Challenge Problem

The goal is to create an object-oriented truss analysis program that

1. is based on the finite element method
2. handles arbitrary trusses in 2D
3. allows for arbitrary loading
4. can plot its undeformed and deformed shape

In order to succeed, we shall devide the task and conquer one class per team. We will use Friday's session to combine your components into a single application.

The program will need and use the following objects:

Node

Each instance represents one node in the system

Node class methods

method	input	returns	description
__init__(x,y)	coordinates of the point as two floats		constructor. Sets position and initializes displacement and force to zeros.
fixDOF(idx)	*idx* of the degree of freedom (dof)		set internal flag for this dof accordingly.
isFixed(idx)	*idx* of the degree of freedom (dof)	True|False	test function returning True if dof at *idx* is fixed, False otherwise.
setDisp(u,v)	components of displacement		overwrited the displacements for this node.
getDisp()		*np.array([u,v])*	returns displacement vector
getPos()		*np.array([x,y])*	returns initial position
getDeformedPos(factor)		*np.array([x+factor*u,y+factor*v])*	returns current position with displacement magnified by factor. Would be good to have a default factor of 1.0 if none given.
addLoad(Px,Py)	components of load		add this load to nodal load
setLoad(Px,Px)	components of load		replace current load by provided load
getLoad()		*np.array([Px,Py])*	returns current load

Node class variables

name	type	description
pos	*np.array([x,y])*	holds *x* and *y* coordinates of the points
index	*int*	index position of this *Node()* in a *System().nodes* list. This needs to be set by *System().addNode(thisNode)*, so coordinate this with the *System()* team.
disp	*np.array([u,v])*	holds *x* and *y* components of nodal displacement
fixity	list of two *True*|*False*	*fixity[i]* is *True* if *i*-th degree of freedom is fixed, *False* otherwise. Note: *i=0*|*1*
force	*np.array([Px,Py])*	holds *x* and *y* components of the nodal load vector.

Equations

1. Deformed position node 0: $\mathbf{x}_0 = \mathbf{X}_0 + (factor) * \mathbf{U}_0$
2. Deformed position node 1: $\mathbf{x}_1 = \mathbf{X}_1 + (factor) * \mathbf{U}_1$

Element

Each instance represents one truss member

Element class methods

method	input	returns	description
__init__(nd0, nd1, material)	two *Node()* objects, one *Material()* object.		constructor.
getForce()		list of (1d) np.array objects	A list of nodal forces. Each nodal force shall be represented as a 1d np.array with two components of the force.
getStiffness()		2-by-2 list of 2-by-2 np.array	A list of lists (matrix) containing nodal tangent matrices as *np.array([[.,.],[.,.]])*

Note: a *Node()* object may have changed its state between calls, so you need to recompute every time!

Element class variables

name	type	description
nodes	List of *Node()* instances	representing the two nodes at either end of a truss.
material	*Material()*	pointer to an instance of *Material()*. Needed to compute stress and tangent modulus.
force	2-list of (1d) np.array objects.	holding nodal forces *P0* and *P1*
Kt	2-by-2 array of 2-by-2 np.array objects.	tangent stiffness matrix. Representing all nodal stiffness matrices.

Equations

1. $\mathbf{L} = \mathbf{X}_1 - \mathbf{X}_0$
2. $\ell = ||\mathbf{L}||$
3. $\mathbf{n} = \frac{1}{\ell}\,\mathbf{L}$

4. Strain: $\varepsilon = \frac{1}{\ell}\,\mathbf{n}\cdot(\mathbf{U}_1 - \mathbf{U}_0)$
5. Force: $f = \sigma(\varepsilon) A$ using *material.setStrain(eps)* and *material.getStress()*.
6. Nodal force vector: $\mathbf{P}^e = f\,\mathbf{n}$
7. $\mathbf{P}_0 = -\mathbf{P}^e \qquad \mathbf{P}_1 = \mathbf{P}^e$
8. Nodal stiffness matrix: $\mathbf{k}^e = \frac{E_t(\varepsilon)\;A}{\ell}\;\mathbf{n}\otimes\mathbf{n}$ using *material.getStiffness()* to find E_t.
9. $\mathbf{k}_{00} = \mathbf{k}_{11} = \mathbf{k}^e \qquad \mathbf{k}_{01} = \mathbf{k}_{10} = -\mathbf{k}^e$

Material

This class is provided as a demonstration example.

Material class methods

method	input	returns	description
__init__(...)	parameters as *{'E':10.0}*		constructor. Sets parameters for this material and initializes all internal variables
getArea()		A	return cross section area from *parameters['A']*
getStress()		σ	request axial stress
getStiffness		E_t	request axial stiffness
setStrain(eps)	strain ε		update state for a user provided axial strain value

Element class variables

name	type	description
params	dict	default parameters: *{'E':100., 'nu':0.0, 'fy:1.0e30}* Holds user provided parameters (MOE, Poisson's ratio, yield stress)
plastic_strain	float	internal state variable.
sig	float	holds current stress
Et	float	holds current materil tangent modulus

Equations

1. Elastic trial state:

   1.1 $\sigma = E * (\varepsilon - \varepsilon_P)$

   1.2 $E_t = E$

2. Yield check: $f = ||\sigma|| - f_y$
3. IF $f \geq 0$:

   3.1. $\Delta\varepsilon_P = \mathrm{sign}(\sigma) * \frac{f}{E}$

   3.2. $\sigma = \sigma - E * \Delta\varepsilon_P$

   3.3. $E_t = E_t - E$

System

Creates an instance of a truss model

System class methods

method	input	returns	description
__init__(...)			constructor.
addNode(newNode)	*Node(...)* object		add one *Node()* object to your list of elements (the model)
addElement(newElem)	*Element(...)* object		add one *Element()* object to your list of elements (the model)
solve()			assemble $[K_t]$ and $\{P\}$, solve for $\{u\} = [K_t]^{-1}\{P\}$, loop through nodes and update nodal displacement, compute unbalanced force $\{R\} = \{P\} - \{F\}$
plot(factor=1.0)			collect node info and send it to the plotter. Request the plot.
report()			print a summary report: list of nodal position, load, displacement, unbalanced force.

System class variables

name	type	description
nodes	List of *Node()* objects	holds all the nodes in the model
elements	List of *Element()* objects	holds all the elements in the model
plotter	*Plotter()*	pointer to *Plotter()* object to handle plotting
disp	*np.array([...])*	system sized displacement vector
loads	*np.array([...])*	system sized load vector

Equations

1. each element has *node0 = elem.nodes[0]* and *node1 = elem.nodes[1]*
2. node indices *i = node0.index* and *j = node1.index*
3. a local d.o.f. $u \to k = 0$ or $v \to k = 1$ of node i belongs at global index $K = 4 * i + k$
4. a local d.o.f. $u \to m = 0$ or $v \to m = 1$ of node j belongs at global index $M = 4 * j + m$

Assembly:

- $\mathbf{F}$: element force from *elem.getForce()*
- $\mathbf{K_t}$: element stiffness from *elem.getStiffness()*
- $\mathbf{R}_{sys}$: system force
- $\mathbf{K_t}_{sys}$: system stiffness

5. Loop over *nodes*: $\mathbf{R}_{sys}[K] = nodes[i].getLoad()[k]$ (this should return 0 if no load at this node and d.o.f.)
6. Loop over *elements*: $\mathbf{R}_{sys}[K] = \mathbf{R}_{sys}[K] - \mathbf{F}[i][k]$
7. Loop over *elements*: $\mathbf{K_t}_{sys}[K, M] = \mathbf{K_t}_{sys}[K, M] + \mathbf{K_t}[i, j][k, m]$
8. Loop over *nodes*: if a d.o.f. at *K* is fixed, set $\mathbf{R}_{sys}[K] = 0$ and $\mathbf{K_t}_{sys}[K, K] = 1.0e20$.
9. Solve system of equations: $\mathbf{U} = \mathbf{K_t}_{sys}^{-1}\mathbf{R}_{sys}$
10. Loop over *nodes*: $nodes[i].setDisp(u, v)$ using $u = \mathbf{U}[2 * i]$ and $v = \mathbf{U}[2 * i + 1]$
11. Recompute: $\mathbf{R}_{sys}$ as in steps 5 and 6 (**do not repeat steps 7-11**). If everything was done correctly, fixed d.o.f.s will contain the support reactions and free d.o.f.s should hold numeric zeros.

Plotter

Creates undeformed and deformed plots of the system.

Plotter class methods

method	input	returns	description
__init__()			constructor. Initialize the plotter object to sensible default settings, as needed.
setMesh(verts,lines)	list of points, list of line indices		replace *self.vertices* and *self.lines* information.
setDisplacements(disp)	list of displacement vectors		replace *self.disp* information.
setValues(vals)	list of line (force) values.		replace *self.values* information.
displacementPlot(file=None)	a string		creates a plot showing undeformed in black and deformed model in red lines. If *file* is given, save a copy of the plot to a file of that name
valuePlot(deformed=False, file=None)	a string		creates a plot showing the undeformed|deformed system (based on the user input) with lines colored based on *values*. Add a colormap/colorbar as legend. If *file* is given, save a copy of the plot to a file of that name

Plotter class variables

name	type	description
vertices	List of *np.array([X,Y])*	list of coordinate pairs representing points (nodes in the model)
lines	List of List	list of 2-element lists of indices. The two lists shall contain the indices of the start and end point of a line in the *vertices list*, respectively.
disp	list of *np.array([u,v])*	list of point displacements for deformed plot. This list must be of identical shape as the *vertices* list such that respective entries represent point position and displacement, respectively.
values	*np.array([...])*	list containing the force values for each line (element). This list must be of identical shape as the *lines* list.