MARMARA UNIVERSITY
FACULTY OF ENGINEERING

DEVELOPMENT OF A COMPUTER PROGRAM FOR THE FINITE ELEMENT ANALYSIS AND DESIGN OF PLANAR TRUSS STRUCTURES

MEHMET GÖRKEM SEVEN

GRADUATION PROJECT REPORT
Department of Mechanical Engineering

Supervisor
Prof. Dr. Mustafa ÖZDEMİR

ISTANBUL, 2024

MARMARA UNIVERSITY
FACULTY OF ENGINEERING

DEVELOPMENT OF A COMPUTER PROGRAM FOR THE FINITE ELEMENT ANALYSIS AND DESIGN OF PLANAR TRUSS STRUCTURES

MEHMET GÖRKEM SEVEN (150420052)

GRADUATION PROJECT REPORT
Department of Mechanical Engineering

Supervisor
Prof. Dr. Mustafa ÖZDEMİR

ISTANBUL, 2024

MARMARA UNIVERSITY
FACULTY OF ENGINEERING

Development of a computer program for the finite element analysis and design of planar truss structures

by

Mehmet Görkem Seven

May 28, 2024, Istanbul

SUBMITTED TO THE DEPARTMENT OF MECHANICAL ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE

OF

BACHELOR OF SCIENCE

AT

MARMARA UNIVERSITY

The author(s) hereby grant(s) to Marmara University permission to reproduce and to distribute publicly paper and electronic copies of this document in whole or in part and declare that the prepared document does not in anyway include copying of previous work on the subject or the use of ideas, concepts, words, or structures regarding the subject without appropriate acknowledgement of the source material.

Signature of Author(s) .....Mehmet Görkem SEVEN

Department of Mechanical Engineering

Certified By .....Prof. Dr. Mustafa ÖZDEMİR

Project Supervisor, Department of Mechanical Engineering

Accepted By.....Prof. Dr. Bülent EKİCİ

Head of the Department of Mechanical Engineering

ACKNOWLEDGEMENT

First of all, I would like to thank my supervisor Prof. Dr. Mustafa ÖZDEMİR, for the valuable guidance and advice on preparing this thesis and giving me moral and material support.

May, 2024 Mehmet Görkem SEVEN

CONTENTS

LIST OF FIGURES

ABSTRACT

Development of a Computer Program for The Finite Element Analysis and Design of Planar Truss Structures

This study aims to develop a computer program that performs finite element analysis and design of planar truss structures, which are widely used in the field of engineering. Truss structures are very common structural elements in the field of engineering and in our lives. We can give examples such as bridges, cranes, roof systems to the truss structures we see in daily life. This project aims to design a program that will help us analyze and design these truss structures more effectively.

ÖZET

Sonlu Eleman Analizi ve Düzlemsel Kafes Yapıların Tasarımı için Bir Bilgisayar Programının Geliştirilmesi

Bu çalışma, düzlemsel kafes yapıların mühendislik alanında çokça kullanılan sonlu elemanlar analizini ve bu yapıların tasarımını gerçekleştiren bir bilgisayar programı geliştirmeyi amaçlamaktadır. Kafes yapıları mühendislik alanında ve hayatımızda oldukça fazla karşımıza çıkan yapı elemanlarıdır. Günlük hayatta gördüğümüz kafes yapılara köprüler, vinçler, çatı sistemleri gibi örnekler verebiliriz. Bu proje de bu kafes yapılarını daha etkili bir şekilde analiz etmemize ve tasarlamamıza yardımcı olacak bir program tasarlamayı hedeflemektedir.

SYMBOLS

f: Axial load

A: Cross-sectional area

$u(x)$: Displacement function

u: Element displacement

$[k_e]$: Element stiffness matrix

$[K]$: Global stiffness matrix

$N(x)$: Inpolation function

L: Length

P: Loading

E: Modulus of elasticity

J: Polar moment of inertia

I: Principal moment of inertia

ρ: Radius of deflected curve

G: Shear modulus

k: Spring stiffness

δ: Spring deformation

ε_x: Strain in axial direction

σ_x: Stress in axial direction

T: Torque

$[R]$: Transformation matrix

$\{U\}$: Vector of nodal displacement

$\{F\}$: Vector of applied nodal force

1. INTRODUCTION

Although the name of the finite element method has been given in recent years, the use of this method goes back centuries. For example, centuries ago, an attempt was made to obtain an approximate solution by using the perimeter of a polygon to find the perimeter of a circle [1]. This method, which dates to the time of Archimedes, was used by Archimedes as follows. Archimedes first divided the area of an unusual shape he wanted to find into small geometric pieces. Then, the sum of all the areas formed by the small parts gives us the total area of the shape. We cannot say that the finite element method emerged from single research. As can be seen, the finite element method emerged because of much research and was formulated and started to be used [2].

The main purpose of the finite element method is to solve a complex problem by replacing it with a simpler problem. Since the problem you are solving is replaced with a simpler problem, we will get an approximate solution rather than an exact solution. In the process of finding a solution, we will have the opportunity to find an approximate solution instead of an exact solution, as the problem is replaced with a simpler one. Known mathematical methods will not be sufficient to find the exact solution to the problem. For this reason, the finite element method is one of the first methods that come to mind that can be applied to a problem for which there are no suitable methods [1].

The finite element method is a method used to obtain an approximate solution to problems involving elliptical partial differential equations with boundary conditions. This method divides the domain considered in the problem into a set of finite elements. For example, if we analyze a structure with the finite element method, we first divide the entire domain into a series of finite small areas and apply loads to these divided domains. Then, we find the unknowns according to the analyzes we make, and these analyzes are made with an utility analysis program. These utility analysis programs make the analysis of any engineering problem very fast. Thanks to this, the infinite element method has been highly adopted by engineers because it reduces the number of experiments of physical samples in the form of procedures prepared by engineers and modifies the components in the sample more quickly. Examples of such software are NASTRAN, ANSYS, FEAST and can be written in any computer language such as C and C++. Initially, the finite element method was used to solve mathematical

formulations faster and easier, but with the development of finite element analysis and the development of software programs, finite element method applications have started to be used in statistical structure analysis, heat transfer problems, as well as in many fields such as bioengineering, nuclear engineering, metallurgy [2].

The formulations in the finite element method to be applied for a physical problem include the same steps for all problem types. We can give examples of these problem types such as structural, heat transfer or fluid flow. We can explain these steps as follows.

1. Preprocessing

Preprocessing is defined as defining the model we are dealing with in the problem. It includes defining the domain of the problem to be analyzed, defining the type or types of elements to be used, defining the material types of the elements, defining the geometric properties of these elements such as length and area, defining boundary conditions and loads.

The preprocessing step is very important for the problem. If there is a problem that has been defined incorrectly and solved perfectly, it means nothing to us anymore because the problem has been solved incorrectly [3].

2. Solution

In the solution phase, the finite element analysis program writes the governing algebraic equations in matrix form and calculates the unknown variables. The calculated values are used to calculate derived variables such as stress and reaction stresses [3].

3. Postprocessing

The evaluation of the solutions found in the previous step, the solution step, is done in the postprocessing step. The software used in this step performs complex operations such as sorting, printing, or plotting the results [3].

2. INTRODUCTION TO FINITE ELEMENT ANALYSIS

2.1. The Linear Spring Element

Spring element is a one-dimensional finite element that supports only uniaxial load. When we apply the load to the spring element, there is a shortening or elongation in the direction of the applied load on the spring. The applied load can cause a deformation in the spring element. The spring element wants to show a resistance against the deformation. This resistance is called spring stiffness and is denoted by k.

Let us try to obtain the finite element formulation for the spring element in Figure 1. Let us call the axis along the length of the spring the x-axis. This axis is also our element or local coordinate system and is embedded in each element. Element coordinate system makes it easy to analyse the properties of each element. Element or local coordinate system is related to global coordinate system. The global coordinate system is the coordinate system in which the behaviour of the whole structure will be defined, not a single element.

Each spring element has two nodes. Let's call these nodes 1 and 2 nodes from left to right respectively. The forces applied to the nodes can be called f_1 and f_2. Due to the applied loads, a displacement occurs at the nodes, and these are called u_1 and u_2, as shown in Figure 1.

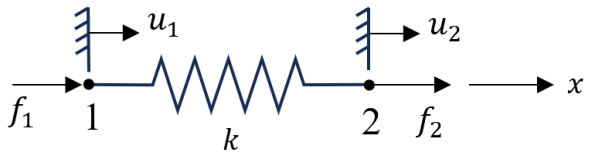


Figure 1: Nodes, displacement, forces shown on a linear spring element.

Assuming that the displacement at both nodes is zero, the net spring deformation can be written as follows.

$$\delta = u_2 - u_1 \tag{1}$$

We know that the net axial force is written as follows,

$$f = k\delta = k(u_2 - u_1) \tag{2}$$

For equilibrium we can write the following equation between f_1 and f_2.

$$f_1 + f_2 = 0 \tag{3}$$

$$f_1 = -f_2 \tag{4}$$

Also f_1 and f_2 can be written as follows.

$$f_1 = -k(u_2 - u_1) \tag{5}$$

$$f_2 = k(u_2 - u_1) \tag{6}$$

If we express the nodal forces in matrix form, it can be written as follows.

$$\begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} \tag{7}$$

or

$$[k_e]\{u\} = \{f\} \tag{8}$$

where

$$[k_e] = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \tag{9}$$

$$\{u\} = \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} \tag{10}$$

$$\{f\} = \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} \tag{11}$$

The k_e matrix we created here is the element stiffness matrix in the element coordinate system. u matrix is the displacement matrix of the nodal points. The f matrix is a matrix formed by the forces at the nodal points.

In the example in Figure 1, we have derived the necessary formulations in the element coordinate system for the calculations. Now let us consider the example in Figure 2 to

obtain the complete behavior of the structure in the global coordinate system.

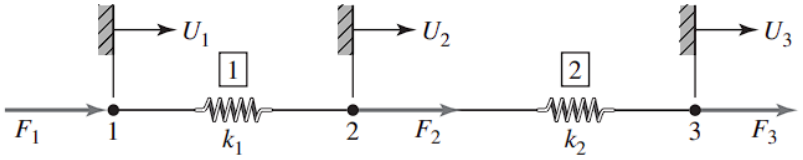


Figure 2: Two spring systems [3].

We find the stiffness matrix of the spring element by realizing the equilibrium conditions. The same procedure can be used for systems with more than one spring. Writing the equilibrium equations for each node separately will be a better method for us to reach our result.

In Figure 2, we see two springs connected to each other. Assume that the spring constants of these two springs are different and let them be k_1 and k_2 respectively. U_1, U_2 and U_3 shown in Figure 2 are the global node displacements. The forces applied to the nodes are F_1, F_2 and F_3.

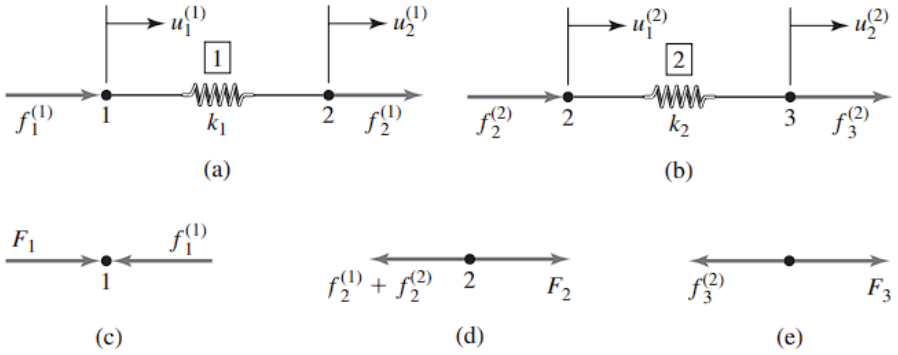


Figure 3: Free-body diagrams of elements and nodes for the system shown in Figure 2 [3].

We have assumed that the two spring systems are in equilibrium. Accordingly, we can show the free-body diagrams of each nodal point as shown in Figure 3. We can write the following equations for each nodal point for which we know the free body diagrams.

$$f_1^{(1)} = -k_1\left(u_2^{(1)} - u_1^{(1)}\right) \tag{12}$$

$$f_2^{(1)} = k_1\left(u_2^{(1)} - u_1^{(1)}\right) \tag{13}$$

and

$$f_2^{(2)} = -k_2\left(u_2^{(2)} - u_1^{(2)}\right) \tag{14}$$

$$f_3^{(2)} = k_2\left(u_2^{(2)} - u_1^{(2)}\right) \tag{15}$$

If we express it in matrix form, it can be written as follows.

$$\begin{bmatrix} k_1 & -k_1 \\ -k_1 & k_1 \end{bmatrix} \begin{Bmatrix} u_1^{(1)} \\ u_2^{(1)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(1)} \\ f_2^{(1)} \end{Bmatrix} \tag{16}$$

$$\begin{bmatrix} k_2 & -k_2 \\ -k_2 & k_2 \end{bmatrix} \begin{Bmatrix} u_1^{(2)} \\ u_2^{(2)} \end{Bmatrix} = \begin{Bmatrix} f_2^{(2)} \\ f_3^{(2)} \end{Bmatrix} \tag{17}$$

The superscripts shown here with 1 and 2 are the element number.

The relationship between element displacements and global displacements can be given as follows.

$$u_1^{(1)} = U_1 \tag{18}$$

$$u_2^{(1)} = U_2 \tag{19}$$

$$u_1^{(2)} = U_2 \tag{20}$$

$$u_2^{(2)} = U_3 \tag{21}$$

If we substitute these equations in Equation 16 and Equation 17, we obtain the following equations.

$$\begin{bmatrix} k_1 & -k_1 \\ -k_1 & k_1 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \end{Bmatrix} = \begin{Bmatrix} f_1^{(1)} \\ f_2^{(1)} \end{Bmatrix} \tag{22}$$

and

$$\begin{bmatrix} k_1 & -k_1 \\ -k_1 & k_1 \end{bmatrix} \begin{Bmatrix} U_2 \\ U_3 \end{Bmatrix} = \begin{Bmatrix} f_2^{(2)} \\ f_3^{(2)} \end{Bmatrix} \tag{23}$$

we use the notation $f_i^{(j)}$ to represent the force exerted on element j at node i.

It is not possible to combine the equations in this form. For this we expand the matrices to $3x3$ and thus obtain the following equations.

$$\begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ 0 \end{Bmatrix} = \begin{Bmatrix} f_1^{(1)} \\ f_2^{(1)} \\ 0 \end{Bmatrix} \tag{24}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \begin{Bmatrix} 0 \\ U_2 \\ U_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ f_2^{(2)} \\ f_3^{(2)} \end{Bmatrix} \tag{25}$$

If we sum these two matrices,

$$\begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \end{Bmatrix} = \begin{Bmatrix} f_1^{(1)} \\ f_2^{(1)} + f_2^{(2)} \\ f_3^{(2)} \end{Bmatrix} \tag{26}$$

We can write F_1, F_2 and F_3 values as follows.

$$F_1 = f_1^{(1)} \tag{27}$$

$$F_2 = f_2^{(1)} + f_2^{(2)} \tag{28}$$

$$F_3 = f_3^{(2)} \tag{29}$$

Substituting into Equation 26, we obtain the result:

$$\begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ F_2 \\ F_3 \end{Bmatrix} \tag{30}$$

The global stiffness matrix is shown as follows.

$$[K] = \begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \tag{31}$$

2.2. The Bar Element

Consider an elastic bar of length L as shown in Figure 4. Let the axial displacements of the rod at nodes 1 and 2 be u_1 and u_2 respectively. From u_1 and u_2 we have a new variable called $u(x)$, the continuous field variable. The discretization is performed with $N_1(x)$ and $N_2(x)$, known as the interpolation function. The displacement of the bar at any position can be given by

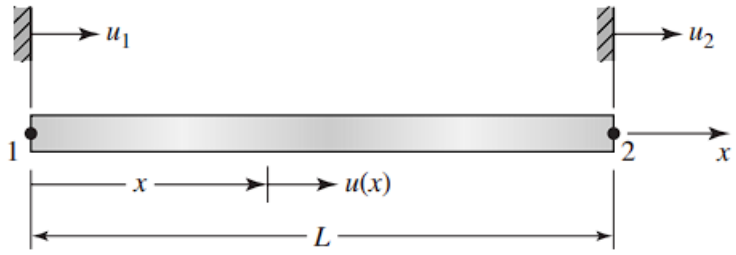


Figure 4: A bar element [3].

$$u(x) = N_1(x)u_1 + N_2(x)u_2 \tag{32}$$

u_1 and u_2 can be written as follows with reference to Figure 4.

$$u_1 = u(x = 0) \tag{33}$$

$$u_2 = u(x = L) \tag{34}$$

The boundary conditions that will occur are as follows.

$$N_1(0) = 1 \qquad N_2(0) = 0 \tag{35}$$

$$N_1(L) = 0 \qquad N_2(L) = 1 \tag{36}$$

The interpolation functions must satisfy the boundary conditions. The displacement function in Equation 32 must satisfy the end conditions because since these nodes are

the connection points of the bar element, the displacement continuity conditions will be applied to these connections. Let us obtain the interpolation functions.

$$N_1(x) = a_0 + a_1 x \tag{37}$$

$$N_2(x) = b_0 + b_1 x \tag{38}$$

To find the coefficients of the interpolation functions, we apply the boundary conditions in Equation 35 and Equation 36. The coefficients we obtain are as follows.

$$a_0 = 1 \tag{39}$$

$$b_0 = 0 \tag{40}$$

$$a_1 = -\frac{1}{L} \tag{41}$$

$$b_1 = \frac{1}{L} \tag{42}$$

If the found coefficients are substituted in Equation 37 and Equation 38, the interpolation functions can be written as follows.

$$N_1(x) = 1 - \frac{x}{L} \tag{43}$$

$$N_2(x) = \frac{x}{L} \tag{44}$$

If we substitute the interpolation functions in Equation 32, the displacement function is as follows.

$$u(x) = \left(1 - \frac{x}{L}\right) u_1 + \left(\frac{x}{L}\right) u_2 \tag{45}$$

If we write it in matrix form, it can be written as follows.

$$u(x) = [N_1(x) \quad N_2(x)] \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = [N]\{u\} \tag{46}$$

where $[N]$ is the row matrix of interpolation functions and $\{u\}$ is the column matrix

(vector) of nodal displacements.

For a bar subjected to a load P, we know that the deflection is as follows.

$$\delta = \frac{PL}{AE} \tag{47}$$

where L is the bar length, A is the uniform cross-sectional area of bar and E is the modulus of elasticity of the material.

We can give the elastic spring constant for a bar element as follows.

$$k = \frac{P}{\delta} = \frac{AE}{L} \tag{48}$$

Normal strain for an element containing uniaxial load such as a bar element is given as follows.

$$\varepsilon_x = \frac{du}{dx} \tag{49}$$

If we substitute Equation 45 in Equation 49, we obtain the following equation.

$$\varepsilon_x = \frac{u_2 - u_1}{L} \tag{50}$$

If we use Hooke's law, we get the following equation.

$$\sigma_x = E\varepsilon_x = E\frac{u_2 - u_1}{L} \tag{51}$$

Normal stress is also given as follows,

$$\sigma_x = \frac{P}{A} \tag{52}$$

where P is the axial force applied to the bar and A is the cross-sectional area of the bar. Then, axial force is,

$$P = \sigma_x A = \frac{AE}{L}(u_2 - u_1) \tag{53}$$

If we use Equation 53 to write the node forces at nodes 1 and 2, the following equations are obtained. Where f_1 and f_2 are the forces at node 1 and node 2 respectively. If we take f_2 to be positive, f_1 and f_2 can be written as follows for equilibrium.

$$f_1 = -\frac{AE}{L}(u_2 - u_1) \tag{54}$$

$$f_2 = \frac{AE}{L}(u_2 - u_1) \tag{55}$$

If we express it in matrix form, we can write it as follows.

$$\frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} \tag{56}$$

The element stiffness matrix of a bar element in the element coordinate system is,

$$[k_e] = \frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{57}$$

2.3. The Beam Element

To analyze the beam element, let us consider a beam element as shown in Figure 5 The length of the beam is L. To examine this case, let's call the axis parallel to the beam $\hat{x}$ and the axis perpendicular to the beam $\hat{y}$. As can be seen from Figure 5, local transverse nodal displacements are given by $\widehat{d_{\iota y}}$ and rotations by $\widehat{\phi_\iota}$. At the same time, local nodal forces are denoted by $\widehat{f_{\iota y}}$ and bending moments by $\widehat{m_\iota}$. We initially neglect all axial effects.

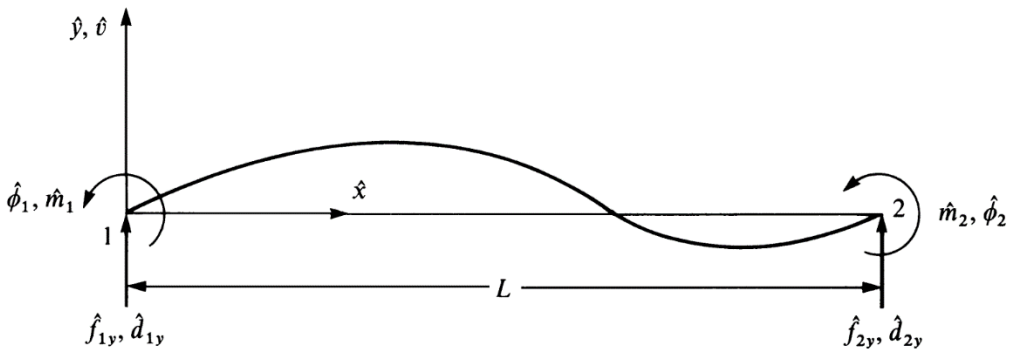


Figure 5: The beam element [4].

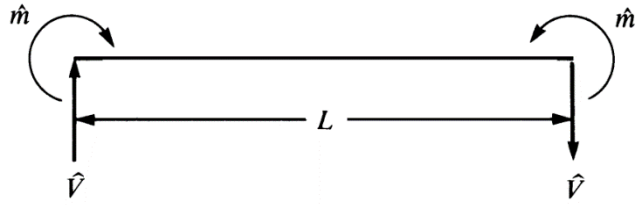


Figure 6: Beam theory sign conventions for shear forces and bending moments [4].

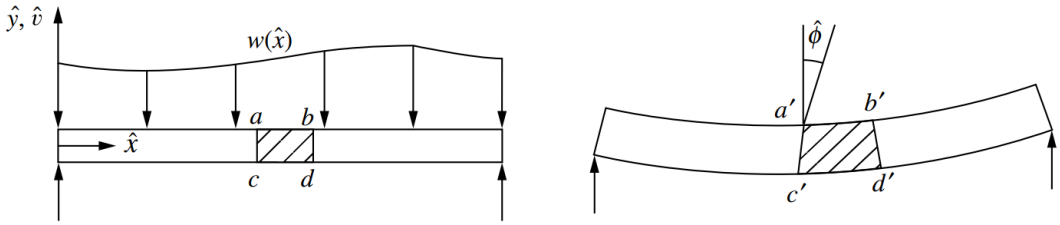
As shown in Figure 7a, if a distributed load represented by $w(\hat{x})$ is applied to our beam element, we can obtain the following differential equations according to the equilibrium principles of the beam element. If we want to write the force and moment equilibrium equation for the differential beam element shown in Figure 7c, we can write it as follows:

$$\sum F_y = 0: V - (V + dV) - w(\hat{x})dx = 0 \tag{58}$$

Simplifying,

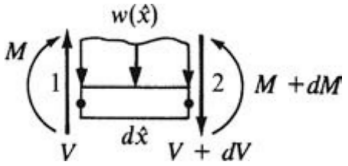
$$-wd\hat{x} - dV = 0 \ \ or \ \ w = -\frac{dV}{d\hat{x}} \tag{59}$$

$$\sum M_2 = 0: \ -Vdx + dM + w(\hat{x})d\hat{x}\left(\frac{d\hat{x}}{2}\right) = 0 \ \ or \ \ V = \frac{dM}{d\hat{x}} \tag{60}$$



(a) Undeformed beam under load $w(\hat{x})$

(b) Deformed beam due to applied loading



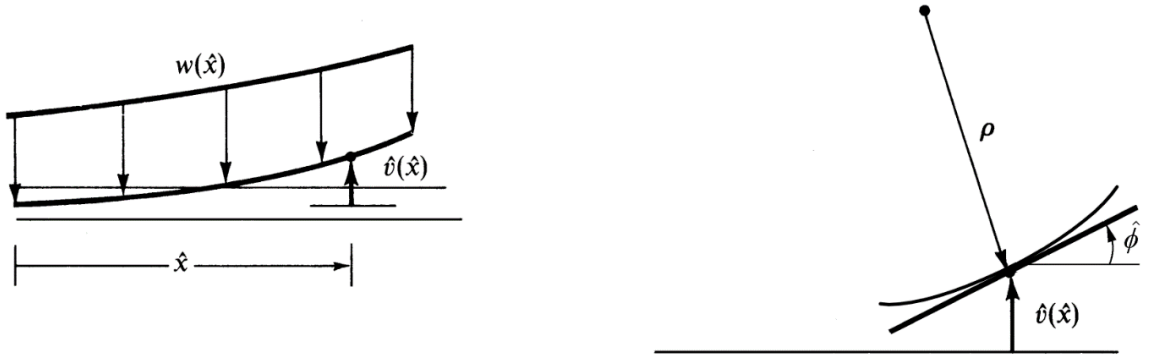
(c) Differential beam element

Figure 7: Beam under distributed load [4].

It is known that the curvature κ of the beam is related to the moment. Accordingly, we can give the equation as follows:

$$\kappa = \frac{1}{\rho} = \frac{M}{EI} \tag{61}$$

where ρ is the radius of deflected curve shown in Figure 8b, $\hat{v}$ is the transverse displacement function in the $\hat{y}$ direction E is the modulus of elasticity, and I is the principal moment of inertia about the axis $\hat{z}$.



(a) Portion of deflected curve of beam (b) Radius of deflected curve at $\hat{v}(\hat{x})$

Figure 8: Deflected curve of beam [4].

The curvature for small slopes $\hat{\phi} = d\hat{v}/d\hat{x}$ is given by

$$\kappa = \frac{d^2\hat{v}}{d\hat{x}^2} = \frac{M}{EI} \tag{62}$$

If we solve the Equation 62 for M, we get the following equation.

$$\frac{d^2}{d\hat{x}^2}\left(EI\frac{d^2\hat{v}}{d\hat{x}^2}\right) = -w(\hat{x}) \tag{63}$$

We can rewrite Equation 63 for a constant EI and only nodal forces and moments as follows:

$$EI\frac{d^4\hat{v}}{d\hat{x}^4} = 0 \tag{64}$$

As the first of the steps we need to do to obtain the stiffness matrix of the beam element I give in Figure 5, we can determine the nodes of the beam element and number these nodes.

Then, assume that there is a transverse displacement change along the beam element.

$$\hat{v}(\hat{x}) = a_1\hat{x}^3 + a_2\hat{x}^2 + a_3\hat{x} + a_4 \tag{65}$$

Since each node has one transverse displacement $\hat{d}_{1y}$ and rotation $\hat{\phi}_1$, we have a total of four degrees of freedom. Equation 65 is therefore a suitable equation for this case.

The transverse displacement function $\hat{v}(\hat{x})$ can be expressed in terms of nodal degrees of freedom $\hat{d}_{1y}$, $\hat{d}_{2y}$, $\hat{\phi}_1$, and $\hat{\phi}_2$ as follows.

$$\hat{v}(0) = \hat{d}_{1y} = a_4 \tag{66}$$

$$\frac{d\hat{v}(0)}{d\hat{x}} = \hat{\phi}_1 = a_3 \tag{67}$$

$$\hat{v}(L) = \hat{d}_{2y} = a_1L^3 + a_2L^2 + a_3L + a_4 \tag{68}$$

$$\frac{d\hat{v}(L)}{d\hat{x}} = \hat{\phi}_2 = 3a_1L^2 + 2a_2L + a_3 \tag{69}$$

where $\hat{\phi} = d\hat{v}/d\hat{x}$ for the assumed small rotation $\hat{\phi}$. If we solve the above equations and substitute them in Equation 65, we obtain the following equation.

$$\begin{aligned}\hat{v} = &\left[\frac{2}{L^3}\left(\hat{d}_{1y} - \hat{d}_{2y}\right) + \frac{1}{L^2}\left(\hat{\phi}_1 + \hat{\phi}_2\right)\right]\hat{x}^3 \\ &+ \left[-\frac{3}{L^2}\left(\hat{d}_{1y} - \hat{d}_{2y}\right) - \frac{1}{L}\left(2\hat{\phi}_1 + \hat{\phi}_2\right)\right]\hat{x}^2 + \hat{\phi}_1\hat{x} + \hat{d}_{1y}\end{aligned} \tag{70}$$

We can write Equation 70 in matrix form as follows.

$$\hat{v} = [N]\{\hat{d}\} \tag{71}$$

where

$$\{\hat{d}\} = \begin{Bmatrix} \hat{d}_{1y} \\ \hat{\phi}_1 \\ \hat{d}_{2y} \\ \hat{\phi}_2 \end{Bmatrix} \tag{72}$$

and

$$[N] = [N_1 \quad N_2 \quad N_3 \quad N_4] \tag{73}$$

$$N_1 = \frac{1}{L^3}(2\hat{x}^3 - 3\hat{x}^2 L + L^3) \tag{74}$$

$$N_2 = \frac{1}{L^3}(\hat{x}^3 L - 2\hat{x}^2 L^2 + \hat{x}L^3) \tag{75}$$

$$N_3 = \frac{1}{L^3}(-2\hat{x}^3 + 3\hat{x}^2 L) \tag{76}$$

$$N_4 = \frac{1}{L^3}(\hat{x}^3 L - \hat{x}^2 L^2) \tag{77}$$

N_1, N_2, N_3, and N_4 are shape functions for a beam element.

We can assume the below equation,

$$\varepsilon_x(\hat{x}, \hat{y}) = \frac{d\hat{u}}{d\hat{x}} \tag{78}$$

where $\hat{u}$ is the axial displacement function.

$$\hat{u} = -\hat{y}\frac{d\hat{v}}{d\hat{x}} \tag{79}$$

If we substitute Equation 79 into the Equation 78, we obtain that following equation.

$$\varepsilon_x(\hat{x}, \hat{y}) = -\hat{y}\frac{d^2\hat{v}}{d\hat{x}^2} \tag{80}$$

Knowing that the bending moment and shear force are related to the transverse displacement function, we can write the following equations.

$$\hat{m}(\hat{x}) = EI\frac{d^2\hat{v}}{d\hat{x}^2} \tag{81}$$

$$\hat{V} = EI\frac{d^3\hat{v}}{d\hat{x}^3} \tag{82}$$

I can write the following equations with the help of Equation 70, Equation 81 and Equation 82 for the bending moment and shear forces that I have shown in Figure 5 and Figure 6, taking into account the signs.

$$\hat{f}_{1y} = \hat{V} = EI\frac{d^3\hat{v}(0)}{d\hat{x}^3} = \frac{EI}{L^3}\left(12\hat{d}_{1y} + 6L\hat{\phi}_1 - 12\hat{d}_{2y} + 6L\hat{\phi}_2\right) \tag{83}$$

$$\hat{m}_1 = -\hat{m} = -EI\frac{d^2\hat{v}(0)}{d\hat{x}^2} = \frac{EI}{L^3}\left(6L\hat{d}_{1y} + 4L^2\hat{\phi}_1 - 6L\hat{d}_{2y} + 2L^2\hat{\phi}_2\right) \tag{84}$$

$$\hat{f}_{2y} = -\hat{V} = -EI\frac{d^3\hat{v}(L)}{d\hat{x}^3} = \frac{EI}{L^3}\left(-12\hat{d}_{1y} - 6L\hat{\phi}_1 + 12\hat{d}_{2y} - 6L\hat{\phi}_2\right) \tag{85}$$

$$\hat{m}_2 = \hat{m} = EI\frac{d^2\hat{v}(L)}{d\hat{x}^2} = \frac{EI}{L^3}\left(6L\hat{d}_{1y} + 2L^2\hat{\phi}_1 - 6L\hat{d}_{2y} + 4L^2\hat{\phi}_2\right) \tag{86}$$

If we write the above equations in matrix form, we can obtain the following equations.

$$\begin{Bmatrix} \widehat{f_{1y}} \\ \widehat{m_1} \\ \widehat{f_{2y}} \\ \widehat{m_2} \end{Bmatrix} = \frac{EI}{L^3}\begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix}\begin{Bmatrix} \widehat{d_{1y}} \\ \widehat{\phi_1} \\ \widehat{d_{2y}} \\ \widehat{\phi_2} \end{Bmatrix} \tag{87}$$

The stiffness matrix for the beam element will be as follows.

$$[k_e] = \frac{EI}{L^3}\begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix} \tag{88}$$

2.4. The Plane Truss Element

To analyze the plane truss, let's consider a two-dimensional case as shown in Figure 9. With this case we will deal with the transition of local coordinates to global coordinates. The figure represents a basic two-dimensional truss consisting of two structural members connected by pin joints and subjected to external forces. The pin connections act as nodes for the two bar elements. The node and element numbers are given in the figure together with the global coordinate system chosen. Figure 9b shows the corresponding global displacements.

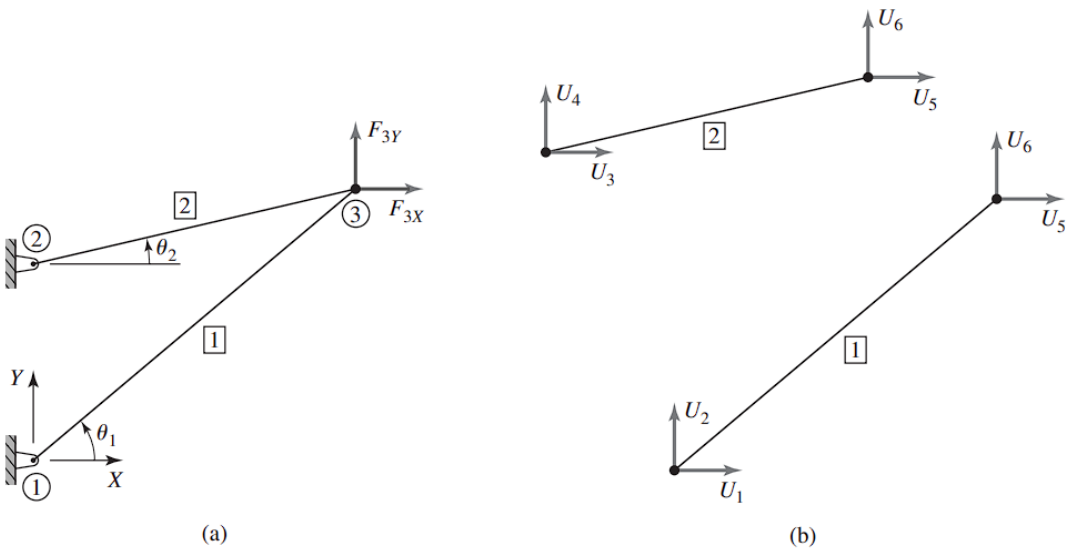


Figure 9: (a) A two-element truss. (b) Global displacement notation [3].

In Figure 10, nodes are shown, and free body diagrams are drawn to write the equilibrium conditions in the two-dimensional case we are considering. As shown in Figure 10a for Node 1, the equilibrium equations in the global X and Y directions can be written as follows.

$$F_1 - f_1^{(1)} \cos\theta_1 = 0 \tag{89}$$

$$F_2 - f_1^{(1)} \sin\theta_1 = 0 \tag{90}$$

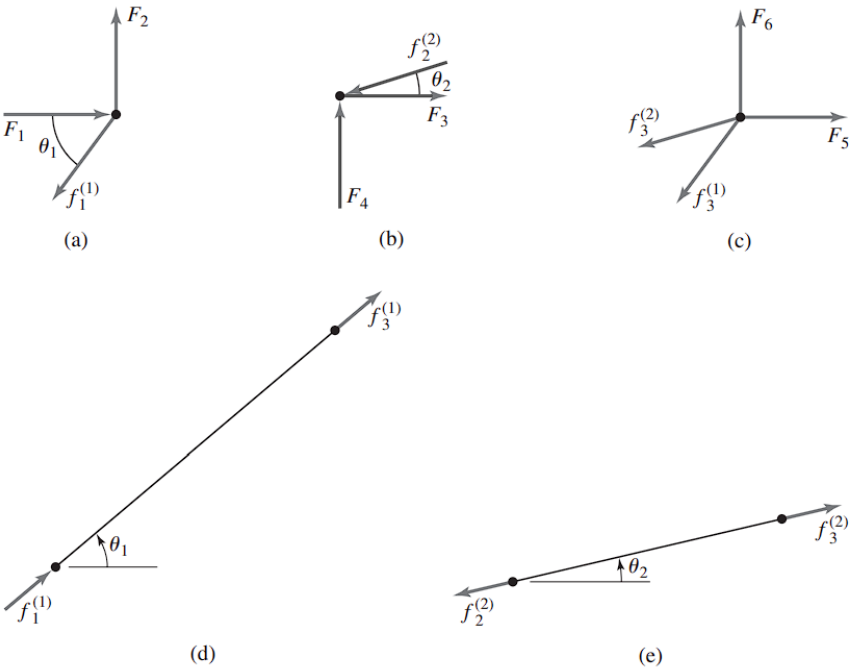


Figure 10: (a)–(c) Nodal free-body diagrams. (d) and (e) Element free-body diagrams [3].

For node 2,

$$F_3 - f_2^{(2)} \cos\theta_2 = 0 \tag{91}$$

$$F_4 - f_2^{(2)} \sin\theta_2 = 0 \tag{92}$$

For node 3,

$$F_5 - f_3^{(1)} \cos\theta_1 - f_3^{(2)} \cos\theta_2 = 0 \tag{93}$$

$$F_6 - f_3^{(1)} \sin\theta_1 - f_3^{(2)} \sin\theta_2 = 0 \tag{94}$$

Figure 11a shows the transformation of a bar element connected to nodes i and j into displacements. Due to external loading, nodes i and j are subjected to two-dimensional displacement as shown in Figure 11b. At the connection points, for the bar member to maintain the connection, the element nodes show a two-dimensional displacement showing both axial and rotational motion. The addition of vertical displacements v_1 and

v_2 at element nodes 1 and 2 allows rotation without affecting element stiffness. These displacements perpendicular to the x-axis ensure that the element remains connected to the structural connection by aligning with the joint displacements.

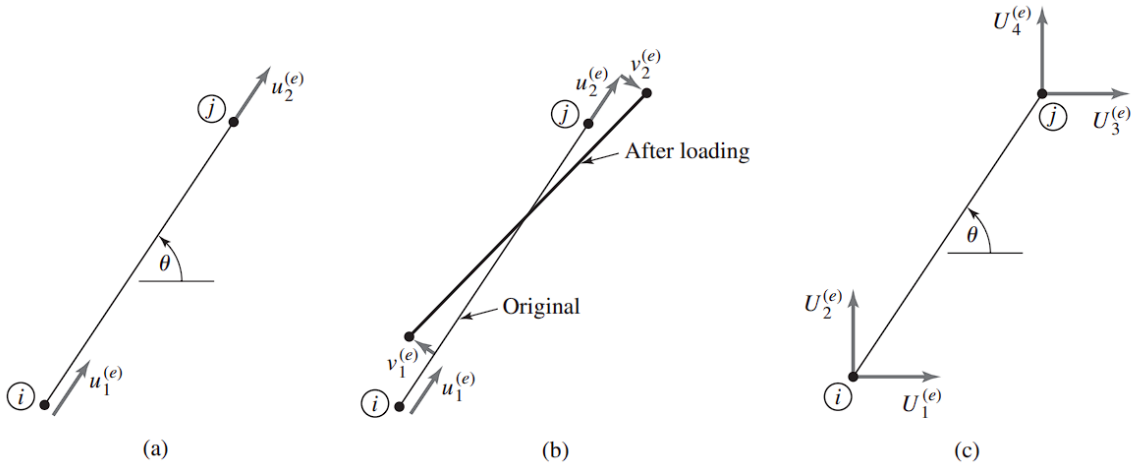


Figure 11: (a) Bar element at orientation θ. (b) Displacements of a bar element. (c) Global displacements for the bar element [3].

Figure 11c shows the connection between the element node displacements in element coordinates and their corresponding displacements in global coordinates.

$$U_1^{(e)} = \text{element node 1 displacement in the global } X \text{ direction}$$
$$U_2^{(e)} = \text{element node 1 displacement in the global } Y \text{ direction}$$
$$U_3^{(e)} = \text{element node 2 displacement in the global } X \text{ direction}$$
$$U_4^{(e)} = \text{element node 2 displacement in the global } Y \text{ direction}$$

We can give the relationship between element displacement and global displacement as follows:

$$u_1^{(e)} = U_1^{(e)} \cos\theta + U_2^{(e)} \sin\theta \tag{95}$$

$$v_1^{(e)} = -U_1^{(e)} \sin\theta + U_2^{(e)} \cos\theta \tag{96}$$

$$u_2^{(e)} = U_3^{(e)} \cos\theta + U_4^{(e)} \sin\theta \tag{97}$$

$$v_2^{(e)} = -U_3^{(e)} \sin\theta + U_4^{(e)} \cos\theta \tag{98}$$

We can write the axial deformation of the element as follows:

$$\delta^{(e)} = u_2^{(e)} - u_1^{(e)} = \left(U_3^{(e)} - U_1^{(e)}\right)\cos\theta + \left(U_4^{(e)} - U_2^{(e)}\right)\sin\theta \tag{99}$$

The net axial force acting on the element is then,

$$f^{(e)} = k^{(e)}\delta^{(e)} = k^{(e)}\left\{\left(U_3^{(e)} - U_1^{(e)}\right)\cos\theta + \left(U_4^{(e)} - U_2^{(e)}\right)\sin\theta\right\} \tag{100}$$

Using Equation 100, we can write the force in element 1 as follows:

$$f_3^{(1)} = -f_1^{(1)} = k^{(1)}[(U_5 - U_1)\cos\theta_1 + (U_6 - U_2)\sin\theta_1] \tag{101}$$

and similarly for element 2:

$$f_3^{(2)} = -f_2^{(2)} = k^{(2)}[(U_5 - U_3)\cos\theta_2 + (U_6 - U_4)\sin\theta_2] \tag{102}$$

Substituting Equations 101 and 102 into the nodal equilibrium conditions (Equations 89–94) yields

$$-k^{(1)}[(U_5 - U_1)\cos\theta_1 + (U_6 - U_2)\sin\theta_1]\cos\theta_1 = F_1 \tag{103}$$

$$-k^{(1)}[(U_5 - U_1)\cos\theta_1 + (U_6 - U_2)\sin\theta_1]\sin\theta_1 = F_2 \tag{104}$$

$$-k^{(2)}[(U_5 - U_3)\cos\theta_2 + (U_6 - U_4)\sin\theta_2]\cos\theta_2 = F_3 \tag{105}$$

$$-k^{(2)}[(U_5 - U_3)\cos\theta_2 + (U_6 - U_4)\sin\theta_2]\sin\theta_2 = F_4 \tag{106}$$

$$\begin{aligned} &k^{(2)}[(U_5 - U_3)\cos\theta_2 + (U_6 - U_4)\sin\theta_2]\cos\theta_2 \\ &+k^{(1)}[(U_5 - U_3)\cos\theta_1 + (U_6 - U_4)\sin\theta_1]\cos\theta_1 = F_5 \end{aligned} \tag{107}$$

$$\begin{aligned} &k^{(2)}[(U_5 - U_3)\cos\theta_2 + (U_6 - U_4)\sin\theta_2]\sin\theta_2 \\ &+k^{(1)}[(U_5 - U_1)\cos\theta_1 + (U_6 - U_2)\sin\theta_1]\sin\theta_1 = F_6 \end{aligned} \tag{108}$$

Equations 103 through 108 are equivalent to the matrix form

$$\begin{bmatrix} k^{(1)}c^2\theta_1 & k^{(1)}s\theta_1 c\theta_1 & 0 & 0 & -k^{(1)}c^2\theta_1 & -k^{(1)}s\theta_1 c\theta_1 \\ k^{(1)}s\theta_1 c\theta_1 & k^{(1)}s^2\theta_1 & 0 & 0 & -k^{(1)}s\theta_1 c\theta_1 & -k^{(1)}s^2\theta_1 \\ 0 & 0 & k^{(2)}c^2\theta_2 & k^{(2)}s\theta_2 c\theta_2 & -k^{(2)}c^2\theta_2 & -k^{(2)}s\theta_2 c\theta_2 \\ 0 & 0 & k^{(2)}s\theta_2 c\theta_2 & k^{(2)}s^2\theta_2 & -k^{(2)}s\theta_2 c\theta_2 & -k^{(2)}s^2\theta_2 \\ -k^{(1)}c^2\theta_{12} & -k^{(1)}s\theta_1 c\theta_1 & -k^{(2)}c^2\theta_2 & -k^{(2)}s\theta_2 c\theta_2 & k^{(1)}c^2\theta_1 + k^{(2)}c^2\theta_2 & k^{(1)}s\theta_1 c\theta_1 + k^{(2)}s\theta_2 c\theta_2 \\ -k^{(1)}s\theta_1 c\theta_1 & -k^{(1)}s^2\theta_1 & -k^{(2)}s\theta_2 c\theta_2 & -k^{(2)}s^2\theta_2 & k^{(1)}s\theta_1 c\theta_1 + k^{(2)}s\theta_2 c\theta_2 & k^{(1)}s^2\theta_1 + k^{(2)}s^2\theta_2 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \end{Bmatrix} \quad (109)$$

Equation 109 is of the form,

$$[K]\{U\} = \{F\} \quad (110)$$

where $[K]$ is the global stiffness matrix, $\{U\}$ is the vector of nodal displacements, and $\{F\}$ is the vector of applied nodal forces.

Recalling the bar element equations as,

$$\frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{Bmatrix} u_1^{(e)} \\ u_2^{(e)} \end{Bmatrix} = \begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix}\begin{Bmatrix} u_1^{(e)} \\ u_2^{(e)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(e)} \\ f_2^{(e)} \end{Bmatrix} \quad (111)$$

If we write in the global coordinate system, our matrix equation will be as follows.

$$[K_e]\begin{Bmatrix} U_1^{(e)} \\ U_2^{(e)} \\ U_3^{(e)} \\ U_4^{(e)} \end{Bmatrix} = \begin{Bmatrix} F_1^{(e)} \\ F_2^{(e)} \\ F_3^{(e)} \\ F_4^{(e)} \end{Bmatrix} \quad (112)$$

where $[K_e]$ represents the element stiffness matrix in the global coordinate system, the vector $\{F^{(e)}\}$ on the right-hand side contains the element nodal force components in the global frame, displacements $U_1^{(e)}$ and $U_3^{(e)}$ are parallel to the global X axis, while $U_2^{(e)}$ and $U_4^{(e)}$ are parallel to the global Y axis.

$$u_1^{(e)} = U_1^{(e)}\cos\theta + U_2^{(e)}\sin\theta \quad (113)$$

$$u_2^{(e)} = U_3^{(e)}\cos\theta + U_4^{(e)}\sin\theta \quad (114)$$

If we write in matrix form, we can write it as follows.

$$\begin{Bmatrix} u_1^{(e)} \\ u_2^{(e)} \end{Bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \end{bmatrix} \begin{Bmatrix} U_1^{(e)} \\ U_2^{(e)} \\ U_3^{(e)} \\ U_4^{(e)} \end{Bmatrix} = [R] \begin{Bmatrix} U_1^{(e)} \\ U_2^{(e)} \\ U_3^{(e)} \\ U_4^{(e)} \end{Bmatrix} \tag{115}$$

where

$$[R] = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \end{bmatrix} \tag{116}$$

is the transformation matrix of element axial displacements to global displacements.

Substituting Equation 116 into Equation 111

$$\begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \end{bmatrix} \begin{Bmatrix} U_1^{(e)} \\ U_2^{(e)} \\ U_3^{(e)} \\ U_4^{(e)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(e)} \\ f_2^{(e)} \end{Bmatrix} \tag{117}$$

or

$$\begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} [R] \begin{Bmatrix} U_1^{(e)} \\ U_2^{(e)} \\ U_3^{(e)} \\ U_4^{(e)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(e)} \\ f_2^{(e)} \end{Bmatrix} \tag{118}$$

If we multiply both sides of the equation by $[R]^T$, the equation we will obtain is as follows.

$$[R]^T \begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} [R] \begin{Bmatrix} U_1^{(e)} \\ U_2^{(e)} \\ U_3^{(e)} \\ U_4^{(e)} \end{Bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & \cos\theta \\ 0 & \sin\theta \end{bmatrix} \begin{Bmatrix} f_1^{(e)} \\ f_2^{(e)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(e)} \cos\theta \\ f_1^{(e)} \sin\theta \\ f_2^{(e)} \cos\theta \\ f_2^{(e)} \sin\theta \end{Bmatrix} \tag{119}$$

So,

$$[R]^T \begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} [R] \begin{Bmatrix} U_1^{(e)} \\ U_2^{(e)} \\ U_3^{(e)} \\ U_4^{(e)} \end{Bmatrix} = \begin{Bmatrix} F_1^{(e)} \\ F_2^{(e)} \\ F_3^{(e)} \\ F_4^{(e)} \end{Bmatrix} \tag{120}$$

The element stiffness matrix in the global coordinate frame is,

$$\left[K^{(e)}\right] = [R]^T \begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} [R] \tag{121}$$

After multiplying the above equation, we can find the element stiffness matrix in the global coordinate system as follows. In the formula below, $\cos\theta$'s are shown as c and $\sin\theta$'s are shown as s.

$$\left[K^{(e)}\right] = k_e \begin{bmatrix} c^2 & sc & -c^2 & -sc \\ sc & s^2 & -sc & -s^2 \\ -c^2 & -sc & c^2 & sc \\ -sc & -s^2 & sc & s^2 \end{bmatrix} \tag{122}$$

where $k_e = AE/L$ is the characteristic axial stiffness of the element.

2.5. The Spatial Truss Element

A spatial truss can be modelled using bar elements provided that the connections transmit only axial loads. To analyze the spatial truss, we can consider a case like the one shown in Figure 12. In Figure 12 we see a one-dimensional bar element connected to nodes i and j in a 3D global reference frame. For our analysis, we can define the unit vector along the element axis in the global system as follows:

$$\boldsymbol{\lambda}^{(e)} = \frac{1}{L}\left[\left(X_j - X_i\right)\mathbf{I} + \left(Y_j - Y_i\right)\mathbf{J} + \left(Z_j - Z_i\right)\mathbf{K}\right] \tag{123}$$

or

$$\boldsymbol{\lambda}^{(e)} = \cos\theta_x\,\mathbf{I} + \cos\theta_y\,\mathbf{J} + \cos\theta_z\,\mathbf{K} \tag{124}$$

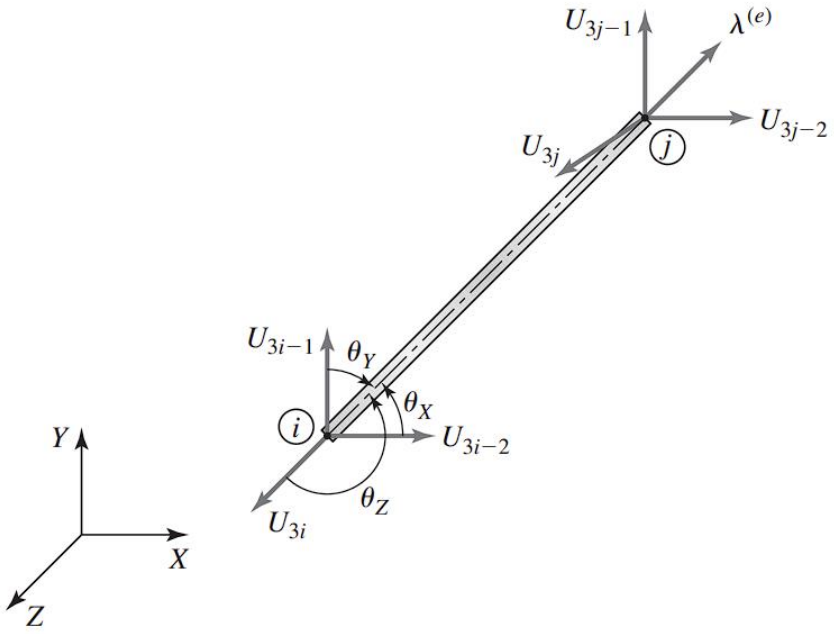


Figure 12: Bar element in a 3D frame [3].

Based on Equation 124, we can write the components of the element displacements in the 3D global system as follows:

$$u_1^{(e)} = U_1^{(e)} \cos\theta_x + U_2^{(e)} \cos\theta_y + U_3^{(e)} \cos\theta_z \tag{125}$$

$$u_2^{(e)} = U_4^{(e)} \cos\theta_x + U_5^{(e)} \cos\theta_y + U_6^{(e)} \cos\theta_z \tag{126}$$

Equations 125 and 126 can be expressed in matrix form as follows:

$$\begin{Bmatrix} u_1^{(e)} \\ u_2^{(e)} \end{Bmatrix} = \begin{bmatrix} \cos\theta_x & \cos\theta_y & \cos\theta_z & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\theta_x & \cos\theta_y & \cos\theta_z \end{bmatrix} \begin{Bmatrix} U_1^{(e)} \\ U_2^{(e)} \\ U_3^{(e)} \\ U_4^{(e)} \\ U_5^{(e)} \\ U_6^{(e)} \end{Bmatrix}$$

$$= [R]\{U^{(e)}\} \tag{127}$$

Here $[R]$ represents the transformation matrix that converts one-dimensional element displacements into a three-dimensional global coordinate system.

Also, the two-dimensional element stiffness matrix must be transformed into a three-dimensional global coordinate system.

$$\left[K^{(e)}\right] = [R]^T \begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} [R] \tag{128}$$

If we substitute the transformation matrix $[R]$ to obtain the element stiffness matrix in the global coordinate system, we obtain the following equation.

$$\left[K^{(e)}\right] = k_e \begin{bmatrix} c_x^2 & c_x c_y & c_x c_z & -c_x^2 & -c_x c_y & -c_x c_z \\ c_x c_y & c_y^2 & c_y c_z & -c_x c_y & -c_y^2 & -c_y c_z \\ c_x c_z & c_y c_z & c_z^2 & -c_x c_z & -c_y c_z & -c_z^2 \\ -c_x^2 & -c_x c_y & -c_x c_z & c_x^2 & c_x c_y & c_x c_z \\ -c_x c_y & -c_y^2 & -c_y c_z & c_x c_y & c_y^2 & c_y c_z \\ -c_x c_z & -c_y c_z & -c_z^2 & c_x c_z & c_y c_z & c_z^2 \end{bmatrix} \tag{129}$$

where

$$\begin{aligned} c_x &= \cos\theta_x \\ c_y &= \cos\theta_y \\ c_z &= \cos\theta_z \end{aligned} \tag{130}$$

2.6. The Plane Frame Element

In order to analyze the plane frame element, let's consider the case as in Figure 13. In Figure 13, we see a beam element that contains both axial and transverse loading. We are faced with a case that we have previously analyzed in beam element. We will examine the analysis of the plane frame element by adding an axial load to the case in the beam element analysis. As can be seen in Figure 13, this element offers a more comprehensive application by accommodating both axial and transverse loading. This axial load can cause many effects on the element. For example, if the axial load compresses the element, buckling can be observed.

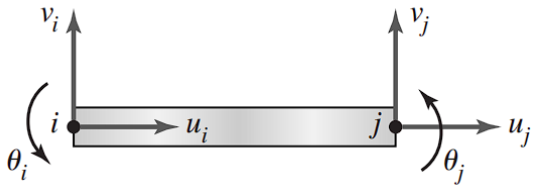


Figure 13: Beam element with nodal displacement [3].

Figure 14 shows the effect of axial load on bending. The effect of the axial load on bending is directly related to the deflection. The deflection at a given point acts as a moment arm for the axial load.

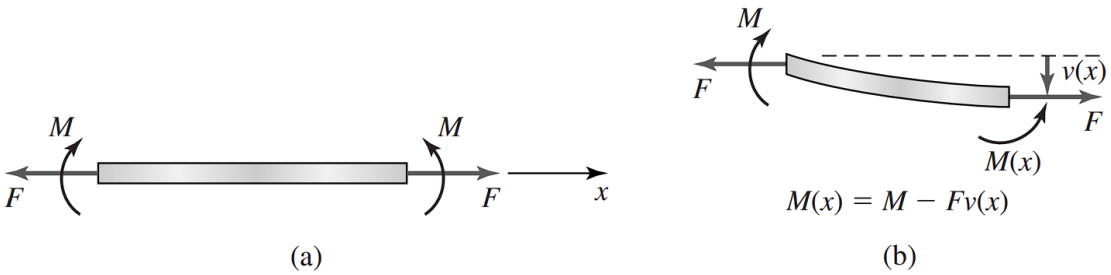


Figure 14: (a) Beam under bending moment and axial load. (b) Section of beam with deflection [3].

To obtain the element stiffness matrix of the beam element with axial load, we need bar element stiffness matrix and beam element stiffness matrix. We can give the element stiffness matrix of the beam element with axial load as follows.

$$[k_e] = \begin{bmatrix} \frac{AE}{L} & -\frac{AE}{L} & 0 & 0 & 0 & 0 \\ -\frac{AE}{L} & \frac{AE}{L} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{12EI_z}{L^3} & \frac{6EI_z}{L^2} & -\frac{12EI_z}{L^3} & \frac{6EI_z}{L^2} \\ 0 & 0 & \frac{6EI_z}{L^2} & \frac{4EI_z}{L} & -\frac{6EI_z}{L^2} & \frac{2EI_z}{L} \\ 0 & 0 & -\frac{12EI_z}{L^3} & -\frac{6EI_z}{L^2} & \frac{12EI_z}{L^3} & -\frac{6EI_z}{L^2} \\ 0 & 0 & \frac{6EI_z}{L^2} & \frac{2EI_z}{L} & -\frac{6EI_z}{L^2} & \frac{4EI_z}{L} \end{bmatrix} \quad (131)$$

This is simply,

$$[k_e] = \begin{bmatrix} [k_{axial}] & [0] \\ [0] & [k_{beam}] \end{bmatrix} \quad (132)$$

To write the element stiffness matrix in Equation 131 in a more appropriate way, if we give the element displacement vector as follows, the element displacement vector becomes,

$$\{\delta\} = \begin{Bmatrix} u_1 \\ v_1 \\ \theta_1 \\ u_2 \\ v_2 \\ \theta_2 \end{Bmatrix} \quad (133)$$

The element stiffness matrix $[k_e]$ becomes:

$$[k_e] = \begin{bmatrix} \frac{AE}{L} & 0 & 0 & -\frac{AE}{L} & 0 & 0 \\ 0 & \frac{12EI_z}{L^3} & \frac{6EI_z}{L^2} & 0 & -\frac{12EI_z}{L^3} & \frac{6EI_z}{L^2} \\ 0 & \frac{6EI_z}{L^2} & \frac{4EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & \frac{2EI_z}{L} \\ -\frac{AE}{L} & 0 & 0 & \frac{AE}{L} & 0 & 0 \\ 0 & -\frac{12EI_z}{L^3} & -\frac{6EI_z}{L^2} & 0 & \frac{12EI_z}{L^3} & -\frac{6EI_z}{L^2} \\ 0 & \frac{6EI_z}{L^2} & \frac{2EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & \frac{4EI_z}{L} \end{bmatrix} \quad (134)$$

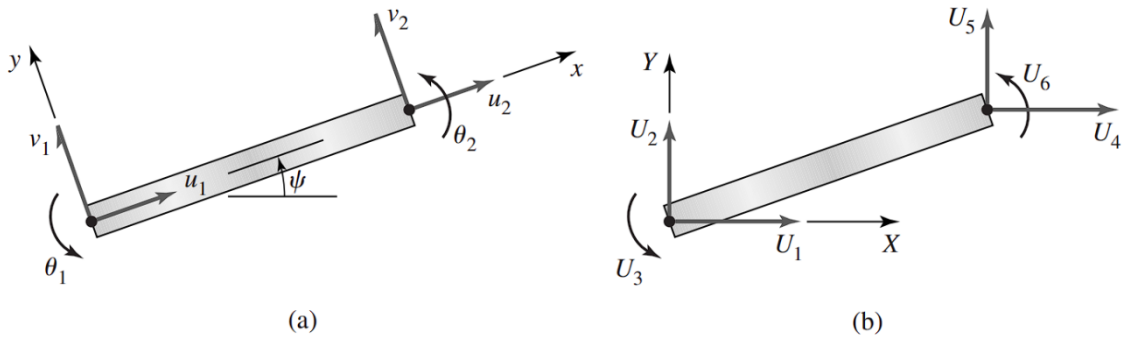


Figure 15: (a) Local displacement. (b) Global displacement [3].

Figure 15a shows the nodal displacements of the element used in Equation 133. Figure 15b shows the global displacements of the element. Using these figures, we can give the relationship between element displacement and global displacement as follows:

$$u_1 = U_1 \cos\psi + U_2 \sin\psi \tag{135}$$

$$v_1 = -U_1 \sin\psi + U_2 \cos\psi \tag{136}$$

$$\theta_1 = U_3 \tag{137}$$

$$u_2 = U_4 \cos\psi + U_5 \sin\psi \tag{138}$$

$$v_2 = -U_4 \sin\psi + U_5 \cos\psi \tag{139}$$

$$\theta_2 = U_6 \tag{140}$$

If the equations from Equation 135 to Equation 140 are written in matrix form, they are shown as follows:

$$\begin{Bmatrix} u_1 \\ v_1 \\ \theta_1 \\ u_2 \\ v_2 \\ \theta_2 \end{Bmatrix} = \begin{bmatrix} \cos\psi & \sin\psi & 0 & 0 & 0 & 0 \\ -\sin\psi & \cos\psi & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\psi & \sin\psi & 0 \\ 0 & 0 & 0 & -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \end{Bmatrix} = [R]\{U\} \tag{141}$$

where $[R]$ is the transformation matrix.

2.7. The Spatial Frame Element

To analyze the spatial frame case, we can consider a three-dimensional beam element as shown in Figure 16. The three-dimensional beam element includes axial and torsional deflections as well as two-plane bending. To obtain the stiffness matrix, we consider the beam element in bending and torsional.

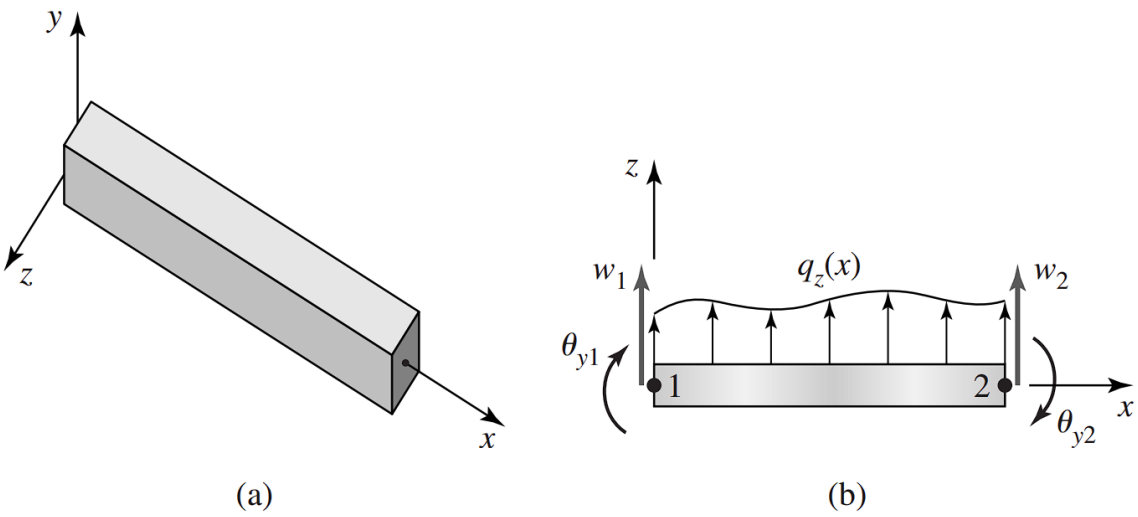


Figure 16: (a) Three-dimensional beam element. (b) Nodal displacements in element xz plane [3].

In Figure 16b we see a beam element with a load $q_z(x)$ distributed in the positive z direction. This beam element has displacements w_1 and w_2 at nodes 1 and 2 and these displacements are z-direction displacements. The notations θ_{y1} and θ_{y2} are used for the rotations at the nodes. To specifically identify the axis of rotation, rotations in the xy plane are denoted by θ_{z1} and θ_{z2}. For the case in Figure 16b, according to the right-hand rule, the rotations about the y-axis will be positive.

The element showing bending in xz plane, as shown in Figure 16, the element stiffness matrix can be written as follows:

$$[k_e]_{xz} = \frac{EI_y}{L^3}\begin{bmatrix} 12 & -6L & -12 & -6L \\ -6L & 4L^2 & 6L & 2L^2 \\ -12 & 6L & 12 & 6L \\ -6L & 2L^2 & 6L & 4L^2 \end{bmatrix} \tag{142}$$

The matrix form of the equations of equilibrium for a two-plane bending element with axial stiffness is expressed as follows:

$$\begin{bmatrix} [k_{axial}] & [0] & [0] \\ [0] & [k_{bending}]_{xy} & [0] \\ [0] & [0] & [k_{bending}]_{xz} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ v_1 \\ \theta_{z1} \\ v_2 \\ \theta_{z2} \\ w_1 \\ \theta_{y1} \\ w_2 \\ \theta_{y2} \end{Bmatrix} = \begin{Bmatrix} f_{x1} \\ f_{x2} \\ f_{y1} \\ M_{z1} \\ f_{y2} \\ M_{z2} \\ f_{z1} \\ M_{y1} \\ f_{z2} \\ M_{y2} \end{Bmatrix} \tag{143}$$

Using the equation above, we can write the element stiffness matrix as follows:

$$[k_e] = \begin{bmatrix} [k_{axial}] & [0] & [0] \\ [0] & [k_{bending}]_{xy} & [0] \\ [0] & [0] & [k_{bending}]_{xz} \end{bmatrix} \tag{144}$$

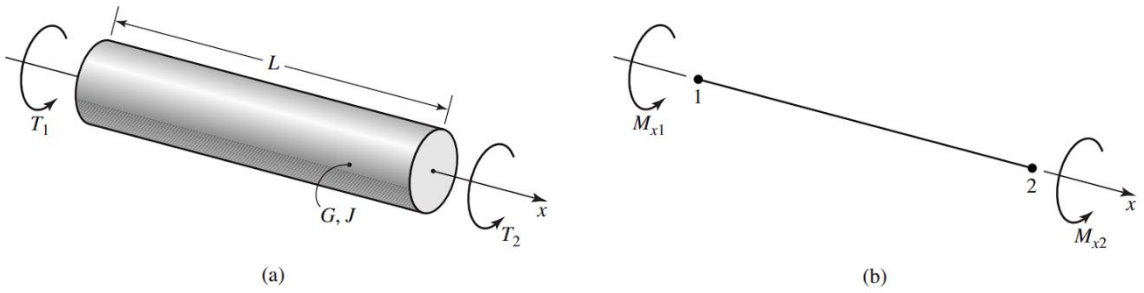


Figure 17: (a) Circular cylinder under torsion. (b) Torsional finite element notation [3].

To add torsion to the beam element as shown in Figure 17a, we need a circular cylinder subjected to torsion due to the twisting moments applied at its ends. The corresponding torsional finite element is shown in Figure 17b with nodes 1 and 2. The positive twisting moments are shown according to the right-hand rule. For a uniformly elastic circular cylinder subjected to a torque T, the twist angle per unit length is given by,

$$\phi = \frac{T}{JG} \tag{145}$$

where J is polar moment of inertia of the cross-sectional area and G is the shear modulus.

The total twist angle of the element can be given as follows:

$$\theta_{x2} - \theta_{x1} = \frac{TL}{JG} \tag{146}$$

or

$$T = \frac{JG}{L}(\theta_{x2} - \theta_{x1}) = k_T(\theta_{x2} - \theta_{x1}) \tag{147}$$

Considering the equilibrium condition $M_{x1} + M_{x2} = 0$, we can write the element equilibrium equation as follows:

$$\frac{JG}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{Bmatrix} \theta_{x1} \\ \theta_{x2} \end{Bmatrix} = \begin{Bmatrix} M_{x1} \\ M_{x2} \end{Bmatrix} \tag{148}$$

Equation 148 gives us the torsional stiffness matrix. We can express the torsional stiffness matrix as,

$$[k_{torsion}] = \frac{JG}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{149}$$

If include the torsional stiffness matrix we obtained in Equation 149 in Equation 144 to obtain the element stiffness matrix of the 3D beam element, our equation will be as:

$$\begin{bmatrix} [k_{axial}] & [0] & [0] & [0] \\ [0] & [k_{bending}]_{xy} & [0] & [0] \\ [0] & [0] & [k_{bending}]_{xz} & [0] \\ [0] & [0] & [0] & [k_{torsion}] \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ v_1 \\ \theta_{z1} \\ v_2 \\ \theta_{z2} \\ w_1 \\ \theta_{y1} \\ w_2 \\ \theta_{y2} \\ \theta_{x1} \\ \theta_{x2} \end{Bmatrix} = \begin{Bmatrix} f_{x1} \\ f_{x2} \\ f_{y1} \\ M_{z1} \\ f_{y2} \\ M_{z2} \\ f_{z1} \\ M_{y1} \\ f_{z2} \\ M_{y2} \\ M_{x1} \\ M_{x2} \end{Bmatrix} \tag{150}$$

3. PROGRAM DESIGN PROCEDURE

3.1. MATLAB

MATLAB is a special programming platform designed specifically for scientists and engineers, designed for the purpose of analyzing systems and data, and provides convenience in creating designs and applications. Using the matrix-based MATLAB language, it enables the expression of computational mathematics for problem solving. In this project, I decided to use the MATLAB program because I think it will provide me with a lot of support and convenience.

3.2. User Input: Getting Number Nodes and Elements

To facilitate user interaction, I have implemented a dialogue box prompting the user to input the number of nodes and elements required for the truss structure analysis. The 'inputdlg' function creates this dialogue box, displaying the prompt "Enter the number of nodes:" and "Enter the number of elements:", with a title "Input".

```
prompt = {'Enter the number of nodes:', 'Enter the number of elements:'};
dlgtitle = 'Input';
dims = [1 50];
definput = {'3', '3'}; % Default values
answer = inputdlg(prompt, dlgtitle, dims, definput);
```

Here, the 'dims = [1 50]' defines the size of the input fields in the dialogue box, ensuring they are wide enough to accommodate reasonable input lengths.

Following the prompt, the code enters a validation loop to ensure that the user provides valid input. It checks if the user cancels the input or leaves any input field empty. If so, it displays an error message asking the user to enter values for both fields.

```
% Validate user input
while true
    if isempty(answer)
        disp('User canceled input.');
```

```
    return;
end
if any(cellfun(@isempty, answer))
    errMsg = 'Please enter values for both fields.';
    errTitle = 'Missing Input';
    uiwait(msgbox(errMsg, errTitle, 'error'));
    answer = inputdlg(prompt, dlgtitle, dims, definput);
    continue;
end
```

Additionally, it checks whether the entered values are positive integers. If either the number of nodes or elements is less than or equal to zero, it displays an error message stating that both the number of elements and nodes must be greater than zero.

```
number_nodes = str2double(answer{1});
number_elements = str2double(answer{2});
if number_elements <= 0 || number_nodes <= 0
    errMsg = 'Number of elements and nodes must be greater than zero.';
    errTitle = 'Invalid Input';
    uiwait(msgbox(errMsg, errTitle, 'error'));
    answer = inputdlg(prompt, dlgtitle, dims, definput);
else
    break; % Exit loop if input is valid
end
```

3.3. Getting Node Coordinates

Now, I will gather the coordinates of each node in the truss structure.

```
% Node Coordinates

nodes = zeros(number_nodes, 2);

for i = 1:number_nodes

   prompt = {sprintf('Enter x coordinate for Node %d:', i), sprintf('Enter y coordinate for Node %d:', i)};

   dlgtitle = sprintf('Node %d Coordinates', i);

   dims = [1 50];

   defaultInput = {'0', '0', '0'}; % Default values

   nodeCoord = inputdlg(prompt, dlgtitle, dims, defaultInput);

   % Convert user inputs to numerical values

   x = str2double(nodeCoord{1});

   y = str2double(nodeCoord{2});

   % Store coordinates in the matrix

   nodes(i, :) = [x, y];

end
```

In this section, I iterate over each node, prompting the user to input the x and y coordinates for each node through a dialog box. The dialog box title dynamically displays the node number for clarity. The default values for x and y coordinates are set to '0'.

The 'inputdlg' function allows for user interaction and input validation, ensuring that valid numerical values are entered for each coordinate. Once the user provides the coordinates, they are stored in the 'nodes' matrix, with each row representing a node and its respective x and y coordinates.

3.4. Getting Connections of Elements

Next, I will determine the connectivity of elements in the truss structure. This involves specifying which nodes are connected by each element, establishing the structural framework necessary for subsequent calculations.

```
% Elements Connectivity

elements = zeros(number_elements, 2);

for i = 1:number_elements

    % Prompt for the connectivity of the current element

    prompt = {sprintf('First node of element %d:', i), sprintf('Second node of element %d:', i)};

    dlgtitle = sprintf('Element %d Connectivity', i);

    dims = [1 50];

    definput = {'1', '2'};

    answer = inputdlg(prompt, dlgtitle, dims, definput);

    % Convert input to numeric values

    connectivity = str2double(answer);

    % Store the connectivity in the elements array

    elements(i, :) = connectivity;

end
```

Here, I iterate over each element, prompting the user to specify the connectivity by providing the node indices of the first and second nodes of each element. The dialog box title dynamically displays the element number for clarity. Default values are set to '1' and '2' for the first and second nodes, respectively.

The 'inputdlg' function ensures user interaction and input validation, guaranteeing that valid numeric values are entered for the node indices. Once the user provides the connectivity, the node indices are stored in the 'elements' matrix, with each row representing an element and its respective node connections.

3.5. Plotting the Planar Truss System

Now, I will visualize the truss system based on the provided node coordinates and element connectivity. This step offers a graphical representation of the truss structure, aiding in the visualization and understanding of its spatial configuration.

```
% Plotting the Truss System with Node Numbering
figure;
hold on;
for i = 1:number_elements
    node1_index = elements(i, 1);
    node2_index = elements(i, 2);

    x1 = nodes(node1_index, 1);
    y1 = nodes(node1_index, 2);
    x2 = nodes(node2_index, 1);
    y2 = nodes(node2_index, 2);

    plot([x1, x2], [y1, y2], 'b-', 'LineWidth', 2);
end

% Plot nodes and number them
for i = 1:number_nodes
    plot(nodes(i,1), nodes(i,2), 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r');
    text(nodes(i,1), nodes(i,2), sprintf(' %d', i), 'VerticalAlignment','bottom', 'HorizontalAlignment','right', 'FontSize', 20, 'FontWeight', 'bold', 'Color', 'green');
end
```

```
xlabel('X');
ylabel('Y');
title('Planar Truss System');
grid on;
hold off;
```

In this section, I create a new figure and hold it for further plotting. Then, I iterate over each element, retrieving the coordinates of its connecting nodes from the 'nodes' matrix and plotting the corresponding lines to represent the truss elements. Each element is drawn with a blue line of width 2 for clarity.

Subsequently, I plot the nodes on the truss structure by iterating over each node. Nodes are represented by red circles with a marker size of 10. Additionally, each node is labeled with its corresponding index number, facilitating identification and reference during analysis.

Lastly, I provide labels for the x and y axes, a title for the plot, enable grid lines for better visualization, and release the hold on the figure for further processing.

3.6. Obtaining Material Properties and Cross-Sectional Areas

Now, I will obtain the material properties and cross-sectional areas for each truss element.

```
% Young's Modulus and Cross-sectional Area for each element
E = zeros(number_elements, 1);
A = zeros(number_elements, 1);


for i = 1:number_elements
    % Prompt for Young's Modulus and Cross-sectional Area
    prompt = {sprintf('Enter Young"s Modulus for element %d (kPa):', i), sprintf('Enter cross-sectional area for element %d (m^2):', i)};
    dlgtitle = sprintf('Element Properties %d', i);
```

```
dims = [1 50];
definput = {'210e6', '1e-4'};
answer = inputdlg(prompt, dlgtitle, dims, definput);

% Validate Young's Modulus
E_str = answer{1};
while isempty(E_str) || isnan(str2double(E_str)) || str2double(E_str) <= 0
    if isempty(E_str)
        errMsg = 'Please enter a value for Young''s Modulus.';
    elseif isnan(str2double(E_str)) || str2double(E_str) <= 0
        errMsg = 'Please enter a valid positive numeric value for Young''s Modulus.';
    end
    errTitle = 'Invalid Input';
    uiwait(errordlg(errMsg, errTitle, 'error'));

    % Prompt again for Young's Modulus
    answer = inputdlg(prompt, dlgtitle, dims, definput);
    E_str = answer{1};
end
E(i) = str2double(E_str);

% Validate Cross-sectional Area
A_str = answer{2};
while isempty(A_str) || isnan(str2double(A_str)) || str2double(A_str) <= 0
    if isempty(A_str)
```

```
        errMsg = 'Please enter a value for cross-sectional area.';
    elseif isnan(str2double(A_str)) || str2double(A_str) <= 0
        errMsg = 'Please enter a valid positive numeric value for cross-sectional area.';
    end
    errTitle = 'Invalid Input';
    uiwait(errordlg(errMsg, errTitle, 'error'));

    % Prompt again for Cross-sectional Area
    answer = inputdlg(prompt, dlgtitle, dims, definput);
    A_str = answer{2};
  end
  A(i) = str2double(A_str);
end
```

In this section, I prompt the user to input the Young's Modulus (E) and cross-sectional area (A) for each truss element using a dialog box. The title of the dialog box displays the element number for clarity, and default values are set to '210e6' kilopascal for Young's Modulus and '1e-4' square meters for the cross-sectional area. The 'inputdlg' function ensures that valid numeric values are entered, with error handling prompting the user for correct inputs if invalid or missing values are provided.

To validate the user input, I employ a while loop that repeatedly prompts the user until valid positive numeric values are entered. The loop checks if the input is empty, not a number, or less than or equal to zero, displaying error messages as needed to guide the user. Once valid values are obtained, they are stored in the respective vectors E and A for subsequent calculations. After gathering the material properties and cross-sectional areas for all elements, these values are stored in the E (Young's Modulus) and A (cross-sectional area) vectors for further use in the analysis.

3.7. Calculation of Element Lengths

To calculate the lengths of each element in the truss structure, I use the 'CalculateElementLengths' function. I call this function with the 'nodes' and 'elements' matrices as inputs:

```
lengths = CalculateElementLengths(nodes, elements);
```

In this line of code, the 'CalculateElementLengths' function computes the lengths of all elements based on the coordinates of the nodes they connect. The 'nodes' matrix contains the coordinates of each node, while the 'elements' matrix defines which nodes each element connects. This step is essential as it determines the geometric properties of the truss elements, which are crucial for further calculations, such as stress and strain analysis.

Next, I define the 'CalculateElementLengths' function to perform these calculations. The function iterates over each element, retrieves the coordinates of the nodes it connects, and calculates the Euclidean distance between these nodes to determine the element length.

```
function lengths = CalculateElementLengths(nodes, elements)
    num_elements = size(elements, 1);
    lengths = zeros(num_elements, 1);


    for i = 1:num_elements
        node1_index = elements(i, 1);
        node2_index = elements(i, 2);


        x1 = nodes(node1_index, 1);
        y1 = nodes(node1_index, 2);
        x2 = nodes(node2_index, 1);
        y2 = nodes(node2_index, 2);
```

```
        lengths(i) = sqrt((x2 - x1)^2 + (y2 - y1)^2);
    end
end
```

Within the 'CalculateElementLengths' function, I first determine the number of elements from the size of the 'elements' matrix and initialize the 'lengths' vector to store the length of each element. The for loop iterates over each element, extracting the indices of the nodes that form the endpoints of the element. For each element, I retrieve the x and y coordinates of both nodes from the 'nodes' matrix. Using these coordinates, I calculate the Euclidean distance between the nodes, which represents the length of the element. Then, stored in the 'lengths' vector.

3.8. Calculation of Element Stiffness Matrices

To calculate the element stiffness matrices for each element in the truss structure, I iterate over all elements and compute their respective stiffness matrices using their material properties and geometric configurations.

```
for i = 1:number_elements
    node1_index = elements(i, 1);
    node2_index = elements(i, 2);

    x1 = nodes(node1_index, 1);
    y1 = nodes(node1_index, 2);
    x2 = nodes(node2_index, 1);
    y2 = nodes(node2_index, 2);

    L = lengths(i);
    theta = atan2(y2 - y1, x2 - x1) * 180 / pi;
```

```
    K_element = PlaneTrussElementStiffness(E(i), A(i), L, theta);
end
```

In this segment of the code, I start by iterating through each element in the truss structure. For each element, I first determine the indices of the nodes that define the endpoints of the element from the 'elements' matrix. Using these indices, I extract the x and y coordinates of the nodes from the 'nodes' matrix. These coordinates are necessary for calculating the length and orientation of the element.

Next, I retrieve the precomputed length of the current element from the 'lengths' vector. The orientation of the element is determined by calculating the angle 'theta' using the 'atan2' function, which provides the angle in radians between the element and the horizontal axis. I then convert this angle to degrees for easier interpretation.

With the length and orientation known, I call the 'PlaneTrussElementStiffness' function to calculate the element stiffness matrix. This matrix, 'K_element', is essential for assembling the global stiffness matrix of the truss system, as it encapsulates the material properties and geometric configuration of the element.

Next, I define the 'PlaneTrussElementStiffness' function to perform these calculations.

```
function K_element = PlaneTrussElementStiffness(E, A, L, theta)
    x = theta * pi / 180;
    C = cos(x);
    S = sin(x);
    K_element = E * A / L * [C^2, C*S, -C^2, -C*S;
                             C*S, S^2, -C*S, -S^2;
                             -C^2, -C*S, C^2, C*S;
                             -C*S, -S^2, C*S, S^2];
end
```

In the 'PlaneTrussElementStiffness' function, I start by converting the angle theta from degrees to radians, as trigonometric functions in MATLAB expect angles in radians. I then calculate the cosine (C) and sine (S) of the angle x. These trigonometric values are

used to construct the stiffness matrix.

The stiffness matrix 'K_element' is computed using the formula below:

$$K = \frac{EA}{L}[C^2, CS, -C^2, -CS;\ CS, S^2, -CS, -S^2;\ -C^2, -CS, C^2, CS;\ -CS, -S^2, CS, S^2\ ] (151)$$

where E is the Young's Modulus, A is the cross-sectional area, and L is the length of the element. This matrix relates the forces and displacements in the element and is essential for finite element analysis.

3.9. Assembling the Global Stiffness Matrix

To construct the global stiffness matrix for the truss system, I start by initializing 'K_global' as a zero matrix of size 2×number_nodes. This matrix will eventually hold the stiffness information for the entire truss structure. I then iterate over each element in the truss to calculate its contribution to the global stiffness matrix.

```
K_global = zeros(2 * number_nodes);
for i = 1:number_elements
    node1_index = elements(i, 1);
    node2_index = elements(i, 2);

    x1 = nodes(node1_index, 1);
    y1 = nodes(node1_index, 2);
    x2 = nodes(node2_index, 1);
    y2 = nodes(node2_index, 2);

    L = lengths(i);
    theta = atan2(y2 - y1, x2 - x1) * 180 / pi;

    K_element = PlaneTrussElementStiffness(E(i), A(i), L, theta);
```

```
    K_global = PlaneTrussAssemble(K_global, K_element, node1_index, node2_index);
end
```

In this segment, for each element, I first identify the indices of its two nodes. Using these indices, I extract the coordinates of the nodes from the 'nodes' matrix. With the coordinates in hand, I calculate the length (L) of the element and its orientation angle (theta) using the 'atan2' function. This angle is then converted to degrees for consistency with the previous calculations.

Next, I call the 'PlaneTrussElementStiffness' function, passing in the material properties (Young's Modulus E and cross-sectional area A), the length L, and the angle theta. This function returns the element stiffness matrix 'K_element', which encapsulates the stiffness characteristics of the individual element.

With the element stiffness matrix calculated, I then update the global stiffness matrix 'K_global' by calling the 'PlaneTrussAssemble' function. This function integrates the element stiffness matrix into the appropriate positions within the global stiffness matrix, ensuring that the stiffness contributions of all elements are correctly accounted for.

Next, I define the 'PlaneTrussAssemble' function to handle the assembly process.

```
function K_global = PlaneTrussAssemble(K_global, k, i, j)
    index_i = 2 * i - 1;
    index_j = 2 * j - 1;

    K_global(index_i:index_i+1, index_i:index_i+1) = K_global(index_i:index_i+1, index_i:index_i+1) + k(1:2, 1:2);
    K_global(index_i:index_i+1, index_j:index_j+1) = K_global(index_i:index_i+1, index_j:index_j+1) + k(1:2, 3:4);
    K_global(index_j:index_j+1, index_i:index_i+1) = K_global(index_j:index_j+1, index_i:index_i+1) + k(3:4, 1:2);
    K_global(index_j:index_j+1, index_j:index_j+1) = K_global(index_j:index_j+1,
```

```
index_j:index_j+1) + k(3:4, 3:4);
end
```

In the ‘PlaneTrussAssemble’ function, I start by determining the appropriate indices for the nodes within the global stiffness matrix. Since each node has two degrees of freedom (x and y directions), the indices for node 'i' are '2*i-1' and '2*i', and similarly for node 'j'.

I then proceed to update the global stiffness matrix 'K_global' by adding the corresponding submatrices from the element stiffness matrix 'k'. The element stiffness matrix 'k' is a 4x4 matrix that describes the stiffness relationship between the degrees of freedom of the two nodes. These submatrices are added to the appropriate positions in the global stiffness matrix to account for the stiffness contributions of the element.

By iterating through all elements and assembling their stiffness contributions into the global stiffness matrix, I ensure that the global stiffness matrix accurately represents the entire truss structure, incorporating the stiffness characteristics of all elements.

3.10. Inclined Support Check

In this section of the code, I address the possibility of an inclined support in the truss structure. First, I prompt the user to confirm if there is an inclined support by using the 'questdlg' function. If the user indicates that an inclined support exists by selecting 'Yes', the code proceeds to gather further details about this support.

```
% Inclined Support
inclined_support_exists = questdlg('Is there an inclined support?', 'Inclined Support', 'Yes', 'No', 'No');
if strcmpi(inclined_support_exists, 'Yes')
    % Inclined Support Node
    prompt = {'Enter the node number for the inclined support: '};
    dlgtitle = 'Inclined Support Node';
    dims = [1 50];
    definput = {'1'};
```

```
    answer = inputdlg(prompt, dlgtitle, dims, definput);
    inclined_node = str2double(answer{1});

    % Inclined Support Angle
    prompt = {'Enter the angle of inclination for the support (in degrees): '};
    dlgtitle = 'Inclined Support Angle';
    dims = [1 50];
    definput = {'45'};
    answer = inputdlg(prompt, dlgtitle, dims, definput);
    inclined_angle = str2double(answer{1});
```

To determine the location and orientation of the inclined support, I first prompt the user to input the node number where the inclined support is applied. The node number entered by the user is then converted from string to numeric format for further processing. Next, I prompt the user to specify the angle of inclination for the support. The angle is input through a dialog box and converted from string to numeric format.

With the node number and angle of inclination now known, I proceed to create a transformation matrix 'T'. The transformation matrix is initially set as an identity matrix of size 2×number_nodes to ensure it matches the dimensions of the global stiffness matrix. This identity matrix serves as a base, which I will modify to account for the inclined support.

```
    % Create the transformation matrix
    T = eye(2*number_nodes); % Create unit matrix
    T = PlaneTrussInclinedSupport(T, inclined_node, inclined_angle);

    % Update the global stiffness matrix
    K_global = T * K_global * T';
end
```

To incorporate the inclined support into the transformation matrix, I call the function 'PlaneTrussInclinedSupport', passing the transformation matrix 'T', the node number 'inclined_node', and the angle 'inclined_angle' as arguments. This function adjusts the transformation matrix to reflect the rotation due to the inclined support. Once the transformation matrix 'T' is correctly set up, I update the global stiffness matrix 'K_global' by pre- and post-multiplying it with the transformation matrix and its transpose. This transformation aligns the global stiffness matrix with the orientation of the inclined support, ensuring accurate representation of the truss structure's stiffness.

Next, I define the 'PlaneTrussInclinedSupport' function, which is responsible for modifying the transformation matrix to account for the inclined support at a specific node and angle.

```
function T = PlaneTrussInclinedSupport(T, i, alpha)
    x = alpha * pi / 180; % degrees to radians
    T(2*i-1, 2*i-1) = cos(x);
    T(2*i-1, 2*i) = sin(x);
    T(2*i, 2*i-1) = -sin(x);
    T(2*i, 2*i) = cos(x);
end
```

In the 'PlaneTrussInclinedSupport' function, I first convert the angle 'alpha' from degrees to radian. This conversion is necessary because trigonometric functions in MATLAB use radians. I then update the appropriate elements of the transformation matrix 'T' for the specified node 'i' to reflect the rotation by the angle 'alpha'. The cosine and sine of the angle are used to populate the transformation matrix, ensuring that the support's inclination is correctly represented in the global stiffness matrix. This adjusted transformation matrix is then returned and used to update the global stiffness matrix 'K_global', completing the process of incorporating the inclined support into the truss analysis.

3.11. Storing the Original Global Stiffness Matrix

```
K_global_original = K_global;
```

In this part of the code, I am storing the original global stiffness matrix before applying any modifications due to boundary conditions. This is done by assigning 'K_global' to a new variable 'K_global_original'. The purpose of this step is to retain a copy of the initial global stiffness matrix, which represents the system's stiffness characteristics without any constraints or transformations applied.

By doing this, I ensure that I have access to the unaltered stiffness matrix later in the analysis. This is particularly important when calculating the reactions at the supports after solving for the nodal displacements. The unmodified 'K_global_original' will be used to compute these reactions accurately, reflecting the true forces in the structure before any boundary condition adjustments were made.

3.12. Initialization of Vectors

```
F = zeros(2*number_nodes, 1); % External forces vector

U = zeros(2*number_nodes, 1); % Displacements vector

known_U = false(2*number_nodes, 1); % Logical array to track known displacements
```

In this section of the code, I am initializing three essential vectors for the truss analysis. First, I create 'F', a vector of zeros with a length twice the number of nodes, which will represent the external forces applied to each node in the structure. Each node has two corresponding entries in this vector, one for the x-direction and one for the y-direction.

Next, I initialize 'U', another vector of zeros with the same length, which will store the displacements of each node. Like the force vector, each node has two entries in the displacement vector, corresponding to the displacements in the x and y directions.

Finally, I set up 'known_U', a logical array also with a length of twice the number of nodes, initialized to false. This array is used to track which displacements are known or specified as boundary conditions. Each entry in this logical array corresponds to an entry in the displacement vector 'U', indicating whether the displacement at that position is known (true) or unknown (false). This setup is crucial for later stages of the analysis, where I will apply boundary conditions and solve for the unknown displacements.

3.13. Applying Boundary Conditions

In this section, I handle the imposition of boundary conditions on the truss system.

```
% Boundary conditions
for i = 1:number_nodes
   % Check for roller support
   roller_support = questdlg(sprintf('Is there a roller support at Node %d?', i),'Roller Support','Yes','No','No');
   if strcmpi(roller_support, 'Yes')
      % Prompt for direction of roller support
      direction = questdlg(sprintf('In which direction is the roller support at Node %d?', i),'Roller Support Direction','X','Y','X');
      if strcmpi(direction, 'X')
         prompt = {sprintf('Enter the vertical force applied to Node %d (Fy in kN): ', i)};
         dlgtitle = 'Vertical Force';
      elseif strcmpi(direction, 'Y')
         prompt = {sprintf('Enter the horizontal force applied to Node %d (Fx in kN): ', i)};
         dlgtitle = 'Horizontal Force';
      end
      dims = [1 50];
      definput = {'0'};
      answer = inputdlg(prompt, dlgtitle, dims, definput);
      force = str2double(answer{1});
      if strcmpi(direction, 'X')
         F(2*i) = force;
         known_U(2*i-1) = true; % X displacement is zero
```

```
        elseif strcmpi(direction, 'Y')
            F(2*i-1) = force;
            known_U(2*i) = true; % Y displacement is zero
        end
    else
        % Check for boundary condition type
        bc_type = questdlg(sprintf('Is the boundary condition force (F) or displacement (U) at Node %d?', i), ...
                'Boundary Condition Type', 'Force', 'Displacement', 'Force');

        if strcmpi(bc_type, 'Force')
            % Prompt for force components
            prompt = {sprintf('Enter the force along x-axis (Fx in kN) at Node %d:', i), ...
                    sprintf('Enter the force along y-axis (Fy in kN) at Node %d:', i)};
            dlgtitle = 'Force';
            dims = [1 50];
            definput = {'0', '0'};
            answer = inputdlg(prompt, dlgtitle, dims, definput);
            % Convert input to numeric values
            force_components = str2double(answer);
            % Assign force components to force vector
            F(2*i-1:2*i) = force_components;
        elseif strcmpi(bc_type, 'Displacement')
            % Prompt for displacement components
            prompt = {sprintf('Enter the displacement along x-axis (Ux in mm) at Node %d:', i), ...
```

```
            sprintf('Enter the displacement along y-axis (Uy in mm) at Node %d:', i)};
        dlgtitle = 'Displacement';
        dims = [1 50];
        definput = {'0', '0'};
        answer = inputdlg(prompt, dlgtitle, dims, definput);
        % Convert input to numeric values
        displacement_components = str2double(answer);
        % Assign displacement components to displacement vector
        U(2*i-1:2*i) = displacement_components;
        known_U(2*i-1:2*i) = true; % Mark displacements as known
    end
  end
end
```

I start by iterating over each node in the truss system using a for loop. For each node, I check if there is a roller support using a dialog box (questdlg). This dialog box asks whether there is a roller support at the current node. If the user selects "Yes," another dialog box appears, asking for the direction of the roller support, either X or Y.

If the roller support direction is X, I prompt the user to enter the vertical force (Fy) applied to the node using an 'inputdlg' function. The title of this input dialog is 'Vertical Force.' Conversely, if the roller support direction is Y, I prompt the user to enter the horizontal force (Fx) applied to the node, with the dialog title set to 'Horizontal Force.'

After the user inputs the force value, I convert it to a numeric value using the 'str2double' function. If the roller support direction is X, I assign the vertical force to the corresponding element in the force vector 'F' and mark the X displacement as zero by setting 'known_U(2*i-1)' to true. Similarly, if the direction is Y, I assign the horizontal force to the appropriate element in the force vector and mark the Y displacement as zero by setting 'known_U(2*i)' to true.

If there is no roller support at the node, I prompt the user to specify whether the boundary condition is a force or a displacement using another dialog box. If the user selects "Force," I prompt for the force components along the x-axis (Fx) and y-axis (Fy) separately, converting the user inputs to numeric values and assigning them to the respective indices in the force vector 'F'.

If the user selects "Displacement," I prompt for the displacement components along the x-axis (Ux) and y-axis (Uy) separately. After converting these inputs to numeric values, I assign them to the corresponding elements in the displacement vector 'U' and mark these displacements as known by setting 'known_U(2*i-1:2*i)' to true.

3.14. Modify the Global Stiffness Matrix and Force Vector Based on Known Displacements

In this part of the code, I modify the global stiffness matrix and the force vector to account for the known displacements in the system.

```
% Modify the global stiffness matrix and force vector based on known displacements
for i = 1:length(known_U)
   if known_U(i)
      K_global(i, :) = 0; % Set the row to zero
      K_global(i, i) = 1; % Set the diagonal to one
      F(i) = U(i); % Set the force equal to the known displacement
   end
end
```

I start by iterating over each element in the 'known_U' array using a for loop. The 'known_U' array is a logical array where each element indicates whether the corresponding displacement is known (true) or unknown (false). For each iteration, I check if the current displacement is known using an if statement (if known_U(i)). If the displacement is known, I proceed with several modifications.

First, I set the entire row of the global stiffness matrix 'K_global' corresponding to the known displacement to zero. This is achieved by assigning a row of zeros to 'K_global(i, :)'. This step effectively removes the influence of the current equation from the system, as the known displacement will be directly imposed rather than solved for. Then, I set the diagonal element of the global stiffness matrix 'K_global' corresponding to the known displacement to one (K_global(i, i) = 1). This step ensures that the equation remains mathematically consistent, as setting the diagonal to one preserves the identity in the matrix equation. By doing this, I ensure that the known displacement value is directly imposed without affecting the other unknowns.

Finally, I set the corresponding element in the force vector 'F' to the value of the known displacement (F(i) = U(i)). This step ensures that the force vector correctly reflects the constraints imposed by the known displacements. By setting F(i) to U(i), I impose the known displacement value directly into the force vector, which is necessary for solving the system of equations accurately.

These modifications to the global stiffness matrix and force vector are crucial for incorporating the boundary conditions into the system of equations. By handling the known displacements in this way, I ensure that the constraints are accurately represented, allowing for the correct solution of the unknown displacements and forces in the truss system.

3.15. Solving for Nodal Displacements and Nodal Forces

To solve for the unknown displacements in the truss system, I start by using the modified global stiffness matrix and the force vector. The matrix equation K_global \ F is used to find the unknown displacements. This operation yields the vector 'U_unknown', which contains the values of the unknown displacements.

```
% Solve for unknown displacements

U_unknown = K_global \ F;
```

Once I have the 'U_unknown' vector, I proceed to update the displacements vector 'U' with these solved values. The 'known_U' array is a logical array that indicates which displacements are known (true) and which are unknown (false). By using logical indexing with ~known_U, I can update only the positions in the U vector corresponding

to the unknown displacements. This step ensures that the U vector now contains both the known displacements (input earlier) and the newly solved unknown displacements.

```
% Update displacements vector with solved values
U(~known_U) = U_unknown(~known_U);
```

Next, I calculate the reactions at the supports using the original global stiffness matrix 'K_global_original' and the updated displacements vector U. The reactions are obtained by multiplying the original global stiffness matrix by the updated displacements vector, which is done through the matrix multiplication operation 'K_global_original * U'. This operation yields the vector F, which now contains the reaction forces at the supports. By using the original stiffness matrix, I ensure that the reactions are calculated based on the initial configuration of the system before any modifications were made for the known displacements.

```
% Calculate reactions at supports using the original global stiffness matrix
F = K_global_original * U;
```

Through these steps, I effectively solve for the unknown displacements in the truss system and calculate the reactions at the supports.

3.16. Displaying Results

To present the results of the truss analysis, I begin by displaying the nodal displacements and nodal forces. The displacements indicate how much each node has moved under the applied loads, while the forces reveal the reactions at each node.

First, I display the nodal displacements. Using a loop that iterates over each node, I extract the x and y components of displacement for each node from the U vector. I then use the 'fprintf' function to print these displacements in a formatted manner, showing the node number and the displacement value in meters to four decimal places. This loop ensures that the displacements for all nodes are displayed sequentially.

```
% Display Nodal Displacements
disp('Nodal Displacements (U in m):');
for i = 1:number_nodes
```

```
    Ux = U(2*i-1);
    Uy = U(2*i);
    fprintf('U%d_x = %.4f m\n', i, Ux);
    fprintf('U%d_y = %.4f m\n', i, Uy);
end
```

Next, I display the nodal forces in a similar manner. Again, I use a loop to iterate over each node, extracting the x and y components of the force from the F vector. The 'fprintf' function is employed to print the forces, showing the node number and the force value in kN to four decimal places. This ensures that the reaction forces at all nodes are displayed clearly.

```
% Display Nodal Forces
disp('Nodal Forces (F in kN):');
for i = 1:number_nodes
    Fx = F(2*i-1);
    Fy = F(2*i);
    fprintf('F%d_x = %.4f kN\n', i, Fx);
    fprintf('F%d_y = %.4f kN\n', i, Fy);
end
```

To provide a summary of the results in a more user-friendly format, I create a message box that consolidates the nodal displacements and forces. I construct a cell array 'result_message' that contains strings representing the headings and numerical values of the displacements and forces. The ‘num2str’ function is used to convert the vectors U and F into string format for inclusion in the message box. The 'msgbox' function is then used to display this cell array in a dialog box with the title 'Analysis Results'.

```
% Display results in a dialog box
result_message = {...
    'Nodal Displacements (U in m):', ...
```

```
    num2str(U), ...
    '', ...
    'Nodal Forces (F in kN):', ...
    num2str(F) };
dlgtitle = 'Analysis Results';
msgbox(result_message, dlgtitle);
```

3.17. Calculating Stresses in Elements

In this section, I calculate the stress for each element in the truss system.

First, I start by iterating over each element. For each element, I need to identify the nodes that define the element. I obtain the indices of these nodes from the 'elements' matrix. Then, I extract the nodal displacement vector 'u' for these nodes from the displacement vector 'U'. This vector contains the displacements of both nodes that form the element, in the order [Ux1, Uy1, Ux2, Uy2].

Next, I calculate the length and angle of the element. The coordinates of the nodes are obtained from the 'nodes' matrix. Using these coordinates, I calculate the length 'L' of the element. The angle theta of the element is computed using the 'atan2' function, which gives the angle in radians. I then convert this angle to degrees for easier interpretation.

With the length and angle known, I proceed to calculate the stress in the element. I call the function 'PlaneTrussElementStress', passing the Young's modulus E, the length L, the angle theta, and the displacement vector u. This function returns the stress sigma in the element. I store the absolute value of this stress in the 'stress_values' array.

Finally, I format and print the stress for each element. If the stress sigma is negative, it indicates compressive stress, while a positive value indicates tensile stress. I use conditional statements to append the appropriate label (Compressive or Tensile) to the printed output.

```
% Loop that calculates the stress for all elements in the system
for i = 1:number_elements
```

```
node1_index = elements(i, 1);
node2_index = elements(i, 2);

% Pull the nodal displacement vector for each element
u = U([2*node1_index-1, 2*node1_index, 2*node2_index-1, 2*node2_index]);

% Calculate the length and angle of the element
x1 = nodes(node1_index, 1);
y1 = nodes(node1_index, 2);
x2 = nodes(node2_index, 1);
y2 = nodes(node2_index, 2);
L = lengths(i);
theta = atan2(y2 - y1, x2 - x1) * 180 / pi;

% Calculate the stress
sigma = PlaneTrussElementStress(E(i), L, theta, u);
stress_values(i) = abs(sigma);

% Edit output according to stress value
if sigma < 0
    fprintf('Stress on element %d (sigma%d): %f kPa (Compressive)\n', i, i, sigma);
elseif sigma > 0
    fprintf('Stress on element %d (sigma%d): %f kPa (Tensile)\n', i, i, sigma);
else
    fprintf('Stress on element %d (sigma%d): %f kPa\n', i, i, sigma);
```

```
    end
end
```

To calculate the stress in an element, I use the ‘PlaneTrussElementStress’ function. This function takes four arguments: the Young's modulus E, the length L of the element, the angle theta of the element, and the displacement vector ‘u’. Inside the function, I first convert the angle theta from degrees to radians. Then, I compute the cosine and sine of the angle. These trigonometric values are used to construct a transformation matrix that relates the global displacements to the local coordinate system of the element. The stress ‘sigma’ is then calculated by applying this transformation matrix to the displacement vector u and scaling by the Young's modulus E divided by the length L of the element. The function returns this stress value.

```
function sigma = PlaneTrussElementStress(E, L, theta, u)
    x = theta * pi / 180;
    C = cos(x);
    S = sin(x);
    sigma = E / L * [-C -S C S] * u;
end
```

3.18. Visualizing Stress in the Truss Elements

In this section, I visualize the stress levels in each truss element by plotting them with colors corresponding to their stress magnitudes. This helps in understanding how different elements are stressed under the given load conditions.

First, I determine the maximum and minimum stress values from the 'stress_values array'. These values will be used to normalize the stress values for color mapping. I use the 'jet' colormap, which provides a range of colors from blue (low stress) to red (high stress).

Next, I create a new figure for plotting and hold it to add multiple elements to the same plot. For each element, I extract the indices of the nodes that define the element from the 'elements' matrix. Using these indices, I retrieve the coordinates of these nodes from

the 'nodes' matrix.

I then calculate the stress index for each element. The stress index is a normalized value between 0 and 1, calculated based on the element's stress relative to the minimum and maximum stress values. This index is used to determine the appropriate color from the 'jet' colormap. I round the stress index to the nearest integer within the range of the colormap size to get the corresponding color.

After determining the color, I plot the element using the 'plot' function, setting the 'Color' property to the calculated color and the 'LineWidth' property to 2 for better visibility. The coordinates ' [x1, y1] ' and ' [x2, y2] ' define the endpoints of the element.

Finally, I plot the nodes of the truss system as black circles with a size of 10 and filled with black color to distinguish them from the elements. I label the x and y axes, set the title of the plot to "Planar Truss System with Stress Visualization", and enable the grid for better readability. The 'hold off' command releases the plot hold, completing the visualization process.

```
% Define color map based on stress levels
max_stress = max(stress_values);
min_stress = min(stress_values);

% Plot elements with colors based on stress levels
figure;
hold on;
color_map = jet;
for i = 1:number_elements
    node1_index = elements(i, 1);
    node2_index = elements(i, 2);

    % Calculate stress index for color mapping
```

```
    stress_index = (stress_values(i) - min_stress) / (max_stress - min_stress);
    color_index = round(stress_index * (size(color_map, 1) - 1)) + 1;
    color = color_map(color_index, :);

    % Plot element with color
    x1 = nodes(node1_index, 1);
    y1 = nodes(node1_index, 2);
    x2 = nodes(node2_index, 2);
    y2 = nodes(node2_index, 2);
    plot([x1, x2], [y1, y2], 'Color', color, 'LineWidth', 2);
end

plot(nodes(:,1), nodes(:,2), 'ko', 'MarkerSize', 10, 'MarkerFaceColor', 'k');
xlabel('X');
ylabel('Y');
title('Planar Truss System with Stress Visualization');
grid on;
hold off;
```

3.19. Adding a Color Scale to Indicate Stress Levels

To enhance the visualization of stress levels in the truss elements, I add a color scale that provides a reference for interpreting the colors representing different stress magnitudes.

First, I create a color scale using the 'colorbar' function. This color scale will be displayed alongside the truss plot, allowing us to correlate the colors of the elements with specific stress values. I set the colormap to 'jet' to ensure that the color scale matches the colors used in the element plotting.

Next, I use the 'caxis' function to define the range of stress values represented by the color scale. The minimum and maximum stress values are used as the limits, ensuring that the full range of stress magnitudes in the truss is covered.

To make the color scale more informative, I label it with 'Stress Levels (kPa)' using the 'ylabel' function. This label clarifies that the colors correspond to stress levels measured in kilopascals (kPa).

I then adjust the ticks on the color scale for better readability. I use the 'linspace' function to create a set of evenly spaced tick values between the minimum and maximum stress levels. This ensures that the color scale has enough reference points without overcrowding. The 'arrayfun' function is used to format these tick values into strings with two decimal places, ensuring clarity and precision in the labels.

Finally, I set the tick values and their corresponding labels on the color scale using the 'set' function. This step completes the setup of the color scale, making it a useful tool for interpreting the stress levels indicated by the colors in the truss plot.

```
% Create color scale indicating stress levels
color_scale = colorbar;
colormap(jet); % Set the colormap to jet
caxis([min_stress, max_stress]);
ylabel(color_scale, 'Stress Levels (kPa)');


% Display stress values on the color scale
yticks = linspace(min_stress, max_stress, 5); % Adjust the number of ticks as needed
yticklabels = arrayfun(@(x) sprintf('%0.2f', x), yticks, 'UniformOutput', false);
set(color_scale, 'YTick', yticks, 'YTickLabel', yticklabels);
```

Now I would like to solve an example to better understand Section 3.14 and Section 3.15. For this, let us consider a system of matrix equations in two unknowns as shown below.

$$\begin{bmatrix} 2 & 4 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ U_2 \end{bmatrix} = \begin{bmatrix} R_1 \\ 13 \end{bmatrix} \tag{152}$$

Since we know the value of U_1, as I have shown in Section 3.14, the rows of the known displacements in the global stiffness matrix K will be updated to zero and our equation will look as follows.

$$\begin{bmatrix} 0 & 0 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ U_2 \end{bmatrix} = \begin{bmatrix} R_1 \\ 13 \end{bmatrix} \tag{153}$$

Then we will update the diagonals of the rows of the known displacements in the global stiffness matrix K as 1 as I have shown in Section 3.14 and the equation will look as follows.

$$\begin{bmatrix} 1 & 0 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ U_2 \end{bmatrix} = \begin{bmatrix} R_1 \\ 13 \end{bmatrix} \tag{154}$$

Then, to obtain the unknown U_2 value, we will write the U_1 value, that is, the value 1, instead of the unknown R_1 value. The equation will take the following form.

$$\begin{bmatrix} 1 & 0 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ U_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 13 \end{bmatrix} \tag{155}$$

After this step, we can now find the value of U_2.

$$\begin{bmatrix} 1 \\ U_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 3 & 5 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 13 \end{bmatrix} \tag{156}$$

$$\begin{bmatrix} 1 \\ U_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \tag{157}$$

As a result of the operations I performed, I saw that the value of U_2 is equal to 2 and thus I obtained my displacement matrix U. Now I can obtain my force matrix.

$$\begin{bmatrix} 2 & 4 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} R_1 \\ 13 \end{bmatrix} \tag{158}$$

$$\begin{bmatrix} R_1 \\ 13 \end{bmatrix} = \begin{bmatrix} 10 \\ 13 \end{bmatrix} \tag{159}$$

Thus, we have seen the operation of my MATLAB codes in Section 3.14 and Section

3.15. I tried to show the step-by-step operations.

4. VALIDATION EXAMPLE

In order to verify the MATLAB code I have written, I will solve an example from the book Kattan, Peter I. MATLAB guide to finite elements: an interactive approach, which I have taken as my reference book. In order to verify the results in the MATLAB code I wrote, I will refer to Example 5.2 in the fifth chapter of the book, The Plane Truss Element.

Example 5.2:

Consider the plane truss with an inclined support as shown in Figure 5.4. Given $E = 70$ GPa and $A = 0.004$ m^2, determine:

1. the global stiffness matrix for the structure,
2. the displacements at nodes 2, 3, and 4,
3. the reactions at nodes 1 and 4,
4. the stress in each element.

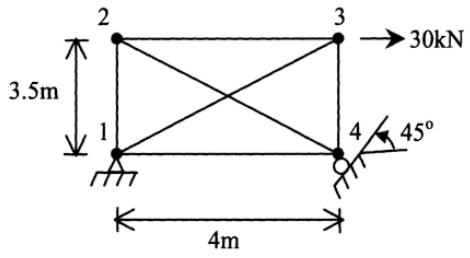


Figure 18: Validation problem [5].

When we run our code, we are asked for the number of nodes and elements of the structure we want to examine in a dialog box. To initiate the validation process, I commenced by inputting essential parameters for the planar truss structure, depicted in the accompanying figure. As can be seen in Figure 18, the system has 4 nodes and 6 elements. So, the user must enter 4 for the number of nodes field and 6 for the number of elements field as we can see the Figure 19.

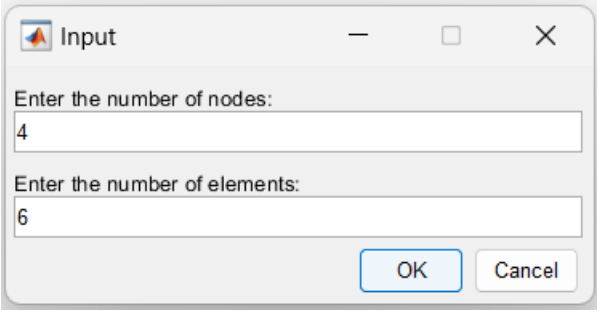


Figure 19: Input screen for number of nodes and elements.

If the user does not fill in the required field like 'enter the number of nodes' and the user click the 'OK' button, the program will show the user an error message as we see in Figure 20. After the user presses the 'OK' button, the program is designed to ask the user to enter the number of nodes and elements in Figure 19 again to make a valid entry.

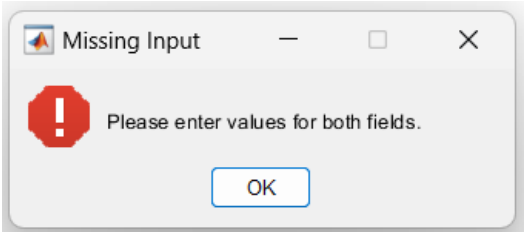


Figure 20: Error screen for missing input.

If the user wants to cancel the input or clicks the 'Cancel' button, the program will terminate with an error message as in Figure 21.

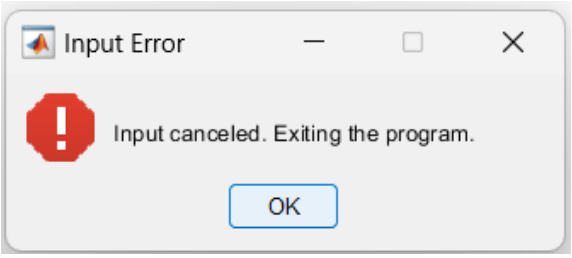


Figure 21: Error screen for input canceled.

Or if the user enters 0 or a value less than 0, the program will encounter the following error message.

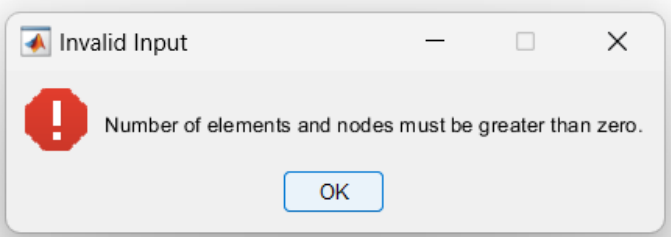


Figure 22: Error screen for invalid input.

According to validation problem as shown in Figure 18, the number of nodes should be 4, the number of elements should be 6. After these numbers inputted correctly, the program asks for coordinates each node in order It starts by asking for the coordinates of node 1. For ease of use, I preferred to separate the inputs as x and y coordinates as shown in Figure 23.

If we look at our validation problem in Figure 18, we can accept the coordinates of node 1 as (0,0). So, 0 must be entered for the x and y coordinates for node 1.

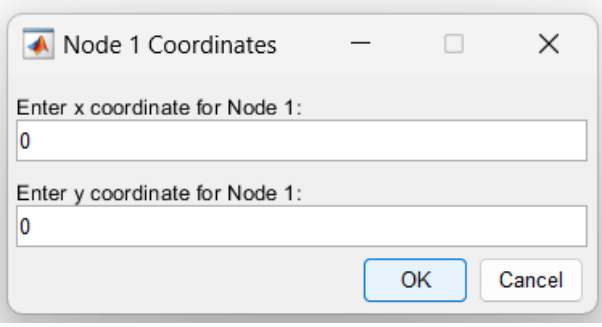


Figure 23: Input screen for node 1 coordinates.

Since we have taken the coordinates of node 1 as (0, 0), the coordinates of node 2 will be (0, 3.5) as seen in Figure 18 and the user will enter these values.

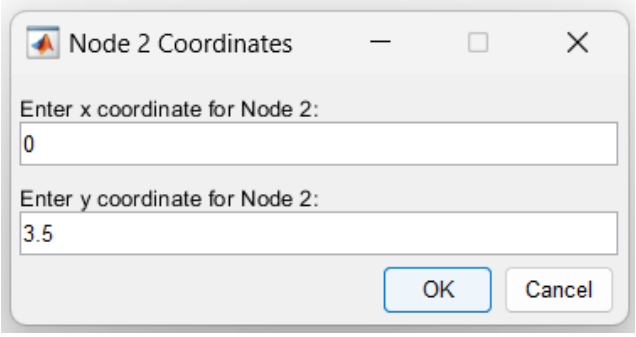


Figure 24: Input screen for node 2 coordinates.

The coordinates of node 3 will be (4, 3.5) as seen in Figure 18 and the user will enter these values.

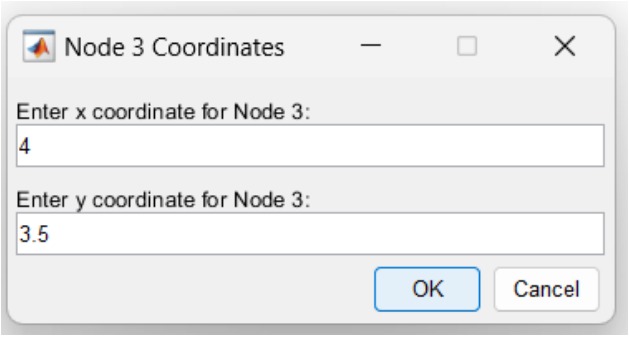


Figure 25: Input screen for node 3 coordinates.

The coordinates of node 4 will be (4, 0) as seen in Figure 18 and the user will enter these values.

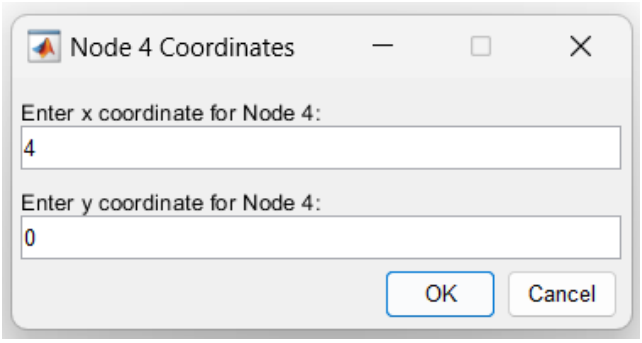


Figure 26: Input screen for node 4 coordinates.

After the user enters the node coordinate inputs, the program will ask the user about the element connectivity of the planar truss system.

Element Number	Node *i*	Node *j*
1	1	2
2	1	4
3	1	3
4	2	4
5	2	3
6	3	4

Figure 27: Element connectivity information for validation problem [5].

In the solution in the book I took as reference, the node numbers to which the element numbers are connected are given as shown in Figure 27. By looking at Figure 27 or Figure 18, we see that element 1 is connected to node 1 and node 2. So, user inputs the element 1's connectivity as shown below figure.

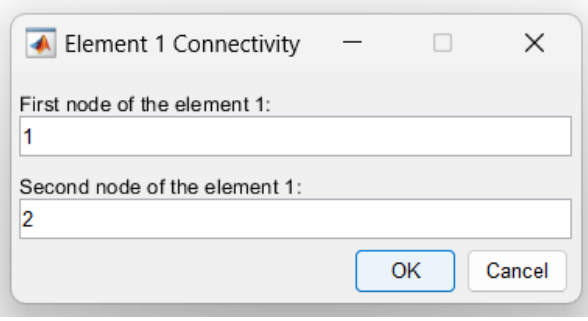


Figure 28: Input screen for element 1 connectivity.

After user inputted the element 1's connectivity, the program will ask element 2's connectivity and so on for as many elements as there are. As you can see in Figure 18 or in Figure 27, element 2 is connected to node 1 and node 4. So, user inputs the element 2's connectivity as shown in Figure 29.

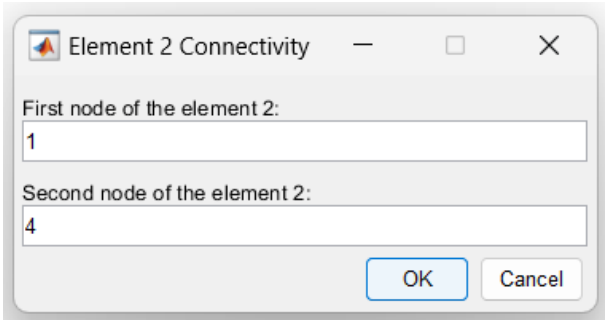


Figure 29: Input screen for element 2 connectivity.

As you can see in Figure 18 or in Figure 27, element 3 is connected to node 1 and node 3. So, user inputs the element 3's connectivity should be as shown in figure below.

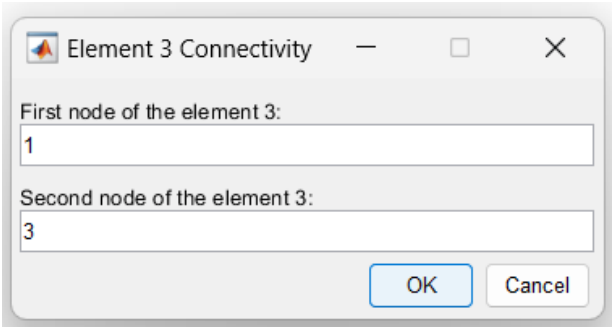


Figure 30: Input screen for element 3 connectivity.

As you can see in Figure 18 or in Figure 27, element 4 is connected to node 2 and node 4. So, user inputs the element 4's connectivity should be as shown in figure below.

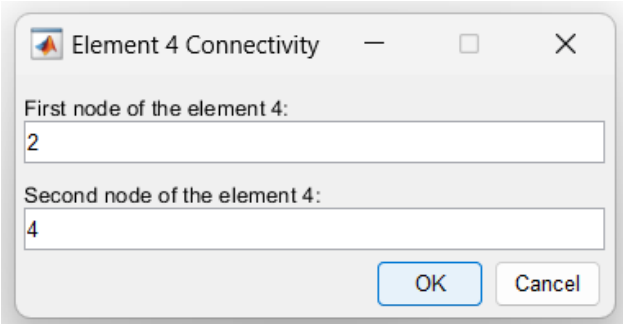


Figure 31: Input screen for element 4 connectivity.

As you can see in Figure 18 or in Figure 27, element 5 is connected to node 2 and node 3. So, user inputs the element 5's connectivity should be as shown in Figure 32.

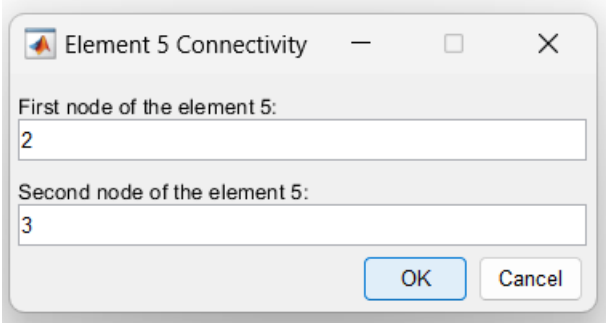


Figure 32: Input screen for element 5 connectivity.

As you can see in Figure 18 or in Figure 27, element 6 is connected to node 3 and node 4. So, user inputs the element 6's connectivity should be as shown in Figure 33.

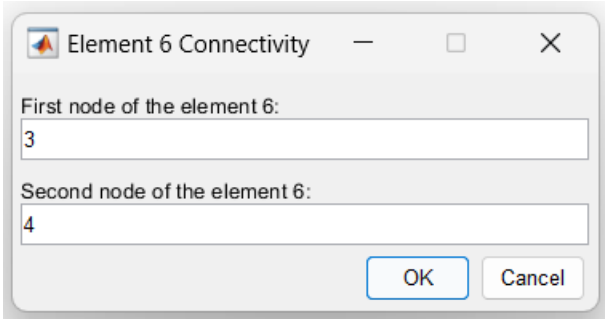


Figure 33: Input screen for element 6 connectivity.

After the user enters the connectivity of each element, the program will ask the user for the Young's Modulus and cross-sectional area of each element. First, it will start asking the material properties of element 1 and finally it will ask the material properties of element 6 because there are 6 elements in the problem we took as an example.

If we look at Figure 18 again to remember the question, Young's Modulus and cross-sectional areas of each element are the same. Therefore, $70\ GPa$ should be entered for the Young's Modulus value of each element. However, the program accepts $kilopascal\ (kPa)$ values as input, as seen in Figure 34. So, when entering the Young's Modulus value of each element, $70e6\ kPa$ must be entered. At the same time, if we look at Figure 18, we see that the cross-sectional areas of each element are the same. Therefore, $0.004\ m^2$ should be entered for the cross-sectional area value of each element. Since the program only accepts m^2 area unit for cross-sectional area inputs, we do not need to change any units. The user must enter the values $70e6\ kPa$ and

$0.004\ m^2$ for the Young's Modulus and cross-sectional area for element 1 properties, respectively.

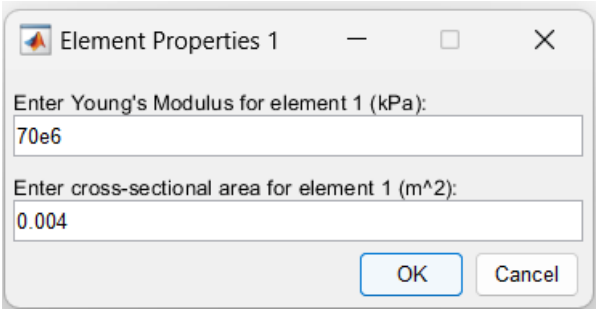


Figure 34: Input screen for element 1 properties.

As shown in Figure 18, element 2, like all elements, has a Young's Modulus of $70e6\ kPa$ and a cross-sectional area value of $0.004\ m^2$. So, user inputs the element 2's properties as shown in Figure 35.

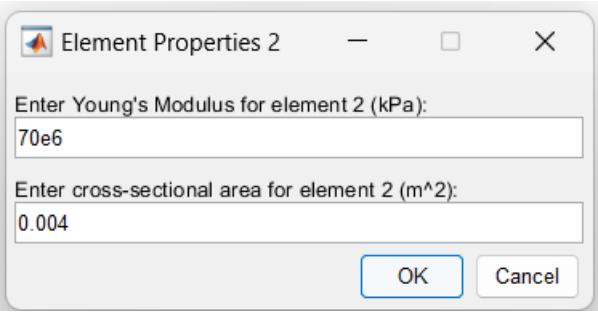


Figure 35: Input screen for element 2 properties.

As shown in Figure 18, element 3, like all elements, has a Young's Modulus of s $70e6\ kPa$ and a cross-sectional area value of $0.004\ m^2$. So, user inputs the element 3's properties as shown in Figure 36.

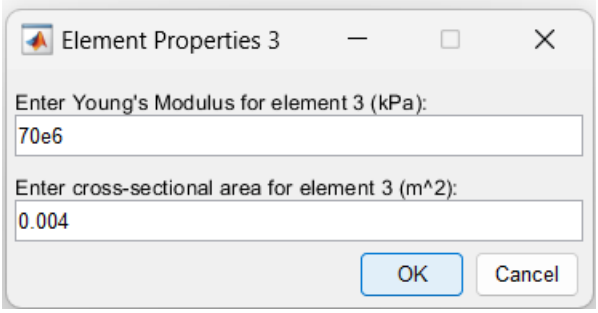


Figure 36: Input screen for element 3 properties.

The user inputs the element 4's properties as shown in Figure 37.

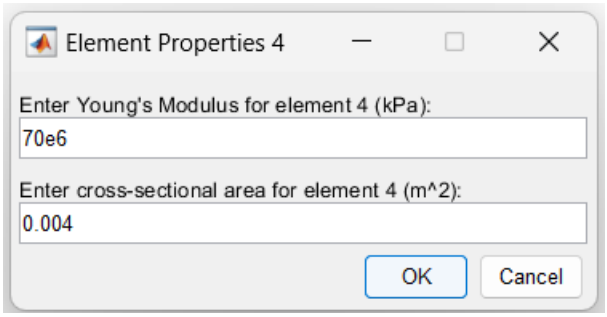


Figure 37: Input screen for element 4 properties.

The user inputs the element 5's properties as shown in figure below.

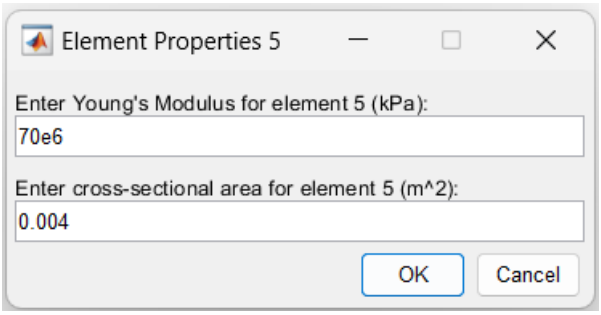


Figure 38: Input screen for element 5 properties.

By entering Young's Modulus and cross-sectional area values in Element 6, as shown in Figure 39, we enter the properties values of each element into the program. This step plays an important role in creating the stiffness matrix of each element and obtaining the global stiffness matrix.

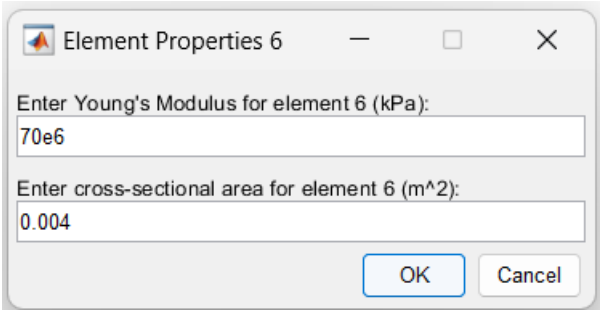


Figure 39: Input screen for element 6 properties.

After this step, the program asks the user if there is inclined support in the structure. As we can see in Figure 18, there is a 45-degree inclined support on node 4. The user should answer 'Yes' to the question 'Is there an inclined support?' in Figure 40.

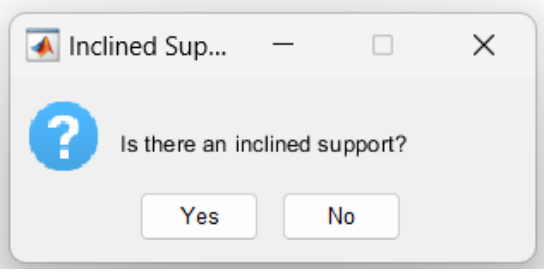


Figure 40: Inclined support.

After the user clicks the 'Yes' button, the program asks the user to enter the node number of the node with inclined support. Since the user knows that there is an inclined support in node 4, the user should input as shown in figure below.

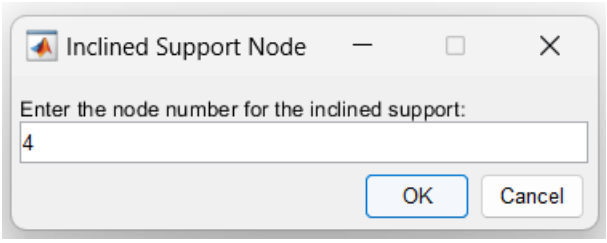


Figure 41: Input screen for entered the node number for inclined support.

After the user enters the node number containing the inclined support, the program asks the user for the inclination angle in degrees. If we look at Figure 18 again, we can see that there is a 45-degree inclination angle at node 4. So, the user should input as shown in figure below.

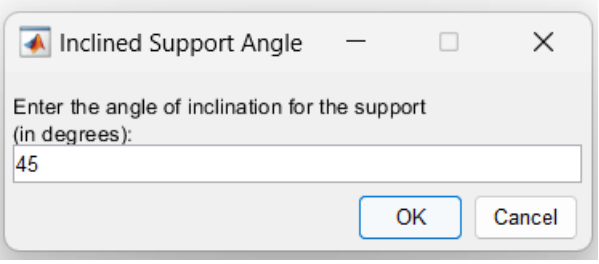


Figure 42: Input screen for entered the angle of inclination for the support.

After these steps are completed, the program asks for boundary conditions. The program first asks the user about the boundary conditions at node 1. First, program asks the user if there is roller support in node 1. If we look at Figure 18 again, we will see that there is no roller support at node 1. So, the user should click on 'No' button in Figure 43.

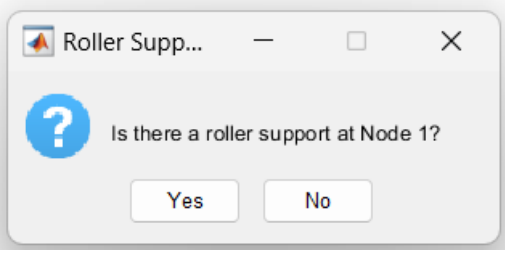


Figure 43: Roller support selection for node 1.

Then, after the user selects "No roller support" for node 1, the program asks the user for the boundary condition type for node 1, as shown in Figure 44. We know that every node must have a boundary condition. Since node 1 is a fixed node, the user must select "Displacement".

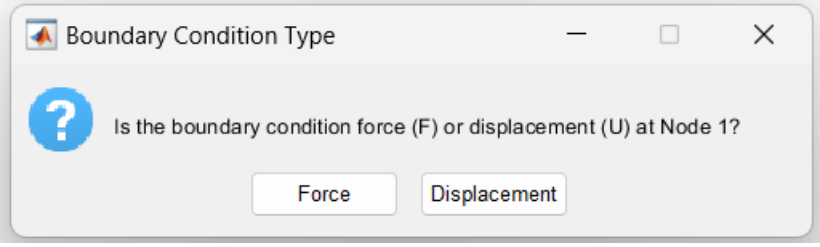


Figure 44: Boundary condition type selection for node 1.

After the user selected displacement in Figure 44, the program will ask for the value of the displacement. For our validation example, node 1 is a fixed node, so its displacements in the x and y directions will be 0. So, the user can enter the inputs like figure below.

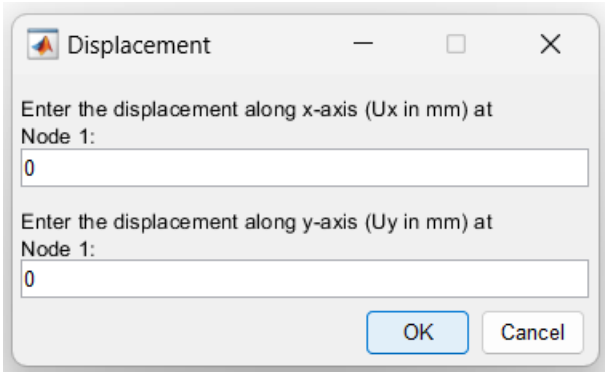


Figure 45: Input screen for displacement values at node 1.

After this step, the program will keep asking for boundary conditions according to the node number inputted by user. For our validation problem node number is 4. So, the program will repeat these steps four times.

Now, the program will ask if there is roller support at Node 2. As we see in Figure 18, there is no roller support at Node 2. So, user should click on 'No' button, as shown in Figure 46.

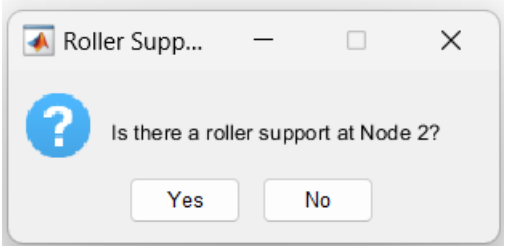


Figure 46: Roller support selection for node 2.

Then, after the user selects "No roller support" for node 2, the program asks the user for the boundary condition type for node 2, as shown in Figure 47. Node 2 is a node located at the top of the structure. No external force is not applied to Node 2. This node can move freely and is connected to the nodes below it (Node 1 and Node 4). So, the user should select 'Force' boundary condition for node 2.

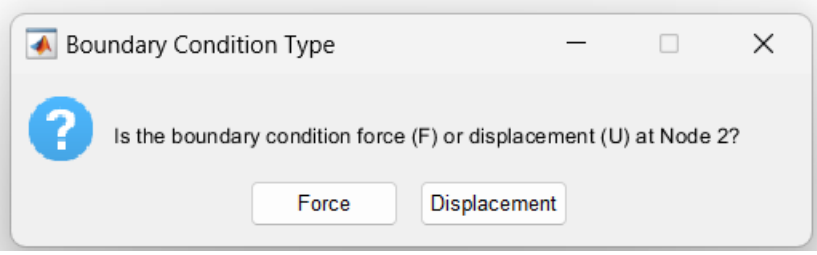


Figure 47: Boundary condition selection for node 2.

After the user selected force in Figure 47, the program will ask for the value of the force. For our validation problem, we see that there is no external force at node 2. So, the user must enter 0 in the force values in the x and y directions for node 2.

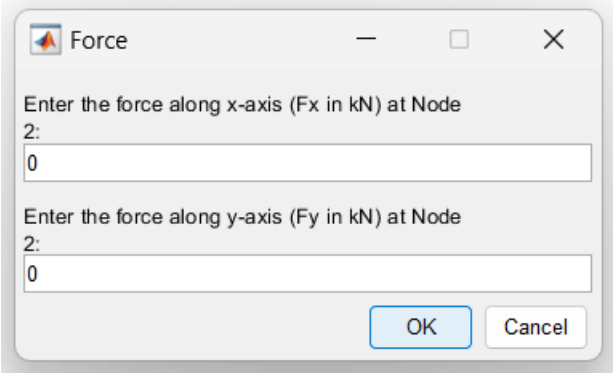


Figure 48: Input screen for force values at node 2.

Now, the program will ask if there is roller support at Node 3. As we see in Figure 18, there is no roller support at Node 3. So, user should click on 'No' button, as shown in Figure 49.

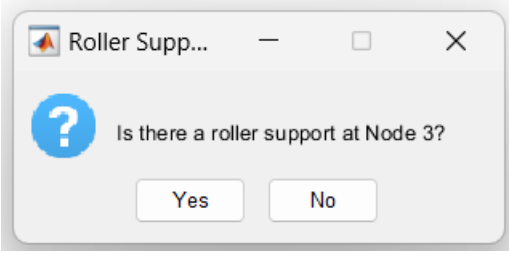


Figure 49: Roller support selection for node 3.

After the user clicks on the 'No' button, the program asks the user for the boundary condition type for node 3. For node 3, according to Figure 18, boundary condition is

given in the question as a force value at x direction which has a magnitude of 30 kN. So, the user should select F as boundary condition type in Figure 50.

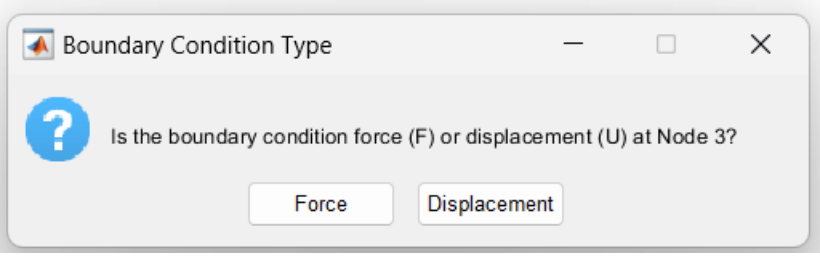


Figure 50: Boundary condition selection for node 3.

After the user selected F in the figure above, the program will ask for the force values to the user. In the validation problem, the value is given as 30 kN in the x direction. There is no external force in the y direction. So, the user should input as shown in Figure 51.

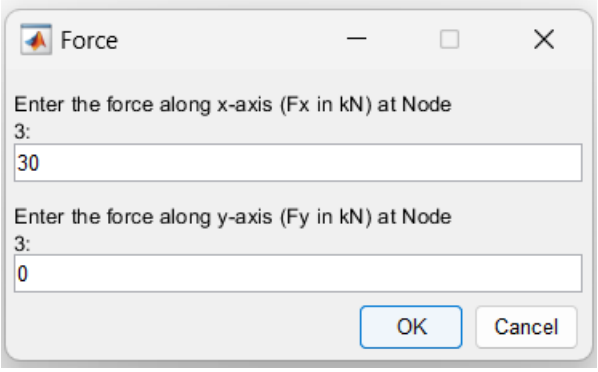


Figure 51: Input screen for force values at node 3.

Then, the program will ask if there is roller support at Node 4. As we see in Figure 18, there is a roller support at Node 4. So, user should click on 'Yes' button, as shown in Figure 52.

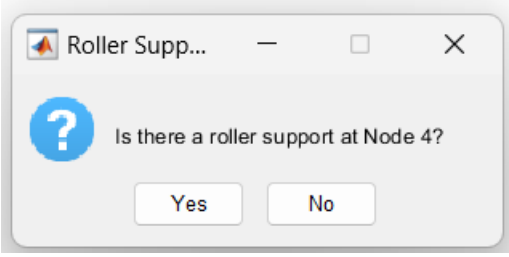


Figure 52: Roller support selection for node 4.

After the user clicks the 'Yes' button, the program will ask the user for the direction of the roller support. If we recall Figure 40, Figure 41 and Figure 42, the program asked the user which node the inclined support is in and the inclination angle. We know from Figure 18 or Figure 41 and Figure 45 that there is an inclined support at node 4 and that this inclined support has an angle of 45 degrees. There is roller support on this 45-degree inclined support. We can say that the direction of the roller support on node 4 is in the y direction because the roller support shows a reaction force in the y direction. So, user should click on 'Y' button, as shown in Figure 53.

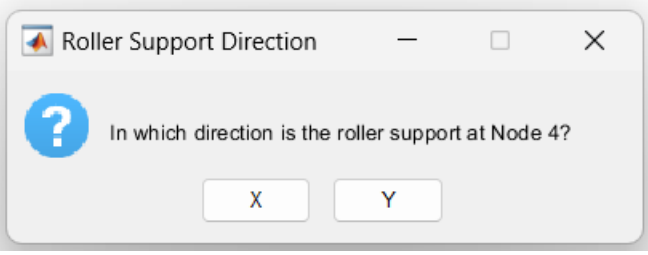


Figure 53: Roller support direction selection for node 4.

After the user clicks on the Y button, the program asks the user to enter the value of any external force applied to node 4 in the horizontal (x) direction. In our validation problem, there is no external force in the horizontal (x) direction of node 4. So, the user must enter the value 0, as shown in Figure 54.

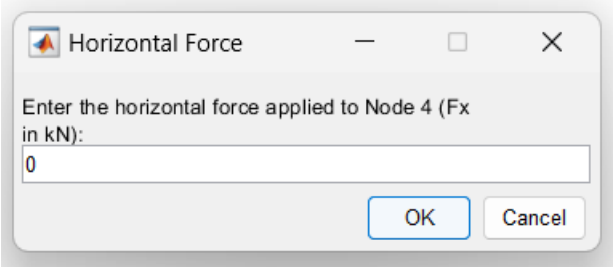


Figure 54: Input screen for entered the horizontal force applied to node 4.

Finally, the input section is over. Now, after all the boundary conditions entered by the user, the program calculates the nodal displacements and nodal forces of each node. The results of the program for our validation problem are shown in Figure 55 and Figure 56.

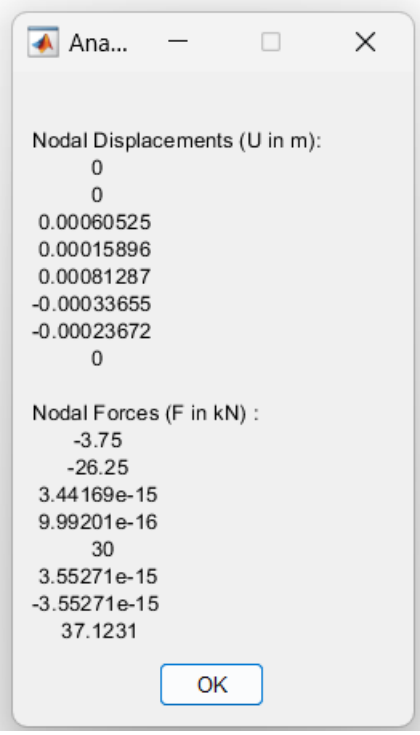


Figure 55: Analysis result message box.

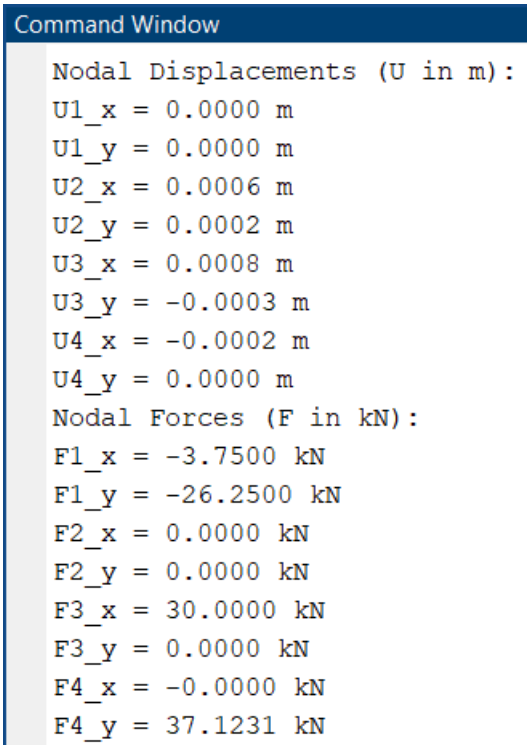


Figure 56: Results in command window on MATLAB.

These results perfectly matched our validation problem according to the referenced book. In Figure 57, you can see the displacement values from reference book solution for validation problem.

```
U =

  1.0e-003 *

         0
         0
    0.6053
    0.1590
    0.8129
   -0.3366

   -0.2367
         0
```

Figure 57: Nodal Displacement values from reference book solution manuel [4].

Also, the nodal forces shown in below figure from the book solution.

```
F =

   -3.7500
  -26.2500
         0
   -0.0000
   30.0000
    0.0000
         0
   37.1231
```

Figure 58: Nodal Force values from reference book solution manuel [4].

In Figure 55, since the F_x and F_y values of node 2, F_y value of node 3, and F_x value of node 4 are significantly small, the book I referenced accepted these values as 0.

For stress analysis in each element. The program gives results in command window as follows:

```
Stress on element 1 (sigma1): 3179.120179 kPa (Tensile)
Stress on element 2 (sigma2): -4142.542362 kPa (Compressive)
Stress on element 3 (sigma3): 5137.974406 kPa (Tensile)
Stress on element 4 (sigma4): -6966.627584 kPa (Compressive)
Stress on element 5 (sigma5): 3633.280204 kPa (Tensile)
Stress on element 6 (sigma6): -6731.059588 kPa (Compressive)
```

Figure 59: Program results for element stresses.

Reference book results for element stresses perfectly match the program results for element stresses. The reference book results for element stresses are given in the figures below.

```
» sigma1=PlaneTrussElementStress(E,L1,theta1,u1)

sigma1 =

  3.1791e+003
```

Figure 60: Result of reference book for stress value of element 1 [5].

```
» sigma2=PlaneTrussElementStress(E,L2,theta2,u2)

sigma2 =

 -4.1425e+003
```

Figure 61: Result of reference book for stress value of element 2 [5].

```
» sigma3=PlaneTrussElementStress(E,L3,theta3,u3)

sigma3 =

  5.1380e+003
```

Figure 62: Result of reference book for stress value of element 3 [5].

```
» sigma4=PlaneTrussElementStress(E,L4,theta4,u4)

sigma4 =

 -6.9666e+003
```

Figure 63: Result of reference book for stress value of element 4 [5].

```
» sigma5=PlaneTrussElementStress(E,L5,theta5,u5)

sigma5 =

  3.6333e+003
```

Figure 64: Result of reference book for stress value of element 5 [5].

```
» sigma6=PlaneTrussElementStress(E,L6,theta6,u6)

sigma6 =

 -6.7311e+003
```

Figure 65: Result of reference book for stress value of element 6 [5].

These values from the reference book are given in kilopascal (kPa) unit.

Another thing from the results section for my program is the geometry of the validation problem. The planar truss structure from Figure 18 is plotted and visualized by my program as shown in Figure 66.

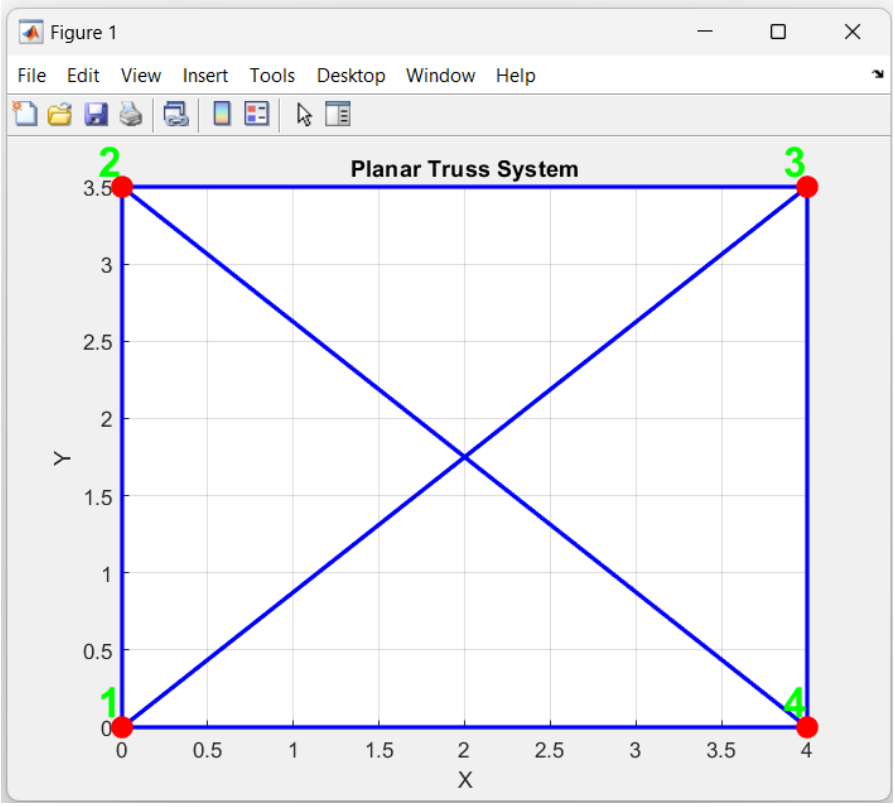


Figure 66: Validation problem structure plotted by my program.

The main purpose of having this geometry drawn by the program and displayed to the user is that the user can easily check the accuracy of the general inputs he has entered like node coordinates, number of elements and element connectivity information.

If we look at Figure 18 again, the geometry drawn by the program for the validation problem is correct. As we see in Figure 66, our planar truss structure consists of 4 nodes and 6 elements, which gives the same structure as Figure 18.

Last thing for my program's results section is stress visualization figure. In this figure there is a stress scale in kPa units on the right-hand side of the figure and there is again the geometry of truss structure drawn by program. The importance of this figure is elements in geometry are colored according to the true stresses applied on them. Most stress applied element is colorized to red, less stress applied element is colorized to

blue. The top color of the color scale corresponds to the highest stress value on the elements, which is element 4. The bottom color of the color scale corresponds to the element with the lowest stress value among the elements, which is element 1. The planar truss structure with stress visualization figure is shown below for validation problem.

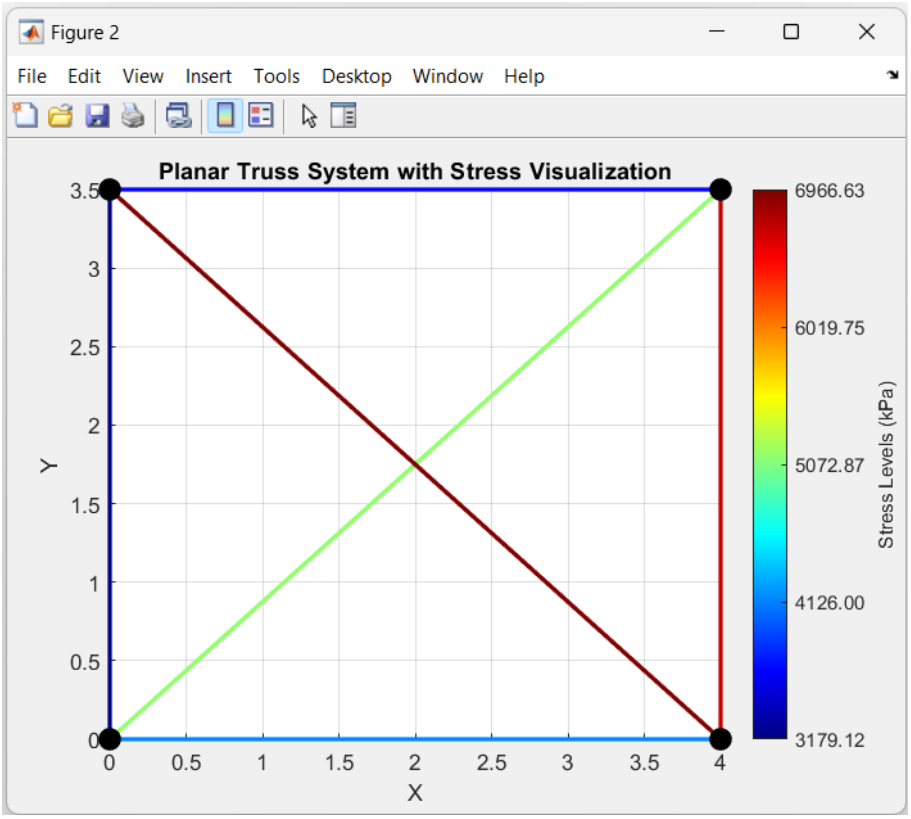


Figure 67: Planar truss structure with stress visualization for validation problem.

As you can see from Figure 60 to Figure 65, stress visualization and true stress values are perfectly matched to reference book.

5. CONCLUSION

In this report firstly I researched finite elements analysis and history of the finite elements analysis. While doing this research, I show the necessary sources in the reference section of my report. As a result of my research, I have carried out in-depth investigations of many structures such as spring elements, bar elements, planar truss systems, etc. My notes required while carrying out these investigations are shown in the project. I have studied these structures, especially mathematical models, in depth to gain a comprehensive understanding of their behavior and applications in finite element analysis.

The preparation for the next semester was thoroughly completed, building a strong theoretical base. With the foundations I acquired, I focused heavily on my subject, Planar Truss Elements. During the latter half of the course, I dedicated my efforts to studying Planar Truss Elements. For this segment, I developed user-friendly code in MATLAB. The code I created is versatile, capable of solving both simple and complex issues related to Planar Truss Elements.

The code I have written consists of a main script. I created 5 functions to help this main script to work. Thanks to some of these functions my code gives related diagrams for a better solution for the asked problem. So, users can be more satisfied with using the code. We can see and analyze the codes I wrote for solving any planar truss structure under the heading 3.

For this project, I developed a MATLAB code to perform the necessary planar truss analysis. Cost analysis is not applicable since this project involved writing MATLAB code for planar truss analysis.

In conclusion, the creation and testing of my code for analyzing planar truss structures has shown that it is both powerful and accurate. By carefully using finite element methods and MATLAB for simulation and analysis, I have built a reliable tool to study how planar truss structures behave under different loads. I used Peter I. Kattan's book "MATLAB Guide to Finite Elements: An Interactive Approach" as a reference. By examining the problem and solution in Example 5.2 from this book, I verified the accuracy of my code in the section titled "Validation Example," which is the third main

section of my report. As a result of the examinations I have made, the results in the book I have taken as a reference and the MATLAB program I have written have given the same results. Thus, it has been verified that the program works accurately and correctly. This approach ensured the code's robustness and reliability.

REFERENCES

[1] Rao, Singiresu S. *The finite element method in engineering*. Butterworth-heinemann, 2017.

[2] Das, Bibhuti Bhusan, et al., eds. *Recent Developments in Sustainable Infrastructure: Select Proceedings of ICRDSI 2019*. Springer, 2021.

[3] Hutton, David V. *Fundamentals of finite element analysis*. The McGraw Hill Companies, 2004.

[4] Logan, Daryl L. *A first course in the finite element method*. Vol. 4. Thomson, 2002.

[5] Kattan, Peter I. *MATLAB guide to finite elements: an interactive approach*. Springer Science & Business Media, 2003.