PRECEPT 4 (HW 2)

CEE 361-513: Introduction to Finite Element Methods

Wednesday Oct. 10

PROBLEM 1

Consider the following 1-D truss system.

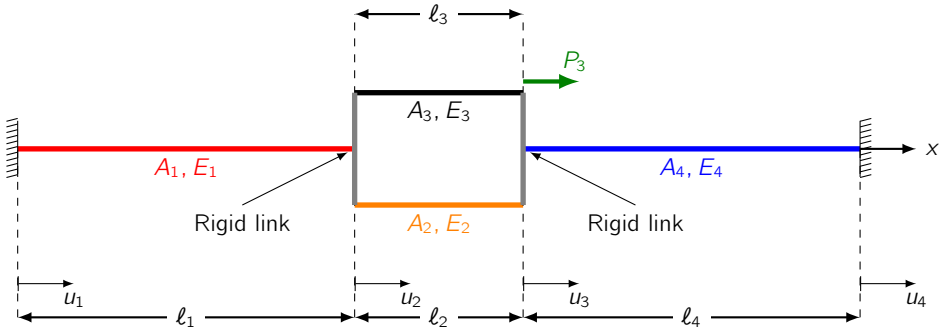

Figure 1: The 1-D Truss system

Using the information provided below, solve for u_2, P_3 and the reactions.

$$\ell_1 = \ell_4 = 2.0\text{m} \qquad \ell_2 = \ell_3 = 1.0\text{m}$$

$$A_1E_1 = 200\text{kN} \qquad A_2E_2 = 300\text{kN} \qquad A_3E_3 = 400\text{kN} \qquad A_4E_4 = 500\text{kN}$$

$$u_3 = 4\text{mm}$$

Solution :

1.1 Connectivity + Free Body Diagram

The first step is to prepare the labeling conventions: write the connectivity matrix for the system (relating the local node numbers with the global node numbers) and draw a free body diagram of the system (include prescribed loads, prescribed displacements, and unknown reactions, unknown loads, and unknown displacements).

element	i node	j node
1	1	2
2	2	3
3	2	3
4	3	4

Table 1: Connectivity Matrix

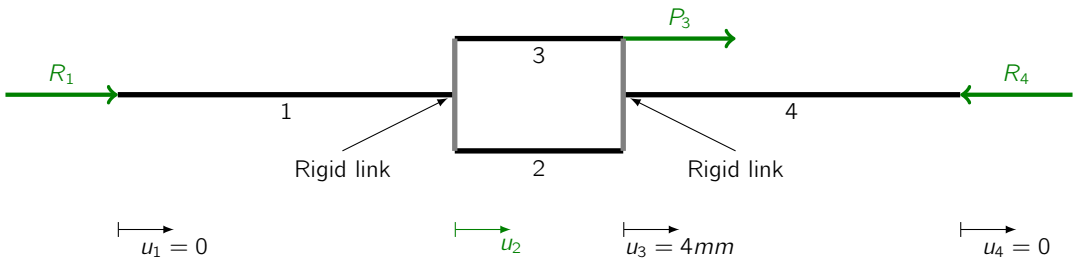

Figure 2: The Complete FBD

1.2 Element Stiffness Formulation

We then write the element stiffness matrices. The notation convention is the same as that followed in class for the local node labels $(i, j$.

$$\begin{bmatrix} -f_i^1 \\ f_j^1 \end{bmatrix} = \begin{bmatrix} \frac{A_1E_1}{\ell_1} & -\frac{A_1E_1}{\ell_1} \\ -\frac{A_1E_1}{\ell_1} & \frac{A_1E_1}{\ell_1} \end{bmatrix} \begin{bmatrix} u_i^1 \\ u_j^1 \end{bmatrix} \qquad \begin{bmatrix} -f_i^2 \\ f_j^2 \end{bmatrix} = \begin{bmatrix} \frac{A_2E_2}{\ell_2} & -\frac{A_2E_2}{\ell_2} \\ -\frac{A_2E_2}{\ell_2} & \frac{A_2E_2}{\ell_2} \end{bmatrix} \begin{bmatrix} u_i^2 \\ u_j^2 \end{bmatrix}$$

$$\begin{bmatrix} -f_i^3 \\ f_j^3 \end{bmatrix} = \begin{bmatrix} \frac{A_3E_3}{\ell_3} & -\frac{A_3E_3}{\ell_3} \\ -\frac{A_3E_3}{\ell_3} & \frac{A_3E_3}{\ell_3} \end{bmatrix} \begin{bmatrix} u_i^3 \\ u_j^3 \end{bmatrix} \qquad \begin{bmatrix} -f_i^4 \\ f_j^4 \end{bmatrix} = \begin{bmatrix} \frac{A_4E_4}{\ell_4} & -\frac{A_4E_4}{\ell_4} \\ -\frac{A_4E_4}{\ell_4} & \frac{A_4E_4}{\ell_4} \end{bmatrix} \begin{bmatrix} u_i^4 \\ u_j^4 \end{bmatrix}$$

1.3 Local to Global DOF

Using the connectivity matrix we replace the local node numbers $(i, j$ with the global node numbers in the element stiffness.

$$\begin{bmatrix} -f_i^1 \\ f_j^1 \end{bmatrix} = \begin{bmatrix} \frac{A_1E_1}{\ell_1} & -\frac{A_1E_1}{\ell_1} \\ -\frac{A_1E_1}{\ell_1} & \frac{A_1E_1}{\ell_1} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \qquad \begin{bmatrix} -f_i^2 \\ f_j^2 \end{bmatrix} = \begin{bmatrix} \frac{A_2E_2}{\ell_2} & -\frac{A_2E_2}{\ell_2} \\ -\frac{A_2E_2}{\ell_2} & \frac{A_2E_2}{\ell_2} \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \end{bmatrix}$$

$$\begin{bmatrix} -f_i^3 \\ f_j^3 \end{bmatrix} = \begin{bmatrix} \frac{A_3E_3}{\ell_3} & -\frac{A_3E_3}{\ell_3} \\ -\frac{A_3E_3}{\ell_3} & \frac{A_3E_3}{\ell_3} \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \end{bmatrix} \qquad \begin{bmatrix} -f_i^4 \\ f_j^4 \end{bmatrix} = \begin{bmatrix} \frac{A_4E_4}{\ell_4} & -\frac{A_4E_4}{\ell_4} \\ -\frac{A_4E_4}{\ell_4} & \frac{A_4E_4}{\ell_4} \end{bmatrix} \begin{bmatrix} u_3 \\ u_4 \end{bmatrix}$$

1.4 Nodal Equilibrium

Writing the equilibrium equations for the 4 nodes:

$$\begin{aligned} R_1 &= -f_i^1 \\ P_2 &= f_j^1 - f_i^2 - f_i^3 \\ P_3 &= -f_j^2 - f_j^3 + f_i^4 \\ R_4 &= f_j^4 \end{aligned}$$

1.5 Assemble global stiffness matrix from element stiffness matrices

Let $k_i = A_i E_i/\ell_i$ for $i = 1...3$. We can write down the equilibrium equations in matrix form. Namely, as we did in class, write the equilibrium equations with a load vector containing reactions and external forces, denoted it by $\{P\}$, the stiffness matrix denoted by $[K]$, and the vector of displacements $\{U\}$ such that

$$[K]\{U\} = \{P\}$$

Let us denote

$$k_1 = \frac{A_1 E_1}{\ell_1} \qquad k_2 = \frac{A_2 E_2}{\ell_2} \qquad k_3 = \frac{A_3 E_3}{\ell_3} \qquad k_4 = \frac{A_4 E_4}{\ell_4}$$

$$\begin{bmatrix} R_1 \\ P_2 \\ P_3 \\ R_4 \end{bmatrix} = \begin{bmatrix} -f_i^1 \\ f_j^1 - f_i^2 - f_i^3 \\ -f_j^2 - f_j^3 + f_i^4 \\ f_i^4 \end{bmatrix} = \begin{bmatrix} k_1 & -k_1 & 0 & 0 \\ -k_1 & k_1 + k_2 + k_3 & -k_2 - k_3 & 0 \\ 0 & -k_2 - k_3 & k_2 + k_3 + k_4 & -k_4 \\ 0 & 0 & -k_4 & k_4 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

1.6 Apply Boundary Conditions

For our given problem:

$$u_1 = 0 \qquad u_3 = 4\text{mm} \qquad u_4 = 0$$
$$R_1 = ? \qquad P_3 = ? \qquad R_4 = ? \qquad u_2 = ?$$

Modifying our stiffness and force matrices to reflect the knowns and unknowns:

$$\begin{bmatrix} 0 \\ 0 \\ 4 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -k_1 & k_1 + k_2 + k_3 & -k_2 - k_3 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

1.7 Solve for unknown displacements and reactions

The only unknown is u_2 and we can solve directly for it. Recall that the rows of the global formulation just represent equations:

$$u_2 = \frac{4\,(k_2 + k_3)}{(k_1 + k_2 + k_3)}$$

The reactions can be found by multiplying the corresponding rows of the **original** stiffness matrix with the displacement vector.

$$R_1 = k_1 u_1$$
$$P_3 = (-k_2 - k_3)\, u_2 + (k_2 + k_3 + k_4)\, u_3$$
$$R_4 = -k_4 u_3$$

In general we would not solve the matrix by hand (too tedious). We now write a python code for solving the same problem following the steps mentioned in Problem 4 of homework 2. We can use our hand calculation to verify our code.

```
"""
Solves the python problem for the precept #3
Author(s) : Vivek Kumar
Last Modified : 2nd October 2017
"""
import numpy as np
import sympy as sp
import numpy.linalg as LA
# Total number of elements
nel = 4
# Number of nodes in an element
en = 2
# Total number of nodes
nnp = 4
# number of degrees of freedom per node
ndf = 1
# total degrees of freedom in an element
ele_dof = nen*ndf
# total degrees of freedom in the system
num_dof = nnp*ndf

# Define the material and geometrical properties
E = [200.0, 300.0, 400.0, 500.0] #kN/mm^2
A = [1.0, 1.0, 1.0, 1.0] #mm^2
I = [2000.0, 1000.0, 1000.0, 2000.0] #mm

# Define the connectivity matrix
connectivity = np.array([[0,1],[1,2], [1,2], [2,3]])
# Define the coordinates of the nodes
coordinates = [0.0,I[0],I[1]+I[0],I[1]+I[0]+I[3] ]

# Function to return the global degree of freedom from the local degree of freedom
def local_to_global_dof(connectivity_array,element_number,local_dof):
        return connectivity[element_number,local_dof]

# Function to return the element stiffness matrix
def element_stiffness(young_modulus, area, x_i, x_j):
        K_e = young_modulus*area/(x_j-x_i)*np.array([[1,-1],[-1,1]])
        return K_e

# Initialize the global stiffness matrix
KG = np.zeros((num_dof,num_dof))

# Loop over all elements
for e in range(nel):
        x_i = coordinates[connectivity[e,0]] # The i coordinate of the element
        x_j = coordinates[connectivity[e,1]] # The j coordinate of the element
        E_e = E[e] # The young's modulus of the element
        A_e = A[e] # The area of the element
        I_e = I[e] # The length of the element
        K_e = element_stiffness(E_e,A_e,x_i,x_j) # Obtain the element stiffness matrix
```

```
        # Assemble the global stiffness matrix
        for p in range(ele_dof):
                global_p = local_to_global_dof(connectivity,e,p)
                for q in range(ele_dof):
                        global_q = local_to_global_dof(connectivity,e,q)
                        KG[global_p,global_q] += K_e[p,q]
# Print KG
print(KG)
# Given load
P = np.zeros(num_dof)
P[1] = 0. #kN

# Nodes of known displacement
# Set one if known else 0
bc = [1,0,1,1]

# Dirichlet Boundary conditions
g = np.zeros(num_dof)
g[0] = 0.0
g[2] = 4.0
g[3] = 0.0

# Updated force matrix
F = np.zeros(num_dof)

# Initialize a new matrix with KG values
K = KG.copy()

# Updated Stiffness matrix
for b in range(len(bc)):
        for num in range(num_dof):
                if bc[b] == 1:
                        if b == num:
                                K[b,num] = 1.0
                        else:
                                K[b,num] = 0.0

# Updated Stiffness matrix
for b in range(len(bc)):
        if bc[b] == 1:
                F[b] = g[b]
        else:
                F[b] = P[b]

# Solve for unknown u
print(F)
u = np.dot(LA.inv(K),F.T) # take the inverse and multiply
#u = LA.solve(K,F.T) # ask numpy to solve

# Find reactions:
R1 = np.dot(KG[:,0],u)
P3 = np.dot(KG[:,2],u)
```

```
R4 = np.dot(KG[:,3],u)

print(u)
print(R1)
print(P3)
print(R4)
```