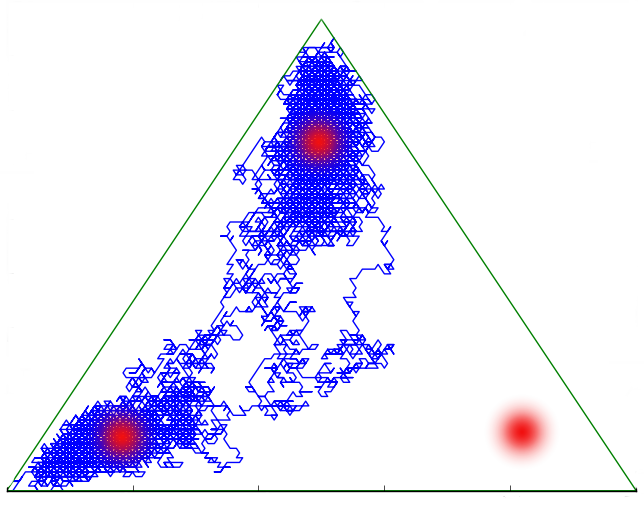
EE 511

Simulation Methods of Stochastic Systems

PROJECT #8

Markov Chain Monte Carlo



**By**

SHIVA SHANMUGAPPA PATRE

**USC ID: 6129918110**

[spatre@usc.edu](mailto:spatre@usc.edu)

**Question 1**

**Description**

Monte Carlo methods are a form of stochastic integration used to approximate expectations by invoking the law of large numbers. Importance sampling is a variance reduction technique that can be used in the Monte Carlo method. The idea behind importance sampling is that certain values of the input random variables in a simulation have more impact on the parameter being estimated than others. It involves multiplying the integrand by 1 to yield an expectation of a quantity that varies less than the original integrand over the region of integration.

For example, let h(x) be a density for the random variable X which takes values only in A so that



Then,



so long as h(x) != 0 for any x ∈ A for which g(x) != 0, and where Eh denotes the expectation with respect to the density h. This gives a Monte Carlo estimator:



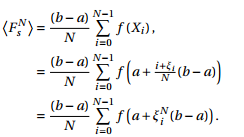
It can be shown that the variance of the above expression is minimized when h(x) is proportional to g(x).

In summary, a good importance sampling function h(x) should have the following properties:

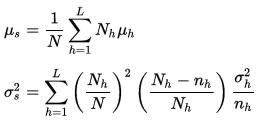
* h(x) > 0 whenever g(x) != 0
* h(x) should be close to being proportional to |g(x)|
* it should be easy to simulate values from h(x)
* it should be easy to compute the density h(x)

*Stratified sampling*

Stratified sampling works by splitting up the original integral into a sum of integrals over sub-domains. In its simplest form, stratified sampling divides the domain [a,b] into N sub-domains (or strata) and places a random sample within each of these intervals. If using a uniform PDF, with  this can be expressed as



The mean and variance for stratified sampling are given by



where,

{\displaystyle N=}N= Size of entire population, should equal to sum of all stratum sizes

{\displaystyle N\_{h}=} Nh=Size of stratum {\displaystyle h}

{\displaystyle n\_{h}=} nh=Number of observations in stratum {\displaystyle h}

{\displaystyle L=}L= Count of strata

Sigma(h) and mu(h) are the standard deviation and sample mean of stratum h

 {\displaystyle h}It can be shown that stratified sampling can never result in higher variance than pure random sampling. In fact, stratified sampling is asymptotically better, since the error reduces linearly with the number of samples, σ ∝ 1/N. Therefore, whenever possible, stratified sampling should be used. Stratified sampling does place more restrictions on the sampling process. This can become problematic when, for instance, the number of samples is not known in advance. Furthermore, though it is possible to extend stratified sampling to higher dimensions, in general, N d strata would need to be created for a d dimensional domain. This explosion of samples can make fine-grained control of simulation times difficult.

**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);

%%%%%%%%%%%%%%% Part 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);

**Output**

Mean and Variance using simple Monte Carlo

2.1387e+20

3.2444e+19

Mean and Variance using Stratified Sampling

2.0453e+20

3.1858e+18

Mean and Variance using Importance Sampling

1.5799e+20

3.4472e+19

Theoretical integral value

2.0460e+20

Mean and Variance using simple Monte Carlo

-0.0814

0.1779

Mean and Variance using Stratified Sampling

-0.0032

0.0091

Mean and Variance using Importance Sampling

6.2578e+09 + 2.2562e+08i

1.8570e+09

Theoretical integral value

-0.1471

**Analysis**

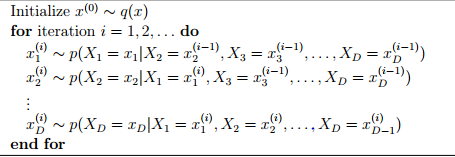
**Question 2**

**Description**

***Gibbs Sampling:*** The idea in Gibbs sampling is to generate posterior samples by sweeping through each variable (or block of variables) to sample from its conditional distribution with the remaining variables fixed to their current values. For instance, consider the random variables X1, X2, and X3. We start by setting these variables to their initial values x1(0) , x2 (0), and x3(0) (often values sampled from a prior distribution q).

At iteration i, we sample x1(i) ∼ p(X1 = x1|X2 = x2(i−1), X3 = x3(i−1) ), sample x2 ∼ p(X2 = x2|X1 = x1(i), X3 = x3(i−1)), and sample x3 ∼ p(X3 = x3|X1 = x1(i) , X2 = x2(i)). This process continues until “convergence” (the sample values have the same distribution as if they were sampled from the true posterior joint distribution).

Algorithm



**Code:**

clc;

clear all;

disp('Part A');

n = 3;

N = 1000;

X = 2\*ones(1,n);%Initialize ones to the column matrix

for k = 1:N

i = ceil(n\*rand);%Round up to a higher integer

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('Part B');

X = 2\*ones(1,n);%Initialize ones to the column matrix

for k = 1:N

i = ceil(n\*rand);%Round up to a higher integer

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**

**Part A**

1.0e+100 \*

2.1123 1.9525

**Part B**

1.0e+11 \*

-3.9981 1.2619

**Analysis**

**Question 3**

**Description**

The key features of a Schwefel function are:

* The function is continuous.
* The function is not convex.
* The function can be defined on n-dimensional space.
* The function is multimodal.

The function can be defined on any input domain but it is usually evaluated on the hypercube xi∈[−500,500] xi∈[−500,500] for i=1..ni=1..n.

**Code**

%% Contour Plotting

N=100;

x = linspace(-500,500); % Generate a row vector of 100 linearly equally spaced points between -512 and 512

y = linspace(-500,500); % Generate a row vector of 100 linearly equally spaced points between -512 and 512

[X,Y] = meshgrid(x,y); % Obtain 100x100 pairs of points in matrix form from vectors x and y

Z = 418.9829\*2 - X.\*sin(sqrt(abs(X))) - Y.\*sin(sqrt(abs(Y))); % Compute f(x,y) in matrix form

figure(1)

contour(X,Y,Z); % Plot a contour plot

colorbar;

xlabel('X1');

ylabel('X2');

figure(2)

mesh(X,Y,Z); % Plot a mesh plot

colorbar;

xlabel('X1');

ylabel('X2');

zlabel('Z');

xlim([-500 500]);

ylim([-500 500]);

%% Eggholder Function(Simulated Annealing)

R = zeros(N);

S = zeros(N);

T = zeros(N);

T = 100;

R\_a = zeros(N);

S\_a = zeros(N);

for t = 1:N

R\_a(t + 1) = R(t) + normrnd(0,10);

S\_a(t + 1) = S(t) + normrnd(0,10);

Z1 = 418.9829\*2 - R\_a(t+1).\*sin(sqrt(abs(R\_a(t+1)))) - S\_a(t+1).\*sin(sqrt(abs(S\_a(t+1))));

Z2 = 418.9829\*2 - R(t).\*sin(sqrt(abs(R(t)))) - S(t).\*sin(sqrt(abs(S(t))));

alpha = exp(Z2 - Z1)/T(t);

if((Z1 <= Z2) || (rand(1) < alpha))

R(t+1) = R\_a(t+1);

S(t+1) = S\_a(t+1);

else

R(t+1) = R(t);

S(t+1) = S(t);

end

T(t+1) = 100/(t+1);

end

display('The minimum values of X\* and Y\* are:');

display(R\_a(N));

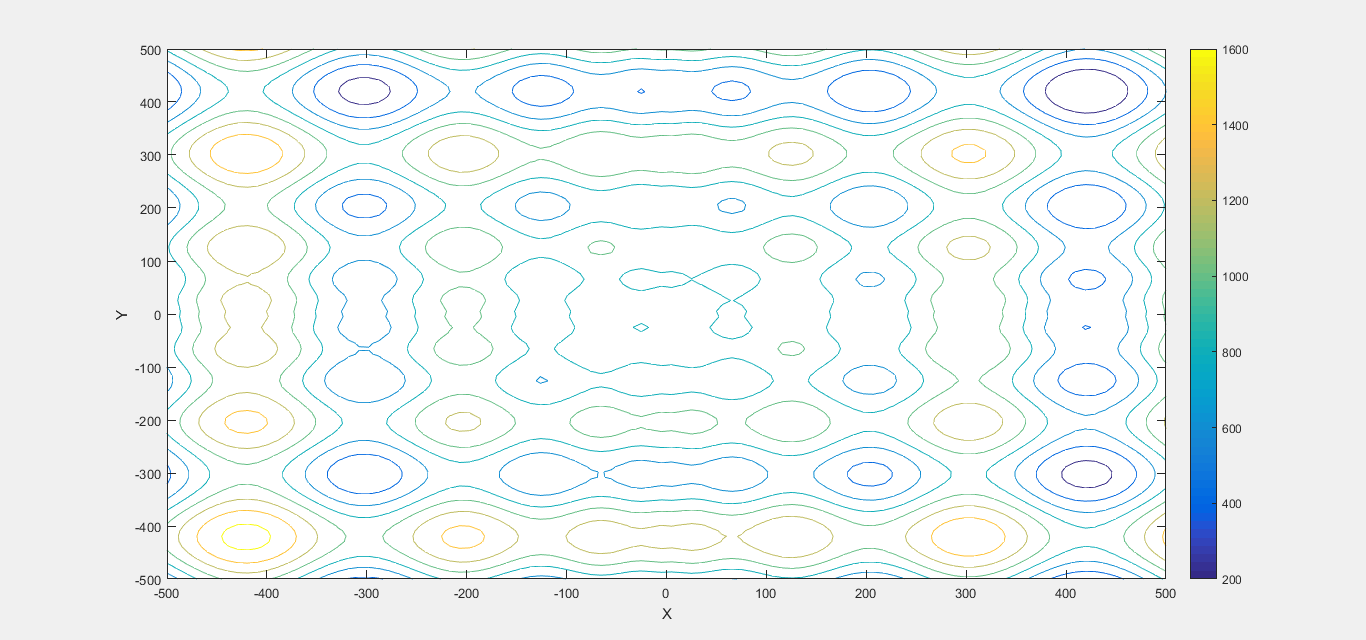
display(S\_a(N));

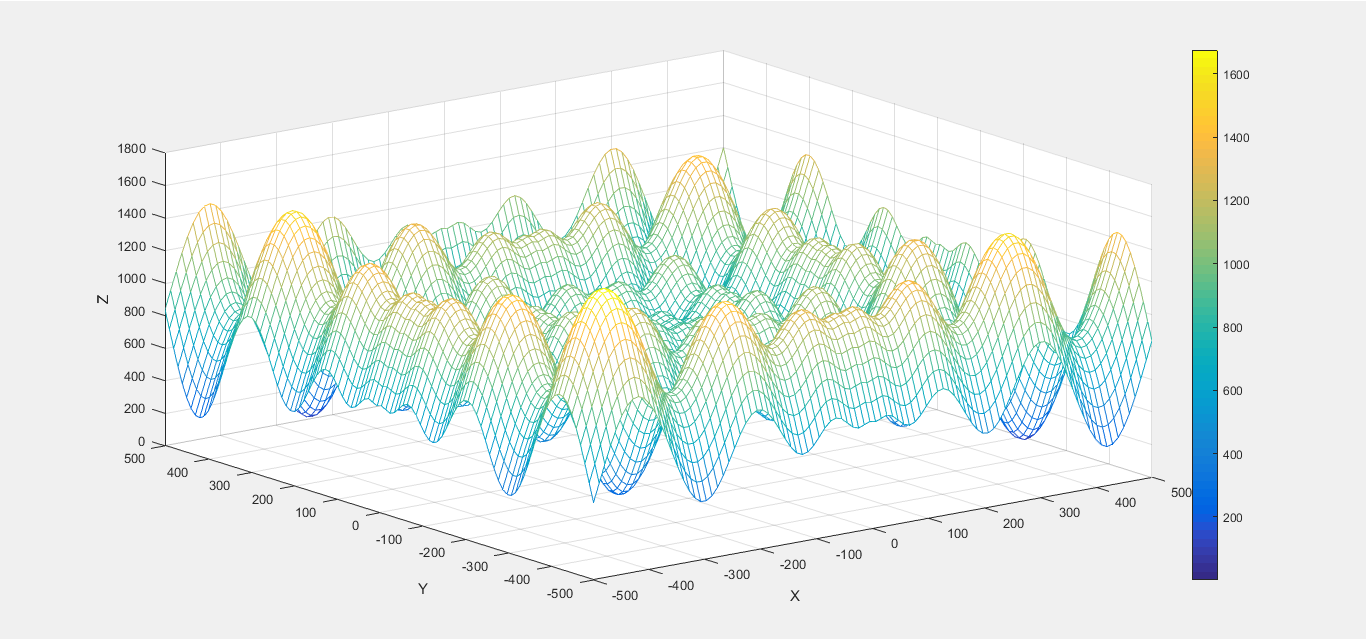
display('The minimum values of the surface is:');

display(418.9829\*2 - R\_a(N).\*sin(sqrt(abs(R\_a(N)))) - S\_a(N).\*sin(sqrt(abs(S\_a(N)))));

**Output**

The minimum values of the surface is 811.2287





**Analysis**

**Question 4**

**Description**

***Simulated annealing***

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).

Applying the simulated annealing technique to the traveling salesman problem can be summed up as follow:

* Create the initial list of cities by shuffling the input list (i.e: make the order of visit random).
* At every iteration, two cities are swapped in the list. The cost value is the distance traveled by the salesman for the whole tour.
* If the new distance, computed after the change, is shorter than the current distance, it is kept.
* If the new distance is longer than the current one, it is kept with a certain probability.
* We update the temperature at every iteration by slowly cooling down.

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));

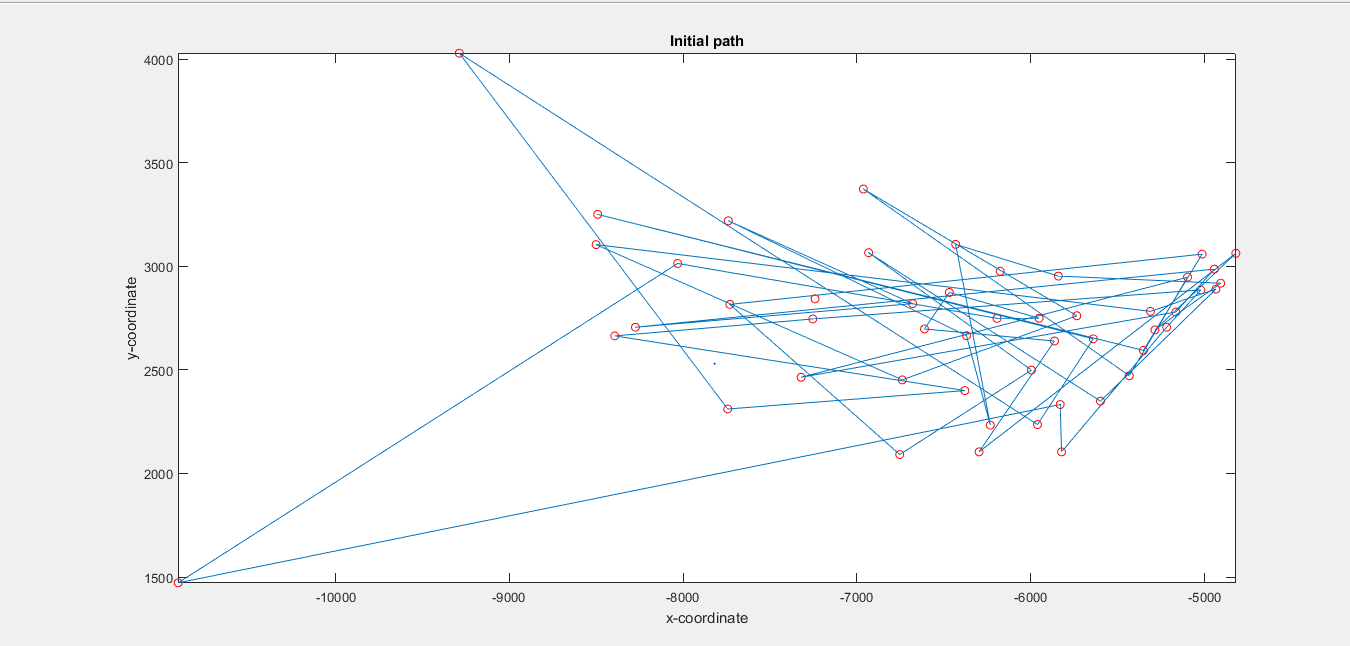
title('Final path of TSP using simulated annealing');

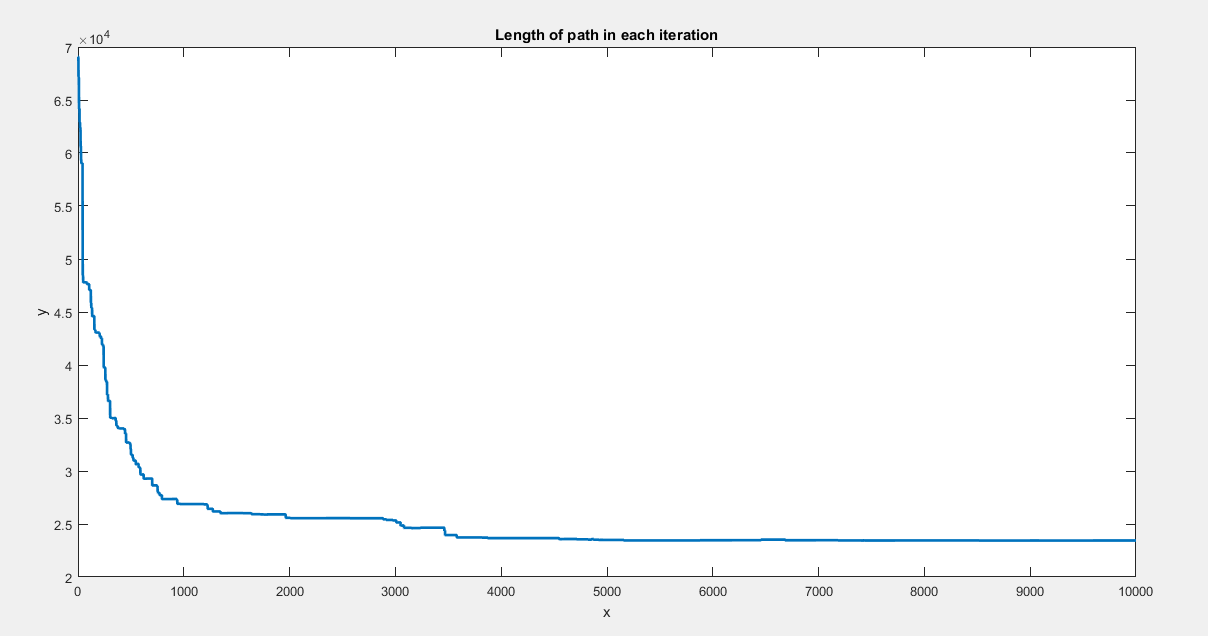
hold off

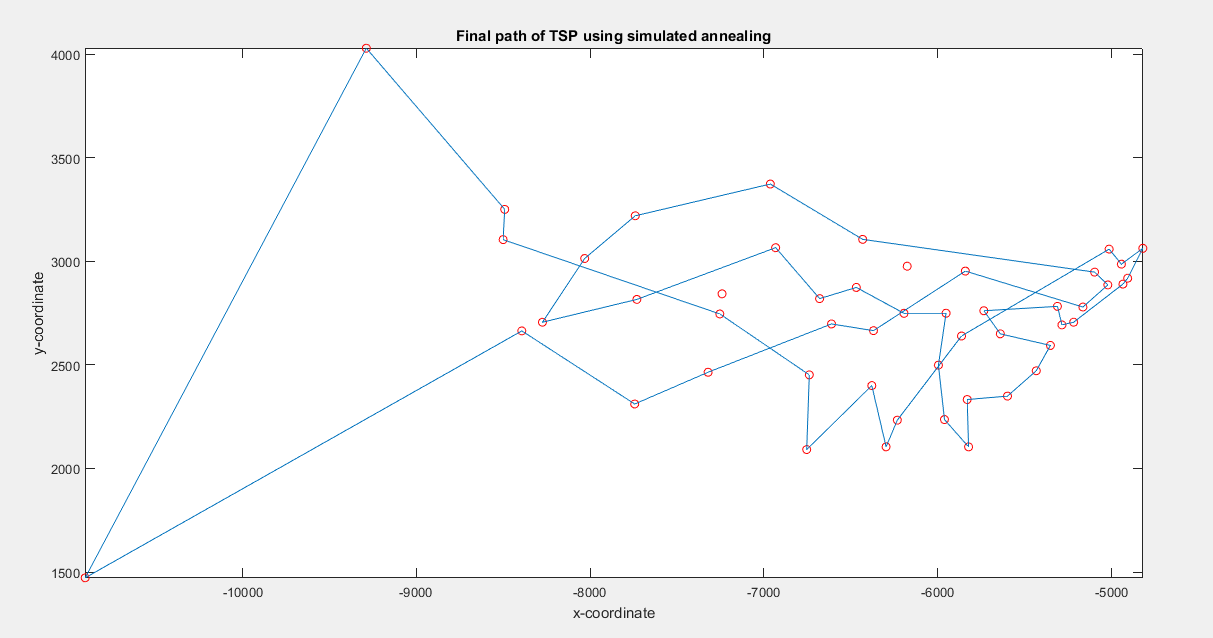
fprintf('\nDistance of the shortest path that visits each of the 48 state capital cities is: %f', len);

**Output**

Distance of the shortest path that visits each of the 48 state capital cities is: 21946.356297







**Analysis**