in.

Depth h = 12 in.

Thickness t = 1.0 in.

Material Properties:

Modulus of elasticity E = 30000 ksi.

Poisson's ratio n = 0.25

Boundary Conditions:

Restraints are provided at the left end (nodes 1, 13, and 6) of the cantilever.

Loading:

A concentrated load P_1 = 6.67 kips is applied at node 10.

A concentrated load P_2 = 26.67 kips is applied at node 27.

A concentrated load P_3 = 6.67 kips is applied at node 5.

Comparison of Results:

The displacements at nodes 10 and 8 are shown in Table 7.5. Also shown in the table are the stresses at node 6. The difference in displacements obtained from the program is less than 1%. The stresses obtained from the program are almost identical to those obtained from SAP 2000.

Table 7.5 Displacements and Stresses for Test Example 5

Location		Result from Program	Result from SAP-2000	% Difference
Node – 10	UX	-0.057484	-0.057074	**0.718%**
	UY	0.317406	0.316064	**0.425%**
Node – 8	UX	-0.042772	-0.042774	**-0.005%**
	UY	0.101260	0.101265	**-0.005%**
Node - 6	S11	-70.300130	-70.304114	**-0.006%**
	S22	-17.575032	-17.576029	**-0.006%**
	S12	18.407756	18.408688	**-0.006%**

Test Example 6

The same example problem of a plate with a semicircular hole, analyzed in Example 3 was chosen for the verification of the four node quadrilateral elements.. The plate is fixed at the top. A downward vertical load of 0.67 kips/in is applied to the free edge of the plate. The length of plate is 16 in., the width is 6 in. and the thickness is 0.45 in. The plate is modeled using 52 four node quadrilateral plane elements. To model the boundaries of the plate, 6 three node triangular (CST) elements were also included in the FE model. This test example was chosen to verify the accuracy of the program for a hybrid mesh that includes two different types of elements i.e. three node triangular and four node quadrilateral elements. The finite element model for the plate is shown in Fig. 7.6.

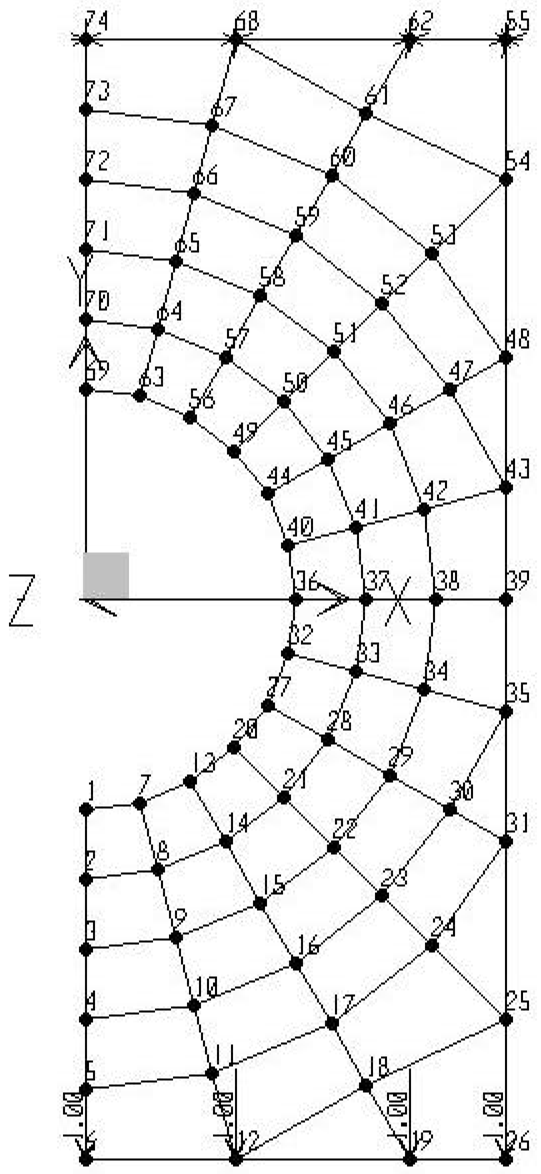


Fig. 7.6 FE Model for Test Example 6 - Plate with Semi Circular Hole.

Geometric Data:

Length L = 16.0 in.

Width b = 16 in.

Thickness t = 0.45 in

Material Properties:

Modulus of elasticity E = 30000 ksi.

Poisson’s ratio n = 0.3

Boundary Conditions:

Restraints are provided at top of the plate (i.e., nodes 55, 62, 68, and 74)

Loading:

Concentrated loads of 1.0 kips are applied at each node at the bottom of the plate. i.e., nodes 6, 12, 19, and 26.

Comparison of Results:

Table 7.6 represents the comparison of displacements and stresses obtained at various nodes from both programs. The displacements at nodes 1 and 6 are tabulated. The differences in results obtained from the two programs are less than 2 %. The stresses at nodes 32 and 36 are obtained from the program are compared to those from SAP 2000. The differences in stresses are in the range 4% to 7%. The element stresses are calculated at the element edge using the derivatives of the displacements. The stresses calculated at the edge of one element may differ significantly from the stresses calculated at the edge of an adjacent element. The reason for this difference is that the stresses are not in equilibrium with the externally applied traction and hence are not continuous across element boundaries. The difference may be significant if the generated finite element mesh is coarse. This difference can be reduced using a finer mesh. Therefore, stresses averaged at any single point may be higher or lower depending on mesh pattern. The commercial finite element analysis program SAP 2000 uses an error estimation scheme that reduces the error in stresses and thus give results that are more accurate. However, the results obtained here from the program are in good agreement with those obtained from the SAP 2000.

Table 7.6 Displacements and Stresses for Test Example 6

Location		Result from Program	Result from SAP-2000	% Difference
Node – 1	**UX**	0.001951	0.001984	**-1.663%**
	UY	-0.003847	-0.003852	**-0.129%**
Node-6	**UX**	0.005384	0.005437	**-0.974%**
	UY	-0.004070	-0.004075	**-0.122%**
Node - 32	**S11**	2.12129	1.98348	**6.947%**
	S22	10.76729	10.57779	**1.791%**
	S12	1.653674	1.585048	**4.329%**
Node - 36	**S11**	1.91307	1.775941	**7.721%**
	S22	12.28807	11.841794	**3.768%**

7.3 Example Problems for Verification of Three Node Triangular Plate Bending (DKT) Element

To verify the accuracy of the three node plate elements, four finite element models were analyzed using the program and SAP 2000. The results for these four test cases are described in the following sections.

Test Example 7

The first test example is a square plate of size 12 ft $\times$ 12 ft having a thickness of 6 in. The plate is subjected to a uniform surface load of 0.1 kips/sq. in. All four edges of the plate are fixed. The finite element model was generated using 128 three node triangular plate bending (DKT) elements.

Two mesh patterns were considered in the analysis of the plate. Both mesh patterns have the same number of nodes and elements. The only difference between the two mesh patterns is the orientation of the elements. The finite element models for the two mesh patterns are shown in Figures 7.7 and 7.8.

Geometric Data:

Length L = 144.0 in.

Width = 144.0 in.

Thickness t = 6.0 in.

Material Properties:

Modulus of elasticity E = 3600 ksi.

Poisson's ratio n = 0.2

Boundary Conditions:

All four edges of the plate are fixed.

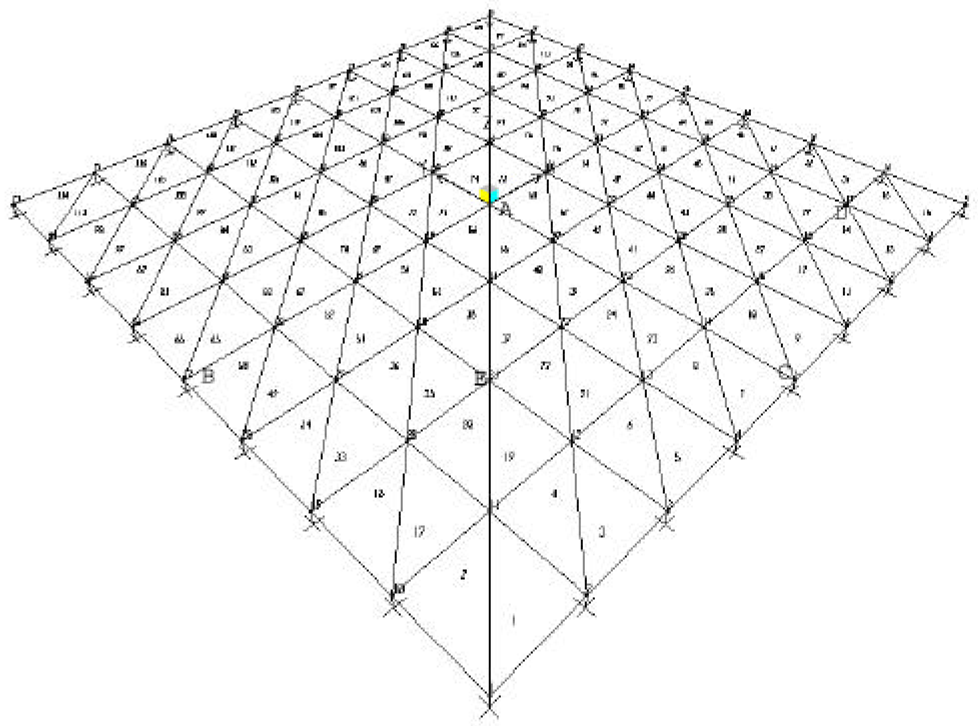


Fig. 7.7 FE Model for Test Example 7 – Square Plate– Mesh Pattern A.

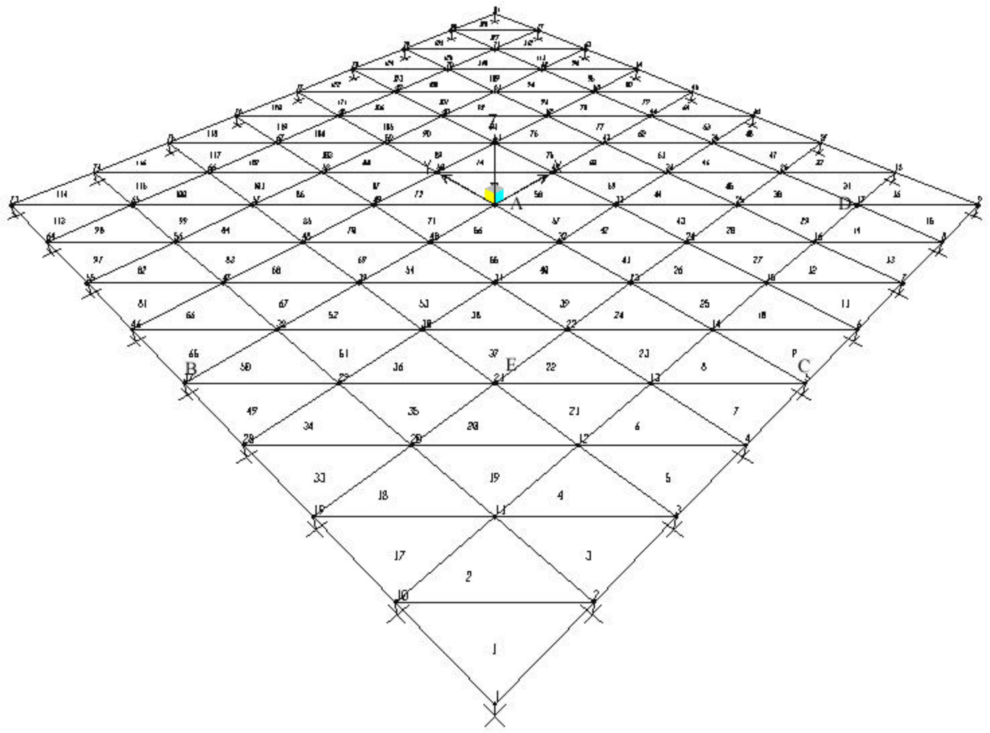


Fig. 7.8 FE Model for Test Example 7 – Square Plate– Mesh Pattern B.

Loading:

A uniform surface load of 0.1 kips/sq. in. is applied to the plate.

Comparison of Results:

The displacements at nodes A and E, and the stresses at nodes B, C, and D for mesh pattern A and B are shown in Tables 7.7 and 7.8. It is seen from the tabulated values that among the two mesh patterns, mesh pattern A gives more accurate results than mesh pattern B. A similar observation was made by Batoz *et al.* (1980). The stresses at nodes B, C and D for SAP 2000 are taken from the stress diagram of the SAP 2000 graphical user interface rather than from the SAP 2000 output file for the structures analyzed using triangular plate elements. The comparison of stresses at various nodes to those obtained from the stress diagram of SAP 2000 graphical user interface are in good agreement.

Table 7.7 Displacements and Stresses for Test Example 7 - Mesh Pattern - A

Location		Results from Program	Results from SAP-2000	% Difference
Point – A (Node – 41)	**UZ**	-0.82345	-0.82920	**-0.694%**
Point – E (Node – 21)	**UZ**	-0.295672	-0.304909	**-3.029%**
Point – B (Node – 37)	**S11**	15.44666	16.8050	**9.538%**
Point – C (Node – 5)	**S22**	17.30481	16.9098	**2.336%**
Point – D (Node – 17)	**S12**	-2.87364	-3.0190	**-4.185%**

Table 7.8 Displacements and Stresses for Test Example 7 – Mesh Pattern - B

Location		Results from Program	Results from SAP-2000	% Difference
Point – A (Node – 41)	**UZ**	-0.79443	-0.82920	**-4.193%**
Point – E (Node – 21)	**UZ**	-0.27932	-0.30010	**-6.924%**
Point – B (Node – 37)	**S11**	14.02095	16.8584	**-16.831%**
Point – C (Node – 5)	**S22**	14.85041	16.9312	**-12.289%**
Point – D (Node – 17)	**S12**	-2.90833	-2.9001	**6.241%**

Test Example 8

The second example is a rectangular plate of size 8 ft × 12 ft with a hole of size 4 ft × 2 ft. The thickness of the plate is 10 in. A uniform surface load of 0.2 kips/sq. in. is applied over the entire plate. All inside and outside edges of the plate are simply supported. The finite element model of the plate contains 96 three node triangular plate bending (DKT) elements (See Figures 7.9 and 7.10).

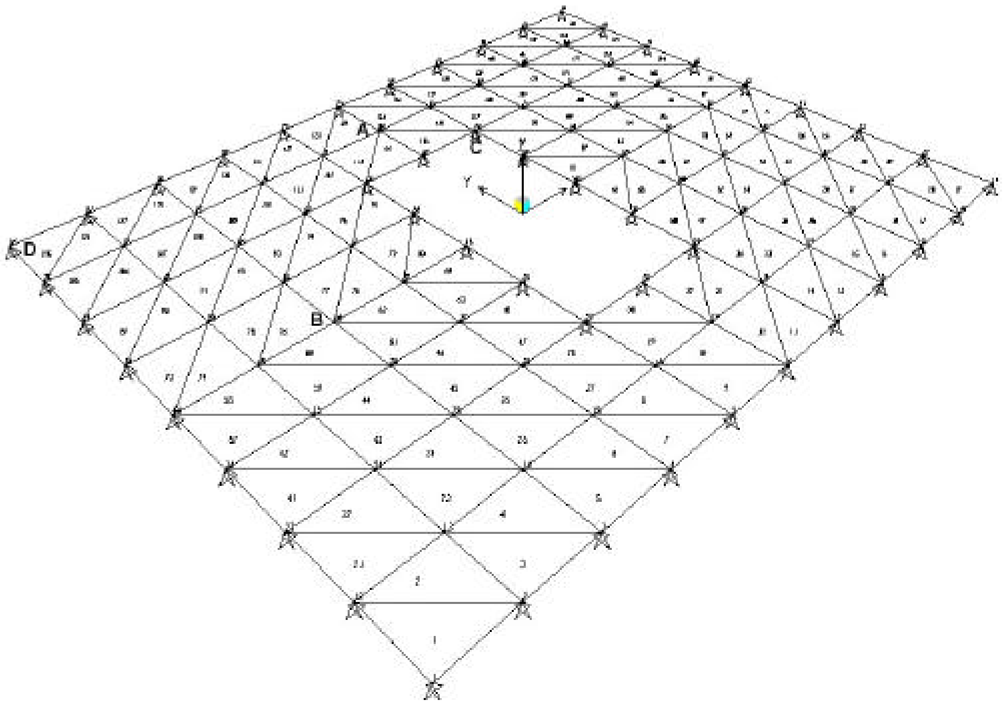


Fig. 7.9 FE Model for Test Example 8 – Rectangular Plate with Hole.

Geometric Data:

Out side length L_1 = 96.0 in.

Outside width B_1 = 144.0 in.

Inside length L_2 = 48 in.

Inside width B_2 = 24 in.

Thickness t = 10.0 in.

Material Properties:

Modulus of elasticity E = 3600 ksi.

Poisson's ratio n = 0.2

Boundary Conditions:

All inside and outside edges of the plate are simply supported.

Loading:

A uniform surface load of 0.2 kips/sq. in. is applied to the plate.

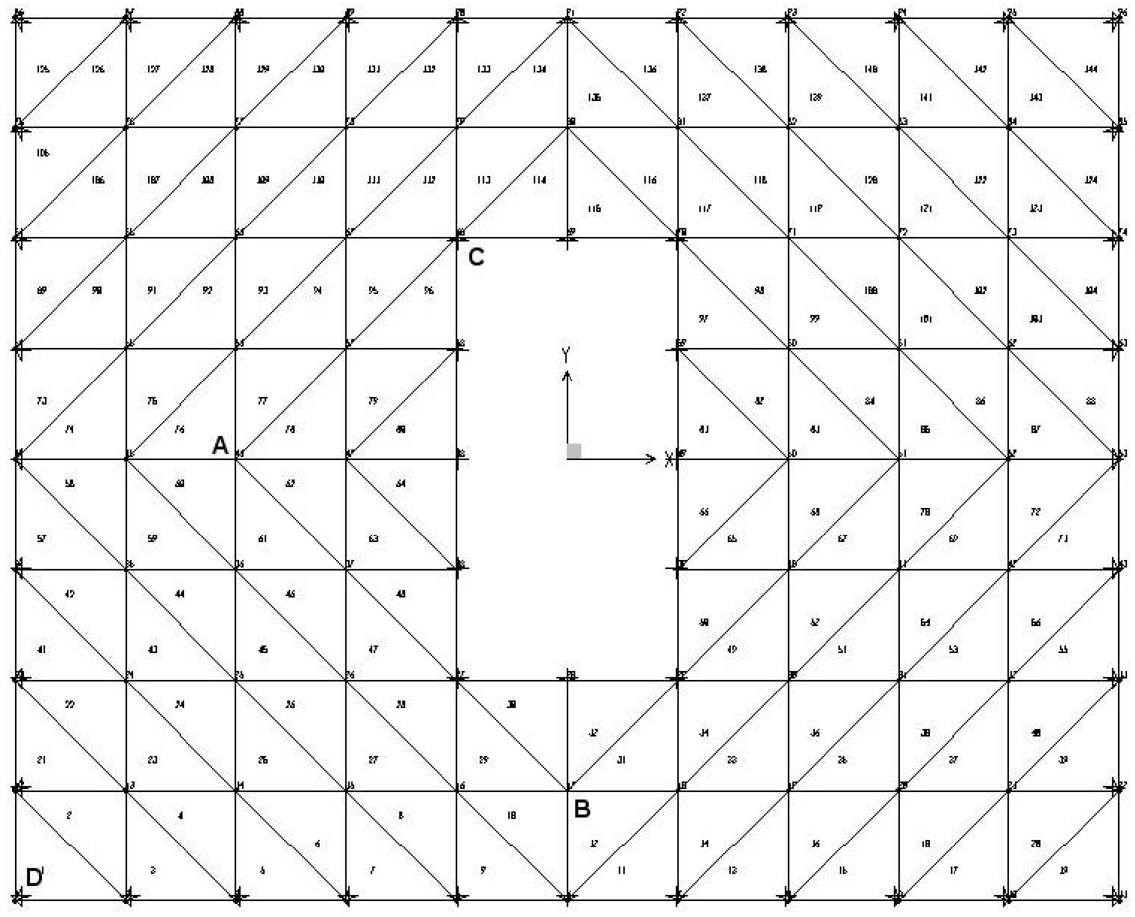


Fig. 7.10 FE Model for Test Example 8 – Rectangular Plate with Hole.

Comparisons of Results:

The comparison of displacements and stresses obtained from the program and SAP 2000 for this example is tabulated in Table 7.9. The displacements are compared at points A and B, while the stresses are compared at points C and D as shown in Figures 7.9 and 7.10. The results obtained from the program are found to be similar with those obtained from SAP 2000. However, the normal stress in the y direction at point D is approximately 9% less than that given by the SAP 2000. The difference in results is because of the different triangular plate bending elements used in the program and SAP 2000.

Table 7.9 Displacements and Stresses for Test Example 8

Location		**Result from Program**	**Result from SAP-2000**	**% Difference**
Point – A (Node – 46)	**UZ**	-0.021537	-0.021647	**-0.508%**
Point – B (Node – 17)	**UZ**	-0.001274	-0.001288	**-1.087%**
Point – C (Node – 68)	**S11**	3.02900	3.0126	**0.544%**
	S22	1.577566	1.7364	**-9.147%**
Point – D (Node – 1)	**S12**	1.022454	1.0103	**1.203%**

Test Example 9

The third verification problem is a cantilever plate of size 8 ft x 8 ft. The thickness of the plate is 6 in. A uniformly distributed surface load of 0.01 kips/sq. in. is applied to the plate. The finite element model consists 128 three node triangular plate bending (DKT) elements. The finite element model of the plate is shown in Fig. 7.11.

Geometric Data:

Length L = 96.0 in.

Width = 96.0 in.

Thickness t = 6.0 in.

Material Properties:

Modulus of elasticity E = 3600 ksi.

Poisson's ratio n = 0.2

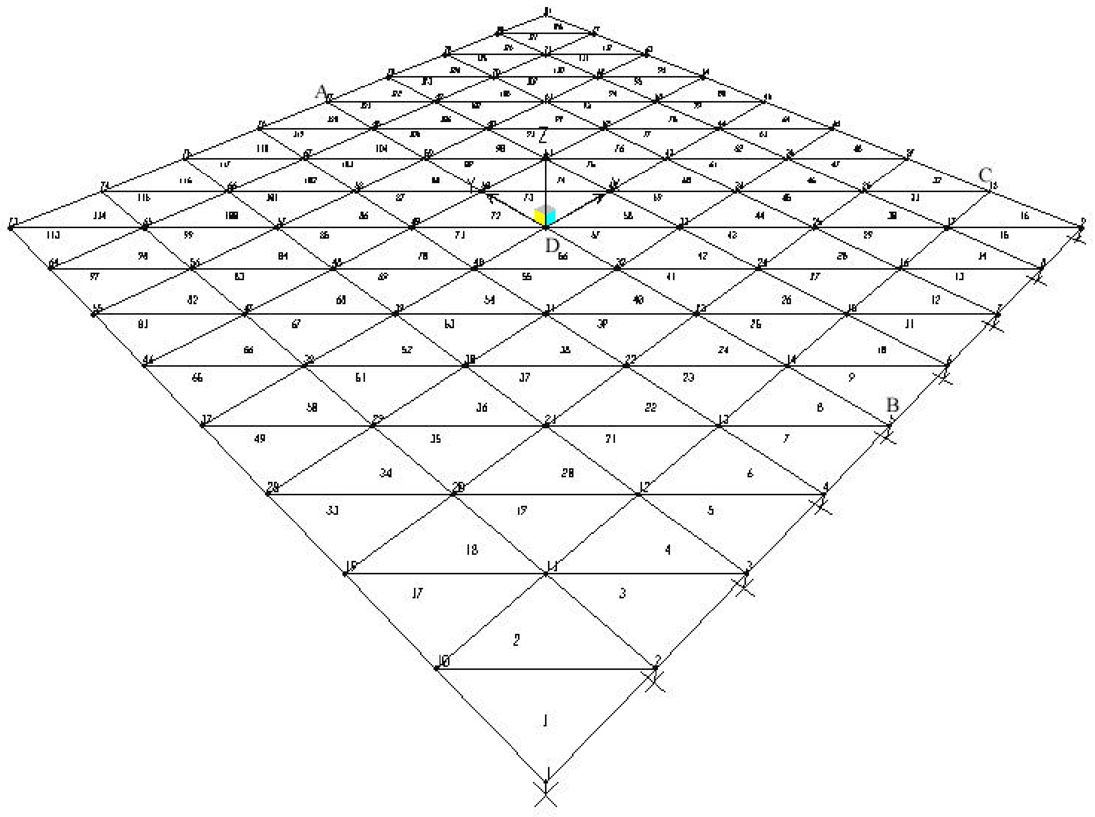


Fig. 7.11 FE Model for Test Example 9– Cantilever Plate.

Boundary Conditions:

One edge of the cantilever plate is fixed.

Loading:

A uniform surface load of 0.01 kips/sq. in. is applied to the plate.

Comparisons of Results:

Displacements at points A and D and stresses at points B and C obtained from the program are compared to those obtained from SAP 2000 (See Table 7.10). The displacements obtained from the program are within 5% from those obtained from SAP 2000. The difference between stresses obtained from the developed program and those obtained from the SAP 2000 is less than 5%. One reason for this difference is use of different elements in the author's program and SAP 2000. Another reason can be the higher order numerical integration scheme used by SAP 2000. SAP 2000 uses four to eight point numerical integration scheme for its quadrilateral and triangular element while the author has used three point numerical integration scheme in the formulation of the element stiffness matrix for DKT element.

Table 7.10 Displacements and Stresses for Test Example 9

Location		**Result from Program**	**Result from SAP-2000**	**% Difference**
Point – A (Node – 77)	**UZ**	-1.60400	-1.68787	**-4.960%**
Point – D (Node – 41)	**UZ**	-0.569561	-0.599412	**-4.980%**
Point – B (Node – 5)	**S11**	1.74398	1.8325	**-5.431%**
Point – B (Node – 5)	**S22**	8.07994	8.2984	**-2.633%**
Point – C (Node – 18)	**S12**	0.40987	0.4288	**-4.415%**

7.4 Example Problems for Verification of Four Node Quadrilateral Plate Bending (DKQ) Element.

Verification of the four node quadrilateral plate bending (DKQ) elements developed in Java is performed by analyzing the same three structures used for the verification of triangular plate bending elements. The structures are now modeled using four node quadrilateral plate bending (DKQ) elements. The description of the example problems and comparison of results is discussed in the following sections.

Test Example 10

The first example problem is a square plate of size 12 ft $\times$ 12 ft and has a thickness of 6 in. A uniform surface load of 0.1 kips/sq. in. is applied to the plate. All four edges of the plate are fixed. The finite element model was generated using 64 four node quadrilateral plate bending (DKQ) elements (See Fig. 7.12).

Geometric Data:

Length L = 144.0 in

Width = 144.0 in.

Thickness t = 6.0 in.

Material Properties:

Modulus of elasticity E = 3600 ksi.

Poisson's ratio n = 0.3

Boundary Conditions:

All four edges of the plate are fixed.

Loading:

A uniform surface load of 0.01 kips/sq. in. is applied to the plate.

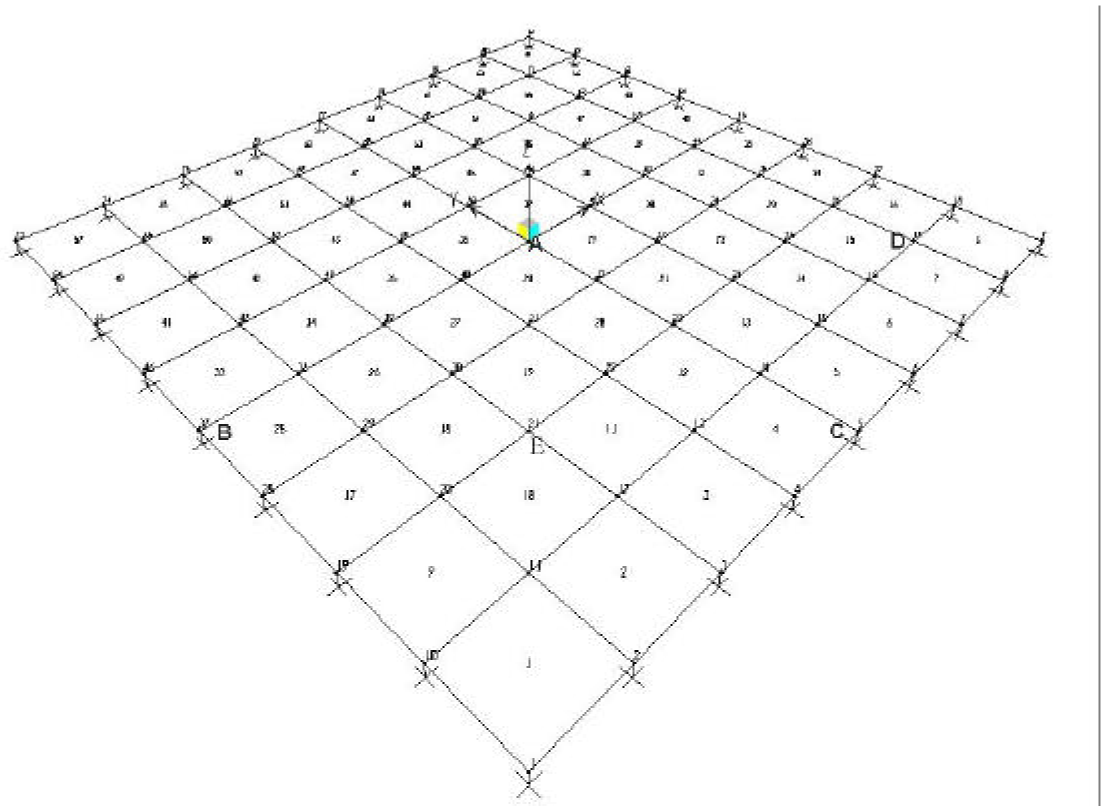


Fig. 7.12 FE Model for Test Example 10 – Square Plate

Comparison of Results:

Table 7.11 shows of displacements at points A and E and stresses at point B, C and D obtained from the program and from SAP 2000. The displacements at points A and E are approximately the same as those from SAP 2000. The difference is 0.006 % in the normal stresses in x and y direction at the fixed support (where the stresses are maximum). The difference in shearing stress from the program and that from the SAP 2000 is 5 %.

Table 7.11 Displacements and Stresses for Test Example 10

Location		Result from Program	Result from SAP 2000	% Difference
Point – A (Node – 41)	**UZ**	-0.84005	-0.84005	**-0.000%**
Point – E (Node – 21)	**UZ**	-0.304782	-0.304782	**0.000%**
Point – B (Node – 37)	**S11**	17.51227	17.51111	**-0.006%**
Point – C (Node –5)	**S22**	17.51227	17.51111	**-0.006%**
Point – D (Node – 17)	**S12**	-2.82021	-2.65871	**-5.726%**

Test Example 11

The second example is the same rectangular plate with hole, which was used for the verification of DKT element. A plate is of size 8 ft $\times$ 12 ft and has a hole of size 4 ft $\times$ 2 ft. The thickness of plate is 10 in. A uniform surface load of 0.2 kips/sq. in. is applied to the plate. All inside and outside edges of the plate are simply supported. A finite element discretization for this example problem includes 72 four node quadrilateral (DKQ) elements. (See Figures 7.13 and 7.14).

Geometric Data:

Out side length L_1 = 96.0 in.

Outside width B_1 = 144.0 in.

Inside length L_2 = 48 in.

Inside width B_2 = 24 in.

Thickness t = 10.0 in.

Material Properties:

Modulus of elasticity E = 3600 ksi.

Poisson’s ratio n = 0.2

Boundary Conditions:

All inside and outside edges of the plate are simply supported.

Loading:

A uniform surface load of 0.2 kips/sq. in. is applied to the plate.

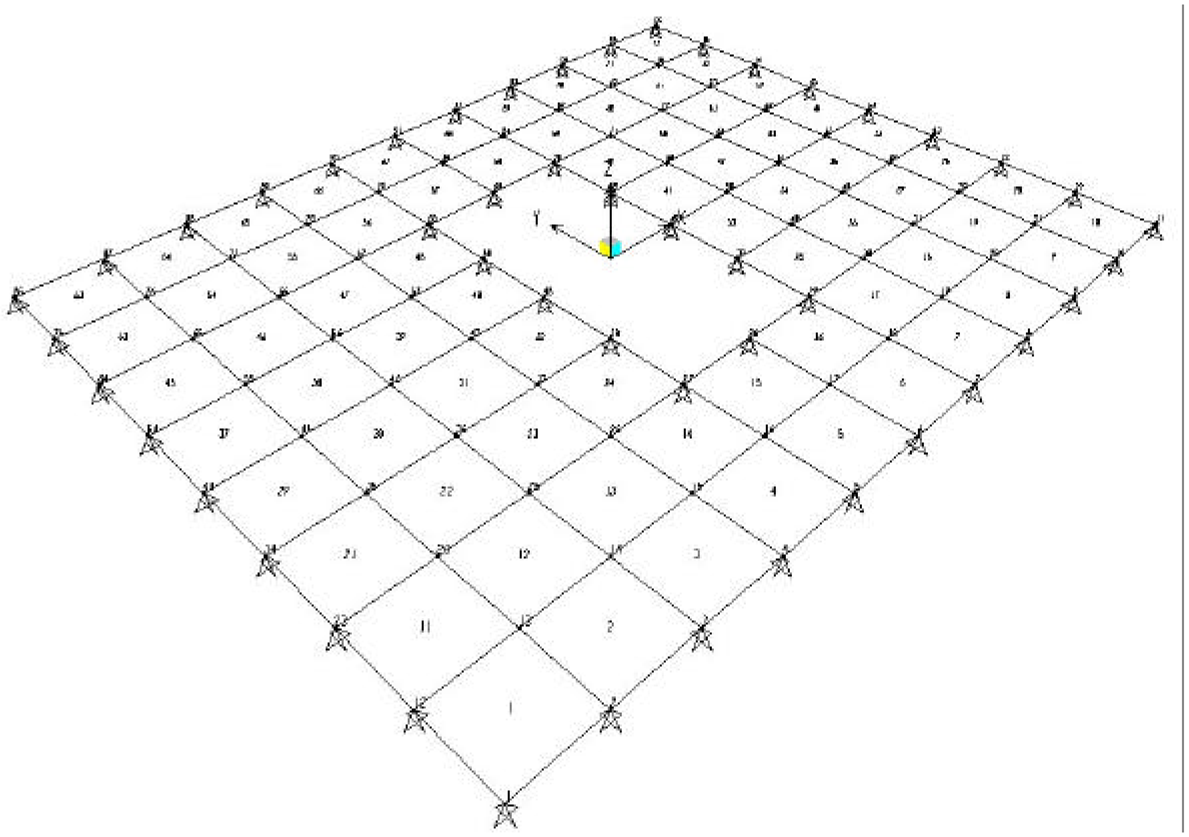


Fig. 7.13 FE Model for Test Example 11 – Rectangular Plate with Hole.

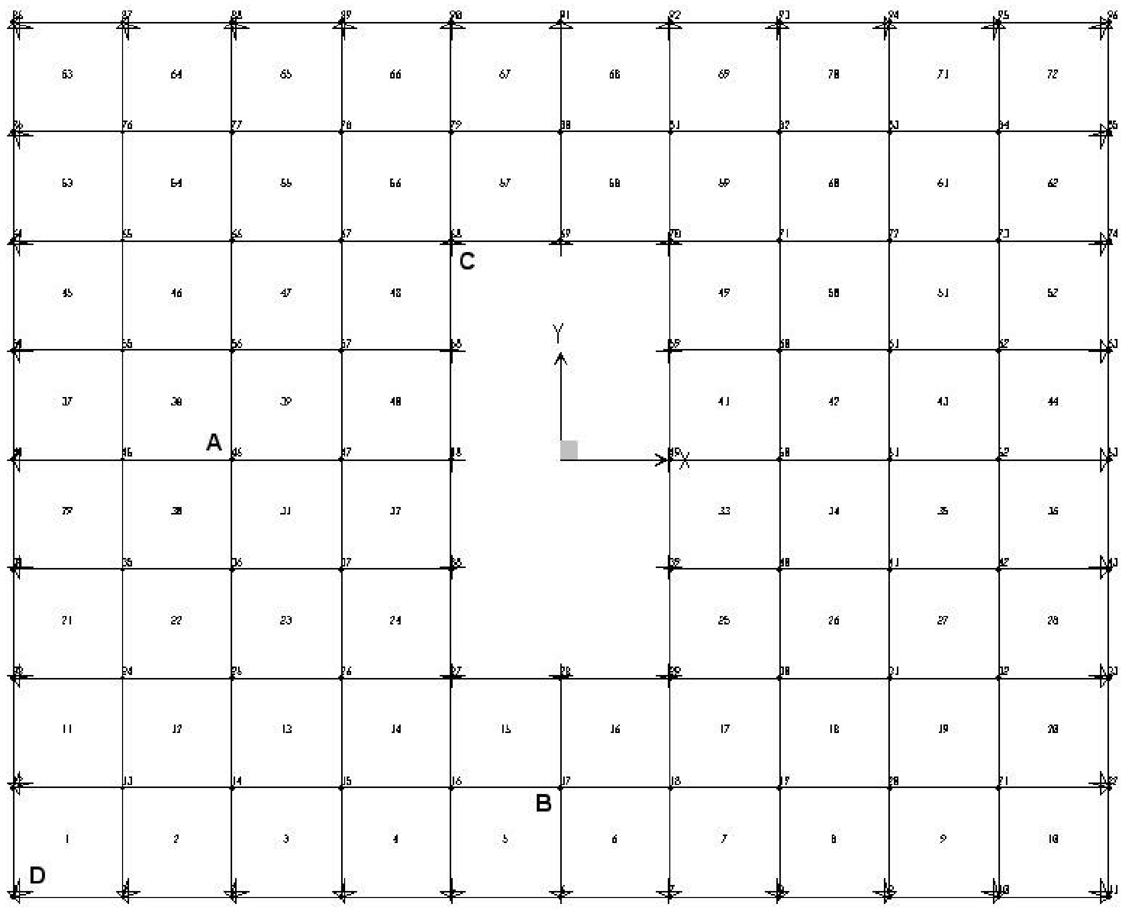


Fig. 7.14 FE Model for Test Example 11 – Rectangular Plate with Hole.

Comparison of Results:

The displacements and stresses obtained from the developed program and from SAP 2000 are tabulated in Table 7.12. The displacements are compared at points A and B while the stresses are compared at points C and D as shown in Figures 7.13 and 7.14. As can be seen from Table 7.12 the difference in displacements is insignificant. The stresses are also very close, however there is a difference of 5 % in shearing stress.

Table 7.12 Displacements and Stresses for Test Example 11

Location		**Result from Program**	**Result from SAP - 2000**	**% Difference**
Point – A (Node – 46)	**UZ**	-0.030028	-0.030028	**0.00%**
Point – B (Node – 17)	**UZ**	-0.001273	-0.001272	**-0.078%**
Point – C (Node – 68)	**S11**	3.7204	3.7202	**-0.005%**
	S22	1.637831	1.63769	**-0.008%**
Point – D (Node – 1)	**S12**	1.149302	1.095553	**-4.676%**

Test Example 12

The third verification example for the four node quadrilateral plate bending (DKQ) element consists of same cantilever plate that was used for the verification of the triangular plate bending (DKT) element. A cantilever plate is of size 8 ft x 8 ft and has a thickness of 6 in. A uniformly distributed surface load of 0.01 kips/sq. in. is applied to a plate. The plate is modeled using 64 four node quadrilateral plate bending (DKQ) elements. A finite element model of the plate for this example problem is shown in Fig. 7.15.

Geometric Data:

Length L = 96.0 in.

Width = 96.0 in.

Thickness t = 6.0 in.

Material Properties:

Modulus of elasticity E = 3600 ksi.

Poisson's ratio n = 0.2

Boundary Conditions:

One edge of cantilever plate is fixed.

Loading:

A uniform surface load of 0.01 kips/sq. in. is applied to the plate.

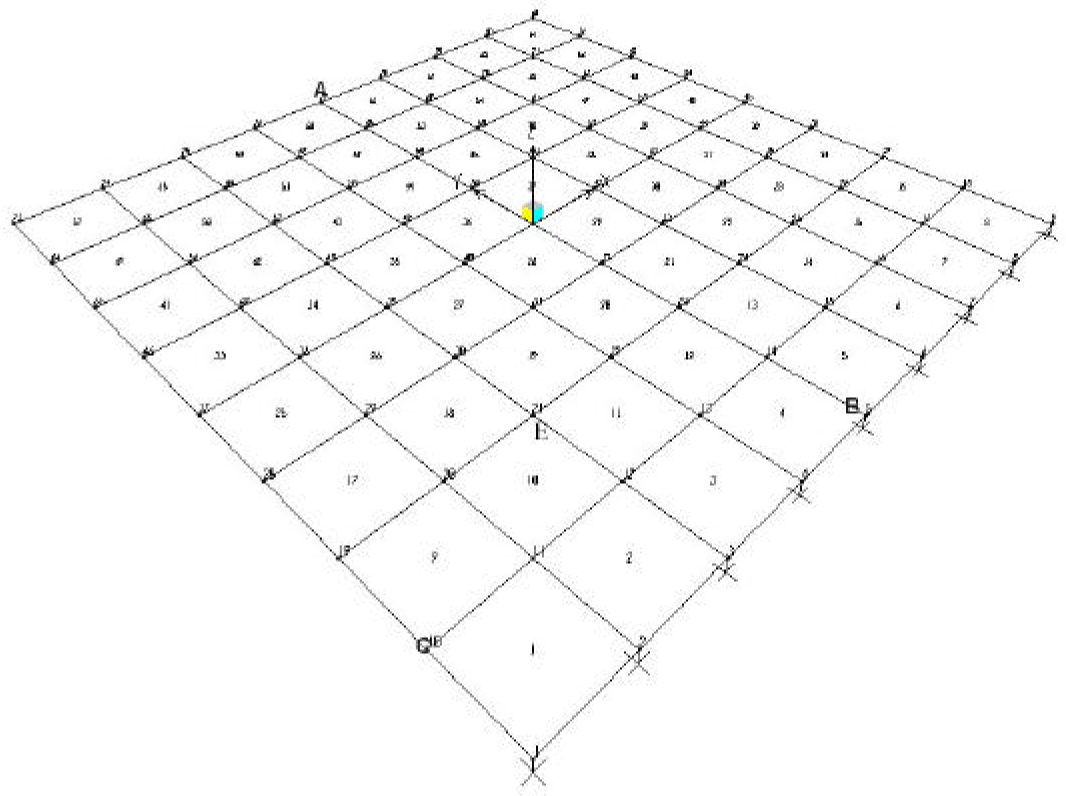


Fig. 7.15 FE Model for Test Example 12 – Cantilever Plate.

Comparisons of Results:

Table 7.13 shows the comparison of displacements and stresses obtained from the developed program and SAP 2000. The displacements are compared at nodes A and E (See Fig. 7.15). It is seen that the displacements obtained from the program are identical to those obtained from the SAP 2000. A similar observation can be made for the normal stresses. However, the shearing stress compared at point C differed by approximately 7%. It can be concluded from the tabulated results that the four node quadrilateral plate bending elements developed for this program is efficient and gives satisfactory results as compared to commercial finite element analysis program SAP 2000.

Table 7.13 Displacements and Stresses for Test Example 12

Location		**Result from Program**	**Result from SAP 2000**	**% Difference**
Point – A (Node – 77)	**UZ**	-1.605057	-1.605057	**0.00%**
Point – E (Node – 21)	**UZ**	-0.169455	-0.169455	**0.00%**
Point – B (Node – 5)	**S11**	1.593686	1.593686	**0.00%**
Point – B (Node – 5)	**S22**	7.968432	7.968432	**0.00%**
Point – C (Node – 10)	**S12**	0.455595	0.488347	**-6.706%**

7.5 Example Problems for Verification of Three Node Triangular Shell Elements

The three node triangular shell element developed in this study is a combination of CST element and DKT element. To verify the accuracy of three node triangular shell element, three example problems were selected. The comparison of displacements and stresses for each example problem is done at different points and the discussion is presented in the following sections.

Test Example 13

A cantilever I – beam is analyzed using the developed program and the SAP 2000. The length of the I – beam is 40 in., the height is 5 in. and the flange widths are 10 in. A load of 1.6 kips is applied at the top and bottom flanges of the I - beam in two opposite directions as shown in Fig. 7.16. This example is one of the verification examples presented in Alladin v. 1.0. A finite element model of the cantilever beam consists of 96 three node triangular shell elements.

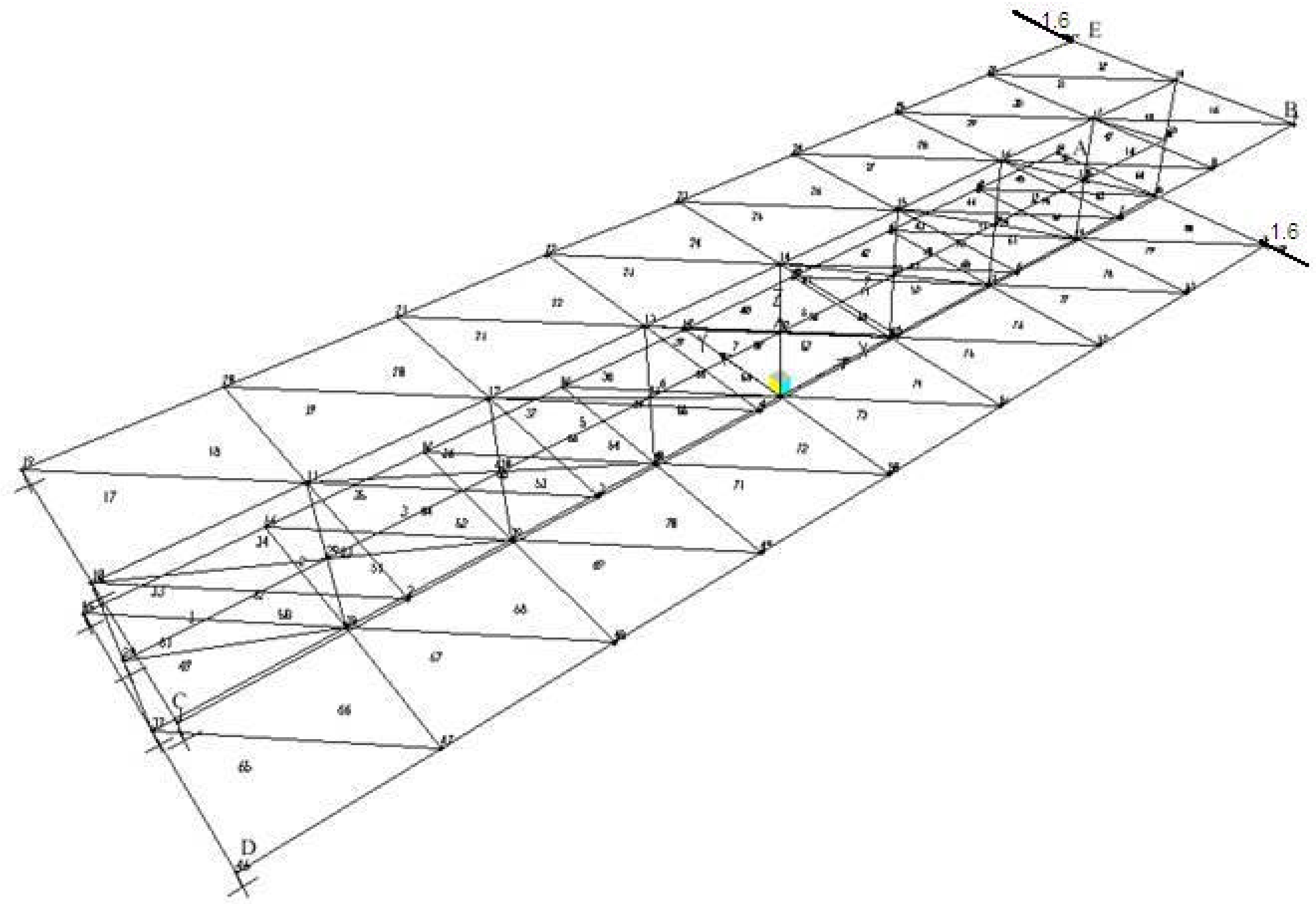


Fig. 7.16 FE Model for Test Example 13 Cantilever I – Beam

Geometric Data:

Length L = 40.0 in.

Width = 10.0 in.

Height h = 5.0 in.

Thickness t = 0.25 in.

Material Properties:

Modulus of elasticity E = 10000 ksi.

Poisson's ratio n = 0.3

Boundary Conditions:

One end of the cantilever is fixed.

Loading:

A concentrated load of 1.6 kips is applied at the top and bottom of the flange in opposite directions as shown in Fig. 7.16.

Comparison of Results:

This verification example was selected, as it is very efficient in determining the effect of inplane rotations in the element. The displacements at nodes 63 and 9 and the stresses at nodes 2, 46 and 27 are compared to those obtained from the SAP 2000 (See Table 7.14). The percentage difference in displacements is within 3 %, which is acceptable because the different plate bending elements are used for the development of three node triangular shell elements in the author's program and SAP 2000. The stresses from the program differ considerably from those obtained from SAP 2000. In this test example, torsional load is applied to the structure that causes inplane rotations in the elements. These inplane rotations are not included in the shell elements developed in this research while the triangular shell element in SAP 2000 include the rotational degrees of freedom in the development of element stiffness matrix of the shell element and hence is more accurate. The results indicate that the shell element developed is suitable for computing deflections but additional work is necessary to modify the element to take inplane rotations in to account. This is left on a topic of future study.

Table 7.14 Displacements and Stresses for Test Example 13

Location		**Result from Program**	**Result from SAP-2000**	**% Difference**
Point – A (Node – 63)	**UX**	-0.015376	-0.014921	**3.049%**
	UY	0.088021	0.085471	**2.983%**
	UZ	0.150498	0.146070	**3.031%**
Point – B (Node – 9)	**UX**	-0.015281	-0.014834	**3.013%**
	UY	-0.088024	-0.085475	**2.982%**
	UZ	-0.148408	-0.144533	**2.681%**

Table 7.14 Displacements and Stresses for Test Example 13 (Continue)

Point – C (Node – 2)	**S11**	-6.517417	-5.441100	**19.78%**
Point – D (Node – 46)	**S22**	2.688420	2.009748	**33.769%**
Point – E (Node – 27)	**S12**	-1.4427	-1.450483	**0.539%**

Test Example 14

The second verification example is a folded plate structure as shown in the Fig. 7.18. The length of the plate is 7.62 in. and the thickness is 1 in. A uniformly distributed surface load of 1 kips/ sq. in. is applied at the top and the two inclined sides of the plate. A finite element model of the folded plate structure is generated using 128 three node triangular plate bending elements. The folded plate structure is a good example for studying the effect of membrane and bending coupling that occurs at the edges of the shell elements. The finite element model of the folded plate is shown in Fig. 7.18

.

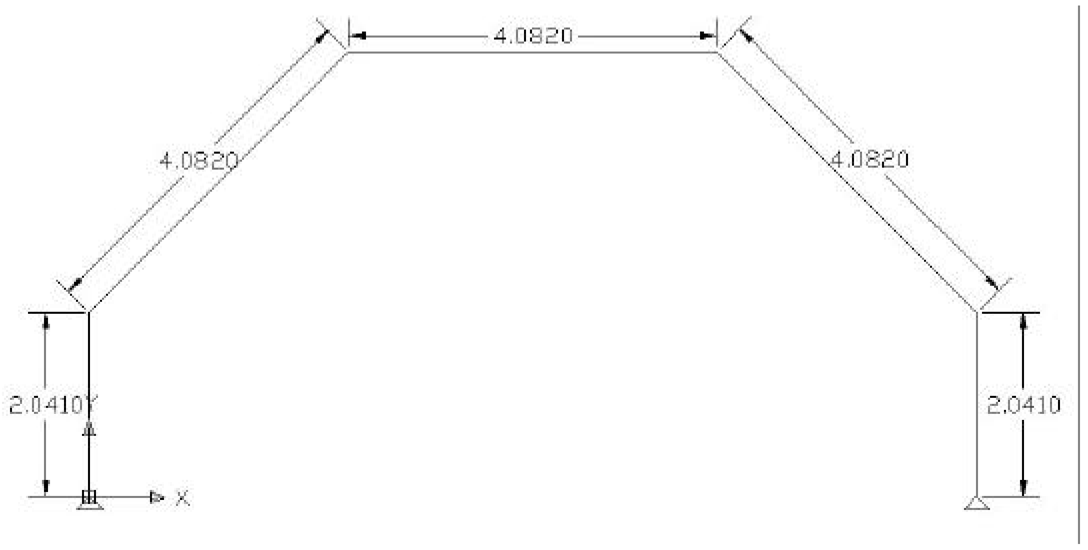


Fig. 7.17 Dimensions of the Folded Plate Structure

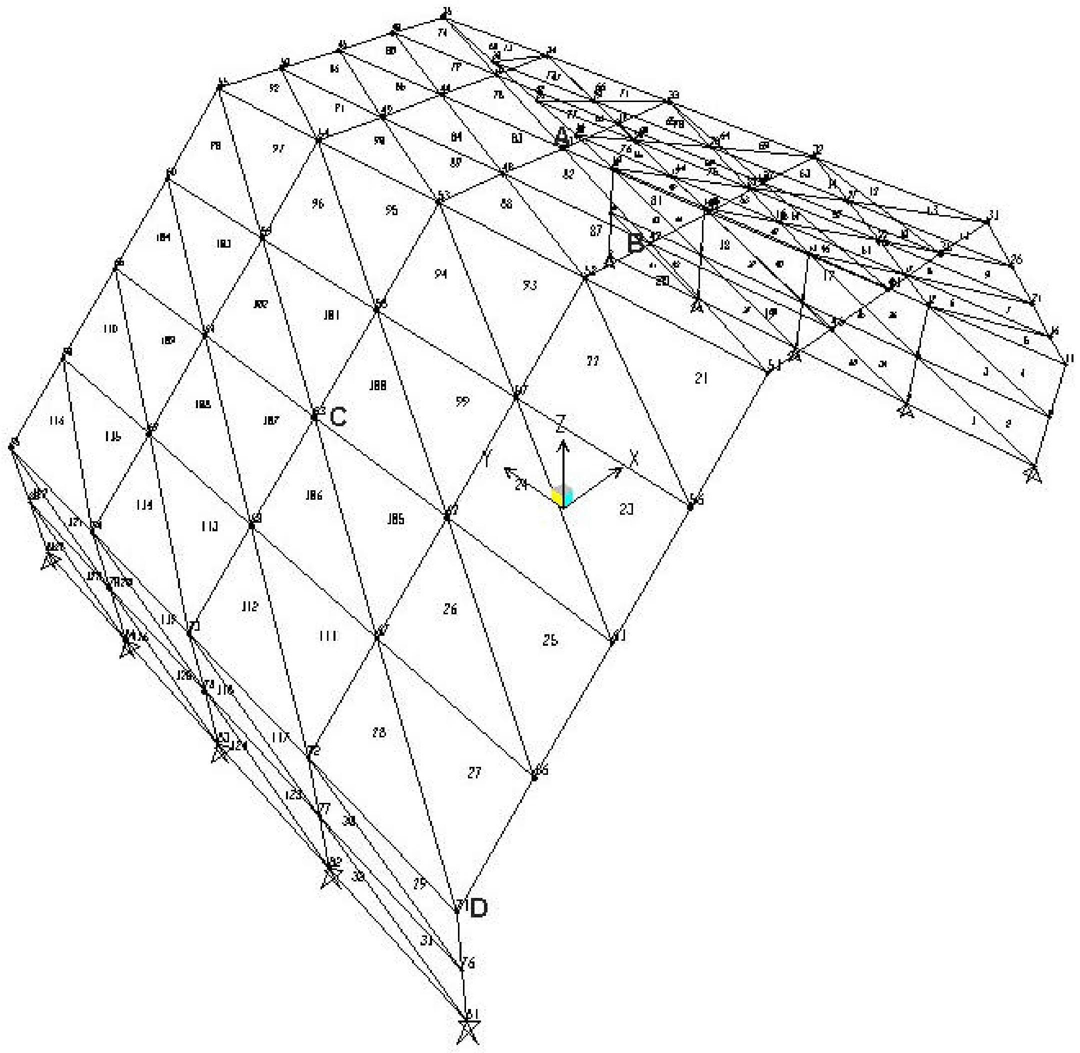


Fig. 7.18 FE Model for Test Example 14 – Folded Plate.

Geometric Data:

Length of the plate L = 7.62 in.

The other geometric data are shown in Fig. 7.17

Material Properties:

Modulus of elasticity E = 3600 ksi.

Poisson's ratio n = 0.2

Boundary Conditions:

The folded plate structure is simply supported at the bottom.

Loading:

A uniformly distributed surface load of 1 kips/sq.in. is applied to the top and two inclined faces of the folded plate.

Comparison of Results:

Table 7.15 shows analysis results at a few points obtained from the program and SAP 2000. The displacements at nodes 43 and node 63 which are the center nodes of the top and inclined faces of the folded plate structure are shown in the table. It can be seen that displacements obtained from the program are almost identical except for the displacements in y direction that have percentage difference of approximately 2 %. The stresses at nodes 43, 63 and 71 given by the program are different from those obtained from SAP 2000. Again, as in the previous example the reason for this difference is due to the fact that drilling degrees of freedom are not included in the triangular shell elements developed in this research. The inclusion of drilling degrees of freedom can have significant effect on stresses while inplane bending stresses are present in the structure.

Table 7.15 Displacements and Stresses for Test Example 14

Location		**Result from Program**	**Result from SAP-2000**	**% Difference**
Point – A (Node – 43)	**UX**	0.003291	0.003293	**-0.061%**
	UY	0.000567	0.000581	**-2.410%**
	UZ	-0.012889	-0.012844	**0.350%**
Point – C (Node – 63)	**UX**	0.008567	0.008551	**0.187%**
	UY	0.000422	0.000431	**-2.088%**
	UZ	-0.013089	-0.013055	**0.260%**
Point – A (Node – 43)	**S11**	-4.02949	-3.62061	**11.293%**
Point – C (Node – 63)	**S22**	-6.03507	-6.90214	**-12.562%**
Point – D (Node 71)	**S12**	-0.362579	-0.28533	**27.074%**

Test Example 15

The third verification problem is the standard test for triangular shell elements which is the Scordelis – Lo roof problem (Chen, 1992). The length of the roof is 50 in. and the radius of 25 in. The angle of inclination is $40°$. A uniform surface load of 90 kips/sq. in. is applied to the cylindrical shell roof. Restraints are provided at the two ends of the roof. The geometric and material properties are described in the following sections. The finite element mesh is generated using 32 three node triangular shell elements. The finite element model of Scordelis – Lo roof is shown in Fig. 7.19.

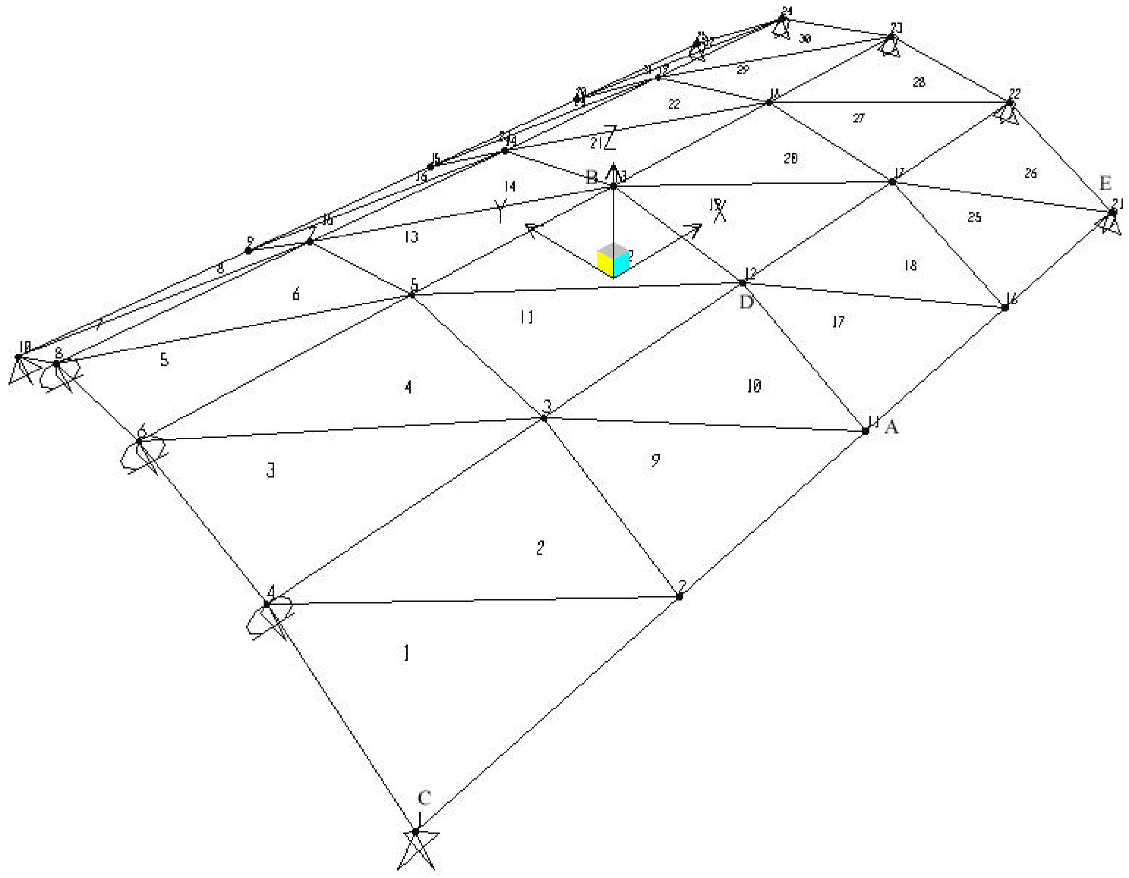


Fig. 7.19 FE Model for Test Example 15 - Scordelis – Lo Roof.

Geometric Data:

Length of the plate L = 50.0 in.

Radius = 25.0 in.

Angle f = 40°

Material Properties:

Modulus of elasticity E = 43200000 ksi.

Poisson’s ratio n = 0

Loading:

A uniformly distributed surface load of 90 kips/sq.in. is applied to the roof.

Comparison of Results:

The standard test consists of computing the downward displacement at center of the free edge of the roof (node 11). The exact value of this displacement is 0.3024 in. The displacement at node 11 from the program and from SAP 2000 are in good agreement with the theoretical value. The percentage difference is less than 3% (See Table 7.16). The displacements at the top of the roof (node 13) are also compared to those obtained from the SAP 2000 and are found satisfactory. Stresses at nodes 1, 12 and 21 are compared and the percentage difference was approximately less than 5 % in results was observed. It can be concluded that for this test example, that the developed triangular flat shell element gives accurate results, however; the SAP 2000 triangular shell element is more accurate since the inplane rotational degrees of freedom are included and hence more accurately represents the inplane bending behavior of the structure.

Table 7.16 Displacements and Stresses for Test Example 15

Location		**Result from Program**	**Result from SAP-2000**	**% Difference**
Point – A (Node – 11)	**UX**	0.002265	0.002148	**5.447%**
	UY	0.178983	0.176408	**1.460%**
	UZ	-0.328725	-0.323447	**1.632%**
Point – B (Node – 13)	**UX**	0.002281	0.002164	**5.407%**
	UY	-0.000005	-0.0000049	**2.041%**
	UZ	0.036906	0.035615	**3.625%**
Point – C (Node – 1)	**S11**	32942.582	34730.135	**-5.147%**
Point – D (Node – 12)	**S22**	227009.341	222166.846	**2.189%**
Point – E (Node – 21)	**S12**	214256.364	220455.581	**-2.812%**

7.6 Example Problems for Verification of Four Node Quadrilateral Shell Elements

This section presents the comparison of displacements and stresses obtained from the developed program with those obtained from SAP 2000. Three example problems were selected for the verification of four node quadrilateral shell element. A description of each example problem and the discussion of the results are presented in the following sections.

Test Example 16

The first example problem for the verification of the four node quadrilateral shell elements is a cantilever channel section with tip load applied to the free end. The length of the cantilever is 6 in., the height 3 in., and the flange width is 2 in. The thickness of the channel section is 1 in. A tip load of 30 kips is applied at the free end of the cantilever. The finite element model is generated using 56 four node quadrilateral shell elements. Fig. 7.20 shows the finite element discretization for the structure. One end of the cantilever is restrained in all six directions and the concentrated loads of 10 kips each (in the downward vertical direction) are applied to the three nodes at the free end.

Geometric Data:

Length = 6.0 in.

Flange width = 2.0 in.

Height of the section = 3.0 in.

Thickness t = 1.0 in.

Material Properties:

Modulus of elasticity E = 3600 ksi.

Poisson's ratio n = 0.2

Boundary Conditions:

Restraints in all six directions are provided at the left end of the cantilever.

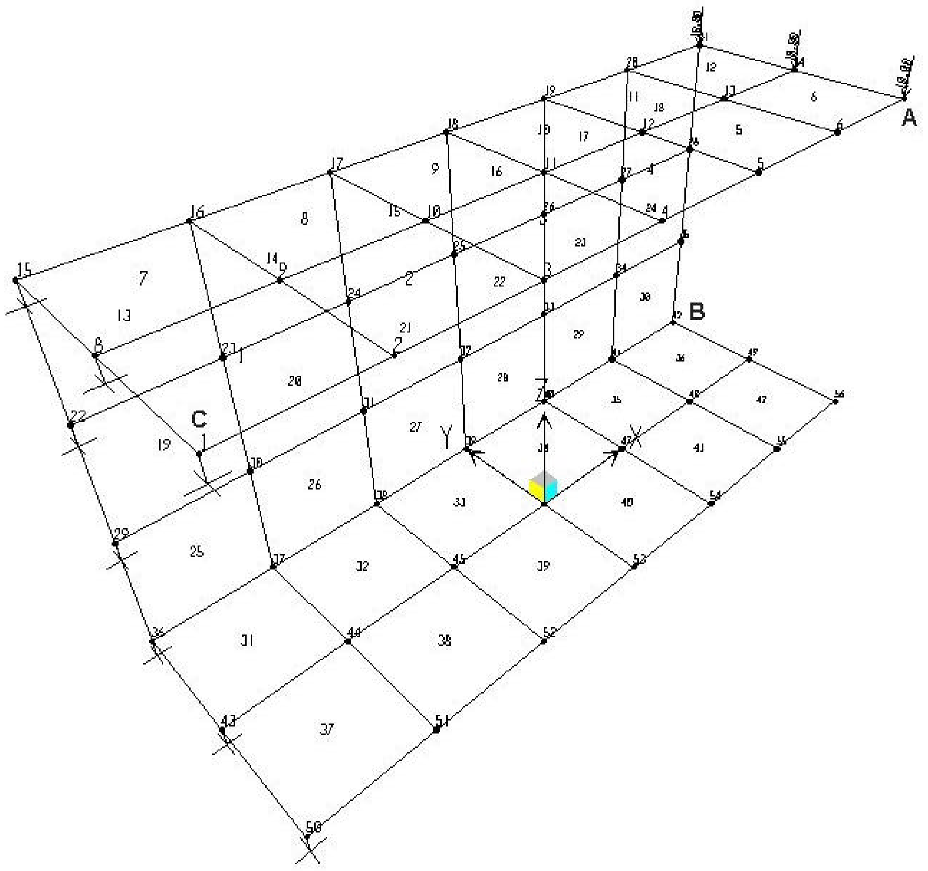


Fig. 7.20 FE Model for Test Example 16 – Cantilever Channel Section

Loading:

A concentrated loads of 10 kips each is applied to the nodes at the top of the free end of the cantilever (nodes 7, 14, and 21) as shown in Fig. 7.20.

Comparison of Results:

Table 7.17 shows the comparison of displacements and stresses for the verification example. The displacements at nodes 11 and 14 are compared with those obtained from SAP 2000. The difference in the displacements is less than 5%. This difference is because of the different element types are used in the author's program and SAP 2000. It is observed that the membrane action of the four node quadrilateral shell element in SAP 2000 is internally represented by eight node quadrilateral plane elements. Also the SAP 2000 shell element includes inplane rotational degrees of freedom in the membrane part of the shell element. In the developed program a four node quadrilateral plane element is used in combination with four node quadrilateral plate bending (DKQ) element. The four node quadrilateral plate element is initially formulated as an eight node quadrilateral plate element and then constraints are provided at the mid nodes. However, the shape functions of eight node quadrilateral element is used in the development of the four node quadrilateral plate bending element. When combining this four node quadrilateral plate bending element with four node quadrilateral plane element to develop the quadrilateral shell element, it was concluded that convergence may not be accurate at the edges of the element and hence the difference in the results were found different. Also, the four node quadrilateral shell element developed in this study does not include the inplane rotational degrees of freedom which is important while analyzing the structure in which pure membrane or bending action is not present. The stresses obtained at nodes 1 and 14 are compared to those obtained from SAP 2000 and the difference in the results are found less than 3 % as shown in Table 7.17.

Table 7.17 Displacements and Stresses for Test Example 16

Location		**Result from Program**	**Result from SAP 2000**	**% Difference**
Point – A (Node – 14)	**UX**	0.014098	0.014522	**-3.263%**
	UY	-0.074297	-0.077952	**-4.689%**
	UZ	-0.218509	-0.223337	**-2.162%**
Point – B (Node – 11)	**UX**	0.009755	0.010005	**-2.499%**
	UY	-0.024764	-0.025712	**-3.687%**
	UZ	-0.069415	-0.070682	**-1.793%**
Point – C (Node – 1)	**S11**	60.253877	59.538842	**-1.187%**
Point – C (Node – 1)	**S22**	11.907768	12.050775	**-1.187%**
Point – D (Node – 7)	**S12**	-26.076878	-26.862638	**-2.925%**

Test Example 17

The second example consists of the same cantilever I – beam used to verify the three node triangular shell elements. This example tests the behavior of the element when the inplane bending stresses are severe. A cantilever I – beam has a length of 40 in., height of 5 in. and flange width of 10 in. The thickness of the beam is 0.25 in. A torque is applied to the I – section by applying point loads of 1.6 kips each in two opposite directions at the top and bottom flanges of the beam (See Fig. 7.21). The finite element model of this example problem contains 48 four node quadrilateral shell elements and is shown in Fig. 7.21.

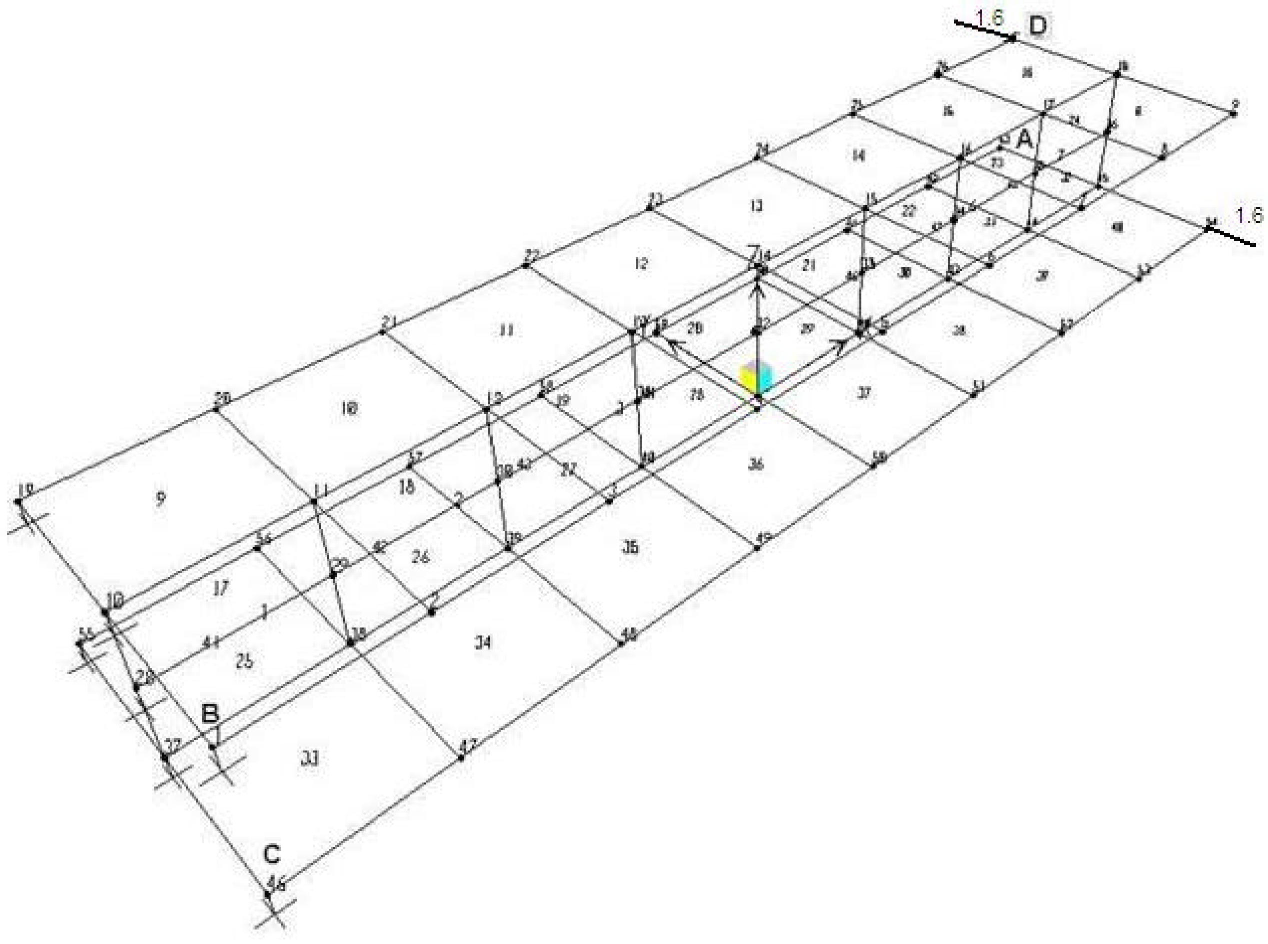


Fig. 7.21 FE Model for Example 17 - Cantilever I – Beam (Alladin v. 1.0, 1996).

Geometric Data:

Length L = 40.0 in.

Width = 10.0 in.

Height h = 5.0 in.

Thickness t = 0.25 in.

Material Properties:

Modulus of elasticity E = 10000 ksi.

Poisson's ratio n = 0.3

Boundary Conditions:

One end of the cantilever is fixed.

Loading:

A concentrated load of 1.6 kips is applied at the top and bottom of the flange in opposite directions as shown in Fig. 7.21.

Comparison of Results:

The displacements at nodes 9 and 63 and stresses at nodes 1, 46, and 27 are obtained from the developed program and SAP 2000 are shown in Table 7.18. It is seen from the Table 7.18 that the effect of inplane bending stresses in the four node quadrilateral shell element is more severe than in the triangular shell element. The displacements are compared at nodes 63 and node 9, which are the nodes opposite to the nodes where the loads are applied. When a torque is applied to a cantilever I – beam it was expected that the displacements in y and z should be in the opposite directions and of same value. The tabulated results agree with the expected results thus verifying the accuracy of the assembly of structure stiffness matrix and the equation solver used in this program. There is a large difference in the results obtained for stresses from the two programs for the same reasons indicated for the Test Example 16.

Table 7.18 Displacements and Stresses for Test Example 17

Location		**Result from Program**	**Result from SAP-2000**	**% Difference**
Point – A (Node – 63)	**UX**	-0.025120	-0.027162	**-7.518%**
	UY	0.139963	0.151049	**-7.339%**
	UZ	0.209469	0.255308	**-17.954%**
Point – E (Node – 9)	**UX**	-0.025120	-0.027162	**-7.518%**
	UY	-0.139963	-0.151049	**-7.339%**
	UZ	-0.209469	-0.255308	**-17.954%**
Point – B (Node – 1)	**S11**	-10.824349	-12.256818	**-11.687%**
Point – C (Node – 46)	**S22**	4.430982	4.683511	**-5.392%**
Point – D (Node – 27)	**S12**	-1.523552	-1.683754	**-9.515%**

Text Example 18

The structure chosen for the third verification example for quadrilateral shell elements is the same cantilever I – beam as in the previous example but now a concentrated tip load of 15 kips is applied at the free end of the cantilever. The length of the cantilever is 40 in., flange width is 10 in., height is 5 in. and thickness is 0.25 in. The cantilever I – beam has same number of elements as in Test Example 17 and is modeled using four node quadrilateral shell elements. The cantilever I – beam was chosen to verify the magnitude of the error in analysis due to inplane bending stresses present in the structure. The finite element model of the cantilever I – beam is shown in Fig. 7.22.

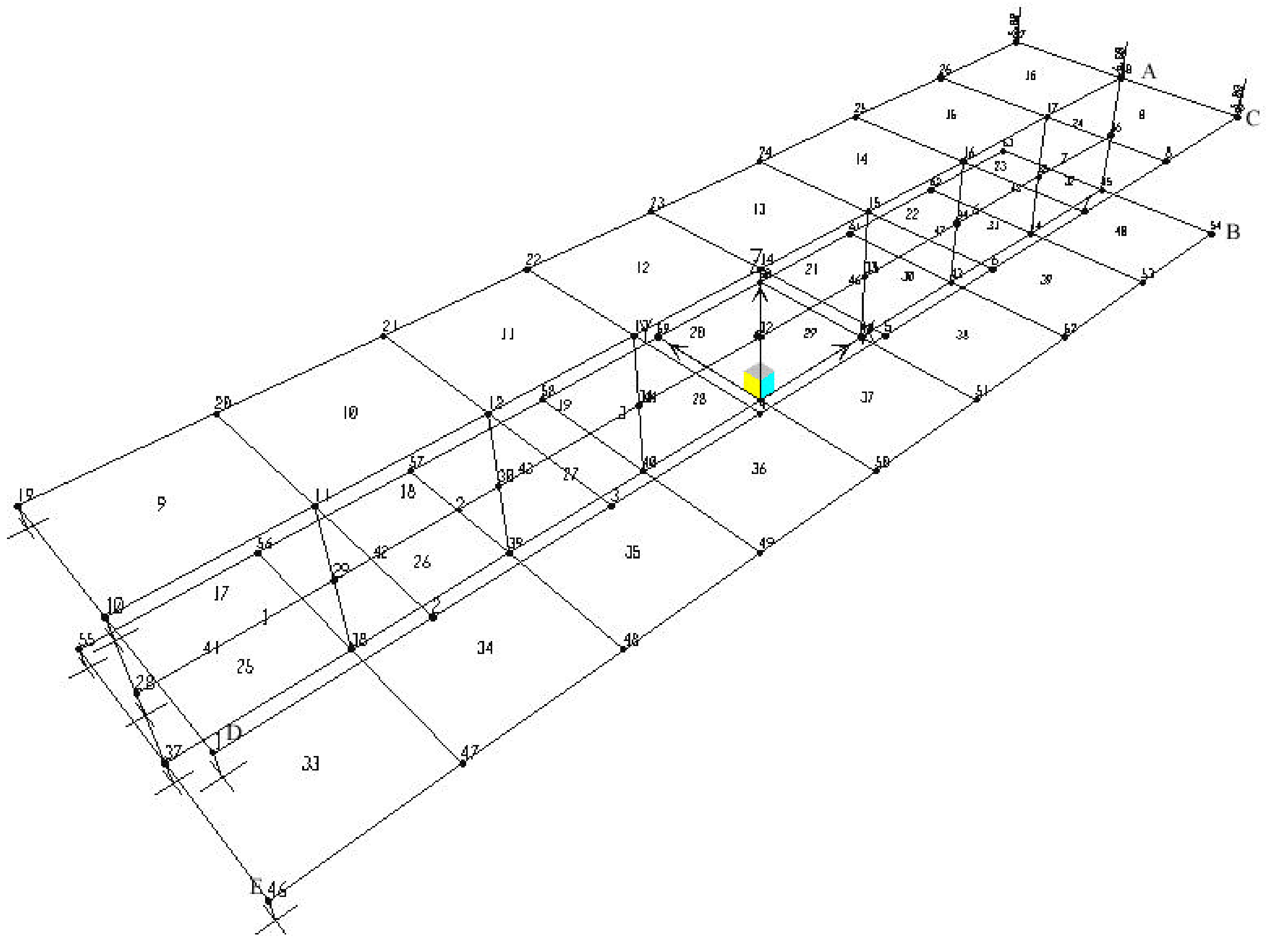


Fig. 7.22 FE Model for Test Example 18 – Tip Loaded Cantilever I – Beam.

Geometric Data:

Length L = 40.0 in.

Width = 10.0 in.

Height h = 5.0 in.

Thickness t = 0.25 in.

Material Properties:

Modulus of elasticity E = 10000 ksi.

Poisson’s ratio n = 0.3

Boundary Conditions:

Restraints in all six directions are provided at the left end of the cantilever.

Loading:

A concentrated load of 5 kips each is applied to nodes 9, 18 and 21.

Comparison of Results:

From Table 7.19 it can be seen that the difference in the displacements at nodes 18 and 54 is approximately 3 %. The maximum displacement is observed at node 9 where the difference between the results obtained from the developed program and SAP 2000 is found 0.6 %. Normal stresses at nodes 1 and 46 differ by 3 %. For shearing stress the difference is approximately 8 %. Thus it can be concluded that the four node quadrilateral shell element gives better results when inplane bending stresses are not significant.

Table 7.19 Displacements and Stresses for Test Example 18

Location		Result from Program	Result from SAP-2000	% Difference
Point – A (Node – 18)	**UX**	0.086634	0.089327	**-3.015%**
	UY	0.00	0.00	**0.00%**
	UZ	-1.051681	-1.084128	**-2.993%**
Point – B (Node – 54)	**UX**	-0.082022	-0.084834	**-3.315%**
	UZ	-1.045458	-1.082865	**-3.454%**
Point – C (Node – 9)	**UZ**	-4.658466	-4.630571	**0.602%**
Point – D (Node – 1)	**S11**	44.204776	42.776400	**-3.231%**
Point – E (Node – 1)	**S22**	13.261433	12.832920	**-3.231%**
Point – E (Node – 46)	**S12**	-4.119114	-4.437667	**7.734%**

Chapter 8
Summary and Conclusions

8.1 Summary

The purpose of this study was to develop membrane, plate and flat shell elements using an object oriented approach and the Java programming language. The membrane elements developed in the program included the three node triangular plane (CST) element and a four node quadrilateral plane element. The plate elements developed were based on the discrete Kirchoff theory. The two plate elements developed were the three node discrete Kirchoff triangular (DKT) element and the four node discrete Kirchoff quadrilateral (DKQ) element. The triangular flat shell element was developed by combining the CST element and the DKT Element (Batoz *et al.*, 1980). The quadrilateral flat shell element was developed by assembling the four node isoparametric quadrilateral element and the DKQ Element (Batoz and Tahar, 1982). A computer program for finite element analysis was also written in Java programming language to check and verify the accuracy of results obtained from the developed membrane, plate and flat shell elements. The program is based on the object oriented approach.

Input to the program is through a text file. The format for the input file is similar to the SAP 2000 S2k file. This makes it possible to generate the finite element model for the verification examples using the SAP 2000 graphical user interface. However, a few changes were made in the format of input text file for compatibility with the developed program. The loads that can be applied to the structure include concentrated loads, and uniformly distributed surface loads. Various load combinations can also be used. The program computes displacements and stresses at each node of the finite element model.

Several test examples were analyzed using the program and results were compared with those obtained from the commercial finite element analysis program SAP

2000. Results were compared at the points of maximum displacements and stresses. The average stress was taken in to consideration to calculate stresses at specific point.

8.2 Conclusions

This thesis presented the development of six finite elements using the object oriented programming concept in Java as an alternative to the traditional procedural programming approach. A finite element analysis program was developed to verify the accuracy of the results.

A series of test example problems were analyzed using the developed program. The results from these analyses were compared with those obtained from the commercial finite element analysis program SAP 2000 in order to verify the accuracy of the developed program.

The results obtained from the analysis of the example problems using the plane stress triangular elements and the plane stress quadrilateral elements were found to be very accurate when compared to those obtained from the SAP 2000. The difference in displacements computed by the two programs was less than 1 %. The difference in stresses was also quite close. However, stresses in a few cases differed by 6 to 7 %.

The displacements for the verification examples using the triangular plate bending (DKT) elements (Batoz *et al.*, 1980) were in agreement with those obtained from the SAP 2000. The difference in displacements was found to be less than 3 % for the three node triangular plate bending (DKT) element and less than 5 % for the four node quadrilateral plate bending (DKQ) element. The computed stresses were also in agreement for most cases. The margin of difference in stresses was about 10 %.

An important observation was made when analyzing the clamped plate using the DKT elements. It was founded that orientation of the mesh pattern affects the analysis results in clamped plate problem analyzed using DKT elements.

Three structures were analyzed using the three node triangular flat shell elements. The displacements found from these analyses were found to be in good agreement. However, there was a significant difference in the stress results. The reason for the difference is because the drilling degrees of freedom were neglected in the development of the element. It can be concluded that although the developed element gives reasonable results for displacements it does not accurately model the behavior of the shell structures when it comes to computing stresses. Thus, it is proposed that this element be modified by including drilling degrees of freedom in the membrane part of the element.

The results from the quadrilateral shell elements were satisfactory only when either membrane or bending action is present in the structure, but gave poor results when inplane bending stresses were also present in the structure. The main reason for poor results is again the negligence of inplane rotational stiffness in the membrane stiffness matrix. The stiffness parameter for the drilling degree of freedom was approximated in the quadrilateral shell element developed in this study, while SAP 2000 uses a quadrilateral shell element with inplane rotational degree of freedom and hence gives more accurate results. McNeal and Harder (1988) have arrived at a similar conclusion. According to them, the flat shell element gives more accurate results when inplane rotational degree of freedom is included. Knowles *et al.* (1976) also observed in his studies that, the flat shell elements are more accurate when the response of the structure is either membrane action or bending, but when membrane-bending coupling is present in the structure the flat shell element gives poor results. They also demonstrated the complete failure in performance of flat shell elements while analyzing the torsional behavior of a slit cylinder.

In conclusion, the plane stress and plate bending elements developed in this study were found to be accurate. The triangular shell element performed better than the quadrilateral shell elements. From an analysis of different structures using the program, it can be concluded that the assembly of the structure stiffness matrix and the equation solver used in the program are accurate.

8.3 Future Development

This study illustrates the use of the object oriented Java programming language for developing membrane, plate bending and flat shell elements. Suggestions for future work include: (1) developing flat shell elements in which the membrane elements have rotational degrees of freedom, (2) using a band storage scheme for storing the structure stiffness matrix and band solvers in the program to solve large finite element analysis problems, (3) modifying the program to include a graphical user interface and, (4) extending the program to include other elements such as truss, and frame elements.

References:

Bathe K. J., *Finite Element Procedures*, Prentice – Hall, Englewood Cliffs, New Jersey, 1996.

Bathe, K. J., and Ho, L. W., A Simple and Effective Element for Analysis of General Shell Structures, *Computers and Structures*, Vol. 13, pp. 673-681, 1981.

Bathe K. J., Wilson E. L., *Numerical Methods in Finite Element Analysis*, Prentice-Hall Inc., 1976.

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 Dhatt G., Development of Two Simple Shell Elements, *AIAAJ*, Vol. 10, No. 2, 1972, pp. 237-238.

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).

Bazeley, G. P., Cheung Y. K., Irons B. M. and Zienkiewicz, O. C., Triangular Elements in Plate Bending, Confirming and Non – Confirming Solutions, *Proc. 1st Conference on Matrix Methods in Structural Mechanics*, pp. 547-576, Wright Patterson AF Base, Ohio, 1966.

Chen H. C. Evaluation of Allaman Triangular Membrane Element used in General Shell Analysis, *Computers and Structures*, Vol. 43, No. 5, pp. 881-887, 1992.

Chen H. C., Evaluation of Allman Triangular Membrane Element used in general Shell Analysis, *Computers and Structures*, Vol. 43 (5), 1992, pp. 881-887.

Clough R. W. and Tocher J. L., Finite Element Stiffness Matrices for Analysis of Plate Bending, *Proc. Conference on Matrix Methods in Structural Mechanic*, WPAFB, Ohio, 1965, pp. 515-545.

Clough R. W., The Finite Element Method in Plane Stress Analysis. *Proceedings American Society of Civil Engineers*, 2nd Conference on Electronic Computations, Pittsbutgh, Pennsylvania, 23, 1960, pp. 345-378

C-MOLD Shrinkage & Warpage User's Guide, Copyright © 1995, 1997 Advanced CAE Technology, Inc.

Cook R. D., *Concepts and Applications of Finite Element Analysis*, John Wiley & Sons, 1974.

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

Driver J. J., A Methodology of Analysis of Structures Using an Object-Oriented Representation of the Structural Model, M.S. Thesis, Virginia Polytechnic Institute and State University, 1994.

Ergatoudis I., Iron B. M., and Zeinkiewicz O. C., Curved Isoparametric ‘Quadrilateral’ elements for Finite Element Analysis, *International Journal of Solids and Structures*, Vol. 4, No.1, 1968, pp. 31-42.

Gallagher R. H., *Finite Element Analysis Fundamentals*, Prentice-Hall, 1975.

Green B. E., Strome D. R., and Weikel R. C., Application of the stiffness method to the analysis of shell structures, *Procedures on Aviation Conference, American Society of Mechanical Engineers*, Los Angeles, March 1961.

Holzer S. M., *Computer Analysis of Structures - Matrix Structural Analysis Structured Programming*, Elsevier Science Publishing Co., Inc., 1985.

Hrabok M. M., and Hrudey T. H., A Review and Catalogue of Plate Bending Finite Elements, *Computers and Structures*, Vol. 19, No. 3, 1984, pp. 475-495.

Irons B. M., Engineering Application of Numerical Integration in Stiffness Methods, *AIAAJ*, Vol. 4, No. 11, 1966, pp. 2035-2037.

Knight Jr. N. F., The Raasch Challenge for Shell Elements, *AIAAJ*, Vol. 35, 1997, pp.375-388.

Mark A. and Blattau M., Alladin v. 1.0 - A complete Toolkit For Engineering Matrix and Finite Element Analysis, 1996

McNeal R. H., A Simple Quadrilateral Shell Element, *Computers and Structures*, Vol. 8, 1978, pp. 175-183.

McNeal R. H. and Harder R. L., Refined Four Node Membrane Element with Rotational Degrees of Freedom, *Computers and Structures*, Vol. 28, 1988, pp. 75-84.

Robinson J. and Haggenmacher G., Lora – an accurate four nodes stress plate bending element, *International Journal of Numerical Methods in Engineering*, Vo. 14 (2), 1979, pp. 296-306.

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., *The complete reference – JAVA 2*, 4th ed., Tata McGraw-Hill Publishing Company Limited, 2001.

Timoshenko S. and Woinowsky-Krieger S., *Theory of Plates and Shells*, 2nd ed., 1959.

Ugural A. C., *Stresses in Plates and Shells*, 2nd ed. 1999.

Weaver W. Jr. and Johnston P. R., *Finite Elements for Structural Analysis*, Prentice-Hall, 1984.

Yang H. T., Saigal S., Masud A., Kapania R. A Survey of Recent Shell Finite Elements, *International Journal of Numerical Methods in Engineering*, 2000, pp. 101-127.

Yang H. T. Y., Saigal S., and Liaw, D. G., Advances of Thin Shell Finite Elements and some applications – version – I, *Computers and Structures*, 35, pp. 481-504, 1990.

Zhang Y. X., Cheung Y. K., and Chen W. J. Two refined non-conforming quadrilateral at shell elements, *International Journal of Numerical Methods in Engineering*, Vol. 49, pp. 355-382.

Zienkiewicz O. C., *The Finite Element Method in Engineering Science*, 2nd ed., McGraw-Hill, 1971.

Appendix – A

Input File Format for Program

The input for the program can be provided in the form of a text file. The format of the text file essentially follows that used in the SAP 2000 program. However, several modifications were made to this format.

All input data is provided in inches and kips. The data to be provided is in the form of data blocks. Each data block is separated by a specific title, which defines and separates the data blocks. The data blocks must be in the same order as shown in Table A-1.

Table A-1 List of data blocks in input file

Title of the data block	Function
SYSTEM	Defines system properties.
JOINT	Defines joint coordinates.
RESTRAINT	Defines the restraints provided to nodes.
MATERIAL	Defines material properties.
SHELL SECTION	Defines shell section properties.
SHELL	Defines the joint connectivity for each element.
LOAD	Defines applied loads applied and load cases.
END	Ends the input file.

SYSTEM Data Block:

SYSTEM data block provides information regarding the degrees of freedom for the whole structure. The restraint conditions for the structure can be derived from this data block.

Table A- 2 SYSTEM data block

SYSTEM	
DOF = UX, UY, UZ, RX, RY, RZ	Shell Element
DOF = UX, RY, RZ or UY, RX, RZ or UZ, RX, RY	Plate Element
DOF = UX, UY or UY, UZ or UX, UZ	Plane Element

JOINT Data Block:

The JOINT data block contains data for joint coordinates. For each joint, X, Y and Z coordinates must be provided.

Table A- 3 JOINT data block

JOINT			
1	X=0.5	Y=0.5	Z=0
Node number	X-coordinate	Y-coordinate	Z-coordinate

RESTAINT Data Block:

The RESTRAINT data block provides data for the restraints provided at different nodes of the structure. The notations U1, U2, U3 and R1, R2, R3 represents translational and rotational restraints in THE global X, Y, and Z directions respectively.

Table A-4 RESTRAINT data block

RESTRAINT	
ADD=1	DOF=U1, U2, U3, R1, R2, R3
Adds restraint to joint number 1	Provides restraints in the specified direction for joint 1.

MATERIAL Data Block:

The MATERIAL data block defines material properties for the elements such as modulus of elasticity and Poisson’s ratio.

Table A-5 MATERIAL data block

MATERIAL	
E=29000	U=0.3
Modulus of elasticity	Poisson’s ratio

SHELL SECTION Data Block:

The SHELL SECTION data block provides information regarding section properties such as the thickness of the plane, plate or shell element. This data block also provides information on the type of stress condition (plane stress or plane strain).

Table A-6 SHELL SECTION data block

SHELL SECTION	
TH=1.0	TYPE=STRESS or STRAIN
Thickness of the element.	Stress conditions (plane stress or plane strain)

SHELL Data Block:

SHELL data block provides information regarding joint connectivity for each element.

Table A-7 SHELL data block

SHELL	
1	J=1,2,3,4
Element no.	Defines nodes **i, j, k,** and **l** in counterclockwise direction (Quadrilateral element)
1	J=1,2,3
Element no.	Defines nodes **i, j,** and **k** in counterclockwise direction (Triangular element)

LOAD Data Block:

LOAD data block defines load values, load cases and load types. Two types of loads are considered in this study. One is a concentrated load that can be applied at any joint and the other is a uniformly distributed surface load that can be applied to elements in the element local Z – direction. A multiplication factor for any load case can also be defined in this data block.

Table A-8 LOAD data block

LOAD	
NAME = DEAD LOAD	MULT = 1
Defines the load case.	Defines the multiplication factor.
TYPE = FORCE	
Defines the load type for this load case.	
ADD = 1	UX = 5, UY = 2, UZ = 4, RX = 4, RY = 1, RZ = 2
Adds joint load to specified node.	Defines the value of load in specified direction.
TYPE = UDL	
Defines uniformly distributed load.	
ADD=1	P = -1
Applies surface load to specified element.	Defines the value of the applied load.

END Data Block:

The END data block must be placed at the end of the input text file to indicate the end of all input data.

Vita

Kaushalkumar M. Kansara was born in Palanpur, Gujarat, India March, 17, 1979. He completed his high school at Shri Vividh Laxi Vidya Mandir in 1996. He graduated with Bachelor of Engineering in Civil Engineering with First Class Distinction from Lukhdheerji Engineering College, Morvi, India. He joined Virginia Tech for pursuing Masters of Science in Civil Engineering in August 2002. He is working as a structural engineer at Alliance Structural Engineers in Vienna, Virginia.