```
#By Joshua Herman
#For INTRO TO FEM class Binghamton Continuing Ed
#Instructor:Mahdi Farahikia
#Program is slight modification of Assignment 2
import numpy as np
import sympy as sym
import pandas as pd



def create_title(unit_tuples_dict,problem_title,problem_type):
    """unit_tuples_dict={1:("y displacements d","inches"),2:("z displacements","radians")}
       problem_title='HW Part 2',
       problem type='FEA FRAME ANALYSIS'"""
    Title="%s: %s :\n"%(problem_type,problem_title)
    spaces="\n"+"   "*3
    for i in range(1,1+len(unit_tuples_dict)):
        Title+=spaces+"All %s in %s"%(unit_tuples_dict[i][0],unit_tuples_dict[i][1])
    return Title
def create_TRUSS_TITLE(dpsifm_units,problem_title):
    unit_tuples_dict={1:('x/y displacements d',dpsifm_units[0]),\
                     2:('x/y forces f',dpsifm_units[1])}
    problem_type="FEA TRUSS ANALYSIS"

    return create_title(unit_tuples_dict,problem_title,problem_type)

def print_Matrix(K):
    Kdivider=np.mean(np.abs(K))
    if Kdivider==0:
        Kdivider=1
        print("Warning: Matrix contains only zeros")
    Kdivider=(10**np.ceil(np.log10(Kdivider)))
    dK=pd.DataFrame(np.round(K/Kdivider,decimals=5))

    print(np.format_float_scientific(Kdivider),' *\n')
    print(dK)
def FEA_TRUSS(E_list,A_list,L_list,thetalist,M,fdpairs,title="FEA TRUSS ANALYSIS"):
    """
    E_list is list of Elastic Moduli E one for each element. Must be entered
        as array where ith array element is E for ith finite element
    A_list is list of cross sectional areas for each element input in manner describe above
    L_list is list of lengths for each element input in manner as described
        above
    thetalist is list of angles for each element (angle measured from global x axis to elemental x axis,
        where origin of element is at node 1 of element)
    M is Matrix Containing Nodal Information. Must be input as array. ith element of eth row of
        array is global node number for ith local node of eth element

    fdpairs is dictionary containing known values of forces/displacements for each
        global node of frame. Dictionary entry must be in following format:
        {1:[[3,'d'],[4,'f']]} means 1st global node has its global x displacement (zeroth index) fixed at 3,global y force(first index)
fixed at force fixed at 4.
         Dictionary element force or displacement must
        be specified for all global nodes with no nodes referenced twice.  """
    thetalist=thetalist/180*np.pi
    keval_list=E_list*A_list/L_list#Element stiffness calculation
    fdpairs0=fdpairs.copy()
    fdpairs=[]
    for e in range(1,len(fdpairs0)+1):
        fdpairs.append(fdpairs0[e][0])
        fdpairs.append(fdpairs0[e][1])

    if len(fdpairs)!=2*np.max(M):
        raise Exception('Error: Number of dofs to be specified must equal twice the number of global nodes')

    #Collect indicial positions of known displacement/forces .ALso collect dof values of known forces in f1 and known displacemnets in d1
    dknownindices=[]
    fknownindices=[]
    d1=[]
    f2=[]
    for i in range(len(fdpairs)):
        if fdpairs[i][2-1]=='d':
            dknownindices.append(i)
            d1.append(fdpairs[i][1-1])
        elif fdpairs[i][2-1]=='f':
            fknownindices.append(i)
            f2.append(fdpairs[i][1-1])
        else:
            raise Exception('Error: fdpairs second value in each pair must be either "f" or "d"')
    d1=np.array([d1]).T
    f2=np.array([f2]).T
    #B: Nodal DOF Matrix (Bei=Global degree of freedom(DOF)of ith local DOF of eth element. Each Node has two dofs)
    B=np.vstack([np.array([2*M[e-1,1-1]-1,2*M[e-1,1-1],2*M[e-1,2-1]-1,2*M[e-1,2-1]]) for e in range(1,len(M)+1)])
    N=len(B)#Number of elements


    #create symbolic angle variable
    theta,keval=sym.symbols('theta,keval')

    #T_hat_e: XY to xy coordinate rotation transformation matrix(from global to local CS) for one node
    T_hat_e=sym.Matrix([[sym.cos(theta),sym.sin(theta)],
                   [-sym.sin(theta),sym.cos(theta)]])#theta is CCW (counter clockwise)

    #T_e:Coordinate rotation transformation for one element (2 nodes)
    Te=sym.zeros(4,4)
    Te[1-1:2,1-1:2]=T_hat_e
    Te[3-1:4,3-1:4]=T_hat_e

    #K_hat_e:Local stiffness matrix for all for dof of element e
    K_hat_e=sym.zeros(4,4)
    K_hat_e[1-1,1-1]=1
    K_hat_e[3-1,1-1]=-1
    K_hat_e[1-1,3-1]=-1
    K_hat_e[3-1,3-1]=1
    K_hat_e*=keval

    #Ke: Global stiffness matrix for all dof of element e
    Ke=Te.T*K_hat_e*Te

    #list of Global stiffness matrices for each element
    Kelist=[np.array(Ke.subs({theta:thetalist[e-1],keval:keval_list[e-1]})).astype(np.float) for e in range(1,N+1)]

    #Assemble stiffness matrix K for global DOFs
    Ndof=np.max(B)
    K=np.zeros((Ndof,Ndof))
    for e in range(1,N+1):
        for i in range(1,4+1):
            for j in range(1,4+1):
                K[B[e-1,i-1]-1,B[e-1,j-1]-1]+=Kelist[e-1][i-1,j-1]
        #i corresponds to ith force of eth element, j corresponds to jth displacement of eth element
        #Bei and Bej corespond to corresponding global dofs.Local dof matrices are effectively converted
        # to corresponding global dof matrix of same element and then all added together to get the total
        # global dof stiffnes matrix

    #Below is the stiffness matrix separated into 4 parts. 1 stands for known displacement and unkown force, 2 stands for known force and
uknown displacement
    K11=K[dknownindices][:,dknownindices]
    K12=K[dknownindices][:,fknownindices]
    K21=K[fknownindices][:,dknownindices]
    K22=K[fknownindices][:,fknownindices]

    #Above 4 matrices are used to calculate global dofs
    d2=np.linalg.solve(K22,f2-K21.dot(d1))
    f1=K11.dot(d1)+K12.dot(d2)

    #Combine calculated dofs plus given variable values into vectors d and f
    d=np.zeros((Ndof,1))
    f=np.zeros((Ndof,1))

    d[dknownindices]=d1
    d[fknownindices]=d2
    f[dknownindices]=f1
    f[fknownindices]=f2

    #Now assign displacements to indivual elements
    de_list=[np.array([[d[B[e,1-1]-1,0],d[B[e,2-1]-1,0],d[B[e,3-1]-1,0],d[B[e,4-1]-1,0]]]).T for e in range(N)]

    #Calculate local coordinate displacements (d_hat_e=T_e*d_e)
    d_hat_e_list=[np.array(Te.subs({theta:thetalist[e]})).astype(np.float).dot(de_list[e]) for e in range(N)]

    #Next calculate local coordinate stresses (sigma_hat_e=K_hat_e[3,:]*d_hat_e/A)
    sigma_hat_e_list=np.array([[(np.array(K_hat_e.subs({keval:keval_list[e]})).astype(np.float)[[3-1]]).dot(d_hat_e_list[e])[0,0] for e
in range(N)]]).T/A_list

    #Next Calculate local coordinate forces
    f_hat_e_list=[(np.array(K_hat_e.subs({keval:keval_list[e]})).astype(np.float)).dot(d_hat_e_list[e]) for e in range(N)]

    #Print Input Data for FEA Problem[(E,A,L,theta) for each element
    thetalist=np.round(thetalist*180/np.pi,4)
    separatorstr="-------------------------------------"
    print((separatorstr*2+'\n')*3)
    print(title)
    print("\n"+separatorstr*1)
    print(separatorstr)
    print("\nElemental Input Data:")

    for e in range(N):
        print("\nInput Data for Element %s:"%(e+1))
        print("E%s = %s"%(e+1,np.format_float_scientific(E_list[e][0])))
        print("A%s = %s"%(e+1,np.format_float_scientific(A_list[e][0])))
        print("L%s = %s"%(e+1,np.format_float_scientific(L_list[e][0])))
        print("theta%s = %s"%(e+1,thetalist[e]))
    #Print Input Fixed DOF data for each Node
    print(separatorstr)
    print("Fixed DOFs")
    j=0
    for i in range(len(fdpairs)):

        if i%2==0:
            j+=1
            print('%s%sx = %s'%(fdpairs[i][1],j,fdpairs[i][0]))
        else:

            print('%s%sy = %s'%(fdpairs[i][1],j,fdpairs[i][0]))

    #Print Nodal connectivity data for each Node
    print(separatorstr)

    print("Below is Chart containing Nodal Connectivity Data.\n")
    print("The ith entry of the eth row represents the global node number associated with the ith local node \nof the eth finite
element")
    dM=pd.DataFrame(M)


    display(dM)
    #Print Global coordinate Stiffness Matrix for each element
    print(separatorstr)
    print("\nGlobal coordinate stiffness Matrices:")
    for e in range(len(Kelist)):
        print("\nGlobal Coordinate Stiffness Matrix for element %s:\n"%(e+1))
        print_Matrix(Kelist[e])
        print("\n")

    #Print Total Global coordinate Stiffness Matrix
    print(separatorstr)
    print("Total Stiffness Matrix:\n")
    Kdivider=np.mean(keval_list)
    print_Matrix(K)

    #Print Global Node Forces and displacements
    print(separatorstr)
    print("\nBelow are Global Node Forces:\n")


    for i in range(len(f)):
        if i%2==0:
            print('fx%s = %s'%(int((i/2+1)),np.format_float_scientific(f[i][0],precision=4)))
        elif i%2==1:
            print('fy%s = %s'%(int((i+1)/2),np.format_float_scientific(f[i][0],precision=4)))
    print(separatorstr)
    print("\nBelow are Global Node Displacements:\n")
    for i in range(len(d)):
        if i%2==0:
            print('dx%s = %s'%(int((i/2+1)),np.format_float_scientific(d[i][0],precision=4)))
        else:
            print('dy%s = %s'%(int((i+1)/2),np.format_float_scientific(d[i][0],precision=4)))

    print(separatorstr)
    print("\nBelow are the FEA results by element in local coordinates:\n")

    for e in range(N):
        print("Below are local coordinate forces, displacements, and axial stresses for Element %s:"%(e+1))
        print("f_%s_x%s = %s"%(e+1,1,np.format_float_scientific(f_hat_e_list[e][1-1,0],precision=4)))
        print("f_%s_y%s = %s"%(e+1,1,np.format_float_scientific(f_hat_e_list[e][2-1,0],precision=4)))
        print("f_%s_x%s = %s"%(e+1,2,np.format_float_scientific(f_hat_e_list[e][3-1,0],precision=4)))
        print("f_%s_y%s = %s"%(e+1,2,np.format_float_scientific(f_hat_e_list[e][4-1,0],precision=4)))
        print("\n")
        print("d_%s_x%s = %s"%(e+1,1,np.format_float_scientific(d_hat_e_list[e][1-1,0],precision=4)))
        print("d_%s_y%s = %s"%(e+1,1,np.format_float_scientific(d_hat_e_list[e][2-1,0],precision=4)))
        print("d_%s_x%s = %s"%(e+1,2,np.format_float_scientific(d_hat_e_list[e][3-1,0],precision=4)))
        print("d_%s_y%s = %s"%(e+1,2,np.format_float_scientific(d_hat_e_list[e][4-1,0],precision=4)))
        print('\n')
        print("sigma_%s = %s"%(e+1,np.format_float_scientific(sigma_hat_e_list[e],precision=4)))
        print('\n')
    print((separatorstr*2+"\n")*3)
```