EE 511: Spring 2018
Simulation Methods for Stochastic Systems
Project #5: Optimization & Sampling via MCMC

[MCMC for Sampling]

```
nsamples=1000;
delta=0.5;
pd1 =@(x) betapdf(x,1,8);
pd2= @(x)betapdf(x,9,1);
pdf = @(x)( 0.6*pd1(x) + 0.4*pd2(x));


start=0;
proppdf = @(x,y)normpdf(y-x,0,1);
proprnd = @(x) x + rand*2*delta - delta;
tic;
smpl =
mhsample(start,nsamples,'pdf',pdf,'proppdf',proppdf,'proprnd',proprnd);
t=toc;
xxhat = cumsum(smpl.^2)./(1:nsamples)';
figure;
plot(1:nsamples,xxhat)

figure;
h = histfit(smpl,50);
h(1).FaceColor = [.8 .8 1];
```

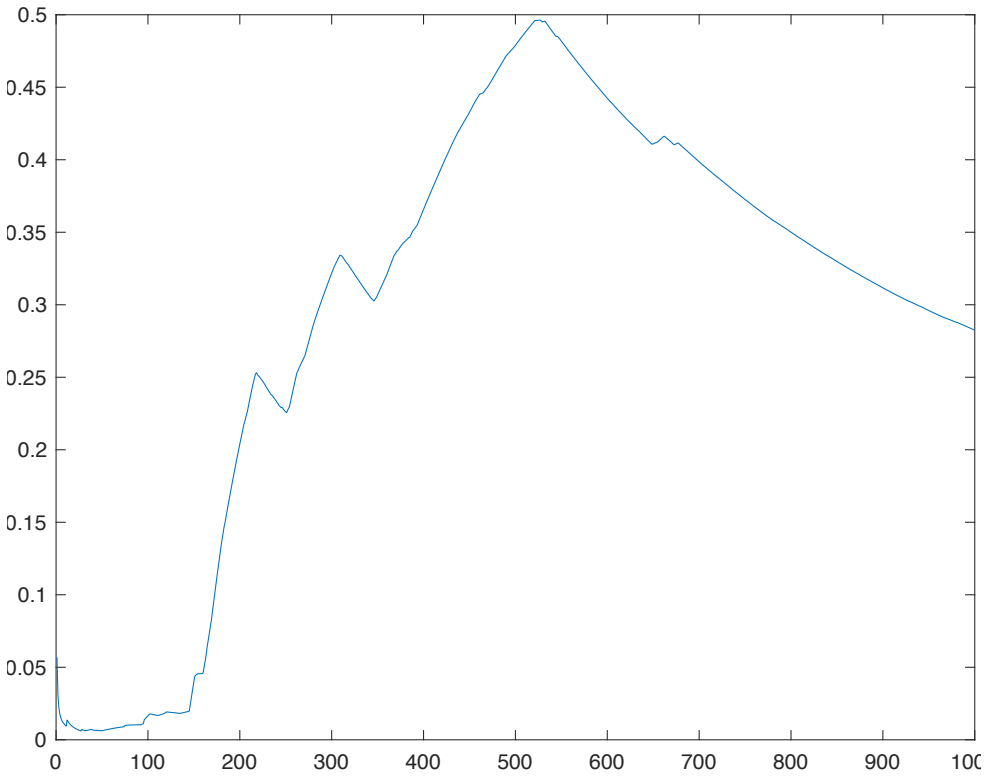


Figure 1: MH Algo sample path gausian start from 0

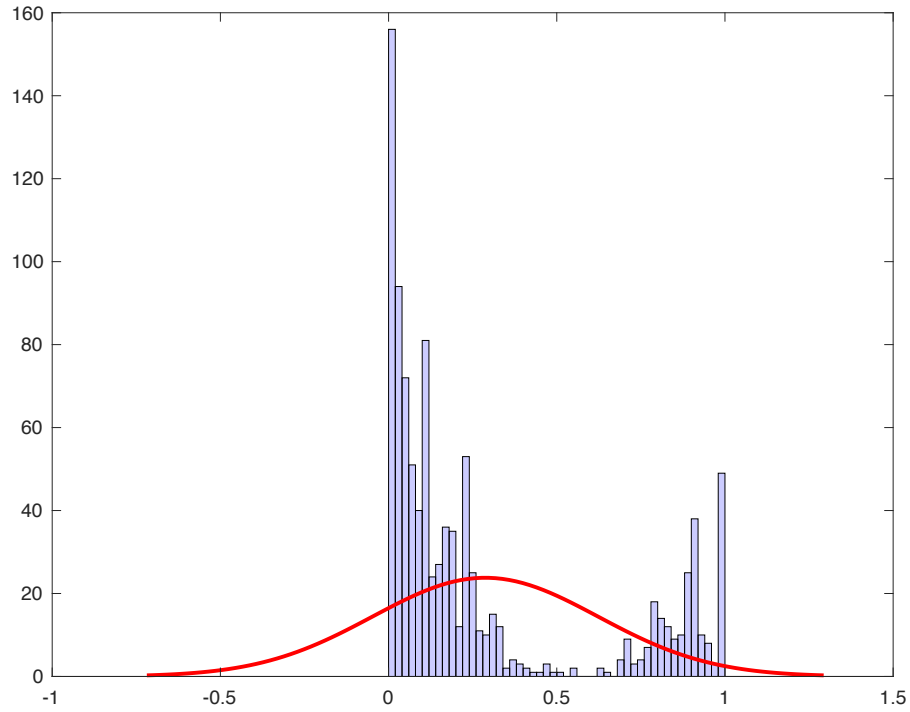


Figure 2: MH algorithm histogram start from 0

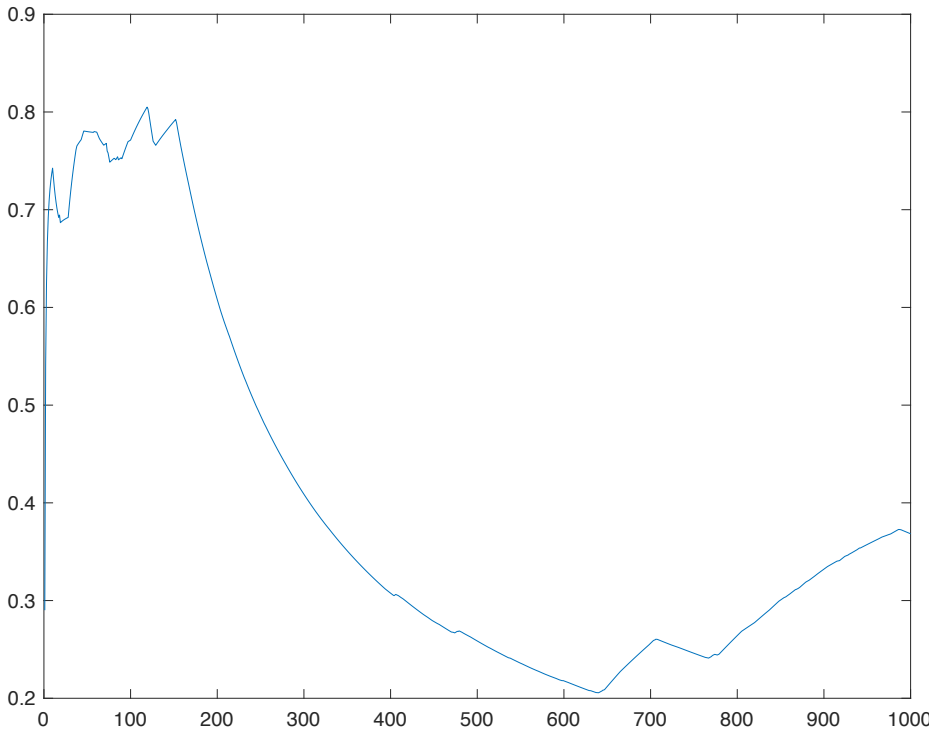


Figure 3: MH algorithm sample path gaussian pdf

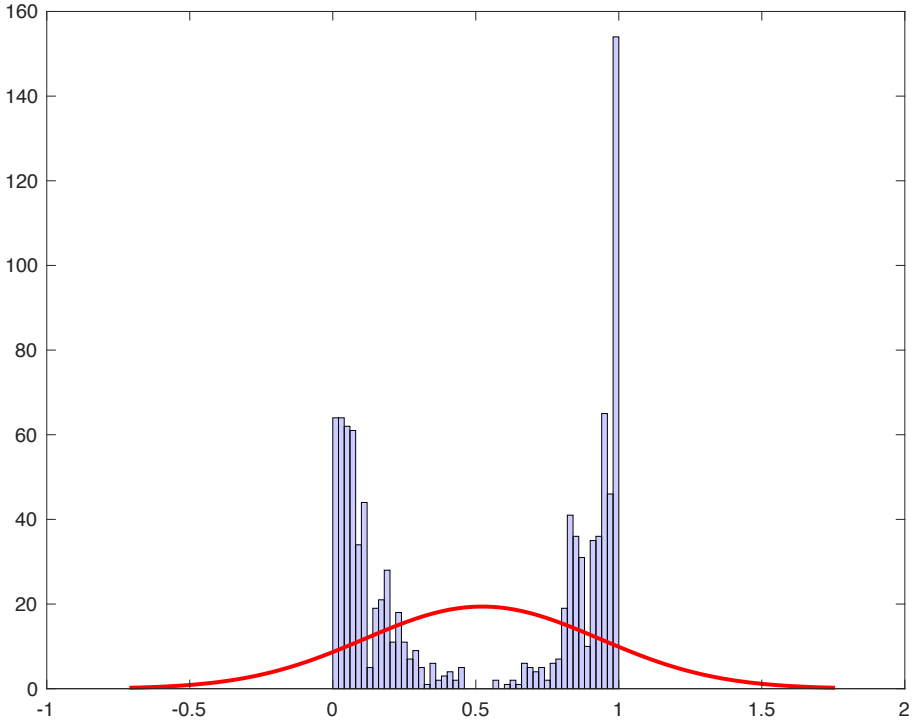


Figure 4L MH algo histogram start from 0.5

Figure 5: MH algorithm for uniformpdf (-.5,+.5)

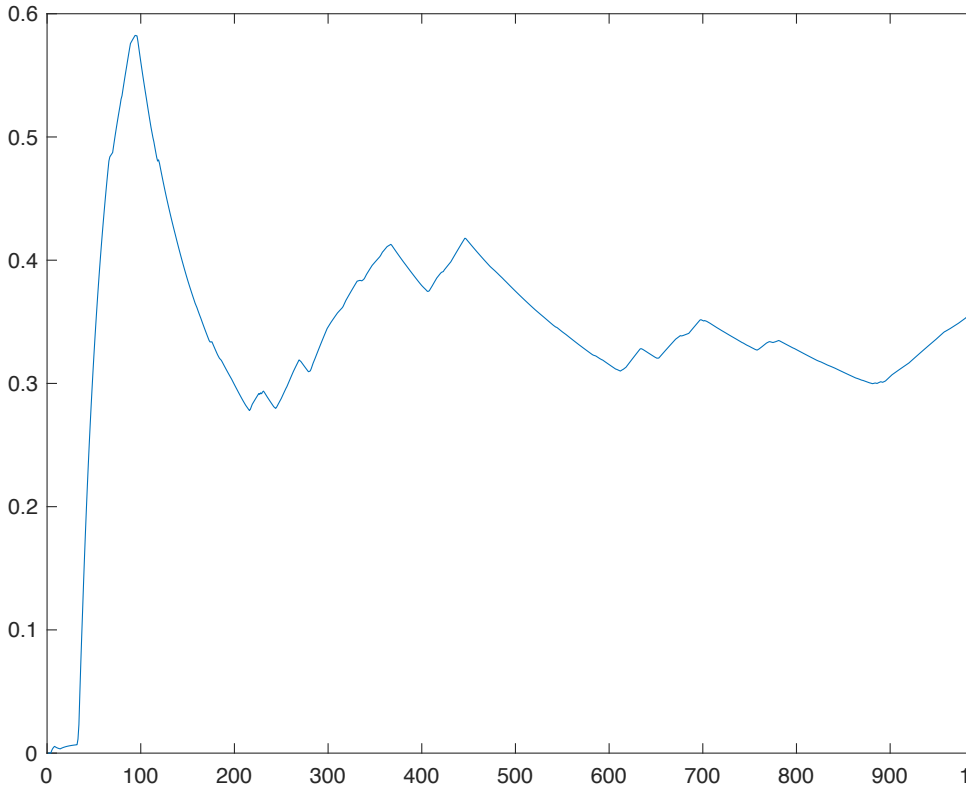


Figure 6: MH algorithm sample path uniform pdf

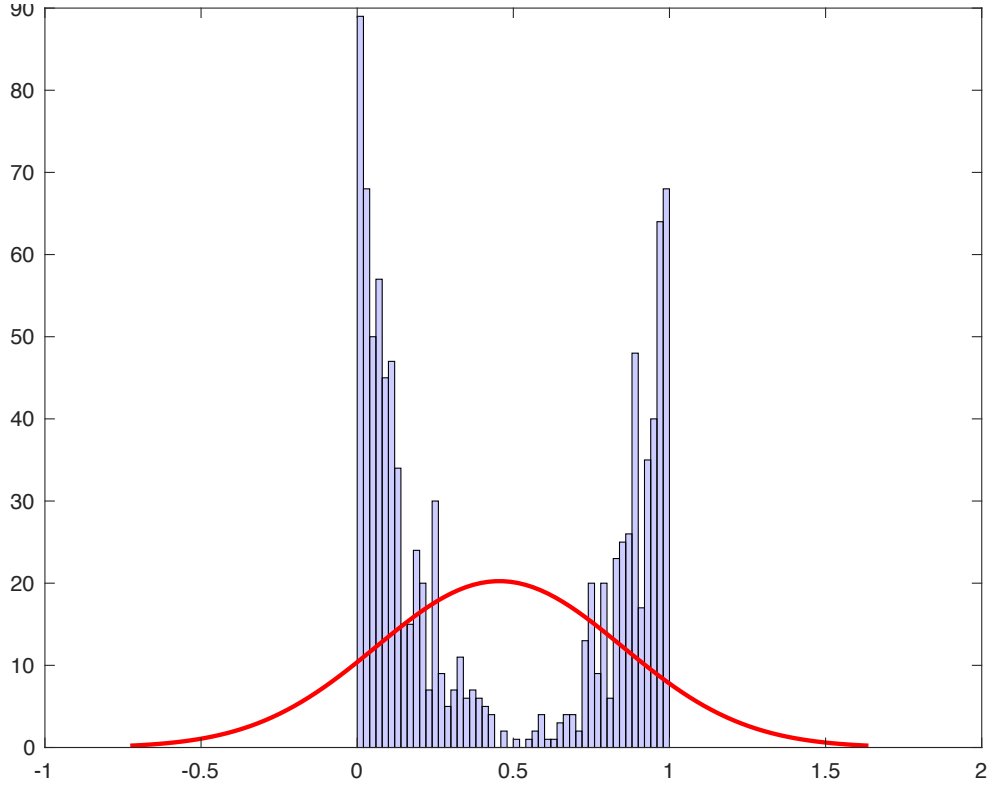


Figure 7: MH algo for uniformpdf (-.5,+.5)

Discussion:

Above code uses Metropolis-Hastings algorithm to generate samples from mixture of Beta distributions. Metropolis-Hastings algorithm simulates samples from a probability distribution. It makes use of the full joint density function and proposal distributions for each variable user interested in.

The main loop of the algorithm consists of generating a proposal sample from the proposal distribution. Then we check the probability of acceptance of this sample by computing the acceptance function. Acceptance function is crucial in the sense of it forces the sampling method to visit higher probability areas under full density. The proposal distribution might be symmetric or asymmetric. If the proposal distribution is symmetric, this procedure is known as 'Random-walk MH algorithm'. In my case, I used symmetric proposal pdf (Gaussian) as my proposal pdf.

I run the algorithm with two different starting points. One starting from 0 and the other starting from 0.5. We can easily see that starting from zero gave more samples that are accepted around zero. But when we start from 0.5 we don't get too many samples for that region this is because the original distribution is less dense within that region. We can say that the algorithm converges to it's equilibrium distribution more rapidly if we start from a initially denser point. For starting from 0.5 it took 3.861320 seconds for it to converge, whereas starting from 0 took 3.308173.

In terms of change in variances, it took 5.235680 seconds to converge for the algorithm with Gaussian pdf and variance=100. Whereas it took 2.738232 seconds for same pdf with variance=1. If the problems get more complicated and samples are increased, the differences between these computation times are also going to get increased.

[MCMC for Optimization]

i.

```
[X,Y] = meshgrid(-500:0.5:500,-500:0.5:500);
Z= (418.9829*2-(X.*sin(sqrt(abs(X)))+Y.*sin(sqrt(abs(Y)))));
C = X.*Y;
surf(X,Y,Z,C)
figure
contour(X,Y,Z)
```

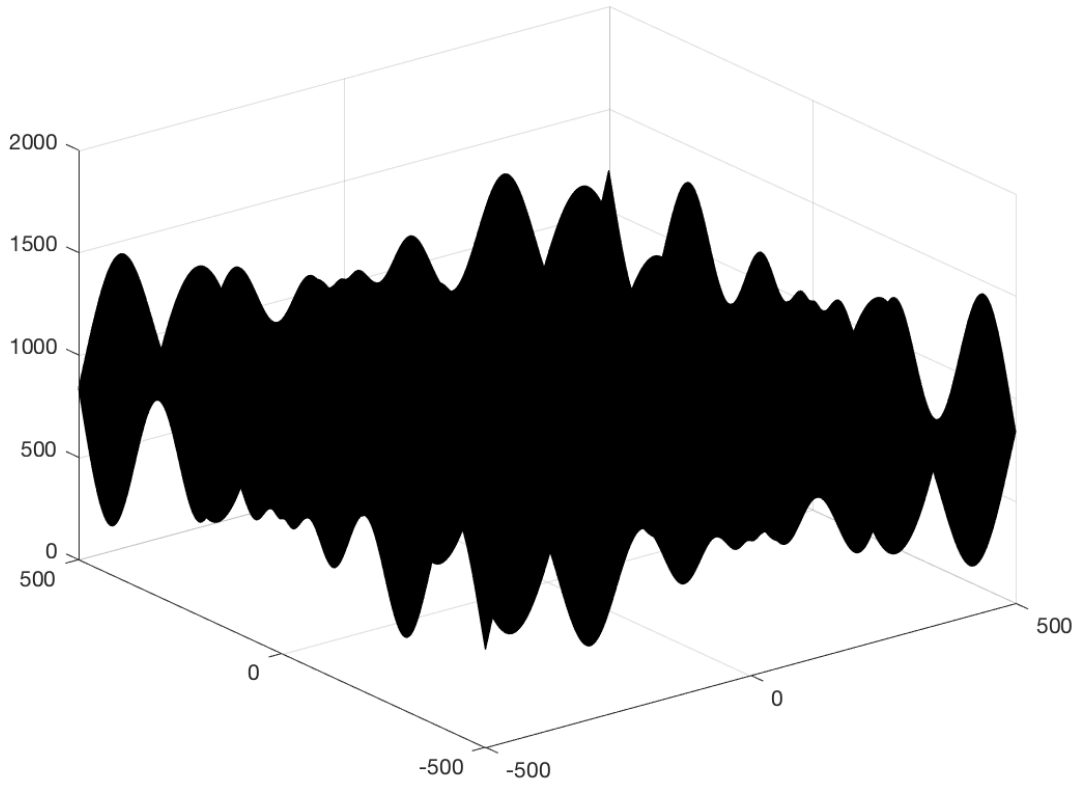


Figure 8: Surface plot of 2D Schwefel

Above the surface plot is generated. Even though I tried to implement colormap, somehow, I could not able to show it. But, it is clear that the surface is very bumpy and it has lots of local maxima, minima such that classical optimization methods would take much longer to converge to a global optimal solution. That is why we implement MCMC method.

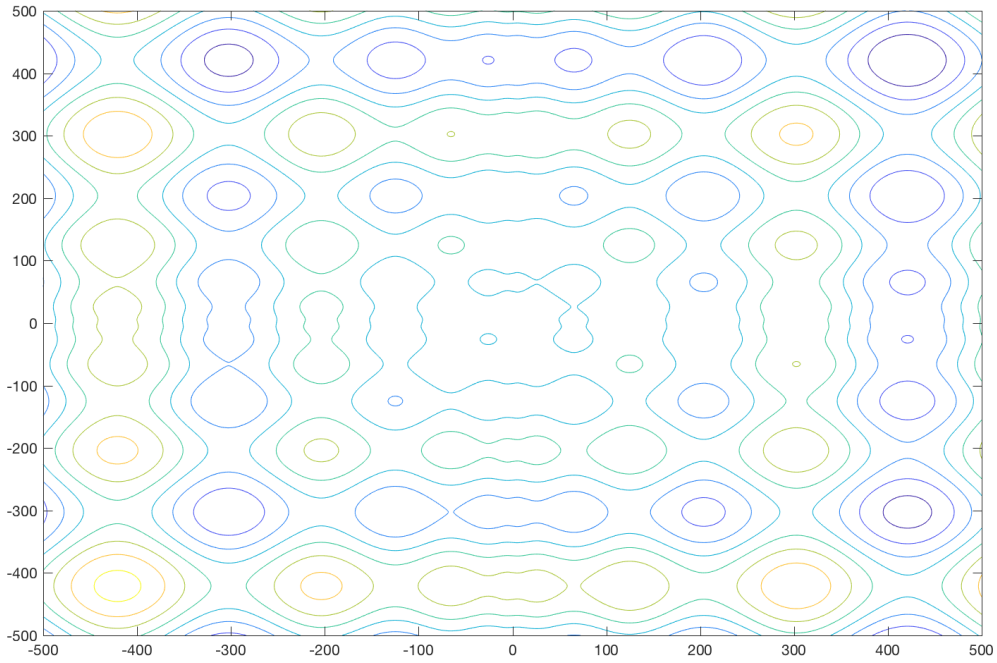


Figure 9: Schwefel 2D contour plot

Contour plot generated above also gives an idea about level surfaces of the 2-D Schwefel function.

ii.

```
%---------------Simulated Annealing function implementation%--------------$

function [minimum,fval] = anneal(loss, parent, options)

def = struct(...
        'CoolSched',@(T) (log(T)),...
        'Generator',@(x) (-500 + (500-(-500)).*rand(1,2)),...
        'InitTemp',1,...
        'MaxConsRej',1000,...
        'MaxSuccess',20,...
        'MaxTries',300,...
        'StopTemp',1e-8,...
        'StopVal',-Inf,...
        'Verbosity',1);

% Check input
if ~nargin %user wants default options, give it and stop
    minimum = def;
    return
elseif nargin<2 %user gave only objective function, throw error
    error('MATLAB:anneal:noParent','You need to input a first guess.');
elseif nargin<3 %user gave no options structure, use default
    options=def;
else %user gave all input, check if options structure is complete
    if ~isstruct(options)
        error('MATLAB:anneal:badOptions',...
            'Input argument ''options'' is not a structure')
    end
```

```
    fs = {'CoolSched','Generator','InitTemp','MaxConsRej',...
        'MaxSuccess','MaxTries','StopTemp','StopVal','Verbosity'};
    for nm=1:length(fs)
        if ~isfield(options,fs{nm}), options.(fs{nm}) = def.(fs{nm}); end
    end
end

% main settings
newsol = options.Generator;      % neighborhood space function
Tinit = options.InitTemp;        % initial temp
minT = options.StopTemp;         % stopping temp
cool = options.CoolSched;        % annealing schedule
minF = options.StopVal;
max_consec_rejections = options.MaxConsRej;
max_try = options.MaxTries;
max_success = options.MaxSuccess;
report = options.Verbosity;
k = 1;                           % boltzmann constant

% counters etc
itry = 0;
success = 0;
finished = 0;
consec = 0;
T = Tinit;
initenergy = loss(parent);
oldenergy = initenergy;
total = 0;
if report==2, fprintf(1,'\n  T = %7.5f, loss = %10.5f\n',T,oldenergy); end

while ~finished
    itry = itry+1; % just an iteration counter
    current = parent;

    % % Stop / decrement T criteria
    if itry >= max_try || success >= max_success
        if T < minT || consec >= max_consec_rejections
            finished = 1;
            total = total + itry;
            break;
        else
            T = cool(T);  % decrease T according to cooling schedule
            if report==2 % output
                fprintf(1,'  T = %7.5f, loss = %10.5f\n',T,oldenergy);
            end
            total = total + itry;
            itry = 1;
            success = 1;
        end
    end

    newparam = newsol(current);
    newenergy = loss(newparam);

    if (newenergy < minF),
        parent = newparam;
        oldenergy = newenergy;
        break
    end

    if (oldenergy-newenergy > 1e-6)
        parent = newparam;
```

```
        oldenergy = newenergy;
        success = success+1;
        consec = 0;
    else
        if (rand < exp( (oldenergy-newenergy)/(k*T) ))
            parent = newparam;
            oldenergy = newenergy;
            success = success+1;
        else
            consec = consec+1;
        end
    end
end

minimum = parent;
fval = oldenergy;

if report
    fprintf(1, '\n  Initial temperature:     \t%g\n', Tinit);
    fprintf(1, '  Final temperature:       \t%g\n', T);
    fprintf(1, '  Consecutive rejections:  \t%i\n', consec);
    fprintf(1, '  Number of function calls:\t%i\n', total);
    fprintf(1, '  Total final loss:        \t%g\n', fval);
end

%---------------Simulated Annealing function implementation%--------------$




%---------------Using simulated annealing function--------------$


camel = @(x,y)(418.9829*2-(x*sin(sqrt(abs(x)))+y*sin(sqrt(abs(y)))));
loss = @(p)camel(p(1),p(2));
[mini, fvali] = anneal(loss,[0,0]);

%---------------Using simulated annealing function--------------$
```

mini =

  420.7000, 429.4613

fvali =

    9.1068

iii.

To change the cooling schedule, we modify the default cooling function of the anneal.m as exponential, polynomial, and logarithmic respectively. For each cooling function we change the initial temperature as [20,50,100,1000] from the same function. To impement different cooling functions, the reference website is used to generate relevant cooling schedules.

Exponential Cooling Function: `T*(0.8)^(itry) %% itry is the iteration number`

This cooling function is proposed by Kirkpatrick, Gelatt and Vecchi (1983) for the exponential cooling schedule.

```
camel = @(x,y)(418.9829*2-(x*sin(sqrt(abs(x)))+y*sin(sqrt(abs(y)))));
loss = @(p)camel(p(1),p(2));
minn=zeros(100,2);
fval=zeros(100,1);
for k=1:1:100
[mini, fvali] = anneal(loss,[0,0]);
minn(k,1)=mini(1);
minn(k,2)=min(2);
fval(k)=fvali;
end

[m,I]=min(fval);
min_pair(1)=minn(I,1);
min_pair(2)=minn(I,2);
histogram(fval);
```

T=20

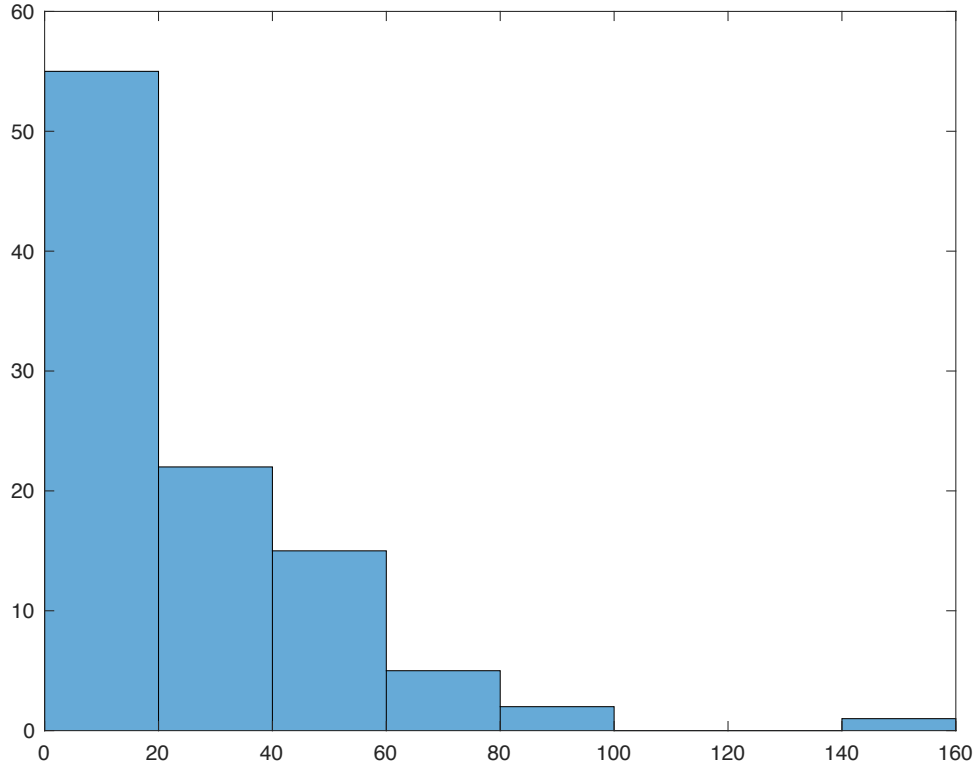


Figure 10: Histogram t=20

m = 0.0711
min_pair = 421.2102, 421.6791

T=50

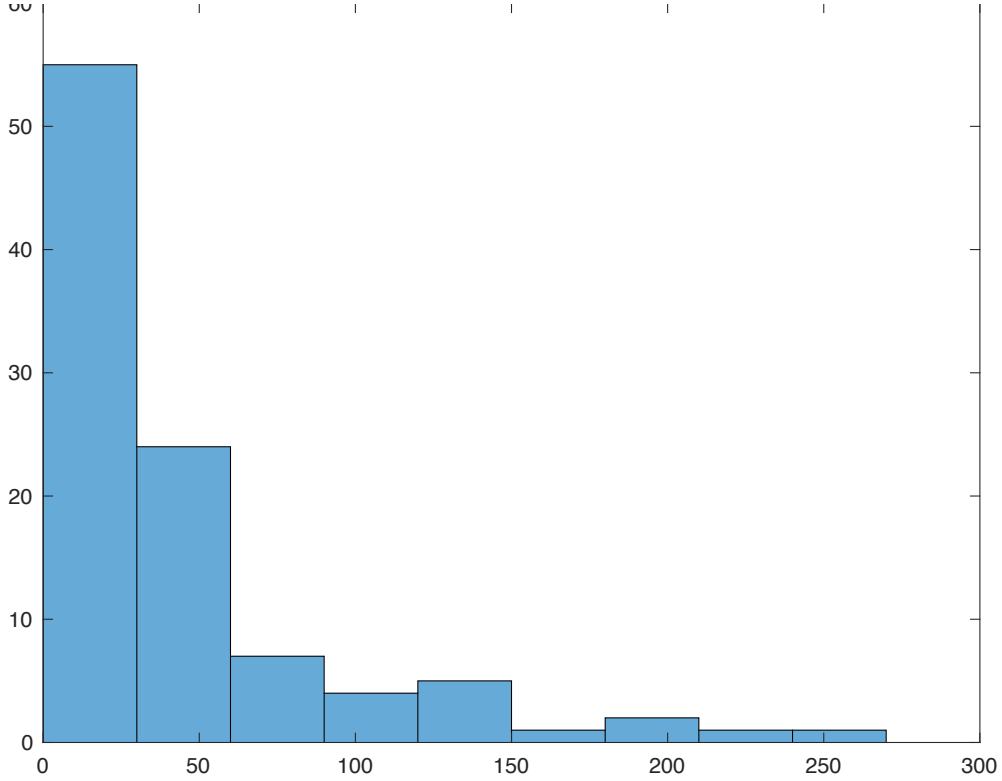


Figure 11: Histogram for t=50

m =3.0711
min_pair =419.6273 425.7147

T=100

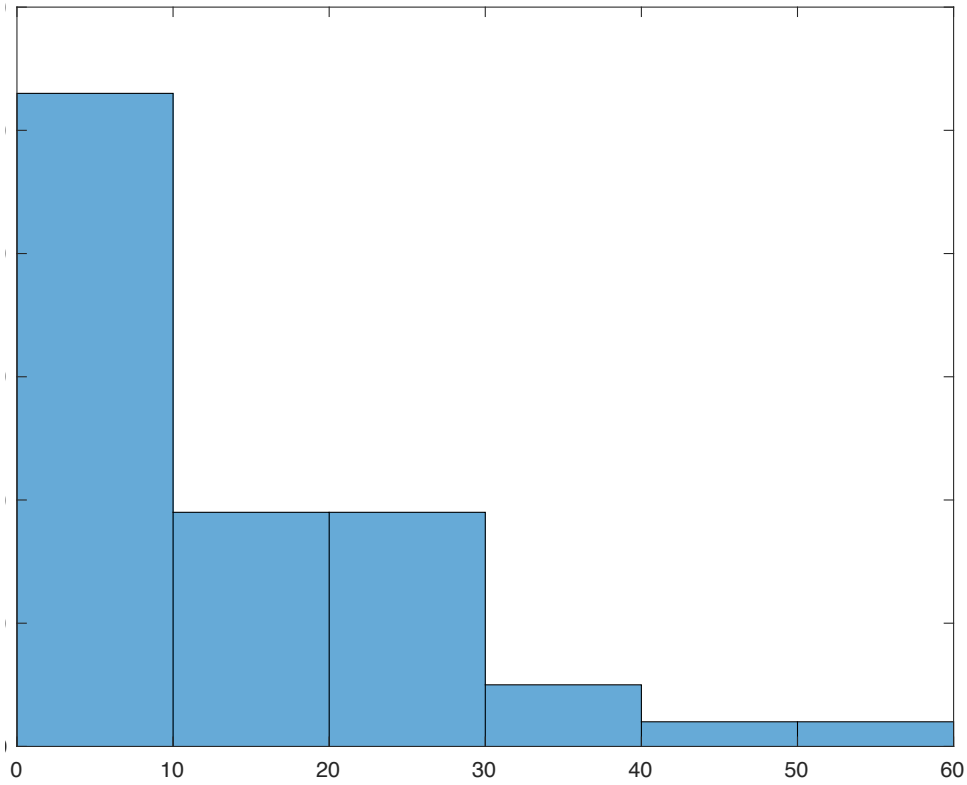


Figure 12: : Histogram for t=100

m = 0.0019
min_pair = 420.9553 421.0898

T=1000

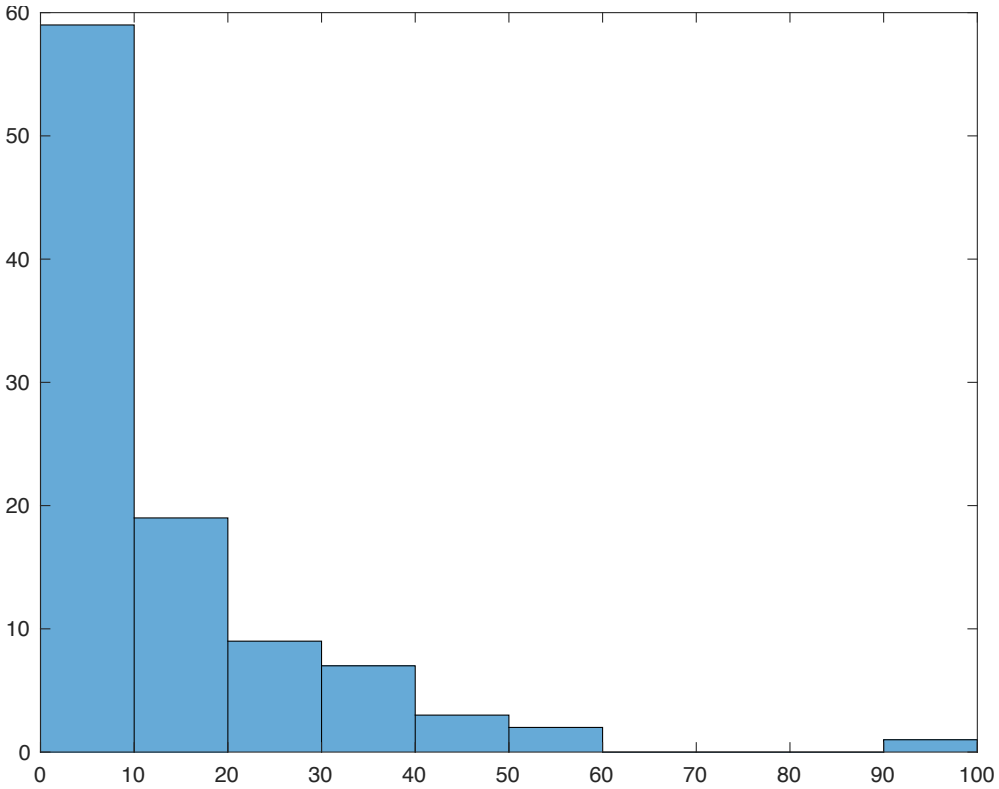


Figure 13: : Histogram for t=1000

m = 0.0247
min_pair =421.3862 420.8242

Polynomial Cooling Function: `(T/(1+(0.5*itry)))`
T=20

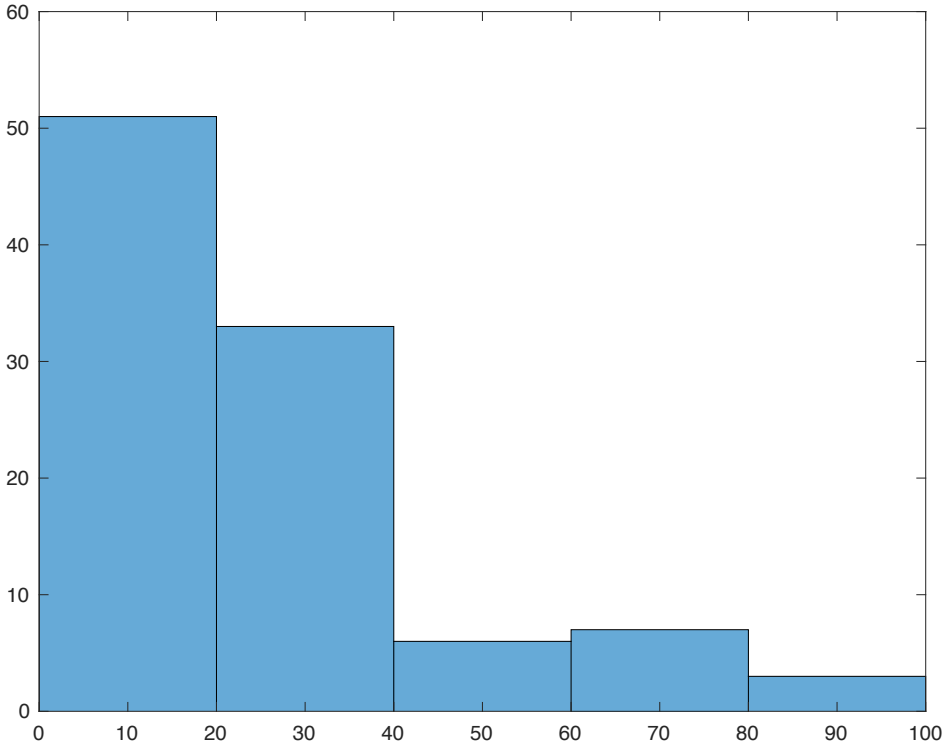


Figure 14: Histogram for t=20

m = 0.0188
min_pair=420.6103, 421.1117

T=50

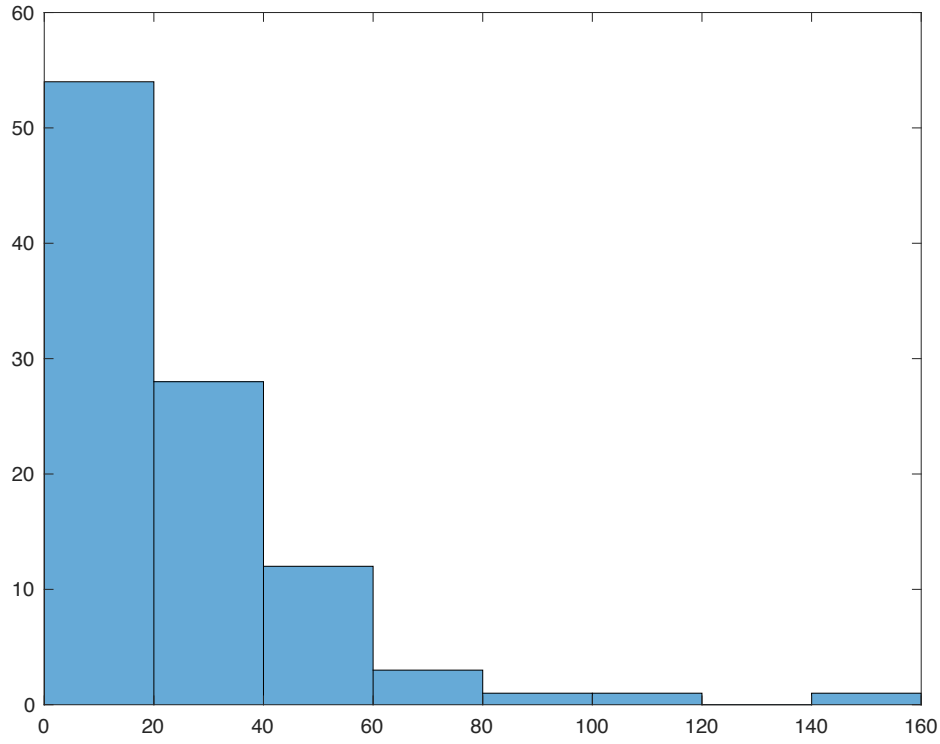


Figure 15: histogram for t=50

m =0.0110
min_pair =421.1581 421.1940

T=100

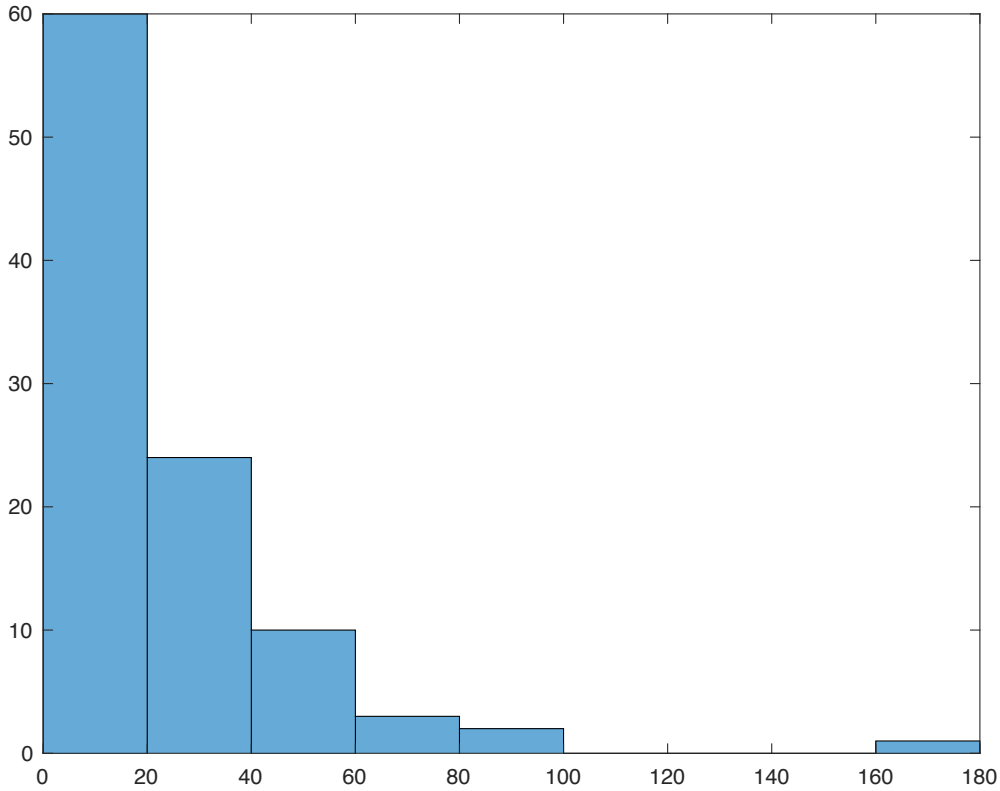


Figure 16: histogram for t=100

m =0.4770
min_pair = 421.6044 419.1305

T=1000

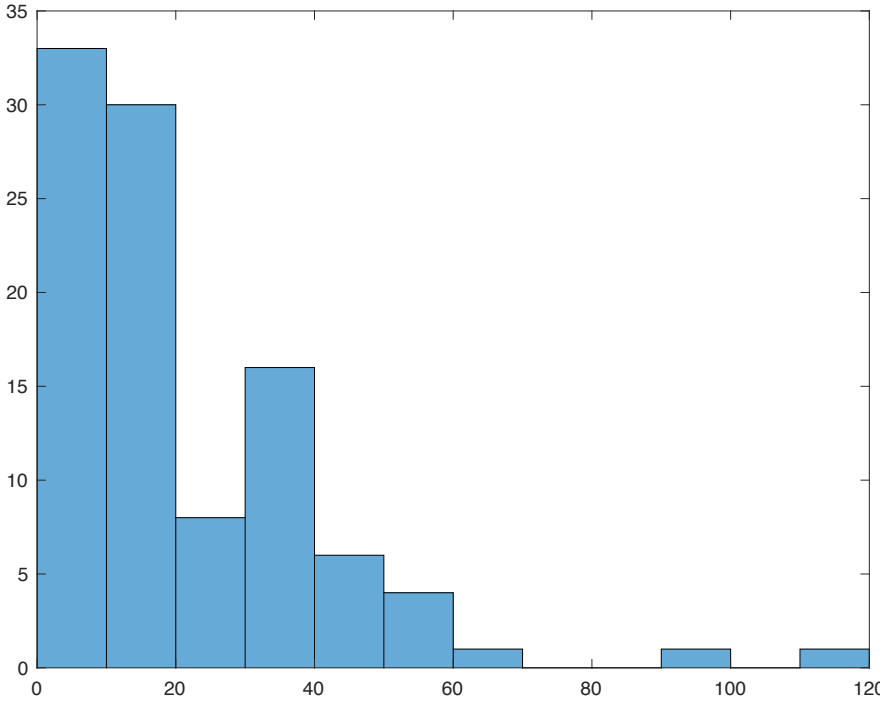


Figure 17: Histogram for t=1000

m =0.4909
min_pair = 422.4599 422.2591

Logarithmic Cooling Function: `(5/(log(1+T)))`
T=20

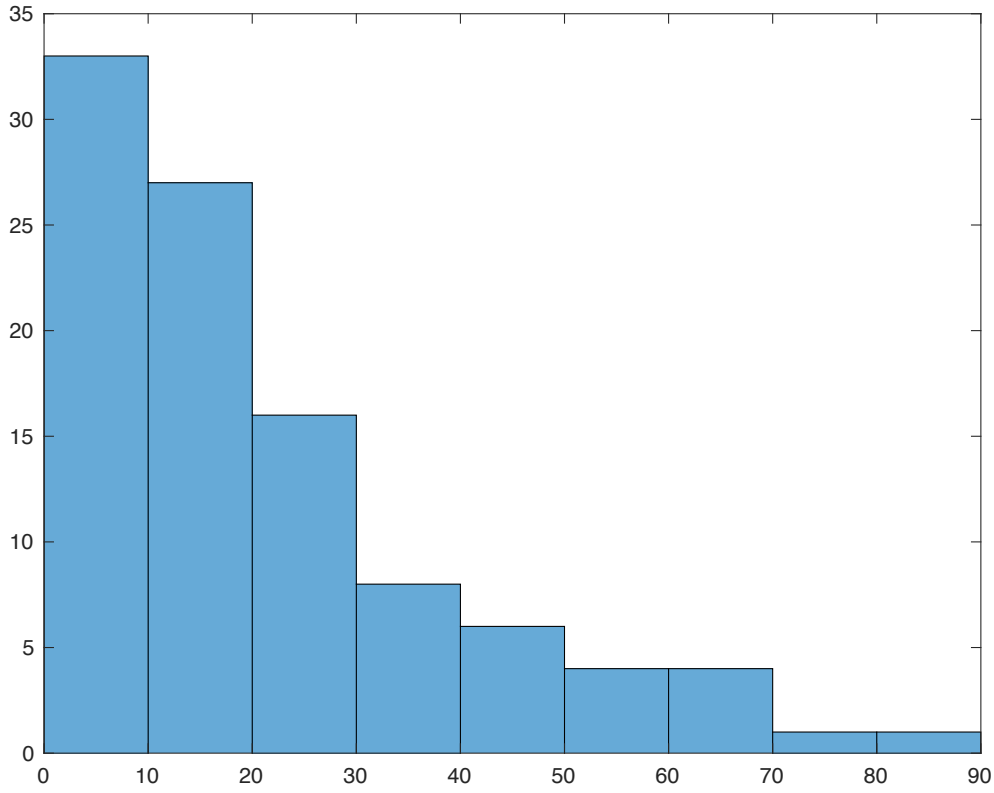


Figure 18: Histogram for t=20

m = 0.3245
min_pair = 421.8492 419.6283

T=50

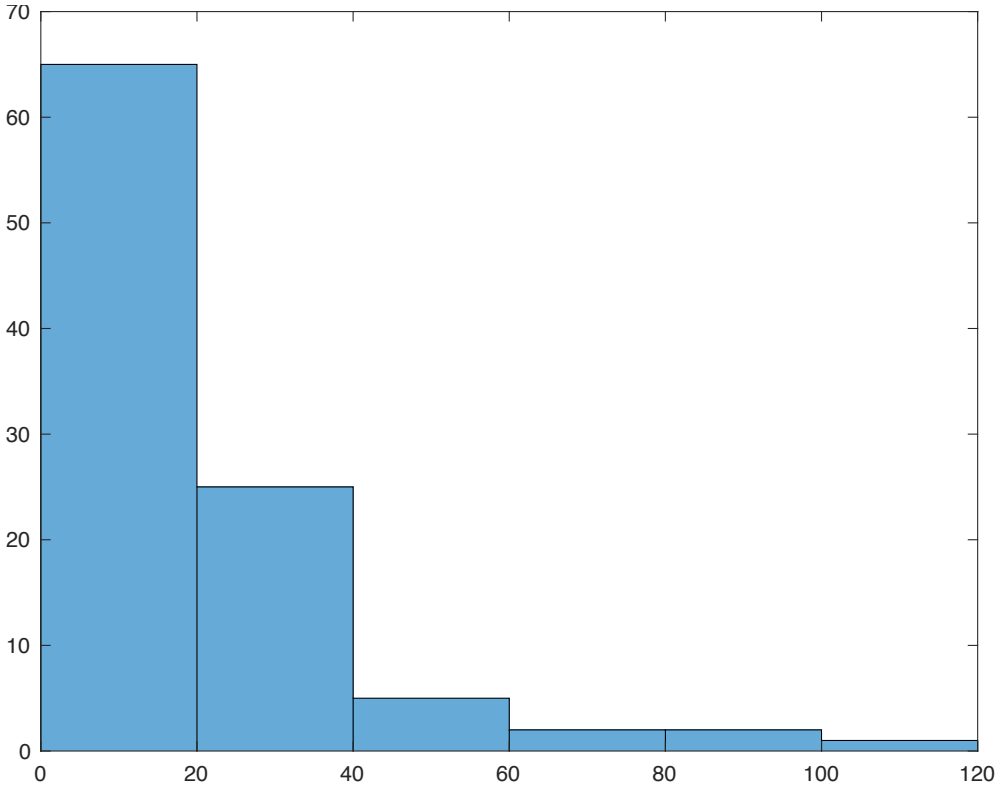


Figure 19: Histogram for t=50

m = 0.0096
min_pair =421.0148 420.6971

T=100

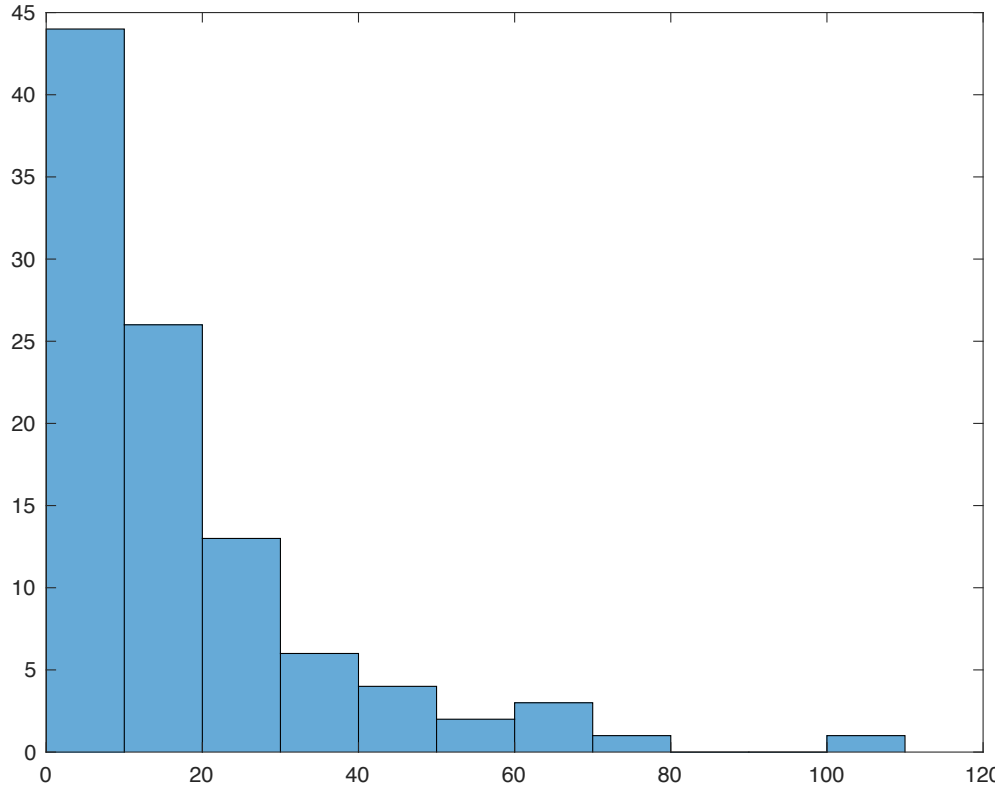


Figure 20: histogram for t=100

m = 0.1115
min_pair =420.0286 420.9830

T=1000

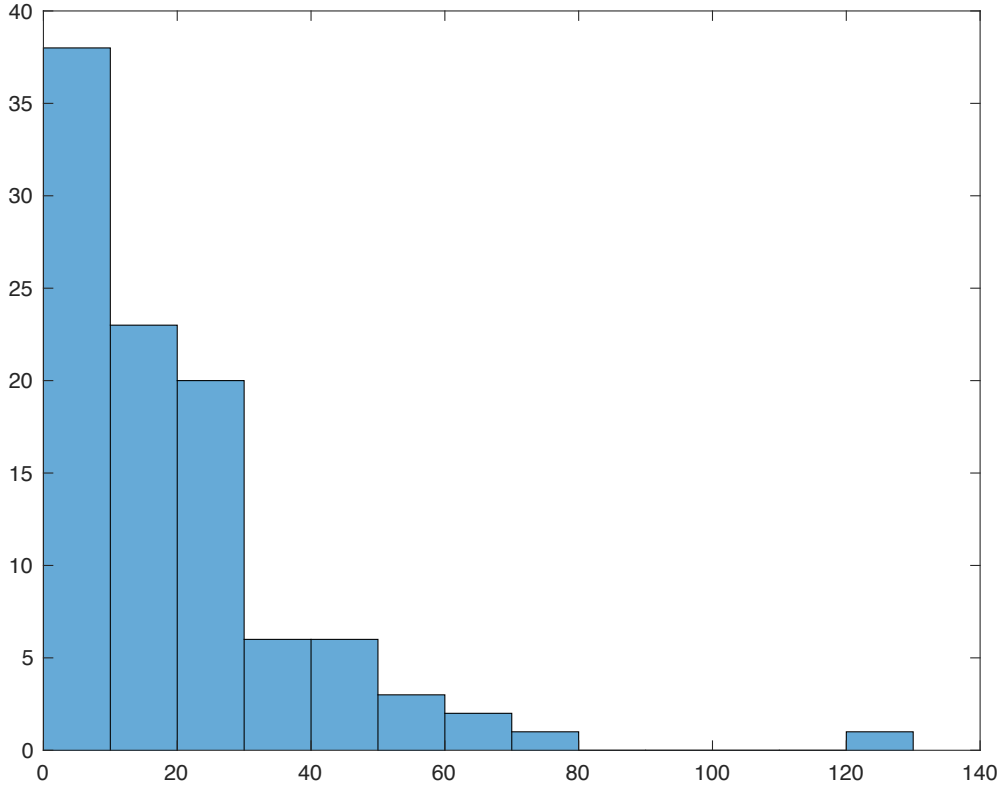


Figure 21: histogram for t=1000

m = 0.0730
min_pair =421.5877 420.5273

iv.

Looking at the histograms and the minimum values achieved by the algorithms, the minimum value achieved was **m = 0.0019** with x and y pairs as **420.9553 421.0898** using exponential cooling schedule with initial **T=100**. The solution pairs on top of the contour is shown below.

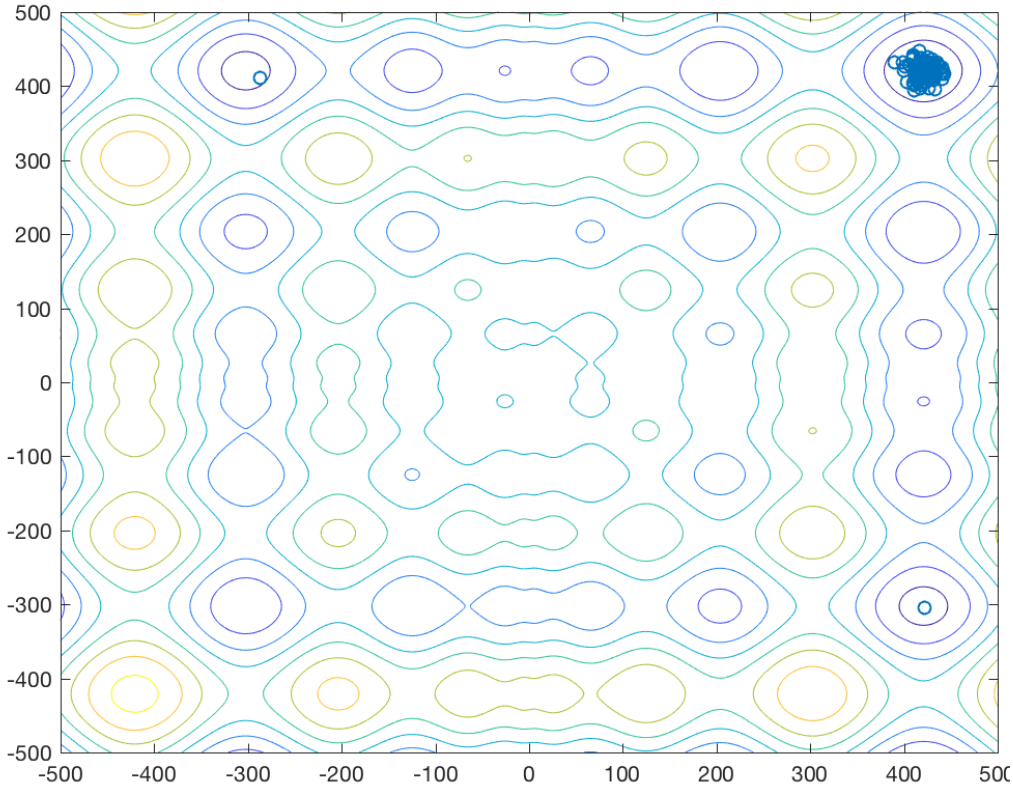


Figure 22: Contour plot with the pairs

[Optimal Paths]

```
function simulatedAnnealingTSP(no_cities, temperature_start,
cooling_factor)

cities_mat = unifrnd(0, 1000, no_cities, 2);
% matrix of city coordinates

distance_mat = squareform(pdist(cities_mat));
% matrix of intercity distances

temperature = temperature_start;
% process temperature variable

distance_prev = Inf;
% set initial tour distance to infinity

counter = 0;
% iteration counter variable

random_tour = randperm(no_cities);
% create initial tour as a random combination of cities

%% simulated annealing process as long as temperature is positive

while temperature > 1

    % below code randomly swaps to cities in the tour. This it the
    % most efficient stochastic method of simulating all possible tours.
```

```
    tour_position1 = round(unifrnd(1, no_cities));
    tour_position2 = round(unifrnd(1, no_cities));
    % randomly select two tour positions from current random tour

    swap_city1 = random_tour(tour_position1);
    swap_city2 = random_tour(tour_position2);
    % extract the cities at these random positions

    random_tour(tour_position1) = swap_city2;
    random_tour(tour_position2) = swap_city1;
    % swap the cities

    tour_distance = 0;
    % reinitialize the tour distance for each iteration

    % calculate the sum of intercity distance along the current random
    % route
    for i = 1 : no_cities - 1
        tour_distance = tour_distance + distance_mat(random_tour(i),
random_tour(i + 1));
    end

    difference = tour_distance - distance_prev;
    % calculate the differnece between current random tour and previous
    % random tour

    % calculate the acceptance probability of the computed tour distance
    % accoridng to exp(-difference/temperature) > rand(0, 1). If either
    % condition below holds true, set the distance as current optimal, and
    % tour as current optimal tour.

    if(difference < 0 | exp(-difference / temperature) > rand(0, 1))
        distance_prev = tour_distance;
        optimal_tour = random_tour;
    end

    counter = counter + 1;
    % counter variable to track iterations

    temperature_vec(counter, :) = temperature;
    % vector tracks annealing temperature at each iteration

    temperature = temperature * (1 - cooling_factor);
    % cool the process by discounting temperatuee by cooling factor

    optim_distance(counter, :) = distance_prev;
    calc_distance(counter, :) = tour_distance;
    % vectors to store the current optimal distance as well as currently
    % computed distance
end

%% Plot of results

plot(1: 1: counter, temperature_vec, '-', 'Linewidth', 2);
xlabel('Number of Steps');
ylabel('Temperature');
title('Decay of Annealing Temperature');
% plot of annealing temperature vs iterations

figure;
```

```
plot(1: 1: counter, optim_distance, '-', 'Linewidth', 2); hold on;
plot(1: 1: counter, calc_distance, '-', 'Linewidth', 2);
legend;
xlabel('Number of Steps');
ylabel('Distance');
title('Simulated Annealing of Traveling Salesman Problem');
% Plot of calculated distcance and currently optimal distance vs.
% iterations

figure;
scatter(cities_mat(1 : no_cities, 1), cities_mat(1 : no_cities, 2), 'o');
hold on;
plot(cities_mat(optimal_tour, 1), cities_mat(optimal_tour, 2), '-'); grid
on;
xlabel('Coordinates');
ylabel('Coordinates');
title('Optimal Tour of Traveling Salesman Problem');
% plot of city coordinates and optimal route
```

Set of random tours evaluated at each iteration, evaluated distance is compared to the previous tour distance. If the distance is less it is assumed optimal, but since we are using simulated annealing procedure if the distance is more it is not disregarded. It is held off with a certain probability. I am using exponential cooling schedule since it performed best for the MCMC for Optimization problem.

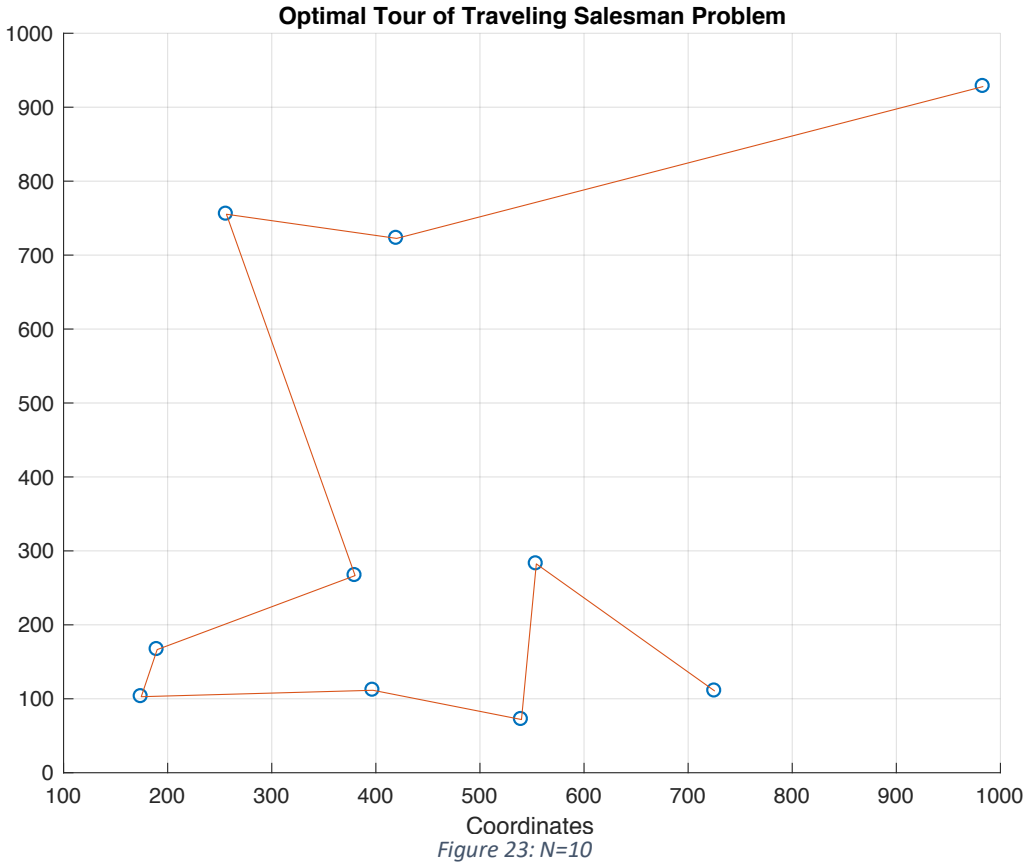


Figure 23: N=10

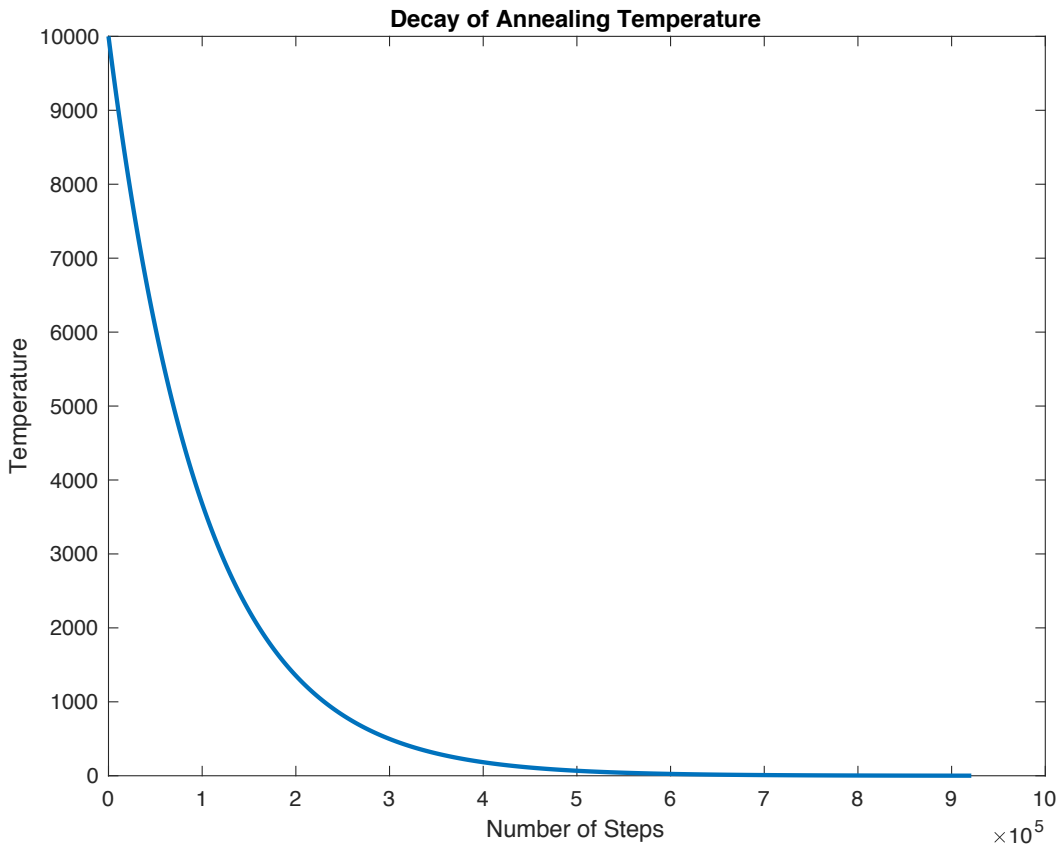
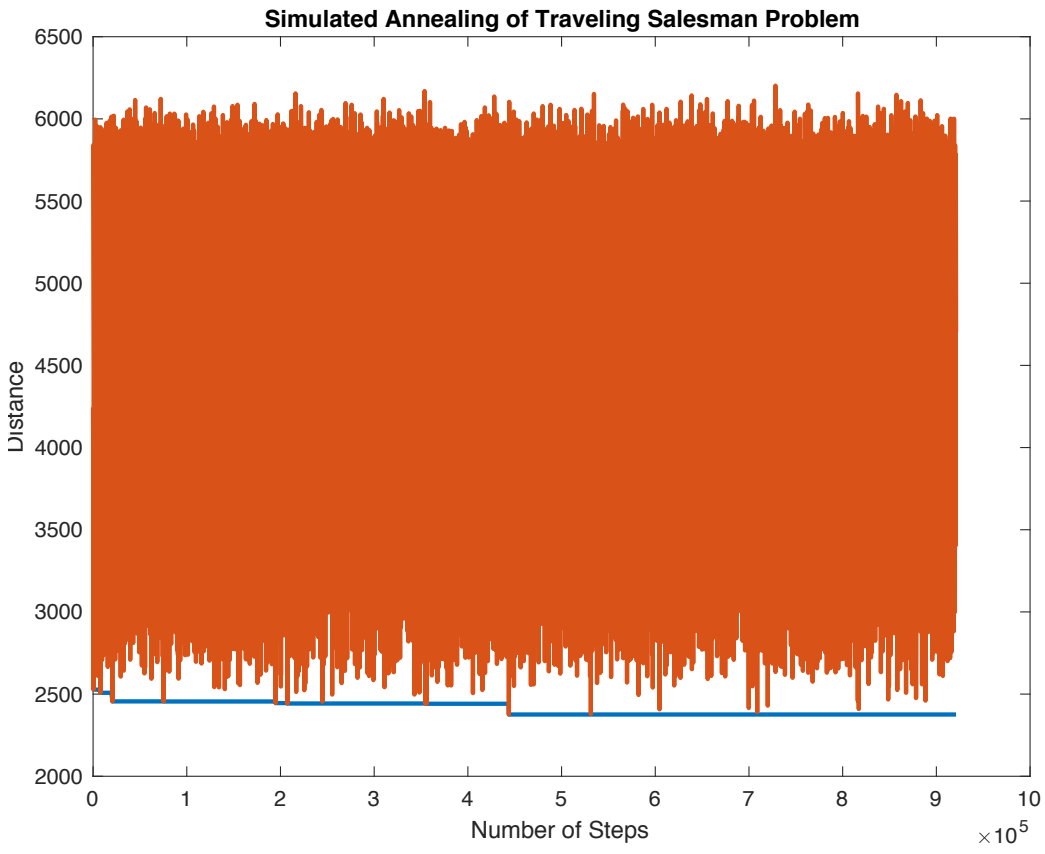


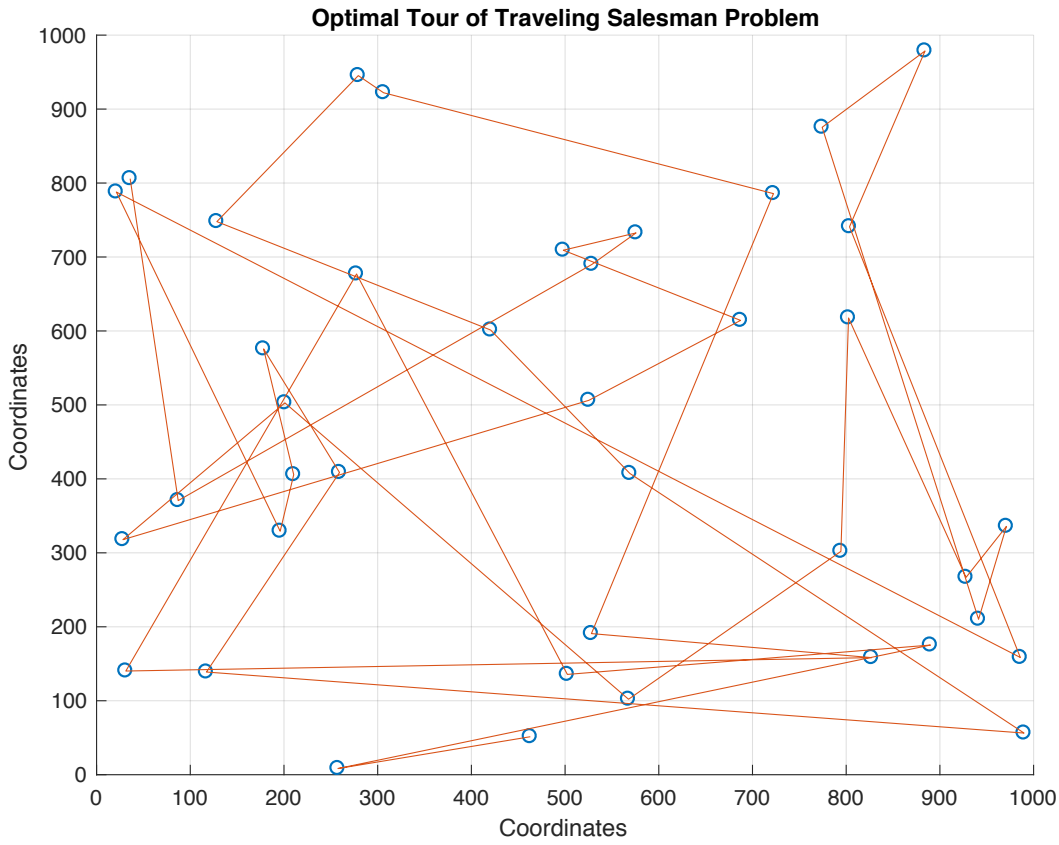
Figure 24: temperature decay for N=10

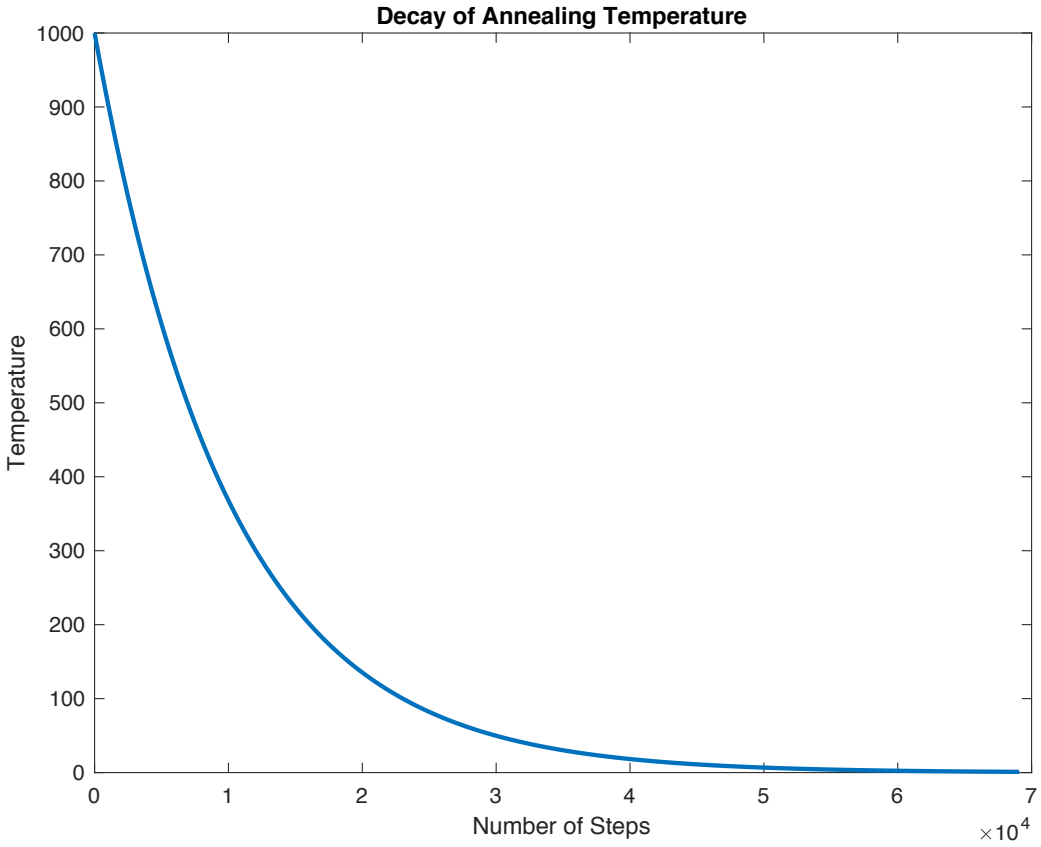


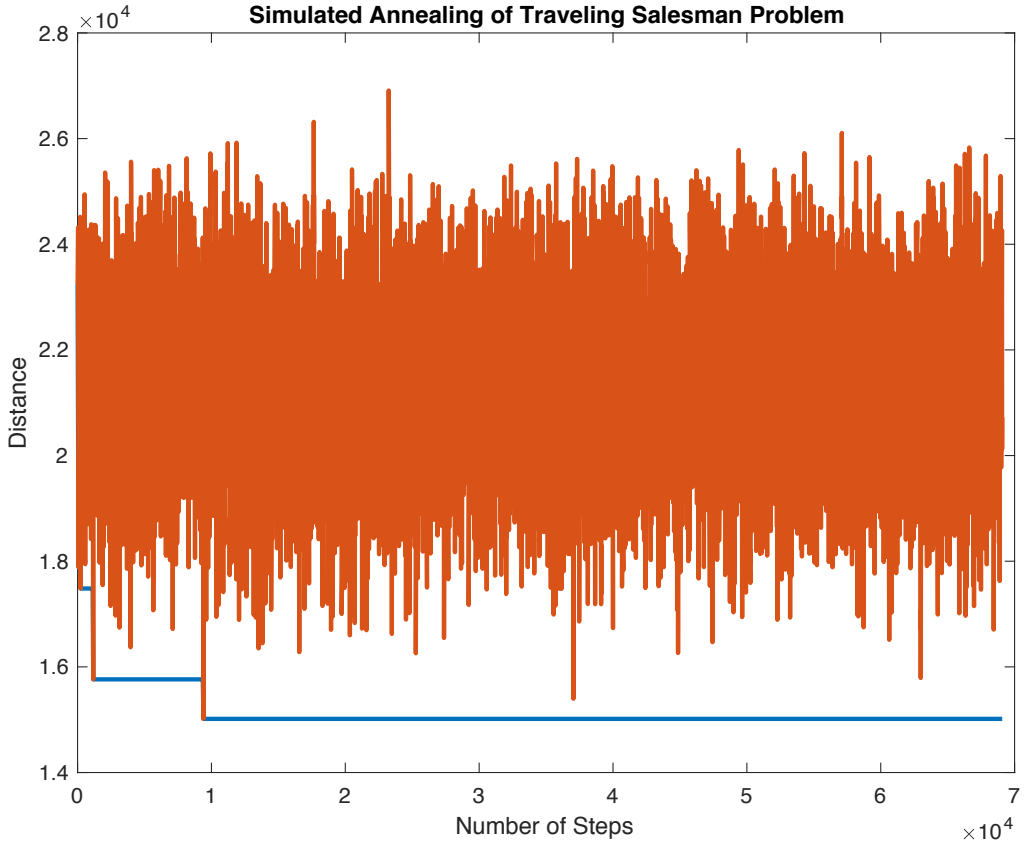
When I choose N=10, Initial Temperature as 10000 and decay rate as 0.00001 with exponential cooling schedule. For N=40, 400, 1000 the elapsed time for the computation of the algorithm is given below.

Elapsed time is 11.784691 seconds. N=10
>> annealingtest
Elapsed time is 11.872613 seconds. N=40
>> annealingtest
Elapsed time is 21.178397 seconds. N=400
>> annealingtest
Elapsed time is 32.370761 seconds N=1000

It is clearly seen that the number of cities increases the computation time significantly.
The optimal tour, Decay of the temperature and the calculated distances are plotted below for N=40.




Decay of Annealing Temperature
Temperature
Number of Steps
×10^4
1000
900
800
700
600
500
400
300
200
100
0
0
1
2
3
4
5
6
7


Simulated Annealing of Traveling Salesman Problem
×10^4
Distance
Number of Steps
×10^4
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
0
1
2
3
4
5
6
7

References:

(1) Taylan Cemgil's lecture slides on Monte Carlo Methods (http://www.cmpe.boun.edu.tr/courses/cmpe58n/fall2009/)

(2) http://what-when-how.com/artificial-intelligence/a-comparison-of-cooling-schedules-for-simulated-annealing-artificial-intelligence/

(3) https://de.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/10548/versions/1/previews/anneal.m/index.html?access_key=

(4) https://github.com/kufer/Traveling-Salesman-Problem-by-Simulated-Annealing/blob/master/simulatedAnnealingTSP.m