PRECEPT 5

CEE 361-513: Introduction to Finite Element Methods

Monday Oct. 16

PROBLEM 1

Consider the frame system shown below. Foreach node z = 1, 2, 3 we have associated coordinates $\boldsymbol{q}_z$ and associated global degrees of freedom $\boldsymbol{u}_z$ and θ_z, where both $\boldsymbol{q}$ and $\boldsymbol{u}$ are vectors while θ_z are scalar rotations. At node 1 the frame is free to rotate but is constrained to move along a plane whose normal is given by $\boldsymbol{m}_s$.

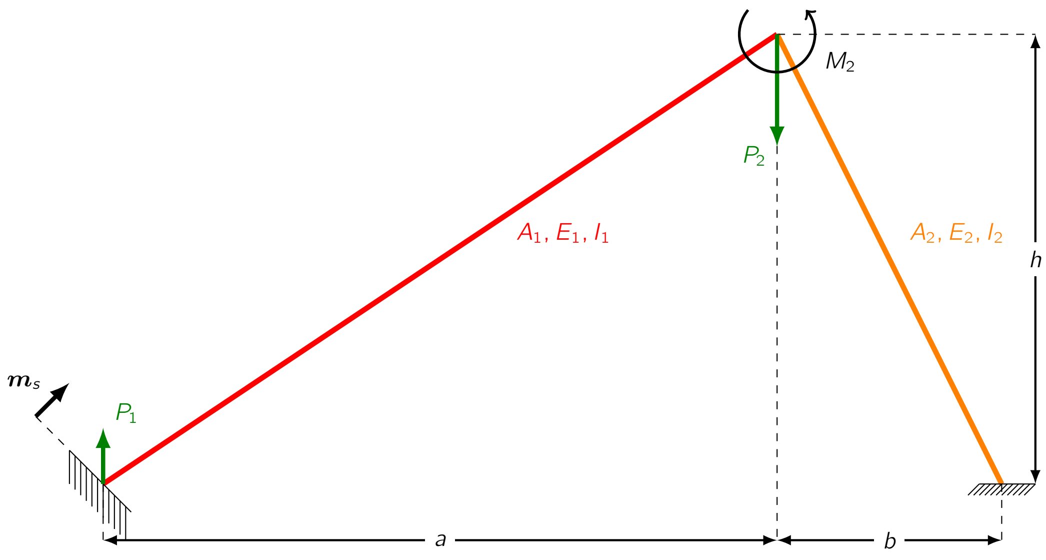

Figure 1: The 2-D Beam system

Let $a = 3.0,\ b = 1.0,\ h = 2.0$ and $A_1 = E_1 = I_1 = 2,\ A_2 = E_2 = I_2 = 3.$ Further Let $\boldsymbol{P}_1 = 1\mathbf{e}_2$, $\boldsymbol{P}_2 = -2\mathbf{e}_2$ and $M_2 = 1.$ The normal $\boldsymbol{m}_s = cos(\pi/4)\mathbf{e}_1 + sin(\pi/4)\mathbf{e}_2$ and the node 3 is completely constrained, i.e. $\boldsymbol{w}_3 = \mathbf{0}$. Using the information provided solve for the displacements, rotations and the reactions.

Solution :

The first step is to write the connectivity matrix for the system, relating the local node numbers with the global node numbers.

element	i node	j node
1	1	2
2	2	3

Table 1: Connectivity Matrix

We then generate the local stiffness matrix for the frame elements.

For element 1:

$$\boldsymbol{q}_i^1 = [0.0, 0.0] \qquad \boldsymbol{q}_j^1 = [3.0, 2.0]$$

$$\boldsymbol{n}^1 = \frac{\boldsymbol{q}_j - \boldsymbol{q}_i}{|\boldsymbol{q}_j - \boldsymbol{q}_i|} = \frac{[3.0, 2.0]}{\sqrt{13}}$$

The rotation operation is:

$$\boldsymbol{R} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

Hence, $\boldsymbol{s}$ is:

$$\begin{aligned} \boldsymbol{s}^1 &= \boldsymbol{R} \cdot \boldsymbol{n}^1 \\ &= \frac{[-2.0, 3.0]}{\sqrt{13}} \end{aligned}$$

Next we obtain the projection tensors:

$$\begin{aligned} \boldsymbol{n}^1 \otimes \boldsymbol{n}^1 &= \frac{1}{13} \begin{bmatrix} 9.0 & 6.0 \\ 6.0 & 4.0 \end{bmatrix} \\ \boldsymbol{s}^1 \otimes \boldsymbol{s}^1 &= \frac{1}{13} \begin{bmatrix} 4.0 & -6.0 \\ -6.0 & 9.0 \end{bmatrix} \end{aligned}$$

We then write the equilibrium equations in matrix form:

$$\begin{bmatrix} \boldsymbol{F}_i \\ M_i \\ \boldsymbol{F}_j \\ M_j \end{bmatrix} = \begin{bmatrix} [\boldsymbol{K_{fw}}] & [\boldsymbol{k_{f\theta}}] & [-\boldsymbol{K_{fw}}] & [\boldsymbol{k_{f\theta}}] \\ [\boldsymbol{k_{mw}}]^T & [k_{m\theta}] & [-\boldsymbol{k_{mw}}]^T & [\hat{k}_{m\theta}] \\ [-\boldsymbol{K_{fw}}] & [-\boldsymbol{k_{f\theta}}] & [\boldsymbol{K_{fw}}] & [-\boldsymbol{k_{f\theta}}] \\ [\boldsymbol{k_{mw}}]^T & [\hat{k}_{m\theta}] & [-\boldsymbol{k_{mw}}]^T & [k_{m\theta}] \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_i \\ \theta_i \\ \boldsymbol{w}_j \\ \theta_j \end{bmatrix}$$

where:

$$\begin{aligned} \boldsymbol{K_{fw}} &= \frac{A_1 E_1}{\ell_1} \boldsymbol{n}^1 \otimes \boldsymbol{n}^1 + \frac{12 E_1 I_1}{\ell_1^3} \boldsymbol{s}^1 \otimes \boldsymbol{s}^1 \\ k_{m\theta} &= \frac{4 E_1 I_1}{\ell_1} \\ \hat{k}_{m\theta} &= \frac{2 E_1 I_1}{\ell_1} \\ \boldsymbol{k}_{mw} = \boldsymbol{k_{f\theta}} &= \frac{6 E_1 I_1}{\ell_1^2} \boldsymbol{s}^1 \end{aligned}$$

The values for element 1 are:

$$\begin{aligned} \boldsymbol{K}^1_{\boldsymbol{fw}} &= \begin{bmatrix} 1.08 & 0.04 \\ 0.04 & 1.05 \end{bmatrix} \\ k^1_{m\theta} &= 4.44 \\ \hat{k}^1_{m\theta} &= 2.22 \\ \boldsymbol{k}^1_{mw} = \boldsymbol{k}^1_{\boldsymbol{f}\theta} &= \begin{bmatrix} -1.02 \\ 1.54 \end{bmatrix} \end{aligned}$$

This would give us our first element stiffness matrix and similarly we can find the stiffness matrix for the other element.Performing similar computation we obtain the following values of the terms for element 2.

$$\begin{aligned} \boldsymbol{K}^2_{\boldsymbol{fw}} &= \begin{bmatrix} 8.53 & 2.25 \\ 2.25 & 5.15 \end{bmatrix} \\ k^2_{m\theta} &= 16.1 \\ \hat{k}^2_{m\theta} &= 8.05 \\ \boldsymbol{k}^2_{mw} = \boldsymbol{k}^2_{\boldsymbol{f}\theta} &= \begin{bmatrix} 9.65 \\ 4.83 \end{bmatrix} \end{aligned}$$

We now need to assemble the element stiffness matrices.
We need to determine the size of the global stiffness matrix.

```
number of nodes (nnodes) = 3
Spatial dimension (space_dim) =2
local degree of freedom (local_dof)= space_dim + (space_dim-1) #each node has space_dim
number of translation +\ (space_dim-1) number of rotation
number of degree of freedom = (local_dof)*nnodes
```

$$\begin{bmatrix} \boldsymbol{V}_1 \\ M_1 \\ \boldsymbol{V}_2 \\ M_2 \\ \boldsymbol{V}_3 \\ M_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{K}^1_{\boldsymbol{fw}} & \boldsymbol{k}^1_{\boldsymbol{f}\theta} & -\boldsymbol{K}^1_{\boldsymbol{fw}} & \boldsymbol{k}^1_{\boldsymbol{f}\theta} & \boldsymbol{O} & \mathbf{0} \\ [\boldsymbol{k}^1_{m\boldsymbol{w}}]^T & k^1_{m\theta} & -[\boldsymbol{k}^1_{m\boldsymbol{w}}]^T & \hat{k}^1{}_{m\theta} & \mathbf{0}^T & 0 \\ -\boldsymbol{K}^1_{\boldsymbol{fw}} & -\boldsymbol{k}^1_{\boldsymbol{f}\theta} & \boldsymbol{K}^1_{\boldsymbol{fw}} + \boldsymbol{K}^2_{\boldsymbol{fw}} & -\boldsymbol{k}^1_{\boldsymbol{f}\theta} + \boldsymbol{k}^2_{\boldsymbol{f}\theta} & -\boldsymbol{K}^2_{\boldsymbol{fw}} & \boldsymbol{k}^2_{\boldsymbol{f}\theta} \\ [\boldsymbol{k}^1_{m\boldsymbol{w}}]^T & \hat{k}^1{}_{m\theta} & [-\boldsymbol{k}^1_{m\boldsymbol{w}}]^T + [\boldsymbol{k}^2_{m\boldsymbol{w}}]^T & k^1_{m\theta} + k^2_{m\theta} & -[\boldsymbol{k}^2_{m\boldsymbol{w}}]^T & \hat{k}^2{}_{m\theta} \\ \boldsymbol{O} & \mathbf{0} & -\boldsymbol{K}^2_{\boldsymbol{fw}} & -\boldsymbol{k}^2_{\boldsymbol{f}\theta} & \boldsymbol{K}^2_{\boldsymbol{fw}} & -\boldsymbol{k}^2_{\boldsymbol{f}\theta} \\ \mathbf{0}^T & 0 & [\boldsymbol{k}^2_{m\boldsymbol{w}}]^T & \hat{k}^2{}_{m\theta} & [-\boldsymbol{k}^2_{m\boldsymbol{w}}]^T & k^2_{m\theta} \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_1 \\ \theta_1 \\ \boldsymbol{w}_2 \\ \theta_2 \\ \boldsymbol{w}_3 \\ \theta_3 \end{bmatrix}$$

Next we need to apply the constraint and the boundary conditions. The normal to the plane is given by:

$$\boldsymbol{m}_s = cos(\pi/4)\mathbf{e}_1 + sin(\pi/4)\mathbf{e}_2$$

We need to add an extra unkown λ and add and extra row and column to our stiffness matrix.

$$\begin{bmatrix} \boldsymbol{V}_1 \\ M_1 \\ \boldsymbol{V}_2 \\ M_2 \\ \boldsymbol{V}_3 \\ M_3 \\ 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{K}^1_{\boldsymbol{fw}} & \boldsymbol{k}^1_{\boldsymbol{f}\theta} & -\boldsymbol{K}^1_{\boldsymbol{fw}} & \boldsymbol{k}^1_{\boldsymbol{f}\theta} & \boldsymbol{O} & \mathbf{0} & -\boldsymbol{m}_s \\ [\boldsymbol{k}^1_{m\boldsymbol{w}}]^T & k^1_{m\theta} & -[\boldsymbol{k}^1_{m\boldsymbol{w}}]^T & \hat{k}^1{}_{m\theta} & \mathbf{0}^T & 0 & 0 \\ -\boldsymbol{K}^1_{\boldsymbol{fw}} & -\boldsymbol{k}^1_{\boldsymbol{f}\theta} & \boldsymbol{K}^1_{\boldsymbol{fw}} + \boldsymbol{K}^2_{\boldsymbol{fw}} & -\boldsymbol{k}^1_{\boldsymbol{f}\theta} + \boldsymbol{k}^2_{\boldsymbol{f}\theta} & -\boldsymbol{K}^2_{\boldsymbol{fw}} & \boldsymbol{k}^2_{\boldsymbol{f}\theta} & \mathbf{0} \\ [\boldsymbol{k}^1_{m\boldsymbol{w}}]^T & \hat{k}^1{}_{m\theta} & [-\boldsymbol{k}^1_{m\boldsymbol{w}}]^T + [\boldsymbol{k}^2_{m\boldsymbol{w}}]^T & k^1_{m\theta} + k^2_{m\theta} & -[\boldsymbol{k}^2_{m\boldsymbol{w}}]^T & \hat{k}^2{}_{m\theta} & 0 \\ \boldsymbol{O} & \mathbf{0} & -\boldsymbol{K}^2_{\boldsymbol{fw}} & -\boldsymbol{k}^2_{\boldsymbol{f}\theta} & \boldsymbol{K}^2_{\boldsymbol{fw}} & -\boldsymbol{k}^2_{\boldsymbol{f}\theta} & \mathbf{0} \\ \mathbf{0}^T & 0 & [\boldsymbol{k}^2_{m\boldsymbol{w}}]^T & \hat{k}^2{}_{m\theta} & [-\boldsymbol{k}^2_{m\boldsymbol{w}}]^T & k^2_{m\theta} & 0 \\ \boldsymbol{m}_s^T & 0 & \mathbf{0}^T & 0 & \mathbf{0}^T & 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_1 \\ \theta_1 \\ \boldsymbol{w}_2 \\ \theta_2 \\ \boldsymbol{w}_3 \\ \theta_3 \\ \lambda \end{bmatrix}$$

Update the matrix based on boundary condition:

$$\begin{bmatrix} \boldsymbol{V}_1 \\ M_1 \\ \boldsymbol{V}_2 \\ M_2 \\ \mathbf{0} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \boldsymbol{K}^1_{\boldsymbol{fw}} & \boldsymbol{k}^1_{\boldsymbol{f}\theta} & -\boldsymbol{K}^1_{\boldsymbol{fw}} & \boldsymbol{k}^1_{\boldsymbol{f}\theta} & \boldsymbol{O} & \mathbf{0} & -\boldsymbol{m}_s \\ [\boldsymbol{k}^1_{m\boldsymbol{w}}]^T & k^1_{m\theta} & -[\boldsymbol{k}^1_{m\boldsymbol{w}}]^T & \hat{k}^1{}_{m\theta} & \mathbf{0}^T & 0 & 0 \\ -\boldsymbol{K}^1_{\boldsymbol{fw}} & -\boldsymbol{k}^1_{\boldsymbol{f}\theta} & \boldsymbol{K}^1_{\boldsymbol{fw}} + \boldsymbol{K}^2_{\boldsymbol{fw}} & -\boldsymbol{k}^1_{\boldsymbol{f}\theta} + \boldsymbol{k}^2_{\boldsymbol{f}\theta} & -\boldsymbol{K}^2_{\boldsymbol{fw}} & \boldsymbol{k}^2_{\boldsymbol{f}\theta} & \mathbf{0} \\ [\boldsymbol{k}^1_{m\boldsymbol{w}}]^T & \hat{k}^1{}_{m\theta} & [-\boldsymbol{k}^1_{m\boldsymbol{w}}]^T + [\boldsymbol{k}^2_{m\boldsymbol{w}}]^T & k^1_{m\theta} + k^2_{m\theta} & -[\boldsymbol{k}^2_{m\boldsymbol{w}}]^T & \hat{k}^2{}_{m\theta} & 0 \\ \boldsymbol{O} & \mathbf{0} & \boldsymbol{O} & \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0}^T & 0 & \mathbf{0}^T & 0 & \mathbf{0}^T & 1 & 0 \\ \boldsymbol{m}_s^T & 0 & \mathbf{0}^T & 0 & \mathbf{0}^T & 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_1 \\ \theta_1 \\ \boldsymbol{w}_2 \\ \theta_2 \\ \boldsymbol{w}_3 \\ \theta_3 \\ \lambda \end{bmatrix}$$

We can now solve our system.

PROBLEM 2

The python code for the above problem

```

# Define the element properties
elements = {};
elements[0]={'A':2.0,'E':2.0, 'I':2.0, 0:0, 1:1}
elements[1]={'A':3.0,'E':3.0, 'I':3.0, 0:1, 1:2}
# Define the coordinates
coordinates = np.array([[0.0,0.0],[3.0,2.0],[4.0,0.0]])
# Define element degree of freedom
ele_dof = 2
# total number of elements
nel = 2
```

```
def local_stiffness_frame(elts, crds, e):
        A, E, I = (elts[e]['A'], elts[e]['E'], elts[e]['I'])
        n = np.array(crds[elts[e][1]])-np.array(crds[elts[e][0]])
        L = LA.norm(n)
        n /= L
        R = np.array([[0,-1],[1,0]])
        s = np.dot(R,n)
        Kfw = A*E/L*np.outer(n,n)+12*E*I/(L**3)*np.outer(s,s)
        kmt = 4*E*I/L
        khmt = 2*E*I/L
        kmw = 6*E*I/(L**2)*s
        kft = 6*E*I/(L**2)*s
        space_dim = n.size
        n_nodes = 2
        n_dof = space_dim*n_nodes+n_nodes
        Ke = np.zeros((n_dof, n_dof))
        # Diagonal terms
        Ke[0:2,0:2] = Ke[3:5,3:5] = Kfw
        Ke[2,2] = Ke[5,5] = kmt
        # Non-diagonal terms
        Ke[0:2,2] = Ke[0:2,5] =kft
        Ke[0:2,3:5] = -Kfw
        Ke[2,3:5] = -kmw
        Ke[2,5] = khmt
        Ke[3:5,5] = -kft
        lower_indices = np.tril_indices(n_dof,-1)
        Ke[lower_indices] = 0.
        Ke += np.triu(Ke,1).T
        return Ke
```

```
nnodes = 3
space_dim = 2
local_dof = space_dim+(space_dim-1)
#each node has space_dim number of translation + (space_dim-1) number of rotation
num_dof = space_dim*nnodes+nnodes*(space_dim-1)
KG = np.zeros((num_dof,num_dof))

# Loop over all elements
for e in range(nel):
# Obtain the element stiffness matrix
KE = local_stiffness_frame(elements, coordinates, e)
# Assemble the global stiffness matrix
for p in range(ele_dof):
global_p = elements[e][p]
for q in range(ele_dof):
global_q = elements[e][q]
KG[global_p*local_dof:(global_p+1)*local_dof,global_q*local_dof:(global_q+1)*local_dof]\
                += KE[p*local_dof:(p+1)*local_dof,q*local_dof:(q+1)*local_dof]
```

```
ms = np.array([np.cos(np.pi/4),np.sin(np.pi/4)])
row = np.array([ms])
row = np.append(row,np.zeros(7))
row = np.resize(row,(1,num_dof))

col = np.append(np.array([-ms]),0.)
col = np.append(col,np.zeros(7))
col = np.resize(col,(num_dof+1,1))

K_new = KG.copy()

K_new = np.vstack([K_new, row])
K_new = np.hstack([K_new, col])
```

```
# Nodes of known displacement
# Set one if known else 0
bc = np.zeros(num_dof+1)
bc = [0,0,0,0,0,0,1,1,1,0]
# Given load
P = np.zeros(len(bc))
P[1] = 1.0
P[4] = -2.0
P[5] = 1.0
# Dirichlet Boundary conditions
g = np.zeros(len(bc))
# Updated force matrix
F = np.zeros(len(bc))
#F = np.array([[u1, P, u3, u4]])
# Initialize a new matrix with KG values
K = K_new.copy()
# Updated Stiffness matrix
for b in range(len(bc)):
        for num in range(len(bc)):
                if bc[b] == 1:
                        if b == num:
                                K[b,num] = 1.0
                        else:
                                K[b,num] = 0.0

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

# Solve for unknown u
u = LA.solve(K,F.T)
R = np.dot(K_new,u)
```