ECE 580: Homework 3

Rahul Deshmukh

April 8, 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 atListing 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:

$$
\begin{aligned}
a_1 &= \begin{bmatrix} 0.8147 & 0.9058 \end{bmatrix}^T \\
a_2 &= \begin{bmatrix} 0.1270 & 0.9134 \end{bmatrix}^T \\
A_1 = a_1 \otimes a_1 &= \begin{bmatrix} 0.6638 & 0.7380 \\ 0.7380 & 0.8205 \end{bmatrix} \\
A_2 = a_2 \otimes a_2 &= \begin{bmatrix} 0.0161 & 0.1160 \\ 0.1160 & 0.8343 \end{bmatrix} \\
A_2^\dagger A_1^\dagger &= \begin{bmatrix} 0.0604 & 0.0672 \\ 0.4348 & 0.4834 \end{bmatrix} \qquad (1) \\
(A_1 A_2)^\dagger &= \begin{bmatrix} 0.0881 & 0.0979 \\ 0.6334 & 0.7042 \end{bmatrix} \qquad (2)
\end{aligned}
$$

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 $[-5, 5]$x$[5, 5]$ is at Figure 1.

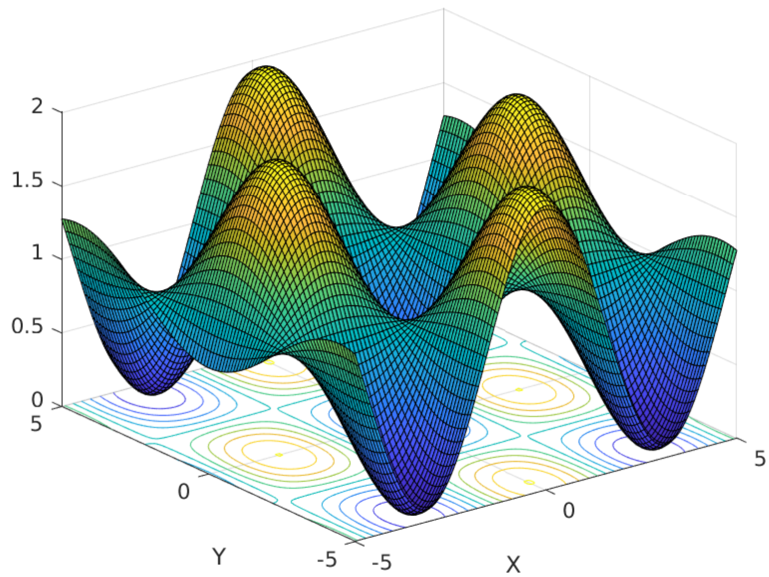

Figure 1: Surface plot of Griewank function

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 $\boldsymbol{x} = \begin{bmatrix} 0.0007 & 0.0023 \end{bmatrix}^T$ with a function value 1.6217e-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.

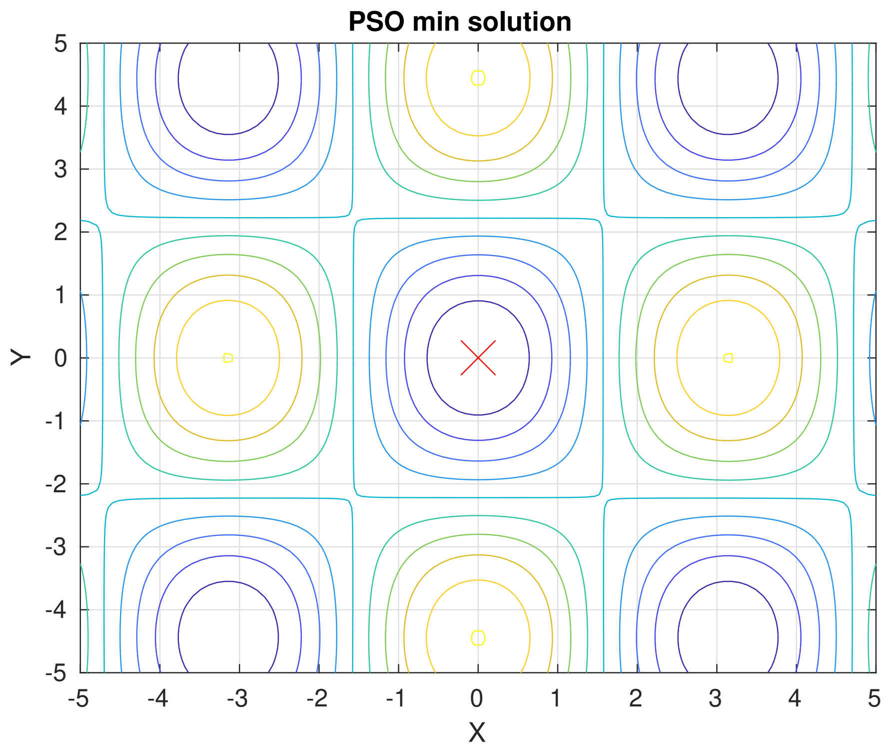

Figure 2: Plot of optimal solution(red X) on contours of objective function

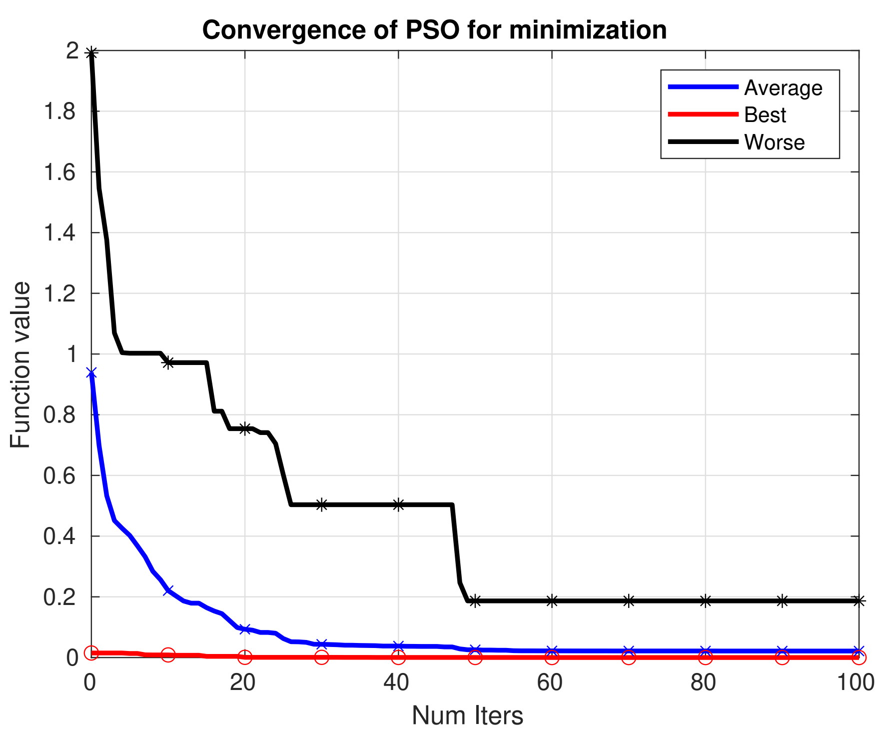

Figure 3: Plot of Average, Best and Worse function values for PSO

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 $\boldsymbol{x} = \begin{bmatrix}0.0000 & 4.4474\end{bmatrix}^T$ 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.

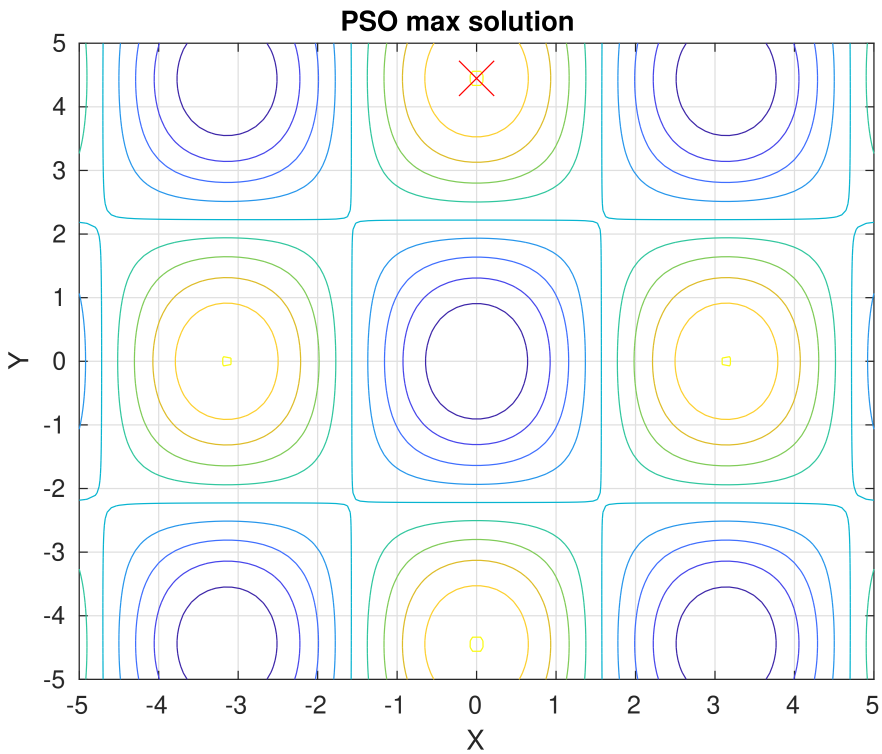

Figure 4: Plot of optimal solution (red X) on contours of objective function

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.

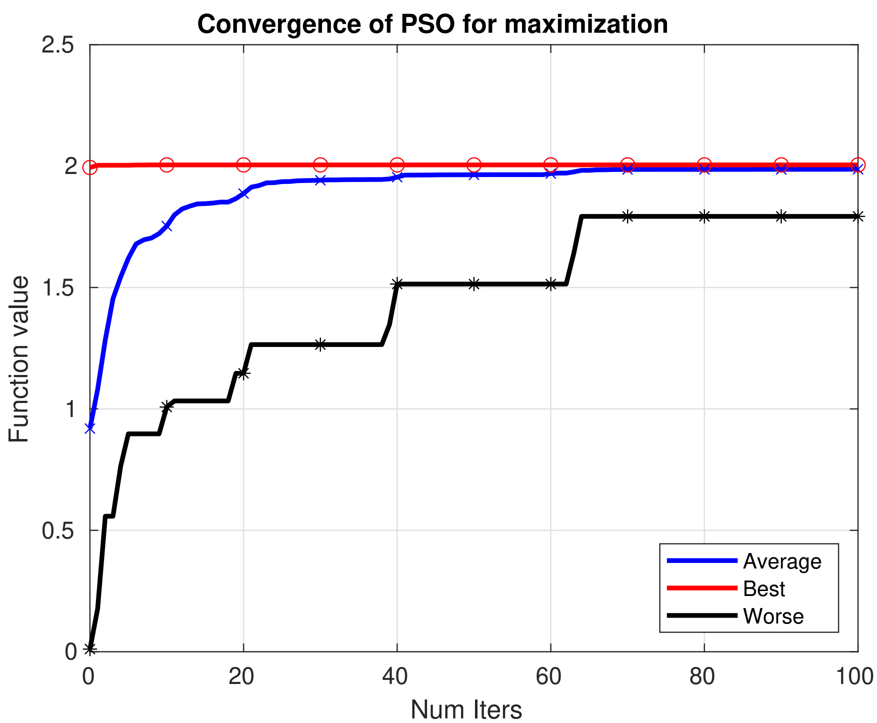

Figure 5: Plot of Average, Best and Worse function values for PSO

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 evaulation. We then carry out selection, cross-over, elitism and fitness evaluation repeadtedly till the number of iterations are satisfied.

For crossover, we dont want to carry-out an operation which might result in a 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 $\begin{bmatrix} 9 & 10 & 1 & 6 & 2 & 5 & 3 & 4 & 8 & 7 \end{bmatrix}$.

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 refered 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 refered at Listing 12 at page 19 & Listing 13 at page 19 respectively.

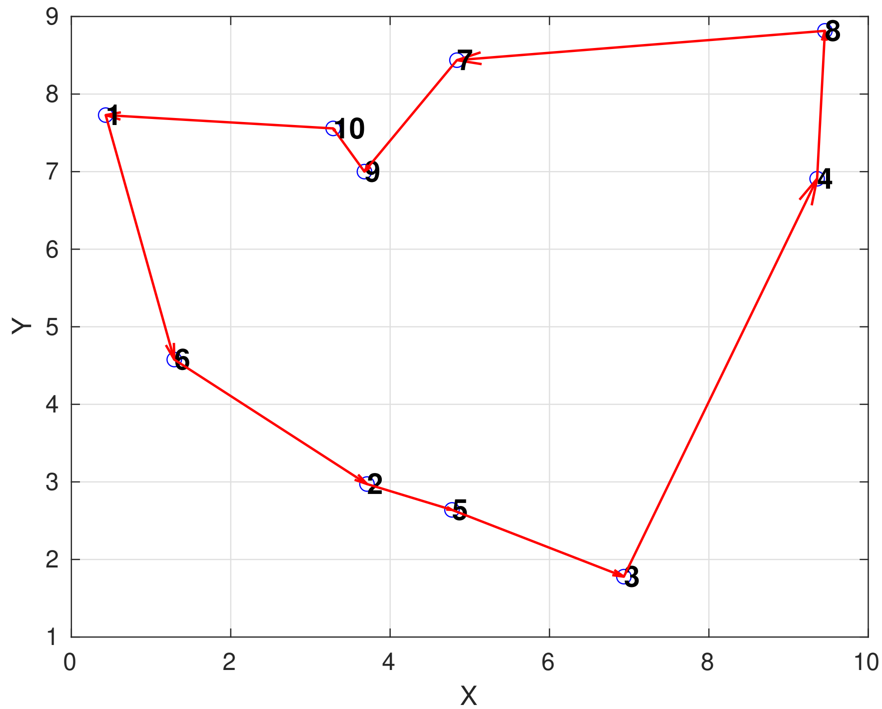

Figure 6: Shortest route calculated using GA

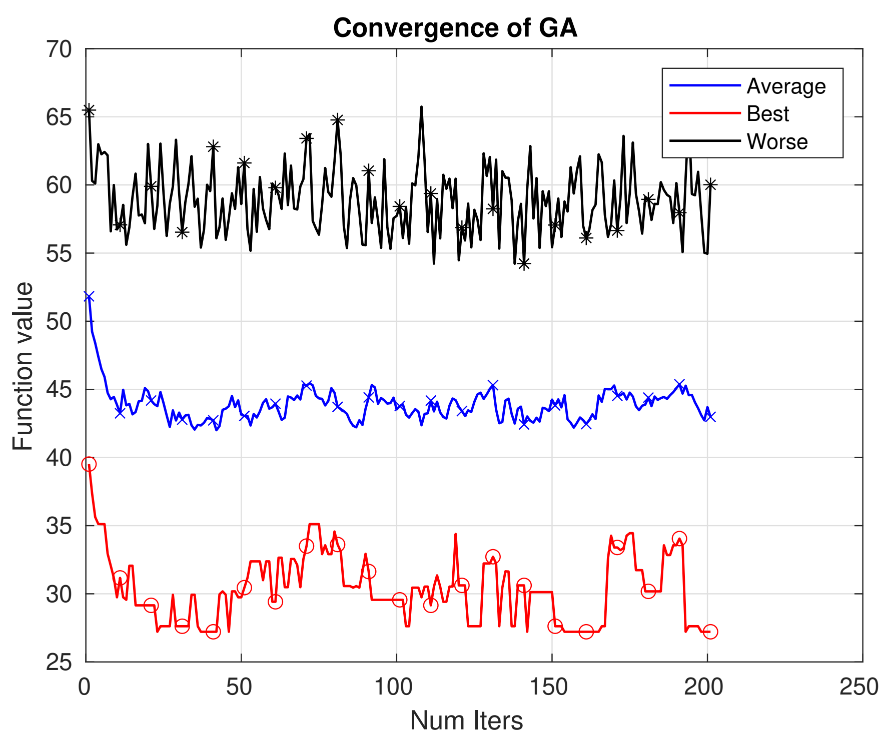

Figure 7: Plot of Average, Best and Worse function values for GA

Exercise 5

For this problem we are required to solve the following problem

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmax}}\, \boldsymbol{c}^T\boldsymbol{x}$$
$$\text{subject to } A\boldsymbol{x} \le \boldsymbol{b}$$
$$\boldsymbol{x} \ge \boldsymbol{0}$$
$$\text{where}$$
$$\boldsymbol{c}^T = \begin{bmatrix} 6 & 4 & 7 & 5 \end{bmatrix}$$
$$A = \begin{bmatrix} 1 & 2 & 1 & 2 \\ 6 & 5 & 3 & 2 \\ 3 & 4 & 9 & 12 \end{bmatrix}$$
$$\boldsymbol{b} = \begin{bmatrix} 20 \\ 100 \\ 75 \end{bmatrix}$$

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

We obtain an optimal solution as $\boldsymbol{x}^* = \begin{bmatrix} 15.0 & 0.0 & 3.3333 & 0.0 \end{bmatrix}^T$ 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

```
clc; clear all;
format short;
rng('default')

a_1 = rand(2,1)
a_2 = rand(2,1)
A_1 = a_1*a_1'
A_2 = a_2*a_2'

A_1_pinv = pinv(A_1);
A_2_pinv =pinv(A_2);

A_2_pinv_A_1_pinv = A_2_pinv*A_1_pinv
A_1A_2_pinv = pinv(A_1*A_2)

diff = A_1A_2_pinv - A_2_pinv_A_1_pinv

if abs(sum(diff))> 1e-16
    fprintf('pinv(A_2)pinv(A_1) not equals pinv(A_1A_2)\n')
end
```

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

```
a_1 =

    0.8147
    0.9058


a_2 =

    0.1270
    0.9134


A_1 =

    0.6638    0.7380
    0.7380    0.8205


A_2 =

    0.0161    0.1160
    0.1160    0.8343


A_2_pinv_A_1_pinv =

    0.0604    0.0672
    0.4348    0.4834

```

```

A_1A_2_pinv =

    0.0881    0.0979
    0.6334    0.7042


diff =

    0.0276    0.0307
    0.1986    0.2208

pinv(A_2)pinv(A_1) not equals pinv(A_1A_2)
```

Listing 3: Main Code

```
% ECE 580 HW4
% Rahul Deshmukh
% deshmuk5@purdue.edu
clc; clear all; close all;
format short;
%% include paths
addpath('../OptimModule/optimizers/global/');
save_dir = './pix/';

%% Problem 2: PSO min
%plot griewank fun
x = linspace(-5,5,100);
[X,Y] = meshgrid(x,x);
[h,w] = size(X);
Z = zeros(h,w);
for ih=1:h
    for iw=1:w
        Z(ih,iw) = griewank_fun([X(ih,iw);Y(ih,iw)]);
    end
end
fig = figure(1);
surfc(X,Y,Z);grid on;
view(3);
xlabel('X');
ylabel('Y');
saveas(fig,strcat(save_dir,'surf_plot'),'epsc');

a = [-5;-5];
b = [5;5];
[x_star_min, history_min] = particleswarm(@(x)griewank_fun(x), a, b);
x_star_min
fval = history_min.data(history_min.Niters+1).gbest_fval
fig2= figure(2);
hold on;grid on;
pso_conv_plot(history_min,1);
hold off;
box('on');
xlabel('Num Iters'); ylabel('Function value');
title('Convergence of PSO for minimization');
saveas(fig2,strcat(save_dir,'plot_pso_min'),'epsc');

fig3= figure(3);
```

```
hold on;grid on;
contour(X,Y,Z);
plot_pso_traj(history_min);
xlabel('X'); ylabel('Y');
title('PSO min solution')
hold off;
xlim([a(1),b(1)]);
ylim([a(2),b(2)]);
xticks(a(1):1:b(1));
yticks(a(2):1:b(2));
box('on');
saveas(fig3,strcat(save_dir,'pso_min_traj'),'epsc')

%% Problem 3: PSO max
[x_star_max, history_max] = particleswarm(@(x)griewank_fun(x,0), a, b);
x_star_max
fval = -1*history_max.data(history_max.Niters+1).gbest_fval
fig4= figure(4);
hold on;grid on;
pso_conv_plot(history_max,0);
hold off;
box('on');
xlabel('Num Iters'); ylabel('Function value');
title('Convergence of PSO for maximization');
saveas(fig4,strcat(save_dir,'plot_pso_max'),'epsc');

fig5= figure(5);
hold on;grid on;
contour(X,Y,Z);
plot_pso_traj(history_max);
xlabel('X'); ylabel('Y');
title('PSO max solution')
hold off;
xlim([a(1),b(1)]);
ylim([a(2),b(2)]);
xticks(a(1):1:b(1));
yticks(a(2):1:b(2));
box('on');
saveas(fig5,strcat(save_dir,'pso_max_traj'),'epsc')

%% Problem 5: Linprog
fprintf('-----------Linear Programming----------------');
A = [1, 2, 1, 2;
    6, 5, 3, 2;
    3, 4, 9, 12];
b = [20; 100; 75];
Aeq= []; beq = [];
lb = [0, 0, 0, 0];
ub = Inf*[1, 1, 1, 1];
c = [6, 4, 7, 5];
[x_star_linprog,fval] = linprog(-1*c, A, b, Aeq, beq, lb, ub);
x_star_linprog
-1*fval

%% Local helper functions for plotting
% plotting for PSO
function pso_conv_plot(history, min_bool)
    av = [];
    gbest = [];
```

```
    worse =[];
    for i=1:history.Niters + 1
       av = [av; history.data(i).pbest_av];
       gbest = [gbest; history.data(i).gbest_fval];
       worse = [worse; history.data(i).pbest_worse];
    end
    if ~min_bool
       av=-1*av; gbest = -1*gbest; worse = -1*worse;
    end
    x = 0:1:history.Niters;
    h1 =plot(x,av,'-b','LineWidth',2);
    h2 = plot(x,gbest,'-r','LineWidth',2);
    h3 = plot(x,worse,'-k', 'LineWidth',2);
    v = 1:10:history.Niters+1;
    plot(x(v),av(v),'bx');
    plot(x(v),gbest(v),'ro');
    plot(x(v),worse(v),'k*');
    if min_bool
        legend([h1,h2,h3],{'Average','Best','Worse'},'Location','northeast');
    else
        legend([h1,h2,h3],{'Average','Best','Worse'},'Location','southeast');
    end
end
function plot_pso_traj(history)
best_x = history.data(history.Niters + 1).gbest_x;
plot(best_x(1,:),best_x(2,:),'rx','MarkerSize',20);
end
```

Listing 4: Griewank Function

```
function y = griewank_fun(X_swarm,min_bool)
%     d dimensionalfriewank function
switch nargin
    case 1
        min_bool=1;
end

[x_dim ,Nswarm] = size(X_swarm);
y = zeros(Nswarm,1);
for k=1:Nswarm
    sum = 0;
    prod = 1;
    x = X_swarm(:,k);
    for i=1:x_dim
       x_i = x(i);
       sum = sum + x_i^2/4000;
       prod = prod * cos(x_i/sqrt(i));
    end
    y(k) = sum - prod +1;
end
if ~min_bool
    y = -1*y;
end
end
```

Listing 5: Particle Swarm

```
function [x_star, history_out] = particleswarm(fun,a,b, Nswarm, Niters,...
    inert_const, cog_const,social_const, constricted, vmax_prop)
%   a,b are the limits of the feasible domains of x i.e. x \in (a,b)
    history.name = 'Global Optimizer: PSO';
%     rng('default');
    switch nargin
        case 3
            Nswarm = 40;
            Niters = 100;
            constricted = 1;
            inert_const = 0.8;
            cog_const = 2;
            social_const = 2;
            vmax_prop = 0.1;
        case 4
            Niters = 100;
            constricted = 1;
            inert_const = 0.8;
            cog_const = 2;
            social_const = 2;
            vmax_prop = 0.1;
        case 5
            inert_const = 0.8;
            cog_const = 2;
            social_const = 2;
            constricted = 1;
            vmax_prop = 0.1;
        case 6
            cog_const = 2;
            social_const = 2;
            constricted = 1;
            vmax_prop = 0.1;
        case 7
            social_const = 2;
            constricted = 1;
            vmax_prop = 0.1;
        case 8
            constricted = 1;
            vmax_prop = 0.1;
        case 9
            vmax_prop = 0.1;
    end
    x_dim = length(a);
    vmax = vmax_prop*(b-a);
    history.parameter.x_dim= x_dim;
    history.parameter.Nswarm = Nswarm;
    history.parameter.Niters = Niters;
    history.parameter.inert_const = inert_const;
    history.parameter.cog_const = cog_const;
    history.parameter.social_const = social_const;
    history.parameter.constricted = constricted;
    history.parameter.vmax = vmax;
    if constricted
       phi = cog_const + social_const;
       kappa = 2/abs(2-phi -sqrt(phi^2 -4*phi));
    end

    count = 0;
```

```
    % generate the swarm randomly
    X_swarm = rand(x_dim, Nswarm); % positions \in (0,1)
    V_swarm = 2*rand(x_dim, Nswarm)-1; % velocities \in (-1,1)
    V_swarm =  min(vmax,max(-vmax,V_swarm)); % \in (-vmax,vmax)
    % scale to the domain
    X_swarm = (b-a).*X_swarm + a;
    % update pbest and gbest
    pbest_x = X_swarm;
    pbest_fval = fun(X_swarm);
    [gbest_fval, idx] = min(pbest_fval);
    gbest_x = pbest_x(:,idx);
    % write to history
    history.data(count+1).pbest_fval = pbest_fval;
    history.data(count+1).gbest_x = gbest_x;
    history.data(count+1).gbest_fval = gbest_fval;
    history.data(count+1).pbest_av = mean(pbest_fval);
    history.data(count+1).pbest_worse = max(pbest_fval);

    for count=1:Niters
        % generate r and s
        r = rand(x_dim,Nswarm);
        s = rand(x_dim,Nswarm);
        % update velocity
        V_swarm = inert_const*V_swarm + cog_const*(r.*(pbest_x-X_swarm)) + ...
                  social_const*(s.*(gbest_x-X_swarm));
        if constricted
           V_swarm = kappa*V_swarm;
        end
        % clamp velocities
        V_swarm =  min(vmax,max(-vmax,V_swarm)); % \in (-vmax,vmax)
        %update position
        X_swarm = X_swarm + V_swarm;
        %update pbest
        new_fval = fun(X_swarm);
        for i=1:Nswarm
           if new_fval(i)< pbest_fval(i)
              pbest_fval(i) = new_fval(i);
              pbest_x(:,i) = X_swarm(:,i);
           end
        end
        %update gbest
        if sum(pbest_fval < gbest_fval) > 0
           [gbest_fval,idx] = min(pbest_fval);
           gbest_x = X_swarm(:,idx);
        end
        % write to history
        history.data(count+1).pbest_fval = pbest_fval;
        history.data(count+1).gbest_x = gbest_x;
        history.data(count+1).gbest_fval = gbest_fval;
        history.data(count+1).pbest_av = mean(pbest_fval);
        history.data(count+1).pbest_worse = max(pbest_fval);
    end
    history.Niters = count;
    x_star = gbest_x;
    if nargout>1
        history_out = history;
    end
end
```

Genetic Algorithm Code

Listing 6: GA Main Code

```
% ECE 580 HW4: Problem 4
% Rahul Deshmukh
% deshmuk5@purdue.edu
clc; clear all; close all;
format long;
save_dir = '../../../../hw4/pix/';
%% TSP setup
% map coordinates
x_pos = [ 0.4306
 3.7094
 6.9330
 9.3582
 4.7758
 1.2910
 4.83831
 9.4560
 3.6774
 3.2849];

y_pos = [ 7.7288
 2.9727
 1.7785
 6.9080
 2.6394
 4.5774
 8.43692
 8.8150
 7.0002
 7.5569];

Num_city = length(x_pos);
lb = 1*ones(1,Num_city);
ub = Num_city*ones(1,Num_city);
resolution = ones(1,Num_city);
coded_lens = ceil(log2((ub-lb)./resolution));


%% GA: solver params
total_possible_path = factorial(Num_city-1)
N_pop = 100;
p_xover = 0.8;
p_mut = 0.05;
Niters = 200;
selection_method = 'tournament_method2';

%% GA starts

% intialize collectors
best_f = [];
av_f = [];
worse_f = [];

% choose type of selector
if strcmp(selection_method, 'roulette')
```

```
    selection = @(x,f) roulette(x,f);
elseif strcmp(selection_method, 'tournament_method1')
    selection = @(x,f) tournament_selection(x,f,1);
elseif strcmp(selection_method, 'tournament_method2')
    selection = @(x,f) tournament_selection(x,f,2);
end

% draw initial population: all possible permuations of route
X = zeros(N_pop, Num_city);
for i=1:N_pop
   ith_route = randperm(Num_city);
   X(i,:) = ith_route;
end
%encode X
parents = encode(X, lb, ub, coded_lens);
% evaluate fitness of parents
f_parent = -1*fitness(parents, lb, coded_lens, resolution, x_pos, y_pos);
[best_f, av_f, worse_f] = log_f(f_parent, best_f, av_f, worse_f);
for i=1:Niters
   % generate mating pool using selection
   mating_pool = selection(parents, f_parent);
   %perform crossover
   parents = crossover(mating_pool, p_xover, Num_city, coded_lens);
   %perform mutation

   %perform elitism
   parents = elitism(parents, f_parent);
   %evaluate fitness of offspring
   f_parent = -1*fitness(parents, lb, coded_lens, resolution, x_pos, y_pos);
   [best_f, av_f, worse_f] = log_f(f_parent, best_f, av_f, worse_f);
end
% find the best offspring
[f_star, k_star] = max(f_parent);
fprintf(strcat('Shortest Route Lenght: ',num2str(-1*f_star)))
x_star_coded = parents(k_star,:);
x_star = decode(x_star_coded, lb, coded_lens, resolution)

%% Convergence Plotting
fig1 = figure(1);
hold on; grid on;
x = 1:Niters+1;
h1 =plot(x,-1*av_f,'-b','LineWidth',1);
h2 = plot(x,-1*best_f,'-r','LineWidth',1);
h3 = plot(x,-1*worse_f,'-k', 'LineWidth',1);
v = 1:10:Niters+1;
plot(x(v),-1*av_f(v),'bx');
plot(x(v),-1*best_f(v),'ro');
plot(x(v),-1*worse_f(v),'k*');
legend([h1,h2,h3],{'Average','Best','Worse'},'Location','northeast');
hold off;
box('on');
xlabel('Num Iters'); ylabel('Function value');
title('Convergence of GA');
saveas(fig1,strcat(save_dir,'ga_conv'),'epsc');
%% Route plotting
fig2 = figure(2);
hold on;grid on;
scatter(x_pos,y_pos,'ob');
x_star_end = [x_star(2:end), x_star(1)];
```

```
for i=1:Num_city
    x = x_pos(x_star(i));
    y = y_pos(x_star(i));
    u = x_pos(x_star_end(i)) - x;
    v = y_pos(x_star_end(i)) - y;
    text(x,y,num2str(x_star(i)),'FontSize',12, 'FontWeight','bold',...
        'HorizontalAlignment','left', 'VerticalAlignment','middle' );
    quiver(x,y,u,v,'r','Autoscale','off','LineWidth',1);
end
box('on');hold off;
xlabel('X'); ylabel('Y');
saveas(fig2,strcat(save_dir,'ga_best_route'),'epsc');
title('Optimal Route')
```

Listing 7: Fitness function

```
function f = fitness(X_coded, lb, code_lens, resolution,...
                    x_pos, y_pos)
X = decode(X_coded, lb, code_lens, resolution);
[N_pop,¬] = size(X);
f = zeros(N_pop,1);
for i=1:N_pop
   ith_route = X(i,:);
   f(i) = route_len(ith_route,x_pos,y_pos);
end
end

function d = route_len(r, x_pos, y_pos)
r_end = [r(2:end),r(1)];
Δ_x = x_pos(r_end) - x_pos(r);
Δ_y = y_pos(r_end) - y_pos(r);
d = sum(sqrt(Δ_x.^2 + Δ_y.^2));
end
```

Listing 8: Encoding function

```
function X_coded = encode(X, lb, ub, code_lens)
[N_pop,¬] = size(X);
L = sum(code_lens);
cumsum_code_lens = [0, cumsum(code_lens)];
X_coded = zeros(N_pop,L);
Num_var = length(lb);
for i = 1:N_pop
    x = X(i,:) - lb;
    x_coded = zeros(1,L);
    for j = 1:Num_var
       xj = x(j);
       x_coded(cumsum_code_lens(j) + 1 : cumsum_code_lens(j+1)) = de2bi(xj ...
           ,code_lens(j));
    end
    X_coded(i,:) = x_coded;
end
```

Listing 9: Decoding function

```
function X = decode(X_coded, lb, code_lens, resolution)
[N_pop,¬] = size(X_coded);
L = sum(code_lens);
cumsum_code_lens = [0, cumsum(code_lens)];
Num_var = length(lb);
X = zeros(N_pop,Num_var);
for i=1:N_pop
   x_coded = X_coded(i,:);
   x = zeros(1,Num_var);
   for j=1:Num_var
       xj_coded = x_coded(cumsum_code_lens(j) + 1 : cumsum_code_lens(j+1));
       x(j) = resolution(j)*bi2de(xj_coded);
   end
   X(i,:) = x + lb;
end
end
```

Listing 10: Roulette-wheel selection function

```
function mating_pool = roulette(parent, f_parent)
[N_pop,¬] = size(parent);
f_min = min(f_parent);
f = f_parent - f_min;
F = sum(f);
p = f/F;
q = cumsum(p);
rand_nums = rand(N_pop,1);
mating_idx = zeros(N_pop,1);
temp = q' -rand_nums;
for k=1:N_pop
   mating_idx(k) = find(temp(k,:) > 0, 1);
end
mating_pool = parent(mating_idx, :);
end
```

Listing 11: Tournament selection function

```
function mating_pool = tournament_selection(parent, f_parent, method)
[N_pop,¬] = size(parent);
mating_idx = zeros(N_pop,1);
if method == 1
   a = randi([1, N_pop], 1, N_pop) ;
   b = randi([1, N_pop], 1, N_pop) ;
   fa = f_parent(a);
   fb = f_parent(b);
   for k=1:N_pop
       if fa(k)>fb(k)
           mating_idx(k) = a(k);
       else
           mating_idx(k) = b(k);
       end
   end
elseif method == 2
    a = randi([1, N_pop], 1, N_pop);
    fa = f_parent(a);
    for k=1:N_pop
```

```
        if fa(k)>f_parent(k)
            mating_idx(k) = a(k);
        else
            mating_idx(k) = k;
        end
    end
end
mating_pool = parent(mating_idx, :);
end
```

Listing 12: Cross-over function

```
function offspring = crossover(mating_pool, p_xover, Num_var, coded_lens)
[N_pop,¬] = size(mating_pool);
cumsum_coded_lens = [0, cumsum(coded_lens)];

rand_nums = rand(1,N_pop);
do_xover = rand_nums > (1-p_xover);
offspring = mating_pool;
for k = 1:N_pop
    if do_xover(k)
        k_offspring = offspring(k,:);
        js = randi([1,Num_var],1,2);
        j1_idx = cumsum_coded_lens(js(1)) + 1 : cumsum_coded_lens(js(1)+1);
        j2_idx = cumsum_coded_lens(js(2)) + 1 : cumsum_coded_lens(js(2)+1);
        j1_code = k_offspring(j1_idx);
        k_offspring(j1_idx) = k_offspring(j2_idx);
        k_offspring(j2_idx) = j1_code;
        offspring(k,:) = k_offspring;
    end
end
end
```

Listing 13: Elitism function

```
function new_pop = elitism(pop, fitness)
new_pop = pop;
temp_fit = fitness;
[¬, max_fit_idx] = max(temp_fit);
temp_fit(max_fit_idx) = min(temp_fit);
[¬, other_max_fit_idx] = max(temp_fit);
new_pop([1,2],:) = pop([max_fit_idx, other_max_fit_idx],:);
end
```

Listing 14: Logging function

```
function [best_f, av_f, worse_f] = log_f(f_parent, best_f, av_f, worse_f)

[best, best_id] = max(f_parent);
av = mean(f_parent);
worse = min(f_parent);

best_f = [best_f, best];
av_f = [av_f, av];
worse_f = [worse_f, worse];
```

```
end
```

Listing 15: Output

```
x_star_min =

    0.0007
    0.0023


fval =

   1.6217e-06


x_star_max =

    0.0000
    4.4474


fval =

    2.0049

-----------Linear Programming----------------
Optimal solution found.


x_star_linprog =

   15.0000
         0
    3.3333
         0


ans =

  113.3333
```