INDIAN INSTITUTE OF TECHNOLOGY, MADRAS

DEPARTMENT OF OCEAN ENGINEERING

TERM PAPER ON

SPACE TRUSS ANALYSIS USING MATLAB

UNDER GUIDANCE

PROF. S. A. SANNASIRAJ

SUBMITTED BY

VIKASH C SHARMA (OE23M030)

SHUBHAM SINGH (OE23M021)

AMIT PAWAR (OE23M024)

INTRODUCTION

In the realm of structural engineering and construction, the utilization of 3D space trusses has become ubiquitous due to their remarkable ability to efficiently support loads over large spans while minimizing material usage. These intricate structural systems are composed of straight members arranged in a three-dimensional space to form interconnected triangular or tetrahedral units, offering inherent rigidity and strength. As a result, space trusses have emerged as preferred choices for roofing structures, owing to their superior rigidity in all three dimensions compared to traditional trusses. However, the analysis of space trusses poses significant challenges, especially when subjected to complex external loads such as tension and compression. Finite Element Analysis (FEA) serves as a powerful tool for simulating and understanding the behavior of these structures under various loading conditions. Despite its effectiveness, FEA can become increasingly intricate as the size of the matrix representing the truss increases, demanding sophisticated computational techniques for accurate analysis.

In this context, MATLAB, a software developed by MathWorks, Inc., emerges as a pivotal tool for the analysis of 3D space trusses. Renowned for its robust matrix operations capabilities, MATLAB provides a versatile programming environment equipped with functions tailored for matrix manipulations, conditionals, and loops. Moreover, MATLAB offers powerful graphical plotting tools, facilitating 2D and 3D visualization of structural configurations and load distributions. This term paper aims to provide a comprehensive overview of the analysis of 3D space trusses using MATLAB. Through a systematic exploration of MATLAB's functionalities, we delve into the intricacies of modeling and simulating space trusses, considering various loading scenarios and structural configurations. By elucidating the application of MATLAB in structural analysis, this paper seeks to underscore its significance in engineering practice, particularly in the domain of structural design and optimization.

Through a blend of theoretical insights and practical demonstrations, this paper endeavors to elucidate the efficacy of MATLAB as a versatile tool for the analysis and design of 3D space trusses. By navigating through the complexities of structural analysis, we aim to equip engineers and researchers with the necessary knowledge and skills to leverage MATLAB effectively in tackling real-world challenges in structural engineering and construction.

ABSTRACT

Finite Element Analysis (FEA) stands as a cornerstone technique in the evaluation of three-dimensional trusses, employing the stiffness matrix methodology. This paper presents a comprehensive overview of the FEA process applied to truss structures, facilitated by MATLAB, a renowned software developed by The MathWorks, Inc. The methodology outlined encompasses nine key steps: structure discretization, node and element numbering, material behavior identification, determination of element stiffness matrices, assembly of the global stiffness matrix, imposition of boundary conditions, application of forces, computation of unknown displacements, and derivation of stress and strain distributions within each truss member.

Through MATLAB's robust matrix operations and versatile programming environment, engineers can efficiently conduct structural analyses, navigating complexities inherent in truss systems. This paper elucidates each step of the FEA process, emphasizing MATLAB's pivotal role in executing tasks such as numerical computation, visualization, and result interpretation. Furthermore, MATLAB's extensive toolbox offerings and user-friendly graphical interfaces augment its utility in engineering applications, enabling practitioners to tackle real-world challenges with precision and efficacy.

By exploring the integration of FEA and MATLAB in truss analysis, this paper not only provides a theoretical framework but also offers practical insights into structural engineering practices. The methodologies discussed herein serve as a valuable resource for engineers and researchers seeking to leverage computational tools for the design, optimization, and evaluation of complex truss structures. Ultimately, this paper underscores the significance of MATLAB as a powerful software platform in advancing the field of structural engineering, paving the way for innovative solutions to modern engineering challenges.

PROBLEM STATEMENT

The aim of this term paper is to analyze a specific space truss structure, as illustrated in Figure. The truss is supported at nodes **2** coordinate **(50, 30 0)**, node **3** coordinate **(0, 40, 80)** and node **4** coordinate **(0, 0, 60)** by ball-and-socket joints that permit rotation but not translation. With known material properties **(E = 200 GPa)** and cross-sectional areas **(A14 = 0.01 m^2, A12 = 0.01 m^2, A13 = 0.01 m^2)**, and subjected to a load of **P = 12 kN** at node **1** coordinate **(40, 24, 0)**, the following objectives will be pursued:

1. Develop the global stiffness matrix for the given space truss structure.
2. Determine the displacements at node **1** resulting from the applied load.
3. Calculate the reactions at nodes **2**, **3** and **4** due to the loading conditions.
4. Analyze and compute the stress experienced by each element within the space truss under the applied load.

Through comprehensive analysis and computations, this study seeks to provide a detailed understanding of the structural behavior of the space truss and its response to external loads, thereby contributing to the knowledge of space truss analysis and design principles.

METHODOLOGY

Discretization of the structure

Nodes and elements numbering

Identify material behaviour (E,A)

Element stiffness matrix (K)

Assemble stiffness matrix

Apply boundary conditions

Find unknown displacements using Gauss elimination method

Finding reaction

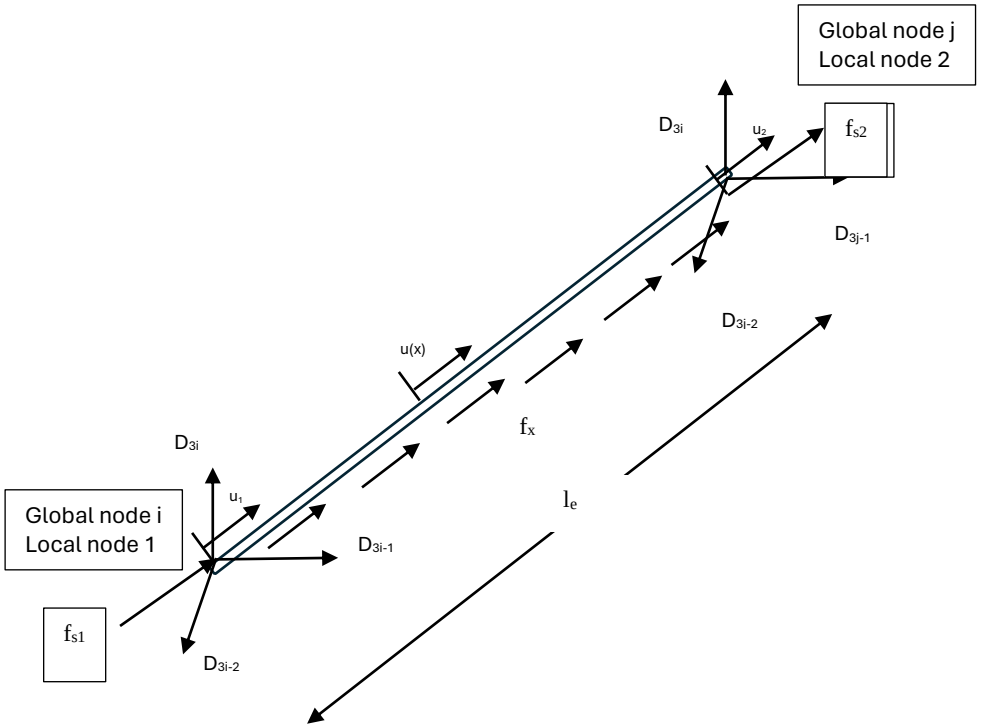
Find stress and strain on each member

Stiffness Matrix derivation

Imagine a structure comprising several trusses or bar members. Each member can be viewed as a uniform cross-section truss/bar element bounded by two nodes (node = 2). Let's focus on a bar element with nodes 1 and 2 positioned at each end. The length of the element is represented as le. The local x-axis aligns with the axial direction of the element, originating from node 1.

In the local coordinate system, each node of the element has only one degree of freedom (DOF),

representing the axial displacement. Thus, the element possesses a total of two DOFs, denoted as (node = 2).



For our stiffness matrix derivation our governing equation as follows:

$$AE\frac{du}{dx} = 0$$

In the Finite Element Method (FEM) discussed earlier, the displacement in an element should be expressed in the following form:

$$u^h(x) = N(x)\, d_e$$

We adhere to the standard procedure for constructing shape functions, assuming that the axial displacement in the truss element can be expressed in a general form.

$$u^h(x) = \propto_0 + \propto_1 * x = \{1 \quad x\}\begin{Bmatrix}\propto_0 \\ \propto_1\end{Bmatrix} = p^T \propto$$

where u^h represents the approximation of the displacement, α is the vector of two unknown constants, α_0 and α_1 is the vector of polynomial basis functions (or monomials). In this specific problem, we utilize up to the first order of polynomial basis. Depending on the nature of the problem, a higher order of polynomial basis functions could be employed. The order of polynomial basis functions up to the n_{th} order can be expressed as:

$$p^T = \{1 \quad x \quad \ldots. \quad x^n\}$$

In deriving the shape function**,** we use

$$\text{At} \quad x = 0\,, \; u(x=0) = u_1$$
$$\text{At} \quad x = l_e\,, \; u(x=l_e) = u_2$$

we have

$$\begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & l_e \end{bmatrix} \begin{Bmatrix} \propto_0 \\ \propto_1 \end{Bmatrix}$$

$$\begin{Bmatrix} \propto_0 \\ \propto_1 \end{Bmatrix} = \begin{bmatrix} 1 & 0 \\ -1/l_e & 1/l_e \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix}$$

Solving the above equation for a, we have

$$u(x) = P^T \alpha = \{1 \;\; x\} \begin{bmatrix} 1 & 0 \\ -\frac{1}{l_e} & \frac{1}{l_e} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

$$u(x) = \begin{Bmatrix} 1 - \frac{x}{l_e} & \frac{x}{l_e} \\ N_1(x) & N_2(x) \end{Bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = N(x) d_e$$

The matrix of shape functions is then obtained in the form

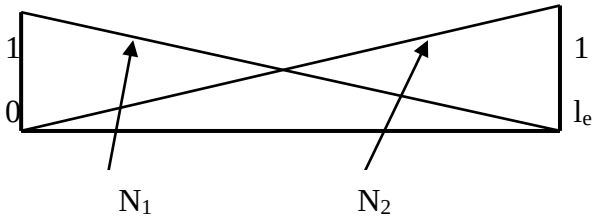
$$\text{N(x)} = [N_1\,(X) \quad N_2\,(X)]$$

where the shape functions for a truss element can be written as

$$N_1\,(X) = 1 - \frac{X}{l_e}$$

$$N_2\,(X) = \frac{X}{l_e}$$

We obtained two shape functions because we have two DOFs in the truss elements.



It can be readily verified that these two shape functions satisfy the delta function property as defined by Equation and the partitions of unity as defined by Equation. We leave this confirmation to the reader as a simple exercise.

The graphical representation of the linear shape functions is depicted in Figure. It is evident that N_i

illustrates the shape of the contribution from nodal displacement at node i, hence its designation as a shape function. In this instance, the shape functions exhibit linear variation across the element, earning them the designation of linear shape functions. Substituting Equations we have

$$u(x) = N_1(x)u_1 + N_2(x)u_2 = u_1 + \frac{u_2 - u_1}{l_e}x$$

which clearly states that the displacement within the element varies linearly. The element is therefore called a linear element.

In a truss structure, there exists only one stress component denoted as σ. The corresponding strain associated with this stress can be determined by

$$\in_x = \frac{du}{dx} = \frac{u_2 - u_1}{l_e}$$

This singular stress component, denoted as σ_x, arises directly from the differentiation of the equation with respect to x. It's worth noting that the strain mentioned in the equation remains constant throughout the element.

Differentiating with respect to x, we obtain the differential equation

$$\frac{d}{dx}\left(\mathrm{AE}\frac{du}{dx}\right) = 0$$

Applying Galerkin's criterion,

$$\int_0^l \frac{d}{dx}\left(\mathrm{AE}\frac{du}{dx}\right) \mathrm{N_i dx} = 0$$

Now we have to apply integration by parts

$$AE \int_0^l \frac{\mathrm{dNi}}{dx}\begin{bmatrix} -\frac{1}{l} & \frac{1}{l} \end{bmatrix} \mathrm{dx} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \left[\mathrm{Ni}AE\ \frac{du}{dx}\right]_0^l$$

$$AE \int_0^l \left[-\frac{1}{l}\right]\begin{bmatrix} -\frac{1}{l} & \frac{1}{l} \end{bmatrix} \mathrm{dx} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix} = \mathrm{f_1 x}$$

$$\boxed{\begin{Bmatrix} f_{1x} \\ f_{2x} \end{Bmatrix} = AE \begin{bmatrix} -\frac{1}{l} & \frac{1}{l} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \end{Bmatrix}}$$

Element Matrices in the Local Coordinate System:

Upon acquiring the strain matrix B, we can proceed to derive the stiffness matrix for truss elements, employing the equation mentioned in the preceding discussion:

$$k = \frac{AE}{l}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

The area of the cross-section of the truss element is denoted by A in the equation. It's important to highlight that the material constant matrix, denoted as c, simplifies to the elastic modulus, E, for the one-dimensional truss element, as illustrated in Equation.

Additionally, it's worth noting that the element stiffness matrix, as depicted in Equation, exhibits symmetry.

$$f_e = \int_{V_e} N^T f_b dV + \int_{S_c} N^T f_s dS = f_x \int_0^{l_e} \begin{bmatrix} N_1 \\ N_2 \end{bmatrix} dx + \begin{Bmatrix} f_{s_1} \\ f_{s_2} \end{Bmatrix} = \begin{Bmatrix} \frac{f_x l_e}{2} + f_{s_1} \\ \frac{f_x l_e}{2} + f_{s_1} \end{Bmatrix}$$

$$D_e = \begin{Bmatrix} D_{3i-2} \\ D_{3i-1} \\ D_{3i} \\ D_{3j-2} \\ D_{3j-1} \\ D_{3j} \end{Bmatrix}$$

Element Matrices in the Global Coordinate System:

$$d_e = \mathrm{TD_e}$$

T is the transformation matrix for the truss element, given by

$$\mathrm{T} = \begin{bmatrix} \mathrm{l}_{ij} & \mathrm{m}_{ij} & \mathrm{n}_{ij} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathrm{l}_{ij} & \mathrm{m}_{ij} & \mathrm{n}_{ij} \end{bmatrix}$$

$$\mathrm{l}_{ij} = cos(\mathrm{x,X}) = \frac{X_j - X_i}{l_e}$$

$$\mathrm{m}_{ij} = cos(\mathrm{x,Y}) = \frac{Y_j - Y_i}{l_e}$$

$$\mathrm{n}_{ij} = cos(\mathrm{x,Z}) = \frac{Z_j - Z_i}{l_e}$$

are the direction cosines of the axial axis of the element. It is easy to confirm that

$$\mathrm{TT^T = I}$$

where I is an identity matrix of 2 x 2. Therefore, matrix T is an orthogonal matrix. The length of the element, le, can be calculated using the global coordinates of the two nodes of the element by

$$l_e = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}$$

The transformation matrix also applies to the force vectors between the local and global coordinate systems:

$$F_e = \begin{Bmatrix} F_{3i-2} \\ F_{3i-1} \\ F_{3i} \\ F_{3j-2} \\ F_{3j-1} \\ F_{3j} \end{Bmatrix}$$

$$f_e = TF_e$$

Pre-multiply Tr to both sides in the above equation to obtain:

$$\mathrm{K_e = T^T\, k_e\, T}$$

$$= \frac{AE}{l_e} \begin{bmatrix} l_{ij}^2 & l_{ij}m_{ij} & l_{ij}n_{ij} & -l_{ij}^2 & -l_{ij}m_{ij} & -l_{ij}n_{ij} \\ l_{ij}m_{ij} & m_{ij}^2 & m_{ij}n_{ij} & -l_{ij}m_{ij} & -m_{ij}^2 & -m_{ij}n_{ij} \\ l_{ij}n_{ij} & m_{ij}n_{ij} & n_{ij}^2 & -l_{ij}n_{ij} & -m_{ij}n_{ij} & -n_{ij}^2 \\ -l_{ij}^2 & -l_{ij}m_{ij} & -l_{ij}n_{ij} & l_{ij}^2 & l_{ij}m_{ij} & l_{ij}n_{ij} \\ -l_{ij}m_{ij} & -m_{ij}^2 & -m_{ij}n_{ij} & l_{ij}m_{ij} & m_{ij}^2 & m_{ij}n_{ij} \\ -l_{ij}n_{ij} & -m_{ij}n_{ij} & -n_{ij}^2 & l_{ij}n_{ij} & m_{ij}n_{ij} & n_{ij}^2 \end{bmatrix}$$

SOLUTION

Step 1:- Nodes coordinates and connection between elements establish the geometric layout and connectivity of the structure.

Node coordinates 1 **(40, 24, 0)**

Node coordinates 2 **(50, 30, 0)**

Node coordinates 3 **(0, 40, 80)**

Node coordinates 4 **(0, 0, 60)**

Connection between elements **[1 2], [1 3]** and **[1 4].**

Step 2:- Discretization of the matrix involves breaking down the stiffness matrix of each element into smaller components within its local coordinate system.

Step 3:- Global stiffness matrix:- The global stiffness matrix combines the contributions from all elements to represent the overall structural stiffness.

Stiffness matrix(K) =

137657.3	78328.66	-19466	-126102	-75661.1	0	-4265.75	1706.3	8531.501	-7289.69	-4373.82	10934.54
78328.66	48703.49	-3148.12	-75661.1	-45396.7	0	1706.3	-682.52	-3412.6	-4373.82	-2624.29	6560.723
-19466	-3148.12	33464.81	0	0	0	8531.501	-3412.6	-17063	10934.54	6560.723	-16401.8
-126102	-75661.1	0	126101.9	75661.14	0	0	0	0	0	0	0
-75661.1	-45396.7	0	75661.14	45396.68	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
-4265.75	1706.3	8531.501	0	0	0	4265.751	-1706.3	-8531.5	0	0	0
1706.3	-682.52	-3412.6	0	0	0	-1706.3	682.5201	3412.6	0	0	0
8531.501	-3412.6	-17063	0	0	0	-8531.5	3412.6	17063	0	0	0
-7289.69	-4373.82	10934.54	0	0	0	0	0	0	7289.692	4373.815	-10934.5
-4373.82	-2624.29	6560.723	0	0	0	0	0	0	4373.815	2624.289	-6560.72
10934.54	6560.723	-16401.8	0	0	0	0	0	0	-10934.5	-6560.72	16401.81

$$[K]\{u\} = \{F\}$$

$$[K]\begin{Bmatrix} u_{1x} \\ u_{1y} \\ u_{1z} \\ u_{2x} \\ u_{2y} \\ u_{2z} \\ u_{3x} \\ u_{3y} \\ u_{3z} \\ u_{4x} \\ u_{4y} \\ u_{4z} \end{Bmatrix} = \begin{Bmatrix} F_{1x} \\ F_{1y} \\ F_{1z} \\ F_{2x} \\ F_{2y} \\ F_{2z} \\ F_{3x} \\ F_{3y} \\ F_{3z} \\ F_{4x} \\ F_{4y} \\ F_{4z} \end{Bmatrix}$$

Step 4:- Boundary condition:- Constrain at support 2, 3, and 4. Applying force at node 1 that is 12kN in x-direction, 0kN in y-direction and 0kN in z-direction.

$$F_{1x} = 12kN, F_{1y} = 0, F_{1z} = 0$$

$$u_{2x} = u_{2y} = u_{2z} = u_{3x} = u_{3y} = u_{3z} = u_{4x} = u_{4y} = u_{4z} = 0$$

$$[K]\begin{Bmatrix} u_{1x} \\ u_{1y} \\ u_{1z} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix} = \begin{Bmatrix} 12 \\ 0 \\ 0 \\ F_{2x} \\ F_{2y} \\ F_{2z} \\ F_{3x} \\ F_{3y} \\ F_{3z} \\ F_{4x} \\ F_{4y} \\ F_{4z} \end{Bmatrix}$$

Step 5:- Displacement at nodes:- Displacement at nodes indicates how much each node has moved or deformed under applied loads.

$$u = \begin{Bmatrix} 0.002203 \\ -0.00348 \\ 0.000954 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{Bmatrix}$$

Step 6:- Reaction at nodes:- Reaction at nodes refers to the forces and moments exerted on the nodes due to constraints or external loads.

$$R = \begin{Bmatrix} 12 \\ 0 \\ 0 \\ -14.4 \\ -8.64 \\ 0 \\ -7.2 \\ 2.88 \\ 14.4 \\ 9.6 \\ 5.76 \\ -14.4 \end{Bmatrix}$$

Step 7:- Stress in elements:- Stress in elements reveals the internal forces and deformations experienced by each element, crucial for assessing structural integrity and performance.

$$\sigma = \begin{Bmatrix} -1679.3141 \\ 1635.5256 \\ -1824 \end{Bmatrix}$$

Step 8 :- Strain in elements:- rain in truss members refers to the amount of deformation or elongation experienced by the individual structural components of a truss under load. It is crucial for determining the overall stability and safety of the truss structure.

$$\in = \begin{Bmatrix} -8.4E-6 \\ -8.18E-6 \\ -9.1E-6 \end{Bmatrix}$$

CONCLUSION

In conclusion, this term paper has delved into a comprehensive analysis of a specific space truss structure, aiming to understand its structural behavior under external loads. The methodology outlined a systematic approach, starting from establishing node coordinates and element connections to computing global stiffness matrices, displacements, reactions at nodes, and stresses in elements.

By developing the global stiffness matrix, determining node displacements, calculating reactions, and analyzing element stresses, a detailed insight into the structural response of the space truss was obtained. The results contribute significantly to the knowledge of space truss analysis and design principles, highlighting the importance of understanding structural behavior for optimizing performance and ensuring structural integrity.

This study underscores the significance of computational methods and analytical techniques in structural engineering, emphasizing the need for accurate modeling and analysis for effective design and assessment of complex structures like space trusses. Overall, the findings of this term paper provide valuable insights into the behavior of space trusses under loading conditions, laying a foundation for further research and development in this field.

REFERENCE

- FEM Classnotes By Prof. S A Sannasiraj
- MATLAB Guide to Finite Elements: An Interactive Approach by Peter I. Kattan
- DaryILogan - A First Course in the Finite Element Method

MATLAB code

```
clc;clear all;close all;
%% % Input Data
NN = xlsread('Truss_input.xls','Trussinput','F3');%Number of Nodes
Coor = xlsread('Truss_input.xls','Trussinput','D9:F100');%Coordinate
for j = 1:NN
    i = j;
    X(i) = Coor(j,1);Y(i) = Coor(j,2);Z(i) = Coor(j,3);
end
NE = xlsread('Truss_input.xls','Trussinput','F4');%Number of elements
Pro = xlsread('Truss_input.xls','Trussinput','H9:L100');%Connectivity
for j = 1:NE
    i = Pro(j,1);E(i) = Pro(j,2);A(i) = Pro(j,3);
    Dir(i,1) = i;Dir(i,2) = Pro(j,4);Dir(i,3) = Pro(j,5);
end
NC = xlsread('Truss_input.xls','Trussinput','R3');%No. of constraints
Cons = xlsread('Truss_input.xls','Trussinput','P9:Q141');
NF = xlsread('Truss_input.xls','Trussinput','Y3');
Forc = xlsread('Truss_input.xls','Trussinput','V9:Y100');
elementNodes = xlsread('Truss_input.xls','Trussinput','K9:L100');
%% % Define Global Stiffness Matrix Element for each element
for i = 1:NE
    L(i) = sqrt((X(Dir(i,3)) - X(Dir(i,2)))^2 + (Y(Dir(i,3)) - Y(Dir(i,2)))^2 +
(Z(Dir(i,3)) - Z(Dir(i,2)))^2);
    Cx(i) = (X(Dir(i,3)) - X(Dir(i,2))) / L(i);
    Cy(i) = (Y(Dir(i,3)) - Y(Dir(i,2))) / L(i);
    Cz(i) = (Z(Dir(i,3)) - Z(Dir(i,2))) / L(i);
end
%% plot
figure;
hold on;
for i = 1:NE% Plot elements
    node1 = elementNodes(i, 1);node2 = elementNodes(i, 2);
    plot3([Coor(node1, 1), Coor(node2, 1)], [Coor(node1, 2), Coor(node2, 2)],
[Coor(node1, 3), Coor(node2, 3)], 'b');
end
scatter3(Coor(:, 1), Coor(:, 2), Coor(:, 3), 'r', 'filled');
for i = 1:NN% Display node coordinates
    text(Coor(i, 1), Coor(i, 2), Coor(i, 3), sprintf('(%g, %g, %g)', Coor(i, 1), Coor(i,
2), Coor(i, 3)), 'FontSize', 8);
end
xlabel('X'); ylabel('Y'); zlabel('Z');
title('Nodes,Elements');grid on; axis equal;
%% % Initialize the stiffness matrix

K = zeros(3*NN,3*NN);
for i = 1:NE % Loop to calculate k1, k2, and k3 matrices
    k_i = (A(i) * E(i) / L(i)) * [Cx(i)^2, Cx(i)*Cy(i), Cx(i)*Cz(i), -Cx(i)^2, -
Cx(i)*Cy(i), -Cx(i)*Cz(i);
                                  Cx(i)*Cy(i), Cy(i)^2, Cy(i)*Cz(i), -Cx(i)*Cy(i), -
Cy(i)^2, -Cy(i)*Cz(i);
                                  Cx(i)*Cz(i), Cy(i)*Cz(i), Cz(i)^2, -Cx(i)*Cz(i), -
Cy(i)*Cz(i), -Cz(i)^2;
                                 -Cx(i)^2, -Cx(i)*Cy(i), -Cx(i)*Cz(i), Cx(i)^2,
Cx(i)*Cy(i), Cx(i)*Cz(i);
                                 -Cx(i)*Cy(i), -Cy(i)^2, -Cy(i)*Cz(i), Cx(i)*Cy(i),
Cy(i)^2, Cy(i)*Cz(i);
```

```
                                    -Cx(i)*Cz(i), -Cy(i)*Cz(i), -Cz(i)^2, Cx(i)*Cz(i),
Cy(i)*Cz(i), Cz(i)^2];
    indice = elementNodes(i, :);% Element degrees of freedom (Dof)
    elementDof = [3*indice(1)-2, 3*indice(1)-1, 3*indice(1), 3*indice(2)-2, 3*indice(2)-
1, 3*indice(2)];
    K(elementDof, elementDof) = K(elementDof, elementDof) + k_i;% Assemble the global
stiffness matrix
end
disp('Global Stiffness(kN/m):');disp(K);

%% % Calculate the bandwidth of the stiffness matrix K
HBW = halfBandwidthMatrix(K, 1);
disp('Half bandwidth matrix:');
disp(HBW);
%% % Definition of Primary Nodal Forces
F = zeros(3*NN,1);
for i = 1:NF
    f = 3*Forc(i,1);
    F(f-2,1) = Forc(i,2);
    F(f-1,1) = Forc(i,3);
    F(f,1) = Forc(i,4);
end
S = K;% Elimination of rows and columns of K-matrix with respect to Supports
for i = 1:NC
    r = 3*Cons(i,1);
    if Cons(i,2) == 0
        S(r-2,:) = 0; S(:,r-2) = 0; S(r-2,r-2) = 1;
        S(r-1,:) = 0; S(:,r-1) = 0; S(r-1,r-1) = 1;
        S(r,:) = 0; S(:,r) = 0; S(r,r) = 1;
    elseif Cons(i,2) == 1
        S(r-2,:) = 0; S(:,r-2) = 0; S(r-2,r-2) = 1;
    elseif Cons(i,2) == 2
        S(r-1,:) = 0; S(:,r-1) = 0; S(r-1,r-1) = 1;
    elseif Cons(i,2) == 3
        S(r,:) = 0; S(:,r) = 0; S(r,r) = 1;
    end
end

%% Gauss elimination method
GM = rref([S F]);
d = GM(:, end);
disp(d);
disp('Displacements (m):');% Display displacements
for i = 1:NN
     fprintf('dx%g',i); fprintf('=%g\n',d(3*i-2))
     fprintf('dy%g',i); fprintf('=%g\n',d(3*i-1))
     fprintf('dz%g',i); fprintf('=%g\n',d(3*i))
end
disp('-------------------------------')

disp('Support Reactions:(kN)')
%% % Calculation of Nodal Reaction, stress and strain
W = K * d;
for i = 1:NN
    fprintf('Wx%g',i); fprintf('=%g\n',W(3*i-2))
    fprintf('Wy%g',i); fprintf('=%g\n',W(3*i-1))
    fprintf('Wz%g',i); fprintf('=%g\n',W(3*i))
end
disp('-------------------------------')
disp('Stress (kPa):')
```

```
ff = zeros(NE, 6);
for e = 1:NE
    indice = elementNodes(e, :);
    elementDof = [3 * indice(1) - 2, 3 * indice(1) - 1, 3 * indice(1), 3 * indice(2) -
2, 3 * indice(2) - 1, 3 * indice(2)];
    x1 = Coor(indice(1), 1); y1 = Coor(indice(1), 2); z1 = Coor(indice(1), 3);
    x2 = Coor(indice(2), 1); y2 = Coor(indice(2), 2); z2 = Coor(indice(2), 3);
    L = sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2);
    Cx = (x2 - x1) / L; Cy = (y2 - y1) / L; Cz = (z2 - z1) / L;
    u = d(elementDof);
    memberstress(e) = E(e) / L * [-Cx, -Cy, -Cz, Cx, Cy, Cz] * u;
    fprintf('%3d %12.8f\n', e, memberstress(e));
end
disp('-------------------------------')
disp('Elements Strain:')
for i = 1:NE
    strain(i)=memberstress(i)/E(i);
    fprintf('Strain %g:\t', i);
    fprintf('%g\n', strain(i));
end
%Write the matrices to Excel
xlswrite('Trussresult2.xlsx', K, 'Stiffness_Matrix');
xlswrite('Trussresult2.xlsx', d, 'Displacements');
xlswrite('Trussresult2.xlsx', W,'Support_Reactions');
xlswrite('Trussresult2.xlsx', memberstress', 'Member_Stresses');
xlswrite('Trussresult2.xlsx', strain', 'Member_Strain');
xlswrite('Trussresult2.xlsx', HBW', 'Half_band_width_Matrix');
disp('Results have been successfully exported to Excel.');

%% Function for Half band width matrix
function HBW = halfBandwidthMatrix(K, half_bandwidth)
    % Get the size of matrix A
    [m, n] = size(K);
    % Initialize a zero matrix for the half-bandwidth matrix
    HBW = zeros(m, half_bandwidth*2+1);

    % Loop through the elements of A to form the half-bandwidth matrix
    for i = 1:m
        for j = max(1, i - half_bandwidth):min(n, i + half_bandwidth)
            HBW(i, j - (i - half_bandwidth) + 1) = K(i, j);
        end
    end
end
```