University of Southern California EE511

Markov Chain Monte Carlo

Project #8

Name: Wu Jiawei
USCID: 9600392575

Abstract

In the project of Markov chain Monte Carlo, four experiments are conducted using Matlab. The project is conducted to estimate defined integral, conditional expectation and find the minimum result for optimization. The theories of Markov chain system, Gibbs sampling, simulated annealing simulation and stratification method are applied comprehensively in the lab. The experiments are repeated multiple times and the outcomes are shown in diagrams and calculations.

Introduction

Four experiments are conducted in the lab. All the samples are generated by random selection and the experiments are repeated multiple times. The goal of the first trial is to follow the Monte Carlo theory to finish integral estimation and compare the outcomes with stratification and importance sampling results. The aim of the second experiment is to estimate the conditional expectation values of independent exponential random variables using Gibbs sampling. The objective of the third experiment is running simulation to choose different parameters and find the single global minimum according to the Schwefel function. The aim of the fourth experiment is to estimate the shortest path between states with the simulated annealing method and generate the total tour distance as a function of the simulation time.

Methodology & Results

Experiment No.1

Description of Algorithm:

1. Markov chain simulation

In Markov chain system, the probability of moving to the next state depends only on the present state and not on the previous states.

P(Xn = xn|Xn-1 = xn-1, Xn-2 = xn-2,..., X0=x0) = P(Xn = xn|Xn-1 = xn-1)

2. Computing definite integrals with important sampling

By generating numbers in crude Monte Carlo methods, information is spread all over the interval that sampled over. A simple transformation of the problem exist for which Monte Carlo can generate a better result. Suppose a function g(x) exists such that h(x)=f(x)/g(x) is almost constant over the domain of integration. Restate the problem

$$\int f(x)dx = \int \frac{f(x)}{g(x)}g(x)dx = \int h(x)g(x)dx$$

Then the integral f can be obtained by sampling from h(x).

The approximated integral is given by

$$\hat{I}_f^{is} = \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{g(x_i)}$$

With variance

$$\begin{aligned}\sigma^2_{\hat{I}_f} &= \frac{1}{N}(\int \frac{f(x)^2}{g(x)^2}g(x)dx - (\int \frac{f(x)}{g(x)}g(x)dx)^2) \\ &= \frac{1}{N}(\int \frac{f(x)^2}{g(x)}dx - (\int f(x)dx)^2)\end{aligned}$$

3. Computing definite integrals with stratified sampling

The theory of stratified sampling is to split up the domain D of X into separate regions, take a sample of points from each such region and combine the results to estimate E(f(X)).

The sample space is stratified to r disjoint region

Set $\lambda \in (0,1)$ and draw $N_a = \lambda N$ data points over $[0,\lambda]$ and $N_b = N - N_a = (1-\lambda)N$ over $[\lambda, 1]$. The integral is then evaluated by

$$\hat{I}_f^s = \frac{1}{N_a}\sum_{i=1}^{N_a} f(x_i^a) + \frac{1}{N_b}\sum_{i=1}^{N_b} f(x_i^b)$$

where $x_i^a \in [0,\lambda]$ and $x_i^b \in [\lambda, 1]$.

Variance of this estimator is:

$$\frac{\lambda^2}{N_a}\mathrm{Var}_a(f(x)) + \frac{(1-\lambda)^2}{N_b}\mathrm{Var}_b(f(x)) = \frac{\lambda}{N}\mathrm{Var}_a(f(x)) + \frac{(1-\lambda)}{N}\mathrm{Var}_b(f(x))$$

Description of Method:

The experiment runs simulation to obtain two integrals of two dimensions random variables x1 and x2. Three methods which are Monte Carlo estimates, stratification and importance sampling are utilized under same budget n=1000. The comparison of quality and sample variance are conducted between three Monte Carlo integral estimates.

Part 1: The first integral

Code:

```
clc;
clear;
%Simple MC
g = @(x)exp(sum(5 * abs(x - 5)));
N = 1000;
X = g(rand(2,N)); % Simple MC
Integral_MC = mean(X);
Variance_MC = 2*std(X)/sqrt(N);

%Stratified
K = 10;
Nij = N/K^2; % Stratified
for i = 1:K
```

```
for j = 1:K
XS = g([i-1+rand(1,Nij);j-1+rand(1,Nij)]/K);
XSb(i,j) = mean(XS);
SS(i,j) = var(XS);
end
end
SST = mean(mean(SS/N));
%MCSS = ([mean(mean(XSb)) 2*sqrt(SST) ]);
Integral_Stratified = mean(mean(XSb));
Variance_Stratified = 2*sqrt(SST);

%Important sampling
N = 1000;
U = rand(2,N);
e = exp(1);
X = log(1+(e-1)*U);
T = (e-1)^2*exp(sum(5 * abs(X - 5))-sum(abs(X)));
%MCIS = ([mean(T) 2*std(T)/sqrt(N)]);
Integral_Importsamping = mean(T);
Variance_Importsampling = 2*std(T)/sqrt(N);
```

Simulation Result:

Workspace

Name	Value
e	2.7183
g	@(x)exp(sum(5*abs(x-5)))
i	10
Integral_Importsamping	2.2644e+20
Integral_MC	2.0089e+20
Integral_Stratified	2.0572e+20
j	10
K	10
N	1000
Nij	10
SS	*10x10 double*
SST	1.1485e+37
T	*1x1000 double*
U	*2x1000 double*
Variance_Importsampling	5.5818e+19
Variance_MC	2.9468e+19
Variance_Stratified	6.7780e+18
X	*2x1000 double*
XS	[4.1143e+17,3.6629e+17,...
XSb	*10x10 double*

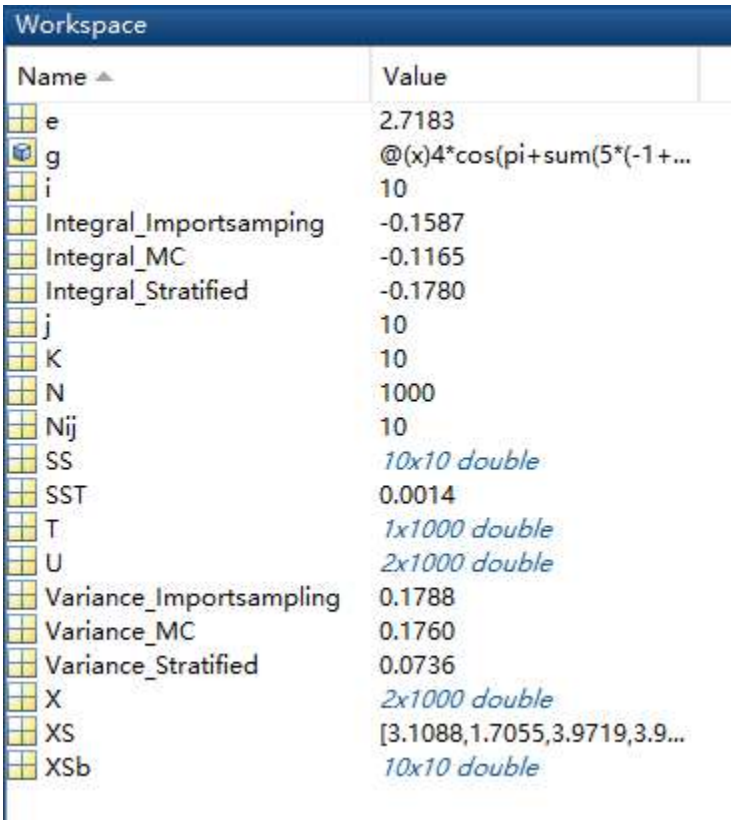
Part 2: The second integral

Code

```
clc; clear; close all;
```

```
%Simple MC
g = @(x) 4 * cos(pi+sum(5 * (-1+2*x)));
N = 1000;
U = rand(2,N);
X = g(U); % Simple MC
Integral_MC = mean(X);
Variance_MC = 2*std(X)/sqrt(N);

%Stratified
K = 10; Nij = N/K^2;
for i = 1:K
for j = 1:K
XS = g([i-1+rand(1,Nij);j-1+rand(1,Nij)]/K);
XSb(i,j) = mean(XS);
SS(i,j) = var(XS);
end
end
SST = mean(mean(SS/N));
Integral_Stratified = mean(mean(XSb));
Variance_Stratified = 2*sqrt(SST);

%Important sampling
e = exp(1);
X = log(1+(e-1)*U);
T = (e-1)^2 * (4*cos(pi+sum(5 * (-1+2*X)))).*3;
Integral_Importsamping = mean(T);
Variance_Importsampling = 2*std(T)/sqrt(N);
```

Simulation Result:



Finding:

1). For the first integral $\exp\left(\sum_{i=1}^{2} 5 \cdot |x_i - 5|\right)$ for $0 \le x_i \le 1.$. The theoretical value of integral is $2.04603*10^{20}$.

It is shown in the result that the Monte Carlo estimation of the integral is $2.2644*10^{20}$.

The stratification estimation of the integral is $2.0089*10^{20}$.

The importance sampling estimation of the integral is $2.0572*10^{20}$.

Therefore, the three different Monte Carlo integral estimations are close to the theoretical value.

The sample variance of the Monte Carlo estimation of the integral is $2.9468*10^{19}$.

The sample variance of the stratification estimation of the integral is $6.778*10^{18}$.

The sample variance of the importance sampling of the integral is $5.5818*10^{19}$.

Thus, the sample variance of the stratification estimation of the integral is less than the sample variance of the Monte Carlo and importance sampling estimation.

2). For the first integral $\cos\left(\pi + \sum_{i=1}^{2} 5x_i\right)$ for $-1 \le x_i \le 1.$. The theoretical value of integral is -0.147126.

It is shown in the result that the Monte Carlo estimation of the integral is -0.1165.

The stratification estimation of the integral is -0.1780.

The importance sampling estimation of the integral is -0.1587.

Therefore, the three different Monte Carlo integral estimations are close to the theoretical value.

The sample variance of the Monte Carlo estimation of the integral is 0.1760.

The sample variance of the stratification estimation of the integral is 0.0736.

The sample variance of the importance sampling of the integral is 0.1788.

Thus, the sample variance of the stratification estimation of the integral is less than the sample variance of the Monte Carlo and importance sampling estimation.

Experiment No.2

Description of Algorithm:

1.The theory of Gibbs Sampling:

The underlying logic of MCMC sampling is that we can estimate any desired expectation by ergodic average. That is, we can compute any statistic of a posterior distribution if we have N simulated samples from that distribution.

$$E[f(s)]_{\mathcal{P}} \approx \frac{1}{N}\sum_{i=1}^{N} f(s^{(i)})$$

where P is the posterior distribution of interest, f(s) is the desired expectation, and f(s(i)) is the i(th) simulated sample from P.

Gibbs sampling is on MCMC technique to obtain samples from the posterior distribution.

Step1: Derive the full joint density, and the posterior conditionals for each of the random variables in the model.

Step2: Simulate samples from the posterior joint distribution based on the posterior conditionals.

2.The summary of Gibbs sampler algorithm is following.

Initialize $x^{(0)} \sim q(x)$
for iteration $i = 1, 2, \ldots$ **do**
$x_1^{(i)} \sim p(X_1 = x_1 | X_2 = x_2^{(i-1)}, X_3 = x_3^{(i-1)}, \ldots, X_D = x_D^{(i-1)})$
$x_2^{(i)} \sim p(X_2 = x_2 | X_1 = x_1^{(i)}, X_3 = x_3^{(i-1)}, \ldots, X_D = x_D^{(i-1)})$
$\vdots$
$x_D^{(i)} \sim p(X_D = x_D | X_1 = x_1^{(i)}, X_2 = x_2^{(i)}, \ldots, X_D = x_{D-1}^{(i)})$
end for

Description of Method:

The aim of simulation is to estimate the conditional expectations. The Gibbs sampling theory is used to utilize independent exponential random variables X1, X2 and X3. N samples are initialized

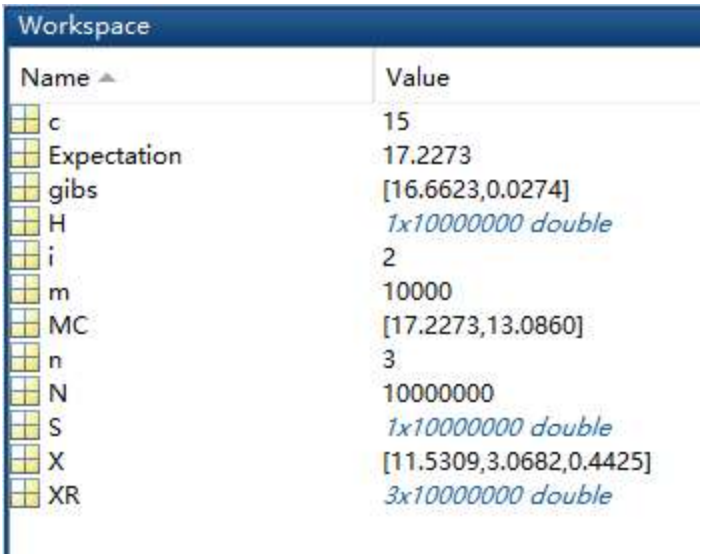
from the probability function. Then the sample is updated based on the conditional probability. The above steps are repeat N times based on Markov Chain Monte Carlo system to estimate the experimental value of expectation.

Part 1: The first conditional expectation

Code:

```
clc; clear; close all;

N = 10000;
n = 3;
c = 15;
X = ones(1,n); % Initial X
for m = 1 : N
  i = ceil(n*rand);    % random coordinate index i
  S = sum(X) - X(i);
  X(i) = max(c-S,0) - log(rand)/i; % new X(i)
  H(m) = S + i * X(i); % new sum for state m
end
gibs = ([mean(H) 2*std(H)/sqrt(N)]);

N = 10000000;
XR = -diag(1./(1:n))*log(rand(n,N));
S = sum(XR);
H = S.*(S>15)/mean(S>15);
MC = ([mean(H) 2*std(H)/sqrt(N)]);
Expectation = mean(H);
```

Simulation Result:



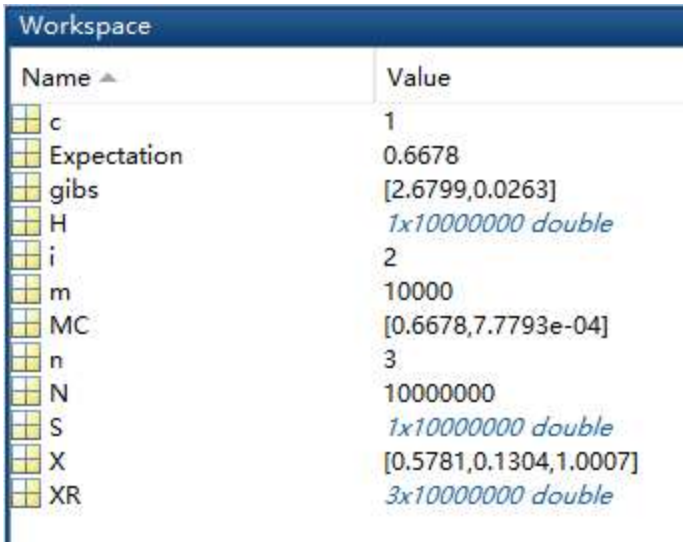
Part2: The second conditional expectation

Code:

```
clc; clear; close all;
```

```
N = 10000;
n = 3;
c = 1;
X = [0.1 0.1 0.1]; % Initial X
for m = 1 : N
  i = ceil(n*rand);    % random coordinate index i
  S = sum(X) - X(i);
  X(i) = max(c-S,0) - log(rand)/i; % new X(i)
  H(m) = S + i * X(i);                  % new sum for state m
end
gibs = ([mean(H) 2*std(H)/sqrt(N)]);

N = 10000000;
XR = -diag(1./(1:n))*log(rand(n,N));
S = sum(XR);
H = S.*(S<1)/mean(S<1);
MC = ([mean(H) 2*std(H)/sqrt(N)]);
Expectation = mean(H);
```

Simulation Result:



Finding:

The Gibbs sampler is used to estimate the value of expectations.

1). For the first expectation $\mathrm{E}[X_1 + 2X_2 + 3X_3 | X_1 + 2X_2 + 3X_3 > 15]$. The theoretical value is appropriate 18.

It is shown in the simulation result that the experimental expectation is 17.2273. Thus, the estimated expectation is close to the theoretical value.

2). For the first expectation $\mathrm{E}[X_1 + 2X_2 + 3X_3 | X_1 + 2X_2 + 3X_3 < 1]$. The theoretical value is appropriate 0.72.

It is shown in the simulation result that the experimental expectation is 0.6678. Thus, the estimated expectation is close to the theoretical value.

Experiment No.3

Description of Algorithm:

1.Theory of Schwefel function

The Schwefel function is a standard optimization benchmark that is useful to evaluate characteristic of optimization algorithm.

The Schwefel function is given by

$$f(x) = 418.9829d - \sum_{i=1}^{d} x_i \sin\left(\sqrt{|x_i|}\right)$$

The Schwefel function has many local minimums and a single global minimum.

2. The theory of simulated annealing

The simulated annealing is a method for finding a good solution to an optimization problem.

The algorithm of simulated annealing is following

1.First, generate a random solution

2.Calculate its cost using some cost function you've defined

3. Generate a random neighboring solution

4. Calculate the new solution's cost

5. Compare them:

i).If cnew < cold: move to the new solution

ii).If cnew > cold: maybe move to the new solution

6. Repeat steps 3-5 above until an acceptable solution is found or you reach some maximum number of iterations.

The advantage of simulated annealing is that it can overcome the tendency to get stuck in local optimums and is much better on average at finding an approximate global optimum.

Description of Method:

The experiment utilizes the method of simulated annealing to find the single global minimum of the 2-dimensional Schwefel function. Simulated annealing starts with an initial path p(0) with cost l(p(0)). The simulation might get stuck on a local optimal if p(t+1)=p′ is to be accepted as the next path when l(p′)≤l(p(t)) and remain p(t+1)=p(t) if l(p′)>l(p(t)). The process of risk taking that accept p(t+1)=p′ though p′ is worse than p(t) is utilized to makes the algorithm better to find the global optimization value. The comparisons of the simulation behaviors are conducted under different cooling schedules and iteration counts. The 2-D sample path of the best estimates of the global minimums are overlay on the contour plot to show the optimization result.

The process of simulated annealing is as followed

1.Choose initial guess for solution (e.g.(0,0))

2.Generate candidate by sampling from normal (or other jump distribution) for each coordinate.

3.Compute α by evaluating the function at the old and new point.

4.Make the jump or stay.

5.Cooling schedule.

6.Repeat.

Code:

```
%%Run simulation with different iteration counts.
%%N=50;
%%N=200;
%%N=1000;


%%Comparing the behavior of simulation using exponential, polynomial and
%%logarithmic cooling schedules
%%exponential cooling schedule:p = exp((temp_z - z(i-1))/T);
%%polynomial cooling schedule:p = (temp_z - z(i-1))/T;
%%logarithmic cooling schedule:p = log((temp_z - z(i-1))/T);

x1 = linspace(-500,500); % Generate a row vector of linearly equally spaced
points between -500 and 500
x2 = linspace(-500,500); % Generate a row vector of linearly equally spaced
points between -500 and 500
[X1,X2] = meshgrid(x1,x2);  % Obtain 100x100 pairs of points in matrix form
from vectors x1 and x2
d = 2;
Z = 4.189829 * d - X1 .* sin(sqrt(abs(X1))) - X2 .* sin(sqrt(abs(X2)));
N=1000;
%%Run simulation with different iteration counts.
%%N=50;
%%N=200;
%%N=1000;
z_min(1:N) = 100;
global_min = z_min(1);
for j = 1:N
    point(1,:) = [0, 0];
    z(1) = 4.189829 * d - 0 .* sin(sqrt(abs(0))) - 0 .* sin(sqrt(abs(0)));
    N = 1000;
    T = 100;
    for i = 2 : N
        mu = 0;
        sigma = 5.5;
        noise = normrnd(mu,sigma,[1,2]);
        temp_point = point(i-1,:) + noise;
        temp_z = 4.189829 * d - temp_point(1) * sin(sqrt(abs(temp_point(1))))
- temp_point(2) * sin(sqrt(abs(temp_point(2))));
        p = exp((temp_z - z(i-1))/T);
%%Comparing the behavior of simulation using exponential, polynomial and
```

```
%%logarithmic cooling schedules
%%exponential cooling schedule:p = exp((temp_z - z(i-1))/T);
%%polynomial cooling schedule:p = (temp_z - z(i-1))/T;
%%logarithmic cooling schedule:p = log((temp_z - z(i-1))/T);
        if (z(i-1) > temp_z)
            z(i) = temp_z;
            point(i, :) = temp_point;
        else
            if (rand(1) > p)
                z(i) = z(i-1);
                point(i, :) = point(i-1, :);
            else
                z(i) = temp_z;
                point(i, :) = temp_point;
            end
        end
        T = 100*log(i+1);
        if (z(i) < z_min(j))
            z_min(j) = z(i);
            if z_min(j) < global_min
                global_min = z_min(j);
                path = point;
            end
        end
    end
end
figure(1);
hist(z_min);
xlabel('The Global Minimum');
ylabel('The Occurence Time');
title('Histogram Showing the Global minimums for Each Case');
figure(2);
contour(X1,X2,Z); % Generate a contour plot
colorbar;
title('The Contour Plot');
figure(3);
contour(X1,X2,Z); % Generate a contour plot overlay with best estimations
colorbar;
hold on
plot(path(:,1), path(:,2), 'ro');
line(path(:,1),path(:,2));
title('The Contour Plot overlayed with the Best Estimates');
hold off;
figure(4);
mesh(X1,X2,Z); % Generate a mesh plot
colorbar;
xlim([-500 500]);
ylim([-500 500]);
title('The Mesh Plot');
```

Simulation Result:

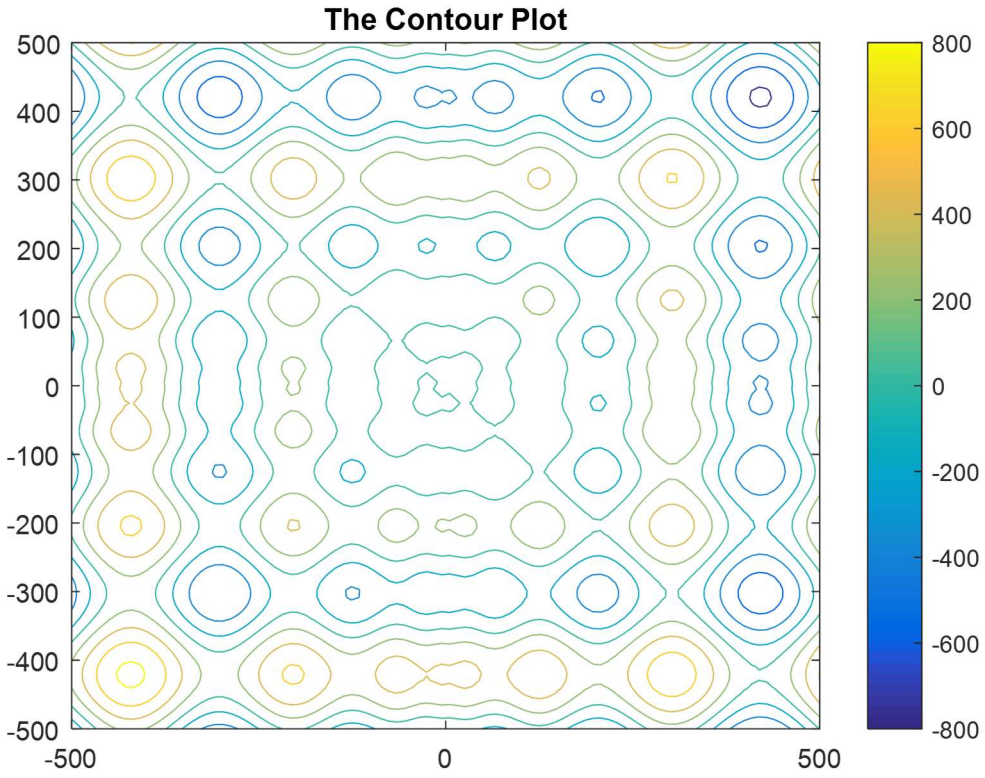
i). Exponential cooling schedule

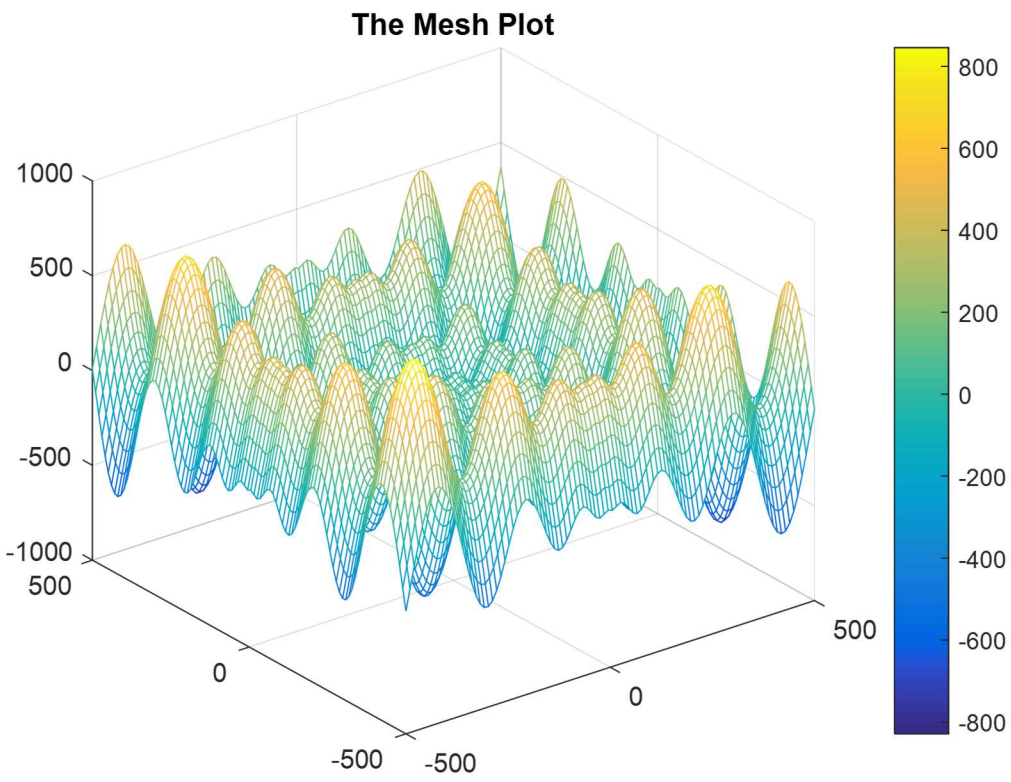
%%N=50;

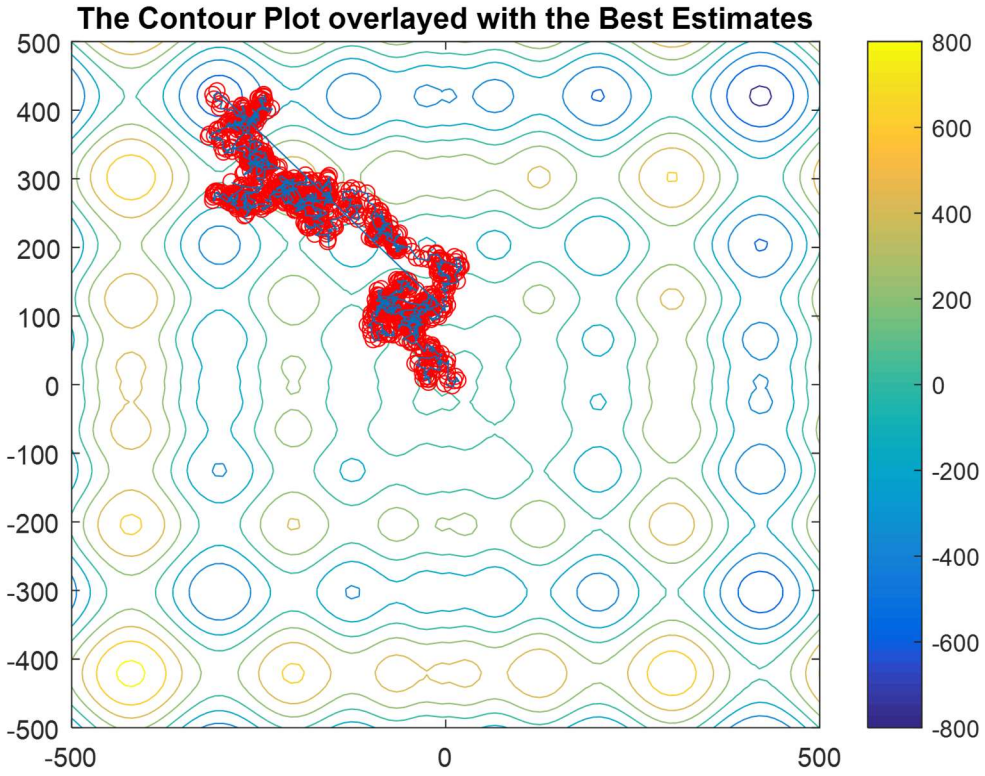
%%N=200;

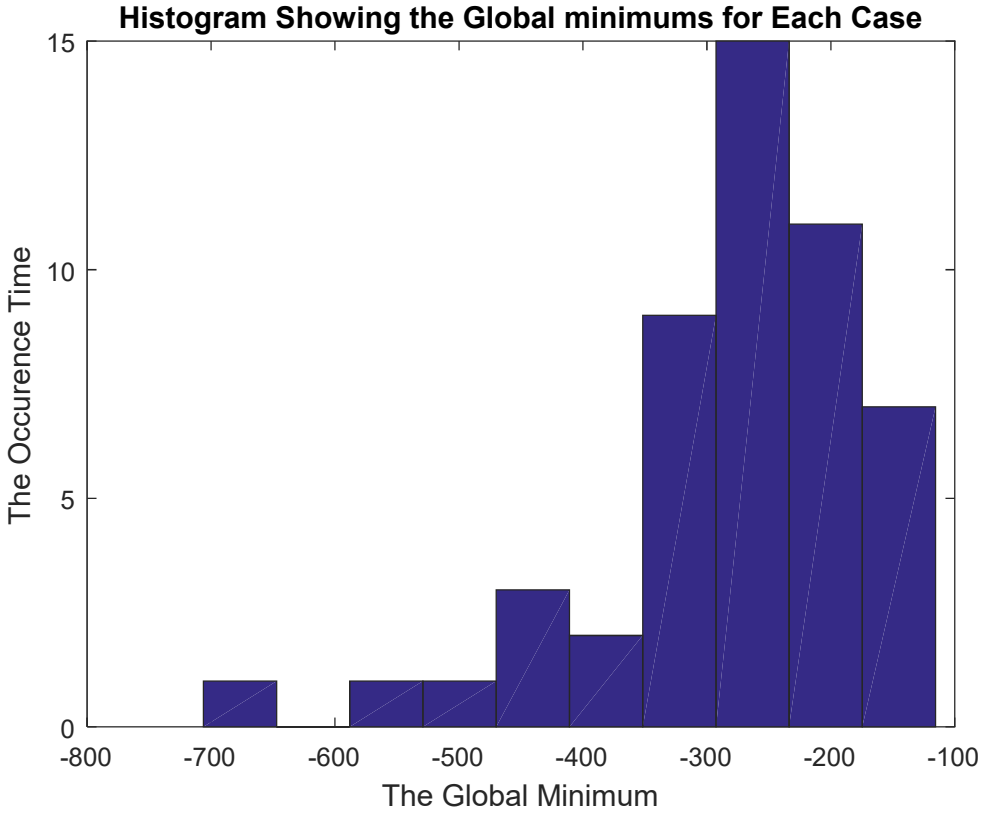
%%N=1000;

1). N=50

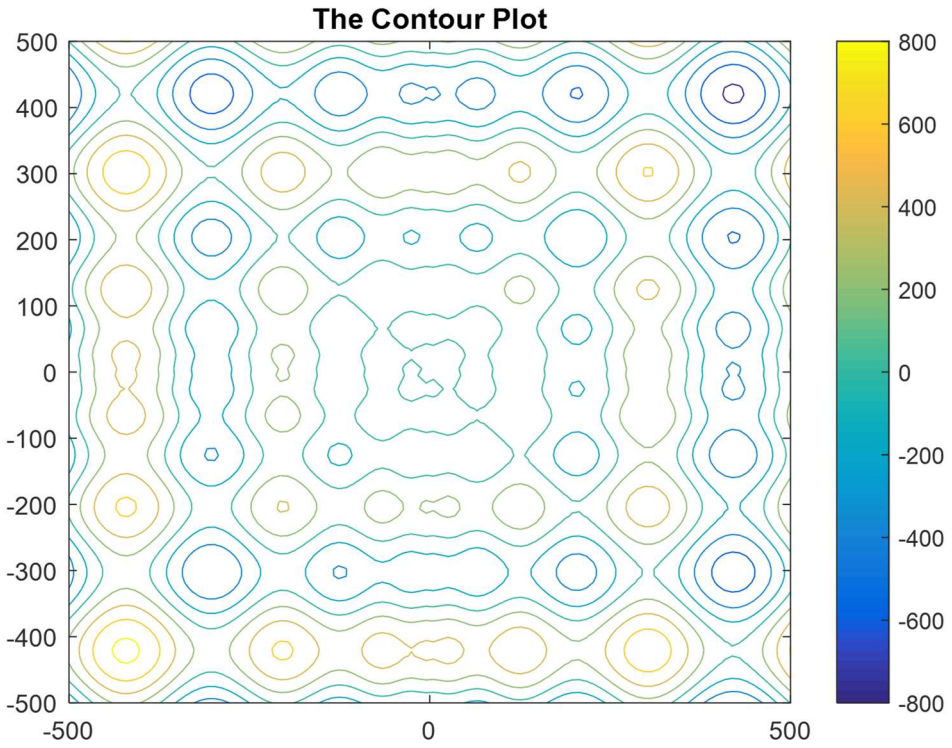


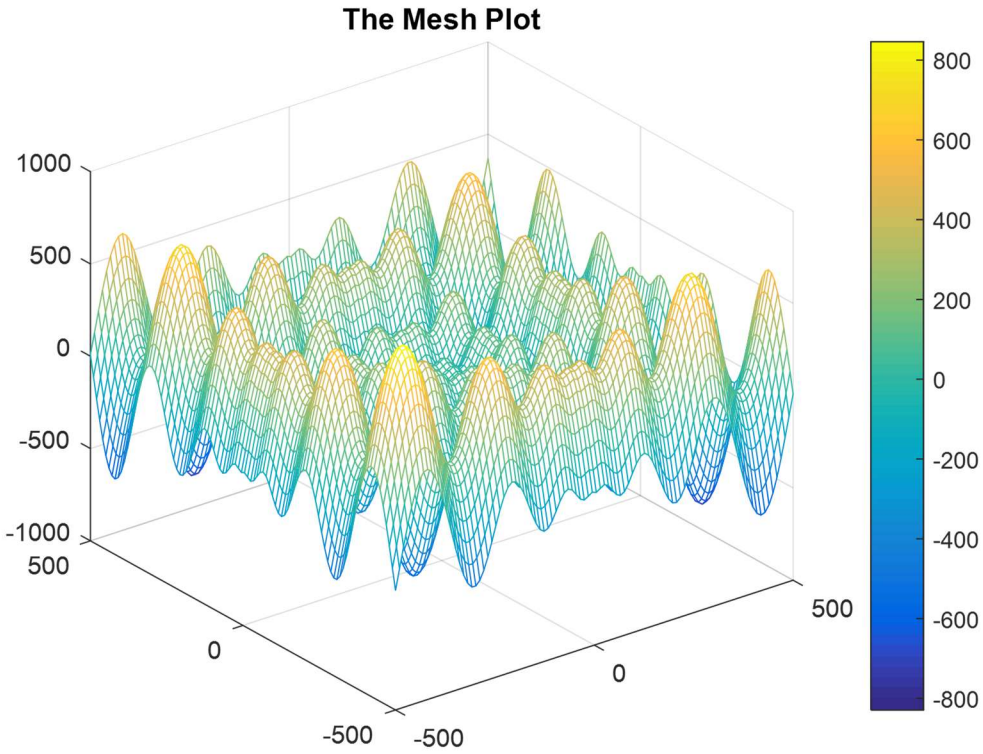

The Mesh Plot
1000
500
0
-500
-1000
500
0
-500
-500
0
500
800
600
400
200
0
-200
-400
-600
-800

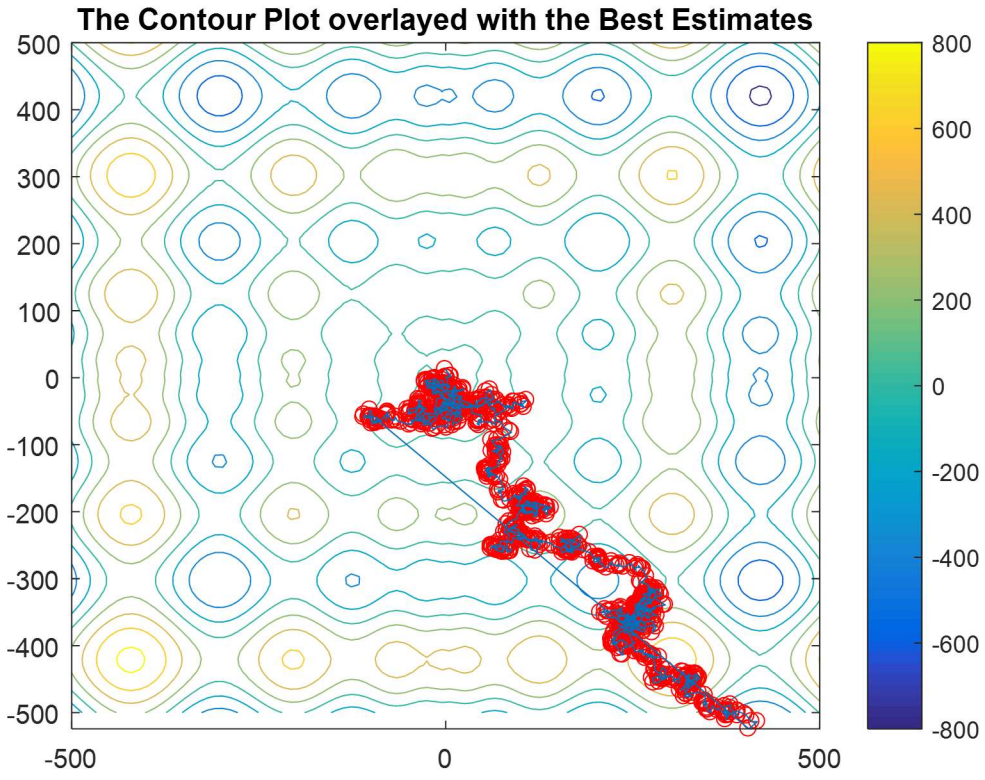

The Contour Plot overlayed with the Best Estimates
500
400
300
200
100
0
-100
-200
-300
-400
-500
-500
0
500
800
600
400
200
0
-200
-400
-600
-800

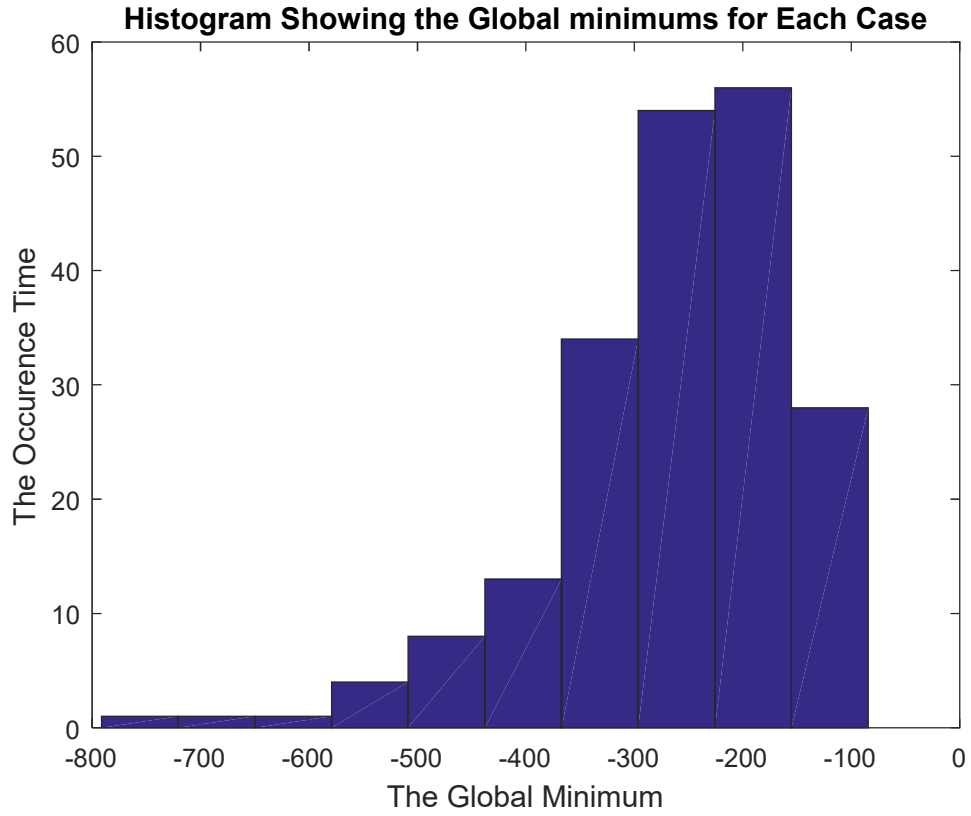

Histogram Showing the Global minimums for Each Case
15
10
5
0
The Occurence Time
-800
-700
-600
-500
-400
-300
-200
-100
The Global Minimum

2).N=200

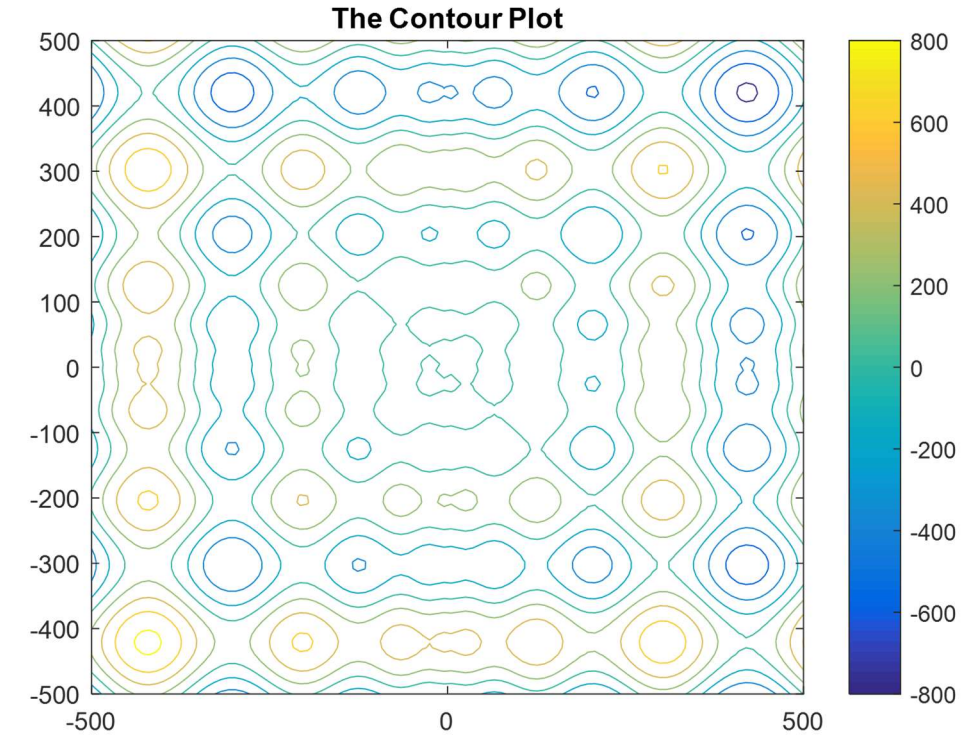

The Contour Plot
500
400
300
200
100
0
-100
-200
-300
-400
-500
-500
0
500
800
600
400
200
0
-200
-400
-600
-800

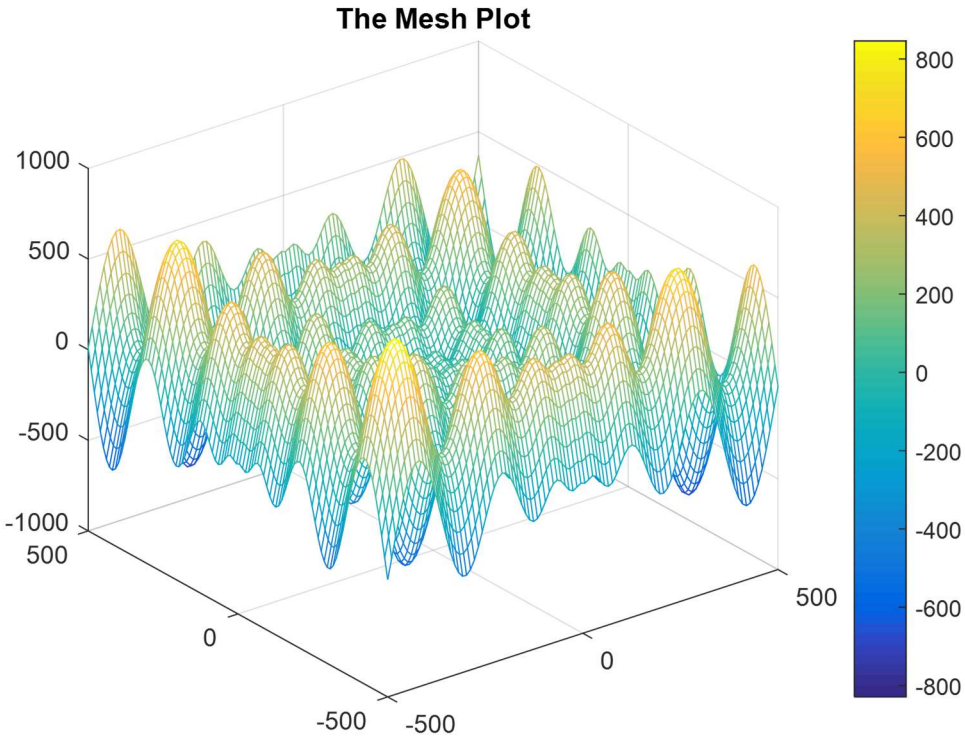

The Mesh Plot
1000
500
0
-500
-1000
500
0
-500
-500
0
500
800
600
400
200
0
-200
-400
-600
-800

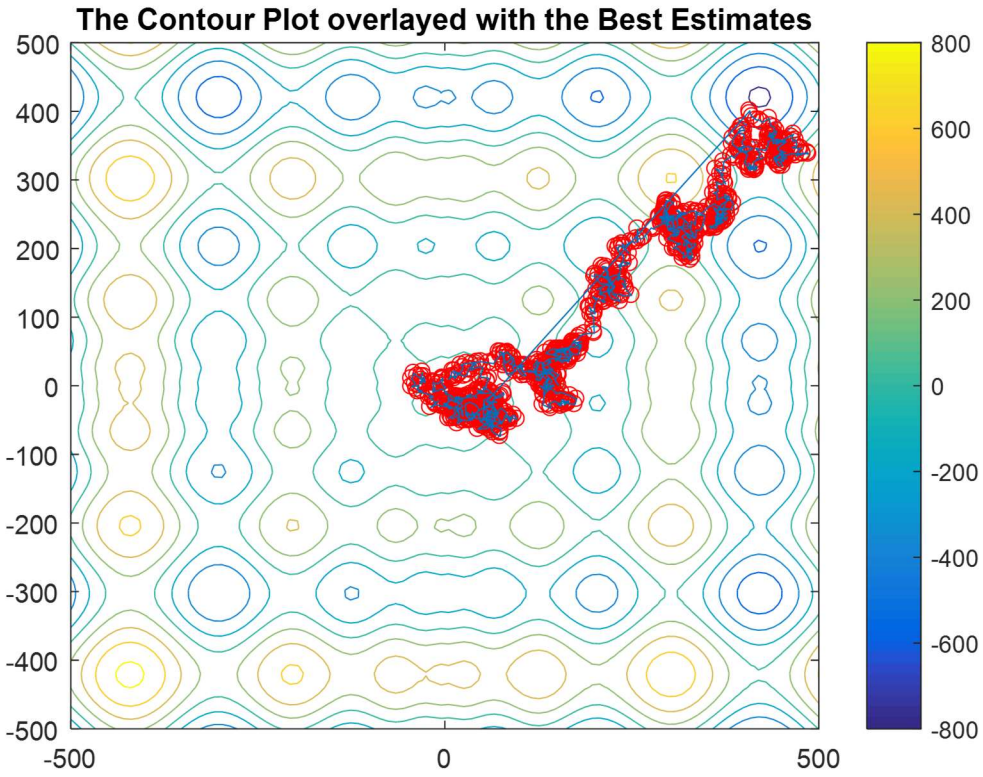

The Contour Plot overlayed with the Best Estimates
500
400
300
200
100
0
-100
-200
-300
-400
-500
-500
0
500
800
600
400
200
0
-200
-400
-600
-800

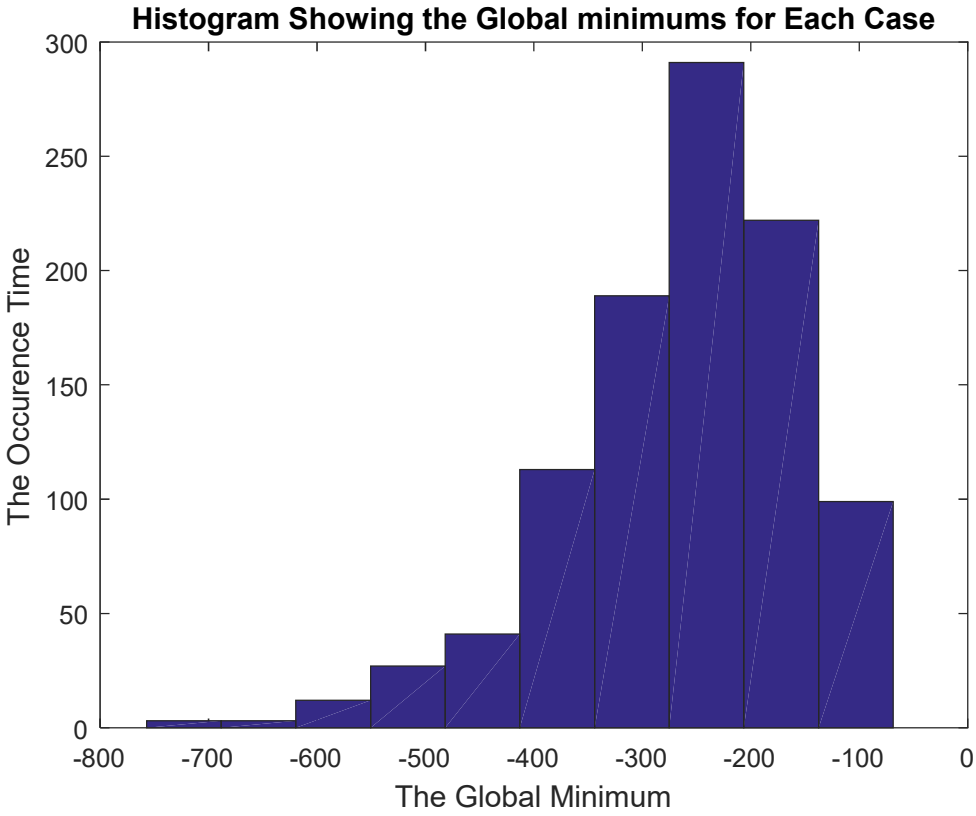

Histogram Showing the Global minimums for Each Case
The Occurence Time
60
50
40
30
20
10
0
-800
-700
-600
-500
-400
-300
-200
-100
0
The Global Minimum

3).N=1000


The Contour Plot
500
400
300
200
100
0
-100
-200
-300
-400
-500
-500
0
500
800
600
400
200
0
-200
-400
-600
-800

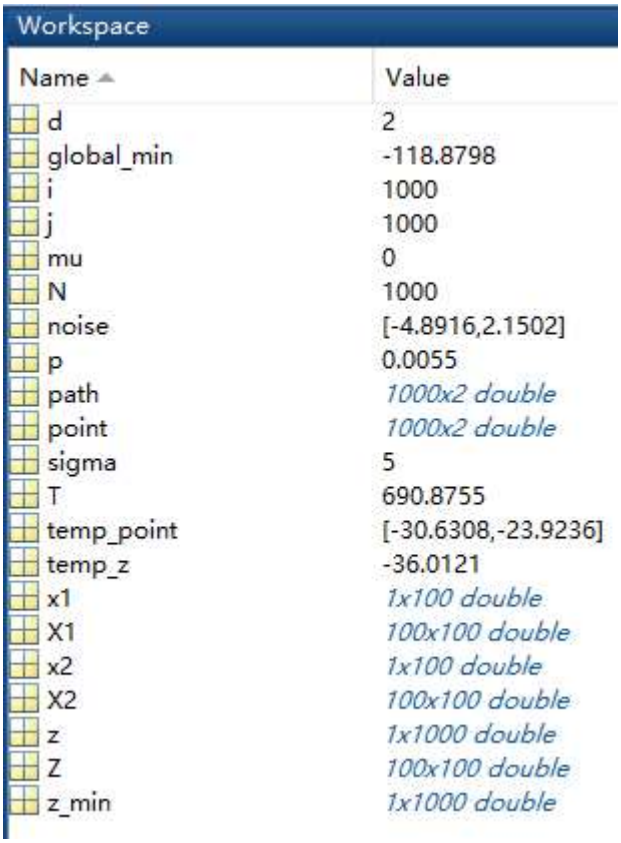

The Mesh Plot
1000
500
0
-500
-1000
500
0
-500
-500
0
500
800
600
400
200
0
-200
-400
-600
-800


The Contour Plot overlayed with the Best Estimates
500
400
300
200
100
0
-100
-200
-300
-400
-500
-500
0
500
800
600
400
200
0
-200
-400
-600
-800


Histogram Showing the Global minimums for Each Case
The Occurence Time
300
250
200
150
100
50
0
-800
-700
-600
-500
-400
-300
-200
-100
0
The Global Minimum

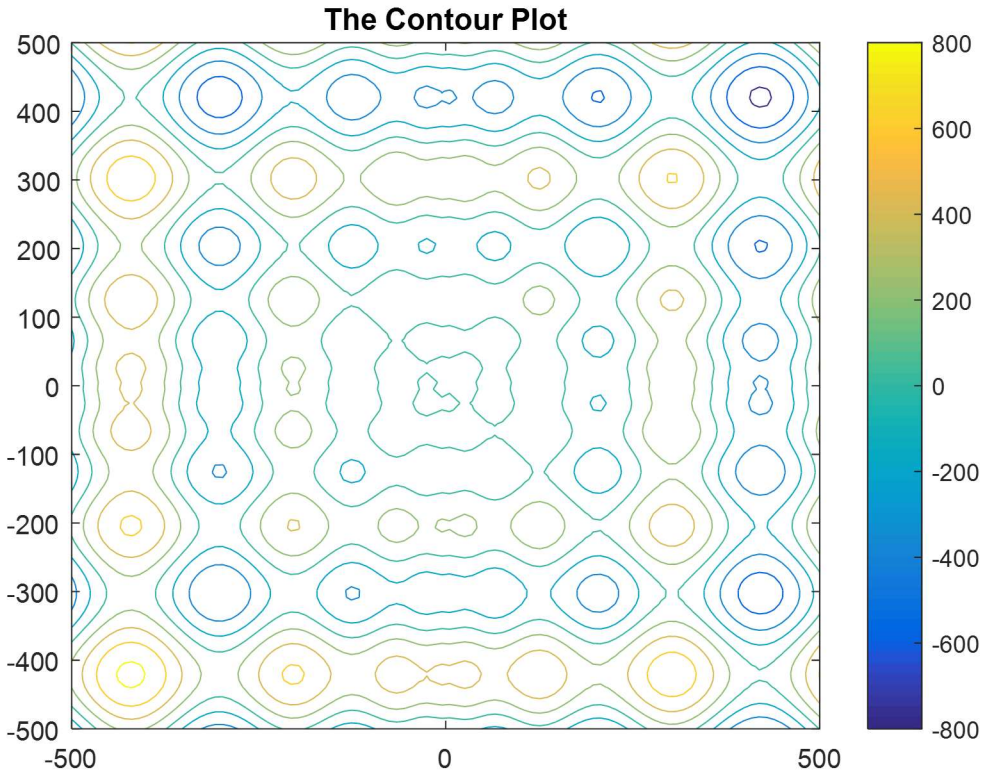
ii). Polynomial cooling schedule

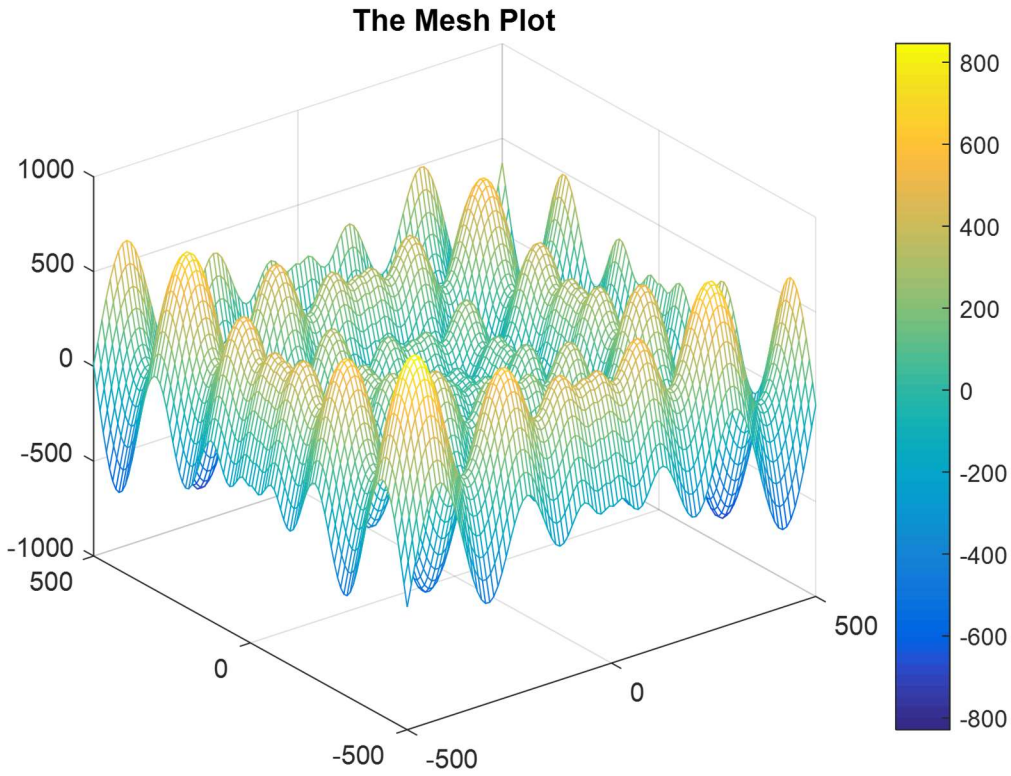
%%N=1000;

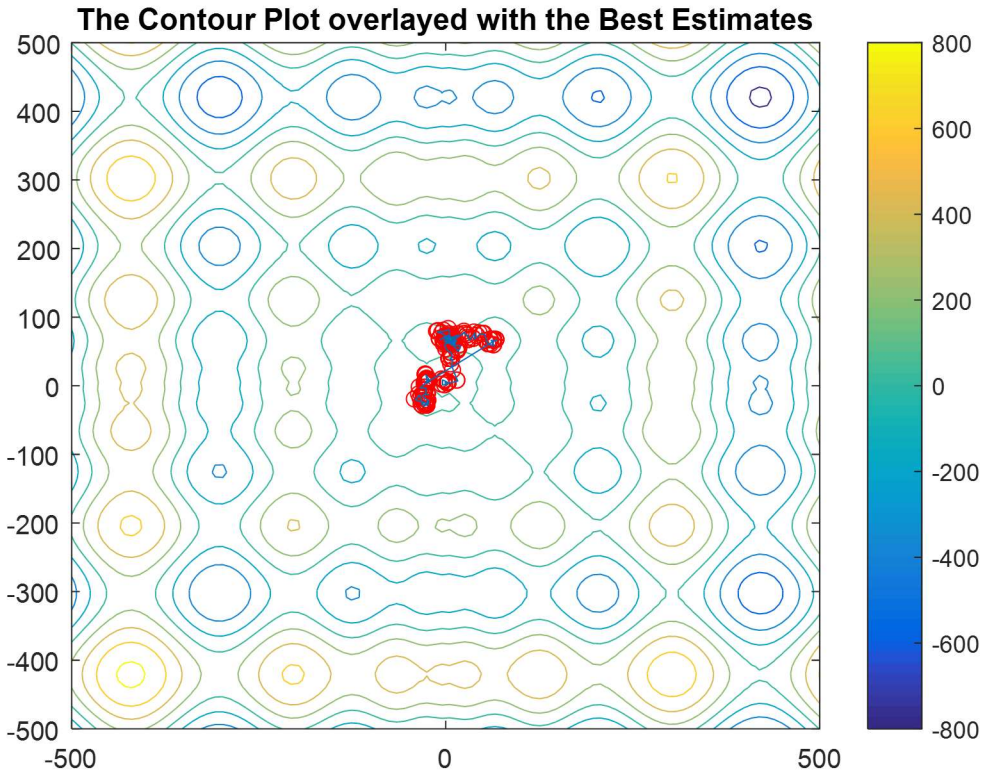


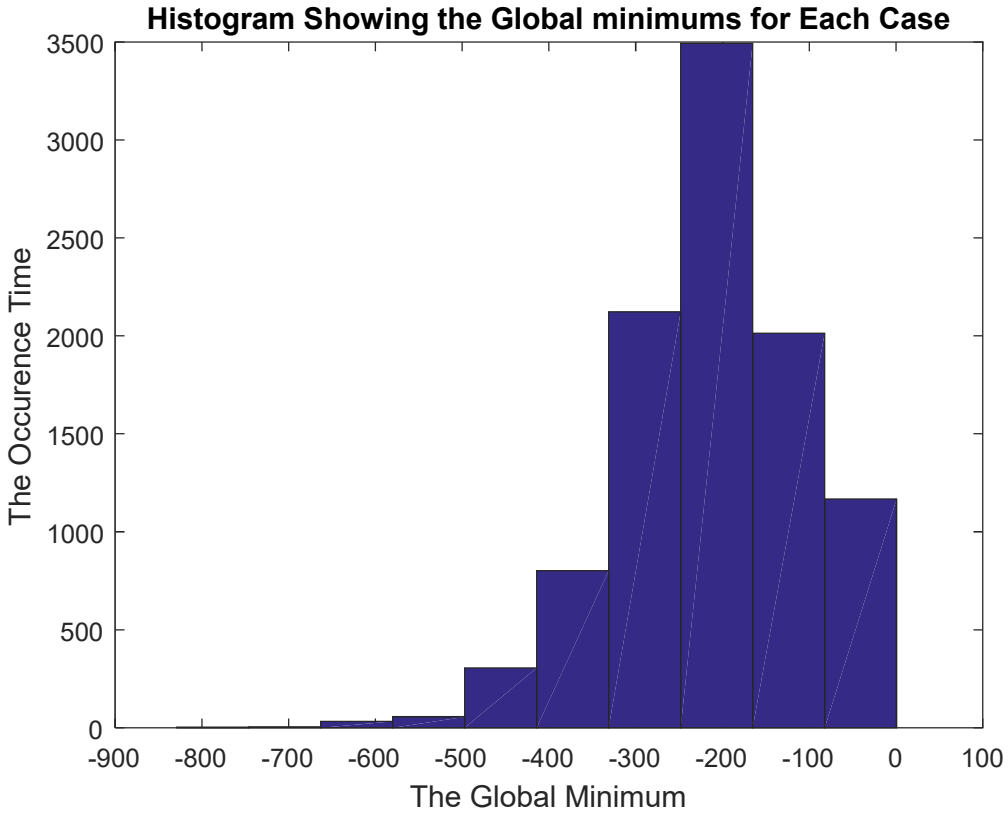
Workspace

Name	Value
d	2
global_min	-118.8798
i	1000
j	1000
mu	0
N	1000
noise	[-4.8916,2.1502]
p	0.0055
path	*1000x2 double*
point	*1000x2 double*
sigma	5
T	690.8755
temp_point	[-30.6308,-23.9236]
temp_z	-36.0121
x1	*1x100 double*
X1	*100x100 double*
x2	*1x100 double*
X2	*100x100 double*
z	*1x1000 double*
Z	*100x100 double*
z_min	*1x1000 double*


The Contour Plot
500
400
300
200
100
0
-100
-200
-300
-400
-500
-500
0
500
800
600
400
200
0
-200
-400
-600
-800

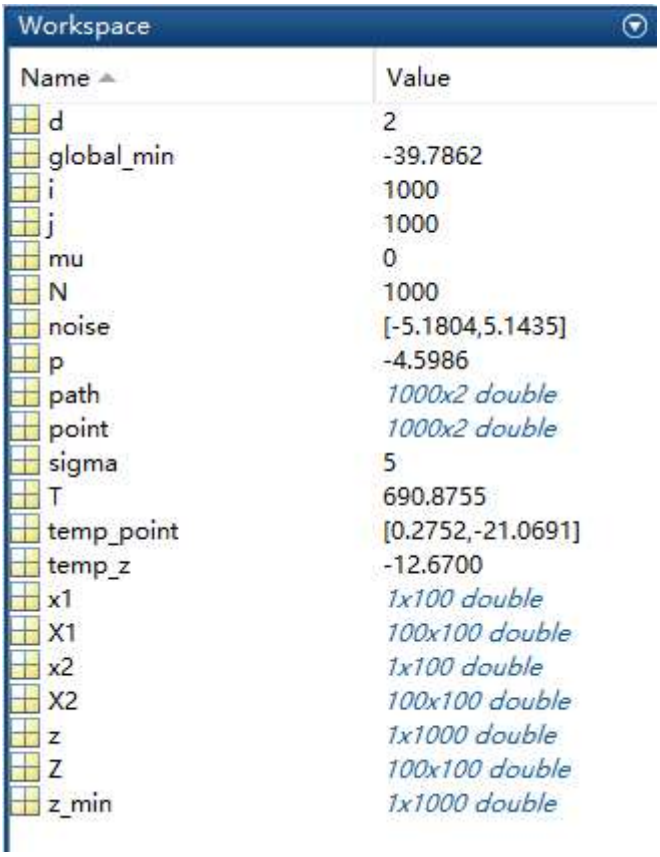

The Mesh Plot
1000
500
0
-500
-1000
500
0
-500
-500
0
500
800
600
400
200
0
-200
-400
-600
-800

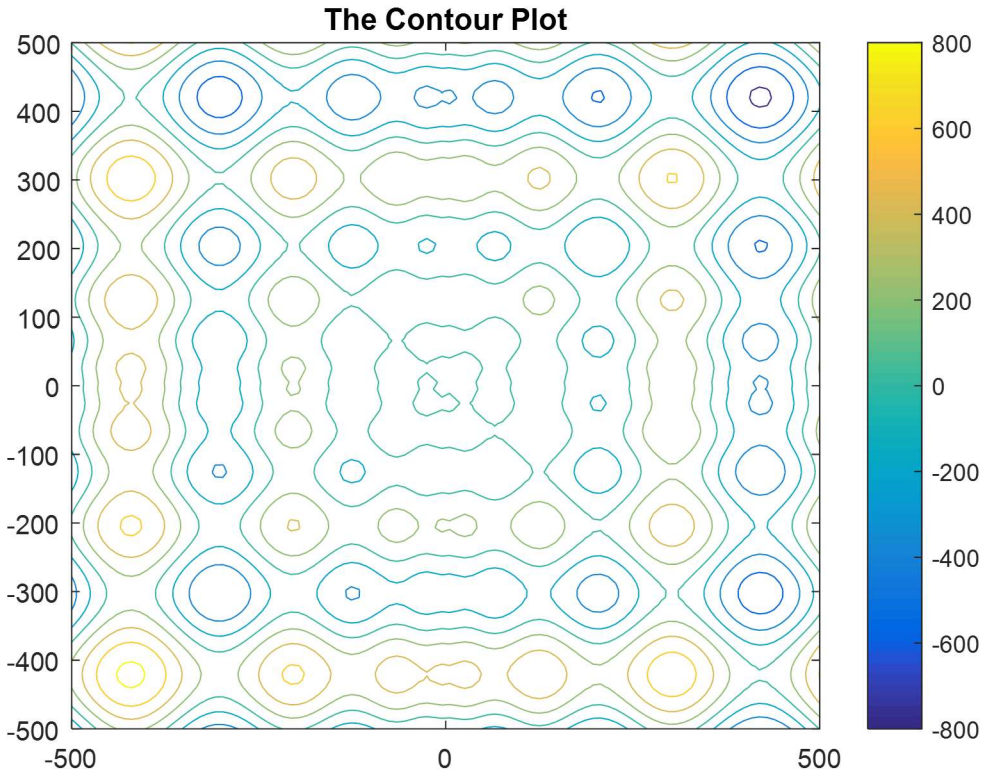

The Contour Plot overlayed with the Best Estimates
500
400
300
200
100
0
-100
-200
-300
-400
-500
-500
0
500
800
600
400
200
0
-200
-400
-600
-800

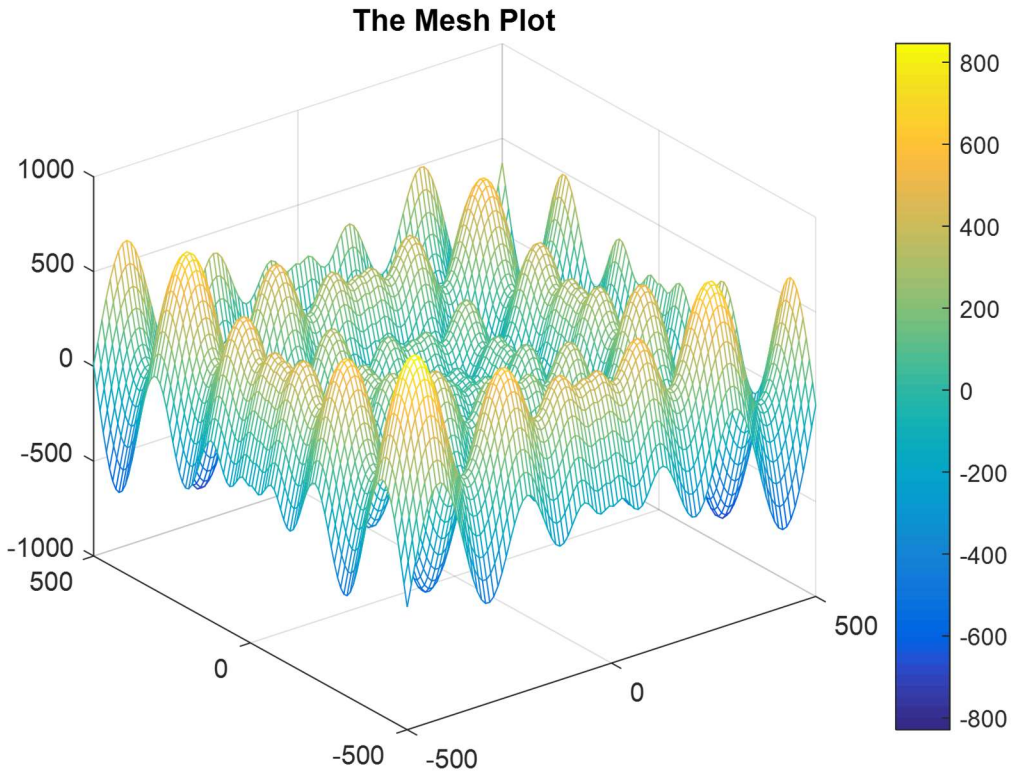

Histogram Showing the Global minimums for Each Case
The Occurence Time
3500
3000
2500
2000
1500
1000
500
0
-900
-800
-700
-600
-500
-400
-300
-200
-100
0
100
The Global Minimum

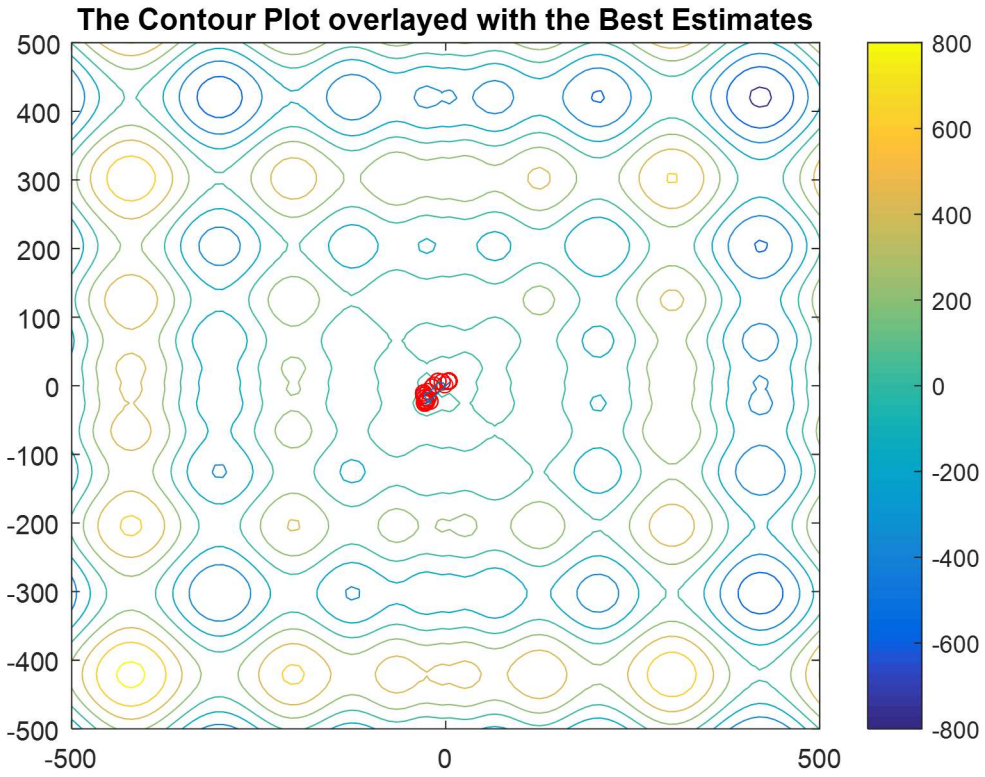
iii). Logarithmic cooling schedule

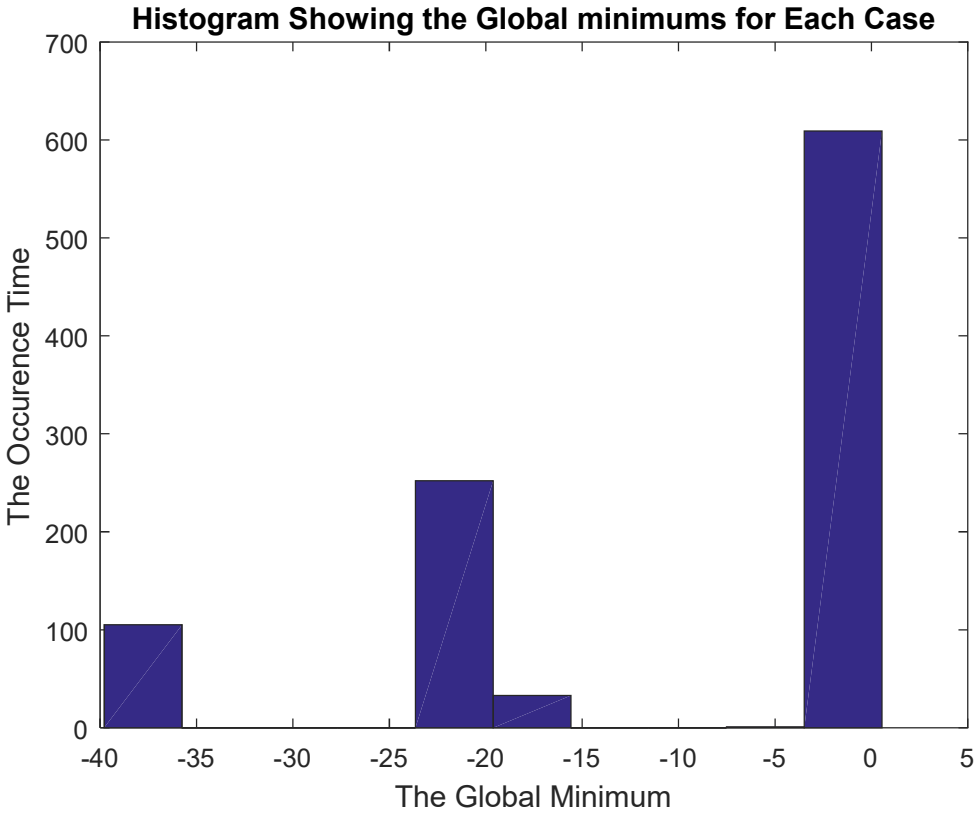
%%N=1000;




The Contour Plot
500
400
300
200
100
0
-100
-200
-300
-400
-500
-500
0
500
800
600
400
200
0
-200
-400
-600
-800


The Mesh Plot
1000
500
0
-500
-1000
500
0
-500
-500
0
500
800
600
400
200
0
-200
-400
-600
-800


The Contour Plot overlayed with the Best Estimates
500
400
300
200
100
0
-100
-200
-300
-400
-500
-500
0
500
800
600
400
200
0
-200
-400
-600
-800


Histogram Showing the Global minimums for Each Case
The Occurence Time
700
600
500
400
300
200
100
0
-40
-35
-30
-25
-20
-15
-10
-5
0
5
The Global Minimum

Finding:

1). It is shown in the simulation result that given the same value of iteration count N=1000.

The best estimate of global minimum for exponential cooling schedule is far from the origin (0,0).

This is because the convergence rate of exponential cooling schedule is large and the global minimum value is small.

The best estimate of global minimum for polynomial cooling schedule is near the origin (0,0).

This is because the convergence rate of polynomial cooling schedule is less the convergence rate of exponential schedule and the global minimum value is big.

The best estimate of global minimum for logarithmic cooling schedule is very close to origin (0,0).

This is because the convergence rate of logarithmic cooling schedule is the smallest and less than the convergence rate of exponential and polynomial cooling schedule, so the global minimum value is largest.

2). It is shown in the simulation result that given the same cooling schedule as the exponential cooling schedule.

The probability of avoid and the local minimum and convergence to the global minimum increase when the value of iteration count increase from 50 to 1000. This is because the global minimum decrease and the method successfully converges to the global minimum.

3). It is shown in the simulation result that given the same cooling schedule as the exponential cooling schedule.

The global minimum for the Schwefel function Is -706.2311 when iteration count N=50.

The global minimum for the Schwefel function Is -791.2658 when iteration count N=200.

The global minimum for the Schwefel function Is -757.2031 when iteration count N=1000.

4). It is shown in the simulation result that given the same value of iteration count N=1000.

The global minimum for the Schwefel function Is -757.2031 using exponential cooling schedule.

The global minimum for the Schwefel function Is -118.8798 using polynomial cooling schedule.

The global minimum for the Schwefel function Is -39.7862 using logarithmic cooling schedule.

The simulation results verify the conclusion that the convergence rate of exponential cooling schedule is largest since the global minimum is smallest, and the convergence rate of logarithmic cooling schedule is smallest since the global minimum is largest.

Experiment No.4

Description of Algorithm:

1.The simulated annealing method is applied to determine the minimal path in the country.

Here, the simulated annealing process is used to find the shortest path through the 48 state capitals of the contiguous United States.

1).Start with the selected cities.

2). Pick a new candidate tour at random from all neighbors of the existing tour. This candidate tour might be better or worse compared to the existing tour which is shorter or longer.

3). If the candidate tour is better than the existing tour, accept it as the new tour.

4). If the candidate tour is worse than the existing tour, still maybe accept it, according to some probability. The probability of accepting an inferior tour is a function of how much longer the candidate is compared to the current tour and the temperature of the annealing process. A higher temperature makes you more likely to accept an inferior tour.

5). Go back to step 2 and repeat many times, lowering the temperature a bit at each iteration, until you get to a low temperature and arrive at your (hopefully global, possibly local) minimum. If you're not sufficiently satisfied with the result, try the process again, perhaps with a different temperature cooling schedule.

The key to the simulated annealing method is in step 4: even if we're considering a tour that is *worse* than the tour we already have, we still sometimes accept the worse tour temporarily, because it might be the stepping stone that gets us out of a local minimum and ultimately closer to the global minimum.

2. The Euclidean distance is used to estimate the distance between two cities given (x, y) coordinates.

$$d(c_1, c_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Description of Method:

The steps to solve the TSP problem with simulated annealing is as followed:

1. Initialize the path $p^{(0)}$ and calculate its cost $l(p^{(0)})$.
2. Iteratively find the next path $p^{(t+1)}$ from $p^{(t)}$ via:
   - Generate a candidate path p' by swapping two cities in $p^{(t)}$ randomly.
   - If $l(p') \leq l(p^{(t)})$, then $p^{(t+1)} = p'$.
     Else $p^{(t+1)} = \begin{cases} p' & \text{with probability } q \\ p^{(t)} & \text{with probability } 1-q \end{cases}$
     where q decreases as the iteration goes.
     Common choices: $q = (t+1)^{\frac{l(p^{(t)}) - l(p')}{c}}$ or $q = e^{\frac{l(p^{(t)}) - l(p')}{c \cdot a^t}}$ with $a \in (0, 1)$.

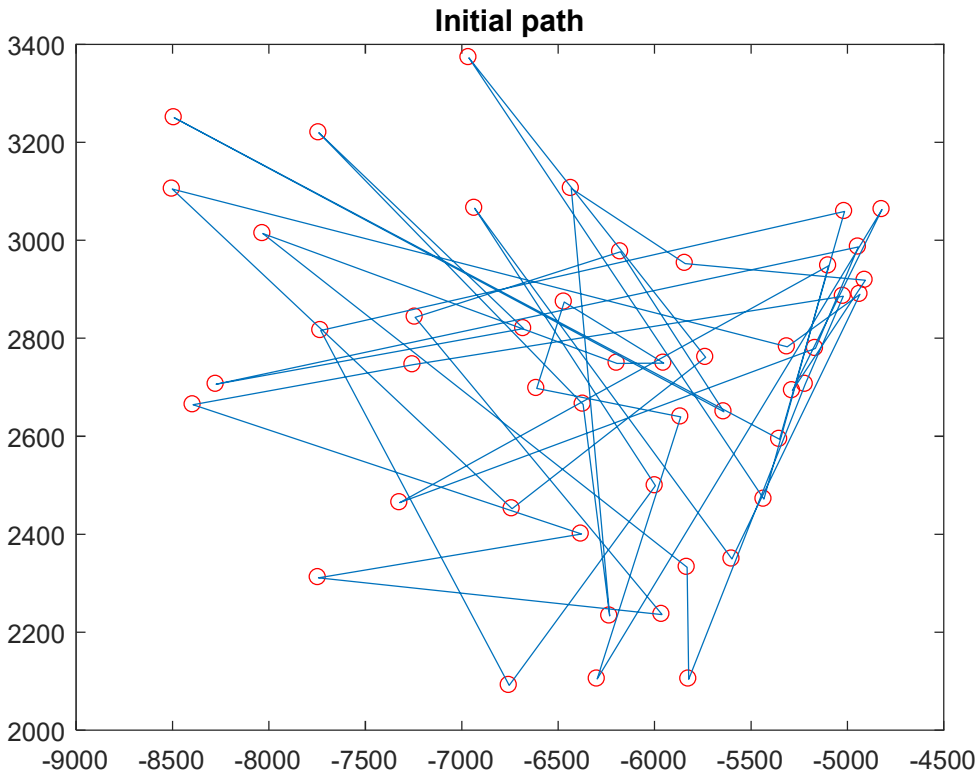
The distance of the shortest path is estimated. The best path is plotted on the x-y axis. The function of total tour distance is generated as the function of the simulation time.

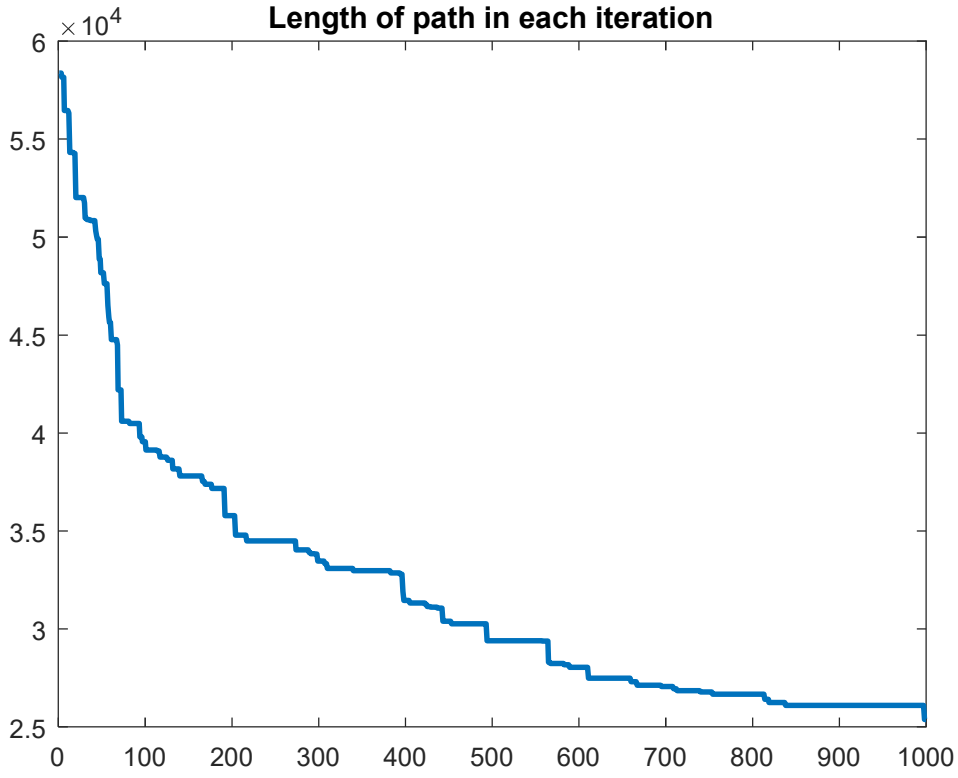
Code:

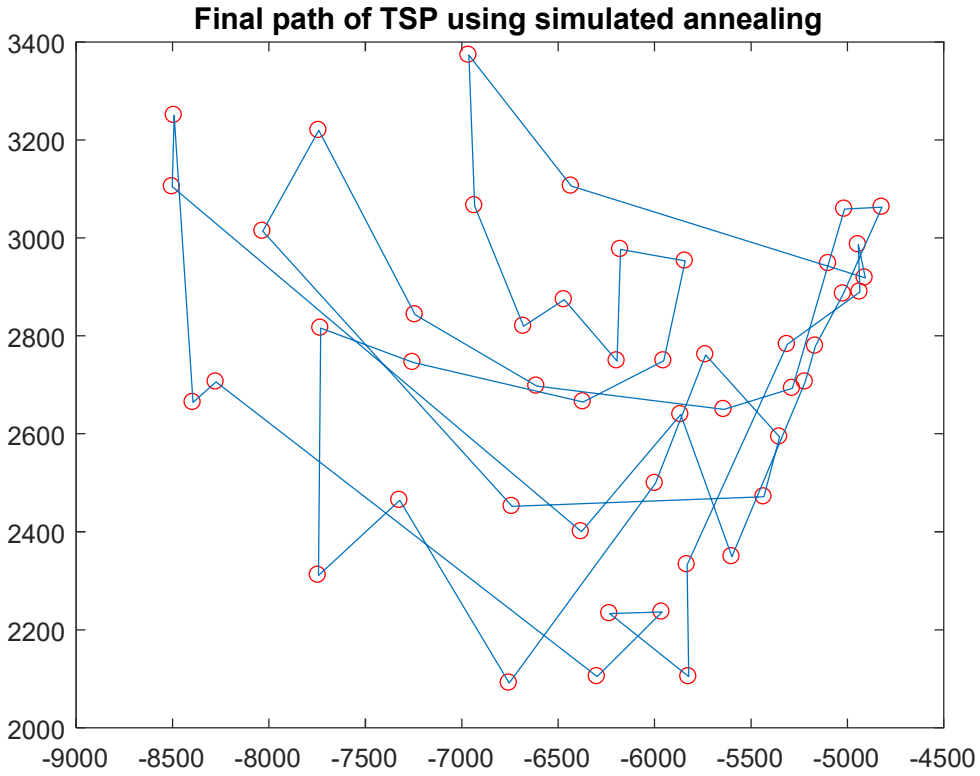
```
clear; clc; close all;
%import data%
city = importdata('uscap_xy.txt');
city = city([1 3:10 12:50],:);
states = importdata('uscap_name.txt');
n = 48;

% Simulated annealing algorithm
distance = pdist2(city, city);
% Parameters
num_iter = 1000; % number of iterations
c = 1;
% Initial path p
p = 1:n;
% 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([-9000 -4500]);
ylim([2000 3400]);
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([-9000 -4500]);
ylim([2000 3400]);
hold on
line(city([p(:); p(1)],1), city([p(:); p(1)],2));
title('Final path of TSP using simulated annealing');
hold off
```

Simulation Result:




Length of path in each iteration
×10^4
6
5.5
5
4.5
4
3.5
3
2.5
0
100
200
300
400
500
600
700
800
900
1000


Final path of TSP using simulated annealing
3400
3200
3000
2800
2600
2400
2200
2000
-9000
-8500
-8000
-7500
-7000
-6500
-6000
-5500
-5000
-4500

Finding:

The simulation inputs the files show the name and the coordinate of each city.

It is shown in the result that the length of path in each iteration decreases when the number of iteration increases. The final length of path is approximated $2.5*10^4$ km. Given the same convergence rate, the simulation time increase when the iteration count increases. Given the same number of iteration, the simulation time to estimate the global minimum decrease when the convergence rate increases.

The initial path and the final path are plotted on the x-y axis. It is shown in the result that the final path contains less routes and distance than the initial path. This is because the simulated annealing accepts the worse tour temporarily to avoid the local minimum and ultimately close to the global minimum. In the initial plot that an arbitrary initial tour is selected from the set of valid tours and the neighboring tour is chosen randomly, so the tour jumps all over the places and temperature is very high at the beginning of the annealing process. As the tours with shorter distance are accepted in the simulated annealing process and used to update the model. The temperature is reduced until we converge on the globally optimal tour.

Conclusion

Overall, the simulations of four experiments focus on Markov chains Monte Carlo theory and its application. The integral simulations are completed in the first experiment using stratification and importance sampling. The second trial simulates two conditional expectations using the Gibbs sampler. The third experiment uses simulated annealing method for Schwefel function to generate the contour plot of function surface and obtain the global minimum. The last experiment runs simulation to determine the minimal path and plot the distance as a function of the simulation time. In conclusion, the project uses mathematical methods to simulate Markov chain Monte Carlo system to solve optimization problems.