**\*\*Title: Finite Element Analysis of 2D Geometry\*\***

**\*\*Your Name\*\***

**\*Affiliation\***

**Email Address**

Word Count: [2368]

**Abstract**

This study conducted a detailed Finite Element Analysis (FEA) to explore the structural response of a 2D model under varied loading angles, utilizing a custom MATLAB script. The primary objective was to investigate how the structure's displacement and strain distributions are affected by directional loading. The analysis focused on a geometric mesh comprising quadrilateral elements, subjected to forces at angles of 0°, 45°, and 90°. Material properties were varied in terms of Young's Modulus (30e9, 40e9, and 50e9 Pascal), while maintaining a constant Poisson's Ratio of 0.3, to simulate different stiffness conditions. The methodology involved the computation of the global stiffness matrix for each scenario, followed by the application of fixed boundary conditions at selected nodes to replicate real-world structural constraints.

Key findings indicated a pronounced impact of load orientation on the structural behaviour, particularly in terms of strain and stress distribution. The analysis revealed that varying the Young's Modulus significantly alters the displacement responses, highlighting the importance of material selection in design considerations. Results were systematically recorded in text files for each combination of load angle and material stiffness, providing a comprehensive dataset for the structural response.

The study concludes that directional loading plays a critical role in structural behaviour, underlining the need for careful consideration of load angles in structural design and integrity assessments. The findings offer valuable insights for engineers and designers in optimizing structures to withstand different loading conditions, thereby enhancing their performance and safety.

**Introduction**

**1. Problem Description**

In the realm of structural engineering and design, understanding how structures respond to varying loads is paramount. The problem addressed in this Finite Element Analysis (FEA) study is the investigation of a structure's behaviour under different loading angles and material properties, a crucial aspect often encountered in real-world engineering scenarios. This analysis is particularly significant as loads on structures do not always apply vertically but can occur in various directions due to wind, earthquakes, or other operational conditions.

The importance of this study lies in its ability to provide insights into the directional stiffness and strength of structures. Different load angles can lead to varying stress distributions and deformation patterns, which are critical for assessing the structural integrity and safety. Inadequate understanding and planning for these variations can lead to design flaws, resulting in over-engineered or underperforming structures, potentially causing safety hazards and increased costs.

In this study we developed model using one of many computational methods in MATLAB for finite element analysis. After that we use ANSYS Mechanical APDL to validate our model.

**Literature Review**

The exploration of structural response under varying loading conditions is a well-established area in structural engineering, with a rich body of research. This study builds upon several key themes and findings from existing literature:

1. **Directional Loading on Structures:**

Past research has extensively analysed the effects of directional loads, primarily focusing on wind and seismic forces. Notably, studies by(Hammad & Moustafa, 2021) have highlighted how non-vertical loads can significantly alter stress distribution and deformation patterns in buildings and bridges.

1. **Finite Element Analysis (FEA) in Structural Engineering:**

FEA has been a cornerstone technique for analysing structural behaviour under various conditions. The work of (Sussman, Bathe, & Structures, 1987) has been instrumental in advancing FEA methodologies, providing a robust framework for simulating complex load scenarios on different structures.

1. **Material Properties and Structural Response:**

The influence of material properties, such as Young's Modulus, on the structural response has been a key focus area. Research by (Meo, Rossi, & Technology, 2006)and (Lazarus et al., 2020) has shed light on how material stiffness impacts the ability of a structure to withstand different loading orientations, emphasizing the need for material optimization in design.

**iv) Impact of Load Angles on Structures:**

The specific area of load angles and their impact on structural integrity has seen growing interest. Studies by (Jockwer & Dietsch, 2018) have demonstrated that load angles can dramatically influence the failure modes and safety margins of structures, advocating for more comprehensive load angle considerations in design standards.

**v) Structural Optimization and Safety:**

This study aims to contribute to this body of knowledge by providing a focused analysis of how varying load angles and young’s modulus affect a 2D structural model, utilizing and expanding upon the established principles of FEA. The integration of these diverse insights from past literature forms the foundation for the current investigation, aiming to offer practical implications for structural design and safety in the face of complex loading scenarios.

**Aim of the Study**

The primary aim of this Finite Element Analysis (FEA) study is to comprehensively understand and quantify the impact of varying load angles on the structural behaviour of a 2D model. Specifically, the research seeks to achieve the following objectives:

**i) Evaluate Structural Response:** To analyse how the structure responds in terms of displacement and strain under loads applied at different angles (0°, 45°, and 90°). This aspect is crucial for assessing the directional dependence of the structure's performance.

**ii) Assess the Influence of Material Properties:** To investigate the role of material stiffness, represented by varying Young's Modulus values, in influencing the structural response under these loading conditions. This will provide insight into how different materials behave under the same loading scenarios.

**iii) Identify Stress Distribution and Weak Points:** To determine the stress distribution patterns within the structure for each loading angle and material property. This analysis will help in identifying potential weak points or areas that are more susceptible to failure, thereby guiding design improvements.

**iv) Optimize Design and Material Selection:** By understanding the interplay between load direction, material properties, and structural behaviour, the study aims to contribute valuable data for optimizing structural designs and material selections in real-world engineering applications.

In essence, the study endeavours to bridge the gap in knowledge regarding the effect of load orientation on structural behaviour, offering practical insights for engineers and designers in the field of structural engineering.

**Materials and Methods**

1. **Selection Criteria of Materials**

Materials for the study were chosen at random in order to better understand the range of responses they had to various loading conditions. But in real-world engineering, choosing the right material is a complex process that depends on a number of important variables. Above all, the material's strength and durability should match the anticipated loading conditions. This includes taking into account the material's resilience to external forces and its elasticity, specifically its Young's Modulus. The decision-making process also heavily weighs availability, affordability, and adherence to standard practices. Sustainability and the environmental impact of materials are becoming more and more significant factors in the context of modern engineering, with an emphasis on using materials that minimise ecological footprints while yet fulfilling structural and operational requirements.

1. **Method of Computing FEA**

The "Direct Stiffness Method" is the general term for the Finite Element Analysis (FEA) approach utilised in MATLAB. The majority of FEA tools and methods, including those used in MATLAB, are based on this technique, which is a fundamental approach in computational mechanics. The description of model used in this study are given below:

1. **Model Description:**

This report presents the findings from a Finite Element Analysis (FEA) conducted to investigate the effects of varying load angles and material properties on a specified geometry. The analysis was performed using a custom MATLAB script, tailored to compute displacements, strains, and stresses under different conditions.

1. **Geometry and Mesh Description**

The geometry of the structure under analysis is defined by the following node coordinates in a 2D plane:

**- Nodes =** **[0,0; 15,0; 30,0; 30,15; 45,15; 60,15; 60,30; 45,30; 30,30; 15,30; 0,30; 0,15; 15,15]**

This configuration forms a mesh composed of quadrilateral elements, detailed as follows

**- Elements Connectivity:**

**[ 1, 2, 13, 12;**

**13, 10, 11, 12;**

**2, 3, 4, 13;**

**13, 4, 9, 10;**

**4, 5, 8, 9;**

**5, 6, 7, 8 ]**

The mesh forms a structured grid with elements likely representing a planar structure. Final geometry of problem structure is shown in (Figure 1).

**ii) Material Properties:**

The material properties used in the analysis are:

**- Young's Modulus (E):** Varied as [30e9, 40e9, 50e9] Pascal

**- Poisson's Ratio (ν):** 0.3

**- Thickness:** 2 mm

1. **Boundary Conditions and Loads:**

The loading conditions involved applying forces at different angles (0°, 45°, 90°) on specified nodes. The forces were decomposed into X and Y components based on the angle of application. Boundary conditions illustrated in (Figure 1)

**Loading Nodes:** [ 6, 7]

The following nodes were fixed to simulate support conditions:

**- Fixed Nodes:** [1, 11]

This constrains movements in these nodes, mimicking a real-world scenario where parts of the structure are anchored.

**Element Type and Mesh Convergence:**

In this study, the Finite Element Model is constructed using quadrilateral elements, a choice guided by their versatility and effectiveness in modelling a wide range of geometries, particularly in 2D plane stress and plane strain problems. The selection of quadrilateral elements is supported by the literature that suggests their suitability for accurately representing complex shapes and boundary conditions (Zienkiewicz & Taylor, 2005)

Mesh convergence is a critical aspect of the FEA process, ensuring that the results are independent of the mesh size. To address this, a mesh convergence study was conducted. The study involved progressively refining the mesh and observing the effects on key output parameters, such as displacements and strains. This approach aligns with best practices in computational mechanics, ensuring that the final mesh is sufficiently fine to capture the necessary details of the structural response, yet coarse enough to keep computational demands within reasonable limits. Mesh convergence was done using Ansys and element size were **5 , 2.5** and **1** **mm.**

Results from each mesh are compared to check the difference. In mesh convergence the mesh is reduced until the values of resulted variables match or are so close that they consider to be same. So in this study we mesh our geometry with 3 element sizes shown in (shown in Figure 3,Figure 4 and Figure 5).

**Simulations and Measurements**

The Finite Element Analysis (FEA) in this study was conducted using a series of custom MATLAB functions, each designed to perform specific steps in the FEA process. The analysis followed a structured approach, starting from defining the geometry to post-processing the results.

**i) Solving for Displacements:**

The primary FEA equation, *[K\_global]\*[displacements] = [F\_global],* is solved. This equation relates the global stiffness matrix to the nodal displacements and the applied forces. The solution gives the displacement of each node in the structure (Table 3).

**ii) Post-Processing for Strain and Stress Calculation *(calculateStrain, calculateStress):***

* + The *calculateStrain* function computes the strain for each element based on the nodal displacements. This involves calculating the strain-displacement (B) matrix and the Jacobian for the transformation of coordinates.
  + Strain values are then used to compute stress using the `calculateStress` function, considering material properties. Calculated stress and strain tensors are shown in (Figure 2 and Figure 1)
  + Results, including displacement vectors and stress-strain distributions, are visualized for analysis.
  + Additionally, results are exported to text files for documentation and further analysis.

Each step in this process is crucial for ensuring the accuracy and reliability of the FEA. The methodology combines theoretical aspects of structural mechanics with practical computational techniques, offering a comprehensive analysis of the structure under varying load conditions.

**Results**

The analysis generated results for displacements and strains for first case with loading of 200 MPa along x-axis on right side of the geometry while elastic Modulus is kept at 40 GPa. The second part of the study is to analyse results from each of the three load angles and three values of young’s modulus. These results were saved in text files. The files contain raw data outlining how the structure responded to the varying conditions. The Matrix computed from MATLAB are shown in (Table 2 and Table 1) while displacement matrix is shown in (Table 3)

**Discussion**

The computing for simple model with 40 GPa elastic modulus and boundary load of 200 MPa shows that our model is deforming as shown in Figure 3, Figure 4 and Figure 5. The displacements computed from MATLAB are much smaller. Comparison of this model with Ansys shows variation due to specific change of boundary conditions discuss in last paragraph.

The variation in load angles provided insights into the directional stiffness and strength of the structure. Higher stiffness values (Young's Modulus) correspond to reduced displacements and strains, indicating a stiffer material response. The load angle had a significant impact on the distribution of strains and stresses within the structure, highlighting areas of potential weakness or failure under different loading scenarios. The comparison of load orientation with displacement shows displacement is maximum when angle is 90 shown in (Table 6)

The analysis assumed linear elastic behaviour, and the results are valid within the limits of this assumption. Comparison is shown in (Table 5) as stiffness increases displacement slightly decreases. However, doesn’t show very large differences in the values.

Variation in results from Ansys and MATLAB can be due to boundary conditions used in Ansys, are fixed boundary constraint (making node 12 displacements 0) while in MATLAB we used two fixed nodes [1, 11]. boundary load in Ansys while in MATLAB we use point loads by reducing boundary load at load points. That significantly alters the calculations. Another reason is the number of elements used in ANSYS are higher give more variations in results. As we know the finer the mesh the accurate the results provided discretization should be done properly. Results from Ansys for simple model are show in (Table 4)

**Conclusion**

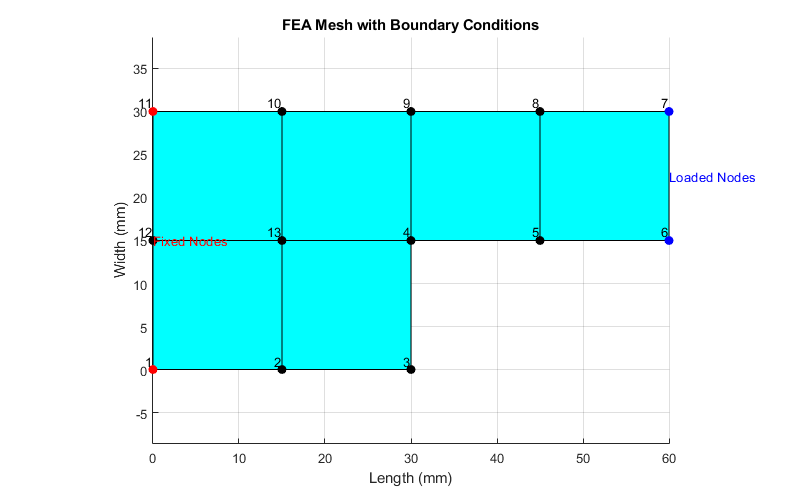
Conclusion drawn from our analysis was behaviour of our geometry due to variating load orientation, as the angle of applied force changes the displacement components changes also increases displacements focusing on loaded nodes. On the other hand increase in stiffness reduces the displacements by slight differences verifying the Stokes law. Moreover, validation of our MATLAB model with ANSYS shows same trend but with slightly higher displacements values in ANSYS. By describing boundary conditions more precisely and matching the meshing will reduce the difference between both results. Moreover, by working more on this model by addressing boundary conditions, we can develop good results for further implications.

**Reference:**

1. Hammad, A., & Moustafa, M. A. J. B. o. E. E. (2021). Numerical analysis of special concentric braced frames using experimentally-validated fatigue and fracture model under short and long duration earthquakes. *19*, 287-316.
2. Jockwer, R., & Dietsch, P. J. E. S. (2018). Review of design approaches and test results on brittle failure modes of connections loaded at an angle to the grain. *171*, 362-372.
3. Lazarus, B. S., Velasco-Hogan, A., Gómez-del Río, T., Meyers, M. A., Jasiuk, I. J. J. o. M. R., & Technology. (2020). A review of impact resistant biological and bioinspired materials and structures. *9*(6), 15705-15738.
4. Meo, M., Rossi, M. J. C. S., & Technology. (2006). Prediction of Young’s modulus of single wall carbon nanotubes by molecular-mechanics based finite element modelling. *66*(11-12), 1597-1605.
5. Sussman, T., Bathe, K.-J. J. C., & Structures. (1987). A finite element formulation for nonlinear incompressible elastic and inelastic analysis. *26*(1-2), 357-409.
6. Zienkiewicz, O. C., & Taylor, R. L. (2005). *The finite element method for solid and structural mechanics*: Elsevier.

**Figures and Tables**

Table : Stress Tensor from Matlab

A screenshot of a calculator

Description automatically generatedA screenshot of a table

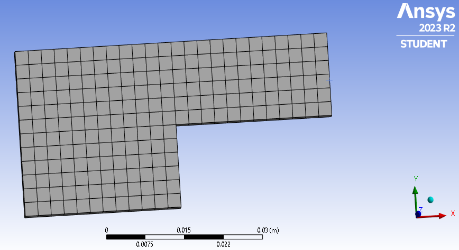
Description automatically generatedTable : Displacements

Table : Strain Tensor from Matlab

Figure : MATLAB problem definition

|  |  |
| --- | --- |
| U1 | 0 |
| V1 | 0 |
| U2 | 1.27361745785848e-08 |
| V2 | -1.12542684834757e-08 |
| U3 | 7.40192961819775e-09 |
| V3 | -4.39831943370922e-08 |
| U4 | 5.55253319486833e-08 |
| V4 | -4.62434953896264e-08 |
| U5 | 9.29346777896021e-08 |
| V5 | -9.81572209244087e-08 |
| U6 | 1.30447791284312e-07 |
| V6 | -1.50593627974582e-07 |
| U7 | 1.82795803666441e-07 |
| V7 | -1.61798944955571e-07 |
| U8 | 1.45282690171731e-07 |
| V8 | -1.09539327241486e-07 |
| U9 | 1.07873344330812e-07 |
| V9 | -5.67570280120094e-08 |
| U10 | 6.29017582036347e-08 |
| V10 | -1.37366482552259e-08 |
| U11 | 0 |
| V11 | 0 |
| U12 | 4.22521799404257e-08 |
| V12 | 3.78768441718017e-09 |
| U13 | 3.89305705150785e-08 |
| V13 | -8.79964902599401e-09 |

A grey square object with a blue background

Description automatically generated with medium confidence

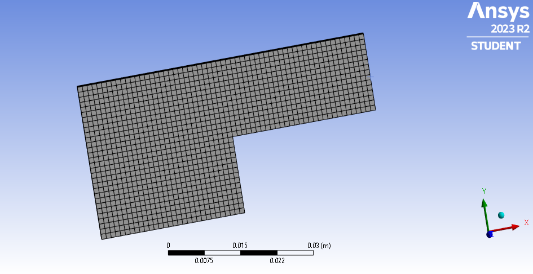
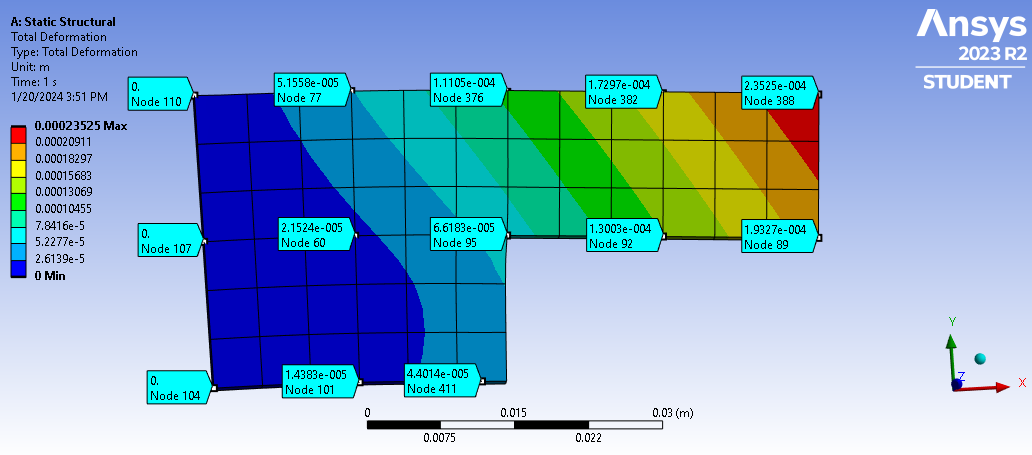


Figure 2: Meshes used in Ansys a)5 mm b) 2.5 mm c) 1 mm



**Left side is fixed boundary making [1,11, 12] nodes fixed (no displacement)**

Figure : Mesh convergence plots using ANSYS a) 5 mm -->Element Size

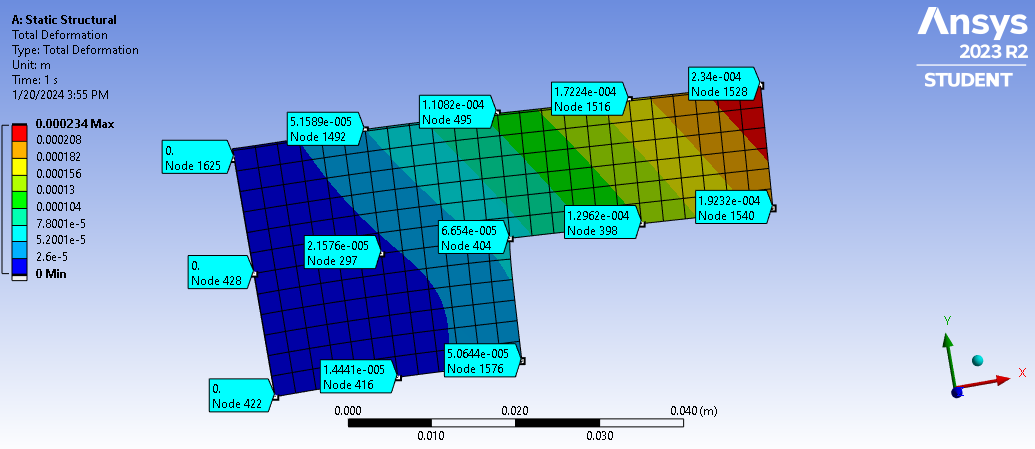


Figure : Mesh convergence plots using ANSYS b) 2.5 mm -->Element Size

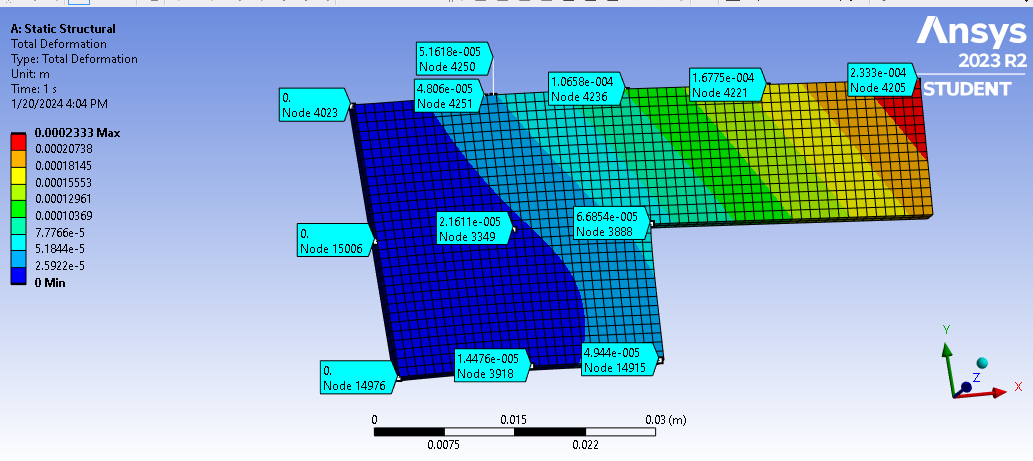


Figure : Mesh convergence plots using ANSYS c) 1 mm -->Element Size

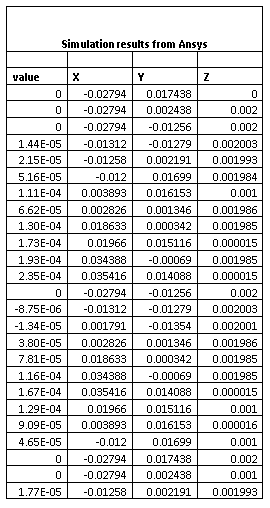
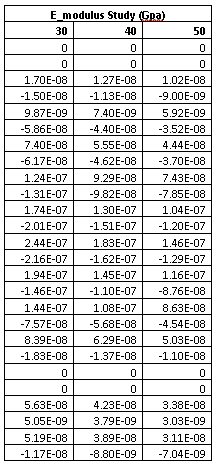


Table : Simulation results from Ansys for E= 40 GPa and Loading is 200 MPa along x-axis

Table : Comparison of different E\_modulus



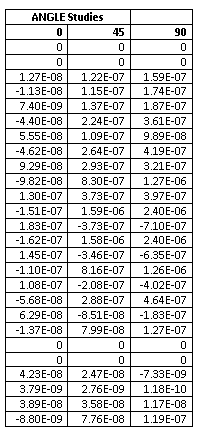
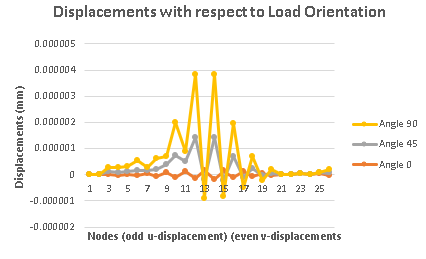
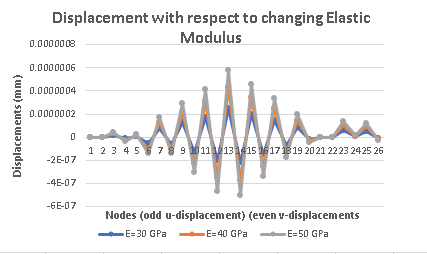


Figure : Displacements with Elastic Modulus Variations

Figure : Displacements with changing Load Orientations

Table : Comparison in displacement over Load Orientations

**APPENDIX**

**MATLAB CODES**

%1)MAIN CODE

% Define geometry, material properties, and mesh

L= 30; % assigning according to your name initials

nodes = [0,0; 15,0; L,0; 30,15; 45,15; 60,15; 60,30; 45,30; L,30; 15,30; 0,30; 0,15; 15,15]; % Node coordinates

elements = [1,2,13,12; 13,10,11,12; 2,3,4,13; 13,4,9,10; 4,5,8,9; 5,6,7,8]; % Element connectivity

E = 40e9; % Young's modulus

nu = 0.3; % Poisson's ratio

thickness = 2; %thickness

numNodes = size(nodes, 1);

dof = 2; % Degrees of freedom per node in 2D problem

% Initialize global stiffness matrix and load vector

K\_global = zeros(numNodes \* dof);

F\_global = zeros(numNodes \* dof, 1);

%elementsNodes = 4;

% Loop over elements to compute and assemble stiffness matrices

for i = 1:6

% Compute element stiffness matrix for plane stress

K\_local = computeStiffnessMatrix(nodes, elements(i,: ), E, nu, thickness);

% Assemble into global stiffness matrix

K\_global = assembleGlobalMatrix(K\_global, K\_local, elements(i, :));

end

% Define fixed and loaded nodes

fixedNodes = [1, 11,]; % Example fixed nodes

loadedNodes = [6, 7]; % Nodes where force is applied

% calculating the force magnitude on each node

% force magnitude= ((Distributive Load)---> pressure \* Total length)/2

forceMagnitude = 1500; % Define the force magnitude

% Apply boundary conditions

[K\_global, F\_global] = applyBoundaryConditions(K\_global, F\_global, fixedNodes, loadedNodes, forceMagnitude);

% Solve for displacements

displacements = K\_global \ F\_global;

% Post-process results

% Calculate strain

strain = calculateStrain(displacements, nodes, elements);

% Calculate stress

stress = calculateStress(strain, E, nu);

% Visualize results (assuming visualizeResults is already implemented)

visualizeResults(nodes, elements, displacements, stress);

*Functions used in above code are given below;*

1. **Local Stiffness Matrix**

function K\_local = computeStiffnessMatrix(nodes, elementNodes, E, nu, thickness)

% Extract the coordinates of the nodes of this element

elementCoords = nodes(elementNodes, :);

% Plane stress constitutive matrix

C = E / (1 - nu^2) \* [1, nu, 0; nu, 1, 0; 0, 0, (1 - nu) / 2];

% Initialize the local stiffness matrix

K\_local = zeros(8, 8); % 4 nodes, 2 DOFs per node

% Gaussian quadrature points and weights (2x2 integration)

gaussPoints = [-1/sqrt(3), 1/sqrt(3)];

gaussWeights = [1, 1];

% Loop over Gaussian quadrature points using indices

for i = 1:length(gaussPoints)

for j = 1:length(gaussPoints)

xi = gaussPoints(i);

eta = gaussPoints(j);

% Calculate Jacobian matrix and its determinant

[Jacobian, detJ] = calculateJacobian(elementCoords, xi, eta);

% Calculate B matrix (strain-displacement matrix)

B = calculateBMatrix(elementCoords, Jacobian, xi, eta);

% Add contribution to the stiffness matrix

K\_local = K\_local + B' \* C \* B \* detJ \* thickness \* gaussWeights(i) \* gaussWeights(j);

end

end

end

1. **Additional Functions in above code are given below:**
2. Jacobian ii) B-Matrix

function B = calculateBMatrix(elementCoords, Jacobian, xi, eta)

% Calculate the derivatives of the shape functions with respect to x and y using the Jacobian

dN\_dxi = [-(1-xi)/4,(1-xi)/4,(1+xi)/4,-(1+xi)/4];

dN\_deta = [-(1-eta)/4,-(1+eta)/4,(1+eta)/4,(1-eta)/4];

% Assuming dN\_dxi and dN\_deta are already defined, and the Jacobian matrix is calculated

invJacobian = inv(Jacobian); % Inverse of the Jacobian matrix

% Initialize arrays to store the derivatives with respect to x and y

dN\_dx = zeros(size(dN\_dxi));

dN\_dy = zeros(size(dN\_deta));

% Calculate the derivatives with respect to x and y for each node

for i = 1:length(dN\_dxi)

dN\_dxi\_eta = [dN\_dxi(i); dN\_deta(i)]; % Column vector of derivatives w.r.t. xi and eta

dN\_dx\_dy = invJacobian \* dN\_dxi\_eta; % Transform to derivatives w.r.t. x and y

dN\_dx(i) = dN\_dx\_dy(1);

dN\_dy(i) = dN\_dx\_dy(2);

end

% Construct the B matrix

B = zeros(3, 8); % 3 rows for εx, εy, γxy; 8 columns for 4 nodes each with 2 DOFs

% Loop through each node to construct the B matrix

for i = 1:4 % 4 nodes for a quadrilateral element

% Node i contributes to columns 2\*i-1 and 2\*i of the B matrix

B(1, 2\*i-1) = dN\_dx(i); % Contribution to εx from node i's x-displacement

B(2, 2\*i) = dN\_dy(i); % Contribution to εy from node i's y-displacement

% For shear strain γxy, contributions come from both x and y displacements

B(3, 2\*i-1) = dN\_dy(i); % Contribution to γxy from node i's x-displacement

B(3, 2\*i) = dN\_dx(i); % Contribution to γxy from node i's y-displacement

end

end

function [Jacobian, detJ] = calculateJacobian(elementCoords, xi, eta)

% Number of nodes per element (4 for a quadrilateral)

numNodes = size(elementCoords, 1);

% Initialize the Jacobian matrix

Jacobian = zeros(2, 2);

% Calculate derivatives of shape functions with respect to xi and eta

% for a 4-node quadrilateral element

% These should be replaced with the actual derivatives of the shape functions

dN\_dxi = [-(1-xi)/4,(1-xi)/4,(1+xi)/4,-(1+xi)/4]; % Placeholder

for derivatives of N with respect to xi

dN\_deta = [-(1-eta)/4,-(1+eta)/4,(1+eta)/4,(1-eta)/4]; % Placeholder

for derivatives of N with respect to eta

% Loop through each node to construct the Jacobian matrix

for i = 1:numNodes

x = elementCoords(i, 1);

y = elementCoords(i, 2);

Jacobian(1, 1) = Jacobian(1, 1) + dN\_dxi(i) \* x;

Jacobian(1, 2) = Jacobian(1, 2) + dN\_dxi(i) \* y;

Jacobian(2, 1) = Jacobian(2, 1) + dN\_deta(i) \* x;

Jacobian(2, 2) = Jacobian(2, 2) + dN\_deta(i) \* y;

end

% Calculate the determinant of the Jacobian

detJ = det(Jacobian);

end

1. **Global Stiffness Matrix**

function K\_global = assembleGlobalMatrix(K\_global, K\_local, elementNodes)

numNodesPerElement = length(elementNodes);

dofPerNode = 2; % Degrees of freedom per node

% Loop over each node in the element

for i = 1:numNodesPerElement

for j = 1:numNodesPerElement

% Global DOF indices for the current pair of nodes

global\_i\_indices = (elementNodes(i) - 1) \* dofPerNode + (1:dofPerNode);

global\_j\_indices = (elementNodes(j) - 1) \* dofPerNode + (1:dofPerNode);

% Local DOF indices

local\_i\_indices = (i - 1) \* dofPerNode + (1:dofPerNode);

local\_j\_indices = (j - 1) \* dofPerNode + (1:dofPerNode);

% Add local stiffness matrix of the element to the global matrix

K\_global(global\_i\_indices, global\_j\_indices) = K\_global(global\_i\_indices, global\_j\_indices) + K\_local(local\_i\_indices, local\_j\_indices);

end

end

end

**3) Apply Boundary Conditions**

function [K\_mod, F\_mod] = applyBoundaryConditions(K\_global, F\_global, fixedNodes, loadedNodes, forceMagnitude)

K\_mod = K\_global;

F\_mod = F\_global;

dof = 2; % Degrees of freedom per node (X and Y)

% Apply Dirichlet boundary conditions (fixed nodes)

for node = fixedNodes

for i = (node-1)\*dof+1 : node\*dof

K\_mod(i, :) = 0; % Set the row to 0

K\_mod(:, i) = 0; % Set the column to 0

K\_mod(i, i) = 1; % Set the diagonal to 1

F\_mod(i) = 0; % Set the force to 0

end

end

% Apply Neumann boundary conditions (forces in X-direction)

for node = loadedNodes

F\_mod((node-1)\*dof + 1) = F\_mod((node-1)\*dof + 1) + forceMagnitude; % Apply force in X-direction

end

end

1. **Strain Tensor**

function strain = calculateStrain(displacements, nodes, elements)

% Initialize strain array

numElements = size(elements, 1);

strain = zeros(numElements, 3); % Assuming 2D strain: εxx, εyy, and γxy

gaussPoints = [-1/sqrt(3), 1/sqrt(3)];

gaussWeights = [1, 1];

for elemIndex = 1:numElements

elementNodes = elements(elemIndex, :);

elementCoords = nodes(elementNodes, :);

elementDisplacements = [];

for nodeIndex = 1:length(elementNodes)

nodeDOFStart = (elementNodes(nodeIndex) - 1) \* 2;

elementDisplacements = [elementDisplacements; displacements(nodeDOFStart + 1); displacements(nodeDOFStart + 2)];

end

% Initialize element strain

elementStrain = zeros(3, 1);

for i = 1:length(gaussPoints)

for j = 1:length(gaussPoints)

xi = gaussPoints(i);

eta = gaussPoints(j);

% Calculate Jacobian matrix for the element at (xi, eta)

[Jacobian, detJ] = calculateJacobian(elementCoords, xi, eta);

% Calculate B matrix (strain-displacement matrix)

B = calculateBMatrix(elementCoords, Jacobian, xi, eta);

% Integrate strain over the element

tempStrain = B \* elementDisplacements; % Matrix multiplication

elementStrain = elementStrain + tempStrain \* detJ \* gaussWeights(i) \* gaussWeights(j); % Scalar multiplication

end

end

% Store the strain for this element

strain(elemIndex, :) = elementStrain; % Strain for each element

end

end

1. **Stress Tensor**

function stress = calculateStress(strain, E, nu)

% Plane stress constitutive matrix

C = E / (1 - nu^2) \* [1, nu, 0; nu, 1, 0; 0, 0, (1 - nu) / 2];

% Calculate stress for each element

numElements = size(strain, 1);

stress = zeros(numElements, 3); % σxx, σyy, and τxy

for i = 1:numElements

stress(i, :) = C \* strain(i, :)';

end

end

1. **Visualizing Results**

function visualizeResults(nodes, elements, displacements, stress)

numNodes = size(nodes, 1);

dof = 2; % Degrees of freedom per node in 2D

% Calculate the deformed coordinates

deformedNodes = nodes + reshape(displacements, dof, numNodes)';

% Plotting the original and deformed mesh

figure;

subplot(1, 2, 1);

plotMesh(nodes, elements, 'Original Mesh');

subplot(1, 2, 2);

plotMesh(deformedNodes, elements, 'Deformed Mesh');

end

function plotMesh(nodes, elements, titleStr)

hold on;

for i = 1:size(elements, 1)

elemNodes = elements(i, :);

X = nodes(elemNodes, 1);

Y = nodes(elemNodes, 2);

fill(X, Y, 'cyan'); % Modify as needed for better visualization

end

title(titleStr);

axis equal;

hold off;

end