INDIAN INSTITUE OF TECHNOLOGY KHARAGPUR

MT21104 - GENETIC ALGORITHMS
IN ENGINEERING PROCESS MODELING
By - Prof. Nirupam Chakraborti

PROJECT – IMPLEMENTATION OF REAL ENCODING SIMPLE GENETIC ALGORITHM TO EVALUATE THE OPTIMUM VALUE OF A GIVEN FUNCTION.

By:

Karthik Reddy Yeredla

19MA20058

Department of Mathematics

2nd year Under graduate student

IIT Kharagpur.

Pseudo Code of Simple Genetic Algorithm used here:

```
Initialization of random population
Let N = number of generations
 for i = 1 to N
            // Selection of parents from the population
                 Stochastic Remainder Selection
           // Crossover for the selected parents with probability
                 Crossover
           // Mutation for the children after the crossover
          // with probability
                 Mutation
         // Update the population to the children after mutation
End
Return the optimized value of the given function
```

Matlab Implementation of the algorithm:

Initially, for simplicity we are taking only one-dimensional decision variable space.

Let the function that need to be optimized be F(x) = $3x^2 - x^3$.

Store the function as a matlab function f(x).

Code:

```
% THIS FUNCTION IS TO BE OPTIMIZED
% BY REAL ENCODING SIMPLE GENETIC ALGORITHM

function y = f(x)
    y = 3*x*x - x*x*x;
end
```

Here we need to optimize the function f(x) then fitness function can be described as shown below. But here it returns a vector of fitness values for the given population.

Fitness function code:

```
function y = fitness(x,N)
    y = zeros(N,1);
    for i = (1:N)
        y(i) = f(x(i));
    end
end
```

Next, we are going to define the operators used for the algorithm.

Selection operator:

The process of selection used here is **stochastic remainder selection***.*

Matlab code for selection function:

```
% THIS FUNCTION RETURNS THE PARENTS FROM THE GIVEN
POPULATION
% USING STOCHASTIC REMAINDER SELECTION

function S = selection(P,N)
   F = fitness(P,N);
   S = floor((F./sum(F)).*N);
   parents = P;
   count = 1;
   for i =(1:N)
       if S(i) > 0
           for j = (count:count+S(i)-1)
               parents(j)= P(i);
           end
           count = count + S(i);
       end
   end
   if sum(S) < N
       prob_distribution = (F./sum(F)).*N -
floor((F./sum(F)).*N);
       k = randsample(P.',N-
count+1,true,prob_distribution.').';
       for i = (count:N)
           parents(i) = k(i-(count-1));
       end
   end
   S = parents;
end
```

Crossover operator:
Let's take the parents to be P_1 and P_2. The method for crossover between two parents is as described below.

Let ϕ be a random generated number between 0 and 1.

As the crossover occurs with probability(p_cross). Generally p_cross will be in the range of (0.7,1).

If the random generated number is less than p_cross (ϕ < p_cross) the crossover occurs. Then the child for the parents will be

$$\mathbf{C = \phi * P_1 + (1 - \phi) * P_2.}$$

Matlab code for Crossover function:

```
%THIS FUNCTION RETURNS THE CHILDREN AFTER CROSSOVER

function C = crossover(P,N)
    pcross = 0.8;
    count = 0;
    F = fitness(P,N);
    C = P;
    prob_distribution = F./sum(F);
       while count < N
           x = rand;
           parents = 
randsample(P.',2,true,prob_distribution.').';
           if x < pcross
               count = count + 1;
               C(count) = x*parents(1) + (1-
x)*parents(2);
           end
       end
end
```

Mutation Operator:

Generally, for mutation of a solution x is done as shown below.

$$x_\mu = x + \beta * x$$

where β can take any value positive or negative. Here we are taking β in the range of (-1,1) and condition that x after mutation must be in the bounds of the decision variable space.

$$x_L \leq x_\mu \leq x_U$$

Also, mutation occurs with probability(p_mutation). Generally, p_mutation value is in the range of (0, 0.2). Let k be a random generated number between 0 and 1. Then β is taken as (2k-1).

If k is less than p_mutation (k<p_mutation) then mutation occurs.

Matlab code for the Mutation function:

```
%THIS FUNCTION RETURNS THE CHILDREN AFTER MUTATION

function M = mutation(P,N,X)
     p_mutation = 0.1;
     M = P;
     for i = (1:N)
         x = rand;
         if x < p_mutation && (2*x)*P(i)>X(1) &&
(2*x)*P(i)<X(2) && f((2*x)*P(i)) > 0
             M(i) = (2*x)*P(i);
         end
     end
end
```

Now we need to write the main function for the algorithm:
Let's take F(x) = $3x^2 - x^3$.

It is as shown below in matlab:

```
% IMPLEMENTATION OF REAL ENCODING SIMPLE GENETIC
ALGORITHM
% XL = lower limit of the decision variable
% XU = Upper limit of the decision variable
% X = [XL XU]

X =[0.5 3];
N = 10;
Population = zeros(N,1);
count = 1;
while count <= N
    x = rand(1,1)*(X(2) - X(1))+X(1);
    if f(x) > 0
        Population(count) = x;
        count = count +1;
    end
end
figure
for i = (1:5000)
    Parents = selection(Population,N);
    Children = crossover(Parents,N);
    Children_M = mutation(Children,N,X);
    Population = Children_M;
end
fplot(@(s) 3*s*s-s*s*s,[X(1) X(2)]);
hold on
xlabel('x')
ylabel('f(x)')
title('Plot of f(x) and the optimized solution')
for j = (1:N)
    plot(Population(j),f(Population(j)),'-o');
end
```

Here we are taking the population size as 10, and no of generations as 5000.

From gradient analysis F(x) = $3x^2 - x^3$ *has an optimum value at*

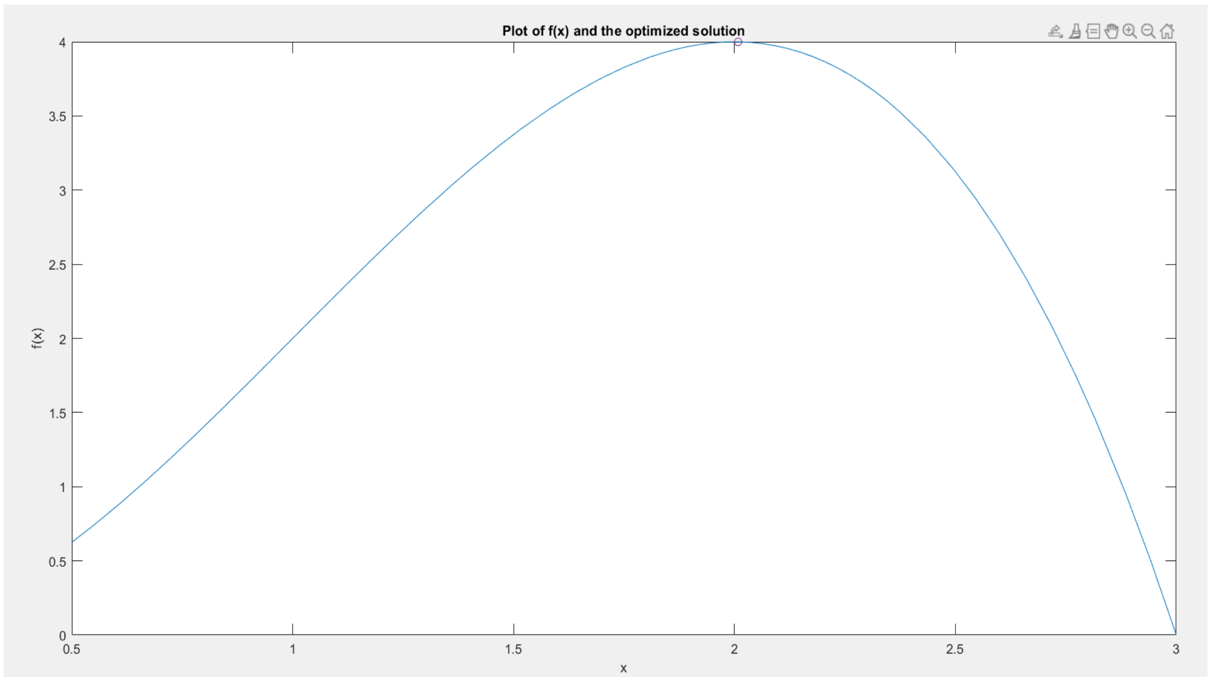
$x = 2$*. Let's see what we get from the designed algorithm.*

After 5000 generations we are getting the population as:

```
2.00820246001510
2.00820246001510
2.00820246001510
2.00820246001510
2.00820246001510
2.00820246001510
2.00820246001510
2.00820246001510
2.00820246001510
2.00820246001510
```

We can clearly see that the population has converged to a single value.

We get the plot as:



We can see that our solution from the algorithm (small violet circle in the above plot) is very close to actual optimum value.

Note: *We need to know that the algorithm will give different results for every time the algorithm runs as the population is random and sometimes pre-mature convergence may occur. But every time the results will be closer to the actual optimum solution.*

Now, let's see for another function ***F(x) =*** $\sqrt{1-x^2}$

In ***the interval (-1,1)***

Again, for initial population size of 10 and after the 5000 generations we get the population as:

1.04138780822739e-237

4.28974464462559e-237

1.34158700533584e-237

1.42825662641643e-237

5.43141357376036e-238

1.11562597661660e-239

2.25957227417907e-237

3.30097355251540e-237

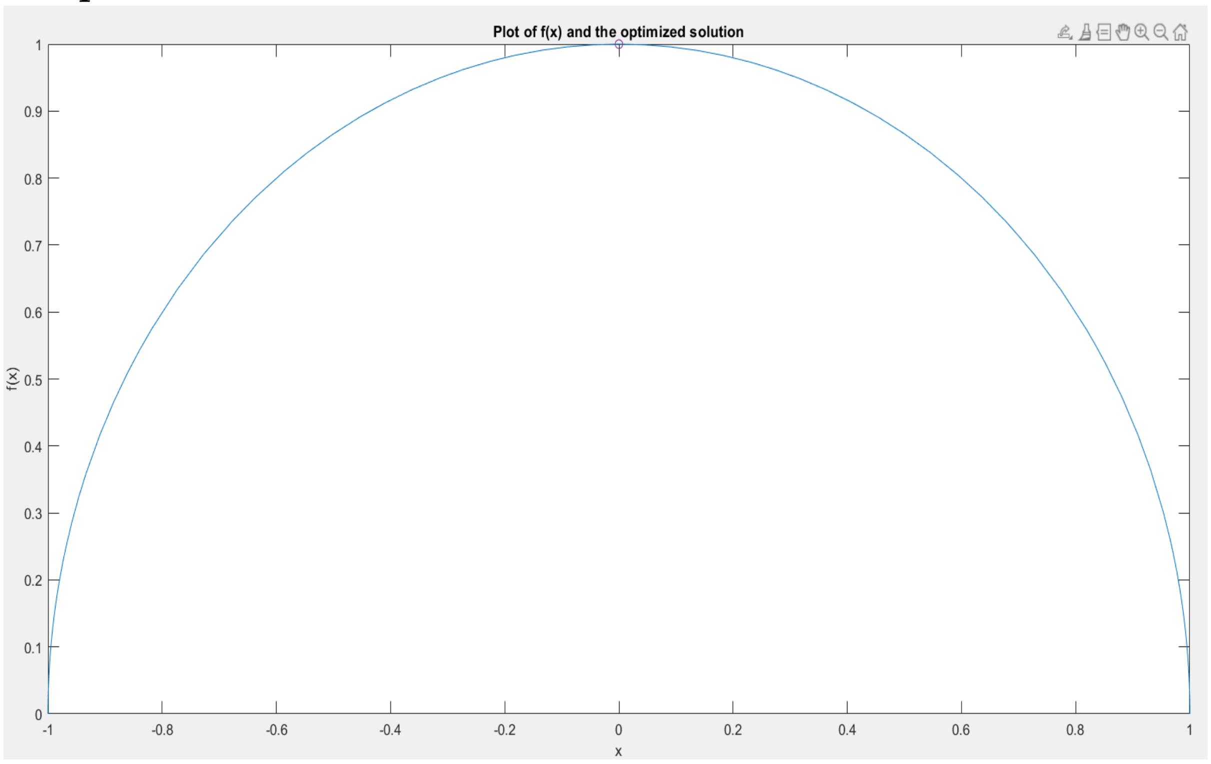
2.70216173989803e-237

3.96581569220574e-237

Actual optimum solution of the given function F(x) = $\sqrt{1-x^2}$ *is at x=0.*

Here e-237 means that they are of the order 10^{-237}*, which is very close to actual optimum solution (0).*

The plot is as shown below:



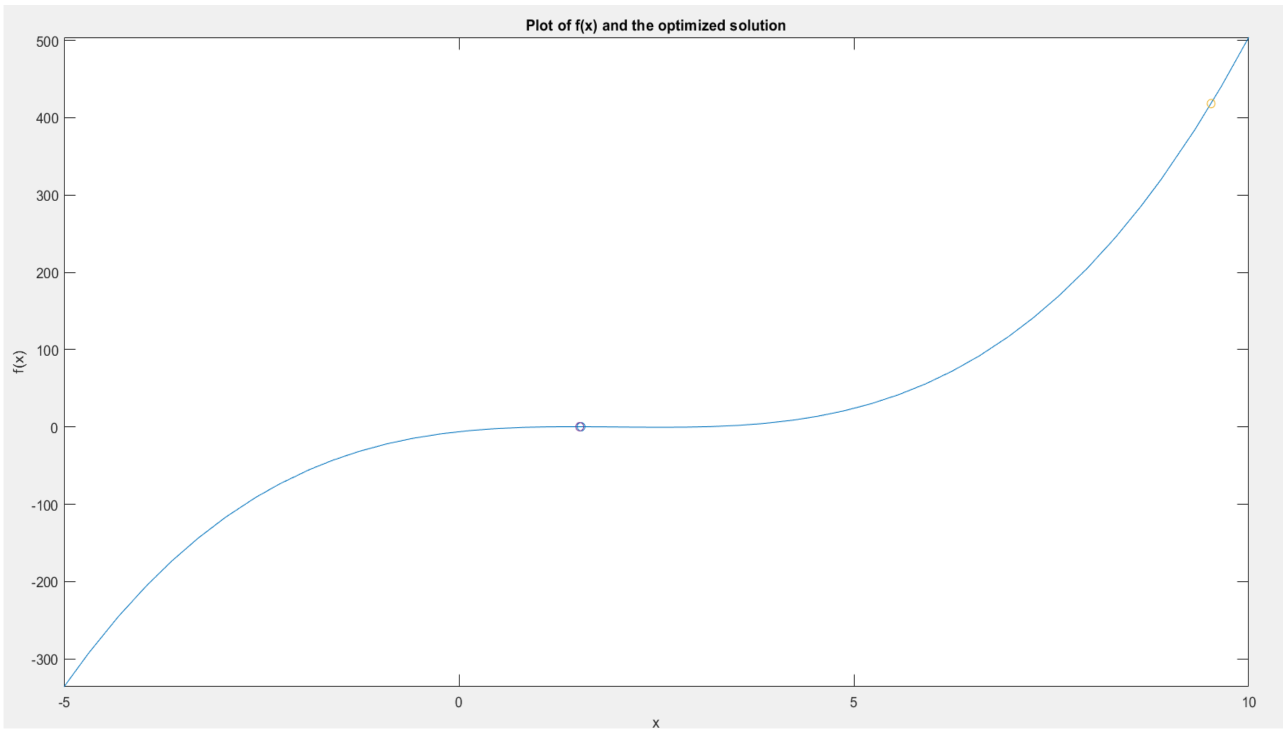
Analysis of role of crossover and mutation:

Let's take function that needs to be optimized be x^3-6x^2+11x-6 in the interval of (-5,10).

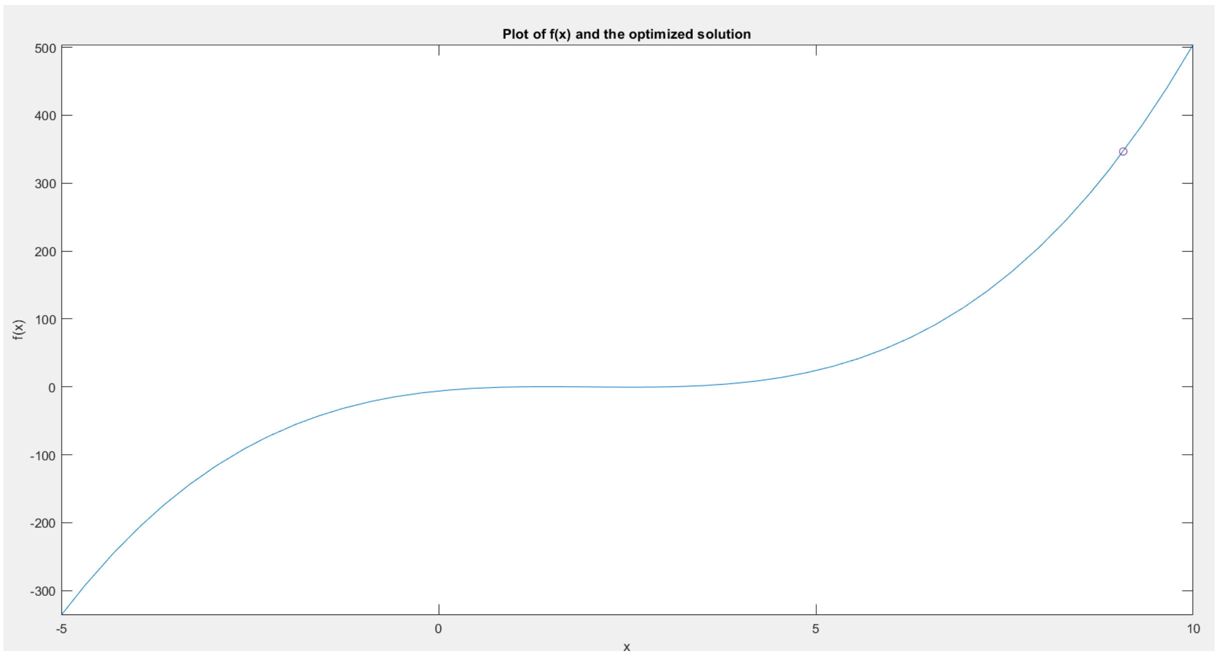
First case:

$p_{cross} = 0.8$ *$p_{mutatation} = 0.1$* *$N = 7000$*

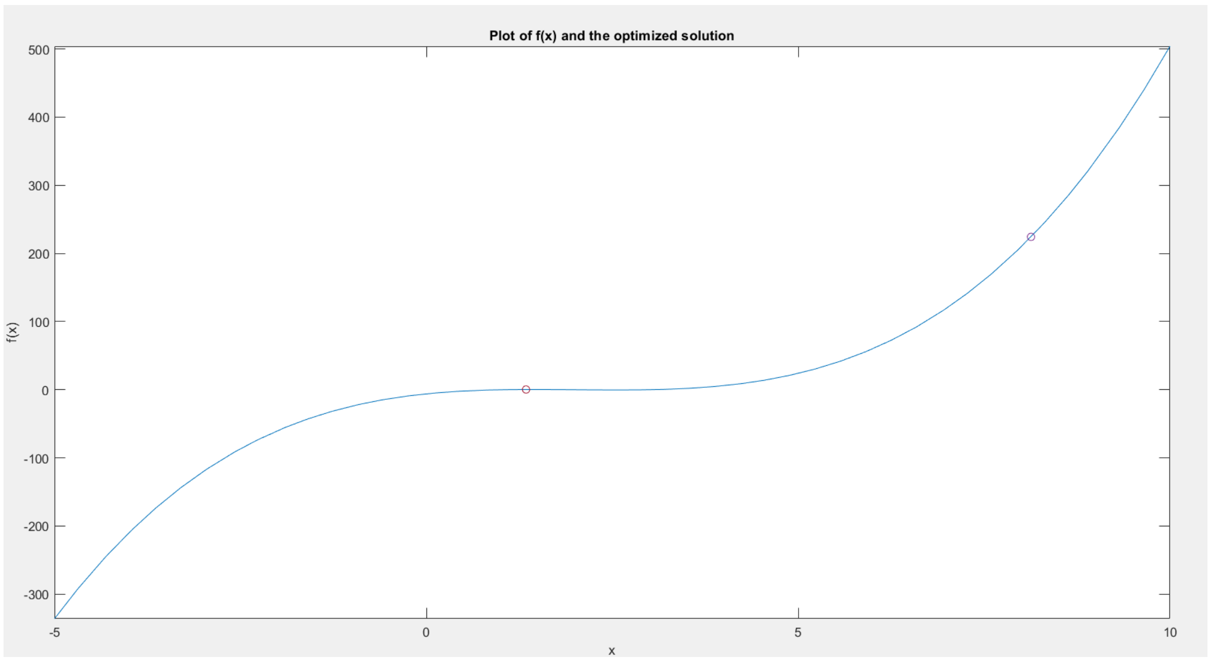
We can clearly see that in two different regions circles are pointed on the graph which are obtained from the algorithm. The blue circle is at the local optimum and yellow circle is at global optimum.



If we increase the number of generations to 11000 keeping the probabilities of crossover and mutation as same:



The solution obtained is near the global optimum which some times may not occur always as shown for N = 10000



By the graphs shown above even for N= 11000 it is showing the correct optimum it may often happen that some solutions are near local optimum.

Therefore, it is not often feasible in optimizing the functions that are having local optimums by using this algorithm.

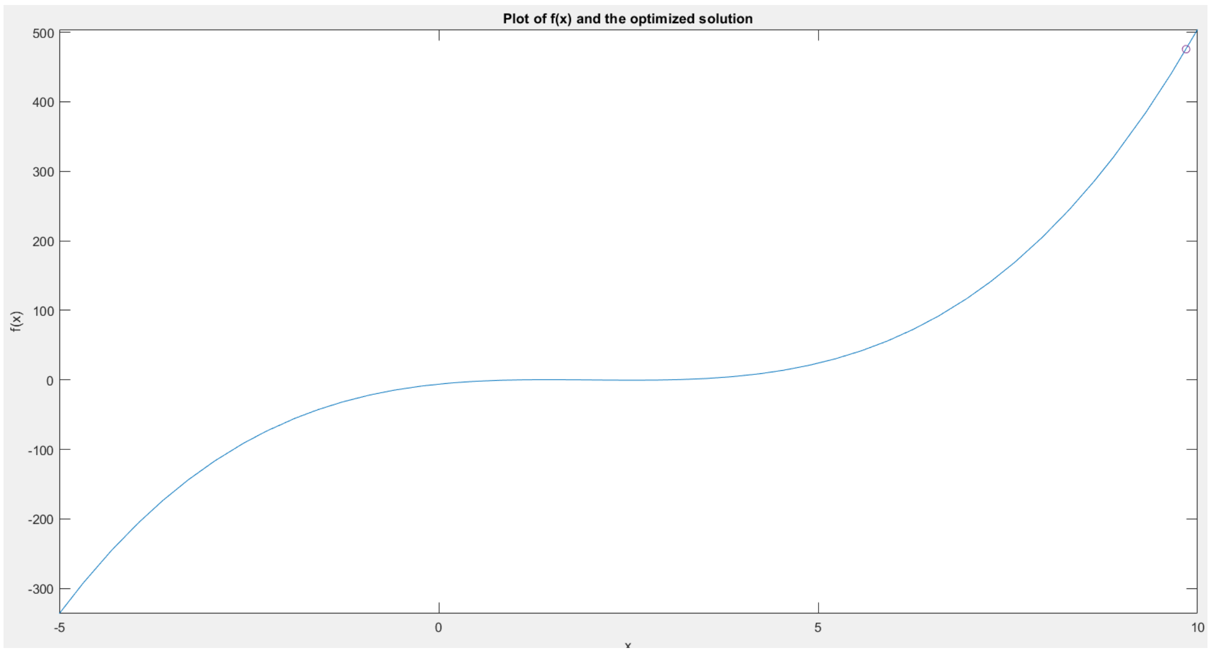
Second case:

Let's increase the probability of crossover to1 keeping probability of mutation as zero

$p_{cross} = 1$ $p_{mutatation} = 0$ $N = 11000$

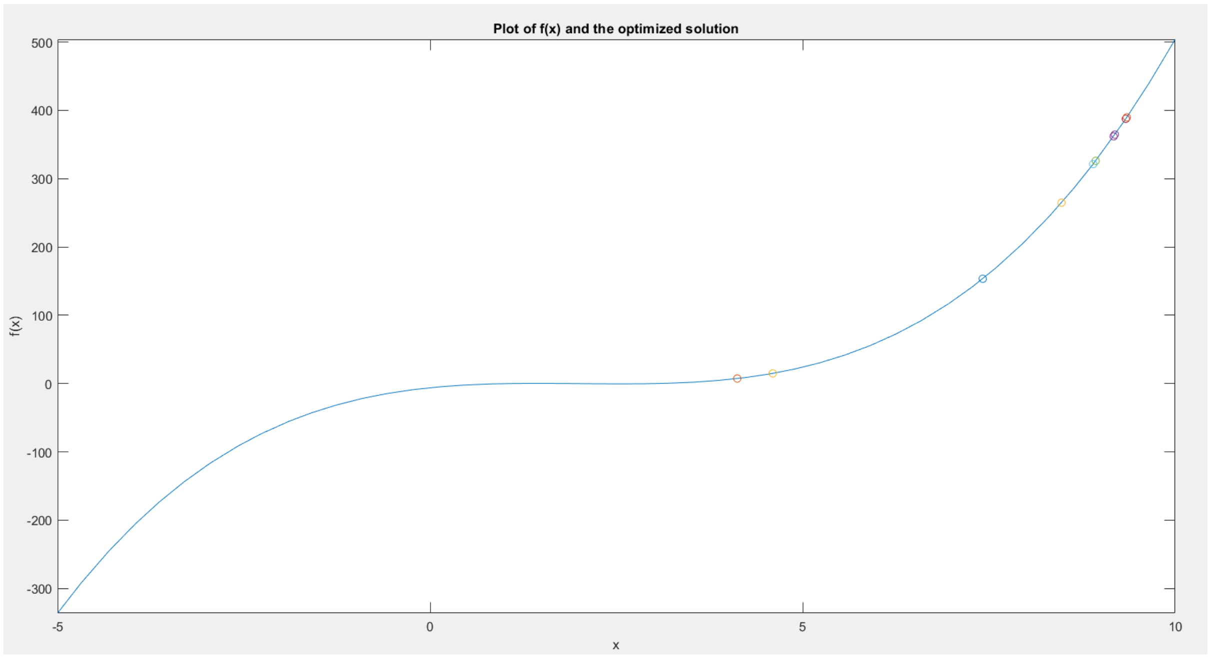
The graph is as shown in the top of the next page.

The solution from the algorithm is very close to the actual optimum (small violet circle on the top right of the graph). From this we can say that even if there is no mutation the solution obtained is very much feasible. But it is essential to have $p_{mutation}$ greater than 0 as it helps when solutions are stagnated at some local optimum. But $p_{mutation}$ should be kept low. In the next case let's see if $p_{mutation}$ is kept high.



Third Case: *$p_{mutation}$ is kept high*

p_{cross} = 0.9 $p_{mutatation}$ = 0.6   N = 11000



We can clearly see that the solutions from the algorithm are not converging. Hence, is $p_{mutatation}$ kept low.

Therefore, the Simple genetic algorithm is not mutation driven algorithm, where p_{cross} should be kept close to 1 and $p_{mutatation}$ should be close to zero.

Now we are taking two-dimensional variable space:

Let's take the function that need to be optimized be

$$F(x,y) = \sqrt{(1-x^2-y^2)}$$

Store this function in the f function as shown. Here f takes an input as vector which contains the variables as x and y.

Function f:

```
% THIS FUNCTION IS TO BE OPTIMIZED
% BY REAL ENCODING SIMPLE GENETIC ALGORITHM

function z = f(x)
    z = sqrt(1-x(1)*x(1)-x(2)*x(2));
end
```

Now define the fitness function as shown below. This function returns a vector containing all the fitness values of the given input Population.

Fitness Function:

```
%THIS FUNCTION RETURNS FITNESS VALUES OF THE POPULATION

function y = fitness(x,N)
    y = zeros(1,N);
    for i = (1:N)
        y(i) = f(x(:,i));
    end
end
```

Next, we need to define the operators of the algorithm:

Selection operator (Stochastic Remainder Selection)

Selection function:

```
% THIS FUNCTION RETURNS THE PARENTS FROM THE GIVEN
POPULATION
% USING STOCHASTIC REMAINDER SELECTION

function S = selection(P,N)
   F = fitness(P,N);
   S = floor((F./sum(F)).*N);
   parents = P;
   count = 1;
   for i =(1:N)
       if S(i) > 0
           for j = (count:count+S(i)-1)
               parents(:,j)= P(:,i);
           end
           count = count + S(i);
       end
   end
   if sum(S) < N
       prob_distribution = (F./sum(F)).*N -
floor((F./sum(F)).*N);
       x = 1:N;
       k = randsample(x,N-
count+1,true,prob_distribution);
       for i = (count:N)
           parents(:,i) = P(:,k(i-(count-1)));
       end
   end
   S = parents;
end
```

Crossover Operator:

It is same concept as the operator used in the single-dimensional variable space.

Crossover function:

```
%THIS FUNCTION RETURNS THE CHILDREN AFTER CROSSOVER

function C = crossover(P,N)
    pcross = 0.8;
    count = 0;
    F = fitness(P,N);
    C = P;
    prob_distribution = F./sum(F);
       while count < N
           x = rand;
           k = randsample(1:N,2,true,prob_distribution);
           if x < pcross
               count = count + 1;
               C(:,count) = x*P(:,k(1)) + (1-
x)*P(:,k(2));
           end
       end
end
```

Mutation Operator:

It is same concept as the operator used in the single-dimensional variable space. There is one more function that need to be used. That is boundary check function which ensures that the generated children are bound within the bounds of the decision variable space.

Boundary Check function:

```
function y = bound_check(a,b)
      y = a(1) >= b(1,1) && a(1) <= b(1,2) && a(2) >=
b(2,1) && a(2) <= b(2,2);
end
```

Mutation function:

```
%THIS FUNCTION RETURNS THE CHILDREN AFTER MUTATION

function M = mutation(P,N,X)
     p_mutation = 0.1;
     M = P;
     for i = (1:N)
         x = rand;
         if x < p_mutation && bound_check((2*x)*P(:,i),X)
&& f((2*x)*P(:,i)) > 0
             M(:,i) = (2*x)*P(:,i);
         end
     end
end
```

Now we need to write the main function for the algorithm:
Let's take F(x,y) = √(1-x^2-y^2).

It is as shown below in matlab:

```
% IMPLEMENTATION OF REAL ENCODING SIMPLE GENETIC
ALGORITHM
% n = NUMBER OF DIMENSIONS IN DESICION VARIABLE SPACE
% B CONTAINS THE BOUNDS OF THE DECISION VARIABLE SPACE

n = 3;
B =[-1 1;-1 1];
N = 10;
%INITIALIZATION OF THE RANDOM POPULATION
Population = zeros(2,N);
count = 1;
while count <= N
    x = rand(2,1).*(B(:,2) - B(:,1))+B(:,1);
    if (f(x)>0)
        Population(:,count) = x;
        count = count +1;
    end
end
%SIMPLE SGA RUNS FOR N GENERATIONS
for i = (1:5000)
    Parents = selection(Population,N);
    Children = crossover(Parents,N);
    Children_M = mutation(Children,N,B);
    Population = Children_M;
end
%GRAPHICAL VISUALIZATION OF OPTIMUM SOLUTION
[X,Y] = meshgrid(-1:0.005:1);
Z = real(sqrt(1-X.^2 - Y.^2));
s = surf(X,Y,Z,'FaceAlpha',1);
s.EdgeColor ='none';
for i = (1:N)
    hold on

plot3(Population(1,i),Population(2,i),f(Population(:,i)),
'p-k');
end
```

Here we are taking the population size as 10, and no of generations as 5000.

From the gradient analysis of the given we get that the optimum value occurs at $x = 0$ and $y = 0$.

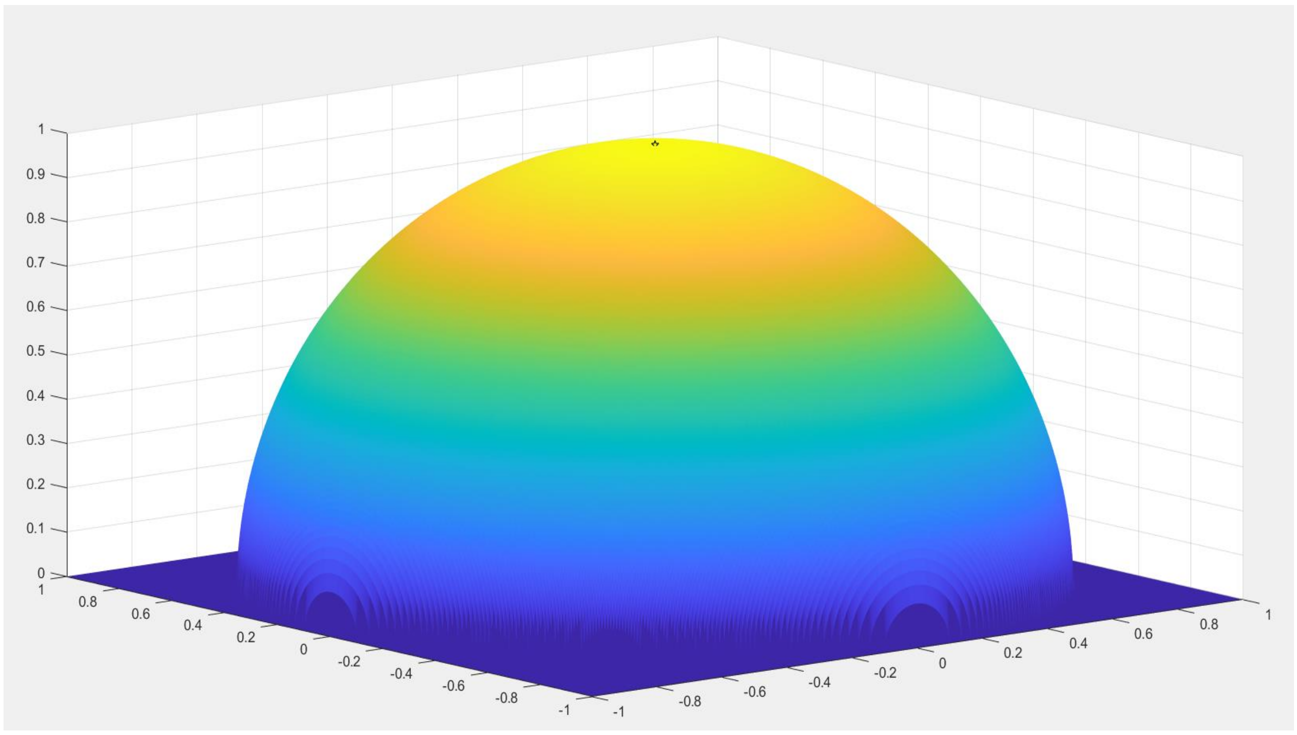
Final population generated after 5000 generations is shown below:

x	**y**
-5.435161801937 98e-226	-2.10383259724342e-225
-1.98266261839476e-225	-7.67445459384041e-225
-2.94987018436505e-226	-1.14183041419131e-225
-4.76871231558232e-226	-1.84586453577536e-225
-1.97660031536724e-225	-7.65098874095795e-225
-1.35334672759797e-226	-5.23851002904477e-226
-2.01008675570527e-225	-7.78060744839676e-225
-1.80174100685313e-225	-6.97414649303764e-225
-3.28466762751048e-225	-1.27142320278354e-224
-7.74580704926353e-226	-2.99823297926254e-225

Here e-226 means that they are of the order 10^{-226}, which is very close to actual optimum solution (0,0).

The final population is converged to a single point (0,0) after 5000 generations.

The Plot of the optimal solution and the function $F(x,y) = \sqrt{(1-x^2-y^2)}$ is shown below:



There is a star on the top of the yellow region which is our optimal solution generated by the algorithm.

We take another example for optimization.

$F(x,y) = x^2 + y^2$

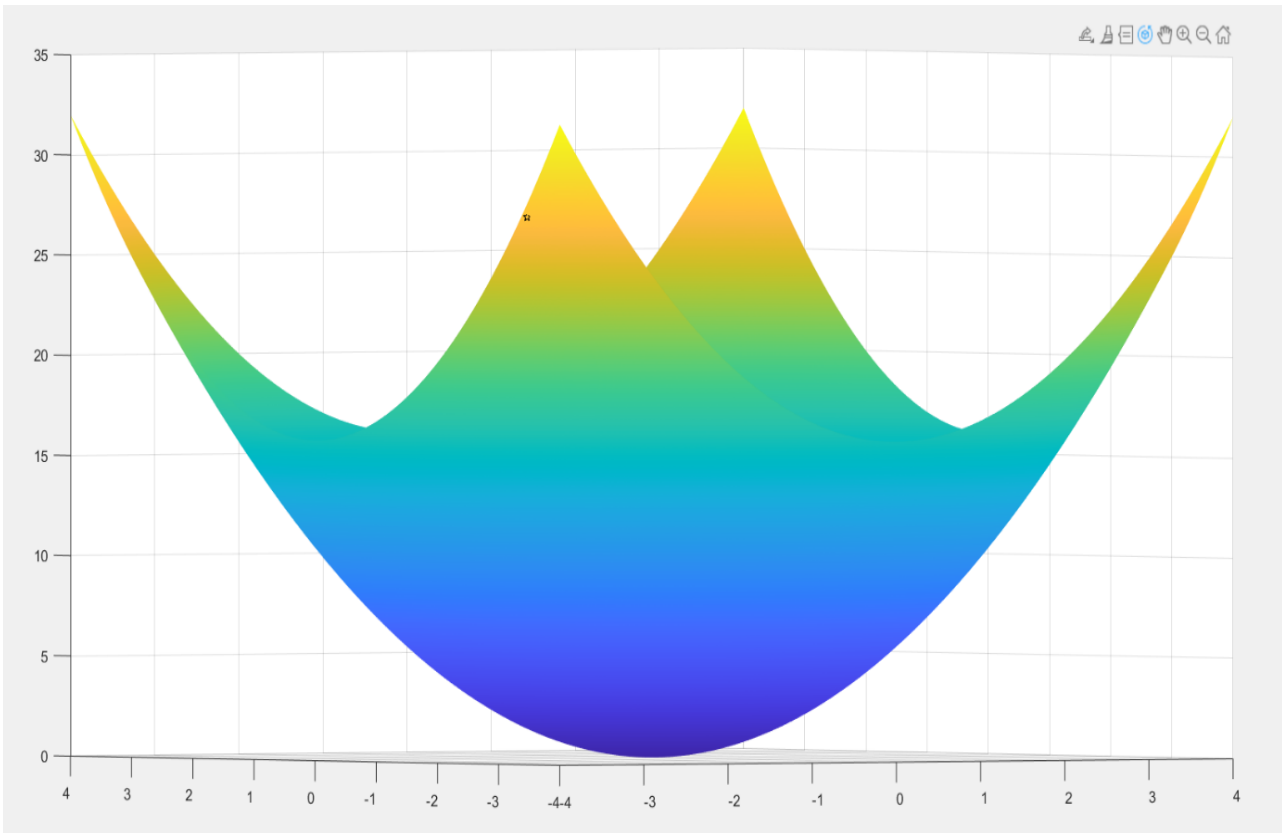
Interval bounds:

x : [-4 4]

y : [-4 4]

It is of cone shape. It doesn't have single optimum value in the given bounds. It has 4 optimum solutions in the intervals of the variable space given. Let's see what is the solution and the plot of the solution given by the algorithm.

Plot of the optimum solution given by algorithm and the given function:



We can clearly see that there is more than one optimum solution i.e. there are 4 optimums. There is a small black star on one of the 4 yellow regions which is our solution from the algorithm. The solution is close to optimum but not very close to actual optimum.

Thus, we can say that the solution from the solution is not the optimum. This is due to pre-mature convergence of the solution and there are multiple optimum in the given intervals of the variable space.

Limitations:

There are some limitations to the above algorithm:

1) *Let the function that needs to be optimized be F(x).*

Then F(x) must always be positive in the given bounds of the variable space.

Reason:

In the selection operator we need to select the population based on the process of stochastic remainder process which requires us to use the probabilities. Probability should always be positive. We are defining the probability of selection of a parent S by

$$P(S) = \frac{fitness(S)}{sum\ of\ fitnesses\ of\ all\ parents} - \left[\frac{fitness(S)}{sum\ of\ fitnesses\ of\ all\ parents}\right]$$

Where [ ] indicates G.I.F (Greatest Integer Function).

We are directly using fitness as the function that need to be optimized. As probability is always positive F(x) also must be positive in the bounds of the variable space.

2) *In order to run the designed algorithm one should guess the number of generations required for the convergence of the solution.*

The user who runs the algorithm must guess the number of generations or should have prior experience with the algorithm to know the number of generations required for convergence of the optimal solution.

3) *For more than single dimensional variable space or for multi-dimensional variable space, some times the optimal solution obtained by the algorithm doesn't converge exactly to the actual optimum of the function.*
4) *This designed Simple genetic algorithm cannot be used if we want to get all the local optimal solution.*

This algorithm in general converge to the global optimum. Some times it may happen that we get local optimum solution which is due to Pre-mature convergence.