MARMARA UNIVERSITY

FACULTY OF ENGINEERING

DEVELOPMENT OF A COMPUTER PROGRAM FOR THE FINITE ELEMENT ANALYSIS AND DESIGN OF SPATIAL FRAME STRUCTURES

Emrecan Zengin

GRADUATION PROJECT REPORT

Department of Mechanical Engineering

Supervisor

Prof.Dr. Mustafa Özdemir

ISTANBUL, 2024

MARMARA UNIVERSITY

FACULTY OF ENGINEERING

DEVELOPMENT OF A COMPUTER PROGRAM FOR THE FINITE ELEMENT ANALYSIS AND DESIGN OF SPATIAL FRAME STRUCTURES

by

Emrecan Zengin

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

Signature of Author(s) ....Emrecan ZENGİN
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 Özdemir, for the valuable guidance and advice on preparing this thesis and giving me moral and material support.

May, 2024 **Emrecan Zengin**

CONTENTS

ABSTRACT

Development of a Computer Program for The Finite Element Analysis and Design of Spatial Frame Structures

This project is about creating a tool to help design and understand certain types of structures, like machines and buildings. We want to use a method called the Finite Element Method, which is a common way to solve design problems. Our focus is on structures with many angles, and we aim to make it easier for engineers and designers to work with them. The goal is to improve how we plan and design these structures, making the whole process simpler and more effective for everyone involved in.

ÖZET

Uzaysal Çerçeve Yapıların Sonlu Elemanlar Analizi ve Tasarımına Yönelik Bilgisayar Programının Geliştirilmesi

Bu proje, özellikle makineler ve binalar gibi belirli yapı türlerinin tasarlanması ve anlaşılması sürecine yardımcı olacak bir araç geliştirmeyi amaçlamaktadır. Bu hedef doğrultusunda, tasarım problemlerini çözmede yaygın olarak kullanılan Sonlu Elemanlar Yöntemi adlı metod bu projede başarıyla uygulanmıştır. Odak noktamız, çeşitli açılara sahip yapılar olup, mühendislerin ve tasarımcıların bu yapılarla çalışmasını kolaylaştırmayı hedeflemekteyiz. Amacımız, bu yapıları planlama ve tasarlama süreçlerini geliştirerek, bu süreçleri katılan herkes için daha basit ve etkili hale getirmektir.

SYMBOLS

f: Axial force

A: Cross-sectional area

$u \ and \ v$: Element displacement

$[k_e] \ or \ [k^{(e)}]$: Element stiffness matrix

$\left[K^{(e)}\right]$: Global stiffness matrix

L: Length

P: Loading

E: Modulus of elasticity

M: Moment

I: Moment of inertia

J: Polar moment of inertia

ρ: Radius of deflected curve

G: Shear modulus of elasticity

k: Spring stiffness

δ: Spring deformation

ε_x: Strain in axial direction

σ_x: Stress in axial direction

U: System displacement

T: Torque

$[R]$: Transformation matrix

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

LIST OF FIGURES

1. INTRODUCTION

The finite element method is a powerful computational technique employed in engineering to approximate solutions for boundary value problems. Basically, the real problem is replaced by a simpler one in computing the solution but when we do that, we can only get an approximate result, not the exact one. This is shown in Figure 1. [1] Most real-world problems are too complicated to solve exactly or even approximately using existing knowledge we have. Because of that we usually use the finite element method to find a solution. Additionally, in the finite element method, we can get better approximate results by doing better computer programs or doing better software to use the finite element method.

In the finite element method, we usually solve the problems which are boundary value problems. To solve these problems, we use boundary conditions of the field. Usually these boundaries are physical displacement, temperature, heat flux, and fluid velocity. [1]

The finite element method simplifies challenging problems by dividing the solution area into small connected parts called finite elements. To understand that we can consider complicated machines which are milling machines or fighter aircraft. We can use finite element method to obtain stresses when these machines are working. To do that we separate these machines into small parts. This is shown in Figure 2 and Figure 3. [2] The purpose of doing that is to get an approximate solution, especially when exact solutions are too hard or impossible to obtain.

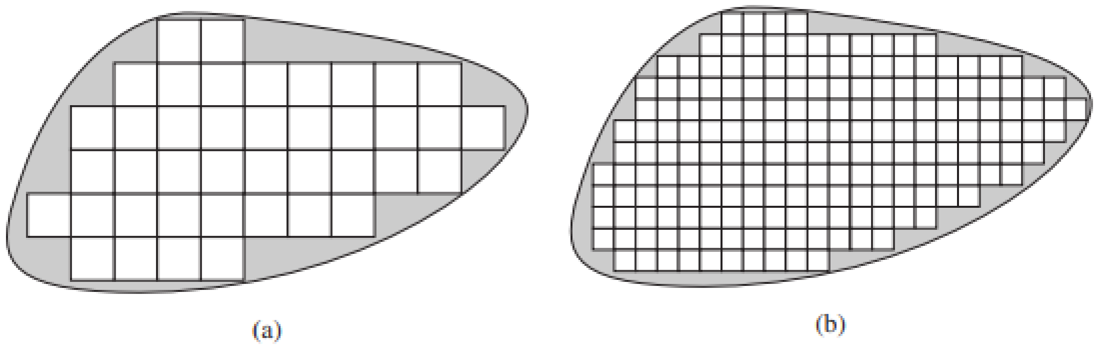


Figure 1: (a) Approximate approach. (b) More exact approach but still not exact one.[1]

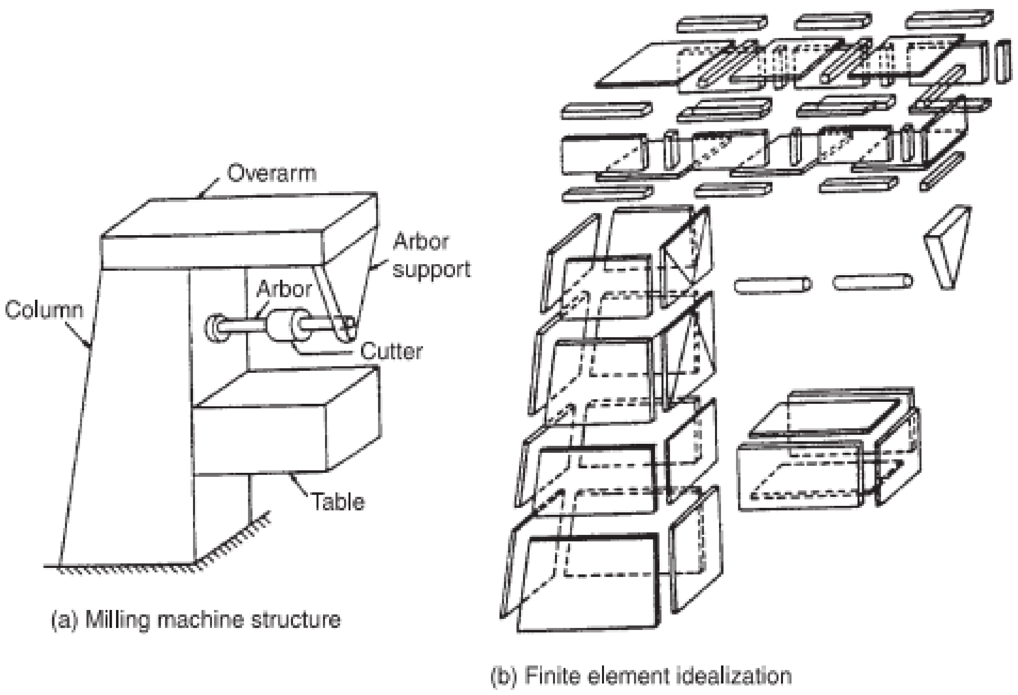

Overarm
Arbor support
Column
Arbor
Cutter
Table
(a) Milling machine structure
(b) Finite element idealization

Figure 2: Representation of a Milling Machine Structure by Finite Elements [2]

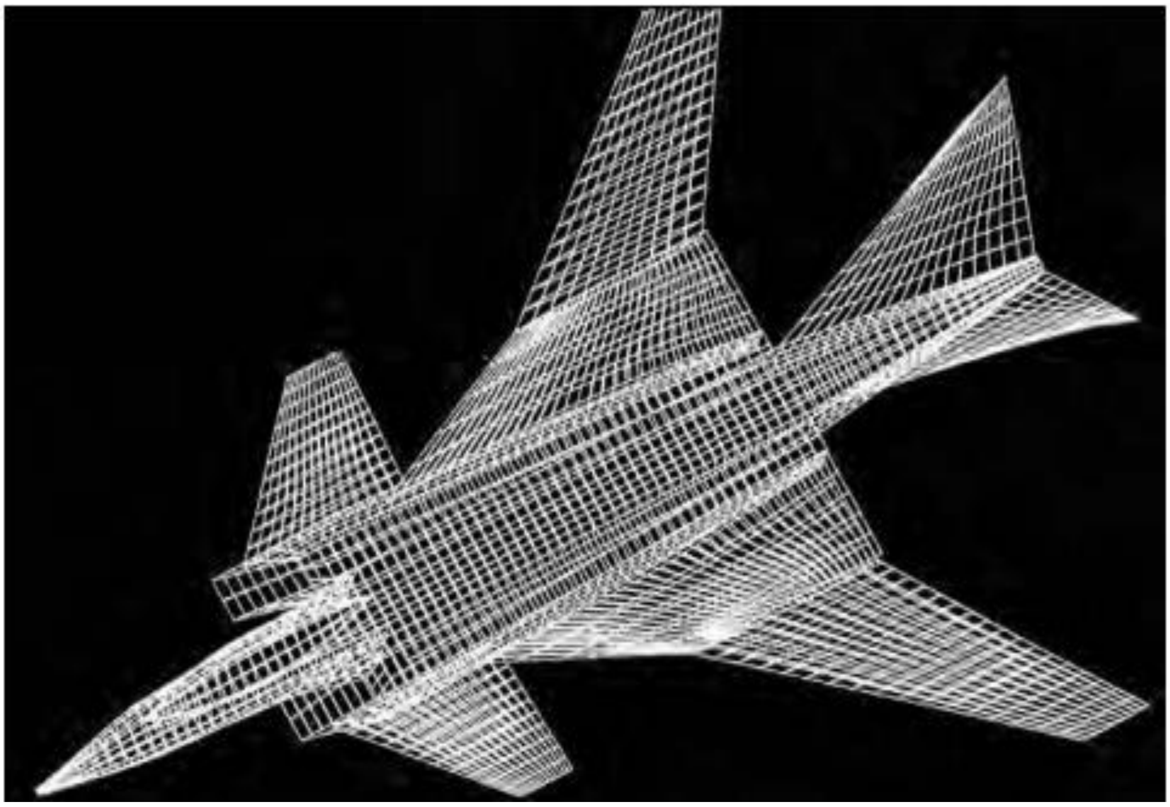


Figure 3: Finite Element Mesh of a Fighter Aircraft [2]

1.1 Brief History of The Finite Element Method

The beginnings of the finite element technique date back at least fifty years. Even before that, there were methods for solving numerical equations, and these go back much further. Messrs. Rayleigh and Ritz used what we call test functions, like guesses, to estimate solutions to numerical problems. Galerkin used the same idea to find solutions. However, compared to the modern finite element technique, there was a disadvantage to the older methods: the conjecture had to cover the entire problem area.

Things really began for the finite element technique in the 1940s when Courant introduced the concept of using piecewise solutions. Around the same time, aeronautical engineers were grappling with the invention of the jet engine and needed better ways to analyze aircraft structures for higher speeds. They came up with matrix methods (known as the flexibility method), where they treated the unknowns as forces and those known as displacements. The finite element technique, in its most common form, corresponds to the displacement method. This means that we are interested in knowing the displacements of the system when forces are applied.

The term "finite element" was first used in 1960 by Clough in the context of plate stress analysis and has been widely used since. During the 1960s and 1970s, the finite element technique expanded to address problems in plate bending, shell bending, pressure vessels, and other three-dimensional problems in structural analysis. It also addressed fluid flow and heat transfer. During this time, there were additional developments to address large displacements and dynamic analysis.

The finite element technique involves a lot of computational work because it deals with very large matrices. In its early days, mainframe computers were used. A significant development was the creation of NASTRAN in the 1960s, the first major software code for the finite element technique. Since then, many commercial software packages such as ANSYS, ALGOR, and COSMOS/M have been introduced for finite element analysis. Today, these packages can be used on regular computers to solve important problems in structural analysis, heat transfer, fluid flow, electromagnetism, and seismic response.

2. LINEAR SPRING AS A FINITE ELEMENT

A linear elastic spring is a mechanical device that can withstand axial loads only, and the stretching or compressing of the spring is directly related to the applied axial load. The factor that determines the relationship between deformation and load is known as the spring constant, spring rate, or spring stiffness (k), and it is measured in force per unit length. Since an elastic spring supports axial loading exclusively, we choose an element coordinate system (also called a local coordinate system) with an x-axis oriented along the length of the spring, as illustrated. [1]

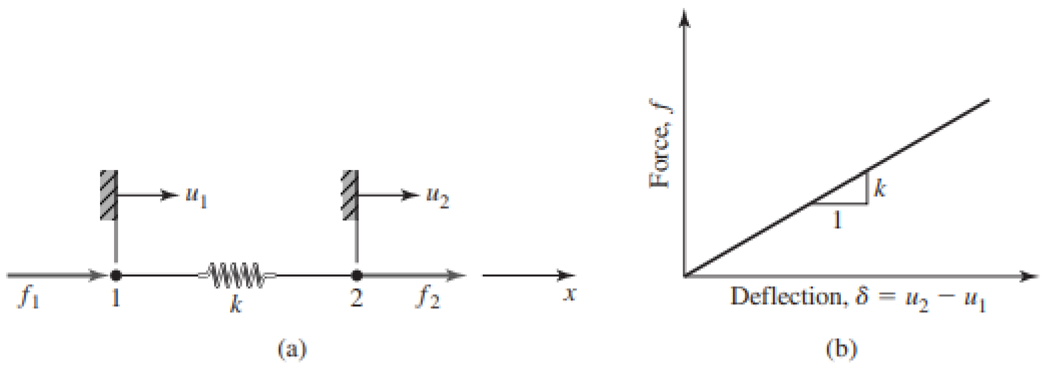


Figure 4: (a) Linear spring element with nodes, nodal displacements, and nodal forces. (b) Load-deflection curve [1]

In Figure 4 u_1 and u_2, f_1 and f_2 are in the positive sense.

Assuming that both the nodal displacements are zero when the spring is undeformed, the net spring deformation is given by

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

and the resultant axial force in the spring is

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

For equilibrium,

$$f_1 + f_2 = 0 \; or \; f_1 = -f_2 \tag{3}$$

Then, in terms of the applied nodal forces as

$$f_1 = -k(u_2 - u_1) \tag{4}$$

$$f_2 = k(u_2 - u_1) \tag{5}$$

which can be expressed in matrix form as

$$\begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} \tag{6}$$

$$[k_e]\{u\} = \{f\} \tag{7}$$

Where $[k_e]$ is the stiffness matrix for one spring element

$$[k_e] = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \tag{8}$$

Solution for the unknown nodal displacements

$$\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = [k_e]^{-1} \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} \tag{9}$$

Also, the matrix is singular and therefore not invertible. That is because the problem as defined is incomplete and does not have a solution: boundary conditions are required.

2.1 System of Two Springs

Derivation of the element stiffness matrix for a spring element was based on equilibrium conditions. We can use this same idea for a bunch of springs connected. To do this, we write down the balance equation for each connection point.

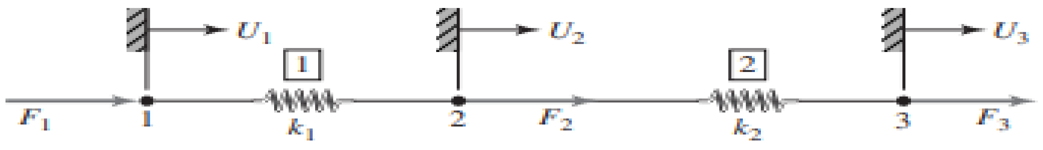


Figure 5: System of two springs with node numbers, element numbers, nodal displacements, and nodal forces [1]

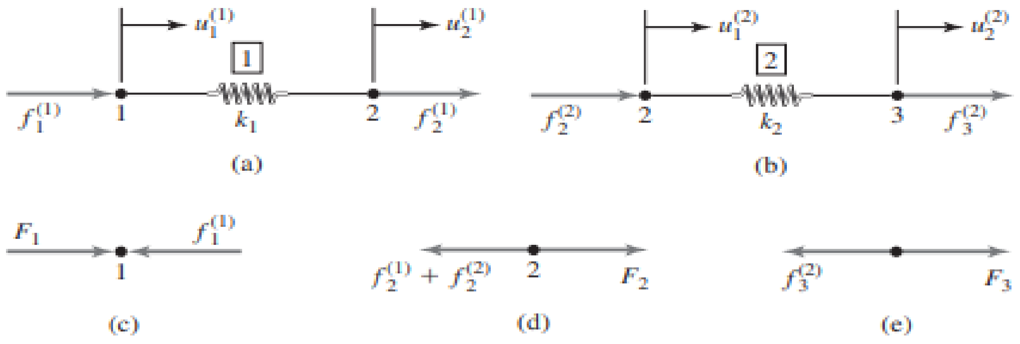


Figure 6: Free-body diagrams of elements and nodes for the two-element system on Figure 5 [1]

Writing the equations for each spring in matrix form

$$\begin{bmatrix} k_1 & -k_1 \\ -k_1 & k_1 \end{bmatrix} \begin{Bmatrix} u_1^{(1)} \\ u_2^{(1)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(1)} \\ f_2^{(1)} \end{Bmatrix} \tag{10}$$

$$\begin{bmatrix} k_2 & -k_2 \\ -k_2 & k_2 \end{bmatrix} \begin{Bmatrix} u_1^{(2)} \\ u_2^{(2)} \end{Bmatrix} = \begin{Bmatrix} f_2^{(2)} \\ f_3^{(2)} \end{Bmatrix} \tag{11}$$

To begin assembling the equilibrium equations describing the behavior of the system of two springs, the displacement compatibility conditions, which relate element displacements to system displacements, are written as:

$$u_1^{(1)} = U_1, \qquad u_2^{(1)} = U_2, \qquad u_1^{(2)} = U_2, \qquad u_2^{(2)} = U_3$$

And therefore:

$$\begin{bmatrix} k_1 & -k_1 \\ -k_1 & k_1 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \end{Bmatrix} = \begin{Bmatrix} f_1^{(1)} \\ f_2^{(1)} \end{Bmatrix} \tag{12}$$

$$\begin{bmatrix} k_2 & -k_2 \\ -k_2 & k_2 \end{bmatrix} \begin{Bmatrix} U_2 \\ U_3 \end{Bmatrix} = \begin{Bmatrix} f_2^{(2)} \\ f_3^{(2)} \end{Bmatrix} \tag{13}$$

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

Expand each equation in matrix form:

$$\begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ 0 \end{Bmatrix} = \begin{Bmatrix} f_1^{(1)} \\ f_2^{(1)} \\ 0 \end{Bmatrix} \tag{14}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \begin{Bmatrix} 0 \\ U_2 \\ U_3 \end{Bmatrix} = \begin{Bmatrix} 0 \\ f_2^{(2)} \\ f_3^{(2)} \end{Bmatrix} \tag{15}$$

Summing member by member:

$$\begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \end{Bmatrix} = \begin{Bmatrix} f_1^{(1)} \\ f_2^{(1)} + f_2^{(2)} \\ f_3^{(2)} \end{Bmatrix} \tag{16}$$

Next, we refer to the free-body diagrams of each of the three nodes:

$$f_1^{(1)} = F_1, \qquad f_2^{(1)} + f_2^{(2)} = F_2, \qquad f_3^{(2)} = F_3$$

Final form:

$$\begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ F_2 \\ F_3 \end{Bmatrix} \tag{17}$$

Where the stiffness matrix:

$$[k_e] = \begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \tag{18}$$

3.ELASTIC BAR ELEMENT

While the concept of the stiffness matrix is introduced using a linear elastic spring, its practical application in finite element analysis is somewhat limited. Springs are indeed utilized in machinery, and having a finite element representation of a linear spring can be beneficial in certain cases. The spring element is also commonly employed to model the elastic characteristics of supports in more complex systems. An element that is more broadly applicable, yet similar in nature, is an elastic bar experiencing axial forces only. This element, referred to as a bar element, proves particularly useful when analyzing both two- and three-dimensional frame or truss structures.

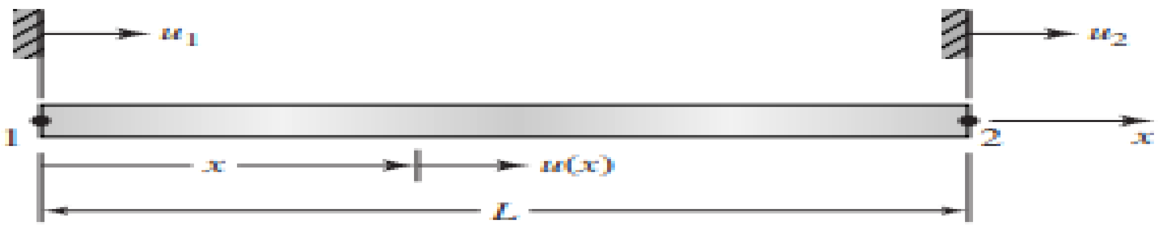


Figure 7: A bar element with element coordinate system and nodal displacement notation [1]

Bar element blending function is:

$$u(x) = N_1(x)u_1 + N_2(x)u_2 \tag{19}$$

Boundary Values:

$$u(x=0) = u_1, \;\; u(x=L) = u_2$$

$$N_1(0) = 1, \qquad N_2(0) = 0, \qquad N_1(L) = 0, \qquad N_2(L) = 1$$

Therefore, the functions are:

$$N_1(x) = 1 - \frac{x}{L}, \quad N_2(x) = \frac{x}{L} \tag{20}$$

$$u(x) = \left(1 - \frac{x}{L}\right)u_1 + \left(\frac{x}{L}\right)u_2 \tag{21}$$

In matrix form:

$$u(x) = [N_1(x) \quad N_2(x)]\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = [N]\{u\} \tag{22}$$

elementary strength of materials that the deflection of an elastic bar of length L and uniform cross-sectional area A when subjected to axial load P is given by

$$\delta = \frac{PL}{AE} \tag{23}$$

where E is the modulus of elasticity of the material. And the equivalent spring constant of an elastic bar as

$$k = \frac{P}{\delta} = \frac{AE}{L} \tag{24}$$

The normal strain component, defined as

$$\varepsilon_x = \frac{du}{dx} = \frac{u_2 - u_1}{L} \tag{25}$$

The axial stress, by Hooke's law:

$$\sigma_x = E\varepsilon_x = E\frac{u_2 - u_1}{L} \tag{26}$$

and the associated axial force is

$$P = \sigma_x A = \frac{AE}{L}(u_2 - u_1) \tag{27}$$

nodal force f_2 must be in the positive coordinate direction while nodal force f_1 must be equal and opposite for equilibrium; therefore,

$$f_1 = -\frac{AE}{L}(u_2 - u_1) \tag{28}$$

$$f_2 = \frac{AE}{L}(u_2 - u_1) \tag{29}$$

Expressed in matrix form as:

$$\frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} \tag{30}$$

Finally, the stiffness matrix for the bar element is given by

$$[k_e] = \frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{31}$$

4. BEAM ELEMENT

The beam element is a type of two-dimensional finite element where the local and global coordinates match up. It is defined by linear shape functions. The beam element possesses properties like modulus of elasticity (E), moment of inertia (I), and length (L). Each beam element comprises two nodes and is considered horizontal, as illustrated in Figure 8. The beam is of length L with axial local coordinate $\hat{x}$ and transverse local coordinate $\hat{y}$. The local transverse nodal displacements are given by $\widehat{d_{iy}}$ and the rotations by $\widehat{\phi_i}$. The local nodal forces are given by $\widehat{f_{iy}}$ and the bending moments by $\widehat{m_i}$'s as shown. We initially neglect all axial effects.

At all nodes, the following sign conventions are used:

1. Moments are positive in the counterclockwise direction.

2. Rotations are positive in the counterclockwise direction.

3. Forces are positive in the positive $\hat{y}$ direction.

4. Displacements are positive in the positive $\hat{y}$ direction.

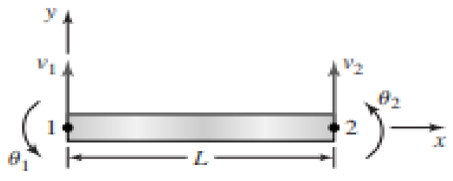


Figure 8: Beam element nodal displacements shown in a positive sense [1]

In figure 8 v_1 and v_2 are the same meaning with $\widehat{d_{iy}}$ and $\widehat{d_{2y}}$ in figure 9. Also, $\Theta_1$and Θ_2 are the same meaning with $\widehat{\phi_i}$ and $\widehat{\phi_2}$ in figure 9.

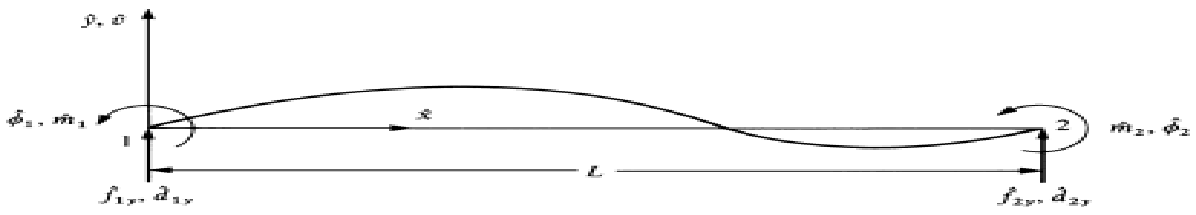
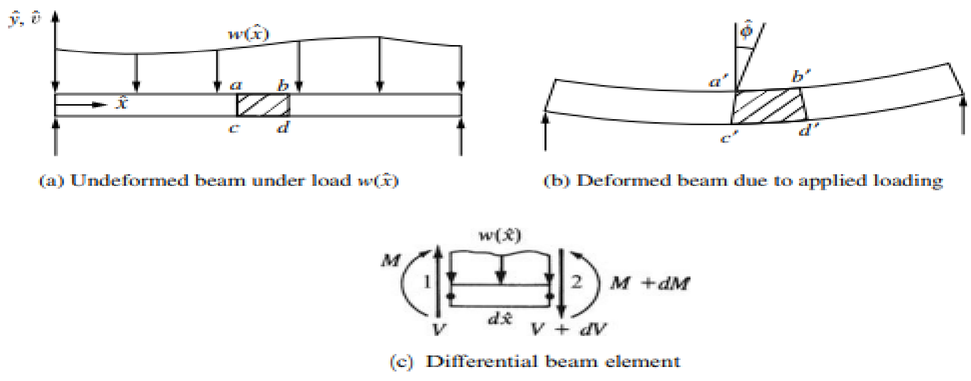


Figure 9: The Beam Element [3]



$\hat{y}, \hat{v}$
$w(\hat{x})$
$\hat{x}$
a
b
c
d
(a) Undeformed beam under load $w(\hat{x})$
$\hat{\phi}$
a'
b'
c'
d'
(b) Deformed beam due to applied loading
$w(\hat{x})$
M
1
2
$M + dM$
V
$d\hat{x}$
$V + dV$
(c) Differential beam element

Figure 10: Beam under distributed load [3]

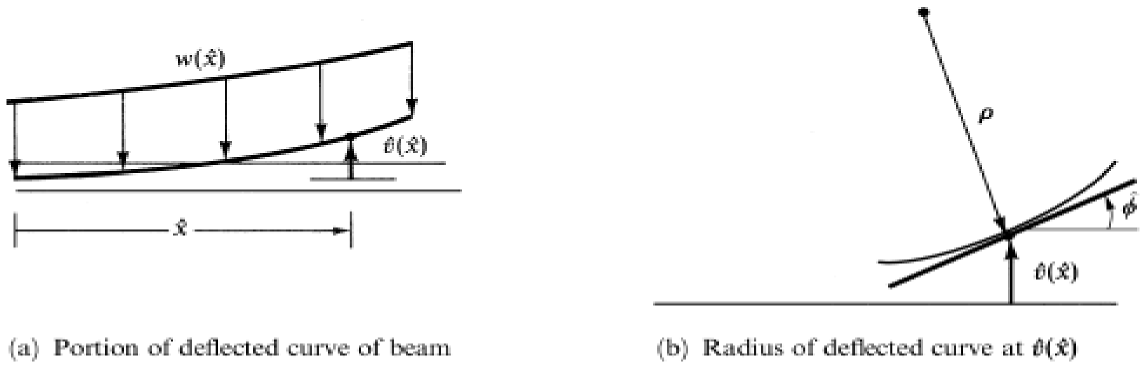
The differential equation is established through the analysis of a beam depicted in Figure 9, which is exposed to a distributed loading denoted as $w(\hat{x})$ (force per unit length). By applying force and moment equilibrium principles to a differential element of the beam, illustrated in Figure 9(c), the following relationship is obtained:

$$\sum F_y = 0 : V - (V + dV) - w(\hat{x})dx = 0 \tag{32}$$

Simplifying:

$$-wd\hat{x} - dV = 0 \;\; or \;\; w = -\frac{dV}{d\hat{x}} \tag{33}$$

$$\sum M_2 = 0: \; -Vdx + dM + w(\hat{x})d\hat{x}\left(\frac{d\hat{x}}{2}\right) = 0 \;\; or \;\; V = \frac{dM}{d\hat{x}} \tag{34}$$



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

Figure 11: Deflected curve of beam [3]

Also, the curvature of the beam, which is κ, related to the moment by

$$\kappa = \frac{1}{\rho} = \frac{M}{EI} \tag{35}$$

Where ρ is the radius of deflected curve shown in figure 10b, $\hat{v}$ 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 $\hat{z}$.

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

$$\kappa = \frac{d^2\hat{v}}{d\hat{x}^2} = \frac{M}{EI} \tag{36}$$

Solving the equation above for M

$$\frac{d^2}{d\hat{x}^2}\left(EI\frac{d^2\hat{v}}{d\hat{x}^2}\right) = -w(\hat{x}) \tag{37}$$

For constant EI and only nodal forces and moments the above equation becomes

$$EI\frac{d^4\hat{v}}{d\hat{x}^4} = 0 \tag{38}$$

Now we will follow the steps which are below to develop the stiffness matrix and equations for a beam element.

4.1 Step 1 Element Type

Represent the beam by labeling nodes at each end and in general by labeling the element number (Figure 8).

4.2 Step 2 Select a Displacement Function

Assume the transverse displacement variation through the element length to be:

$$\hat{v}(\hat{x}) = a_1\hat{x}^3 + a_2\hat{x}^2 + a_3\hat{x} + a_4 \tag{39}$$

There are four total degrees of freedom so the complete cubic displacement function which is above is appropriate. A transverse displacement $\widehat{d_{\iota y}}$ and a small rotation $\widehat{\phi_\iota}$ at each node are four total degrees of freedom.

We express $\hat{v}$ as a function of the nodal degrees of freedom $\widehat{d_{1y}}$, $\widehat{d_{2y}}$, $\widehat{\phi_1}$, $\widehat{\phi_2}$ in below:

$$\hat{v}(0) = \widehat{d_{1y}} = a_4 \tag{40}$$

$$\frac{d\hat{v}(0)}{d\hat{x}} = \widehat{\phi_1} = a_3 \tag{41}$$

$$\hat{v}(L) = \widehat{d_{2y}} = a_1L^3 + a_2L^2 + a_3L^3 + a_4 \tag{42}$$

$$\frac{d\hat{v}(L)}{d\hat{x}} = \widehat{\phi_2} = 3a_1L^2 + 2a_2L + a_3 \tag{43}$$

Where $\hat{\phi}$ =d$\hat{v}/d\hat{x}$ for the assumed small rotation $\hat{\phi}$. Solving the above equations a_1 through a_4 in terms of the nodal degrees of freedom and substituting into equation:

$$\hat{v}(\hat{x}) = a_1\hat{x}^3 + a_2\hat{x}^2 + a_3\hat{x}^3 + a_4 \tag{44}$$

$$\hat{v} = \left[\frac{2}{L^3}(\widehat{d_{1y}} - \widehat{d_{2y})} + \frac{1}{L^2}\left(\widehat{\phi_1} + \widehat{\phi_2}\right)\right]\hat{x}^3\#(45)$$

$$+\left[-\frac{3}{L^2}(\widehat{d_{1y}} - \widehat{d_{2y})} - \frac{1}{L}\left(2\widehat{\phi_1} + \widehat{\phi_2}\right)\right]\hat{x}^2 + \widehat{\phi_1}\hat{x} + \widehat{d_{\imath y}} \tag{45}$$

The above equation in matrix form:

$$\hat{v} = [N]\{\hat{d}\} \tag{46}$$

Where

$$\{\hat{d}\} = \begin{Bmatrix} \widehat{d_{1y}} \\ \widehat{\phi_1} \\ \widehat{d_{2y}} \\ \widehat{\phi_2} \end{Bmatrix} \tag{47}$$

And where

$$[N] = [N_1\ N_2\ N_3\ N_4] \tag{48}$$

And

$$N_1 = \frac{1}{L^3}(2\hat{x}^3 - 3\hat{x}^2L + L^3) \tag{49}$$

$$N_2 = \frac{1}{L^3}(\hat{x}^3L - 2\hat{x}^2L^2 + \hat{x}L^3) \tag{50}$$

$$N_3 = \frac{1}{L^3}(-2\hat{x}^3 + 3\hat{x}^2L) \tag{51}$$

$$N_4 = \frac{1}{L^3}(\hat{x}^3L - \hat{x}^2L^2) \tag{52}$$

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

4.3 Step 3 Define the Strain/Displacement and Stress/Strain Relationships

Assume the following axial strain/displacement relationship to be valid:

$$\varepsilon_x(\hat{x},\hat{y}) = \frac{d\hat{u}}{d\hat{x}} \tag{53}$$

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

$$\hat{u} = -\hat{y}\frac{d\hat{v}}{d\hat{x}} \tag{54}$$

The fundamental assumption is that cross sections of the beam, like cross section ABCD, which are initially planar before experiencing bending deformation, retain their planar configuration after deformation. In essence, a small angular rotation represented by the angle $(\frac{d\hat{v}}{d\hat{x}})$. Using above last two equations, we obtain:

$$\varepsilon_x(\hat{x},\hat{y}) = -\hat{y}\frac{d^2\hat{v}}{d\hat{x}^2} \tag{55}$$

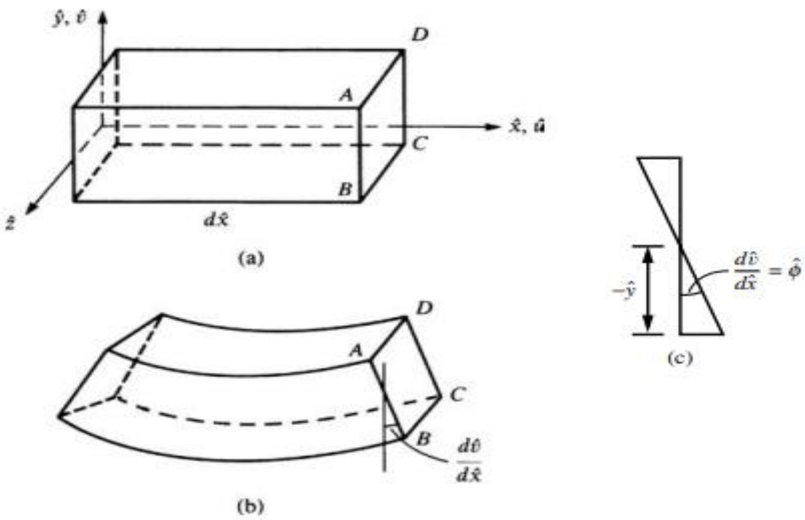


Figure 12: Beam segment (a) before deformation and (b) after deformation; (c) angle of rotation of cross section ABCD [3]

From elementary beam theory, the bending moment and shear force are:

$$\hat{m}(\hat{x}) = EI\frac{d^2\hat{v}}{d\hat{x}^2}\ and\ \hat{V} = EI\frac{d^3\hat{v}}{d\hat{x}^3} \tag{56}$$

4.4 Step 4 Derive the Element Stiffness Matrix and Equations

Begin by obtaining the element stiffness matrix and equations through a direct equilibrium approach. Next, establish the correlation between nodal and beam theory sign conventions for shear forces and bending moments, as illustrated in Figures 8 and 9. Along with above equations to obtain:

$$\widehat{f_{1y}} = \hat{V} = EI\frac{d^3\hat{v}(0)}{d\hat{x}^3} = \frac{EI}{L^3}\left(12\widehat{d_{1y}} + 6L\widehat{\phi_1} - 12\widehat{d_{2y}} + 6L\widehat{\phi_2}\right) \tag{57}$$

$$\widehat{m_1} = -\hat{m} = -EI\frac{d^2\hat{v}(0)}{d\hat{x}^2} = \frac{EI}{L^3}\left(6L\widehat{d_{1y}} + 4L^2\widehat{\phi_1} - 6L\widehat{d_{2y}} + 2L^2\widehat{\phi_2}\right) \tag{58}$$

$$\widehat{f_{2y}} = -\hat{V} = -EI\frac{d^3\hat{v}(L)}{d\hat{x}^3} = \frac{EI}{L^3}\left(-12\widehat{d_{1y}} - 6L\widehat{\phi_1} + 12\widehat{d_{2y}} - 6L\widehat{\phi_2}\right) \tag{59}$$

$$\widehat{m_2} = \hat{m} = EI\frac{d^2\hat{v}(L)}{d\hat{x}^2} = \frac{EI}{L^3}\left(\widehat{6Ld_{1y}} + 2L^2\widehat{\phi_1} - 6L\widehat{d_{2y}} + 4L^2\widehat{\phi_2}\right) \tag{60}$$

The above equations in matrix form:

$$\begin{Bmatrix} \widehat{f_{1y}} \\ \widehat{m_1} \\ \widehat{f_{2y}} \\ \widehat{m_2} \end{Bmatrix} = \frac{EI}{L^3}\begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix}\begin{Bmatrix} \widehat{d_{1y}} \\ \widehat{\phi_1} \\ \widehat{d_{2y}} \\ \widehat{\phi_2} \end{Bmatrix} \tag{61}$$

Where the stiffness matrix for beam element is then:

$$[k_e] = \frac{EI}{L^3}\begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix} \tag{62}$$

5. THE PLANE TRUSS ELEMENT

In this section we will use transformations to determine the global stiffness matrix for a bar element. Which will be the plane truss element. Bar element stiffness matrix to the plane truss element stiffness matrix we will use direct assembly procedure.

Recall the bar element equations in equation (30) as

$$\frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} \tag{63}$$

The equation (63) can be write as follows

$$\begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix}\begin{Bmatrix} u_1^{(e)} \\ u_2^{(e)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(e)} \\ f_2^{(e)} \end{Bmatrix} \tag{64}$$

Transforming above equations into the global coordinate system as

$$\left[K^{(e)}\right]\begin{Bmatrix} U_1{}^{(e)} \\ U_2{}^{(e)} \\ U_3{}^{(e)} \\ U_4{}^{(e)} \end{Bmatrix} = \begin{Bmatrix} F_1{}^{(e)} \\ F_2{}^{(e)} \\ F_3{}^{(e)} \\ F_4{}^{(e)} \end{Bmatrix} \tag{65}$$

For the plane truss element, we can think following figure

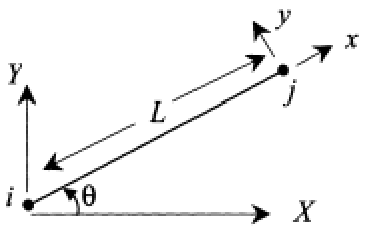


Figure 13: The plane truss element [5]

In equation (65) $\left[K^{(e)}\right]$ means the stiffness matrix in the global coordinate system, the vector $\left\{F^{(e)}\right\}$ include 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. Relationship element displacement coordinate system to the element displacements in global coordinates as follows

$$u_1{}^{(e)} = U_1{}^{(e)}\cos\theta + U_2{}^{(e)}\sin\theta \tag{66}$$

$$u_2{}^{(e)} = U_3{}^{(e)} \cos\theta + U_4{}^{(e)} \sin\theta \tag{67}$$

In matrix form as follows

$$\begin{Bmatrix} u_1^{(e)} \\ u_2^{(e)} \end{Bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \end{bmatrix} \begin{Bmatrix} U_1{}^{(e)} \\ U_2{}^{(e)} \\ U_3{}^{(e)} \\ U_4{}^{(e)} \end{Bmatrix} = [R] \begin{Bmatrix} U_1{}^{(e)} \\ U_2{}^{(e)} \\ U_3{}^{(e)} \\ U_4{}^{(e)} \end{Bmatrix} \tag{68}$$

Where

$$[R] = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \end{bmatrix} \tag{69}$$

Substituting equations (69) into equation (64)

$$\begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \end{bmatrix} \begin{Bmatrix} U_1{}^{(e)} \\ U_2{}^{(e)} \\ U_3{}^{(e)} \\ U_4{}^{(e)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(e)} \\ f_2^{(e)} \end{Bmatrix} \tag{70}$$

Or

$$\begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} [R] \begin{Bmatrix} U_1{}^{(e)} \\ U_2{}^{(e)} \\ U_3{}^{(e)} \\ U_4{}^{(e)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(e)} \\ f_2^{(e)} \end{Bmatrix} \tag{71}$$

After transforming equilibrium equations from element displacements to global displacements, which were initially expressed in the element coordinate system, multiplying the equations by $\cos\theta$ and $\sin\theta$ to obtain X and Y direction equilibrium equations in the global coordinate system. Same procedure done for node 2. If we multiply both side with $[R]^T$ that would be the same what we described before. $[R]^T$ is the transpose of the transformation matrix. As

$$[R]^T \begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} [R] \begin{Bmatrix} U_1{}^{(e)} \\ U_2{}^{(e)} \\ U_3{}^{(e)} \\ U_4{}^{(e)} \end{Bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & \cos\theta \\ 0 & \sin\theta \end{bmatrix} \begin{Bmatrix} f_1^{(e)} \\ f_2^{(e)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(e)} \cos\theta \\ f_1^{(e)} \sin\theta \\ f_2^{(e)} \cos\theta \\ f_2^{(e)} \sin\theta \end{Bmatrix} \tag{72}$$

Clearly, the right-hand side of the equation (72) shows the components of the element forces in the global coordinate system, as follows

$$[R]^T \begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} [R] \begin{Bmatrix} U_1{}^{(e)} \\ U_2{}^{(e)} \\ U_3{}^{(e)} \\ U_4{}^{(e)} \end{Bmatrix} = \begin{Bmatrix} F_1{}^{(e)} \\ F_2{}^{(e)} \\ F_3{}^{(e)} \\ F_4{}^{(e)} \end{Bmatrix} \tag{73}$$

If we compare equation (65) and (73), it can be seen as follows

$$\left[K^{(e)}\right] = [R]^T \begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} [R] \tag{74}$$

Doing multiplication in equation (74), we can get element stiffness matrix in global coordinate system. Since $c = \cos\theta\,, s = \sin\theta$ in below equation

$$\left[K^{(e)}\right] = \frac{AE}{L} \begin{bmatrix} c^2 & sc & -c^2 & -sc \\ sc & s^2 & -sc & -s^2 \\ -c^2 & -sc & c^2 & sc \\ -sc & -s^2 & sc & s^2 \end{bmatrix} \tag{75}$$

6. THE SPACE TRUSS ELEMENT

The space truss element is a three-dimensional finite element with both local and global coordinates [5]. To analyze the space truss element, we will use the bar element in 3-D coordinates.

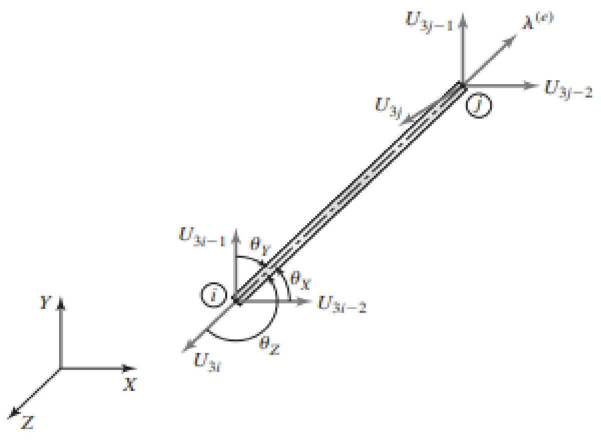


Figure 14: Bar element in a 3-D global coordinate system [1].

In Figure 13, exemplify a one-dimensional bar element attached to nodes i and j within a 3-D global reference frame, the unit vector along the element axis, referred to as the element reference frame, when expressed in the global coordinate system can be described as follows:

$$\lambda^{(e)} = \frac{1}{L}\left[(X_j - X_i)I + (Y_j - Y_i)J + (Z_j - Z_i)K\right] \tag{76}$$

or can be described as follows

$$\lambda^{(e)} = \cos\theta_x I + \cos\theta_y J + \cos\theta_z K \tag{77}$$

Therefore, the displacements of the elements are represented in components within the 3-D global system.

$$u_1^{(e)} = U_1^{(e)}\cos\theta_x + U_2^{(e)}\cos\theta_y + U_3^{(e)}\cos\theta_z \tag{78}$$

$$u_2^{(e)} = U_4^{(e)}\cos\theta_x + U_5^{(e)}\cos\theta_y + U_6^{(e)}\cos\theta_z \tag{79}$$

In equation (78) and (79), we adopt the notation n that element displacements 1 and 4 are in the global X direction, displacements 2 and 5 are in the global Y direction, and element displacements 3 and 6 are in the global Z direction.

We do same procedure with equation (78) and (79), like in equation (68)

$$\begin{Bmatrix} u_1^{(e)} \\ u_2^{(e)} \end{Bmatrix} = \begin{bmatrix} \cos\Theta_x & \cos\Theta_y & \cos\Theta_z & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\Theta_x & \cos\Theta_y & \cos\Theta_z \end{bmatrix} \begin{bmatrix} {U_1}^{(e)} \\ {U_2}^{(e)} \\ {U_3}^{(e)} \\ {U_4}^{(e)} \\ {U_5}^{(e)} \\ {U_6}^{(e)} \end{bmatrix} = [R]\{U^{(e)}\} \quad (80)$$

The same procedure in equation (74) will be done, where $[R]$ is the transformation matrix.

$$[K^{(e)}] = [R]^T \begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} [R] \qquad (81)$$

Multiplying matrices in equations (81) gives

$$[K^{(e)}] = k_e \begin{bmatrix} {c_x}^2 & c_xc_y & c_xc_z & -{c_x}^2 & -c_xc_y & c_xc_z \\ c_xc_y & {c_y}^2 & c_yc_z & -c_xc_x & -{c_y}^2 & -c_yc_z \\ c_xc_z & c_yc_z & -{c_z}^2 & -c_xc_z & -c_yc_z & -{c_z}^2 \\ -{c_x}^2 & -c_xc_x & -c_xc_z & {c_x}^2 & c_xc_y & c_xc_z \\ -c_xc_y & -{c_y}^2 & -c_yc_z & c_xc_y & {c_y}^2 & c_yc_z \\ -c_xc_z & -c_yc_z & -{c_z}^2 & c_xc_z & c_yc_z & {c_z}^2 \end{bmatrix} \qquad (82)$$

Where

$$k_e = \frac{AE}{L}, c_x = \cos\Theta_x\,, c_y = \cos\Theta_y\,, c_z = \cos\Theta_z \qquad (83)$$

7.THE PLANE FRAME ELEMENT

The plane frame element is a two-dimensional finite element with both local and global coordinates [5].

To determine the stiffness matrix for plane frame element we will use figure 7 and figure 8. So if we combine degrees of freedoms in figure 7 and figure 8 we can get six degrees of which are below

$$\begin{Bmatrix} u_1 \\ v_1 \\ \theta_1 \\ u_2 \\ v_2 \\ \theta_2 \end{Bmatrix} \tag{84}$$

Six degrees of freedom means we generated six by six matrix for the stiffness matrix for plane frame element. To do that we can use equations (31) and (62). If we are numbering these matrices according to the order of the terms in equation (84) it will be like that

$$\begin{array}{lcc} & \boxed{\begin{matrix}1 & \quad 4\end{matrix}} & \\ & \begin{matrix}\downarrow & \quad \downarrow\end{matrix} & \\ [k_e] = \frac{AE}{L} & \begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix} & \begin{matrix}\leftarrow\\ \leftarrow\end{matrix}\ \boxed{\begin{matrix}1\\4\end{matrix}} \end{array}$$

$$\begin{array}{lcc} & \boxed{\begin{matrix}2 & 3 & 5 & 6\end{matrix}} & \\ & \begin{matrix}\downarrow & \downarrow & \downarrow & \downarrow\end{matrix} & \\ [k_e] = \frac{EI}{L^3} & \begin{bmatrix}12 & 6L & -12 & 6L\\ 6L & 4L^2 & -6L & 2L^2\\ -12 & -6L & 12 & -6L\\ 6L & 2L^2 & -6L & 4L^2\end{bmatrix} & \begin{matrix}\leftarrow\\ \leftarrow\\ \leftarrow\\ \leftarrow\end{matrix}\ \boxed{\begin{matrix}2\\3\\5\\6\end{matrix}} \end{array}$$

So, if we insert these matrices according to their yields into six-by-six matrix for the stiffness matrix for plane frame element

$$[k^{(e)}] = \begin{bmatrix} \frac{AE}{L} & 0 & 0 & \frac{-AE}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & \frac{-12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & \frac{-6EI}{L^2} & \frac{2EI}{L} \\ \frac{-AE}{L} & 0 & 0 & \frac{AE}{L} & 0 & 0 \\ 0 & \frac{-12EI}{L^3} & \frac{-6EI}{L^2} & 0 & \frac{12EI}{L^3} & \frac{-6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & \frac{-6EI}{L^2} & \frac{4EI}{L} \end{bmatrix} \tag{85}$$

If we want global stiffness matrix for the plane truss element can follow the steps below

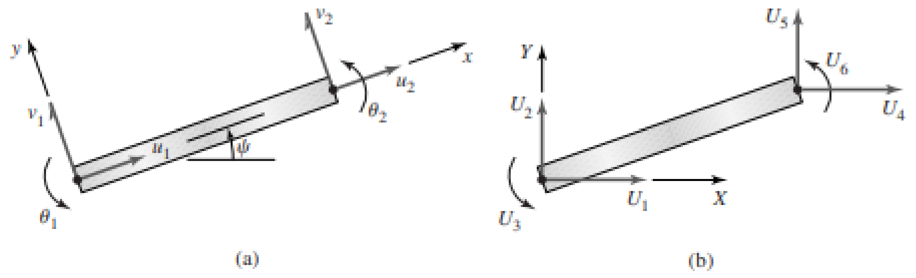


Figure 15: (a) Nodal displacements in the element coordinate system. (b) Nodal displacements in the global coordinate system [1]

In figure 15 (a) the angle Ψ is taking symbol θ in above and below equations in heading 7.

Using figure 15, we can write global displacements with element displacements.

$$\begin{aligned} u_1 &= U_1 \cos\theta + U_2 \sin\theta \\ v_1 &= -U_1 \sin\theta + U_2 \cos\theta \\ \theta_1 &= U_3 \\ u_2 &= U_4 \cos\theta + U_5 \sin\theta \\ v_2 &= -U_4 \sin\theta + U_5 \cos\theta \\ \theta_2 &= U_6 \end{aligned} \tag{86}$$

Also, equations (86) can be written in matrix form as

$$\begin{Bmatrix} u_1 \\ v_1 \\ \theta_1 \\ u_2 \\ v_2 \\ \theta_2 \end{Bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\theta & \sin\theta & 0 \\ 0 & 0 & 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \end{Bmatrix} = [R][U] \qquad (87)$$

Where $[R]$ is the transformation matrix. So, like section 5 and section 6 we will do same procedure to determine the global stiffness matrix.

$$\left[K^{(e)}\right] = [R]^T\left[k^{(e)}\right][R] \qquad (88)$$

8. THE SPACE FRAME ELEMENT

The three-dimensional finite element known as the space frame element employs both local and global coordinates. It possesses material properties such as modulus of elasticity (E), shear modulus of elasticity (G), cross-sectional area (A), moments of inertia (I_x and I_y), polar moment of inertia (J), and length (L) [5].

To determine the stiffness matrix of the space frame element, firstly we expand the beam-axial component discussed in the previous sections to add two-plane bending. Subsequently, we introduce torsional capability.

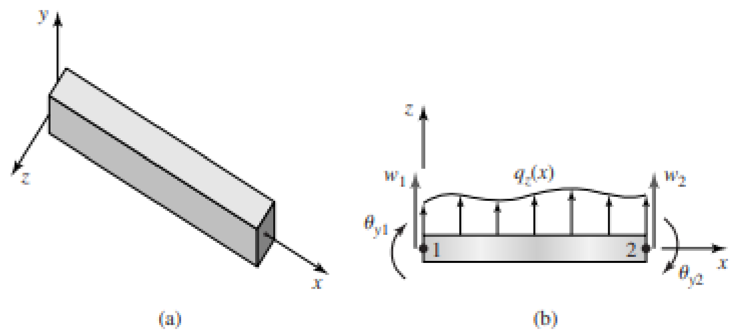


Figure 16: (a) Three-dimensional beam element. (b) Nodal displacements in element *xz* plane [1]

It can be shown in figure 16 the beam element has bending about the *z* axis and bending about the *y* axis. The nodal displacements in the *z* direction given v_1 and v_2, rotations given θ_{z1} and θ_{z2}. The nodal displacements in the *y* direction given w_1 and w_2, rotations given θ_{y1} and θ_{y2}.

Element stiffness matrix for bending about the *z* axis:

$$[k_e]_{xy} = \frac{EI_z}{L^3}\begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix} \tag{89}$$

Element stiffness matrix for bending about the y axis:

$$[k_e]_{xz} = \frac{EI_y}{L^3}\begin{bmatrix} 12 & -6L & -12 & -6L \\ -6L & 4L^2 & 6L & 2L^2 \\ -12 & 6L & 12 & 6L \\ -6L & 2L^2 & 6L & 4L^2 \end{bmatrix} \tag{90}$$

Also $[k_e]_{axial}$ introduced in section 7 and given:

$$[k_e]_{axial} = \frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{91}$$

Further, we need to add the torsion to the beam element to determine the element stiffness matrix. Our calculation will be based on figure 17. In Figure 17 the given directions are taken positively for below equations.

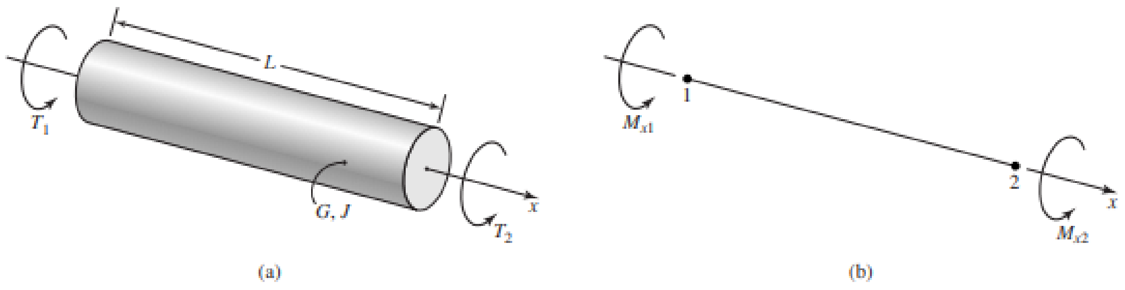


Figure 17: (a) Circular cylinder subjected to torsion. (b) Torsional finite element notation [1]

In basic strength of materials, it is commonly given that the angle of twist per unit length for a uniform, elastic circular cylinder under the given in Figure 17 of torque T is expressed as:

$$\phi = \frac{T}{JG} \tag{91}$$

where J is polar moment of inertia of the cross-sectional area and G is the shear modulus of the material. The total angle of twist given:

$$\theta_{x2} - \theta_{x1} = \frac{TL}{JG} \tag{92}$$

Torque given

$$T = \frac{JG}{L}(\theta_{x2} - \theta_{x1}) = k_T(\theta_{x2} - \theta_{x1}) \tag{93}$$

Also, equation (93) can be given for the equilibrium condition $M_{x1} + M_{x2} = 0$

$$\frac{JG}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{Bmatrix} \theta_{x1} \\ \theta_{x2} \end{Bmatrix} = \begin{Bmatrix} M_{x1} \\ M_{x2} \end{Bmatrix} \tag{94}$$

The torsional stiffness matrix is

$$[k_e]_{torsion} = \frac{JG}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{95}$$

Finally, we add to torsional characteristics to the general beam element

$$\begin{bmatrix} [k_e]_{axial} & [0] & [0] & [0] \\ [0] & [k_e]_{xy} & [0] & [0] \\ [0] & [0] & [k_e]_{xz} & [0] \\ [0] & [0] & [0] & [k_e]_{torsion} \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ \theta_{z1} \\ w_1 \\ \theta_{y1} \\ \theta_{x1} \\ u_2 \\ v_2 \\ \theta_{z2} \\ w_2 \\ \theta_{y2} \\ \theta_{x2} \end{Bmatrix} = \begin{Bmatrix} f_{x1} \\ f_{y1} \\ M_{z2} \\ f_{z1} \\ M_{y1} \\ M_{x1} \\ f_{x2} \\ f_{y2} \\ M_{z2} \\ f_{z2} \\ M_{y2} \\ M_{x2} \end{Bmatrix} \tag{96}$$

$$\left[k^{(e)}\right]=\begin{bmatrix} \frac{EA}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{EA}{L} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} & 0 & -\frac{12EI_y}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} \\ 0 & 0 & \frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 & 0 & 0 & -\frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 \\ 0 & 0 & 0 & \frac{GJ}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{GJ}{L} & 0 & 0 \\ 0 & 0 & -\frac{6EI_y}{L^2} & 0 & \frac{4EI_y}{L} & 0 & 0 & 0 & \frac{6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & 0 \\ 0 & \frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{4EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{2EI_z}{L} \\ -\frac{EA}{L} & 0 & 0 & 0 & 0 & 0 & \frac{EA}{L} & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{12EI_z}{L^3} & 0 & 0 & 0 & -\frac{6EI_z}{L^2} & 0 & \frac{12EI_z}{L^3} & 0 & 0 & 0 & -\frac{6EI_z}{L^2} \\ 0 & 0 & -\frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 & 0 & 0 & \frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 \\ 0 & 0 & 0 & -\frac{GJ}{L} & 0 & 0 & 0 & 0 & 0 & \frac{GJ}{L} & 0 & 0 \\ 0 & 0 & -\frac{6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & 0 & 0 & 0 & \frac{6EI_y}{L^2} & 0 & \frac{4EI_y}{L} & 0 \\ 0 & \frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{2EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{4EI_z}{L} \end{bmatrix} \tag{97}$$

After we described element stiffness matrix of the space frame element, we can perform the global stiffness matrix for the space frame element. To do that we follow the same procedure which is we use rotation matrix and then we multiply the element stiffness matrix with rotation matrix's transpose on the right-hand side and multiply with rotation matrix on the left-hand side.

$$[K^{(e)}] = [R]^T[k^{(e)}][R] \tag{98}$$

Where the rotation matrix [R] is given by

$$[R] = \begin{bmatrix} [r] & 0 & 0 & 0 \\ 0 & [r] & 0 & 0 \\ 0 & 0 & [r] & 0 \\ 0 & 0 & 0 & [r] \end{bmatrix} \tag{99}$$

Where [r] is the 3 X 3 matrix of direction cosines given as follows

$$[r] = \begin{bmatrix} C_{Xx} & C_{Yx} & C_{Zx} \\ C_{Xy} & C_{Yy} & C_{Zy} \\ C_{Xz} & C_{Yz} & C_{Zz} \end{bmatrix} \tag{100}$$

Where $C_{Xx} = \cos\theta_{Xx}$, $C_{Yx} = \cos\theta_{Yx}$,……., etc.

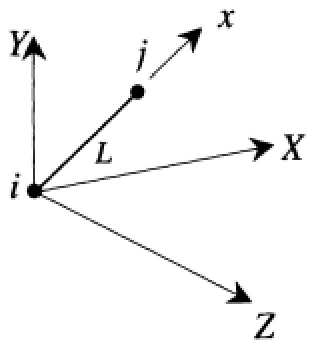


Figure 18: The Space Frame Element [5]

It can be also shown like where L is the length

$$C_{Xx} = \frac{x_2 - x_1}{L} \tag{101}$$

$$C_{Yx} = \frac{y_2 - y_1}{L} \tag{102}$$

$$C_{Zx} = \frac{z_2 - z_1}{L} \tag{103}$$

$$D = \sqrt{{C_{Xx}}^2 + {C_{Yx}}^2} \tag{104}$$

$$C_{Xy} = \frac{-C_{Yx}}{D} \tag{105}$$

$$C_{Yy} = \frac{C_{Xx}}{D} \tag{106}$$

$$C_{Zy} = 0 \tag{107}$$

$$C_{Xz} = \frac{-C_{Xx} * C_{Zx}}{D} \tag{108}$$

$$C_{Yz} = \frac{-C_{Yx} * C_{Zx}}{D} \tag{109}$$

$$C_{Zz} = D \tag{110}$$

9. PROGRAM DESINGN PROCEDURE

9.1 MATLAB

MATLAB is a programming platform designed specifically for engineers and scientists to analyze and design systems and products that transform our world. The heart of MATLAB is the MATLAB language, a matrix-based language allowing the most natural expression of computational mathematics [6].

9.2 Prompting for Number of Elements and Nodes

My code starts by asking the user to input the number of elements and nodes in the system. These inputs will be used later to all my code. You can see the first part of my code below. Explanation of first part will be given below of my code.

```
% Initialize num_elements and num_nodes to zero initially

num_elements = 0;

num_nodes = 0;

while num_elements <= 0 || num_nodes <= 0

    % Prompt the user for the number of elements and nodes

    prompt = {'Number of elements:', 'Number of nodes:'};

    dlgtitle = 'Input';

    dims = [1 50];

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

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


    % Convert user inputs to numerical values

    num_elements = str2double(userInput{1});

    num_nodes = str2double(userInput{2});
    % Check if any input is invalid (zero or negative)

    if num_elements <= 0 || num_nodes <= 0

        % Display error message in a dialog box

        errMsg = 'Number of elements and nodes must be greater than zero.';

        errTitle = 'Invalid Input';

        uiwait(msgbox(errMsg, errTitle, 'error'));

    end

end
```

The code begins by initializing two variables, **num_elements** and **num_nodes**, to zero. These variables will store the number of elements and nodes in the space frame structure.

Next, a **while** loop is used to repeatedly prompt the user for the number of elements and nodes until valid (positive) inputs are provided. The condition **while num_elements <= 0 || num_nodes <= 0** ensures that the loop continues running as long as either **num_elements** or **num_nodes** is zero or negative. Within the loop, a prompt dialog box is displayed to the user, asking for the number of elements and nodes. This is achieved using the **inputdlg** function. The **prompt** variable contains the messages to be displayed, **dlgtitle** sets the title of the dialog box, **dims** specify the dimensions of the input fields, and **defaultInput** provides default values (empty in this case). The user inputs are then retrieved from the dialog box and converted from strings to numerical values using the **str2double** function. These converted values are stored in the **num_elements** and **num_nodes** variables. After converting the inputs, the code checks if either **num_elements** or **num_nodes** is zero or negative. If this condition is met, an error message is displayed in a message box using the **msgbox** function, informing the user that the number of elements and nodes must be greater than zero. The **uiwait** function is used to pause the execution of the code until the user dismisses the message box.

Basically, this part of the code ensures that the user enters valid numbers for the elements and nodes required for further calculations in the space frame structure analysis.

9.3 Collecting Node Coordinates

```
% Initialize variables to store node coordinates
node_coordinates = zeros(num_nodes, 3);
% Loop through each node to collect coordinates
for i = 1:num_nodes
    prompt = {sprintf('Enter x coordinate for Node %d:', i), sprintf('Enter y coordinate for Node %d:', i), sprintf('Enter z 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});
    z = str2double(nodeCoord{3});
    % Store coordinates in the matrix
    node_coordinates(i, :) = [x, y, z];
end
```

In this part of the code, I was initializing variables to store the coordinates of each node in the space frame structure. First, I created a matrix **node_coordinates** with dimensions corresponding to the number of nodes and three columns, one for each coordinate (x, y, and z). I used the **zeros** function to initialize this matrix with zeros.

Next, I looped through each node to collect its coordinates from the user. For each iteration, I created a prompt asking for the x, y, and z coordinates of the current node. The prompt was displayed in a dialog box with a title indicating which node's coordinates were being entered. Default values of '0' were provided in the input dialog to facilitate quick entry if needed.

After displaying the prompt, I converted the user's input from strings to numerical values using **str2double**. Finally, I stored these numerical coordinates in the corresponding row of the **node_coordinates** matrix. This ensured that all the nodes' coordinates were systematically collected and stored for later use in the analysis.

9.4 Computing Element Stiffness Matrices

```
% Initialize connections matrix

connections = zeros(num_elements, 2);

% Loop through each element to compute stiffness matrices

for i = 1:num_elements

    % Keep asking for input until valid values are provided

    while true

        prompt = {'First node of the element:', 'Second node of the element:', 'Modulus of elasticity (kPa):', 'Shear modulus of elasticity (kPa):', 'Cross sectional Area (m^2):', 'Moment of inertia about the y axis (m^4):', 'Moment of inertia about the z axis (m^4):', 'Torsional constant (m^4):'};

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

        dims = [1 50];

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

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

        % Convert user inputs to numerical values

        node1 = str2double(elementProps{1});

        node2 = str2double(elementProps{2});

        E = str2double(elementProps{3});

        G = str2double(elementProps{4});

        A = str2double(elementProps{5});

        Iy = str2double(elementProps{6});
        Iz = str2double(elementProps{7});
        J = str2double(elementProps{8});
        % Check if any input is invalid (zero or negative)
        if any([node1, node2, E, G, A, Iy, Iz, J] <= 0) || node1 > num_nodes || node2 > 
num_nodes
            % Display error message in a dialog box
            errMsg = sprintf('All inputs must be greater than zero and node1, node2 numbers must 
be less than or equal to %d which is the total number of nodes.', num_nodes);
            errTitle = 'Invalid Input';
            errDlg = errordlg(errMsg, errTitle);
            uiwait(errDlg); % Wait for the user to close the error dialog
        else
            % Inputs are valid, exit the loop
            break;
        end
    end
    % Continue with further computations for the current element
    % Store node connections
    connections(i, :) = [node1, node2];
    % Extract coordinates of the nodes for the current element
    x1 = node_coordinates(node1, 1);
    y1 = node_coordinates(node1, 2);
    z1 = node_coordinates(node1, 3);
    x2 = node_coordinates(node2, 1);
    y2 = node_coordinates(node2, 2);
    z2 = node_coordinates(node2, 3);
% Compute element stiffness matrix
    element_stiffness_matrices{i} = SpaceFrameElementStiffness(E, G, A, Iy, Iz, J, x1, y1, z1,
x2, y2, z2);
    % Display element stiffness matrix
    disp(['Element Stiffness Matrix for Element ', num2str(i), ':']);
    disp(element_stiffness_matrices{i});
end
```

In this part of the code, I was initializing the connections matrix and computing stiffness matrices for each element in the space frame structure. Here's a detailed explanation of how I approached this task:

First, I initialized the **connections** matrix with dimensions corresponding to the number of elements and two columns, representing the two nodes that each element connects. I used the **zeros** function to initialize this matrix with zeros.

Next, I looped through each element to gather its properties from the user and compute its stiffness matrix. For each element, I prompted the user for input via a dialog box. The dialog box asked for the first and second nodes of the element, modulus of elasticity (E), shear modulus (G), cross-sectional area (A), moments of inertia about the y-axis (Iy) and z-axis (Iz), and the torsional constant (J). I displayed the prompt in a dialog box titled with the current element number and provided empty strings as default values.

Then, I converted the user inputs from strings to numerical values using **str2double**. I included a while loop to ensure that valid inputs were provided. If any input was invalid (zero, negative, or node numbers exceeding the total number of nodes), an error message was displayed in a dialog box. The loop continued to prompt the user until valid values were entered.

Once valid inputs were obtained, I stored the node connections for the current element in the **connections** matrix. I also extracted the coordinates of the nodes for the current element from the **node_coordinates** matrix.

To compute the element stiffness matrix, I called the **SpaceFrameElementStiffness** function with the appropriate parameters (E, G, A, Iy, Iz, J, and the node coordinates). That function script can be seen in the next code and the explanations for that function script given below the code. I stored the resulting stiffness matrix in the **element_stiffness_matrices** cell array and displayed it using the **disp** function.

By structuring the code in this way, I ensured that all necessary properties for each element were accurately collected, validated, and used to compute the corresponding stiffness matrices. This approach facilitated a systematic and error-checked process for preparing the space frame structure for further analysis.

```
function y = SpaceFrameElementStiffness(E,G,A,Iy,Iz,J,x1,y1,z1,x2,y2,z2)

L = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1) + (z2-z1)*(z2-z1));

w1 = E*A/L ;

w2 = 12*E*Iz/(L*L*L);

w3 = 6*E*Iz/(L*L);

w4 = 4*E*Iz/L;

w5 = 2*E*Iz/L;

w6 = 12*E*Iy/(L*L*L);

w7 = 6*E*Iy/(L*L);

w8 = 4*E*Iy/L;

w9 = 2*E*Iy/L;

w10 = G*J/L;

kprime = [w1 0 0 0 0 0 -w1 0 0 0 0 0;

0 w2 0 0 0 w3 0 -w2 0 0 0 w3;

0 0 w6 0 -w7 0 0 0 -w6 0 -w7 0;

0 0 0 w10 0 0 0 0 0 -w10 0 0;

0 0 -w7 0 w8 0 0 0 w7 0 w9 0;
0 w3 0 0 0 w4 0 -w3 0 0 0 w5;
-w1 0 0 0 0 0 w1 0 0 0 0 0;
0 -w2 0 0 0 -w3 0 w2 0 0 0 -w3;
0 0 -w6 0 w7 0 0 0 w6 0 w7 0;
0 0 0 -w10 0 0 0 0 0 w10 0 0 ;
0 0 -w7 0 w9 0 0 0 w7 0 w8 0 ;
0 w3 0 0 0 w5 0 -w3 0 0 0 w4];


if x1 == x2 & y1 == y2
 if z2 > z1
  Lambda = [0 0 1 ; 0 1 0 ; -1 0 0] ;
 else
  Lambda = [0 0 -1 ; 0 1 0 ; 1 0 0];
 end
else
CXx = (x2-x1)/L ;
CYx = (y2-y1)/L;
CZx = (z2-z1)/L;
D = sqrt(CXx*CXx + CYx*CYx);
CXy = -CYx/D;
CYy = CXx/D ;
CZy = 0;
CXz = -CXx*CZx/D;
CYz = -CYx*CZx/D;
```

```
CZz = D;

Lambda = [CXx CYx CZx ; CXy CYy CZy ; CXz CYz CZz];

end

R = [Lambda zeros(3) zeros(3) zeros(3)

zeros(3) Lambda zeros(3) zeros(3) ;

zeros(3) zeros(3) Lambda zeros(3) ;

zeros(3) zeros(3) zeros(3) Lambda];

y = R'*kprime*R;
```

Firstly, I will give the meaning of the inputs of the function.

E: Young's modulus of the material

G: Shear modulus of the material

A: Cross-sectional area of the element

Iy: Second moment of area about the y-axis

Iz: Second moment of area about the z-axis

J: Torsional constant (or polar moment of inertia)

x1, y1, z1: Coordinates of the first node of the element

x2, y2, z2: Coordinates of the second node of the element

Then, the function calculates the length L of the element using the Euclidean distance formula between the two nodes. After, I computed various coefficients (w1 through w10) based on the material and cross-sectional properties of the element. After, I created the local element stiffness matrix (kprime) based on the stiffness coefficients. The matrix kprime is a 12x12 matrix representing the stiffness of the element in local coordinates. The last thing is the R which is the transformation matrix. I calculated the transformation matrix R based on the orientation of the element. It transforms the local stiffness matrix into the global coordinate system. Finally, the transformed stiffness matrix (**y**) represents the stiffness of the element in the global coordinate system.

9.5 Assembling the Global Stiffness Matrix

```
% Assemble the global stiffness matrix

K_global = zeros(6*num_nodes, 6*num_nodes);

for i = 1:num_elements

    node_i = connections(i, 1);

    node_j = connections(i, 2);

    K_element = element_stiffness_matrices{i};

    K_global = SpaceFrameAssemble(K_global, K_element, node_i, node_j);

end

% Display the global stiffness matrix

disp('Global Stiffness Matrix (K_global):');

disp(K_global);
```

In this part of the code, I was focused on assembling the global stiffness matrix for the entire space frame structure. Here's how I accomplished this task:

First, I initialized the global stiffness matrix, **K_global**, with dimensions that account for all degrees of freedom in the system. Since each node in a space frame has six degrees of freedom (three translational and three rotational), the size of the global stiffness matrix is **6 * num_nodes** by **6 * num_nodes**. I used the **zeros** function to create a matrix filled with zeros of this size.

Next, I looped through each element to integrate its stiffness matrix into the global stiffness matrix. For each element, I performed the following steps:

1. **Identify Nodes**: I identified the nodes connected by the current element by extracting the node numbers from the **connections** matrix. **node_i** and **node_j** represent the first and second nodes of the element, respectively.

2. **Retrieve Element Stiffness Matrix**: I retrieved the stiffness matrix for the current element from the **element_stiffness_matrices** cell array.

3. **Assemble Global Stiffness Matrix**: I called the **SpaceFrameAssemble** function to integrate the element stiffness matrix into the appropriate locations within the global stiffness matrix. This function takes the global stiffness matrix, the element stiffness matrix, and the node numbers as inputs, and returns the updated global stiffness matrix. It ensures that the stiffness contributions of each element are correctly placed in the global matrix according to the connectivity of the nodes. More information about that function can be seen after the function script below.

After looping through all the elements and assembling their stiffness matrices into the global matrix, I displayed the resulting global stiffness matrix, **K_global**, using the **disp** function. This matrix represents the combined stiffness properties of the entire space frame structure, accounting for the interactions and connections between all its elements.

By assembling the global stiffness matrix in this manner, I ensured that the structural behavior of the space frame under various loads and conditions could be accurately analyzed in subsequent steps.

```
function y = SpaceFrameAssemble(K,k,i,j)



K(6*i-5 ,6*i-5) = K(6*i-5,6*i-5) + k(1 ,1) ;

K(6*i-5,6*i-4) = K(6*i-5,6*i-4) + k(1,2);

K(6*i-5,6*i-3) = K(6*i-5,6*i-3) + k(1,3);

K(6*i-5 ,6*i-2) = K(6*i-5,6*i-2) + k(1,4) ;

K(6*i-5,6*i-1) = K(6*i-5,6*i-1) + k(1,5);

K(6*i-5 ,6*i) = K(6*i-5,6*i) + k(1,6);

K(6*i-5,6*j-5) = K(6*i-5,6*j-5) + k(1 ,7) ;

K(6*i-5,6*j-4) = K(6*i-5 ,6*j-4) + k(1,8);

K(6*i-5,6*j-3) = K(6*i-5,6*j-3) + k(1 ,9) ;

K(6*i-5,6*j-2) = K(6*i-5,6*j-2) + k(1,10) ;

K(6*i-5,6*j-1) = K(6*i-5,6*j-1) + k(1,11);
```

```
K(6*i-5,6*j) = K(6*i-5,6*j) + k(1,12);

K(6*i-4,6*i-5) = K(6*i-4,6*i-5) + k(2,1);

K(6*i-4,6*i-4) = K(6*i-4,6*i-4) + k(2,2);

K(6*i-4,6*i-3) = K(6*i-4,6*i-3) + k(2,3);

K(6*i-4 ,6*i-2) = K(6*i-4,6*i-2) + k(2,4);

K(6*i-4 ,6*i-1) = K(6*i-4 ,6*i-1) + k(2,5);

K(6*i -4,6*i) = K(6*i-4 ,6*i) + k(2,6);

K(6*i-4,6*j-5) = K(6*i-4,6*j-5) + k(2,7) ;

K(6*i-4,6*j-4) = K(6*i-4,6*j-4) + k(2,8);

K(6*i-4,6*j-3) = K(6*i-4 ,6*j-3) + k(2,9);

K(6*i-4 ,6*j-2) = K(6*i-4,6*j-2) + k(2,10) ;

K(6*i-4,6*j-1) = K(6*i-4,6*j-1) + k(2,11);

K(6*i-4,6*j) = K(6*i-4,6*j) + k(2,12);

K(6*i-3,6*i-5) = K(6*i-3,6*i-5) + k(3,1);

K(6*i-3,6*i-4) = K(6*i -3,6*i-4) + k(3,2);

K(6*i-3,6*i-3) = K(6*i-3,6*i-3) + k(3,3);

K(6*i-3,6*i-2) = K(6*i-3 ,6*i-2) + k(3,4);

K(6*i-3,6*i-1) = K(6*i-3,6*i-1) + k(3 ,5);

K(6*i-3,6*i) = K(6*i-3,6*i) + k(3,6);

K(6*i-3,6*j-5) = K(6*i-3,6*j-5) + k(3,7);

K(6*i-3,6*j-4) = K(6*i-3,6*j-4) + k(3,8);

K(6*i-3,6*j-3) = K(6*i-3,6*j-3) + k(3,9);

K(6*i-3,6*j-2) = K(6*i-3,6*j-2) + k(3,10);

K(6*i-3,6*j-1) = K(6*i-3,6*j-1) + k(3,11);
```

```
K(6*i-3,6*j) = K(6*i-3,6*j) + k(3,12);

K(6*i-2,6*i-5) = K(6*i-2,6*i-5) + k(4,1);

K(6*i-2,6*i-4) = K(6*i-2,6*i-4) + k(4,2);

K(6*i-2,6*i-3) = K(6*i-2,6*i-3) + k(4,3);

K(6*i-2,6*i-2) = K(6*i-2,6*i-2) + k(4,4);

K(6*i-2,6*i-1) = K(6*i-2,6*i-1) + k(4,5) ;

K(6*i-2,6*i) = K(6*i-2,6*i) + k(4,6) ;

K(6*i-2,6*j-5) = K(6*i-2,6*j-5) + k(4,7);

K(6*i -2,6*j-4) = K(6*i-2,6*j-4) + k(4,8);

K(6*i-2,6*j-3) = K(6*i-2,6*j-3) + k(4,9) ;

K(6*i-2 ,6*j-2) = K(6*i-2,6*j-2) + k(4,10);

K(6*i-2,6*j-1) = K(6*i-2,6*j-1) + k(4 ,11);

K(6*i-2,6*j) = K(6*i-2,6*j) + k(4,12);

K(6*i-1,6*i-5) = K(6*i-1,6*i-5) + k(5,1) ;

K(6*i-1,6*i-4) = K(6*i-1,6*i-4) + k(5,2) ;

K(6*i-1,6*i-3) = K(6*i-1,6*i-3) + k(5,3);

K(6*i-1,6*i-2) = K(6*i-1,6*i-2) + k(5,4) ;

K(6*i-1,6*i-1) = K(6*i-1,6*i-1) + k(5,5);

K(6*i-1,6*i) = K(6*i-1,6*i) + k(5,6);

K(6*i-1,6*j-5) = K(6*i-1,6*j-5) + k(5,7);

K(6*i-1,6*j-4) = K(6*i-1,6*j-4) + k(5,8) ;

K(6*i-1,6*j-3) = K(6*i-1,6*j-3) + k(5,9);

K(6*i-1,6*j-2) = K(6*i-1,6*j-2) + k(5,10);

K(6*i-1,6*j-1) = K(6*i-1,6*j-1) + k(5,11);
```

```
K(6*i-1,6*j) = K(6*i-1,6*j) + k(5,12) ;

K(6*i,6*i-5) = K(6*i,6*i-5) + k(6,1);

K(6*i,6*i-4) = K(6*i,6*i-4) + k(6,2);

K(6*i,6*i-3) = K(6*i,6*i-3) + k(6,3) ;

K(6*i,6*i-2) = K(6*i,6*i-2) + k(6,4);

K(6*i,6*i-1) = K(6*i,6*i-1) + k(6,5);

K(6*i,6*i) = K(6*i,6*i) + k(6,6);

K(6*i,6*j-5) = K(6*i ,6*j-5) + k(6,7);

K(6*i,6*j-4) = K(6*i,6*j-4) + k(6 ,8);

K(6*i,6*j-3) = K(6*i ,6*j-3) + k(6 ,9);

K(6*i,6*j-2) = K(6*i,6*j-2) + k(6,10);

K(6*i,6*j-1) = K(6*i,6*j-1) + k(6,11);

K(6*i,6*j) = K(6*i,6*j) + k(6,12);

K(6*j-5,6*i-5) = K(6*j-5,6*i-5) + k(7,1);

K(6*j-5,6*i-4) = K(6*j-5,6*i-4) + k(7,2);

K(6*j-5,6*i-3) = K(6*j-5,6*i-3) + k(7,3);

K(6*j-5 ,6*i-2) = K(6*j-5,6*i-2) + k(7,4);

K(6*j-5,6*i-1) = K(6*j-5,6*i-1) + k(7,5);

K(6*j-5,6*i) = K(6*j-5,6*i) + k(7,6);

K(6*j-5,6*j-5) = K(6*j-5,6*j-5) + k(7,7);

K(6*j-5,6*j-4) = K(6*j-5,6*j-4) + k(7,8);

K(6*j -5,6*j-3) = K(6*j-5,6*j-3) + k(7,9);

K(6*j-5,6*j-2) = K(6*j-5,6*j-2) + k(7,10);

K(6*j-5,6*j-1) = K(6*j-5,6*j-1) + k(7,11);
```

```
K(6*j-5,6*j) = K(6*j-5 ,6*j) + k(7,12);

K(6*j-4,6*i-5) = K(6*j-4,6*i-5) + k(8,1) ;

K(6*j-4,6*i-4) = K(6*j-4,6*i-4) + k(8,2);

K(6*j-4,6*i-3) = K(6*j-4,6*i-3) + k(8,3);

K(6*j-4,6*i-2) = K(6*j-4,6*i-2) + k(8,4);

K(6*j-4,6*i-1) = K(6*j-4,6*i-1) + k(8,5);

K(6*j-4,6*i) = K(6*j-4,6*i) + k(8,6);

K(6*j-4,6*j-5) = K(6*j-4,6*j-5) + k(8,7);

K(6*j-4,6*j-4) = K(6*j -4,6*j-4) + k(8,8) ;

K(6*j-4,6*j-3) = K(6*j-4,6*j-3) + k(8,9);

K(6*j-4,6*j-2) = K(6*j-4,6*j-2) + k(8,10);

K(6*j-4,6*j-1) = K(6*j -4,6*j-1) + k(8,11);

K(6*j-4,6*j) = K(6*j-4,6*j) + k(8,12);

K(6*j-3,6*i-5) = K(6*j -3,6*i-5) + k(9,1);

K(6*j -3,6*i-4) = K(6*j-3,6*i-4) + k(9,2);

K(6*j-3,6*i-3) = K(6*j-3,6*i-3) + k(9,3);

K(6*j-3,6*i-2) = K(6*j-3,6*i-2) + k(9,4);

K(6*j-3,6*i-1) = K(6*j-3,6*i-1) + k(9,5);

K(6*j-3,6*i) = K(6*j-3,6*i) + k(9,6);

K(6*j-3,6*j-5) = K(6*j-3,6*j-5) + k(9,7);

K(6*j-3,6*j-4) = K(6*j-3,6*j-4) + k(9,8);

K(6*j-3,6*j-3) = K(6*j -3,6*j-3) + k(9,9);

K(6*j-3,6*j-2) = K(6*j-3,6*j-2) + k(9,10);

K(6*j-3,6*j-1) = K(6*j-3,6*j-1) + k(9,11) ;
```

```
K(6*j-3,6*j) = K(6*j-3,6*j) + k(9,12) ;

K(6*j-2 ,6*i-5) = K(6*j-2,6*i-5) + k(10,1);

K(6*j-2,6*i-4) = K(6*j-2,6*i-4) + k(10,2);

K(6*j-2,6*i-3) = K(6*j-2,6*i-3) + k(10,3);

K(6*j-2,6*i-2) = K(6*j-2,6*i-2) + k(10,4);

K(6*j-2,6*i-1) = K(6*j-2,6*i-1) + k(10,5);

K(6*j-2,6*i) = K(6*j -2,6*i) + k(10,6);

K(6*j-2,6*j-5) = K(6*j-2,6*j-5) + k(10,7);

K(6*j-2,6*j-4) = K(6*j-2,6*j-4) + k(10,8);

K(6*j-2,6*j-3) = K(6*j-2,6*j-3) + k(10,9) ;

K(6*j-2,6*j-2) = K(6*j-2,6*j-2) + k(10,10);

K(6*j-2,6*j-1) = K(6*j-2,6*j-1) + k(10,11);

K(6*j-2,6*j) = K(6*j-2,6*j) + k(10,12);

K(6*j-1,6*i-5) = K(6*j-1,6*i-5) + k(11,1);

K(6*j-1,6*i-4) = K(6*j-1,6*i-4) + k(11 ,2) ;

K(6*j-1,6*i-3) = K(6*j-1,6*i-3) + k(11,3);

K(6*j-1,6*i-2) = K(6*j-1,6*i-2) + k(11,4);

K(6*j-1,6*i-1) = K(6*j-1,6*i-1) + k(11,5);

K(6*j-1,6*i) = K(6*j-1,6*i) + k(11,6);

K(6*j-1,6*j-5) = K(6*j-1,6*j-5) + k(11,7) ;

K(6*j-1 ,6*j-4) = K(6*j-1,6*j-4) + k(11,8) ;

K(6*j-1,6*j-3) = K(6*j-1,6*j-3) + k(11,9) ;

K(6*j-1 ,6*j-2) = K(6*j-1,6*j-2) + k(11,10) ;

K(6*j-1,6*j-1) = K(6*j-1,6*j-1) + k(11,11);
```

```
K(6*j-1,6*j) = K(6*j-1 ,6*j) + k(11,12);

K(6*j,6*i-5) = K(6*j ,6*i-5) + k(12 ,1) ;

K(6*j,6*i-4) = K(6*j,6*i-4) + k(12,2) ;

K(6*j,6*i-3) = K(6*j,6*i-3) + k(12,3) ;

K(6*j,6*i-2) = K(6*j,6*i-2) + k(12,4);

K(6*j,6*i-1) = K(6*j,6*i-1) + k(12 ,5);

K(6*j ,6*i) = K(6*j,6*i) + k(12 ,6) ;

K(6*j ,6*j-5) = K(6*j,6*j-5) + k(12,7);

K(6*j,6*j-4) = K(6*j ,6*j-4) + k(12,8);

K(6*j ,6*j-3) = K(6*j,6*j-3) + k(12,9) ;

K(6*j,6*j-2) = K(6*j,6*j-2) + k(12,10);

K(6*j,6*j-1) = K(6*j,6*j-1) + k(12,11);

K(6*j,6*j) = K(6*j,6*j) + k(12 ,12);

y = K;
```

Firstly, I will give the meanings of the function inputs below:

K: The global stiffness matrix of the entire structure.

k: The stiffness matrix of the individual element to be assembled.

i, **j**: The node indices corresponding to the element being assembled.

The function updates the global stiffness matrix K by adding the contributions from the stiffness matrix k of the individual element. It places the components of k into the appropriate locations of K based on the node indices i and j. Then function maps the local degrees of freedom of the element to the global degrees of freedom of the entire structure. Finally, the function returns the updated global stiffness matrix K.

9.6 Boundary Condition Input

```
% Initialize force,moment and displacement,rotation vectors
F = zeros(6*num_nodes, 1); % External forces vector and moments
U = zeros(6*num_nodes, 1); % Displacements vector and rotations
known_U = false(6*num_nodes, 1); % Logical array to track known displacements
% Ask the user for boundary condition information for each node
for i = 1:num_nodes
  prompt = sprintf('Is the boundary condition at Node %d force/moment (F) or displacement/rotation (U)?', i);
  dlgtitle = 'Boundary Condition Type';
  bc_type = questdlg(prompt, dlgtitle, 'F', 'U', 'F'); % Default to force/moment
  if strcmpi(bc_type, 'F')


    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), ...
      sprintf('Enter the force along z-axis (Fz in kN) at Node %d:', i), ...
      sprintf('Enter the moment about x-axis (Mx in kN*m) at Node %d:', i), ...
      sprintf('Enter the moment about y-axis (My in kN*m) at Node %d:', i), ...
      sprintf('Enter the moment about z-axis (Mz in kN*m) at Node %d:', i) };
    dlgtitle = sprintf('Force and Moment at Node %d', i);
    dims = [1 50];
    defaultInput = {'0', '0', '0', '0', '0', '0'}; % Default values
    forces = inputdlg(prompt, dlgtitle, dims, defaultInput);
```

```
        F(6*i-5:6*i) = str2double(forces);
    elseif strcmpi(bc_type, 'U')
        prompt = {...
            sprintf('Enter the displacement along x-axis (Ux in m) at Node %d:', i), ...
            sprintf('Enter the displacement along y-axis (Uy in m) at Node %d:', i), ...
            sprintf('Enter the displacement along z-axis (Uz in m) at Node %d:', i), ...
            sprintf('Enter the rotation about x-axis (Rx in rad) at Node %d:', i), ...
            sprintf('Enter the rotation about y-axis (Ry in rad) at Node %d:', i), ...
            sprintf('Enter the rotation about z-axis (Rz in rad) at Node %d:', i) };
        dlgtitle = sprintf('Displacement and Rotation at Node %d', i);
        dims = [1 50];
        defaultInput = {'0', '0', '0', '0', '0', '0'}; % Default values
        displacements = inputdlg(prompt, dlgtitle, dims, defaultInput);
        U(6*i-5:6*i) = str2double(displacements);
        known_U(6*i-5:6*i) = true; % Mark displacements and rotations as known
    end
end
```

In this part of the code, I was focused on initializing the vectors for external forces and moments (**F**), as well as the vectors for displacements and rotations (**U**). Here's a detailed explanation of what I was doing:

First, I initialized three vectors:

- **F**: This vector stores the external forces and moments applied at each node in the structure. It has a length of **6 * num_nodes** to account for all possible forces and moments (three forces and three moments) at each node.
- **U**: This vector stores the displacements and rotations at each node. Like **F**, it has a length of **6 * num_nodes**.

- **known_U**: This is a logical array that tracks which displacements and rotations are known (specified by boundary conditions). Initially, all entries are set to **false**.

Next, I looped through each node to collect boundary condition information. For each node, I prompted the user to specify whether the boundary condition is a force/moment (**F**) or a displacement/rotation (**U**). I used the **questdlg** function to present a dialog box with options "F" and "U", defaulting to "F".

If the user selected "F" (force/moment), another dialog box appeared asking for the force and moment values along the x, y, and z axes for the current node. I used the **inputdlg** function to collect these values, with default inputs of **0**. The entered values were then converted to numerical format using **str2double** and stored in the corresponding positions in the **F** vector.

If the user selected "U" (displacement/rotation), a similar dialog box appeared asking for the displacement and rotation values along the x, y, and z axes for the current node. Again, I used **inputdlg** to collect these values and **str2double** to convert them to numerical format. The values were stored in the corresponding positions in the **U** vector, and the **known_U** logical array was updated to mark these entries as **true**, indicating that these displacements and rotations are known.

By the end of this process, I had collected all the necessary boundary condition information for each node, with the **F** vector containing the specified external forces and moments, the **U** vector containing the specified displacements and rotations, and the **known_U** array indicating which displacements and rotations are known. This setup is crucial for the subsequent steps in analyzing the space frame structure, as it provides the necessary boundary conditions for solving the system of equations that describe the structure's behavior.

9.7 Visualization of Space Frame Element

```
% Create a new figure for Space Frame Element

figure;

% Plot nodes

plot3(node_coordinates(:, 1), node_coordinates(:, 2), node_coordinates(:, 3), 'bo', 'MarkerSize', 10); hold on;

% Plot connections
```

```
for i = 1:num_elements

    node_i = connections(i, 1);

    node_j = connections(i, 2);

    x = [node_coordinates(node_i, 1), node_coordinates(node_j, 1)];

    y = [node_coordinates(node_i, 2), node_coordinates(node_j, 2)];

    z = [node_coordinates(node_i, 3), node_coordinates(node_j, 3)];

    plot3(x, y, z, 'k-', 'LineWidth', 1);

end

% Set labels and title

xlabel('X');

ylabel('Y');

zlabel('Z');

title('Space Frame Element');

% Add labels for node numbers with a slight offset

offset = 0.15; % Adjust the offset as needed

for i = 1:num_nodes

    text(node_coordinates(i, 1) + offset, node_coordinates(i, 2) + offset, node_coordinates(i, 3)
+ offset, num2str(i), 'Color', 'red', 'FontSize', 15);

end

% Adjust aspect ratio and view angle

axis equal;

view(3);
```

In this part of the code, I was focused on creating a visual representation of the space frame structure. Here's an explanation of what I was doing step by step:

1. **Creating a New Figure:**
    - I started by creating a new figure window using the **figure** command. This sets up a separate window for plotting the space frame structure.
2. **Plotting the Nodes:**
    - I used the **plot3** function to plot the nodes in 3D space. The coordinates of the nodes are stored in the **node_coordinates** matrix, and the 'bo' argument specifies that the nodes should be plotted as blue circles with a marker size of 10. The **hold on** command allows subsequent plotting commands to add to the current figure without replacing the existing plot.
3. **Plotting the Connections:**
    - I looped through each element to plot the connections between the nodes. For each element, I extracted the node numbers (**node_i** and **node_j**) that define the element. Then, I retrieved the coordinates of these nodes from the **node_coordinates** matrix.
    - Using the **plot3** function, I plotted a line between the two nodes representing the element. The 'k-' argument specifies that the line should be black with a default line width of 1.
4. **Setting Labels and Title:**
    - I labeled the x, y, and z axes using the **xlabel**, **ylabel**, and **zlabel** functions, respectively.
    - I set the title of the plot to 'Space Frame Element' using the **title** function.
5. **Adding Labels for Node Numbers:**
    - To enhance clarity, I added labels for each node number. I used a small offset to avoid overlapping the labels with the nodes themselves. The **text** function was used to place the labels, with the node number specified by **num2str(i)**, colored red, and set to a font size of 15.

6. **Adjusting Aspect Ratio and View Angle:**
    - I set the aspect ratio to be equal using the **axis equal** command, ensuring that the units are the same along all axes, which helps in visualizing the structure without distortion.
    - Finally, I adjusted the view angle to a 3D perspective using the **view(3)** command, which provides a better overall view of the space frame structure.

This plotting section of the code was crucial for visually verifying the geometry and connectivity of the space frame elements and nodes, ensuring that the data input process was accurate and that the structure was correctly defined.

9.8 Modification of Global Stiffness Matrix and Force Vector

```
% Modify the global stiffness matrix and force vector based on known displacements
K_global_original = K_global; % Save the original global stiffness matrix for reaction calculations
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 or rotation
   end
end
```

In this section, I saved the original global stiffness matrix which is K_global in previous code into K_global_original. Because in this step I rearranged the K_global to solve the unknown displacements and rotations in next step. Simply the code modifies the global stiffness matrix K_global and force vector F based on known displacements. More detail, if a displacement is known at a node, code sets the corresponding row of K_global to zeros and assigns a value of one to the diagonal element. It also updates the corresponding force value.

9.9 Solving for Displacements, Rotations and Forces, Moments

```
% Solve for unknown displacements and rotations

U_unknown = K_global \ F;

% Update displacements and rotations vector with solved values

U(~known_U) = U_unknown(~known_U);

% Calculate reactions at supports using the original global stiffness matrix

R = K_global_original * U ;

% Display results

disp('Node Displacements and Rotations (U):');

disp(U);

disp('Reactions (R):');

disp(R);

% Display results in a dialog box

result_message = {...

    'Node Displacements and Rotations (U in m and rad):', ...

    num2str(U), ...

    '', ...

    'Reactions (R in kN and kN*m) :', ...

    num2str(R) };

dlgtitle = 'Analysis Results';

msgbox(result_message, dlgtitle);
```

Firstly, in this part I solved for unknown displacements and rotations using the modified global stiffness matrix and force vector. Then, code updates the U vector with the solved values for unknown displacements and rotations. Then I calculated the reactions using the original global stiffness matrix and the updated displacement vector. Finally, I displayed the results, including node displacements and rotations (U) and reactions (R). Also, I give the outputs in to a message box for a better representation.

9.10 Calculating Nodal Displacements and Element Forces

```
% Initialize element force vectors cell array
f = cell(1, num_elements);
% Loop through each element to compute stiffness matrices and force vectors
for i = 1:num_elements
    % Extract node numbers for the current element
    node_i = connections(i, 1);
    node_j = connections(i, 2);
    % Extract coordinates of the nodes for the current element
    x1 = node_coordinates(node_i, 1);
    y1 = node_coordinates(node_i, 2);
    z1 = node_coordinates(node_i, 3);
    x2 = node_coordinates(node_j, 1);
    y2 = node_coordinates(node_j, 2);
    z2 = node_coordinates(node_j, 3);
    % Collect displacements for the nodes of the current element
    u1_indices = (6*node_i-5):(6*node_i);
    u2_indices = (6*node_j-5):(6*node_j);
    u_a = U(u1_indices);
    u_b = U(u2_indices);
    u = [u_a;u_b];
    disp(['Nodal Displacement vector in meter for Element ', num2str(i), ':']);
```

```
   disp(u);
   % Compute element force vector using the collected displacements
   f{i} = SpaceFrameElementForces(E, G, A, Iy, Iz, J, x1, y1, z1, x2, y2, z2, u);
   % Display element force vector
   disp(['Element Force Vector in kN for Element ', num2str(i), ':']);
  disp(f{i});
 % Create message box content
   msg = {...
       ['Element ', num2str(i), ' Results:'], ...
       sprintf('Nodal Displacements (u) in meter and Element Forces (f) in kN:\n')};
   % Add displacements and forces to the message
   for j = 1:numel(u)
       msg{end+1} = sprintf('u%d = %.10f,   f%d = %.10f\n', j, u(j), j, f{i}(j));
   end
   % Convert message cell array to a string
   msg_str = strjoin(msg, '\n');
   % Display message box
   msgbox(msg_str, ['Element ', num2str(i), ' Results']);
```

In this part of the code, the program computes the force vectors for each element in the space frame structure. It initializes a cell array to store these vectors, then iterates through each element, extracting the necessary node numbers and coordinates. Using the collected displacement data from the global displacement vector, it calculates the force vector for each element by using the SpaceFrameElementForces function. Below I will give and explain the function. After computation, the code generates message boxes displaying the nodal displacements and element forces in a user-friendly format. Thanks to this computation we can analysis every element of the structure separately. The above part of my code can be seen start with for loop, that loop has not completed yet. The loop completed next heading ‘9.11’.

```
function y =SpaceFrameElementForces(E,G,A,Iy,Iz,J,x1 ,y1,z1,x2,y2,z2,u)

L = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1) + (z2-z1)*(z2-z1));

w1 = E*A/L ;

w2 = 12*E*Iz/(L*L*L);

w3 = 6*E*Iz/(L*L);

w4 = 4*E*Iz/L;

w5 = 2*E*Iz/L;

w6 = 12*E*Iy/(L*L*L);

w7 = 6*E*Iy/(L*L);

w8 = 4*E*Iy/L;

w9 = 2*E*Iy/L;

w10 = G*J/L;

kprime = [w1 0 0 0 0 0 -w1 0 0 0 0 0;

0 w2 0 0 0 w3 0 -w2 0 0 0 w3;

0 0 w6 0 -w7 0 0 0 -w6 0 -w7 0;

0 0 0 w10 0 0 0 0 0 -w10 0 0;

0 0 -w7 0 w8 0 0 0 w7 0 w9 0;

0 w3 0 0 0 w4 0 -w3 0 0 0 w5;

-w1 0 0 0 0 0 w1 0 0 0 0 0;

0 -w2 0 0 0 -w3 0 w2 0 0 0 -w3;

0 0 -w6 0 w7 0 0 0 w6 0 w7 0;

0 0 0 -w10 0 0 0 0 0 w10 0 0 ;

0 0 -w7 0 w9 0 0 0 w7 0 w8 0 ;

0 w3 0 0 0 w5 0 -w3 0 0 0 w4];
```

```
if x1 == x2 & y1 == y2
 if z2 > z1
  Lambda = [0 0 1 ; 0 1 0 ; -1 0 0] ;
 else
  Lambda = [0 0 -1 ; 0 1 0 ; 1 0 0];
 end
else
CXx = (x2-x1)/L ;
CYx = (y2-y1)/L;
CZx = (z2-z1)/L;
D = sqrt(CXx*CXx + CYx*CYx);
CXy = -CYx/D;
CYy = CXx/D ;
CZy = 0;
CXz = -CXx*CZx/D;
CYz = -CYx*CZx/D;
CZz = D;
Lambda = [CXx CYx CZx ; CXy CYy CZy ; CXz CYz CZz];
end
R = [Lambda zeros(3) zeros(3) zeros(3)
zeros(3) Lambda zeros(3) zeros(3) ;
zeros(3) zeros(3) Lambda zeros(3) ;
zeros(3) zeros(3) zeros(3) Lambda];
```

```
y = kprime * R * u;
```

The SpaceFrameElementForces function calculates the element force vector for a space frame element based on various input parameters. These parameters are almost the same as in heading 9.4. As an extra, that function uses the nodal displacement vector which is ‘u’ to obtain the element forces.

9.11 Plotting Diagrams

```
% Compute the length of elements
    L{i} = SpaceFrameElementLength(x1, y1, z1, x2, y2, z2);


    % Plot axial force diagram for the current element
    figure; % Create a new figure for each element
    SpaceFrameElementAxiaIDiagram(f{1,i}, L{i});
    xlabel('Length in meter');
    ylabel('Axial Force in kN');
    title(['Axial Force Diagram for Element ', num2str(i)]);


     % Plot shear force diagram in Y direction for the current element
    figure; % Create a new figure for each element
    SpaceFrameElementShearYDiagram(f{1,i}, L{i});
    xlabel('Length in meter');
    ylabel('Shear Force in kN');
    title(['Shear Force Diagram in Y Direction for Element ', num2str(i)]);


    % Plot shear force diagram in Z direction for the current element
    figure; % Create a new figure for each element
```

```
SpaceFrameElementShearZDiagram(f{1,i}, L{i});
xlabel('Length in meter');
ylabel('Shear Force in kN');
title(['Shear Force Diagram in Z Direction for Element ', num2str(i)]);

 % Plot torsion diagram for the current element
figure; % Create a new figure for each element
SpaceFrameElementTorsionDiagram(f{1,i}, L{i});
xlabel('Length in meter');
ylabel('Torsion in kN*m');
title(['Torsion Diagram for Element ', num2str(i)]);

 % Plot Bending Moment diagram Along Y axis for the current element
figure; % Create a new figure for each element
SpaceFrameElementMomentYDiagram(f{1,i}, L{i});
xlabel('Length in meter');
ylabel('Bending Moment in kN*m');
title(['Bending Moment Diagram along Y axis for Element ', num2str(i)]);

% Plot Bending Moment diagram Along Z axis for the current element
figure; % Create a new figure for each element
SpaceFrameElementMomentZDiagram(f{1,i}, L{i});
xlabel('Length in meter');
ylabel('Bending Moment in kN*m');
```

```
    title(['Bending Moment Diagram along Z axis for Element ', num2str(i)]);

end
```

Firstly, length (L) of each element found by using function which is 'SpaceFrameElementLength'. This function can be seen below.

```
function y = SpaceFrameElementLength(x1, y1, z1, x2, y2, z2)

    % Calculate the length using the Euclidean distance formula

    y = sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2);

end
```

After that, plots are which are Axial Force Diagram, Shear Force Diagram in Y Direction, Shear Force Diagram in Z Direction, Torsion Diagram, Bending Moment Diagram along Y axis, Bending Moment Diagram along Z axis respectively drawn by my program. While drawing these diagrams, the corresponding element forces (f) in the functions used. The functions scripts include a short explanation also. The functions can be seen below

Axial Force Diagram:

```
function y = SpaceFrameElementAxiaIDiagram(f, L)

%SpaceFrameElementAxialDiagram This function plots the axial force diagram for the space
frame element with nodal force vector f and length L

x = [0 ; L];

z = [-f(1) ; f(7)] ;

hold on;

title('Axial Force Diagram');

plot(x,z, 'r') ;

y1 = [0 ; 0];

plot (x, y1 , 'b' )
```

Shear Force Diagram in Y Direction:

```
function y = SpaceFrameElementShearYDiagram(f, L)

%SpaceFrameElementShearYDiagram This function plots the shear force diagram for the space frame element with nodal force vector f and length L.

x = [0 ; L];

z = [f(2) ; -f(8)];

hold on ;

title('Shear Force Diagram in Y Direction ');

plot(x,z,'r');

y1 = [0 ; 0] ;

plot(x,y1, 'b')
```

Shear Force Diagram in Z Direction:

```
function y = SpaceFrameElementShearZDiagram(f, L)

% SpaceFrameElementShearZDiagram This function plots the shear force diagram for the space frame element with nodal force vector f and length L.

x = [0 ; L];

z = [f(3) ; -f(9)];

hold on;

title('Shear Force Diagram in Z Direction');

plot(x,z,'r');

y1 = [0 ; 0];

plot(x,y1, 'b' )
```

Torsion Diagram:

```
function y = SpaceFrameElementTorsionDiagram(f, L)
```

```
%SpaceFrameElementTorsionDiagram This function plots the torsion diagram for the space frame element with nodal force vector f and length L .

x = [0 ; L];

z = [f(4) ; -f(10)];

hold on;

title('Torsion Diagram');

plot(x,z, 'r');

y1 = [0 ; 0] ;

plot(x,y1, 'b')
```

Bending Moment Diagram along Y Axis:

```
function y = SpaceFrameElementMomentYDiagram(f, L)

% SpaceFrameElementMomentYDiagram This function plots the bending moment diagram for the space frame element with nodal force vector f and length L .

x = [0 ; L];

z = [f(5) ; -f(11)];

hold on;

title('Bending Moment Diagram along Y AXis');

plot(x,z,'r');

y1 = [0 ; 0];

plot(x,y1, 'b')
```

Bending Moment Diagram along Z Axis:

```
function y = SpaceFrameElementMomentZDiagram(f, L)

% SpaceFrameElementMomentZDiagram This function plots the bending moment diagram for the space frame element with nodal force vector f and length L.
```

```
x = [0 ; L];

z = [f(6) ; -f(12)] ;

hold on ;

title('Bending Moment Diagram along Z Axis');

plot(x,z,'r') ;

yl = [0 ; 0];

plot(x,yl, 'b')
```

9.12 Small Example for Explanations How the Code Works

In this heading, I would like to solve a simple question using my codes. This section is done to explain in detail what the code does in section 9.8 and section 9.9.

Firstly, the main idea is to solve an equation with unknowns. For our case, we can write the equation as follows.

$$[K] \times [U] = [F] \tag{111}$$

For the above equation, we can think of K as the global matrix, U as the displacement vector, and F as the force vector. Then let's write a 3 × 3 matrix for K. Let's do this as a vector of 3 × 1 in U. And let's place the F vector corresponding to K and U in equation (111) in order below.

$$\begin{bmatrix} 2 & 2 & 0 \\ 0 & 3 & 3 \\ 4 & 0 & 4 \end{bmatrix} \times \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 10 \\ 21 \\ 24 \end{bmatrix} \tag{112}$$

Now let's assume that U_1, F_2 and F_3 are not known in the above equation.

$$\begin{bmatrix} 2 & 2 & 0 \\ 0 & 3 & 3 \\ 4 & 0 & 4 \end{bmatrix} \times \begin{bmatrix} U_1 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 10 \\ \mathrm{F}_2 \\ \mathrm{F}_3 \end{bmatrix} \tag{113}$$

As explained in Section 9.8, the code will first make the rows of known displacements zero in the K matrix, and then make the diagonals of these rows 1 like below.

$$\begin{bmatrix} 2 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \mathrm{U}_1 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 10 \\ \mathrm{F}_2 \\ \mathrm{F}_3 \end{bmatrix} \tag{114}$$

Then, code will assign known displacements to unknown forces, as explained in section 9.8.

$$\begin{bmatrix} 2 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \mathrm{U}_1 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 10 \\ 3 \\ 4 \end{bmatrix}$$

Since the first thing is to find the U vector, the known U values that we assign in the F vector will be written directly into the U vector after the below calculation.

$$U = [K_{new}]^{-1} \times F \tag{115}$$

$$U = \begin{bmatrix} 2 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \times \begin{bmatrix} 10 \\ 3 \\ 4 \end{bmatrix} \tag{116}$$

$$U = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \tag{117}$$

We see above that the U vector gives the desired result. Now we can find the vector F by multiplying the known vector U by the matrix K.

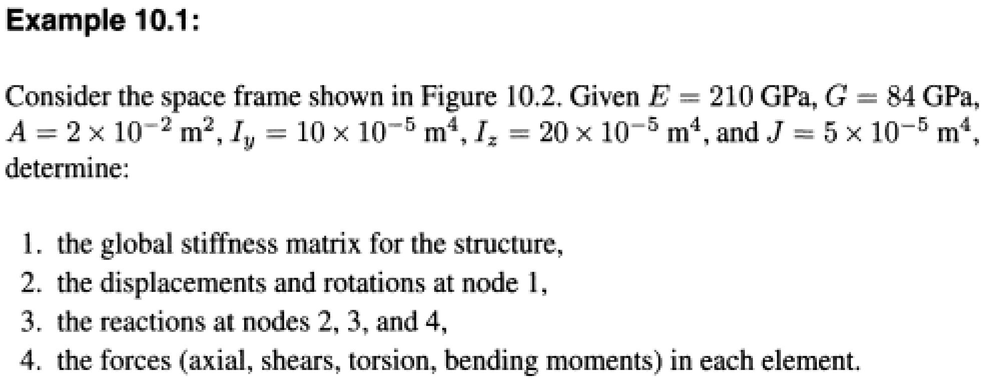
$$[K] \times [U] = [F] \tag{118}$$

$$\begin{bmatrix} 2 & 2 & 0 \\ 0 & 3 & 3 \\ 4 & 0 & 4 \end{bmatrix} \times \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} = [F] = \begin{bmatrix} 10 \\ 21 \\ 24 \end{bmatrix} \tag{119}$$

In the above steps, I have shown how to solve an equation with unknowns step by step using the working principle of the code I wrote.

10. EXAMPLE

To validate the reliability and accuracy of my computer program for finite element analysis, I will employ a well-established example from "MATLAB Guide to Finite Elements: An Interactive Approach" by P. I. Kattan (2003). The chosen example, 10.1 from the space frame element section. By comparing the outcomes generated by my program with the documented results, I aim to verify the accuracy of my computational approach in analyzing space frame elements.



Example 10.1:

Consider the space frame shown in Figure 10.2. Given $E = 210$ GPa, $G = 84$ GPa, $A = 2 \times 10^{-2}$ m^2, $I_y = 10 \times 10^{-5}$ m^4, $I_z = 20 \times 10^{-5}$ m^4, and $J = 5 \times 10^{-5}$ m^4, determine:

1. the global stiffness matrix for the structure,
2. the displacements and rotations at node 1,
3. the reactions at nodes 2, 3, and 4,
4. the forces (axial, shears, torsion, bending moments) in each element.

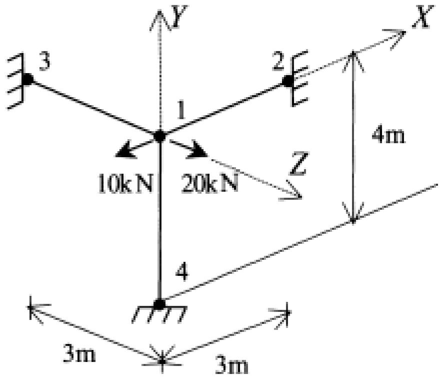
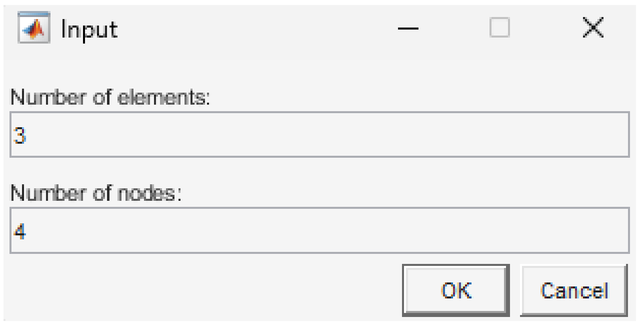


Figure 19: Example Problem [5].

After running my code, in MATLAB program first my program asks the user what the number of elements is and what is the number of nodes in a dialog box. According to my chosen example from the fifth of my reference book the number of elements and the number of nodes is 3 and 4 respectively. The dialog box can be seen below

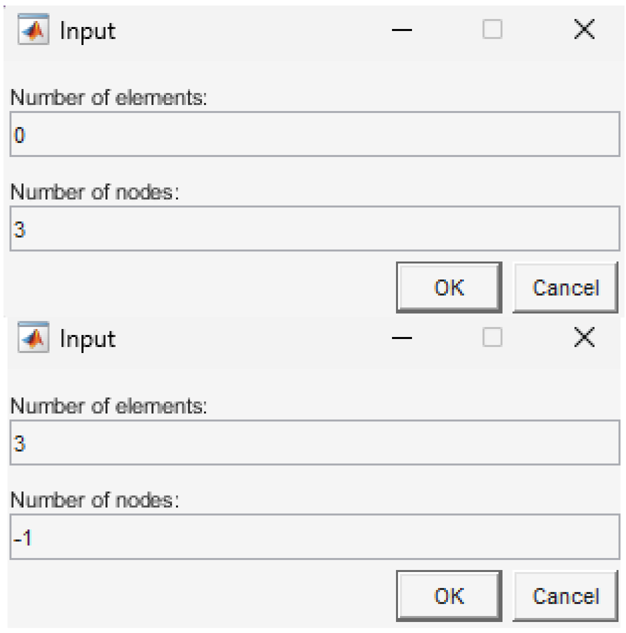

Input

Number of elements:
3
Number of nodes:
4

Figure 20: Input screen for number of elements and number of nodes

If the user enters these inputs negative number or zero my program warns the user with an error message in a box. The error message is ‘Number of elements and nodes must be greater than zero’. That message continues until valid inputs are provided by the user. If the inputs are valid the user can continue the program.

Below I show screenshots from my program about after the invalid input for number of elements and number of nodes provided by use.


Input

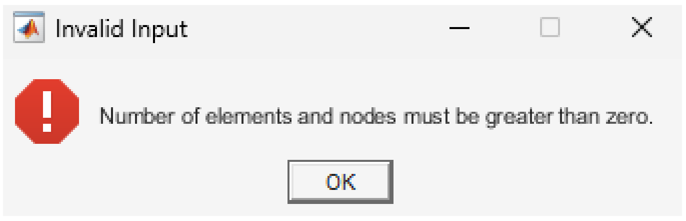
Number of elements:
0
Number of nodes:
3

Input

Number of elements:
3
Number of nodes:
-1

Figure 21: Invalid inputs

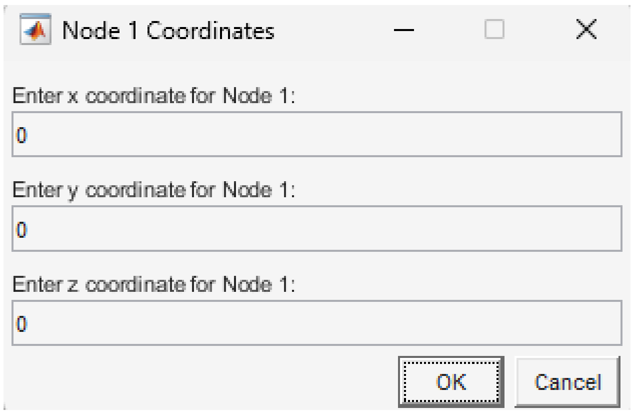
After those invalid inputs program gives an error message


Invalid Input

Number of elements and nodes must be greater than zero.

Figure 22: Error message for invalid inputs

After the valid inputs are taken from the user, the program ask the user x, y and z coordinates of the nodes. Again, the pop-up screen appears on the screen like figure 23 below. From figure 19, node 1 coordinates are (0, 0, 0).



Node 1 Coordinates

Enter x coordinate for Node 1:

0

Enter y coordinate for Node 1:

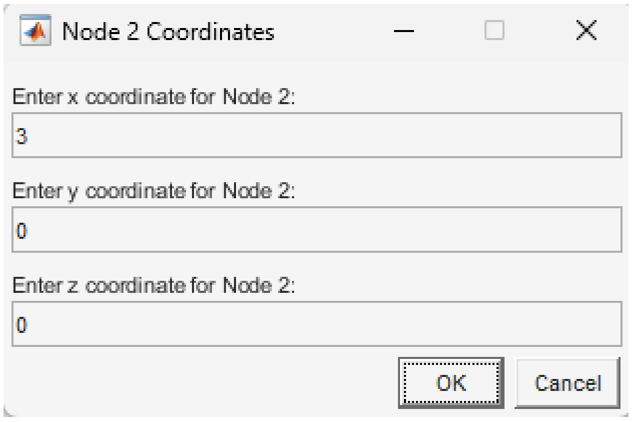
0

Enter z coordinate for Node 1:

0

Figure 23: Inputs for node 1 coordinates

Program ask the user 4 times according to given number of nodes in figure 20. And the coordinates given according to structure geometry from example problem in figure 19. So, user can continue with other node coordinates. From figure 19, node 2 coordinates are (3, 0, 0), node 3 coordinates are (0, 0, -3), node 4 coordinates are (0, -4, 0).



Node 2 Coordinates

Enter x coordinate for Node 2:

3

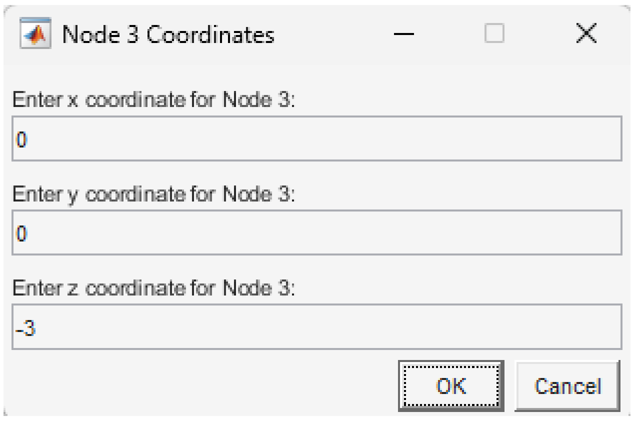
Enter y coordinate for Node 2:

0

Enter z coordinate for Node 2:

0

Figure 24: Inputs for node 2 coordinates


Node 3 Coordinates

Enter x coordinate for Node 3:

0

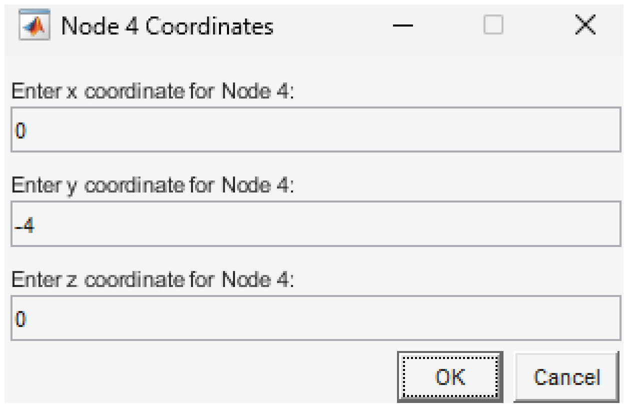
Enter y coordinate for Node 3:

0

Enter z coordinate for Node 3:

-3

Figure 25: Inputs for node 3 coordinates


Node 4 Coordinates

Enter x coordinate for Node 4:

0

Enter y coordinate for Node 4:

-4

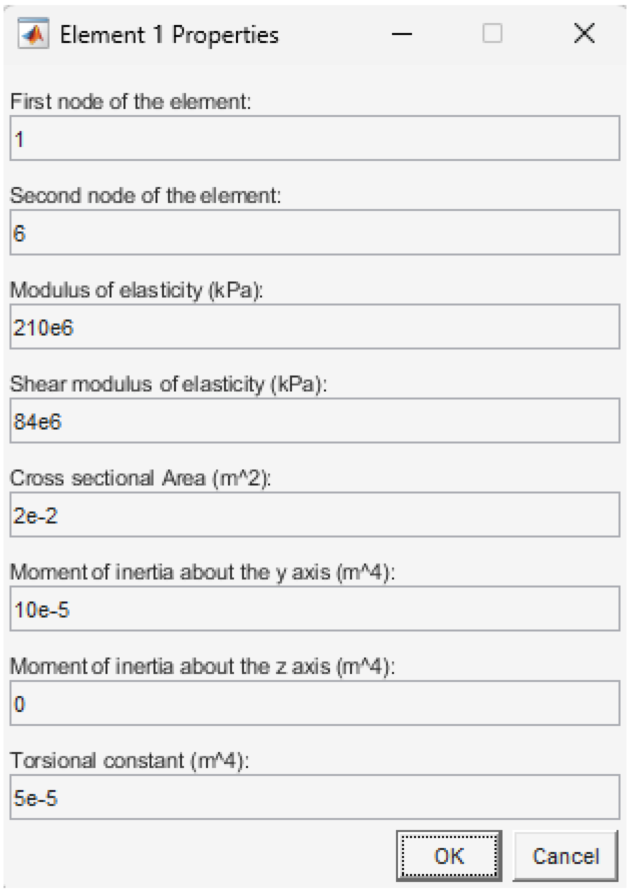
Enter z coordinate for Node 4:

0

Figure 26: Inputs for node 4 coordinates

After the coordinates of are taken from the user, program continue with elements connections and elements' material properties. Again, this part of my program controls the user inputs valid or invalid. The invalid inputs are negative or zero given for the material properties which are modulus of elasticity, shear modulus of elasticity, cross sectional area, moment of inertias about the y and z axis, torsional constant. Also, program control the connections input for elements. For example, element 1 in the structure is between two nodes which are node 1 and node 2. These connections are taken from the user, so these inputs also cannot be zero or negative. My program checks all these inputs and if one of them invalid, program warn the user an error message 'All inputs must be greater than zero and node1, node2 numbers must be less than or equal to the total number of nodes.'

Now I give an error example in figure 27 below. For that I give a wrong input for a material property.


Element 1 Properties

First node of the element:
1

Second node of the element:
6

Modulus of elasticity (kPa):
210e6

Shear modulus of elasticity (kPa):
84e6

Cross sectional Area (m^2):
2e-2

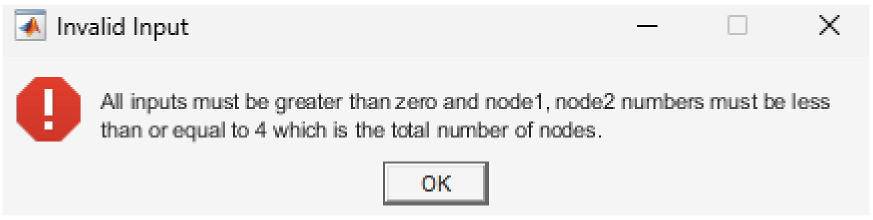
Moment of inertia about the y axis (m^4):
10e-5

Moment of inertia about the z axis (m^4):
0

Torsional constant (m^4):
5e-5

Figure 27: Invalid input for a material property

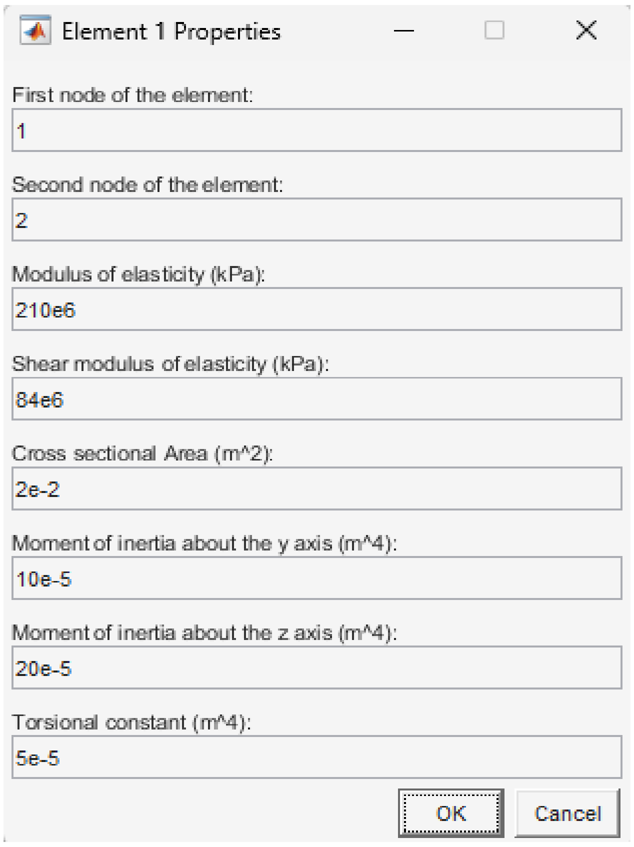
In figure 27, all inputs can be given positive, but the program also checks the first two inputs which are connections parts of the element cannot be higher total number of nodes. For example, which is in figure 19. We have 4 total number of nodes. The user cannot give the connections higher 4. If the user gave the higher number, then total number of nodes for connections input my program also warn the user in a same error message for the case in the figure 27.


Invalid Input

All inputs must be greater than zero and node1, node2 numbers must be less than or equal to 4 which is the total number of nodes.

Figure 28: Error message to user

We can continue to example problem with valid inputs according to given knowledge in the question in figure 19. It can be seen inputs in figure 29, 30, 31. The user will see 3 screens for that part according to given number of elements which is 3.


Element 1 Properties

First node of the element:
1

Second node of the element:
2

Modulus of elasticity (kPa):
210e6

Shear modulus of elasticity (kPa):
84e6

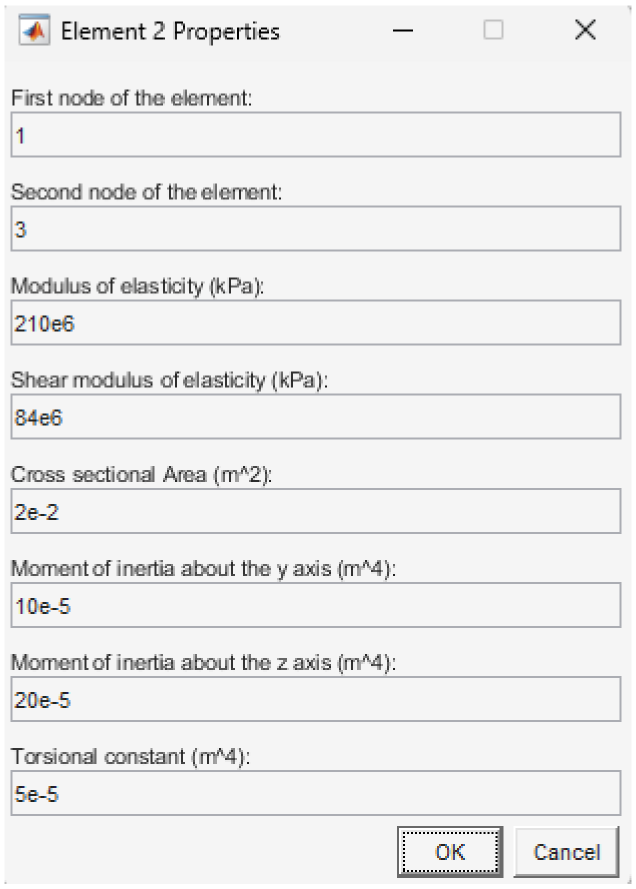
Cross sectional Area (m^2):
2e-2

Moment of inertia about the y axis (m^4):
10e-5

Moment of inertia about the z axis (m^4):
20e-5

Torsional constant (m^4):
5e-5

Figure 29: Element 1 properties


Element 2 Properties

First node of the element:
1

Second node of the element:
3

Modulus of elasticity (kPa):
210e6

Shear modulus of elasticity (kPa):
84e6

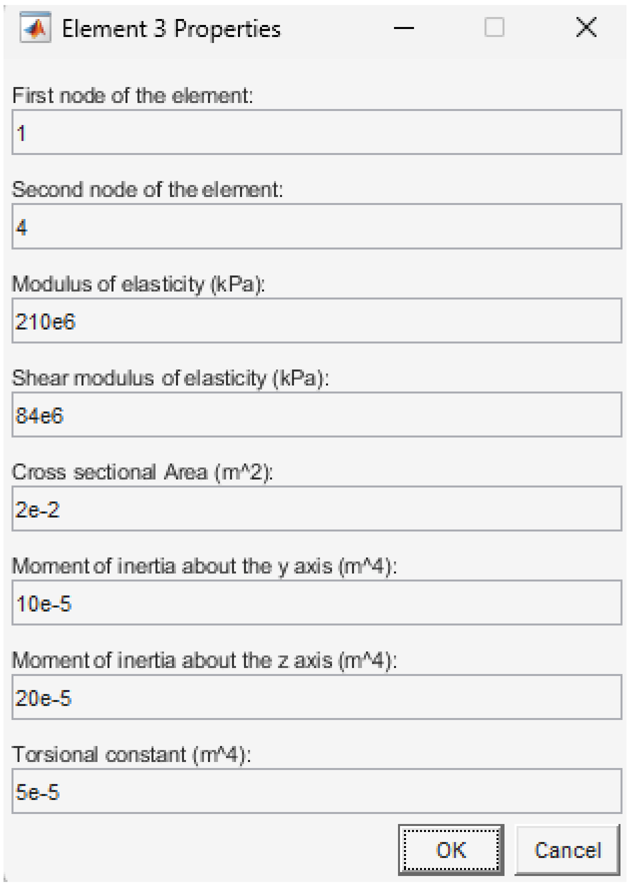
Cross sectional Area (m^2):
2e-2

Moment of inertia about the y axis (m^4):
10e-5

Moment of inertia about the z axis (m^4):
20e-5

Torsional constant (m^4):
5e-5

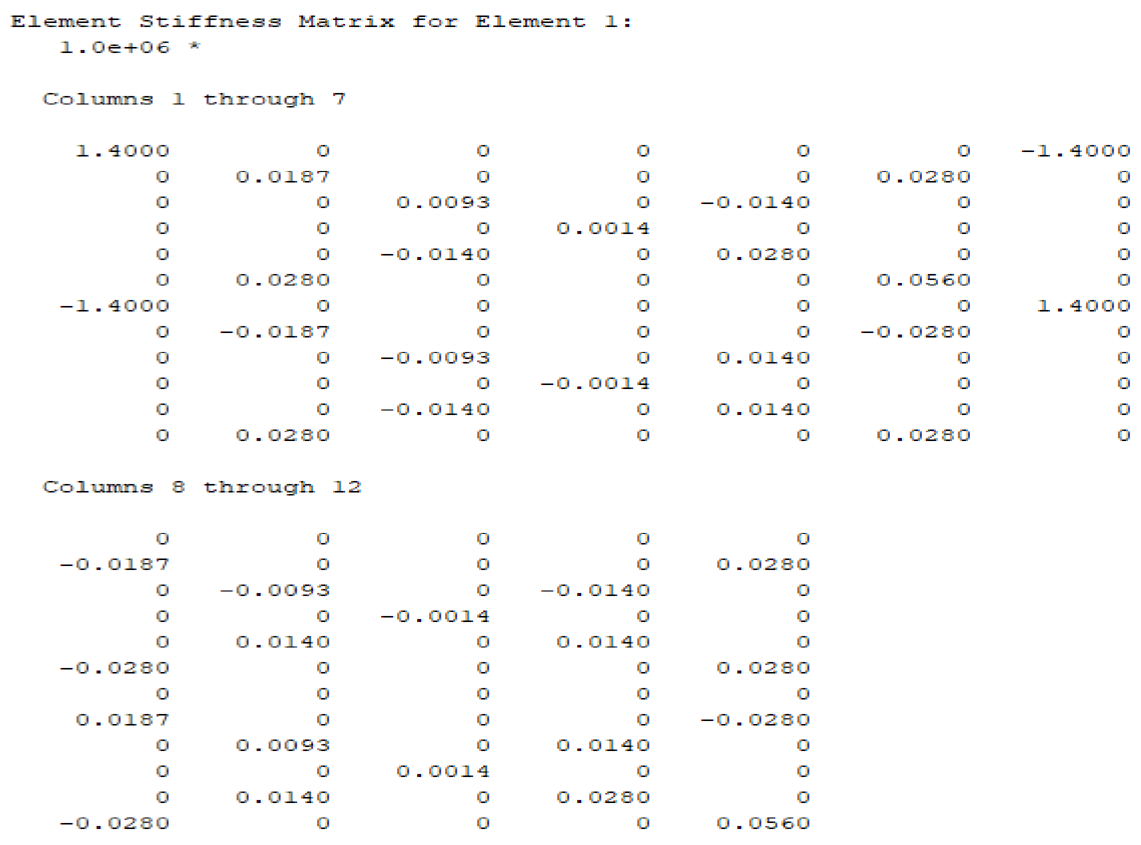
Figure 30: Element 2 properties



First node of the element:
1
Second node of the element:
4
Modulus of elasticity (kPa):
210e6
Shear modulus of elasticity (kPa):
84e6
Cross sectional Area (m^2):
2e-2
Moment of inertia about the y axis (m^4):
10e-5
Moment of inertia about the z axis (m^4):
20e-5
Torsional constant (m^4):
5e-5

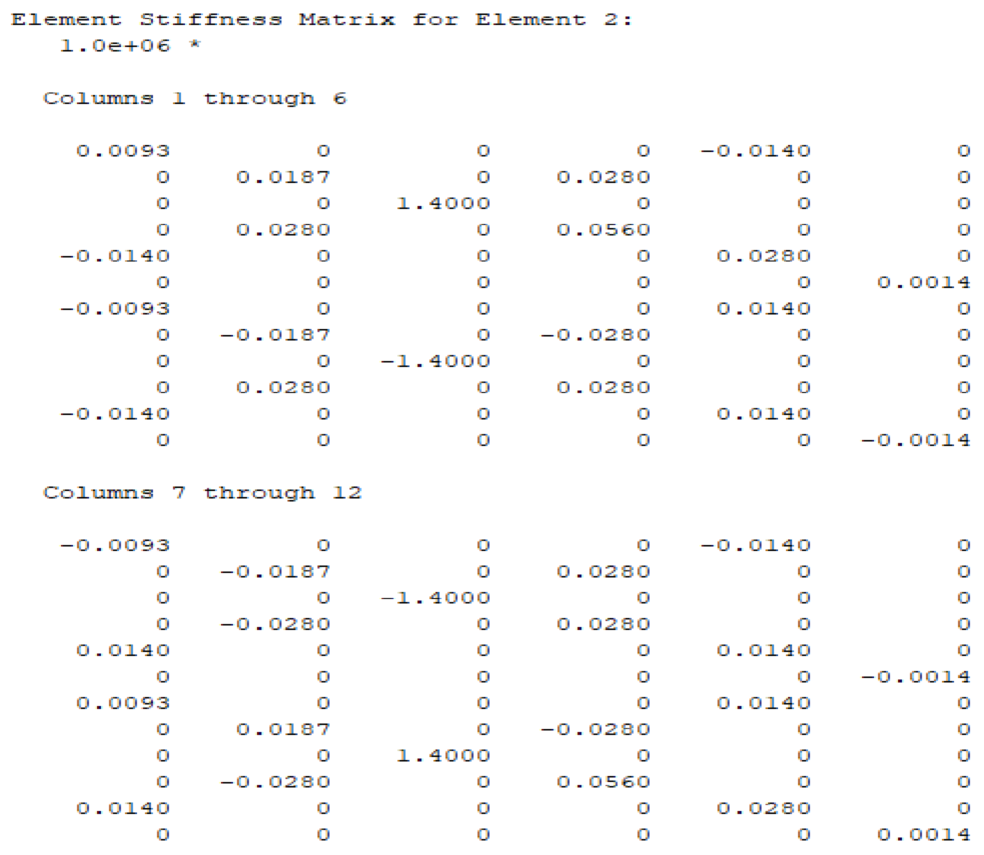
Figure 31: Element 3 properties

After the properties of elements given by the user, code found the element of stiffness matrices and shows the command window in MATLAB.



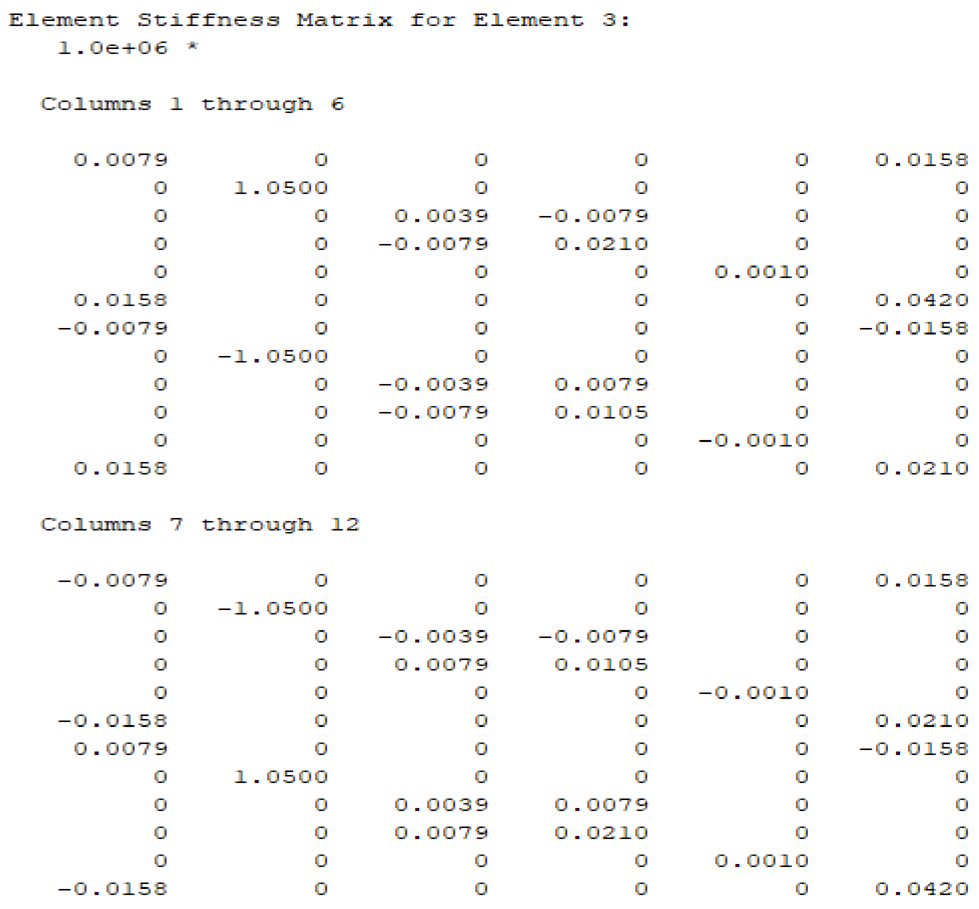
```
Element Stiffness Matrix for Element 1:
   1.0e+06 *

  Columns 1 through 7

    1.4000         0         0         0         0         0   -1.4000
         0    0.0187         0         0         0    0.0280         0
         0         0    0.0093         0   -0.0140         0         0
         0         0         0    0.0014         0         0         0
         0         0   -0.0140         0    0.0280         0         0
         0    0.0280         0         0         0    0.0560         0
   -1.4000         0         0         0         0         0    1.4000
         0   -0.0187         0         0         0   -0.0280         0
         0         0   -0.0093         0    0.0140         0         0
         0         0         0   -0.0014         0         0         0
         0         0   -0.0140         0    0.0140         0         0
         0    0.0280         0         0         0    0.0280         0

  Columns 8 through 12

         0         0         0         0         0
   -0.0187         0         0         0    0.0280
         0   -0.0093         0   -0.0140         0
         0         0   -0.0014         0         0
         0    0.0140         0    0.0140         0
   -0.0280         0         0         0    0.0280
         0         0         0         0         0
    0.0187         0         0         0   -0.0280
         0    0.0093         0    0.0140         0
         0         0    0.0014         0         0
         0    0.0140         0    0.0280         0
   -0.0280         0         0         0    0.0560
```

Figure 32: Element Stiffness Matrix for element 1



```
Element Stiffness Matrix for Element 2:
   1.0e+06 *

  Columns 1 through 6

    0.0093         0         0         0   -0.0140         0
         0    0.0187         0    0.0280         0         0
         0         0    1.4000         0         0         0
         0    0.0280         0    0.0560         0         0
   -0.0140         0         0         0    0.0280         0
         0         0         0         0         0    0.0014
   -0.0093         0         0         0    0.0140         0
         0   -0.0187         0   -0.0280         0         0
         0         0   -1.4000         0         0         0
         0    0.0280         0    0.0280         0         0
   -0.0140         0         0         0    0.0140         0
         0         0         0         0         0   -0.0014

  Columns 7 through 12

   -0.0093         0         0         0   -0.0140         0
         0   -0.0187         0    0.0280         0         0
         0         0   -1.4000         0         0         0
         0   -0.0280         0    0.0280         0         0
    0.0140         0         0         0    0.0140         0
         0         0         0         0         0   -0.0014
    0.0093         0         0         0    0.0140         0
         0    0.0187         0   -0.0280         0         0
         0         0    1.4000         0         0         0
         0   -0.0280         0    0.0560         0         0
    0.0140         0         0         0    0.0280         0
         0         0         0         0         0    0.0014
```

Figure 33: Element Stiffness Matrix for element 2

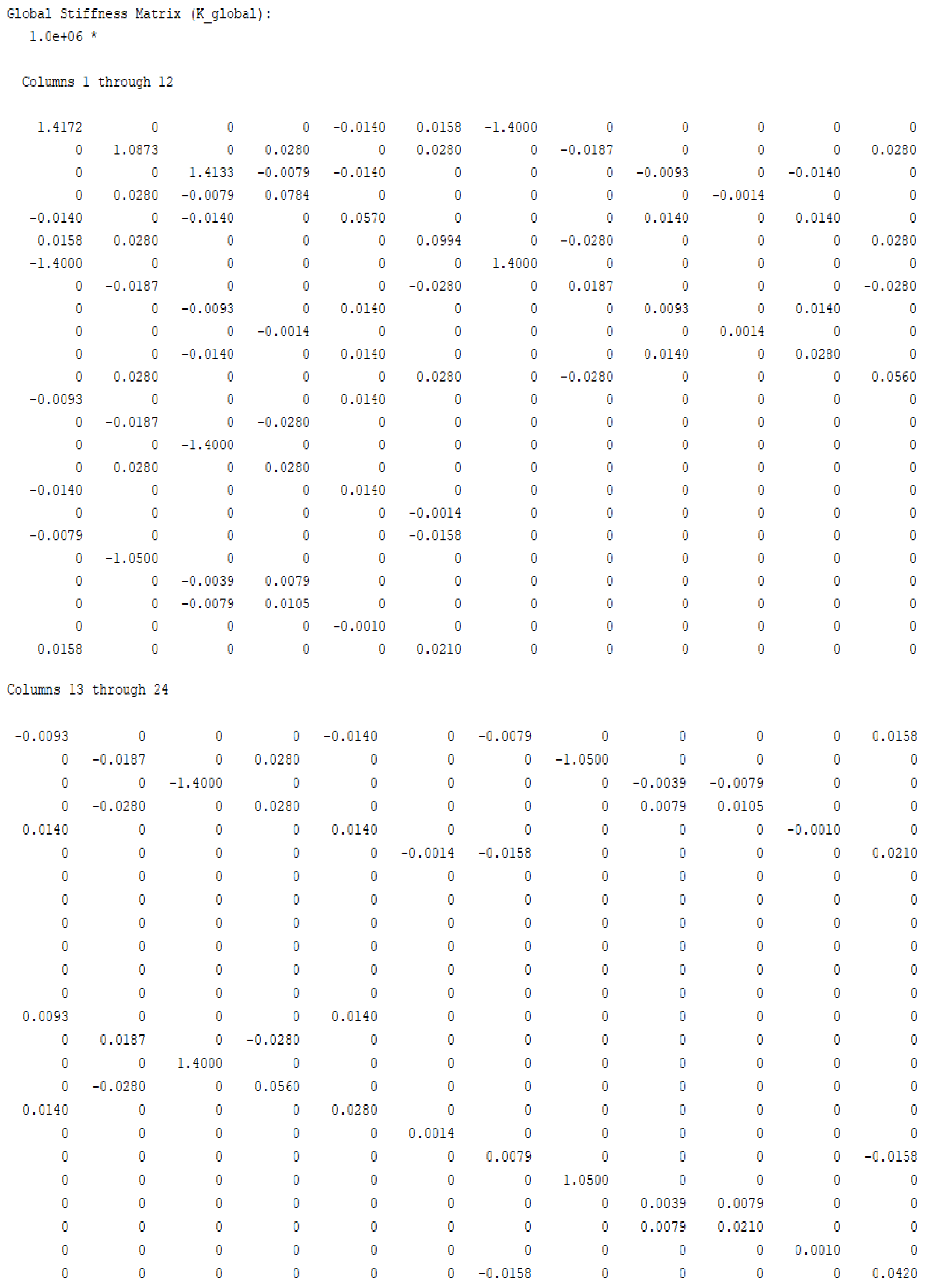


```
Element Stiffness Matrix for Element 3:
   1.0e+06 *

  Columns 1 through 6

    0.0079         0         0         0         0    0.0158
         0    1.0500         0         0         0         0
         0         0    0.0039   -0.0079         0         0
         0         0   -0.0079    0.0210         0         0
         0         0         0         0    0.0010         0
    0.0158         0         0         0         0    0.0420
   -0.0079         0         0         0         0   -0.0158
         0   -1.0500         0         0         0         0
         0         0   -0.0039    0.0079         0         0
         0         0   -0.0079    0.0105         0         0
         0         0         0         0   -0.0010         0
    0.0158         0         0         0         0    0.0210

  Columns 7 through 12

   -0.0079         0         0         0         0    0.0158
         0   -1.0500         0         0         0         0
         0         0   -0.0039   -0.0079         0         0
         0         0    0.0079    0.0105         0         0
         0         0         0         0   -0.0010         0
   -0.0158         0         0         0         0    0.0210
    0.0079         0         0         0         0   -0.0158
         0    1.0500         0         0         0         0
         0         0    0.0039    0.0079         0         0
         0         0    0.0079    0.0210         0         0
         0         0         0         0    0.0010         0
   -0.0158         0         0         0         0    0.0420
```

Figure 34: Element Stiffness Matrix for Element 3

These matrix results are checked with the solution manual of the reference book.

Once all properties of elements in the structure are taken by the user, code can calculate the global stiffness matrix. Also, that matrix appears on the command window of MATLAB.

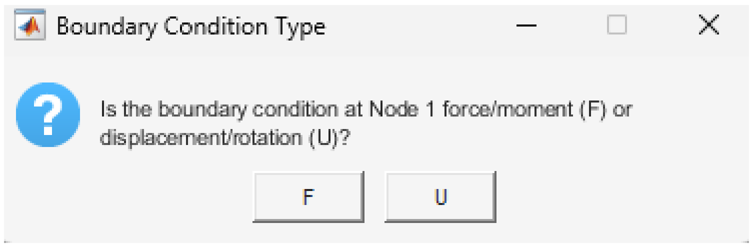


```
Global Stiffness Matrix (K_global):
   1.0e+06 *

  Columns 1 through 12

    1.4172         0         0         0   -0.0140    0.0158   -1.4000         0         0         0         0         0
         0    1.0873         0    0.0280         0    0.0280         0   -0.0187         0         0         0    0.0280
         0         0    1.4133   -0.0079   -0.0140         0         0         0   -0.0093         0   -0.0140         0
         0    0.0280   -0.0079    0.0784         0         0         0         0         0   -0.0014         0         0
   -0.0140         0   -0.0140         0    0.0570         0         0         0    0.0140         0    0.0140         0
    0.0158    0.0280         0         0         0    0.0994         0   -0.0280         0         0         0    0.0280
   -1.4000         0         0         0         0         0    1.4000         0         0         0         0         0
         0   -0.0187         0         0         0   -0.0280         0    0.0187         0         0         0   -0.0280
         0         0   -0.0093         0    0.0140         0         0         0    0.0093         0    0.0140         0
         0         0         0   -0.0014         0         0         0         0         0    0.0014         0         0
         0         0   -0.0140         0    0.0140         0         0         0    0.0140         0    0.0280         0
         0    0.0280         0         0         0    0.0280         0   -0.0280         0         0         0    0.0560
   -0.0093         0         0         0    0.0140         0         0         0         0         0         0         0
         0   -0.0187         0   -0.0280         0         0         0         0         0         0         0         0
         0         0   -1.4000         0         0         0         0         0         0         0         0         0
         0    0.0280         0    0.0280         0         0         0         0         0         0         0         0
   -0.0140         0         0         0    0.0140         0         0         0         0         0         0         0
         0         0         0         0         0   -0.0014         0         0         0         0         0         0
   -0.0079         0         0         0         0   -0.0158         0         0         0         0         0         0
         0   -1.0500         0         0         0         0         0         0         0         0         0         0
         0         0   -0.0039    0.0079         0         0         0         0         0         0         0         0
         0         0   -0.0079    0.0105         0         0         0         0         0         0         0         0
         0         0         0         0   -0.0010         0         0         0         0         0         0         0
    0.0158         0         0         0         0    0.0210         0         0         0         0         0         0

Columns 13 through 24

   -0.0093         0         0         0   -0.0140         0   -0.0079         0         0         0         0    0.0158
         0   -0.0187         0    0.0280         0         0         0   -1.0500         0         0         0         0
         0         0   -1.4000         0         0         0         0         0   -0.0039   -0.0079         0         0
         0   -0.0280         0    0.0280         0         0         0         0    0.0079    0.0105         0         0
    0.0140         0         0         0    0.0140         0         0         0         0         0   -0.0010         0
         0         0         0         0         0   -0.0014   -0.0158         0         0         0         0    0.0210
         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0         0         0         0
    0.0093         0         0         0    0.0140         0         0         0         0         0         0         0
         0    0.0187         0   -0.0280         0         0         0         0         0         0         0         0
         0         0    1.4000         0         0         0         0         0         0         0         0         0
         0   -0.0280         0    0.0560         0         0         0         0         0         0         0         0
    0.0140         0         0         0    0.0280         0         0         0         0         0         0         0
         0         0         0         0         0    0.0014         0         0         0         0         0         0
         0         0         0         0         0         0    0.0079         0         0         0         0   -0.0158
         0         0         0         0         0         0         0    1.0500         0         0         0         0
         0         0         0         0         0         0         0         0    0.0039    0.0079         0         0
         0         0         0         0         0         0         0         0    0.0079    0.0210         0         0
         0         0         0         0         0         0         0         0         0         0    0.0010         0
         0         0         0         0         0         0   -0.0158         0         0         0         0    0.0420
```

Figure 35: Global Stiffness Matrix

After matrixes are calculated, the program continues with boundary conditions to calculate the displacements, rotations and force, moments.

As known every node must have one boundary condition which could be force, moment or displacement, rotation for the space frame elements my program ask that like figure 36 below.

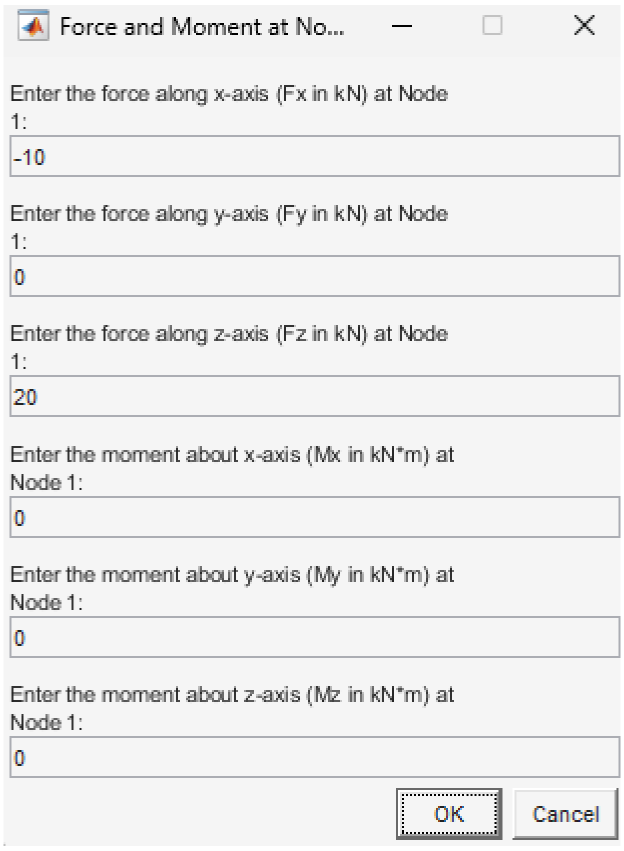


Boundary Condition Type

Is the boundary condition at Node 1 force/moment (F) or displacement/rotation (U)?

Figure 36: Boundary condition type

According to example problem from figure 19, in the first node has force boundary condition so user can choose F in figure 36 then enter the forces in x direction -10 kilonewton and in z direction 20 kilonewton. There is no force in y direction and there are no moments in all directions. So, the user can enter the inputs like figure 37 below.



Force and Moment at No...

Enter the force along x-axis (Fx in kN) at Node 1:
-10

Enter the force along y-axis (Fy in kN) at Node 1:
0

Enter the force along z-axis (Fz in kN) at Node 1:
20

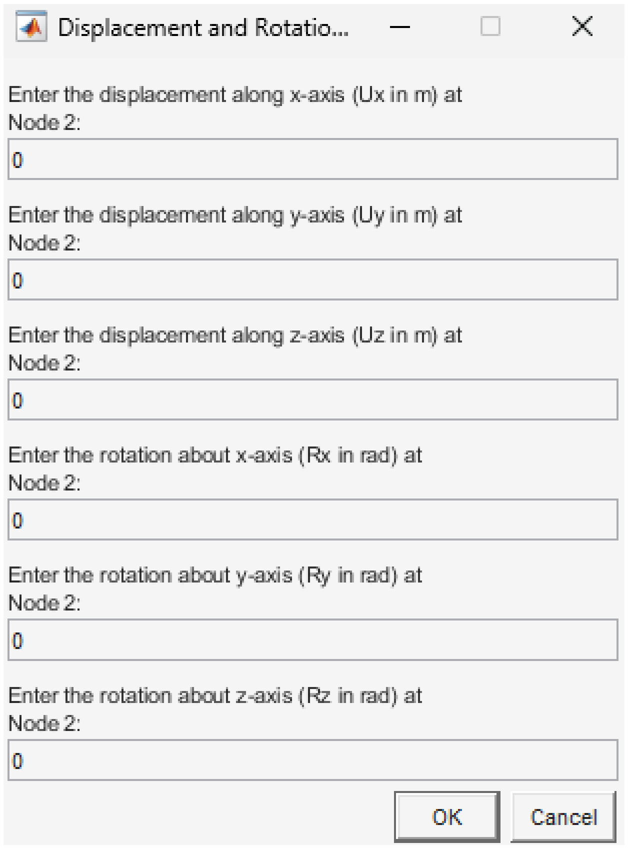
Enter the moment about x-axis (Mx in kN*m) at Node 1:
0

Enter the moment about y-axis (My in kN*m) at Node 1:
0

Enter the moment about z-axis (Mz in kN*m) at Node 1:
0

Figure 37: Force and moments at node 1

Nodes 2,3 and 4 are fixed so user can choose U in the other screen appears after entered inputs in figure 37 and enter the inputs like figure 38,39,40 below.


Displacement and Rotatio...

Enter the displacement along x-axis (Ux in m) at Node 2:
0

Enter the displacement along y-axis (Uy in m) at Node 2:
0

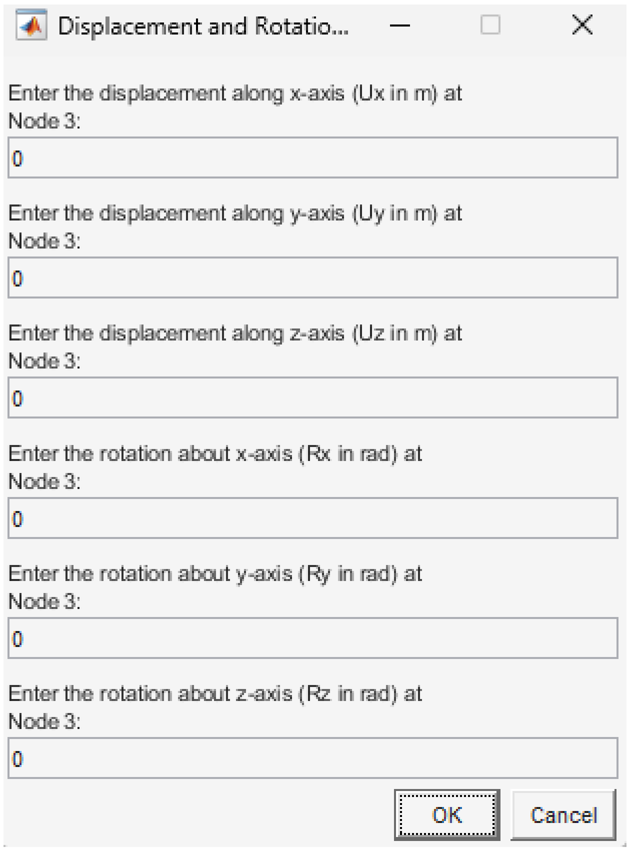
Enter the displacement along z-axis (Uz in m) at Node 2:
0

Enter the rotation about x-axis (Rx in rad) at Node 2:
0

Enter the rotation about y-axis (Ry in rad) at Node 2:
0

Enter the rotation about z-axis (Rz in rad) at Node 2:
0

Figure 38: Displacements and rotations at node 2


Displacement and Rotatio...

Enter the displacement along x-axis (Ux in m) at Node 3:
0

Enter the displacement along y-axis (Uy in m) at Node 3:
0

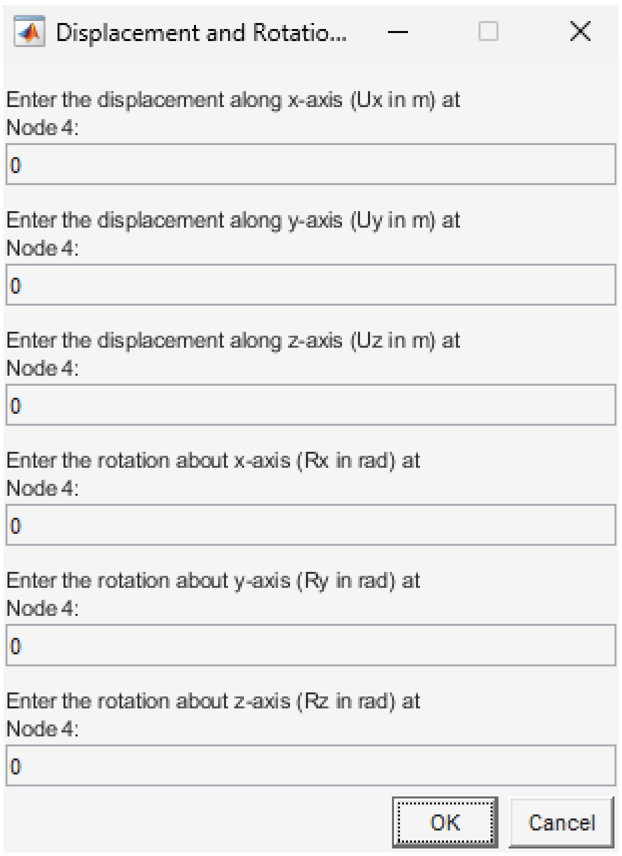
Enter the displacement along z-axis (Uz in m) at Node 3:
0

Enter the rotation about x-axis (Rx in rad) at Node 3:
0

Enter the rotation about y-axis (Ry in rad) at Node 3:
0

Enter the rotation about z-axis (Rz in rad) at Node 3:
0

Figure 39: Displacements and rotation at node 3


Displacement and Rotatio...

Enter the displacement along x-axis (Ux in m) at Node 4:
0

Enter the displacement along y-axis (Uy in m) at Node 4:
0

Enter the displacement along z-axis (Uz in m) at Node 4:
0

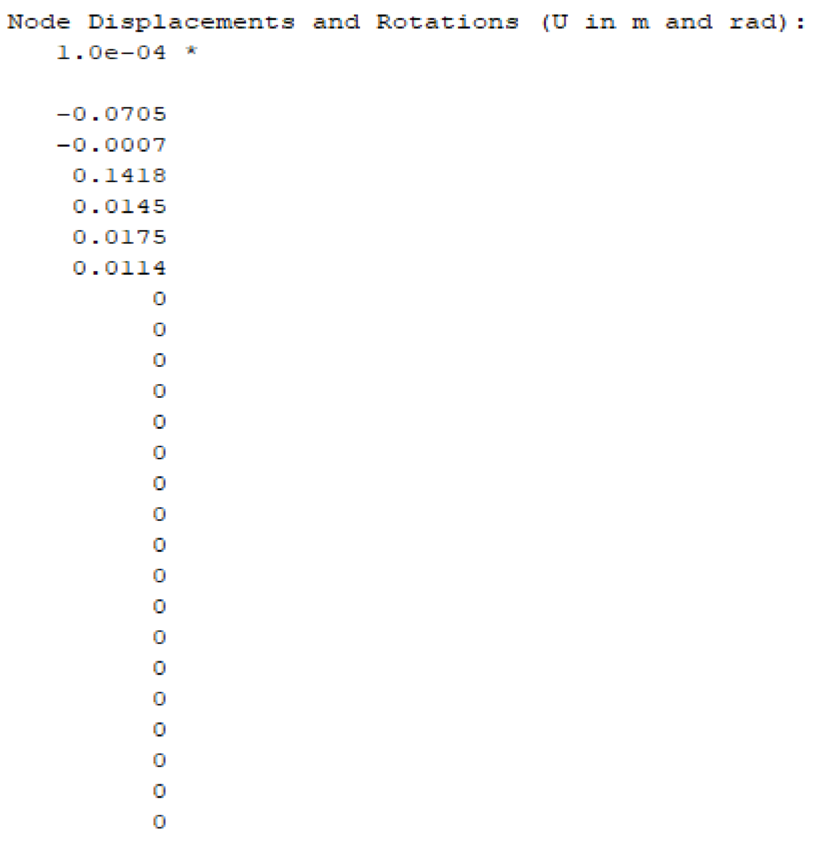
Enter the rotation about x-axis (Rx in rad) at Node 4:
0

Enter the rotation about y-axis (Ry in rad) at Node 4:
0

Enter the rotation about z-axis (Rz in rad) at Node 4:
0

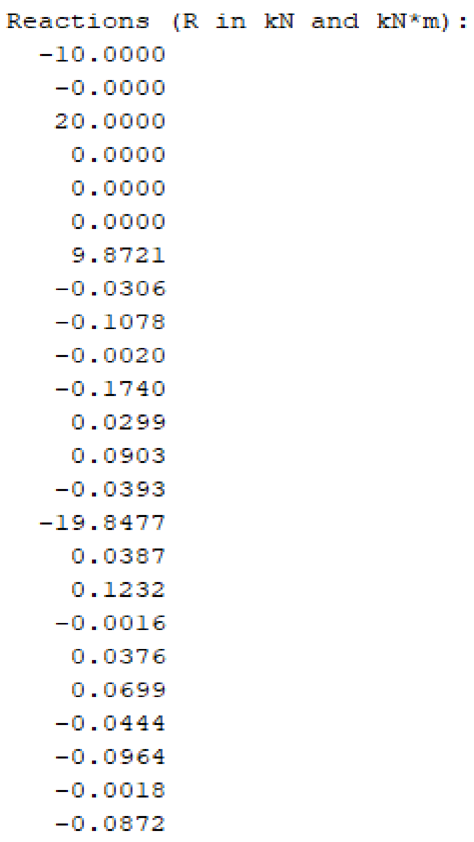
Figure 40: Displacements and rotation at node 4

After all boundary conditions given by the user the program can calculate the global nodal displacement vector and global nodal force vector. Also, the program calculates the nodal displacement vectors for each element and element force vectors for each element. These results are given below. And these results are checked with solution manual. These vectors are given in the pop-up screens and given in the command window on MATLAB.



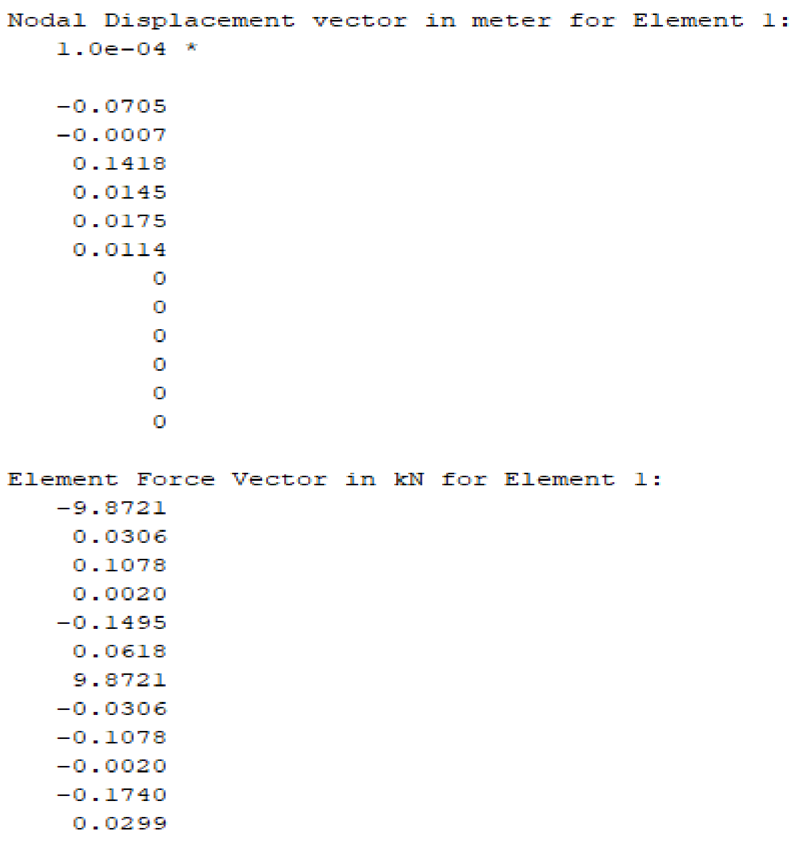
```
Node Displacements and Rotations (U in m and rad):
   1.0e-04 *

   -0.0705
   -0.0007
    0.1418
    0.0145
    0.0175
    0.0114
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
         0
```

Figure 41: Global nodal displacement vector



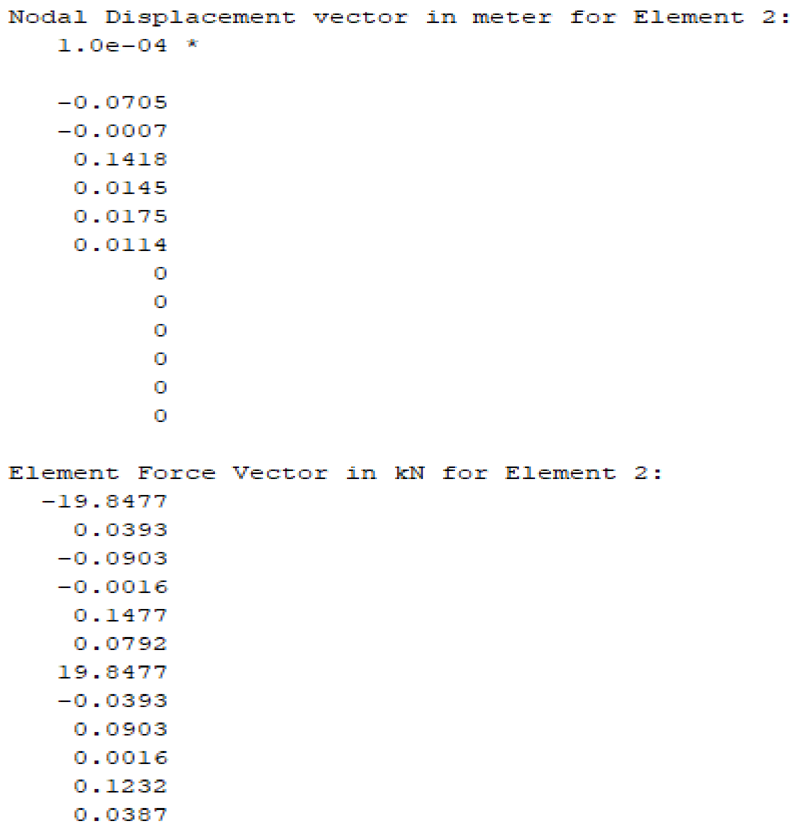
```
Reactions (R in kN and kN*m):
  -10.0000
   -0.0000
   20.0000
    0.0000
    0.0000
    0.0000
    9.8721
   -0.0306
   -0.1078
   -0.0020
   -0.1740
    0.0299
    0.0903
   -0.0393
  -19.8477
    0.0387
    0.1232
   -0.0016
    0.0376
    0.0699
   -0.0444
   -0.0964
   -0.0018
   -0.0872
```

Figure 42: Global nodal force vector



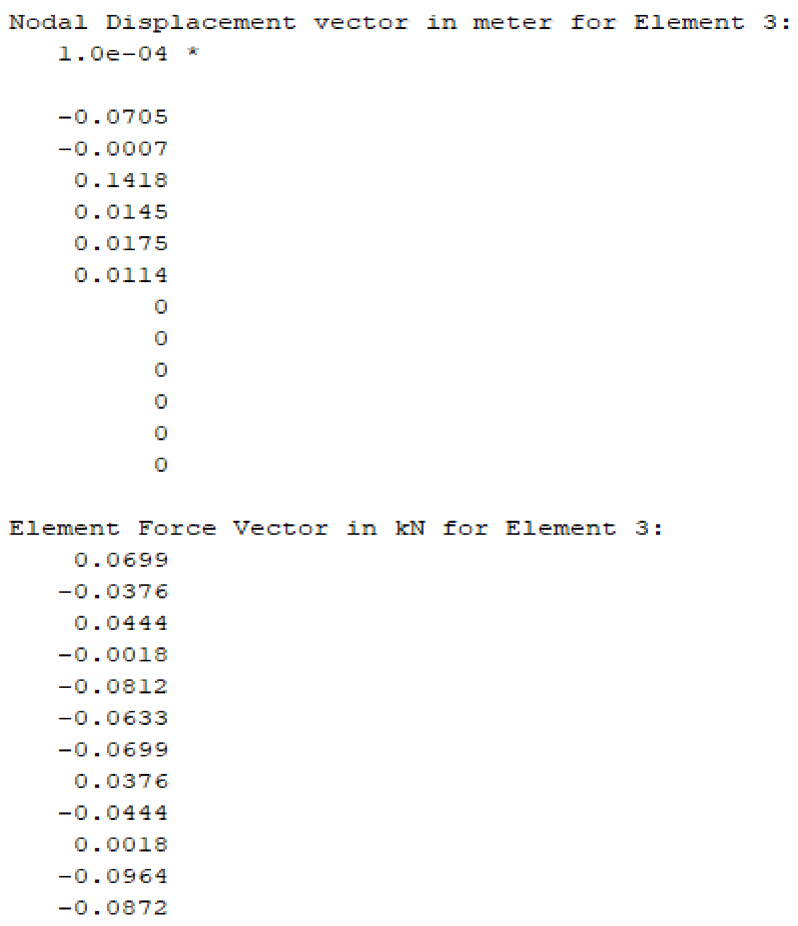
```
Nodal Displacement vector in meter for Element 1:
   1.0e-04 *

   -0.0705
   -0.0007
    0.1418
    0.0145
    0.0175
    0.0114
         0
         0
         0
         0
         0
         0

Element Force Vector in kN for Element 1:
   -9.8721
    0.0306
    0.1078
    0.0020
   -0.1495
    0.0618
    9.8721
   -0.0306
   -0.1078
   -0.0020
   -0.1740
    0.0299
```

Figure 43: Nodal displacement and element force vectors for element 1



```
Nodal Displacement vector in meter for Element 2:
   1.0e-04 *

   -0.0705
   -0.0007
    0.1418
    0.0145
    0.0175
    0.0114
         0
         0
         0
         0
         0
         0

Element Force Vector in kN for Element 2:
  -19.8477
    0.0393
   -0.0903
   -0.0016
    0.1477
    0.0792
   19.8477
   -0.0393
    0.0903
    0.0016
    0.1232
    0.0387
```

Figure 44: Nodal displacement and element force vectors for element 2



```
Nodal Displacement vector in meter for Element 3:
   1.0e-04 *

   -0.0705
   -0.0007
    0.1418
    0.0145
    0.0175
    0.0114
         0
         0
         0
         0
         0
         0

Element Force Vector in kN for Element 3:
    0.0699
   -0.0376
    0.0444
   -0.0018
   -0.0812
   -0.0633
   -0.0699
    0.0376
   -0.0444
    0.0018
   -0.0964
   -0.0872
```

Figure 45: Nodal displacement and element force vectors for element 3

Also, my program can draw the structure according to given inputs by user. Drawn structure can be seen in figure 46 and can compare with original structure in figure 19. In the drawn structure the node numbers are given in red color in the figure for clearance.

Thanks to the drawn structure in figure 46, users can compare the drawn structure with original, to check that the problem is fully understood by the program or not. We can see that the drawn structure is same as original so it can be said that the program works.

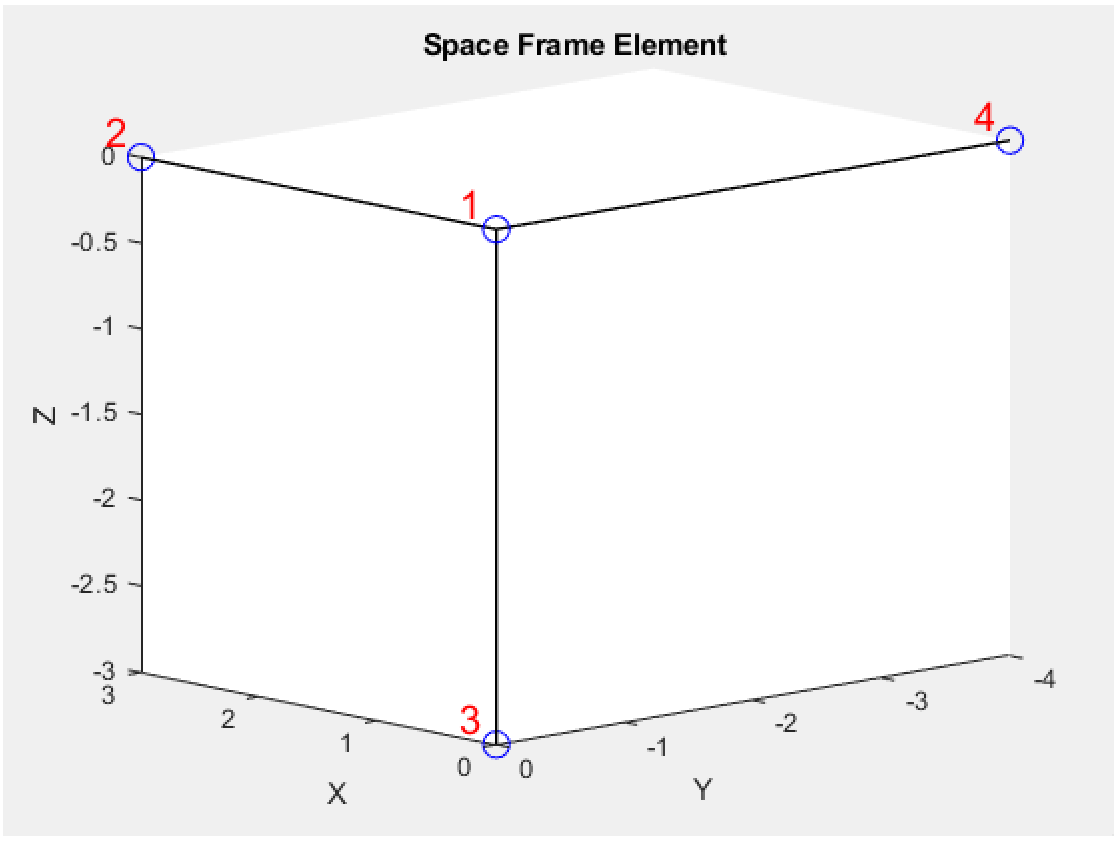


Figure 46: Drawn space frame element by program

Moreover, the program gives the related diagrams which are Axial Force Diagram, Shear Force Diagram in Y Direction, Shear Force Diagram in Z Direction, Torsion Diagram, Bending Moment Diagram along Y axis, Bending Moment Diagram along Z axis. These diagrams are drawn for each element of the structure. In short, the program draws six plots per element.

In the diagrams blue line represents the zero line for a better visualization.

The diagrams for the example problem in figure 19 can be seen below.

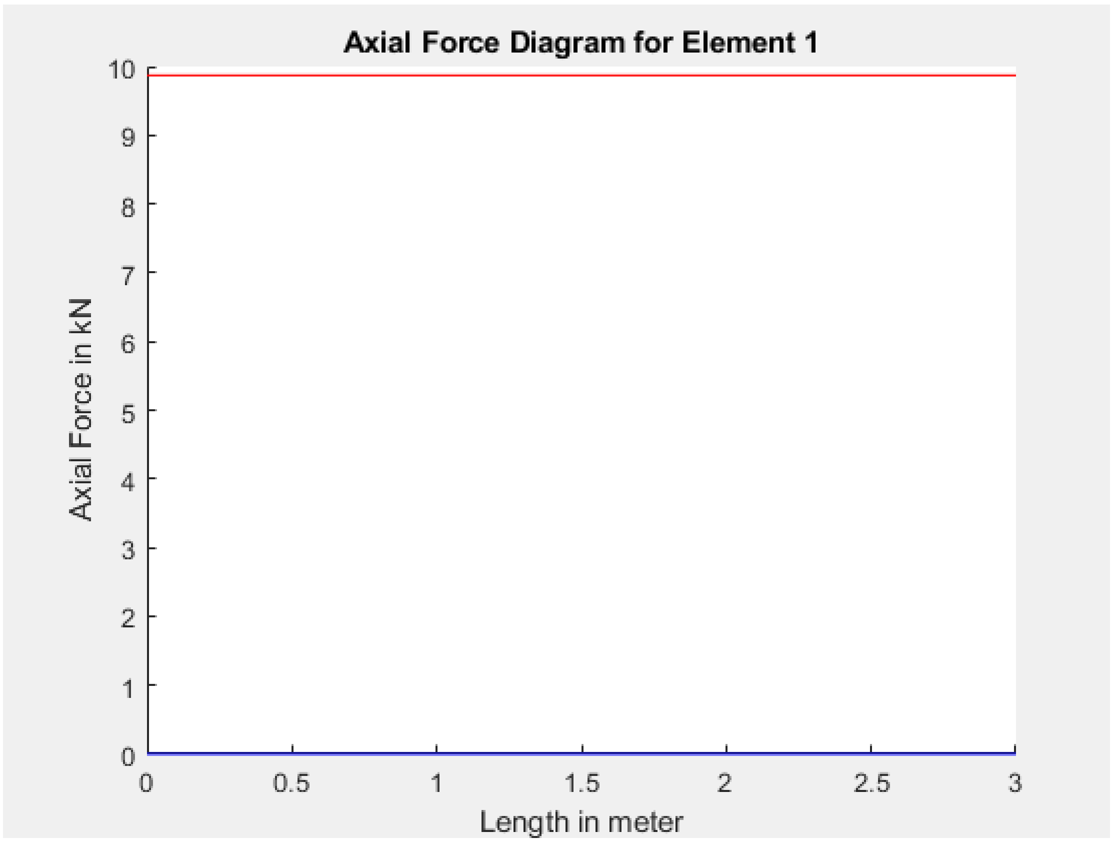


Figure 47: Axial force diagram for element 1

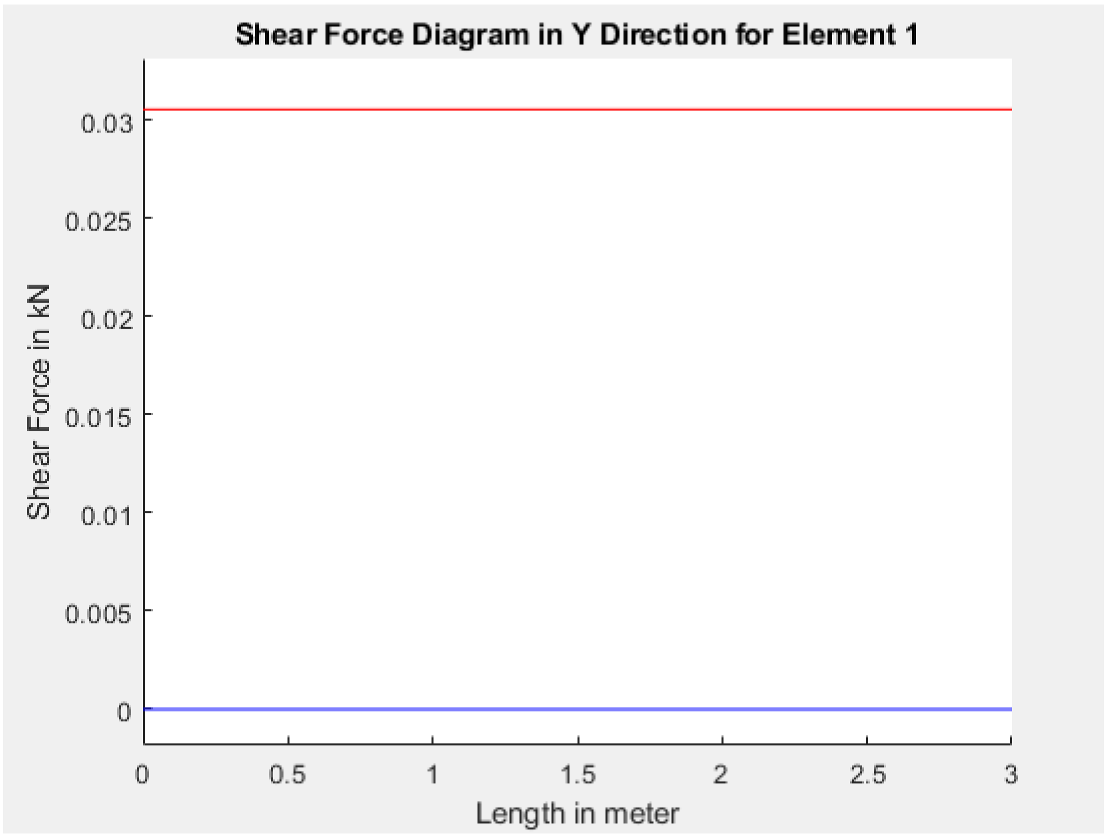


Figure 48: Shear force diagram in Y direction for element 1

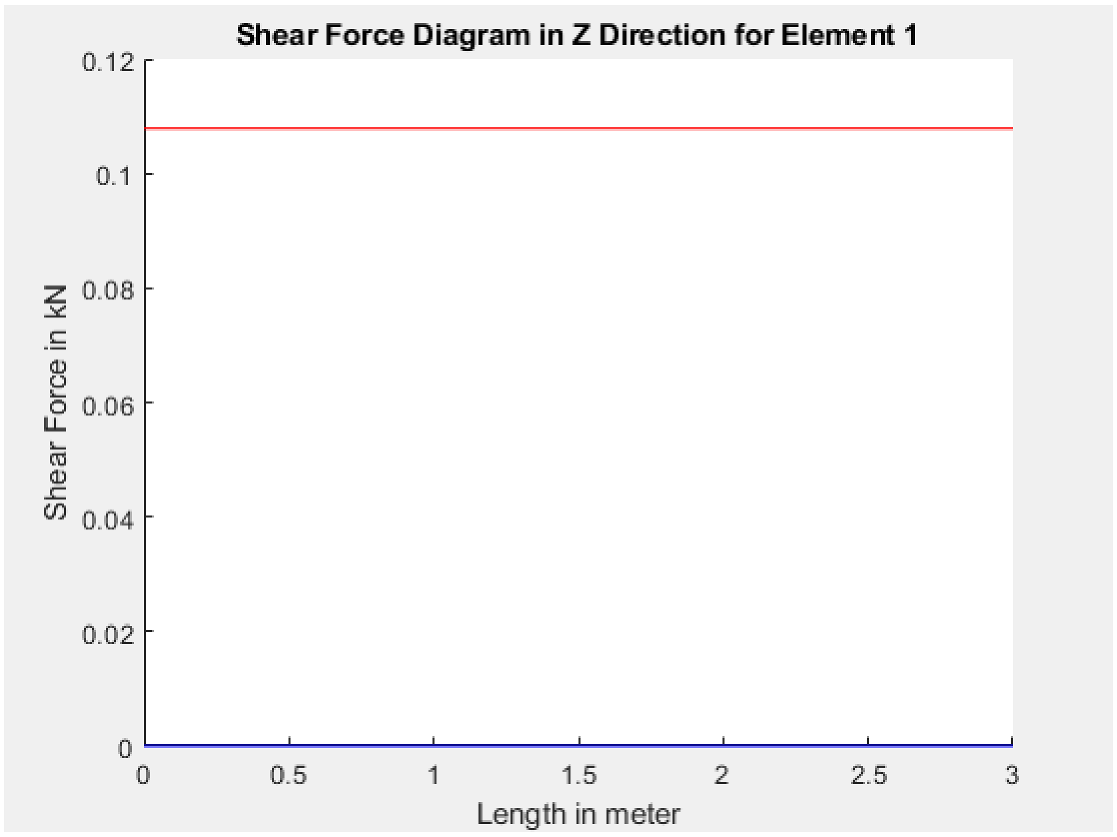


Figure 49: Shear force diagram in Z direction for element 1

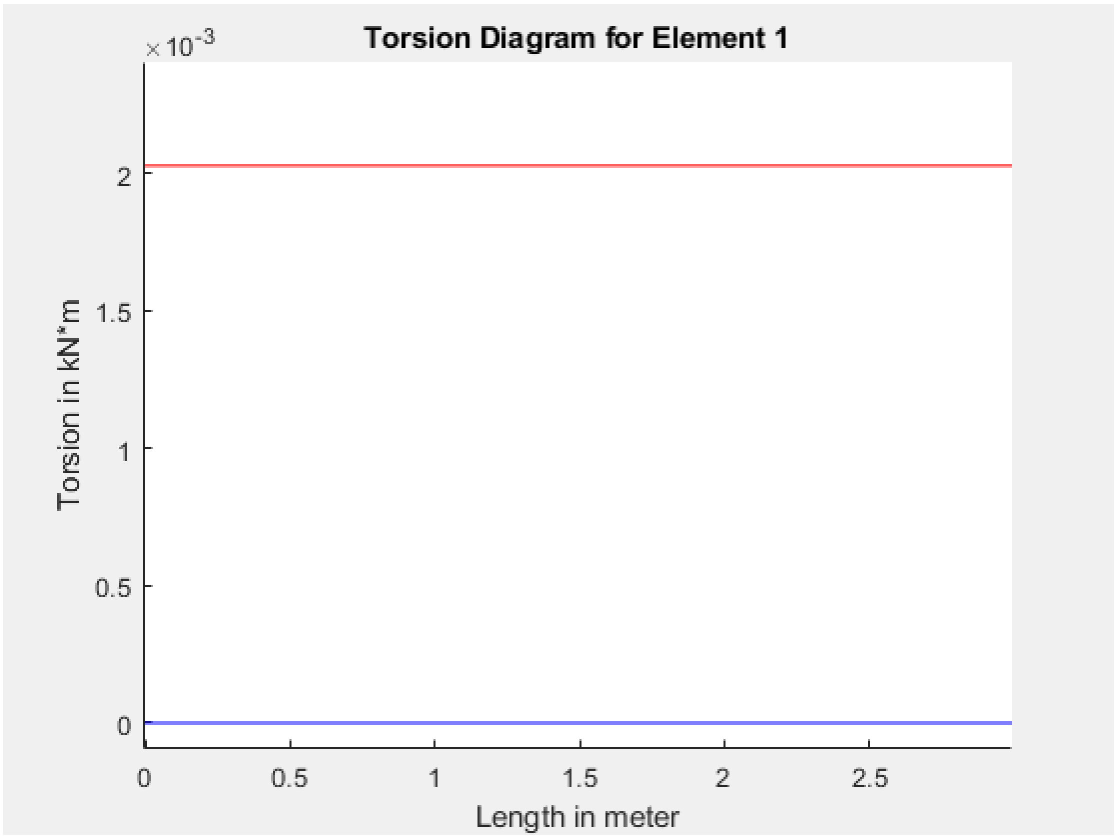


Figure 50: Torsion diagram for element 1

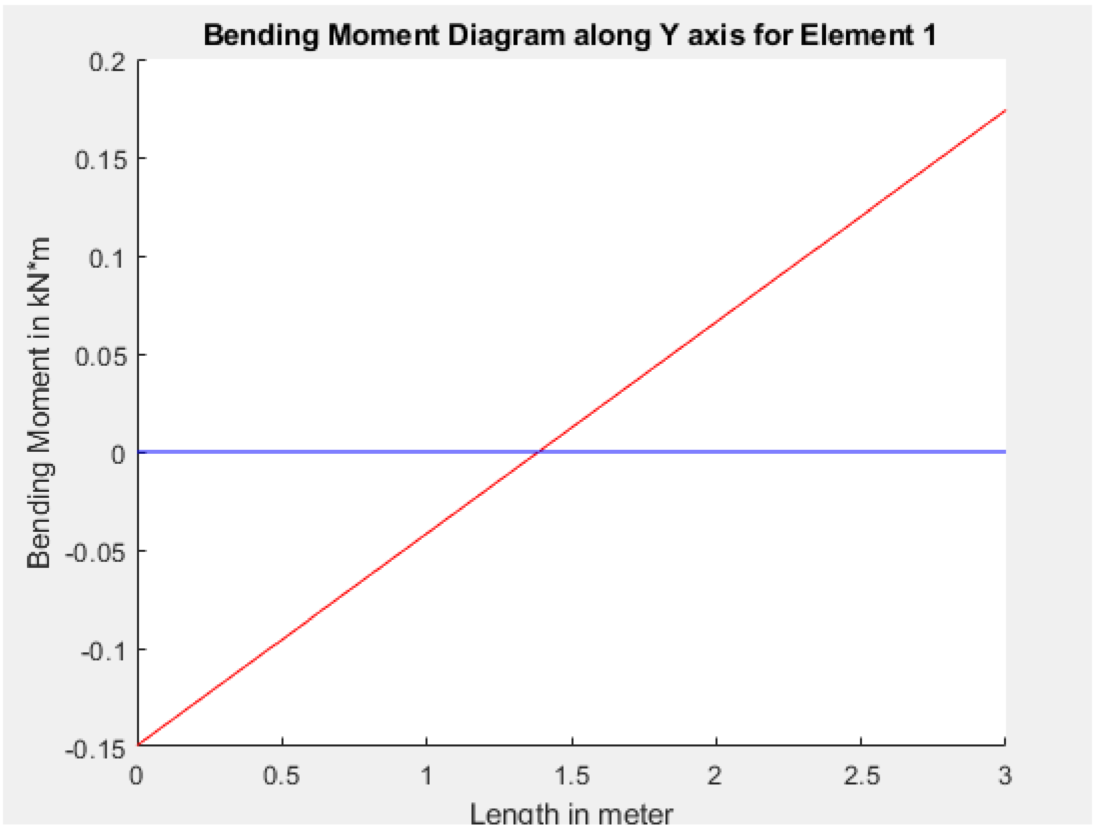


Figure 51: Bending moment diagram along Y axis for element 1

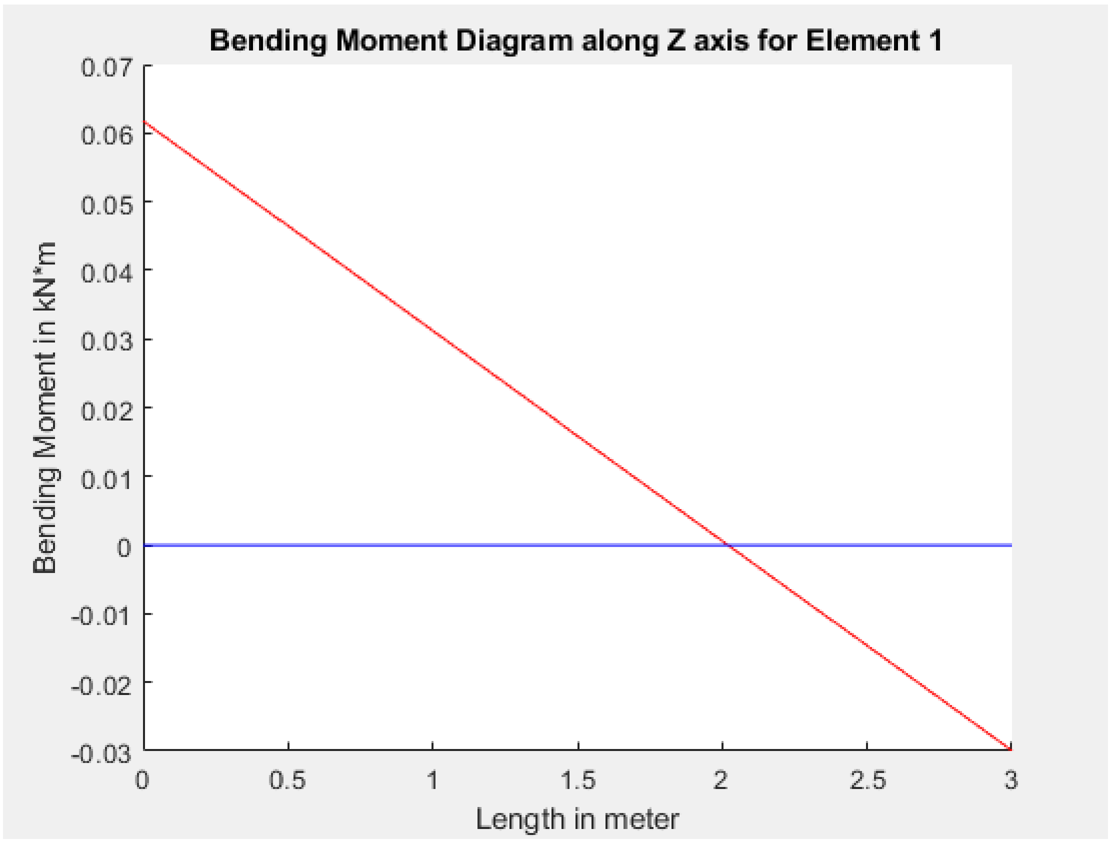


Figure 52: Bending moment diagram along Z axis for element 1

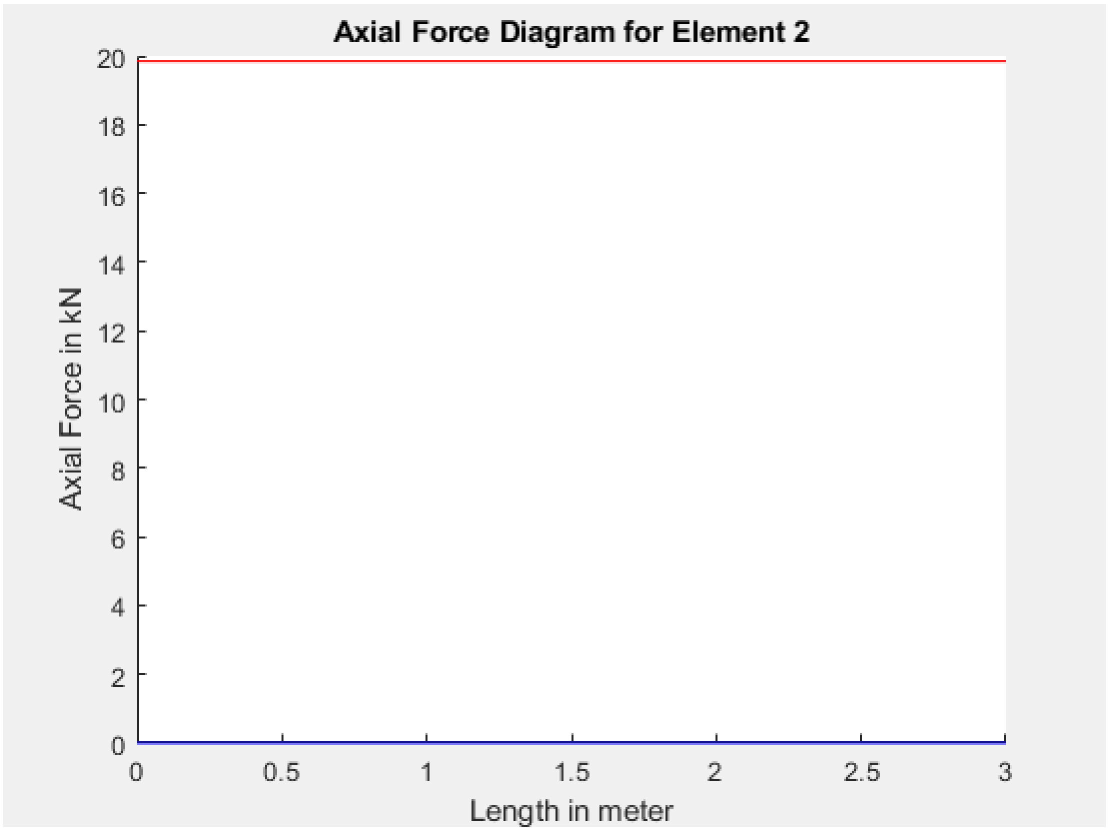


Figure 53: Axial force diagram for element 2

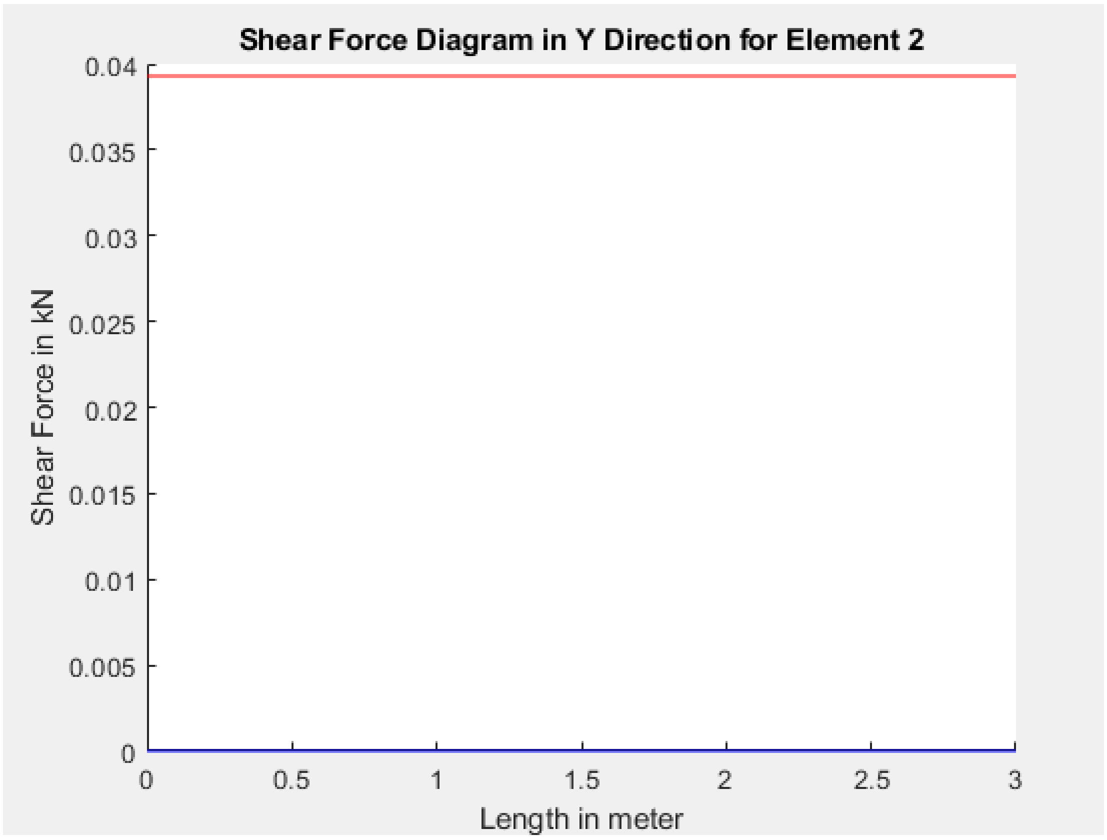


Figure 54: Shear force diagram in Y direction for element 2

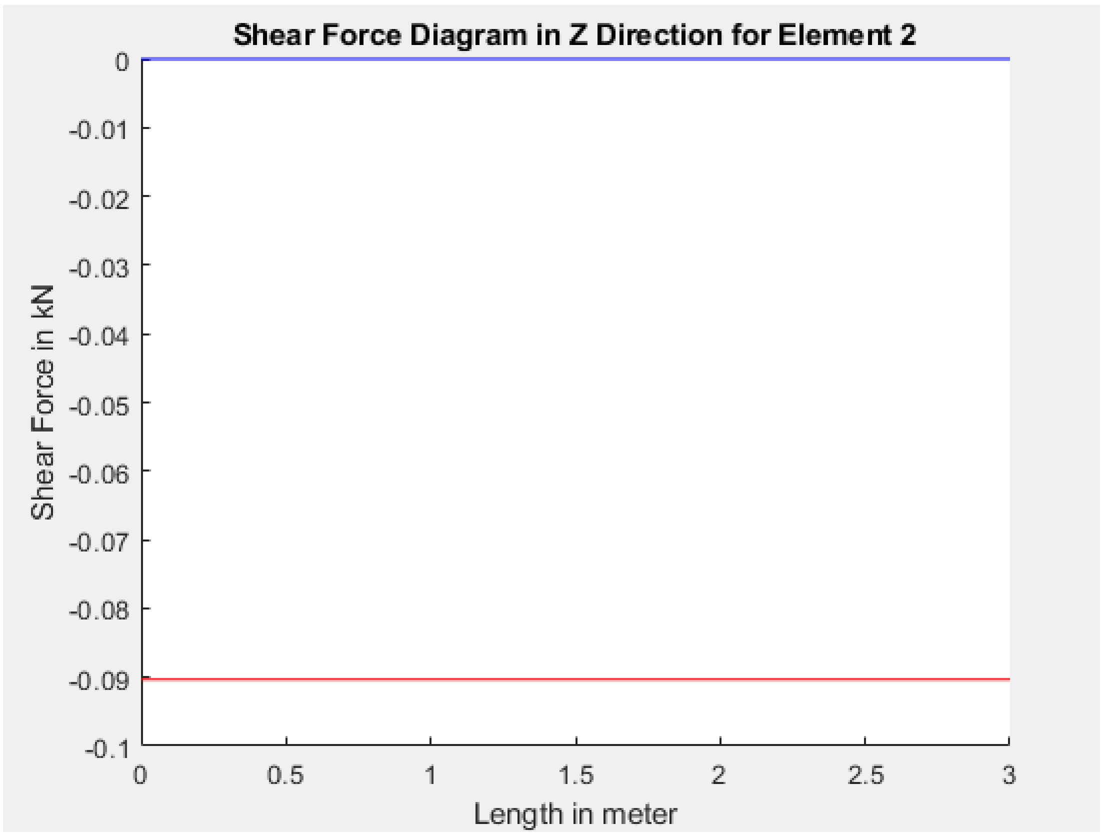


Figure 55: Shear force diagram in Z direction for element 2

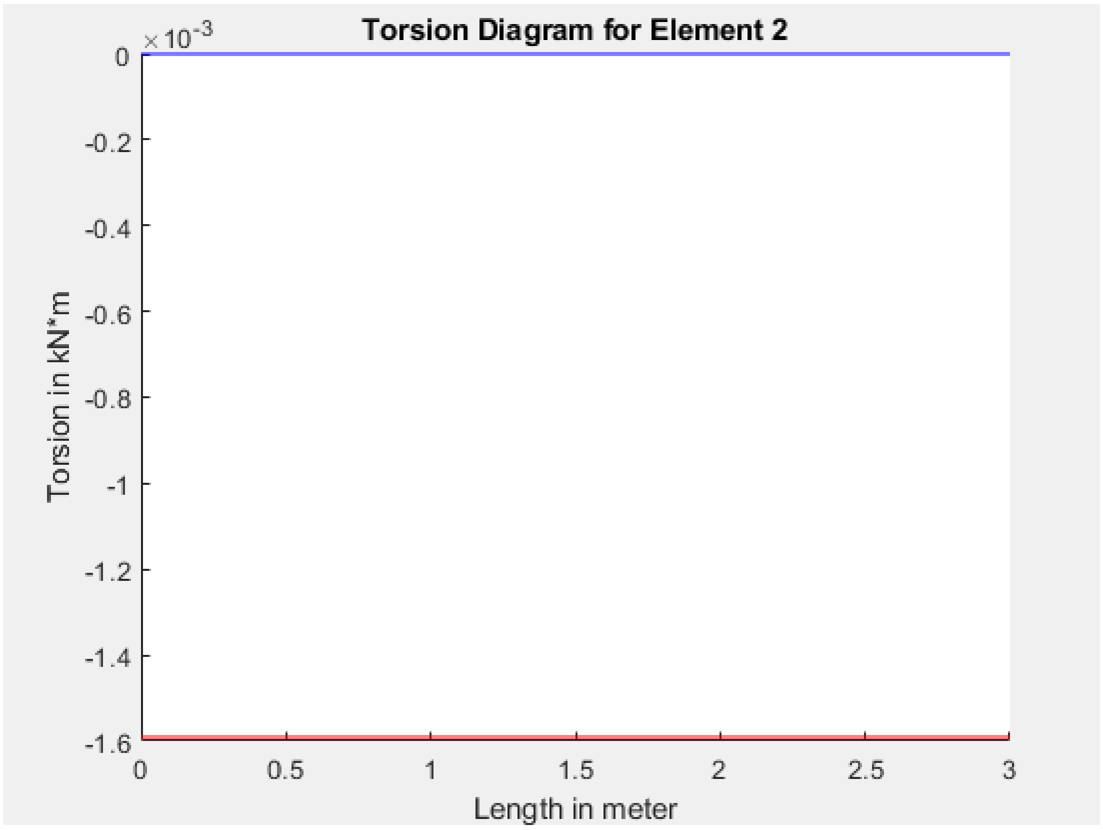


Figure 56: Torsion diagram for element 2

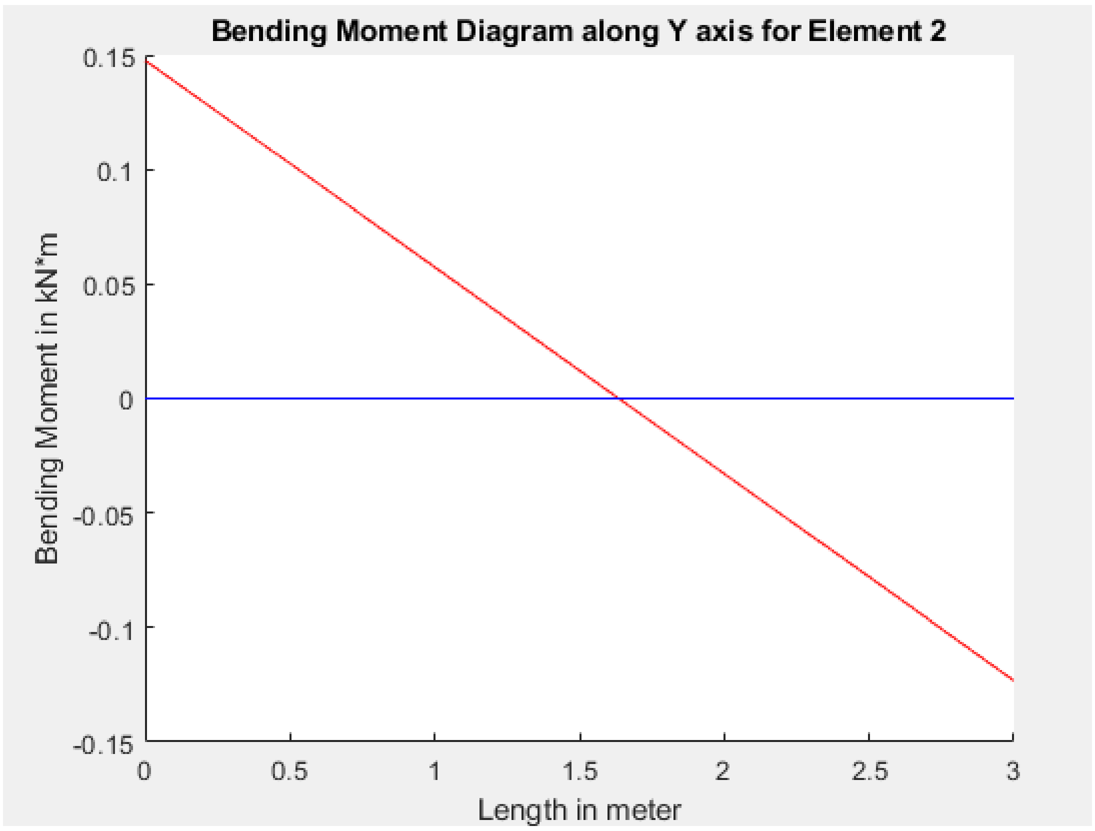


Figure 57: Bending moment diagram along Y axis for element 2

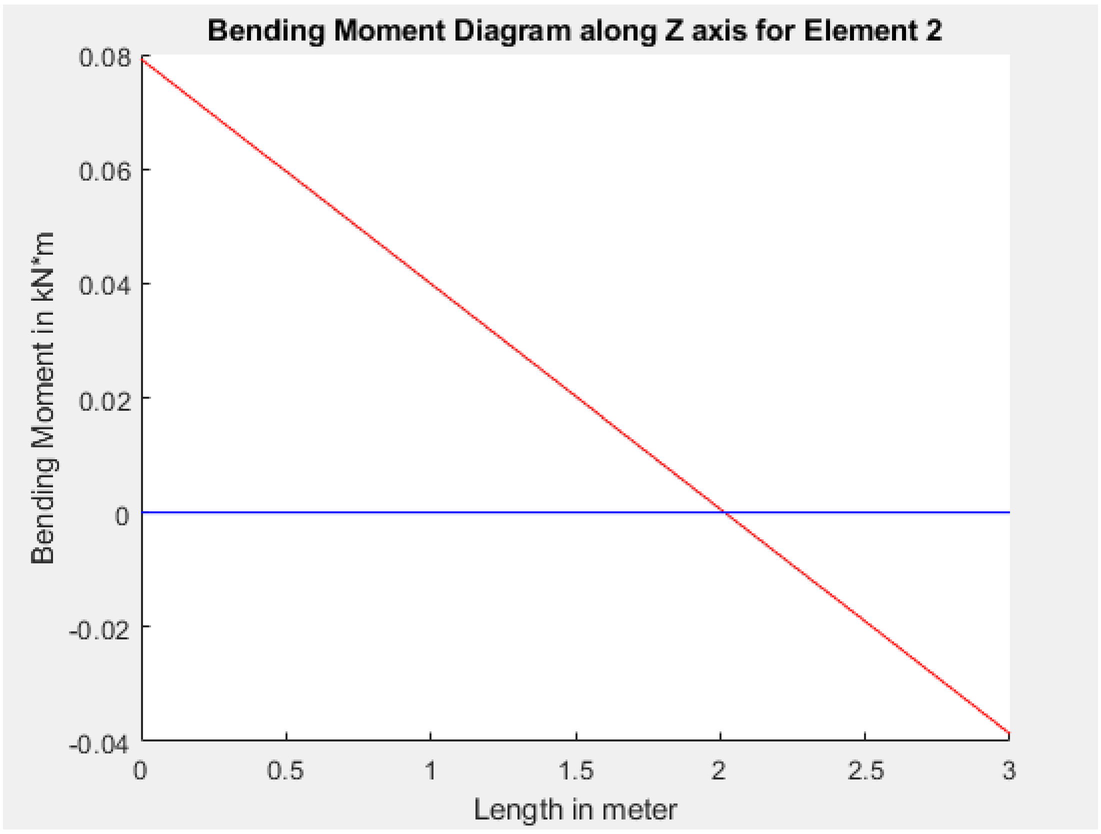


Figure 58: Bending moment diagram along Z axis for element 2

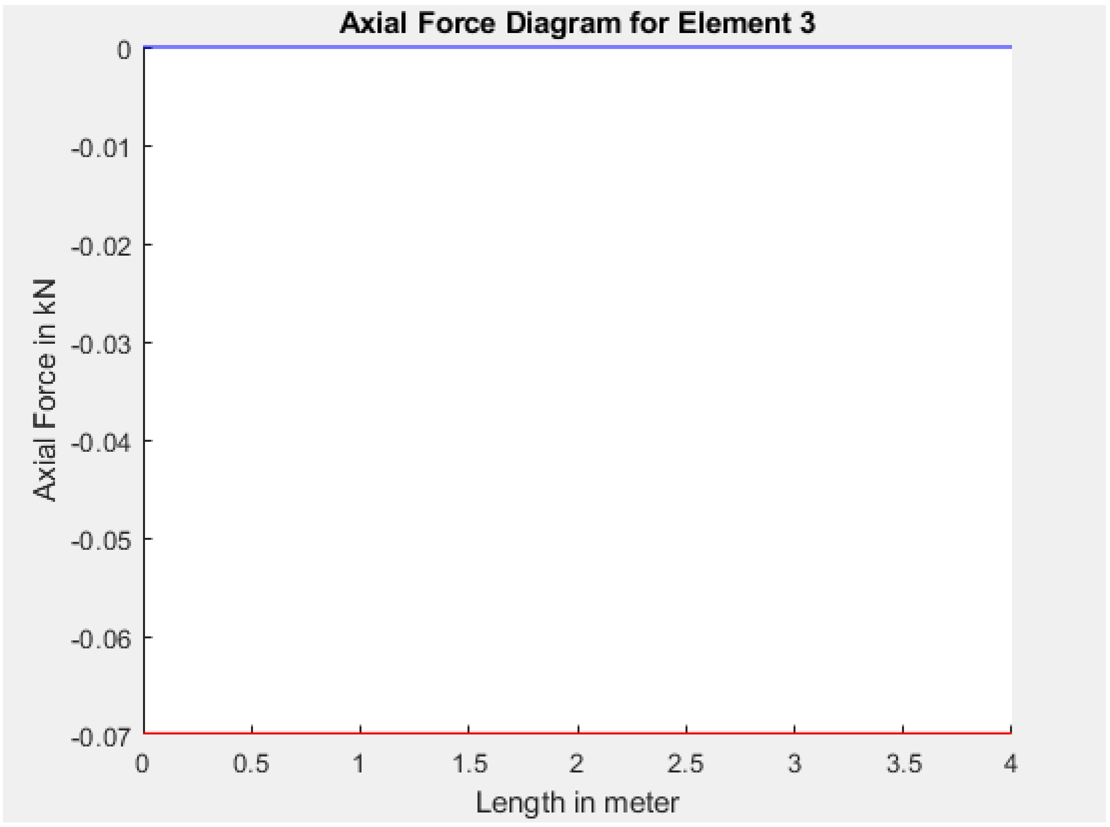


Figure 59: Axial force diagram for element 3

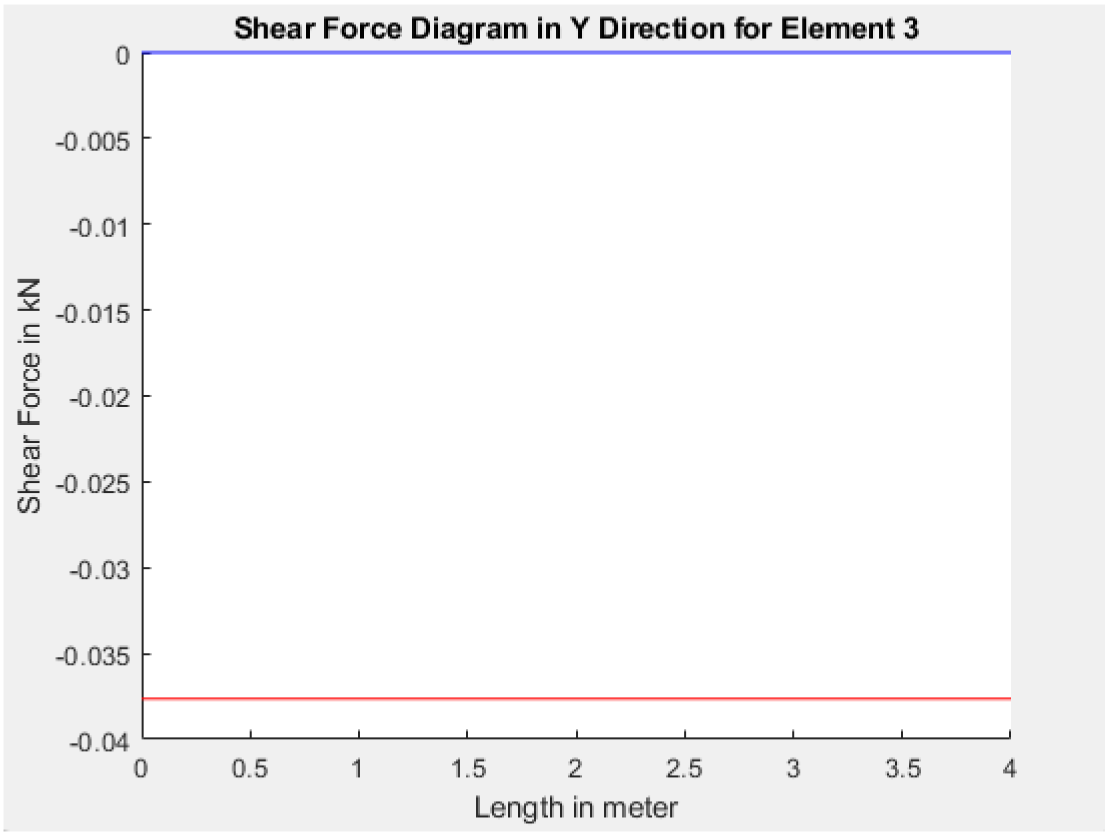


Figure 60: Shear force diagram in Y direction for element 3

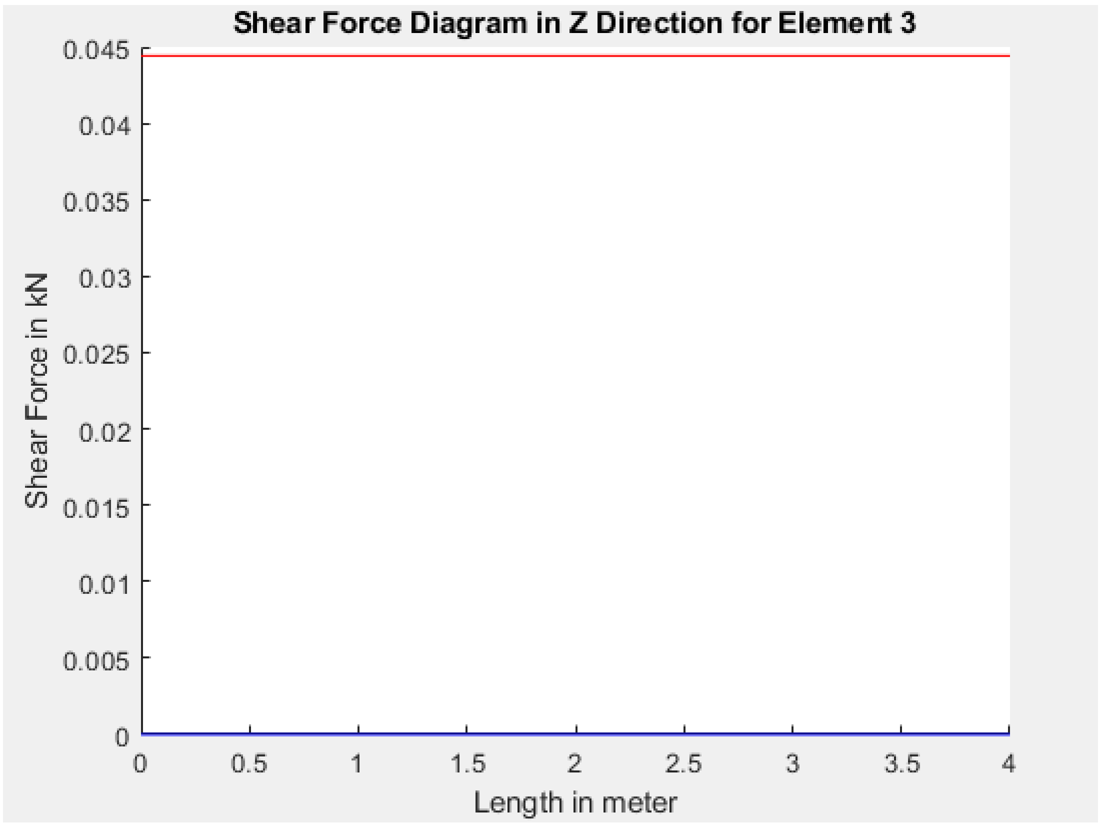


Figure 61: Shear force diagram in Z direction for element 3

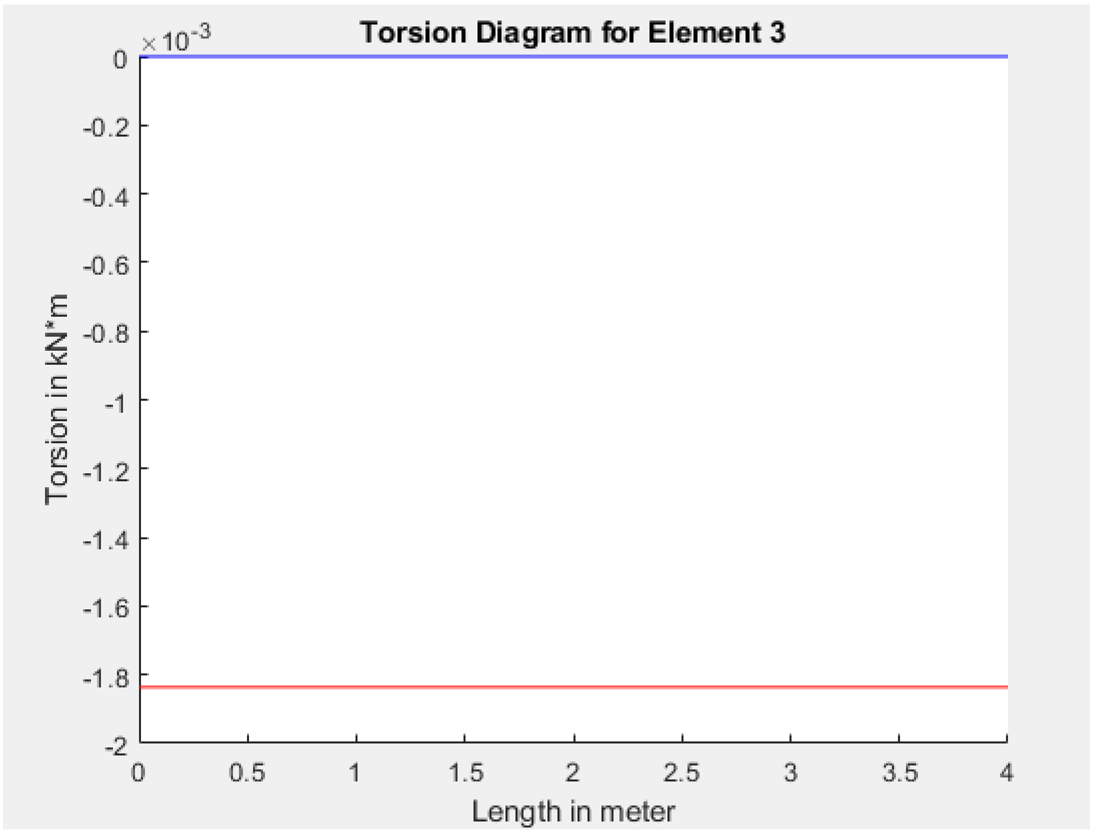


Figure 62: Torsion diagram for element 3

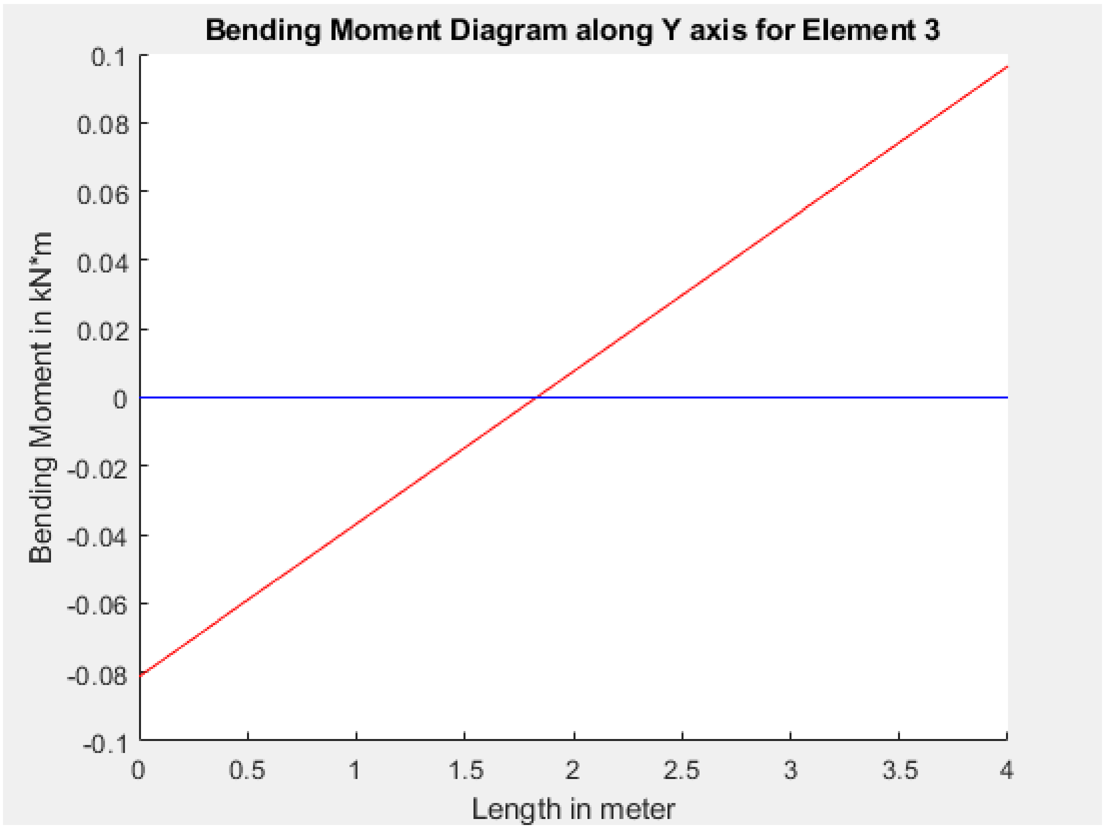


Figure 63: Bending moment diagram along Y axis for element 3

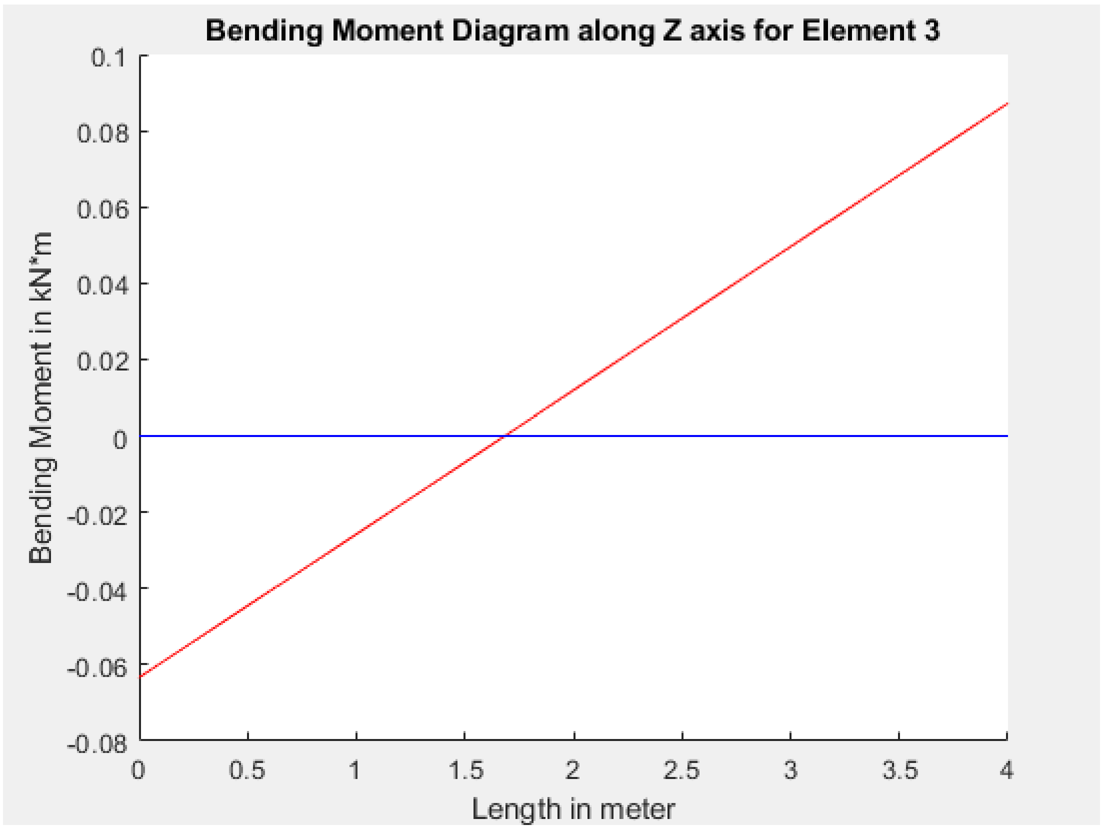


Figure 64: Bending moment diagram along Z axis for element 3

11. CONCLUSION

In this report firstly I researched finite elements analysis and history of the finite elements analysis. To do that I did source search and examined the relative search with finite elements analysis. Accordingly, to my search I understood the importance of finite elements analysis method then I wrote about it in that report with references.

Secondly, starting from springs, which are considered as an introduction to finite element analysis, the stiffness matrices of some important elements were calculated with relevance figures. After calculated element stiffness matrices also, their global stiffness matrices were calculated according to knowledge I had from the course by Mr. Mustafa Özdemir and references which I mentioned.

In the second part of that course, I focused on my own subject which is Spatial Frame Elements. In this part of the course, I wrote code in a user friendly on the MATLAB program. This code I wrote can be used to solve simple or complex problems involving Space Frame Elements.

My code consists of a main script and 11 function scripts that help to work the main script. 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. These scripts can be seen in detail and explained in the heading 9.

After explaining the code in detail and step by step in heading 9. I solved an example problem with my code for a better understanding of the working of my code from my reference book, which I mentioned in heading 10, then I compared the results to check the accuracy of my code. And I found that I achieved the same results as the reference book.

Since this project involved writing a MATLAB code for analyzing spatial frame structures, cost analysis is not applicable.

In conclusion, creating and testing the computational framework for analyzing spatial frame structures has shown that it is both strong and accurate. By carefully using finite element methods and MATLAB for simulation and analysis, I have built a reliable tool to study how spatial frame structures behave under different loads. Testing with Example 10.1 from "MATLAB Guide to Finite Elements: An Interactive Approach" by P. I. Kattan (2003) has confirmed that the results are precise, proving that the algorithms work well.

REFERENCES

[1]: D.V. Hutton, Fundamentals of Finite Element Analysis, McGraw-Hill,2004

[2]: Singiresu S. Rao, The Finite Element Method In Engineering, Fourth Edition

[3]: Dary L. Logan, A First Course in the Finite Element Method, Fourth Edition

[4]: Sennett, R., Matrix Analysis of Structures, Prentice Hall, 1994

[5]: P.I. Kattan, Matlab Guide to Finite Elements An Interactive Approach, 2003

[6]: https://www.mathworks.com/