1 Exercise PP. 7 - 1

Considering the Rosenbrock's parabolic valley:

$$f(x_1, x_2) = 100(x_1\ x_1^2)^2 + (1 - x_1)^2$$

Obtain the point of minima using a Genetic Algorithm.

1.1 MATLAB Files

The genetic algorithm that follows performs the minimisation of the objective function for 10 different initial populations and outputs the results from the first one and the average of all of them, plotting the search path, the population at different generation and a comparison graph between the standard deviation, the mean and the best fitness values.

GA_main.m

```
%% GA_main.m
% Main script of the GA
% Calls: genetic

addpath([pwd,'\Functions';],[pwd,'\Functions\Plot'],[pwd,'\Functions\Genetic_Operators']);
clc
clear
close all

time_total = 0;
X_total = [0,0];
f_total = 0;
fitness_total = 0;
k_total = 0;

for m=1:10
    [time,X_best, best_fitness,f,k] = genetic(m);
    time_total = time_total + time;
    X_total = X_total + X_best;
    fitness_total = fitness_total + best_fitness(1);
    k_total = k_total + k;

end

time_average = time_total/m;
X_average = X_total/m;
fitness_average = fitness_total/m;
```

```
k_average = k_total/m;


% Results Print
f_average = f(X_average(1),X_average(2));
error = abs(f_average-f(1,1));
accuray = (1-error)*100;

fprintf([ ...
         'Average results of %i different initial  populations:
             \n', ...
         'Number of generations: %.0f\n', ...
         'Approximate Solution: [%.5f , %.5f]\n', ...
         'Objective function value: %.4f \n', ...
         'Maximum fitness value: %.4f \n', ...
         'Precision: %.5f%% \n', ...
         'Average time: %.4fs\n'], ...
         m, k_average, X_average(1),X_average(2),f_average,
            fitness_average, accuray, time_average)
```

PP7_data.m

```
%% PP7_data.m
% Problem Data
% Calls: truss_constraints

% Data
lb = [0; 0];    % Lower limits
ub = [2; 2]; % Upper limits

% Objective Function
f = @(x1,x2) 100.*(x2-x1.^2).^2+(1-x1).^2;

% Population Parameters
N_pop = 100;         % Number of individuals
l_c = 300;           % Chromosome Length
b = [l_c/2 l_c/2]; % Number of alleles per gene

% Max number of iterations
kmax = 1000;

% Tolerance
tol = 1e-6;
```

genetic.m

```
%% genetic.m
% Genetic algorithm
% Calls: truss_data, population_generation, fitness_function,
   crossover, mutation, Evolutionary_cycle, plot_bestfitness,
   plot_bestfitness_path, plot_pop_evolution,
   plot_pop_evolution2

function [time,X_best,fitness,f,k] = genetic(m)
tic

% Data Initialization
PP7_data

% Initial Population of solutions generation
[Population]=population_generation(N_pop,l_c);
k=0;

% Fitness Evaluation
pop_final=0;
[fitness,X_pop,Population] = fitness_function(Population,N_pop,
   lb,ub,l_c,f);

if m == 1
    % Plot - Fittest Individual for each Generation
    plot_bestfitness(f,lb,ub);
end

while k<kmax && std(fitness)>tol

    k=k+1;

    % Save Current Generation
    X_pop_old = X_pop;

    if m == 1
        pop_evolution(:,:,k) = X_pop;
        std_fitness(k) = std(fitness);
        mean_fitness(k) = mean(fitness);
        best_fitness(k) = fitness(1);
    end

    % Genetic Operators
```

```
    Offsprings_population = crossover(l_c,N_pop,fitness,Population);
    Offsprings_population = mutation(Offsprings_population,size(Offsprings_population,1),l_c);

    % Fitness evaluation of the individuals in the sons' population
    [fitness_offsprings,~,Offsprings_population] = fitness_function(Offsprings_population,size(Offsprings_population,1),lb,ub,l_c,f);

    [fitness,~,Population] = fitness_function(Population,N_pop,lb,ub,l_c,f);

    % Survivors selection
    [Population_new,fitness_new] = Evolutionary_cycle(Population,N_pop,Offsprings_population,fitness,fitness_offsprings);
    Population = Population_new;
    fitness = fitness_new;

    % Decoding
    X_pop = decoding(Population,N_pop,lb,ub,l_c);

    if m == 1
        % Plot - Current Generation
        plot_bestfitness_path(X_pop,X_pop_old,pop_final)
    end

end

% Final Population Fittest Individual
X_best = X_pop(1,:);

time = toc;

if m == 1
    % Plot - Final Population
    pop_final=1;
    plot_bestfitness_path(X_pop,X_pop_old,pop_final);

    % Plot - Generations Evolution
    pop_evolution(:,:,k+1) = X_pop;
    plot_pop_evolution(pop_evolution,X_pop,k,f,lb,ub);

```

```
    % Plot - Generations Evolution - Standard Deviation and
       Mean
    std_fitness(k+1) = std(fitness);
    mean_fitness(k+1) = mean(fitness);
    best_fitness(k+1) = fitness(1);
    plot_pop_evolution2(std_fitness,mean_fitness,best_fitness,k
       );

    % Results
    fobj=f(X_best(1),X_best(2));
    fprintf(['First Initial Population: \n',...
             'Number of generations: %d\n', ...
             'Approximate Solution: [%.5f , %.5f]\n', ...
             'Objective function value: %.4f \n\n\n'], ...
             k,X_best(1),X_best(2),fobj)
end
```

population_generation.m

```
%% population_generation.m
% Initial population generation

function [Population]=population_generation(N_pop,l_c)

Population1 = randi([0 1], N_pop, l_c);

% Eliminate Clones
Population = unique(Population1,'rows');

while size(Population1,1) > size(Population,1)
    difference = size(Population1,1) - size(Population,1);
    Population_add = randi([0 1], difference, l_c);
    Population1 = [Population; Population_add];
    Population = unique(Population1,'rows');
end
```

Evolutionary_cycle.m

```
%% Evolutionary_cycle.m
% Survivors Selection

function [Population_new,fitness_new] = Evolutionary_cycle(
   Population, ...
```

```
            N_pop,Offsprings_population,fitness,
               fitness_offsprings)

% Selection (Fitness)
Natural_selection = [Population; Offsprings_population];
Natural_selection_fitness = [fitness; fitness_offsprings];

[Ordered_fitness,I] = sort(Natural_selection_fitness,1,'descend
   ');
Population_new=Natural_selection(I(1:N_pop),:);
fitness_new = Ordered_fitness(1:N_pop);
```

fittness_fucntion.m

```
%% fitness_function.m
% Fitness Function
% Calls: decoding

function [fitness,X_real_ordered,population_ordered] =
   fitness_function(population,N_pop,lb,ub,l_c,f)

% Decoding
X_real = decoding(population,N_pop,lb,ub,l_c);

% Fitness evaluation
fitness=-f(X_real(:,1),X_real(:,2));

% Sorted Population (descend)
[fitness,position_pop]=sort(fitness,'descend');
X_real_ordered = X_real(position_pop,:);
population_ordered = population(position_pop,:);
end
```

decoding.m

```
%% decoding.m
% Genotype decoding (bits) to fenotype (real) - binary coding
   emulating
% real representation

function [X_real] = decoding(Population,N_pop,lb,ub,l_c)

X_bin=zeros(N_pop,2);
```

```
X_real=zeros(N_pop,2);

for i=1:N_pop
    for j=1:l_c/2
        %x1
        X_bin(i,1) = X_bin(i,1)+Population(i,j)*2^(l_c/2-j);

        %x2
        X_bin(i,2) = X_bin(i,2)+Population(i,j+l_c/2)*2^(l_c/2-
           j);
    end
end

for i=1:N_pop
    %x1
    X_real(i,1) = lb(1)+(ub(1)-lb(1))/(2^(l_c/2) - 1)*X_bin(i
       ,1);

    %x2
    X_real(i,2) = lb(2)+(ub(2)-lb(2))/(2^(l_c/2) - 1)*X_bin(i
       ,2);
end
```

selection.m

```
%% selection.m
% Selection Operator

function [parent1, parent2, roulette] = selection(fitness,N_pop
   ,population,i, roulette)

if i == 1
    % Selection probability for each individual
    selection_prob = fitness/sum(fitness);

    % Roulette intervals
    roulette=zeros(N_pop,1);
    roulette(1) = selection_prob(1);
    for i=2:N_pop
        roulette(i)=roulette(i-1) + selection_prob(i);
    end
end

% Select first parent
```

```
selection_value = rand();
parent_position1 = find(selection_value<roulette,1);
parent1 = population(parent_position1,:);

% Select second parent
selection_value = rand();
parent_position2 = find(selection_value<roulette,1);

while parent_position2==parent_position1
selection_value = rand();
parent_position2 = find(selection_value<roulette,1);
end

parent2 = population(parent_position2,:);
```

crossover.m

```
%% crossover.m
% Crossover operator
% Calls: selection

function [offspring_pop] = crossover(l_c,N_pop,fitness,
   population)

offspring_pop = zeros(N_pop,l_c);
son1 = zeros(1,l_c);
son2 = zeros(1,l_c);
roulette=0;

for i=1:N_pop/2
    % Parent Selection
    [parent1, parent2,roulette] = selection(fitness,N_pop,
       population,i,roulette);

    % Mask
    mask = randi([0,1],1,l_c);

    for j=1:l_c
        if mask(j) == 0
            son1(j) = parent1(j);
            son2(j) = parent2(j);
        else
            son1(j) = parent2(j);
            son2(j) = parent1(j);
```

```
        end
    end

    offspring_pop(i*2-1,:) = son1; % Son 1
    offspring_pop(i*2,:) = son2;   % Son 2
end
```

mutation.m

```
%% mutation.m
% Mutation operator

function [offspring_pop] = mutation(offspring_pop,N_pop,l_c)

for i=1:N_pop
    for j=1:l_c
        prob_mut = rand();

        if prob_mut <= 0.01
            if offspring_pop(i,j)==0
                offspring_pop(i,j) = 1;
            else
                offspring_pop(i,j) = 0;
            end
        end
    end
end
```

plot_bestfitness.m

```
%% File: plot_bestfitness.m
% Plot fittest indivual for each generation

function [fig] = plot_bestfitness(ff,lb,ub)

fig = figure(1);
fcontour(ff,[lb(1) ub(1) lb(2) ub(2)])
title('$f(x_1,x_2)=100(x_2-x_1^2)^2+(1-x_1)^2$','Interpreter','
    latex')
xlabel('$x_1$',Interpreter='latex')
ylabel('$x_2$',Interpreter='latex')
grid on
hold on
```

plot_bestfitness _path.m

```
%% plot_bestfitness_path.m
% Plot - Current Population

function [fig] = plot_bestfitness_path(X_pop,X_pop_old,
    pop_final)

if pop_final == 0
    plot([X_pop_old(1,1),X_pop(1,1)],[X_pop_old(1,2),X_pop(1,2)
        ],'o-r');
else
    plot([X_pop_old(1,1),X_pop(1,1)],[X_pop_old(1,2),X_pop(1,2)
        ],'o-k',MarkerFaceColor='k');
end
drawnow
```

plot_pop _evolution.m

```
%% plot_pop_evolution.m
% Generation evolution plot

function [fig] = plot_pop_evolution(pop_evolution,X_pop,k,ff,lb
    ,ub)

fig = figure(2);
tiledlayout('flow')

a=round((k+1)/5);
for i=1:a:(1+4*a)
    nexttile
    fcontour(ff,[lb(1) ub(1) lb(2) ub(2)])
    xlabel('$x_1$','Interpreter','latex')
    ylabel('$x_2$','Interpreter','latex')
    hold on

    scatter(pop_evolution(:,1,i),pop_evolution(:,2,i),"red")
    title(['Generation ',num2str(i-1)],'Interpreter','latex')
end

nexttile
fcontour(ff,[lb(1) ub(1) lb(2) ub(2)])
xlabel('$x_1$','Interpreter','latex')
ylabel('$x_2$','Interpreter','latex')
```

```
hold on

scatter(X_pop(:,1),X_pop(:,2),"red")
title(['Generation  ',num2str(k)],'Interpreter','latex')
```

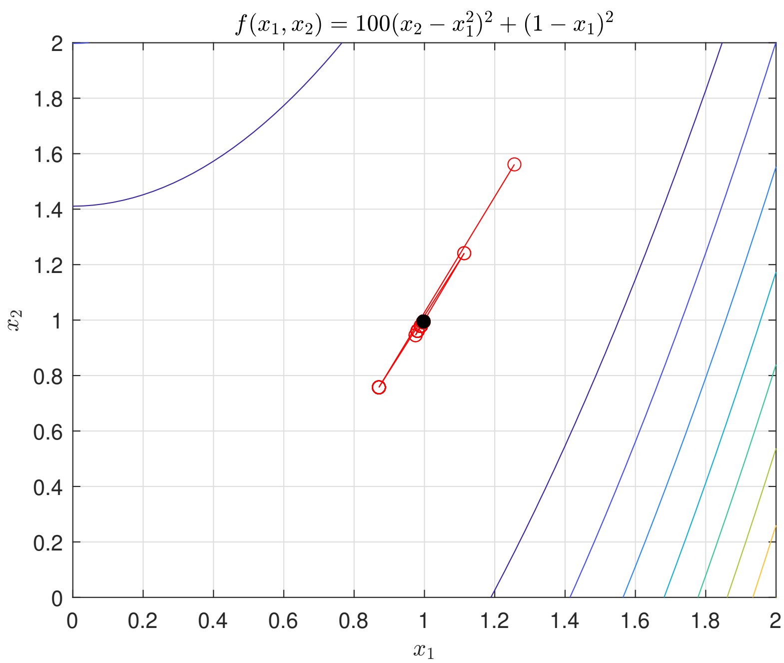
plot_pop _evolution2.m

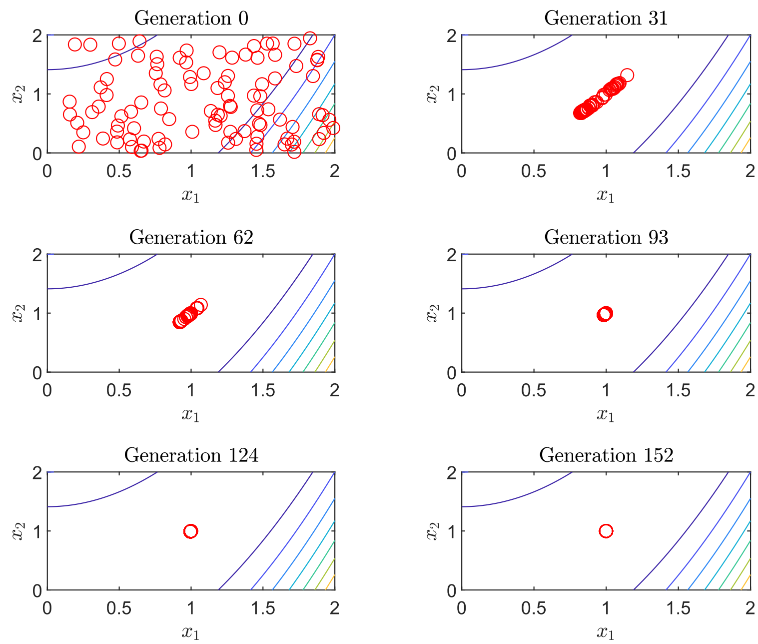
```
%% plot_pop_evolution2.m
% Generation Evolution plot - stantdard deviation and mean

function [fig] = plot_pop_evolution2(std_fitness,mean_fitness,
    best_fitness,k)

fig = figure(3);

plot(0:k,std_fitness,"blue",'LineWidth',2) % Plot standard
    deviation
hold on

plot(0:k,mean_fitness,"red",'LineWidth',2) % Plot mean
hold on

plot(0:k,best_fitness,"green",'LineWidth',2) % Plot best
    fitness

title('Fitness','Interpreter','latex')
xlabel('Generation','Interpreter','latex')
xlim([0 k])

legend('Standard Deviation','Mean','Best Fitness','Interpreter'
    , ...
        'latex','Location', 'northeast')
```

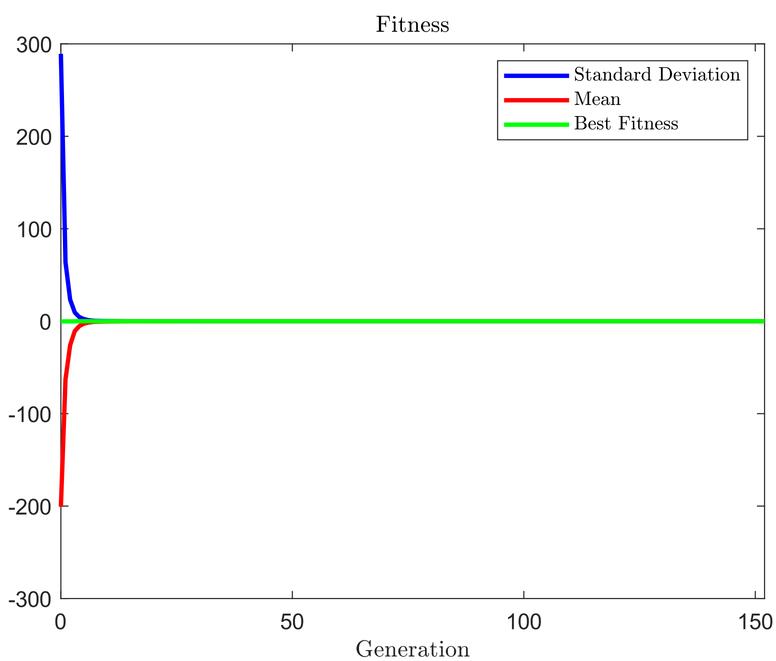
1.2 Results

The results below were obtained for a population with 100 individuals, a chromosome length equal to 300 and with 150 alleles per gene. The precision shown in the results was calculated using the objective function value for the exact point of minima, which is $f(1,1) = 0$.

```
First Initial Population:
Number of generations: 152
Approximate Solution: [0.99992 , 0.99985]
Objective function value: 0.0000


Average results of 10 different initial  populations:
Number of generations: 142
Approximate Solution: [0.99989 , 0.99978]
Objective function value: 0.0000
Maximum fitness value: -0.0000
Precision: 100.00000%
Average time: 1.1266s
```




Generation 0
Generation 31
Generation 62
Generation 93
Generation 124
Generation 152
x_1
x_2
0
0.5
1
1.5
2


Fitness
Standard Deviation
Mean
Best Fitness
300
200
100
0
-100
-200
-300
0
50
100
150
Generation