HOMEWORK 4

CEE 361-513: Introduction to Finite Element Methods

Due: Friday Nov 9 @ Midnight

NB: Students taking CEE 513 must complete all problems. All other students will not be graded for problems marked with $\star$, but are encourage to attempt them anyhow.

PROBLEM 1

Consider the frame 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 3 the frame is free to rotate but is constrained to move along a plane whose normal is given by $\boldsymbol{m}_s$.

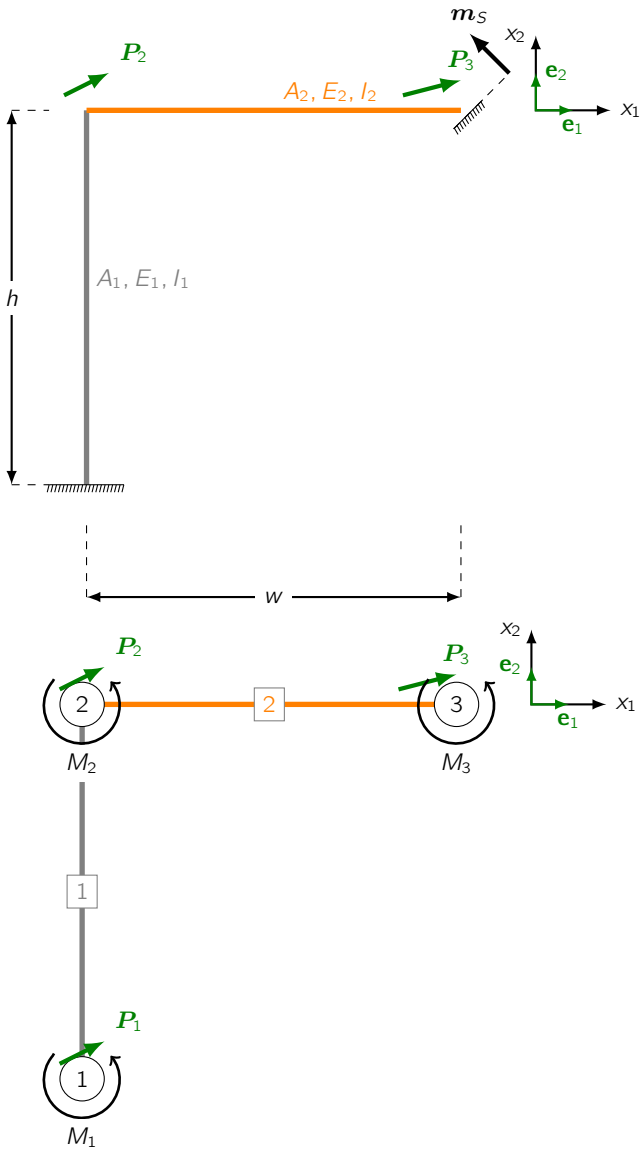

Figure 1: The system of uniaxial rods

1. For each node write the equilibrium equations in terms of the external forces $\boldsymbol{P}_k$, $k = 1, 2, 3$ and moments M_k, $k = 1, 2, 3$, and the internal forces $\boldsymbol{f}^e_{i,j}$ and moments $m^e_{i,j}$.

2. Write the general expression of internal forces (and moments) as the matrix vector operation of the *local element stiffness* and the *local degrees of freedom*.

3. For each element write the internal forces (and moments) as the matrix vector operation of the *local element stiffness* and the *GLOBAL degrees of freedom* using the connectivity array.

4. Using $\boldsymbol{K}^e_{\boldsymbol{fw}}, \boldsymbol{k}^e_{\boldsymbol{f}\theta}, \ldots$ (cf. lecture notes), write down the equilibrium equations in matrix form.

5. At node 1 we prevent the frame from moving (i.e. all displacements and rotations are zero). At node 3 we allow the truss to move along a plane whose unit normal is $\boldsymbol{m}_S$ as well as to rotate freely. Apply the aforementioned conditions to the matrix form of the previous step.

PROBLEM 2

Consider the frame of Problem 1. Let $w = 10, h = 10$, and $A_1 = E_1 = I_1 = 1, A_2 = E_2 = I_2 = 2$. Further let $\boldsymbol{P}_2 = 10\mathbf{e}_1 + 5\mathbf{e}_2$ and $M_2 = 3$, as well as $\boldsymbol{P}_3 = 2\mathbf{e}_1 + 5\mathbf{e}_2$ (all other external loads are zero). Using as reference the starter code below (cf. "fill me here" comments for missing items) do the following:

1. Write a function for the local stiffness matrix of a frame element (combined axial and bending), following the template below
   `def local_stiffness_frame(elts,crds,e) .`

2. Assemble the global stiffness matrix and load vector.

3. Knowing that $\boldsymbol{w}_1 = \mathbf{0}$ and $\theta_1 = 0$ as well as $\boldsymbol{m}_S = -cos(\pi/4)\mathbf{e}_1 + \sin(\pi/4)\mathbf{e}_2$, apply the appropriate boundary conditions.

4. Solve for the displacements and rotations.

5. Plot the deformed shape of the frame.

```
elements = {}
elements[ ] = {'A': , 'E':  , 'I': , 0: , 1: } #fill me here

def local_stiffness_truss(elts,crds,e):
A, E = (elts[e]['A'],elts[e]['E'])
# Compute the director vector between the nodes
n = crds[elts[e][1]] - crds[elts[e][0]]
# Compute the lenght of the element
L = la.norm(n)
# Normalize the director vector
n /= L
# Compute the stiffness tensor
k = A*E/L*np.outer(n,n)
# The individual tensor entries of the local element stiff
ke = np.array([[k,-k],[-k,k]])
# Resize it
space_dim = n.size # the space dimensions
n_nodes = 2 # the number of nodes
n_dof = space_dim*n_nodes # the number of element degrees of freedom
ke = ke.reshape(n_dof, n_dof )
return ke
```

```
# Assemble the local stiffness matrix for a frame element
def local_stiffness_beam(elts,crds,e):
# Get element properties
A, E, I = (elts[e]['A'],elts[e]['E'],elts[e]['I'])
# Compute the director vector between the nodes
n = crds[elts[e][1]] - crds[elts[e][0]]
# Compute the lenght of the element
L = la.norm(n)
# Normalize the director vector
n /= L
# Define the rotation operation
R = np.array([[0,-1],[1,0]])
# Compute normal
s = # fill me here
# Compute the coefficients
Kfw =  # fill me here
kmt =  # fill me here
khmt =  # fill me here
kmw =  # fill me here
kft =  # fill me here
# The elt stiffness
space_dim = n.size # the space dimensions
n_nodes = 2 # the number of nodes
n_dof = space_dim*n_nodes + n_nodes # the number of element degrees of freedom
Ke = np.zeros((n_dof,n_dof))
# Add contributions
# The Diagonal blocks
Ke[0:2,0:2] = Ke[3:5,3:5] = Kfw
Ke[2,2] = Ke[5,5] = kmt
# The upper triangular portion
Ke[0:2,2] = Ke[0:2,5] =  kft
Ke[0:2,3:5] = Ke[0:2,3:5] = -Kfw
Ke[2,3:5] = -kmw
Ke[2,5] = khmt
Ke[3:5,5] = -kft
# Copy upper triangular to lower triangular (it's a symmetric matrix)
lower_indeces = np.tril_indices(n_dof,-1)
Ke[lower_indeces] = 0
Ke += np.triu(Ke,1).T
return Ke
```