Problem 1:

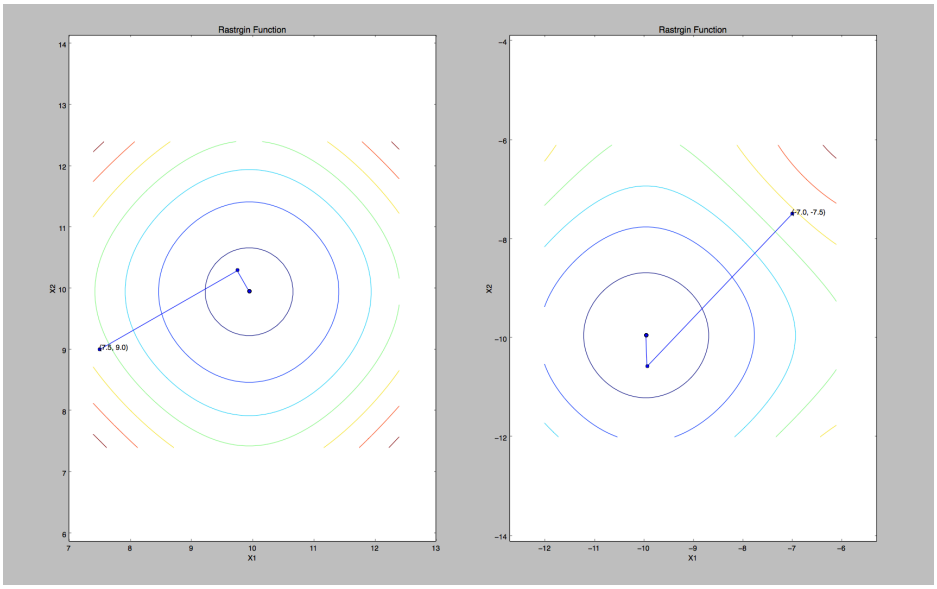

Figure 1: Steepest descent method.

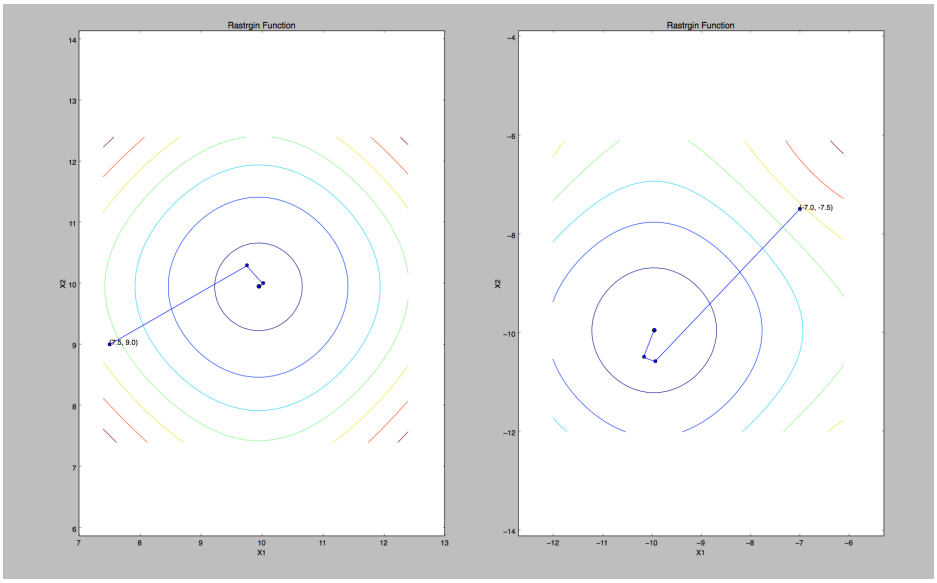

Figure 2: Conjugate gradient method.

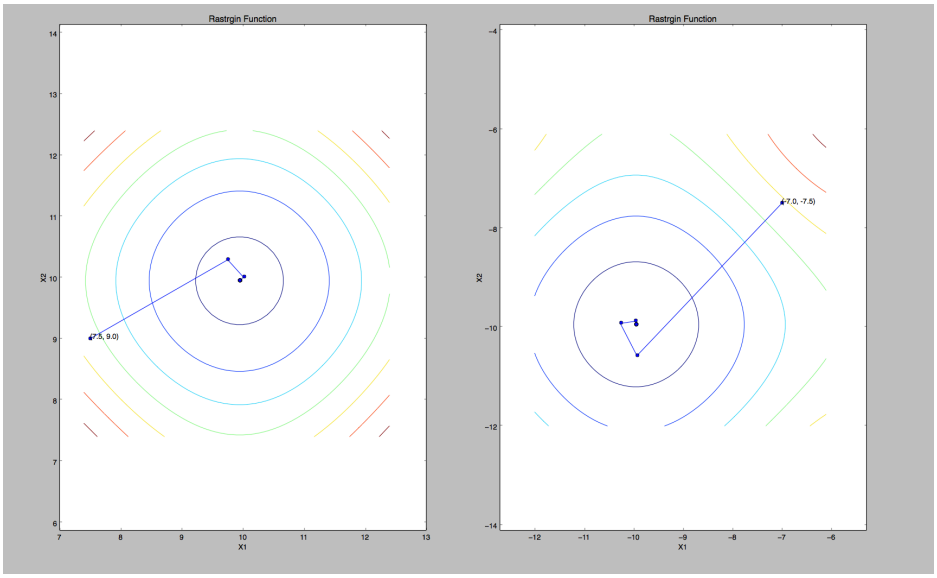

Figure 3: Rank one correction

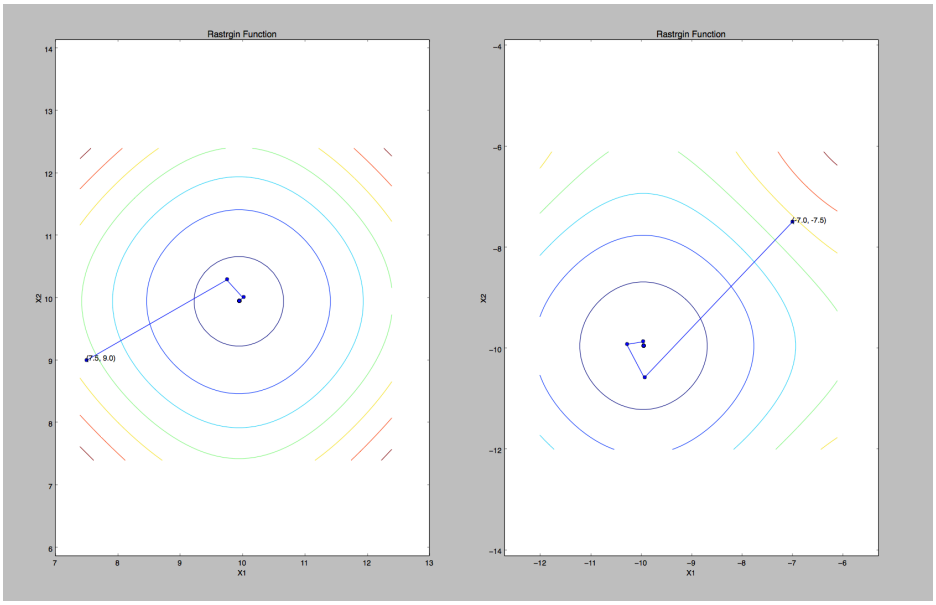

Figure 4: DFP method.

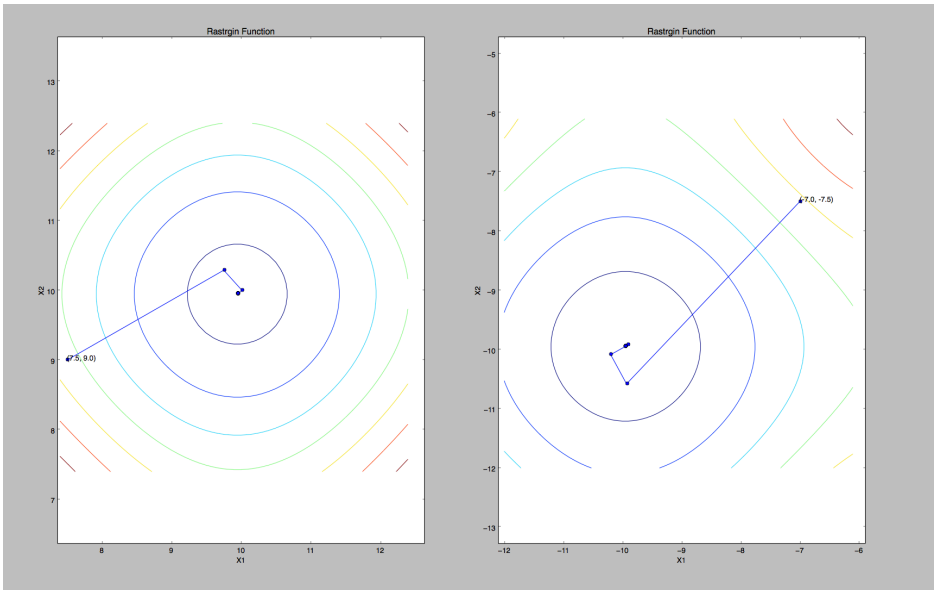

Figure 5: BFGS method.

Table 1: Minimization result summary for Rastrigin's function with different starting points

Method	X_0: [7.5,9.0]	X_0: [-7.0,-7.5]
Steepest Gradient	[9.94958639528, 9.94958638731]	[-9.94958637149, -9.9495863764]
Conjugate Gradient	[9.94958638332, 9.94958640373]	[-9.94958637058, -9.94958637687]
Rank one correction	[9.94958641809, 9.94958633077]	[-9.94958632695, -9.9495867058]
DFP	[9.94958641892, 9.94958632983]	[-9.94958638618, -9.94958631318]
BFGS	[9.94958635865, 9.94958639946]	[-9.94958647941, -9.94958646598]

Code (python):

```
1
2  import matplotlib.pyplot as plt
3
4  import numpy as np
5  def rastriginFunc(x1, x2):
6      return 20 + 0.01*x1**2 + 0.01*x2**2 \
7             - 10*(np.cos(0.2*np.pi*x1)+np.cos(0.2*np.pi*x2))
8
9
10 def plotContour(ax, xlim, ylim):
11     Δ = 0.1
12     x = np.arange(xlim[0], xlim[1], Δ)
13     y = np.arange(ylim[0], ylim[1], Δ)
14     X1, X2 = np.meshgrid(x, y)
15     Z = rastriginFunc(X1,X2)
16     ax.set_title("Rastrgin Function")
17     ax.set_xlabel("X1")
18     ax.set_ylabel("X2")
19     cs = ax.contour(X1, X2, Z)
20     return cs
21
22 def gradientRastrigin(x1, x2):
23     # df/dx1:
24     g1 = 0.02*x1 +2*np.pi*np.sin(0.2*np.pi*x1)
25     g2 = 0.02*x2 +2*np.pi*np.sin(0.2*np.pi*x2)
26     return g1, g2
27
28
29 def lineSearchFunc(X, alpha, D):
30     x1, x2 = X
31     d1, d2 = D
32     return rastriginFunc(x1 + alpha * d1, x2 + alpha * d2)
33
34 def goldenSection(func, X, Direction,  left ,right, tol):
35     rho = (np.sqrt(5)-1)/2  # 1.618
36     length = right - left
37     if abs(length) < tol:
38         return (right + left)/2
39     mR = left  +  rho * length
40     mL = right -  rho * length
41
42     if func(X, mL,Direction ) < func( X, mR, Direction):
43         return goldenSection(func,X, Direction,  left, mR,tol)
44     else:
45         return goldenSection(func,X, Direction,  mL, right, tol)
46
47
48
49 def bracket(Alpha0, func, X, D, epsilon):
50
51     Xleft = Alpha0
52     Xright = 0
53     i = 1
54     while i<100:     ## i can not be too large.
```

```
55          Xright += Xleft + i* epsilon
56          #print Xright, func(X, Xright, D)
57          if func(X, Xright, D) ≥ func(X, Xleft, D):
58              return Xleft, Xright
59          else:
60              i+=1
61              Xleft = Xright
62
63
64  def steepesDescent(X0, tol):
65      x1List = []
66      x2List = []
67      x1_current = X0[0]
68      x2_current = X0[1]
69      g1, g2 = 1.0, 1.0
70      counter = 0
71      while np.sqrt(g1**2 + g2**2) > tol:
72          x1List.append(x1_current)
73          x2List.append(x2_current)
74          #print x1_current, x2_current
75          g1, g2 = gradientRastrigin(x1_current, x2_current)
76          iLeft, iRight = bracket(0, lineSearchFunc, (x1_current, ...
                x2_current), (-g1, -g2), epsilon=0.001)
77          alpha = goldenSection(lineSearchFunc,(x1_current, ...
                x2_current),(-g1, -g2), iLeft, iRight, tol)
78          x1_current = x1_current - alpha * g1
79          x2_current = x2_current - alpha * g2
80          counter += 1
81          if counter>100:
82              print "Iteration more than 100"
83              break
84
85
86      return x1List, x2List
87
88
89  def conjugateGradient(X0, tol) :
90      x1List = []
91      x2List = []
92      x1_current = X0[0]
93      x2_current = X0[1]
94      g1, g2 = gradientRastrigin(x1_current, x2_current)
95      d1, d2 = -g1, -g2
96      counter = 0
97      while np.sqrt(g1**2 + g2**2) > tol: # and counter < 10:
98          x1List.append(x1_current)
99          x2List.append(x2_current)
100         #print x1_current, x2_current
101         g1, g2 = gradientRastrigin(x1_current, x2_current)
102         iLeft, iRight = bracket(0, lineSearchFunc, (x1_current, ...
                x2_current), (d1, d2), epsilon=0.001)
103         alpha = goldenSection(lineSearchFunc,(x1_current, ...
                x2_current),(d1, d2), iLeft, iRight, tol)
104
105         x1_current = x1_current + alpha * d1
106         x2_current = x2_current + alpha * d2
107
108         g1_new, g2_new = gradientRastrigin(x1_current, x2_current)
109         beta = max(0, (g1_new*(g1_new - g1) + ...
                g2_new*(g2_new-g2))/(g1*g1 + g2*g2) )
110         d1 = -g1_new + beta * d1
111         d2 = -g2_new + beta * d2
112
113
114         counter += 1
115         if counter>100:
116             print "Iteration more than 100"
117             break
118     return x1List, x2List
119
120 def rankone(X0, tol):
```

```
121
122     x1List = []
123     x2List = []
124     x1_current = X0[0]
125     x2_current = X0[1]
126     g1, g2 = gradientRastrigin(x1_current, x2_current)
127     H = np.identity(2)
128     D =-H * np.matrix([[g1], [g2]])
129     d1, d2 = D.tolist()[0][0], D.tolist()[1][0]
130     counter = 0
131     while np.sqrt(g1**2 + g2**2) > tol: # and counter < 100:
132         x1List.append(x1_current)
133         x2List.append(x2_current)
134         #print x1_current, x2_current
135         g1, g2 = gradientRastrigin(x1_current, x2_current)
136         iLeft, iRight = bracket(0, lineSearchFunc, (x1_current, ...
                x2_current), (d1, d2), epsilon=0.001)
137         alpha = goldenSection(lineSearchFunc,(x1_current, ...
                x2_current),(d1, d2), iLeft, iRight, tol)
138
139
140
141         x1_current = x1_current + alpha * d1
142         x2_current = x2_current + alpha * d2
143
144         g1_new, g2_new = gradientRastrigin(x1_current, x2_current)
145         diff_x = alpha * np.matrix([[d1], [d2]])
146         diff_g = np.matrix([[g1_new-g1], [g2_new-g2]])
147         denom =np.dot( diff_g.transpose(), (diff_x - ...
                H*diff_g)).tolist()[0][0]
148         numerator = (diff_x - H*diff_g)*((diff_x - ...
                H*diff_g).transpose())
149         H =  H + 1.0/denom*numerator
150         D = -H* np.matrix([[g1_new], [g2_new]])
151         d1, d2 = D.tolist()[0][0], D.tolist()[1][0]
152
153         counter += 1
154         if counter>100:
155             print "Iteration more than 100"
156             break
157
158
159     return x1List, x2List
160
161 def DFP(X0, tol):
162
163     x1List = []
164     x2List = []
165     x1_current = X0[0]
166     x2_current = X0[1]
167     g1, g2 = gradientRastrigin(x1_current, x2_current)
168     H = np.identity(2)
169     D =-H * np.matrix([[g1], [g2]])
170     d1, d2 = D.tolist()[0][0], D.tolist()[1][0]
171     counter = 0
172     while np.sqrt(g1**2 + g2**2) > tol: # and counter < 100:
173         x1List.append(x1_current)
174         x2List.append(x2_current)
175         #print x1_current, x2_current
176         g1, g2 = gradientRastrigin(x1_current, x2_current)
177         iLeft, iRight = bracket(0, lineSearchFunc, (x1_current, ...
                x2_current), (d1, d2), epsilon=0.001)
178         alpha = goldenSection(lineSearchFunc,(x1_current, ...
                x2_current),(d1, d2), iLeft, iRight, tol)
179
180
181
182         x1_current = x1_current + alpha * d1
183         x2_current = x2_current + alpha * d2
184
185         g1_new, g2_new = gradientRastrigin(x1_current, x2_current)
```

```
186         diff_x = alpha * np.matrix([[d1], [d2]])
187         diff_g = np.matrix([[g1_new-g1], [g2_new-g2]])
188         denom1 = np.dot(diff_x.transpose(), diff_g).tolist()[0][0]
189         numerator1 = diff_x * (diff_x.transpose())
190 
191         denom2 =np.dot( diff_g.transpose(), H*diff_g).tolist()[0][0]
192         numerator2 = (H*diff_g)*((H*diff_g).transpose())
193         H =  H + numerator1/denom1 - numerator2/denom2  ...
                #denom*numerator
194         D = -H* np.matrix([[g1_new], [g2_new]])
195         d1, d2 = D.tolist()[0][0], D.tolist()[1][0]
196 
197         counter += 1
198         if counter>100:
199             print "Iteration more than 100"
200             break
201 
202     return x1List, x2List
203 
204 def BFGS(X0, tol):
205 
206     x1List = []
207     x2List = []
208     x1_current = X0[0]
209     x2_current = X0[1]
210     g1, g2 = gradientRastrigin(x1_current, x2_current)
211     H = np.identity(2)
212     D =-H * np.matrix([[g1], [g2]])
213     d1, d2 = D.tolist()[0][0], D.tolist()[1][0]
214     counter = 0
215     while np.sqrt(g1**2 + g2**2) > tol:
216         x1List.append(x1_current)
217         x2List.append(x2_current)
218         #print x1_current, x2_current
219         g1, g2 = gradientRastrigin(x1_current, x2_current)
220         iLeft, iRight = bracket(0, lineSearchFunc, (x1_current, ...
                x2_current), (d1, d2), epsilon=0.001)
221         alpha = goldenSection(lineSearchFunc,(x1_current, ...
                x2_current),(d1, d2), iLeft, iRight, tol)
222 
223 
224 
225         x1_current = x1_current + alpha * d1
226         x2_current = x2_current + alpha * d2
227 
228         g1_new, g2_new = gradientRastrigin(x1_current, x2_current)
229         diff_x = alpha * np.matrix([[d1], [d2]])
230         diff_g = np.matrix([[g1_new-g1], [g2_new-g2]])
231 
232         denom0      =   np.dot(diff_g.transpose(), ...
                diff_x).tolist()[0][0]
233         numerator0  =   np.dot(diff_g.transpose(), ...
                H*diff_g).tolist()[0][0]
234 
235 
236         denom1      = np.dot(diff_x.transpose(), diff_g)
237         numerator1  = diff_x * (diff_x.transpose())
238         denom2 =np.dot( diff_g.transpose(), diff_x).tolist()[0][0]
239         numerator2 = H*diff_g*(diff_x.transpose()) + ...
                (H*diff_g*(diff_x.transpose())).transpose()
240 
241         H =  H + (1+numerator0/denom0)*numerator1/denom1 - ...
                numerator2/denom2  #denom*numerator
242 
243         D = -H* np.matrix([[g1_new], [g2_new]])
244         d1, d2 = D.tolist()[0][0], D.tolist()[1][0]
245         counter +=  1
246         if counter>100:
247             print "Iteration more than 100"
248             break
249 
```

```
250
251
252     return x1List, x2List
253
254
255
256 def minfinder(method):
257
258     X0_p= (7.5, 9.0)
259     X0_n= (-7.0, -7.5)
260
261     f, (ax1, ax2) = plt.subplots(1,2, sharex=False)
262     ## For starting point (7.5, 9.0)
263     plotContour(ax1, [7.4,12.5] , [7.4,12.5])
264     x1List, x2List = method(X0_p, tol=1.0e-8)
265     ax1.annotate( (%s, %s) % X0_p, xy= X0_p, textcoords= data  )
266     ax1.plot(x1List, x2List,  o- )
267     ax1.axis( equal )
268     print "[" + str(x1List[-1]) + ", "  + str(x2List[-1]) + "]"
269
270     ## For starting point (-7.0, -7.5)
271     plotContour(ax2, [-12.0,-6.00] , [-12.0,-6.0])
272     x1List, x2List = method(X0_n, tol=1.0e-8)
273     ax2.annotate( (%s, %s) % X0_n, xy= X0_n, textcoords= data  )
274     ax2.plot(x1List, x2List,  o- )
275     ax2.axis( equal )
276
277
278     print "[" + str(x1List[-1]) + ", "  + str(x2List[-1]) + "]"
279     plt.show()
280
281 def main():
282     #minfinder(method = steepesDescent)
283     #minfinder(method = conjugateGradient)
284     #minfinder(method=rankone)
285     #minfinder(method= DFP)
286     minfinder(method=BFGS)
287
288 if  __name__ == "__main__":
289     main()
```