MARMARA UNIVERSITY
FACULTY OF ENGINEERING

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

Fatih Mirveyisoğlu

GRADUATION PROJECT REPORT
Department of Mechanical Engineering

Supervisor
Prof. Dr. Mustafa ÖZDEMİR

ISTANBUL, 2024

MARMARA UNIVERSITY
FACULTY OF ENGINEERING

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

by

Fatih Mirveyisoğlu

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) .....Fatih MİRVEYİSOĞLU
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 Fatih Mirveyisoğlu

CONTENTS

ABSTRACT

The project "Development of a computer program for finite element analysis and design of spatial truss structures" goals to satisfy the growing need in structural engineering for sophisticated tools to empower the productivity of analysis of spatial truss structures. The main aim is to create a reliable computer program that combines efficient design algorithms with finite element analysis methods to offer engineers a truss structural analysis tool.

For the testing cases and validation, the things that program capable of are shown, highlighting its accuracy and reliability in the analysis of spatial truss structures. Real and practical cases are presented to simulate the applications of the program, underlining its contribution to the optimization of truss structural designs in different engineering contents.

ÖZET

"Sonlu elemanlar analizi ve mekansal kafes kiriş yapılarının tasarımı için bir bilgisayar programının geliştirilmesi" projesi, mekansal kafes kiriş yapılarının analizinin verimliliğini güçlendirmek için yapısal mühendislikte karmaşık araçlara yönelik artan ihtiyacı karşılamayı amaçlamaktadır. Temel amaç, verimli tasarım algoritmalarını sonlu elemanlar analiz yöntemleriyle birleştirerek mühendislere bir kafes kiriş yapısal analiz aracı sunan güvenilir bir bilgisayar programı oluşturmaktır.

Test durumları ve doğrulama için, programın kapasitesi gösterilir ve mekansal kafes yapıların analizindeki doğruluğu ve güvenilirliği vurgulanır. Programın uygulamalarını simüle etmek için gerçek ve pratik örnekler sunulmakta, farklı mühendislik içeriklerindeki kafes yapı tasarımlarının optimizasyonuna katkısı vurgulanmaktadır.

SYMBOLS

f: Axial force

A: Cross-sectional area

$u(x)$: Displacement function

u: Element displacement

$[k_e]$: Element stiffness matrix

$[K]$: Global stiffness matrix

$N(x)$: Inpolation function

L: Length

P: Applied Load

E: Elastic Modulus

J: Polar moment of inertia

I: Principal moment of inertia

ρ: Radius of deflected curve

G: Shear modulus

k: Spring stiffness

δ: Spring deformation

ε_x: Strain in axial direction

σ_x: Stress in axial direction

T: Torque

Φ: Twist angle

$[R]$: Transformation matrix

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

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

LIST OF FIGURES

1. INTRODUCTION

The finite element method, also known as finite element analysis, is a computational technique used to find approximate solutions to boundary condition problems. Simply explained as, the boundary condition problem is a problem which needs some information about at any point of valid domain created for problem. By using this boundary information, problems can be solved for system of equations. Another name of boundary problem is field problem. Fields are the regions where the problem is stated. In most cases fields refer to physical structures. The field variable is the dependent variable where the problem stated is determined by the differential equation systems. There are some types of problems that boundary variables may vary. Here are some examples of different field variables; physical displacements, temperatures, and heat fluxes [1].

When a difficult and complicated problem is faced, Finite Element Method basically changes the problem to a simpler version. This concept is the basic idea of finite element method. By changing the real problem to a simpler version, the solution can only be found approximately [2].

1.1 A Brief History of Finite Element Method

The name Finite Element Method might be a new name, but the methodology and idea of Finite Element Method goes a long way through Archimedes. Archimedes calculated the area of a face of a unique body, divided the face into triangles so he could calculate the area. He used triangles because he knew how to calculate the area of a triangle. As mentioned before, in the introduction part, Archimedes used the idea of finite element method which is changing complicated problems into a simpler version.

In the late 19th century, the idea of finite element method became formulations. In most boundary condition problems has a solution created by differential equations that requires formulations.

In 1851, Schellbach used finite element analysis concepts to find the equation of the surface of an area enclosed by a closed curve in space. He divided the surface into triangles and wrote an expression using finite differences for the entire area. At the time, Schellbach suggested no other uses for this idea. However, nowadays, finite element analysis is widely used to simplify calculations involving differential equations. In finite element analysis, we replace these complex equations with simpler polynomial equations to obtain more practical solutions.

In 1941, people started using finite element analysis for plane elasticity and bending problems. The finite element method we use today probably came from a scientist named Richard Courant. In his 1943 paper, he analyzed the torsional strength of a hollow shaft. He divided the entire plane into triangles like Archimedes and created a stress function. From the nodal values, he calculated stress functions for each triangle. According to some research, Richard Courant created modern finite element method [3].

2. FINITE ELEMENT ANALYSIS

2.1 Linear Spring Element

Spring is designed to carry the loadings applied on axial direction only. Applied loading creates elongation on the spring. The amount of elongation depends on the spring constant, also called spring stiffness, and the amount of load applied. Consider a system as shown in figure 1 [1].

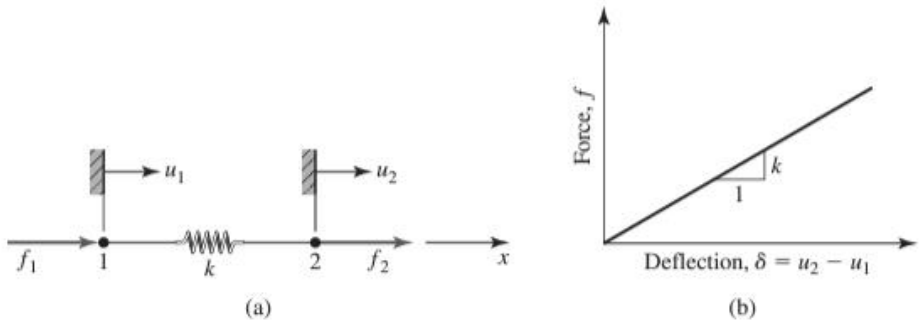


Figure 1. (a) Nodes, elongations, forces shown on a linear spring element. (b) Load versus deflection curve [3].

Deformation can be expressed by assuming nodal displacements are zero:

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

And the axial force is:

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

Depending on equilibrium and equations 1 and 2 we can write:

$$f_1 = -k(u_2 - u_1) \tag{3a}$$

$$f_2 = k(u_2 - u_1) \tag{3b}$$

We can write equations 3a and 3b in matrix form as:

$$\begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} \tag{4}$$

Here, k matrix is $[k_e]$ element stiffness matrix in element coordinate system:

$$[k_e] = \begin{bmatrix} k & -k \\ -k & k \end{bmatrix} \tag{5}$$

From beginning to this point, we analyzed spring element in element (local) coordinate system. For global coordinate system reference figure 2[1].

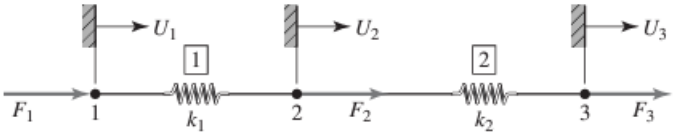


Figure 2.Assembly of 2 spring elements [3].

In Figure 2 node numbers, nodal displacements, and nodal forces are shown. Here is the free body diagram of this system:

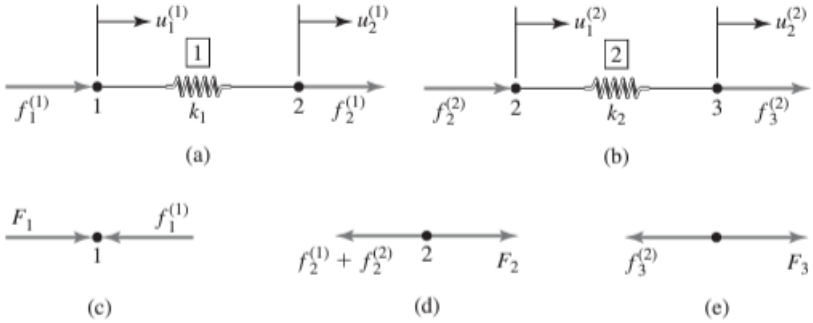


Figure 3. Free body diagram of system shown in figure 2 [3].

Assume the system in figure 2 to be in equilibrium. For each spring, the spring constants are k_1 and k_2 respectively, using equation 4, we can create the equations below:

$$\begin{bmatrix} k_1 & -k_1 \\ -k_1 & k_1 \end{bmatrix} \begin{Bmatrix} {u_1}^{(1)} \\ {u_2}^{(1)} \end{Bmatrix} = \begin{Bmatrix} {f_1}^{(1)} \\ {f_2}^{(1)} \end{Bmatrix} \tag{6a}$$

$$\begin{bmatrix} k_2 & -k_2 \\ -k_2 & k_2 \end{bmatrix} \begin{Bmatrix} {u_1}^{(2)} \\ {u_2}^{(2)} \end{Bmatrix} = \begin{Bmatrix} {f_1}^{(2)} \\ {f_2}^{(2)} \end{Bmatrix} \tag{6b}$$

Next step is relating element displacements to system displacements.

$${u_1}^{(1)} = U_1 \tag{7a}$$

$$u_2{}^{(1)} = U_2 \tag{7b}$$

$$u_1{}^{(2)} = U_2 \tag{7c}$$

$$u_2{}^{(2)} = U_3 \tag{7d}$$

We should do the same process for element forces. Refer to the free body diagram of system (figure 3), we can write the global force F_2 as $f_2{}^{(2)} + f_2{}^{(1)} = F_2$ .Then to match the vectors and nodes, we can expand both matrices from 2x2 to 3x3. After this step, global system of equations will be as shown:

$$\begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ F_2 \\ F_3 \end{Bmatrix} \tag{8a}$$

$$[K]\{U\} = \{F\} \tag{8b}$$

Here [K] is global stiffness matrix.

$$[K] = \begin{bmatrix} k_1 & -k_1 & 0 \\ -k_1 & k_1 + k_2 & -k_2 \\ 0 & -k_2 & k_2 \end{bmatrix} \tag{9}$$

2.2 Elastic Bar Element

Elastic bar formulation in finite element analysis differs from the linear spring formulation.
Elastic bar element requires some certain conditions which are:

1. The bar should be straight.
2. The material should obey Hooke's Law.
3. Forces should only apply at the tips of the bar.
4. Axial loading should be applied.

Let's define the bar using a figure:

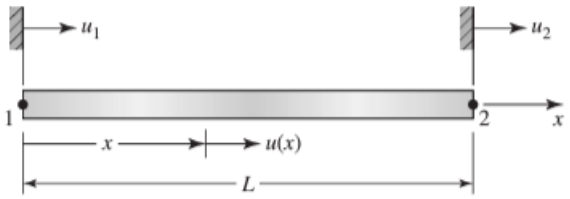


Figure 4.Elastic bar, displacements, length and nodes are shown [3].

Axial displacement at any point on the bar defined as u(x)=x. We can express this boundary variable:

$$u(x) = N_1(x)u_1 + N_2(x)u_2 \tag{10}$$

$N_1(x)$ and $N_2(x)$ are interpolating functions, u_1 and u_2 are nodal displacement variables.

To find the interpolation functions, we need boundary conditions for u(x) and N(x).

$$u(x = 0) = u_1 \; and \; u(x = L) = u_2 \tag{11a}$$

$$N_1(0) = 1 \qquad N_2(0) = 0 \tag{11b}$$

$$N_1(L) = 0 \qquad N_2(L) = 1 \tag{11c}$$

Let's get to the deflection and stiffness matrix for elastic bar element. For a elastic bar element, we know that deflection is given by:

$$\delta = \frac{PL}{AE} \tag{12}$$

Here P is loading, L is length, A is cross sectional area, E is modulus elasticity for elastic bar element.

If we obtain it like spring constant, we can say:

$$k = \frac{AE}{L} \tag{13}$$

In axial loading, the strain can be expressed as:

$$\varepsilon_x = \frac{u_2 - u_1}{L} \tag{14}$$

By using Hooke's Law:

$$\sigma_X = E\varepsilon_x = E\frac{u_2-u_1}{L} \tag{15}$$

We know, $P = \sigma_X A,$ depending on this information, we can relate nodal forces to nodal displacements:

$$f_1 = -\frac{AE}{L}(u_2 - u_1) \tag{16a}$$

$$f_2 = -\frac{AE}{L}(u_2 - u_1) \tag{16b}$$

Let's write the equations 16a and 16b in matrix form.

$$\frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} \tag{17}$$

So, the stiffness matrix for elastic bar element is:

$$[k_e] = \frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{18}$$

2.3 Beam Element

One kind of two-dimensional finite element where the local and global coordinates coincide is called a beam element.The parameters of the beam element are length (L), moment of inertia (I), and modulus of elasticity (E). As seen in Figure 5, each beam element has two nodes and is regarded as horizontal. 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 $\widehat{d_{\iota y}}$ and the rotations by $\widehat{\phi_\iota}$. The local nodal forces are given by $\widehat{f_{\iota y}}$ and the bending moments by $\widehat{m_\iota}$'s as shown. We initially neglect all axial effects.

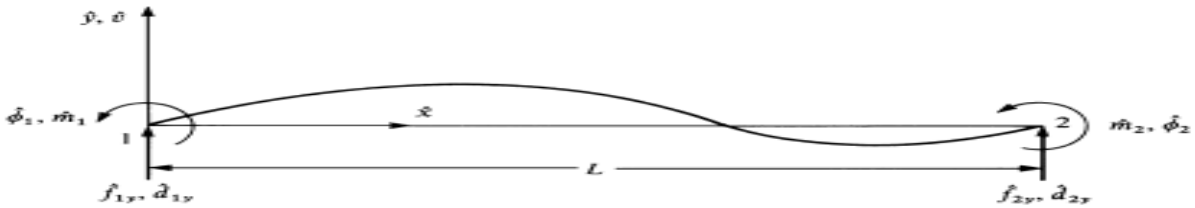


Figure 6. The Beam Element [3].

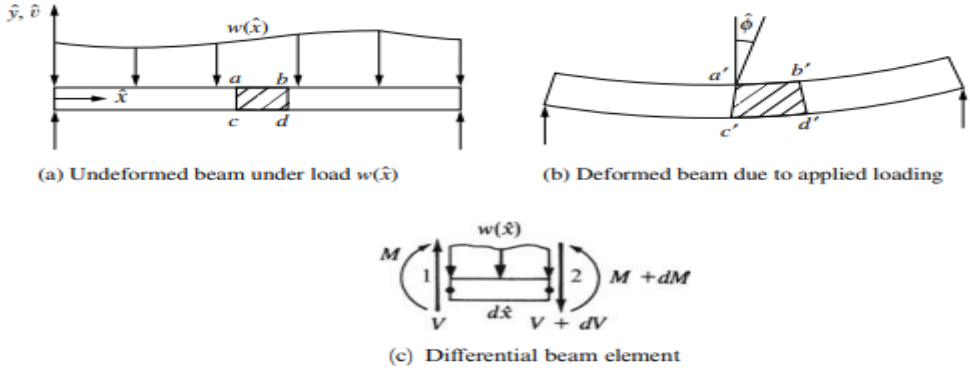


Figure 5. Beam under distributed load [3].

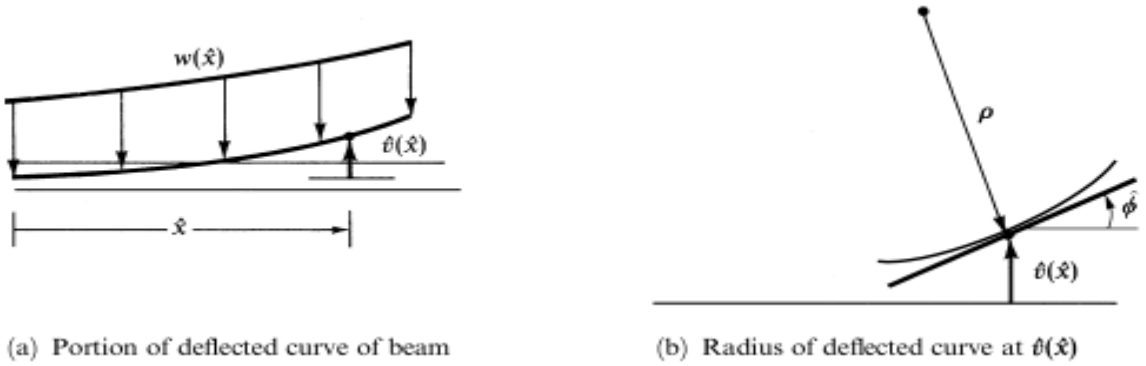
The differential equation is derived by examining the beam shown in Figure 9 that is subjected to a distributed loading represented by the symbol w(x) (force per unit length). The following connection can be found by applying the concepts of force and moment equilibrium to a differential element of the beam.

$$\sum F_y = 0: V - (V + dV) - w(\hat{x})dx = 0 \tag{19}$$

Simplifying:

$$-wd\hat{x} - dV = 0 \ \ or \ \ w = -\frac{dV}{d\hat{x}} \tag{20}$$

$$\sum M_2 = 0: \ -Vdx + dM + w(\hat{x})d\hat{x}\left(\frac{d\hat{x}}{2}\right) = 0 \ \ or \ \ V = \frac{dM}{d\hat{x}} \tag{21}$$



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

Figure 7. Portion of deflected curve of beam [3].

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

$$\kappa = \frac{1}{\rho} = \frac{M}{EI} \tag{22}$$

Where ρ is the radius of deflected curve shown in figure 7, $\hat{v}$ is the transverse displacement function in the $\hat{y}$ direction E is the modulus of elasticity, and I is the principal moment of inertia.

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

$$\kappa = \frac{d^2\hat{v}}{d\hat{x}^2} = \frac{M}{EI} \tag{23}$$

Solving the equation above for M

$$\frac{d^2}{d\hat{x}^2}\left(EI\frac{d^2\hat{v}}{d\hat{x}^2}\right) = -w(\hat{x}) \tag{24}$$

Finally, we get the below equilibrium for EI.

$$EI\frac{d^4\hat{v}}{d\hat{x}^4} = 0 \tag{25}$$

Firstly, we should represent the beam by labeling nodes and elements.

Assume transverse displacement function to be:

$$\hat{v}(\hat{x}) = a_1\hat{x}^3 + a_2\hat{x}^2 + a_3\hat{x} + a_4 \tag{26}$$

For the beam element, at each node, the number of degrees of freedom is 4. The transverse displacement function expressed by nodal degrees of freedom which are $\widehat{d_{1y}}$, $\widehat{d_{2y}}$, $\widehat{\phi_1}$, $\widehat{\phi_2}$.

$$\hat{v}(0) = \widehat{d_{1y}} = a_4 \tag{27}$$

$$\frac{d\hat{v}(0)}{d\hat{x}} = \widehat{\phi_1} = a_3 \tag{28}$$

$$\hat{v}(L) = \widehat{d_{2y}} = a_1 L^3 + a_2 L^2 + a_3 L^3 + a_4 \tag{29}$$

$$\frac{d\hat{v}(L)}{d\hat{x}} = \widehat{\phi_2} = 3a_1 L^2 + 2a_2 L + a_3 \tag{30}$$

Where $\hat{\phi}$ =d$\hat{v}/d\hat{x}$ for the tiny rotation $\hat{\phi}$. By solving the equations 27,28,29,30, we get:

$$\hat{v}(\hat{x}) = a_1 \hat{x}^3 + a_2 \hat{x}^2 + a_3 \hat{x}^3 + a_4 \tag{31}$$

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

$$+\left[-\frac{3}{L^2}(\widehat{d_{1y}} - \widehat{d_{2y}}) - \frac{1}{L}\left(2\widehat{\phi_1} + \widehat{\phi_2}\right)\right]\hat{x}^2 + \widehat{\phi_1}\hat{x} + \widehat{d_{\iota y}} \tag{32}$$

Writing the equations 31,32,33 in matrix form.

$$\hat{v} = [N]\{\hat{d}\} \tag{33}$$

Where:

$$\{\hat{d}\} = \begin{Bmatrix} \widehat{d_{1y}} \\ \widehat{\phi_1} \\ \widehat{d_{2y}} \\ \widehat{\phi_2} \end{Bmatrix} \tag{34}$$

$$[N] = [N_1\ N_2\ N_3\ N_4] \tag{35}$$

$$N_1 = \frac{1}{L^3}(2\hat{x}^3 - 3\hat{x}^2 L + L^3) \tag{36}$$

$$N_2 = \frac{1}{L^3}(\hat{x}^3 L - 2\hat{x}^2 L^2 + \hat{x} L^3) \tag{37}$$

$$N_3 = \frac{1}{L^3}(-2\hat{x}^3 + 3\hat{x}^2 L) \tag{38}$$

$$N_4 = \frac{1}{L^3}(\hat{x}^3 L - \hat{x}^2 L^2) \tag{39}$$

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

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

$$\varepsilon_x(\hat{x}, \hat{y}) = \frac{d\hat{u}}{d\hat{x}} \tag{40}$$

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

$$\hat{u} = -\hat{y}\frac{d\hat{v}}{d\hat{x}} \tag{41}$$

The basic presumption is that the beam's cross sections, such as cross section ABCD, which are initially planar before bending deformation, would remain planar after deformation. Essentially, the angle ($\frac{d\hat{v}}{d\hat{x}}$) represents a tiny angular rotation. By using the equations 40 and 41 above, we get:

$$\varepsilon_x(\hat{x}, \hat{y}) = -\hat{y}\frac{d^2\hat{v}}{d\hat{x}^2} \tag{42}$$

For bending moment and shear force, we can obtain:

$$\hat{m}(\hat{x}) = EI\frac{d^2\hat{v}}{d\hat{x}^2} \; and \; \hat{V} = EI\frac{d^3\hat{v}}{d\hat{x}^3} \tag{43}$$

Depending on figures 5 and 6, sign conventions, illustrations for moments, with the equations given above for the beam element, we can use direct equilibrium approach.

$$\widehat{f_{1y}} = \hat{V} = EI\frac{d^3\hat{v}(0)}{d\hat{x}^3} = \frac{EI}{L^3}\left(12\widehat{d_{1y}} + 6L\widehat{\phi_1} - 12\widehat{d_{2y}} + 6L\widehat{\phi_2}\right) \tag{44}$$

$$\widehat{m_1} = -\hat{m} = -EI\frac{d^2\hat{v}(0)}{d\hat{x}^2} = \frac{EI}{L^3}\left(6L\widehat{d_{1y}} + 4L^2\widehat{\phi_1} - 6L\widehat{d_{2y}} + 2L^2\widehat{\phi_2}\right) \tag{45}$$

$$\widehat{f_{2y}} = -\hat{V} = -EI\frac{d^3\hat{v}(L)}{d\hat{x}^3} = \frac{EI}{L^3}\left(-12\widehat{d_{1y}} - 6L\widehat{\phi_1} + 12\widehat{d_{2y}} - 6L\widehat{\phi_2}\right) \tag{46}$$

$$\widehat{m_2} = \hat{m} = EI\frac{d^2\hat{v}(L)}{d\hat{x}^2} = \frac{EI}{L^3}\left(6L\widehat{d_{1y}} + 2L^2\widehat{\phi_1} - 6L\widehat{d_{2y}} + 4L^2\widehat{\phi_2}\right) \tag{47}$$

Take these equations in matrix form:

$$\begin{Bmatrix} \widehat{f_{1y}} \\ \widehat{m_1} \\ \widehat{f_{2y}} \\ \widehat{m_2} \end{Bmatrix} = \frac{EI}{L^3}\begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix}\begin{Bmatrix} \widehat{d_{1y}} \\ \widehat{\phi_1} \\ \widehat{d_{2y}} \\ \widehat{\phi_2} \end{Bmatrix} \tag{48}$$

So, the stiffness matrix for beam element is,

$$[k_e] = \frac{EI}{L^3}\begin{bmatrix} 12 & 6L & -12 & 6L \\ 6L & 4L^2 & -6L & 2L^2 \\ -12 & -6L & 12 & -6L \\ 6L & 2L^2 & -6L & 4L^2 \end{bmatrix} \tag{49}$$

2.4 Planar Truss Structures

This section will cover the part planar truss structures and the global stiffness matrix for the planar trusses by applying the transformation matrixes to the bar element equations shown in the previous sections.

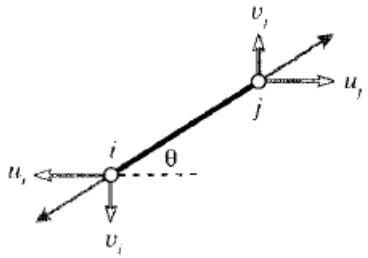


Figure 8. Planar truss element [3].

We will apply the direct assembly method to convert the bar element stiffness matrix into the plane truss element stiffness matrix. Recall the bar element equations as,

$$\frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \end{Bmatrix} \tag{50}$$

We know this equation can be expressed as,

$$\begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} \begin{Bmatrix} u_1^{(e)} \\ u_2^{(e)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(e)} \\ f_2^{(e)} \end{Bmatrix} \tag{51}$$

Now, transformation applied to the above equation which creates:

$$\left[K^{(e)}\right] \begin{Bmatrix} {U_1}^{(e)} \\ {U_2}^{(e)} \\ {U_3}^{(e)} \\ {U_4}^{(e)} \end{Bmatrix} = \begin{Bmatrix} {F_1}^{(e)} \\ {F_2}^{(e)} \\ {F_3}^{(e)} \\ {F_4}^{(e)} \end{Bmatrix} \tag{52}$$

So, $\left[K^{(e)}\right]$ means the element 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

$${u_1}^{(e)} = {U_1}^{(e)} \cos\theta + {U_2}^{(e)} \sin\theta \tag{53}$$

$${u_2}^{(e)} = {U_3}^{(e)} \cos\theta + {U_4}^{(e)} \sin\theta \tag{54}$$

In matrix form as follows

$$\begin{Bmatrix} u_1^{(e)} \\ u_2^{(e)} \end{Bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \end{bmatrix} \begin{Bmatrix} {U_1}^{(e)} \\ {U_2}^{(e)} \\ {U_3}^{(e)} \\ {U_4}^{(e)} \end{Bmatrix} = [R] \begin{Bmatrix} {U_1}^{(e)} \\ {U_2}^{(e)} \\ {U_3}^{(e)} \\ {U_4}^{(e)} \end{Bmatrix} \tag{55}$$

R can be expressed as,

$$[R] = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \end{bmatrix} \tag{56}$$

Now, substitute [R] matrix into equation 21.

$$\begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ 0 & 0 & \cos\theta & \sin\theta \end{bmatrix} \begin{Bmatrix} {U_1}^{(e)} \\ {U_2}^{(e)} \\ {U_3}^{(e)} \\ {U_4}^{(e)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(e)} \\ f_2^{(e)} \end{Bmatrix} \tag{57}$$

In general form, we get:

$$\begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} [R] \begin{Bmatrix} {U_1}^{(e)} \\ {U_2}^{(e)} \\ {U_3}^{(e)} \\ {U_4}^{(e)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(e)} \\ f_2^{(e)} \end{Bmatrix} \tag{58}$$

X and Y-direction equilibrium equations in the global coordinate system are obtained by multiplying the equilibrium equations by cos(θ) and sin(θ) after they have been converted from element displacements to global displacements, which were first represented in the element coordinate system. The process for node 2 is the same. The result would be the same as previously described if we multiplied both sides by $[R]^T$. The transformation matrix is transposed as $[R]^T$.

$$[R]^T \begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} [R] \begin{Bmatrix} {U_1}^{(e)} \\ {U_2}^{(e)} \\ {U_3}^{(e)} \\ {U_4}^{(e)} \end{Bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & \cos\theta \\ 0 & \sin\theta \end{bmatrix} \begin{Bmatrix} f_1^{(e)} \\ f_2^{(e)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(e)} \cos\theta \\ f_1^{(e)} \sin\theta \\ f_2^{(e)} \cos\theta \\ f_2^{(e)} \sin\theta \end{Bmatrix} \tag{59}$$

We know, the right side of the equilibrium is forces.

$$[R]^T \begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} [R] \begin{Bmatrix} {U_1}^{(e)} \\ {U_2}^{(e)} \\ {U_3}^{(e)} \\ {U_4}^{(e)} \end{Bmatrix} = \begin{Bmatrix} {F_1}^{(e)} \\ {F_2}^{(e)} \\ {F_3}^{(e)} \\ {F_4}^{(e)} \end{Bmatrix} \tag{60}$$

Let's put the equations 30 and 22 in a general form.

$$\left[K^{(e)}\right] = [R]^T \begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} [R] \tag{61}$$

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.

$$[K^{(e)}] = \frac{AE}{L}\begin{bmatrix} c^2 & sc & -c^2 & -sc \\ sc & s^2 & -sc & -s^2 \\ -c^2 & -sc & c^2 & sc \\ -sc & -s^2 & sc & s^2 \end{bmatrix} \tag{62}$$

2.5 Spatial Truss Structures

When analyzing spatial (3D) trusses, we can use bar elements if the connections allow only axial load transfer. This is valuable information for determining the structural spatial trusses required in the design. Figure 5[1], a one-dimensional rod element connects nodes i and j in a 3-dimensional spherical reference frame. The unit vector along the axis of the element expressed in the global system is important for analysis.

$$\lambda^e = cos\theta_x I + cos\theta_Y J + cos\theta_Z K \tag{63}$$

Equation 33 shows the unit vector in the global coordinate system.

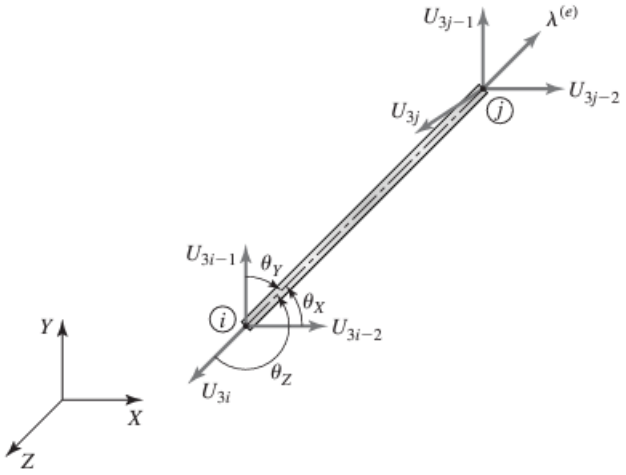


Figure 9. Bar element in x-y-z(3D) frame [3].

Depending on equation 25, we can express the displacements in global system.

$$u_1{}^{(e)} = U_1{}^{(e)}cos\theta_x + U_2{}^{(e)}cos\theta_y + U_3{}^{(e)}cos\theta_z \tag{64}$$

$$u_2{}^{(e)} = U_4{}^{(e)}cos\theta_x + U_5{}^{(e)}cos\theta_y + U_6{}^{(e)}cos\theta_z \tag{65}$$

Using equations 34 and 35 we can create a matrix form.

$$\begin{Bmatrix} {u_1}^{(e)} \\ {u_2}^{(e)} \end{Bmatrix} = \begin{bmatrix} cos\theta_x & cos\theta_y & cos\theta_z & 0 & 0 & 0 \\ 0 & 0 & 0 & cos\theta_x & cos\theta_y & cos\theta_z \end{bmatrix} \begin{Bmatrix} {U_1}^{(e)} \\ {U_2}^{(e)} \\ {U_3}^{(e)} \\ {U_4}^{(e)} \\ {U_5}^{(e)} \\ {U_6}^{(e)} \end{Bmatrix} = [R]\{U^{(e)}\} \qquad (66)$$

Where, [R] is transformation matrix for converting the one-dimensional elements into three dimensional global systems coordinates. Also, the two-dimensional element stiffness matrix needs to be transformed into a three-dimensional global coordinate system.

$$\left[K^{(e)}\right] = [R]^T \begin{bmatrix} k_e & -k_e \\ -k_e & k_e \end{bmatrix} [R] \qquad (67)$$

Now, substitute the [R] matrix in equation to get element stiffness matrix in global coordinate system.

$$\left[K^{(e)}\right] = k_e \begin{bmatrix} {c_x}^2 & c_x c_y & c_x c_z & -{c_x}^2 & -c_x c_y & c_x c_z \\ c_x c_y & {c_y}^2 & c_y c_z & -c_x c_x & -{c_y}^2 & -c_y c_z \\ c_x c_z & c_y c_z & -{c_z}^2 & -c_x c_z & -c_y c_z & -{c_z}^2 \\ -{c_x}^2 & -c_x c_x & -c_x c_z & {c_x}^2 & c_x c_y & c_x c_z \\ -c_x c_y & -{c_y}^2 & -c_y c_z & c_x c_y & {c_y}^2 & c_y c_z \\ -c_x c_z & -c_y c_z & -{c_z}^2 & c_x c_z & c_y c_z & {c_z}^2 \end{bmatrix} \qquad (68)$$

2.6 Planar Frame Element

To study the plane frame element, let us look at the scenario as shown in figure given below.

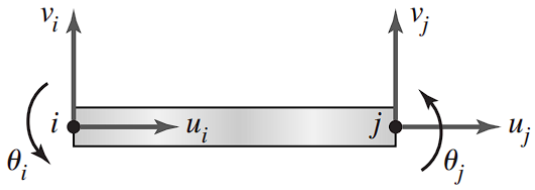


Figure 10. The nodes, nodal displacements under axial loading [3].

A beam element with axial and transverse loading is shown in figure 10. In the beam element we are faced with a situation that we have already analyzed. By including an axial load in the analysis of the

beam element, we will investigate the analysis of the straight frame element. Figure 10 shows how this part adapts to both axial and transverse stress, allowing a wider use. The element can be affected by this axial load in various ways. If there is axial loading on element, there might be buckling [3].

To understand bending under effects of axial loading, consider the case shown in figure given below.

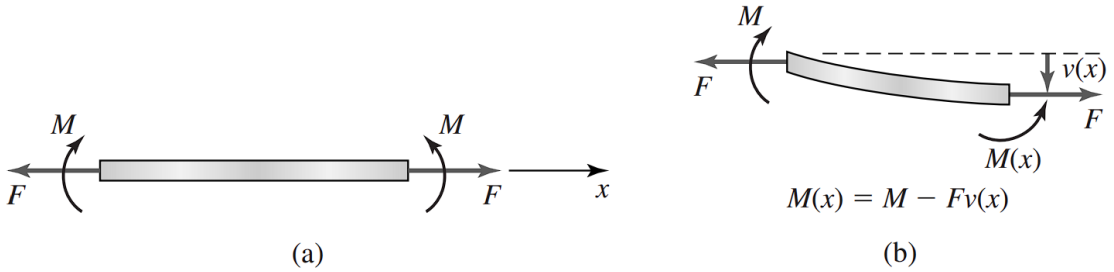


Figure 11. (a) A basic beam under compressive bending. (b) Deflection caused by bending moment and bending function [3].

We already know beam elements stiffness matrix from section 2.3. To obtain the stiffness matrix for the beam shown in figure 10, we will use beam elements stiffness matrix and plane truss elements stiffness matrix.

$$[k_e] = \begin{bmatrix} \frac{AE}{L} & -\frac{AE}{L} & 0 & 0 & 0 & 0 \\ -\frac{AE}{L} & \frac{AE}{L} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{12EI_z}{L^3} & \frac{6EI_z}{L^2} & -\frac{12EI_z}{L^3} & \frac{6EI_z}{L^2} \\ 0 & 0 & \frac{6EI_z}{L^2} & \frac{4EI_z}{L} & -\frac{6EI_z}{L^2} & \frac{2EI_z}{L} \\ 0 & 0 & -\frac{12EI_z}{L^3} & -\frac{6EI_z}{L^2} & \frac{12EI_z}{L^3} & -\frac{6EI_z}{L^2} \\ 0 & 0 & \frac{6EI_z}{L^2} & \frac{2EI_z}{L} & -\frac{6EI_z}{L^2} & \frac{4EI_z}{L} \end{bmatrix} \quad (69)$$

After this step, deflections under axial loading can be expressed as,

$$\{\delta\} = \begin{Bmatrix} u_1 \\ v_1 \\ \theta_1 \\ u_2 \\ v_2 \\ \theta_2 \end{Bmatrix} \quad (70)$$

In a simpler form for element stiffness matrix, consider the figure for comparison local and global

displacements shown below.

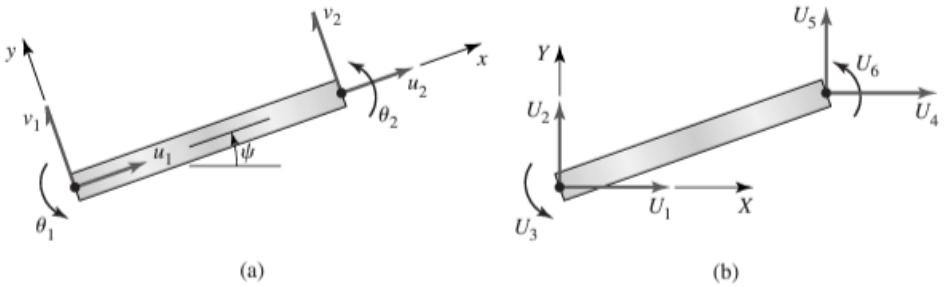


Figure 12. (a) Local displacements. (b) Global displacements [3].

According to the figure 12, we can simply write the stiffness matrix

$$[k_e] = \begin{bmatrix} \frac{AE}{L} & 0 & 0 & -\frac{AE}{L} & 0 & 0 \\ 0 & \frac{12EI_z}{L^3} & \frac{6EI_z}{L^2} & 0 & -\frac{12EI_z}{L^3} & \frac{6EI_z}{L^2} \\ 0 & \frac{6EI_z}{L^2} & \frac{4EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & \frac{2EI_z}{L} \\ -\frac{AE}{L} & 0 & 0 & \frac{AE}{L} & 0 & 0 \\ 0 & -\frac{12EI_z}{L^3} & -\frac{6EI_z}{L^2} & 0 & \frac{12EI_z}{L^3} & -\frac{6EI_z}{L^2} \\ 0 & \frac{6EI_z}{L^2} & \frac{2EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & \frac{4EI_z}{L} \end{bmatrix} \tag{71}$$

Using figure 12, we can convert the local element stiffness displacement to global.

$$u_1 = U_1 \cos\psi + U_2 \sin\psi \tag{72}$$

$$v_1 = -U_1 \sin\psi + U_2 \cos\psi \tag{73}$$

$$\theta_1 = U_3 \tag{74}$$

$$u_2 = U_4 \cos\psi + U_5 \sin\psi \tag{75}$$

$$v_2 = -U_4 \sin\psi + U_5 \cos\psi \tag{76}$$

$$\theta_2 = U_6 \tag{77}$$

Let's write these equations in a matrix.

$$\begin{Bmatrix} u_1 \\ v_1 \\ \theta_1 \\ u_2 \\ v_2 \\ \theta_2 \end{Bmatrix} = \begin{bmatrix} \cos\psi & \sin\psi & 0 & 0 & 0 & 0 \\ -\sin\psi & \cos\psi & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos\psi & \sin\psi & 0 \\ 0 & 0 & 0 & -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \end{Bmatrix} = [R]\{U\} \tag{78}$$

Where, [R] is the transformation matrix.

Lastly, using transformation matrix, the stiffness matrix for planar frame element comes from the equation given below.

$$[K_e] = [R]^T[k_e][R] \tag{79}$$

Where, $[R]^T$ is transpose of the transformation matrix $[R]$.

2.7 Spatial Frame Element

To study the plane frame element, let us look at the scenario as shown in figure given below.

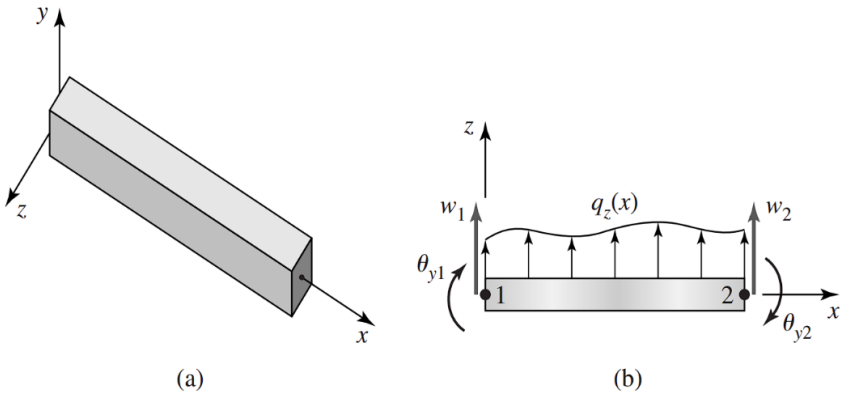


Figure 13. (a) Three-dimensional beam element. (b) Nodal displacements in element x-z plane [3].

In Figure 13b, there is a distributed load $q_z(x)$ positive along the z direction, there are displacements w_1 and w_2 at nodes 1 and 2. The notations θ_{y1} and θ_{y2} are given. Depending on the figure, The element stiffness matrix for the xz plane can be expressed as:

$$[k_e]_{xz} = \frac{EI_y}{L^3}\begin{bmatrix} 12 & -6L & -12 & -6L \\ -6L & 4L^2 & 6L & 2L^2 \\ -12 & 6L & 12 & 6L \\ -6L & 2L^2 & 6L & 4L^2 \end{bmatrix} \tag{80}$$

The element equilibrium equations for a two-plane bending element with axial stiffness are expressed in matrix form as follows: combining the spar element stiffness matrix, the x-y plane flexure stiffness matrix, and the x-z plane stiffness matrix form:

$$\begin{bmatrix} [k_{axial}] & [0] & [0] \\ [0] & [k_{bending}]_{xy} & [0] \\ [0] & [0] & [k_{bending}]_{xz} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ v_1 \\ \theta_{z1} \\ v_2 \\ \theta_{z2} \\ w_1 \\ \theta_{y1} \\ w_2 \\ \theta_{y2} \end{Bmatrix} = \begin{Bmatrix} f_{x1} \\ f_{x2} \\ f_{y1} \\ M_{z1} \\ f_{y2} \\ M_{z2} \\ f_{z1} \\ M_{y1} \\ f_{z2} \\ M_{y2} \end{Bmatrix} \tag{81}$$

Let's pull out the stiffness matrix from equation 81.

$$[k_e] = \begin{bmatrix} [k_{axial}] & [0] & [0] \\ [0] & [k_{bending}]_{xy} & [0] \\ [0] & [0] & [k_{bending}]_{xz} \end{bmatrix} \tag{82}$$

We know, the stiffness matrix is a 10x10 matrix created by the $[k_{axial}]$, $[k_{bending}]_{xy}$, $[k_{bending}]_{xz}$ and the zero matrixes.

To provide torsion to the beam element, we need a circular cylinder subjected to torsion due to the torsional moments applied to its ends, as shown in figure 13a. Torsion finite element at nodes 1 and 2, with the axis of the cylinder aligned with the x-axis and matching the positive torsional moments shown according to the right rule figure 13b. The torsion angle per unit length of a circular cylinder with uniform elastic properties applied to a torque is given by T.

$$\phi = \frac{T}{JG} \tag{83}$$

Where J is the polar moment of inertia, G is shear modulus.

The overall angle of twist of the element can be represented in terms of the nodal rotations and twisting moments as the angle of twist per unit length is constant.

$$\theta_{x2} - \theta_{x1} = \frac{TL}{JG} \tag{84}$$

To take it further, notice that there is a moment equilibrium as,

$$M_{x1} + M_{x2} = 0 \tag{85}$$

According to the equation 85, the element matrix equation is given as follows,

$$\frac{JG}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\begin{Bmatrix} \theta_{x1} \\ \theta_{x2} \end{Bmatrix} = \begin{Bmatrix} M_{x1} \\ M_{x2} \end{Bmatrix} \tag{86}$$

So, the torsional stiffness matrix is,

$$[k_{torsion}] = \frac{JG}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{87}$$

Adding the torsional characteristics to the general beam element, the element equations become,

$$\begin{bmatrix} [k_{axial}] & [0] & [0] & [0] \\ [0] & [k_{bending}]_{xy} & [0] & [0] \\ [0] & [0] & [k_{bending}]_{xz} & [0] \\ [0] & [0] & [0] & [k_{torsion}] \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ v_1 \\ \theta_{z1} \\ v_2 \\ \theta_{z2} \\ w_1 \\ \theta_{y1} \\ w_2 \\ \theta_{y2} \\ \theta_{x1} \\ \theta_{x2} \end{Bmatrix} = \begin{Bmatrix} f_{x1} \\ f_{x2} \\ f_{y1} \\ M_{z1} \\ f_{y2} \\ M_{z2} \\ f_{z1} \\ M_{y1} \\ f_{z2} \\ M_{y2} \\ M_{x1} \\ M_{x2} \end{Bmatrix} \tag{88}$$

Let's pull out the stiffness matrix from equation 88.

$$\begin{bmatrix} [k_{axial}] & [0] & [0] & [0] \\ [0] & [k_{bending}]_{xy} & [0] & [0] \\ [0] & [0] & [k_{bending}]_{xz} & [0] \\ [0] & [0] & [0] & [k_{torsion}] \end{bmatrix} \tag{89}$$

We know, the stiffness matrix is a 10x10 matrix created by the $[k_{axial}]$, $[k_{bending}]_{xy}$, $[k_{bending}]_{xz}$, $[k_{torsion}]$ and the zero matrixes.

3. DESIGN PROCESS OF COMPUTER PROGRAM

For this program, I will use MATLAB to create an optimal design. MATLAB is a computational matrix-based computer program used by most engineering professionals. I decided to use MATLAB for this unique project.

3.1 Initialization

```
clc;
clear all;
```

These commands are typical MATLAB launch procedures. To create a tidy interface, the command window is cleared by the CLC. MATLAB eliminates all variables from the workspace, guaranteeing that the data no longer obstructs accurate computations.

3.2 User Input: Number of Elements and Nodes

```
% Prompt the user for the number of elements and nodes
valid_input = false;
while ~valid_input
    prompt = {'Enter the number of elements:', 'Enter the number of nodes:'};
    dlg_title = 'Input';
    num_lines = 1;
    default_answer = {'3', '4'};
    answer = inputdlg(prompt, dlg_title, num_lines, default_answer);

    % Check if the dialog was canceled
    if isempty(answer)
        errordlg('Input canceled. Exiting the program.', 'Input Error');
        return; % Exit the program
    end

    % Convert inputs to numeric values
    num_elements = str2double(answer{1});
    num_nodes = str2double(answer{2});

    % Check if inputs are valid numbers and not empty
```

```
    if any(cellfun(@isempty, answer)) || isnan(num_elements) || isnan(num_nodes)
        % Show error message if inputs are not valid
        h = errordlg('Invalid input. Please enter numeric values for the number of elements and nodes.', 'Input Error');
        uiwait(h); % Wait for user to close the error dialog
    else
        valid_input = true;
    end
end
```

In the section responsible for obtaining the number of elements and nodes, the code initiates by prompting the user to input these parameters for the spatial truss structure. Using the **inputdlg** function, a dialog box appears, presenting fields for the user to specify the number of elements and nodes required for the analysis.

Following the user's input, the code validates the provided values. If the user cancels the input dialog, an error message alerts the user of the cancellation, and the program terminates. Subsequently, the entered values undergo verification to ensure they are numeric and not empty. This verification process ensures the integrity of the input data.

Once the validation process is complete, the code converts the string inputs representing the number of elements and nodes into numeric values using the **str2double** function. This conversion facilitates subsequent mathematical operations and indexing.

In the event of invalid input, such as non-numeric characters or empty fields, the program displays an error message prompting the user to input numeric values for both the number of elements and nodes. This iterative process continues until valid numeric inputs are provided for both parameters.

Upon successful validation, the obtained numeric values representing the number of elements and nodes are stored in the variables **num_elements** and **num_nodes**, respectively. These variables serve as crucial inputs for further computations and analyses within the code.

3.3 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
    valid_input = false;
    while ~valid_input
        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)};
        dlg_title = sprintf('Node %d Coordinates', i);
        num_lines = 1;
        default_answer = {'0', '0', '0'};
        answer = inputdlg(prompt, dlg_title, num_lines, default_answer);

        % Check if the dialog was canceled
        if isempty(answer)
            errordlg('Input canceled. Exiting the program.', 'Input Error');
            return; % Exit the program
        end
        % Convert inputs to numeric values
        x = str2double(answer{1});
        y = str2double(answer{2});
        z = str2double(answer{3});
        % Check if inputs are valid numbers and not empty
        if any(cellfun(@isempty, answer)) || isnan(x) || isnan(y) || isnan(z)
            % Show error message if inputs are not valid
            h = errordlg('Invalid input. Please enter numeric values for coordinates.', 'Input Error');
            uiwait(h); % Wait for user to close the error dialog
```

```
        else
            % Store coordinates in the matrix if inputs are valid
            node_coordinates(i, :) = [x, y, z];
            valid_input = true;
        end
    end
end
```

The code initializes by creating an empty matrix, **node_coordinates**, to store the coordinates of each node. It allocates memory for the matrix based on the number of nodes previously obtained from the user input.

Subsequently, a loop iterates over each node to collect its coordinates. Within this loop, the code prompts the user to input the x, y, and z coordinates for the current node using the **inputdlg** function. The dialog box specifies fields for entering the coordinates, with prompts tailored to each dimension.

Once the user submits the coordinates, the code validates the input to ensure it is numeric and not empty. This validation process guarantees the integrity of the provided coordinates.

Upon successful validation, the code converts the string inputs representing the coordinates into numeric values using the **str2double** function. This conversion enables subsequent mathematical operations involving the coordinates.

In cases of invalid input, such as non-numeric characters or empty fields, the program displays an error message, prompting the user to input valid numeric values for the coordinates. This iterative process continues until valid coordinates are obtained for each node.

Upon obtaining valid coordinates for a node, the code stores them in the **node_coordinates** matrix at the corresponding row, effectively recording the spatial location of each node within the truss structure.

3.4 Connectivity of Elements

```
% Initialize matrices for element connections and properties
connections = zeros(num_elements, 2);
```

```
element_stiffness_matrices = cell(num_elements, 1);

% Loop through each element to compute stiffness matrices
for i = 1:num_elements
    prompt = {sprintf('Enter first node for Element %d:', i), ...
            sprintf('Enter second node for Element %d:', i)};
    dlg_title = sprintf('Element %d Nodes', i);
    num_lines = 1;
    default_answer = {'', ''};
    answer = inputdlg(prompt, dlg_title, num_lines, default_answer);
    node1 = str2double(answer{1});
    node2 = str2double(answer{2});

    connections(i, :) = [node1, node2];
```

The code initializes by creating an empty matrix, **connections**, to store the node connections for each element. This matrix is sized based on the number of elements previously specified by the user.

Subsequently, a loop iterates over each element to collect its node connections. Within this loop, the code prompts the user to input the node numbers representing the start and end nodes of the current element using the **inputdlg** function. The dialog box prompts the user to enter the node numbers corresponding to the start and end nodes of the current element.

Once the user provides the node numbers, the code converts the string inputs representing the node numbers into numeric values using the **str2double** function. This conversion facilitates subsequent operations involving the node numbers.

The code then stores the obtained node connections in the **connections** matrix, with each row representing an element and the two columns containing the node numbers representing the start and end nodes of the element.

This process repeats for each element in the spatial truss structure until the connections for all elements are obtained.

3.5 Obtaining Material Properties and Cross-Sectional Areas

```
% Initialize matrices for material properties and cross-sectional areas
modulus_of_elasticity = zeros(num_elements, 1);
```

```
cross_sectional_areas = zeros(num_elements, 1);

% Loop through each element to prompt user for material properties and cross-sectional areas
for i = 1:num_elements
    % Prompt user for material properties
    prompt = {sprintf('Enter modulus of elasticity (Kpa for Element %d:', i), ...
              sprintf('Enter cross-sectional area (m^2) for Element %d:', i)};
    dlg_title = sprintf('Element %d Properties', i);
    num_lines = 1;
    default_answer = {'200e6', '0.001'};
    answer = inputdlg(prompt, dlg_title, num_lines, default_answer);

    % Check if the dialog was canceled
    if isempty(answer)
        errordlg('Input canceled. Exiting the program.', 'Input Error');
        return; % Exit the program
    end

    % Convert inputs to numeric values
    E = str2double(answer{1});
    A = str2double(answer{2});

    % Validate input
    if any(cellfun(@isempty, answer)) || isnan(E) || isnan(A)
        % Show error message if inputs are not valid
        h = errordlg('Invalid input. Please enter numeric values for material properties.', 'Input Error');
        uiwait(h); % Wait for user to close the error dialog
        return; % Exit the program
    end

    % Store material properties and cross-sectional areas
    modulus_of_elasticity(i) = E;
```

```
    cross_sectional_areas(i) = A;
end
```

The code initializes by creating empty matrices to store material properties and cross-sectional areas for each element. These matrices are sized based on the number of elements in the truss structure.

Next, a loop iterates over each element to prompt the user for material properties and cross-sectional areas. Within this loop:

- The code displays a dialog box using the **inputdlg** function, requesting the modulus of elasticity (in kPa) and the cross-sectional area (in square meters) for the current element.
- The dialog box provides fields for the user to input these values, with prompts indicating the type of information required for each field.

Once the user submits the material properties and cross-sectional area values, the code validates the input to ensure that the provided values are numeric and not empty. This validation step ensures the integrity of the input data.

Upon successful validation, the code converts the string inputs representing the material properties and cross-sectional areas into numeric values using the **str2double** function. This conversion facilitates subsequent calculations involving these parameters.

The obtained material properties and cross-sectional areas are then stored in the respective matrices allocated at the beginning of the section. Each row of these matrices corresponds to an element, with the modulus of elasticity and cross-sectional area values assigned accordingly.

This process repeats for each element in the spatial truss structure until material properties and cross-sectional areas are obtained for all elements.

3.6 Calculation of Element Lengths

```
% Loop through each element to compute element lengths and stiffness matrices

for i = 1:num_elements

    % Extract coordinates of the nodes for the current element

    x1 = node_coordinates(connections(i, 1), 1);

    y1 = node_coordinates(connections(i, 1), 2);

    z1 = node_coordinates(connections(i, 1), 3);

    x2 = node_coordinates(connections(i, 2), 1);
```

```
    y2 = node_coordinates(connections(i, 2), 2);
    z2 = node_coordinates(connections(i, 2), 3);

    % Compute element length
    L = SpaceTrussElementLength(x1, y1, z1, x2, y2, z2);

    % Compute direction cosines
    if L ~= 0
        Cx = (x2 - x1) / L;
        Cy = (y2 - y1) / L;
        Cz = (z2 - z1) / L;
    else
        Cx = 0;
        Cy = 0;
        Cz = 0;
    end

    % Compute element stiffness matrix
    element_stiffness_matrices{i} = SpaceTrussElementStiffness(E, A, L, Cx, Cy, Cz);

    % Display element stiffness matrix
    fprintf('Element Stiffness Matrix for Element %d:\n', i);
    disp(element_stiffness_matrices{i});
end
```

In this segment, the code iterates over each element of the spatial truss structure. For each element, it performs several calculations essential for structural analysis.

The first step involves extracting the coordinates of the start and end nodes of the current element from the **node_coordinates** matrix. These coordinates serve as the basis for determining the length of the element.

With the node coordinates in hand, the code proceeds to compute the length (L) of the current element using geometric principles. This length is a fundamental parameter for subsequent calculations related to the element's stiffness and behavior under load.

Following the calculation of element length, the code determines the direction cosines (Cx, Cy, Cz) to establish the orientation of the element in three-dimensional space. These direction cosines are computed based on the vector between the start and end nodes of the element.

Using the material properties (modulus of elasticity, cross-sectional area), element length, and direction cosines, the code computes the stiffness matrix for the current element. This stiffness matrix represents the element's resistance to deformation under applied loads and is crucial for analyzing the overall structural behavior.

Finally, the stiffness matrix for the current element is displayed to provide insights into its structural properties. This information aids in understanding how the element responds to different loading conditions and contributes to the overall stability and performance of the truss structure.

3.7 Assembling the Global Stiffness Matrix

```
% Assemble the global stiffness matrix
K_global = zeros(3*num_nodes, 3*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 = SpaceTrussAssemble(K_global, K_element, node_i, node_j);
end
```

```
% Store the original global stiffness matrix for reaction calculations
K_global_original = K_global;


% Display the global stiffness matrix
disp('Global Stiffness Matrix (K_global):');
disp(K_global);
```

The code initializes an empty matrix named **K_global** to store the global stiffness matrix. This matrix is sized based on the total degrees of freedom in the truss structure, which is determined by multiplying the number of nodes by 3 (since each node has three degrees of freedom).

Following initialization, a loop iterates over each element of the truss structure. For each element, the code retrieves the node numbers representing the start and end nodes from the **connections** matrix.

Next, the stiffness matrix (**K_element**) corresponding to the current element is obtained from the precomputed array **element_stiffness_matrices**.

The **SpaceTrussAssemble** function is then called to assemble the stiffness contribution of the current element into the global stiffness matrix **K_global**.

Once the loop completes, the assembled global stiffness matrix is displayed to provide insight into the structural properties of the truss system.

Additionally, the original global stiffness matrix (**K_global_original**) is stored for potential use in calculating reactions at supports or other analyses.

This section of the code is crucial as it determines the stiffness characteristics of the entire spatial truss structure, laying the foundation for subsequent analyses and simulations.

3.8 Applying Boundary Conditions and Solving

```
% Initialize force and displacement vectors
F = zeros(3*num_nodes, 1); % External forces vector
U = zeros(3*num_nodes, 1); % Displacements vector
```

```
known_U = false(3*num_nodes, 1); % Logical array to track known displacements


% Ask the user for boundary condition information for each node
for i = 1:num_nodes
    bc_type = questdlg(sprintf('Select the type of boundary condition at Node %d:', i), 'Boundary Condition Type', 'Force (F)', 'Displacement (U)', 'Displacement (U)');
    if strcmpi(bc_type, 'Force (F)')
        prompt = {'Enter the force applied to Node in kN (Fx):', 'Enter the force applied to Node in kN (Fy):', 'Enter the force applied to Node in kN (Fz):'};
        dlg_title = sprintf('Force at Node %d', i);
        num_lines = 1;
        default_answer = {'0', '0', '0'};
        answer = inputdlg(prompt, dlg_title, num_lines, default_answer);
        F(3*i-2:3*i) = str2double(answer);
    elseif strcmpi(bc_type, 'Displacement (U)')
        prompt = {'Enter the displacement of Node in meters (Ux):', 'Enter the displacement of Node in meters (Uy):', 'Enter the displacement of Node in meters (Uz):'};
        dlg_title = sprintf('Displacement at Node %d', i);
        num_lines = 1;
        default_answer = {'0', '0', '0'};
        answer = inputdlg(prompt, dlg_title, num_lines, default_answer);
        U(3*i-2:3*i) = str2double(answer);
        known_U(3*i-2:3*i) = true; % Mark displacements as known
    end
end
```

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

% Solve for unknown displacements
U_unknown = K_global \ F;

% Update displacements 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 in a dialog box
result_message = {...
    'Nodal Displacements (U in m):', ...
    num2str(U), ...
    '', ...
    'Nodal Forces (F in kN) :', ...
```

```
   num2str(R) };
dlgtitle = 'Analysis Results';
msgbox(result_message, dlgtitle);
disp('Nodal Displacements in meters')
disp(U)
disp('Nodal Forces in kN')
disp(R)
```

In this section of the code, boundary conditions are applied to the spatial truss structure, and the system of equations representing the structure is solved to determine displacements and reactions.

Initially, three vectors are initialized: **F**, representing the external forces acting on the nodes; **U**, representing the displacements vector storing the unknown displacements at each node; and **known_U**, a logical array indicating known or unknown displacements at each degree of freedom.

The code then prompts the user for boundary condition information for each node of the truss structure. For each node, a dialog prompts the user to select the type of boundary condition: either force (F) or displacement (U). Depending on the selected type:

- If the boundary condition is force (F), the user is prompted to input the forces in the x, y, and z directions for the node, which are then stored in the **F** vector.
- If the boundary condition is displacement (U), the user is prompted to input the displacements in the x, y, and z directions for the node, which are stored in the **U** vector. Additionally, the corresponding entries in the **known_U** array are marked as true to indicate known displacements.

Following the acquisition of boundary condition information, the code modifies the global stiffness matrix (**K_global**) and force vector (**F**) based on the known displacements. This involves zeroing out rows corresponding to known displacements in the stiffness matrix, setting diagonal entries corresponding to known displacements to one, and setting forces equal to known displacements in the force vector.

Subsequently, the system of equations is solved for unknown displacements (**U_unknown**) using the modified global stiffness matrix and force vector.

The displacements vector (**U**) is then updated with the solved values.

Reactions at supports are calculated using the original global stiffness matrix (**K_global_original**) and the solved displacements.

Finally, the results, including nodal displacements (**U**) and nodal forces (**R**), are displayed in a dialog box and printed to the console for analysis.

This section of the code plays a critical role in analyzing the structural behavior of the truss system under applied loads and constraints.

Let me make it clear with a simplified example. Consider a linear equation system with 2 unknowns and 2 equations. Let the matrix form for this system be:

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 4 \\ U_2 \end{bmatrix} = \begin{bmatrix} R_1 \\ 3 \end{bmatrix} \tag{90}$$

Here, since U_1 is known we will set the first row of matrix to zero then our matrix becomes:

$$\begin{bmatrix} 0 & 0 \\ 1 & -1 \end{bmatrix} \tag{91}$$

Then, again since U_1 is known we will set diagonal of first row to 1.

$$\begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix} \tag{92}$$

After the step F(i) = U(i) our system becomes:

$$\begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 4 \\ U_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \end{bmatrix} \tag{93}$$

Solve for U_2, result is 1.

I made these arrangements only to find unknown displacements not reaction forces. The reaction forces will be found later with the stored global stiffness matrix.

3.9 Solving for Reactions and Displaying Results

```
% Pulling elemental displacements
for i = 1:num_elements
   node1 = connections(i, 1disp(U);
   node2 = connections(i, 2);
   u_element = [U(3*node1-2); U(3*node1-1); U(3*node1); U(3*node2-2); U(3*node2-1); U(3*node2)];
```

```
    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 length
    L = SpaceTrussElementLength(x1, y1, z1, x2, y2, z2);

    % Compute direction cosines using given formulas
    thetax = acos((x2 - x1) / L) * 180 / pi;
    thetay = acos((y2 - y1) / L) * 180 / pi;
    thetaz = acos((z2 - z1) / L) * 180 / pi;

    % Compute the stress in the element
    sigma = SpaceTrussElementStress(E, L, thetax, thetay, thetaz, u_element);
    stress_values(i) = abs(sigma);

    % Display the stress result
    fprintf('Stress in Element %d: %.4e kPa\n', i, sigma);
end
```

In this part of the code, I iterate through each element of the spatial truss structure. For every element: I retrieve the node numbers representing the start and end nodes from the **connections** matrix. Using these node numbers, I extract the nodal displacements from the **U** vector to create the **u_element** vector, containing the displacements for both nodes of the element. Then, I extract the coordinates of the start and end nodes from the **node_coordinates** matrix. With these coordinates, I compute the element length **L** using the **SpaceTrussElementLength** function. Next, I calculate the direction cosines (**thetax**, **thetay**, **thetaz**) by taking the arccosine of the ratio of coordinate differences to the element length. Using the displacements and direction cosines obtained, I compute the stress in the

element with the **SpaceTrussElementStress** function. This function incorporates material properties, element length, displacements, and direction cosines to determine stress. The computed stress is stored in the **sigma** variable, and its absolute value is assigned to the corresponding index in the **stress_values** array. Finally, I display the calculated stress for the current element using a formatted output statement.

This code section is essential for assessing the stress distribution within each element of the truss structure, providing crucial insights into its structural integrity and performance under applied loads.

3.10 Visualization Results

```
% Visulazation Results


max_stress = max(stress_values);
min_stress = min(stress_values);


% Plot spatial truss structure
figure;
scatter3(node_coordinates(:, 1), node_coordinates(:, 2), node_coordinates(:, 3), 100, 'r', 'filled'); hold on;
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, 'b-', 'LineWidth', 2);
end
```

```
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Spatial Truss Structure');
grid on
offset = 0.1; % 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', 'green', 'FontSize', 12);
end
axis equal;
view(3);
```

In this section of the code, the visualization of the spatial truss structure, along with stress visualization, is performed.

First, the maximum and minimum stress values are computed from the **stress_values** array.

Then, the spatial truss structure is plotted. A new figure is created, and the node coordinates are scatter-plotted in red, representing each node as a filled circle. Additionally, a loop iterates through each element to plot its connections using **plot3**, drawing lines between the corresponding nodes. These lines are displayed in blue with a line width of 2.

The axes are labeled as X, Y, and Z, and the title of the plot is set to "Spatial Truss Structure". Grid lines are added to the plot using the **grid on** command.

For better visualization, the text position is adjusted using an offset, and each node is labeled with its corresponding index using the **text** function. Node indices are displayed in green with a font size of 12.

Lastly, the aspect ratio of the plot is set to be equal along all axes using **axis equal**, and the viewing angle is adjusted to a 3D perspective with **view(3)**.

This visualization provides a clear representation of the spatial truss structure, allowing for visual

inspection of its geometry and connectivity, aiding in understanding its structural behavior under applied loads.

3.11 Plotting Spatial Truss Structure with Stress Visualization

```
% Plot spatial truss structure with stress visualization
figure;
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)];

    % Define color based on stress level
    stress_index = (stress_values(i) - min_stress) / (max_stress - min_stress);
    color_map = jet;
    color_index = round(stress_index * (size(color_map, 1) - 1)) + 1;
    color = color_map(color_index, :);

    % Plot connection with stress-based coloring
    plot3(x, y, z, 'Color', color, 'LineWidth', 2); hold on;
end
scatter3(node_coordinates(:,1), node_coordinates(:,2), node_coordinates(:,3), 100, 'k', 'filled');
xlabel('X');
ylabel('Y');
zlabel('Z');
```

```
title('Spatial Truss Structure with Stress Visualization');
grid on
axis equal;
view(3);
```

A new figure is created, and for each element, the connections between nodes are plotted using **plot3**. The coordinates of the nodes are retrieved from the **node_coordinates** matrix, and lines connecting the nodes are drawn in colors corresponding to stress levels. The stress level is determined based on the stress values calculated earlier and normalized to the range between the minimum and maximum stress values. The **jet** colormap is used to map stress levels to colors, and the **plot3** function is employed to visualize the connections with stress-based coloring.

Additionally, scatter points are plotted for each node to mark their positions. These points are displayed in black and filled.

Axes labels are set for X, Y, and Z axes, and the title of the plot is specified as "Spatial Truss Structure with Stress Visualization". Grid lines are added to the plot using **grid on**.

To ensure a consistent aspect ratio, the **axis equal** command is used, and the viewing angle is adjusted to a 3D perspective with **view(3)**.

This visualization provides insight into stress distribution within the spatial truss structure, highlighting regions of higher and lower stress concentrations.

3.12 Color Scaling

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

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

A color scale is generated using the **colorbar** function, which creates a color bar legend to represent the stress levels. The colormap is set to 'jet' using the **colormap** function, which maps stress values to colors ranging from blue to red, indicating low to high stress levels, respectively.

The color scale range is set to span from the minimum stress value to the maximum stress value using the **caxis** function.

A label is added to the color scale axis, specifying the units of stress as 'Stress Levels (kPa)', using the **ylabel** function.

To improve readability, tick marks and labels are set on the color scale axis. The **linspace** function is used to generate evenly spaced tick positions between the minimum and maximum stress values. The **arrayfun** function is then utilized to format the tick labels in scientific notation with two decimal places. Finally, the tick positions and labels are set using the **set** function.

4. VALIDATION EXAMPLE

In this validation phase, I will utilize a verified example from my reference book, "MATLAB Guide to Finite Elements: An Interactive Approach" by P. I. Kattan (2003). Specifically, I will refer to Example 6.1 from the section on spatial truss analysis to corroborate the results obtained from my computer program.

Example 6.1:

Consider the space truss shown in Figure 6.2. The supports at nodes 1, 2, and 3 are ball-and-socket joints allowing rotation but no translation. Given $E = 200$ GPa, $A_{14} = 0.001$ m^2, $A_{24} = 0.002$ m^2, $A_{34} = 0.001$ m^2, and $P = 12$ kN, determine:

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

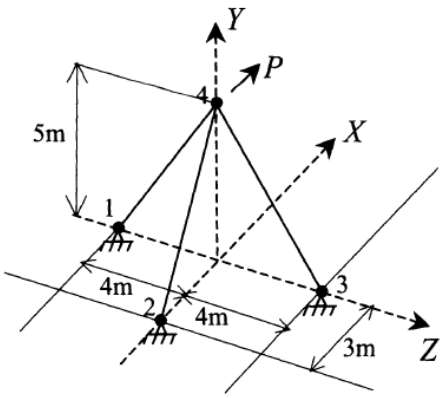


Figure 14. Validation Problem [4].

To initiate the validation process, I commenced by inputting essential parameters for the spatial truss structure, depicted in the accompanying figure. A dialog box prompted me to specify the number of nodes and elements within the structure. Aligning with the specifications outlined in the book, I opted for 4 nodes and 3 elements for this example. This initial input holds significant importance as it establishes the groundwork for subsequent finite element analysis procedures.

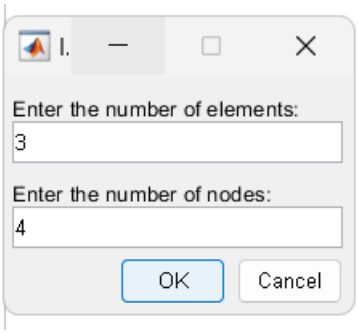


Figure 15. Input screen for number of elements and nodes

If any invalid or non-numeric value inputted at this stage, the program gives error to user as shown in figure 16. After the user clicks 'OK' button, the input screen in figure 15 pops up again for user to enter valid input. This error screen pops up every time user inputted invalid or non-numeric value at every stage of the program.

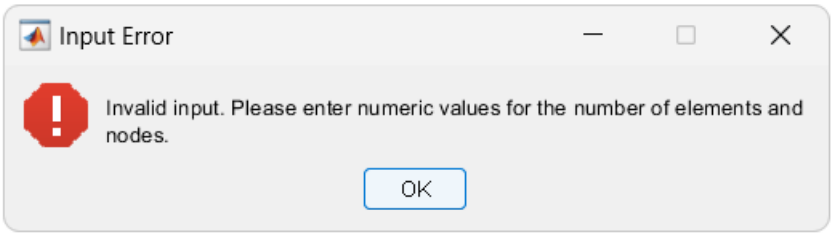


Figure 16. Error screen for non-numeric input.

If the user clicks the close button or cancel button of any input screen the program will show the below message box and terminates the program.

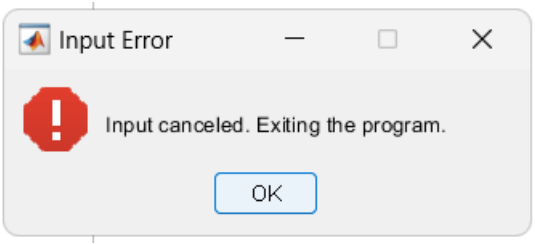


Figure 17.Message box

According to validation problem as shown in figure 14, the number of elements should be 3, the number of nodes should be 4. After these numbers inputted correctly, the program asks for coordinates of node 1. For ease of use, I preferred to separate the inputs as x-y-z coordinates as shown in figure 18.

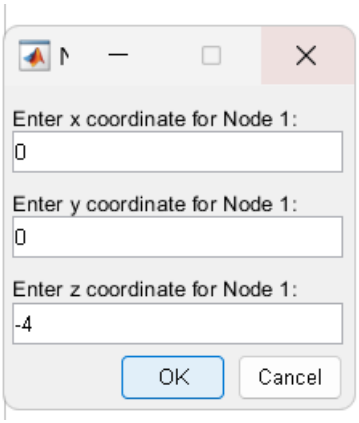


Figure 18. Input screen for node 1 coordinates.

As you can see from figure 14, node 1 coordinates are (0,0, -4).

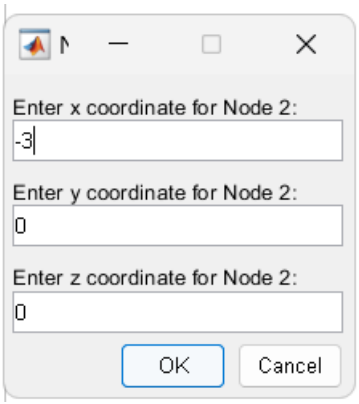


Figure 19. Input screen for node 2 coordinates.

As you can see from figure 14, node 2 coordinates are (-3,0,0).

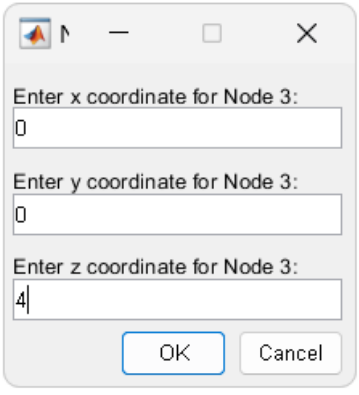


Figure 20. Input screen for node 3 coordinates.

As you can see from figure 14, node 3 coordinates are (0,0,4).

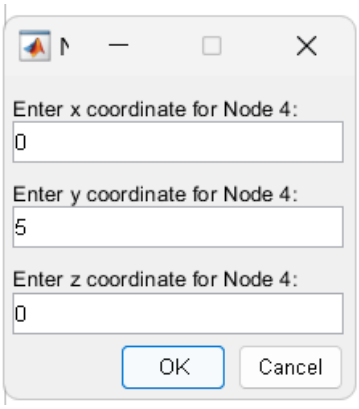


Figure 21. Input screen for node 4 coordinates.

As you can see from figure 14, node 4 coordinates are (0,5,0).

After input of node coordinates stage, program starts asking to user for element connectivity of the spatial truss structure.

Element Number	Node i	Node j
1	1	4
2	2	4
3	3	4

Figure 22. Element connectivity information for validation problem [4].

As you can see in figure 14 or in figure 22, element 1 is connected to node 1 and node 4. So, user inputs the element 1's connectivity as shown below figure.

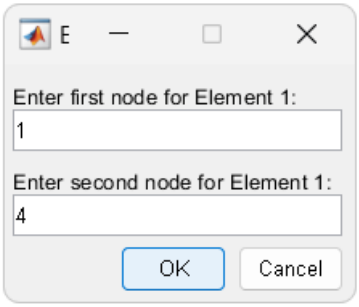


Figure 23. Input screen for element 1 connectivity.

After user inputted the element 1's connectivity, the program will ask element 1's material properties. As shown in figure 14, every element for this example has the same elastic modulus which is 200 GPa. Elements 1 and 3 have the same cross-sectional area which is 0.001 meters. Element 2 has a cross sectional area of 0.002 meters. So, the user should enter the elastic modulus and cross-sectional area for element 1 as shown in figure 24.

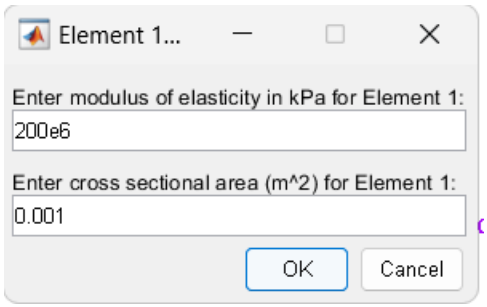


Figure 24. Input screen for material properties of element 1.

As you can see in figure 14 or in figure 22, element 2 is connected to node 2 and node 4. So, user inputs the element 2's connectivity as shown in figure 25.

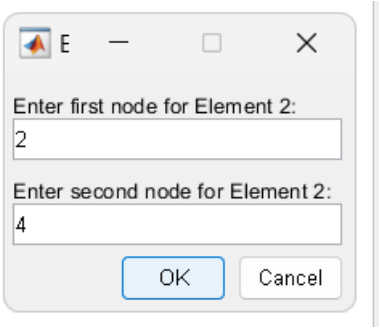


Figure 25.Input screen for element 2 connectivity.

Now, the program will ask the user to element properties for element 2.

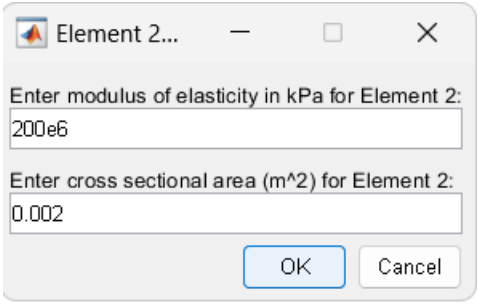


Figure 26. Input screen for material properties of element 2

As you can see in figure 14 or in figure 22, element 3 is connected to node 3 and node 4. So, user

inputs the element 3's connectivity should be as shown in figure below.

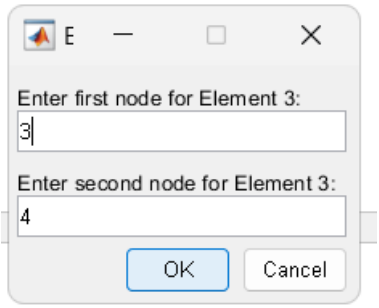


Figure 27. Input screen for element 3 connectivity.

Now, the program will ask the user to element properties for element 3.

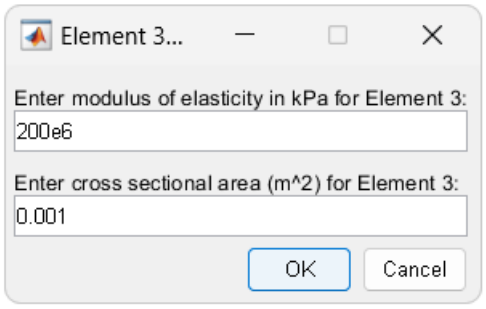


Figure 28. Input screen for material properties of element 3.

After these steps are completed, the program asks for boundary conditions. At every node there must be 1 boundary condition. First, the program asks for boundary condition type. The user should select between displacement or force.

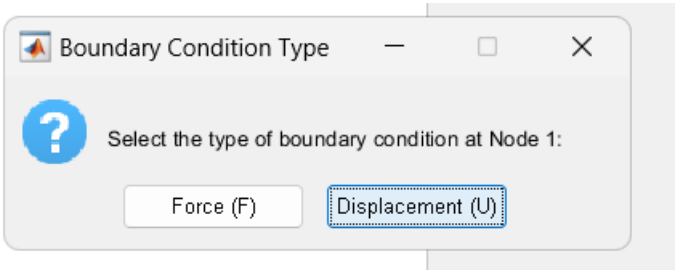


Figure 29. Boundary condition type selection for node 1.

As you can see from figure 14, node 1 is a fixed node. So, the user should select displacement in screen shown in figure 29.

After the user selected displacement in figure 29, the program will ask for the value of the

displacement. For our validation example, since node 1 is a fixed node displacement values at the directions x-y-z are 0.

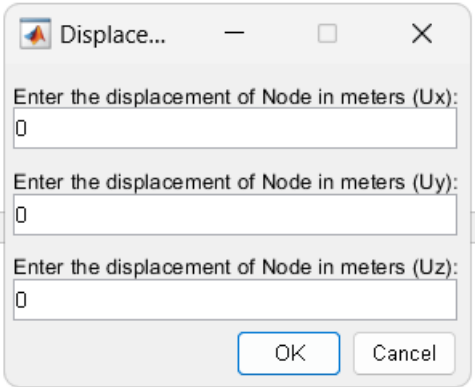


Figure 30. Input screen for displacement values at node 1.

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

Now, the program will ask the user for boundary condition type at node 2. As you can see from figure 14, node 2 is a fixed node. So, the user should select the boundary condition type as displacement.

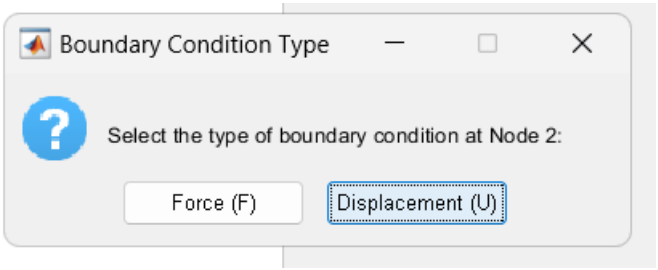


Figure 31.Boundary condition selection screen for node 2.

After the user selected displacement in figure 31, the program will ask for the value of the displacement. For our validation example, since node 2 is a fixed node displacement values at the directions x-y-z are 0.

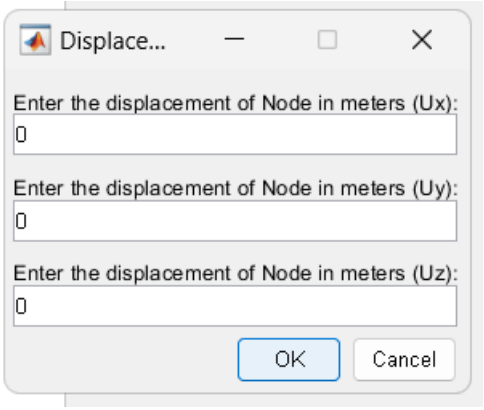


Figure 32. Input screen for displacement values at node 2.

After this step, the user should select the boundary condition type for node 3. As figure 14 shows, node 3 is a fixed node. So, the user should select the displacement boundary condition in figure below.

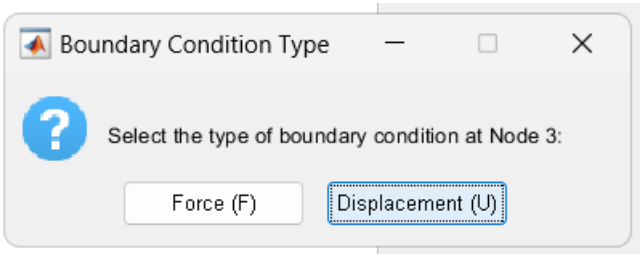


Figure 33. Boundary condition selection screen for node 3.

After the user selected displacement in figure above, the program will ask for the value of the displacement. According to figure 14, since node 3 is a fixed node displacement values at the directions x-y-z are 0.

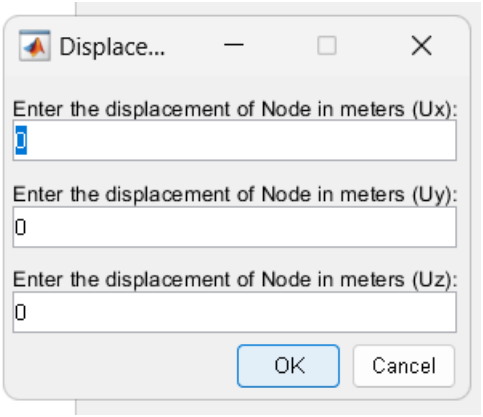


Figure 34. Input screen for displacement values at node 3.

Now, again the program will ask for boundary condition type. For node 4, according to figure 14, boundary condition is given in the question as a force value at x direction which has a magnitude of 12 kN. So, the user should select F as boundary condition type in figure 35.

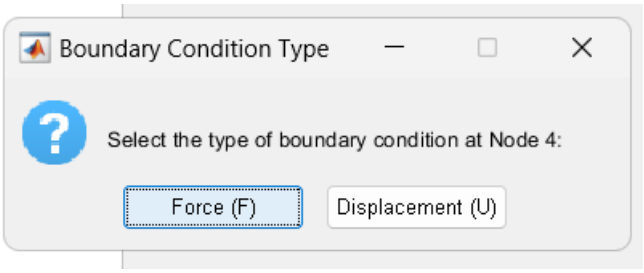


Figure 35. Boundary condition selection screen for node 4.

After the user selected F in the figure above, the program will ask for the force values to the user. Since value is given in validation problem as 12 kN in x direction. There is no force in the y and z directions. So, the user should input as shown in figure 36.

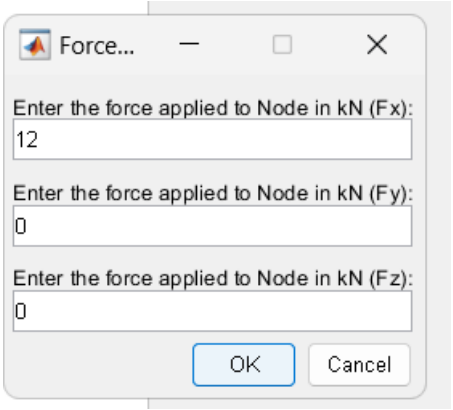


Figure 36. Input screen for force values at node 4.

Finally, the input section is over. Now the program will show the results, the spatial truss structure geometry and colorized stress levels according to the magnitude of element stress values. For our validation problem the results are shown in figure 37.

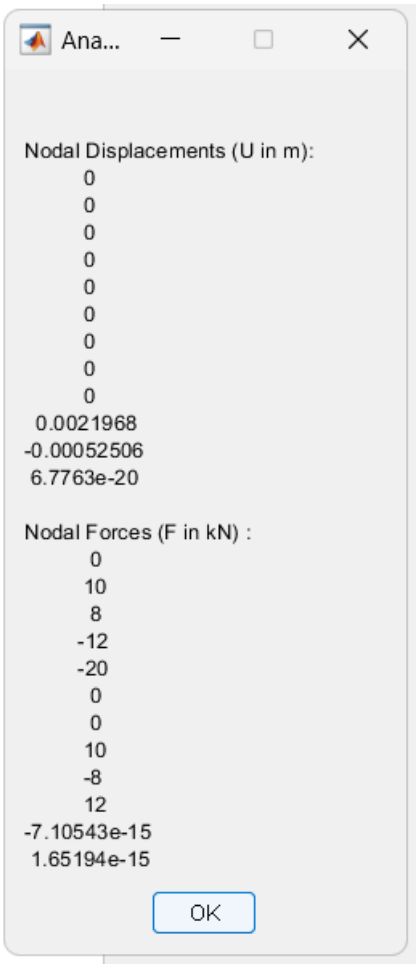


Figure 37. Results message box.

These results perfectly matched our validation problem according to the referenced book. In figure 38, you can see the displacement values from reference book solution for validation problem.

```
U =

         0
         0
         0
         0
         0
         0
         0
         0
         0
    0.0015
   -0.0005
   -0.0000
```

Figure 38. Nodal Displacement values from book solution [4].

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

```
F =

    0.0000
   10.0000
    8.0000
  -12.0000
  -20.0000
   -0.0000
    0.0000
   10.0000
   -8.0000
   12.0000
   -0.0000
   -0.0000
```

Figure 39. Nodal forces from book solution [4].

In figure 37, Since $U_Z, F_y \; and \; F_Z$ at node 4 are significantly small, the book accepted those values as 0.

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

```
Stress in Element 1: -1.2806e+04 kPa
Stress in Element 2: 1.1662e+04 kPa
Stress in Element 3: -1.2806e+04 kPa
```

Figure 40. Program results for element stresses.

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

```
» sigma1=SpaceTrussElementStress
         (E,L1,theta1x,theta1y,theta1z,u1)

sigma1 =

 -1.2806e+004


» sigma2=SpaceTrussElementStress
         (E,L2,theta2x,theta2y,theta2z,u2)

sigma2 =

  1.1662e+004


» sigma3=SpaceTrussElementStress
         (E,L3,theta3x,theta3y,theta3z,u3)

sigma3 =

 -1.2806e+004
```

Figure 41. Reference book results for element stresses [4].

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

Another thing from the results section for my program is the geometry of the validation problem. The spatial truss structure from figure 14 is plotted and visualized by my program as shown in figure 42.

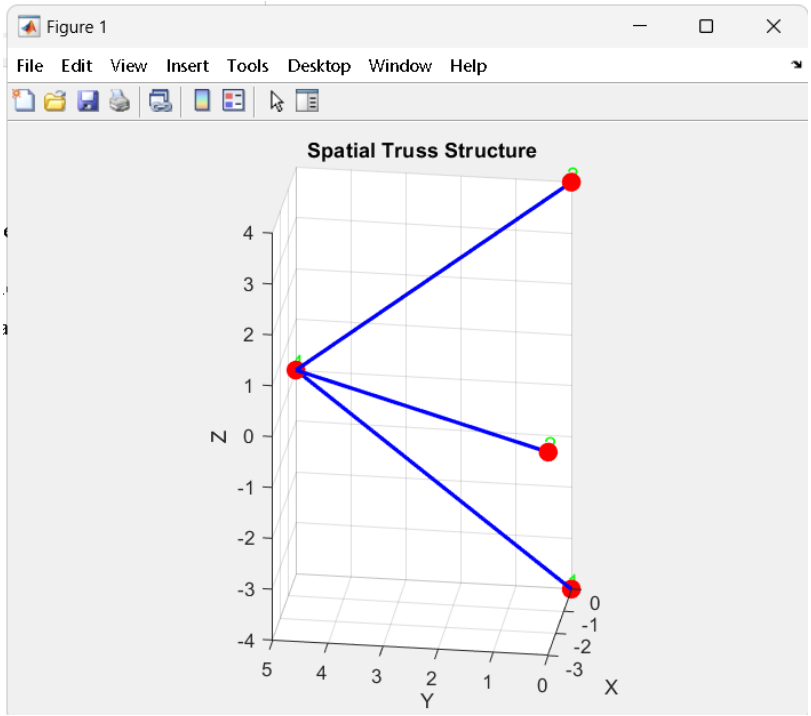


Figure 42. Validation problem structure plotted by my program.

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

As we can see from figure 14, the geometry drawn by the program for validation problem is correct, just oriented for user to easily understand the figure of the spatial truss structure.

Last thing for my program's results section is stress visualization figure. In this figure there is a stress scale in kPa units on the right-hand side of the figure and there is again the geometry of truss structure drawn by program. The importance of this figure is elements in geometry are colorized according to the true stresses applied on them. Most stress applied element is colorized to red, less stress applied element is colorized to blue. In between these two colors colorization is made by program according to stress scale on the right-hand side of the figure. The spatial truss structure with stress visualization figure is shown below for validation problem.

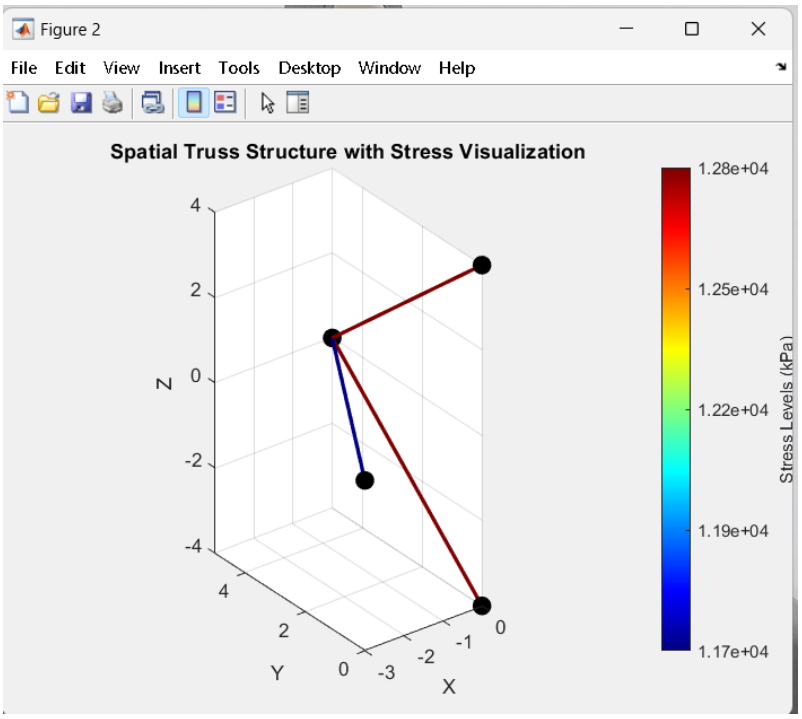


Figure 43. Spatial truss structure with stress visualization for validation problem.

As you can see from figure 41, stress visualization and true stress values are perfectly matched to reference book.

5.CONCLUSION

In this project, I have explored the finite element method through extensive research from various sources. Initially, I delved into the history of finite element analysis, providing an introductory overview of its general principles and applications. Following this foundational knowledge, I focused on specific elements such as the spring element, bar element, and spatial truss element. These topics were examined in depth, particularly their mathematical models, to gain a comprehensive understanding of their behavior and applications within finite element analysis.

The preparation for the next semester was thoroughly completed, building a strong theoretical base. With this foundation, the development of a MATLAB program for Finite Element Analysis (FEA) of spatial truss structures began. The initial phase involved drafting a preliminary version of the program, which outlined its structure and functionality. This draft was essential in providing a clear roadmap, facilitating a systematic and organized approach to the coding process.

To ensure the program's accuracy and reliability, multiple example problems were tested. These examples were carefully selected from referenced textbooks, serving as benchmarks to gauge the program's performance. The results were encouraging, as the program successfully computed the displacements and reactions for these examples, aligning perfectly with the exact solutions provided in the literature.

One of the significant outcomes of this project is the generation of visual figures by the MATLAB code. The program is capable of creating detailed plots of the spatial truss structures, clearly illustrating the nodal coordinates and the connections between elements. Additionally, it can visualize the stress distribution within the truss elements using a color-coded scheme, which provides an intuitive understanding of stress variations. These figures not only enhance the interpretability of the results but also serve as valuable tools for verifying and presenting the analysis in a visually appealing manner.

The development and testing of the MATLAB program for spatial truss analysis were successfully completed, demonstrating its capability to accurately analyze and solve spatial truss structures. The process, which began with drafting the program and followed a meticulously planned work strategy, concluded with rigorous testing against textbook examples. This systematic approach ensured the robustness and reliability of the implemented code.

Moreover, the original contribution of this project to the literature is its fully automated nature. While there have been other projects and studies on similar topics, this project distinguishes itself by providing a fully automated solution. This automation significantly reduces the manual input required, streamlining the analysis process, and making the tool more user-friendly and efficient.

Cost analysis is not suitable for this project since this project focused on developing a MATLAB program for spatial truss analysis.

In conclusion, the project not only facilitated the creation of a reliable MATLAB program for spatial truss analysis but also reinforced the theoretical concepts of finite element analysis. The successful validation of the program through benchmark problems underscores its potential as a valuable tool for structural analysis. The ability to generate informative visual figures further enhances its applicability and utility. The fully automated nature of the program marks a significant advancement, setting it apart from previous works and contributing original value to the literature. This project lays a solid foundation for future research and applications in structural engineering, marking a significant step forward in the academic and practical understanding of finite element analysis.

REFERENCES

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

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

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

[4] Kattan, P. I. (2003). MATLAB guide to finite elements: an interactive approach. Springer Science & Business Media.