TERM PROJECT REPORT

On

2D TRUSS ANALYSIS USING PYHTON

Submitted to

International Institute of Information Technology

by

MEDARI VIKAS – 2024210010

ABHISHEK TIWARI – 2024210012

Master of Technology
Computer Aided Structural Engineering

Under the Guidance of

K. SRIRANJANI

Professor

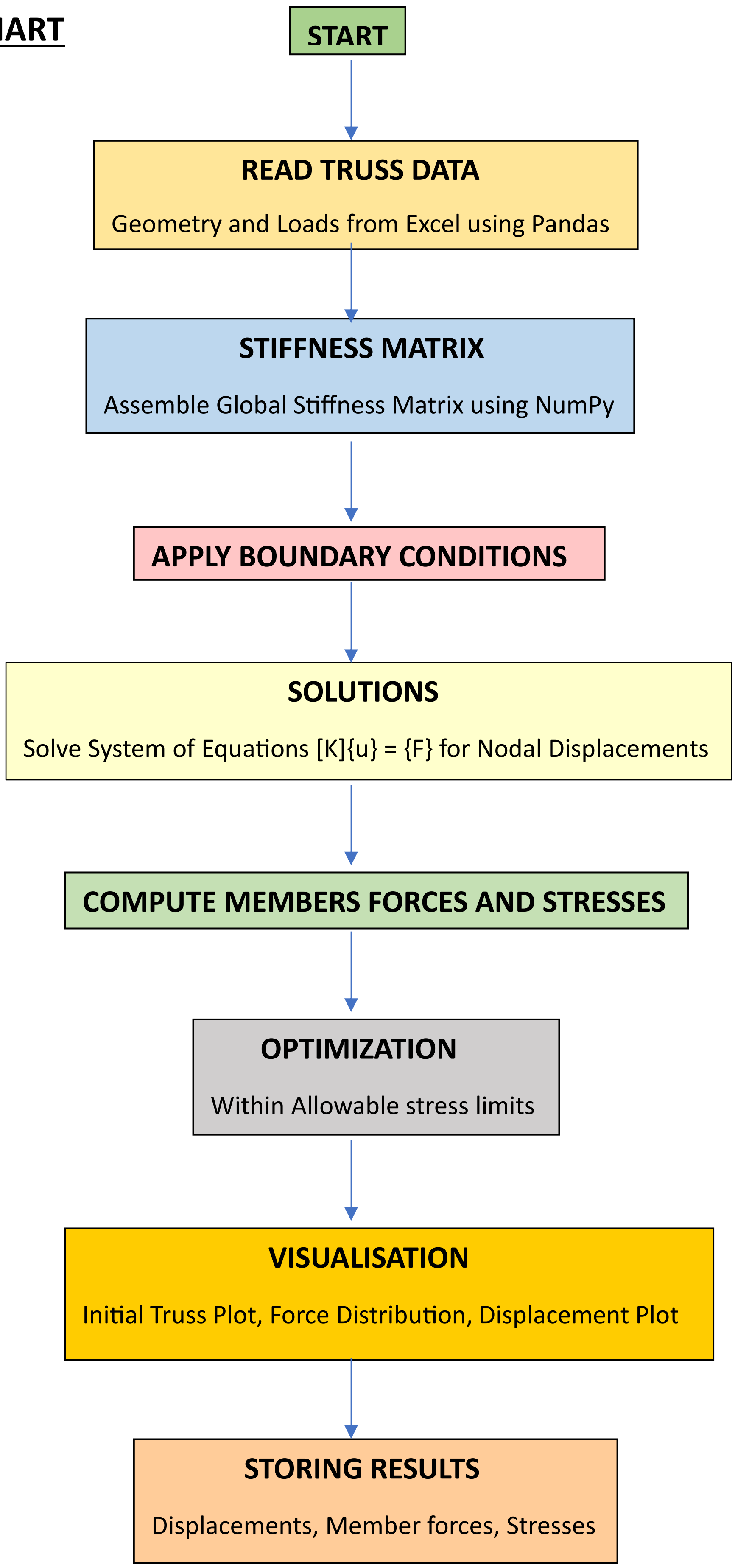
EARTHQUAKE ENGINEERING RESEARCH CENTRE
INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

1. Introduction

This report presents a detailed structural analysis of a 2D truss system using Python. The analysis includes:

- **Data Extraction:** Reading truss geometry, material properties, and loading conditions from an Excel file.
- **Stiffness Matrix Formulation:** Computing local and global stiffness matrices.
- **Displacement Calculation:** Solving for nodal displacements using boundary conditions.
- **Force and Stress Analysis:** Determining member forces and stresses.
- **Visualization:** Plotting the initial truss, deformed shape, and force distribution.
- **Optimization Check:** Verifying if stresses are within allowable limits.
- **Data Export:** Storing results in an Excel file.

2. FLOWCHART


START
READ TRUSS DATA
Geometry and Loads from Excel using Pandas
STIFFNESS MATRIX
Assemble Global Stiffness Matrix using NumPy
APPLY BOUNDARY CONDITIONS
SOLUTIONS
Solve System of Equations [K]{u} = {F} for Nodal Displacements
COMPUTE MEMBERS FORCES AND STRESSES
OPTIMIZATION
Within Allowable stress limits
VISUALISATION
Initial Truss Plot, Force Distribution, Displacement Plot
STORING RESULTS
Displacements, Member forces, Stresses

3. Methodology

3.1 Data Extraction from Excel

The initial truss data is imported using pandas and includes:

- **Node coordinates** (x, y)
- **Member connectivity** (start node, end node)
- **Cross-sectional area (A)** and **Young's modulus (E)**
- **Support conditions** (fixed or pinned)
- **Applied loads** (forces at nodes)

3.2 Stiffness Matrix Formulation

1. **Member Stiffness Matrix (Local Coordinates):**
    - Each member has a 4×4 stiffness matrix derived from truss element theory.
    - Transformation to global coordinates using direction cosines.
2. **Global Stiffness Matrix Assembly:**
    - Combining individual member stiffness matrices into the global system.
    - Applying boundary conditions to remove rigid-body motion.

3.3 Solving for Displacements

- The system of equations:

$$[K_{global}]\{U\}=\{F\}$$

where:

- [Kglobal] = Global stiffness matrix
- {U} = Displacement vector
- {F} = Applied load vector

- Solved using numpy.linalg.solve() after applying boundary conditions.

3.4 Member Forces and Stresses

- **Axial Force:**

$$F = EA/L * \quad (uj - ui)\cos\theta + (vj - vi)\sin\theta$$

- **Stress:**

$$\sigma = F/A$$

- **Check against allowable stress:**
    - If $|\sigma| > \sigma allowable$, optimization is needed.

3.5 Data Export

- Displacements, forces, and stresses are stored in an Excel file using pandas.

3.6 Visualization

- **Initial Truss Plot:** Nodes, members, and applied loads.
- **Force Distribution:** Color-coded tension (blue) and compression (red).
- **Displacement Plot:** Exaggerated deformed shape.

4. Python Implementation

4.1 Required Libraries

- numpy
  - Efficient numerical computations, especially for matrices and linear algebra.
- pandas
  - Data handling and manipulation of Excel files or tabular data.
- matplotlib.pyplot
  - Visualization of data and plots.
- openpyxl
  - Excel workbook manipulation (specifically for .xlsx files).

4.2 Data Reading from Excel

- **def (**read_single_sheet**):**
    - Reads Excel data and organizes it into structured DataFrames.

- **def** main():

```
data = read_single_sheet("truss_analysis.xlsx")
nodes = data['NODES']
elements = data['ELEMENTS']
loads = data['LOADS']
supports = data['SUPPORTS']
node_coords = nodes[['X', 'Y']].values
num_nodes = len(node_coords)
```

NODES		
Node	X	Y
1	0	0
2	10	15
3	20	25
4	30	35
5	40	25
6	50	15
7	60	0

```
element_nodes = elements[['StartNode', 'EndNode']].values.astype(int) - 1
element_areas = elements['Area'].astype(float).values
element_E = elements['E'].astype(float).values
num_elements = len(element_nodes)
```

ELEMENTS				
Element	StartNode	EndNode	Area	E
1	1	2	500.00	2.00E+05
2	2	3	500.00	2.00E+05
3	3	4	500.00	2.00E+05
4	4	5	500.00	2.00E+05
5	5	6	500.00	2.00E+05
6	6	7	500.00	2.00E+05
7	1	6	500.00	2.00E+05
8	2	7	500.00	2.00E+05
9	2	5	500.00	2.00E+05
10	3	6	500.00	2.00E+05
11	3	5	500.00	2.00E+05

```
loads_vector = np.zeros(2 * num_nodes)
  for _, row in loads.iterrows():
      node = int(row['Node']) - 1
      loads_vector[2 * node] = row['Fx']
      loads_vector[2 * node + 1] = row['Fy']
```

LOADS		
Node	Fx	Fy
4	100000	-500000
3	0	-200000
5	0	-100000

```
support_conditions = np.zeros(2 * num_nodes, dtype=int)
    for _, row in supports.iterrows():
        node = int(row['Node']) - 1
        support_conditions[2 * node] = row['Xfixed']
        support_conditions[2 * node + 1] = row['Yfixed']
```

SUPPORTS		
Node	Xfixed	Yfixed
1	1	1
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	1	1

4.3 Stiffness Matrix Assembly

- Computes **local stiffness** for each truss element.

```
for i in range(num_elements):
    start, end = element_nodes[i]
    E = element_E[i]
    A = element_areas[i]

    x1, y1 = node_coords[start]
    x2, y2 = node_coords[end]
    dx = x2 - x1
    dy = y2 - y1
    L = np.sqrt(dx**2 + dy**2)

    c = dx / L
    s = dy / L

    k = (E * A / L) * np.array([
        [ c*c,  c*s, -c*c, -c*s],
        [ c*s,  s*s, -c*s, -s*s],
        [-c*c, -c*s,  c*c,  c*s],
        [-c*s, -s*s,  c*s,  s*s]
    ])
```

4.4 Applying Boundary Conditions

- Modifies stiffness matrix to account for **fixed supports**.
- Assembles into a **global stiffness matrix**.

```
support_conditions = np.zeros(2 * num_nodes, dtype=int)
for _, row in supports.iterrows():
    node = int(row['Node']) - 1
    support_conditions[2 * node] = row['Xfixed']
    support_conditions[2 * node + 1] = row['Yfixed']

idx = [2 * start, 2 * start + 1, 2 * end, 2 * end + 1]

    for r in range(4):
        for c_ in range(4):
            global_stiffness[idx[r], idx[c_]] += k[r, c_]

for i in range(2 * num_nodes):
    if support_conditions[i] == 1:
        global_stiffness[i, :] = 0
        global_stiffness[:, i] = 0
        global_stiffness[i, i] = 1
        loads_vector[i] = 0
```

4.5 Solving for Displacements

```
displacements = np.linalg.solve(global_stiffness, loads_vector)
```

4.6 Computing Member Forces & Stresses

```
member_forces = np.zeros(num_elements)
member_stresses = np.zeros(num_elements)
for i in range(num_elements):
    start, end = element_nodes[i]
    E = element_E[i]
    A = element_areas[i]

    u1, v1 = displacements[2 * start], displacements[2 * start + 1]
    u2, v2 = displacements[2 * end], displacements[2 * end + 1]

    x1, y1 = node_coords[start]
    x2, y2 = node_coords[end]
    dx = x2 - x1
    dy = y2 - y1
    L = np.sqrt(dx**2 + dy**2)

    c = dx / L
    s = dy / L

    delta = (u2 - u1) * c + (v2 - v1) * s
    force = (E * A / L) * delta
    member_forces[i] = force
    member_stresses[i] = force / A
```

4.7 Export Results to Excel

```
# Displacement
disp_df = pd.DataFrame([{
    'Node': i + 1,
    'X-Displacement (m)': displacements[2 * i],
    'Y-Displacement (m)': displacements[2 * i + 1]
} for i in range(num_nodes)])

# Forces and Stress
force_df = pd.DataFrame([{
    'Element': i + 1,
    'Force (N)': member_forces[i],
    'Stress (Pa)': member_stresses[i],
    'Nature': "Tension" if member_forces[i] > 0 else "Compression"
} for i in range(num_elements)])

with pd.ExcelWriter('truss_analysis.xlsx', engine='openpyxl', mode='a', if_sheet_exists='replace') as writer:
    disp_df.to_excel(writer, sheet_name='Displacements', index=False)
    force_df.to_excel(writer, sheet_name='Member Forces', index=False)
```

4.8 Visualization

```
plot_truss_structure(node_coords, element_nodes, loads_vector)
```

- **Truss Structure**: Shows geometry and loads.

```
plot_member_forces(node_coords, element_nodes, member_forces)
```

- **Member Forces**: Visualizes force distribution

5. Results & Discussion

5.1 Displacements

- Maximum displacement occurs at Node **4** with Ux=0.0314, Uy=-0.324

Node	X-Displacement (m)	Y-Displacement (m)
1	0	0
2	-0.06694921	-0.059748376
3	0.039655127	-0.275474579
4	0.031471443	-0.323859438
5	0.011011877	-0.259466191
6	0.097631665	-0.056653467
7	0	0

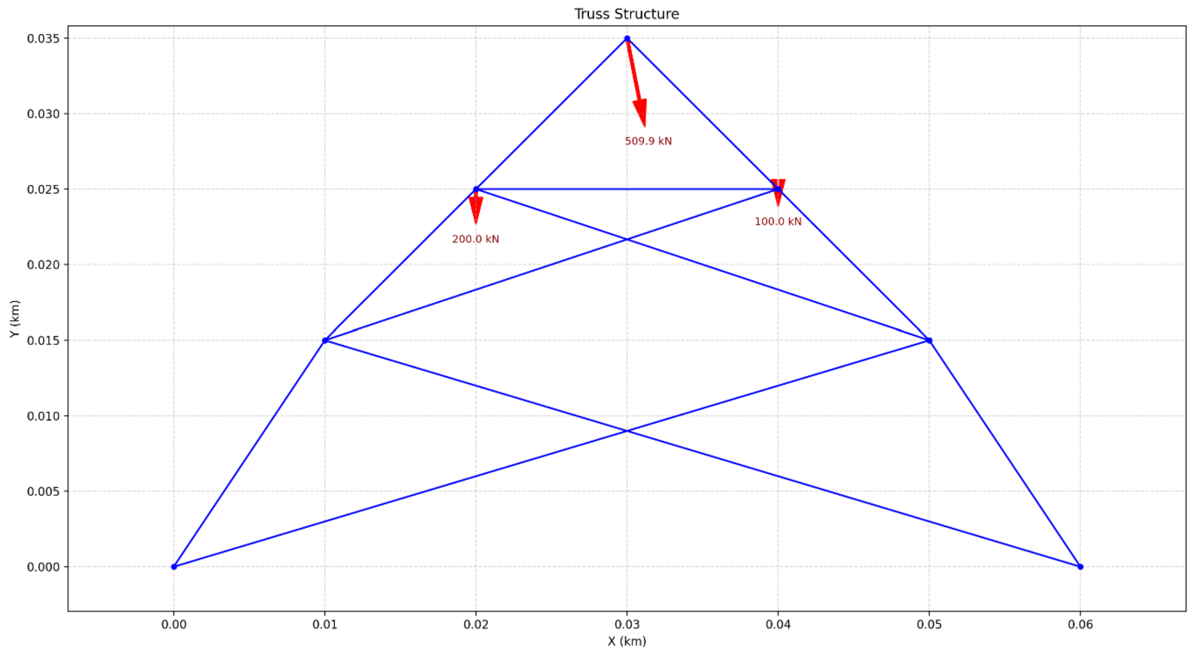
5.2 Member Forces & Stresses

- **Tension Members:** Highlighted in blue.
- **Compression Members:** Highlighted in red.
- **Critical Members:** Members exceeding allowable stress require optimization.

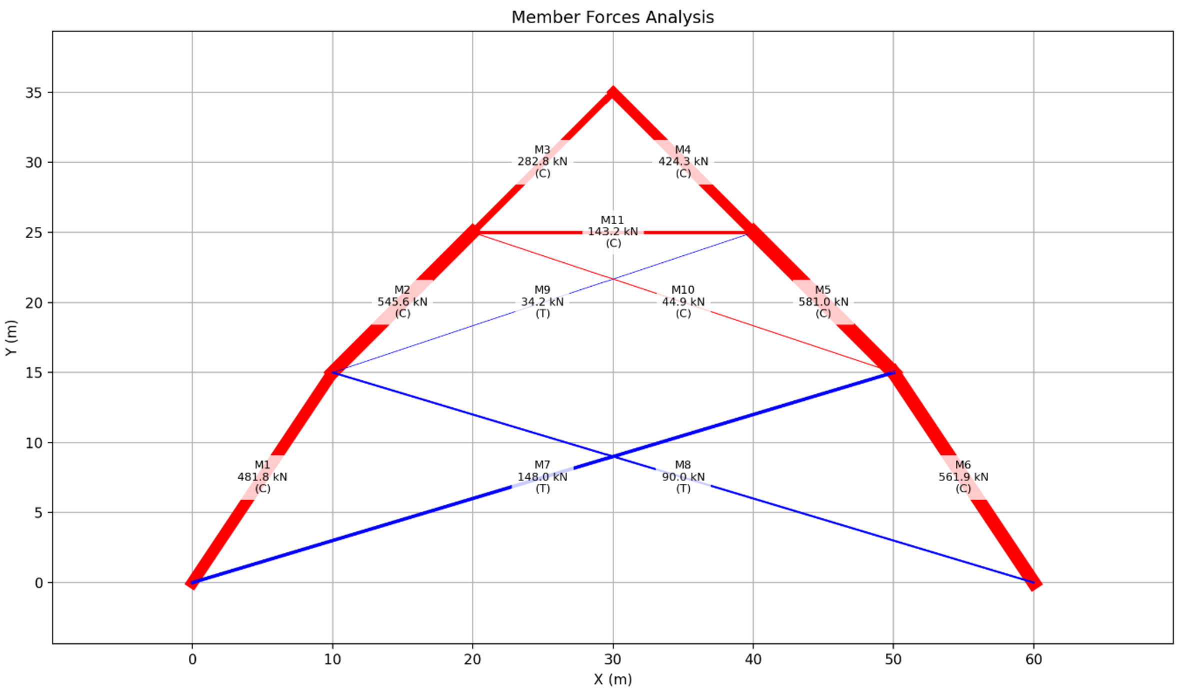
Element	Force (N)	Stress (Pa)	Nature	Optimization Needed
1	-481759.302	-963.5186036	Compression	Required
2	-545609.337	-1091.218675	Compression	Required
3	-282842.712	-565.6854249	Compression	Required
4	-424264.069	-848.5281374	Compression	Required
5	-580964.676	-1161.929353	Compression	Required
6	-561882.663	-1123.765327	Compression	Required
7	147955.2754	295.9105508	Tension	Not Required
8	89953.57256	179.9071451	Tension	Not Required
9	34165.44469	68.33088939	Tension	Not Required
10	-44891.4968	-89.78299362	Compression	Not Required
11	-143216.249	-286.432498	Compression	Not Required

5.3 Visualization

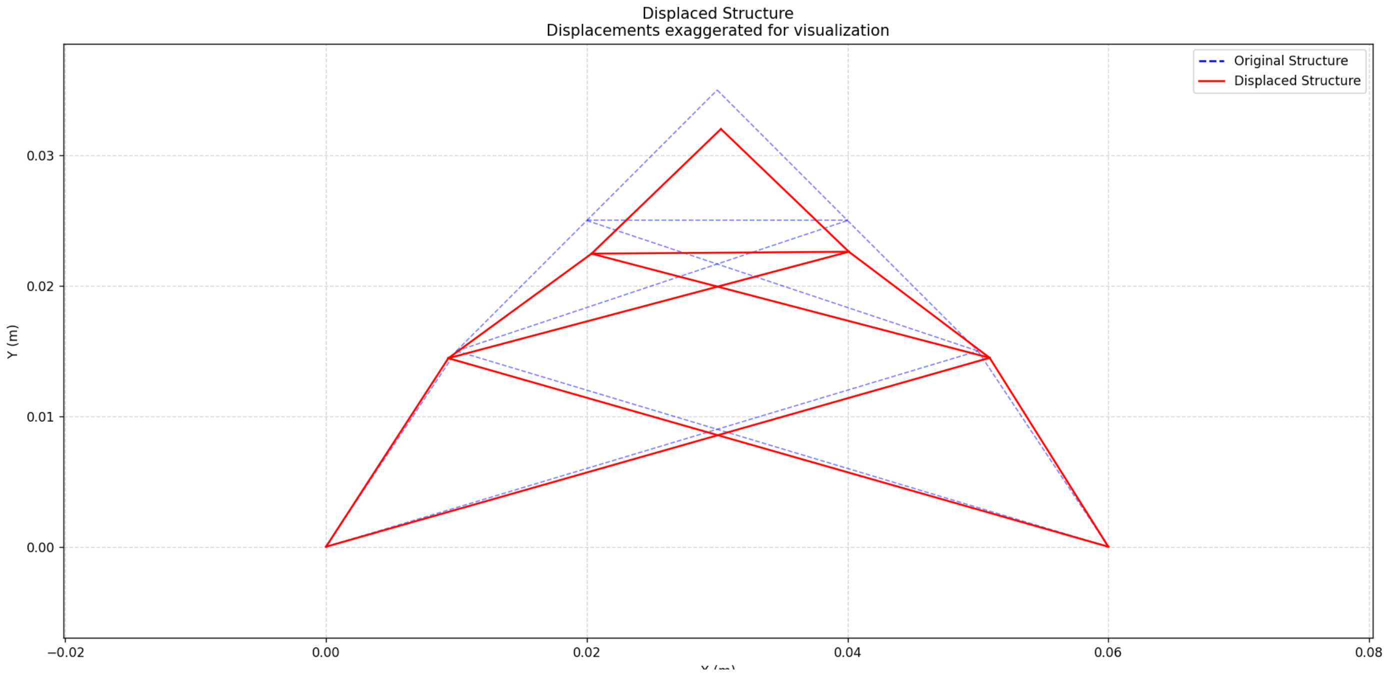
- **Initial Truss:** Shows geometry and applied loads.



- **Force Distribution:** Color-coded based on tension/compression.



- **Deformation Plot:** Exaggerated for clarity.



6. Conclusion

- The truss analysis successfully computed displacements, forces, and stresses.
- **Optimization Needed.**
    - If $\sigma_{max} > \sigma_{allowable}$, cross-section or material changes are required.
- Python provided an efficient and automated solution for structural analysis.