

ECE 580: Homework 3

Rahul Deshmukh

April 17, 2020

Exercise 1

For this problem, we need to construct two matrices A_1, A_2 such that $A_2^\dagger A_1^\dagger \neq (A_1 A_2)^\dagger$.

We can construct A_1, A_2 randomly with the constraint that atleast one of them is rank deficient. The script at Listing 1 at page 9 generates two random vectors $a_1, a_2 \in \mathbb{R}^2$ and then constructs the matrices A_1, A_2 by taking the outer product of the vectors. We then evaluate the LHS: $A_2^\dagger A_1^\dagger$ and the RHS: $(A_1 A_2)^\dagger$. It turns out that LHS \neq RHS. To illustrate, the computation for default seed is shown below:

$$a_1 = [0.8147 \quad 0.9058]^T$$

$$a_2 = [0.1270 \quad 0.9134]^T$$

$$A_1 = a_1 \otimes a_1 = \begin{bmatrix} 0.6638 & 0.7380 \\ 0.7380 & 0.8205 \end{bmatrix}$$

$$A_2 = a_2 \otimes a_2 = \begin{bmatrix} 0.0161 & 0.1160 \\ 0.1160 & 0.8343 \end{bmatrix}$$

$$A_2^\dagger A_1^\dagger = \begin{bmatrix} 0.0604 & 0.0672 \\ 0.4348 & 0.4834 \end{bmatrix} \tag{1}$$

$$(A_1 A_2)^\dagger = \begin{bmatrix} 0.0881 & 0.0979 \\ 0.6334 & 0.7042 \end{bmatrix} \tag{2}$$

Therefore from Eq. 1 & Eq. 2 we can see that $A_2^\dagger A_1^\dagger \neq (A_1 A_2)^\dagger$. The output of the script is at Listing 2 at page 9. We will observe the same phenomenon for all randomly generated matrices (you need to comment out line 3 in Listing 1 at page 9).

Exercise 2

For Exercise 2 & Exercise 3 we will be working with the Griewank function which is defined by the Listing 4 at page 12. The function plot over the domain $[-5, 5] \times [-5, 5]$ is at Figure 1.

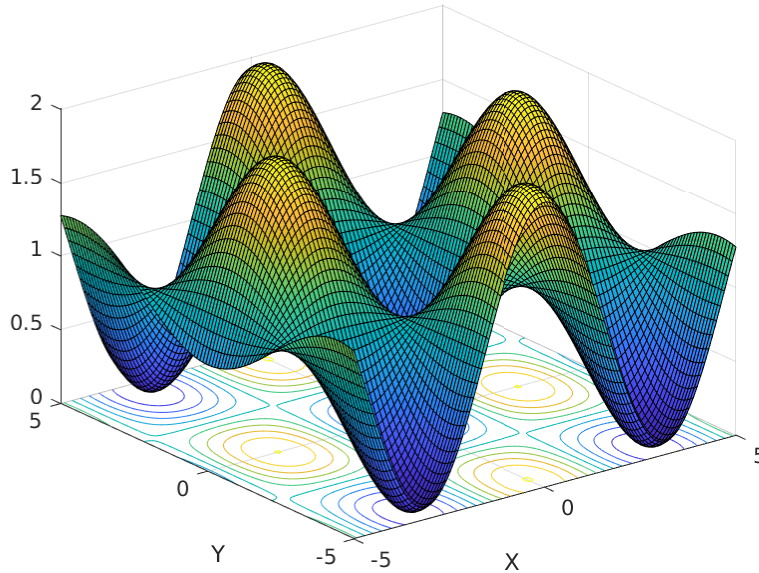


Figure 1: Surface plot of Griewank function

In my particle swarm algorithm, I am using the following parameter settings:

- Swarm size: $d = 40$
- Number of iterations: 100
- Inertial constant: $\omega = 0.8$
- Cognitive constant: $c_1 = 2$
- Social constant: $c_2 = 2$

After several trials, I get the optimal solution as $\mathbf{x} = [0.0007 \ 0.0023]^T$ with a function value $1.6217\text{e-}06$. It should be noted that the PSO algorithm can get stuck in local minima as it not gauranteed to find the global minima due to finite size of the population and finite number of iterations.

The location of the optimal solution on contour plot is at Figure 2.

The plot for best, average, and the worst objective function values in the population for every generation is at Figure 3

For MATLAB function for this problem refer to Listing 5 at page 12 & Listing 4 at page 12 and the call to the function can be referred at Listing 3 at page 10 with corresponding output at Listing 15 at page 20.

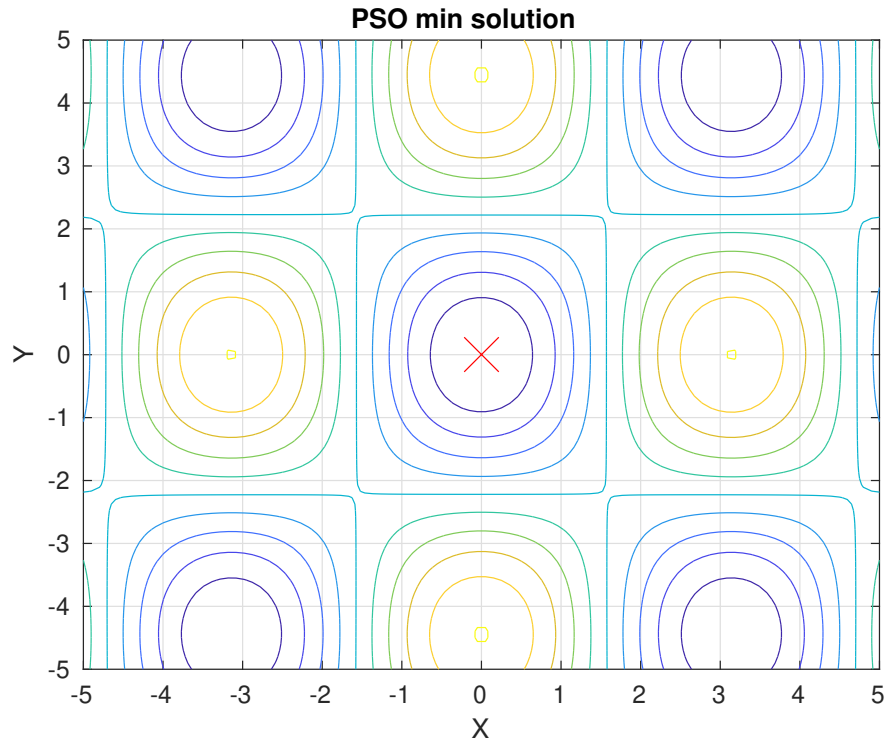


Figure 2: Plot of optimal solution(red X) on contours of objective function

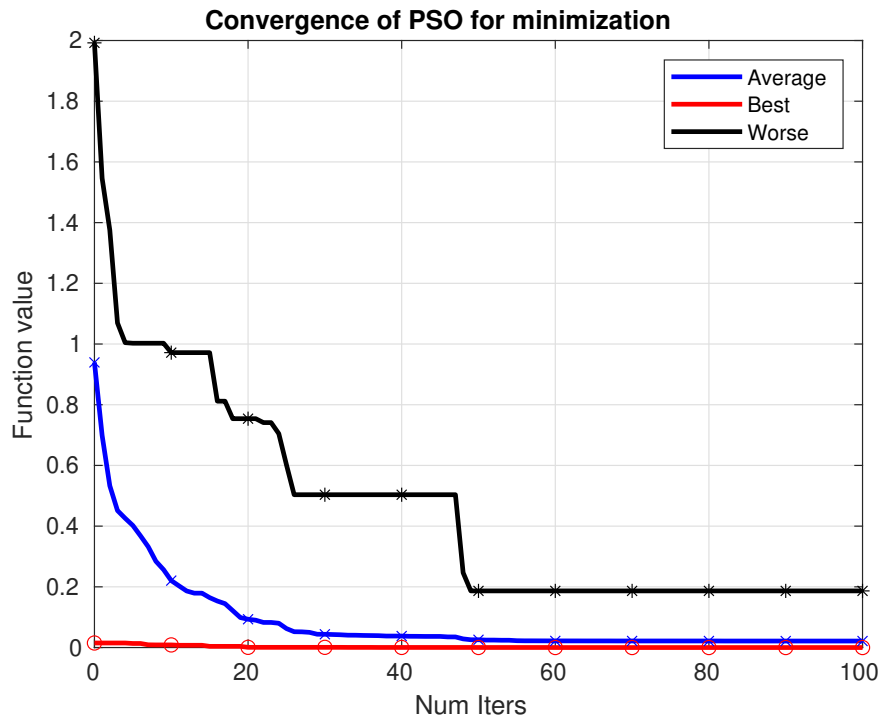


Figure 3: Plot of Average, Best and Worse function values for PSO

Exercise 3

For Maximization problem, we just multiply the Griewank function with negative one and then minimize it with the same parameter settings. After several trials, I get the optimal solution as $\mathbf{x} = [0.0000 \ 4.4474]^T$ with a function value 2.0049.

The location of the optimal solution on contour plot is at Figure 4.

The plot for best, average, and the worst objective function values in the population for every generation is at Figure 5.

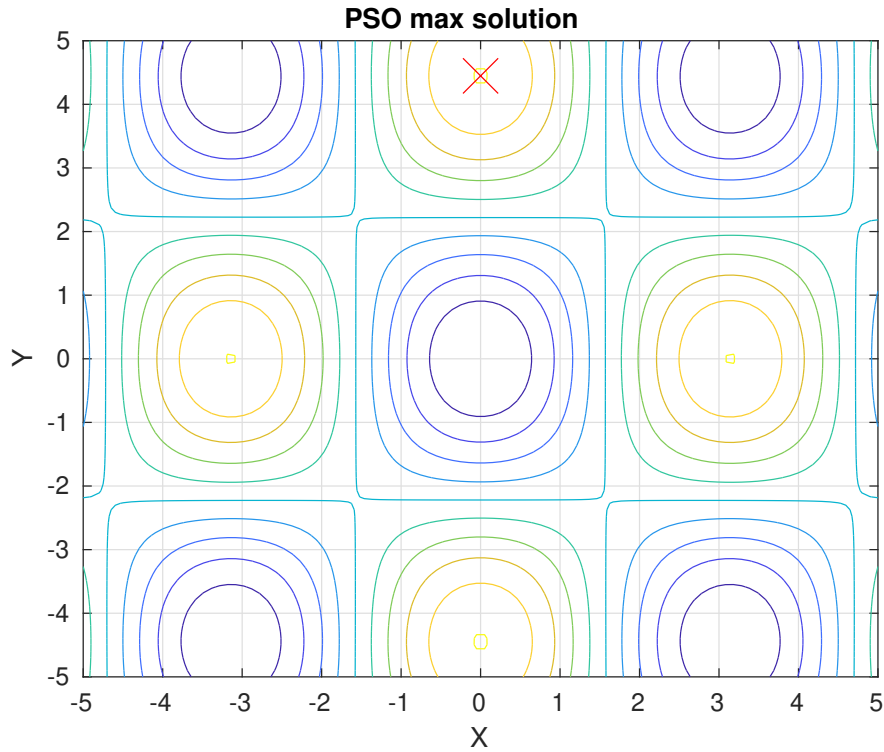


Figure 4: Plot of optimal solution (red X) on contours of objective function

For MATLAB function for this problem refer to Listing 5 at page 12 & Listing 4 at page 12 and the call to the function can be referred at Listing 3 at page 10 with corresponding output at Listing 15 at page 20.

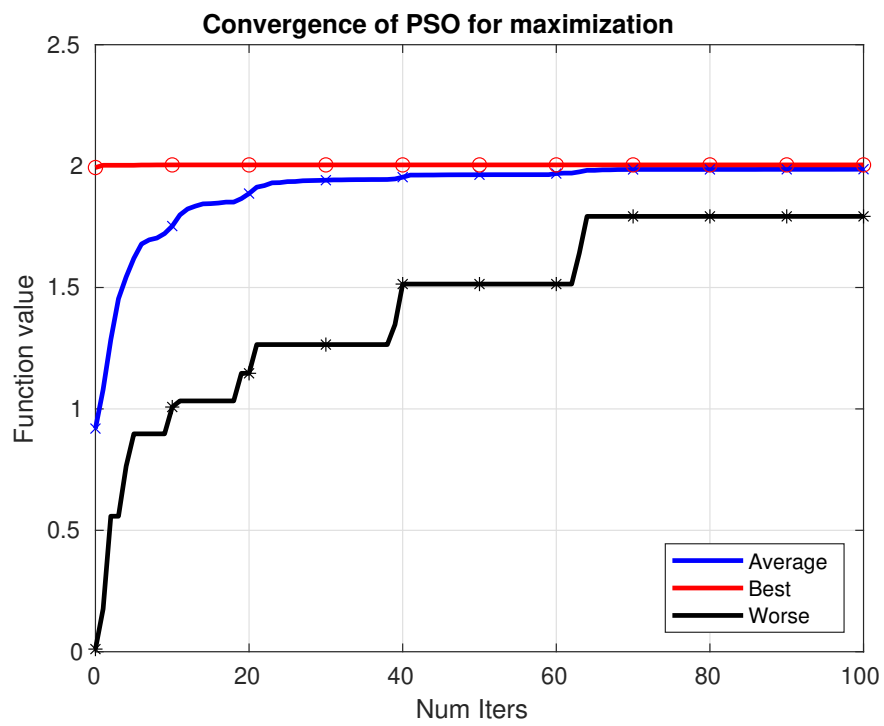


Figure 5: Plot of Average, Best and Worse function values for PSO

Exercise 4

In my GA algorithm, I am using the following parameter settings:

- Population size: 100
- Number of iterations: 200
- Probability for cross-over: 0.8

For the TSP, our design variable (\mathbf{x}) is a 10 dimensional vector with each individual component (x_i) indicating the city visited at the i^{th} turn. We can have a total of $9! = 362880$ possible routes.

To obtain the initial population we randomly permute the numbers in the range 1-10 and then proceed with fitness evaluation. We then carry out selection, cross-over, elitism and fitness evaluation repeatedly till the number of iterations are satisfied.

For crossover, we don't want to carry-out an operation which might result in an in-feasible sample. For example, with 10 cities and a resolution of 1 we can represent the decimal number with 4 bits. However, the coded word cannot be binary representations of numbers greater than 9. Therefore to avoid such a problem, we carry out cross-over by just inverting the visiting order between two randomly chosen coordinates of parent-vector.

For selection, I am using the method-2 of tournament-selection.

After carrying out several trials, I obtain a shortest route of 27.2133. The order of cities for this route is [9 10 1 6 2 5 3 4 8 7].

The shortest route found using GA is at Figure 6. The plot for best, average, and the worst objective function values in the population for every generation is at Figure 7.

For main file for GA refer to Listing 6 at page 15. The fitness function can be referred at Listing 7 at page 17. The encoding and decoding functions can be found at Listing 8 at page 17 & Listing 9 at page 17 respectively. The function for Tournament selection is at Listing 11 at page 18. The function for crossover and elitism can be referred at Listing 12 at page 19 & Listing 13 at page 19 respectively.

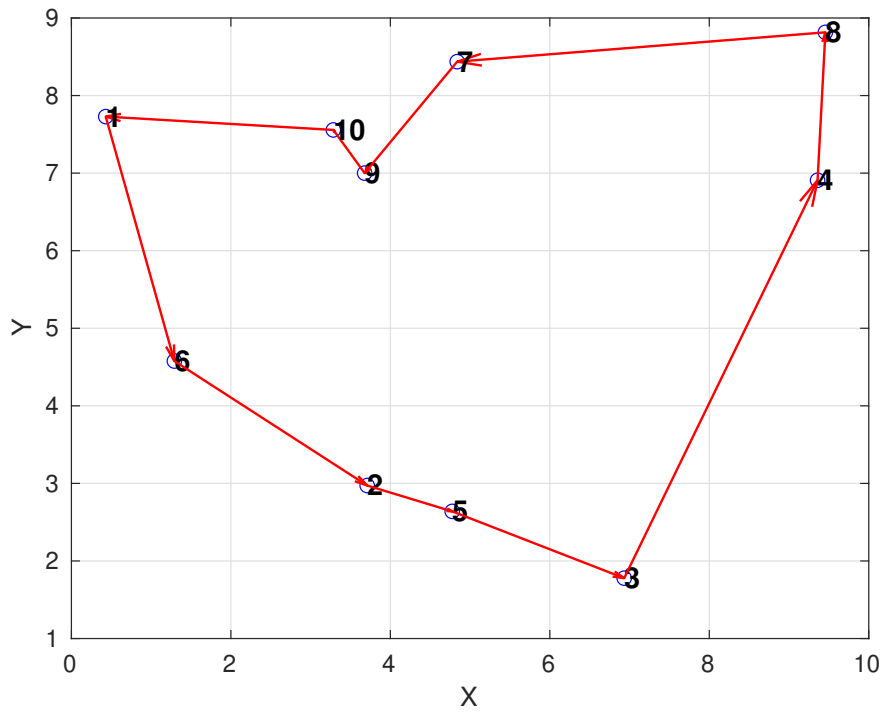


Figure 6: Shortest route calculated using GA

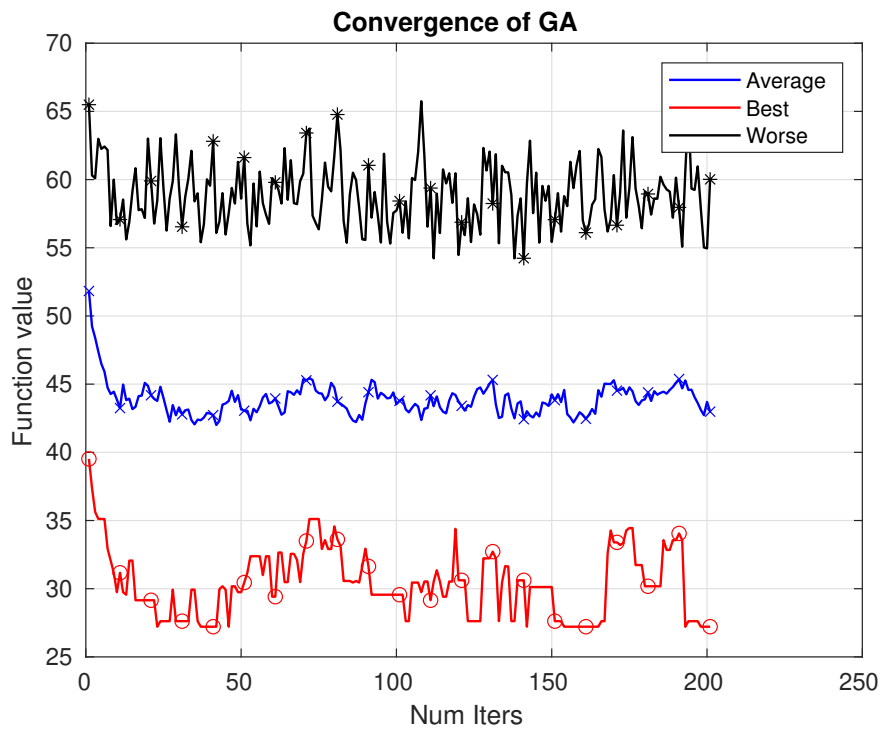


Figure 7: Plot of Average, Best and Worse function values for GA

Exercise 5

For this problem we are required to solve the following problem

$$\begin{aligned} \mathbf{x}^* &= \underset{\mathbf{x}}{\operatorname{argmax}} \mathbf{c}^T \mathbf{x} \\ \text{subject to } A\mathbf{x} &\leq \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

where

$$\begin{aligned} \mathbf{c}^T &= [6 \quad 4 \quad 7 \quad 5] \\ A &= \begin{bmatrix} 1 & 2 & 1 & 2 \\ 6 & 5 & 3 & 2 \\ 3 & 4 & 9 & 12 \end{bmatrix} \\ \mathbf{b} &= \begin{bmatrix} 20 \\ 100 \\ 75 \end{bmatrix} \end{aligned}$$

We convert the above problem to a minimization problem by multiplying \mathbf{c}^T by negative one and then solve using MATLAB's `linprog()` function which works only for a minimization problem.

We obtain an optimal solution as $\mathbf{x}^* = [15.0 \quad 0.0 \quad 3.3333 \quad 0.0]^T$ with the maximum function value of 113.3333.

The MATLAB code for `linprog` can be found at Listing 3 at page 10 with corresponding output at Listing 15 at page 20.

MATLAB Code

Listing 1: Problem-1 code

```
1 clc; clear all;
2 format short;
3 rng('default')
4
5 a_1 = rand(2,1)
6 a_2 = rand(2,1)
7 A_1 = a_1*a_1'
8 A_2 = a_2*a_2'
9
10 A_1_pinv = pinv(A_1);
11 A_2_pinv =pinv(A_2);
12
13 A_2_pinv_A_1_pinv = A_2_pinv*A_1_pinv
14 A_1A_2_pinv = pinv(A_1*A_2)
15
16 diff = A_1A_2_pinv - A_2_pinv_A_1_pinv
17
18 if abs(sum(diff))> 1e-16
19     fprintf('pinv(A_2)pinv(A_1) not equals pinv(A_1A_2)\n')
20 end
```

Listing 2: Output of Problem-1 (for default seed)

```
1 a_1 =
2
3     0.8147
4     0.9058
5
6
7 a_2 =
8
9     0.1270
10    0.9134
11
12
13 A_1 =
14
15     0.6638     0.7380
16     0.7380     0.8205
17
18
19 A_2 =
20
21     0.0161     0.1160
22     0.1160     0.8343
23
24
25 A_2_pinv_A_1_pinv =
26
27     0.0604     0.0672
28     0.4348     0.4834
29
```

```

30
31 A_1A_2_pinv =
32
33     0.0881    0.0979
34     0.6334    0.7042
35
36
37 diff =
38
39     0.0276    0.0307
40     0.1986    0.2208
41
42 pinv(A_2)pinv(A_1) not equals pinv(A_1A_2)

```

Listing 3: Main Code

```

1  % ECE 580 HW4
2  % Rahul Deshmukh
3  % deshmun5@purdue.edu
4  clc; clear all; close all;
5  format short;
6  %% include paths
7  addpath('.../OptimModule/optimizers/global/');
8  save_dir = './pix/';
9
10 %% Problem 2: PSO min
11 %plot griewank fun
12 x = linspace(-5,5,100);
13 [X,Y] = meshgrid(x,x);
14 [h,w] = size(X);
15 Z = zeros(h,w);
16 for ih=1:h
17     for iw=1:w
18         Z(ih,iw) = griewank_fun([X(ih,iw);Y(ih,iw)]);
19     end
20 end
21 fig = figure(1);
22 surf(X,Y,Z);grid on;
23 view(3);
24 xlabel('X');
25 ylabel('Y');
26 saveas(fig, strcat(save_dir, 'surf_plot'), 'epsc');
27
28 a = [-5;-5];
29 b = [5;5];
30 [x_star_min, history_min] = particleswarm(@(x)griewank_fun(x), a, b);
31 x_star_min
32 fval = history_min.data(history_min.Niters+1).gbest_fval
33 fig2= figure(2);
34 hold on;grid on;
35 pso_conv_plot(history_min,1);
36 hold off;
37 box('on');
38 xlabel('Num ITERS'); ylabel('Function value');
39 title('Convergence of PSO for minimization');
40 saveas(fig2, strcat(save_dir, 'plot_pso_min'), 'epsc');
41
42 fig3= figure(3);

```

```

43 hold on;grid on;
44 contour(X,Y,Z);
45 plot_pso_traj(history_min);
46 xlabel('X'); ylabel('Y');
47 title('PSO min solution')
48 hold off;
49 xlim([a(1),b(1)]);
50 ylim([a(2),b(2)]);
51 xticks(a(1):1:b(1));
52 yticks(a(2):1:b(2));
53 box('on');
54 saveas(fig3, strcat(save_dir, 'pso.min-traj'), 'eps')
55
56 %% Problem 3: PSO max
57 [x_star_max, history_max] = particleswarm(@(x)griewank_fun(x,0), a, b);
58 x_star_max
59 fval = -1*history_max.data(history_max.Niters+1).gbest_fval
60 fig4= figure(4);
61 hold on;grid on;
62 pso_conv_plot(history_max,0);
63 hold off;
64 box('on');
65 xlabel('Num Iters'); ylabel('Function value');
66 title('Convergence of PSO for maximization');
67 saveas(fig4, strcat(save_dir, 'plot.pso.max'), 'eps');
68
69 fig5= figure(5);
70 hold on;grid on;
71 contour(X,Y,Z);
72 plot_pso_traj(history_max);
73 xlabel('X'); ylabel('Y');
74 title('PSO max solution')
75 hold off;
76 xlim([a(1),b(1)]);
77 ylim([a(2),b(2)]);
78 xticks(a(1):1:b(1));
79 yticks(a(2):1:b(2));
80 box('on');
81 saveas(fig5, strcat(save_dir, 'pso.max-traj'), 'eps')
82
83 %% Problem 5: Linprog
84 fprintf('-----Linear Programming-----');
85 A = [1, 2, 1, 2;
86      6, 5, 3, 2;
87      3, 4, 9, 12];
88 b = [20; 100; 75];
89 Aeq= []; beq = [];
90 lb = [0, 0, 0, 0];
91 ub = Inf*[1, 1, 1, 1];
92 c = [6, 4, 7, 5];
93 [x_star_linprog, fval] = linprog(-1*c, A, b, Aeq, beq, lb, ub);
94 x_star_linprog
95 -1*fval
96
97 %% Local helper functions for plotting
98 % plotting for PSO
99 function pso_conv_plot(history, min_bool)
100     av = [];
101     gbest = [];

```

```

102     worse = [];
103     for i=1:history.Niters + 1
104         av = [av; history.data(i).pbest_av];
105         gbest = [gbest; history.data(i).gbest_fval];
106         worse = [worse; history.data(i).pbest_worse];
107     end
108     if ~min_bool
109         av=-1*av; gbest = -1*gbest; worse = -1*worse;
110     end
111     x = 0:1:history.Niters;
112     h1 =plot(x,av,'-b','LineWidth',2);
113     h2 = plot(x,gbest,'-r','LineWidth',2);
114     h3 = plot(x,worse,'-k','LineWidth',2);
115     v = 1:10:history.Niters+1;
116     plot(x(v),av(v),'bx');
117     plot(x(v),gbest(v),'ro');
118     plot(x(v),worse(v),'k*');
119     if min_bool
120         legend([h1,h2,h3],{'Average','Best','Worse'},'Location','northeast');
121     else
122         legend([h1,h2,h3],{'Average','Best','Worse'},'Location','southeast');
123     end
124 end
125 function plot_pso_traj(history)
126 best_x = history.data(history.Niters + 1).gbest_x;
127 plot(best_x(1,:),best_x(2:,:), 'rx', 'MarkerSize',20);
128 end

```

Listing 4: Griewank Function

```

1 function y = griewank_fun(X_swarm,min_bool)
2 %     d dimensionalfriewank function
3 switch nargin
4     case 1
5         min_bool=1;
6 end
7
8 [x_dim ,Nswarm] = size(X_swarm);
9 y = zeros(Nswarm,1);
10 for k=1:Nswarm
11     sum = 0;
12     prod = 1;
13     x = X_swarm(:,k);
14     for i=1:x_dim
15         x_i = x(i);
16         sum = sum + x_i^2/4000;
17         prod = prod * cos(x_i/sqrt(i));
18     end
19     y(k) = sum - prod +1;
20 end
21 if ~min_bool
22     y = -1*y;
23 end
24 end

```

Listing 5: Particle Swarm

```

1 function [x_star, history_out] = particleswarm(fun,a,b, Nswarm, Niters,...
2     inert_const, cog_const, social_const, constricted, vmax_prop)
3 %   a,b are the limits of the feasible domains of x i.e. x \in (a,b)
4 history.name = 'Global Optimizer: PSO';
5 %   rng('default');
6 switch nargin
7     case 3
8         Nswarm = 40;
9         Niters = 100;
10        constricted = 1;
11        inert_const = 0.8;
12        cog_const = 2;
13        social_const = 2;
14        vmax_prop = 0.1;
15    case 4
16        Niters = 100;
17        constricted = 1;
18        inert_const = 0.8;
19        cog_const = 2;
20        social_const = 2;
21        vmax_prop = 0.1;
22    case 5
23        inert_const = 0.8;
24        cog_const = 2;
25        social_const = 2;
26        constricted = 1;
27        vmax_prop = 0.1;
28    case 6
29        cog_const = 2;
30        social_const = 2;
31        constricted = 1;
32        vmax_prop = 0.1;
33    case 7
34        social_const = 2;
35        constricted = 1;
36        vmax_prop = 0.1;
37    case 8
38        constricted = 1;
39        vmax_prop = 0.1;
40    case 9
41        vmax_prop = 0.1;
42 end
43 x_dim = length(a);
44 vmax = vmax_prop*(b-a);
45 history.parameter.x_dim= x_dim;
46 history.parameter.Nswarm = Nswarm;
47 history.parameter.Niters = Niters;
48 history.parameter.inert_const = inert_const;
49 history.parameter.cog_const = cog_const;
50 history.parameter.social_const = social_const;
51 history.parameter.constricted = constricted;
52 history.parameter.vmax = vmax;
53 if constricted
54     phi = cog_const + social_const;
55     kappa = 2/abs(2-phi -sqrt(phi^2 -4*phi));
56 end
57
58 count = 0;

```

```

59 % generate the swarm randomly
60 X_swarm = rand(x_dim, Nswarm); % positions \in (0,1)
61 V_swarm = 2*rand(x_dim, Nswarm)-1; % velocities \in (-1,1)
62 V_swarm = min(vmax, max(-vmax, V_swarm)); % \in (-vmax, vmax)
63 % scale to the domain
64 X_swarm = (b-a).*X_swarm + a;
65 % update pbest and gbest
66 pbest_x = X_swarm;
67 pbest_fval = fun(X_swarm);
68 [gbest_fval, idx] = min(pbest_fval);
69 gbest_x = pbest_x(:, idx);
70 % write to history
71 history.data(count+1).pbest_fval = pbest_fval;
72 history.data(count+1).gbest_x = gbest_x;
73 history.data(count+1).gbest_fval = gbest_fval;
74 history.data(count+1).pbest_av = mean(pbest_fval);
75 history.data(count+1).pbest_worse = max(pbest_fval);
76
77 for count=1:Niters
78     % generate r and s
79     r = rand(x_dim, Nswarm);
80     s = rand(x_dim, Nswarm);
81     % update velocity
82     V_swarm = inert_const*V_swarm + cog_const*(r.*(pbest_x-X_swarm)) + ...
83             social_const*(s.*(gbest_x-X_swarm));
84     if constricted
85         V_swarm = kappa*V_swarm;
86     end
87     % clamp velocities
88     V_swarm = min(vmax, max(-vmax, V_swarm)); % \in (-vmax, vmax)
89     %update position
90     X_swarm = X_swarm + V_swarm;
91     %update pbest
92     new_fval = fun(X_swarm);
93     for i=1:Nswarm
94         if new_fval(i) < pbest_fval(i)
95             pbest_fval(i) = new_fval(i);
96             pbest_x(:, i) = X_swarm(:, i);
97         end
98     end
99     %update gbest
100    if sum(pbest_fval < gbest_fval) > 0
101        [gbest_fval, idx] = min(pbest_fval);
102        gbest_x = X_swarm(:, idx);
103    end
104    % write to history
105    history.data(count+1).pbest_fval = pbest_fval;
106    history.data(count+1).gbest_x = gbest_x;
107    history.data(count+1).gbest_fval = gbest_fval;
108    history.data(count+1).pbest_av = mean(pbest_fval);
109    history.data(count+1).pbest_worse = max(pbest_fval);
110 end
111 history.Niters = count;
112 x_star = gbest_x;
113 if nargout>1
114     history_out = history;
115 end
116 end

```

Genetic Algorithm Code

Listing 6: GA Main Code

```
1 % ECE 580 HW4: Problem 4
2 % Rahul Deshmukh
3 % deshmun5@purdue.edu
4 clc; clear all; close all;
5 format long;
6 save_dir = '../.../hw4/pix/';
7 %% TSP setup
8 % map coordinates
9 x_pos = [ 0.4306
10 3.7094
11 6.9330
12 9.3582
13 4.7758
14 1.2910
15 4.83831
16 9.4560
17 3.6774
18 3.2849];
19
20 y_pos = [ 7.7288
21 2.9727
22 1.7785
23 6.9080
24 2.6394
25 4.5774
26 8.43692
27 8.8150
28 7.0002
29 7.5569];
30
31 Num_city = length(x_pos);
32 lb = 1*ones(1,Num_city);
33 ub = Num_city*ones(1,Num_city);
34 resolution = ones(1,Num_city);
35 coded_lens = ceil(log2((ub-lb)./resolution));
36
37
38 %% GA: solver params
39 total_possible_path = factorial(Num_city-1)
40 N_pop = 100;
41 p_xover = 0.8;
42 p_mut = 0.05;
43 Niters = 200;
44 selection_method = 'tournament_method2';
45
46 %% GA starts
47
48 % initialize collectors
49 best_f = [];
50 av_f = [];
51 worse_f = [];
52
53 % choose type of selector
54 if strcmp(selection_method, 'roulette')
```

```

55     selection = @(x,f) roulette(x,f);
56 elseif strcmp(selection_method, 'tournament_method1')
57     selection = @(x,f) tournament_selection(x,f,1);
58 elseif strcmp(selection_method, 'tournament_method2')
59     selection = @(x,f) tournament_selection(x,f,2);
60 end
61
62 % draw initial population: all possible permutations of route
63 X = zeros(N_pop, Num_city);
64 for i=1:N_pop
65     ith_route = randperm(Num_city);
66     X(i,:) = ith_route;
67 end
68 %encode X
69 parents = encode(X, lb, ub, coded_lens);
70 % evaluate fitness of parents
71 f_parent = -1*fitness(parents, lb, coded_lens, resolution, x_pos, y_pos);
72 [best_f, av_f, worse_f] = log_f(f_parent, best_f, av_f, worse_f);
73 for i=1:Niters
74     % generate mating pool using selection
75     mating_pool = selection(parents, f_parent);
76     %perform crossover
77     parents = crossover(mating_pool, p_xover, Num_city, coded_lens);
78     %perform mutation
79
80     %perform elitism
81     parents = elitism(parents, f_parent);
82     %evaluate fitness of offspring
83     f_parent = -1*fitness(parents, lb, coded_lens, resolution, x_pos, y_pos);
84     [best_f, av_f, worse_f] = log_f(f_parent, best_f, av_f, worse_f);
85 end
86 % find the best offspring
87 [f_star, k_star] = max(f_parent);
88 fprintf(strcat('Shortest Route Length: ', num2str(-1*f_star)))
89 x_star_coded = parents(k_star,:);
90 x_star = decode(x_star_coded, lb, coded_lens, resolution)
91
92 %% Convergence Plotting
93 fig1 = figure(1);
94 hold on; grid on;
95 x = 1:Niters+1;
96 h1 = plot(x, -1*av_f, '-b', 'LineWidth', 1);
97 h2 = plot(x, -1*best_f, '-r', 'LineWidth', 1);
98 h3 = plot(x, -1*worse_f, '-k', 'LineWidth', 1);
99 v = 1:10:Niters+1;
100 plot(x(v), -1*av_f(v), 'bx');
101 plot(x(v), -1*best_f(v), 'ro');
102 plot(x(v), -1*worse_f(v), 'k*');
103 legend([h1,h2,h3], {'Average', 'Best', 'Worse'}, 'Location', 'northeast');
104 hold off;
105 box('on');
106 xlabel('Num Iters'); ylabel('Function value');
107 title('Convergence of GA');
108 saveas(fig1, strcat(save_dir, 'ga_conv'), 'eps');
109 %% Route plotting
110 fig2 = figure(2);
111 hold on; grid on;
112 scatter(x_pos, y_pos, 'ob');
113 x_star_end = [x_star(2:end), x_star(1)];

```



```

114 for i=1:Num_city
115     x = x_pos(x_star(i));
116     y = y_pos(x_star(i));
117     u = x_pos(x_star_end(i)) - x;
118     v = y_pos(x_star_end(i)) - y;
119     text(x,y,num2str(x_star(i)),'FontSize',12, 'FontWeight','bold',...
120          'HorizontalAlignment','left', 'VerticalAlignment','middle' );
121     quiver(x,y,u,v,'r','Autoscale','off','LineWidth',1);
122 end
123 box('on');hold off;
124 xlabel('X'); ylabel('Y');
125 saveas(fig2, strcat(save_dir, 'ga_best_route'), 'eps');
126 title('Optimal Route')

```

Listing 7: Fitness function

```

1 function f = fitness(X_coded, lb, code_lens, resolution,...
2                     x_pos, y_pos)
3 X = decode(X_coded, lb, code_lens, resolution);
4 [N_pop,~] = size(X);
5 f = zeros(N_pop,1);
6 for i=1:N_pop
7     ith_route = X(i,:);
8     f(i) = route_len(ith_route,x_pos,y_pos);
9 end
10 end
11
12 function d = route_len(r, x_pos, y_pos)
13 r_end = [r(2:end),r(1)];
14 Δ_x = x_pos(r_end) - x_pos(r);
15 Δ_y = y_pos(r_end) - y_pos(r);
16 d = sum(sqrt(Δ_x.^2 + Δ_y.^2));
17 end

```

Listing 8: Encoding function

```

1 function X_coded = encode(X, lb, ub, code_lens)
2 [N_pop,~] = size(X);
3 L = sum(code_lens);
4 cumsum_code_lens = [0, cumsum(code_lens)];
5 X_coded = zeros(N_pop,L);
6 Num_var = length(lb);
7 for i = 1:N_pop
8     x = X(i,:) - lb;
9     x_coded = zeros(1,L);
10    for j = 1:Num_var
11        xj = x(j);
12        x_coded(cumsum_code_lens(j) + 1 : cumsum_code_lens(j+1)) = de2bi(xj ...
13            ,code_lens(j));
14    end
15    X_coded(i,:) = x_coded;
16 end

```

Listing 9: Decoding function

```

1 function X = decode(X_coded, lb, code_lens, resolution)
2 [N_pop, ~] = size(X_coded);
3 L = sum(code_lens);
4 cumsum_code_lens = [0, cumsum(code_lens)];
5 Num_var = length(lb);
6 X = zeros(N_pop, Num_var);
7 for i=1:N_pop
8     x_coded = X_coded(i, :);
9     x = zeros(1, Num_var);
10    for j=1:Num_var
11        xj_coded = x_coded(cumsum_code_lens(j) + 1 : cumsum_code_lens(j+1));
12        x(j) = resolution(j)*bi2de(xj_coded);
13    end
14    X(i, :) = x + lb;
15 end
16 end

```

Listing 10: Roulette-wheel selection function

```

1 function mating_pool = roulette(parent, f_parent)
2 [N_pop, ~] = size(parent);
3 f_min = min(f_parent);
4 f = f_parent - f_min;
5 F = sum(f);
6 p = f/F;
7 q = cumsum(p);
8 rand_nums = rand(N_pop, 1);
9 mating_idx = zeros(N_pop, 1);
10 temp = q' - rand_nums;
11 for k=1:N_pop
12     mating_idx(k) = find(temp(k, :) > 0, 1);
13 end
14 mating_pool = parent(mating_idx, :);
15 end

```

Listing 11: Tournament selection function

```

1 function mating_pool = tournament_selection(parent, f_parent, method)
2 [N_pop, ~] = size(parent);
3 mating_idx = zeros(N_pop, 1);
4 if method == 1
5     a = randi([1, N_pop], 1, N_pop);
6     b = randi([1, N_pop], 1, N_pop);
7     fa = f_parent(a);
8     fb = f_parent(b);
9     for k=1:N_pop
10        if fa(k) > fb(k)
11            mating_idx(k) = a(k);
12        else
13            mating_idx(k) = b(k);
14        end
15    end
16 elseif method == 2
17     a = randi([1, N_pop], 1, N_pop);
18     fa = f_parent(a);
19     for k=1:N_pop

```

```

20         if fa(k)>f_parent(k)
21             mating_idx(k) = a(k);
22         else
23             mating_idx(k) = k;
24         end
25     end
26 end
27 mating_pool = parent(mating_idx, :);
28 end

```

Listing 12: Cross-over function

```

1 function offspring = crossover(mating_pool, p_xover, Num_var, coded_lens)
2 [N_pop, ~] = size(mating_pool);
3 cumsum_coded_lens = [0, cumsum(coded_lens)];
4
5 rand_nums = rand(1, N_pop);
6 do_xover = rand_nums > (1-p_xover);
7 offspring = mating_pool;
8 for k = 1:N_pop
9     if do_xover(k)
10         k_offspring = offspring(k, :);
11         js = randi([1, Num_var], 1, 2);
12         j1_idx = cumsum_coded_lens(js(1)) + 1 : cumsum_coded_lens(js(1)+1);
13         j2_idx = cumsum_coded_lens(js(2)) + 1 : cumsum_coded_lens(js(2)+1);
14         j1_code = k_offspring(j1_idx);
15         k_offspring(j1_idx) = k_offspring(j2_idx);
16         k_offspring(j2_idx) = j1_code;
17         offspring(k, :) = k_offspring;
18     end
19 end
20 end

```

Listing 13: Elitism function

```

1 function new_pop = elitism(pop, fitness)
2 new_pop = pop;
3 temp_fit = fitness;
4 [~, max_fit_idx] = max(temp_fit);
5 temp_fit(max_fit_idx) = min(temp_fit);
6 [~, other_max_fit_idx] = max(temp_fit);
7 new_pop([1, 2], :) = pop([max_fit_idx, other_max_fit_idx], :);
8 end

```

Listing 14: Logging function

```

1 function [best_f, av_f, worse_f] = log_f(f_parent, best_f, av_f, worse_f)
2
3 [best, best_id] = max(f_parent);
4 av = mean(f_parent);
5 worse = min(f_parent);
6
7 best_f = [best_f, best];
8 av_f = [av_f, av];
9 worse_f = [worse_f, worse];

```

```
10 end
```

Listing 15: Output

```
1 x_star_min =
2
3     0.0007
4     0.0023
5
6
7 fval =
8
9     1.6217e-06
10
11
12 x_star_max =
13
14     0.0000
15     4.4474
16
17
18 fval =
19
20     2.0049
21
22 -----Linear Programming-----
23 Optimal solution found.
24
25
26 x_star_linprog =
27
28     15.0000
29         0
30     3.3333
31         0
32
33
34 ans =
35
36     113.3333
```