

### Problem 1:

$$y_k = ay_{k-1} + bu_k + v_k$$

Then

$$\begin{aligned} y_1 &= ay_0 + bu_1 + v_1 \\ y_2 &= ay_1 + bu_2 + v_2 \\ &= a^2y_0 + abu_1 + av_1 + b_2 + v_2 \\ y_3 &= ay_2 + bu_3 + v_3 \\ &= a^3y_0 + a^2bu_1 + a^2v_1 + abu_2 + av_2 + bu_3 + v_3 \end{aligned}$$

where  $v_k$  represents white noise and  $y_0 = 0$ .

We can write those equation in a matrix form:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} b & 0 & 0 & 0 & \dots & \dots & 0 \\ ab & a & 0 & 0 & \dots & \dots & 0 \\ a^2b & ab & b & 0 & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a^{k-1}b & a^{k-2}b & a^{k-3}b & \dots & \dots & \dots & b \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ \vdots \\ u_k \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & \dots & 0 \\ a & 1 & 0 & 0 & \dots & \dots & 0 \\ a^2 & a & 1 & 0 & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & a^{k-1} & a^{k-2} & a^{k-3} & \dots & \dots & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ \vdots \\ v_k \end{bmatrix}$$

$$\mathbf{y} = \mathbf{C}\mathbf{u} + \mathbf{D}\mathbf{v}$$

$$\mathbf{D}^{-1}\mathbf{y} = \mathbf{D}^{-1}\mathbf{C}\mathbf{u} + \mathbf{v}$$

$$\mathbf{b} = \mathbf{A}\mathbf{x} + \mathbf{v}$$

where

$$\mathbf{C} = \begin{bmatrix} b & 0 & 0 & 0 & \dots & \dots & 0 \\ ab & a & 0 & 0 & \dots & \dots & 0 \\ a^2b & ab & b & 0 & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a^{k-1}b & a^{k-2}b & a^{k-3}b & \dots & \dots & \dots & b \end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & \dots & 0 \\ a & 1 & 0 & 0 & \dots & \dots & 0 \\ a^2 & a & 1 & 0 & \dots & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & a^{k-1} & a^{k-2} & a^{k-3} & \dots & \dots & 1 \end{bmatrix}$$

$$\mathbf{b} = \mathbf{D}^{-1}\mathbf{y}$$

$$\mathbf{A} = \mathbf{D}^{-1}\mathbf{C}$$

Using least square method:

$$u^* = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b}$$

## Problem 2:

Using PSO algorithm, we can find the minimizer:

$$\begin{aligned}x_0 &= 0.000279321964182 \\x_1 &= 0.000193196456243 \\f(x_0, x_1) &= 2.28836604776e - 07\end{aligned}$$

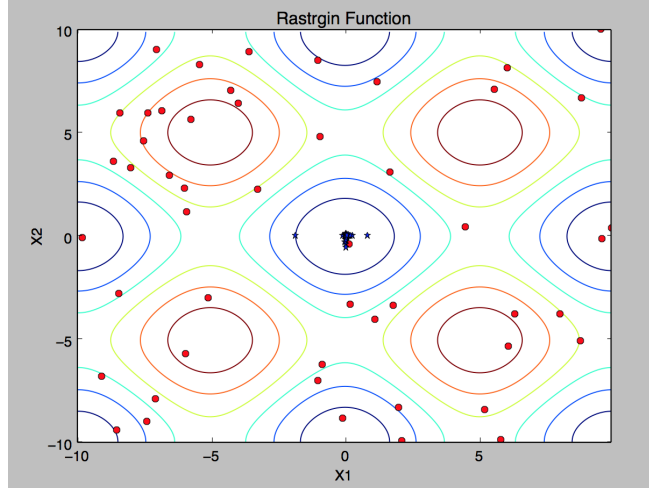


Figure 1: PSO Algorithm (problem 2): circle points are randomly generated 50 initial points. Stars indicate the positions after 50 iterations.

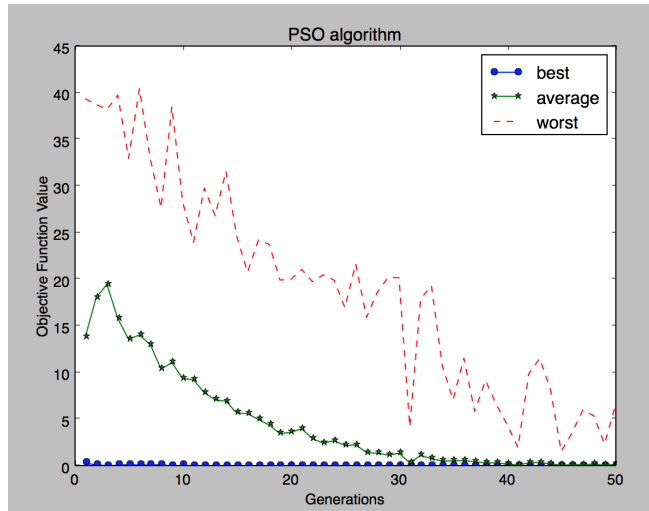


Figure 2: PSO Algorithm (problem 2): plots of the best, average, and the worst objective function values in the population for 50 generations

### Problem 3:

Using PSO algorithm, we can find the maximizer:

$$\begin{aligned}x_0 &= -5.02482780601 \\x_1 &= 5.02524813509 \\f(x_0, x_1) &= -40.5025451078\end{aligned}$$

In fact, there are several other global maximizers. PSO method will converge to different global maximizer depending on the initial points which are randomly chosen.

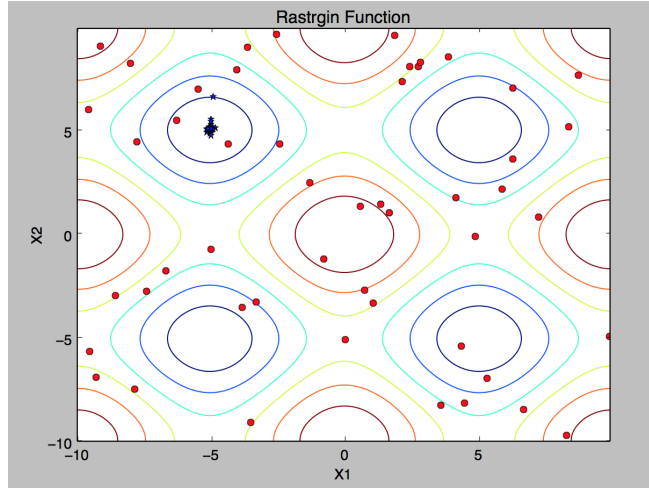


Figure 3: PSO Algorithm (problem 3): circle points are randomly generated 50 initial points. Stars indicate the positions after 50 iterations.

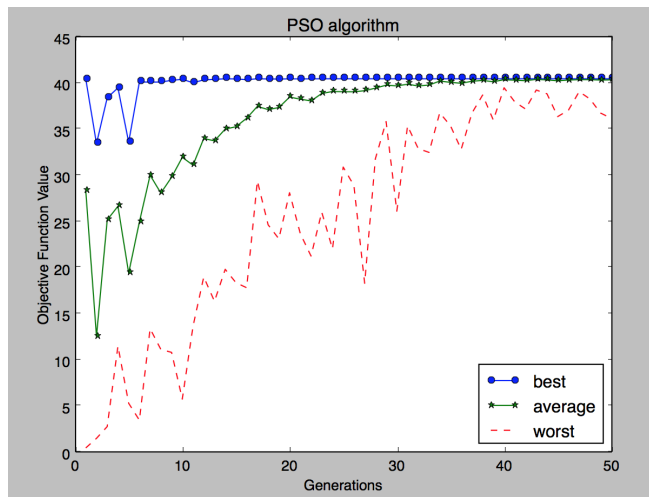


Figure 4: PSO Algorithm (problem 3): plots of the best, average, and the worst objective function values in the population for 50 generations

### Problem 4:

Population size: 50

Number of iterations: 50

For canonical number genetic algorithm, the minimizer is:

$$\begin{aligned}x_1 &= 0.0408935546875 \\x_2 &= 0.0390625 \\f(x_1, x_2) &= 0.00634456702034\end{aligned}$$

For real number genetic algorithm, the minimizer is :

$$\begin{aligned}x_1 &= 0.018313265874 \\x_2 &= 0.0286761643909 \\f(x_1, x_2) &= 0.00229673023909\end{aligned}$$

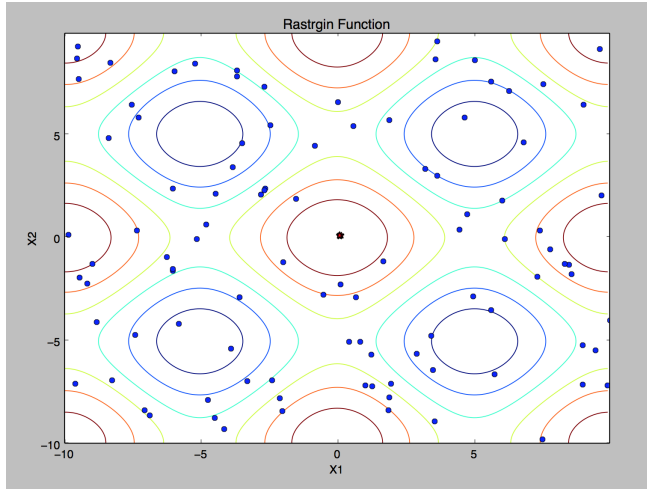


Figure 5: Canonical Genetic Algorithm (problem 4): circle points are randomly generated 50 initial points. Stars indicate the positions after 50 iterations.

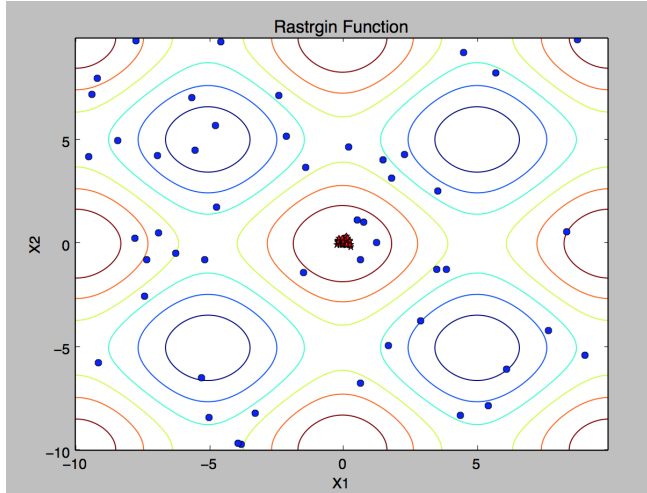


Figure 6: Real Number Genetic Algorithm (problem 4): circle points are randomly generated 50 initial points. Stars indicate the positions after 50 iterations.

## Problem 5:

The shortest path is shown in Figure 9, and the shortest distance is : 37.7222579198

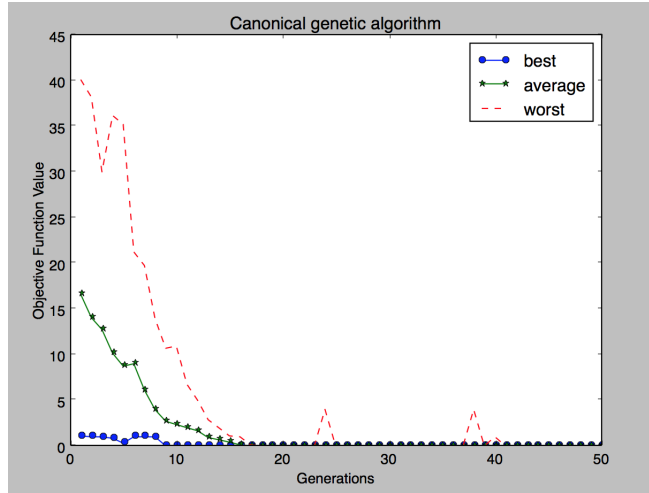


Figure 7: Canonical Genetic Algorithm (problem 4): plots of the best, average, and the worst objective function values in the population for 50 generations

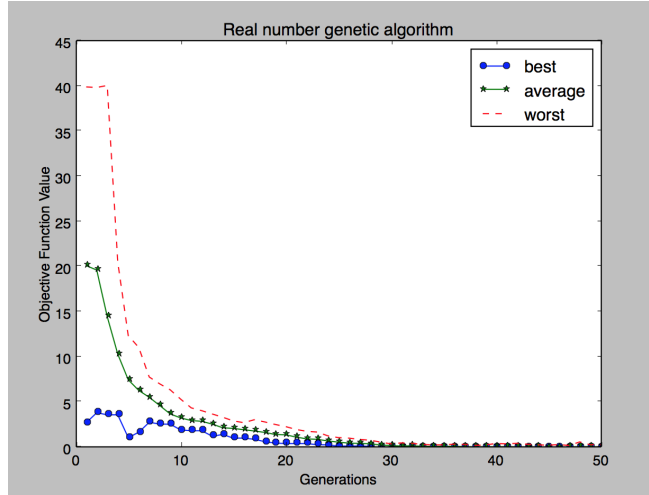


Figure 8: Real Number Genetic Algorithm (problem 4): plots of the best, average, and the worst objective function values in the population for 50 generations

## Problem 6:

Matlab code for Problem 6:

```

1 f = [7 10 14 8 7 11 12 6 5 8 15 9 ];
2 A = [];
3 b = [];
4 Aeq = [1 1 1 1 0 0 0 0 0 0 0 0;
5        0 0 0 0 1 1 1 1 0 0 0 0;
6        0 0 0 0 0 0 0 0 1 1 1 1;
7        1 0 0 0 1 0 0 0 1 0 0 0;
8        0 1 0 0 0 1 0 0 0 1 0 0;
9        0 0 1 0 0 0 1 0 0 0 1 0;
10       0 0 0 1 0 0 0 1 0 0 0 1];
11 beq = [30 40 30 20 20 25 35];
12 lb = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
13 ub = [];
14 x = linprog(f, A, b, Aeq, beq, lb, ub)

```

The output:

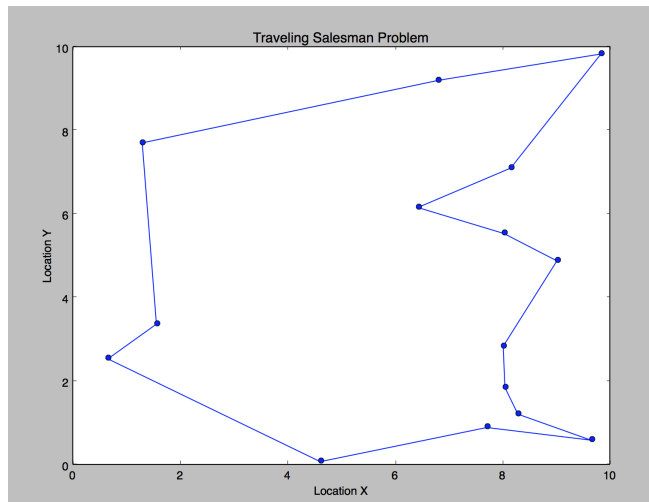


Figure 9: Traveling salesman problem (problem 5): plots of the shortest distance path

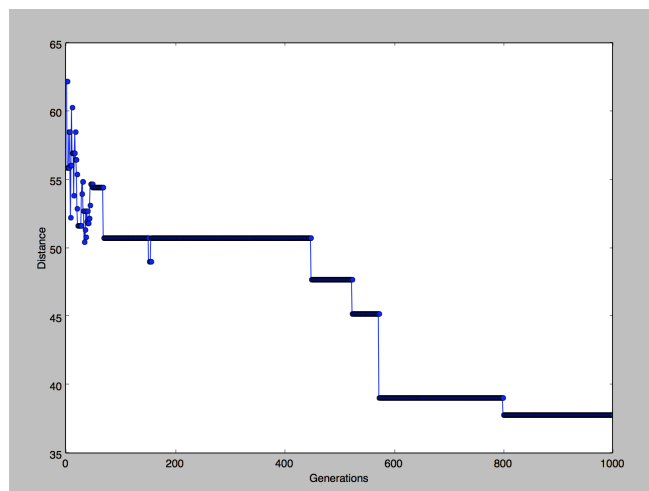


Figure 10: Traveling salesman problem (problem 5): plots of the shortest distance for different combinations of the population for 1000 generations

```

1  x =
2
3      4.8834
4      5.1166
5      8.7304
6      11.2696
7      0.0000
8      0.0000
9      16.2696
10     23.7304
11     15.1166
12     14.8834
13     0.0000
14     0.0000

```

Code problem2:

```

1  import matplotlib.pyplot as plt
2  import numpy as np
3  import sys
4

```

```

5 import numpy as np
6 def func(x1, x2):
7     return -20 - 0.01*x1**2 - 0.01*x2**2 \
8         + 10*(np.cos(0.2*np.pi*x1)+np.cos(0.2*np.pi*x2))
9
10
11 def plotContour(ax, xlim, ylim):
12     Δ = 0.1
13     x = np.arange(xlim[0], xlim[1], Δ)
14     y = np.arange(ylim[0], ylim[1], Δ)
15     X1, X2 = np.meshgrid(x, y)
16     Z = func(X1,X2)
17     ax.set_title("Rastrgin Function")
18     ax.set_xlabel("X1")
19     ax.set_ylabel("X2")
20     cs = ax.contour(X1, X2, Z)
21     return cs
22
23 def getGbest(p0, p1):
24     # find the global best from the personal best.
25     f = []
26     for i in range(len(p0)):
27         f.append(func(p0[i], p1[i]))
28     gBest = min(f)
29     ind = np.argmin(f)
30     g0 = p0[ind]
31     g1 = p1[ind]
32
33     return gBest, g0, g1
34
35 def init(ax, n):
36     #n = 100
37     x0 = 20* (np.random.rand(n)-0.5)
38     x1 = 20* (np.random.rand(n)-0.5)
39     # personal best
40     p0 = x0
41     p1 = x1
42
43     v0 = 0.1*np.random.rand(n)
44     v1 = 0.1*np.random.rand(n)
45     gBest, g0, g1 = getGbest(p0, p1)
46     ax.plot(x0, x1, 'or')
47     pBest = []
48     for i in range(n):
49         pBest.append(func(x0[i], x1[i]))
50     return x0, x1, v0, v1, p0,p1,pBest, g0, g1, gBest
51
52 def getBestPlot(x0_dec, x1_dec):
53
54     f = []
55     for i in range(len(x0_dec)):
56         f.append(func(x0_dec[i], x1_dec[i]))
57     best = min(f)
58     worst = max(f)
59     ave = np.mean(f)
60     ind = np.argmin(f)
61     return ind, best, worst, ave
62
63 def clampVelocity(vCurr, vmax):
64     return min(vmax, max(-vmax, vCurr))
65
66 def pmo(ax):
67     n =50
68     c1 = 2.01
69     c2 = 2.09
70     bestList = []
71     worstList = []
72     aveList = []
73     counter = []
74     x0, x1, v0, v1, p0,p1,pBest, g0, g1, gBest = init(ax, n )
75

```

```

76     k = 0.729
77
78     vmax = 4.0
79     for i in range(50):
80
81         r0 = np.random.rand(n)
82         r1 = np.random.rand(n)
83         s0 = np.random.rand(n)
84         s1 = np.random.rand(n)
85         for j in range(n):
86             v0[j] = k*(v0[j] + c1*r0[j]*(p0[j]-x0[j]) + ...
87                     c2*s0[j]*(g0-x0[j]))
88             v1[j] = k*(v1[j] + c1*r1[j]*(p1[j]-x1[j]) + ...
89                     c2*s1[j]*(g1-x1[j]))
90             # clamping velocity
91             v0[j] = min(vmax, max(-vmax, v0[j]))
92             v1[j] = min(vmax, max(-vmax, v1[j]))
93
94             x0[j] = x0[j] + v0[j]
95             x1[j] = x1[j] + v1[j]
96             val = func(x0[j], x1[j])
97             if val < pBest[j]:
98                 pBest[j] = val
99                 p0[j] = x0[j]
100                p1[j] = x1[j]
101
102            gCurrBest, g0Curr, g1Curr = getGbest(p0, p1)
103            #print g0Curr, g1Curr, gCurrBest, func(g0Curr, g1Curr)
104            if gCurrBest < gBest:
105                gBest = gCurrBest
106                g0 = g0Curr
107                g1 = g1Curr
108                print "gBest", g0, g1, gBest, func(g0, g1)
109
110            ind, best, worst, ave = getBestPlot(x0, x1)
111            counter.append(i+1)
112            bestList.append(-best)
113            worstList.append(-worst)
114            aveList.append(-ave)
115
116            ax.plot(x0, x1, "b")
117            fig2 = plt.figure()
118            ax1 = fig2.add_subplot(111)
119            ax1.plot(counter, bestList, '-o', label='best')
120            ax1.plot(counter, aveList, '-*', label='average')
121            ax1.plot(counter, worstList, '--', label='worst')
122            ax1.set_xlabel("Generations")
123            ax1.set_ylabel("Objective Function Value")
124            ax1.set_title("PSO algorithm")
125            ax1.legend(loc=4)
126            plt.show()
127
128        def main():
129            f, (ax) = plt.subplots(1,1, sharex=False)
130            plotContour(ax, [-10,10] , [-10,10])
131            pmo(ax)
132            plt.show()

```

Canonical GA algorithm (problem 4) code :

```

1  import matplotlib.pyplot as plt
2  import numpy as np
3  from random import sample
4  def func(x1, x2):
5      return -20 - 0.01*x1**2 - 0.01*x2**2 \
6             + 10*(np.cos(0.2*np.pi*x1)+np.cos(0.2*np.pi*x2))
7
8  def plotContour(ax, xlim, ylim):
9      Δ = 0.1

```



```

10     x = np.arange(xlim[0], xlim[1], Δ)
11     y = np.arange(ylim[0], ylim[1], Δ)
12     X1, X2 = np.meshgrid(x, y)
13     Z = func(X1,X2)
14     ax.set_title("Rastrgin Function")
15     ax.set_xlabel("X1")
16     ax.set_ylabel("X2")
17     cs = ax.contour(X1, X2, Z)
18     return cs
19
20
21 def mapping(binNum, n):
22     decVal = 20.0 * binNum/(np.power(2, n-1)) -10
23     return decVal
24
25 def init(ax, n):
26     x0_dec = []
27     x1_dec = []
28     L = 32
29     max = np.power(2, L/2-1)
30     x0_bin = np.random.randint(0, max, size=n)
31     x1_bin = np.random.randint(0, max, size=n)
32     for i in range(n):
33         x0_dec.append(mapping(x0_bin[i], L/2))
34         x1_dec.append(mapping(x1_bin[i], L/2))
35
36     ax.plot(x0_dec, x1_dec, 'ob')
37     return x0_bin, x1_bin, x0_dec, x1_dec
38
39 def selection(x0_bin, x1_bin, x0_dec, x1_dec, k):
40     f = []
41     for i in range(len(x0_dec)):
42         f.append(func(x0_dec[i], x1_dec[i]))
43     base = min(f)
44     f = [x-base for x in f]
45     cumSum = np.cumsum(f)
46     new_x0_dec = []
47     new_x1_dec = []
48     new_x0_bin = []
49     new_x1_bin = []
50     f_index = [0] * k
51
52     alpha = np.random.rand(k)*sum(f)
53     for i in range(k):
54         for j in range(len(x0_dec)):
55             if alpha[i]> cumSum[j]:
56                 f_index[i] += 1
57             else:
58                 break
59
60     for index in f_index:
61         new_x0_dec.append(x0_dec[f_index[index]])
62         new_x1_dec.append(x1_dec[f_index[index]])
63         new_x0_bin.append(x0_bin[f_index[index]])
64         new_x1_bin.append(x1_bin[f_index[index]])
65     return new_x0_bin, new_x1_bin, new_x0_dec, new_x1_dec
66
67
68 def crossover(x0_bin, x1_bin, x0_dec, x1_dec, n):
69
70     pc = 0.65
71     pm = 0.0075
72
73     # choosing crossing site;
74     N = 2* int(pc * n/2) # number of parents who are ready to ...
75     do crossover.
76     L = 32
77     parent_index = sample(xrange(len(x0_dec)), N)
78     i=0
79     while i<len(parent_index):
80         parent1 = "{0:b}".format( ...

```

```

        x0_bin[parent_index[i]]*np.power(2, L/2) + ...
        x1_bin[parent_index[i]]).zfill(L)
80     parent2 = "{0:b}".format( ...
        x0_bin[parent_index[i+1]]*np.power(2, L/2)+ ...
        x1_bin[parent_index[i+1]]).zfill(L)
81     site = np.random.randint(0,L-1)
82     child1 = parent1[0:site] + parent2[site:L]
83     child2 = parent2[0:site] + parent1[site:L]
84
85
86     x0_bin[parent_index[i]] = int(child1[0:L/2], 2)
87     x0_bin[parent_index[i+1]] = int(child2[L/2:L], 2)
88     x1_bin[parent_index[i]] = int(child1[L/2:L], 2)
89     x1_bin[parent_index[i+1]] = int(child2[0:L/2], 2)
90     i+=2
91
92     ## Mutation
93     for i in range(len(x0_bin)):
94         s = "{0:b}".format(x0_bin[i]).zfill(L/2)
95         new_s = ""
96         for j in range(len(s)):
97             val = np.random.rand()
98             if val < pm:
99                 new_s += str(1- int(s[j]))
100            else:
101                new_s += s[j]
102            x0_bin[i] = int(new_s, 2)
103
104            s1 = "{0:b}".format(x1_bin[i]).zfill(L/2)
105            new_s1 = ""
106            for j in range(len(s1)):
107                val = np.random.rand()
108                if val < pm:
109                    new_s1 += str(1- int(s1[j]))
110                else:
111                    new_s1 += s1[j]
112
113            x1_bin[i] = int(new_s1, 2)
114
115            #update x0_dec and x1_dec:
116            for i in range(len(x0_bin)):
117                x0_dec[i] = mapping(x0_bin[i], L/2)
118                x1_dec[i] = mapping(x1_bin[i], L/2)
119
120
121            return x0_bin, x1_bin, x0_dec, x1_dec
122
123    def getBestPlot(x0_dec, x1_dec):
124
125        f = []
126        for i in range(len(x0_dec)):
127            f.append(-func(x0_dec[i], x1_dec[i]))
128        best = min(f)
129        worst = max(f)
130        ave = np.mean(f)
131        ind = np.argmin(f)
132        return ind, best, worst, ave
133
134    def ga(ax):
135        n = 50 # number of initial points.
136        bestList = []
137        worstList = []
138        aveList = []
139        counter = []
140
141        x0_bin, x1_bin, x0_dec, x1_dec = init(ax, n)
142
143        for i in range(50):
144            x0_bin, x1_bin, x0_dec, x1_dec = selection(x0_bin, ...
145                x1_bin, x0_dec, x1_dec, k=len(x0_dec))
146            x0_bin, x1_bin, x0_dec, x1_dec = crossover(x0_bin, ...

```

```

        x1_bin, x0_dec, x1_dec, n) # crossover and mutation
146
147     ind, best, worst, ave = getBestPlot(x0_dec, x1_dec)
148
149     counter.append(i+1)
150     bestList.append(best)
151     worstList.append(worst)
152     aveList.append(ave)
153
154     print x0_dec[ind], x1_dec[ind], -func(x0_dec[ind], x1_dec[ind])
155     plt.plot(x0_dec, x1_dec, '*r')
156
157     fig2 = plt.figure()
158     ax1 = fig2.add_subplot(111)
159     ax1.plot(counter, bestList, '-o', label='best')
160     ax1.plot(counter, aveList, '-*', label='average')
161     ax1.plot(counter, worstList, '--', label='worst')
162     ax1.set_xlabel("Generations")
163     ax1.set_ylabel("Objective Function Value")
164     ax1.set_title("Canonical genetic algorithm")
165     ax1.legend()
166     plt.show()
167
168 def main():
169     f, (ax) = plt.subplots(1,1, sharex=False)
170     plotContour(ax, [-10,10] , [-10,10])
171     ga(ax)
172     plt.show()
173
174 if __name__ == "__main__":
175     main()

```

Real number GA algorithm:

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 from random import sample
4 def func(x1, x2):
5     return -20 - 0.01*x1**2 - 0.01*x2**2 \
6         + 10*(np.cos(0.2*np.pi*x1)+np.cos(0.2*np.pi*x2))
7
8
9 def plotContour(ax, xlim, ylim):
10     Δ = 0.1
11     x = np.arange(xlim[0], xlim[1], Δ)
12     y = np.arange(ylim[0], ylim[1], Δ)
13     X1, X2 = np.meshgrid(x, y)
14     Z = func(X1,X2)
15     ax.set_title("Rastrgin Function")
16     ax.set_xlabel("X1")
17     ax.set_ylabel("X2")
18     cs = ax.contour(X1, X2, Z)
19     return cs
20
21 def mapping(binNum, n):
22     decVal = 20.0 * binNum/(np.power(2, n-1)) -10
23     return decVal
24
25
26 def init(ax, n):
27     x0_dec = []
28     x1_dec = []
29     L = 32
30     max = np.power(2, L/2-1)
31     x0_bin = np.random.randint(0, max, size=n)
32     x1_bin = np.random.randint(0, max, size=n)
33     for i in range(n):
34         x0_dec.append(mapping(x0_bin[i], L/2))
35         x1_dec.append(mapping(x1_bin[i], L/2))

```

```

36
37     ax.plot(x0_dec, x1_dec, 'ob')
38     return x0_bin, x1_bin, x0_dec, x1_dec
39
40 def selection(x0_dec, x1_dec, k):
41     f = []
42     for i in range(len(x0_dec)):
43         f.append(func(x0_dec[i], x1_dec[i]))
44     base = min(f)
45     f = [x-base for x in f]
46     cumSum = np.cumsum(f)
47     new_x0_dec = []
48     new_x1_dec = []
49     #new_x0_bin = []
50     #new_x1_bin = []
51     f_index = [0] * k
52
53
54     alpha = np.random.rand(k)*sum(f)
55     for i in range(k):
56         for j in range(len(x0_dec)):
57             if alpha[i]> cumSum[j]:
58                 f_index[i] += 1
59             else:
60
61                 break
62
63     for index in f_index:
64         new_x0_dec.append(x0_dec[f_index[index]])
65         new_x1_dec.append(x1_dec[f_index[index]])
66         #new_x0_bin.append(x0_bin[f_index[index]])
67         #new_x1_bin.append(x1_bin[f_index[index]])
68     return new_x0_dec, new_x1_dec
69
70
71 def crossover(x0_dec, x1_dec, n):
72
73     pc = 0.65
74     pm = 0.075
75
76     # choosing crossing site;
77     N = 2* int(pc * n/2) # number of parents who are ready to ...
78     do crossover.
79     L = 32
80     parent_index = sample(xrange(len(x0_dec)), N)
81     i=0
82     while i<len(parent_index):
83         parent1_x0 = x0_dec[parent_index[i]]
84         parent1_x1 = x1_dec[parent_index[i]]
85         parent2_x0 = x0_dec[parent_index[i+1]]
86         parent2_x1 = x1_dec[parent_index[i+1]]
87
88         w = np.random.normal(0, 0.1, 4)
89         child1_x0 = (parent1_x0+parent2_x0)/2 + w[0]
90         child1_x1 = (parent1_x1+parent2_x1)/2 + w[1]
91         child2_x0 = (parent1_x0+parent2_x0)/2 + w[2]
92         child2_x1 = (parent1_x1+parent2_x1)/2 + w[3]
93
94         x0_dec[parent_index[i]] = child1_x0
95         x1_dec[parent_index[i]] = child1_x1
96         x0_dec[parent_index[i+1]] = child2_x0
97         x1_dec[parent_index[i+1]] = child2_x1
98         i+=2
99     ## Mutation
100     for i in range(len(x0_dec)):
101         val = np.random.rand()
102
103         if val < pm :
104             w = np.random.normal(0, 0.1, 2)
105             x0_dec[i] += w[0]
106             x1_dec[i] += w[1]

```

```

106     return x0_dec, x1_dec
107
108 def getBestPlot(x0_dec, x1_dec):
109     f = []
110     for i in range(len(x0_dec)):
111         f.append(-func(x0_dec[i], x1_dec[i]))
112     best = min(f)
113     worst = max(f)
114     ave = np.mean(f)
115     ind = np.argmin(f)
116     return ind, best, worst, ave
117
118
119 def ga(ax):
120     n = 50    # number of initial points.
121
122     bestList = []
123     worstList = []
124     aveList = []
125     counter = []
126
127     x0_bin, x1_bin, x0_dec, x1_dec = init(ax, n)
128
129     for i in range(50):
130         x0_dec, x1_dec = selection(x0_dec, x1_dec, k=len(x0_dec))
131         x0_dec, x1_dec = crossover(x0_dec, x1_dec, n) # ...
132         crossover and mutation
133         ind, best, worst, ave = getBestPlot(x0_dec, x1_dec)
134
135         counter.append(i+1)
136         bestList.append(best)
137         worstList.append(worst)
138         aveList.append(ave)
139     print x0_dec[ind], x1_dec[ind], -func(x0_dec[ind], x1_dec[ind])
140     plt.plot(x0_dec, x1_dec, '*r')
141
142     fig2 = plt.figure()
143     ax1 = fig2.add_subplot(111)
144     ax1.plot(counter, bestList, '-o', label='best')
145     ax1.plot(counter, aveList, '-*', label='average')
146     ax1.plot(counter, worstList, '--', label='worst')
147     ax1.set_xlabel("Generations")
148     ax1.set_ylabel("Objective Function Value")
149     ax1.set_title("Real number genetic algorithm")
150     ax1.legend()
151     plt.show()
152
153 def main():
154     f, (ax) = plt.subplots(1,1, sharex=False)
155     plotContour(ax, [-10,10] , [-10,10])
156     ga(ax)
157     plt.show()

```

Traveling salesman problem (problem 5):

```

1
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import random
5 from random import shuffle
6 import sys
7
8 def getDist(comb):
9     locX = zip(*comb)[0]
10    locY = zip(*comb)[1]
11    dist = 0
12    for i in range(len(locX)-1):
13        dx = locX[i]-locX[i+1]
14        dy = locY[i]-locY[i+1]
15        dist += np.sqrt(dx**2+dy**2)

```

```

16     dist += np.sqrt((locX[-1]-locX[0])**2 + (locY[-1]-locY[0])**2)
17
18     return dist
19
20 def plotConnection(comb):
21     locX = zip(*comb)[0]
22     locY = zip(*comb)[1]
23     for i in range(len(locX)-1):
24         plt.plot((locX[i], locX[i+1]), (locY[i], locY[i+1]), 'o-b')
25     plt.plot((locX[0], locX[-1]), (locY[0], locY[-1]), 'o-b')
26     plt.xlabel("Location X")
27     plt.ylabel("Location Y")
28     plt.title("Traveling Salesman Problem")
29     plt.show()
30
31
32 def switchByIndex(ind1, ind2, List):
33     temp = List[ind2]
34     List[ind2] = List[ind1]
35     List[ind1] = temp
36
37     return List
38
39 def init(locX, locY, num):
40     zList = []
41     for i in range(num):
42         z = zip(locX, locY)
43         shuffle(z)
44         zList.append(z)
45     return zList
46
47 def listInsert(l, l1, pivot):
48     # insert l1 into l starting from pivot points:
49     assert (pivot<len(l))
50     new_list = []
51     for i in range(pivot):
52         new_list.append(l[i])
53
54     new_list.extend(l1)
55     for i in np.arange(pivot, len(l)):
56         new_list.append(l[i])
57     return new_list
58
59 def selection(zList):
60     f = []
61     k = len(zList)
62     for comb in zList:
63         f.append(-getDist(comb))
64     base = min(f)
65     f = [x-base for x in f]
66     cumSum = np.cumsum(f)
67     alpha = np.random.rand(k)*sum(f)
68     f_index = [0]* k
69     for i in range(len(alpha)):
70         for j in range(len(cumSum)):
71             if alpha[i] > cumSum[j]:
72                 f_index[i] += 1
73             else:
74                 break
75     new_zList =[]
76     for ind in f_index:
77         new_zList.append(zList[ind])
78
79     return new_zList
80
81 def crossover(zList):
82     pc = 0.75
83     N = int(pc*len(zList)/2)*2    # number of parents (even number)
84     numSub = int(len(zList[0])/3)
85
86     parent_index = random.sample(xrange(len(zList)), N)

```

```

87     i=0
88     while i<len(parent_index):
89         pivot1 = random.sample(xrange(1,len(zList[0])*2/3-1), 1)[0]
90         pivot2 = random.sample(xrange(1,len(zList[0])*2/3-1), 1)[0]
91         #print pivot
92         parent1 = zList[parent_index[i]]
93         parent2 = zList[parent_index[i+1]]
94
95         snap1 = [x for x in parent2 if x not in parent1[pivot1: ...
96                   pivot1+numSub]]
97         snap2 = [x for x in parent1 if x not in parent2[pivot2: ...
98                   pivot2+numSub]]
99
100        zList[parent_index[i]] = listInsert(snap1, ...
101        parent1[pivot1:pivot1+numSub], pivot1)
102        zList[parent_index[i+1]] = listInsert(snap2, ...
103        parent2[pivot2:pivot2+numSub], pivot2)
104
105        #print zList[parent_index[i]]
106        #print zList[parent_index[i+1]]
107
108        i+=2
109    return zList
110
111 def mutation(zList):
112     pm = 0.0075
113
114     #N = int(pc*len(zList))    # number of single parents.
115
116     for i in range(len(zList)):
117         m = np.random.rand()
118         #print m
119         if m<pm:
120             pivot = random.sample(xrange(1,len(zList[0])), 2)
121             zList[i] = switchByIndex(pivot[0], pivot[1], zList[i])
122
123     return zList
124
125 def GA_algorithm(locX, locY):
126     num = 50    # initiate 50 combinations
127
128     zList = init(locX, locY, num)
129
130     counter = []
131     best = []
132
133     for i in range(1000):
134         counter.append(i+1)
135         zList = selection(zList)
136         zList = crossover(zList)
137         zList = mutation(zList)
138         bestDist = sys.maxint
139         for comb in zList:
140             d = getDist(comb)
141             if d<bestDist:
142                 bestDist = d
143                 bestPath = comb
144         best.append(bestDist)
145     print bestDist
146     plt.plot(counter, best, '-')
147     plt.xlabel("Generations")
148     plt.ylabel("Distance")
149     plt.show()
150
151     plotConnection(bestPath)
152     return locX, locY
153
154 def main():
155

```

```
154     locX = [7.7176, 9.8397, 1.3001, 4.6141, 9.0204, 1.5651, 6.4311,  
155             8.0298, 8.2941, 0.6621, 9.6546, 6.8045, 8.0196, ...  
             8.1590, 8.0531]  
156     locY = [0.8955, 9.8352, 7.7070, 0.0784, 4.8838, 3.3703, 6.1604,  
157             5.5397, 1.2147, 2.5383, 0.5923, 9.2022, 2.8386, ...  
             7.1041, 1.8498]  
158     GA_algorithm(locX, locY)
```