Devin Trejo

ECE 3522: Stochastic Processes in Signals and Systems

Department of Electrical and Computer Engineering, Temple University, Philadelphia, PA 19122

I. PROBLEM STATEMENT

The central limit theorem states that no matter the underlying distribution as long as you have a relatively uniform mean and variance the sum of the distributions will become normal. In our experiment we will develop our own function that creates a uniform distribution with a number of samples N composed of a number of random variables n. By observing the actual histogram of our function output we explore how as we increase the number of random variables our PDF becomes more normal. We plot the actual mean, variance, and MSE between the actual pdf and a normal fitted distribution to observe the underlying characteristics of our output.

Activity four of the assignment tasks us with creating a new method for creating a normal distribution of random variables using the Box-Muller technique. To compare the two functions we develop we time and compare the speed and accuracy of our two approaches for creating normal distributions.

II. APPROACH AND RESULTS

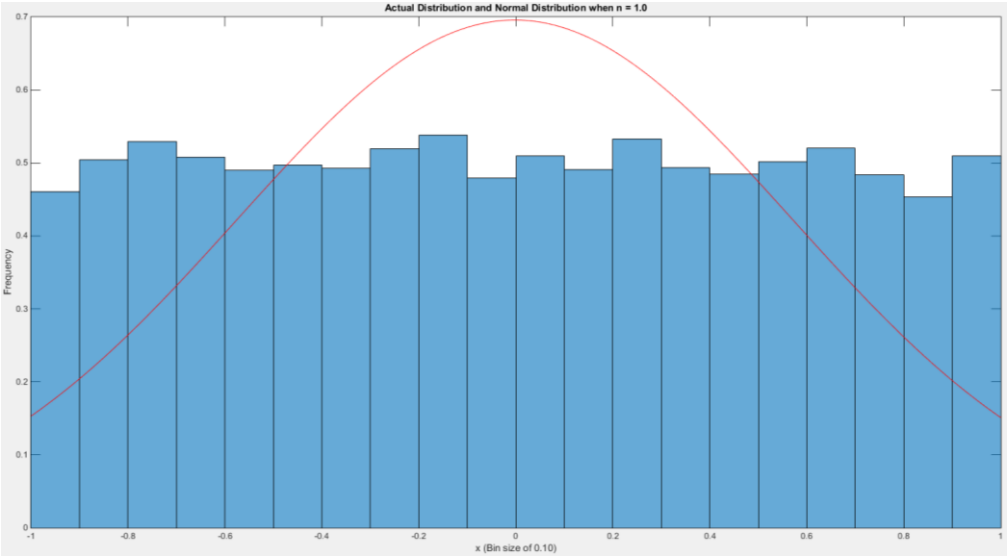

Figure 1: Actual PDF and a Normal Distribution fit when n=1

To begin we use our uniform generating function to create a single uniform distribution. The resulting plot is obviously uniform and does not fit a normal distribution too well. A single uniform distribution had the highest mean squared error out of all the plots (we will look at the MSE later in the assignment). Next we increase the number of random variables (n) to ten and observe whether we see whether we can see application of the central limit theorem.

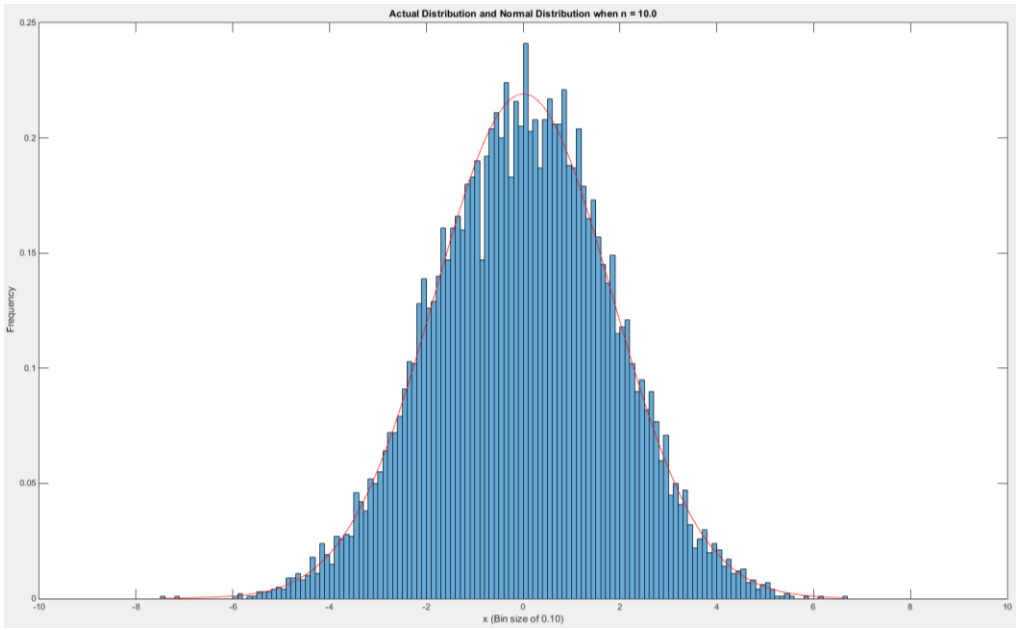

Figure 2: Actual PDF and a Normal Distribution fit when n=10

At ten random variables we see a pretty normal distribution from the sum of the samples returned using our uniform generating function. We extrapolated from our MSE plot (Figure 1) a MSE of 0.01. Also note how the possible range of data has increased from [-1,1] to [-10, 10]. Next we will increase the number of random variables to one hundred and observe the actual vs normal fitted PDF.

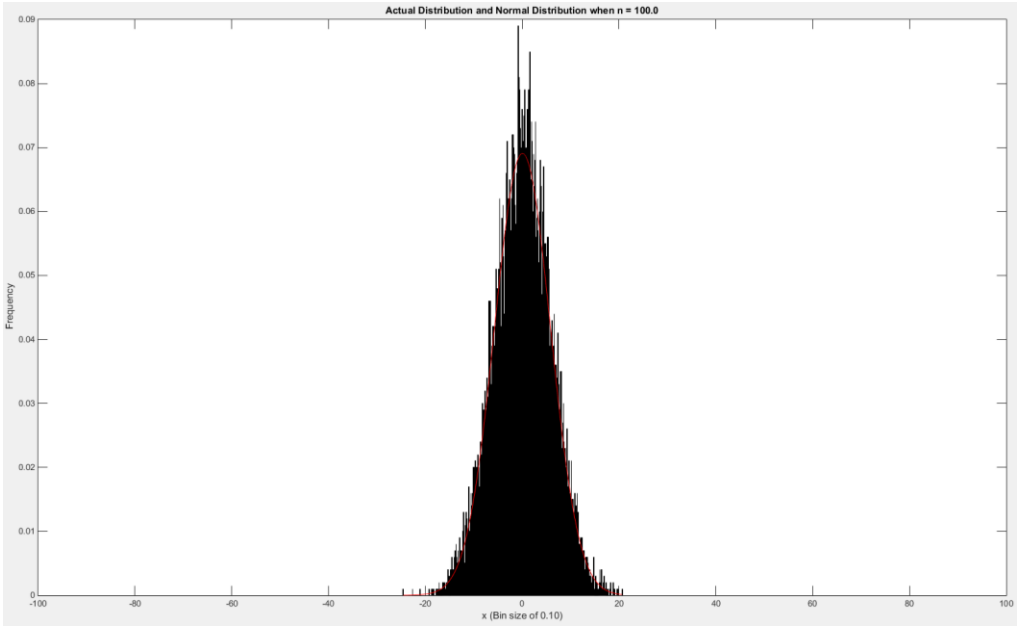

Figure 3: Actual PDF and a Normal Distribution fit when n=100

At one hundred samples our sum of uniform distributions has become Gaussian. In our experiment we found that even after 10 random variables our distribution was already Gaussian. As we keep increasing n our distribution will become more Gaussian.

We will now find the error of the mean, variance, and difference between the actual and normal pdf and plot the differences. Using the central limit theorem we can also find the expected mean and variance as we increase our number of random variables (n). The overall mean and variance will be as follows:

$$\overline{S_n} = E[S_n] = n\mu_x$$

$$\sigma_{S_n}^2 = n\sigma_x^2$$

First we find the expected values for mean and variance given a uniform distribution between -1 and 1.

$$\mu_x = \frac{b+a}{2} = \frac{1+(-1)}{2} = 0$$

$$\sigma_x^2 = \frac{(b-a)^2}{12} = \frac{\big(1-(-1)\big)^2}{12} = \frac{1}{3}$$

Thus for example if we increase the number of random variables (n) to 100 our new mean and variance become:

$$\overline{S_n} = E[S_n] = (100)(0) = 0$$

$$\sigma_{S_n}^2 = (100)\frac{1}{3} = \frac{100}{3}$$

Using these concepts we can write code to find the error of our mean and variance as we increase our number of random variables (n).

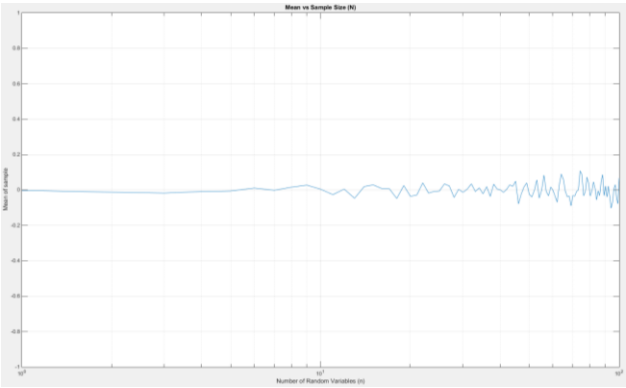

Figure 4: The actual mean as we increase the Number of Random Variables (n)

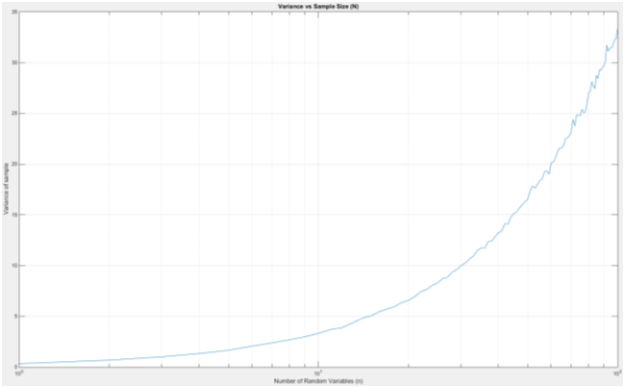

Figure 5: The actual variance as we increase the Number of Random Variables (n)

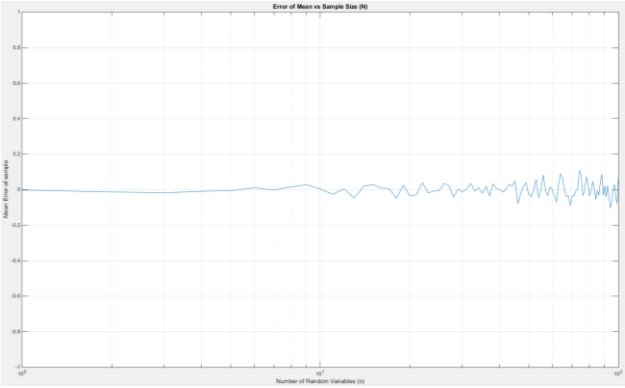

Figure 6: The mean error as we increase the Number of Random Variables (n)

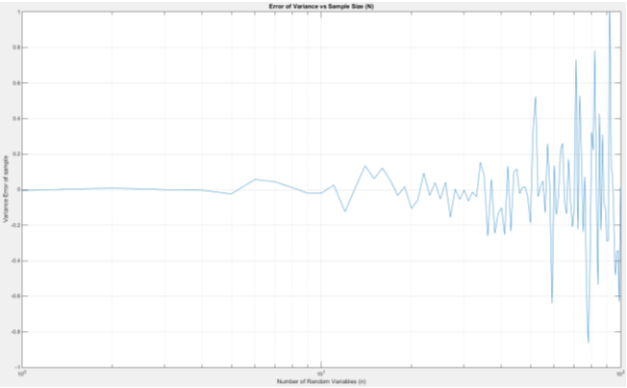

Figure 7: The variance error as we increase the Number of Random Variables (n)

Also, we are interested in finding the difference between our actual PDF and a normal distribution estimated fit.

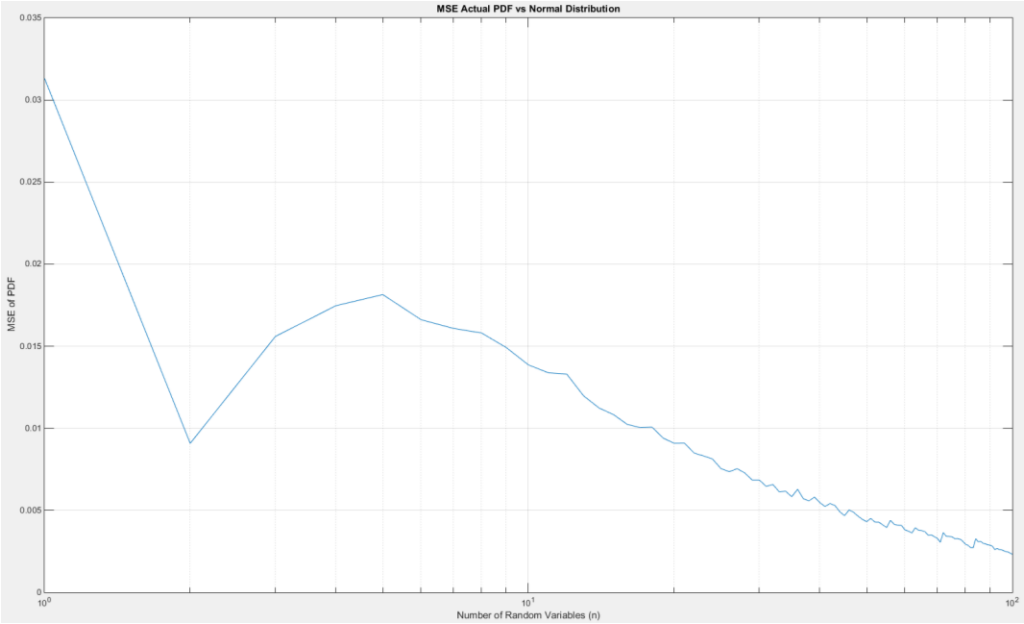

Figure 8: MSE between the actual PDF and a Normal fitted distribution.

We observe how the difference between our means and variance never goes above 1, so our percent difference will always be very small. The MSE also decreases as we increase the number of random variables as the more distributions we combine the more normal our PDF becomes. It is also interesting to note how at a random variable size of two reach a relative minimum and does become lower until our random variable size was twenty. I concluded this characteristics occurs due to my histogram binning process. In the code I increase my bin edges by n. Thus when n is equal to two the range for my histogram is [-2,2] and when n is 10 my histogram ranges from [-10,10]. At a random variable size of 2 we have a triangle histogram (convolve to uniform distributions) where the edges are relatively flat. The probability of having a value of -1 in our first uniform distribution and -1 again in our second uniform distribution is very low. Fitting a Gaussian will fit the edges of the actual histogram very well producing a smaller histogram. Also, with a smaller random variable size our amplitudes are concentrated around the expected value zero. As we increase our variable size our spread of values increases causing the normal distribution to not fit as well. The characteristic does not change until our variable size is greater than eight where our MSE begins to decrease again, thus show casing the properties of the central limit theorem.

Next we look at another way to generate normally distribution random variables using the Box-Muller transform. From the Wikipedia page of Box-Muller Transforms we learn how we can generate two uniformly distributed variables and perform the following operation to create a normally distributed random variable.

$$Z_0 = \sqrt{-2\ln(RV_1)}\cos\left(2\pi(RV_2)\right); where\ RV_{1/2} = Randomly\ Generatrd\ Unifromly\ Distributed\ Variables$$

The Box-Muller technique was developed as a more efficient method to creating normal distributed random variables. In our experiment we will compare the computation time and fit our sum of uniform distribution to the Box-Muller method to see if the theory stands true. In our test scenario we focus on a random variable size (n) of ten with ten thousand samples (N) in each random variable.

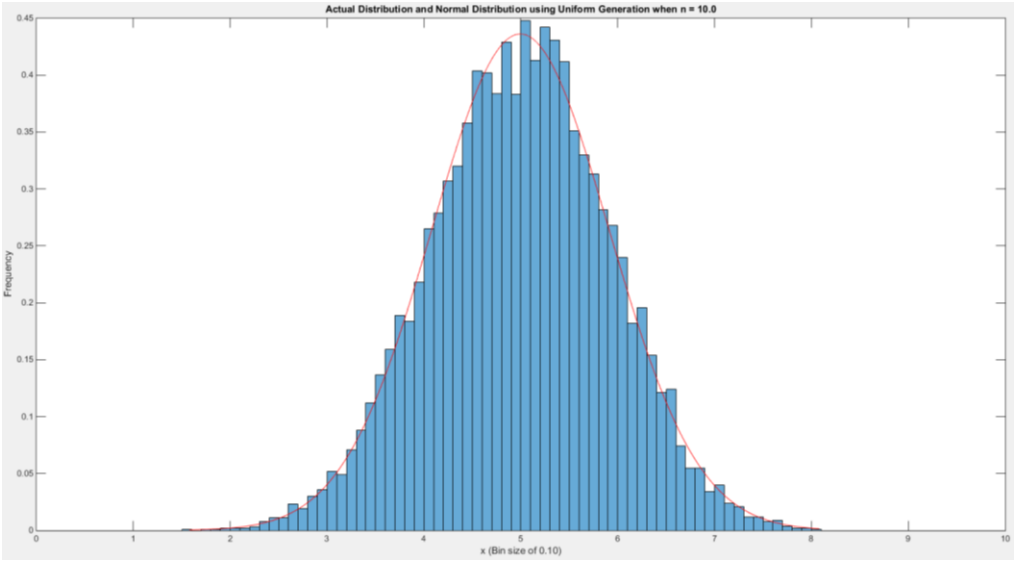

Figure 9: Normal Distributed Array using Sum of Uniform Method

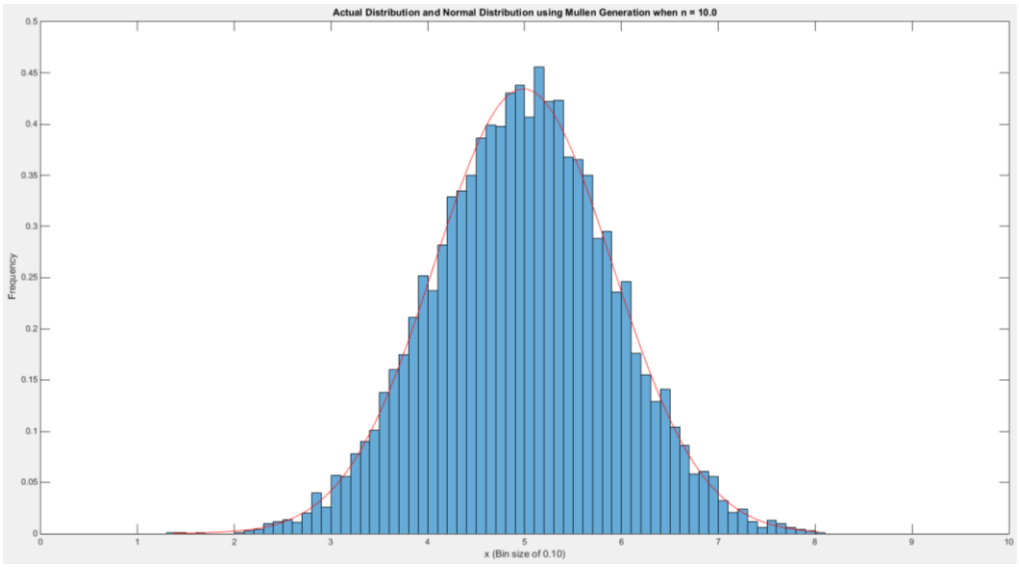

Figure 10: Normal Distributed Array using Sum of Box-Muller Method

Above we plot the actual vs a normal distribution fit for both our sum of uniform and box Muller techniques. The general characteristics of both PDFs showcase pretty normal distributed random variables. The real challenge is comparing the computation time it took to generate those random variable arrays to the accuracy to an actual normal distribution.

Table 1: Sum of Uniform vs Box-Muller Comparison

Method	***Elapsed Time (Milliseconds)***	***MSE (Actual vs Estimated)***
Sum of Uniform	3.823	0.0551
Box-Muller	0.648	0.0548

From Table 1 we observe the better performance and accuracy of the Box-Muller method. The sum of uniform distribution does work however it is slow due to the multiple iterations of random numbers that need to be generated and summed in order to achieve a final result. The Box-Muller technique requires no looping across multiple values and thus produces a results 3.2ms faster.

III. MATLAB CODE

In this assignment we separated our coding process into functions. Each function is outlined below and accompanied by a script that utilizes each function.

```
function sumx = myunifDist(n, N, min, max)

numSam = n;
vecL = N;

% Lower and Upper bound of random Number Generation
lRange = min;
uRange = max;

for i=1:numSam
    x(i,:) = lRange +(uRange-lRange)*rand(vecL,1);
end

% Output the Sum of all randomly generated variables.
sumx = sum(x,1);

end
```

1 The 'myunifDist' function generates a uniformly distributed set of random variables that contains N samples with a range of values from [min, max]. The number of uniformly distributed random variables that is summed up is defined by the input parameter n. The code is simple utilizing a single for loop that generates a uniform random sample array n times. The output of the function is the sum of all n random variables generated.

```
function MSE = myMSE(x, y)

    % Find lengths of the two arrays
    xL = length(x);
    yL = length(y);

    % Find the longer array and down sample according to the smaller array
    if xL > yL
        lMin = yL;
        numSam = xL/yL;
        if mod(numSam, 1) ~= 0
            numSam = floor(numSam);
        end
        Array1 = downsample(x, numSam);
        Array2 = y;
    else
        lMin = xL;
        numSam = yL/xL;
        if mod(numSam, 0) ~= 0
            numSam = floor(numSam);
        end
        Array1 = downsample(y, numSam);
        Array2 = x;
    end

    % Find MSE
    for k = 1:lMin
        g(k) = (Array1(k)-Array2(k))^2;
    end

    MSE = sum(g)/lMin;

end
```

2 'myMSE' is a function the find the mean squared error between two arrays x and y. To begin we need to make sure both sample arrays are the same length. If they are not it is necessary to down sample the longer array to the size of the smaller array. Using a combination of if statements we determine the longer array and down sample accordingly. Then we loop through each element in the input array x and y and find the MSE between the two. The output of the function is the normalized MSE.

```
function out = myNormDist(mu, sigma, N)

    % Find two uniform Variables
    x1 = rand(1,N);
    x2 = rand(1,N);

    y1 = sqrt(-2.*log(x1)).*cos(2.*pi.*x2);
    %y2 = sqrt(-2.*log(x1)).*sin(2.*pi.*x2);

    out = (y1*sigma + mu);
end
```

3 The last function created for the computer assignment is the Box-Muller distribution generator. Using the function outline on the Wikipedia page, we first need three input parameters mean (mu), standard deviation (sigma), and number of samples (N). We generator two random numbers using MatLab's built in random number generator. Using the two random numbers we generated we can create a set of normal distribution using the equation specified. The output is then configured to have properties of the mean and standard deviation specified.

```
%% Part 1, 2, 3
clear; clc; close all;

% Number random Variables
nRandomVar = 100;
NSamples = 10000;
binSize = 0.1;

% Lower and Upper bound of random Number Generation
lRange = -1;
uRange = 1;
```

4 We start our script by defining our constants. The assignment asks for n = 100 and N=10,000 with a range between -1 and 1. I specified a bin size of 0.1 given our minimum value range is from -1 to 1.

```
% For loop
for n=1:nRandomVar
    % Progress Bar
    n

    % Excepted Mean/Variance
    eMean = (uRange+lRange)/2*n;
    eVar = (uRange-lRange)^2/12*n;

    % Histogram Bounds
    bounds = [n*lRange:binSize:n*uRange];
    numBins = length(bounds);

    % Generate Random Array
    samArray = myunifDist(n, NSamples, lRange, uRange);
```

5 The main for loop of the script starts by counting from 1 to the maximum n asked for (n=100). It is important to adjust our expected mean and variance to the number of random variables we have. Also, we redefine our histogram bounds depending on size n to keep in accordance to the properties set forth by the central limit theorem. Lastly, we call our function to provide us with a sum of randomly generator uniformly distributed variables according to our loop counter.

```
% Find mean and Variance
    samMean(n) = mean(samArray);
    samVar(n) = var(samArray);
    meanErr(n) = (samMean(n) - eMean);
    varErr(n) = (samVar(n) - eVar);
```

6 We find the actual mean and variance and compare the values to our expected mean and variance.

```
% Estimate the pdf
    h_samArray = histcounts(samArray, bounds, 'Normalization', 'pdf');

    % Use a function to fit the data to a Normal distribution
    % Sort Array
    samArray_sort = sort(samArray)';

    % Fit to Normal Distribution
    pd = fitdist(samArray_sort, 'Normal');

    % Find values of the PDF for our data
    samArrayNormal = pdf(pd, samArray_sort);
```

7 In this section of the script we find the actual and normal fitted distributions of our sample array we generated. The importance step here is to remember to sort the sample array we generated and pass the sorted array to 'fitdist' so we create a distribution object that correctly formatted. All these functions have been used in previous computer assignments.

```
MSE_samArray(n) = myMSE(h_samArray, samArrayNormal);
```

8 Calling our mean squared error function allows us to find the MSE between our normal fitted and actual pdfs.

```
    % Special Cases we want to observe
    if n==1 || n==10 || n==100
        figure('name','[ECE 3522] Class Assignment [8]');

        % Plot the Histogram and Normal on same plot
        histogram(samArray, bounds, 'Normalization', 'pdf');

        hold on

        % Plot the Normal Distribution
        plot(samArray_sort, samArrayNormal, 'r');

        % Define axis
        title(sprintf('Actual Distribution and Normal Distribution when n = %0.1f',n));
        xlabel(sprintf('x (Bin size of %0.2f)',binSize));
        ylabel('Frequency');

        hold off
    end
end
```

9 We end our for-loop by looking for three special cases which we wish to pick out from the rest of the bunch. The assignment calls for plotting the actual PDF and normal fitted distribution for when n=1, 10, and 100. When our for-loop reaches the desired case we branch into the if statement block and executed plotting the histogram and normal distribution fit to a new figure.

```
%Plot all the things
figure('name','[ECE 3522] Class Assignment [8]');
semilogx(samMean);
ylim([-1 1]);
grid on
xlabel('Number of Random Variables (n)')
ylabel('Mean of sample');
title('Mean vs Sample Size (N)')

figure('name','[ECE 3522] Class Assignment [8]');
semilogx(samVar);
grid on
xlabel('Number of Random Variables (n)')
ylabel('Variance of sample');
title('Variance vs Sample Size (N)')

figure('name','[ECE 3522] Class Assignment [8]');
semilogx(meanErr);
grid on
xlabel('Number of Random Variables (n)')
ylabel('Mean Error of sample');
title('Error of Mean vs Sample Size (N)')

figure('name','[ECE 3522] Class Assignment [8]');
semilogx(varErr);
grid on
xlabel('Number of Random Variables (n)')
ylabel('Variance Error of sample');
title('Error of Variance vs Sample Size (N)')

figure('name','[ECE 3522] Class Assignment [8]');
semilogx(MSE_samArray);
grid on
xlabel('Number of Random Variables (n)')
ylabel('MSE of PDF');
title('MSE Actual PDF vs Normal Distribution')
```

10 Now that we have finished looping through all possible variables sizes n, we plot the results for what our actual mean and variance are. We also plot the error between the expected mean and variance are in comparison to the actual mean and variance. Lastly, we plot the MSE for all variable sizes we computed.

The next section of code outlines the script used to complete part four of the assignment.

```
%% Part 4
clear; clc; close all

% Number random Variables
nRandomVar = 10;
NSamples = 10000;
binSize = 0.1;

% Lower and Upper bound of random Number Generation
lRange = 0;
uRange = 1;

% Excpected Mean/Variance for Uniform Distribution
eMean = (uRange+lRange)/2*nRandomVar;
eVar = (uRange-lRange)^2/12*nRandomVar;
eStd = sqrt(eVar);

% Histogram Bounds
bounds = [nRandomVar*lRange:binSize:nRandomVar*uRange];
numBins = length(bounds);
```

11 As in the previous part we define our input parameters and adjust mean, variance, and histogram bounds accordingly.

```
% Create RN using two techniques

%--------Sum of Uniform Technique--------

fprintf('Sum uniform Generation\n');
% Time the generation
tic
% Generate Random Array
samArrayUGen = myunifDist(nRandomVar, NSamples, lRange, uRange);
toc
```

12 Next we will time the performance of our 'myunifDist' function. The process is what we used in the previous parts to develop a normal distribution of random variables.

```
% Plot
figure('name','[ECE 3522] Class Assignment [8]');
% Estimate the pdf
h_samArrayUGen = histcounts(samArrayUGen, bounds, 'Normalization', 'pdf');
histogram(samArrayUGen, bounds, 'Normalization', 'pdf');
hold on

% Use a function to fit the data to a Normal distribution
% Sort Array
samArrayUGen_sort = sort(samArrayUGen)';

% Fit to Normal Distribution
pd = fitdist(samArrayUGen_sort, 'Normal');

% Find values of the PDF for our data
samArrayUGenNormal = pdf(pd, samArrayUGen_sort);

% Plot the Normal Distribution
plot(samArrayUGen_sort, samArrayUGenNormal, 'r');
title(sprintf('Actual Distribution and Normal Distribution using Uniform Generation when n
= %0.1f',nRandomVar));
xlabel(sprintf('x (Bin size of %0.2f)',binSize));
ylabel('Frequency');
hold off

MSE_samArrayUGen = myMSE(h_samArrayUGen, samArrayUGenNormal)
```

13 As in the previous parts we plot the actual pdf and overlay a normal distrubiton. Using our mean squared error function we find the MSE between our actual and normal fitted PDFs.

```
%---------Muller Generation--------

fprintf('Muller Transform Generation \n');
tic
% Find random Numbers using Muller Transform
samArrayMGen = myNormDist(eMean, eStd, NSamples);
toc

% Plot
figure('name','[ECE 3522] Class Assignment [8]');
% Estimate the pdf
h_samArrayMGen = histcounts(samArrayMGen, bounds, 'Normalization', 'pdf');
histogram(samArrayMGen, bounds, 'Normalization', 'pdf');
hold on

% Use a function to fit the data to a Normal distribution
% Sort Array
samArrayMGen_sort = sort(samArrayMGen)';

% Fit to Normal Distribution
pd = fitdist(samArrayMGen_sort, 'Normal');

% Find values of the PDF for our data
samArrayMGenNormal = pdf(pd, samArrayMGen_sort);

% Plot the Normal Distribution
plot(samArrayMGen_sort, samArrayMGenNormal, 'r');
title(sprintf('Actual Distribution and Normal Distribution using Muller Generation when n =
%0.1f', nRandomVar));
xlabel(sprintf('x (Bin size of %0.2f)',binSize));
ylabel('Frequency');
hold off

MSE_samArrayMGen = myMSE(h_samArrayMGen, samArrayMGenNormal)
```

14 We repeat the steps taken for the uniform distribution method for the Muller method. Again we time using the tic/toc functions built into MatLab.

IV. CONCLUSIONS

In our trials we have shown how we can create a normal distribution of random variables using application of the central limit theorem. The turning point generally accepted as the turning point for creating a normal distribution out of other distributed random variables is a n>=30. In our experiment at only a variable size of 10 we observe quite normally distributed variables. This application of the central limit theorem can be used to understand why white noise is apparent in most signals. In a open environment, you have a combination of sound sources combining together. If you start a recording the microphone will pick up the surrounding sounds and thus create an underlying Gaussian white noise in your recording. Albeit the noise will have less amplitude than you voice which you intended to record, but the noise is still there.

In the latter point of the assignment we compared and contrasted the central limit theorem uniform RV creation of a normal distribution to the Box-Muller technique. In a fraction of the time we were able to create our distribution of normal variables. In the future we can utilize this technique for our analysis purposes if we wanted to create a very large array of samples.