EE 511 Simulation Methods for Stochastic Systems
Project #5 (Chaitra Suresh -7434709345)

[MCMC for Sampling]

The random variable X has a mixture distribution: 60% in a Beta(1,8) distribution and 40% in a Beta(9,1) distribution.

Goal:

i. Implement a Metropolis-Hastings algorithm to generate samples from this distribution.

ii. Run the algorithm multiple times from different initial points. Plot sample paths for the algorithm. Tell if/when the algorithm converges to its equilibrium distribution?
Plot sample paths for the algorithm using different proposal pdfs. Comment on the effect of low-variance vs high-variance proposal pdfs on the behaviour of your algorithm.

Algorithm:

[Metropolis-Hastings] Algorithm:
To sample $\{Xt\}_{t=0}$ from an ergodic markov chain with right target distribution probability density function f_x. Uniform random generator for y axis. Conditional Proposal for x axis which is symmetric.

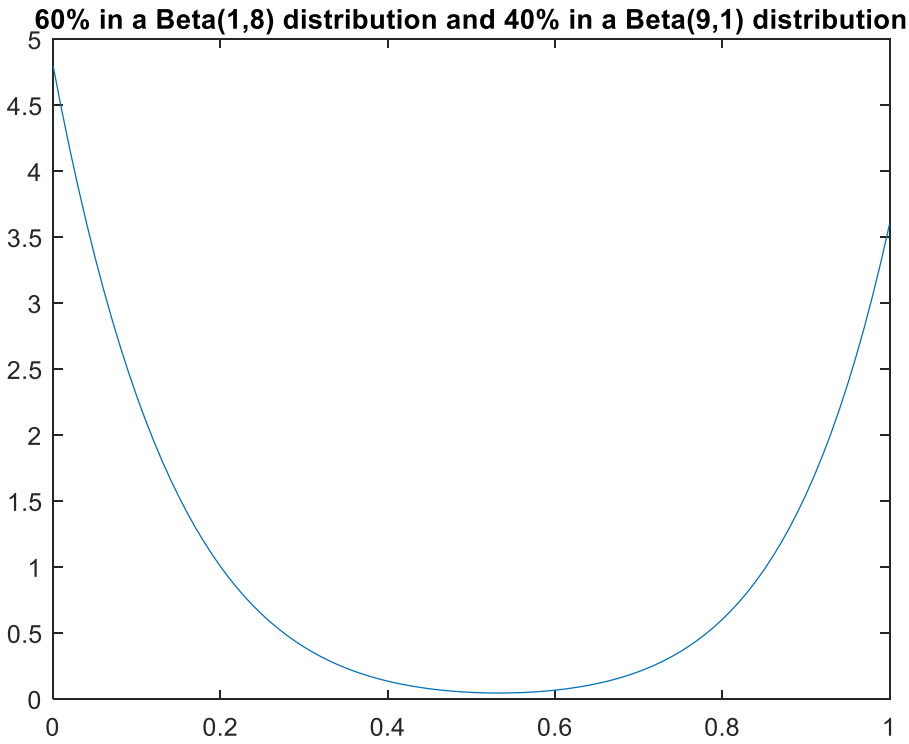
Step 1: Pick any admissible initial sample x0 : $f_x(x_0) > 0$
Step 2: Sample a candidate from $q(y|x_t)$: $y \sim q(y|x_t)$
Step 3: Calculate the Hastings ratio

$$\alpha(x_t,y) = \min\left[1, \frac{f_x(y)}{f_x(x_t)}\right]$$

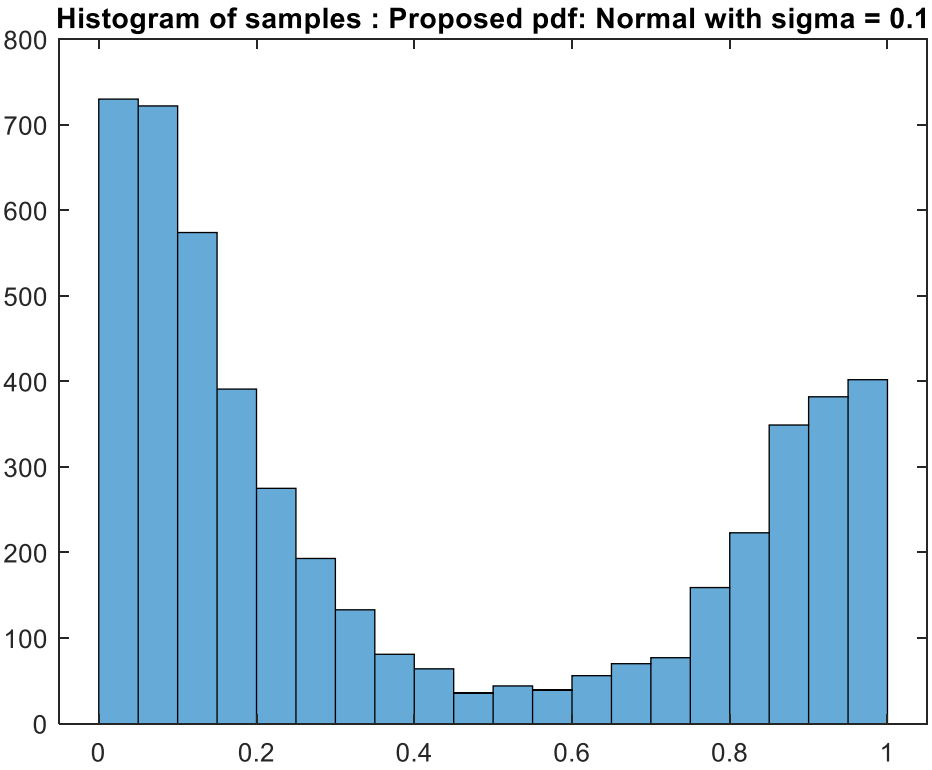
Step 4: Sample Uniformly U~ U[0,1]
Step 5: Accept $x_{t+1} \leftarrow$ y if U<= $\alpha(x_t,y)$ else reject y
Step 6: Go to step (2)

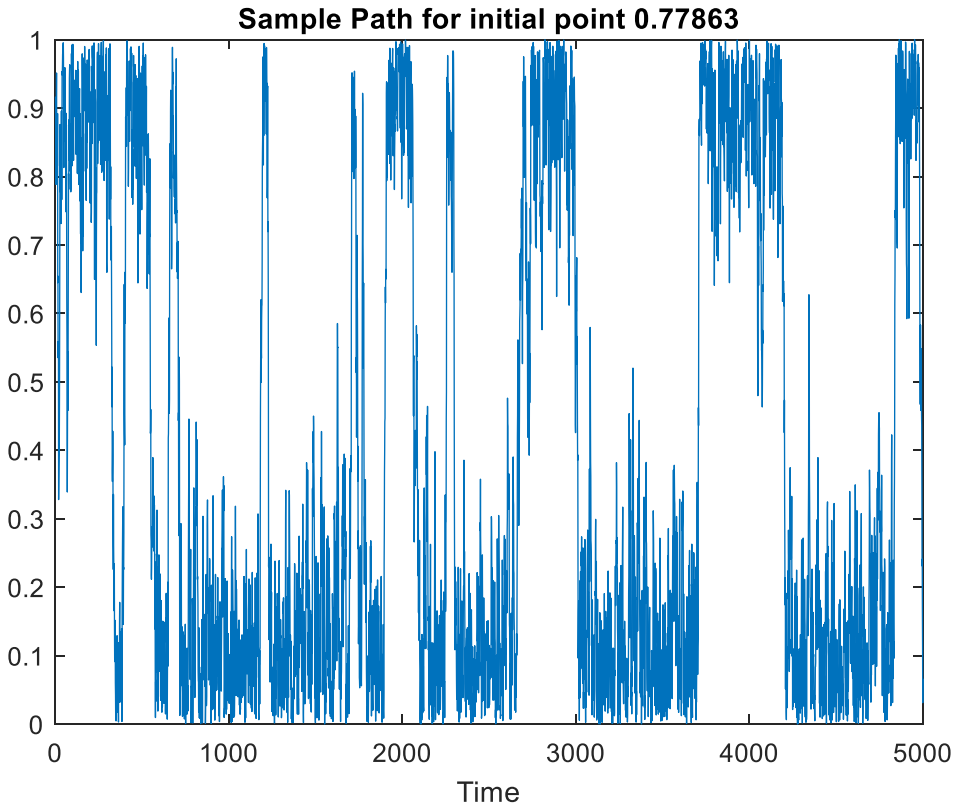
Result:

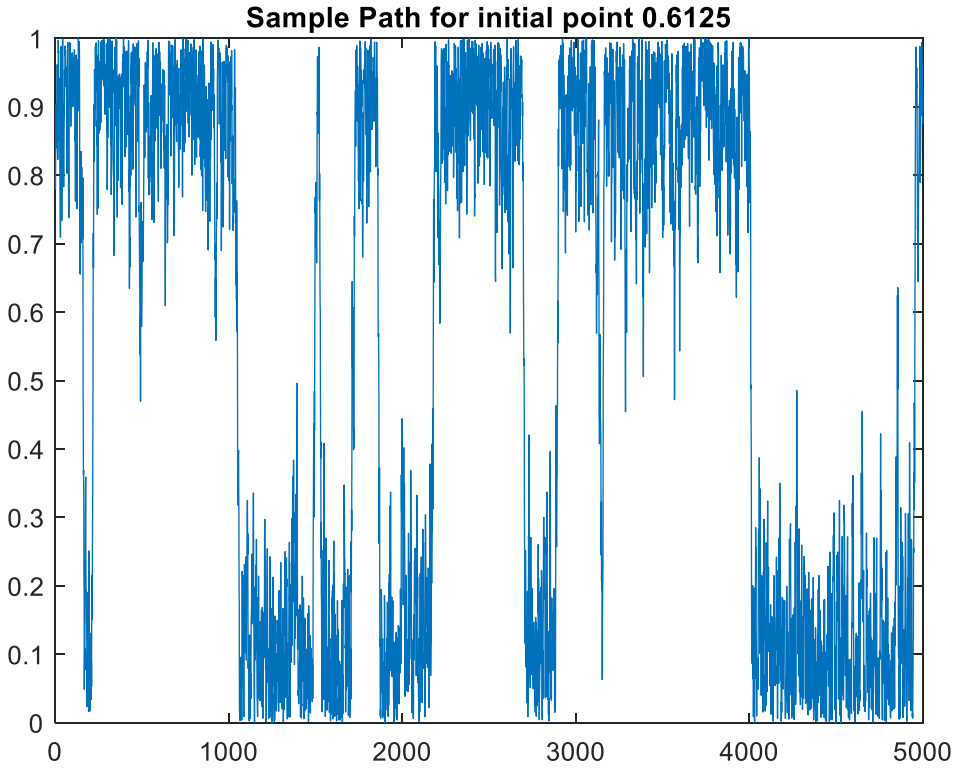
Random Variable X

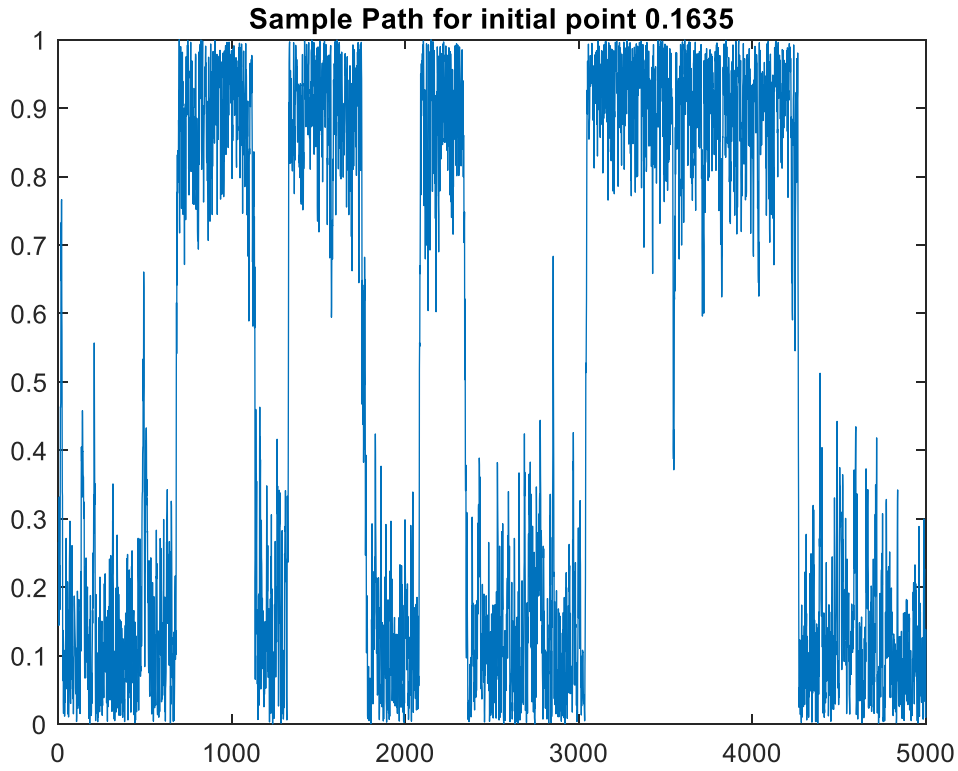


i) Different initial points

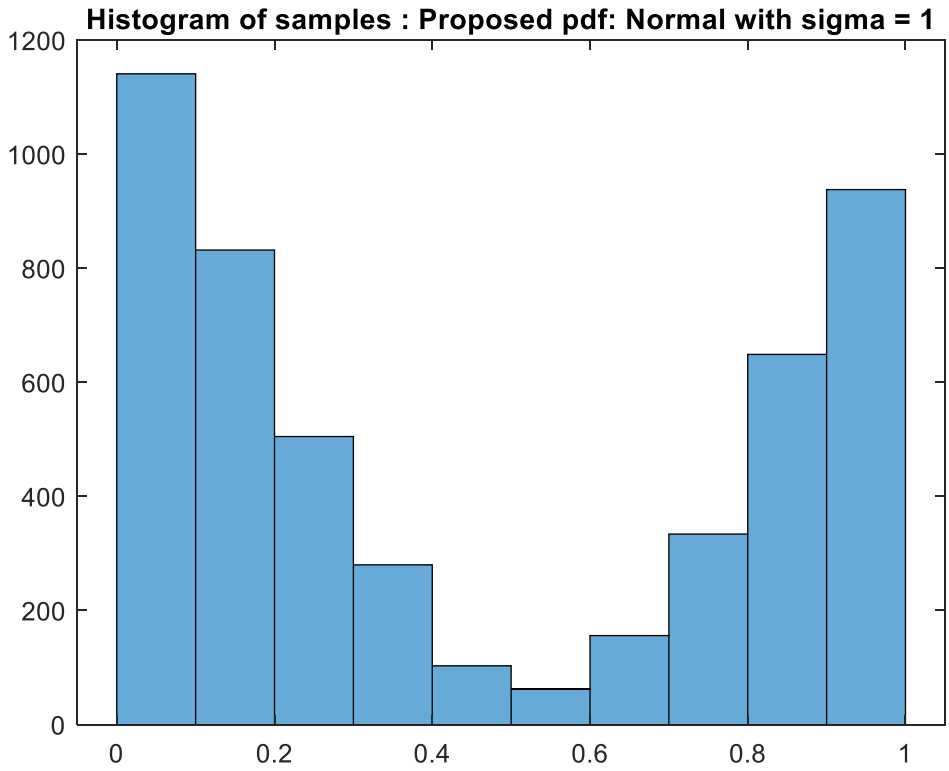


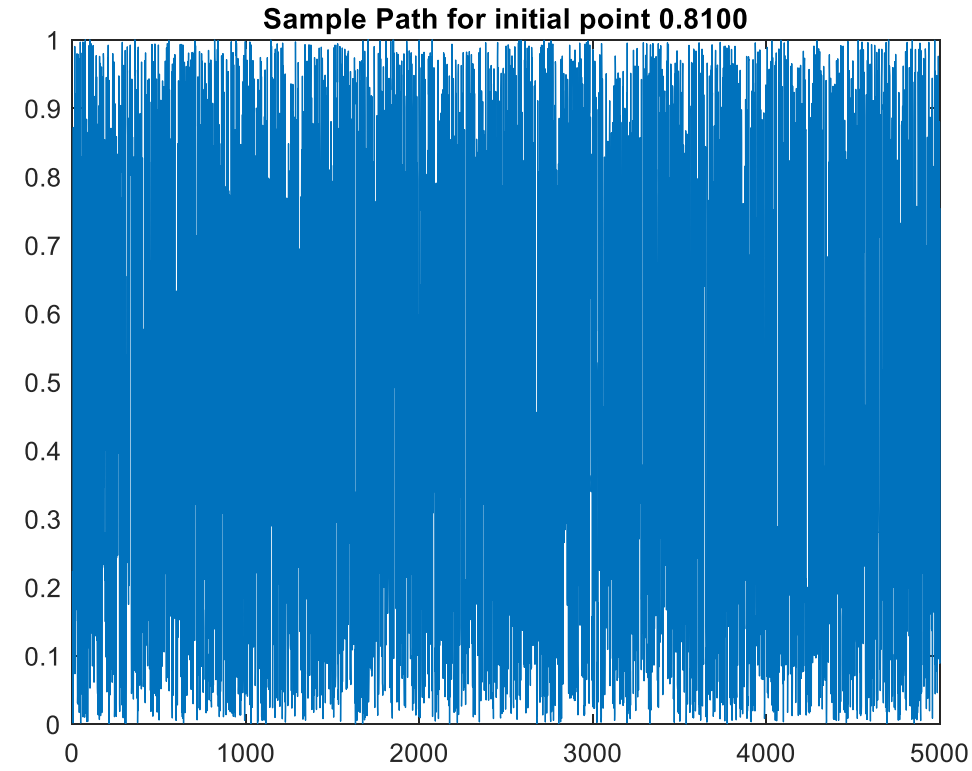

Sample Path for initial point 0.77863
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
1000
2000
3000
4000
5000
Time

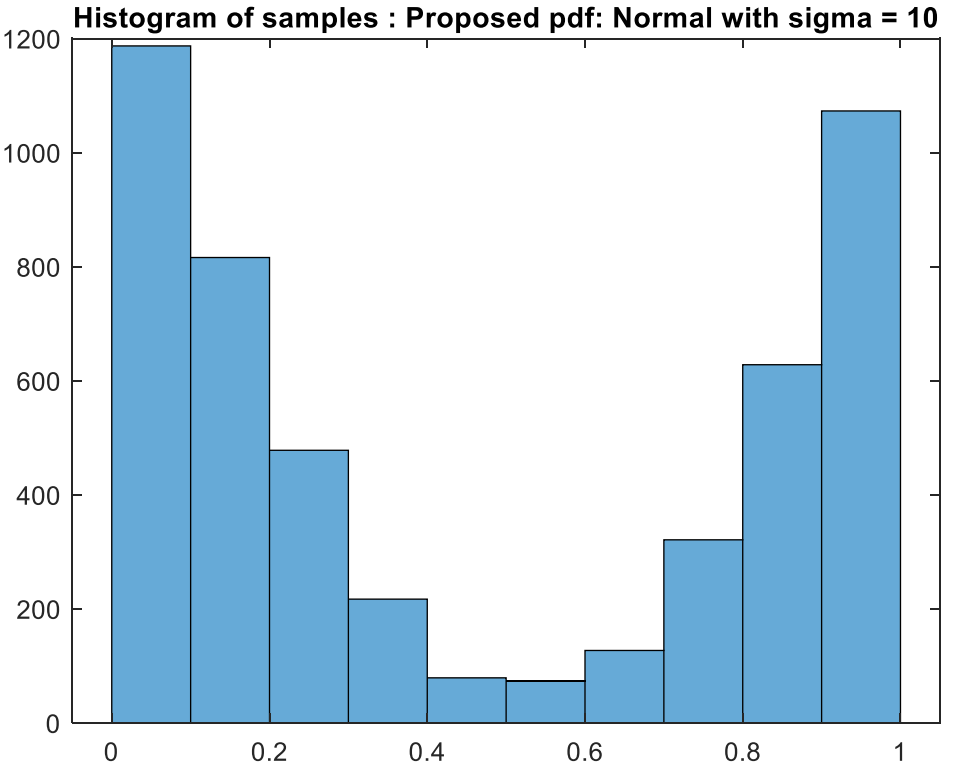

Sample Path for initial point 0.6125
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
1000
2000
3000
4000
5000

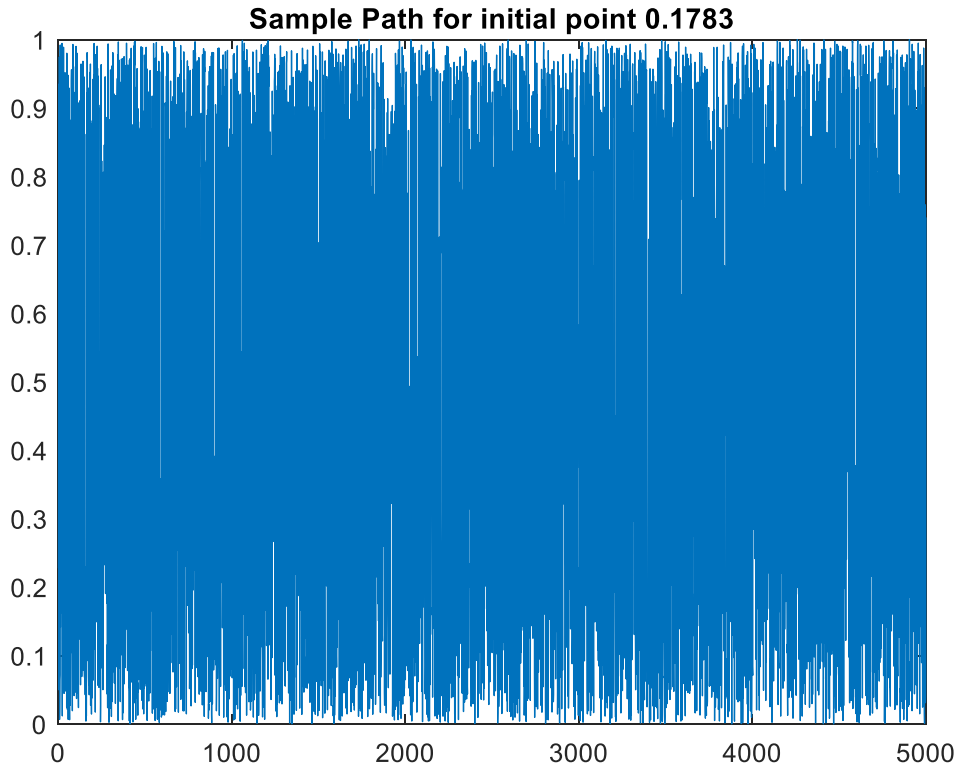


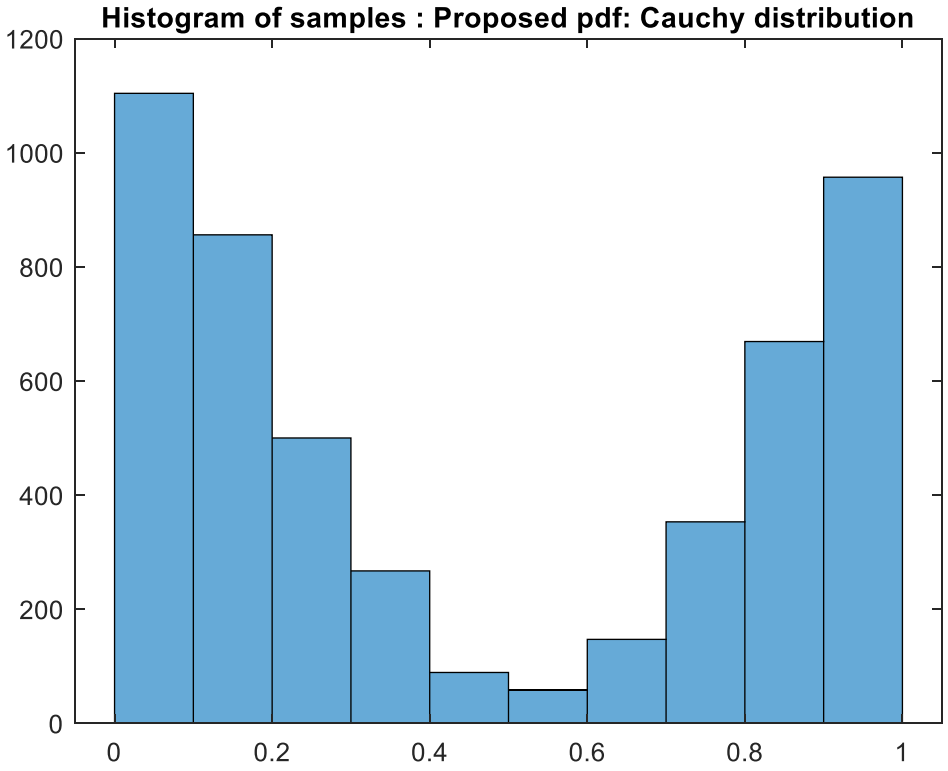
i) Different variance of the proposed probability distributive function

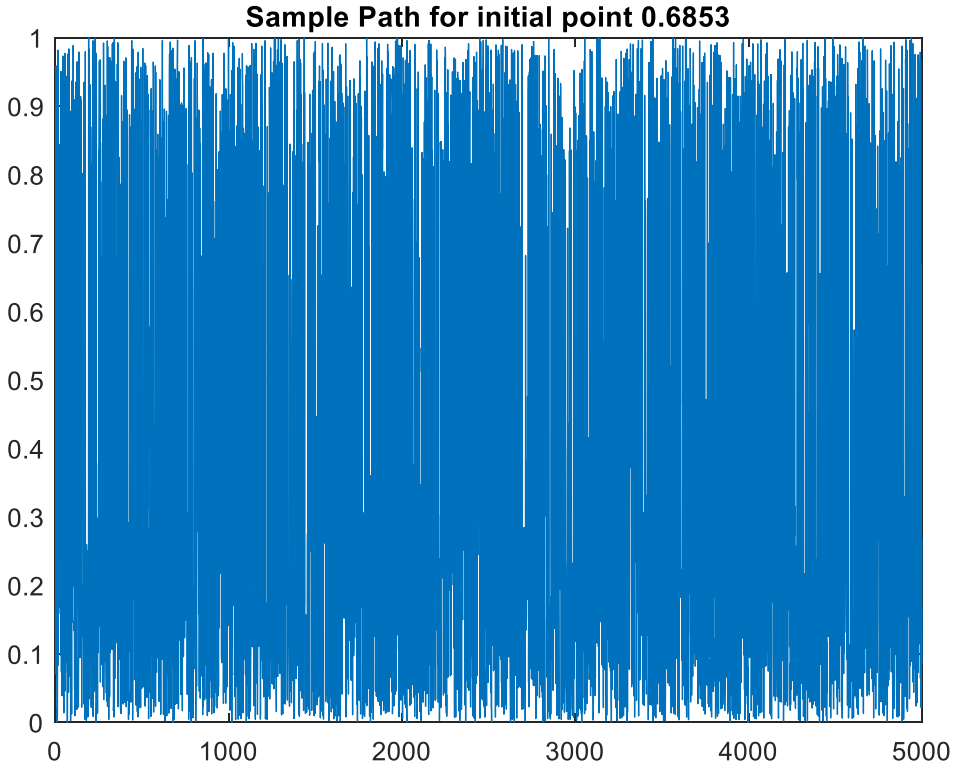



Sample Path for initial point 0.8100
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
1000
2000
3000
4000
5000


Histogram of samples : Proposed pdf: Normal with sigma = 10
1200
1000
800
600
400
200
0
0
0.2
0.4
0.6
0.8
1


Sample Path for initial point 0.1783
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
1000
2000
3000
4000
5000


Histogram of samples : Proposed pdf: Cauchy distribution
1200
1000
800
600
400
200
0
0
0.2
0.4
0.6
0.8
1



Discussion:

- Results depict that Metropolis Hastings algorithm approximate/ sample closely to the given input distribution.
- The module is run for multiple initial values and the Metropolis Hastings algorithm closely approximates the input mixed distribution.
- With very low variance, the Markov chain will mix slowly and takes lot of time to converge while with very high variance, the proposals are likely to be outside the range of the definition of distribution taking lot of time to convergence. Higher the variance better is the convergence rate.
- Acceptance rate is high when low variance proposed probability distribution function is used, while the acceptance rate is too low in case of high variance proposed probability distribution function.
- The proposed probability distribution function used is Normal distribution (symmetric) and Cauchy Distribution. Normal density is better compared to Cauchy.

[MCMC for Optimization]

The n-dimensional Scwefel function
f (x)=418.9829 n − $\sum_{i=1}^{n} xi \sin\sqrt{|x_i|}$ $x_i \in [-500,500]$
Goal:
i. Plot a contour plot of the surface for the 2-D surface
ii. Implement a simulated annealing procedure to find the global minimum of this surface
iii. Explore the behaviour of the procedure starting from the origin with an exponential, a polynomial, and a logarithmic cooling schedule. Run the procedure for t={20, 50, 100, 1000}

iterations for k=100 runs each. Plot a histogram of the function minima your procedure converges to.
iv. Choose your best run and overlay your 2-D sample path on the contour plot of the Schwefel function to visualize the locations your optimization routine explored.

Algorithm:
Simulated Annealing Algorithm:
Goal: To find x* argmin_x g(x)=418.9829 n − $\sum_{i=1}^{n} xi \sin\sqrt{|x_i|}$ $x_i \in [-500,500]$

Requirement: x_0 , Candidate proposal routine $q(.|x_t)$
Cooling schedule: As t tends to infinity, the temperature T_t tends to 0.
Step 1: Sample initial candidate solution x0
Step 2: Generate new candidate y ~ $q(.|x_t)$
Step 3: Calculate Gibbs ratio

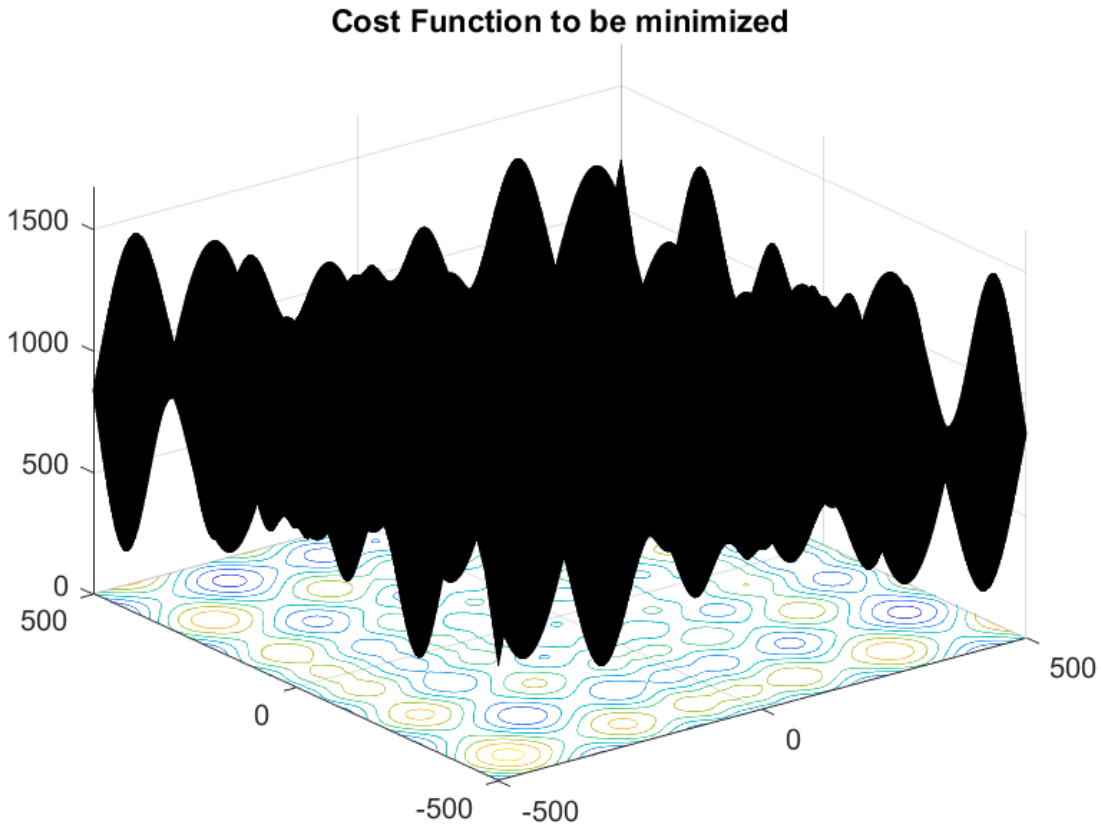
$\alpha(x_t,y) = \min [1, \exp (\frac{(-(g(y)-g(x))}{T})]$

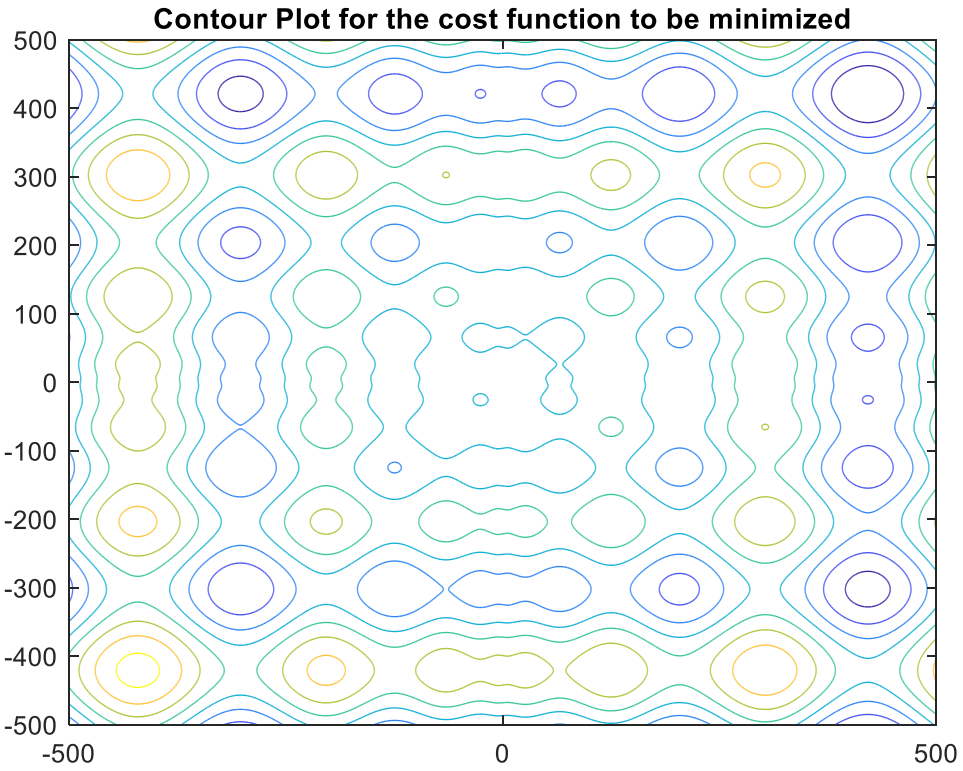
Step 4: Accept y with probability

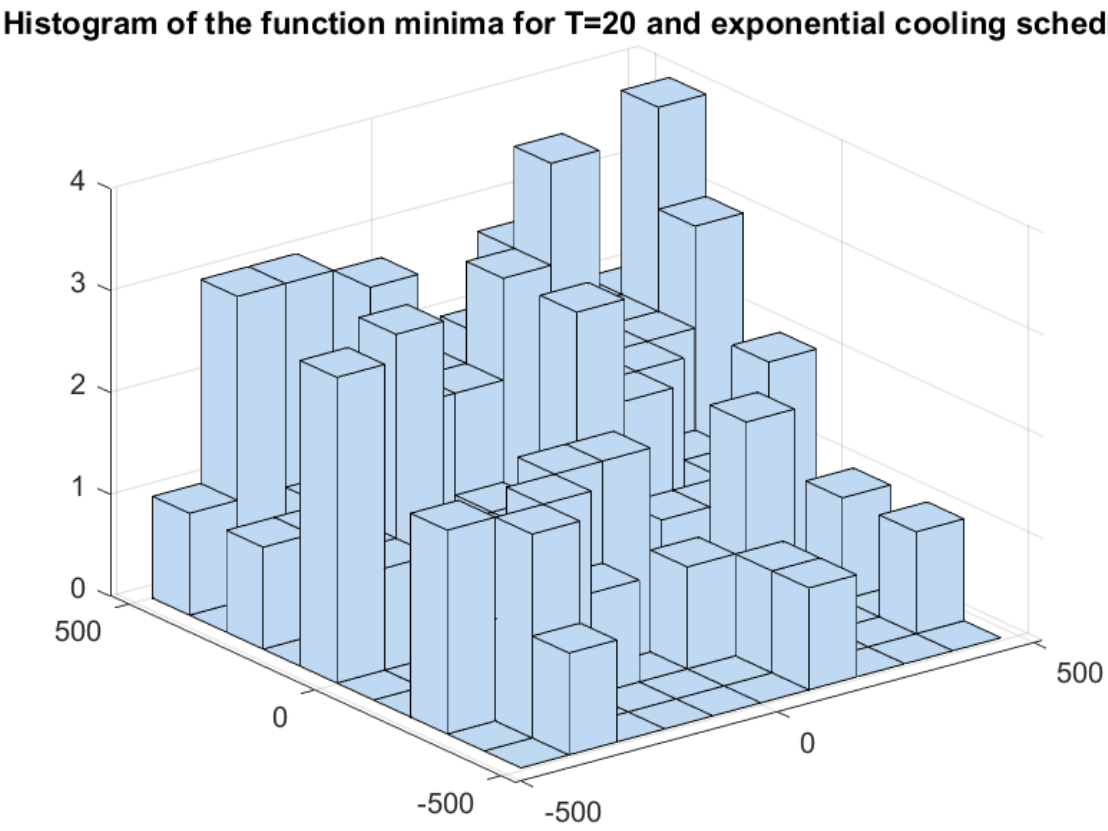
$x_{t+1} \leftarrow y$ if $U <= \alpha(x_t,y)$

Step 5: Go to step 1

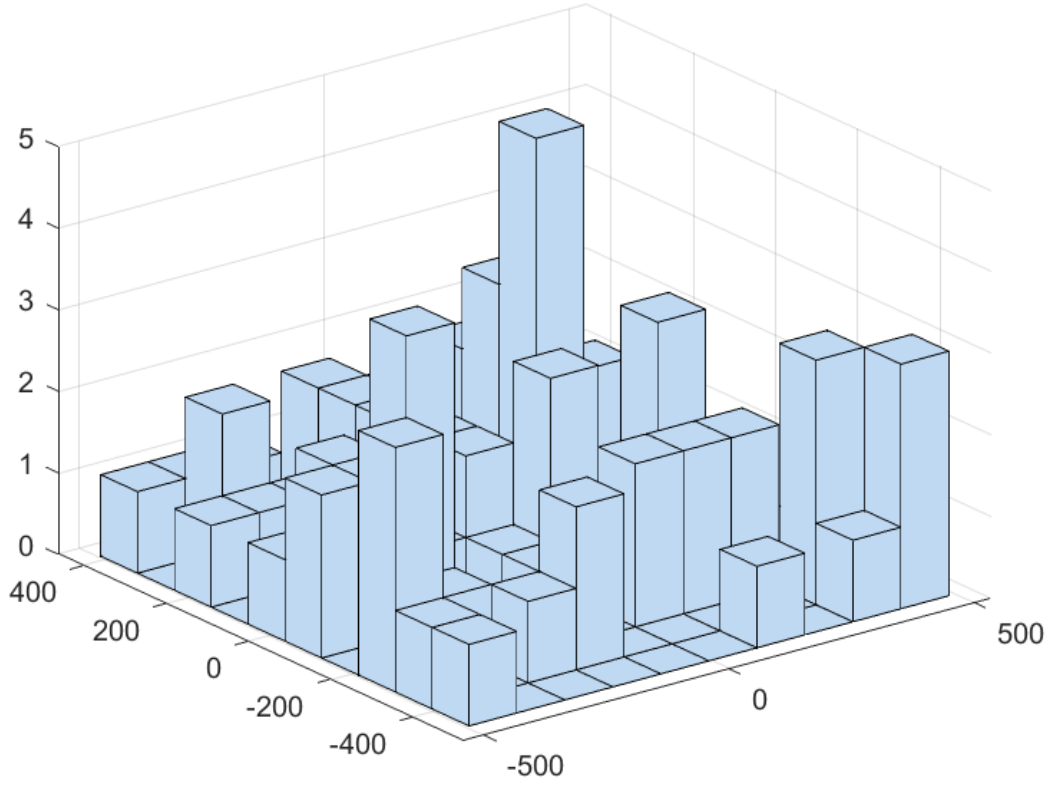
Results:



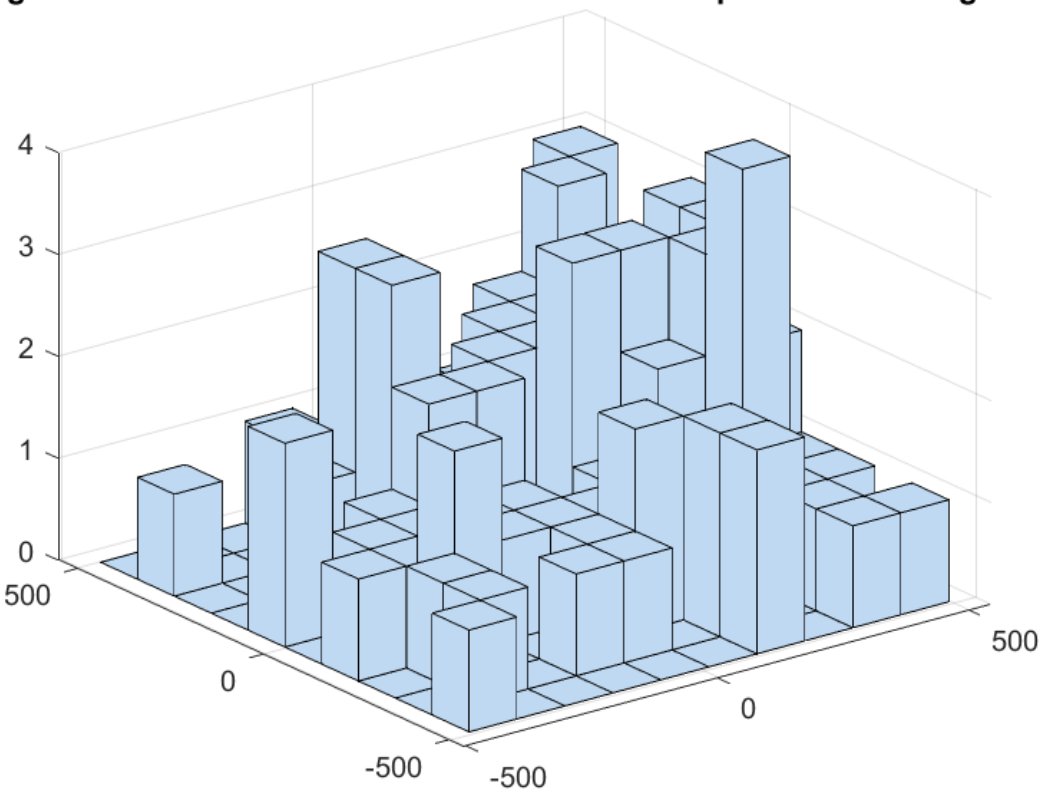

Contour Plot for the cost function to be minimized
500
400
300
200
100
0
-100
-200
-300
-400
-500
-500
0
500


Histogram of the function minima for T=20 and exponential cooling schedule
4
3
2
1
0
500
0
-500
-500
0
500

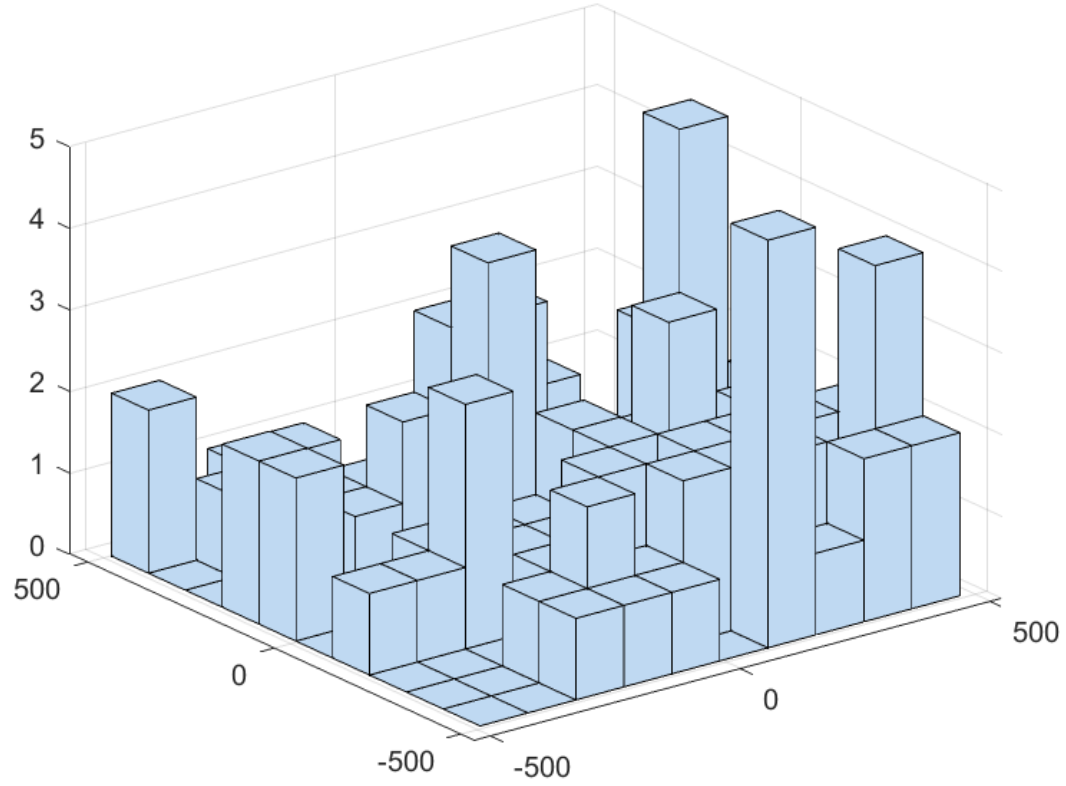
Histogram for the function minima for T=50 with exponential schedule



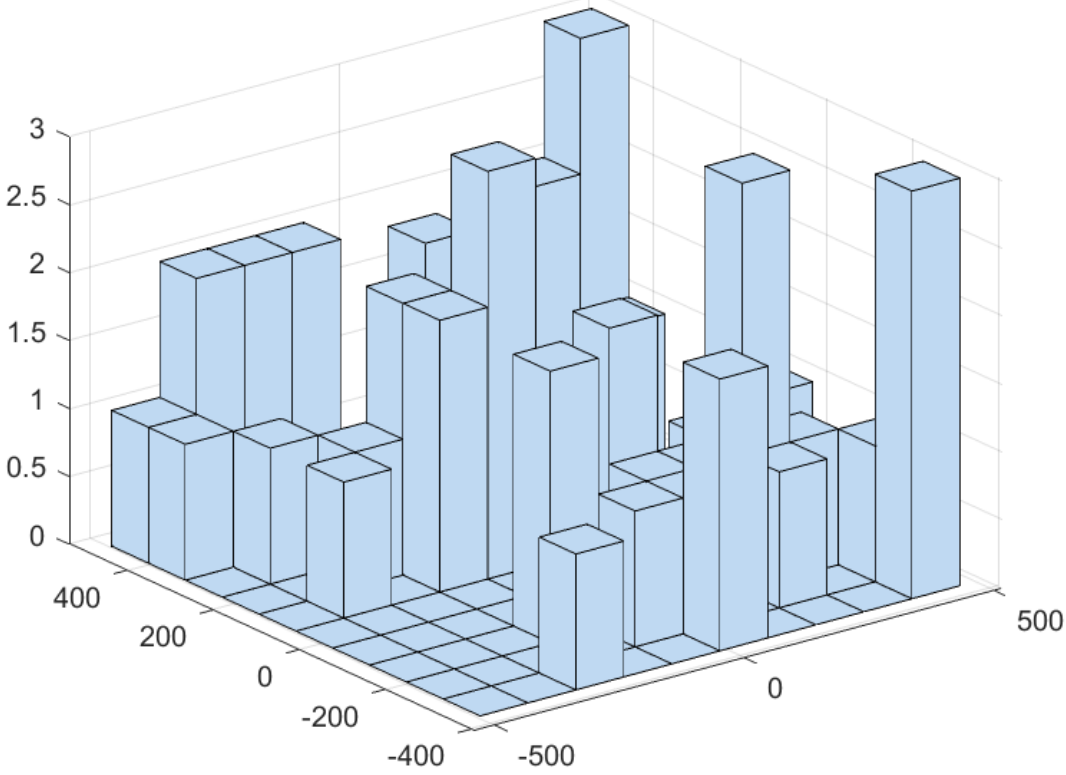
Histogram for the function minima for T =100 with exponential cooling schedul



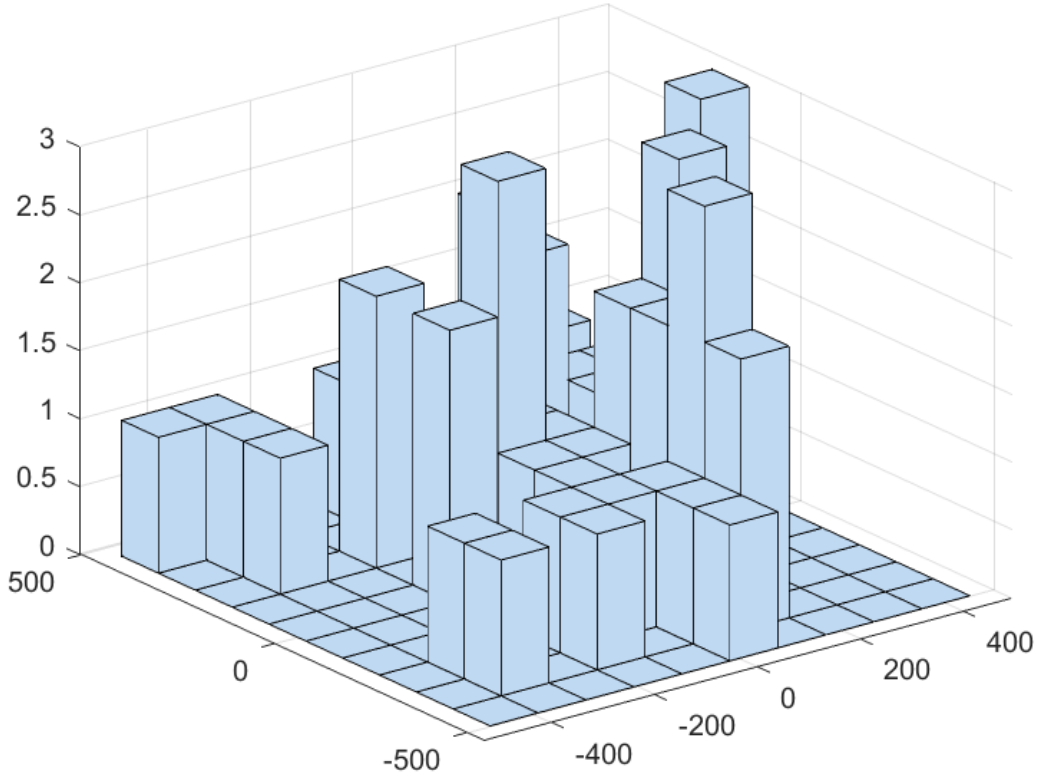
Histogram for function minima with T=1000 and exponential cooling schedule



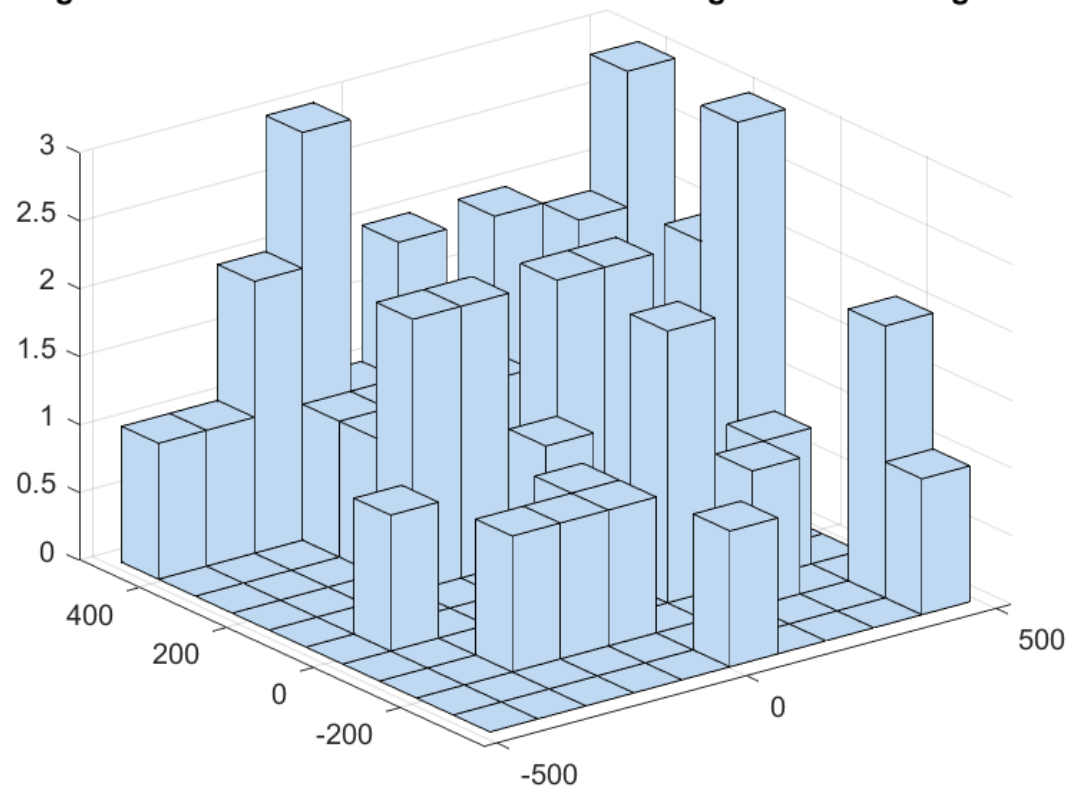
Histogram for function minima with T = 20 and logarithmic cooling schedule



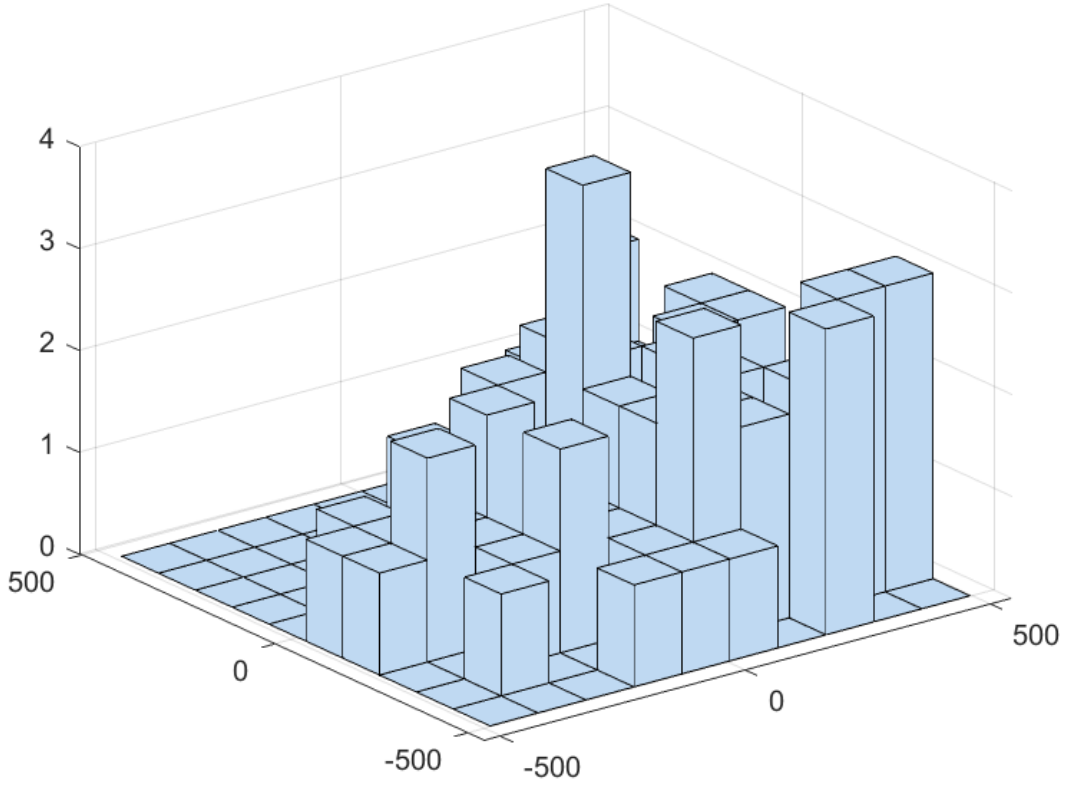
Histogram for function minima with T = 50 and logarithmic cooling schedule



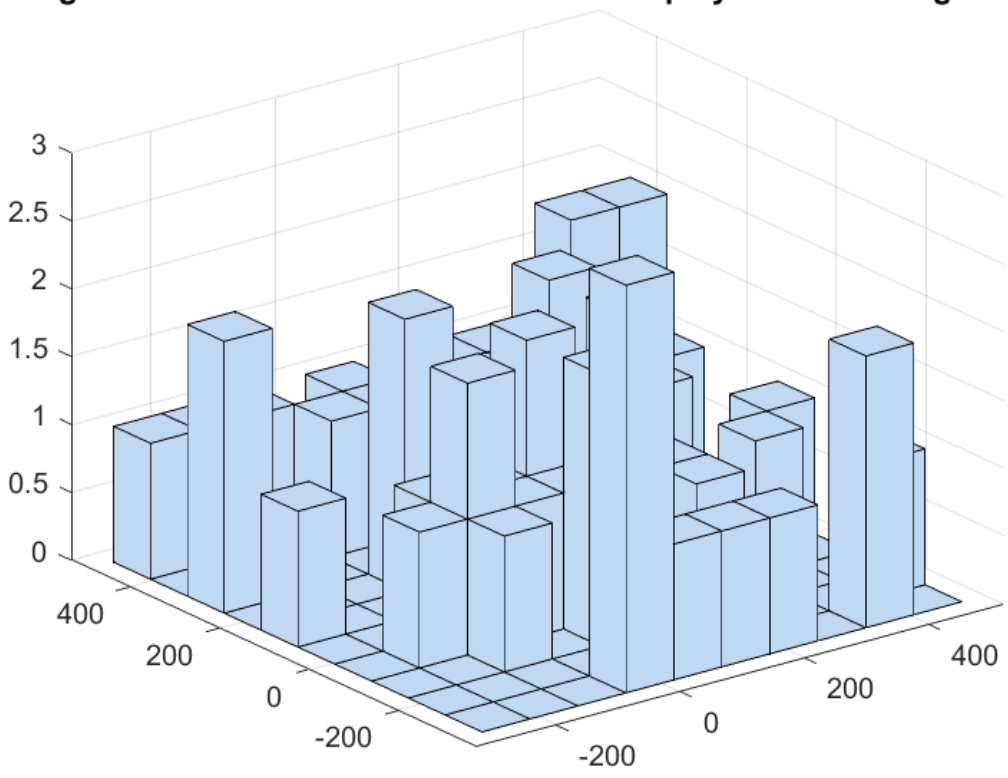
Histogram for function minima with T=100 and logarithmic cooling schedule



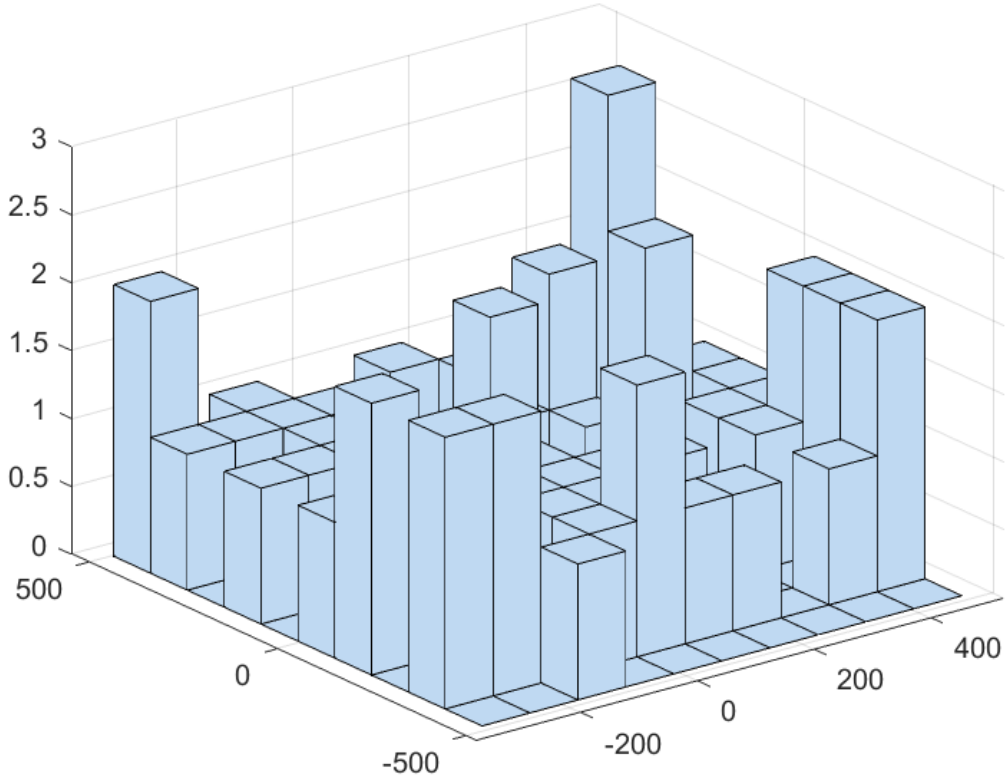
Histogram for function minima with T = 1000 and logarithmic cooling schedule



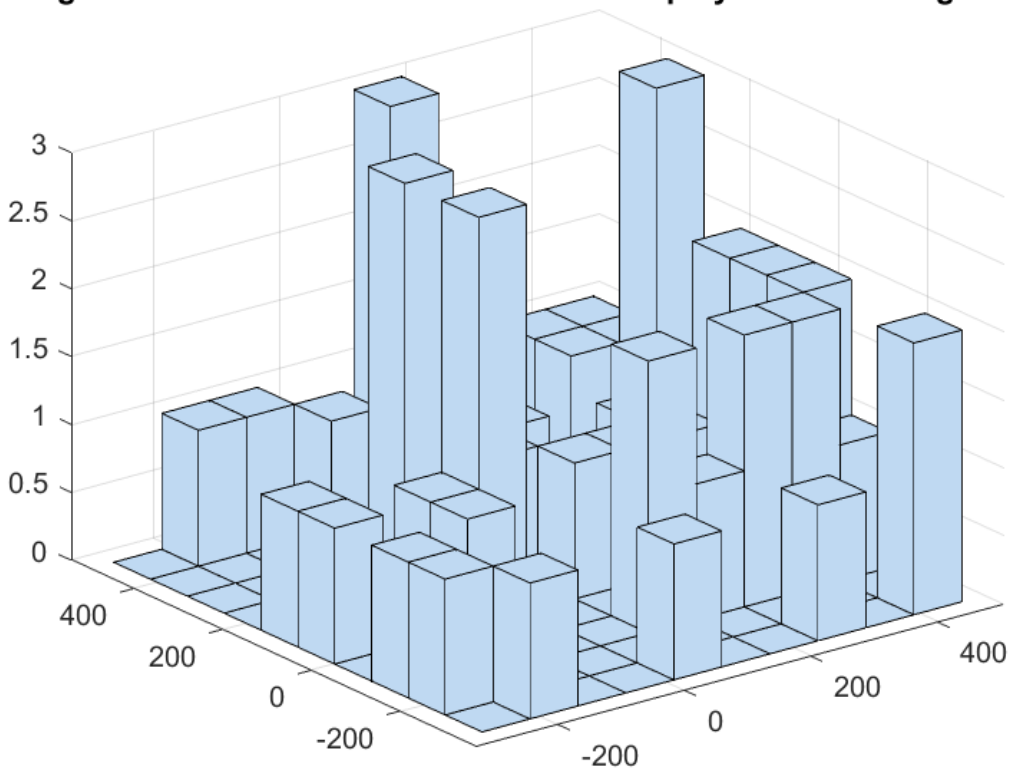
Histogram for function minima with T=20 and polynomial cooling schedule



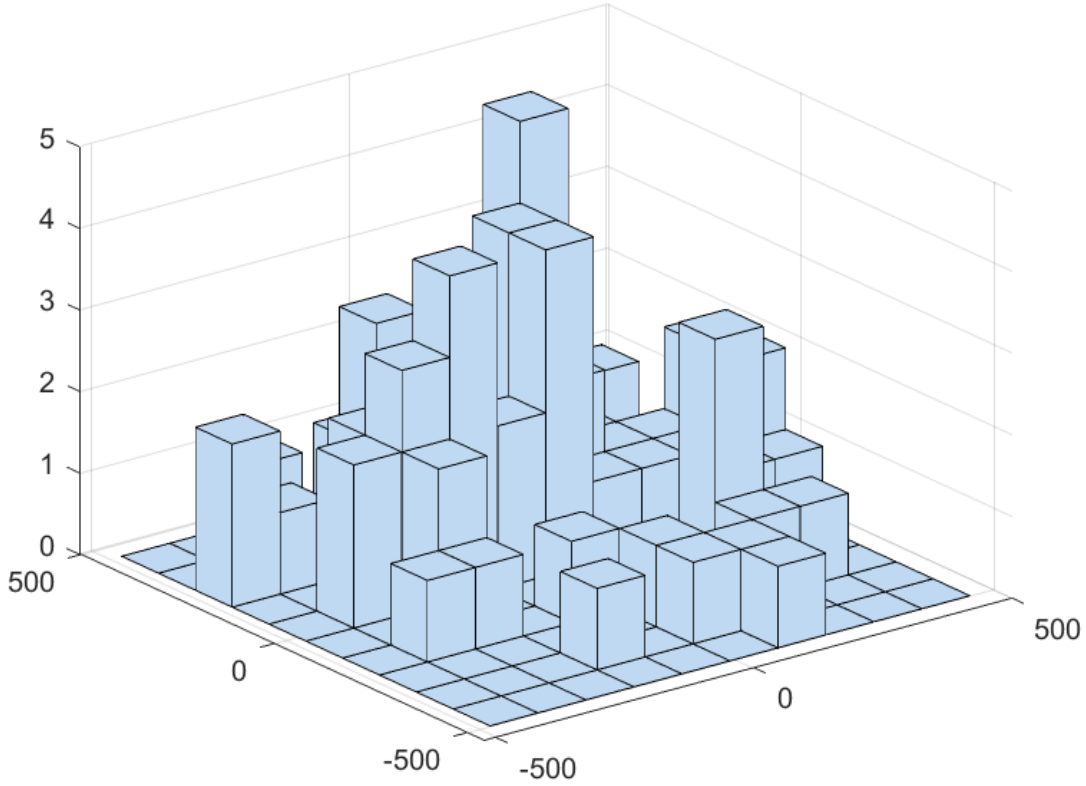
Histogram for function minima with T=50 and polynomial cooling schedule



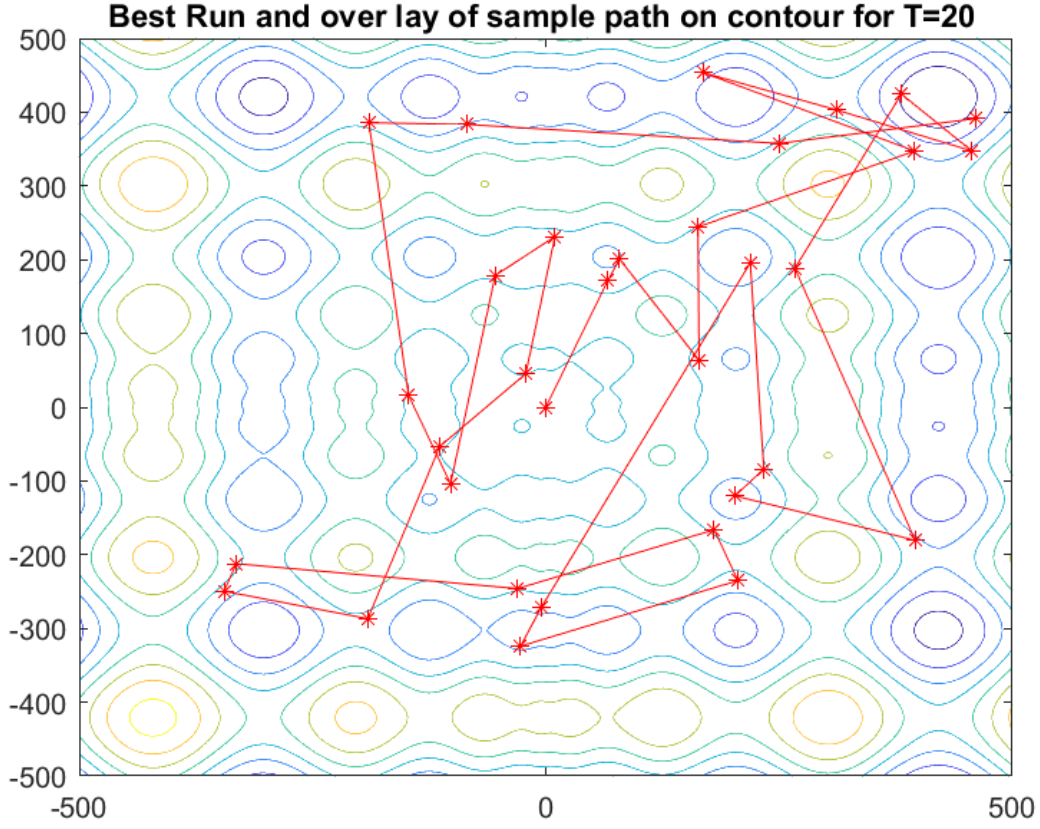
Histogram for function minima for T=100 with polynomial cooling schedule



Histogram of function minima for T=1000 for polynomial cooling schedule



Best Run and over lay of sample path on contour for T=20



Discussion:

- Among exponential, polynomial and logarithmic cooling schedule, polynomial and logarithmic cooling schedule performed better compared to exponential cooling schedule.
- Lower the temperature, faster is the convergence compared to higher temperature. It takes less time to cool from lower temperature similar to simulated annealing.
- T=20 performs better than T=1000
- Proposed probability distributive function is Normal.
- The global function minima is 420.9687, 420.9687. Logarithmic function is a better cooling schedule to converge at global minima despite many local minima present in the function.
- The best model is for T=20 and polynomial/logarithmic cooling schedule.

[Optimal Paths]

The famous Traveling Salesman Problem (TSP) is an NP-hard routing problem. The time to find optimal solutions to
TSPs grows exponentially with the size of the problem (number of cities). A statement of the TSP goes thus:
"A salesman needs to visit each of N cities exactly once and in any order. Each city is connected to other cities via an air transportation network. Find a minimum length path on the network that goes through all N cities exactly once (an optimal Hamiltonian cycle)."
A TSP solution c=(c1, ...c N) is just an ordered list of the N cities with minimum path length. We will be exploring

Goal:

i. Pick N=10 2-D points in the [0,1000]x[0,1000] rectangle. These 2-D samples will represent the locations of N=10 cities.
1. Write a function to capture the objective function of the TSP problem:
D(c)= $\sum_{i=1}^{N-1} \|c(i+1) - c(i)\|$
2. Start with a random path through all N cities c (a random permutation of the cities), an initial high temperature T0, and a cooling schedule Tk=f (T_0 , k).
3. Randomly pick any two cities in your current path. Swap them. Use the difference between the new and old path length to calculate a Gibbs acceptance probability. Update the path accordingly.
4. Update your annealing temperature and repeat the previous city swap step. Run the simulated annealing procedure "to convergence."
5. Plot the values of your objective function from each step. Plot your final TSP city tour.
ii. Run the Simulated Annealing TSP solver you just developed for N = {40, 400, 1000} cities. Explore the speed and convergence properties at these different problem sizes. You might want to play with the cooling schedules.

Algorithm:

General Steps:
Step 1: Pick N=10 points in [0,1000]x[0,1000] – location of 10 cities.
Step 2: Randomly generate a path across all cities
Step 3: Randomly pick two cities in the current path and swap them.
Step 4: Calculate the cost of the current path and the swapped path
Step 5: Simulated Annealing algorithm is used to calculate the Gibbs acceptance probability.
Step 6: If accepted, swapped path is now the current path, else another swapped path is generated

Step 7: The procedure is repeated for different cities with N=40,400,1000 cities.

Simulated Annealing Algorithm:
Goal: To find x* argmin_x g(x)
Requirement: x0, Candidate proposal routine $q(.|x_t)$
Cooling schedule: As t tends to infinity, the temperature T_t tends to 0.
Step 1: Sample initial candidate solution x0
Step 2: Generate new candidate y ~ $q(.|x_t)$
Step 3: Calculate Gibbs ratio

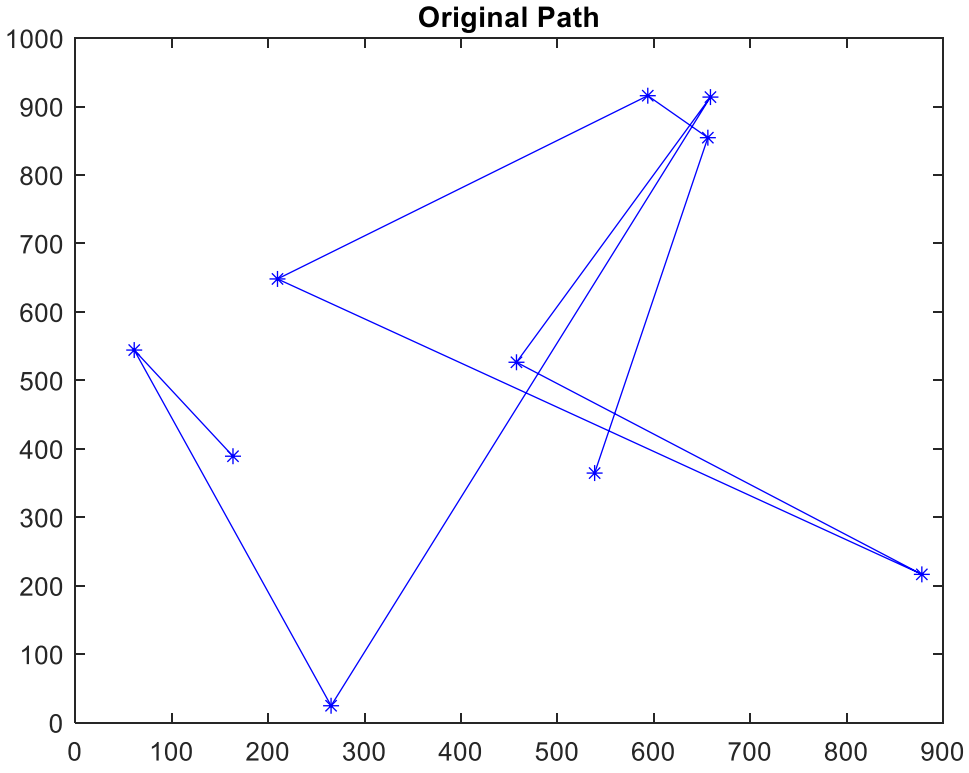
$$\alpha(x_t,y) = \min\left[1, \exp\left(\frac{(-g(y)-g(x))}{T}\right)\right]$$

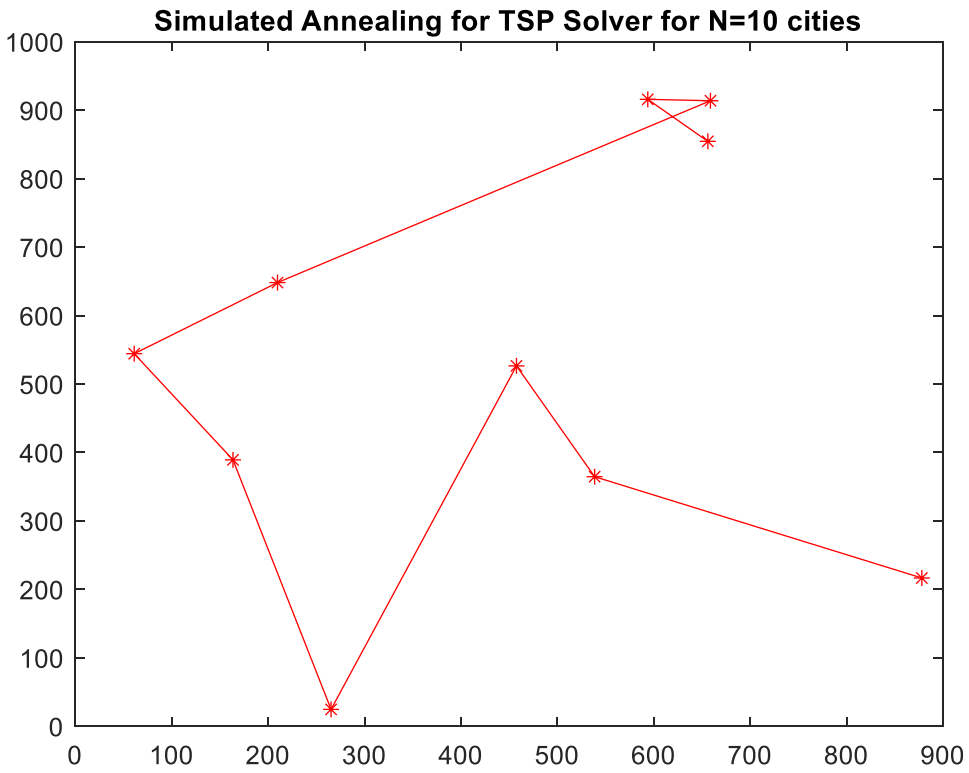
Step 4: Accept y with probability

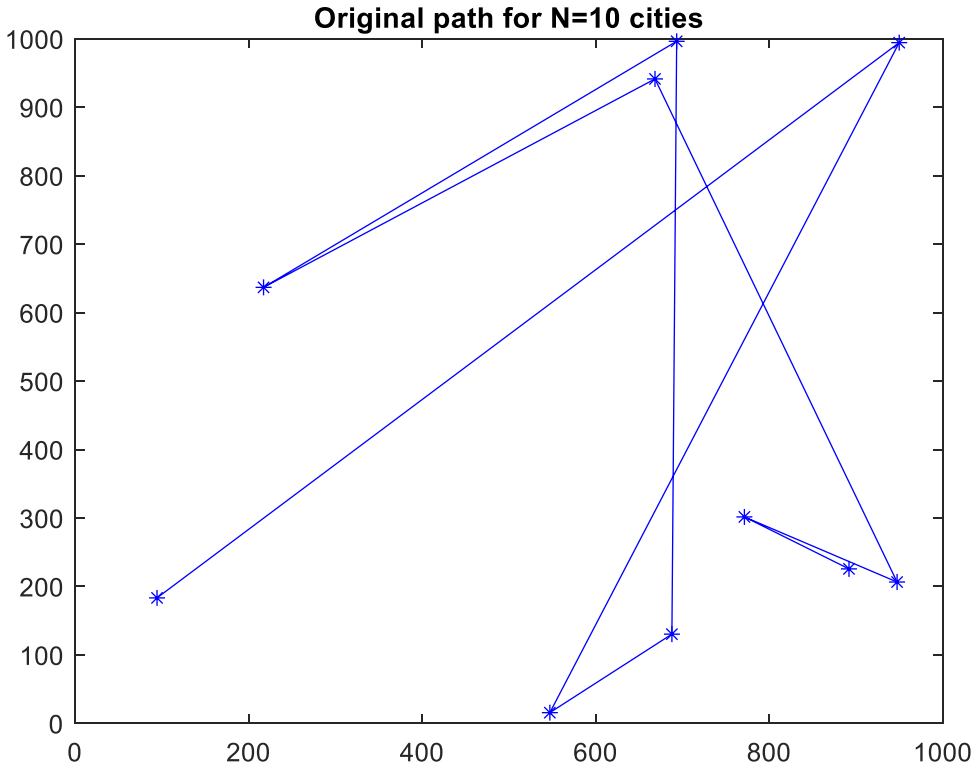
$x_{t+1} \leftarrow y$ if $U <= \alpha(x_t, y)$

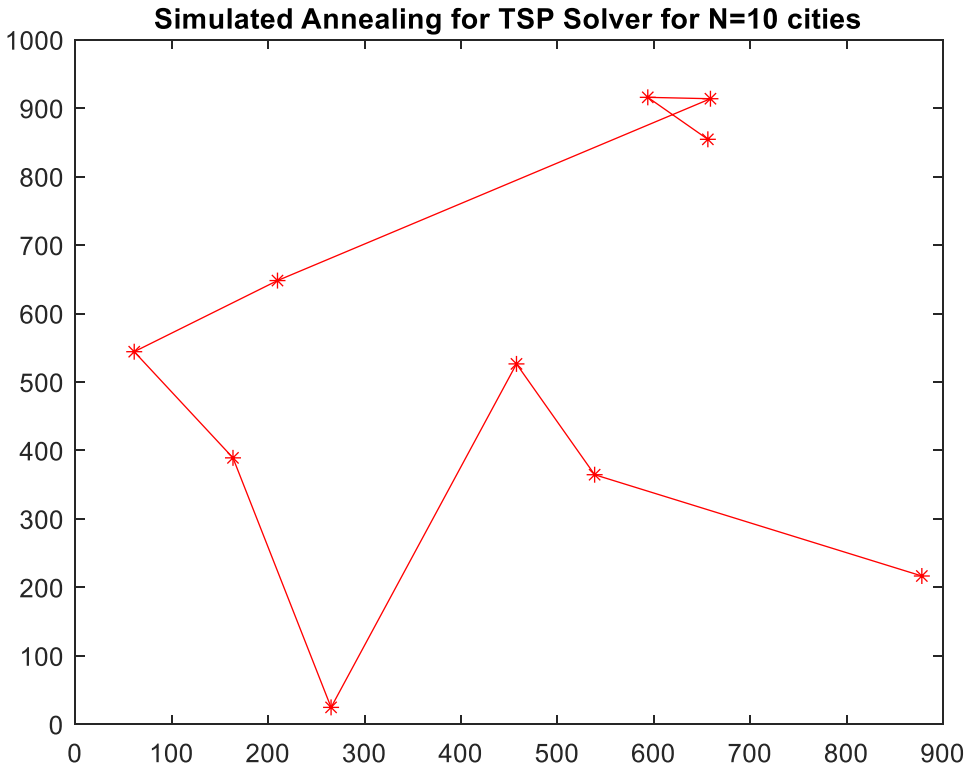
Step 5: Go to step 1 in Simulated Annealing algorithm

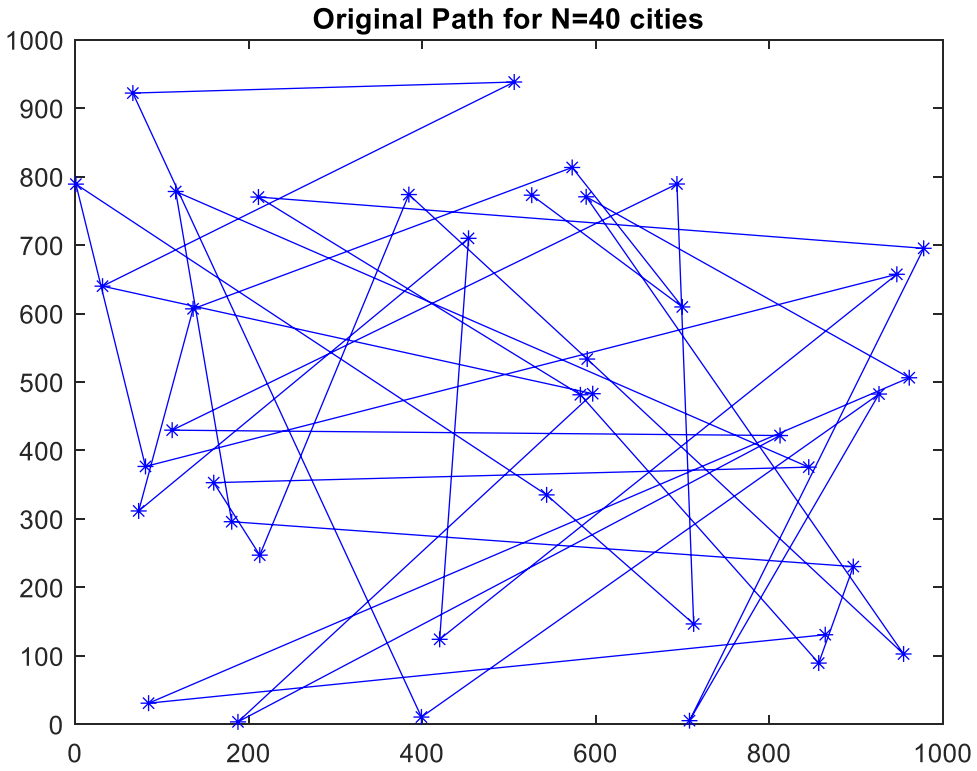
Result:

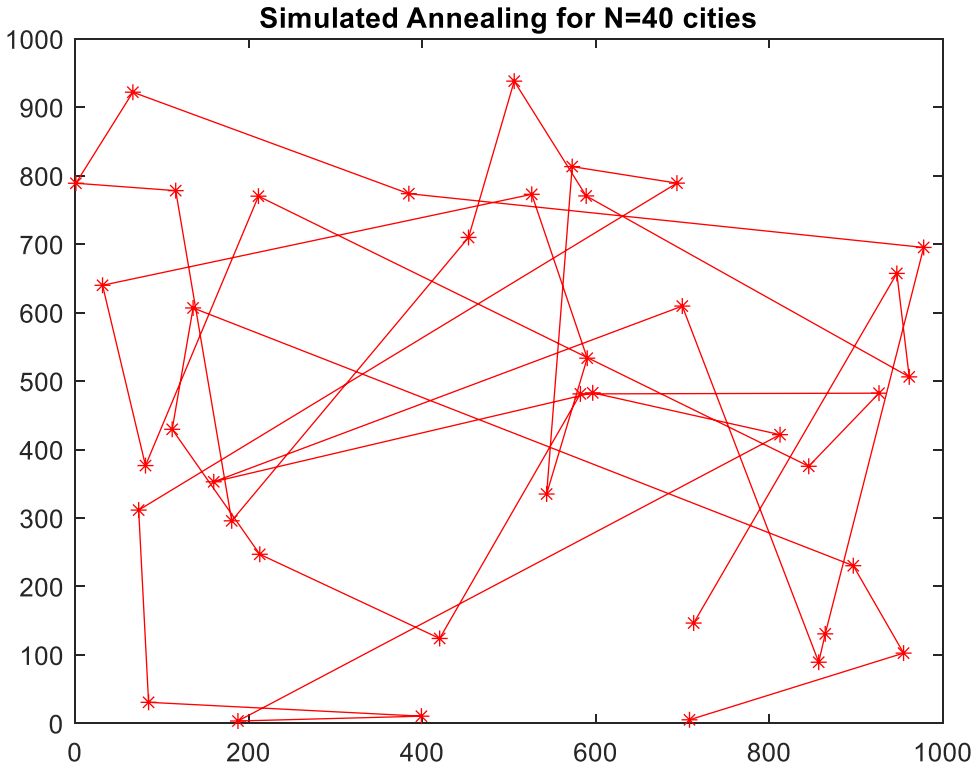


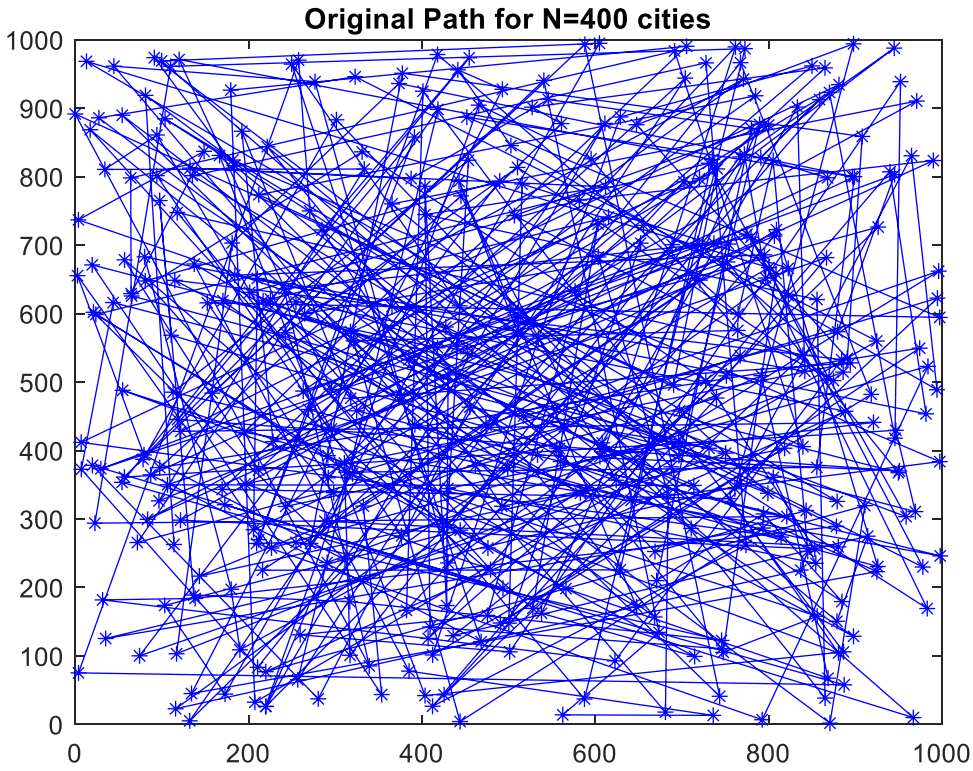

Simulated Annealing for TSP Solver for N=10 cities
1000
900
800
700
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
800
900

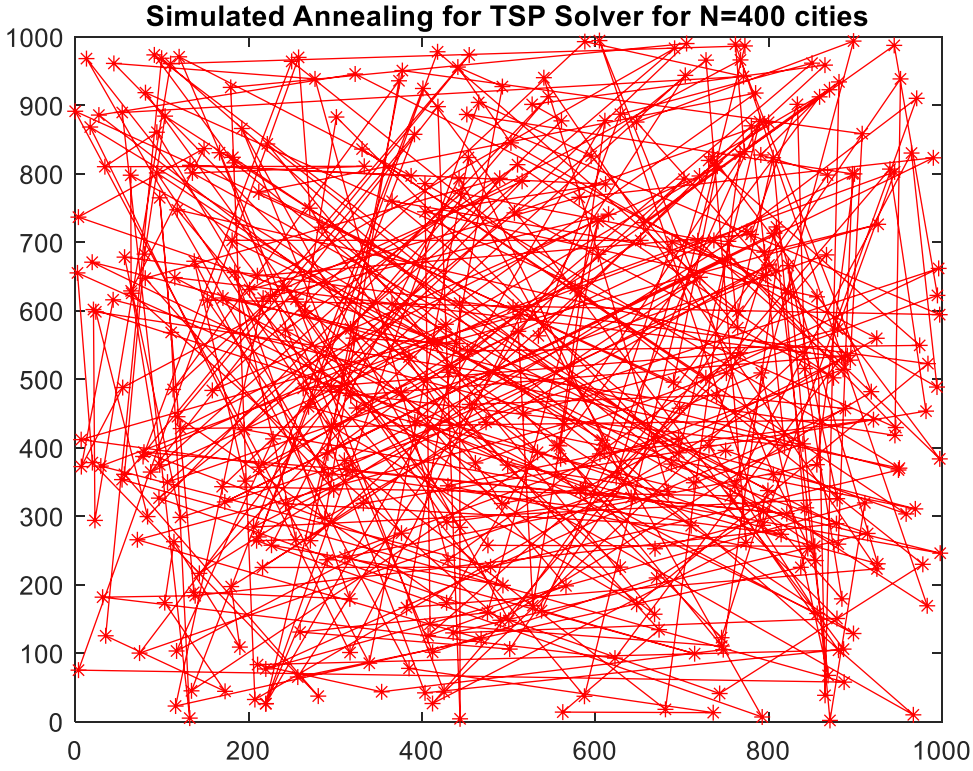

Original path for N=10 cities
1000
900
800
700
600
500
400
300
200
100
0
0
200
400
600
800
1000

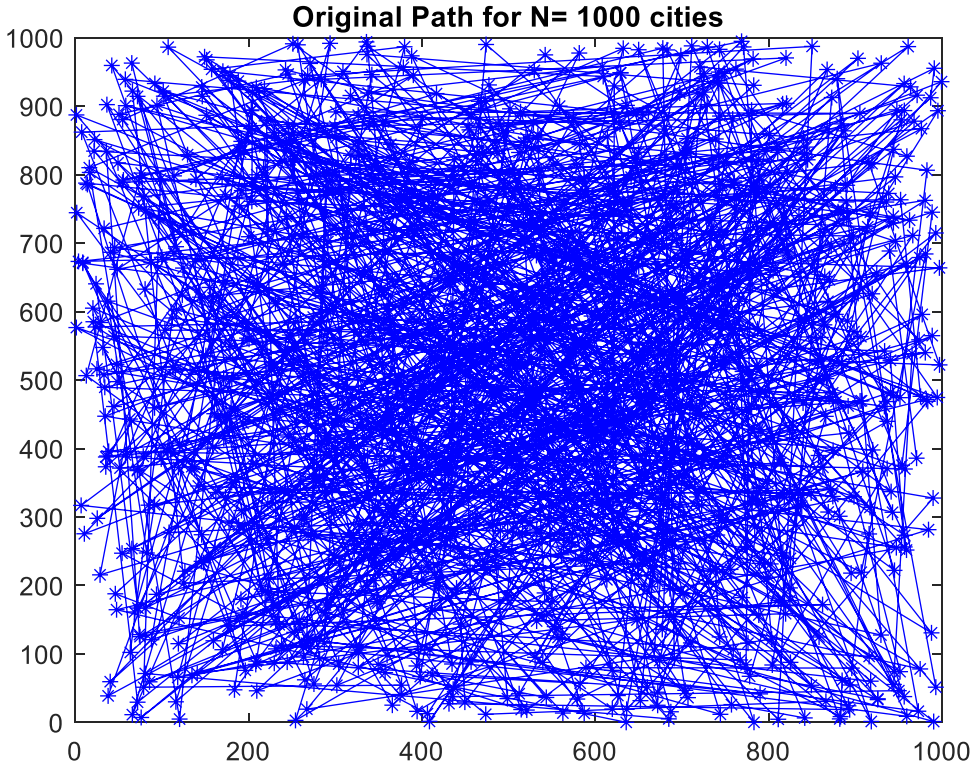

Simulated Annealing for TSP Solver for N=10 cities
1000
900
800
700
600
500
400
300
200
100
0
0
100
200
300
400
500
600
700
800
900

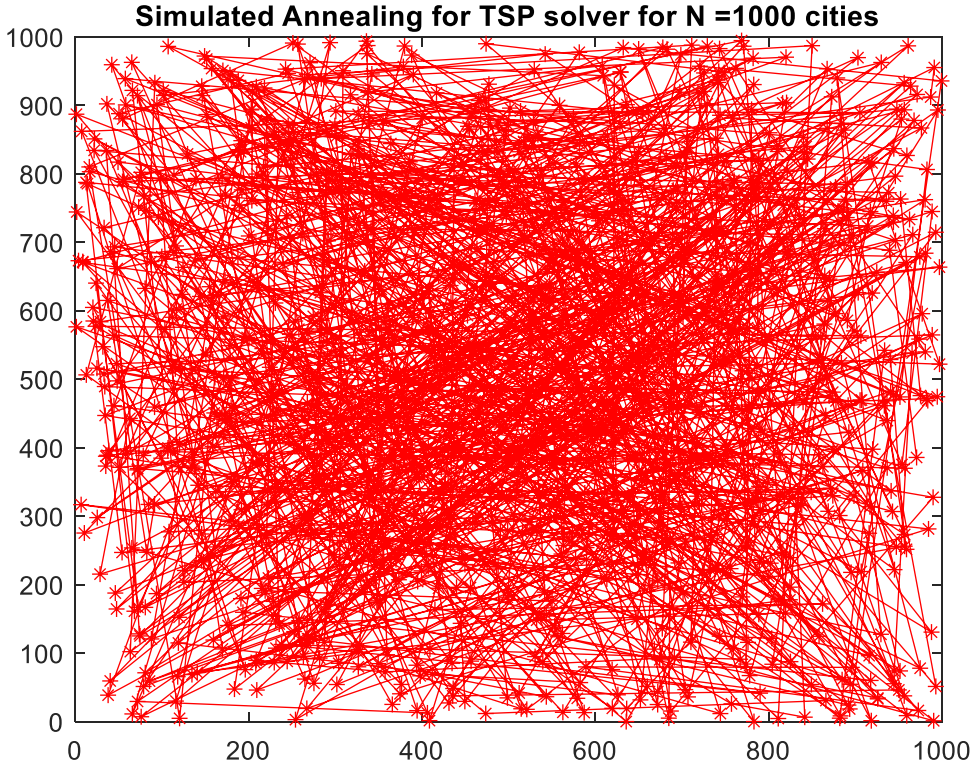

Original Path for N=40 cities
1000
900
800
700
600
500
400
300
200
100
0
0
200
400
600
800
1000


Simulated Annealing for N=40 cities
1000
900
800
700
600
500
400
300
200
100
0
0
200
400
600
800
1000


Original Path for N=400 cities
1000
900
800
700
600
500
400
300
200
100
0
0
200
400
600
800
1000


Simulated Annealing for TSP Solver for N=400 cities
1000
900
800
700
600
500
400
300
200
100
0
0
200
400
600
800
1000


Original Path for N= 1000 cities
1000
900
800
700
600
500
400
300
200
100
0
0
200
400
600
800
1000


Simulated Annealing for TSP solver for N =1000 cities
1000
900
800
700
600
500
400
300
200
100
0
0
200
400
600
800
1000

Discussion:

- With N=10, the solution generated using Simulated annealing for traveling salesman problem, it can be noticed that an optimal path is generated to connect all the cities with least possible distance with each city traversed only once.
- As N is increased from 10,40,400 to 1000, the computational complexity is high. Speed of computation decreases as N increases.
- It takes a lot of time to converge as N increases.
- It is often difficult to converge as N increases with any temperature and cooling schedules.
- Among polynomial, logarithmic and exponential cooling schedules, polynomial and logarithmic perform better in contrast to exponential cooling schedule.
- All the simulations are performed for T=10.

Code:

1. [MCMC for Sampling]

```
i=1;
for x=0:0.01:1
p(i)=0.6*betapdf(x,1,8)+0.4*betapdf(x,9,1);
i=i+1;
end
figure(1)
plot(0:0.01:1,p);

%samples=p(rand(1,100));
accept_count=0;
x(1,1)=rand;
k=1;
sigma=0.1;
i=1;
while(accept_count<5000)
    proposed_pdf(i)=normrnd(x(1,i),sigma);
    %proposed_pdf(i)=x(1,i)+tan(pi*(ab-0.5)); %Cauchy
alpha=min(1,((0.6*betapdf(proposed_pdf(i),1,8)+0.4*betapdf(proposed_
pdf(i),9,1))/(0.6*betapdf(x(1,i),1,8)+0.4*betapdf(x(1,i),9,1))));
    u=rand;
    if u<=alpha
        sample_collection(accept_count+1)=proposed_pdf(i);
        accept_count=accept_count+1;
        x(1,i+1)=proposed_pdf(i);
    else
        x(1,i+1)=x(1,i);
    end
    i=i+1;
end
figure(2);
histogram(sample_collection);
title('Histogram of samples : Proposed pdf: Normal with sigma =
0.1');
```

```
figure(3);
plot(sample_collection);
title('Sample Path for initial point 0.77863');
```

2. [MCMC for Optimization]

```
clear all
close all

n=2;
for i=-500:500
    for j=-500:500
        x(1,1)=i;
        x(1,2)=j;
        function_x(i+501,j+501)=418.9829*n-
((x(1,1)*sin(sqrt(abs(x(1,1)))))+(x(1,2)*sin(sqrt(abs(x(1,2))))));
    end
end
X=-500:500;
Y=-500:500;
figure(1);
surfc(X,Y,function_x);
figure(4);
contour(X,Y,function_x);


accept_count=0;
x(1,1)=0;
x(1,2)=0;
k=1;
sigma=200;
i=1;
original_T=20;
T=original_T;
while(i<100)
    proposed_pdf(i,:)=normrnd(x(i,:),sigma,[1,2]);
    g_y=(418.9829*n -
((proposed_pdf(i,1)*sin(sqrt(abs(proposed_pdf(i,1)))))+(proposed_pdf
(i,2)*sin(sqrt(abs(proposed_pdf(i,2)))))));
    g_x=(418.9829*n-
((x(1,1)*sin(sqrt(abs(x(1,1)))))+(x(1,2)*sin(sqrt(abs(x(1,2)))))));
    if(500>=proposed_pdf(i,1) && proposed_pdf(i,1)>=-500 && -
500<=proposed_pdf(i,2) && proposed_pdf(i,2)<=500)
        alpha=min(1,exp(-(g_y-g_x)/T));
        u=rand;
        if ( u<=alpha )
            sample_collection(accept_count+1,:)=proposed_pdf(i,:);
            accept_count=accept_count+1;
            x(i+1,:)=proposed_pdf(i,:);
        else
            x(i+1,:)=x(i,:);
        end
        i=i+1;
        a=rand;
```

```
%         T=original_T/log(i+1); % logarithm
%         T=original_T/exp(-(i+1)); %exponential
        T=original_T/((i+1).^(a)); %polynomial
    end
end


figure(2);

hist3(sample_collection);


X=-500:500;
Y=-500:500;
figure(5);
contour(X,Y,function_x);
hold on;
plot(sample_collection(:,1),sample_collection(:,2),'-*r');


%
% compPlot = figure('Name', 'Comparison of Stuff');
% ax1 = axes('Parent', compPlot);
% hold(ax1, 'on');
% plot(ax1, a, 'Color', 'blue');
% plot(ax1, b, 'Color', 'red');
% title(ax1, 'Figure 3: Plot of Stuff');
% legend('Thing 1','Thing 2');
% hold(ax1, 'off');
```

3. [Optimal Paths]

```
clear all;
close all;

N=1000;
x=1000*rand(1,N);
y=1000*rand(1,N);


initial_temp=1;
initial_path=randperm(N);
accept_count=0;
j=1;
path=initial_path;
T=initial_temp;
while(j<5000)
    temp=randperm(N,2);
    ind1=find(path(j,:)==temp(1,1));
    ind2=find(path(j,:)==temp(1,2));
    swapped_path(j,:)=path(j,:);
    swapped_path(j,ind1)=temp(1,2);
    swapped_path(j,ind2)=temp(1,1);
    sum1=0;
```

```
    for i=1:N-1
    sum1=sum1+sqrt((x(1,swapped_path(j,i+1))-
x(1,swapped_path(j,i))).^2+(y(1,swapped_path(j,i+1))-
y(1,swapped_path(j,i))).^2);
    end
    D_new_path(1,j)=sum1;
    sum2=0;
    for i=1:N-1
    sum2=sum2+sqrt((x(1,path(j,i+1))-
x(1,path(j,i))).^2+(y(1,path(j,i+1))-y(1,path(j,i))).^2);
    end
    D_old_path(1,j)=sum2;
    alpha=exp(D_old_path(1,j)-D_new_path(1,j))/T;
    u=rand;
    if ( (u<=alpha) || (D_new_path(1,j) < D_old_path(1,j)))
        sample_collection(accept_count+1,:)=swapped_path(j,:);
        accept_count=accept_count+1;
        path(j+1,:)=swapped_path(j,:);
        cost_function(accept_count+1)=D_new_path(1,j);
    else
        path(j+1,:)=path(j,:);
    end
    j=j+1;
    T=T/(i+1);
%     T=initial_temp/(i.^(-0.5));
%     T=T*0.99;

end

figure(1);
plot(x(1,initial_path),y(1,initial_path),'-*b')
figure(2); plot(x(1,path(100,:)),y(1,path(100,:)),'-*r')
% figure(3);plot(cost_function,'*-b')
```