```python
#By Joshua Herman
#For INTRO TO FEM class Binghamton Continuing Ed
#Instructor:Mahdi Farahikia
#Code From Assignment 4
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_FRAME_TITLE(dpsifm_units,problem_title):
    unit_tuples_dict={1:('x/y displacements d',dpsifm_units[0]),2:('z displacements psi',dpsifm_units[1]),\
                     3:('x/y forces f',dpsifm_units[2]),4:("z forces m",dpsifm_units[3])}
    problem_type="FEA FRAME 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)
x_hat=sym.symbols("x_hat")
def FEA_Frame(E_list,A_list,I_list,L_list,theta_list,q_sym_list,M,fdpairs,title="FEA FRAME 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 described above
    I_list is list of 2nd Area Moments for each element input in manner describe above
    L_list is list of lengths for each element input in manner as described
        above
    theta_list is list of ccw angles from global x axis to local x axis for each element
    q_sym_list contains a list of sympy functions where ith element represent distributed load for ith 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'],[0,'d']]} 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,
        while global z displacement (second index) (angular) displacement fixed at 0. Dictionary element force or displacement must
        be specified for all global nodes with no nodes referenced twice.  """
    #Inputs dictionary
    inputs={"E_list":E_list,"A_list":A_list,"I_list":I_list,"L_list":L_list,"theta_list":theta_list,\
          "q_sym_list":q_sym_list,"M":M,"fdpairs":fdpairs,"title":title}

    #Create results and calculations dictionary
    calculations={}
    results={}


    #Create symbolic variables
    L,keval_bar,keval_beam,xbar,theta=sym.symbols('L,keval_bar,keval_beam,xbar,theta')

    #convert theta to radians
    theta_list*=np.pi/180


    keval_beam_list=E_list*I_list;keval_bar_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])
        fdpairs.append(fdpairs0[e][2])

    if len(fdpairs)!=3*np.max(M):
        raise Exception('Error: Number of dofs to be specified must equal thrice 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

    #add to dictionary
    calculations['dknownindices']=dknownindices
    calculations['fknownindices']=fknownindices
    calculations['d1']=d1
    calculations['f2']=f2


    #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([3*M[e-1,1-1]-2,3*M[e-1,1-1]-1,3*M[e-1,1-1],3*M[e-1,2-1]-2,3*M[e-1,2-1]-1,3*M[e-1,2-1]]) for e in
range(1,len(M)+1)])
    N=len(B)#Number of elements

    #add to dictionary
    calculations['B']=B
    calculations['N']=N




    #Beam,bar to frame row/column multiplication matrices
    Mbeam=sym.Matrix([[0,0,0,0],#To convert Beam local stiffness matrix to frame local stiffness
             [1,0,0,0],
             [0,1,0,0],
             [0,0,0,0],
             [0,0,1,0],
             [0,0,0,1]])
    Mbar=sym.Matrix([[1,0],#convert bar local stiffness matrix to frame local stiffness
                    [0,0],
                    [0,0],
                    [0,1],
                    [0,0],
                    [0,0]])





    #K_hat_e:Local stiffness matrix for all for dof of element e
    K_hat_e_beam=1/L**3*sym.Matrix([[12,6*L,-12,6*L],
                        [6*L,4*L**2,-6*L,2*L**2],
                        [-12,-6*L,12,-6*L],
                        [6*L,2*L**2,-6*L,4*L**2]])
    K_hat_e_beam=Mbeam*K_hat_e_beam*Mbeam.T
    K_hat_e_beam*=keval_beam

    K_hat_e_bar=sym.Matrix([[1,-1],
                           [-1,1]])
    K_hat_e_bar=Mbar*K_hat_e_bar*Mbar.T
    K_hat_e_bar*=keval_bar
    K_hat_e=K_hat_e_beam+K_hat_e_bar#superimpose bar and beam stiffness matrices to get frame

    #List of local stiffness matrices for all dof of element e
    K_hat_e_list=[np.array(K_hat_e.subs({L:L_list[e-1],keval_beam:keval_beam_list[e-1],keval_bar:keval_bar_list[e-1]})).astype(np.float)
for e in range(1,N+1)]
    #add to dictionary
    calculations["K_hat_e_list"]=K_hat_e_list

    #Transformation Matrix T
    Tbar=sym.Matrix([[sym.cos(theta),sym.sin(theta),0],
                    [-sym.sin(theta),sym.cos(theta),0],
                    [0,0,1]])
    T=sym.zeros(6,6)
    T[1-1:3,1-1:3]=Tbar
    T[4-1:6,4-1:6]=Tbar
    #Ke: Global stiffness matrix for all dof of element e
    Ke=T.T*K_hat_e*T

    #list of Global stiffness matrices for each element
    Kelist=[np.array(Ke.subs({L:L_list[e-1],theta:theta_list[e-1],keval_beam:keval_beam_list[e-1],keval_bar:keval_bar_list[e-
1]})).astype(np.float) for e in range(1,N+1)]
    Telist=[np.array(T.subs({theta:theta_list[e-1]})).astype(np.float) for e in range(1,N+1)]
    #Add to dictionary
    calculations["Kelist"]=Kelist
    calculations["Telist"]=Telist

    #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(1,6+1):
            for j in range(1,6+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
    #Add to Dictionary
    calculations["Ndof"]=Ndof
    calculations["K"]=K

    #Equivalent Nodal Forces
    def equivalent_nodal_forces(q):
        if q!=0:
            q*=x_hat**0
            N=sym.Matrix([[2*xbar**3-3*xbar**2+1,
                        L*(xbar**3-2*xbar**2+xbar),
                        -2*xbar**3+3*xbar**2,
                        L*(xbar**3-xbar**2)]]) #Shape Functions for beam where xbar=x_hat/L

            f0_hat_e=Mbeam*-L*sym.integrate(q.subs({x_hat:L*xbar})*N.T,(xbar,0,1))#Equivalent Nodal Forces
        else:
            f0_hat_e=sym.zeros(6,1)
        return f0_hat_e

    #CALCULATE EQUIVALENT NODAL FORCES FOR EACH ELEMENT.THEN CONVERT TO GLOBAL COORDINATES

    f_hat_0_e_list=[np.array(equivalent_nodal_forces(q_sym_list[i]).subs({L:L_list[i]})).astype(np.float)  for i in range(N)]
    f_0_e_list=[Telist[e-1].T.dot(f_hat_0_e_list[e-1]) for e in range(1,N+1)]
    #add to dictionary
    calculations["f_hat_0_e_list"]=f_hat_0_e_list
    calculations["f_0_e_list"]=f_0_e_list

    #Assemble equivalent nodal forces vector f0 for global DOFs

    f0=np.zeros((Ndof,1))
    for e in range(1,N+1):
        for i in range(1,6+1):
            f0[B[e-1,i-1]-1,0]+=f_0_e_list[e-1][i-1,0]
        #i corresponds to ith force of eth element
        #Beicoresponds to corresponding global dof.Local equivalent distributed force vector is effectively converted
        # to corresponding global vector of same element and then all added together to get the total
        # global equivalent distributed force vector
    #add to dictionary
    calculations['f0']=f0

    #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]
    #add to dictionary
    calculations['K11']=K11
    calculations['K12']=K12
    calculations['K21']=K21
    calculations['K22']=K22

    #Below same is done for equivalent nodal force vector
    f01=f0[dknownindices]
    f02=f0[fknownindices]
    #add to dictionary
    calculations['f01']=f01
    calculations['f02']=f02

    #Above 4 matrices are used to calculate global dofs
    d2=np.linalg.solve(K22,(f2+f02)-K21.dot(d1))
    f1=K11.dot(d1)+K12.dot(d2)-f01
    #add to dictionary
    calculations['d2']=d2
    calculations['f1']=f1

    #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
    #add to dictionary
    results['d']=d
    results['f']=f

    #Now assign displacements to indivual elements
    d_e_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],d[B[e,5-1]-1,0],d[B[e,6-1]-1,0]]]).T for e in
range(N)]
    #add to dictionary

    results['d_e_list']=d_e_list
    #calculate global coordinate local forces
    f_eff_e_list=[Kelist[e].dot(d_e_list[e]) for e in range(len(d_e_list))]
    calculations['fe_list']=f_eff_e_list
    f_e_list=[f_eff_e_list[e]-f_0_e_list[e] for e in range(len(d_e_list))]
    results['f_e_list']=f_e_list

    #Calculate local coordinate displacements (d_hat_e=T_e*d_e)
    d_hat_e_list=[Telist[e].dot(d_e_list[e]) for e in range(N)]
    #add to dictionary
    results['d_hat_e_list']=d_hat_e_list

    #Next Calculate local coordinate forces
    f_hat_e_eff_list=[K_hat_e_list[e].dot(d_hat_e_list[e]) for e in range(N)]
    f_hat_e_list=[f_hat_e_eff_list[e]-f_hat_0_e_list[e] for e in range(N)]
    #add to dictionary
    calculations['f_hat_e_eff_list']=f_hat_e_eff_list
    results['f_hat_e_list']=f_hat_e_list

    #Print Input Data for FEA Problem (E,A,I,L,theta,q) for each element

    separatorstr="-------------------------------------"
    print((separatorstr*2+'\n')*3)
    print(title)
    print("\n"+separatorstr*1)
    print(separatorstr)
    print("\nElemental Input Data:")

    #convert theta to degrees
    theta_list*=180/np.pi

    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])))
        print("A%s = %s"%(e+1,np.format_float_scientific(A_list[e])))
        print("I%s = %s"%(e+1,np.format_float_scientific(I_list[e])))
        print("L%s = %s"%(e+1,np.format_float_scientific(L_list[e])))
        print("theta%s = %s"%(e+1,np.format_float_scientific(theta_list[e])))
        print("q%s = %s"%(e+1,np.format_float_scientific(q_sym_list[e])))
    #Print Input Fixed DOF data for each Node
    print(separatorstr)
    print("Fixed DOFs")
    j=0
    for i in range(len(fdpairs)):
        if i%3==0:
            j+=1

            if fdpairs[i][1]=='f':
                fdtype='f'
            else:
                fdtype='d'
            print('%s%sx = %s'%(fdtype,j,fdpairs[i][0]))
        elif i%3==1:

            if fdpairs[i][1]=='f':
                fdtype='f'
            else:
                fdtype='d'
            print('%s%sy = %s'%(fdtype,j,fdpairs[i][0]))
        else:
            if fdpairs[i][1]=='f':
                fdtype='m'
            else:
                fdtype='phi'
            print('%s%sz = %s'%(fdtype,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")

    print_Matrix(K)

    #Print Global equivalent Nodal Forces Vector for each element
    print(separatorstr)
    print("\nGlobal equivalent Nodal Forces Vectors:")
    for e in range(len(f_0_e_list)):
        print("\nGlobal equivalent Nodal Force Vector for element %s:\n"%(e+1))
        print(f_0_e_list[e])
        print("\n")

    #Print Total Global equivalent Nodal Forces Vector
    print(separatorstr)
    print("Total Global Equivalent Nodal Force Vector:\n")
    print(f0)

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

    j=0
    for i in range(len(f)):
        if i%3==0:
            j+=1
            print('fx%s = %s'%(int(j),np.format_float_scientific(f[i][0],precision=4)))
        elif i%3==1:
            print('fy%s = %s'%(int(j),np.format_float_scientific(f[i][0],precision=4)))
        elif i%3==2:
            print('mz%s = %s'%(int(j),np.format_float_scientific(f[i][0],precision=4)))
    print(separatorstr)
    print("\nBelow are Global Node Displacements:\n")
    j=0
    for i in range(len(d)):
        if i%3==0:
            j+=1
            print('dx%s = %s'%(int(j),np.format_float_scientific(d[i][0],precision=4)))
        elif i%3==1:
            print('dy%s = %s'%(int(j),np.format_float_scientific(d[i][0],precision=4)))
        else:
            print('phiz%s = %s'%(int(j),np.format_float_scientific(d[i][0],precision=4)))
    #Print Local Node Forces and displacemgents in global coordinate system

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

    for e in range(N):
        print("Below are global coordinate forces and displacements for Element %s:"%(e+1))
        print("f_%s_x%s = %s"%(e+1,1,np.format_float_scientific(f_e_list[e][1-1,0],precision=4)))
        print("f_%s_y%s = %s"%(e+1,1,np.format_float_scientific(f_e_list[e][2-1,0],precision=4)))
        print("m_%s_z%s = %s"%(e+1,1,np.format_float_scientific(f_e_list[e][3-1,0],precision=4)))
        print("f_%s_x%s = %s"%(e+1,2,np.format_float_scientific(f_e_list[e][4-1,0],precision=4)))
        print("f_%s_y%s = %s"%(e+1,2,np.format_float_scientific(f_e_list[e][5-1,0],precision=4)))
        print("m_%s_z%s = %s"%(e+1,2,np.format_float_scientific(f_e_list[e][6-1,0],precision=4)))
        print("\n")
        print("d_%s_x%s = %s"%(e+1,1,np.format_float_scientific(d_e_list[e][1-1,0],precision=4)))
        print("d_%s_y%s = %s"%(e+1,1,np.format_float_scientific(d_e_list[e][2-1,0],precision=4)))
        print("phi_%s_z%s = %s"%(e+1,1,np.format_float_scientific(d_e_list[e][3-1,0],precision=4)))
        print("d_%s_x%s = %s"%(e+1,2,np.format_float_scientific(d_e_list[e][4-1,0],precision=4)))
        print("d_%s_y%s = %s"%(e+1,2,np.format_float_scientific(d_e_list[e][5-1,0],precision=4)))
        print("phi_%s_z%s = %s"%(e+1,2,np.format_float_scientific(d_e_list[e][6-1,0],precision=4)))
        print('\n')

        print('\n')


    #Print Local Node Forces and displacements in local coordinate system

    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 and displacements for Element %s:"%(e+1))
        print("f_hat_%s_x%s = %s"%(e+1,1,np.format_float_scientific(f_hat_e_list[e][1-1,0],precision=4)))
        print("f_hat_%s_y%s = %s"%(e+1,1,np.format_float_scientific(f_hat_e_list[e][2-1,0],precision=4)))
        print("m_hat_%s_z%s = %s"%(e+1,1,np.format_float_scientific(f_hat_e_list[e][3-1,0],precision=4)))
        print("f_hat_%s_x%s = %s"%(e+1,2,np.format_float_scientific(f_hat_e_list[e][4-1,0],precision=4)))
        print("f_hat_%s_y%s = %s"%(e+1,2,np.format_float_scientific(f_hat_e_list[e][5-1,0],precision=4)))
        print("m_hat_%s_z%s = %s"%(e+1,2,np.format_float_scientific(f_hat_e_list[e][6-1,0],precision=4)))
        print("\n")
        print("d_hat_%s_x%s = %s"%(e+1,1,np.format_float_scientific(d_hat_e_list[e][1-1,0],precision=4)))
        print("d_hat_%s_y%s = %s"%(e+1,1,np.format_float_scientific(d_hat_e_list[e][2-1,0],precision=4)))
        print("phi_hat_%s_z%s = %s"%(e+1,1,np.format_float_scientific(d_hat_e_list[e][3-1,0],precision=4)))
        print("d_hat_%s_x%s = %s"%(e+1,2,np.format_float_scientific(d_hat_e_list[e][4-1,0],precision=4)))
        print("d_hat_%s_y%s = %s"%(e+1,2,np.format_float_scientific(d_hat_e_list[e][5-1,0],precision=4)))
        print("phi_hat_%s_z%s = %s"%(e+1,2,np.format_float_scientific(d_hat_e_list[e][6-1,0],precision=4)))
        print('\n')

        print('\n')
    print((separatorstr*2+"\n")*3)
    return inputs,calculations,results
```