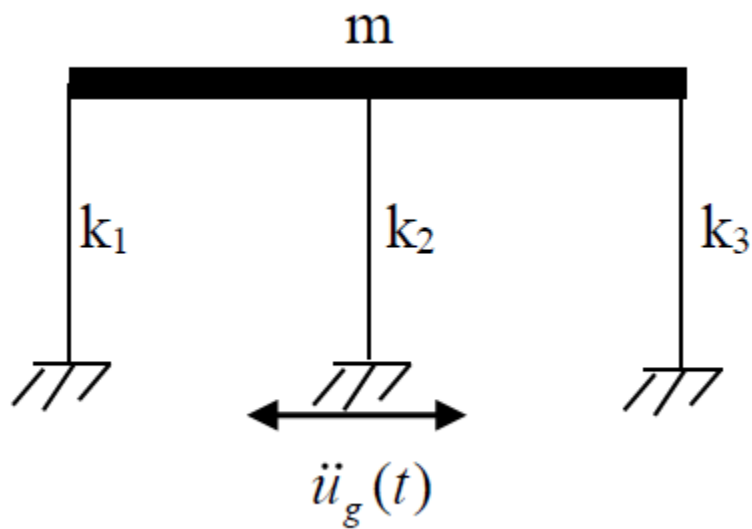


# Single-story building with redundant columns

## 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  $\gamma$  ( $\gamma = 0.1, 0.2, 0.3, 0.4$ ).

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): 5 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

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

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.

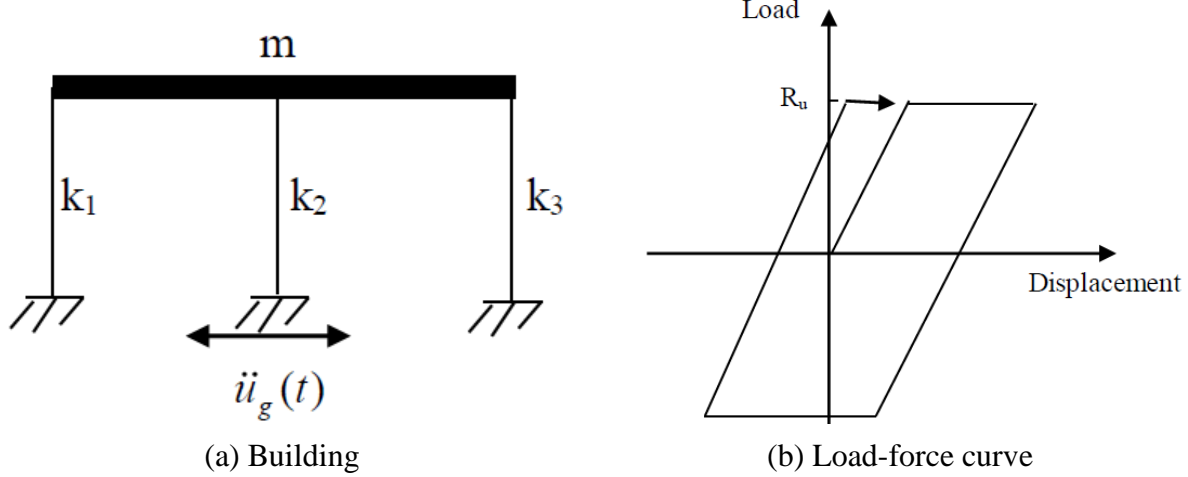


Fig. 1 Building and column characteristics

## 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  $\mu$  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  $\gamma$  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  $\gamma$  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

$$\begin{aligned} m\ddot{u}(t) + 2m\zeta\omega\dot{u}(t) + f_s(u, \dot{u}) &= -m\ddot{u}_g(t) \\ \ddot{u}_g(t) &= s\omega_p p(t) \\ p(t) &= e^{-\zeta_p\omega_p t} \cos\left(\omega_p \sqrt{1 - \zeta_p^2} t + \varphi\right) \\ \varphi &= \sin^{-1} \zeta_p \end{aligned}$$

### 4. Computer analysis

4.1 Solution for 2.1: a ten-curve graphical chart between the relative displacement ductility  $\mu$  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  $\mu$  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.

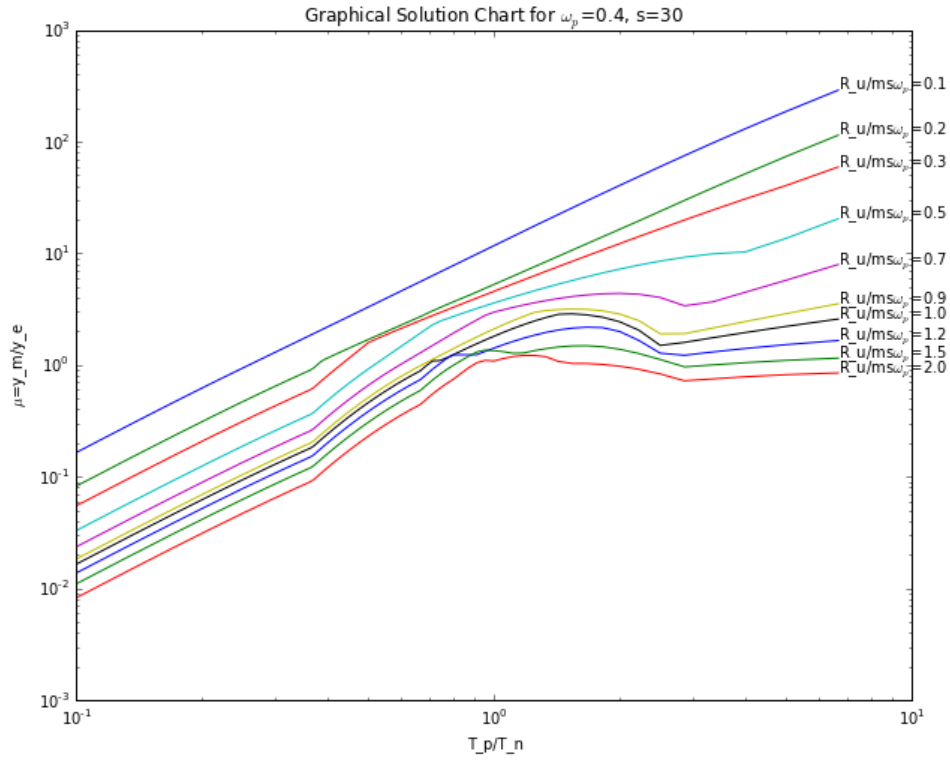


Figure 1 Relationship between  $\mu$  and  $T_p/T_n$  with  $s = 30$  in/sec

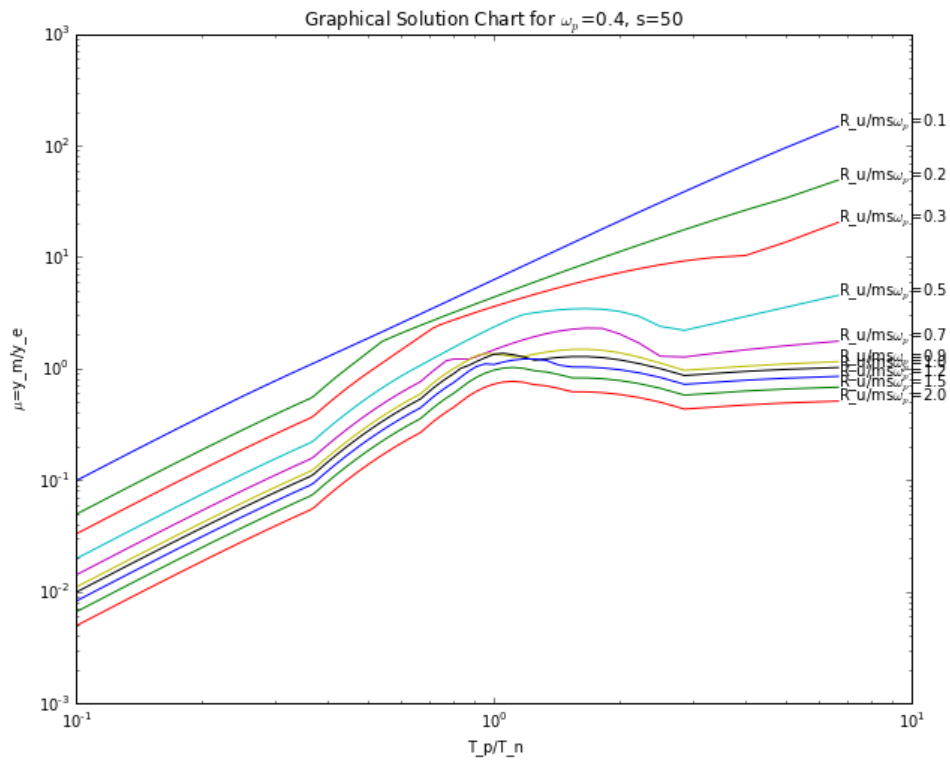
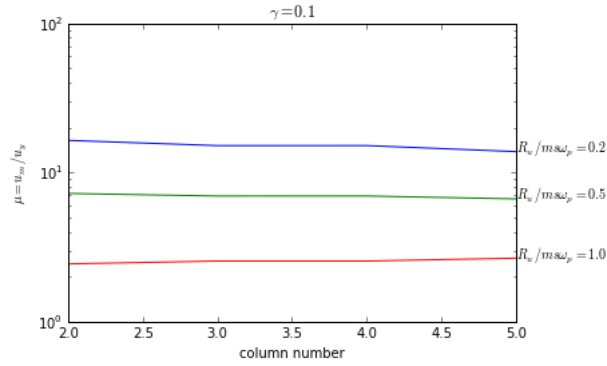
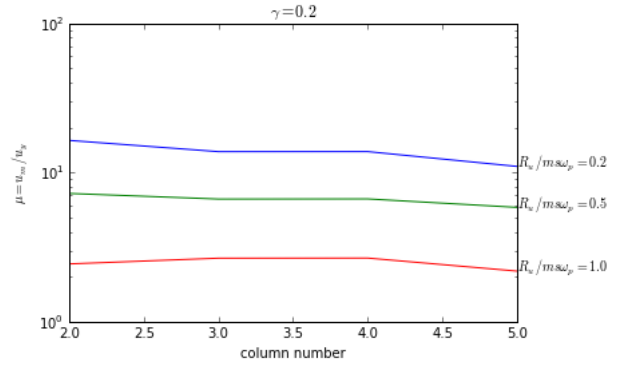


Figure 1 Relationship between  $\mu$  and  $T_p/T_n$  with  $s = 50$  in/sec

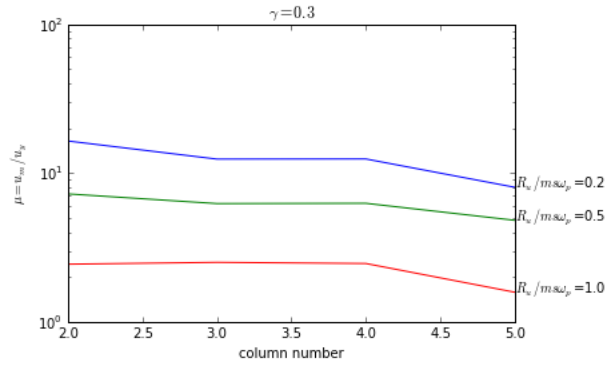
4.2 Solution for 2.2: For each  $\gamma$  value, plot a three-curve chart between the maximum relative displacement ductility and the number of columns corresponding to  $R_u/(ms\omega_p) = 0.2, 0.5, 1.0$ .



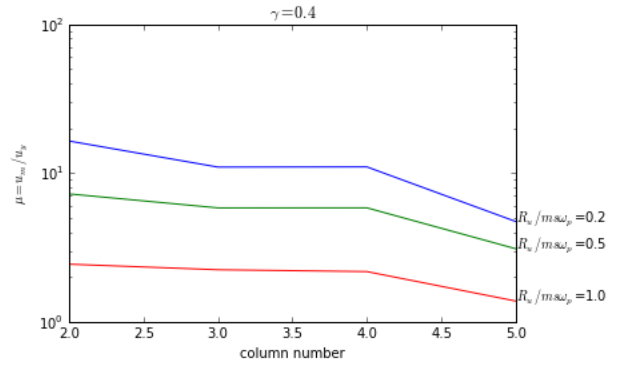
(a)  $\gamma = 0.1$



(b)  $\gamma = 0.2$

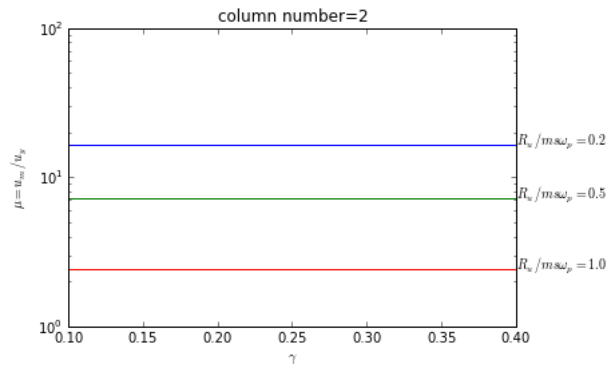


(c)  $\gamma = 0.3$

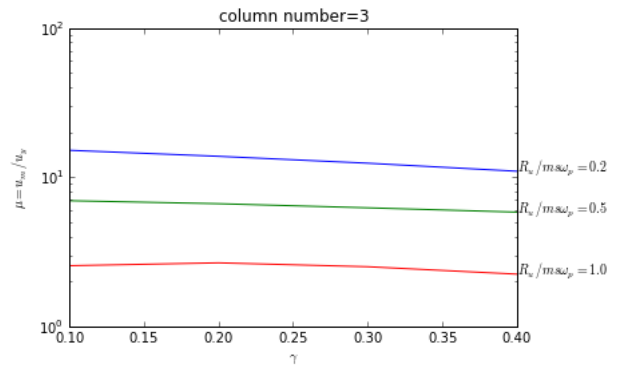


(d)  $\gamma = 0.4$

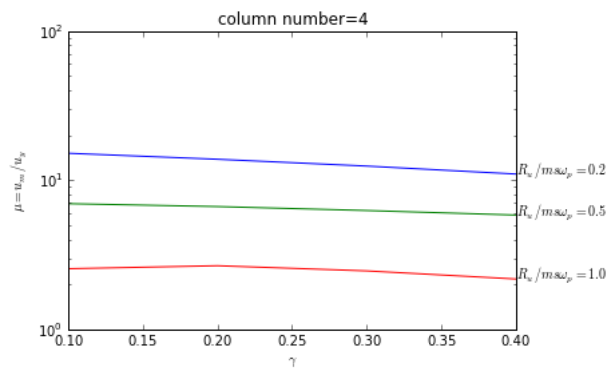
4.3 Solution for 2.3: For each number of columns, plot a three-curve chart between the maximum relative displacement ductility and the  $\gamma$  value corresponding to  $R_u/(ms\omega_p) = 0.2, 0.5, 1.0$ .



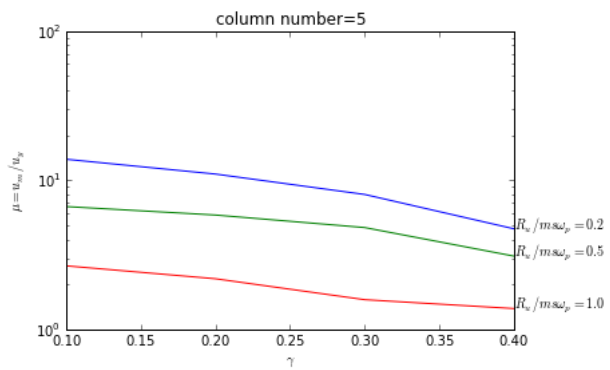
(a) 2 columns



(b) 3 columns



(c) 4 columns



(d) 5 columns

## Code for Problem 2.1

```
1. # -*- coding: utf-8 -*-
2. """
3. Created on Fri May 11 12:54:09 2018
4.
5. @author: xyvm4
6.
7. 6205 structural dynamics & earthquake engineering
8.
9. Final Project Problem-1
10.
11. Xinzhe Yuan
12. """
13.
14. import numpy as np
15. import math
16. import matplotlib.pyplot as plt
17. #####
18.
19. def Newmark_coefficients(dt,garma,belta):
20.
21.     a0=1/(belta*(dt**2))
22.     a1=garma/(belta*dt)
23.     a2=1/(belta*dt)
24.     a3=(1/(2*belta))-1
25.     a4=(garma/belta)-1
26.     a5=(dt/2)*((garma/belta)-2)
27.     a6=dt*(1-garma)
28.     a7=garma*dt
29.
30.     return a0,a1,a2,a3,a4,a5,a6,a7
31.
32. #####
33. # input values
34.
35. garma = 0.5
36. belta = 0.25
37.
38. OmegaP = 0.4
39. TP = 2*math.pi/OmegaP
40. TN = TP*np.arange(10,0.1,-0.05)
41. TP_over_TN = TP/TN
42.
43. stiffness = 1
44. mass = TN**2*stiffness/(2*math.pi)**2
45. massT = zip(mass,TN)
46. vel_s = 50 #in/sec
47. damping_ratio = 0.05
48.
49. loaddata = np.loadtxt('excitation.txt')
50. inD = 0
51. dT = 0.02
52.
53. ratio_force = np.array([0.1,0.2,0.3,0.5,0.7,0.9,1.0,1.2,1.5,2.0])
54. epsilon = 1e-3
55.
56. elasticS = stiffness
57. plasticS = 0
58. a0,a1,a2,a3,a4,a5,a6,a7 = Newmark_coefficients(dT,garma,belta)
```

```

59. umax = np.zeros((len(massT),len(ratio_force)))
60. DuctileFactor = np.zeros((len(massT),len(ratio_force)))
61.
62. #####
63. # computation
64.
65. jj = 0
66. for ratio in ratio_force:
67.     ii = 0
68.     for item in massT:
69.         yieldforce = ratio*item[0]*vel_s*OmegaP
70.         yielddisp = yieldforce/stiffness
71.         eq_force = -item[0]*loaddata
72.         damping = 2*damping_ratio*item[0]*2*math.pi/item[1]
73.         steps = len(eq_force)-1
74.         R = np.zeros(steps)
75.         kT = np.zeros(steps+1); kT[0] = stiffness
76.         kTilter = np.zeros(steps)
77.         deltU = np.zeros(steps)
78.         u = np.zeros(steps+1); u[0] = inD
79.         fs = np.zeros(steps+1); fs[0] = u[0]*kT[0]
80.         ud = np.zeros(steps+1); ud[0] = 0
81.         udd = np.zeros(steps+1); udd[0] = loaddata[0]
82.         pTilter = np.zeros(steps)
83.         pStep = eq_force
84.         cA = a0*item[0] + a1*damping
85.         cB = a2*item[0] + a4*damping
86.         cC = a3*item[0] + a5*damping
87.
88.         for i in range(steps):
89.             u[i+1] = u[i]
90.             kT[i+1] = kT[i]
91.             fs[i+1] = fs[i]
92.             pTilter[i] = pStep[i+1]+cA*u[i]+cB*ud[i]+cC*udd[i]
93.             R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]
94.
95.             while abs(R[i]) > epsilon:
96.                 kTilter[i] = kT[i+1]+cA
97.                 deltU[i] = R[i]/kTilter[i]
98.                 u[i+1] = u[i+1] + deltU[i]
99.                 fs[i+1] = fs[i+1]+kT[i+1]*deltU[i]
100.
101.                 if fs[i+1] >= yieldforce:
102.                     kT[i+1] = plasticS
103.                     fs[i+1] = yieldforce
104.                 elif fs[i+1] <= -yieldforce:
105.                     kT[i+1] = plasticS
106.                     fs[i+1] = -yieldforce
107.                 else:
108.                     kT[i+1] = elasticS
109.                 R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]
110.
111.             ud[i+1] = a1*(u[i+1]-u[i])-a4*ud[i]-a5*udd[i]
112.             udd[i+1] = a0*(u[i+1]-u[i])-a2*ud[i]-a3*udd[i]
113.
114.
115.             if ud[i+1]*ud[i] <= 0:
116.                 kT[i+1] = elasticS
117.
118.             umax[ii][jj] = max(abs(u))
119.             DuctileFactor[ii][jj] = umax[ii][jj]/yielddisp

```



```

120.         ii += 1
121.         jj += 1
122.
123. #####
124. # Chart plot
125.
126. fig = plt.figure(figsize=(10,8))
127. ax = fig.add_subplot(111)
128. for i in range(len(ratio_force)):
129.     text = r'R_u/ms$\omega_p$={}'.format(ratio_force[i])
130.     line, = ax.plot(TP_over_TN,DuctileFactor[:,i])
131.     ax.annotate(text, xy=(TP_over_TN[-1],DuctileFactor[:,i][-1]))
132.     ax.set_yscale('log')
133.     ax.set_xscale('log')
134.
135. plt.xlabel('$T_p/T_n$')
136. plt.ylabel(r'$\mu$=$y_m/y_e$')
137. plt.title(r'Graphical Solution Chart for $\omega_p$=0.4, s=50')
138. fig.savefig('finalp11.png')
139.
140.
141.
142.
143.
144.
145.
146.
147.

```

## Code for Problem 2&3

```

1. # -*- coding: utf-8 -*-
2. """
3. Created on Sat May 12 16:10:49 2018
4.
5. @author: xyvm4
6.
7. 6205 structural dynamics & earthquake engineering
8.
9. Final Project Problem-2
10.
11. Xinzhe Yuan
12. """
13. import numpy as np
14. import math
15. import matplotlib.pyplot as plt
16. #####
17.
18. def Newmark_coefficients(dt,garma,belta):
19.
20.     a0=1/(belta*(dt**2))
21.     a1=garma/(belta*dt)
22.     a2=1/(belta*dt)
23.     a3=(1/(2*belta))-1
24.     a4=(garma/belta)-1
25.     a5=(dt/2)*((garma/belta)-2)

```

```

26.     a6=dt*(1-garma)
27.     a7=garma*dt
28.
29.     return a0,a1,a2,a3,a4,a5,a6,a7
30.
31. #####
32. # input values
33.
34. garma = 0.5
35. belta = 0.25
36.
37. OmegaP = 0.4
38. TP = 2*math.pi/OmegaP
39. TN = TP*0.5
40. stiffness = 1
41. mass = TN**2*stiffness/(2*math.pi)**2
42. vel_s = 30 #in/sec
43. damping_ratio = 0.05
44. loaddata = np.loadtxt('excitation.txt')
45. eq_force = -mass*loaddata
46. pStep = eq_force
47. inD = 0
48. dT = 0.02
49. ratio_force = np.array([0.2,0.5,1.0])
50. epslon = 1e-3
51. elasticS = stiffness
52. a0,a1,a2,a3,a4,a5,a6,a7 = Newmark_coefficients(dT,garma,belta)
53. damping = 2*damping_ratio*mass*2*math.pi/TN
54. steps = len(eq_force)-1
55. cA = a0*mass + a1*damping
56. cB = a2*mass + a4*damping
57. cC = a3*mass + a5*damping
58.
59. gamma = np.array([0.1,0.2,0.3,0.4])
60. col_num = np.array([2,3,4,5])
61.
62. umax = np.zeros((len(col_num),len(ratio_force)))
63. DuctileFactor = np.zeros((len(col_num),len(ratio_force)))
64. solu_23 = np.zeros((len(gamma),len(col_num),len(ratio_force)))
65.
66. for ii in range(len(gamma)):
67.     # kk = 0
68.     gar = gamma[ii]
69.     for kk in range(len(ratio_force)):
70.         ratio = ratio_force[kk]
71.         # yieldforce = ratio*mass*OmegaP*vel_s
72.         # yielddisp = yieldforce/stiffness
73.
74.         for jj in range(len(col_num)):
75.             cnum = col_num[jj]
76.             R = np.zeros(steps)
77.             kT = np.zeros(steps+1); kT[0] = stiffness
78.             kTilter = np.zeros(steps)
79.             deltU = np.zeros(steps)
80.             u = np.zeros(steps+1); u[0] = inD
81.             fs = np.zeros(steps+1); fs[0] = u[0]*kT[0]
82.             ud = np.zeros(steps+1); ud[0] = 0
83.             udd = np.zeros(steps+1); udd[0] = loaddata[0]
84.             pTilter = np.zeros(steps)
85.             yieldforce = ratio*mass*OmegaP*vel_s
86.

```

```

87.         if cnum == 2:
88.             plasticS = 0
89.             yieldddisp = yieldforce/stiffness
90.
91.             for i in range(steps):
92.                 u[i+1] = u[i]
93.                 kT[i+1] = kT[i]
94.                 fs[i+1] = fs[i]
95.                 pTilter[i] = pStep[i+1]+cA*u[i]+cB*ud[i]+cC*udd[i]
96.                 R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]
97.
98.                 while abs(R[i]) > epsilon:
99.                     kTilter[i] = kT[i+1]+cA
100.                     deltu[i] = R[i]/kTilter[i]
101.                     u[i+1] = u[i+1] + deltu[i]
102.                     fs[i+1] = fs[i+1]+kT[i+1]*deltu[i]
103.
104.                     if fs[i+1] >= yieldforce:
105.                         kT[i+1] = plasticS
106.                         fs[i+1] = yieldforce
107.                     elif fs[i+1] <= -yieldforce:
108.                         kT[i+1] = plasticS
109.                         fs[i+1] = -yieldforce
110.                     else:
111.                         kT[i+1] = elasticS
112.                         R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]
113.
114.                     ud[i+1] = a1*(u[i+1]-u[i])-a4*ud[i]-a5*udd[i]
115.                     udd[i+1] = a0*(u[i+1]-u[i])-a2*ud[i]-a3*udd[i]
116.
117.                     if ud[i+1]*ud[i] <= 0:
118.                         kT[i+1] = elasticS
119.
120.                     umax[jj][kk] = max(abs(u))
121.                     DuctileFactor[jj][kk] = umax[jj][kk]/yieldddisp
122.
123.             if cnum == 3:
124.
125.                 plasticS1 = (2-
126.                     gamma[ii])*stiffness/3; force1 = yieldforce/(1+gamma[ii])
127.                 plasticS2 = (1-gamma[ii])*stiffness/3; force2 = (3-
128.                     gamma[ii])*yieldforce/3
129.                 plasticS3 = 0
130.                 yieldddisp = force1/stiffness + (force2-
131.                     force1)/plasticS1 + (yieldforce-force2)/plasticS2
132.
133.                 for i in range(steps):
134.                     u[i+1] = u[i]
135.                     kT[i+1] = kT[i]
136.                     fs[i+1] = fs[i]
137.                     pTilter[i] = pStep[i+1]+cA*u[i]+cB*ud[i]+cC*udd[i]
138.                     R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]
139.
140.                     while abs(R[i]) > epsilon:
141.                         kTilter[i] = kT[i+1]+cA
142.                         deltu[i] = R[i]/kTilter[i]
143.                         u[i+1] = u[i+1] + deltu[i]
144.                         fs[i+1] = fs[i+1]+kT[i+1]*deltu[i]
145.
146.                         if fs[i+1] >= -force1 and fs[i+1] <= force1:
147.                             kT[i+1] = elasticS

```

```

145.         elif fs[i+1] >= force1 and fs[i+1] <= force2:
146.             kT[i+1] = plasticS1
147.         elif fs[i+1] >= force2 and fs[i+1] <= yieldforce:
148.             kT[i+1] = plasticS2
149.         elif fs[i+1] >= yieldforce:
150.             kT[i+1] = plasticS3
151.             fs[i+1] = yieldforce
152.         elif fs[i+1] >= -force2 and fs[i+1] <= -force1:
153.             kT[i+1] = plasticS1
154.         elif fs[i+1] >= -yieldforce and fs[i+1] <= -force2:
155.             kT[i+1] = plasticS2
156.         elif fs[i+1] <= -yieldforce:
157.             kT[i+1] = plasticS3
158.             fs[i+1] = -yieldforce
159.         R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]
160.
161.         ud[i+1] = a1*(u[i+1]-u[i])-a4*ud[i]-a5*udd[i]
162.         udd[i+1] = a0*(u[i+1]-u[i])-a2*ud[i]-a3*udd[i]
163.
164.         if ud[i+1]*ud[i] <= 0:
165.             kT[i+1] = elasticS
166.
167.         umax[jj][kk] = max(abs(u))
168.         DuctileFactor[jj][kk] = umax[jj][kk]/yieldldisp
169.
170.         if cnum == 4:
171.
172.             plasticS1 = (3-gamma[ii])/4; force1 = yieldforce/(1+gamma[ii])
173.             plasticS2 = (1-gamma[ii])/4; force2 = (4-
gamma[ii])*yieldforce/4
174.             plasticS3 = 0;
175.             yieldldisp = force1/stiffness + (force2-
force1)/plasticS1 + (yieldforce-force2)/plasticS2
176.
177.         for i in range(steps):
178.             u[i+1] = u[i]
179.             kT[i+1] = kT[i]
180.             fs[i+1] = fs[i]
181.             pTilter[i] = pStep[i+1]+cA*u[i]+cB*ud[i]+cC*udd[i]
182.             R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]
183.
184.         while abs(R[i]) > epsilon:
185.             kTilter[i] = kT[i+1]+cA
186.             deltU[i] = R[i]/kTilter[i]
187.             u[i+1] = u[i+1] + deltU[i]
188.             fs[i+1] = fs[i+1]+kT[i+1]*deltU[i]
189.
190.         if fs[i+1] >= -force1 and fs[i+1] <= force1:
191.             kT[i+1] = elasticS
192.         elif fs[i+1] >= force1 and fs[i+1] <= force2:
193.             kT[i+1] = plasticS1
194.         elif fs[i+1] >= force2 and fs[i+1] <= yieldforce:
195.             kT[i+1] = plasticS2
196.         elif fs[i+1] >= yieldforce:
197.             kT[i+1] = plasticS3
198.             fs[i+1] = yieldforce
199.         elif fs[i+1] >= -force2 and fs[i+1] <= -force1:
200.             kT[i+1] = plasticS1
201.         elif fs[i+1] >= -yieldforce and fs[i+1] <= -force2:
202.             kT[i+1] = plasticS2
203.         elif fs[i+1] <= -yieldforce:

```

```

204.                kT[i+1] = plasticS3
205.                fs[i+1] = -yieldforce
206.                R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]
207.
208.                ud[i+1] = a1*(u[i+1]-u[i])-a4*ud[i]-a5*udd[i]
209.                udd[i+1] = a0*(u[i+1]-u[i])-a2*ud[i]-a3*udd[i]
210.
211.                if ud[i+1]*ud[i] <= 0:
212.                    kT[i+1] = elasticS
213.
214.                umax[jj][kk] = max(abs(u))
215.                DuctileFactor[jj][kk] = umax[jj][kk]/yielddisp
216.
217.                if cnum == 5:
218.
219.                    plasticS1 = (4-
220.                        2*gamma[ii])*stiffness/5; force1 = yieldforce/(1+2*gamma[ii])
221.                    plasticS2 = (3-3*gamma[ii])*stiffness/5; force2 = (5-
222.                        gamma[ii])*yieldforce/(1+gamma[ii])/5
223.                    plasticS3 = (2-3*gamma[ii])*stiffness/5; force3 = (5-
224.                        3*gamma[ii])*yieldforce/5
225.                    plasticS4 = (1-2*gamma[ii])*stiffness/5; force4 = (5-
226.                        6*gamma[ii])/5/(1-gamma[ii])*yieldforce
227.                    plasticS5 = 0
228.                    yielddisp = force1/stiffness + (force2-
229.                        force1)/plasticS1 + (force3-force2)/plasticS2 + (force4-
230.                        force3)/plasticS3 + (yieldforce-force4)/plasticS4
231.
232.                for i in range(steps):
233.                    u[i+1] = u[i]
234.                    kT[i+1] = kT[i]
235.                    fs[i+1] = fs[i]
236.                    pTilter[i] = pStep[i+1]+cA*u[i]+cB*ud[i]+cC*udd[i]
237.                    R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]
238.
239.                    while abs(R[i]) > epsilon:
240.                        kTilter[i] = kT[i+1]+cA
241.                        deltu[i] = R[i]/kTilter[i]
242.                        u[i+1] = u[i+1] + deltu[i]
243.                        fs[i+1] = fs[i+1]+kT[i+1]*deltu[i]
244.
245.                        if fs[i+1] >= -force1 and fs[i+1] <= force1:
246.                            kT[i+1] = elasticS
247.                        elif fs[i+1] >= force1 and fs[i+1] <= force2:
248.                            kT[i+1] = plasticS1
249.                        elif fs[i+1] >= force2 and fs[i+1] <= force3:
250.                            kT[i+1] = plasticS2
251.                        elif fs[i+1] >= force3 and fs[i+1] <= force4:
252.                            kT[i+1] = plasticS3
253.                        elif fs[i+1] >= force4 and fs[i+1] <= yieldforce:
254.                            kT[i+1] = plasticS4
255.                        elif fs[i+1] >= yieldforce:
256.                            kT[i+1] = plasticS5
257.                            fs[i+1] = yieldforce
258.                        elif fs[i+1] >= -force2 and fs[i+1] <= -force1:
259.                            kT[i+1] = plasticS1
260.                        elif fs[i+1] >= -force3 and fs[i+1] <= -force2:
261.                            kT[i+1] = plasticS2
262.                        elif fs[i+1] >= -force4 and fs[i+1] <= -force3:
263.                            kT[i+1] = plasticS3
264.                        elif fs[i+1] >= -yieldforce and fs[i+1] <= -force4:

```

```

259.             kT[i+1] = plasticS4
260.         elif fs[i+1] <= -yieldforce:
261.             kT[i+1] = plasticS5
262.             fs[i+1] = -yieldforce
263.             R[i] = pTilter[i]-fs[i+1]-cA*u[i+1]
264.
265.             ud[i+1] = a1*(u[i+1]-u[i])-a4*ud[i]-a5*udd[i]
266.             udd[i+1] = a0*(u[i+1]-u[i])-a2*ud[i]-a3*udd[i]
267.
268.             if ud[i+1]*ud[i] <= 0:
269.                 kT[i+1] = elasticS
270.
271.             umax[jj][kk] = max(abs(u))
272.             DuctileFactor[jj][kk] = umax[jj][kk]/yielddisp
273.
274.             jj += 1
275.             kk += 1
276.
277.             solu_23[ii] = DuctileFactor
278.
279.             ii += 1
280.
281. #####
282. # plot the results
283. # problem 2 for each gamma value
284. # key = 0
285. for key in range(len(solu_23)):
286.     item = solu_23[key]
287.     fig = plt.figure(figsize=(6,4))
288.     ax = fig.add_subplot(111)
289.     for pp in range(len(ratio_force)):
290.         text = r'$R_u/ms\omega_p=\{\}$'.format(ratio_force[pp])
291.         line, = ax.plot(col_num,item[:,pp])
292.         ax.annotate(text, xy=(col_num[-1],item[:,pp][-1]))
293.         ax.set_yscale('log')
294.         plt.xlabel('column number')
295.         plt.ylabel(r'$\mu=u_m/u_y$')
296.         plt.title(r'$\gamma=\{\}$'.format(gamma[key]))
297.         fig.savefig('fig{}.png'.format(key+1))
298.
299. # problem 3 for each column number
300.
301. for kkey in range(len(col_num)):
302.     fig = plt.figure(figsize=(6,4))
303.     ax = fig.add_subplot(111)
304.     itemm = solu_23[:,kkey,:]
305.     for ppp in range(len(ratio_force)):
306.         text = r'$R_u/ms\omega_p=\{\}$'.format(ratio_force[ppp])
307.         line, = ax.plot(gamma,itemm[:,ppp])
308.         ax.annotate(text, xy = (gamma[-1],itemm[:,ppp][-1]))
309.         ax.set_yscale('log')
310.         plt.xlabel(r'$\gamma$')
311.         plt.ylabel(r'$\mu=u_m/u_y$')
312.         plt.title('column number=\{\}'.format(col_num[kkey]))
313.         fig.savefig('figg{}.png'.format(kkey+1))

```