University of Michigan Oct. 6 2021

AE567

Statistical Inference, Estimation, and Learning

Project 1:

Stochastic Modeling, Monte Carlo Methods, and Variance Reduction

Deniz Dost

UM ID: 82337583

1. Warm-up: Life without a CLT

For the given PDF for X, the CDF is calculated by simply integrating the given function as shown below:

$$\int_1^x \frac{a}{x^{a+1}}\,dx = \int_1^x a x^{-(a+1)}\,dx = a\left[\frac{1}{-a}\,x^{-a}\right]_1^{\infty} = -x^{-a}\Big|_1^x = -\,x^{-a} + 1 = 1 - x^{-a}$$

Then, the inverse CDF is computed as shown below, where u is sampled from a uniform distribution.

$$\begin{aligned} u &= 1 - x^{-a} \\ x^{-a} &= 1 - u \\ x &= (1-u)^{-\frac{1}{a}} \end{aligned}$$

This inverse CDF is used with 1000 uniformly distributed random variables for 100 times. Monte Carlo (MC) estimate for these evaluations are calculated by taking the running estimate. The resulting running estimates for 100 sequences are shown below in Figure 1.

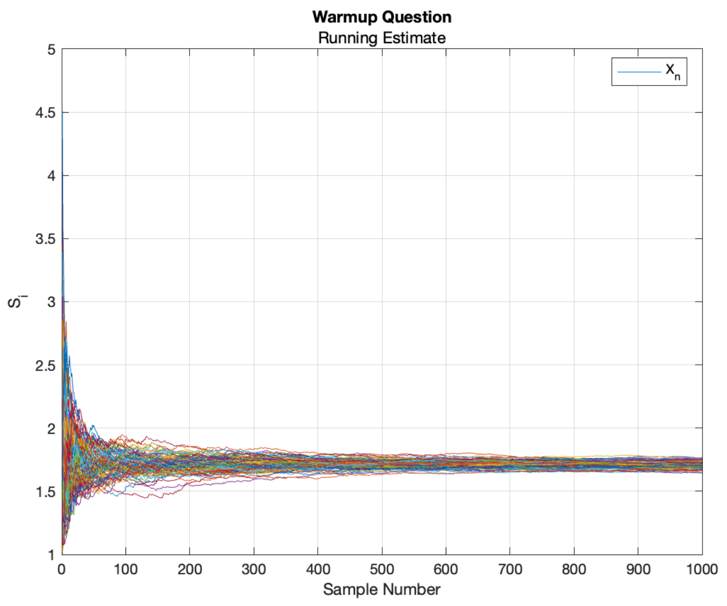

Figure 1

The variance of sequences taken over at each sample converges as shown below in Figure 2.

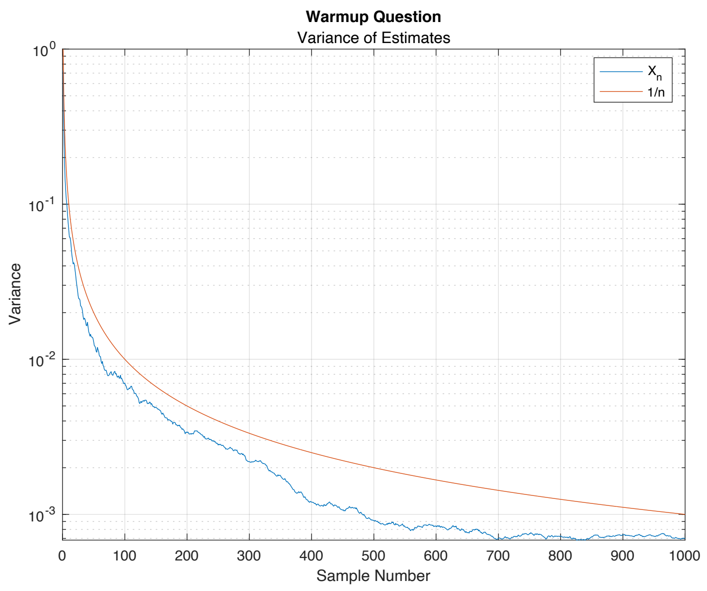

Figure 2

Basically, it follows the 1/n curve since the estimator has 1/n*variance. Also it is observed from the curve in the Figure 2, that the variance drops with increasing sample number. This can also be seen from the histogram of estimates in Figure 3.

The purple distribution which represents the estimates at 200 samples is much more wider than the red distribution which represents the estimates at 1000 samples. This is also intuitively expected.

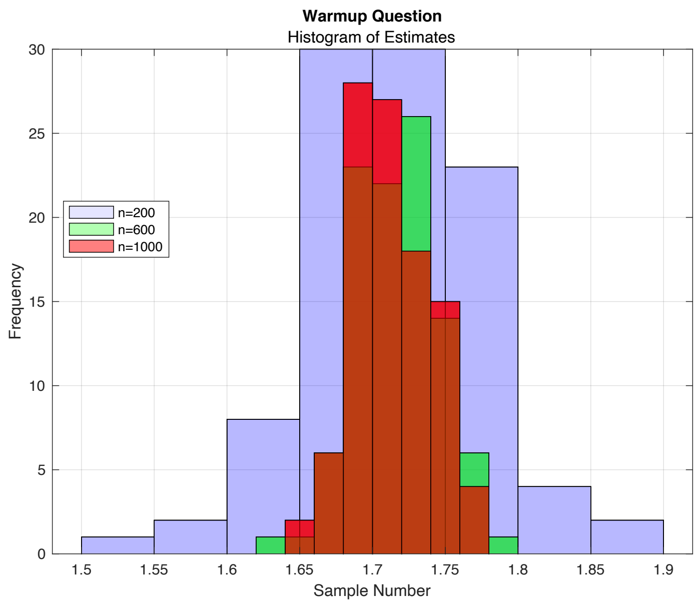

Figure 3

See Appendix 1 for the code for this problem.

2. Importance Sampling for Random Walks

2.1. 1-D Bernoulli Random Walk

Questions 2.1

(a)

In order to simulate the 1-D random walk, a set of steps (either 1 or -1) is created by sampling from a uniform distribution. To assign equal probability to positive and negative steps, a threshold value of 0.5 is used. Random variables above that value are assigned 1, and random variables below that value are assigned -1. Then, the cumulative sum of these steps are taken to find the path travelled. Figure 3 shows an example of a 1D Gaussian walk with 100 steps.

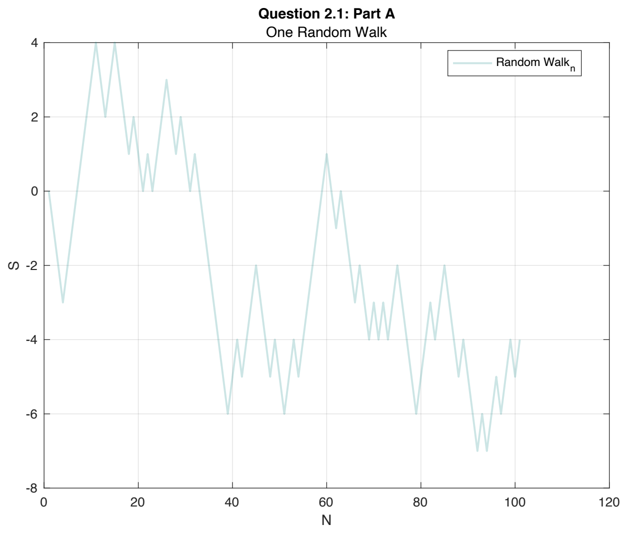

Figure 3

Then, 1000 1D walks are created and the paths are visualized as shown in Figure 4.

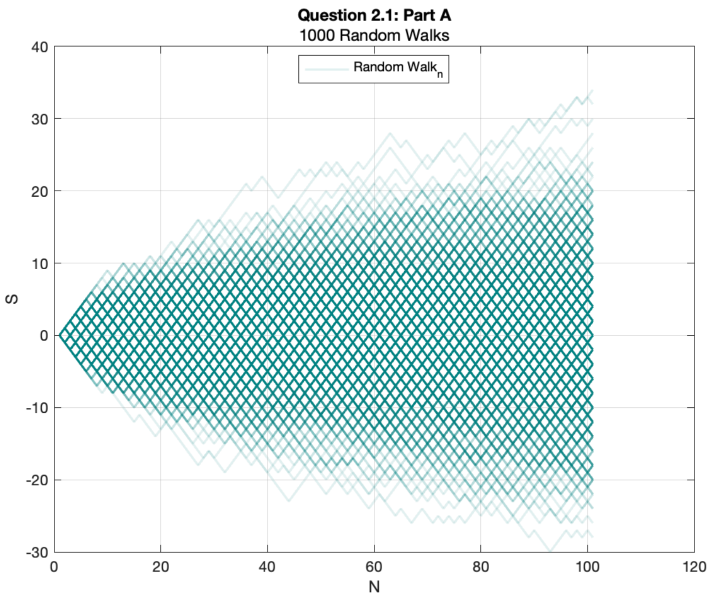

Figure 4

The distance arrived at the end of each path is the S value at N=100 and the distribution of these distances for 1000 steps can be seen in the histogram in Figure 5. This distribution is shaped like a symmetrical binomial distribution as expected with 0.5 p value since there are only two possible outcomes, 1 or -1.

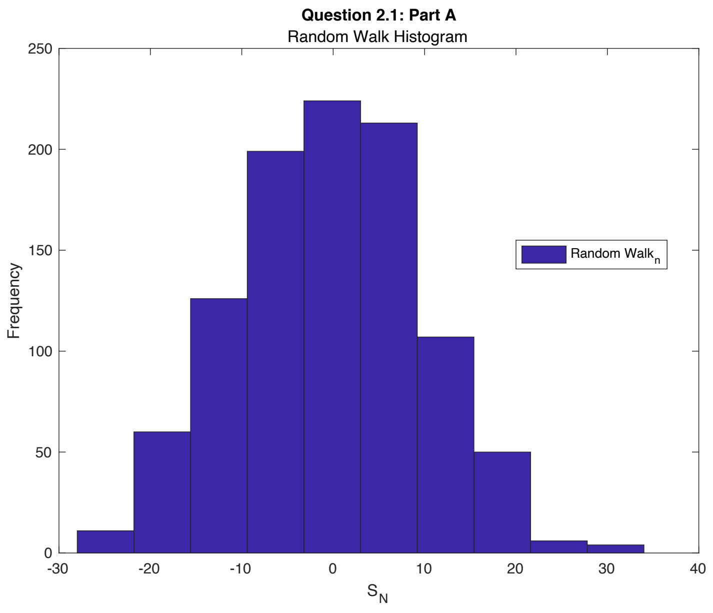

Figure 5

(b)

MC method is used for 100000 trials and the computed probability for P(S>10) is 0.1353. This can also be qualitatively observed from the histogram in Figure 6.

$$P(S > 10) = 0.1353$$

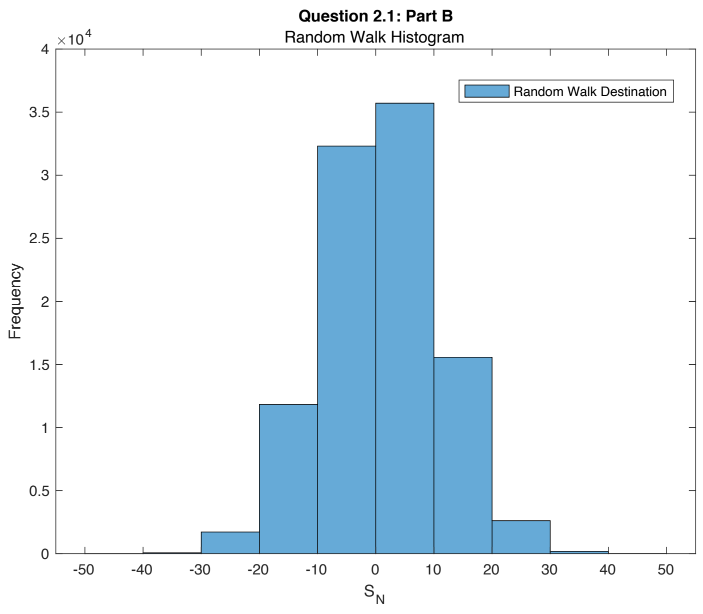

Figure 6

(c)

In order to find this probability, importance sampling is used. Proposed function for this exercise is sampling from the uniform distribution. However, this time the threshold value is selected to be 0.2 so that the walker would take 20% (-1) steps and 80% (+1) steps and the ratio of the original and proposed function (likelihood) is computed with relevant probabilities. The proposed function results in destination values distributed as shown in Figure 7. The proposal function clearly has its mean shifted more towards to the threshold since 55 is not reached frequently with the original distribution. Then, the evaluator function evaluates whether the arrived distance by the proposed distribution is in the desired range. These are used to compute the importance sample MC estimate and resulted value is:

$$P(S > 55) = 7.9526e - 09$$

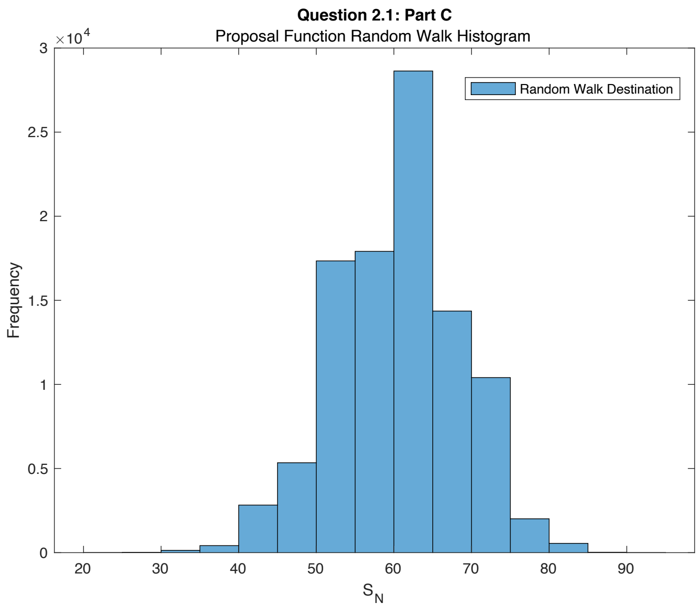

Figure 7

(d)

The analytical expressions are derived as follows:

$$Minimum\ number\ of\ steps\ required\ for\ S > 10 = 10 + \frac{100 - 10}{2} + 1 = 56$$

$$P(S > 10) = \frac{1}{2^{100}} \sum_{k=56}^{100} \binom{100}{k} = 0.1356$$

$$Minimum\ number\ of\ steps\ required\ for\ S > 55 = ceil(55 + \frac{100 - 55}{2}) = 78$$

$$P(S > 55) = \frac{1}{2^{100}} \sum_{k=78}^{100} \binom{100}{k} = 7.9527e - 09$$

These analytical values match with the MC values calculated in (b) and (c).

(e)

(i)

MC standard error can be calculated as

$$std_X[S_n[g]] = \frac{std_x[g]}{\sqrt{n}}$$

The corresponding values are 0.00108 for S>10, and 6.00417e-11 for S>55. Then, the 95% confidence interval is calculated where the upper and lower bounds are set in order to contain 95% of the data. In order to do that, z score which is defined as below is used.

$$H_n = \frac{S_n - \mu}{\sigma/\sqrt{n}}$$

$$P_X(-z \leqslant H_n \leqslant z) = P_X(-z \leqslant \frac{S_n - \mu}{\sigma/\sqrt{n}} \leqslant z) = 0.95$$

$$S_n = \mu z \pm ste_{MC}$$

where z=1.96 for 95%

$$(S > 10)_n^{upper\ bound} = 0.1332$$

$$(S > 10)_n^{lower\ bound} = 0.1375$$

$$(S > 55)_n^{upper\ bound} = 0.8070e - 08$$

$$(S > 55)_n^{lower\ bound} = 0.7835e - 08$$

(ii)

$$P(0.1332 \leqslant P(S > 10)_{analytical} \leqslant 0.1375) = 0.941$$

$$P(0.7835e - 08 \leqslant P(S > 55)_{analytical} \leqslant 0.807e - 08) = 0.954$$

So the interval limits hold.

(iii)

Taking the running mean of all steps each time step for 1000 trials results in a distribution as shown in Figure 8 for S>10 and in Figure 9 for S>55.

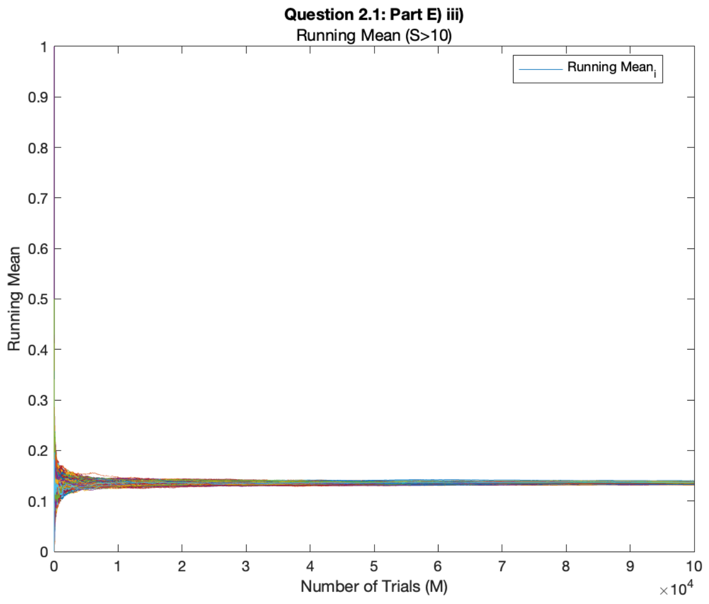

Figure 8

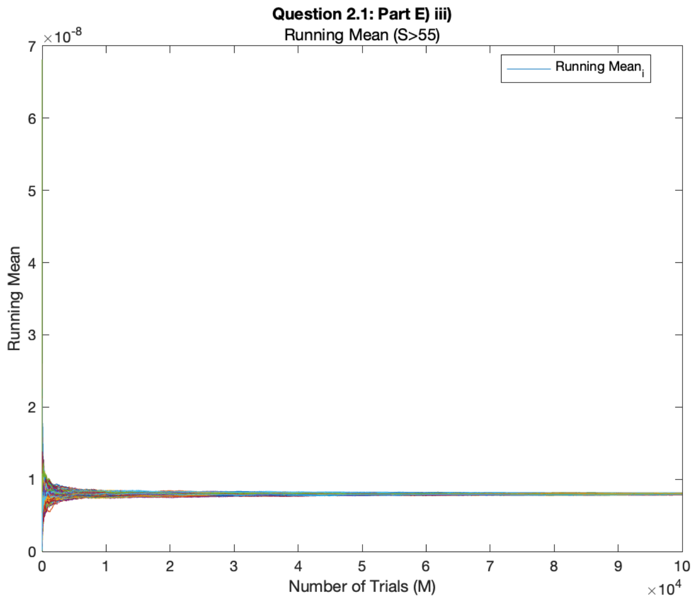

Figure 9

And the envelope created from the minimum and maximum values at each step creates an envelope as below, Figure 10 for S>10 and Figure 11 for S>55.

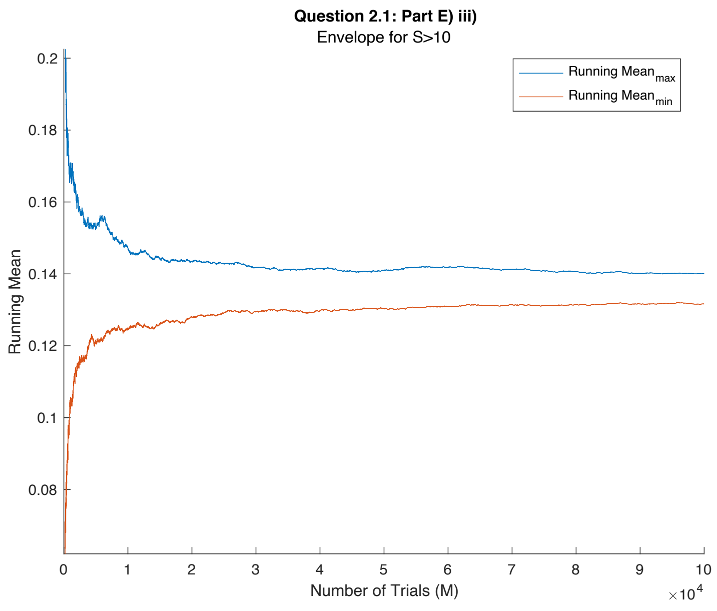

Figure 10

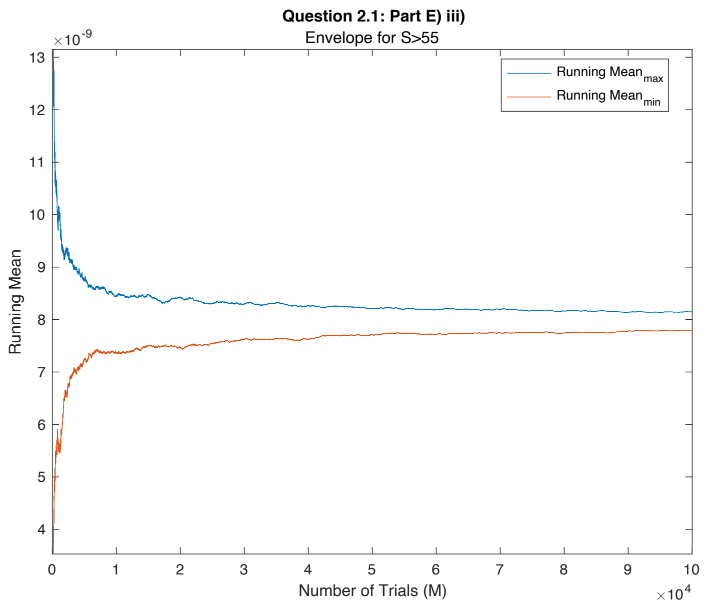

Figure 11

See Appendix 2 for the code for this problem.

2.2. 3-D Gaussian Random Walk

Questions 2.2

(a)

A 3-D Gaussian random walk is generated with 100 steps taken according to a standard normal distribution in all three axis. The histogram of the final distance at the end, step=100, is shown below in Figure 12.

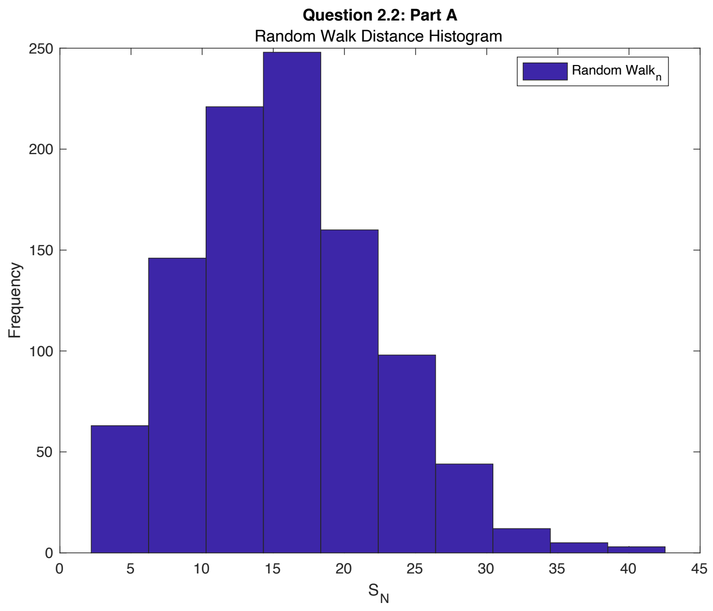

Figure 12

(b)

MC method is used for 100000 trials and the computed probability for P(S>10) is 0.1353. This can also be qualitatively observed from the histogram in Figure 13.

$$P(|S| > 10) = 0.79989$$

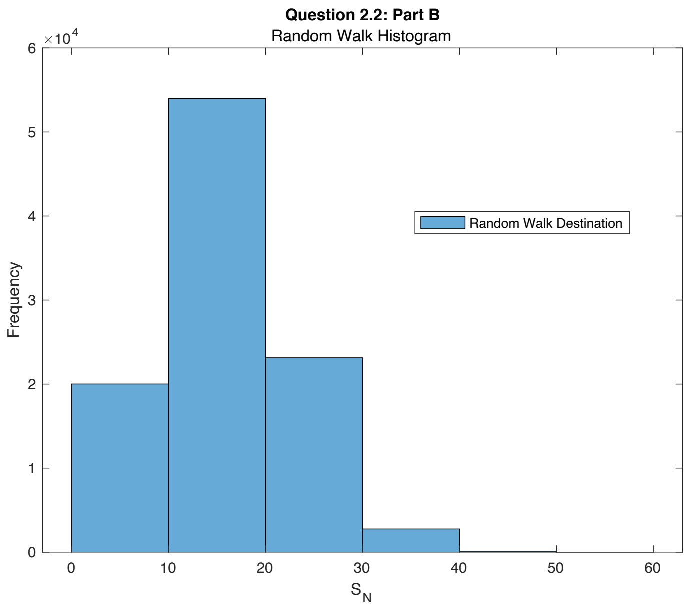

Figure 13

(c)

In order to find this probability, importance sampling is used. Proposed function for this exercise is sampling from the normal distribution. However, this time the mean is selected to be 0.25. The normal curves shifted to the right by this value, so that the walker would take less (-1) steps than (+1) steps and the ratio of the original and proposed function (likelihood) is computed with relevant probabilities. The proposed function results in destination values distributed as shown in Figure 14. The proposal function clearly has its mean shifted more towards to the threshold which is desired. Then, the evaluator function evaluates whether the the arrived distance by the proposed distribution is in the desired range. These are used to compute the importance sample MC estimate and resulted value is:

$$P(|S > 55|) = 2.4148e - 07$$

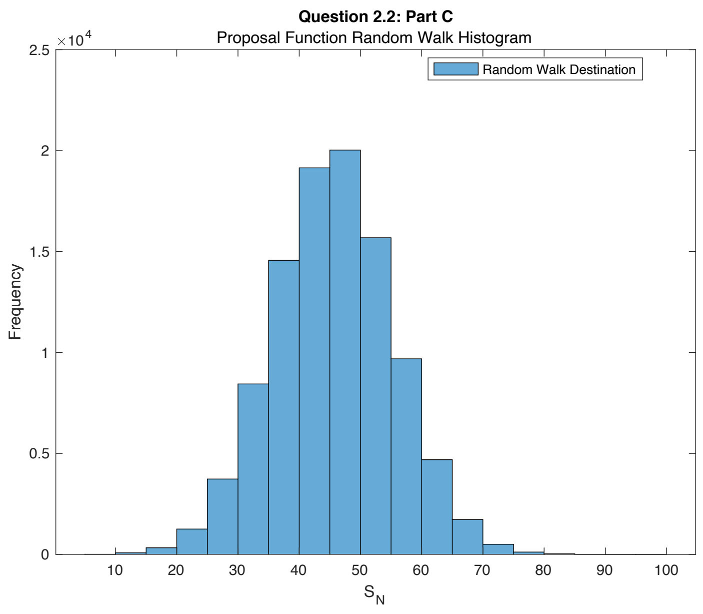

Figure 14

The analytical expression for $P(|S| > 55)$ can be derived as follows:

$$P(|S| > threshold) = P(\sqrt{S_x^2 + S_y^2 + S_x^2} > threshold)$$

$$P(|S|^2 > threshold^2) = P((S_x^2 + S_y^2 + S_x^2) > threshold^2)$$

$$S_X, S_y, S_z \sim N(0,N) \ \text{, } where\ N = 3$$

$$\sum_{i=1}^{N=3} X_i^2 \ \text{, } where\ X_i \sim N(0,\sigma^2) \ \text{, } \sigma^2 = N$$

$$then\ \text{, } Y \sim \Gamma(\frac{N}{2}, 2\sigma^2)$$

$$P(|S| > threshold) = P(\Gamma(\frac{3}{2}, 2N > threshold^2)) = 1 - P(\Gamma(\frac{3}{2}, 2N \leqslant threshold^2))\text{, } where\ N = 100$$

$$P(|S| > 10) = 0.8012$$

$$P(|S| > 55) = 1.2226e - 06$$

If the 3-D case is considered, where there a 8 octants in a sphere where the walker can arrive at, the analytical probability can be multiplied by 8. Since the proposed distribution is shifted to positive walks. This may not be the true approach for the analytic solution. The obtained probability is now 9.7812e-06. These probability values are consistent with the MC outputs.

(d)

As shown above in the 1-D walk case, MC standard error is calculated. The corresponding values are 0.0013 for |S|>10, and 3.0482e-08 for |S|>55. The confidence intervals are also calculated as below:

$$(|S| > 10)_n^{upper\ bound} = 0.8009$$
$$(|S| > 10)_n^{lower\ bound} = 0.7989$$

$$(|S| > 55)_n^{upper\ bound} = 0.241478e - 06$$
$$(|S| > 55)_n^{lower\ bound} = 0.241477e - 06$$

See Appendix 3 for the code for this problem.

3. Multilevel Monte Carlo and Control Variates for Stochastic ODEs

Question 1

The Wiener Process is defined in the problem statement and since $W(t)$ is defined to have Gaussian increments $\Delta W_n = W(t_{n+1}) - W(t_n)$ is also a Gaussian distribution.

$$W(t+\Delta t) = W(t) + \sqrt{\Delta t}\xi\,,\ where\ \xi \sim N(0,1)$$
$$W(t+\Delta t) \sim N(0,\Delta t)$$

3.1.

In order to create a geometric brownian motion, first a set of 100 Wiener Processes are created sampling from a normal distribution according to the following formula.

$$W^n(t) = \frac{1}{\sqrt{n}} \sum_{1 \leqslant k \leqslant [nt]} X_k\ ,\ where\ n = \frac{t_n}{\Delta t}\ and\ t_n = 1$$

This process is the input for brownian motion's diffusion part. For a time step value of $\Delta t = 0.05$ and $0 \leqslant t \leqslant 1$, and the drift and diffusion parameters as 0.05 and 0.2 respectively, a Wiener distribution is created. Figure 15 shows this process.

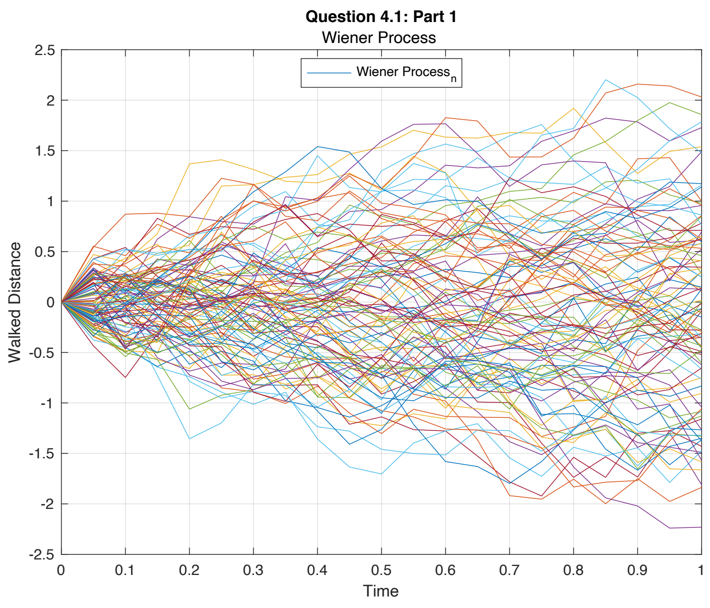

Figure 15

Then, Euler-Maruyama scheme is used to create the geometric Brownian motion as shown in Figure 16.

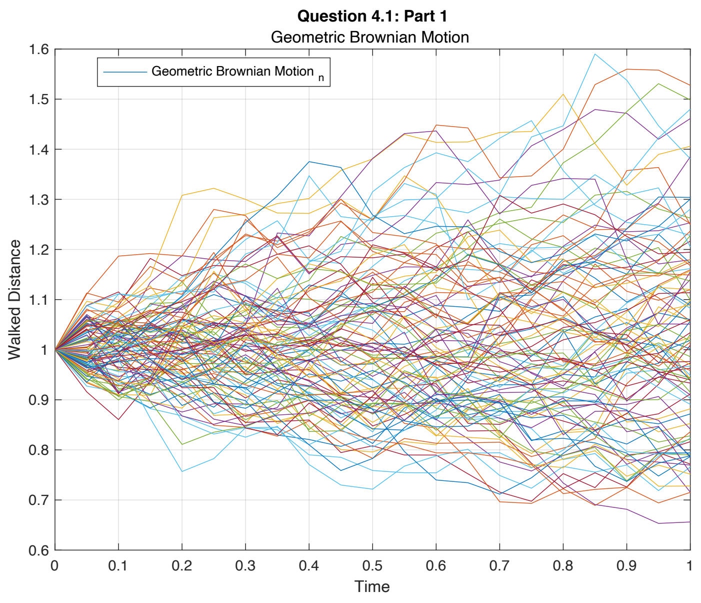

Figure 16

MC method created for this problem resulted in a variance of 0.0385 and an expectation of 1.0326 for Y(1).

The analytical solution is also obtained as follows:

$$E[Y(1)] = E[Y_0 e^{\mu - \frac{\sigma^2}{2} t + \sigma W_t}] \ \ , where\ t = 1$$

$$E[Y(1)] = Y_0 e^{\mu - \frac{\sigma^2}{2}} + E[e^{\sigma W_1}] \ \ , where\ Y_0 = 1$$

From $E[e^x] = e^{\mu_x + \frac{\sigma_x^2}{2}}$,

$$\Delta W \sim N(0, \Delta t)$$

$$W_1 \sim N(0, N\Delta t)$$

$$\sigma^2 W_1 \sim N(0, \sigma N \Delta t)$$

$$E[Y(1)] = e^{\mu - \frac{\sigma^2}{2}} e^{\frac{\sigma^2 N \Delta t}{2}} \ \ , where\ N = \frac{1}{\Delta t}$$

$$E[Y(1)] = e^{\mu - \frac{\sigma^2}{2} + \frac{\sigma^2}{2}} = 1.0513$$

Also,

$$Since\ \ Var(Y) = E[Y^2] - E[Y]^2$$

$$Var(Y(1)) = e^{2\mu - \sigma^2 + 2\sigma^2 N \Delta t} - e^{2\mu - \sigma^2 + \sigma^2 N \Delta t} = 0.0451$$

So the analytical solutions are consistent with the MC outputs.

3.2.

In order to examine Brownian motion with different timescales, Δt and $4\Delta t$, first a function for Wiener Process is defined. Since these fine and coarser motions will have the same statistics, fine scale motion is used to create coarse scale motion. This function takes the Wiener process values form the finer set and creates a ΔW from designated intervals. Therefore, a coarser Wiener process is created as shown below in Figure 17.

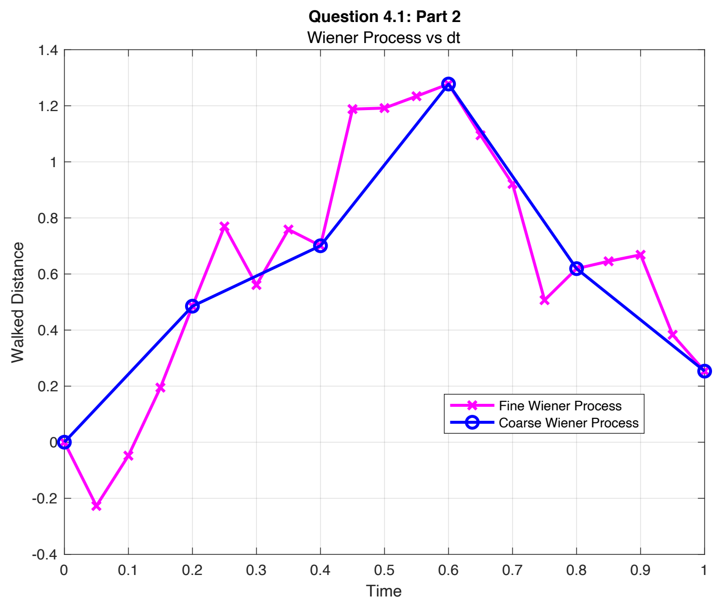

Figure 17

Using these coarser Wiener process values in the Euler-Maruyama scheme, a coarser Brownian walk is created as shown below in Figure 18.

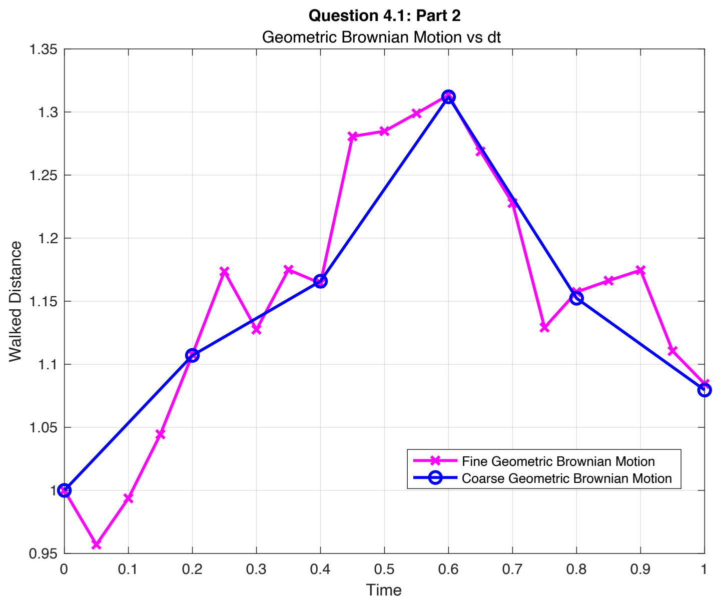

Figure 18

Here, the coarser motion and the finer motion intersect at the multipliers of 4 of Δt but the finer motion follows a smoother path since it takes outputs more samples than the coarser one as expected.

3.3.

For MLMC, the general scheme looks like:

$$E[Y_f] = E[Y_c] + E[Y_f - Y_c]$$

And for the sake of this question where there are 4 levels, the MLMC algorithm is created in accordance with the following expression where Y_4 is the high fidelity model with $\Delta t = 4^{-5}$ and Y_1 the lowest fidelity model with $\Delta t = 4^{-1}$. These levels share the underlying realizations of wiener realization at their respective time steps. This is acquired with using the fine to coarse transformation function as discussed.

$$E[Y_4(1)] = E[Y_1(1)] + \sum_{i=2}^{4} E[Y_i(1) - Y_{i-1}(1)]$$

This results in in the bar graphs shown in the figures below for variance and estimates. It is clear and intuitive that both decrease as the level goes from coarser to finer. Level 1 corresponds to the coarsest level and the fidelity increases as the level number increases. This is expected since a coarser estimator will only take a few samples whereas a finer one takes a lot more samples which reduces the variance as shown in Figure 19 and Figure 20.

The variance value decreasing with increased fidelity can also be derived from the formula

$$Var(Y_l - Y_{l-1}) = Var(Y_l) + Var(Y_{l-1}) - 2Cov(Y_l - Y_{l-1})$$

As the levels get finer, the Y values will be more closer to each other which result in a bigger covariance. So the overall variance of a finer level will be reduced.

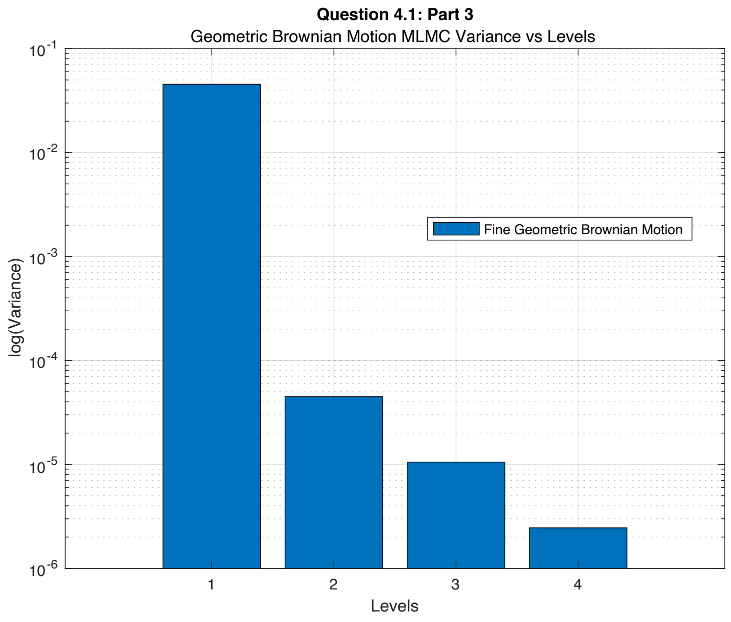

Figure 19

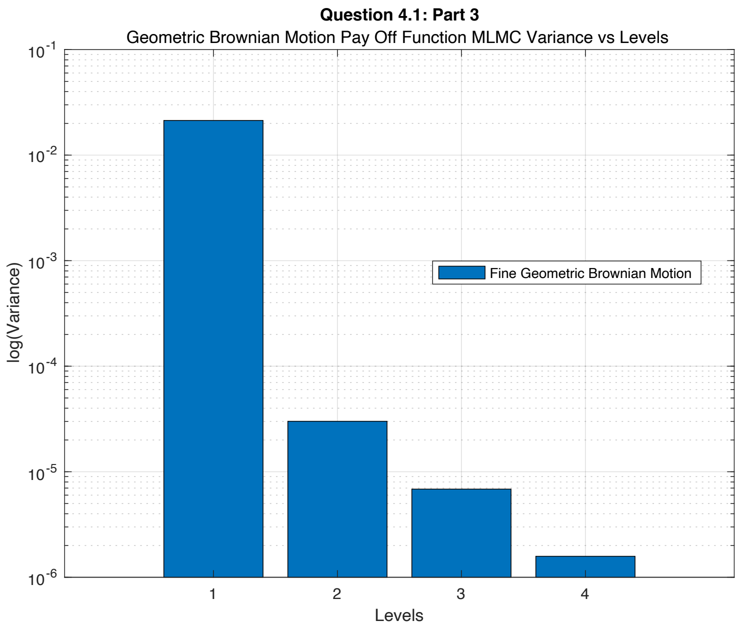

Figure 20

The code ran for MLMC is now modified for the pay-off cost function given. The expectation of this estimator can be seen in the figures below, Figure 21 and Figure 22. The trend from the E[Y(1)] is observed in this case as well, where the expectation increases towards the coarser levels. Intuitively, as the levels become finer, it is expected to converge to the true value better.

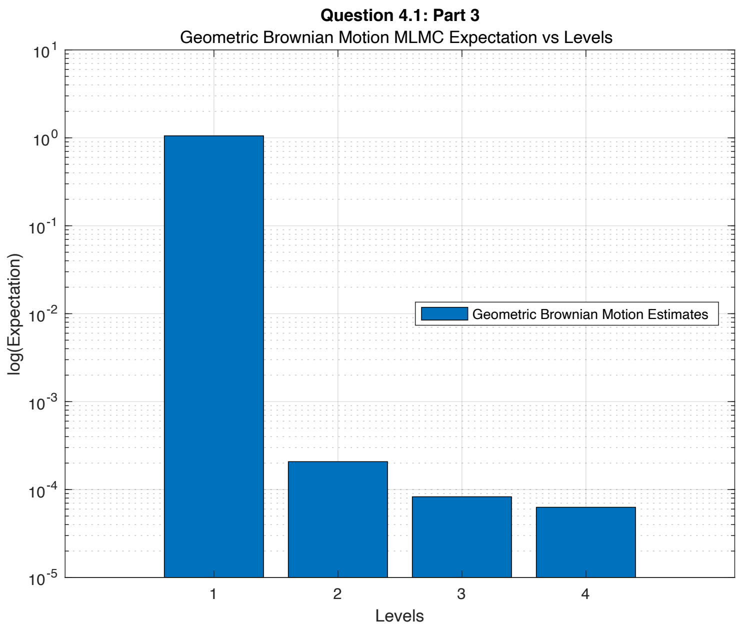

Figure 21

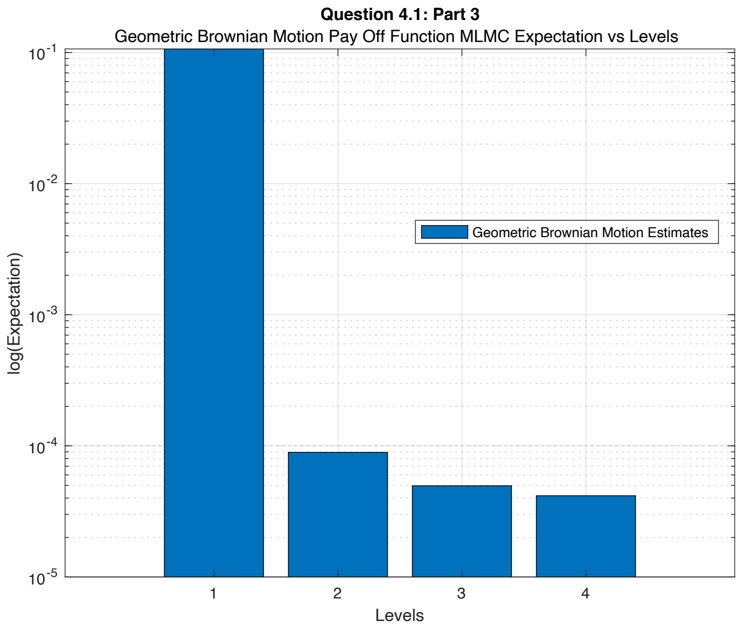

Figure 22

3.4.

In order to compute the theoretical cost of the MLMC estimator, run times obtained from Matlab tic and toc measurements are used. The runtimes for each time is as follows:

$$t_{P_1(Y)} = 0.029989\ seconds$$
$$t_{P_2(Y)-P_1(Y)} = 0.032751\ seconds$$
$$t_{P_3(Y)-P_2(Y)} = 0.041187\ seconds$$
$$t_{P_4(Y)-P_3(Y)} = 0.054871\ seconds$$

$$\lambda = \epsilon^{-2} \sum_{l=0}^{L} \sqrt{V_l \tilde{C}_l}$$

Where variances V_l and costs $\tilde{C}_l$ are calculated above.

$$N_l = \lambda\sqrt{V_l\tilde{C}_l}$$

$$N_{P_1(Y)} = 0.0027e - 12$$
$$N_{P_2(Y)-P_1(Y)} = 0.0793e - 12$$
$$N_{P_3(Y)-P_2(Y)} = 0.1823e - 12$$
$$N_{P_4(Y)-P_3(Y)} = 0.4014e - 12$$

This is also intuitive since the coarsest level would require the least, and the finest level would require the most allocation.

3.5.

In order to find the required sample size, the sample size where the MLMC and MC both have the same statistics can be found and taken as the result.

See Appendix 4 for the code for this problem.

4. APPENDIX

4.1.Appendix 1

```
% Warmup

clear all
close all
clc

seed = 1333; % rng seed for reproducibility


sumType = 0;
alpha_p = 3/2;
sampleNum = 1000;
n = 100;

MCest = zeros(1,n);

sumType = 1;
[estimate, samples, evaluations] = monteCarlow(n,sampleNum,seed,alpha_p,sumType);

plot(estimate')
title('Warmup Question', 'Running Estimate')
xlabel('Sample Number')
ylabel('S_i')
legend('X_n','Location','best')
grid on

j=1;
for i=1:sampleNum

varn(j)=var(estimate(:,i));
 j=j+1;
end

semilogy(varn)

semilogy(varn)
hold on
semilogy(1./(1:sampleNum))
title('Warmup Question', 'Variance of Estimates')
xlabel('Sample Number')
ylabel('Variance')
legend('X_n','1/n','Location','best')
grid on

figure(4)
i=200:400:sampleNum;
h(1)=histogram(estimate(:,i(1)),'FaceColor','b','FaceAlpha',0.1)
hold on
h(2)=histogram(estimate(:,i(2)),'FaceColor','g','FaceAlpha',0.3)
h(3)=histogram(estimate(:,i(3)),'FaceColor','r','FaceAlpha',0.5)
title('Warmup Question', 'Histogram of Estimates')
```

```
xlabel('Sample Number')
ylabel('Frequency')
legend('n=200','n=600','n=1000','Location','best')
grid on

function [estimate, samples, evaluations] =
monteCarlow(runNumber,numSteps,seed,alpha_p,sumType)
s = rng;
s.Seed = seed;
rng(s.Seed);

u = rand(runNumber,numSteps);
u (u>0.9) = 0.1;
x = (1-u).^(-1/alpha_p);
samples = x;
evaluations = x;

if sumType == 0
    estimate = sum(evaluations') / (numSteps); % indicator func sum columnwise
summation
else
    indMatrix = ones(runNumber,1)*(1:numSteps);
    cumsumMatrix = cumsum(evaluations');
    estimate = cumsumMatrix' ./ (1:numSteps); % columnwise summation
end
end
```

4.2. Appendix 2

```
% 567 RW
clc
clear all
close all

seed = 1333; % rng seed for reproducibility

%% PART A

numSteps = 100;
runNumber = 1000;
pThreshold = 0.5;
steps = generateRandomWalkSteps(runNumber,numSteps,pThreshold,seed);
randomWalk = generateRandomWalk(steps);

figure(1)
plot(randomWalk','LineWidth',1.5,'Color',[0, 0.5, 0.5, 0.10])
title('Question 2.1: Part A','1000 Random Walks')
xlabel('N')
ylabel('S')
legend('Random Walk_n','Location','best')
grid on

figure(2)
hist(randomWalk(:,end))
title('Question 2.1: Part A','Random Walk Histogram')
xlabel('S_N')
```

```
ylabel('Frequency')
legend('Random Walk_n','Location','best')

%% PART B

distLimit = 10;
totDist = [];
runNumber = 10^5;


sumType = 0; % 0: sum, 1: cumsum
[estimate, samples,
evaluations]=monteCarlo2(runNumber,numSteps,seed,pThreshold,distLimit,sumType);

PS10 = estimate;
EvalS10 = evaluations;

%% PART C
sumType = 0; % 0: sum, 1: cumsum
pThresholdc = 0.2;
distLimitc = 55;
[estimate, samples, evaluations] =
ISmonteCarlo2(runNumber,numSteps,seed,pThreshold,pThresholdc,distLimit,distLimitc,sumT
ype);

PS55 = estimate;
EvalS55 = evaluations;

%% PART D

minSteps10 = ceil(distLimit + ((100-distLimit)/2))+1;
syms k x
AnalyticalP10 =
double((1/2^numSteps)*symsum(nchoosek(numSteps,k),minSteps10,numSteps));
% y = 1-binocdf(minSteps10,100,0.5)

minSteps55 = ceil(distLimitc + ((100-distLimitc)/2));
AnalyticalP55 =
double((1/2^numSteps)*symsum(nchoosek(numSteps,k),k,minSteps55,numSteps));

%% PART E

% Part i)

stdError10 = std(EvalS10)/sqrt(runNumber);
stdError55 = std(EvalS55)/sqrt(runNumber);

z = 1.96;
% prob 0.95 ==> z score 1.96

meanEvalS10 = mean(EvalS10);
varEvalS10 = var(EvalS10);
intervalS10_ = [meanEvalS10-z*stdError10 meanEvalS10+z*stdError10];


meanEvalS55 = mean(EvalS55);
varEvalS55 = var(EvalS55);
intervalS55_ = [meanEvalS55-z*stdError55 meanEvalS55+z*stdError55];
```

```
% Part ii)

seed = 1333;
duplicateNum = 1000;

PS10_1000 = zeros(1,duplicateNum);
Eval10_1000 = zeros(runNumber,duplicateNum);
PS55_1000 = zeros(1,duplicateNum);
intervalS10 = zeros(duplicateNum,2);
evalEval = zeros(runNumber,1);


for i=1:duplicateNum
seed = seed+i;

[estimate, samples,
evaluations]=monteCarlo2(runNumber,numSteps,seed,pThreshold,distLimit,sumType);
PS10i = estimate;
EvalS10i = evaluations;
stdError10i = std(EvalS10i)/sqrt(runNumber);
Eval10_1000(:,i)  = EvalS10i;
intervalS10i = [PS10i-z*stdError10i PS10i+z*stdError10i];
intervalS10(i,:) = intervalS10i;


[estimate, samples, evaluations] =
ISmonteCarlo2(runNumber,numSteps,seed,pThreshold,pThresholdc,distLimit,distLimitc,sumT
ype);
PS55i = estimate;
EvalS55i = evaluations;
stdError55i = std(EvalS55i)/sqrt(runNumber);
Eval55_1000(:,i)  = EvalS55i;
intervalS55i = [PS55i-z*stdError55i PS55i+z*stdError55i];
intervalS55(i,:) = intervalS55i;

end

confP10 = sum(intervalS10(:,1) <= AnalyticalP10 & AnalyticalP10 <= intervalS10(:,2))/
duplicateNum;
confP55 = sum(intervalS55(:,1) <= AnalyticalP55 & AnalyticalP55 <= intervalS55(:,2))/
duplicateNum;
save('q21e','confP10','confP55','intervalS10','intervalS55','Eval55_1000','Eval10_1000
')

% Part iii)


runningMean10Eval = zeros(duplicateNum,runNumber);
runningMean55Eval = zeros(duplicateNum,runNumber);

cumsumEval = cumsum(Eval10_1000);
indMatrix = ones(duplicateNum,1)*(1:runNumber);
runningMean10Eval=cumsumEval'./indMatrix;

cumsumEval = cumsum(Eval55_1000);
runningMean55Eval=cumsumEval'./indMatrix;
```

```
runningMean10EvalMax = max(runningMean10Eval);
runningMean10EvalMin = min(runningMean10Eval);
runningMean55EvalMax = max(runningMean55Eval);
runningMean55EvalMin = min(runningMean55Eval);

figure(3)
plot(1:runNumber,runningMean10EvalMax
hold on
plot(1:runNumber,runningMean10EvalMin)
title('Question 2.1: Part E) iii)','Envelope for S>10')
xlabel('Number of Trials (M)')
ylabel('Running Mean')
legend('Running Mean_{max}','Running Mean_{min}','Location','best')


figure(4)
plot(1:runNumber,runningMean55EvalMax)
hold on
plot(1:runNumber,runningMean55EvalMin)
title('Question 2.1: Part E) iii)','Envelope for S>55')
xlabel('Number of Trials (M)')
ylabel('Running Mean')
legend('Running Mean_{max}','Running Mean_{min}','Location','best')
%
save(‘q21e','confP10','confP55','intervalS10','intervalS55','Eval55_1000','Eval10_1000
','runningMean10Eval','runningMean55Eval')

% generate a set of steps for a random walk

function X = generateRandomWalkSteps(runNumber,numSteps,threshold,seed)
% rng seed fix
s = rng;
s.Seed = seed;
rng(s.Seed)
X = rand(runNumber,numSteps); % sample from a unifrom
X (X < threshold) = -1;
X (X > threshold) =  1;
end

% generate a rv that is shrunk in space and sped up in time from a sequence of steps

function X = generateRandomWalk(steps)
%     steps=steps';
    stepSize = size(steps);
    X = zeros(1,stepSize(1));
    X = [X; cumsum(steps')];
    X = X';
end
function [estimate, samples, evaluations] =
monteCarlo2(runNumber,numSteps,seed,pThreshold,distLimit,sumType)

steps = generateRandomWalkSteps(runNumber,numSteps,pThreshold,seed);
randomWalk=generateRandomWalk(steps);
samples = randomWalk;
totDist = randomWalk(:,end);
evaluations =evaluatorFunc(totDist,distLimit);
if sumType == 0
    estimate = sum(evaluations) / (runNumber); % indicator func sum columnwise
summation
else
```

```
    estimate = cumsum(evaluations) / (1:runNumber); % columnwise summation
end
end
function [estimate, samples, evaluations] =
ISmonteCarlo2(runNumber,numSteps,seed,pThresholdf,pThresholdq,distLimitf,distLimitq,su
mType)


fsteps = generateRandomWalkSteps(runNumber,numSteps,pThresholdf,seed);
frandomWalk=generateRandomWalk(fsteps);
qsteps = generateRandomWalkSteps(runNumber,numSteps,pThresholdq,seed);
qrandomWalk=generateRandomWalk(qsteps);
q_x = prob_x(qsteps,pThresholdq);
f_x = prob_x(qsteps,pThresholdf);

g_x =evaluatorFunc(qrandomWalk(:,end),distLimitq);

likelihoodfq   = f_x./q_x;
plikelihoodfq  = prod(likelihoodfq');
evaluations    = g_x.*plikelihoodfq';

samples = qrandomWalk;
if sumType == 0
    estimate = sum(evaluations) / (runNumber); % indicator func sum columnwise
summation
else
    estimate = cumsum(evaluations) / (1:runNumber+1); % columnwise summation
end
end

function evaluatorFuncOut = evaluatorFunc(totDist, threshold)
g_xTotDist = totDist;
g_xTotDist (totDist <= threshold) = 0;
g_xTotDist (totDist > threshold) = 1;
probDistLimit = sum(g_xTotDist)/length(g_xTotDist);
evaluatorFuncOut = g_xTotDist;
end

function probWalk = prob_x(inputArray,pValue)
inputArray (inputArray ==  -1) = pValue ;
inputArray (inputArray ==  1) = 1-pValue ;
probWalk = (inputArray);
end
```

4.3. Appendix 3

```
% 567 3D RW
clc
clear all
close all

seedX = 1333; % rng seed for reproducibility
seedY = 4806; % rng seed for reproducibility
seedZ = 1881; % rng seed for reproducibility
seed  = [seedX seedY seedZ];

%% PART A

numSteps  = 100;
```

```
runNumber = 1000;
stepsX = generateRandomWalkStepsNormal(runNumber,numSteps,seed(1));
stepsY = generateRandomWalkStepsNormal(runNumber,numSteps,seed(2));
stepsZ = generateRandomWalkStepsNormal(runNumber,numSteps,seed(3));
randomWalk = cat(3,stepsX,stepsY,stepsZ);

randomWalkT = cat(3,(randomWalk(:,:,1))',(randomWalk(:,:,2))',(randomWalk(:,:,3))');
posEnd = sum(randomWalkT);

posEnd = squeeze(posEnd);
distEnd = vecnorm(posEnd'); % If A is a matrix, then vecnorm returns the norm of each column.

figure(2)
hist(distEnd)
title('Question 2.2: Part A','Random Walk Distance Histogram')
xlabel('S_N')
ylabel('Frequency')
legend('Random Walk_n','Location','best')

%% PART B
distLimit = 10;
runNumber = 100000;
sumType = 0;

[estimate, samples, evaluations] = monteCarloNormal(runNumber,numSteps,seed,distLimit,sumType);
EvalS10=evaluations;
estimate

%% PART C
% IS Sampling Rules
distLimitc = 55;
runNumber = 100000;
sumType = 0;

mu1 = 0;
sigma1 = 1;

mu2 = 0.25;
sigma2 = 1;

[estimate, samples, evaluations] =
ISmonteCarloNormal(runNumber,numSteps,seed,mu1,sigma1,mu2,sigma2,distLimitc,sumType);
EvalS55 = evaluations;
estimate

%% PART D

stdError10 = std(EvalS10)/sqrt(runNumber);

stdError55 = std(EvalS55)/sqrt(runNumber);

z = 1.96;
% prob 0.95 ==> z score 1.96

meanEvalS10 = mean(EvalS10);
varEvalS10 = var(EvalS10);
intervalS10 = [meanEvalS10-z*varEvalS10/(sqrt(runNumber)) meanEvalS10+z*varEvalS10/(sqrt(runNumber))];


meanEvalS55 = mean(EvalS55);
varEvalS55 = var(EvalS55);
intervalS55 = [meanEvalS55-z*varEvalS55/(sqrt(runNumber)) meanEvalS55+z*varEvalS55/(sqrt(runNumber))];
```

```
%% extra
AnalyticalP10 = 1 - gamcdf(distLimit,3/2,200)
AnalyticalP55 = 1 - gamcdf(distLimitc,3/2,200)

% generate a set of steps for a random walk

function X = generateRandomWalkStepsNormal(runNumber,numSteps,seed)
% rng seed fix
s = rng;
s.Seed = seed;
rng(s.Seed)

X = randn(runNumber,numSteps); % sample from a unifrom

end

function [estimate, samples, evaluations] = monteCarloNormal(runNumber,numSteps,seed,distLimit,sumType)

stepsX = generateRandomWalkStepsNormal(runNumber,numSteps,seed(1));
stepsY = generateRandomWalkStepsNormal(runNumber,numSteps,seed(2));
stepsZ = generateRandomWalkStepsNormal(runNumber,numSteps,seed(3));
randomWalk = cat(3,stepsX,stepsY,stepsZ);
samples = randomWalk;

randomWalkT = cat(3,(randomWalk(:,:,1))',(randomWalk(:,:,2))',(randomWalk(:,:,3))');
posEnd = sum(randomWalkT);
posEnd = squeeze(posEnd);
distEnd = vecnorm(posEnd'); % If A is a matrix, then vecnorm returns the norm of each column.
evaluations=evaluatorFunc(distEnd,distLimit);


if sumType == 0
    estimate = sum(evaluations) / (runNumber); % indicator func sum columnwise summation
else
    estimate = cumsum(evaluations) / (1:runNumber+1); % columnwise summation
end
end

% generate a set of steps for a random walk

function X = generateRandomWalkStepsNormalIS(runNumber,numSteps,mu,sigma,seed)
% rng seed fix
s = rng;
s.Seed = seed;
rng(s.Seed)

X = sigma*randn(runNumber,numSteps) + mu; % sample from a unifrom

end

function [estimate, samples, evaluations] =
ISmonteCarloNormal(runNumber,numSteps,seed,mu1,sigma1,mu2,sigma2,distLimit,sumType)

fstepsX = generateRandomWalkStepsNormal(runNumber,numSteps,seed(1));
fstepsY = generateRandomWalkStepsNormal(runNumber,numSteps,seed(2));
fstepsZ = generateRandomWalkStepsNormal(runNumber,numSteps,seed(3));
frandomWalk = cat(3,fstepsX,fstepsY,fstepsZ);
fsamples = frandomWalk;

frandomWalkT = cat(3,(frandomWalk(:,:,1))',(frandomWalk(:,:,2))',(frandomWalk(:,:,3))');
fposEnd = sum(frandomWalkT);
fposEnd = squeeze(fposEnd);
fdistEnd = vecnorm(fposEnd'); % If A is a matrix, then vecnorm returns the norm of each column.
```

```
qstepsX = generateRandomWalkStepsNormalIS(runNumber,numSteps,mu2,sigma2,seed(1));
qstepsY = generateRandomWalkStepsNormalIS(runNumber,numSteps,mu2,sigma2,seed(2));
qstepsZ = generateRandomWalkStepsNormalIS(runNumber,numSteps,mu2,sigma2,seed(3));
qrandomWalk = cat(3,qstepsX,qstepsY,qstepsZ);
qsamples = qrandomWalk;

qrandomWalkT = cat(3,(qrandomWalk(:,:,1))',(qrandomWalk(:,:,2))',(qrandomWalk(:,:,3))');
qposEnd = sum(qrandomWalkT);
qposEnd = squeeze(qposEnd);
qdistEnd = vecnorm(qposEnd'); % If A is a matrix, then vecnorm returns the norm of each column.

q_x=normalDistPDF(mu2,sigma2,qsamples);
f_x=normalDistPDF(mu1,sigma1,qsamples);
g_x  = evaluatorFunc(qdistEnd,distLimit);
plikelihoodfq = zeros(runNumber,1);
likelihoodfq   = f_x./q_x;

for i=1:runNumber
likelihoodStep = squeeze(likelihoodfq(i,:,:));
pEachStep = prod(likelihoodStep');
pEachRun = prod(pEachStep);
plikelihoodfq(i,1) = pEachRun;
end

evaluations    = g_x.*plikelihoodfq';
samples = qrandomWalk;

if sumType == 0
    estimate = sum(evaluations) / (runNumber); % indicator func sum columnwise summation
else
    estimate = cumsum(evaluations) / (1:runNumber+1); % columnwise summation
end
end

function evaluatorFuncOut = evaluatorFunc(totDist, threshold)


g_xTotDist = totDist;
g_xTotDist (totDist <= threshold) = 0;
g_xTotDist (totDist > threshold) = 1;
probDistLimit = sum(g_xTotDist)/length(g_xTotDist);
evaluatorFuncOut = g_xTotDist;
end

function p = normalDistPDF(mu,sigma,x)
p = (1/(sigma*sqrt(2*pi)))*exp(-0.5*((x-mu)/sigma).^2);
end
```

4.4. Appendix 3

```
% 567
% 4.1

clear all
close all
clc

seed = 1333; % rng seed for reproducibility
rng; %seed fix
s = rng;
s.Seed = seed;
rng(s.Seed);
```

```
mu = 0.05;
sigma = 0.2;
Y_0 = 1;
% 0<t<1
tFinal = 1;
% dt =1/N;
dt = 0.05;
% N = 1/dt;
runNumber = 100;


%% PART 1

W = bms(tFinal, dt, runNumber);
Y = gbms(W,mu,sigma,Y_0,dt);

% plots

timeVec = 0:dt:tFinal;

figure(1)
plot(timeVec,W')
title('Question 4.1: Part 1','Wiener Process ')
xlabel('Time')
ylabel('Walked Distance')
legend('Wiener Process_n','Location','best')
grid on

figure(2)
plot(timeVec,Y')
title('Question 4.1: Part 1','Geometric Brownian Motion')
xlabel('Time')
ylabel('Walked Distance')
legend('Geometric Brownian Motion_n','Location','best')
grid on


var_Y=var(Y(:,end));
exp_Y=mean(Y(:,end));


AnalyticExp_Y = exp(mu - (sigma^2)/2 + (sigma^2)/2 );
AnalyticVar_Y = exp(2*mu - sigma^2 + 2*sigma^2)- exp(2*mu - sigma^2 + sigma^2);

%% PART 2

runNumber = 1;
m=4;
dtm = dt*m;     % coarse time step


Wf = bms(tFinal, dt, runNumber,seed);
Wc = bmf2c(tFinal, dtm, runNumber, Wf, m);

Yf = gbms(Wf,mu,sigma,Y_0,dt);
Yc = gbms(Wc,mu,sigma,Y_0,dtm);

% plots

timeVecC = 0:dtm:tFinal;
timeVecF = 0:dt:tFinal;

figure(3)
plot(timeVecF,Wf,'m-x','LineWidth',2,'MarkerSize',8)
```

```
hold on
plot(timeVecC,Wc,'b-o','LineWidth',2,'MarkerSize',8)
title('Question 4.1: Part 2','Wiener Process vs dt')
xlabel('Time')
ylabel('Walked Distance')
legend('Fine Wiener Process','Coarse Wiener Process','Location','best')
grid on

figure(4)
plot(timeVecF,Yf,'m-x','LineWidth',2,'MarkerSize',8)
hold on
plot(timeVecC,Yc,'b-o','LineWidth',2,'MarkerSize',8)
title('Question 4.1: Part 2','Geometric Brownian Motion vs dt')
xlabel('Time')
ylabel('Walked Distance')
legend('Fine Geometric Brownian Motion','Coarse Geometric Brownian Motion','Location','best')
grid on

%% PART 3

runNumber = 1000;
i = [2 3 4 5];

dt = 4^-i(4);
Wf = bms(tFinal, dt, runNumber);
Yf = gbms(Wf,mu,sigma,Y_0,dt);

m1=4;
dtc1 = dt*m1;    % coarse time step 1
Wc1 = bmf2c(tFinal, dtc1, runNumber, Wf, m1);
Yc1 = gbms(Wc1,mu,sigma,Y_0,dtc1);

m2=4^2;
dtc2 = dt*m2;
Wc2 = bmf2c(tFinal, dtc2, runNumber, Wf, m2);
Yc2 = gbms(Wc2,mu,sigma,Y_0,dtc2);

m3=4^3;
dtc3 = dt*m3;
Wc3 = bmf2c(tFinal, dtc3, runNumber, Wf, m3);
Yc3 = gbms(Wc3,mu,sigma,Y_0,dtc3);

% plots

timeVecF = 0:dt:tFinal;
timeVecC1 = 0:dtc1:tFinal;
timeVecC2 = 0:dtc2:tFinal;
timeVecC3 = 0:dtc3:tFinal;

Yf_end = Yf(:,end);
Yc1_end = Yc1(:,end);
Yc2_end = Yc2(:,end);
Yc3_end = Yc3(:,end);

V_Yc3 = var(Yc3_end);
V_Yc2Yc3 = var(Yc2_end-Yc3_end);
V_Yc1Yc2 = var(Yc1_end-Yc2_end);
V_YfYc1 = var(Yf_end-Yc1_end);

VarY = [V_Yc3 V_Yc2Yc3 V_Yc1Yc2 V_YfYc1];

V_Yf = V_Yc3 + V_Yc2Yc3 + V_Yc1Yc2 + V_YfYc1;

E_Yc3    = sum(Yc3_end)/length(Yc3_end);
```

```
E_Yc2Yc3 = sum(Yc2_end-Yc3_end)/length(Yc2_end-Yc3_end);
E_Yc1Yc2 = sum(Yc1_end-Yc2_end)/length(Yc1_end-Yc2_end);
E_YfYc1  = sum(Yf_end-Yc1_end)/length(Yf_end-Yc1_end);

E_Yf = E_Yc3 + E_Yc2Yc3 + E_Yc1Yc2 + E_YfYc1;
EY   = [E_Yc3  E_Yc2Yc3  E_Yc1Yc2  E_YfYc1];

% P

P_Yc3 = exp(-0.05)*max(0,(Yc3_end)-1);
P_Yc2 = exp(-0.05)*max(0,(Yc2_end)-1);
P_Yc1 = exp(-0.05)*max(0,(Yc1_end)-1);
P_Yf = exp(-0.05)*max(0,(Yf_end)-1);

P_Yc2Yc3 = P_Yc2-P_Yc3;
P_Yc1Yc2 = P_Yc1-P_Yc2;
P_YfYc1  = P_Yf-P_Yc1;

% P Expectation

EP_Yc3    = sum(P_Yc3)/length(P_Yc3);
EP_Yc2Yc3 = sum(P_Yc2Yc3)/length(P_Yc2Yc3);
EP_Yc1Yc2 = sum(P_Yc1Yc2)/length(P_Yc1Yc2);
EP_YfYc1  = sum(P_YfYc1)/length(P_YfYc1);

EP_Yf = EP_Yc3 + EP_Yc2Yc3 + EP_Yc1Yc2 + EP_YfYc1;
EYP   = [EP_Yc3 EP_Yc2Yc3 EP_Yc1Yc2 EP_YfYc1];

% P Variance

VP_Yc3 = var(P_Yc3);
VP_Yc2Yc3 = var(P_Yc2Yc3);
VP_Yc1Yc2 = var(P_Yc1Yc2);
VP_YfYc1  = var(P_YfYc1);

VP_Yf = VP_Yc3 + VP_Yc2Yc3 + VP_Yc1Yc2 + VP_YfYc1;
VarYP = [VP_Yc3 VP_Yc2Yc3 VP_Yc1Yc2 VP_YfYc1];

figure(7)
bar((VarY))
set(gca,'YScale','log')
title('Question 4.1: Part 3','Geometric Brownian Motion MLMC Variance vs Levels')
xlabel('Levels')
ylabel('log(Variance)')
legend('Geometric Brownian Motion Variances','Coarse Geometric Brownian Motion #1','Coarse Geometric
Brownian Motion #2','Coarse Geometric Brownian Motion #3','Location','best')
grid on


figure(8)
bar((VarYP))
set(gca,'YScale','log')
title('Question 4.1: Part 3','Geometric Brownian Motion Pay Off Function MLMC Variance vs Levels')
xlabel('Levels')
ylabel('log(Variance)')
legend('Geometric Brownian Motion Variances','Coarse Geometric Brownian Motion #1','Coarse Geometric
Brownian Motion #2','Coarse Geometric Brownian Motion #3','Location','best')
grid on



figure(9)
bar((EY))
set(gca,'YScale','log')
```

```
title('Question 4.1: Part 3','Geometric Brownian Motion MLMC Expectation vs Levels')
xlabel('Levels')
ylabel('log(Expectation)')
legend('Geometric Brownian Motion Estimates','Coarse Geometric Brownian Motion #1','Coarse Geometric
Brownian Motion #2','Coarse Geometric Brownian Motion #3','Location','best')
grid on


figure(10)
bar(((EYP)))
set(gca,'YScale','log')
title('Question 4.1: Part 3','Geometric Brownian Motion Pay Off Function MLMC Expectation vs Levels')
xlabel('Levels')
ylabel('log(Expectation)')
% ylim([10^-6 10^-1])
legend('Geometric Brownian Motion Estimates','Coarse Geometric Brownian Motion #1','Coarse Geometric
Brownian Motion #2','Coarse Geometric Brownian Motion #3','Location','best')
grid on

%% PART 4

t4=0.029989;
t3= 0.032751;
t2=0.041187;
t1=0.054871;

t= t1+t2+t3+t4;

normalized_t4 = t4/t1;
normalized_t3 = t3/t1;
normalized_t2 = t2/t1;
normalized_t1 = t1/t1;
tarVar=0.005;
lambda =
tarVar^(-2)*(sqrt(normalized_t4*VP_Yc3)*sqrt(normalized_t3*VP_Yc2Yc3)*sqrt(normalized_t2*VP_Yc1Yc2)*sqr
t(normalized_t1*VP_YfYc1))^2;
N_L = lambda.*sqrt([VP_Yc3 VP_Yc2Yc3 VP_Yc1Yc2 VP_YfYc1].\[normalized_t4 normalized_t3 normalized_t2
normalized_t1]);

function bmsOut = bms(tFinal, dt, runNumber)
sampleNum = ceil(tFinal/dt);
samples = [zeros(runNumber,1) randn(runNumber,sampleNum)*sqrt(dt)];
bmsOut=cumsum(samples'); %If A is a matrix, then cumsum(A) returns a matrix containing the cumulative
sums for each column of A.
end

function gbmsOut = gbms(W,mu,sigma,Y_0,dt)
W=W';
sizeW = size(W);
Y(1:sizeW(1),1) = Y_0;
for i=1:sizeW(2)-1
    b = mu * Y(:,i);
    h = sigma * Y(:,i);
    dW = W(:,i+1)-W(:,i);
    Y(:,i+1) = Y(:,i) + b.*dt + h.*dW;
end
gbmsOut=Y;
end

function bmf2cOut = bmf2c(tFinal, dt, runNumber, bf, m)
sampleNum = ceil(tFinal/dt)+1;
bc = zeros(sampleNum,runNumber);
bc(1,:) = 0;
for i = 2:sampleNum
```

```
    delta = bf((i-1)*m+1,:) - bf((i-2)*m+1,:);
    bc(i,:) = bc(i-1,:) + delta;
end
bmf2cOut = bc;
end
```