ECE 580: Homework 5

Rahul Deshmukh

April 27, 2020

Exercise 1

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

- Number of bits used to represent each variable: 10 (resolution=0.0098)
- Population size: 40
- Number of iterations: 30
- Probability for cross-over: 0.9
- Probability of Mutation: 0.01
- Selection Method: tournament selection method-2

After carrying out several trials, I obtain an optimal function value of $1.7928e-05$ for the optimal solution as $x^* = \begin{bmatrix}-0.0049 & -0.0049\end{bmatrix}^T$. The plot for best, average, and the worst objective function values in the population for every generation is at Figure 1.

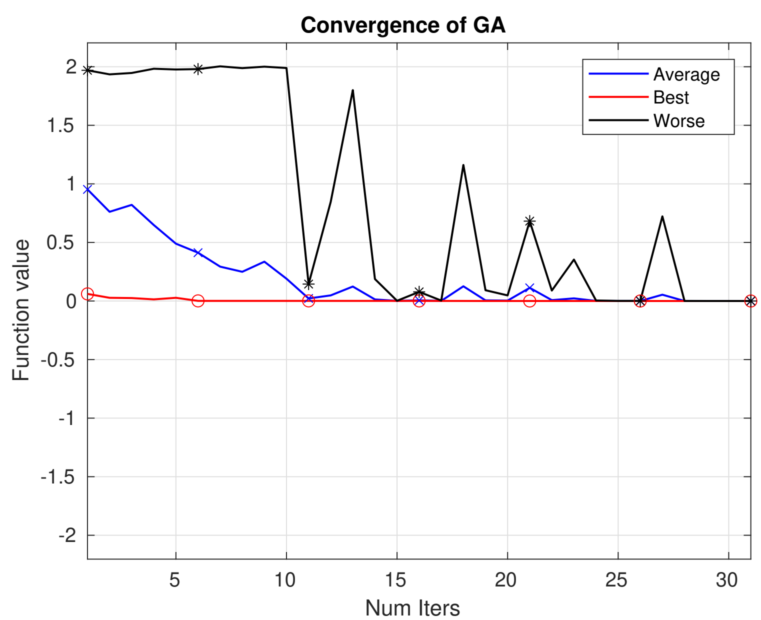

Figure 1: Plot of Average, Best and Worse function values for GA

For main file for GA refer to Listing 1 at page 9. The fitness function can be refered at Listing 2 at page 10 & Listing 3 at page 10. The encoding and decoding functions can be found at Listing 4 at page 11 & Listing 5 at page 11 respectively. The function for Tournament selection is at Listing 7 at page 12. The function for crossover, mutation and elitism can be refered at Listing 8 at page 12, Listing 9 at page 13 & Listing 10 at page 13 respectively.

Exercise 2

In my Real-Number GA, I am using the following parameter settings:

- Population size: 40
- Number of iterations: 30
- Probability for cross-over: 0.9
- Crossover-Method: Convex Combination
- Probability of Mutation: 0.01
- Selection Method: tournament selection method-2

After carrying out several trials, I obtain an optimal function value of $1.7333e-11$ for the optimal solution as $x^* = 1.0e-05 * \begin{bmatrix}0.0195 & 0.8318\end{bmatrix}^T$. The plot for best, average, and the worst objective function values in the population for every generation is at Figure 2.

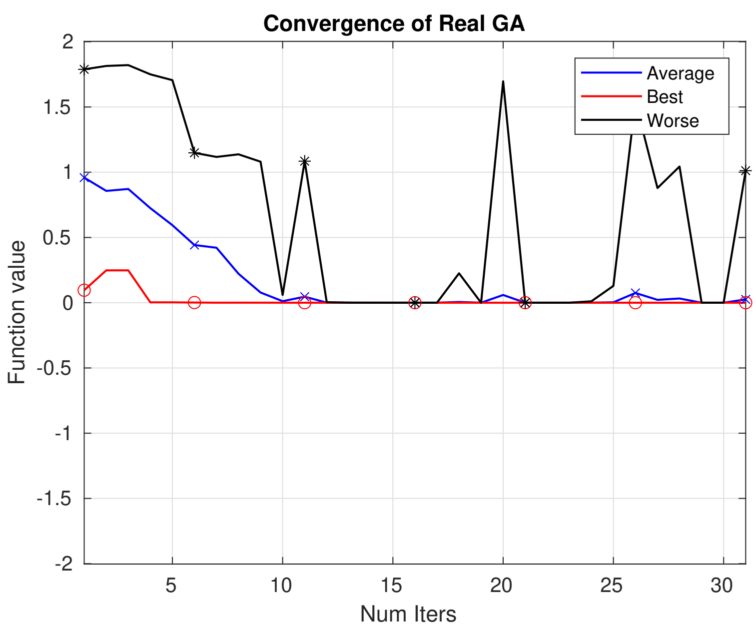

Figure 2: Plot of Average, Best and Worse function values for GA

For main file for GA refer to Listing 12 at page 14. The fitness function can be refered at Listing 2 at page 10 & Listing 3 at page 10. The function for Tournament selection is at Listing 7 at page 12. The function for crossover, mutation and elitism can be refered at Listing 13 at page 15, Listing 14 at page 16 & Listing 10 at page 13 respectively.

Exercise 3

The given LP problem is:

$$\begin{aligned} \text{maximize} \quad & -4x_1 - 3x_2 \\ \text{subject to} \quad & 5x_1 + x_2 \geq 11 \\ & 2x_1 + x_2 \geq 8 \\ & x_1 + 2x_2 \geq 7 \\ & x_1, x_2 \geq 0 \end{aligned}$$

We first convert the above problem to standard form:

$$\begin{aligned} \text{minimize} \quad & 4x_1 + 3x_2 \\ \text{subject to} \quad & 5x_1 + x_2 - x_3 = 11 \\ & 2x_1 + x_2 - x_4 = 8 \\ & x_1 + 2x_2 - x_5 = 7 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \end{aligned}$$

We then solve the above problem using Two-phase simplex method. The computations (with pivot elements in boxes) are as follows:

Phase 1:

$$\begin{aligned}
\begin{bmatrix} \boldsymbol{A} & \boldsymbol{I} & \boldsymbol{b} \\ \boldsymbol{0}^T & \boldsymbol{1}^T & 0 \end{bmatrix} &= \begin{bmatrix} 5 & 1 & -1 & 0 & 0 & 1 & 0 & 0 & 11 \\ 2 & 1 & 0 & -1 & 0 & 0 & 1 & 0 & 8 \\ 1 & 2 & 0 & 0 & -1 & 0 & 0 & 1 & 7 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \\
&= \begin{bmatrix} \boxed{5} & 1 & -1 & 0 & 0 & 1 & 0 & 0 & 11 \\ 2 & 1 & 0 & -1 & 0 & 0 & 1 & 0 & 8 \\ 1 & 2 & 0 & 0 & -1 & 0 & 0 & 1 & 7 \\ -8 & -4 & 1 & 1 & 1 & 0 & 0 & 0 & -26 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 1/5 & -1/5 & 0 & 0 & 1/5 & 0 & 0 & 11/5 \\ 0 & 3/5 & 2/5 & -1 & 0 & -2/5 & 1 & 0 & 18/5 \\ 0 & \boxed{9/5} & 1/5 & 0 & -1 & -1/5 & 0 & 1 & 24/5 \\ 0 & -12/5 & -3/5 & 1 & 1 & 8/5 & 0 & 0 & -42/5 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & -2/9 & 0 & 1/9 & 2/9 & 0 & -1/9 & 5/3 \\ 0 & 0 & \boxed{1/3} & -1 & 1/3 & -1/3 & 1 & -1/3 & 2 \\ 0 & 1 & 1/9 & 0 & -5/9 & -1/9 & 0 & 5/9 & 8/3 \\ 0 & 0 & -1/3 & 1 & -1/3 & 4/3 & 0 & 4/3 & -2 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 & -2/3 & 1/3 & 0 & 2/3 & -1/3 & 3 \\ 0 & 0 & 1 & -3 & 1 & -1 & 3 & -1 & 6 \\ 0 & 1 & 0 & 1/3 & -2/3 & 0 & -1/3 & 2/3 & 2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}
\end{aligned}$$

Phase 2:

$$\begin{bmatrix} \boldsymbol{A} & \boldsymbol{b} \\ \boldsymbol{c}^T & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -2/3 & 1/3 & 3 \\ 0 & 0 & 1 & -3 & 1 & 6 \\ 0 & 1 & 0 & 1/3 & -2/3 & 2 \\ 4 & 3 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & -2/3 & 1/3 & 3 \\ 0 & 0 & 1 & -3 & 1 & 6 \\ 0 & 1 & 0 & 1/3 & -2/3 & 2 \\ 0 & 0 & 0 & 5/3 & 2/3 & -18 \end{bmatrix}$$

The optimal solution is given by $x_1^* = 3, x_2^* = 2$ with maximum function value as $-4x_1^* - 3x_2^* = -18$

For MATLAB function for this problem refer to Listing 16 at page 16 & Listing 17 at page 17 and the call to the function can be referred at Listing 15 at page 16 with corresponding output at Listing 18 at page 19.

Exercise 4

The dual problem is given by:

$$\begin{aligned} \text{maximize} \quad & \boldsymbol{\lambda}^T \boldsymbol{b} \\ \text{subject to} \quad & \boldsymbol{\lambda}^T \boldsymbol{A} \leq c^T \end{aligned}$$

that is:

$$\begin{aligned} \text{maximize} \quad & 11\lambda_1 + 8\lambda_2 + 7\lambda_3 \\ \text{subject to} \quad & \begin{bmatrix} \lambda_1 & \lambda_2 & \lambda_3 \end{bmatrix}^T \begin{bmatrix} 5 & 1 \\ 2 & 1 \\ 1 & 2 \end{bmatrix} \leq \begin{bmatrix} 4 & 3 \end{bmatrix} \\ & \lambda_1, \lambda_2, \lambda_3 \geq 0 \end{aligned}$$

Since we have already obtained the optimal BFS $\boldsymbol{x}^* = \begin{bmatrix} 3 & 2 & 0 & 0 & 0 \end{bmatrix}^T$ corresponding to the optimal basis $\boldsymbol{B} = \begin{bmatrix} 5 & 1 & -1 \\ 2 & 1 & 0 \\ 1 & 2 & 0 \end{bmatrix}$ and cost coefficients $\boldsymbol{c_B} = \begin{bmatrix} 4 & 3 & 0 \end{bmatrix}^T$. From theorem of duality, we have

$$\begin{aligned} \boldsymbol{\lambda}^T \boldsymbol{b} &= \boldsymbol{c}^T \boldsymbol{x} = \boldsymbol{c_B}^T \boldsymbol{B}^{-1} \boldsymbol{b} \\ \Rightarrow \boldsymbol{\lambda}^T &= \boldsymbol{c_B}^T \boldsymbol{B}^{-1} \\ &= \begin{bmatrix} 4 & 3 & 0 \end{bmatrix}^T \begin{bmatrix} 5 & 1 & -1 \\ 2 & 1 & 0 \\ 1 & 2 & 0 \end{bmatrix}^{-1} \\ &= \frac{1}{-3} \begin{bmatrix} 4 & 3 & 0 \end{bmatrix}^T \begin{bmatrix} 0 & -2 & 1 \\ 0 & 1 & -2 \\ 3 & -9 & 3 \end{bmatrix}^T \\ \boldsymbol{\lambda}^{*T} &= \begin{bmatrix} 0 & 5/3 & 2/3 \end{bmatrix} \\ \boldsymbol{\lambda}^{*T} \boldsymbol{b} &= 18 \end{aligned}$$

Exercise 5

We have to solve the following constrained optimization problem:

$$\begin{aligned}
\text{maximize} \quad & f(x) = 4x_1 + x_2^2 \\
\text{eqv minimize} \quad & -f(x) = -4x_1 - x_2^2 \\
\text{subject to} \quad & h_1(x) = x_1^2 + x_2^2 - 9 = 0 \\
\\
\text{lagrangian} \quad & l(\boldsymbol{x}, \lambda_1) = -4x_1 - x_2^2 + \lambda_1(x_1^2 + x_2^2 - 9)
\end{aligned}$$

Using FONC:

$$\begin{aligned}
\nabla_{\boldsymbol{x}} l(\boldsymbol{x}, \lambda_1) &= \begin{bmatrix} -4 & -2x_2 \end{bmatrix} + \lambda_1 \begin{bmatrix} 2x_1 & 2x_2 \end{bmatrix} = \mathbf{0}^T \\
\Rightarrow & \begin{bmatrix} -4 + 2\lambda_1 x_1 \\ 2x_2(\lambda_1 - 1) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
\nabla_{\lambda_1} l(\boldsymbol{x}, \lambda_1) &= x_1^2 + x_2^2 - 9 = 0
\end{aligned}$$

The solution to the above set of equations is:

$$\boldsymbol{x}_a^* = \begin{bmatrix} x_1 \\ x_2 \\ \lambda_1 \end{bmatrix} = \begin{bmatrix} \pm 3 \\ 0 \\ \pm\frac{2}{3} \end{bmatrix}, \quad \boldsymbol{x}_b^* = \begin{bmatrix} x_1 \\ x_2 \\ \lambda_1 \end{bmatrix} = \begin{bmatrix} 2 \\ \pm\sqrt{5} \\ 1 \end{bmatrix}$$

Computing the Hessian of lagrangian:

$$\begin{aligned}
L(\boldsymbol{x}, \lambda_1) &= 2 \begin{bmatrix} \lambda_1 & 0 \\ 0 & (\lambda_1 - 1) \end{bmatrix} \\
\Rightarrow L(\boldsymbol{x}_b^*, \lambda_1^* = 1) &= 2 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \succcurlyeq 0 \Rightarrow \boldsymbol{x}_b^* \text{ is a possible minimizer of } -f(\boldsymbol{x}) \\
L(\boldsymbol{x}_{a2}^*, \lambda_1^* = -\frac{2}{3}) &= 2 \begin{bmatrix} -\frac{2}{3} & 0 \\ 0 & -\frac{5}{3} \end{bmatrix} \preccurlyeq 0 \Rightarrow \boldsymbol{x}_b^* \text{ is a possible maximizer of } -f(\boldsymbol{x})
\end{aligned}$$

For SONC we need to find the Tangent Space $T(\boldsymbol{x}^*)$:

$$\begin{aligned}
T(\boldsymbol{x}_b^*) &= \{y : \nabla_{\boldsymbol{x}} h_1^T(x) y = 0\} \\
&= \{y : \begin{bmatrix} 2x_1^* & 2x_2^* \end{bmatrix} y = 0\} \\
&= \{y : \begin{bmatrix} 2 & \pm\sqrt{5} \end{bmatrix} y = 0\} \\
&= \{y : y = t \begin{bmatrix} 1 \\ \mp\frac{\sqrt{5}}{2} \end{bmatrix}, \quad t \in \mathbb{R} - \{0\}\} \\
y^T L(\boldsymbol{x}_b^*) y &= t^2 \begin{bmatrix} 1 & \mp\frac{\sqrt{5}}{2} \end{bmatrix} 2 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ \mp\frac{\sqrt{5}}{2} \end{bmatrix} \\
&= 2t^2 > 0 \quad \forall \quad t \in \mathbb{R} - \{0\}
\end{aligned}$$

Therefore $\boldsymbol{x}_b^* = \begin{bmatrix} 2 \\ \pm\sqrt{5} \\ 1 \end{bmatrix}$ is a strict minimizer of $-f(x)$ and thus is a maximizer of $f(x)$

Exercise 6

We have to solve the following constrained optimization problem:

$$\begin{aligned} \text{maximize} \quad & f(x) = 18x_1^2 - 8x_1x_2 + 12x_2^2 \\ \text{subject to} \quad & h(x) = 1 - 2x_1^2 + 2x_2^2 = 0 \end{aligned}$$

Converting to minimization problem:

$$\begin{aligned} \text{minimize} \quad & -f(x) = -\frac{1}{2}\boldsymbol{x}^T \begin{bmatrix} 36 & -8 \\ -8 & 24 \end{bmatrix} \boldsymbol{x} \\ \text{subject to} \quad & h(x) = 1 - \frac{1}{2}\boldsymbol{x}^T \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} \boldsymbol{x} \\ \text{lagrangian} \quad & l(x, \lambda) = -\frac{1}{2}\boldsymbol{x}^T \begin{bmatrix} 36 & -8 \\ -8 & 24 \end{bmatrix} \boldsymbol{x} + \lambda(1 - \frac{1}{2}\boldsymbol{x}^T \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} \boldsymbol{x}) \end{aligned}$$

FONC:

$$\begin{aligned} \nabla_\lambda l(\boldsymbol{x}, \lambda) &= 2x_1^2 + 2x_2^2 - 1 = 0 \\ \nabla_{\boldsymbol{x}} l(\boldsymbol{x}, \lambda) &= \begin{bmatrix} -36 & 8 \\ 8 & -24 \end{bmatrix} \boldsymbol{x} - \lambda(\begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} \boldsymbol{x}) = \mathbf{0} \\ &\Rightarrow \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}^{-1} \begin{bmatrix} -36 & +8 \\ +8 & -24 \end{bmatrix} \boldsymbol{x} = \lambda \boldsymbol{x} \\ &\Rightarrow \begin{bmatrix} -9 & +2 \\ +2 & -6 \end{bmatrix} \boldsymbol{x} = \lambda \boldsymbol{x} \\ &\Rightarrow det(\begin{bmatrix} -9 & +2 \\ +2 & -6 \end{bmatrix} - \lambda \boldsymbol{I}_2) = 0 \Rightarrow \lambda = -10, -5 \\ \boldsymbol{x} &= \alpha \begin{bmatrix} -2 \\ 1 \end{bmatrix}, \beta \begin{bmatrix} 1 \\ 2 \end{bmatrix} \end{aligned}$$

The solution to the above equations is:

$$\boldsymbol{x}_a^* = \begin{bmatrix} x_1 \\ x_2 \\ \lambda \end{bmatrix} = \begin{bmatrix} \mp\frac{2}{\sqrt{10}} \\ \pm\frac{1}{\sqrt{10}} \\ -10 \end{bmatrix}, \quad \boldsymbol{x}_b^* = \begin{bmatrix} x_1 \\ x_2 \\ \lambda \end{bmatrix} = \begin{bmatrix} \pm\frac{1}{\sqrt{10}} \\ \pm\frac{2}{\sqrt{10}} \\ -5 \end{bmatrix}$$

SONC:

$$\begin{aligned} L(\boldsymbol{x}, \lambda) &= \begin{bmatrix} -36 & 8 \\ 8 & -24 \end{bmatrix} - \lambda \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} \\ &= (\begin{bmatrix} -9 - \lambda & +2 \\ +2 & -6 - \lambda \end{bmatrix}) \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} \\ \Rightarrow L(\boldsymbol{x}, \lambda = -10) &= \begin{bmatrix} 4 & 8 \\ 8 & 16 \end{bmatrix} \succcurlyeq 0 \text{ is a possible minimizer of } -f(x) \\ \Rightarrow L(\boldsymbol{x}, \lambda = -5) &= \begin{bmatrix} -16 & +8 \\ +8 & -4 \end{bmatrix} \preccurlyeq 0 \text{ is a possible maximizer of } -f(x) \end{aligned}$$

SOSC: We need to find the Tangent Space $T(x^*)$

$$\begin{aligned}
T(\boldsymbol{x}_a^*) &= \{y : \nabla_{\boldsymbol{x}} h_1^T(x) y = 0\} \\
&= \{y : \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} \mp\frac{2}{\sqrt{10}} \\ \pm\frac{1}{\sqrt{10}} \end{bmatrix} y = 0\} \\
&= \{y : y = t \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad t \in \mathbb{R} - \{0\}\} \\
\Rightarrow y^T L(\boldsymbol{x}_a^*) y &= 100t^2 > 0 \quad \forall \quad t \in \mathbb{R} - \{0\}
\end{aligned}$$

Therefore the solution corresponding to $\lambda = -10$, $\boldsymbol{x}_a^* = \begin{bmatrix} \mp\frac{2}{\sqrt{10}} \\ \pm\frac{1}{\sqrt{10}} \end{bmatrix}$ is a strict minimizer of $-f(x)$ and thus is a maximizer of $f(x)$

Exercise 7

MATLAB Code

Canonical GA Code

Listing 1: Canonical GA Main Code

```
% ECE 580 HW5: Problem 1
% Rahul Deshmukh
% deshmuk5@purdue.edu
clc; clear all; close all;
format short;
%% Include paths
addpath('../OptimModule/optimizers/global/GA/');
save_dir = './pix/';
%% set seed
rng(7); % tried 0:10, 7 gave the smallest result, can jump between global and ...
    local min
%% Canonical GA problem setup
lb = -5*ones(1,2);
ub = 5*ones(1,2);
Num_vars = length(lb);
bits = 10;
coded_lens = bits*ones(1,Num_vars);
resolution = (ub-lb)./(2.^coded_lens-1)

%% GA: solver params
N_pop = 40; % cant be odd integer
p_xover = 0.9;
p_mut = 0.01;
Niters = 30;
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
X = rand(N_pop, Num_vars);
% scale to domain
X = (X.*(ub-lb) + lb);
% discretize to resolution
X = floor((X - lb)./resolution).*resolution + lb;

%encode X
parents = encode(X, lb, ub, coded_lens, resolution);
```

```
% evaluate fitness of parents
f_parent = -1*fitness_griewank(parents, lb, coded_lens, resolution);
[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 = two_point_crossover(mating_pool, p_xover);
   %perform mutation
   parents = mutation(parents, p_mut);
   %perform elitism
   parents = elitism(parents, f_parent);
   %evaluate fitness of offspring
   f_parent = -1*fitness_griewank(parents, lb, coded_lens, resolution);
   [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('best fval: \t',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 = 1:5: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');
xlim([1, Niters+1])
ylim(max(abs(worse_f))*(1.1)*[-1,1]);
title('Convergence of GA');
saveas(fig1,strcat(save_dir,'ga_canon_conv'),'epsc');
```

Listing 2: Fitness function

```
function f = fitness_griewank(X_coded, lb, code_lens, resolution)
X = decode(X_coded, lb, code_lens, resolution);
f = griewank_fun(X');
end
```

Listing 3: 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 4: Encoding function

```
function X_coded = encode(X, lb, ub, code_lens, resolution)
[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);
% convert discretized X to integers for encoding
X = round((X - lb)./resolution);
for i = 1:N_pop
    x = X(i,:);
    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 5: 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 6: 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 7: 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 8: Cross-over function

```
function offspring = two_point_crossover(mating_pool, p_xover)

[N_pop,L] = size(mating_pool);
% shuffle parents
mating_pool = mating_pool(randperm(N_pop),:);
% generate rand nums for deciding if do crossover?
rand_nums = rand(1,round(N_pop/2));
do_xover = rand_nums > (1-p_xover);
offspring = zeros(N_pop, L);

for k = 1:round(N_pop/2)
    parents = mating_pool([2*k-1 ,2*k],:);
    if do_xover(k)
        % find crossover point
        xover_pt = randi(L,1);
        % switch genes
        offspring(2*k-1,:)= [parents(1,1:xover_pt), parents(2,xover_pt+1:end)];
        offspring(2*k,:) = [parents(2,1:xover_pt), parents(1,xover_pt+1:end)];
    else
        offspring([2*k-1, 2*k],:) = parents;
    end
end

end
```

Listing 9: Mutation function

```
function mutated = mutation(parents, p_mut)
[N_pop,¬] = size(parents);
rand_nums = rand(N_pop,1);
do_mut_idx = find(rand_nums < p_mut);
mutated = parents;
% complement each bit in parent
mutated(do_mut_idx,:) = 1-parents(do_mut_idx,:);
end
```

Listing 10: 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 11: Logging function

```
function [best_f, av_f, worse_f] = log_f(f_parent, best_f, av_f, worse_f)
best_f = [best_f, max(f_parent)];
av_f = [av_f, mean(f_parent)];
worse_f = [worse_f, min(f_parent)];
end
```

Real GA Code

Listing 12: Real GA Main Code

```
% ECE 580 HW5: Problem 2
% Rahul Deshmukh
% deshmuk5@purdue.edu
clc; clear all; close all;
format short;
%% Include paths
addpath('../OptimModule/optimizers/global/GA/');
addpath('../OptimModule/optimizers/global/GA/Real_Num_GA/');
save_dir = './pix/';
%% set seed
rng(6); % tried 0:10 ,6 was the best and always converges to global min
%% Real GA problem setup
lb = -5*ones(1,2);
ub = 5*ones(1,2);
Num_vars = length(lb);

%% GA: solver params
N_pop = 40; % cant be odd integer
p_xover = 0.9;
p_mut = 0.01;
Niters = 30;
selection_method = 'tournament_method2';
xover_method = 'conv_combo';

%% 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
X = rand(N_pop, Num_vars);
% scale to domain
X = (X.*(ub-lb) + lb);

parents = X;
% evaluate fitness of parents
f_parent = -1*griewank_fun(parents');
[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, xover_method);
```

```
    %perform mutation
    parents = mutation(parents, p_mut, lb, ub);
    %perform elitism
    parents = elitism(parents, f_parent);
    %evaluate fitness of offspring
    f_parent = -1*griewank_fun(parents');
    [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('best fval: \t',num2str(-1*f_star)))
x_star = parents(k_star,:)

%% 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 = 1:5: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');
xlim([1, Niters+1])
ylim(max(abs(worse_f))*(1.1)*[-1,1]);
title('Convergence of Real GA');
saveas(fig1,strcat(save_dir,'ga_real_conv'),'epsc');
```

Listing 13: Cross-over function

```
function offspring = crossover(mating_pool,p_xover, method)
switch nargin
    case 2
        method='av';
end
%shuffle mating pool
[Npop,¬] =  size(mating_pool);
mating_pool = mating_pool(randperm(Npop), :);
rand_num = rand(round(Npop/2),1);
do_xover = rand_num < p_xover;
do_xover_idx = find(do_xover);

alpha = rand(length(do_xover_idx),1);
offspring = mating_pool;
if strcmp(method,'av')
    for i = 1:length(do_xover_idx)
        k = do_xover_idx(i);
        parents = mating_pool([2*k-1 2*k],:);
        offspring([2*k-1 2*k], :) = [sum(parents, 1)/2.0;
                                        parents((alpha(i)> 0.5)+1,:)];
    end
elseif strcmp(method,'conv_combo')
    for i = 1:length(do_xover_idx)
```

```
        k = do_xover_idx(i);
        parents = mating_pool([2*k-1 2*k],:);
        offspring([2*k-1 2*k], :) = [alpha(i), 1-alpha(i);
                                     1-alpha(i), alpha(i)]*parents;
    end
else
    error('Method for crossover not implemented');
end

end
```

Listing 14: Mutation function

```
function mutated = mutation(parents, p_mut, lb, ub)
[N_pop,¬] = size(parents);
rand_nums = rand(N_pop,1);
do_mut_idx = find(rand_nums < p_mut);
mutated = parents;
alpha = rand(length(do_mut_idx),1);
w = rand(length(do_mut_idx),length(lb));
% scale and translate w to domain
w = w.*(ub-lb) + lb;
mutated(do_mut_idx,:) = parents(do_mut_idx,:).*alpha + w.*(1-alpha);
end
```

Linear programming Code

Listing 15: Linprog Main Code

```
% ECE 580 HW5: Problem 3
% Rahul Deshmukh
% deshmuk5@purdue.edu
clc; clear all;
format rat;
%% include paths
addpath('../OptimModule/optimizers/linprog/');
%%
verbose=0;
c = [4; 3];
A = [-5, -1;
     -2, -1;
     -1, -2];
b = [-11; -8; -7];
Aeq = [];
beq = [];
LB=[];
UB=[];
[x_str, fval] = mylinprog(c,A,b,Aeq,beq,LB,UB,verbose)
```

Listing 16: Two Phase Simplex

```
function [x_str, fval] = mylinprog(c,A,b,Aeq,beq,LB,UB,verbose)
% Two Phase Simplex LP solver
```

```
% the problem is not given in std form
% Given: min c'*x
% st:    Aeq*x = beq
%        Ax ≤ b
%        x≥0       %%code not generalized to accept LB,UB
if nargin==7
    verbose=0; %by default
end
if ¬isempty(LB) || ¬isempty(UB)
    error('Not implemented');
end
%% Convert to std form using slack and surplus variables st b_std≥0
A_std =[]; b_std = []; c_std =[];
A_std = [A_std; A, eye(size(A,1))];
b_std = [b_std;abs(b)];
c_std = [c_std;c;zeros(size(A,1),1)];
% make inequality b's positive
neg_b_idx = find(b < 0);
A_std(neg_b_idx,:) = -1*A_std(neg_b_idx,:);
% stack equality equations
if ¬isempty(Aeq)
    neg_b_idx = find(beq < 0);
    Aeq(neg_b_idx,:) = -1*Aeq(neg_b_idx,:);
    A_std = [A_std; Aeq, zeros(size(Aeq,1),size(A,1))];
    b_std = [b_std; abs(beq)];
end
%% Solve std form using two-phase simplex method
if verbose
    fprintf('\\text{Phase 1:}&\\nonumber\\\\\n%%\n');
end
% Phase1: Find initial basis using artificial variables
A1 = [A_std, eye(size(A_std,1))];
c1 = [zeros(size(A_std,2),1);ones(size(A_std,1),1)];
basis_idx = size(A_std,2) + (1:size(A_std,1));
[x_p1, fval_p1, basis_idx_p1, tab]= simplex(A1,b_std,c1, basis_idx, verbose);
if verbose
   fprintf('\\text{Phase 2:}&\\nonumber\\\\\n%%\n') ;
end
% Phase2: Find optimal solution
A2 = tab(1:end-1,1:size(A_std,2));
b2 = tab(1:end-1,end);
[x_str_p2, fval, basis_idx_p2, ¬] = simplex(A2, b2, c_std, basis_idx_p1, verbose);
x_str = x_str_p2(1:length(c));
fprintf('\n** Optimum Solution found using mylinprog **\n');
if verbose
    display(x_str);
    display(fval);
end
end
```

Listing 17: Simplex Method

```
function [x_str, fval, basis_idx, tab]= simplex(A,b,c, basis_idx, verbose)
% simplex method to solve LP in std form:
% Given: min c^Tx
% st:    Ax = b
%        x≥0
% basis_idx the indices are in order of std cartesian basis 1...n
```

```
tol = 1e-6;
tab = [A, b;
       c',0];
if verbose
    print_tab(tab);
end

% make cost coeffs zero for basis idx
for i=1:length(basis_idx)
%     r = find(tab(1:end-1,basis_idx(i))>0);
    tab(end,:) = tab(end,:) - tab(end,basis_idx(i))*tab(i,:);
end

% till all cost coeffs are non-negative do:
while ¬isempty(find(tab(end,1:end-1)< 0))
% choose pivot column
    [¬,p] = min(tab(end,1:end-1));
% find pivot element
    % using only positive a_p's
    pos_ap_idx = find(tab(1:end-1,p)>0);
    [¬,q_idx] = min(tab(pos_ap_idx,end)./tab(pos_ap_idx, p));
    q = pos_ap_idx(q_idx);
    % update basis idx
    basis_idx(q) = p;
% print tab with pivot element
    if verbose
        print_tab(tab,[q,p]);
    end
    % make pivot 1
    tab(q,:) = tab(q,:)/tab(q,p);
% make pivot column
    for r=1:size(tab,1)
        if r≠q
            tab(r,:) = tab(r,:) - tab(r,p)*tab(q,:);
        end
    end
    tab(tab>-tol & tab<tol) = 0;
end
if verbose
    print_tab(tab)
end
%return x_str and basis_idx
x_str = zeros(length(c),1);
x_str(basis_idx) = tab(1:end-1,end);
fval= c'*x_str;
end

function print_tab(tab, pivot)
if nargin==1
    pivot = 0;
end
% display(tab)
% print latex bmatrix
[n_row, n_col] = size(tab);
fprintf('&=\\begin{bmatrix}\n');
for r=1:n_row
    for c=1:n_col-1
        if r==pivot(1) && c== pivot(2)
            % this is the pivot
```

```
            fprintf('\\fbox{{\\color{red}%s}}&\t',strtrim(rats(tab(r,c))));
        else
            fprintf('%s&\t',strtrim( rats(tab(r,c)) ));
        end
    end
    fprintf('%s\\\\\n',strtrim( rats(tab(r,c+1)) ));
end
fprintf('\\end{bmatrix}\\nonumber\n%%\n');
end
```

Listing 18: LP output

```
 ** Optimum Solution found using mylinprog **

x_str =

       3
       2


fval =

      18
```