University of Southern California

EE511 Simulation Methods for Stochastic Systems

Project #8- Markov Chain Monte Carlo

BY

Mohan Krishna Thota

USC ID: 6683486728

mthota@usc.edu

PROBLEM 1:

DESCRIPTION:

- Monte Carlo is the art of approximating an expectation by the sample mean of a function of simulated random variables.
- Monte Carlo is about invoking laws of large numbers to approximate expectations.
- Importance sampling is choosing a good distribution from which to simulate one's random variables.
- It involves multiplying the integrand by 1to yield an expectation of a quantity that varies less than the original integrand over the region of integration.
- As a final word on importance sampling, the tails of the distributions matter! While h(x) might be roughly the same shape as g(x), serious difficulties arise if h(x) gets small much faster than g(x) out in the tails.
- In such a case, though it is improbable, if you do, then your Monte Carlo estimator will take a jolt such an g(x)/h(x) that you see.

PROCEDURE:

- I have generated the two random numbers using **rand()** function.
- The question is asking to conduct the experiment in two parts for the calculation of mean and variance, then calculate the simple monte carlo using stratified sampling and the importance sampling for the series of random numbers.

CODE:

```
clc;
clear all;
close all;
N = 1000;      % No of samples

% Estimating mean and variance using simple Monte Carlo and Stratified
% Sampling

g = @(x1,x2)exp(5.*abs(x1-5) + 5.*abs(x2-5));
a = -1;
b = 1;
u1 = rand(1,N);
u2 = rand(1,N);
X = g(u1,u2);   % Simple Monte Carlo
display('Mean and Variance using simple Monte Carlo');
disp(mean(X));
disp(2*std(X)/sqrt(N));
K = 20; Nij = N/K;      % With Stratified sampling
for i = 1:K
   for j = 1:K
      XS = g((i-1+rand(1,Nij))/K,(j-1+rand(1,Nij))/K);
      XSb(i,j) = mean(XS);
      SS(i,j) = var(XS);
   end
end
SST = mean(mean(SS/N));
display('Mean and Variance using Stratified Sampling');
disp(mean(mean(XSb)));
disp(2*sqrt(SST));

% Estimating mean and variance using Importance
% Sampling

e = exp(1);     % Importance sampling
u3 = rand(1,N);
u4 = rand(1,N);
X1 = log(1+(e-1)*u3);
X2 = log(1+(e-1)*u4);
T = (e-1)^2*exp(5.*abs(X1-5) + 5.*abs(X2-5)-(X1+X2));
display('Mean and Variance using Importance Sampling');
disp(mean(T));
disp(2*std(T)/sqrt(N));
pl = 0;
pu = 1;
ql = 0;
qu = 1;
fun = @(p,q) exp(5.*abs(p-5) + 5.*abs(q-5)); % theoretical integration of funtion(x,y)
c = integral2 (@(p,q)fun(p,q),pl,pu,ql,qu);
disp('Theoretical integral value');
disp(c);
```

1(B).

```
N = 1000;
a = -1;
b = 1;
X12 = rand(1,N); X22=rand(1,N);
V=(b-a)*(b-a)*cos(pi + 5*(a + (b-a)*X12) + 5*(a + (b-a)*X22));
display('Mean and Variance using simple Monte Carlo');
disp(mean(V));
disp(2*std(V)/sqrt(N));
g = @(x1,x2)cos(pi + 5.*x1 + 5.*x2);

K = 20; Nij = N/K;      % With Stratified sampling
for i = 1:K
   for j = 1:K
      r1 = (b-a).*rand(1,Nij) + a;
      r2 = (b-a).*rand(1,Nij) + a;
      XS = g((i-1+r1)/K,(j-1+r2)/K);
      XSb(i,j) = mean(XS);
      SS(i,j) = var(XS);
   end
end
SST = mean(mean(SS/N));
display('Mean and Variance using Stratified Sampling');
disp(mean(mean(XSb)));
disp(2*sqrt(SST));

% Estimating mean and variance using Importance
% Sampling

e = exp(1);    % Importance sampling
r3 = (b-a).*rand(1,Nij) + a;
r4 = (b-a).*rand(1,Nij) + a;
X12 = log(1+(e-1)*r3);
X22 = log(1+(e-1)*r4);
T = (e-1)^2*cos(pi + 5.*X12 + 5.*X22-(X12+X22));
display('Mean and Variance using Importance Sampling');
disp(mean(T));
disp(2*std(T)/sqrt(N));

pl = -1;
pu = 1;
ql = -1;
qu = 1;
fun = @(p,q) cos(pi + 5*p + 5*q);  % theoretical integration of funtion(x,y)
c = integral2 (@(p,q)fun(p,q),pl,pu,ql,qu);
disp('Theoretical integral value');
disp(c);
```

RESULTS:

```
>> problem2
Mean and Variance using simple Monte Carlo
   -0.2773
    0.1793
Mean and Variance using Stratified Sampling
   -0.0039
    0.0092
Mean and Variance using Importance Sampling
  -3.8537e+09 + 5.4900e+09i
   1.8506e+09
Theoretical integral value
   -0.1471

Problem1
Mean and Variance using simple Monte Carlo
   1.9576e+20
   2.9368e+19
Mean and Variance using Stratified Sampling
   2.0542e+20
   3.3274e+18
Mean and Variance using Importance Sampling
   2.1809e+20
   5.3697e+19
Theoretical integral value
   2.0460e+20
```

ANALYSIS:

From the above estimated values, we could see that the three different estimates, Monte carlo estimates are bit closer to the theoretical integral values. And the comparision is shown above.

PROBLEM 2:

DESCRIPTION:

- In statistics, Gibbs sampling or a Gibbs sampler is a Markov chain Monte Carlo(MCMC) algorithm for obtaining a sequence of observations which are approximated from a specified multivariate probability distribution, when direct sampling is difficult.
- This sequence can be used to approximate the joint distribution (e.g., to generate a histogram of the distribution);
- to approximate the marginal distribution of one of the variables, or some subset of the variables (for example, the unknown parameters or latent variables);
- or to compute an integral (such as the expected value of one of the variables). Typically, some of the variables correspond to observations whose values are known,
- hence do not need to be sampled. Gibbs sampling is commonly used as a means of statistical inference, especially Bayesian inference.
- It is a randomized algorithm (i.e. an algorithm that makes use of random numbers) and is an alternative to deterministic algorithms for statistical inference such as the expectation-maximization algorithm (EM).
- As with other MCMC algorithms, Gibbs sampling generates a Markov chain of samples, each of which is correlated with nearby samples.
- As a result, care must be taken if independent samples are desired (typically by thinning the resulting chain of samples by only taking every nth value, e.g. every 100th value).
- In addition, samples from the beginning of the chain (the burn-in period) may not accurately represent the desired distribution.

PROCEDURE:

- I have rounded each element to the nearest integer greater than or equal to that element the two random numbers using ceil function.
- The question is asking to conduct the experiment in two parts for the calculation of mean.

CODE:

```
clc;
clear all;
disp('1st part');
n = 3;
N = 1000;
X = 2*ones(1,n);
for k = 1:N
    i = ceil(n*rand);
    S = sum(X) - i*X(i);
    X(i) = max(15 - S, 0) - log(rand)/1;
    H(k) = S + X(i);
end
disp([mean(H) 2*std(H)/sqrt(N)]);

disp('2nd part');
X = 2*ones(1,n);
for k = 1:N
    i = ceil(n*rand);
    S = sum(X) - i*X(i);
    X(i) = log(rand)/1 - max(1 - S, 0);
    H(k) = S + X(i);
end
disp([mean(H) 2*std(H)/sqrt(N)]);
```

OUTPUT:

```
1st part

   1.0e+93 *

    8.1270    5.0281

2nd part

   1.0e+07 *

   -1.1952    0.4274
```

The estimates of the Gibbs Sampling method is show above

PROBLEM 3:

DESCRIPTION:

- Schwefel's function is deceptive in that the global minimum is geometrically distant, over the parameter space, from the next best local minima.
- Therefore, the search algorithms are potentially prone to convergence in the wrong direction.

Function Definition

f7(x)=sum(-x(i)·sin(sqrt(abs(x(i)))),i=1:n;
-500<=x(i)<=500.

global minimum

f(x)=-n·418.9829; x(i)=420.9687, i=1:n.

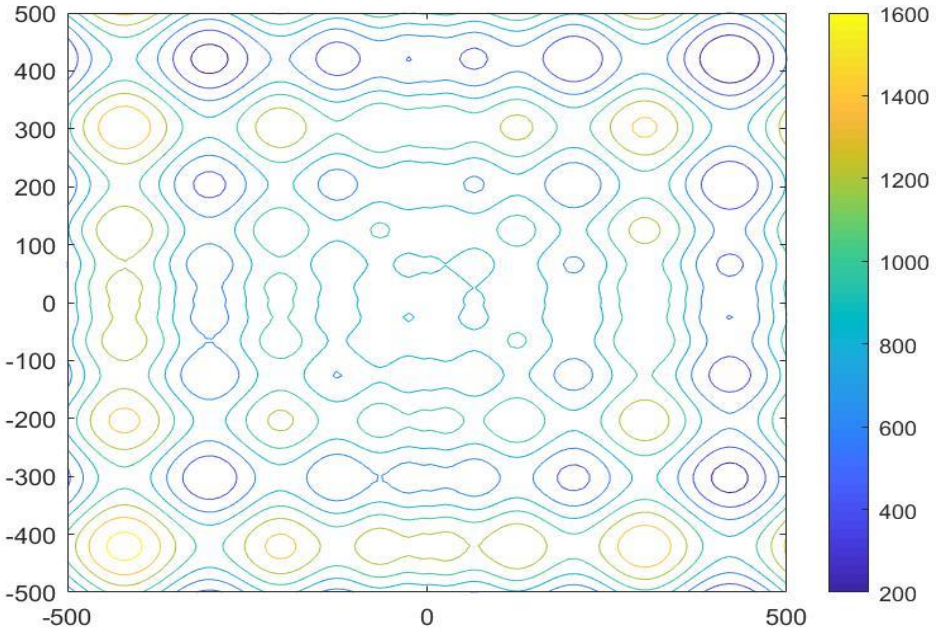
PROCEDURE:

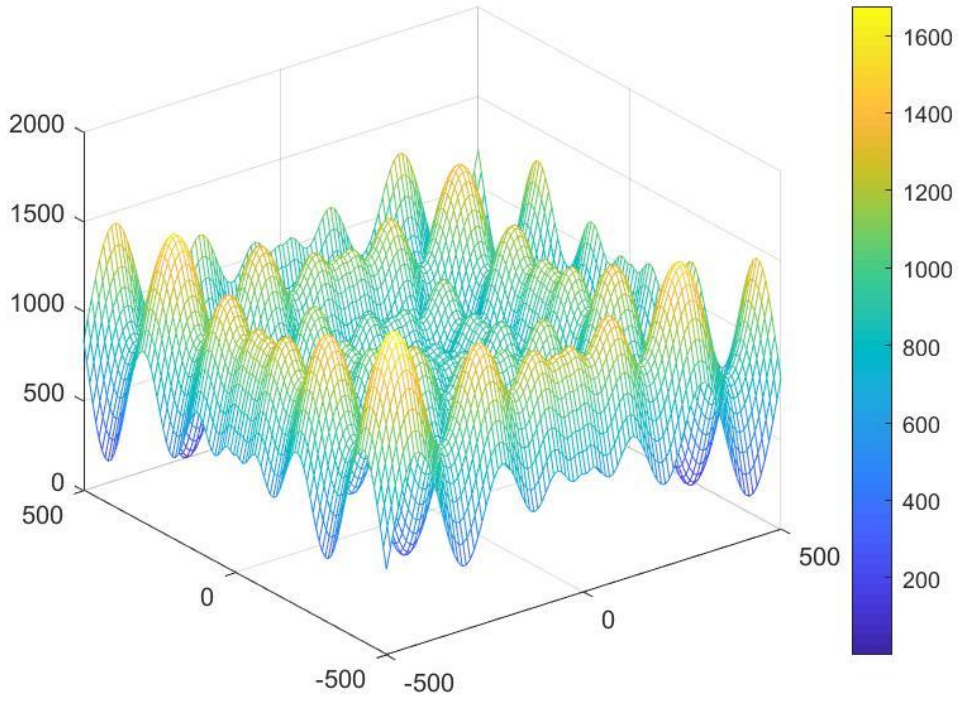
- I have generated two 100 linearly equally spaced points between -512 and 512. Then created the 100x100 pairs of points in matrix form from the vectors x and y.
- Plotted the generated values from the obtained values.

CODE:

```
x1 = linspace(-500,500); % 100 linearly equally spaced points between -512 and 512
x2 = linspace(-500,500); % 100 linearly equally spaced points between -512 and 512
[x1,x2] = meshgrid(x1,x2);  % 100x100 pairs of points in matrix form
Y = 418.9829*2 - x1.*sin(sqrt(abs(x1))) - x2.*sin(sqrt(abs(x2)));
figure(1)
contour(x1,x2,Y); % Plot
colorbar;

figure(2)
mesh(x1,x2,Y); % Plot
colorbar;
xlim([-500 500]);
ylim([-500 500]);

x=zeros(n);
y=zeros(n);
T=zeros(n);
x_star=zeros(n);
y_star=zeros(n);
t=1;
T = 100;
for t = 1 :n

x_star(t+1) = x(t) +normrnd(0,10);
y_star(t+1) =y(t) + normrnd(0,10);

val1 = 418.9829*2 - x(t)*sin(sqrt(abs(x(t)))) - y(t)*sin(sqrt(abs(y(t))));
val2 = 418.9829*2 - x_star(t+1)*sin(sqrt(abs(x_star(t+1)))) - y_star(t+1)*sin(sqrt(abs(y_star(t+1))));
 alpha = exp(val1 -val2)/ T(t);

 if ((val2 <= val1) || (rand(1)<alpha))
    x(t+1) = x_star(t+1);
    y(t+1) = y_star(t+1);

 else
    x(t+1)=x(t);
    y(t+1) = y(t);

 end

 T(t+1) = 100/log(t+1);
end
x_star(n)
y_star(n)
```

RESULT:





From the above figures the minimum value of the surface is 811.2287.

PROBLEM 4:

DESCRIPTION:

- Simulated annealing (SA) is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimization in a large search space.
- It is often used when the search space is discrete (e.g., all tours that visit a given set of cities).
- For problems where finding an approximate global optimum is more important than finding a precise local optimum in a fixed amount of time, simulated annealing may be preferable to alternatives such as gradient descent.
- The notion of slow cooling implemented in the Simulated Annealing algorithm is interpreted as a slow decrease in the probability of accepting worse solutions as the solution space is explored.

PROCEDURE:

- Generated location of cities. rng() function for reproducibility
- Then implemented the annealing algorithm.
- Then opened the given files and simulated the annealing algorithm for the data set.
- Plotted the obtained the values in the graph which are attached below.

Moreover, two major optimizations can be used to speed up the computation of the distances:

- Instead of recomputing the distance between two cities every time it is required, the distances between all pairs of cities can be precomputed in a table, and used thereafter. Actually, a triangular matrix is enough, as the distance between cities A and B is the same as the distance between B and A.

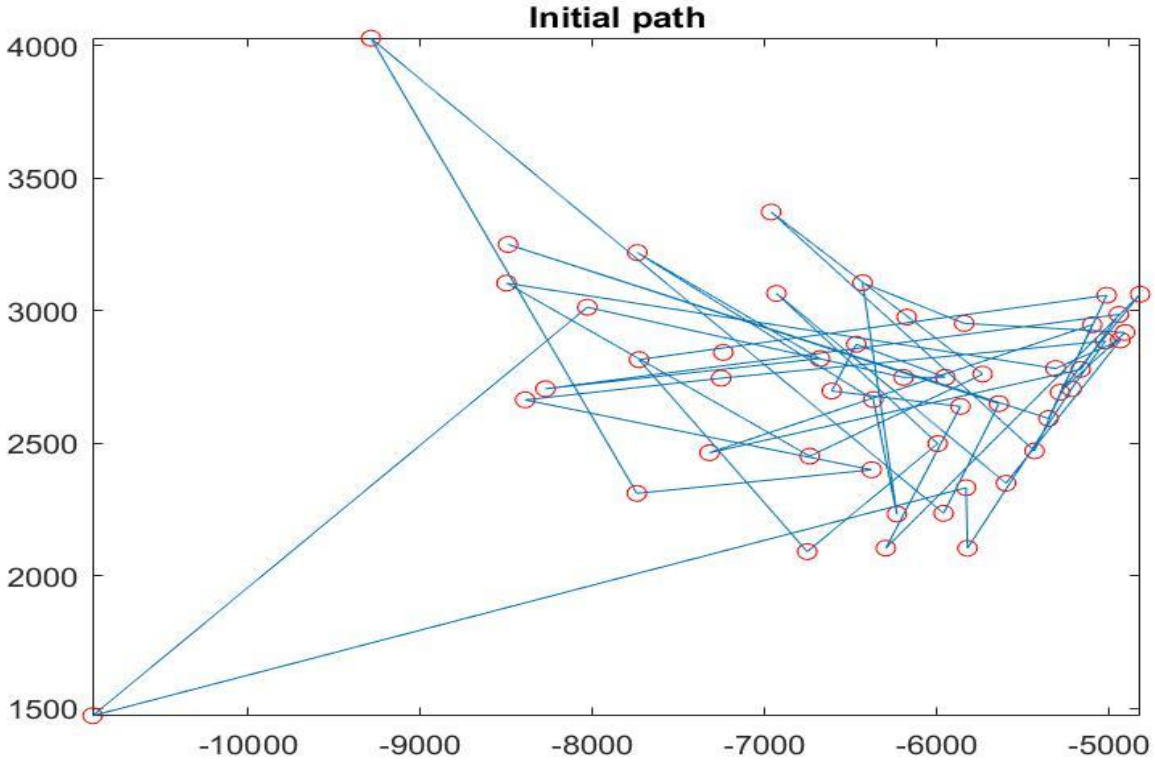
CODE:

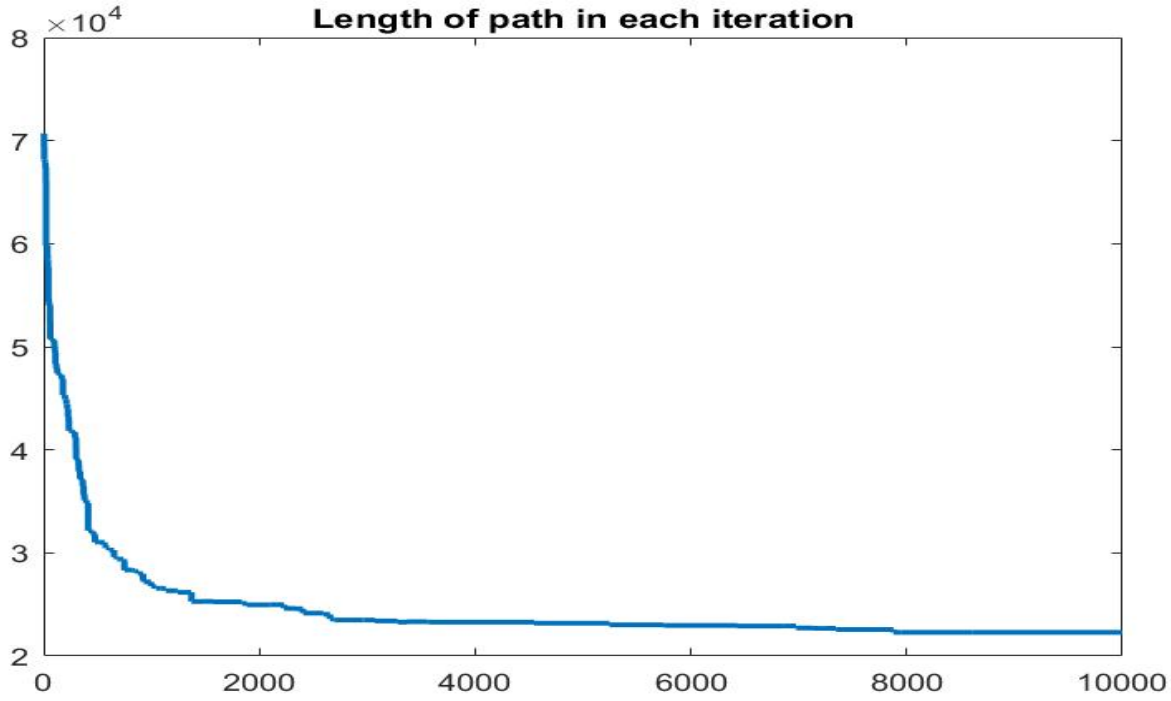
```
clear all;
clc;
content = fileread( 'uscap_xy.txt' ) ;%open the file
%read the contents of  the file
data = textscan( content, '%f %f%*[^\n]','HeaderLines', 0) ;
x = data{1};
y = data{2};
n= 48;
city = [x y];

% Simulated annealing algorithm
distance = pdist2(city, city);
% Parameters
num_iter = 10000; % number of iterations
c = 100;
% Initial path p
p = [4:n 1:3];
% Initial length of p
len = 0;
for a1 = 1:n-1
   len = len + distance(p(a1),p(a1+1));
end
len = len + distance(p(n),p(1));

% Save the paths and lengths
pathHistory = zeros(num_iter,n);
lenHistory = zeros(1,n);

% Plotting intial path
figure(1)
plot(city(:,1), city(:,2), 'ro');
xlim([min(x)-1 max(x)+1]);
ylim([min(y)-1 max(y)+1]);
hold on
line(city([p(:); p(1)],1), city([p(:); p(1)],2));
title('Initial path');
hold off

count = 0;

while(count<num_iter)
   count = count + 1;
   % Create path p2 by randomly swap two cities
   swap_index = randsample(n,2);
   p2 = p;
   temp = p2(swap_index(1));
   p2(swap_index(1)) = p2(swap_index(2));
   p2(swap_index(2)) = temp;
```

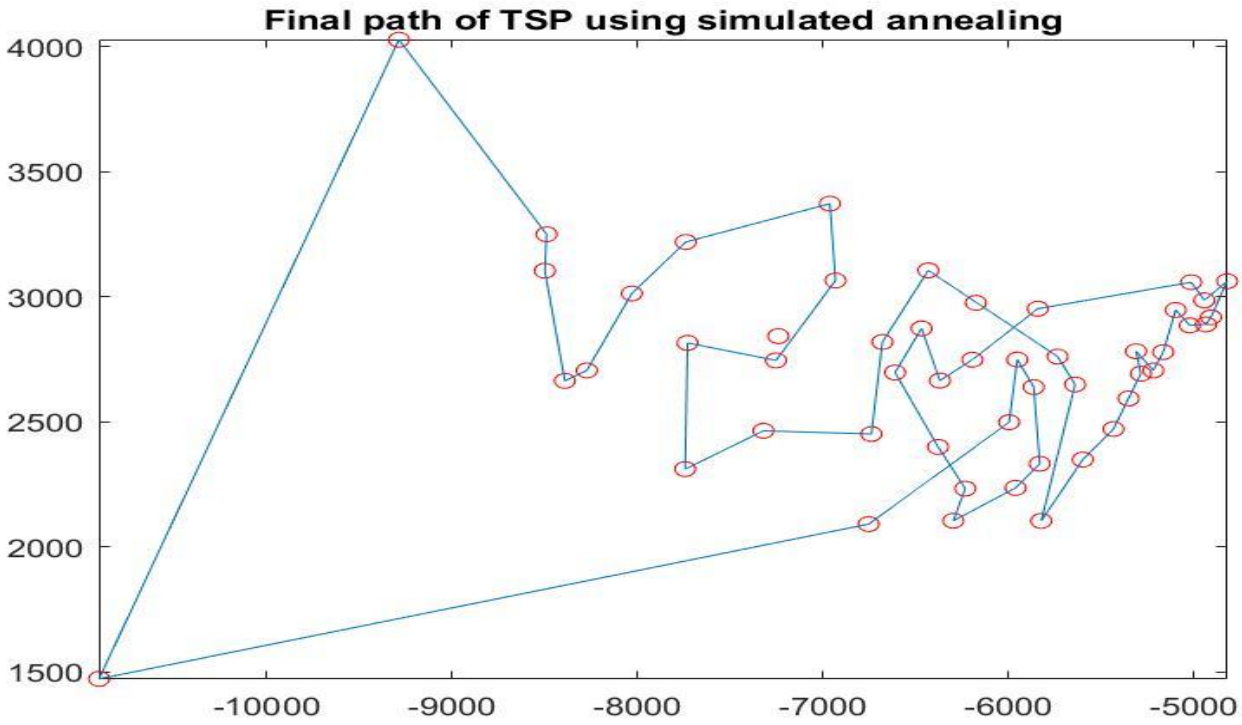
```
    % Cost of p2
    len2 = 0;
    for a1 = 1:n-1
        len2 = len2 + distance(p2(a1),p2(a1+1));
    end
    len2 = len2 + distance(p2(n),p2(1));

    q = (1+count)^((len - len2)/c);

    if len2 - len <= 0
        p = p2;
        len = len2;
    else
        if rand <= q
            p = p2;
            len = len2;
        end
    end
    pathHistory(count,:) = p;
    lenHistory(count) = len;
end

figure(2)
plot(1:num_iter, lenHistory, 'linewidth',2);
title('Length of path in each iteration')

figure(3)
plot(city(:,1), city(:,2), 'ro');
xlim([min(x)-1 max(x)+1]);
ylim([min(y)-1 max(y)+1]);
hold on
line(city([p(:); p(1)],1), city([p(:); p(1)],2));
title('Final path of TSP using simulated annealing');
hold off

fprintf('\nDistance of the shortest path that visits each of the 48 state capital cities is: %f', len);
```

RESULT:

Distance of the shortest path that visits each of the 48 state capital cities is: 22285.792096


Initial path
4000
3500
3000
2500
2000
1500
-10000
-9000
-8000
-7000
-6000
-5000


Length of path in each iteration
×10^4
8
7
6
5
4
3
2
0
2000
4000
6000
8000
10000



ANALYSIS:

It took a total of 10000 iterations to estimate the path i.e.,determine minimal path. Distance of the shortest path that visits each of the 48 state capital cities is: 22285.792096. Convergence rate of the estimate toward minimum increases with iterations