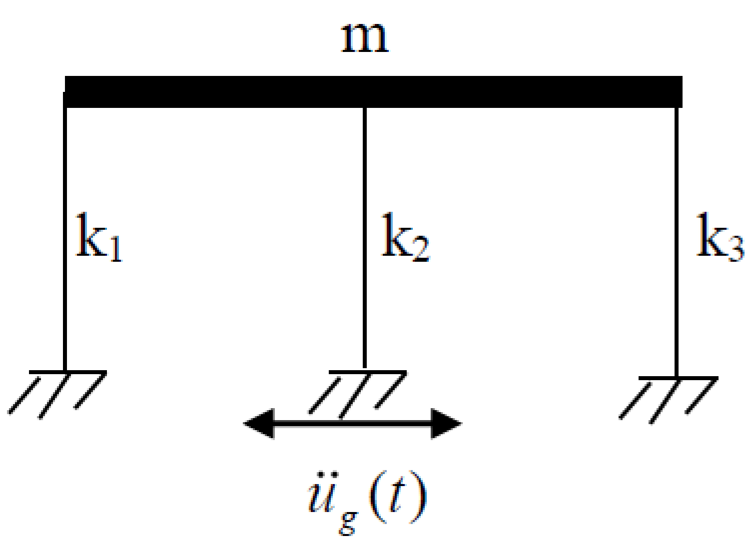
Single-story building with redundant columns

Final Project



Course: CE6205 Structural Dynamics & Earthquake Engineering

Instructor: Prof. Genda Chen

Student Name: Xinzhe Yuan

1. Introduction

A rigid-floor, single-story building of 2, 3, 4, or 5 columns has a total bending stiffness k of each building, regardless of the different number of columns. For each building, the column stiffness distribution factor is represented by γ ($\gamma = 0.1, 0.2, 0.3, 0.4$).

Case (1): 2 columns, $k_1 = k_2 = \frac{k}{2}$

Case (2): 3 columns, $k_1 = \frac{1-\gamma}{3}k, k_2 = \frac{k}{3}, k_3 = \frac{1+\gamma}{3}k$

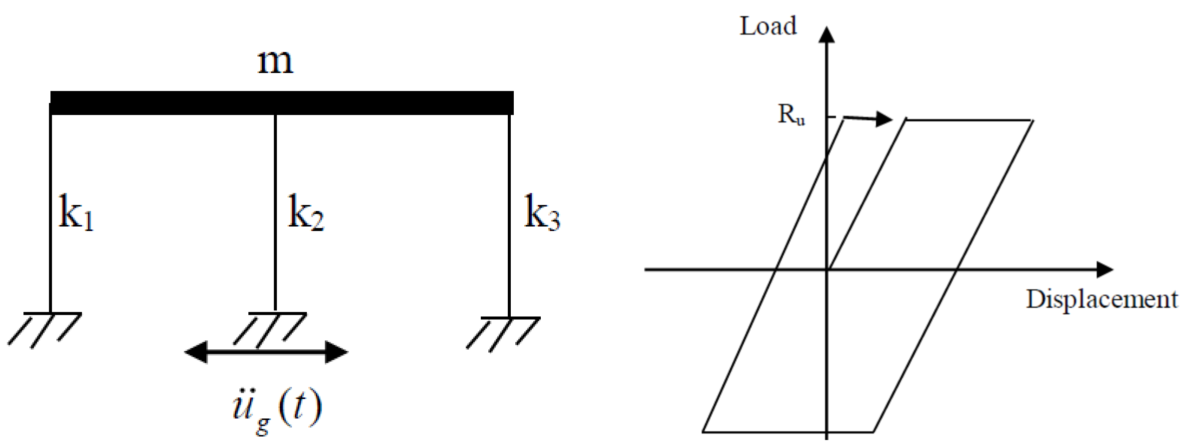
Case (3): 4 columns, $k_1 = \frac{1-\gamma}{4}k, k_2 = k_3 = \frac{k}{4}, k_4 = \frac{1+\gamma}{4}k$

Case (4): 4 columns, $k_1 = \frac{1-2\gamma}{5}k, k_2 = \frac{1-\gamma}{5}k,\ k_3 = \frac{k}{5}, k_4 = \frac{1+\gamma}{5}k, k_5 = \frac{1+2\gamma}{5}k$

The load is an impulsive ground velocity

$$\dot{u}_g(t) = se^{-\zeta_p \omega_p t} sin\omega_p \sqrt{1-\zeta_p^2} t$$

here $\zeta_p = 0.1$ is the damping factor of the decaying sinusoidal excitation, $\omega_p = 2\pi/T_p$ is the frequency of the sinusoid, and s is the initial amplitude of the velocity pulse. The building can be modeled with an elasto-perfectly-plastic bilinear load displacement curve as shown in Fig. 1(b). For each building, the yield force of every column is identical, totaling R_u. For example, each column has a yield force $R_u/5$ for the five-column building. A damping ratio of 5% for low-amplitude vibration is assumed.



(a) Building (b) Load-force curve

Fig. 1 Building and column characteristics

2. Statement of the problem

The objectives of this project are to investigate the relation between structural system redundancy and the maximum ductility requirements, and to understand the elastic and inelastic behavior of buildings under near-field ground motions.

2.1 Develop a ten-curve graphical chart between the relative displacement ductility μ and the ratio between the excitation period and the natural period of the building T_p/T_n with $R_u/(ms\omega_p) = 0.1, 0.2, 0.3, 0.5, 0.7, 0.9, 1.0, 1.2, 1.5, 2.0$. If necessary, assume s=30 in/sec and then investigate whether the final results are independent of the selection of s value.

2.2 For each γ value, plot a three-curve chart between the maximum relative displacement ductility and the number of columns corresponding to $R_u/(ms\omega_p) = 0.2, 0.5, 1.0$.

2.3 For each number of columns, plot a three-curve chart between the maximum relative displacement ductility and the γ value corresponding to $R_u/(ms\omega_p) = 0.2, 0.5, 1.0$.

3. Analytical model

The dynamic equilibrium of the elasto-perfectly-plastic system is

$$m\ddot{u}(t) + 2m\zeta\omega\dot{u}(t) + f_s(u, \dot{u}) = -m\ddot{u}_g(t)$$

$$\ddot{u}_g(t) = s\omega_p p(t)$$

$$p(t) = e^{-\zeta_p\omega_p t} cos\left(\omega_p\sqrt{1-\zeta_p^2}t + \varphi\right)$$

$$\varphi = \sin^{-1}\zeta_p$$

4. Computer analysis

4.1 Solution for 2.1: a ten-curve graphical chart between the relative displacement ductility μ and the ratio between the excitation period and the natural period of the building T_p/T_n with $R_u/(ms\omega_p) = 0.1, 0.2, 0.3, 0.5, 0.7, 0.9, 1.0, 1.2, 1.5, 2.0$.

In this question, assume that the system has 2 columns $k_1 = k_2 = \frac{k}{2}$. $\omega_p = 0.4$, $s = 30$ in/sec. $\omega_p = 0.4$. Figure 1 shows the relationship between the relative displacement ductility μ and the ratio between the excitation period and the natural period of the building T_p/T_n with $R_u/(ms\omega_p) = 0.1, 0.2, 0.3, 0.5, 0.7, 0.9, 1.0, 1.2, 1.5, 2.0$. To investigate whether the value of s will influence the relationship or not, change $s = 50$ in/sec and run the code again. Figure 2 shows the result and it can be seen that the value of s has a slight impact when $R_u/(ms\omega_p)$ is high.

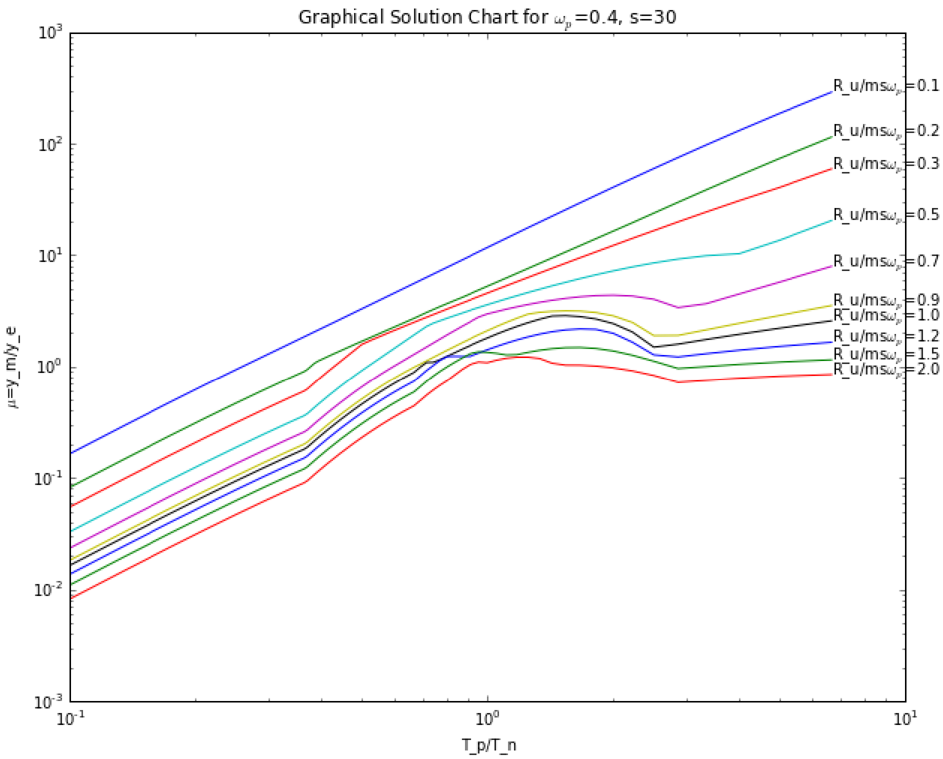


Figure 1 Relationship between μ and T_p/T_n with $s = 30$ in/sec

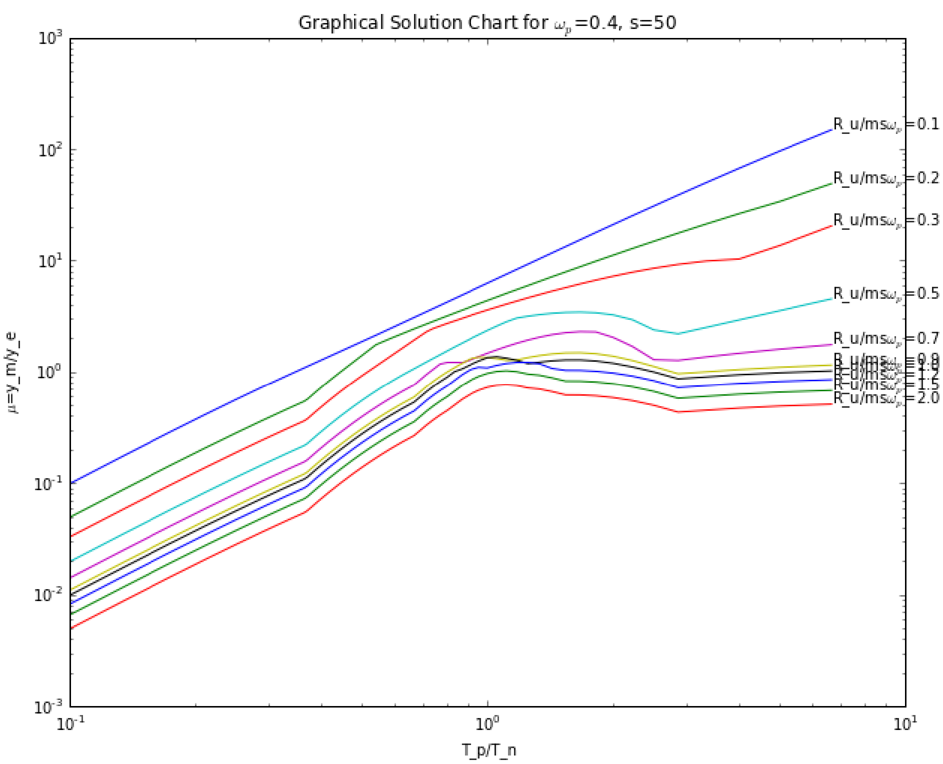
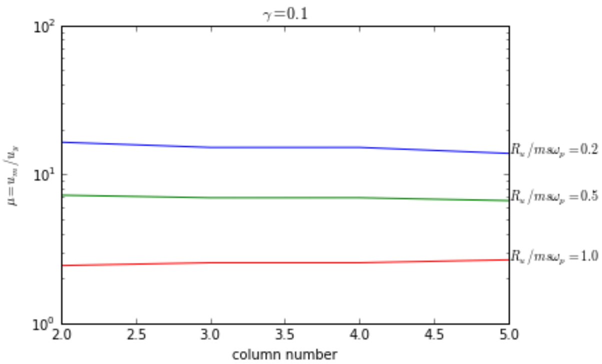
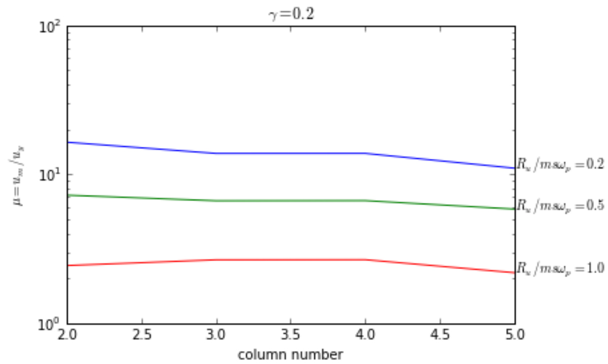


Figure 1 Relationship between μ and T_p/T_n with $s = 50$ in/sec

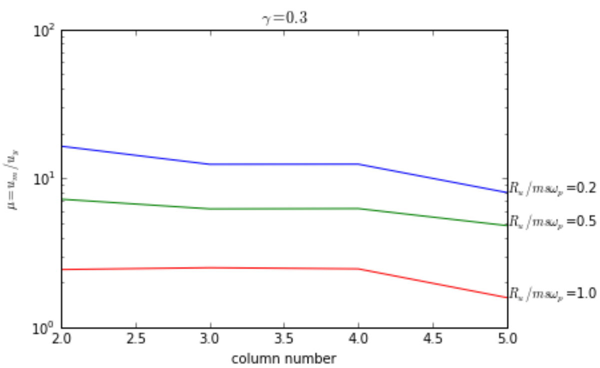
4.2 Solution for 2.2: For each γ value, plot a three-curve chart between the maximum relative displacement ductility and the number of columns corresponding to $R_u/(ms\omega_p) = 0.2, 0.5, 1.0$.



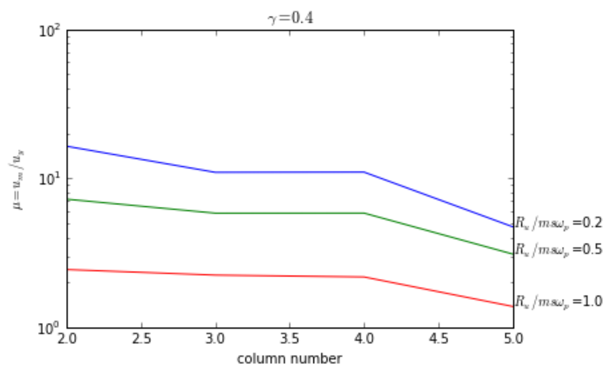
(a) $\gamma = 0.1$



(b) $\gamma = 0.2$

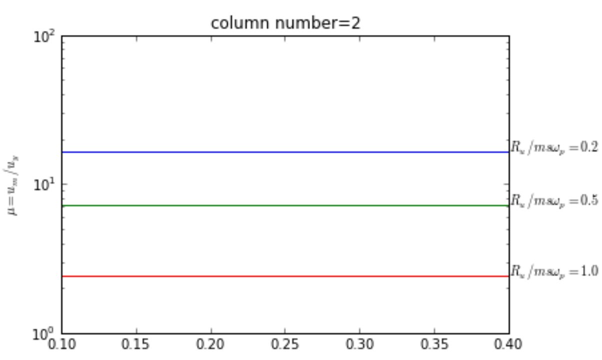


(c) $\gamma = 0.3$

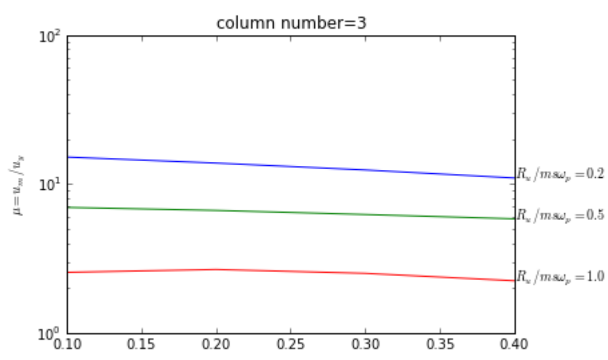


(d) $\gamma = 0.4$

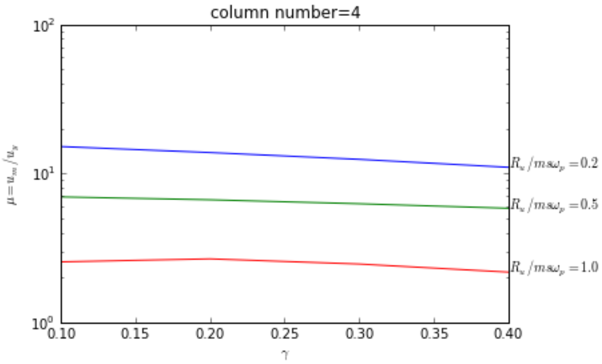
4.3 Solution for 2.3: For each number of columns, plot a three-curve chart between the maximum relative displacement ductility and the γ value corresponding to $R_u/(ms\omega_p) = 0.2, 0.5, 1.0$.



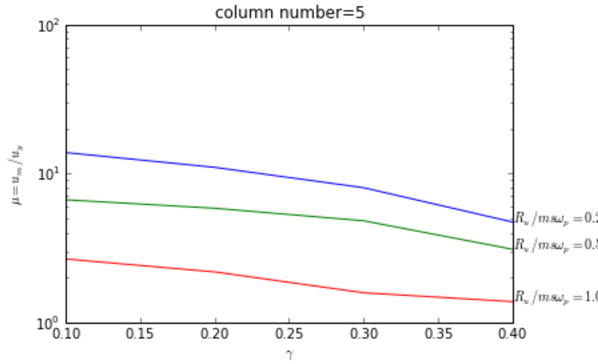
(a) 2 columns



(b) 3 columns



(c) 4 columns



(d) 5 columns

Code for Problem 2.1

```
# -*- coding: utf-8 -*-
"""
Created on Fri May 11 12:54:09 2018

@author: xyvm4

6205 structural dynamics & earthquake engineering

Final Project Problem-1

Xinzhe Yuan
"""

import numpy as np
import math
import matplotlib.pyplot as plt
################################################################################

def Newmark_coefficients(dt,garma,belta):

    a0=1/(belta*(dt**2))
    a1=garma/(belta*dt)
    a2=1/(belta*dt)
    a3=(1/(2*belta))-1
    a4=(garma/belta)-1
    a5=(dt/2)*((garma/belta)-2)
    a6=dt*(1-garma)
    a7=garma*dt

    return a0,a1,a2,a3,a4,a5,a6,a7

################################################################################
# input values

garma = 0.5
belta = 0.25

OmegaP = 0.4
TP = 2*math.pi/OmegaP
TN = TP*np.arange(10,0.1,-0.05)
TP_over_TN = TP/TN

stiffness = 1
mass = TN**2*stiffness/(2*math.pi)**2
massT = zip(mass,TN)
vel_s = 50 #in/sec
damping_ratio = 0.05

loaddata = np.loadtxt('excitation.txt')
inD = 0
dT = 0.02

ratio_force = np.array([0.1,0.2,0.3,0.5,0.7,0.9,1.0,1.2,1.5,2.0])
epslon = 1e-3

elasticS = stiffness
plasticS = 0
a0,a1,a2,a3,a4,a5,a6,a7 = Newmark_coefficients(dT,garma,belta)
```

```
umax = np.zeros((len(massT),len(ratio_force)))
DuctileFactor = np.zeros((len(massT),len(ratio_force)))

##############################################################################
# computation

jj = 0
for ratio in ratio_force:
    ii = 0
    for item in massT:
        yieldforce = ratio*item[0]*vel_s*OmegaP
        yielddisp = yieldforce/stiffness
        eq_force = -item[0]*loaddata
        damping = 2*damping_ratio*item[0]*2*math.pi/item[1]
        steps = len(eq_force)-1
        R = np.zeros(steps)
        kT = np.zeros(steps+1); kT[0] = stiffness
        kTilter = np.zeros(steps)
        deltU = np.zeros(steps)
        u = np.zeros(steps+1); u[0] = inD
        fs = np.zeros(steps+1); fs[0] = u[0]*kT[0]
        ud = np.zeros(steps+1); ud[0] = 0
        udd = np.zeros(steps+1); udd[0] = loaddata[0]
        pTilter = np.zeros(steps)
        pStep = eq_force
        cA = a0*item[0] + a1*damping
        cB = a2*item[0] + a4*damping
        cC = a3*item[0] + a5*damping

        for i in range(steps):
            u[i+1] = u[i]
            kT[i+1] = kT[i]
            fs[i+1] = fs[i]
            pTilter[i] = pStep[i+1]+cA*u[i]+cB*ud[i]+cC*udd[i]
            R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]

            while abs(R[i]) > epslon:
                 kTilter[i] = kT[i+1]+cA
                 deltU[i] = R[i]/kTilter[i]
                 u[i+1] = u[i+1] + deltU[i]
                 fs[i+1] = fs[i+1]+kT[i+1]*deltU[i]

                       if fs[i+1] >= yieldforce:
                           kT[i+1] = plasticS
                           fs[i+1] = yieldforce
                       elif fs[i+1] <= -yieldforce:
                           kT[i+1] = plasticS
                           fs[i+1] = -yieldforce
                       else:
                           kT[i+1] = elasticS
                       R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]


                  ud[i+1] = a1*(u[i+1]-u[i])-a4*ud[i]-a5*udd[i]
                  udd[i+1] = a0*(u[i+1]-u[i])-a2*ud[i]-a3*udd[i]

                  if ud[i+1]*ud[i] <= 0:
                      kT[i+1] = elasticS

              umax[ii][jj] = max(abs(u))
              DuctileFactor[ii][jj] = umax[ii][jj]/yielddisp
```

```
                ii += 1
            jj += 1

        #################################################################################

        # Chart plot

        fig = plt.figure(figsize=(10,8))
        ax = fig.add_subplot(111)
        for i in range(len(ratio_force)):
            text = r'R_u/ms$\omega_p$={}'.format(ratio_force[i])
            line, = ax.plot(TP_over_TN,DuctileFactor[:,i])
            ax.annotate(text, xy=(TP_over_TN[-1],DuctileFactor[:,i][-1]))
            ax.set_yscale('log')
            ax.set_xscale('log')

        plt.xlabel('$T_p/T_n$')
        plt.ylabel(r'$\mu$=y_m/y_e')
        plt.title(r'Graphical Solution Chart for $\omega_p$=0.4, s=50')
        fig.savefig('finalpl1.png')
```

Code for Problem 2&3

```
# -*- coding: utf-8 -*-
"""
Created on Sat May 12 16:10:49 2018

@author: xyvm4

6205 structural dynamics & earthquake engineering

Final Project Problem-2

Xinzhe Yuan
"""
import numpy as np
import math
import matplotlib.pyplot as plt
################################################################################

def Newmark_coefficients(dt,garma,belta):

    a0=1/(belta*(dt**2))
    a1=garma/(belta*dt)
    a2=1/(belta*dt)
    a3=(1/(2*belta))-1
    a4=(garma/belta)-1
    a5=(dt/2)*((garma/belta)-2)
```

```
    a6=dt*(1-garma)
    a7=garma*dt

    return a0,a1,a2,a3,a4,a5,a6,a7

################################################################################
# input values

garma = 0.5
belta = 0.25

OmegaP = 0.4
TP = 2*math.pi/OmegaP
TN = TP*0.5
stiffness = 1
mass = TN**2*stiffness/(2*math.pi)**2
vel_s = 30 #in/sec
damping_ratio = 0.05
loaddata = np.loadtxt('excitation.txt')
eq_force = -mass*loaddata
pStep = eq_force
inD = 0
dT = 0.02
ratio_force = np.array([0.2,0.5,1.0])
epslon = 1e-3
elasticS = stiffness
a0,a1,a2,a3,a4,a5,a6,a7 = Newmark_coefficients(dT,garma,belta)
damping = 2*damping_ratio*mass*2*math.pi/TN
steps = len(eq_force)-1
cA = a0*mass + a1*damping
cB = a2*mass + a4*damping
cC = a3*mass + a5*damping

gamma = np.array([0.1,0.2,0.3,0.4])
col_num = np.array([2,3,4,5])

umax = np.zeros((len(col_num),len(ratio_force)))
DuctileFactor = np.zeros((len(col_num),len(ratio_force)))
solu_23 = np.zeros((len(gamma),len(col_num),len(ratio_force)))

for ii in range(len(gamma)):
#    kk = 0
    gar = gamma[ii]
    for kk in range(len(ratio_force)):
        ratio = ratio_force[kk]
#        yieldforce = ratio*mass*OmegaP*vel_s
#        yielddisp = yieldforce/stiffness

        for jj in range(len(col_num)):
            cnum = col_num[jj]
            R = np.zeros(steps)
            kT = np.zeros(steps+1); kT[0] = stiffness
            kTilter = np.zeros(steps)
            deltU = np.zeros(steps)
            u = np.zeros(steps+1); u[0] = inD
            fs = np.zeros(steps+1); fs[0] = u[0]*kT[0]
            ud = np.zeros(steps+1); ud[0] = 0
            udd = np.zeros(steps+1); udd[0] = loaddata[0]
            pTilter = np.zeros(steps)
            yieldforce = ratio*mass*OmegaP*vel_s

```

```
            if cnum == 2:
                plasticS = 0
                yielddisp = yieldforce/stiffness

                for i in range(steps):
                    u[i+1] = u[i]
                    kT[i+1] = kT[i]
                    fs[i+1] = fs[i]
                    pTilter[i] = pStep[i+1]+cA*u[i]+cB*ud[i]+cC*udd[i]
                    R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]

                    while abs(R[i]) > epslon:
                        kTilter[i] = kT[i+1]+cA
                               deltU[i] = R[i]/kTilter[i]
                               u[i+1] = u[i+1] + deltU[i]
                               fs[i+1] = fs[i+1]+kT[i+1]*deltU[i]

                               if fs[i+1] >= yieldforce:
                                   kT[i+1] = plasticS
                                   fs[i+1] = yieldforce
                               elif fs[i+1] <= -yieldforce:
                                   kT[i+1] = plasticS
                                   fs[i+1] = -yieldforce
                               else:
                                   kT[i+1] = elasticS
                               R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]

                           ud[i+1] = a1*(u[i+1]-u[i])-a4*ud[i]-a5*udd[i]
                           udd[i+1] = a0*(u[i+1]-u[i])-a2*ud[i]-a3*udd[i]

                           if ud[i+1]*ud[i] <= 0:
                               kT[i+1] = elasticS

                       umax[jj][kk] = max(abs(u))
                       DuctileFactor[jj][kk] = umax[jj][kk]/yielddisp

                   if cnum == 3:

                       plasticS1 = (2-gamma[ii])*stiffness/3; force1 = yieldforce/(1+gamma[ii])
                       plasticS2 = (1-gamma[ii])*stiffness/3; force2 = (3-gamma[ii])*yieldforce/3
                       plasticS3 = 0
                       yielddisp = force1/stiffness + (force2-force1)/plasticS1 + (yieldforce-force2)/plasticS2

                       for i in range(steps):
                           u[i+1] = u[i]
                           kT[i+1] = kT[i]
                           fs[i+1] = fs[i]
                           pTilter[i] = pStep[i+1]+cA*u[i]+cB*ud[i]+cC*udd[i]
                           R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]

                           while abs(R[i]) > epslon:
                               kTilter[i] = kT[i+1]+cA
                               deltU[i] = R[i]/kTilter[i]
                               u[i+1] = u[i+1] + deltU[i]
                               fs[i+1] = fs[i+1]+kT[i+1]*deltU[i]

                               if fs[i+1] >= -force1 and fs[i+1] <= force1:
                                   kT[i+1] = elasticS
```

```
                            elif fs[i+1] >= force1 and fs[i+1] <= force2:
                                kT[i+1] = plasticS1
                            elif fs[i+1] >= force2 and fs[i+1] <= yieldforce:
                                kT[i+1] = plasticS2
                            elif fs[i+1] >= yieldforce:
                                kT[i+1] = plasticS3
                                fs[i+1] = yieldforce
                            elif fs[i+1] >= -force2 and fs[i+1] <= -force1:
                                kT[i+1] = plasticS1
                            elif fs[i+1] >= -yieldforce and fs[i+1] <= -force2:
                                kT[i+1] = plasticS2
                            elif fs[i+1] <= -yieldforce:
                                kT[i+1] = plasticS3
                                fs[i+1] = -yieldforce
                            R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]

                        ud[i+1] = a1*(u[i+1]-u[i])-a4*ud[i]-a5*udd[i]
                        udd[i+1] = a0*(u[i+1]-u[i])-a2*ud[i]-a3*udd[i]

                        if ud[i+1]*ud[i] <= 0:
                            kT[i+1] = elasticS

                    umax[jj][kk] = max(abs(u))
                    DuctileFactor[jj][kk] = umax[jj][kk]/yielddisp

                if cnum == 4:

                    plasticS1 = (3-gamma[ii])/4; force1 = yieldforce/(1+gamma[ii])
                    plasticS2 = (1-gamma[ii])/4; force2 = (4-gamma[ii])*yieldforce/4
                    plasticS3 = 0;
                    yielddisp = force1/stiffness + (force2-force1)/plasticS1 + (yieldforce-force2)/plasticS2

                    for i in range(steps):
                        u[i+1] = u[i]
                        kT[i+1] = kT[i]
                        fs[i+1] = fs[i]
                        pTilter[i] = pStep[i+1]+cA*u[i]+cB*ud[i]+cC*udd[i]
                        R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]

                        while abs(R[i]) > epslon:
                            kTilter[i] = kT[i+1]+cA
                            deltU[i] = R[i]/kTilter[i]
                            u[i+1] = u[i+1] + deltU[i]
                            fs[i+1] = fs[i+1]+kT[i+1]*deltU[i]

                            if fs[i+1] >= -force1 and fs[i+1] <= force1:
                                kT[i+1] = elasticS
                            elif fs[i+1] >= force1 and fs[i+1] <= force2:
                                kT[i+1] = plasticS1
                            elif fs[i+1] >= force2 and fs[i+1] <= yieldforce:
                                kT[i+1] = plasticS2
                            elif fs[i+1] >= yieldforce:
                                kT[i+1] = plasticS3
                                fs[i+1] = yieldforce
                            elif fs[i+1] >= -force2 and fs[i+1] <= -force1:
                                kT[i+1] = plasticS1
                            elif fs[i+1] >= -yieldforce and fs[i+1] <= -force2:
                                kT[i+1] = plasticS2
                            elif fs[i+1] <= -yieldforce:
```

```
                                    kT[i+1] = plasticS3
                                    fs[i+1] = -yieldforce
                                R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]

                            ud[i+1] = a1*(u[i+1]-u[i])-a4*ud[i]-a5*udd[i]
                            udd[i+1] = a0*(u[i+1]-u[i])-a2*ud[i]-a3*udd[i]

                            if ud[i+1]*ud[i] <= 0:
                                kT[i+1] = elasticS

                        umax[jj][kk] = max(abs(u))
                        DuctileFactor[jj][kk] = umax[jj][kk]/yielddisp

                    if cnum == 5:

                        plasticS1 = (4-2*gamma[ii])*stiffness/5; force1 = yieldforce/(1+2*gamma[ii])
                        plasticS2 = (3-3*gamma[ii])*stiffness/5; force2 = (5-gamma[ii])*yieldforce/(1+gamma[ii])/5
                        plasticS3 = (2-3*gamma[ii])*stiffness/5; force3 = (5-3*gamma[ii])*yieldforce/5
                        plasticS4 = (1-2*gamma[ii])*stiffness/5; force4 = (5-6*gamma[ii])/5/(1-gamma[ii])*yieldforce
                        plasticS5 = 0
                        yielddisp = force1/stiffness + (force2-force1)/plasticS1 + (force3-force2)/plasticS2 + (force4-force3)/plasticS3 + (yieldforce-force4)/plasticS4

                        for i in range(steps):
                            u[i+1] = u[i]
                            kT[i+1] = kT[i]
                            fs[i+1] = fs[i]
                            pTilter[i] = pStep[i+1]+cA*u[i]+cB*ud[i]+cC*udd[i]
                            R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]

                            while abs(R[i]) > epslon:
                                kTilter[i] = kT[i+1]+cA
                                deltU[i] = R[i]/kTilter[i]
                                u[i+1] = u[i+1] + deltU[i]
                                fs[i+1] = fs[i+1]+kT[i+1]*deltU[i]

                                if fs[i+1] >= -force1 and fs[i+1] <= force1:
                                    kT[i+1] = elasticS
                                elif fs[i+1] >= force1 and fs[i+1] <= force2:
                                    kT[i+1] = plasticS1
                                elif fs[i+1] >= force2 and fs[i+1] <= force3:
                                    kT[i+1] = plasticS2
                                elif fs[i+1] >= force3 and fs[i+1] <= force4:
                                    kT[i+1] = plasticS3
                                elif fs[i+1] >= force4 and fs[i+1] <= yieldforce:
                                    kT[i+1] = plasticS4
                                elif fs[i+1] >= yieldforce:
                                    kT[i+1] = plasticS5
                                    fs[i+1] = yieldforce
                                elif fs[i+1] >= -force2 and fs[i+1] <= -force1:
                                    kT[i+1] = plasticS1
                                elif fs[i+1] >= -force3 and fs[i+1] <= -force2:
                                    kT[i+1] = plasticS2
                                elif fs[i+1] >= -force4 and fs[i+1] <= -force3:
                                    kT[i+1] = plasticS3
                                elif fs[i+1] >= -yieldforce and fs[i+1] <= -force4:
```

```python
                                kT[i+1] = plasticS4
                            elif fs[i+1] <= -yieldforce:
                                kT[i+1] = plasticS5
                                fs[i+1] = -yieldforce
                            R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]

                        ud[i+1] = a1*(u[i+1]-u[i])-a4*ud[i]-a5*udd[i]
                        udd[i+1] = a0*(u[i+1]-u[i])-a2*ud[i]-a3*udd[i]

                        if ud[i+1]*ud[i] <= 0:
                            kT[i+1] = elasticS

                    umax[jj][kk] = max(abs(u))
                    DuctileFactor[jj][kk] = umax[jj][kk]/yielddisp

                jj += 1
            kk += 1

        solu_23[ii] = DuctileFactor

        ii += 1

    ################################################################################

    # plot the results
    # problem 2 for each gamma value
    # key = 0
    for key in range(len(solu_23)):
        item = solu_23[key]
        fig = plt.figure(figsize=(6,4))
        ax = fig.add_subplot(111)
        for pp in range(len(ratio_force)):
            text = r'$R_u/ms\omega_p={}$'.format(ratio_force[pp])
            line, = ax.plot(col_num,item[:,pp])
            ax.annotate(text, xy=(col_num[-1],item[:,pp][-1]))
            ax.set_yscale('log')
        plt.xlabel('column number')
        plt.ylabel(r'$\mu=u_m/u_y$')
        plt.title(r'$\gamma={}$'.format(gamma[key]))
        fig.savefig('fig{}.png'.format(key+1))

    # problem 3 for each column number

    for kkey in range(len(col_num)):
        fig = plt.figure(figsize=(6,4))
        ax = fig.add_subplot(111)
        itemm = solu_23[:,kkey,:]
        for ppp in range(len(ratio_force)):
            text = r'$R_u/ms\omega_p={}$'.format(ratio_force[ppp])
            line, = ax.plot(gamma,itemm[:,ppp])
            ax.annotate(text, xy = (gamma[-1],itemm[:,ppp][-1]))
            ax.set_yscale('log')
        plt.xlabel(r'$\gamma$')
        plt.ylabel(r'$\mu=u_m/u_y$')
        plt.title('column number={}'.format(col_num[kkey]))
        fig.savefig('figg{}.png'.format(kkey+1))
```