**By : KARN AGARWAL  
 2 ND YEAR**

**Finite element analysis**

Building a more detailed finite element analysis (FEA) project for structural analysis requires structuring the code into components and adding functionalities to handle larger systems. We’ll outline an FEA project for a 1D truss structure with multiple elements and nodes, following these steps:

1. **Project Structure and Basic Outline**:
   * Define element properties, geometry, and material.
   * Build the local stiffness matrix for each element.
   * Assemble the global stiffness matrix.
   * Apply loads and boundary conditions.
   * Solve the system to get nodal displacements.
   * Calculate reaction forces and element stresses.
2. **Code Components**:
   * **Classes and Functions** for managing elements, nodes, and the global system.
   * Modular **functions for stiffness matrix assembly**, load application, and boundary conditions.
   * **Input/Output**: Define input for structural layout (nodes, elements, boundary conditions) and output displacements, stresses, and reactions.

**Project Code**

Here's a more structured and scalable version of the FEA project in Python.

**1. Define Constants and Classes for Elements and Nodes**

python

Copy code

import numpy as np

# Define Material and Geometric Properties for a Truss Element

class TrussElement:

def \_\_init\_\_(self, node\_start, node\_end, area, youngs\_modulus, length):

self.node\_start = node\_start # Start node

self.node\_end = node\_end # End node

self.area = area # Cross-sectional area

self.youngs\_modulus = youngs\_modulus # Young's modulus

self.length = length # Element length

def stiffness\_matrix(self):

# Local stiffness matrix for 1D truss element

k = (self.youngs\_modulus \* self.area / self.length) \* np.array([[1, -1], [-1, 1]])

return k

**2. Define the Structural System**

This part manages nodes, elements, and the global stiffness matrix assembly.

python

Copy code

class TrussStructure:

def \_\_init\_\_(self, num\_nodes):

self.num\_nodes = num\_nodes

self.elements = []

self.forces = np.zeros(num\_nodes) # Global force vector

self.displacements = np.zeros(num\_nodes) # Displacements at each node

self.stiffness\_matrix = np.zeros((num\_nodes, num\_nodes)) # Global stiffness matrix

def add\_element(self, element):

self.elements.append(element)

# Add the element's stiffness to the global stiffness matrix

n1, n2 = element.node\_start, element.node\_end

k = element.stiffness\_matrix()

self.stiffness\_matrix[n1, n1] += k[0, 0]

self.stiffness\_matrix[n1, n2] += k[0, 1]

self.stiffness\_matrix[n2, n1] += k[1, 0]

self.stiffness\_matrix[n2, n2] += k[1, 1]

def apply\_force(self, node, force):

self.forces[node] = force

def apply\_boundary\_condition(self, fixed\_nodes):

# Modify stiffness matrix and force vector for fixed boundary conditions

for node in fixed\_nodes:

self.stiffness\_matrix[node, :] = 0

self.stiffness\_matrix[:, node] = 0

self.stiffness\_matrix[node, node] = 1 # Set diagonal to 1 for stability

self.forces[node] = 0

def solve(self):

# Solve the system of equations for displacements

self.displacements = np.linalg.solve(self.stiffness\_matrix, self.forces)

def calculate\_element\_stress(self, element):

# Calculate stress in an element after solving for displacements

u1 = self.displacements[element.node\_start]

u2 = self.displacements[element.node\_end]

strain = (u2 - u1) / element.length

stress = element.youngs\_modulus \* strain

return stress

**3. Define the Main Analysis Process**

This is the main function where elements are created, forces and boundary conditions are applied, and results are calculated.

python

Copy code

# Main function to set up and solve a truss problem

def main():

# Parameters

E = 210e9 # Young's modulus in Pascals

A = 0.01 # Cross-sectional area in m²

L = 1.0 # Length of each element in meters

# Create a structure with 3 nodes (for a 2-element truss system)

structure = TrussStructure(num\_nodes=3)

# Define elements (each element connects two nodes)

element1 = TrussElement(node\_start=0, node\_end=1, area=A, youngs\_modulus=E, length=L)

element2 = TrussElement(node\_start=1, node\_end=2, area=A, youngs\_modulus=E, length=L)

# Add elements to the structure

structure.add\_element(element1)

structure.add\_element(element2)

# Apply forces (e.g., 1000 N at node 2)

structure.apply\_force(node=2, force=1000)

# Apply boundary conditions (e.g., node 0 is fixed)

structure.apply\_boundary\_condition(fixed\_nodes=[0])

# Solve for displacements

structure.solve()

# Output results

print("Nodal Displacements:", structure.displacements)

# Calculate and output stresses in each element

for i, element in enumerate(structure.elements):

stress = structure.calculate\_element\_stress(element)

print(f"Stress in Element {i+1}: {stress} Pa")

**4. Run the Analysis**

Simply call the main() function to execute the analysis for the truss structure.

python

Copy code

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Explanation of Each Component**

1. **TrussElement Class**: Manages properties and stiffness matrix of each truss element. The stiffness matrix is calculated based on the material and geometric properties.
2. **TrussStructure Class**:
   * Maintains the list of elements and nodes.
   * Builds and stores the global stiffness matrix.
   * Has methods for adding elements, applying forces, applying boundary conditions, and solving the displacement equations.
   * Calculates element stresses based on solved displacements.
3. **Main Function (main)**: Sets up the structure, adds elements, applies forces and boundary conditions, and solves for nodal displacements. It also calculates and prints the stresses in each element.

**Expected Output**

When you run this code, the output should include:

* **Nodal Displacements**: Displacements of each node.
* **Stresses in Each Element**: The stress in each truss element based on the calculated displacements.

**Expanding the Project**

You can further expand this project by:

* Adding support for multiple dimensions (2D or 3D truss systems).
* Handling beam elements with bending and shear forces.
* Including dynamic analysis for time-dependent loading.
* Adding visualization for node positions, deformed shapes, and stress distribution.

This setup provides a solid foundation for understanding and building more complex FEA models.

4o