AXISYMMETRIC ANALYSIS USING 4-NODED AND 8-NODED RECTANGULAR ELEMENTS

Project Report

Submitted in partial fulfillment of the requirements of the course

ME6611E Finite Element Method
Master of Technology in Machine Design

by

SAREPALLI RAMHARAN

M240549ME

DEVIKA SANTHOSH S

M240360ME

under the guidance

of

Dr. JAYADEEP U. B.

तमसो मा ज्योतिर्गमय

Department of Mechanical Engineering

NATIONAL INSTITUTE OF TECHNOLOGY, CALICUT

April 2025

Certificate

This is to certify that the project report entitled by "**AXISYMMETRIC ANALYSIS USING 4-NODED AND 8-NODED RECTANGULAR ELEMENTS"** submitted by **SAREPALLI RAMHARAN (M240549ME) and DEVIKA SANTHOSH S (M240360ME).** National Institute of Technology Calicut, towards the partial fulfillment of the requirements for the award of the Degree of Master of Technology in Machine Design, is a Bonafide record of the work carried out by him under my supervision and guidance.

Dr. JAYADEEP U.B.
Associate Professor
Department of
Mechanical Engineering

Place : NIT Calicut
Date : 10 April 2025

DECLARATION

We hereby declare that this project report on "**AXISYMMETRIC ANALYSIS USING 4-NODED AND 8-NODED RECTANGULAR ELEMENTS"** is our own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or other institutes of higher learning, except where due acknowledgement has been made in the text.

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to everyone who contributed to the successful completion of this course project on the “AXISYMMETRIC ANALYSIS USING 4-NODED AND 8-NODED RECTANGULAR ELEMENTS”.

First and foremost, we extend our sincere thanks to our supervisor Dr. Jayadeep U. B. whose expert guidance, valuable insights, and constant encouragement were instrumental in shaping this work. Their knowledge and experience in the field of **Finite Element Method** have greatly contributed to the development of this course project.

We are also grateful to the faculty and staff of the Mechanical Engineering National Institute of technology Calicut for providing the necessary resources and facilities, as access to **MATLAB** in cad-lab.

Special thanks to my colleagues and peers, Friends whose constructive feedback and collaboration helped enhance the quality of this project and also for encouragement, and unwavering support during this project. we deeply appreciate the discussions and suggestions that enriched my understanding of the subject.

ABSTRACT

This study presents a MATLAB-based finite element analysis of axisymmetric structures using both 4-noded (linear) and 8-noded (quadratic) elements. The work focuses on modelling a thick-walled cylindrical structure subjected to internal pressure, leveraging axisymmetric assumptions to reduce computational effort while preserving accuracy. The governing equations are derived from elasticity theory for axisymmetric solids and discretized using shape functions corresponding to the chosen element types.

Element stiffness matrices are formulated using Gauss quadrature integration, and the global system is solved under appropriate boundary conditions. The mechanical response, specifically radial, axial and hoop stress distributions, is computed for both element types. A convergence study is performed, and numerical results are validated against analytical solutions to assess accuracy.

The findings show that while 4-noded elements offer computational efficiency, 8-noded elements provide enhanced precision in capturing stress gradients near boundaries. This work confirms the effectiveness of MATLAB for simulating axisymmetric problems and highlights the trade-offs between element order and solution accuracy, emphasizing the importance of element selection in finite element modelling of pressure vessels and similar structures.

CONTENTS

List of symbols

ε	strain
ε_r	Radial strain
ε_θ	Hoop strain
ε_z	Axial strain
γ_{rz}	Shear strain
σ	Stress
σ_r	Radial stress
σ_θ	Hoop stress
σ_z	Axial stress
τ_{rz}	Shear stress
r_i	Internal radius of cylinder
r_o	External radius of cylinder
B	strain displacement matrix
D	Constitutive matrix
E	Youngs modulus
K	Element stiffness matrix
L	Length of the cylinder
Ni	Shape function

List of figures

INTRODUCTION

1.1 INTRODUCTION

Axisymmetric finite element analysis is a powerful technique used to solve problems involving structures that are symmetric about a central axis. This approach is commonly applied in mechanical and structural engineering for components such as pressure vessels, pipes, and rotating machinery, where the geometry, loading, and boundary conditions remain constant along the circumferential direction. By simplifying a 3D problem into a 2D representation in the *r*–*z* plane, axisymmetric analysis significantly reduces computational cost while maintaining solution accuracy.

In this study, the axisymmetric formulation is implemented using both 4-noded (bilinear) and 8-noded (quadratic) finite elements. These elements are used to discretize the geometry and solve for stress, strain, and displacement under internal pressure loading. MATLAB is employed as the computational platform, allowing for the development of customized FEM code to explore the behaviour of cylindrical structures.

1.2 Assumptions of Axisymmetric Analysis

- The geometry, loading, and boundary conditions are rotationally symmetric.
- Material is isotropic and behaves linearly elastically.
- Small deformation theory is used.
- Out-of-plane shear and displacement components are zero.
- Analysis is conducted in the 2D *r*–*z* plane, with circumferential behavior implied.

1.3 PROBLEM STATEMENT

This MATLAB code aims to analyze a thick-walled cylindrical shell subjected to internal pressure using the Finite Element Method (FEM) under axisymmetric conditions. The objective is to compute the displacement and stress distribution within the cylinder using both 4-noded and 8-noded quadrilateral finite elements.

Dimensions:

1. Internal radius of the cylinder = 30
2. External radius of the cylinder = 50
3. Length of the cylinder = 40
4. Internal pressure = 40

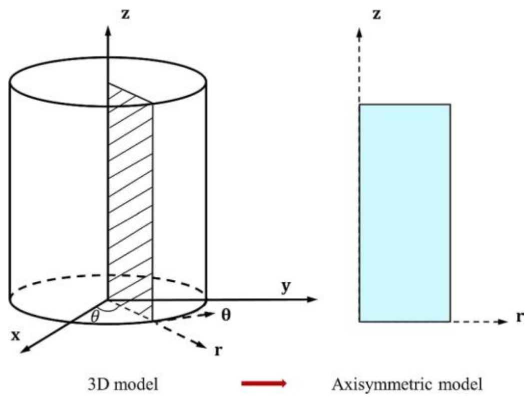

Fig. 1.1 Thick cylinder schematic

Properties Of Materials:

Young's Modulus:

$E_{steel} = 2E5$

Assume Poisson's Ratio:

$\vartheta_{steel} = 0.3$

METHODOLGY

2.1 MATHEMATICAL MODEL

2.1.1 Governing Differential Equation

When the elasticity problem degenerates from three dimension to axisymmetric, two shearing stress components vanish. These vanishing components due to symmetry are $\tau_{r\theta}$ and $\tau_{\theta z}$ coordinate system where r is the radial direction, Θ is the circumferential direction, and z is the axial direction. Hence, the remaining stress components are

$$\{\sigma\} = \{\sigma_r \quad \sigma_\theta \quad \sigma_z \quad \tau_{rz}\}$$

And the strains are

$$\{\varepsilon\} = \{\varepsilon_r \quad \varepsilon_\theta \quad \varepsilon_z \quad \gamma_{rz}\}$$

The material property matrix $[D]$ for the axisymmetric problem is

$$[D] = \frac{E}{(1+\vartheta)(1-2\vartheta)} * \begin{bmatrix} 1-\vartheta & \vartheta & \vartheta & 0 \\ \vartheta & 1-\vartheta & \vartheta & 0 \\ \vartheta & \vartheta & 1-\vartheta & 0 \\ 0 & 0 & 0 & \frac{1-2\vartheta}{2} \end{bmatrix} \tag{2.1}$$

The kinematic equation is

$$\text{Strain vector: } \{\varepsilon\} = \begin{Bmatrix} \varepsilon_r \\ \varepsilon_\theta \\ \varepsilon_z \\ \gamma_{rz} \end{Bmatrix} = \begin{Bmatrix} \frac{\partial u}{\partial r} \\ \frac{u}{r} \\ \frac{\partial u}{\partial z} \\ \frac{\partial u}{\partial z} + \frac{\partial w}{\partial r} \end{Bmatrix} \tag{2.2}$$

The stress vector is

$$\text{Stress vector: } \sigma = \begin{Bmatrix} \sigma_r \\ \sigma_\theta \\ \sigma_z \\ \tau_{rz} \end{Bmatrix} \tag{2.3}$$

2.1.1 Equilibrium Equations in cylindrical coordinates

- $\frac{\partial \sigma_r}{\partial r} + \frac{1}{r} * \frac{\partial \tau_{r\theta}}{\partial r} + \frac{\partial \tau_{rz}}{\partial z} + \frac{1}{r} * (\sigma_r - \sigma_\theta) + F_r = 0$
- $\frac{\partial \tau_{r\theta}}{\partial r} + \frac{1}{r} * \frac{\partial \sigma_\theta}{\partial \theta} + \frac{\partial \tau_{\theta z}}{\partial z} + \frac{2}{r} * \tau_{r\theta} + F_\theta = 0$
- $\frac{\partial \tau_{rz}}{\partial r} + \frac{1}{r} * \frac{\partial \tau_{\theta z}}{\partial \theta} + \frac{\partial \sigma_z}{\partial z} + \frac{1}{r} * \tau_{rz} + F_z = 0$

The terms $\tau_{r\theta}$, $\tau_{\theta z}$ will vanish due to symmetry of the problem

- $\frac{\partial \sigma_r}{\partial r} + \frac{\partial \tau_{rz}}{\partial z} + \frac{1}{r} * (\sigma_r - \sigma_\theta) + F_r = 0$
- $\frac{\partial \tau_{rz}}{\partial r} + \frac{\partial \sigma_z}{\partial z} + \frac{1}{r} * \tau_{rz} + F_z = 0$

2.1.2 AXISYMMETRIC SOLID FORMULATION

$K^e = \int B^T D B \, dV$

Where D is the constitutive matrix,

- It gives the relation between stress and strain.
- $$B = \begin{bmatrix} \frac{\partial N1}{\partial r} & 0 & \frac{\partial N2}{\partial r} & 0 & \frac{\partial N3}{\partial r} & 0 & \frac{\partial N4}{\partial r} & 0 \\ 0 & \frac{\partial N1}{\partial z} & 0 & \frac{\partial N2}{\partial z} & 0 & \frac{\partial N3}{\partial z} & 0 & \frac{\partial N4}{\partial z} \\ \frac{N1}{r} & 0 & \frac{N2}{r} & 0 & \frac{N3}{r} & 0 & \frac{N4}{r} & 0 \\ \frac{\partial N1}{\partial z} & \frac{\partial N1}{\partial r} & \frac{\partial N2}{\partial z} & \frac{\partial N2}{\partial r} & \frac{\partial N3}{\partial z} & \frac{\partial N3}{\partial r} & \frac{\partial N4}{\partial z} & \frac{\partial N4}{\partial r} \end{bmatrix} \qquad (2.4)$$
- The strain-displacement matrix B relates displacements to strains in **axisymmetric** conditions
- Since the term $\left[\frac{Ni}{r}\right]$ in B matrix the matrix is not a constant matrix like the plane stress / plane strain case as a result the integration needs to be undertaken. One simple approximation for the integration is to evaluate B

matrix at the centroid of the element i.e.,

- We calculate the $\vec{r} = \frac{ri+r0}{2}$; $\vec{z} = \frac{zi+z0}{2}$ and find B matrix at the centroid.
- But we generally find the r at gauss points in the FEM

  $K^e = \int B^T DB\, dV$
- $dv = 2 * \pi * r * dr * dz$ accounts for rotational symmetry.

  $K^e = 2\pi \iint r\ [B^T][D][B] drdz$
- Here r is the mean radius of the element.

matrix then,

The displacement field is obtained by solving the system of linear equations

$$\mathrm{K\,U} = \mathrm{F}$$

The force vector is given by

$F_t = \int N^T * t . 2 * \pi * r\, d\Gamma$

Here t is the traction vector.

2.1.3 ISO PARAMETRIC FORMULATION

2.1.3.1 Bilinear Quadrilateral Element

Each bilinear element has four nodes with two in plane degrees of freedom at each node

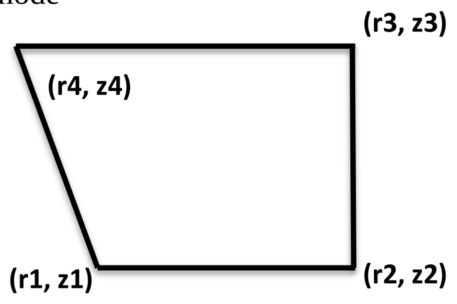

Fig 2.1 4-node Global coordinate systems

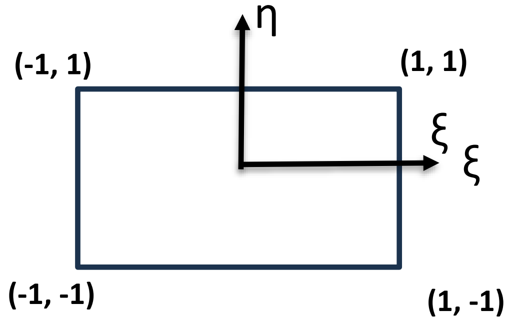

Fig 2.2 4-node natural coordinate system

Where, ξ and η range from -1 to 1.

The element is mapped to a rectangle through the use of the natural coordinates through the use of the natural coordinates ξ and η.

The shape functions are used to determine the geometry

- r= $\sum_{i=1}^{4} N_i\, r_i = N_1 * r_1 + N_2 * r_2 + N_3 * r_3 + N_4 * r_4$
- z= $\sum_{i=1}^{4} N_i\, z_i = N_1 * z_1 + N_2 * z_2 + N_3 * z_3 + N_4 * z_4$

Using the property of Iso – Parametric element the displacement fields are

- u= $\sum_{i=1}^{4} N_i\, u_i = N_1 * u_1 + N_2 * u_2 + N_3 * u_3 + N_4 * u_4$
- w= $\sum_{i=1}^{4} N_i\, w_i = N_1 * w_1 + N_2 * w_2 + N_3 * w_3 + N_4 * w_4$

The shape functions are:

N1 = $\frac{1}{4}$ (1−ξ)(1−η),

N2 = $\frac{1}{4}$ (1+ξ)(1−η),

N3 = $\frac{1}{4}$ (1+ξ)(1+η)

N4 = $\frac{1}{4}$ (1--ξ)(1+η)

For this element the matrix [B] is given by:

[B] = $\frac{1}{|J|}$ $[B_1\ B_2\ B_3\ B_4]$

Where $[B_i]$ is given by:

$$B = \begin{bmatrix} \frac{\partial N1}{\partial r} & 0 & \frac{\partial N2}{\partial r} & 0 & \frac{\partial N3}{\partial r} & 0 & \frac{\partial N4}{\partial r} & 0 \\ 0 & \frac{\partial N1}{\partial z} & 0 & \frac{\partial N2}{\partial z} & 0 & \frac{\partial N3}{\partial z} & 0 & \frac{\partial N4}{\partial z} \\ \frac{N1}{r} & 0 & \frac{N2}{r} & 0 & \frac{N3}{r} & 0 & \frac{N4}{r} & 0 \\ \frac{\partial N1}{\partial z} & \frac{\partial N1}{\partial r} & \frac{\partial N2}{\partial z} & \frac{\partial N2}{\partial r} & \frac{\partial N3}{\partial z} & \frac{\partial N3}{\partial r} & \frac{\partial N4}{\partial z} & \frac{\partial N4}{\partial r} \end{bmatrix}$$

- Since $\frac{\partial N1}{\partial r}$ cannot be calculated directly.
- We use chain rule to evaluate the terms.

The chain rule is as follows:

- $\frac{\partial N1}{\partial r} = \frac{\partial N1}{\partial \xi} * \frac{\partial \xi}{\partial r} + \frac{\partial N1}{\partial \eta} * \frac{\partial \eta}{\partial r}$
- $\frac{\partial N1}{\partial z} = \frac{\partial N1}{\partial \xi} * \frac{\partial \xi}{\partial z} + \frac{\partial N1}{\partial \eta} * \frac{\partial \eta}{\partial z}$
- $\begin{bmatrix} \frac{\partial N1}{\partial r} \\ \frac{\partial N1}{\partial z} \end{bmatrix} = \begin{bmatrix} \frac{\partial \xi}{\partial r} & \frac{\partial \eta}{\partial r} \\ \frac{\partial \xi}{\partial z} & \frac{\partial \eta}{\partial z} \end{bmatrix} \begin{bmatrix} \frac{\partial N1}{\partial \xi} \\ \frac{\partial N1}{\partial \eta} \end{bmatrix} = [J]^{-1} * \begin{bmatrix} \frac{\partial N1}{\partial \xi} \\ \frac{\partial N1}{\partial \eta} \end{bmatrix}$
- The Jacobian matrix:

$$J = \begin{bmatrix} \frac{\partial r}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial r}{\partial \eta} & \frac{\partial z}{\partial \eta} \end{bmatrix}$$

- The Jacobian matrix can be evaluated as

$$J = \begin{bmatrix} \frac{\partial r}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial r}{\partial \eta} & \frac{\partial z}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \sum \frac{\partial Ni}{\partial \xi} * ri & \sum \frac{\partial Ni}{\partial \xi} * zi \\ \sum \frac{\partial Ni}{\partial \eta} * ri & \sum \frac{\partial Ni}{\partial \eta} * zi \end{bmatrix} \qquad (2.5)$$

- Since Ni is a function of ξ and η it can be differentiated.

The element stiffness matrix for the bilinear quadrilateral element is written in terms of double integration as follows:

$$[K] = t \int_{-1}^{1} \int_{-1}^{1} [B]^T \, [D][B]|J| d\xi \, d\eta$$

Strain formulation:

$\{\varepsilon\} = [B][u]$ (2.6)

Stress formulation:

$\{\sigma\} = [D][\varepsilon]$ (2.7)

2.1.3.2 Gauss point integration:

Gauss point integration is a numerical method used to accurately evaluate integrals in finite element analysis, especially for stiffness matrix and force vector calculations. For 4-node quadrilateral elements, a 2x2 Gauss quadrature is commonly applied over the iso-parametric domain.

The gauss points are

Gauss point 1	$(-\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}})$
Gauss point 2	$(\frac{1}{\sqrt{3}}, -\frac{1}{\sqrt{3}})$
Gauss point 3	$(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}})$

- The associated weights are (wi, wj) = (1,1)

The stiffness matrix with gauss integration becomes

$K^e = \sum_{i=1}^{n} \sum_{j=1}^{n} [B^T][D][B] * \det(J) * wi * wj * 2 * \pi * r\_gauss$

2.1.4.3 Quadratic Quadrilateral Element

Each quadratic element has eight nodes with two in plane degrees of freedom at each node

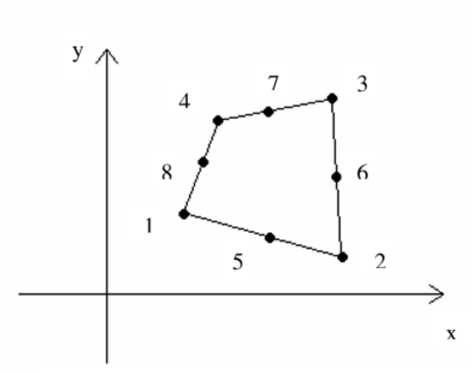

Fig 2.3 8-node quadrilateral in Global coordinate system

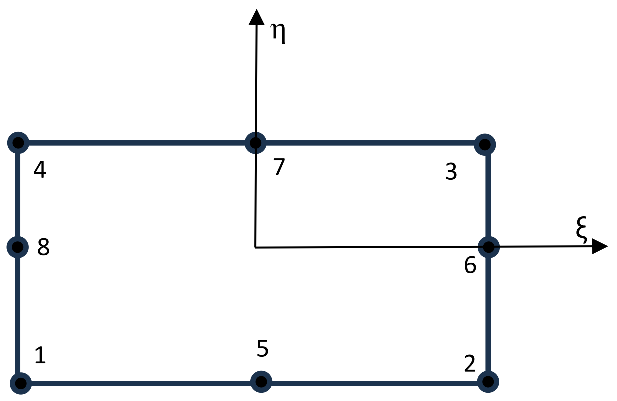

Fig 2.4 8-node quadrilateral in natural coordinate system

Where, ξ and η range from -1 to 1.

The element is mapped to a rectangle through the use of the natural coordinates through the use of the natural coordinates ξ and η.

The shape functions are used to determine the geometry

r= $\sum_{i=1}^{8} N_i \, r_i$

z= $\sum_{i=1}^{4} N_i \, z_i$

Using the property of Iso – Parametric element the displacement fields are

$u = \sum_{i=1}^{8} N_i u_i$

$w = \sum_{i=1}^{8} N_i w_i$

The shape functions are:

$N1 = \frac{1}{4}(1-\xi)(1-\eta)(-\xi - \eta - 1)$

$N2 = \frac{1}{4}(1+\xi)(1-\eta)(\xi - \eta - 1)$

$N3 = \frac{1}{4}(1+\xi)(1+\eta)(\xi + \eta - 1)$

$N4 = \frac{1}{4}(1--\xi)(1+\eta)(-\xi + \eta - 1)$

$N5 = \frac{1}{2}(1--\xi^2)(1-\eta)$

$N6 = \frac{1}{2}(1+\xi)(1-\eta^2)$

$N7 = \frac{1}{2}(1-\xi^2)(1+\eta)$

$N8 = \frac{1}{2}(1-\xi)(1-\eta^2)$

For this element the matrix [B] is given by:

$[B] = \frac{1}{|J|}[B_1\ B_2\ B_3\ B_4]$

Where $[B_i]$ is given by:

The B-matrix has 4 rows and 16 columns

2.1.4.4 Gauss point integration:

Gauss point integration is a reliable numerical technique used to compute integrals in finite element analysis, ensuring accurate evaluation of stiffness matrices and force vectors. For 8-node quadrilateral elements, a 3x3 Gauss quadrature is typically used over the iso-parametric domain to capture the higher-order variation of shape functions.

- The gauss points are : $\left\{-\sqrt{\frac{3}{5}}\ ;\ 0\ ;\ \sqrt{\frac{3}{5}}\right\}$
- The associated weights are: $\left\{\frac{5}{9}\ ;\ \frac{8}{9}\ ;\ \frac{5}{9}\right\}$

The need of using 3 gauss points is in 2D is it can solve the polynomial of up to degree 4th order. Whereas it can solve up to 5th order in 1D

3.MATLAB CODE STRUCTURE

3.1 Flow chart for 4-node quadrilateral.

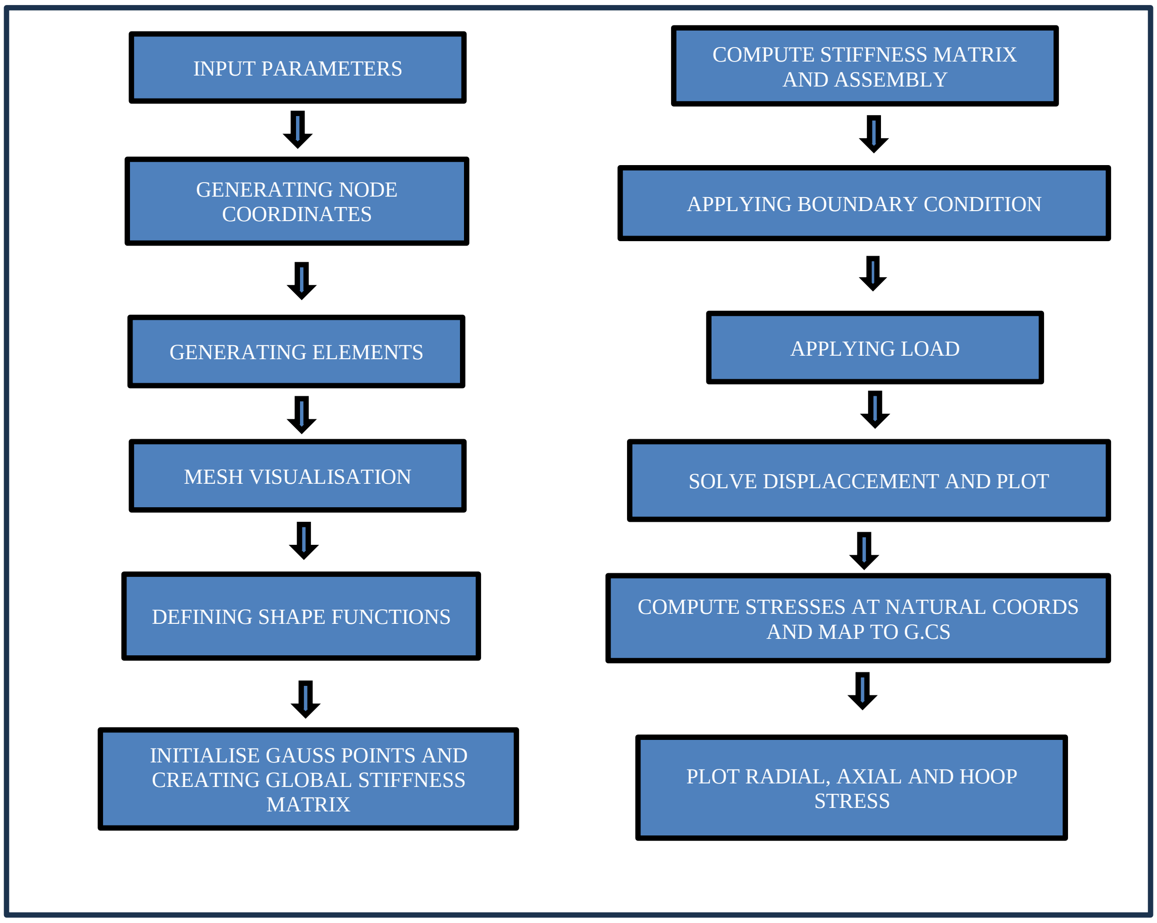
INPUT PARAMETERS
GENERATING NODE COORDINATES
GENERATING ELEMENTS
MESH VISUALISATION
DEFINING SHAPE FUNCTIONS
INITIALISE GAUSS POINTS AND CREATING GLOBAL STIFFNESS MATRIX
COMPUTE STIFFNESS MATRIX AND ASSEMBLY
APPLYING BOUNDARY CONDITION
APPLYING LOAD
SOLVE DISPLACCEMENT AND PLOT
COMPUTE STRESSES AT NATURAL COORDS AND MAP TO G.CS
PLOT RADIAL, AXIAL AND HOOP STRESS

Fig 3.1 Code work flow for 4 – node element

Description: The code workflow for the 4-node quadrilateral element begins with defining input parameters, generating node coordinates and elements, and visualizing the mesh. Shape functions are defined, and Gauss points are initialized to compute the global stiffness matrix. After assembling the stiffness matrix, boundary conditions and loads are applied, and the displacement field is solved and plotted. Finally, stresses are calculated at Gauss points, mapped to global coordinates, and radial, axial, and hoop stresses are plotted for analysis.

3.2 Flow chart for 8-node quadrilateral.

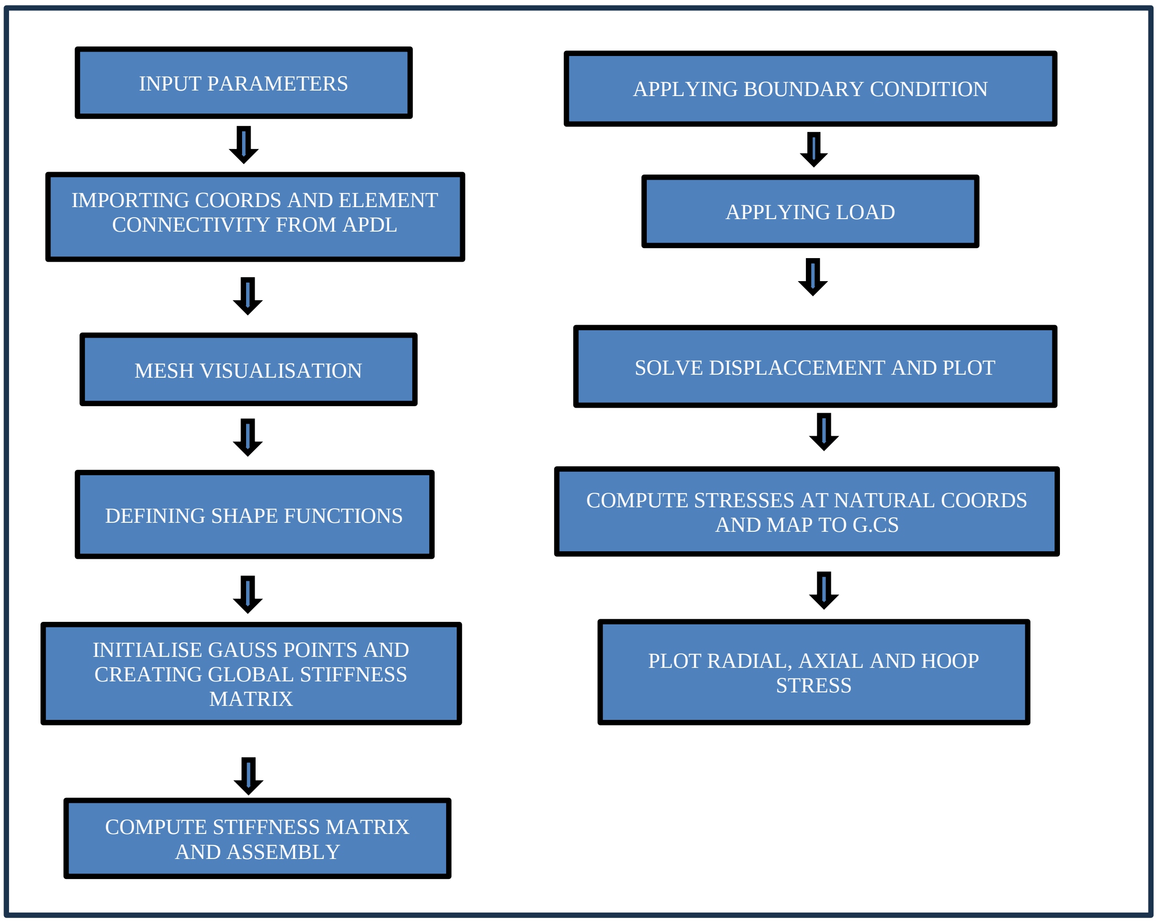
INPUT PARAMETERS
IMPORTING COORDS AND ELEMENT CONNECTIVITY FROM APDL
MESH VISUALISATION
DEFINING SHAPE FUNCTIONS
INITIALISE GAUSS POINTS AND CREATING GLOBAL STIFFNESS MATRIX
COMPUTE STIFFNESS MATRIX AND ASSEMBLY
APPLYING BOUNDARY CONDITION
APPLYING LOAD
SOLVE DISPLACCEMENT AND PLOT
COMPUTE STRESSES AT NATURAL COORDS AND MAP TO G.CS
PLOT RADIAL, AXIAL AND HOOP STRESS

Fig 3.2 Code work flow for 8 – node element

Description: This workflow integrates APDL data into a custom MATLAB code by importing node coordinates and element connectivity. After visualizing the mesh and defining shape functions, Gauss points are initialized and the global stiffness matrix is assembled. Boundary conditions and external loads are applied, followed by solving the displacement field and plotting the results. Finally, stresses are computed at natural coordinates, mapped to the global system, and radial, axial, and hoop stress plots are generated.

4. RESULTS

4.1 COMPARISON OF APDL AND MATLAB RESULTS FOR 4-NODE RECTANULAR ELEMENTS

PARAMETER	APDL	MATLAB
MAXIMUM RADIAL DISPLACEMENT (Ur)	0.0139	0.0139
MINIMUM RADIAL DISPLACEMENT (Ur)	0.002	0.0102
MAXIMUM AXIAL DISPLACEMENT (u_Z)	0	0
MINIMUM AXIAL DISPLACEMENT (u_Z)	0	0
MAXIMUM RADIAL STRESS (σ_r)	-0.614	-0.9210
MINIMUM RADIAL STRESSS (σ_r)	-37.33	-35.9913
MAXIMUM AXIAL STRESS (σ_Z)	13.81	13.5394
MINIMUM AXIAL STRESS (σ_Z)	12.22	13.5011
MAXIMUM HOOP STRESS (σ_θ)	83.62	81.1227
MINIMUM HOOP STRESS (σ_θ)	45.29	45.9246

4.1.1 COMPARISON OF APDL AND MATLAB CONTOURS FOR 4-NODE RECTANULAR ELEMENTS OF RADIAL DISPLACEMENT

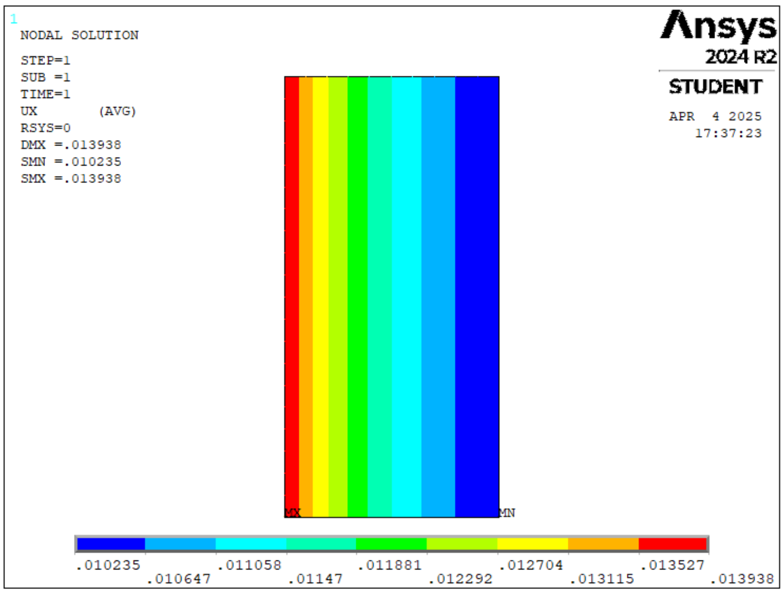

Fig 4.1 *Radial displacement contour from APDL using 4-node rectangular element*

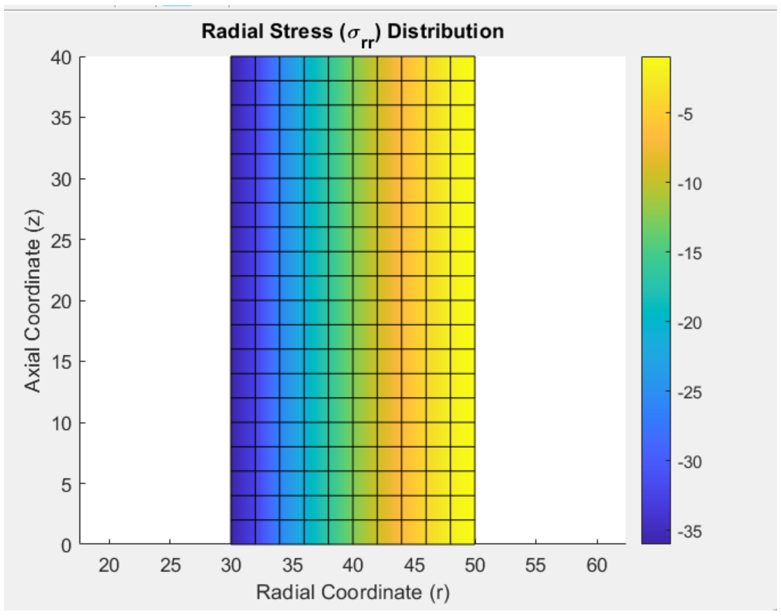

Fig 4.2 *Radial displacement contour from MATLAB using 4-node rectangular element*

4.1.2 COMPARISON OF APDL AND MATLAB CONTOURS FOR 4-NODE RECTANGULAR ELEMENTS OF RADIAL STRESS

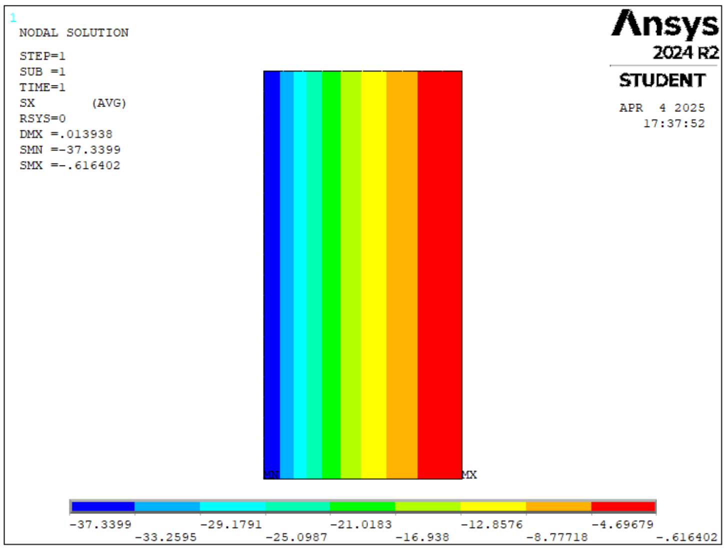

Fig 4.3 *Radial stress contour from APDL using 4-node rectangular element*

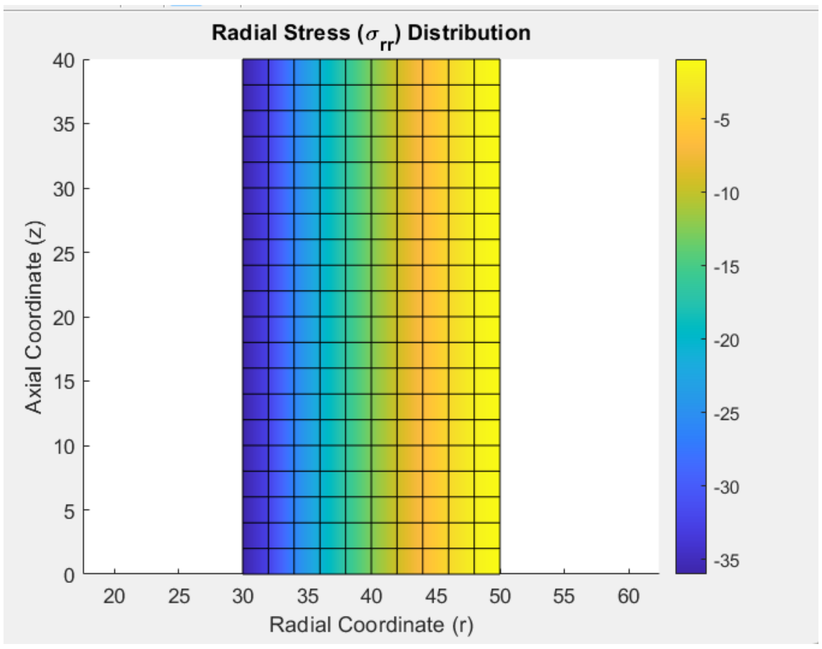

Fig 4.4 *Radial stress contour from MATLAB using 4-node rectangular element*

4.1.3 COMPARISON OF APDL AND MATLAB CONTOURS FOR 4-NODE RECTANGULAR ELEMENTS OF AXIAL STRESS

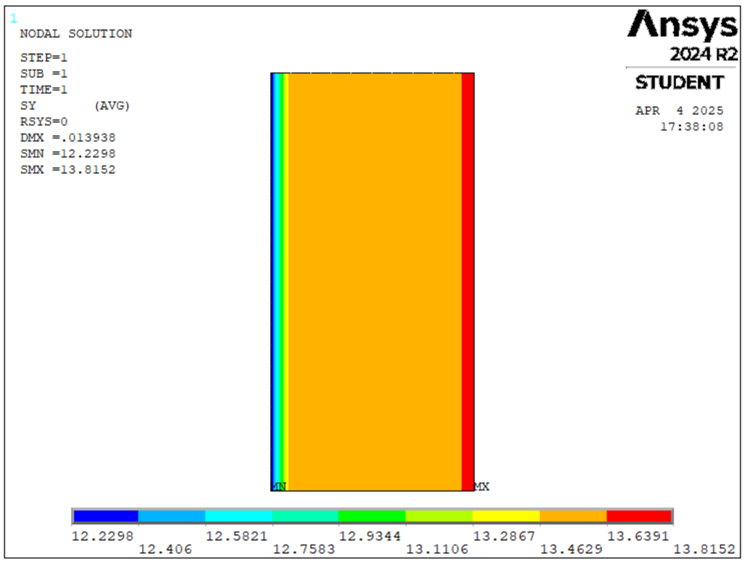

Fig 4.5 *Axial stress contour from APDL using 4-node rectangular element*

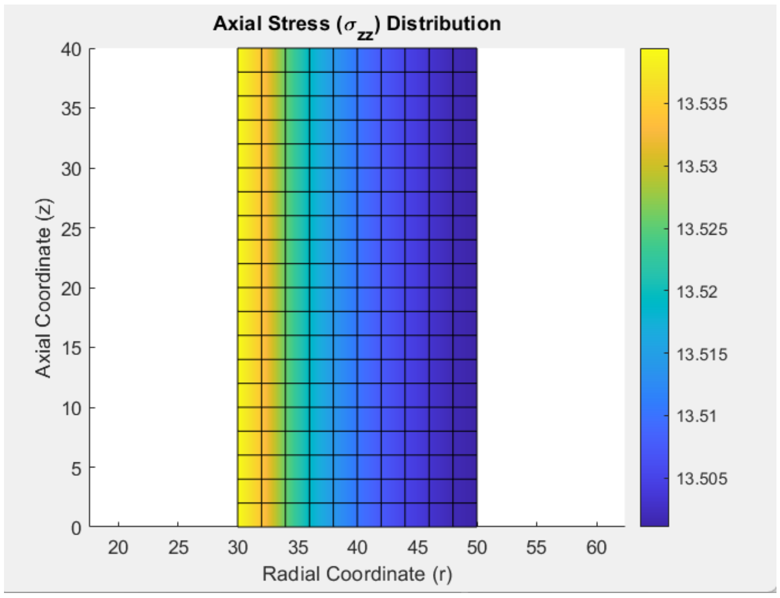

Fig 4.6 *Axial stress contour from MATLAB using 4-node rectangular element*

4.1.4 COMPARISON OF APDL AND MATLAB CONTOURS FOR 4-NODE RECTANGULAR ELEMENTS OF HOOP STRESS

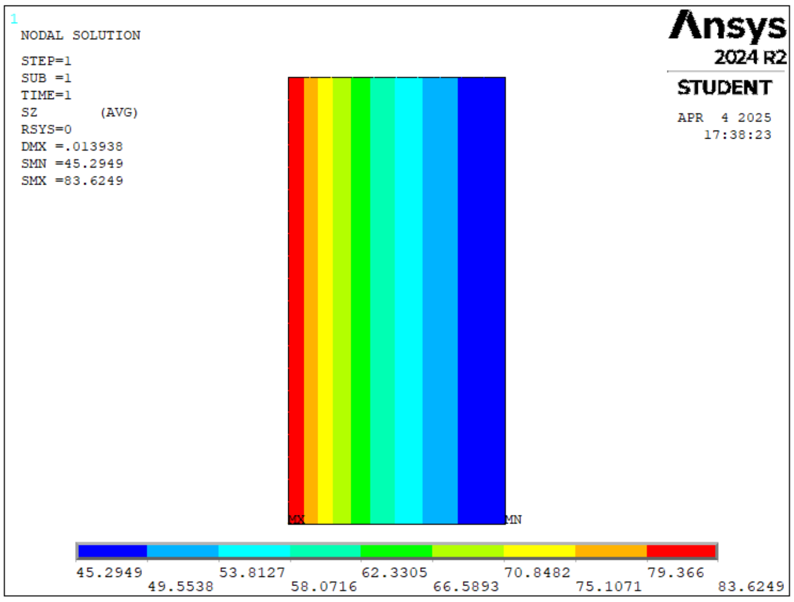

Fig 4.7 *Hoop stress contour from APDL using 4-node rectangular element*

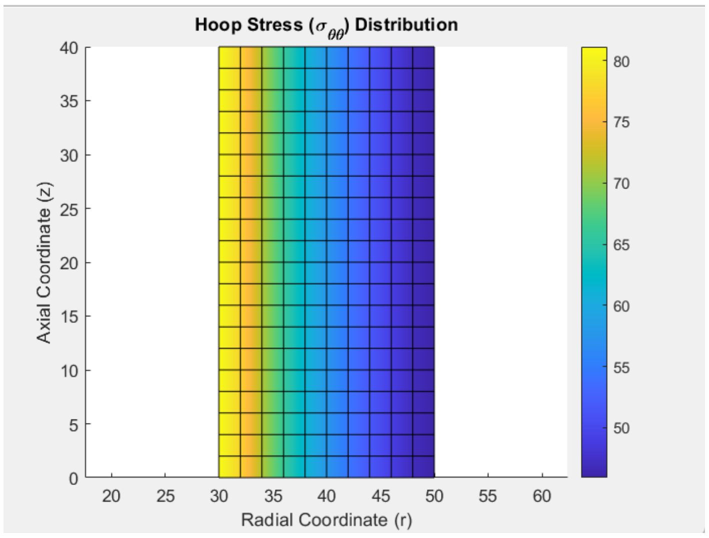

Fig 4.8 *Hoop stress contour from MATLAB using 4-node rectangular element*

4.2 COMPARISON OF APDL AND MATLAB RESULTS FOR 8-NODE RECTANULAR ELEMENTS

PARAMETER	APDL	MATLAB
MAXIMUM RADIAL DISPLACEMENT (Ur)	0.0139	0.0139
MINIMUM RADIAL DISPLACEMENT (Ur)	0.0102	0.0102
MAXIMUM AXIAL DISPLACEMENT (u_Z)	0	0
MINIMUM AXIAL DISPLACEMENT (u_Z)	0	0
MAXIMUM RADIAL STRESS (σ_r)	0.019	0.0335
MINIMUM RADIAL STRESSS (σ_Z)	-39.874	-39.7795
MAXIMUM AXIAL STRESS (σ_Z)	13.5	13.59
MINIMUM AXIAL STRESS (σ_Z)	13.5	13.45
MAXIMUM HOOP STRESS (σ_θ)	84.87	85.09
MINIMUM HOOP STRESS (σ_θ)	44.98	45.01

4.2.1 COMPARISON OF APDL AND MATLAB CONTOURS FOR 8-NODE RECTANGULAR ELEMENTS OF RADIAL DISPLACEMENT

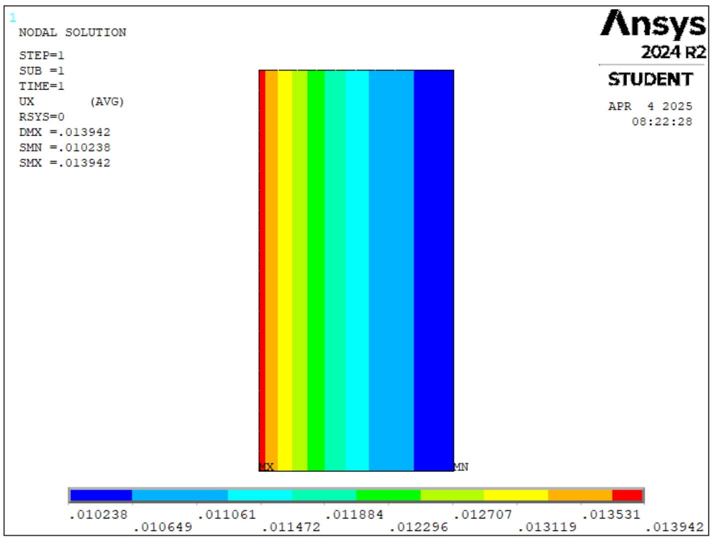

Fig 4.9 *Radial displacement contour from APDL using 8-node rectangular element*

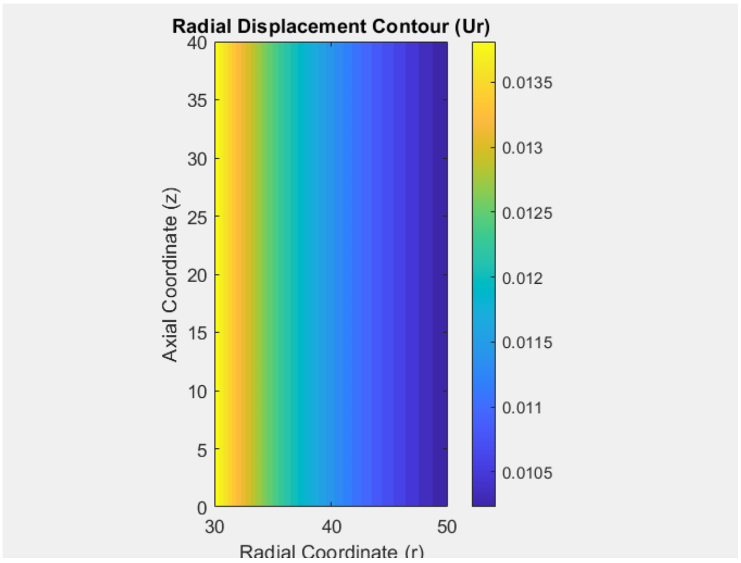

Fig 4.10 *Radial displacement contour from MATLAB using 8-node rectangular element*

4.2.2 COMPARISON OF APDL AND MATLAB CONTOURS FOR 8-NODE RECTANGULAR ELEMENTS OF RADIAL STRESS

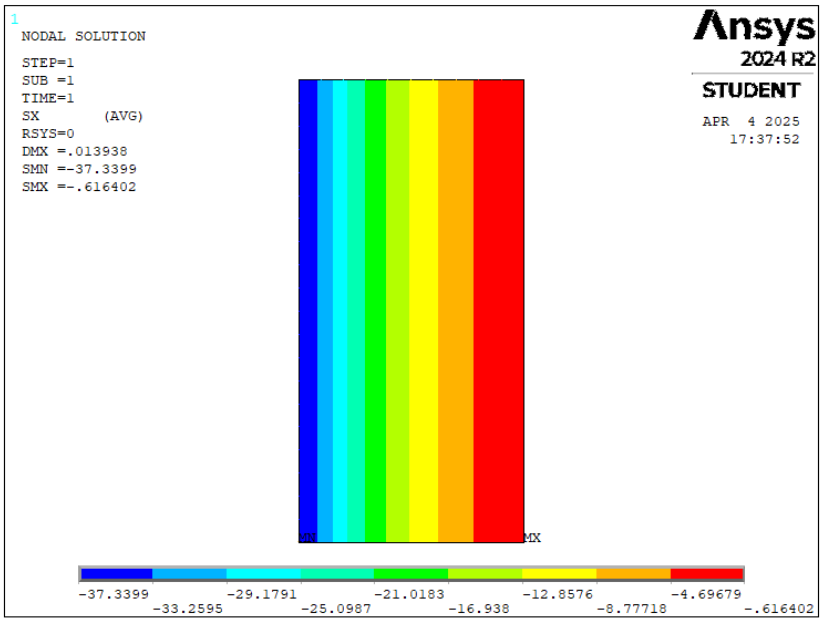

Fig 4.11 *Radial stress contour from APDL using 8-node rectangular element*

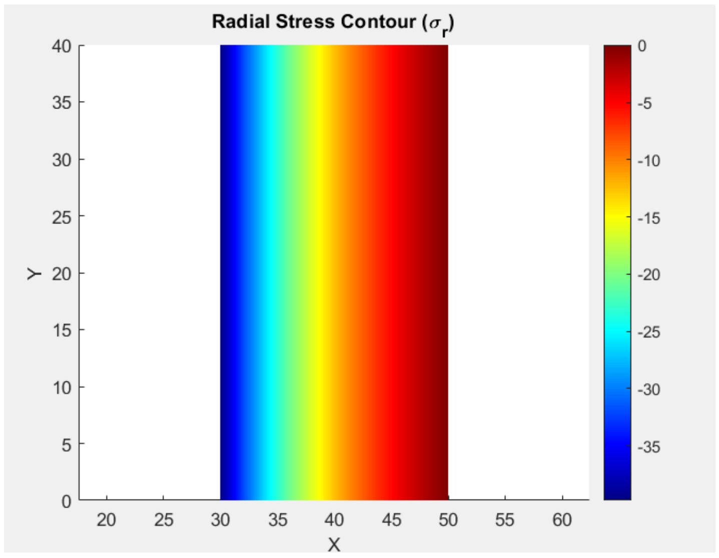

Fig 4.12 *Radial stress contour from APDL using 8-node rectangular element*

4.2.3 COMPARISON OF APDL AND MATLAB CONTOURS FOR 8-NODE RECTANGULAR ELEMENTS OF AXIAL STRESS

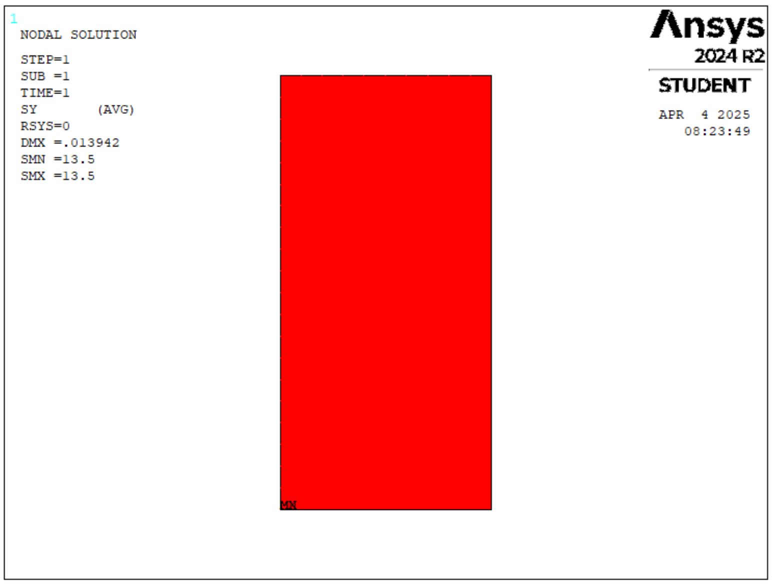

Fig 4.13 *Axial stress contour from APDL using 8-node rectangular element*

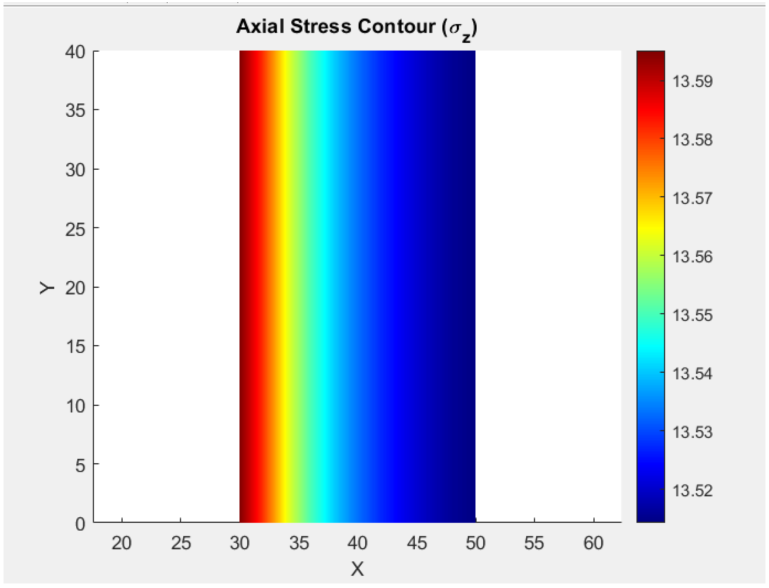

Fig 4.14 *Axial stress contour from MATLAB using 8-node rectangular element*

4.2.4 COMPARISON OF APDL AND MATLAB CONTOURS FOR 8-NODE RECTANGULAR ELEMENTS OF HOOP STRESS

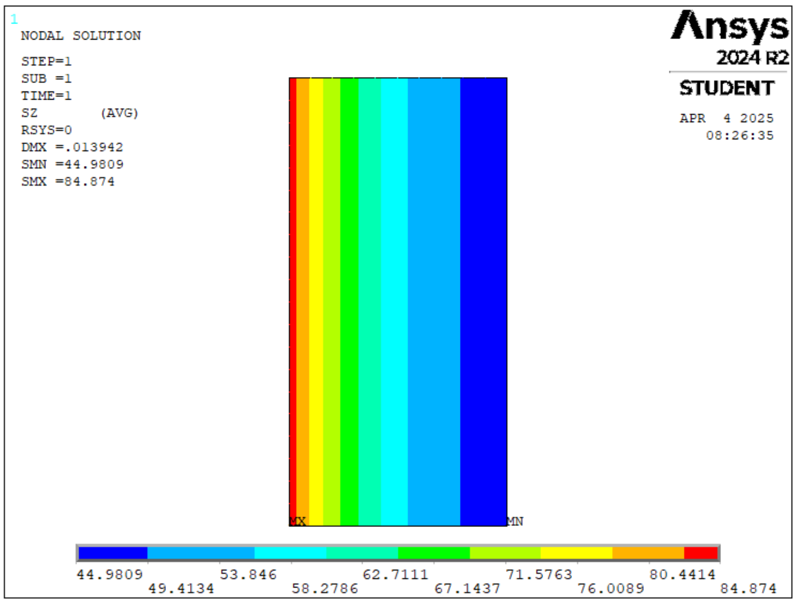

Fig 4.15 *Axial stress contour from MATLAB using 8-node rectangular element*

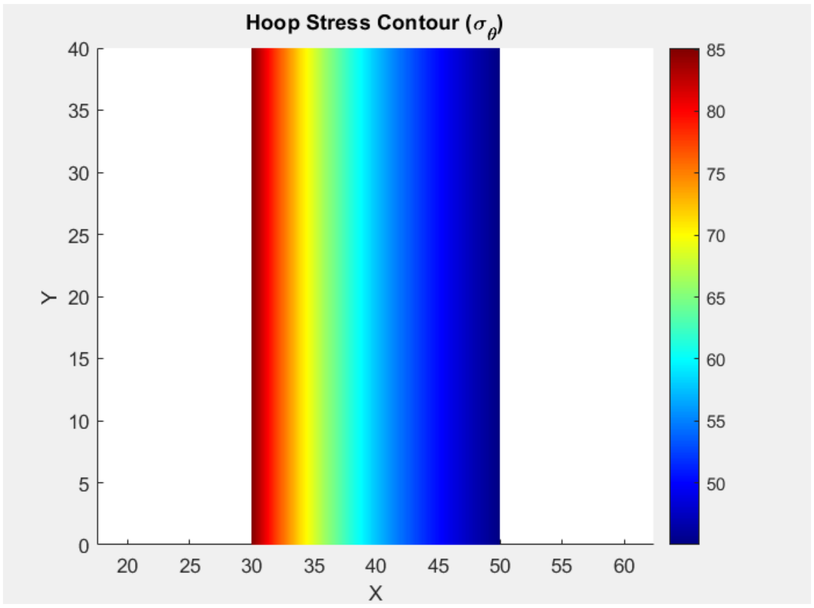

Fig 4.16 *Axial stress contour from MATLAB using 8-node rectangular element*

5. CONCLUSION

The axisymmetric analysis of a thick cylinder under internal pressure was successfully performed using both 4-noded and 8-noded quadrilateral elements. The study demonstrated the formulation and implementation of finite element methods through MATLAB. While both elements provided accurate results, the 8-noded element, due to its higher-order shape functions, showed improved precision in capturing displacement and stress variations, especially in curved geometries. The comparison of results between MATLAB and APDL validated the correctness of the implementation. This analysis highlights the effectiveness of using higher-order elements in finite element modeling for more accurate stress analysis in axisymmetric structures.

REFERENCES

1. The book Young W. Kwon, Hyochoong Bang - The Finite Element Method Using MATLAB, Second Edition -CRC Press (2000)

2. The text_book_of_jn-reddy-introduction-to-the-finite-element-method-4e-mcgraw-hill-2018.

APPENDIX

MATLAB CODE

CODE FOR 4-NODE QUADRILATERAL

```
clc;
clear;
clf;
%% Parameters
ri = input("Enter internal radius="); % Inner radius of the cylinder (m)
ro = input("Enter external radius="); % outer radius of the cylinder (m)
L = input("Enter lenght of the cylinder="); % lenght of the cylinder (m)
nr = input("enter number of divisions in r direction=");  % Number of elements
along r direction
nz = input("enter numbe of divisions in z direction=");  % Number of elements
along y

E = input("enter Youngs moduLus in MPa=");   % Young's modulus (Pa)
v = input("enter poissons ratio=");     % Poisson's ratio
P = input("enter internal pressue in MPa= ");    % Pressure applied on the right side
(N/m²)

% Constitutive matrix
C = (E / ((1+v)*(1-2*v))) * [1-v, v, v, 0;
                        v, 1-v, v, 0;
                        v, v, 1-v, 0;
                        0, 0, 0, (1-2*v)/2];

%% Generate Node Coordinates
r_vals = linspace(ri, ro, nr + 1);  % Radial nodes from inner to outer radius
z_vals = linspace(0, L, nz + 1);    % Axial nodes from 0 to height H

node_coords = [];
for j = 1:length(z_vals)
```

```
    for i = 1:length(r_vals)
        node_coords = [node_coords; r_vals(i), z_vals(j)];
    end
end

num_nodes = size(node_coords, 1);

%% Generate Elements (4-node Quadrilaterals)
elements = [];
for j = 1:nz  % Axial direction (Z)
    for i = 1:nr  % Radial direction (R)
        n1 = (j - 1) * (nr + 1) + i;
        n2 = n1 + 1;
        n3 = n2 + (nr + 1);
        n4 = n1 + (nr + 1);
        elements = [elements; n1, n2, n3, n4];
    end
end

num_elements = size(elements, 1);



%% Visualization of Mesh
figure;
hold on;
for elem = 1:num_elements
    nodes = elements(elem, :);
    coords = node_coords(nodes, :);
    fill(coords(:,1), coords(:,2), 'w', 'EdgeColor', 'k'); % Plot elements
end
axis equal;
title('Axisymmetric Mesh (r-z Plane)');
xlabel('Radial Coordinate (r)');
```

```
ylabel('Axial Coordinate (z)');
%% Gauss Quadrature Points and Weights (2x2)
gauss_points = [-1/sqrt(3), 1/sqrt(3)];
weights = [1, 1];
%% Define Symbolic Shape Functions
syms xi eta

% Shape functions for 4-node quadrilateral
N1 = (1 - xi) * (1 - eta) / 4;
N2 = (1 + xi) * (1 - eta) / 4;
N3 = (1 + xi) * (1 + eta) / 4;
N4 = (1 - xi) * (1 + eta) / 4;
Ni = [N1, N2, N3, N4];

% Compute derivatives symbolically
dN_dxi = [eta/4 - 1/4, 1/4 - eta/4, eta/4 + 1/4, - eta/4 - 1/4];
dN_deta = [xi/4 - 1/4, - xi/4 - 1/4, xi/4 + 1/4, 1/4 - xi/4];
%% Initialize Global Stiffness Matrix
K_global = zeros(2 * num_nodes, 2 * num_nodes);
F_global = zeros(2 * num_nodes, 1);
%% Compute Stiffness Matrix for Axisymmetric Analysis
for elem = 1:num_elements
   % Extract node indices and coordinates for the current element
   node_ids = elements(elem, :);
   coords = node_coords(node_ids, :);

   % Element stiffness matrix
   Ke = zeros(8, 8);

   % Gauss Integration
   for i = 1:2
      for j = 1:2
         xi_val = gauss_points(i);
```

```
        eta_val = gauss_points(j);

        % Evaluate shape function derivatives at Gauss points
        dN_dxi_val = double(subs(dN_dxi, [xi, eta], [xi_val, eta_val]));
        dN_deta_val = double(subs(dN_deta, [xi, eta], [xi_val, eta_val]));

        % Compute Jacobian Matrix
        J = [dN_dxi_val; dN_deta_val] * coords;
        detJ = det(J);
        Jinv = inv(J);

        % Compute B matrix for axisymmetric analysis
        B = zeros(4, 8);

        % Compute r at Gauss point correctly
        Ni_val = double(subs(Ni, [xi, eta], [xi_val, eta_val]));
        r_gauss = Ni_val * coords(:,1);

        for k = 1:4
            dN_nat = [dN_dxi_val(k); dN_deta_val(k)];
            dN_xy = Jinv * dN_nat;

            B(1, 2*k-1) = dN_xy(1);  % dN/dr
            B(2, 2*k)   = dN_xy(2);  % dN/dz
            B(3, 2*k-1) = Ni_val(k) / r_gauss;  % Hoop strain term (N/r)
            B(4, 2*k-1) = dN_xy(2);  % dN/dz
            B(4, 2*k)   = dN_xy(1);  % dN/dr
        end

        % Stiffness matrix at Gauss point
        Ke = Ke + B' * C * B * detJ * weights(i) * weights(j) * 2 * pi * r_gauss;
    end
end
```

```
    % Assemble into Global Stiffness Matrix
    dof_map = [2*node_ids-1; 2*node_ids];
    dof_map = dof_map(:)';
    K_global(dof_map, dof_map) = K_global(dof_map, dof_map) + Ke;
end
%% Apply Boundary Conditions (Fix Only Uz at z = 0 and z = L)
fixed_nodes_z = find(node_coords(:,2) == 0 | node_coords(:,2) == L); % Nodes at
z = 0 (bottom) and z = L (top)
fixed_dofs_z = 2 * fixed_nodes_z; % Fix only Uz (longitudinal displacement)

fixed_dofs = unique(fixed_dofs_z(:)); % Ensure unique DOFs

% Apply boundary conditions to stiffness and force matrices
K_global(fixed_dofs, :) = 0;
K_global(:, fixed_dofs) = 0;
K_global(fixed_dofs, fixed_dofs) = eye(length(fixed_dofs));
F_global(fixed_dofs) = 0; % Ensure force vector consistency

%% Apply Internal Pressure Load at Inner Radius (r = ri)
inner_nodes = find(node_coords(:,1) == ri); % Find nodes at inner radius
inner_dofs = 2 * inner_nodes - 1; % Ur (radial displacement) DOFs

% Compute total force per unit length (Pressure × circumference × element length)
element_length = L / nz;  % Length of each element along z
total_force = P * 2 * pi * ri * element_length;  % Total force per element
(axisymmetric)

% Distribute force across inner-edge nodes using shape functions
for i = 1:length(inner_nodes)
    if i == 1  % First node (only connected to one element)
        F_global(inner_dofs(i)) = total_force / 2;  % Apply half force for first node
    elseif i == length(inner_nodes)  % Last node (only connected to one element)
```

```
        F_global(inner_dofs(i)) = total_force / 2;  % Apply half force for last node
    else  % Internal nodes (connected to two elements)
        F_global(inner_dofs(i)) = total_force;  % Full force for internal nodes
    end
end

%% Solve for Displacements

% Ensure no singularities by modifying fixed DOFs
nonzero_dofs = setdiff(1:length(F_global), fixed_dofs); % Remove fixed DOFs

% Extract submatrices for non-fixed DOFs
K_reduced = K_global(nonzero_dofs, nonzero_dofs);
F_reduced = F_global(nonzero_dofs);

% Solve for reduced displacements
U_reduced = K_reduced \ F_reduced;

% Initialize full displacement vector
U = zeros(length(F_global), 1);
U(nonzero_dofs) = U_reduced; % Assign computed values



%% Extract Displacement Results
Ux = U(1:2:end); % Radial displacements (Ur)
Uz = U(2:2:end); % Axial displacements (Uz)
max(Ux)
min(Ux)
min(Uz)
%% Generate Grid Data
x = node_coords(:,1); % Radial coordinate (r)
y = node_coords(:,2); % Axial coordinate (z)
```

```
% Create interpolant for radial displacement (Ux)
Fx = scatteredInterpolant(x, y, Ux, 'natural', 'none');

% Generate Regular Grid for Contour Plot
[xq, yq] = meshgrid(linspace(min(x), max(x), 100), linspace(min(y), max(y), 100));
Uxq = Fx(xq, yq);  % Interpolated radial displacement

%% Plot Radial Displacement Contour (Ur)
figure;
contourf(xq, yq, Uxq, 30, 'LineColor', 'none'); % Smooth contour plot
colorbar;
title('Radial Displacement Contour (Ur)');
xlabel('Radial Coordinate (r)');
ylabel('Axial Coordinate (z)');
axis equal;

%% Compute Stress at Each Element
element_stress = zeros(num_elements, 4); % Stresses (σ_rr, σ_zz, σ_θθ, τ_rz)

for elem = 1:num_elements
    node_ids = elements(elem, :);
    coords = node_coords(node_ids, :);

    % Extract displacement values for this element (Fixed)
    dof_map = reshape([2*node_ids-1; 2*node_ids], [], 1); % Ensure column vector
    Ue = U(dof_map);

    % Initialize stress accumulator
    stress_at_gauss = zeros(4, 4); % (4 stress components) × (4 Gauss points)

    for i = 1:2
        for j = 1:2
            xi_val = gauss_points(i);
            eta_val = gauss_points(j);

            % Compute shape function derivatives
            dN_dxi_val = double(subs(dN_dxi, [xi, eta], [xi_val, eta_val]));
            dN_deta_val = double(subs(dN_deta, [xi, eta], [xi_val, eta_val]));

            % Compute Jacobian and inverse
            J = [dN_dxi_val; dN_deta_val] * coords;
            Jinv = inv(J);

            % Compute B-matrix
            B = zeros(4, 8);
            Ni_val = double(subs(Ni, [xi, eta], [xi_val, eta_val]));
            r_gauss = Ni_val * coords(:,1);

            for k = 1:4
                dN_nat = [dN_dxi_val(k); dN_deta_val(k)];
                dN_xy = Jinv * dN_nat;

                B(1, 2*k-1) = dN_xy(1);  % dN/dr
                B(2, 2*k)   = dN_xy(2);  % dN/dz
                B(3, 2*k-1) = Ni_val(k) / r_gauss;  % Hoop strain term
                B(4, 2*k-1) = dN_xy(2);  % dN/dz
                B(4, 2*k)   = dN_xy(1);  % dN/dr
            end

            % Compute stress at Gauss point
            stress_at_gauss(:, (i-1)*2 + j) = C * B * Ue;
        end
    end

    % Average stress over Gauss points
    element_stress(elem, :) = mean(stress_at_gauss, 2);
```

```
end

%% Convert Elemental Stresses to Nodal Stresses
nodal_stress_rr = zeros(num_nodes, 1);
nodal_stress_zz = zeros(num_nodes, 1);
node_count = zeros(num_nodes, 1);

for elem = 1:num_elements
    node_ids = elements(elem, :);

    % Accumulate stress values
    nodal_stress_rr(node_ids) = nodal_stress_rr(node_ids) + element_stress(elem,
1); % σ_rr
    nodal_stress_zz(node_ids) = nodal_stress_zz(node_ids) + element_stress(elem,
2); % σ_zz
    node_count(node_ids) = node_count(node_ids) + 1;
end

% Average over shared nodes
nodal_stress_rr = nodal_stress_rr ./ node_count;
minimum_radial_stress=min(nodal_stress_rr)
maximum_radial_stress=max(nodal_stress_rr)
nodal_stress_zz = nodal_stress_zz ./ node_count;
minimum_axial_stress=min(nodal_stress_zz)
maximum_axial_stress=max(nodal_stress_zz)
%% Plot Radial Stress (σ_rr)
figure;
patch('Faces', elements, 'Vertices', node_coords, ...
      'FaceVertexCData', nodal_stress_rr, 'EdgeColor', 'k', 'FaceColor', 'interp');
colorbar;
title('Radial Stress (\sigma_{rr}) Distribution');
xlabel('Radial Coordinate (r)');
ylabel('Axial Coordinate (z)');
```

```
axis equal;

%% Plot Axial Stress (σ_zz)
figure;
patch('Faces', elements, 'Vertices', node_coords, ...
    'FaceVertexCData', nodal_stress_zz, 'EdgeColor', 'k', 'FaceColor', 'interp');
colorbar;
title('Axial Stress (\sigma_{zz}) Distribution');
xlabel('Radial Coordinate (r)');
ylabel('Axial Coordinate (z)');
axis equal;

%% Compute Hoop Stress (σ_θθ) at Nodes
nodal_stress_theta = zeros(num_nodes, 1);

for elem = 1:num_elements
   node_ids = elements(elem, :);

   % Accumulate hoop stresses
   nodal_stress_theta(node_ids)          =          nodal_stress_theta(node_ids)          +
element_stress(elem, 3); % σ_θθ
end

% Average over shared nodes
nodal_stress_theta = nodal_stress_theta ./ node_count;

% Find min/max values
min_hoop_stress = min(nodal_stress_theta);
max_hoop_stress = max(nodal_stress_theta);
disp(['Minimum Hoop Stress (σ_θθ_min): ', num2str(min_hoop_stress)]);
disp(['Maximum Hoop Stress (σ_θθ_max): ', num2str(max_hoop_stress)]);

%% Plot Hoop Stress (σ_θθ) Distribution
```

```
figure;
patch('Faces', elements, 'Vertices', node_coords, ...
    'FaceVertexCData', nodal_stress_theta, 'EdgeColor', 'k', 'FaceColor', 'interp');
colorbar;
title('Hoop Stress (\sigma_{\theta\theta}) Distribution');
xlabel('Radial Coordinate (r)');
ylabel('Axial Coordinate (z)');
axis equal;
```

CODE FOR 8-NODE QUADRILATERAL

```
clf;
clc;
clear;
%% Parameters
ri = 30; % Inner radius of the cylinder (m)
ro = 50; % Outer radius of the cylinder (m)
L = 40; % Length of the cylinder (m)
E = 2e5;   % Young's modulus (Pa)
v = 0.3;   % Poisson's ratio
P = 40;    % Pressure applied on the right side (N/m²)

% Constitutive matrix
C = (E / ((1+v)*(1-2*v))) * [1-v, v, v, 0;
                             v, 1-v, v, 0;
                             v, v, 1-v, 0;
                             0, 0, 0, (1-2*v)/2];

% Define node coordinates
nodes=readmatrix('coords.csv');
nodes(:,3)=nodes(:,3)-ri;
elements1 = readmatrix('element connectivity.csv');
```

```
% Extract main nodes (first 4 columns)
elements = elements1(:, 1:4);
% Extract mid-side nodes (last 4 columns)
elements2 = elements1(:, 5:8);
%% Plot the mesh
figure;
hold on;
for i = 1:size(elements,1)
    elemNodes = elements(i, :);
    r = nodes(elemNodes, 2);
    z = nodes(elemNodes, 3);
    fill(r, z, 'c', 'FaceAlpha', 0.3, 'EdgeColor', 'k', 'LineWidth', 1.5);
end
% Plot nodes
scatter(nodes(:,2), nodes(:,3), 50, 'r', 'filled');
for i = 1:size(nodes,1)
    text(nodes(i,2), nodes(i,3), num2str(nodes(i,1)), 'VerticalAlignment', 'bottom',
'HorizontalAlignment', 'right', 'FontSize', 10);
end
grid on;
axis equal;
xlabel('X Coordinate');
ylabel('Y Coordinate');
title('Mesh Visualization');
hold off;
    %% Define Symbolic Shape Functions for 8-Node Quadrilateral
    syms xi eta

    % Shape functions for 8-node serendipity quadrilateral
    N1 = -(1 - xi) * (1 - eta) * (1 + xi + eta) / 4;
    N2 = -(1 + xi) * (1 - eta) * (1 - xi + eta) / 4;
    N3 = -(1 + xi) * (1 + eta) * (1 - xi - eta) / 4;
    N4 = -(1 - xi) * (1 + eta) * (1 + xi - eta) / 4;
    N5 = (1 - xi^2) * (1 - eta) / 2; % Mid-edge bottom
    N6 = (1 + xi) * (1 - eta^2) / 2; % Mid-edge right
    N7 = (1 - xi^2) * (1 + eta) / 2; % Mid-edge top
    N8 = (1 - xi) * (1 - eta^2) / 2; % Mid-edge left

    Ni = [N1, N2, N3, N4, N5, N6, N7, N8];

    % Compute derivatives symbolically
    dN_dxi = [- ((eta - 1)*(eta + xi + 1))/4 - ((eta - 1)*(xi - 1))/4,
        ((eta - 1)*(eta - xi + 1))/4 - ((eta - 1)*(xi + 1))/4,
        ((eta + 1)*(eta + xi - 1))/4 + ((eta + 1)*(xi + 1))/4,
        ((eta + 1)*(xi - eta + 1))/4 + ((eta + 1)*(xi - 1))/4,
        xi*(eta - 1),
        1/2 - eta^2/2,
        -xi*(eta + 1),
        eta^2/2 - 1/2];
    dN_deta = [- ((xi - 1)*(eta + xi + 1))/4 - ((eta - 1)*(xi - 1))/4,
        ((xi + 1)*(eta - xi + 1))/4 + ((eta - 1)*(xi + 1))/4,
        ((xi + 1)*(eta + xi - 1))/4 + ((eta + 1)*(xi + 1))/4,
        ((xi - 1)*(xi - eta + 1))/4 - ((eta + 1)*(xi - 1))/4,
        xi^2/2 - 1/2,
        -eta*(xi + 1),
        1/2 - xi^2/2,
        eta*(xi - 1)];

     %% Gauss Quadrature Points and Weights (3x3 for 8-Node Elements)
   gauss_points = [-sqrt(3/5), 0, sqrt(3/5)]; % Standard points for 3-point quadrature
weights = [5/9, 8/9, 5/9]; % Weights are all 1 for 3×3 integration
    %% Initialize Global Stiffness Matrix
    num_nodes = size(nodes,1);
    num_dof = 2 * num_nodes;
    K_global = zeros(num_dof, num_dof);
    F_global = zeros(num_dof, 1);
```

```
num_elements = size(elements,1); % number of elements
node_coords(:,1)=nodes(:,2); % storing r coords in new variable
node_coords(:,2)=nodes(:,3); % storing z coords in new variable

%% Compute Stiffness Matrix for Axisymmetric 8-Node Elements
for elem = 1:num_elements
    node_ids = elements1(elem, :);
    coords = node_coords(node_ids, :);
    Ke = zeros(16, 16);

    for i = 1:3  % Use 3x3 Gauss Quadrature
        for j = 1:3
            xi_val = gauss_points(i);
            eta_val = gauss_points(j);

            % Evaluate shape function derivatives at Gauss points
            dN_dxi_val = double(subs(dN_dxi, [xi, eta], [xi_val, eta_val]));
            dN_deta_val = double(subs(dN_deta, [xi, eta], [xi_val, eta_val]));

            % Compute Jacobian Matrix
           J = [dN_dxi_val'; dN_deta_val'] * coords;
            detJ = det(J);
            Jinv = inv(J);

            % Compute B matrix
            B = zeros(4, 16);
            Ni_val = double(subs(Ni, [xi, eta], [xi_val, eta_val]));
            r_gauss = Ni_val * coords(:,1);

            for k = 1:8  %CONSTRUCTION OF B MATRIX
                dN_nat = [dN_dxi_val(k); dN_deta_val(k)];
                dN_xy = Jinv * dN_nat;
                B(1, 2*k-1) = dN_xy(1);
                B(2, 2*k)   = dN_xy(2);
                B(3, 2*k-1) = Ni_val(k) / r_gauss;
                B(4, 2*k-1) = dN_xy(2);
                B(4, 2*k)   = dN_xy(1);
            end

            % Stiffness matrix at Gauss point
            Ke = Ke + B' * C * B * detJ * weights(i) * weights(j) * 2 * pi * r_gauss;
        end
    end

    % Assemble into Global Stiffness Matrix
    dof_map = [2*node_ids-1; 2*node_ids];
    dof_map = dof_map(:)';
    K_global(dof_map, dof_map) = K_global(dof_map, dof_map) + Ke;
  end
%% Apply boundary conditions
fixed_dof = [];
for i = 1:size(nodes, 1) % 21 numbeer of nodes
  if (nodes(i, 3) == 0 || nodes(i, 3) == max(nodes(:, 3))) % Fix top and bottom plane
    fixed_dof = [fixed_dof; i*2]; % fixing Uz dof
  end
end

for i=1:size(fixed_dof,1)
K_global(fixed_dof(i), :) = 0;
K_global(:, fixed_dof(i)) = 0;

K_global((fixed_dof(i)), (fixed_dof(i))) = 1;
F_global(fixed_dof(i)) = 0; % Ensure force vector consistency
end
```

```
%% Apply Internal Pressure Load at Inner Radius (r = ri)
force_dof = [];
for i = 1:size(nodes, 1) % number of nodes
   if (nodes(i, 2) == ri)
      force_dof = [force_dof; i]; % force applied in r direction only
   end
end
for i = 1:size(nodes, 1)
   if nodes(i, 3)~= 0
      zzz = i + 2;
      break;
   end
end
zzzz=nodes(zzz,3); %element size in z direction
full_force = P * 2 * pi * ri *zzzz;
force_dof2=2*force_dof-1; %applying force only to r direction
for i = 1:size(force_dof2,1)
   if i < 3 % applying at the end nodes
      F_global(force_dof2(i)) = F_global(force_dof2(i)) + full_force*(1/6); % Half
force for first two DOFs
   elseif any(elements2(:) == force_dof(i)) % check for mid nodes
      F_global(force_dof2(i)) = F_global(force_dof2(i)) + full_force*(4/6);
   else
      F_global(force_dof2(i))  =  F_global(force_dof2(i))  +  full_force*(2/6);  %
connecting nodes
   end
end

U = K_global\F_global; % Solve for displacement

Ur = []; % Initialize an empty array to store Ux values

for i = 1:2:(size(F_global,1)-1) %all odd terms are Ux
```

```
    Ur = [Ur; U(i)]; % Append values instead of overwriting
end
Uz=[];
for i = 2:2:size(F_global,1)
    Uz = [Uz; U(i)]; % Append values instead of overwriting
end
maxUx=max(Ur)
minUx=min(Ur)
maxUy=max(Uz)
minUy=min(Uz)



%%

% Solve for displacement
U = K_global\F_global;
% Extract displacement components
Ur = U(1:2:end); % X-displacement (radial)
Uz = U(2:2:end); % Y-displacement (axial)

% Compute displacement magnitude
U_magnitude = sqrt(Ur.^2 + Uz.^2);

% Plot displacement contour
figure;
hold on;
for i = 1:size(elements,1)
    elemNodes = elements(i, :);
    r = nodes(elemNodes, 2);
    z = nodes(elemNodes, 3);
    c = U_magnitude(elemNodes); % Displacement magnitude at element nodes
    patch(r, z, c, 'EdgeColor', 'k', 'FaceColor', 'interp');
end
```

```
scatter(nodes(:,2), nodes(:,3), 50, U_magnitude, 'filled'); % Overlay node colors
colorbar;
title('Displacement Contour');
xlabel('X Coordinate');
ylabel('Y Coordinate');
axis equal;
grid on;
hold off;
%% Initialize Storage
stress_r_nodal = zeros(num_nodes, 1);
stress_z_nodal = zeros(num_nodes, 1);
stress_theta_nodal = zeros(num_nodes, 1);
node_count = zeros(num_nodes, 1);

%% Compute Nodal Stresses
for elem = 1:num_elements
    node_ids = elements1(elem, :);
    coords = node_coords(node_ids, :);
    elem_dof = [2*node_ids-1; 2*node_ids];
    elem_dof = elem_dof(:)';
    Ue = U(elem_dof);

    for local_node = 1:8  % Loop over element nodes
        xi_val = [-1, 1, 1, -1, 0, 1, 0, -1]; % Natural coords of 8 nodes
        eta_val = [-1, -1, 1, 1, -1, 0, 1, 0];

        % Compute Shape Function Derivatives
        dN_dxi_val = double(subs(dN_dxi, [xi, eta], [xi_val(local_node),
eta_val(local_node)]));
        dN_deta_val = double(subs(dN_deta, [xi, eta], [xi_val(local_node),
eta_val(local_node)]));

        % Compute Jacobian & B Matrix
        J = [dN_dxi_val'; dN_deta_val'] * coords;
        Jinv = inv(J);
        Ni_val = double(subs(Ni, [xi, eta], [xi_val(local_node), eta_val(local_node)]));
        r_node = Ni_val * coords(:,1); % Compute r at node

        B = zeros(4, 16);
        for k = 1:8
          dN_nat = [dN_dxi_val(k); dN_deta_val(k)];
          dN_xy = Jinv * dN_nat;

          B(1, 2*k-1) = dN_xy(1);  % ε_rr
          B(2, 2*k)   = dN_xy(2);  % ε_zz
          B(3, 2*k-1) = Ni_val(k) / r_node;  % ε_θθ
          B(4, 2*k-1) = dN_xy(2);
          B(4, 2*k)   = dN_xy(1);
        end

        % Compute Stresses
        strain = B * Ue;
        stress = C * strain;

        % Assign stresses to corresponding global node
        global_node = node_ids(local_node);
        stress_r_nodal(global_node) = stress_r_nodal(global_node) + stress(1);
        stress_z_nodal(global_node) = stress_z_nodal(global_node) + stress(2);
        stress_theta_nodal(global_node) = stress_theta_nodal(global_node) + stress(3);
        node_count(global_node) = node_count(global_node) + 1;
    end
end

% Average stress values at shared nodes
```

```
stress_r_nodal = stress_r_nodal ./ node_count;
stress_z_nodal = stress_z_nodal ./ node_count;
stress_theta_nodal = stress_theta_nodal ./ node_count;

%% Display Maximum and Minimum Stresses
fprintf('Maximum Radial Stress: %.4f MPa\n', max(stress_r_nodal));
fprintf('Minimum Radial Stress: %.4f MPa\n', min(stress_r_nodal));
fprintf('Maximum Axial Stress: %.4f MPa\n', max(stress_z_nodal));
fprintf('Minimum Axial Stress: %.4f MPa\n', min(stress_z_nodal));
fprintf('Maximum Hoop Stress: %.4f MPa\n', max(stress_theta_nodal));
fprintf('Minimum Hoop Stress: %.4f MPa\n', min(stress_theta_nodal));

%% Extract X, Y coordinates
X = node_coords(:, 1);
Y = node_coords(:, 2);

%% Plot Stress Contours Using Patch (Ignoring Middle Nodes)
corner_indices = [1, 2, 3, 4]; % Only use corner nodes (no midside nodes)

figure;
hold on;
colormap jet;
for e = 1:num_elements
   nodes = elements1(e, corner_indices); % Use only corner nodes
   patch(X(nodes), Y(nodes), stress_r_nodal(nodes), 'FaceColor', 'interp',
'EdgeColor', 'none');
end
colorbar;
title('Radial Stress Contour (\sigma_r)');
xlabel('X'); ylabel('Y');
axis equal; hold off;

figure;
```

```
hold on;
colormap jet;
for e = 1:num_elements
    nodes = elements1(e, corner_indices);
    patch(X(nodes), Y(nodes), stress_z_nodal(nodes), 'FaceColor', 'interp',
'EdgeColor', 'none');
end
colorbar;
title('Axial Stress Contour (\sigma_z)');
xlabel('X'); ylabel('Y');
axis equal; hold off;

figure;
hold on;
colormap jet;
for e = 1:num_elements
    nodes = elements1(e, corner_indices);
    patch(X(nodes), Y(nodes), stress_theta_nodal(nodes), 'FaceColor', 'interp',
'EdgeColor', 'none');
end
colorbar;
title('Hoop Stress Contour (\sigma_\theta)');
xlabel('X'); ylabel('Y');
axis equal; hold off;
```