Title:

ECE 345 Project 2 - CLT & Radar

Members:

John Schaff, Scott Lee, Tianqi Wang, Parth Patel

Professor:

Dr. Zhou

Date:

December 9, 2016

**Part 1**

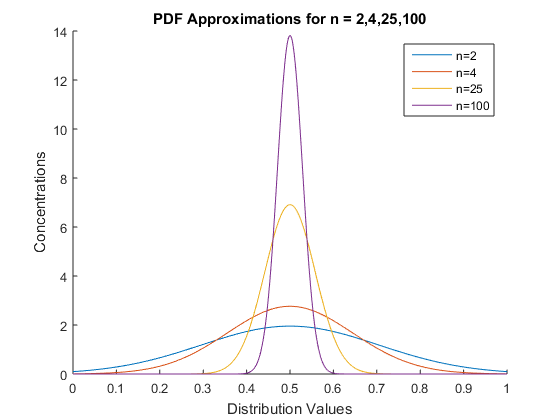
1. Due to the Uniform Distribution for each random variable Xi, we also know that any sum of those random variables will also have the following mean, variance, and standard deviation. Using the general form of the Central Limit Theorem, the probability density function for different values of n can be solved for assuming that the distribution can modeled by a Gaussian distribution. We were given x is a random variable uniformly distributed over [0,1] and the function Mn is sum of these random variables for n summations and then divided by the number of random variables. Even though the Central Limit Theorem is used a limit as n approaches infinity, the approximation comes from assuming that our number of n is sufficient to make this approximation. Looking at the plots in MATLAB, the assumption holds decently once n = 4, as this apparently begins to show more of the Gaussian characteristics of the probability density function. Overall, the central limit theorem is an appropriate approximation for a uniformly distributed variable and the operations performed on them.

CLT Approximation for Mn |

In general form,

General Equation

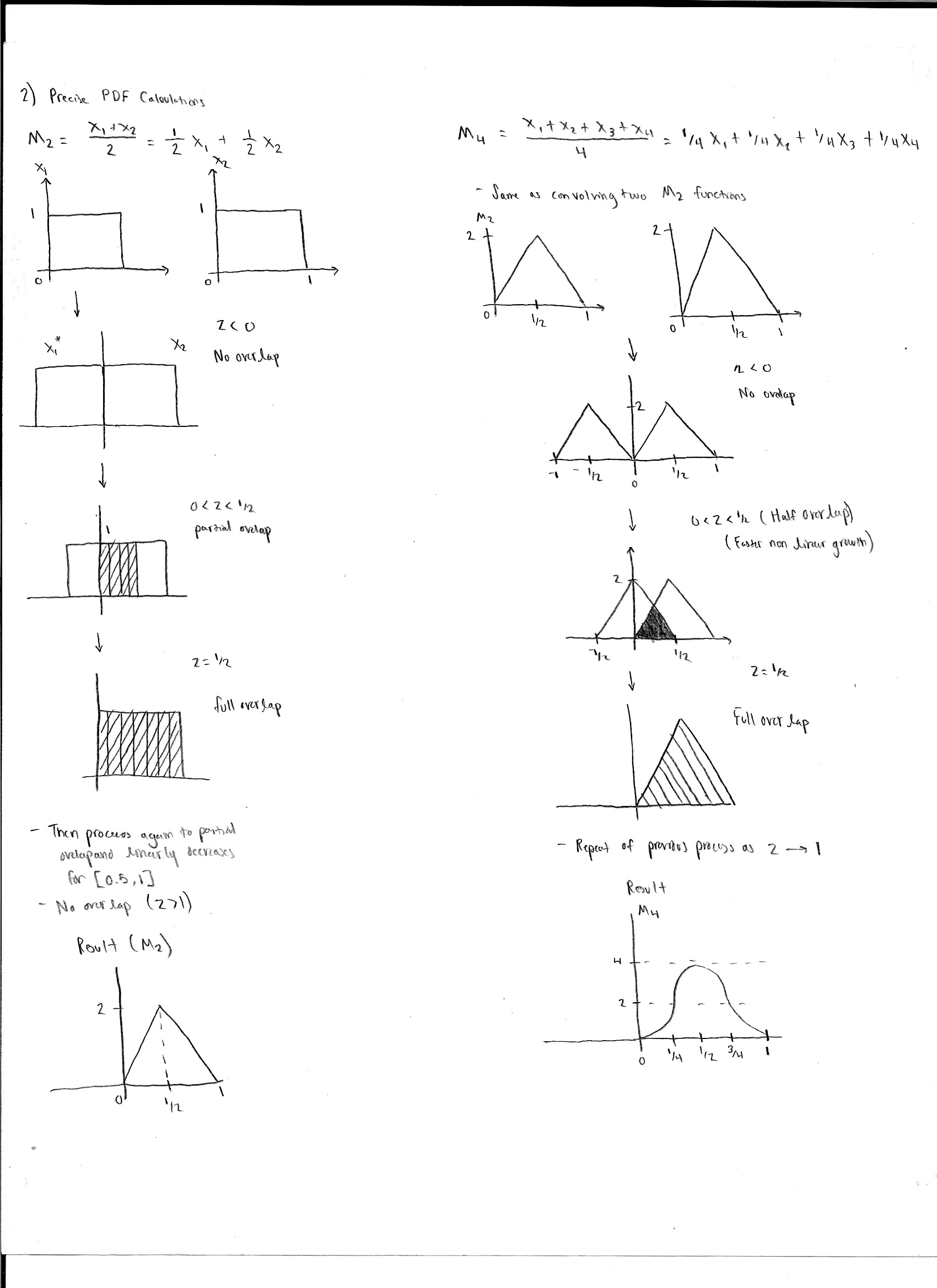
Looking at the plots for the different n distribution probability distributions, the area should stay approximately the same at 1. So it makes sense for the peak to begin increasing and become more narrow as the distribution becomes more and more Gaussian. We used this logic to rationally check that our approximations made sense before proceeding.

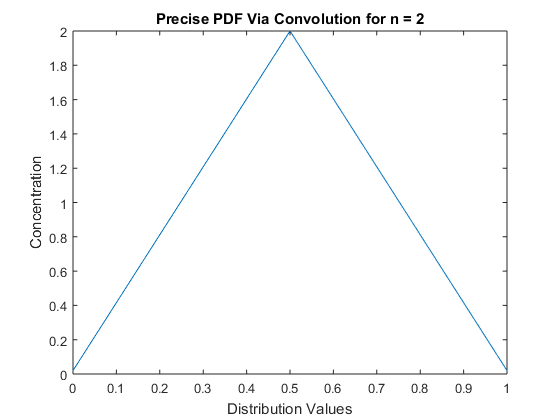


**Figure (Approximate PDF’s from CLT for n = 2,4,25,100)**

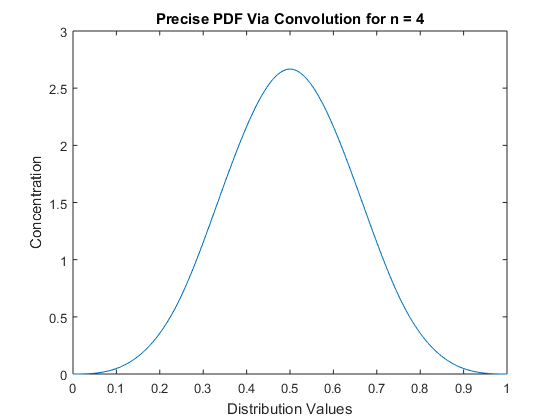
**b)**

Using the Convolution integral : . This is referring to the convolution of this has a denotation of . For the specific case of n=2 and n=4, we will use the convolution integral in the form of. Eventually if the number of convolutions is large enough, then the final form of the probability density function does begin to form a Gaussian distribution. Even after n = 4 convolutions, the form already begins to resemble the other distributions. We computed the convolution in MATLAB after graphically estimating the figure to see how close our graphical estimation was to the more precise determination in MATLAB. Besides slight differences in the height and values on the y axis, the general shape seemed decently close. We simply tried to see what it would look like before making the computations in MATLAB first by approaching a graphical convolution. We started by determining M2, which was significantly easier to do as it was just convolving two rectangles. It would make sense to get a triangle as the result due to the linear increase and decrease as the rectangle was shifted across in the convolution. However the calculation for M4  is a bit more challenging. In this case we knew there were several critical points. The first would be the complete overlap. This point would yield the largest value and the peak of the distribution function. Then the next two critical points were the increasing and decreasing parts of the graph. The increase would initially be a greater rate as it goes up the convolved triangle overlaps with the increasing area of the stationary triangle function. However, the rate would then decrease after it cross halfway through the other triangle. This is the basis for the initial sharp rate to gradual drop-off at the peak, and the inverse after the full overlap had finished. While the by hand calculations weren’t perfectly accurate, it was good stepping stone to see if we were thinking correctly about the convolution and to check our results in MATLAB to see if they made sense. The convolutions are shown below:





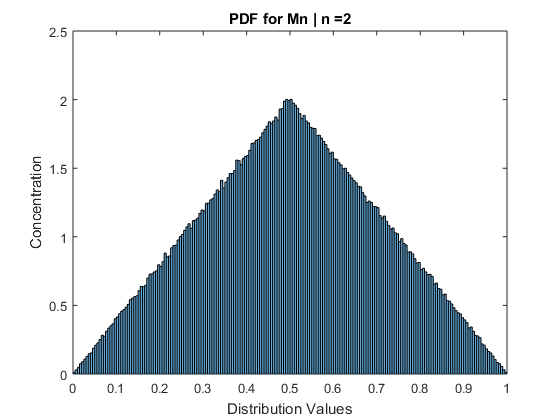
**Figure (Precise PDF for M2 via Convolution)**



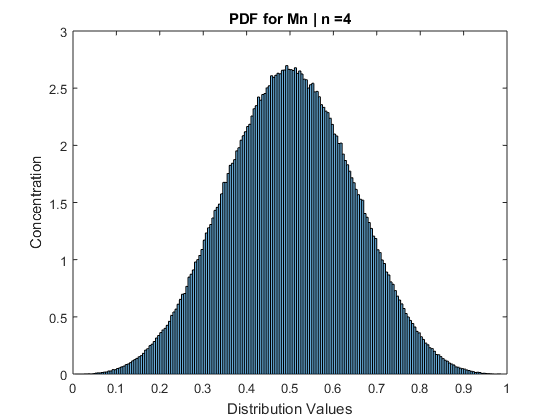
**Figure (Precise PDF for M4 via Convolution)**

**c)**

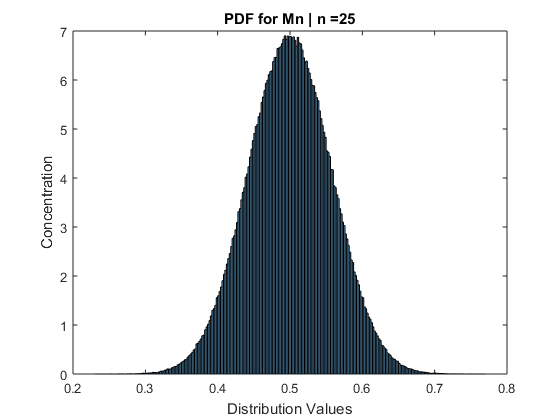
The simulated probability density functions for the uniform distribution from zero to one begin to show the impact of the central limit theorem. In the beginning, the probability density function histogram shows the initial triangular shape shown in the precise probability density function calculations. We expected this after seeing both the initial CLT approximation and the convolution method to determine our initial guess of the function. The mean at an early stage is also represented by the uniform distribution, as it always has the highest distribution at the mean of 0.50 due to the limits being from 0 to 1. This result is an indicator of the low value of n used for the assumption of the central limit theorem. The central limit theorem implicitly requires larger values of n for more accuracy in determining the probability density function. So the Gaussian distribution shouldn’t start to from until n reaches larger values. However, at M4 the probability density function already begins to show the appearance of the Gaussian distribution. Even for a small n value, the central limit theorem can be used quite effectively to give a decent Gaussian approximation. After the lower values of n, the upper values definitely demonstrate the approach to infinity as the difference between M25 and M100 is negligible and difficult to distinguish apart. The main approach for generating these probability density functions in MATLAB came from generating large vectors of data to initially represent the trials of the function. Then I would take the mean of all those values to get more accurate data for plotting. So the larger the n value would have more values to compare to when computing the mean. Then we compiled the data into the histogram function and normalized it to have the appearance of a probability density function. The normalization was important for both comparison and also checking to make sure the functions had the same characteristics of the precise or approximate probability density functions. Luckily the histogram has a built in normalization factor to make this calculation easier to perform. Looking at the initial n =2 and n =4 probability density functions, the values are slightly off from the by hand calculations but still retain the same basic shape shown.



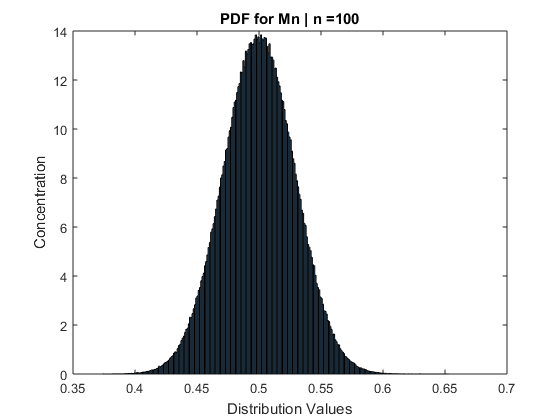
**Figure (PDF for M2)**



**Figure (PDF for M4)**

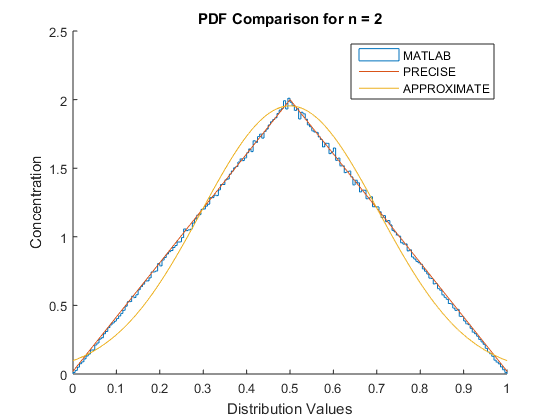


**Figure (PDF for M25)**

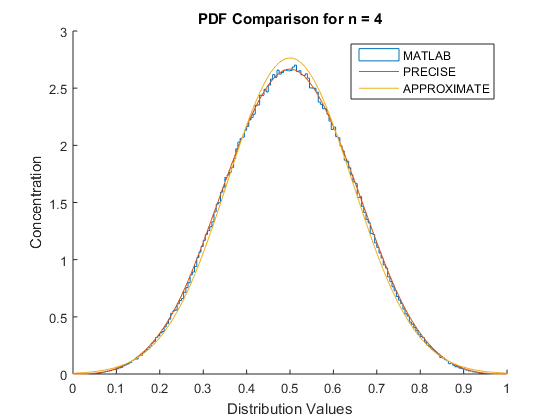


**Figure (PDF for M100)**

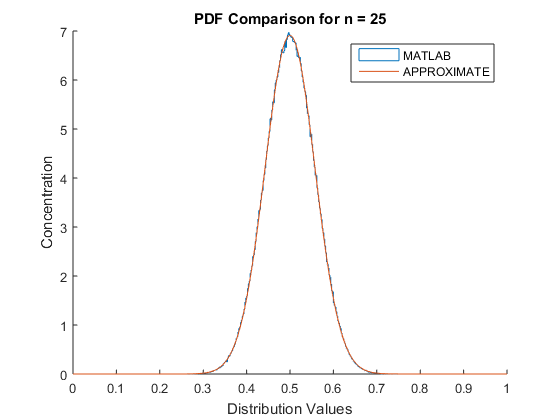
**d)** Looking at all four comparisons, all seem to quite precise and within a very small range of the other distribution attempts. The only major differences can be observed at the small n value for n = 2. At this point, the most amount of errors exists due to the small number of samples compared to the other n values tested. Even in this case, the difference is fairly trivia. The main part for solving the comparison simply involved collecting all the data we had gotten and plotting on the same graph for each n value we had tested. The precise and MATLAB derived function simply to be almost identical for every case, while the approximate shows errors at the lower values of n. This is likely due to the dependence on the number of n values compared to the convolution method. The convolution method may be more computationally expensive to determine, but it relies only on the shape of the random variable. This contrasts with the central limit theorem, which is computationally simple to perform and grasp, yet has a dependence on the number of samples associated with the data and requires a larger number of samples to assume the Gaussian distribution. The MATLAB simulation values will always have some error associated with them, but assuming the number of trials is quite high then it will likely achieve the same error as the precise convolution method.



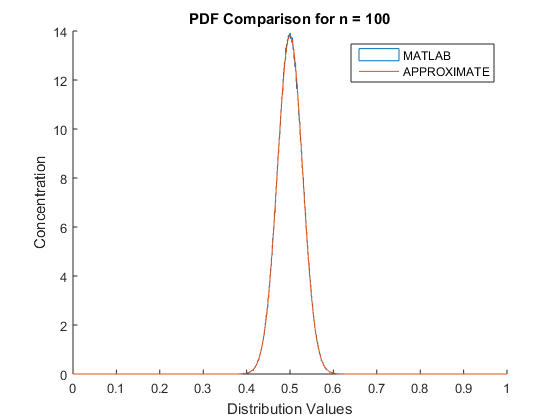
**Figure (PDF Comparison for n = 2)**



**Figure (PDF Comparison for n = 4)**

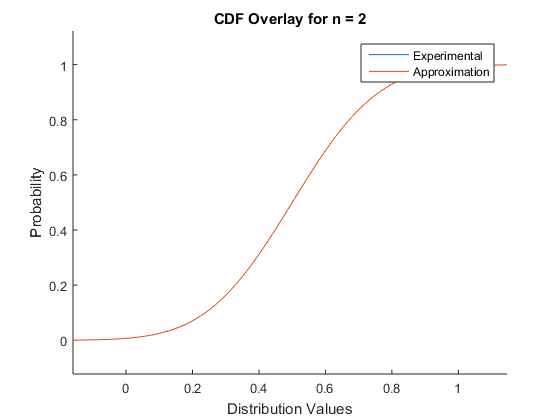


**Figure (PDF Comparison for n = 25)**

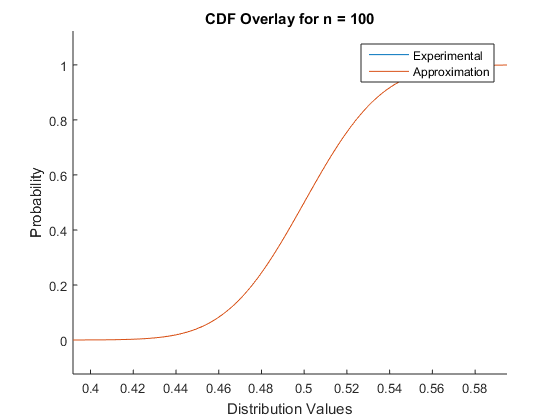


**Figure (PDF Comparison for n = 100)**

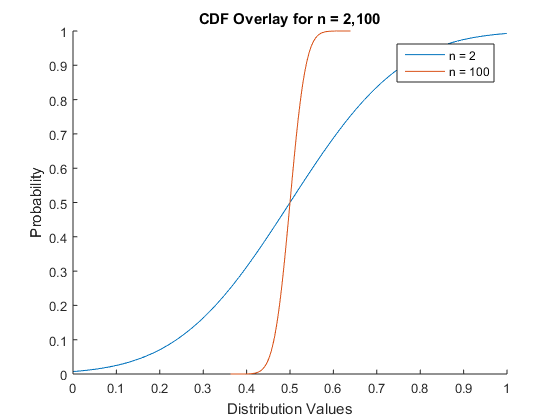
**e)** Since we’ve been working with probability density functions this whole time, the cumulative distribution function is always a nearby tool to examine the data as well to compare with the probability density function. To acquire the cumulative distribution function for the approximation, we had already made the probability density function into an anonymous function in MATLAB. All that we had to do was convert the anonymous function into a symbolic equation and use an integration function over the limits we wanted to see for a cumulative density function. Due to the fact the value beyond 1 will always be 1, we simply decided to integrate the probability function from [0,1]. Then to get the cumulative density function for the experimental simulation data, I decided to use an anonymous function representing the central limit theorem and plug the data into that function to get the normalized data. I then sorted it and determined the appropriate p values from that data before plotting it to get the cumulative distribution function for the simulated MATLAB data as well. This ended up giving quite identical results, as the two cumulative distribution function are nearly indistinguishable from each other. It requires looking at the third graph comparing the simulation data for n = 2 and n = 100 to see the difference. Due to the much narrower probability density function at n = 100, it makes sense for the cumulative density function to reach 1 at a much quicker rate compared to the cumulative density function for the function at n = 2. Regardless, both results seem to reflect the same general trends of the data as the probability density function did, so the results still seem to indicate the success of our simulation.



**Figure (CDF Comparison for n = 2)**



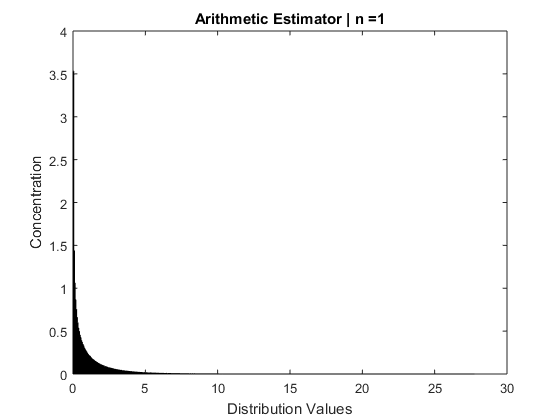
**Figure (CDF Comparison for n =100)**



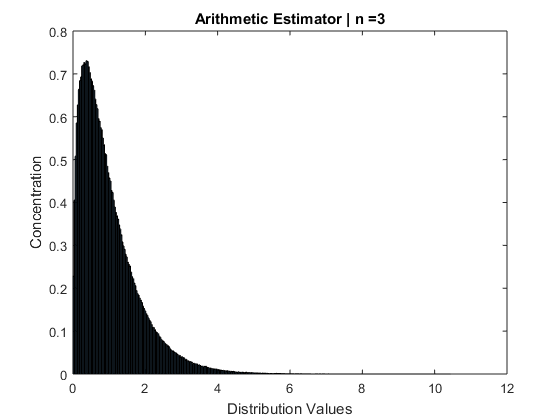
**Figure (CDF Overlay for n = 2 and n = 100)**

**Part 2**

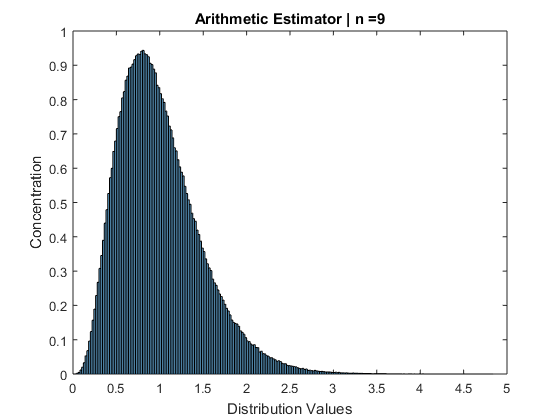
**a)**



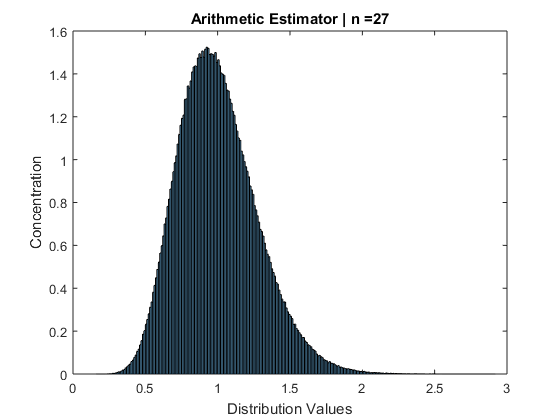
**Figure (Arithmetic Estimator | n=1)**



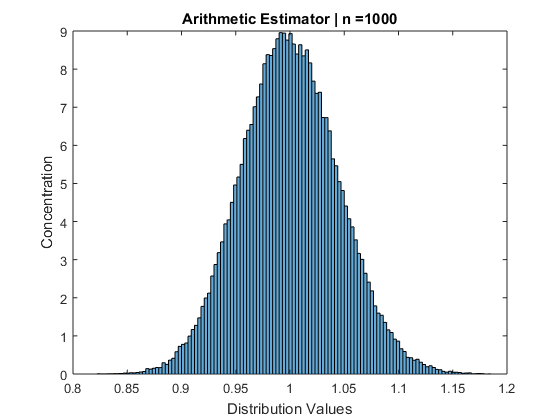
**Figure (Arithmetic Estimator | n=3)**



**Figure (Arithmetic Estimator | n=9)**



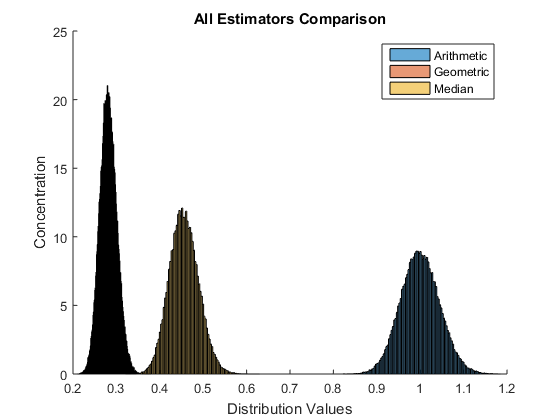
**Figure (Arithmetic Estimator | n =27)**



**Figure (Arithmetic Estimator | n =1000)**

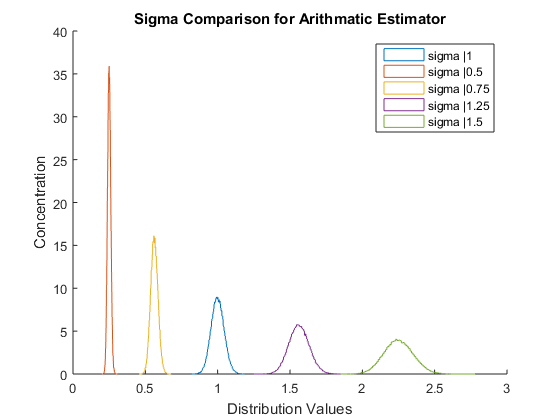
This problem is quite similar to the first part. The arithmetic estimator is the same as Mn from the first function, and the trends shown from this data reflect the same Gaussian trend as the n values increase. The first n value doesn’t even really reflect the estimator, as it just ends up being the original data. This shouldn’t have a trend even close to Gaussian as the number of n values is far too lower to even try to approximate via the central limit theorem. Then as n increases, the distribution values become more normal and the shape of the probability density function narrows to look like the Gaussian curve. Also, the different value for µ is more apparent in this example as the initial mean for low n is much different from large n. The density function shifts slowly with the increase in n to reflect the actual mean being 1 as true. To achieve these results, we used a random function to produce data with a set mean and sigma to try and model the voltage function presented in the problem. We set the mean of the data to be zero and sigma to be 1 based off the voltage function and the specifications dictated in the problem. Then we represented the power function by squaring that data and applying the arithmetic estimator to see how the data appeared. Overall, the results seemed to reflect what we expected for the function, however the mean of 1 seems to differ from the specified mean of the voltage function, where the mean was 0. This may be due to the inherent bias of the estimator for looking at given data. This may be due the squaring of the data when we generate the power vector, as this could impact the biasing of the estimators and which ones will be more or less biased.

**b)** After implementing the arithmetic estimator, we then implemented the geometric and median estimator to see how these estimators compared when modeling the voltage and power functions. We initially implemented the geometric function without using a built-in MATLAB function, but after acknowledging we could implement the geometric function, we decided to use the geomean function in MATLAB for the sake of speed due to the large matrices we were working with and the number of those matrices for comparison. The median estimator simply involved acquiring the median of the data we had generated with random function. We had to a magnitude of 10 less trials for this simulation due to sheer size of the matrices, but the results still came out clean so I shouldn’t impact the results. Seeing the three estimators on the same plot, it’s quite obvious that the geometric and median estimators introduce a fair amount of bias into data. The original mean of the voltage function was 1 and the sigma was variable depending on what the data was given to the voltage function.

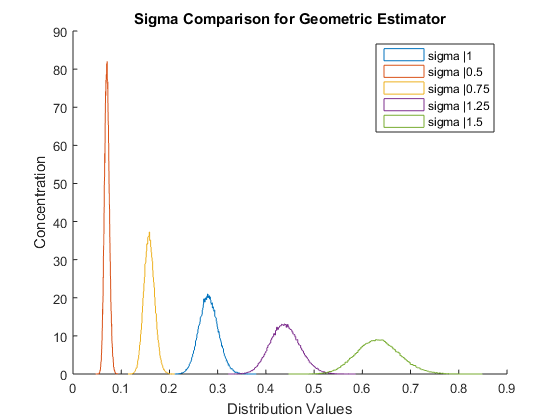


**Figure (Arithmetic, Geometric, and Median Estimators | n =1000)**

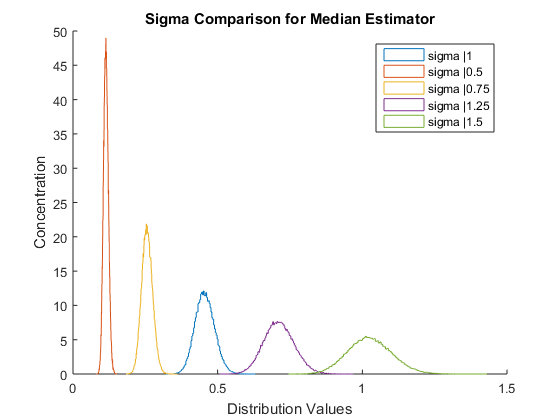
**c)**



**Figure (Arithmetic PDF Comparison for Sigma)**



**Figure (Geometric PDF Comparison for Sigma)**



**Figure (Median PDF Comparison for Sigma)**

**d)** The main indicator of the estimator for this case is by looking at the mean of all three estimators and comparing it to the original voltage and power probability density function. The original probability function for the voltage had mean of 0 and a variable sigma. So for all cases, the unbiased estimators will yield a mean of 0. For the cases we examined in part b, we saw how different each mean was compared to the original mean of the random data. The least biased estimator from the perspective of the mean came from the median estimator, as this estimator was the closest to the actual mean of the voltage function. Then the geometric estimator had a slightly larger mean compared to the median estimator. Finally, the arithmetic estimator was the least accurate having a mean of almost exactly 1. Also, the width of the estimators for n = 1000 and sigma = 1 show that the median estimator is the narrowest, then geometric, and then arithmetic. The tighter distribution values represent the small variance of the estimator, reflecting the strength of the median in this simulation for looking at the several different estimators.