ECE 8527 Homework Number 3: Random Processed Revisited

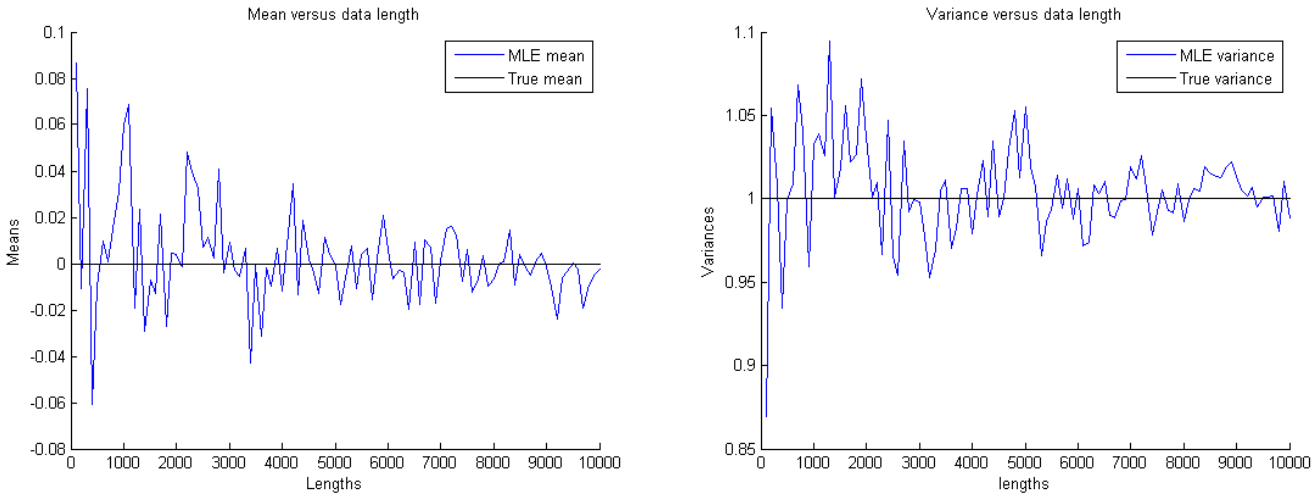
Andrew Powell

March 10, 2014

1 Problem 1

For Problem 1, a series of random data sets are generated with a normal distribution whose mean is $\mu = 0$ and variance is $\sigma^2 = 1$. The length N of each data set in the series changes as the MATLAB vector $N = 1 \times 10^2 : 1 \times 10^2 : 1 \times 10^4$. The results are shown in Figure 1 as two plots, where the leftward plot shows how the Maximum Likelihood estimate of the mean $\hat{\mu}$ changes with respect to the data set's length N and the rightward plot shows how the Maximum Likelihood estimate of the variance $\hat{\sigma^2}$ changes with respect to the length.

Figure 1: Gaussian parameters versus data length N



The results shown in Figure 1 easily demonstrates more data increases the probability the estimated parameters will be closer to the true parameters of the data set's underlining distribution.

2 Problem 2

2.1 Theoretical Results

$$\begin{aligned} R_{xx}[k] &= \mathrm{E}[x_n x_{n-k}] \\ &= \frac{1}{N}\sum_{n=0}^{N-1} x[n]x[n-k] \end{aligned} \tag{1}$$

$$\begin{aligned} C_{i,j} &= \mathrm{E}[x_{n-i} x_{n-j}] \\ &= \frac{1}{N}\sum_{n=0}^{N-1} x[n-i]x[n-j] \end{aligned} \tag{2}$$

There are a number of assumptions and small "modifications" that need to be mentioned before presenting the answers to Problem 2. The autocorrelation and covariance operations utilized for this problem are shown in Equations 2 and 3, respectively. The reason why the fraction $\frac{1}{N}$ is included to "normalize" the two operations is to simply make the results appear more familiar. It is acknowledged in the field of Digital Signal Processing the fraction is typically dropped since the scaling doesn't change the outcome of the results; however, from my point of view, it makes more sense to include the scaling since the expectation operator E[•] informally is defined as the value one would expected as the result of averaging an infinite amount of variates from the random variable for which the expectation is determined (*I hope what I said made sense*). In regards to the definition of covariance in Problem 2, it is assumed the reason why the mean is not subtracted from each data point is because, for both the gaussian and sine wave examples, the mean is assumed zero.

$$R_{xx}[k] = \begin{cases} \sigma^2 & \text{where } k = 0 \\ 0 & \text{where } k \neq 0 \end{cases} \tag{3}$$

$$C_{ij} = \begin{cases} \sigma^2 & \text{where } i = j \\ 0 & \text{where } i \neq j \end{cases} \tag{4}$$

Equations 3 and 4 are the expressions for a Gaussian distribution's autocorrelation $R_{xx}[k]$ and covariance C_{ij}, respectively.

$$\begin{aligned} \text{Power Spectral Density (PSD}(k)) \;&=\; \text{Fourier Transform}\{R_{xx}[l]\} \\ R_{xx}(k) = |X(e^{j2\pi k})|^2 &= \sum_{n=0}^{N-1} R_{xx}[n] e^{-\frac{j2\pi kn}{N}} \end{aligned} \tag{5}$$

The relationship between the autocorrelation of the signal $x[n]$ and the Fourier Transform is known as the Wiener-Khinchin theorem and is shown in Equation 5.

$$R[0] \geq |R[k]| \tag{6}$$

The relationship between $R[0]$ and $|R[k]|$ is shown in Equation 6. Because autocorrelation is essentially multiplying the signal $x[n]$ with a replicate whose phase varies, the point at which the phase is zero, $k = 0$, indicates the signal is being multiplied with itself. Thus, unless the signal is constant, it is impossible for the absolute value of the autocorrelation function $|R[k]|$, where $k \neq 0$, to even be equal to the autocorrelation for which the phase is zero, $R[0]$. As far as the shape of the autocorrelation, the relationship simply implies the center of the autocorrelation—that is, $R(0)$—will either have the largest magnitude or a magnitude equivalent to largest magnitude.

$$x[n] = A \sin\left(\frac{2\pi f}{f_s} n\right) \tag{7}$$

$$
\begin{aligned}
R(k) &= \frac{A^2}{N}\sum_{n=0}^{N-1}\left[\sin\left(\frac{2\pi f}{f_s}n\right)\right]\left[\sin\left(\frac{2\pi f}{f_s}(n-k)\right)\right] \\
&= \frac{A^2}{2N}\left[N\cos\left(\frac{2\pi f}{f_s}k\right) - \sum_{n=0}^{N-1}\cos\left(\frac{2\pi f}{f_s}(2n-k)\right)\right] \\
&= \frac{A^2}{2N}\left[N\cos\left(\frac{2\pi f}{f_s}k\right) - \sum_{n=0}^{N-1}\left[\cos\left(\frac{2\pi f}{f_s}2n\right)\cos\left(\frac{2\pi f}{f_s}k\right) + \sin\left(\frac{2\pi f}{f_s}2n\right)\sin\left(\frac{2\pi f}{f_s}k\right)\right]\right] \\
&= \frac{A^2}{2N}\left[N\cos\left(\frac{2\pi f}{f_s}k\right) - \left[\sum_{n=0}^{N-1}\cos\left(\frac{2\pi f}{f_s}2n\right)\cos\left(\frac{2\pi f}{f_s}k\right) + \sum_{n=0}^{N-1}\sin\left(\frac{2\pi f}{f_s}2n\right)\sin\left(\frac{2\pi f}{f_s}k\right)\right]\right] \\
&= \frac{A^2}{2N}\left[N\cos\left(\frac{2\pi f}{f_s}k\right) - \left[\cos\left(\frac{2\pi f}{f_s}k\right)\sum_{n=0}^{N-1}\cos\left(\frac{2\pi f}{f_s}2n\right) + \sin\left(\frac{2\pi f}{f_s}k\right)\sum_{n=0}^{N-1}\sin\left(\frac{2\pi f}{f_s}2n\right)\right]\right] \\
&= \frac{A^2}{2N}\left[N\cos\left(\frac{2\pi f}{f_s}k\right) - \left[\cos\left(\frac{2\pi f}{f_s}k\right)0 + \sin\left(\frac{2\pi f}{f_s}k\right)0\right]\right] \\
&= \frac{A^2}{2}\cos\left(\frac{2\pi f}{f_s}k\right) \\
&= \sigma^2\cos\left(\frac{2\pi f}{f_s}k\right)
\end{aligned}
\tag{8}
$$

$$
\begin{aligned}
C_{i,j} &= \frac{A^2}{N}\sum_{n=0}^{N-1}\sin\left(\frac{2\pi f}{f_s}(n-i)\right)\sin\left(\frac{2\pi f}{f_s}(n-j)\right) \\
C_{i,j} &= \frac{A^2}{2N}\sum_{n=0}^{N-1}\left[\cos\left(\frac{2\pi f}{f_s}(j-i)\right) - \cos\left(\frac{2\pi f}{f_s}(2n-(i+j))\right)\right] \\
C_{i,j} &= \frac{A^2}{2N}\left[\sum_{n=0}^{N-1}\cos\left(\frac{2\pi f}{f_s}(j-i)\right) - \sum_{n=0}^{N-1}\cos\left(\frac{2\pi f}{f_s}(2n-(i+j))\right)\right] \\
C_{i,j} &= \frac{A^2}{2N}\left[N\cos\left(\frac{2\pi f}{f_s}(j-i)\right) - 0\right] \\
C_{i,j} &= \frac{A^2}{2}\cos\left(\frac{2\pi f}{f_s}(j-i)\right) \\
&= \sigma^2\cos\left(\frac{2\pi f}{f_s}(j-i)\right)
\end{aligned}
\tag{9}
$$

Equations 8 and 9 respectively demonstrate how the autocorrelation $R_{xx}[k]$ and covariance matrix $C_{i,j}$ for a "generic" sine wave are derived. The generic sine wave is assumed $x(t) = A\sin(2\pi ft)$, for which A is the amplitude, f is the frequency, and t is the time. t is defined as $\frac{n}{f_s}$ in order to determine $x(t)$ in samples; that is, $x[n] = A\sin(\frac{2\pi f}{f_s}n)$. It is imperative the sample frequency f_s, which is measured in samples per second, is larger than the Nyquist rate $f_s \geq 2 \times f$ so as to avoid aliasing. N is the "window size" in samples. As shown in the expressions for autocorrelation and covariance (that is, Equations 8 and 9), the window size N causes a number of the summations to sum to zero if the window size N is a multiple of the generic sine wave's period $T = \frac{1}{f}$. In other words, the window size $N = \frac{f}{f_s} \times r$, where r is a positive integer. The sine wave's phase is ignored so as to avoid complications in the simulations, or experimental results generated from MATLAB. The final expression for the generic sine wave

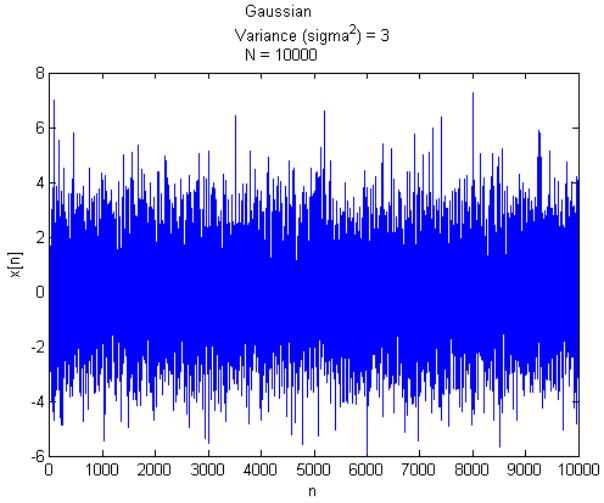
is shown in Equation 7.

According to MATLAB, if the window size N is greater than 1, the rank is always 2; otherwise, the rank is 1. Why? Clearly, only two columns of the covariance matrix can be independent, whereas the other columns are linear combinations of the two independent columns. Please note, the window size N (obviously) pare integers defined for values greater than 0.

On a serious note, the reason for the rank could have a relation to the sine wave's frequency decomposition, wherein there are two impulse functions that define any sinusoidal wave. White noise—the data set of the Gaussian distribution—includes an infinite amount of different frequencies and its covariance matrix is the identity whose rank is then also infinity since white noise would imply the data set's size N is going to infinity.

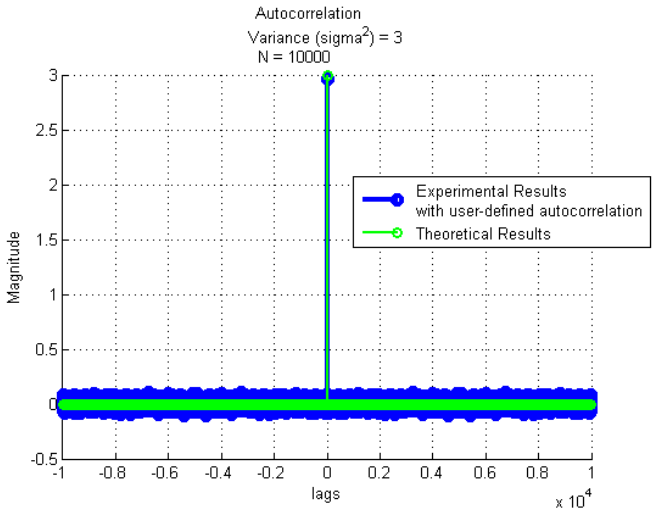
2.2 Experimental Results

Figure 2: The window of white noise $x[n]$ utilized for generating the autocorrelation $R_{xx}[k]$



The white noise utilized for the experimental results for the autocorrelation is shown in Figure 2.

Figure 3: White noise's autocorrelation $R_{xx}[k]$



For the given parameters, the white noises' experimental and theoretical autocorrelations are shown in Figure 3. As described in the plot's legend, the experimental results are generated with an user-defined correlation function. The reason for self defining an correlation function is because the correlation function MATLAB provides only appends zeros to the data when performing the shift and dot product-product between the data—or, in this case, the data and another copy of the data at a different phase.

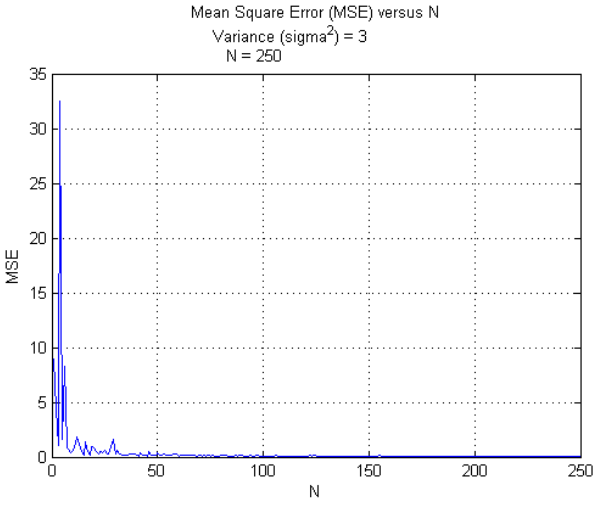
The correlation function defined for Homework 3 instead performs a *circular shift* operation on the second copy of the data. The circular shift operation simply causes the data to shift back a single index and the last data point becomes the first data point in the data. The reason for the circular shift operation instead of appending zeros is to, in a sense, trick the autocorrelation function into thinking the data is periodic for the window size N. This idea of performing a circular shift operation on the second copy of the data when determining the autocorrelation actually makes more sense with the sine wave (whose experimental results are described after the white noise's experimental results). For white noise, new normally distributed data points should have been generated instead of appending zeros. Autocorrelation utilizing the circular shift operation, however, is simpler to implement.

The phase shifts performed for the generation of the experimental covariance matrices also apply the circular shift operations for changes in phase.

It is also worth noticing the impulse function located at lag equal to 0. The height of the impulse function, as expected, is equal to the variance σ^2(or is it more accurate to refer to this variance as the average variance, considering N represents the size of a window, rather than an infinitely sized data set?).

$$\text{MSE} \;=\; \text{E}_{i,j}[(C_{th_{i,j}} - C_{ex_{i,j}})^2] \tag{10}$$

Figure 4: Mean Square Error of white noise's covariance matrix $C_{i,j}$

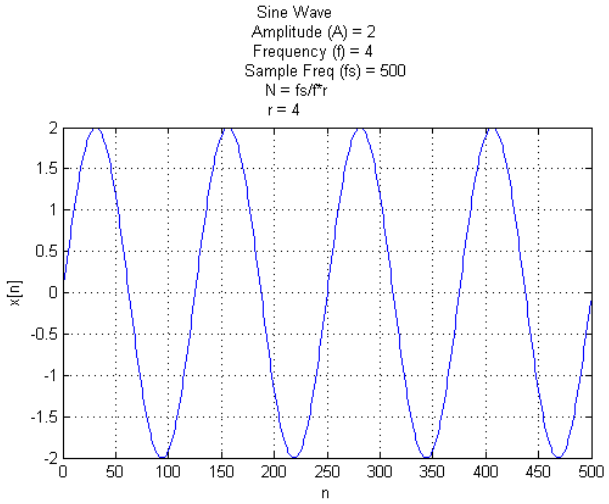


Instead of displaying a few instances of the theoretical and experimental covariance matrices $C_{i,j}$, both covariance matrices are generated for values of N, ranging from 1 to 250. After the theoretical covariance matrix, based on Equation 4, and the experimental covariance matrix, based solely on the data, are generated for each N, the Mean Square Error (MSE) is computed according to Equation 10. All the MSEs are collected into a single vector and are potted against

all the window size N in order to create Figure 4.

As demonstrated from Figure 4, the MSE jumps around relatively high values for values of N approximately less than 100; afterward, the MSE seemingly converges to 0.

Figure 5: Sine wave window $x[n]$ utilized for generating the autocorrelation $R_{xx}[k]$



For the MATLAB experimental results for determining the autocorrelation $R_{xx}[k]$, the sine wave $x[n]$ shown in Figure 5 is utilized. Please note most of the methods utilized to compute the experimental results for the sine wave are identical to the methods utilized for the white noise.

Figure 6: Sine waves's autocorrelation $R_{xx}[k]$

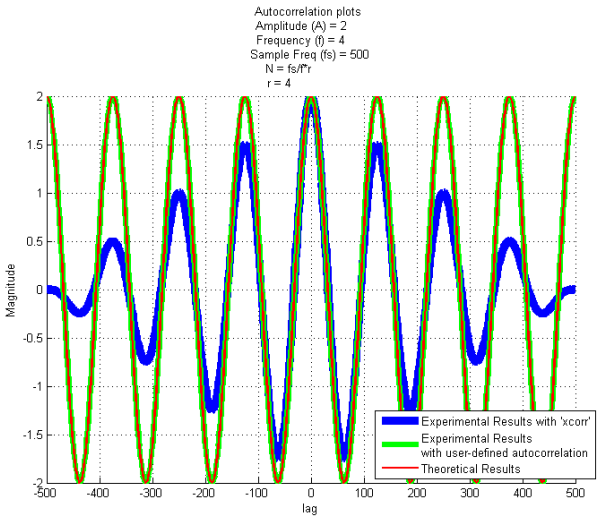
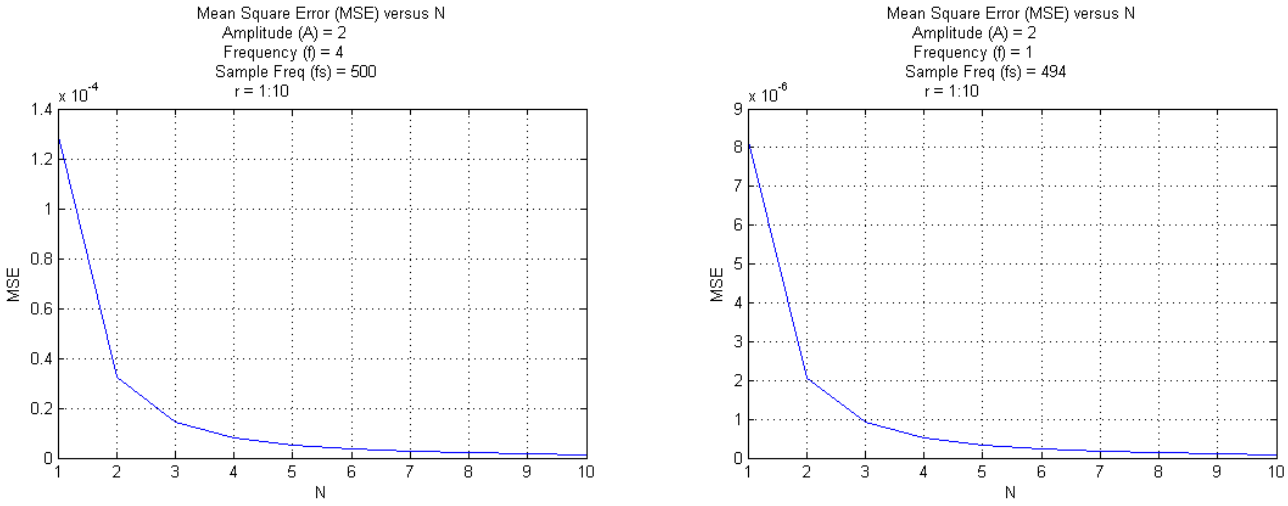


Figure 6 shows three plots of the sine wave's autocorrelation. The red plot is simply the autocorrelation computed from the derived expression for the generic sine wave's autocorrelation, as shown in Equation 8. The blue and green plots are two different versions of the autocorrelation generated from data. The difference is in how the second copy of the data is shifted prior the dot product operation between the data and its shifted copy. The blue plot is the autocorrelation computed from MATLAB's `xcorr`, in which zeros are simply appended to tails of both the data and its copy when there is a phase difference between the data and its copy. The green plot is the

autocorrelation generated from the user-defined function as described in the second paragraph below Figure 3.

As Figure 6 demonstrates, the circular shift operation causes the autocorrelation operation to result in an experimental autocorrelation very similar to the theoretical autocorrelation. The circular shift operation is essentially replicating the chosen window of the sine wave. In other words, the circular shift operation would not have worked so well if the window size N was chosen as a value other than a multiple of the generic sine wave's period, or $\frac{f}{f_s}$. As expected, the autocorrelation generated from `xcorr` linearly diminishes to zero the farther the lag is from 0. If an infinite data set is possible the autocorrelation from `xcorr` would, however, resemble the red and green plots.

Figure 7: Mean Square Error of sine wave's covariance matrix $C_{i,j}$



The last two plots shown in Figure 7 show how the covariance matrices' MSE, calculated in the same manner as the MSE calculated for the white noise, changes with respect to a growing window size N. The window size N, in the case of the generic sine wave, changes with respect to the positive integer f, as $N = \frac{f}{f_s} \times r$ and the theoretical expression for the generic sine wave's covariance matrix (Equation 9) is only valid for positive integers of r.

3 Problem 3

3.1 Derived Results

3.1.1 Discrete Filter: $H(z) = 1 + cz^{-1}$

$$H(z) = 1 + cz^{-1} \tag{11}$$

An expression is determined for calculating the constant c for a filter whose transfer function is Equation 11, input $x[n]$ is normally distributed with a mean of 0, and output data $\tilde{y}[n]$ is known.

$$y[n] = x[n] + cx[n-1] \tag{12}$$

The transfer function shown in Equation 11 is first transformed from the z-domain back to the time domain with the inverse z-transform. The result of the inverse transformation is shown in

Equation 12.

$$\begin{aligned}\text{Mean Square Error (MSE)} &= E[(y[n]-\tilde{y}[n])^2] \\ E[\bullet] &= \frac{1}{N}\sum_{n=0}^{N-1}\bullet_n\end{aligned} \tag{13}$$

In order to determine from the data a potential value for the constant c, an expression that contains both the output data $\tilde{y}[n]$ and the input data $x[n]$ is required. A simple expression that relates the data and the filter's constant c is the Mean Square Error (MSE) (which was previously used in Problem 2 to determine the error between theoretically and experimentally determined covariance matrices). In the context of Problem 3, the expression for determining the MSE is shown in Equation 13. $y[n]$ represents the theoretical results determined from the difference equation shown in Equation 12, whereas $\tilde{y}[n]$ represents the actual output data.

$$\begin{aligned}\text{MSE} &= E[(y[n]-\tilde{y}[n])^2] \\ &= E[((x[n]+cx[n-1])-\tilde{y}[n])^2] \\ &= E\left[x^2[n]+2cx[n]x[n-1]+cx^2[n-1]-2x[n]\tilde{y}[n]-2cx[n-1]\tilde{y}[n]+\tilde{y}^2[n]\right]\end{aligned} \tag{14}$$

Once Equation 12 is substituted in Equation 13 and then expanded, the MSE results in Equation 14.

$$\begin{aligned}\frac{d\,(\text{MSE})}{dc} &= \frac{d\left(E\left[x^2[n]+2cx[n]x[n-1]+cx^2[n-1]-2x[n]\tilde{y}[n]-2cx[n-1]\tilde{y}[n]+\tilde{y}^2[n]\right]\right)}{dc} \\ &= E[2x[n]x[n-1]+2x^2[n-1]c-2x[n-1]\tilde{y}[n]]\end{aligned} \tag{15}$$

Equation 14 is then differentiated with respect to the constant c so as to minimize the MSE. Similar to Maximum Likelihood Estimation (MLE) where an unknown parameter of an assumed distribution is chosen if its maximizes the likelihood a particular data set was produced from the distribution, the constant or parameter c is chosen if it minimizes the MSE; MLE and the minimization of the error (MSE) are similar in methodology, but fundamentally different.

$$\begin{aligned}0 &= E[2x[n]x[n-1]+2x^2[n-1]c-2x[n-1]\tilde{y}[n]] \\ 0 &= E[x[n]x[n-1]]+E[x^2[n-1]]c-E[x[n-1]\tilde{y}[n]] \\ 0 &= R_{xx}[1]+R_{xx}[0]c-R_{yx}[1]\end{aligned}$$

The following is possible since it is known the input is a normal distribution whose mean μ is 0

$$\begin{aligned}0 &= \sigma_x^2 c - R_{yx}[1] \\ c &= \frac{R_{yx}[1]}{\sigma_x^2}\end{aligned} \tag{16}$$

After Equation 15 is set equal to 0, the expression for determining c is finally derived in Equation 16.

3.1.2 Discrete Filter: $H(z) = 1 + c_1 z^{-1} + c_2 z^{-2}$

$$H(z) = 1 + c_1 z^{-1} + c_2 z^{-2} \tag{17}$$

The steps to determine the constants c_1 and c_2 of the new digital filter shown in Equation 17 through the minimization of the Mean Square Error (MSE) are nearly the same as the steps described in Section 3.1.1.

$$y[n] = x[n] + c_1 x[n-1] + c_2 x[n-2] \tag{18}$$

First, the difference equation that calculates the theoretical output data $y[n]$ based on the input $x[n]$ and the constants $c_{1 \text{ and } 2}$ is acquired after performing the inverse z-transform on the filter's transfer function. The new difference equation is shown in Equation 18.

$$\begin{aligned} \text{MSE} \; &= E[[\alpha + 2c_1 x[n-1]x[n] + c_1^2 x^2[n-1] + \\ &\quad 2c_2 x[n-2]x[n] + 2c_1 c_2 x[n-2]x[n-1] + c_2^2 x^2[n-2] \\ &\quad - 2c_1 x[n-1]\tilde{y}[n] - 2c_2 x[n-2]\tilde{y}[n]] \end{aligned} \tag{19}$$

The difference equation in Equation 18 is then substituted in Equation 13 to result in Equation 19.

$$c_1 = \frac{R_{yx}[1]}{\sigma_x^2} \tag{20}$$

$$c_2 = \frac{R_{yx}[2]}{\sigma_x^2} \tag{21}$$

Because there are now two constants for which need to be solved, partial differentiation is applied with respect to the two constants. After simplification, the equations nicely reduce to Equations 20 and 21.

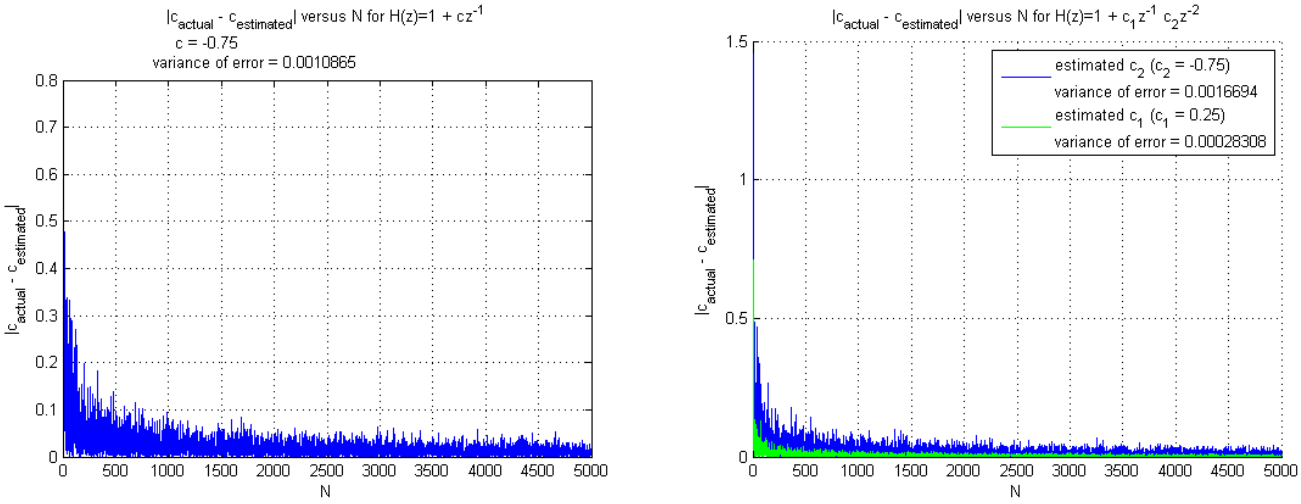
In order to determine the constant c with Bayesian Estimation, a parameterized distribution is required to represent c, which is now considered a random variable.

3.2 Experimental Results

$$e_c = |c_{actual} - c_{estimated}| \tag{22}$$

Equation 22 shows how the error between the actual and the estimated constants are calculated for the experimental results. The estimated constants are calculated from Equations 16, 20, and 21. For each window size N ranging from 3 to 5×10^3, N data points are generated from a normal distribution whose mean μ_x is 0 and variance σ_x^2 is 4 (variance selected arbitrarily). The input data points are passed through each of the two discrete filters to result in two sets of output data. The transfer functions of the two discrete filters respectively correspond to Equations 11 and 17.

Figure 8: Absolute error $e_c$versus window size N



The results are finally presented in Figure 8. The variance of the data produced for the absolute errors e_c is also included to further show how each data set differs from each other. It's difficult to accurately tell how much different the errors e_c for the constants c are from one another—even after running a few trials of the MATLAB simulation. The MATLAB simulation begins to take way too long to run if the window size N is raised too high. However, based on the number of trials and observing primarily the variance of the error e_c, the error e_c of the constant c, referring to the left plot of Figure 8, varies less than the error e_c of constant c_1, referring to the right plot of Figure 8. The estimated constant c_2 consistently appears to vary less than both the constants c and c_1.

4 Problem 4

$$H(z) = 1 - a_n z^{-1} \tag{23}$$

For Problem 4, the first discrete filter of Problem 3 is presented again, however with a constant a that varies as a function of time. Equation 23 shows the discrete filter of importance for Problem 4.

$$\begin{aligned} y[n] &= x[n] - a_n x[n-1] \\ a_n &= a_{actual} + 0.24 \sin\left(2\pi \frac{n}{100}\right) \\ a_{actual} &= 0.75 \end{aligned} \tag{24}$$

Once again, the goal of this problem is to estimate the constant a and then compare the estimated constant $a_{estimated}$ with the actual constant a_{actual}. Equation 24 shows the discrete filter after which the inverse z-transform is applied and how the constant a changes with respect to time measured in samples n. a_n represents what the constant is at a particular instance of time, whereas a_{actual} is the actual constant.

$$\tilde{x}[n] = x[n] \in N(\mu_x, \sigma_x^2) \text{ where } \mu_x = 0 \tag{25}$$

The input $x[n]$ is once again randomly generated as a Gaussian distribution, as shown in Equation 25.

$$m \in \{1N, 2N, 3N, ...\} \tag{26}$$

As in Problem 3, the comparison between the constants a_{actual} and $a_{estimated}$ is done by calculating, and then plotting, the error $e_c$between the actual and estimated constants as a function of the window size N. However, in the case of Problem 4, e_c is also calculated with respect to a series of N-sized, adjacent windows as shown in Equation 26.

$$\tilde{x}[n+m] \sim \tilde{x}[n] \tag{27}$$

$$\begin{aligned} y[n+m] &= x[(n+m)] - a_{n+m}x[(n+m)-1] \\ &= \tilde{x}[(n+m)] - a_{n+m}\tilde{x}[(n+m)-1] \\ &\sim \tilde{x}[n] - a_{n+m}\tilde{x}[n-1] \\ a_{n+m} &= a_{actual} + 0.24\sin\left(2\pi\frac{(n+m)}{100}\right) \\ a_{actual} &= 0.75 \end{aligned} \tag{28}$$

In order to simplify the MATLAB source code developed for Problem 4, approximations are determined for the input $x[n]$ and the discrete filter system $y[n]$. Because m from Equation 26 only shifts the input by equally sized, adjacent windows and the input is normally distributed, a close approximation of $\tilde{x}[n+m]$ is $x[n]$ (Equation 27). A close approximation for the discrete filter's difference equation with respect to m is shown in Equation 28. The close approximations made in Equations 27 and 28 are really windows of the input repeated every window size N. Thus, the larger the window size N is, the closer the approximations are to the true $y[n+m]$ and $\tilde{x}[n+m]$.

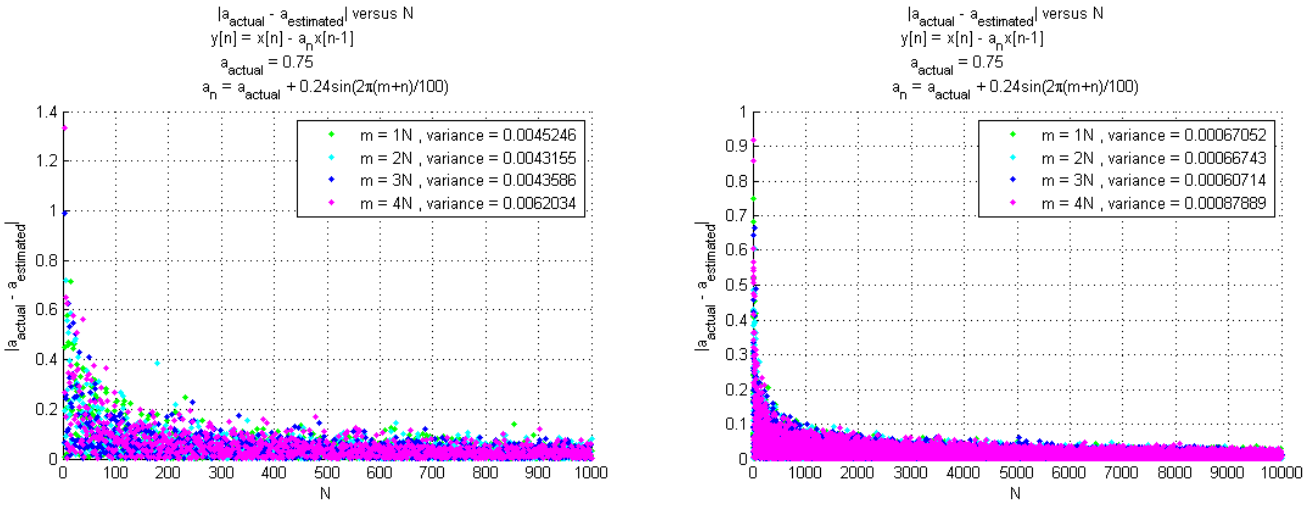
Since the same assumptions made about the input $x[n]$ cannot be drawn for the sinusoidal term of the time-varying constant a_n, $n+m$ is substituted into a_n as shown in Equation 28.

$$\begin{aligned} R_{\tilde{x}\tilde{x}}[k] &= \frac{1}{N}\sum_{n=0}^{N-1}\tilde{x}[m+n]\tilde{x}[m+n-k] \\ &\sim \frac{1}{N}\sum_{n=0}^{N-1}\tilde{x}[n]\tilde{x}[n-k] \end{aligned} \tag{29}$$

The shift m does not cause the gaussian distribution to change!

The equation for determining the constant $a_{estimated}$ is Equation 16 from Problem 3. Figure 29 shows the results generated from a MATLAB source. Judging from the plot, the change in m does not appear to cause much variation in the error e_c. As expected, an increase in the window size N causes the error e_c to drop, though the rate at which the error drops abruptly appears to nearly reach zero as the window size N goes to infinity. Depending on the constraints of the application, it might be more practical to select a relatively lower window size N—let's say approximately within the range of $3{\times}10^3$ and $6{\times}10^3$—than a larger window size, seeing as a larger N results in little benefit.

Figure 9



5 Problem 5

$$x[n] = a^n u[n] \tag{30}$$

$$\begin{aligned} C_{i,j} &= E[(x_{n-i} - \mu_{x_{n-i}})(x_{n-j} - \mu_{x_{n-j}})] \\ &= \frac{1}{N} \sum_{n=n_{start}}^{N-1} (x[n-i] - \mu_{x_{n-i}})(x[n-j] - \mu_{x_{n-j}}) \\ \mu_{x_{n-m}} &= E[x_{n-m}] \\ &= \frac{1}{N} \sum_{n=n_{start}}^{N-1} x[n-m] \end{aligned} \tag{31}$$

The goal of Problem 5 is to generate a set of covariance matrices $C_{j,i}$ for $x[n]$ shown in Equation 30. The covariance matrix $C_{i,j}$ is once again normalized. Since the mean μ_x of the data is not zero, the expression defining the covariance matrix in Equation 4 is instead replaced with the expression shown in Equation 31. The constants a from Equation 30 and the n_{start} from Equation 31 are varied for each covariance matrix $C_{i,j}$. Specifically, a and n_{start} are constants from the sets $\{0.99, 1.01\}$ and $\{0, 100, 1000\}$, respectively.

Equations 32 to 37 are the results generated for Problem 5.

$$\begin{aligned} n_{start} &= 0 \\ N &= 1000 \\ a &= 0.99 \\ C_{dimension=2} &= \begin{bmatrix} 0.040292 & 0.039789 \\ 0.039789 & 0.040292 \end{bmatrix} \end{aligned} \tag{32}$$

$$\begin{aligned} n_{start} &= 100 \\ N &= 1000 \\ a &= 0.99 \\ C_{dimension=2} &= \begin{bmatrix} 0.0053984 & 0.0054529 \\ 0.0054529 & 0.005508 \end{bmatrix} \end{aligned} \tag{33}$$

$$\begin{aligned} n_{start} &= 1000 \\ N &= 1000 \\ a &= 0.99 \\ C_{dimension=2} &= \begin{bmatrix} 7.5095 \cdot 10^{-11} & 7.5854 \cdot 10^{-11} \\ 7.5854 \cdot 10^{-11} & 7.662 \cdot 10^{-11} \end{bmatrix} \end{aligned} \tag{34}$$

The covariance matrices $C_{i,j}$ shown in Equations 32 to 34 are generated with the constant $a = .99$, a value less than 1. This observation is of importance because the product of a set of values, all of which are less than 1, will invariably result in a value less than all the values of the original set. Seeing as the operation computed to determine the covariance matrix $C_{i,j}$, can be considered a dot product for which each term is a product of two values less than 1, the covariance matrix $C_{i,j}$ of $x[n]$, where $a < 1$, will consist of values going to 0 as n_{start} goes to infinity. A larger window size N only lessens the rate at which the covariance matrix's values go to 0. An increase in the covariance matrix's dimension

The fact the elements of the covariance matrix $C_{i,j}$ appear to converge to zero as n_{start} goes to infinity implies the signal $x[n]$ is stable.

$$\begin{aligned} n_{start} &= 0 \\ N &= 1000 \\ a &= 1.01 \\ C_{dimension=2} &= \begin{bmatrix} 1.748 \cdot 10^{7} & 1.7307 \cdot 10^{7} \\ 1.7307 \cdot 10^{7} & 1.7136 \cdot 10^{7} \end{bmatrix} \end{aligned} \tag{35}$$

$$\begin{aligned} n_{start} &= 100 \\ N &= 1000 \\ a &= 1.01 \\ C_{dimension=2} &= \begin{bmatrix} 1.2788 \cdot 10^{8} & 1.2662 \cdot 10^{8} \\ 1.2662 \cdot 10^{8} & 1.2536 \cdot 10^{8} \end{bmatrix} \end{aligned} \tag{36}$$

$$\begin{aligned} n_{start} &= 1000 \\ N &= 1000 \\ a &= 1.01 \\ C_{dimension=2} &= \begin{bmatrix} 7.6788 \cdot 10^{15} & 7.6027 \cdot 10^{15} \\ 7.6027 \cdot 10^{15} & 7.5275 \cdot 10^{15} \end{bmatrix} \end{aligned} \tag{37}$$

The covariance matrices shown in Equations 35 to 37 reflect the case for which the constant a of $x[n]$ is greater than 1. Because the operation calculated to obtain each value of the covariance

matrix $C_{i,j}$ is essentially a sum of a terms, each of which are products of two values greater than 1, the values of the covariance matrix $C_{i,j}$ will undoubtedly increase to infinity.

Another interesting observation is the rank of all the covariance matrices $C_{i,j}$ appears to equal 1, judging from the data presented in Equations 32 to 37. The reason for the rank of 1 could be $x[n]$ has only 1 parameter a.

6 Appendix: MATLAB source code

```
function Homework3Script
close all;

% Problem1;
% Problem2;
% Problem3;
% Problem4;
% Problem5;

end

function Problem1

mean = 0;
variance = 1;
lengths = 1e2:1e2:1e4;

MLEmeans = zeros(1, numel(lengths));
MLEvariances = zeros(1, numel(lengths));

for length = lengths
    X = generateGrv(mean, variance, length);
    [MLEmean, MLEvariance] = getMaximumLikelihoodEstimations(X);
    MLEmeans(length == lengths) = MLEmean;
    MLEvariances(length == lengths) = MLEvariance;
end

figure;
hold on
plot(lengths, MLEmeans, 'b');
plot(lengths, repmat(mean, numel(lengths)), 'k');
legend('MLE mean', 'True mean');
title('Mean versus data length');
xlabel('Lengths');
ylabel('Means');
hold off


figure;
hold on
plot(lengths, MLEvariances, 'b');
plot(lengths, repmat(variance, numel(lengths)), 'k');
legend('MLE variance', 'True variance');
title('Variance versus data length');
xlabel('lengths');
ylabel('Variances');
hold off

end

function Problem2
%% gaussian stuff
% set base parameters for grv
variance = 3;
N = 1e3;
n = 0:N-1;
y = generateGrv(0, variance, N);

% plot the gaussian noise
figure;
plot(n, y);
```

```
title(char('Gaussian', ...
    ['Variance (sigma^2) = ' num2str(variance)], ...
    ['N = ' num2str(N)]));
ylabel('x[n]');
xlabel('n');
grid on

% determine the theoretical and experimental autocorrelations for a
% specific value of N
[Rex, k] = getAutocorrelationCircularShift(y);
Rth = [zeros(1, numel(y)-1) variance zeros(1, numel(y)-1)];

% plot stem plt
figure;
hold on
stem(k, Rex, 'b', 'LineWidth', 3);
stem(k, Rth, 'g', 'LineWidth', 2);
title(char('Autocorrelation', ...
    ['Variance (sigma^2) = ' num2str(variance)], ...
    ['N = ' num2str(N)]));
ylabel('Magnitude');
xlabel('lags');
legend(char('Experimental Results', 'with user-defined autocorrelation'), ...
    'Theoretical Results');
grid on
hold off

% Mean Square Error variables
Ns = 1:250;
errors = zeros(1, numel(Ns));

% get the Mean Square Errors for noise
for N=Ns

    % set parameters
    y = generateGrv(0, variance, N);

    % get theoretical result
    CmTh = variance*eye(N);

    % get result from simulation
    X = zeros(numel(y),N);
    for index=1:N, X(:,index)=circshift(y',index-1);end;
    CmEx = cov(X);

    % get error
    errors(N==Ns) = getError(CmEx, CmTh);

end

% plot how the Mean Square Error (MSE) changes with respect to N
figure;
plot(Ns, errors);
title(char('Mean Square Error (MSE) versus N', ...
    ['Variance (sigma^2) = ' num2str(variance)], ...
    ['N = ' num2str(N)]));
ylabel('MSE');
xlabel('N');
grid on

%% sine stuff
% set base parameters of sine wave
f = 4;
```

```
fs = 2*f+492;
r = 4;
N = fs/f*r;
A = 2;
n = 0:N-1;

getSineWave = @(A,f,fs,n)A*sin(2*pi*f/fs*n);
y = getSineWave(A,f,fs,n);

% generate different versions of the autocorrelation
[Rex, k] = xcorr(y, 'biased'); % baised means R is scaled by 1/numel(y)
Rex2 = getAutocorrelationCircularShift(y);
Rth = A^2/2*cos(2*pi*f/fs*k);

% plot original sine wave
figure;
plot(n, y);
title(char('Sine Wave', ...
    ['Amplitude (A) = ' num2str(A)], ...
    ['Frequency (f) = ' num2str(f)], ...
    ['Sample Freq (fs) = ' num2str(fs)], ...
    'N = fs/f*r', ...
    ['r = ' num2str(r)]));
ylabel('x[n]');
xlabel('n');
grid on

% plot autocorrelations
figure
hold on
plot(k, Rex, 'b', 'LineWidth', 7);
plot(k, Rex2, 'g', 'LineWidth', 5);
plot(k, Rth, 'r', 'LineWidth', 2);
grid on
title(char('Autocorrelation plots', ...
    ['Amplitude (A) = ' num2str(A)], ...
    ['Frequency (f) = ' num2str(f)], ...
    ['Sample Freq (fs) = ' num2str(fs)], ...
    'N = fs/f*r', ...
    ['r = ' num2str(r)]));
ylabel('Magnitude');
xlabel('lag');
legend('Experimental Results with ''xcorr'''', ...
    char('Experimental Results', 'with user-defined autocorrelation'), ...
    'Theoretical Results');
hold off

% defined base parameters for generating the error between the
% theory-determined covariance matrices and the experimental convariance
% matrices
C=@(i,j)A^2/2*cos(2*pi*f/fs*(j-i));
%Ns = 1:500;
rs = 1:10;
errors = zeros(1, numel(rs));

% get the average percent errors for the sine waves
for r=rs

    % generate N
    N = fs/f*r;
    n = 0:N-1;

    % set parameters
```

```
    y = getSineWave(A,f,fs,n);

    % get theoretical result
    CmTh = zeros(N, N);
    for i=0:N-1, CmTh(i+1,:)=C(i,0:N-1); end;

    % get result from simulation
    X = zeros(numel(y),N);
    for index=1:N, X(:,index)=circshift(y',index-1);end;
    CmEx = cov(X);

    % get error
    errors(r==rs) = getError(CmEx, CmTh);

end

% plot how the Mean Square Error (MSE) changes with respect to N
figure
plot(rs, errors);
title(char('Mean Square Error (MSE) versus N', ...
    ['Amplitude (A) = ' num2str(A)], ...
    ['Frequency (f) = ' num2str(f)], ...
    ['Sample Freq (fs) = ' num2str(fs)], ...
    ['r = 1:' num2str(rs(end))]));
ylabel('MSE');
xlabel('N');
grid on

end

function Problem3

% set up the constants
c11 = -.75;
c21 = .25; c22 = -.75;

% set up filter polynomials (transfer function H(z))
filterTransferFunction1 = {[1 c11], 1};
filterTransferFunction2 = {[1 c21 c22], 1};

% set base parameters for grv
variance = 4;
mean = 0;
Ns = 3:5e3;
error1 = zeros(1, numel(Ns));
error2 = zeros(2, numel(Ns));

for N=Ns
    x = generateGrv(mean, variance, N);
    y1 = filter(filterTransferFunction1{1}, filterTransferFunction1{2}, x);
    y2 = filter(filterTransferFunction2{1}, filterTransferFunction2{2}, x);
    estimatedC11 = getEstimatedC(1, x, y1);
    estimatedC21 = getEstimatedC(1, x, y2);
    estimatedC22 = getEstimatedC(2, x, y2);
    error1(N==Ns) = abs(estimatedC11-c11);
    error2(1, N==Ns) = abs(estimatedC21-c21);
    error2(2, N==Ns) = abs(estimatedC22-c22);
end

figure;
plot(Ns, error1);
grid on
title(char('|c_{actual} - c_{estimated}| versus N for H(z)=1 + cz^{-1}', ...
```

```
    ['c = ' num2str(c11)], ...
    ['variance of error = ' num2str(var(error1))]))
ylabel('|c_{actual} - c_{estimated}|')
xlabel('N')

figure;
hold on
plot(Ns, error2(2,:), 'b');
plot(Ns, error2(1,:), 'g');
grid on
legend(char(['estimated c_2 (c_2 = ' num2str(c22) ')'], ...
    ['variance of error = ' num2str(var(error2(2,:)))]), ...
    char(['estimated c_1 (c_1 = ' num2str(c21) ')'], ...
    ['variance of error = ' num2str(var(error2(1,:)))]));
title('|c_{actual} - c_{estimated}| versus N for H(z)=1 + c_1z^{-1} c_2z^{-2}')
ylabel('|c_{actual} - c_{estimated}|')
xlabel('N')
hold off

end

function Problem4

% a = 0.75 + 0.24sin(2pi(1)n/100)
aActual = .75;
getA = @(nOffset, N)aActual + 0.24*sin(2*pi*(1)*(nOffset+[0:(N-1)])/100);

% H(z) = 1-az^{-1}
getFilteredOutput = @(x, a)x-a.*[0 x(1:end-1)];

% determine estimated a
getEstimatedA = @(x, y)-getEstimatedC(1, x, y);

ms = 1:4;
Ns = 3:1e3;
meanw = 0;
variance = 4;
errors = zeros(numel(ms), numel(Ns));

for m = ms
    for N = Ns
        x = generateGrv(meanw, variance, N);
        y = getFilteredOutput(x, getA(m*N, N));
        aEstimated = getEstimatedA(x, y);
        errors(m==ms, N==Ns) = abs(aActual-aEstimated);
    end
end

figure;
hold on
colorSet = varycolor(numel(ms));
legendSet = {};
for m = ms
    plot(Ns, errors(m==ms, :), '.', 'Color', colorSet(m==ms,:));
    legendSet{end+1} = ['m = ' num2str(m) 'N , variance = ' num2str(var(errors(m==ms, :)))];
end
legend(legendSet);
title(char('|a_{actual} - a_{estimated}| versus N', ...
    'y[n] = x[n] - a_nx[n-1]', ...,
    ['a_{actual} = ' num2str(aActual)], ...
    'a_n = a_{actual} + 0.24sin(2\pi(m+n)/100)'));
ylabel('|a_{actual} - a_{estimated}|')
xlabel('N')
```

```
grid on
hold off

end

function Problem5

getX = @(a, n)a.^n.*(n >= 0);
N = 10e3;
nstarts = [0, 100, 1000];
as = [0.99, 1.01];
dimension = 100;

for a=as
    for nstart=nstarts
        C = getCovarianceMatrix(@(n)getX(a, n+nstart), dimension, N);
        disp(char(['nstart = ' num2str(nstart)], ...
            ['a = ' num2str(a)], ...
            convertMatrixToLatexWithPrecision(C, 5), ...
            ' '));
    end
end


end

function error = getError(ex, th)

error = mean(mean((ex-th).^2));

end

function X = generateGrv(mean, covariance, length)

X = mvnrnd(mean', covariance, length)';

end

function [mean, covariance] = getMaximumLikelihoodEstimations(X)

    mean = 1/numel(X(1,:))*sum(X,2);
    covariance = 0;
    for n =1:numel(X(1,:))
        covariance = covariance + (X(:,n)-mean)*(X(:,n)-mean)';
    end
    covariance = covariance/numel(X(1,:));

end

function [R, l] = getAutocorrelationCircularShift(y)

l = -(numel(y)-1):numel(y)-1;
R = numel(l);
for n=1:numel(l), R(n) = 1/numel(y)*sum(y'.*circshift(y', n)); end

end

function [R, l] = getCrosscorrelationCircularShift(x,y)

if numel(x) ~= numel(y), error('x and y must have the same amount of elements'); end
l = -(numel(y)-1):numel(y)-1;
R = numel(l);
for n=1:numel(l), R(n) = 1/numel(y)*sum(x'.*circshift(y', n)); end
```

```
378 end

380 function c = getEstimatedC(l, x, y)

382 [Ryx, lag] = xcorr(y, x, 'unbiased');
    c = Ryx(lag == l)/var(x);
384

    end
386

    function C = getCovarianceMatrix(xFunctionHandle, dimension, N)
388

    [~, d] = rat(dimension);
390 if d ~= 1, error('dimension cannot have a decimal'); end
    if dimension < 1, error('dimension should be greater than 0'), end
392

    X = zeros(N, dimension);
394 for i=1:dimension;
        X(:, i)=xFunctionHandle([0:N-1]-(i-1));
396 end
    C = cov(X);
398

    end
400

    function latexMatrix = convertMatrixToLatexWithPrecision(M,p)
402

    d = digits(p);
404 latexMatrix = latex(sym(vpa(M)));
    digits(d);
406

    end
```

Listing 1: MATLAB source