Quantum Genetic Algorithm

Anirudh K,206002009

Benchmark function to be optimized

Spherical function :

```
function z = MinSphere(x)

    z = sum(x.^2);

end
```

Functions Used:

- Main.m – To define problem and params, call QGA fro 100 iterations
- QGA.m – Performs Conversion of Quantum Population to Classical Population , performs crossover and mutation on Chromosomes. Sort population and store best 2. Reconvert these two as Quantum Chromosomes for next iteration

Code:

Main.c

```
clc;
clear;
close all;
%% Definitions
%problem
problem.CostFunction = @(x) MinSphere(x);
problem.nVar = 3; % No. of genes in a chromosome

% QGA params
params.MaxIt = 100;
params.nPop = 6;
params.qPop = 2;
params.nChromo = 3;
params.nMin = -5;
params.nMax = 5;

% Initializing first two quantum chromosomes
empty_individual.Chromosome = []; % chromosome
empty_individual.BinVal = [];
empty_individual.Cost = [];
qpop = repmat(empty_individual,params.qPop,1);
for i=1:params.qPop
    for j = 1:params.nChromo
                val = params.nMin + (0-(params.nMin))*rand(1,1);
                g = CreateGene(val,val+params.nMax,params.nMin,params.nMax);
                temp(j) = g;
     end
```

Quantum Genetic Algorithm

Anirudh K,206002009

```
            qpop(i).Chromosome = temp;
end

% Now we call the algorithm that perfroms interference , generates
% classical population which is used to obtain two best solutions that are
% again used as input
bestsol = repmat(empty_individual,1,1);
bestsol.Cost = 1000;
for i=1:params.MaxIt
    out = QGA(problem,params,qpop,empty_individual);
    if out.qpop(1).Cost < bestsol.Cost
        qpop = out.qpop;
        bestsol.Cost = qpop(1).Cost;
        bestsol.Chromosome = qpop(1).Chromosome;
    end
    disp(['Iteration' num2str(i) ' :Best Cost = ' num2str(qpop(1).Cost)]);
end

pdf1 = bestsol.Chromosome(1).PDF;
pdf2 = bestsol.Chromosome(2).PDF;
pdf3 = bestsol.Chromosome(3).PDF;
x = -5:.01:5;
figure;
subplot(3,1,1);
plot(x,pdf1,'r','LineWidth',2);
subplot(3,1,2);
plot(x,pdf2,'r','LineWidth',2);
subplot(3,1,3);
plot(x,pdf3,'r','LineWidth',2);
```

QGA.c

```
function out = QGA(problem,params,qpop,empty_individual)

x = -5:.01:5;

%setting values
CostFunction = problem.CostFunction;
nPop = params.nPop;
qPop = params.qPop;
nMax = params.nMax;
nMin = params.nMin;
nChromo = params.nChromo;

%Result of intereference
int = Intereference(qpop);

%Creating classical population - 1
cpop = repmat(empty_individual,nPop,1);
for i=1:2
        temp = CreatePop(int);
        cpop(i).Chromosome = temp.Chromosome;
end

%creating more members using crossover
```

Quantum Genetic Algorithm

Anirudh K,206002009

```
[cpop(3).Chromosome,cpop(4).Chromosome] =
ArithmeticCrossover(cpop(1).Chromosome,cpop(2).Chromosome);
[cpop(5).Chromosome,cpop(6).Chromosome] =
Mutate(cpop(3).Chromosome,cpop(4).Chromosome);

for i=1:nPop
    cpop(i).Cost = CostFunction(cpop(i).Chromosome);
end
cpop = SortPopulation(cpop);
disp("Cpop local best costs : ");
disp(cpop(1).Cost);
disp(cpop(2).Cost);

qpop = repmat(empty_individual,qPop,1);
for i=1:2
        temp = CreateQPop(cpop(i),nMax,nMin,nChromo);
        qpop(i).Chromosome = temp.Chromosome;
        qpop(i).Cost = cpop(i).Cost;

end

out.qpop = qpop;

end
```

CreateQpop

```
function qpop = CreateQPop(cpop,nMax,nMin,nChromo)

    for i=1:nChromo
        val = cpop.Chromosome(i);
        if val<0
            g = CreateGene(val,val+nMax,nMin,nMax);
        else
            g = CreateGene(val-nMax,val,nMin,nMax);
        end
        qpop.Chromosome(i) = g;
    end
end
```

CreateGene

```
function gene = CreateGene(a,b,nMin,nMax)
    %function to create a single gene within a given range

    pd = makedist('Uniform','lower',a,'upper',b);

    x = nMin:.01:nMax;
    PDF = pdf(pd,x);
    CDF = cdf(pd,x);
    %Store values
    gene.lower = pd.Lower;
    gene.upper = pd.Upper;
    gene.PDF = PDF;
    gene.CDF = CDF;
```

Quantum Genetic Algorithm

Anirudh K,206002009

```
end
```

CreatePop [for classical population]

```
function cpop = CreatePop(int)

    index = zeros(3);
    for i = 1:3
        index(i) = (1-0).*rand(1,1) + 0;
    end
    x = linspace(-5, 5, 1000);

    tol = 0.001;
    cg1 = find(abs(int.cdf1 - index(1))<tol);
    cg2 = find(abs(int.cdf2 - index(2))<tol);
    cg3 = find(abs(int.cdf3 - index(3))<tol);

    cpop.Chromosome = [x(cg1(1)),x(cg2(1)),x(cg3(1))];


end
```

Interference function

```
function int = Intereference(qpop)

x = -5:.01:5;

int.pdf1 = (qpop(1).Chromosome(1).PDF+qpop(2).Chromosome(1).PDF)/2;
int.pdf2 = (qpop(1).Chromosome(2).PDF+qpop(2).Chromosome(2).PDF)/2;
int.pdf3 = (qpop(1).Chromosome(3).PDF+qpop(2).Chromosome(3).PDF)/2;
int.cdf1 = (qpop(1).Chromosome(1).CDF+qpop(2).Chromosome(1).CDF)/2;
int.cdf2 = (qpop(1).Chromosome(2).CDF+qpop(2).Chromosome(2).CDF)/2;
int.cdf3 = (qpop(1).Chromosome(3).CDF+qpop(2).Chromosome(3).CDF)/2;
end
```

Arithmetic Crossover

```
function [y1,y2] = ArithmeticCrossover(x1,x2)

  r = rand(1,1);
  y1 = r*x1 + (1-r)*x2;
  y2 = (1-r)*x1 + r*x2;

end
```

Mutate

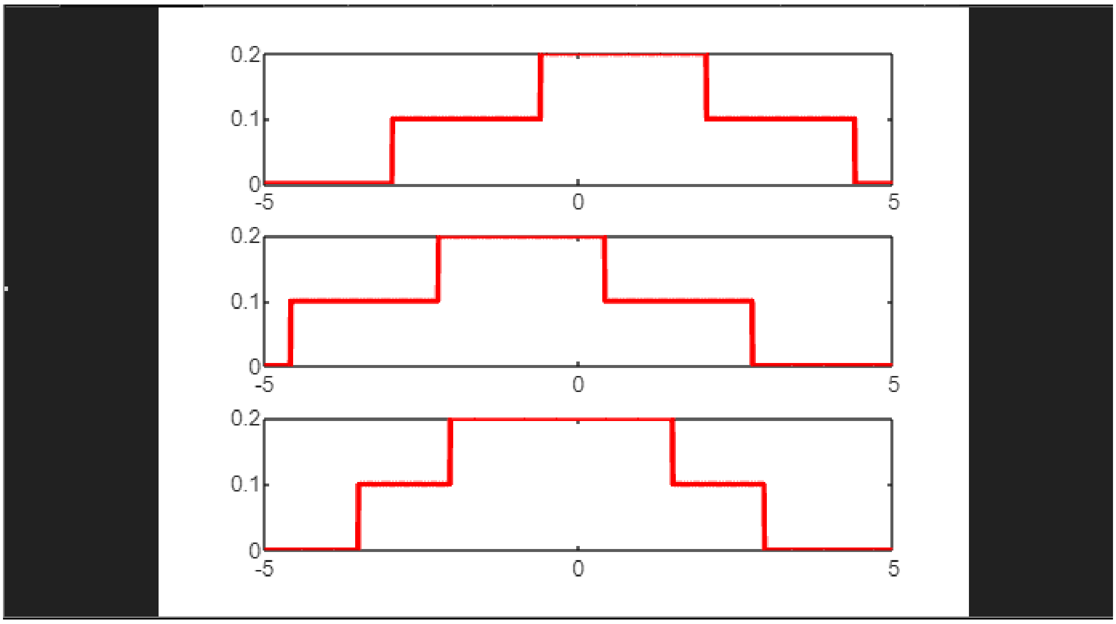
```
function [y1,y2] = Mutate(x1,x2)
```

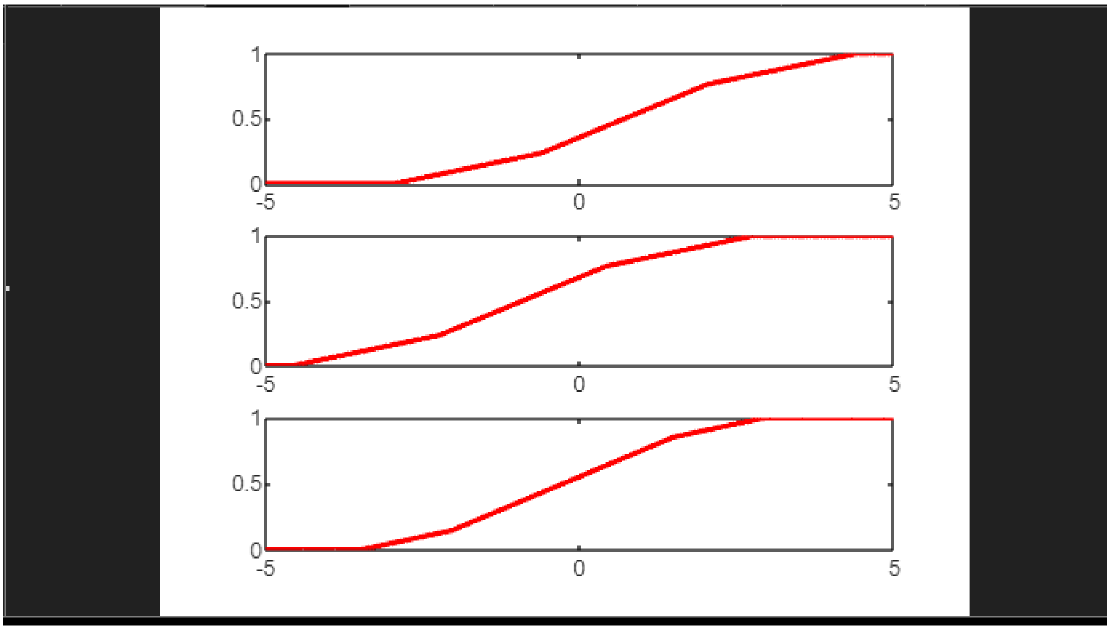
Quantum Genetic Algorithm

Anirudh K,206002009

```
  pd = makedist('Normal','mu',0,'sigma',0.05);
  g1 = random(pd,1,3);
  g2 = random(pd,1,3);
  y1 = x1+g1;
  y2 = x2+g2;
end
```

Interference output [pdf and cdf]

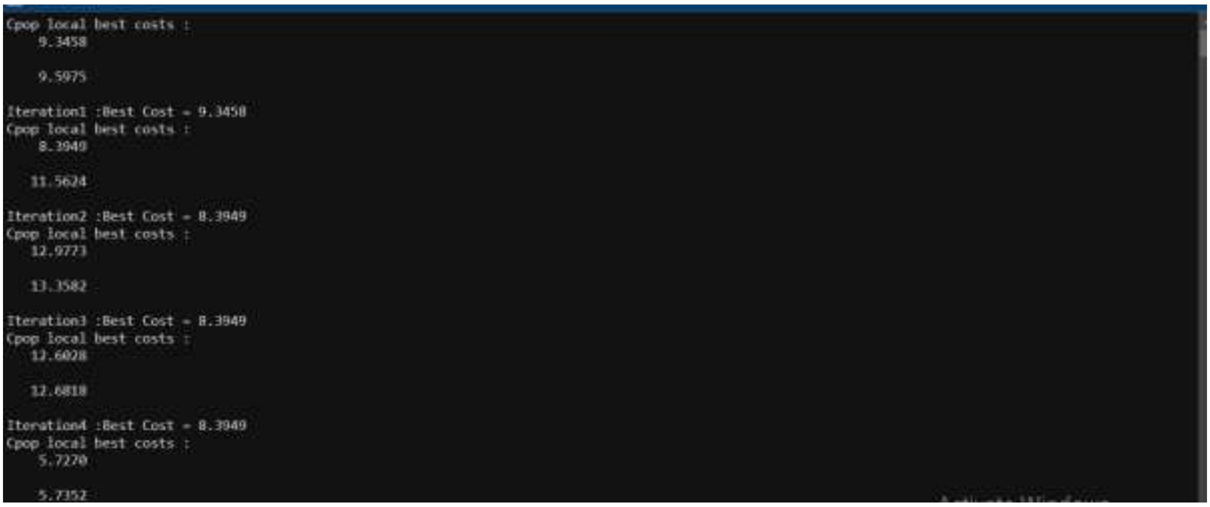


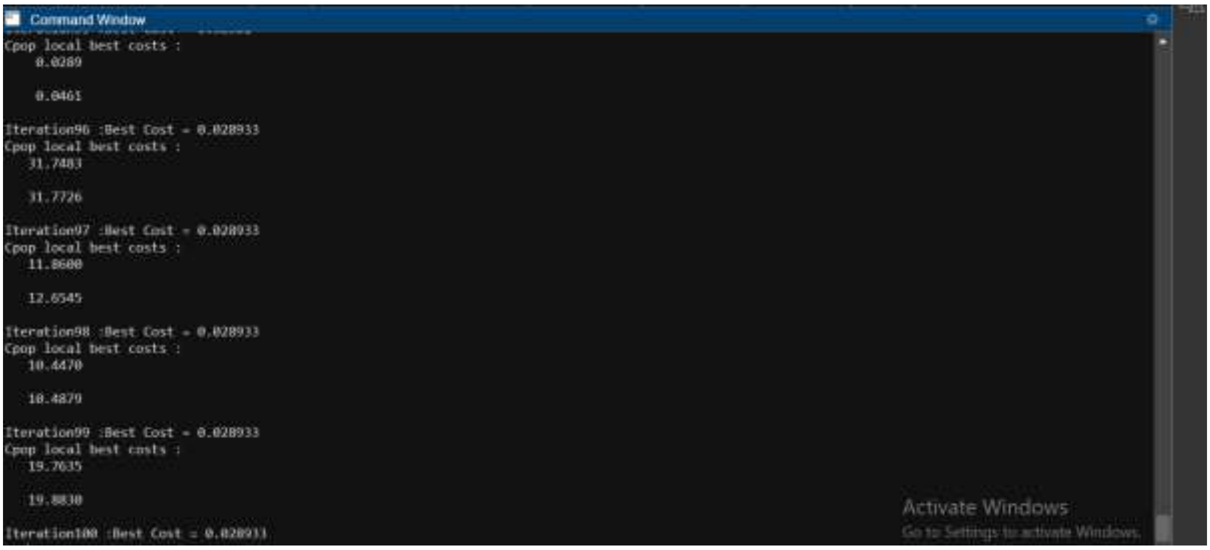


Quantum Genetic Algorithm

Anirudh K,206002009

Terminal output



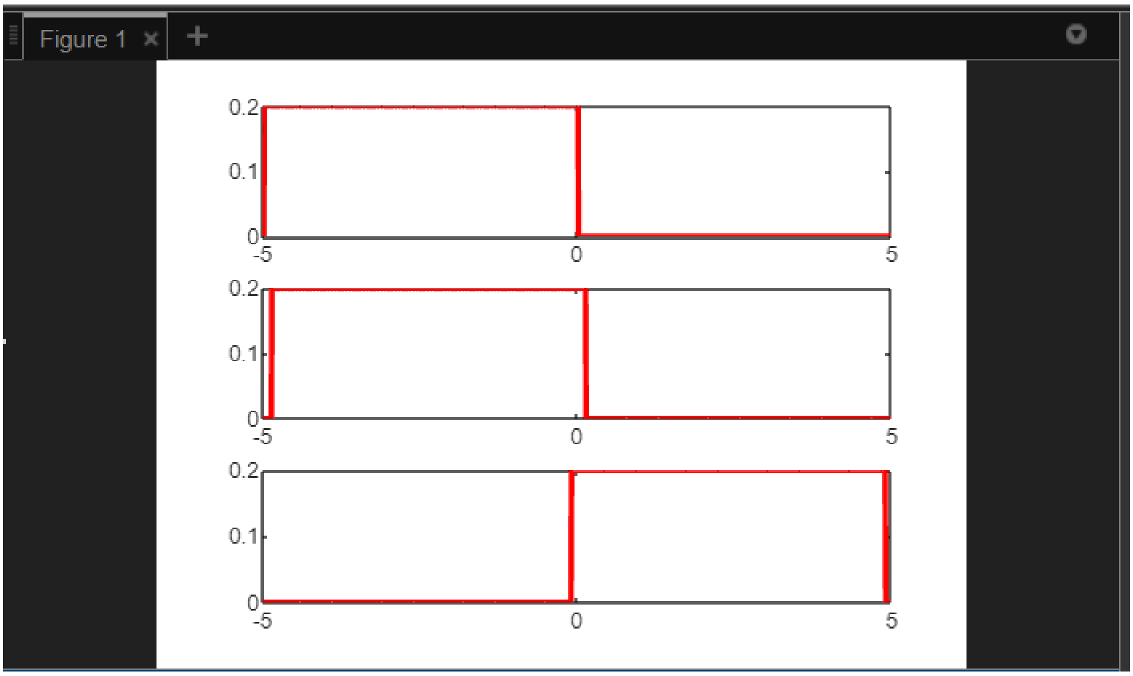


Here we get the solution 0.028933 which is very close to the actual solution of 0

Quantum Genetic Algorithm

Anirudh K,206002009

Final Chromosome



Result

Thus , quantum genetic algorithm is implement to solve the minimization problem of spherical function.