Devin Trejo

ECE 3522: Stochastic Processes in Signals and Systems

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

I. PROBLEM STATEMENT

Chapter 10 of Fundamentals of Applied Probability and Random Processes by Oliver-C-Ibe introduces applied concepts for stochastic processes. A stochastic process is one where the output contains a component of randomness. In this assignment we combine knowledge from Signals and Systems with Stochastic Systems and observe what trends we can conclude.

To begin we will create a function that will provide us a sine wave with a component of noise. In the previous assignment we demonstrated how we could create a function using the Box-Muller technique to create an array of normally distributed variables. We now use that function to create a Gaussian noise aspect and combine it with our signal. The amount of noise we combine with our sine wave is dependent on the signal to noise ratio. We can define the SNR as follows:

$$SNR_{db} = 20\log\left(\frac{Sine_{Power}}{Noise_{Power}}\right) \tag{1}$$

In our function we will be able to take in the desired output frequency in hertz, the length (in seconds), the sample frequency, and SNR.

Now that we are capable of creating a signal with noise we observe the relationship of the autocorrelation to the SNR. Autocorrelation tells us how correlated a signal is to itself at different shift values (τ). In our case we will take the first 16 shifts of a 500Hz sine wave sampled at 8000Hz. Inspecting the autocorrelation at different shifts will tells us at SNR does our noise impact our sine wave considerably. Also at these various SNRs we inspect the Fourier transform of the signal. We are curious to observe what noise looks like in the frequency domain.

Lastly, we explore the idea of the power spectral density function. The PSD is the Fourier transform of the autocorrelation function and tells us the power at certain frequencies of our signal. To demonstrate the concept we create a signal with 30db SNR. Passing the signal through a digital filter shown below provides us with a clearer output. We find the PSD and plot the Fourier transform of our filtered signal squared. We expect the resulting plots should be equal.

$$y[n] = 0.5y[n-1] + x[n]$$

II. APPROACH AND RESULTS

To begin we create a function that produces a sine wave with noise given a certain SNR. Using the Box-Muller method allows us to quickly create an array of normal variables, but we still need to find the variance of the noise. Recall that the variance also equals the power of a signal. Using equation 1 we can find the power of the noise. Then we can feed it into our normal number generator to create a noise signal. We create a table of the power in the entire signal given different SNR.

Table 1: Power at various SNR

SNR (dB)	Power (W)
-30	0.1
-20	0.1
-10	0.1
0.0	0.1
10	0.1
20	0.2
30	0.3

From the table we observe how if we increase the noise in the signal the less power we have. Our sine wave has an amplitude of ±1. Ideally for a pure sine wave the power is as follows:

$$Sine_{Power} = \frac{|A|^2}{2} = \frac{1^2}{2} = \frac{1}{2}W$$

In the ideal case our power in our noisy signal would be 0.5W. As we have a greater signal to noise ratio our power in he signal also increases closer to 1. Comparing the sine wave form for -30dB to 30db we observe the impact of noise. In the -30dB noisy signal you are not really able to realize the sine wave. In the 30dB signal you can clearly see the sine wave (see plots below).

Now we wish to observe the autocorrelation of our signal given various SNRs for the first 16 shifts. We also plot the Fourier transform of the signal to note the frequency response.

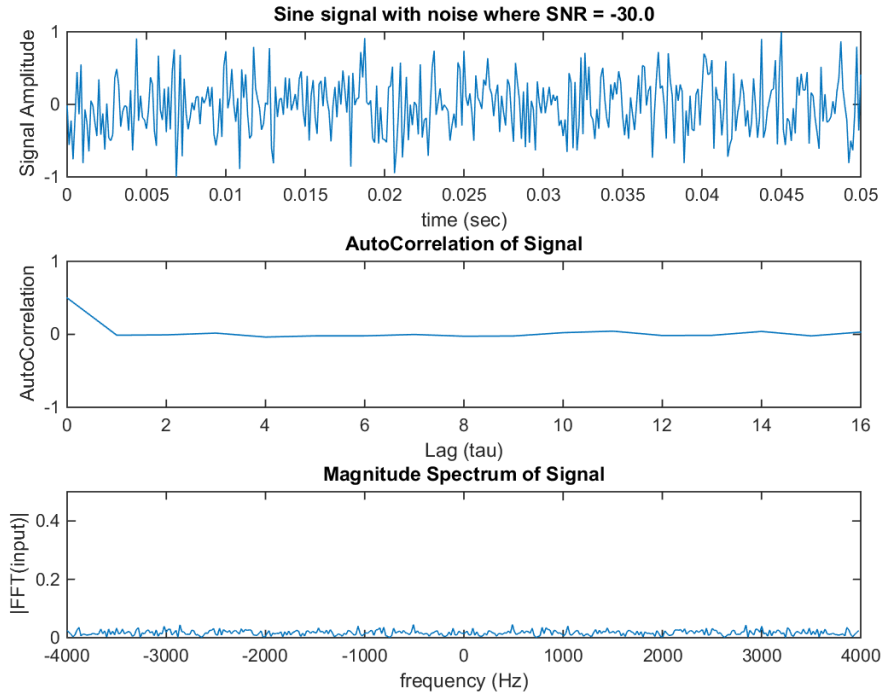

Figure 1: Sine with noise signal where SNR = -30db. ACF of Signal w/ Noise, and FFT of Signal w/ Noise

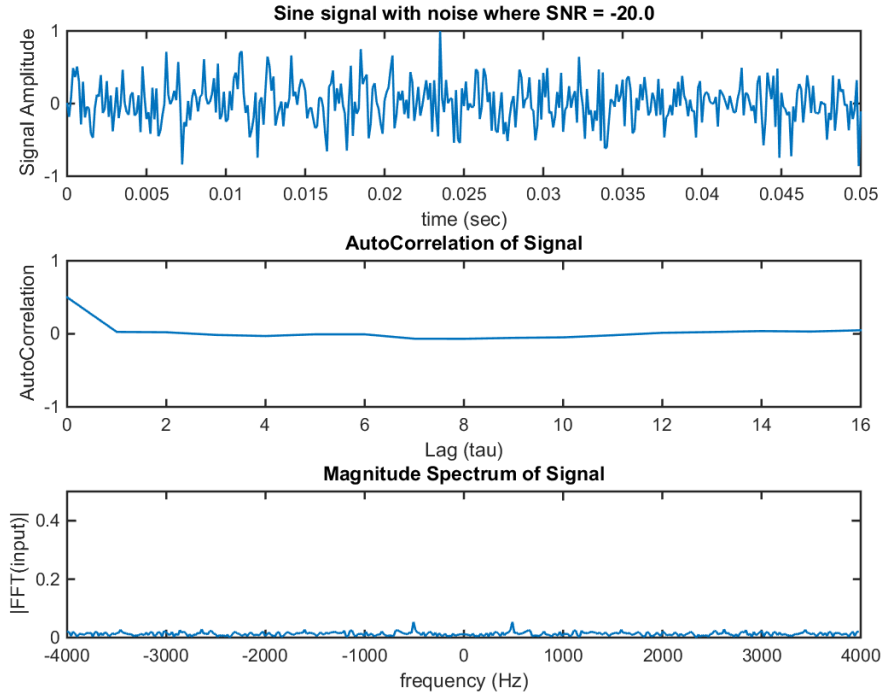

Figure 2: Sine with noise signal where SNR = -20db. ACF of Signal w/ Noise, and FFT of Signal w/ Noise

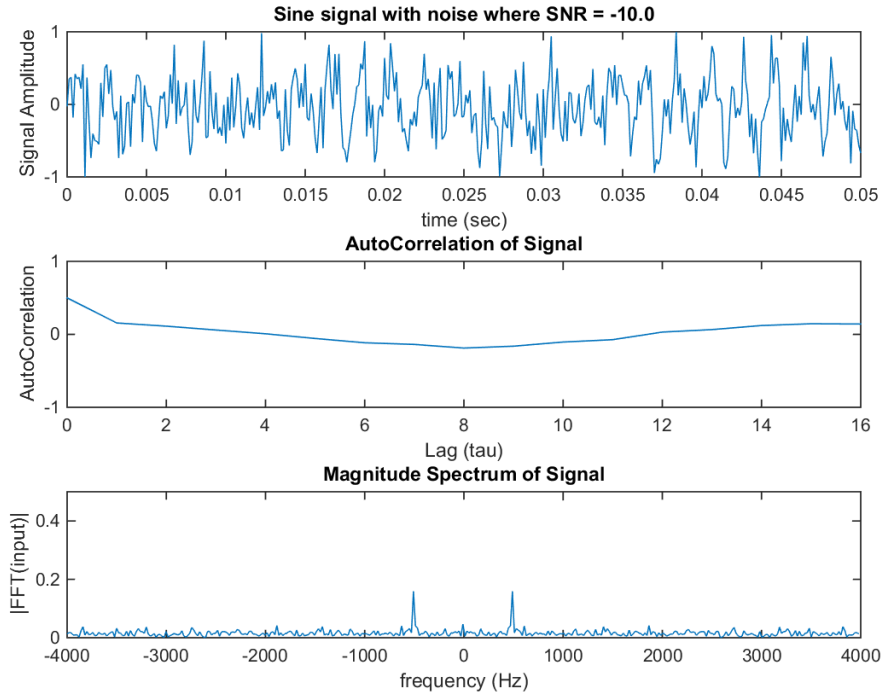

Figure 3: Sine with noise signal where SNR = -10db. ACF of Signal w/ Noise, and FFT of Signal w/ Noise

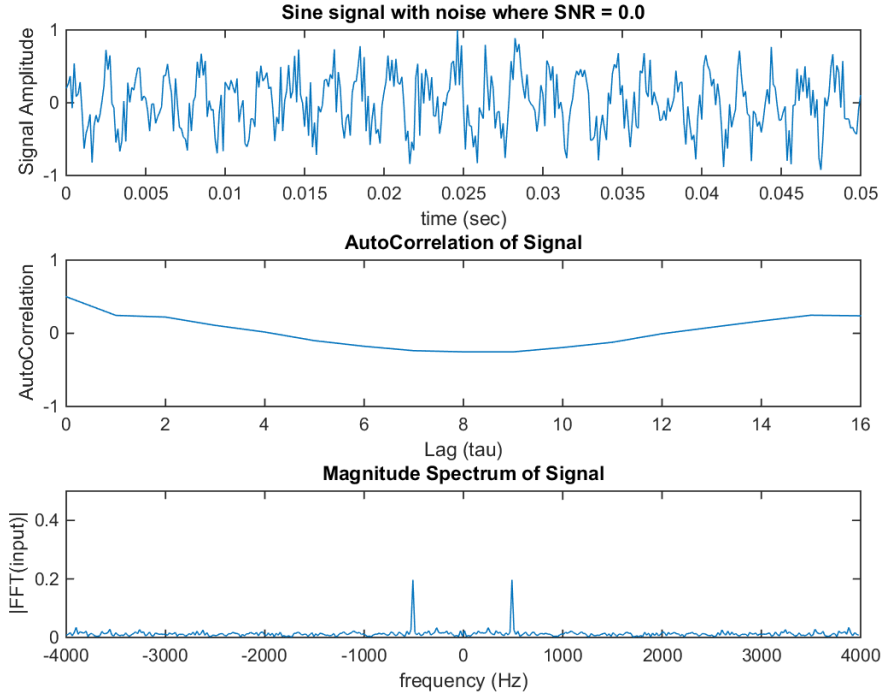

Figure 4: Sine with noise signal where SNR = 0.0db. ACF of Signal w/ Noise, and FFT of Signal w/ Noise

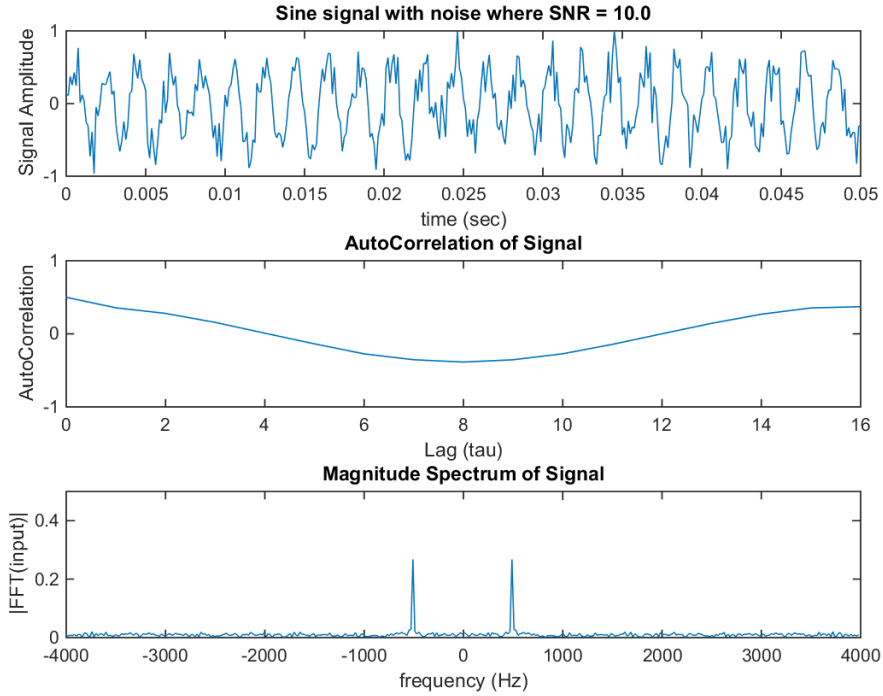

Figure 5: Sine with noise signal where SNR = 10db. ACF of Signal w/ Noise, and FFT of Signal w/ Noise

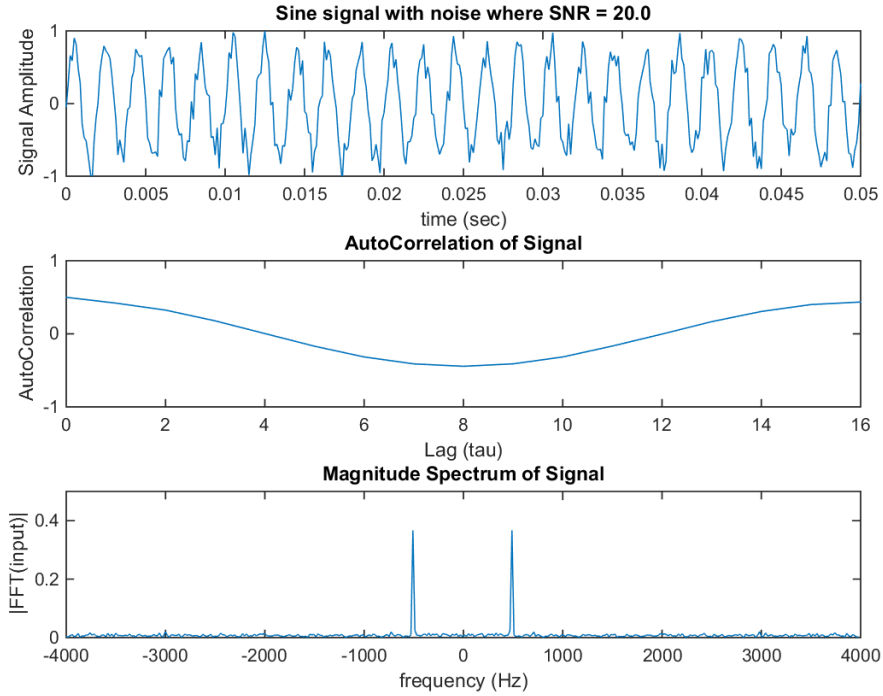

Figure 6: Sine with noise signal where SNR = 20db. ACF of Signal w/ Noise, and FFT of Signal w/ Noise

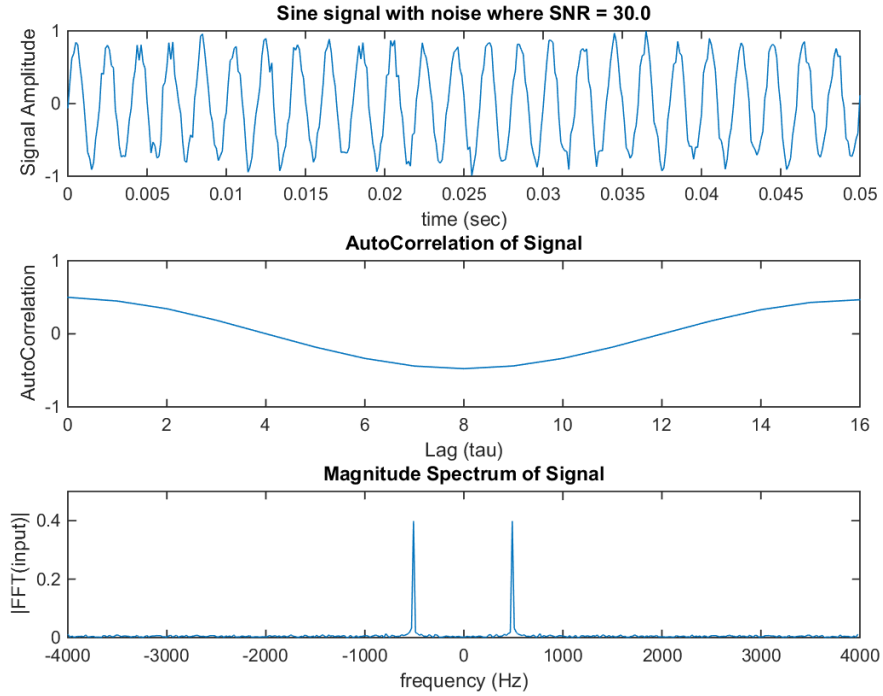

Figure 7: Sine with noise signal where SNR = 30db. ACF of Signal w/ Noise, and FFT of Signal w/ Noise

The plots above showcase the SNRs effect on the signal itself. For a pure sine wave we expect an auto-correlation of a cosine that is time invariant:

$$x = sin(\omega t) \rightarrow R_{xx} = \frac{1}{2}\cos(\omega \tau)$$

However given noise in the signal the autocorrelation function will approach zero since the random noise takes over the signal. In the beginning we had a SNR of -30dB which have no correlation unless τ=0. As we slowly increased the SNR we obtained more a pure sinewave form thus our autocorrelation approached the expected cosine. At a SNR of 0dB (equal noise to signal), our autocorrelation starts to transform into the cosine.

In the frequency domain we see how when there is a lot of noise we have a relatively uniform frequency domain. Since Gaussian noise contains uniform power across all frequencies this characteristic is to be expected. As we increased our SNR our fundamental frequency starts to emerge. Since we are feeding our sineNoise function a 500Hz sine wave we expected a impulse at 500Hz and -500Hz (since Fourier transform is even). At a SNR of -10dB we start to see the fundamental frequency resonate. As we increased the SNR our noise floor decreases.

In the last part of our signal analysis we take a 30dB SNR sinewave and pass it through a digital filter. We compare the power spectral density function to the squared Fourier transform of our filtered signal.

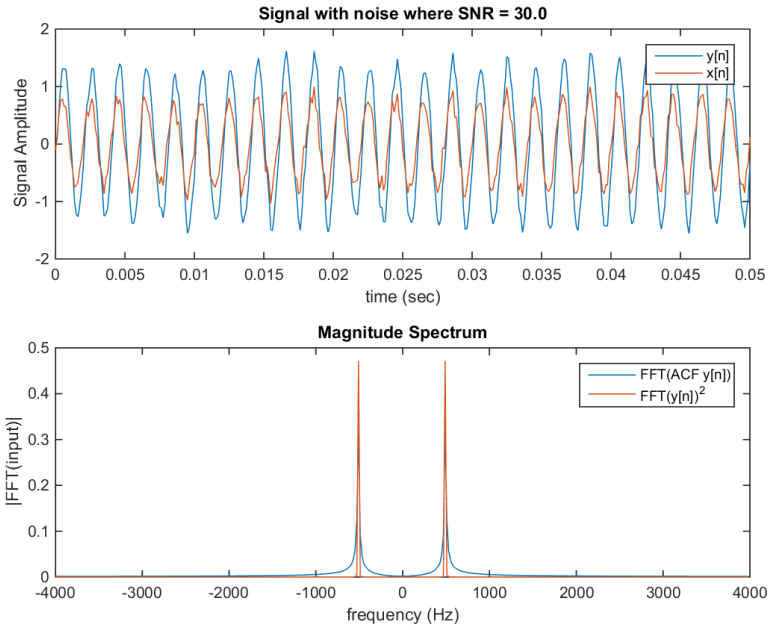

Figure 8: Signal with noise x[n] and filtered signal y[n]. PSD and Squared Fourier transform of filtered signal

In the final part of the assignment we apply a filter that takes a signal and adds half the amplitude of the previous part in the filter. The filter will remove some aspects of the noise we have in our signal. The real observation in this part of the assignment is to compare the power spectral density to the power calculated using conventional means. That is:

$$Power = \frac{1}{N}\sum_{n=1}^{N-1} Y[\Omega]^2$$

What we observe is how the two methods to calculating power match. Using the power spectral density to find the power is proffered for real signals since we always have some component of noise in our signal. The PSD is calculated from the autocorrelation function which is time invariant.

Compare the frequency plot of our filtered signal to that the frequency response of a non-filtered signal seen in Figure 7: Sine with noise signal where SNR = 30db. ACF of Signal w/ Noise, and FFT of Signal w/ NoiseFigure 7. Notice the amount of energy we have in our filtered signal is greater than that of our non-filtered. The difference in energy comes from our filter. Since our filter is combining the previous point in the signal and the last point we are amplifying the signal while suppressing the noise. The result is more energy in our filtered signal at our fundamentals when compared to the non-filtered signal.

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 sig = devGenerate_SineN(Apeak, freqHz, tMax, samFreq, snrDB)

    % Define time between samples
    dt = 1/samFreq;

    % Define time Vector
    t = 0:dt:tMax;
    numSam = length(t);

    % Create Signal
    x = Apeak*sin(freqHz*2*pi*t);

    % Find power in the sine wave
    powX = Apeak^2/2;

    % Find amplitude of noise
    % SNRDB = (Asignal/Anoise)in DB
    powNoise = powX/db2mag(snrDB);

    noiseMean = 0;
    noiseVar = powNoise;
    noiseStd = sqrt(noiseVar);

    xNoise = devNormDist(noiseMean, noiseStd, numSam);

    % Create Noisy Signal and Normalize
    sigNoisy = x + xNoise;
    sig = sigNoisy/max(sigNoisy)*Apeak;

end
```

1 Our first function 'devGenerate_SineN' allows us to create a sine wave with noise. We have five inputs, the amplitude of the sinewave desired, the frequency, the length in time of the signal, the sample frequency, and the SNR measured in dB. We enter the function and define our time vector with respect to the sample frequency and length of time passed into the function. Next we create a sine wave signal with the frequency and amplitude desired. Now we have to create a Guassian noise aspect of the signal. First we find the power of our pure sine wave. Using the SNR formula to can find the power of our noise component. Finally using the Box-Muller technique explored in the last computer assignment we can create a random normally distributed array given the number of sample desired, mean of the noise (=0) and variance (=power). We combine our sine wave with the random array of normal variables (our noise) and normalize so our maximum value never goes above the desired peak.

```
function ccf1 = devAutoCorr(Signal, lags)

    [ccf,lags,bounds] = autocorr(Signal,lags);

    ccf = ccf*max(Signal)^2/2;
    plot(lags, ccf);
    maxA = round(max(ccf));
    ylim([-maxA maxA]);
    xlabel('Lag (tau)');
    ylabel('AutoCorrelation');
    title('AutoCorrelation of Signal');

    ccf1 = ccf;
end
```

2 'devAutoCorr' allows us to find the autocorrelation of a signal given a number of lags tau. Behind the scenes we use MatLab's autocorr function to perform the autocorrelation. We are then provided the number of lags and autocorrelation function. We plot the results.

```
function FTM = devFFTMag2(Signal, fs)

    samL = length(Signal);

    df = fs/samL;
    NFFT = 2^nextpow2(samL);
    f = (-fs/2) : df : (fs/2)-df;

    % Take the fourier transform. The fftshift will duplicate our FT for
    % negative frequencies. (Plot is centered at zero)
    FT = fft(Signal)/samL;
    FTM = abs(FT);

    plot(f,fftshift(FTM, 2));
    xlabel('frequency (Hz)')
    ylabel('|FFT(input)|');
    title('Magnitude Spectrum');
end
```

3 'devFFTMag2 plots the fourier trasnform of a signal using MatLab's fft function. We define our frequency vector we plot against with respect to the sample frequency. This function only plots the magnitude spectrum of transform. The fftshift function allows us to plot positive and negative frequencies. We return an output of the magnitude of the Fourier transform out of the function.

```
clear; close all; clc

fs = 8E3;
f = 500;
sigAmp = 1;
timeL = .05;
tauMax = 16;

% Define time between samples
dt = 1/fs;
periodSig = 1/f;
sampleOneT = 2*floor(periodSig*fs);

% Define time Vector
timeVec = 0:dt:timeL;
samL = length(timeVec);

% Index Variable
i = 1;
```

4 We start our script by defining our constants. The assignment asks a sample frequency of 8kHz, a sine of 500Hz w/ Noise, and shifts up to 16. Also on our length of time we want to observe is 0.05 secs. This time length will allow us to observe 25 cycles of our signal.

```
for SNR = -30:10:30;
    % Create Noisy Signal
    sig = devGenerate_SineN(sigAmp, f, timeL, fs, SNR);
    figure('name','[ECE 3522] Class Assignment [9]');
    subplot(3,1,1);
    plot(timeVec, sig);
    ylim([-sigAmp sigAmp]);
    title(sprintf('Sine signal with noise where SNR = %0.1f',SNR));
    xlabel('time (sec)');
    ylabel('Signal Amplitude');

    % Find Power of signal squared
    pSig = sum(sig.^2)/samL;
    fprintf(sprintf('Power computed taking SUM(sig[n]^2) = %0.001f where SNR = %0.1f\n',
pSig, SNR));

    % Part 2
    subplot(3,1,2);
    % Find AutoCorrelation for the first 16 lags
    devAutoCorr(sig, tauMax);

    % Part 4
    subplot(3,1,3);
    % Fourier Tranform
    devFFTMag2(sig, fs);
    title('Magnitude Spectrum of Signal');
    ylim([0 0.5]);
end
```

5 The next step is the bulk of the script. We sweep our for loop for all the SNR ratios we wish to observe. For each SNR we generate a new signal and plot it. Next we computer the power of our signal. Finding the autocorrelation function and plotting the Fourier transform is easy now that we have our 'devAutoCorr' function. For each SNR we have find the autocorrelation and its frequency spectrum.

```
% Part 5
for SNR = 30
    % Generate your Signal
    sig = devGenerate_SineN(sigAmp, f, timeL, fs, SNR);

    % Insatiate the filtered signal
    y = zeros(1,samL);
    y(1) = sig(1);

    % Digital Filter
    for n = 2:samL
        y(n) = 0.5*y(n-1)+sig(n);
    end

    % Plotting Stuff
    figure('name','[ECE 3522] Class Assignment [9]');

    % Plot the Signal
    subplot(2,1,1);
    plot(timeVec, y);
    hold on
    plot(timeVec, sig);
    hold off
    title(sprintf('Signal with noise where SNR = %0.1f',SNR));
    xlabel('time (sec)');
    ylabel('Signal Amplitude');
    legend('y[n]', 'x[n]');

    % Plot the FFT of AutoCorrelation Function
    subplot(2,1,2);
    acf = devAutoCorr(y, samL-1);
    ftACF = devFFTMag2(acf*2,fs);
```

6 Part five calls for a SNR of 30 dB. In this analysis we will take the autocorrelation function and compute the power spectral density function. First we create a noise signal using our function. Next we pass it through a digital filter. We define the first point since the filter calls for using a sample of y[n-1]. After applying the filter we combining 'devAutoCorr' and 'devFFTMag2' allows us to find the PSD of y[n]. The PSD will tell us the power at certain frequencies.

```
% Plot the FFT of AutoCorrelation Function
    subplot(2,1,2);
    acf = devAutoCorr(y, samL-1);
    ftACF = devFFTMag2(acf*2,fs);

    % Plot FFT of Signal
    hold on
    df = fs/samL;
    NFFT = 2^nextpow2(samL);
    f = (-fs/2) : df : (fs/2)-df;

    % Take the fourier transform. The fftshift will duplicate our FT for
    % negative frequencies. (Plot is centered at zero)
    FTY = fft(y)/samL;
    FTMY = abs(FTY);
    plot(f,fftshift(FTMY, 2).^2);

    FTX = fft(sig)/samL;
    FTMX = abs(FTX);

    plot(f,fftshift(FTMX, 2).^2);
    legend('FFT(ACF y[n])', 'FFT(y[n])^2', 'FFT(x[n])^2');
end
```

7 In comparison we take the Fourier Transform and plot the magnitude spectrum squared. We can't use the Fourier transform function we created since we need to plot the square of the output. Plotting within the same subplot allows us to compare the results of this method to the PSD method.

IV. CONCLUSIONS

Through simulation using MatLab, we have shown that construction of the autocorrelation of real world signals. Last year we learned in Signals and Systems the ideal cases of signals, where no noise was present in our signals. In the ideal case, you do not have to deal with filtering out your signal after it has been transmitted. With the stochastic process we produced we showed how we can remove the noise to some extent. Noise is hard to eliminate since the random nature of its effects can never be predicted thus there is no one way to remove it. Instead we end up approximating what the characteristics of the original signal were and our results show that some definite conclusion can be made using techniques such as finding the power spectral density.