import numpy as np

from scipy.integrate import odeint

import matplotlib.pyplot as plt

class MultiDOF(object):

    '''

    Zeroth node is the relative fixed node.

    '''

    def \_\_init\_\_(self,Masses):

        self.Masses = Masses

        #self.MassTypes = MassTypes

        #self.SpringElementsK = SpringKs

        #self.DamperElementsC = DamperCs

        #self.SpringElements = SpringElements

        #self.LeverSpringElements = LeverSpringElements

        self.SpringCompOnlyType = []

        #self.DamperElements = DamperElements

        #self.LeverDamperElements = LeverDamperElements

        self.LiveK = []

        self.C = []

        self.M = []

        self.initX = []

        self.initXp = []

        self.DisplacementFunctions = []

        self.ForcingFunctions = []

        self.y = 0

        self.once = True

        ###init calcs

        self.n = len(self.Masses)

        #self.\_\_BuildK\_MatrixNew()

        #self.\_\_BuildC\_MatrixNew()

        self.\_\_BuildM\_Matrix()

    def \_\_BuildK\_MatrixNew(self):

        self.LiveK = np.zeros([self.n,self.n])

        for idx,i in enumerate(self.Masses):

            print('mass:',idx)

            line = [0]\*self.n

            if self.MassTypes[idx]=='M':

                for j in self.SpringElements:

                    if (j[1]==idx)or((j[2]==idx)):

                        line[j[1]] = line[j[1]]+self.SpringElementsK[j[0]]

                        line[j[2]] = line[j[2]]-self.SpringElementsK[j[0]]

                for j in self.LeverSpringElements:

                    if ((j[1][0]==idx)or((j[2][0]==idx))):

                        line[j[1][0]] = line[j[1][0]]+self.SpringElementsK[j[0]]

                        line[j[2][0]] = line[j[2][0]]-self.SpringElementsK[j[0]]

                        if(j[1][0]!=j[1][1]):

                            line[j[1][1]] = -line[j[1][1]]-self.SpringElementsK[j[0]]\*j[3]

                        if(j[2][0]!=j[2][1]):

                            line[j[2][1]] = -line[j[2][1]]-self.SpringElementsK[j[0]]\*j[4]

            if self.MassTypes[idx]=='J':

                for j in self.LeverSpringElements:

                    line[j[1][0]] = line[j[1][0]] + self.SpringElementsK[j[0]]\*j[3]

                    line[j[2][0]] = line[j[2][0]] + self.SpringElementsK[j[0]]\*j[3]

                    line[j[1][1]] = line[j[1][1]] + self.SpringElementsK[j[0]]\*j[3]\*\*2

                    line[j[2][1]] = line[j[2][1]] + self.SpringElementsK[j[0]]\*j[4]\*\*2

            print('line',line)

            self.LiveK[:][idx] = line

    def \_\_BuildC\_MatrixNew(self):

        self.C = np.zeros([self.n,self.n])

        for idx,i in enumerate(self.Masses):

            line = [0]\*self.n

            for j in self.DamperElements:

                if (j[1]==idx)or((j[2]==idx)):

                    line[j[1]] = line[j[1]]+self.DamperElementsC[j[0]]

                    line[j[2]] = line[j[2]]+self.DamperElementsC[j[0]]

            for j in self.LeverDamperElements:

                if (j[1]==idx)or((j[2]==idx)):

                    line[j[1]] = line[j[1]]+self.DamperElementsC[j[0]]\*j[3]

                    line[j[2]] = line[j[2]]+self.DamperElementsC[j[0]]\*j[4]

                #print('mass:',idx)

                #print('line',line)

            self.C[:][idx] = line

    def \_\_BuildK\_MatrixOld(self):

        self.LiveK = np.zeros([self.n,self.n])

        for idx,i in enumerate(self.SpringElementsK):

            ans = np.zeros([self.n,self.n])

            #print('this is the error number',idx)

            con = self.SpringElementsConnectivity[idx]

            ans[con[0],con[0]] = i\*con[2]\*con[3]

            ans[con[0],con[1]] = -i\*con[2]\*con[3]

            ans[con[1],con[0]] = -i\*con[2]\*con[3]

            ans[con[1],con[1]] = i\*con[2]\*con[3]

            self.LiveK  = self.LiveK + ans

    def BuildLiveK(self,X):

        self.LiveK = np.zeros([self.n,self.n])

        for idx,i in enumerate(self.SpringElementsK):

            if(self.SpringCompOnlyType[idx]==0)or((X[self.SpringElementsConnectivity[idx][1]]-X[self.SpringElementsConnectivity[idx][0]])<0):

                ans = np.zeros([self.n,self.n])

                con = self.SpringElementsConnectivity[idx]

                ans[con[0],con[0]] = i

                ans[con[0],con[1]] = -i

                ans[con[1],con[0]] = -i

                ans[con[1],con[1]] = i

                self.LiveK = self.LiveK + ans

    def \_\_BuildC\_Matrix(self):

        self.C = np.zeros([self.n,self.n])

        for idx,i in enumerate(self.DamperElementsC):

            ans = np.zeros([self.n,self.n])

            con = self.DamperElements[idx]

            ans[con[0],con[0]] = i

            ans[con[0],con[1]] = -i

            ans[con[1],con[0]] = -i

            ans[con[1],con[1]] = i

            self.C = self.C + ans

    def \_\_BuildM\_Matrix(self):

        self.M = np.zeros([self.n,self.n])

        for idx,i in enumerate(self.Masses):

            self.M[idx][idx] = i

    def DerivativeX(self,X,t):

        Xdis = X[0:self.n]

        Xvel = X[self.n:]

        XdisOld = Xdis

        XvelOld = Xvel

        if self.once:

            print('Xdis,Xvel',Xdis,Xvel)

        Xdis,Xvel = self.OverWriteDisplacement(Xdis,Xvel,t)

        if self.once:

            print('Xdis,Xvel',Xdis,Xvel)

        #self.BuildLiveK(Xdis)

        xd = Xvel

        term1 = np.matmul(-np.linalg.inv(self.M), self.LiveK)

        term2 = np.matmul(-np.linalg.inv(self.M), self.C)

        term3 = np.matmul(np.linalg.inv(self.M), self.ApplyForcingFuncitons(t))

        xdd = np.matmul(term1,Xdis) + np.matmul(term2,Xvel) + term3

        return np.append(xd,xdd)

    def SetSpringsToCompressionOnly(self,S):

        self.SpringCompOnlyType = S

    def SetInitialDisplacements(self,X):

        self.initX = np.array(X)

    def SetInitialVelocities(self,Xp):

        self.initXp = np.array(Xp)

    def SetConstraints(self,FuncDescribingX,nodenumber):

        '''

        FuncDescribingX must be a input function that spits out x,v(displacement,velocity) when given t

        '''

        if self.DisplacementFunctions==[]:

            self.DisplacementFunctions = [0]\*self.n

        self.DisplacementFunctions[nodenumber] = FuncDescribingX

    def SetForcingFunction(self, FuncDescribingF,NodeNumber):

        '''

        FuncDescribingF must take one input t and output a force for one node NodeNumber

        '''

        if self.ForcingFunctions==[]:

            self.ForcingFunctions = [0]\*self.n

        self.ForcingFunctions[NodeNumber] = FuncDescribingF

    def OverWriteDisplacement(self,X,Xp,t):

        #print(self.DisplacementFunctions)

        for idx,i in enumerate(self.DisplacementFunctions):

            if i!=0:

                X[idx],Xp[idx] = i(t)

            #print(X,',',Xp)

        return X,Xp

    def ApplyForcingFuncitons(self,t):

        F = [0]\*self.n

        for idx,i in enumerate(self.ForcingFunctions):

            if i!=0:

                F[idx]= i(t)

        return F

    def Labels(self):

        l = []

        for i in range(self.n):

            l.append('x' + str(i))

        return l

    def SolveSystem(self,t0,t1,res=1000):

        self.res = res

        t = np.linspace(t0,t1,res)

        InitialConditions = np.append(self.initX,self.initXp)

        #print('initial con:',InitialConditions)

        self.y = odeint(self.DerivativeX,InitialConditions,t)

## Nexst

from Vibration import \*

### This is a functiond describing the base exitiation.

raw\_data = np.load('X-Y random road.npy')

""" plt.figure()

plt.plot(raw\_data[0],raw\_data[1])

plt.show() """

speed = 40 #km.h

v = speed\*1000/3600

t2 = 1000/v

def BaseExitation1(t):

    findx = t\*v

    x = raw\_data[0]

    y = raw\_data[1]

    index1 = 0

    for idx, i in enumerate(x):

        if i >= findx:

            index1 = idx

            break

    dis = (y[index1-1])+((findx-x[index1-1])/(x[index1]-x[index1-1]))\*(y[index1]-y[index1-1])

    vel = v\*(y[index1]-y[index1-1])/(x[index1]-x[index1-1])

    return dis,vel

def BaseExitation2(t):

    findx = (t\*v)-1.5

    x = raw\_data[0]

    y = raw\_data[1]

    index1 = 0

    for idx, i in enumerate(x):

        if i >= findx:

            index1 = idx

            break

    dis = (y[index1-1])+((findx-x[index1-1])/(x[index1]-x[index1-1]))\*(y[index1]-y[index1-1])

    vel = v\*(y[index1]-y[index1-1])/(x[index1]-x[index1-1])

    return dis,vel

Masses = [1,1,50,50,500,400,56.8]

a = 1.2

b = 1.2

sys = MultiDOF(Masses)

k1 = 65000

k2= 65000

k3 = 32000

k4 = 32000

k5 = 754500

c1 = 250

c2 = 250

c3 = 7500

c4 = 7500

c5 = 3840

sys.LiveK = [

[k1,0,-k1,0,0,0,0],

[0,k2,0,-k2,0,0,0],

[-k1,0,k1+k3,0,-k3,a\*k3,0],

[0,-k2,0,k2+k4,-k4,-b\*k4,0],

[0,0,-k3,-k4,k3+k4+k5,a\*k3-b\*k4,-k5],

[0,0,a\*k3,-b\*k4,a\*k3-b\*k4,(a\*\*2)\*k3+(b\*\*2)\*k4,0],

[0,0,0,0,-k5,0,k5]

]

sys.C = [

[c1,0,-c1,0,0,0,0],

[0,c2,0,-c2,0,0,0],

[-c1,0,c1+c3,0,-c3,a\*c3,0],

[0,-c2,0,c2+c4,-c4,-b\*c4,0],

[0,0,-c3,-c4,c3+c4+c5,a\*c3-b\*c4,-c5],

[0,0,a\*c3,-b\*c4,a\*c3-b\*c4,(a\*\*2)\*c3+(b\*\*2)\*c4,0],

[0,0,0,0,-c5,0,c5]

]

sys.SetInitialDisplacements([0,0,0,0,0,0,0])

sys.SetInitialVelocities([0,0,0,0,0,0,0])

sys.SetConstraints(BaseExitation1,1)

sys.SetConstraints(BaseExitation2,0)

t0 = 0

t1 = 10

sys.SolveSystem(t0,t1,res=10000)

#sys.plot(0,5)

t = np.linspace(t0,t1,sys.res)

plt.figure(1)

displacements = sys.y[:,0:sys.n]

plt.title('Resonse of vehicle with offsets')

plt.plot(t,displacements[:,0], label = 'Bump rear')

plt.plot(t,displacements[:,1], label = 'Bump front')

plt.plot(t,displacements[:,2]+0.1, label = 'Rear wheel')

plt.plot(t,displacements[:,3]+0.1, label = 'Front wheel')

plt.plot(t,displacements[:,4]+0.2, label = 'Car Body')

plt.plot(t,displacements[:,5], label = 'Car Body rotation(rad)')

plt.plot(t,displacements[:,6]+0.3, label = 'Driver head')

plt.xlabel('time')

plt.ylabel('x(t)')

plt.legend()

plt.grid()

plt.figure(2)

displacements = sys.y[:,0:sys.n]

plt.title('Resonse of vehicle with no offsets')

plt.plot(t,displacements[:,0], label = 'Bump rear')

plt.plot(t,displacements[:,1], label = 'Bump front')

plt.plot(t,displacements[:,2], label = 'Rear wheel')

plt.plot(t,displacements[:,3], label = 'Front wheel')

plt.plot(t,displacements[:,4], label = 'Car Body')

plt.plot(t,displacements[:,5], label = 'Car Body rotation(rad)')

plt.plot(t,displacements[:,6], label = 'Driver head')

plt.xlabel('time')

plt.ylabel('x(t)')

plt.legend()

plt.grid()

k = np.array(sys.C)

print(k-k.T)

plt.show()