ECE 30200 Project 1

Name: Vaibhav Ramachandran

The task in the project was to develop and write a Monte Carlo Algorithm.

**Problem 1:**

The purpose of this problem was to generate 1000 random variables X(n) = (X1(n), X2(n)) where each X(n) represents random coordinates in the unit square S = [0, 1]2. In order to do this I made use of the rand() MATLAB function. So my command window looks like

N = 1000;

x = rand(N,2); %Generate 1000 pairs of random co-ordinates within the square [0,1]^2

**Problem 2:**

The purpose of this problem was to construct a set Ω using the equation,

h(x) = x13 + x22 + 2x1x2 – sin(x1) + cos(2πx1x2) – 1

In order to do this, I created a function called my\_h\_function.m which took the x vector previously created in problem 1 as the input and output a vector with N values which was then stored in h(x).

My my\_h\_function.m file looks like,

function output = my\_h\_function(x)

%Problem (2) Creates a function h(x) taking the vector x which contains N co-ordinates

output = x(:,1).^3 + x(:,2).^2 + 2\*x(:,1).\*x(:,2) - sin(x(:,1)) + cos(2\*pi\*x(:,1).\*x(:,2)) - 1;

end

Similarly, I created another function called my\_g\_function.m which also took the x vector previously created in problem 1 as the input and output a vector with N values which was then stored in g(x). The N values in g(x) were created according to the following formula, g(x) =

My my\_g\_function.m file looks like,

function output = my\_g\_function(x)

%Problem (2) Creates a function g(x) taking the vector x which contains N co-ordinates

output = exp(-(x(:,1).^2 + x(:,2).^2)/2)/(2\*pi);

end

I then proceeded to plot the region represented by g(x) and then highlighted the points where the value of h(x) > 0. So my command window looked like,

N = 1000;

x = rand(N,2);

h = my\_h\_function(x); %Problem (2) Creates a function h(x)

g = my\_g\_function(x); %Problem (2) Creates a function g(x)

scatter3(x(:,1),x(:,2),g,20,h>=0);

The main command was the scatter3 command which created a scatter plot of the N random coordinates using g(x) as the z coordinates and highlighting the points where h(x) > 0. As a result, I obtained a plot which looks like,

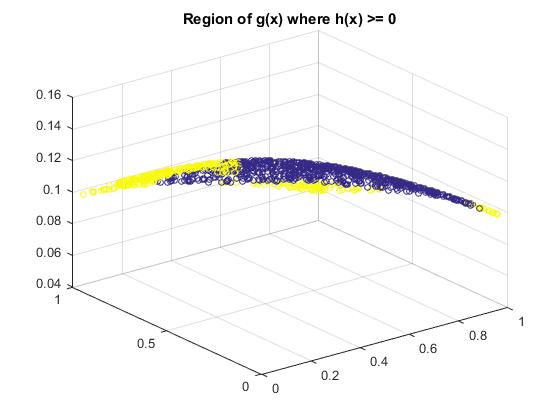


Figure 1: Result of problem 2. Plot of g(x). Blue colored points are the regions where h(x) > 0.

**Problem 3:**

The purpose of this problem was to create a program that would estimate the integral using the formula,

So I created a function called my\_Monte\_Carlo.m which calls my\_h\_function.m and my\_g\_function.m and uses the sum command to compute the summation. The my\_Monte\_Carlo.m file looked like,

function output = my\_Monte\_Carlo

%Problem (3) Evaluates the Monte Carlo integral

N = 1000;

x = rand([N,2]);

h = my\_h\_function(x);

g = my\_g\_function(x);

output = sum(g.\*(h>=0))/N; %Monte Carlo value for h(x) and g(x) for step 3

end

The estimate obtained for after running the function was **0.0315**.

**Problem 4:**

The purpose of this problem was to use the Monte Carlo algorithm previously developed for problem 3 and use it to estimate the value of . As a result, the following modifications were made to the my\_Monte\_Carlo.m function and I created a new function called my\_Monte\_Carlo\_for\_circle.m which looks like,

function output = my\_Monte\_Carlo\_for\_circle

%Problem (4) Estimates the value of pi using the Monte Carlo algorithm

N = 1000;

x = rand([N,2]);

h = (x(:,1).^2 + x(:,2).^2);

g = 1;

output = 4\*sum(g.\*(h>=0 & h<=1))/N;

%Multiplying by 4 since we're calculating pi using radius = 1/2. So in order to get the value of pi we multiply by 4.

end

The h(x) function was changed to represent a circle and the g(x) function was set to 1 in order to calculate the estimate of . The final output was also multiplied by 4 to get the value of since the radius of the circle now becomes ½. (Since the circle is enclosed in a square of side 1)

The estimate of obtained through this procedure was found to be **3.1560**.

**Problem 5:**

The purpose of this problem was to estimate the mean and standard deviation of the random variable I, which was calculated using the procedure outlined in problem 3. Thus 100 random values of N between 10 and 105 were chosen using the function Nset = round(logspace(1,5,100)).

For each value of Nset generated, 500 trials were taken to compute the estimate of I, resulting in a 500-by-100 matrix. Furthermore, for each value of Nset, the resulting mean and standard deviation were also calculated. The raw estimate I, the mean of I and the standard deviation of I were then plotted as a function of N. The semilogx command was used to plot the graphs. The resulting MATLAB file, named ECE\_30200\_Project\_1\_Part\_5.m looked like,

Nset = round(logspace(1,5,100));

for i = 1:100

for trial = 1:500

x = rand(Nset(i),2);

h = my\_h\_function(x);

g = my\_g\_function(x);

I(trial, i) = sum(g.\*(h >= 0))/Nset(i);

end

end

figure;

h1 = semilogx(Nset, I, 'kx');

hold on;

h2 = semilogx(Nset, mean(I), 'b', 'LineWidth', 2);

h3 = semilogx(Nset, mean(I) + std(I), 'r', 'LineWidth', 1);

h4 = semilogx(Nset, mean(I) - std(I), 'r', 'LineWidth', 1);

xlabel ('number of samples');

ylabel ('estimated integral');

legend ([h2, h3], {'mean', 'mean +/ - std'});

grid on

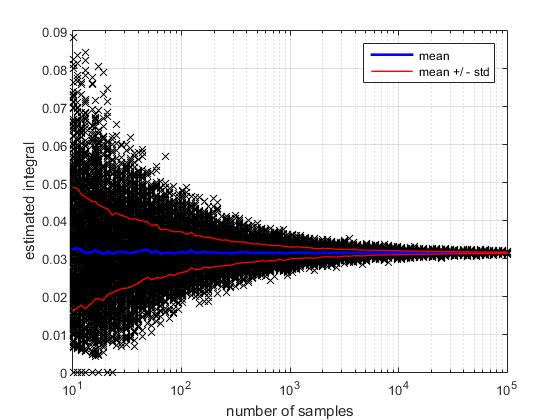
The resulting plot looked like,

Figure 2: Result of problem 5. Plot of raw estimate I, mean(I) and std(I) vs N.

The trend shown on the plot shows that as the number of samples increases, the standard deviation exponentially reduces and becomes nearly zero. As a result, the two red lines representing the difference between the standard deviation and the mean, converge at the mean. The value of the estimated integral is more concentrated around the mean as well, as the number of samples increases.

From inspection of the plot it can be seen that the minimum value of N required for the standard deviation to be within 5% of the mean is approximately **N = 800**.

**Problem 6:**

The purpose of this problem was to prove the following equation holds true,

In order to do this, I took a large enough value for N, N = 100000 and first estimated which was then used to estimate the value,

Finally, I plotted as a function of N and overlapped the curve with the empirical variance found in problem 5. The MATLAB code for this problem built off of the code for the previous question. The empirical variance was calculated using the var() function in MATLAB. The MATLAB file that I created, named ECE\_30200\_Project\_1\_Part\_6.m, thus looked like,

Nset = round(logspace(1,5,100));

for i = 1:100

for trial = 1:500

x = rand(Nset(i),2);

h = my\_h\_function(x);

g = my\_g\_function(x);

I(trial, i) = sum(g.\*(h >= 0))/Nset(i);

end

end

N = 100000;

x = rand(N,2); %Generate 100000 pairs of random co-ordinates within the square [0,1]^2

S = [0,1].^2;

h = my\_h\_function(x);

g = my\_g\_function(x);

E\_x\_squared = sum((g.^2).\*(h>=0))/N;

E\_x = sum(g.\*(h>=0))/N;

V = norm(S)\*(E\_x\_squared - (E\_x.^2)); %Calculated the value of V according to the above equation. Was found to be 0.0027.

figure;

Nset = round(logspace(1,5,100));

semilogx(Nset, V./Nset , 'LineWidth', 2);

hold on;

semilogx(Nset, var(I), 'r', 'LineWidth', 2); % I is calculated from Part (5)

legend ('true variance', 'empirical variance');

xlabel ('number of samples');

ylabel ('variance');

The plot that was thus obtained looked like,

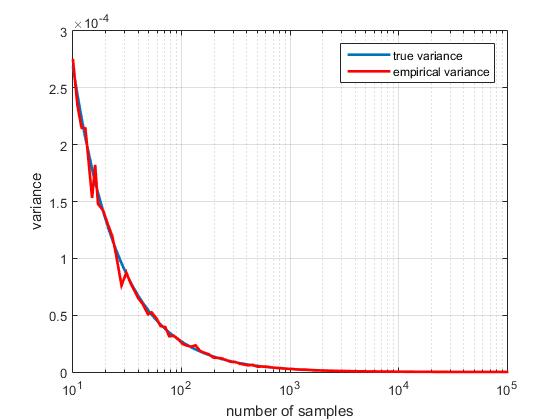


Figure 3: Result of problem 6. Plot of empirical variance and true variance vs number of samples

The difference between the two curves is that for the empirical variance line we only run through the Nset vector once for each point on the plot, whereas for the true variance line we run through the Nset vector 500 times for each point on the blue curve, which is why it has a much smoother transition as compared to the red curve which appears jagged at certain points. Running it more times for each point helps provide a more precise value which is why the blue line is called the true variance.