Emilie Doyle

ECE 3522: Stochastic Process in Signals and Systems

Department of Electrical and Computer Engineering, Temple University, Philadelphia, PA 1912

# Problem Statement

The objective of this computer assignment is to learn more about the signal to noise ratio and filtering through the use of MATLAB coding. The focus of this assignment is to solidify concepts surrounding a signal’s power and variance. This is to be accomplished through the completion of the following tasks. The first task is to create a function to generate a signal that is comprised of a sine wave and Gaussian white noise. The format for the function is given, including the input and output of the function. The power of the sinewave is to be determined and then the element of Gaussian white noise is to be added to the signal based on the given signal to noise ratio (SNR) which is measured in dB. The next task is to cycle through values for the SNR from -30 dB through 30 dB, incrementing at values of 10 dB. These values were then to be used to compute the autocorrelation function of the signal created by our function, with a frequency of 500 Hz sampled at 800 Hz. Next a fast Fourier Transform (FFT) was to be computed and the resulting magnitude spectrum was plotted from 0 to 4000 Hz consisting of 8192 points in the transform. Lastly, a digital filter was to be applied to a signal at 30 dB SNR. The form of the digital filter was given as:

After the filter is applied, the new autocorrelation was to be found and the Fourier Transform magnitude spectrum was to be plot again and compared.

# Approach and Results

First, I started off with making the required function. I put the input of the function as amplitude, frequency (in Hz), duration (in seconds), sample frequency (in Hz) and the signal to noise ratio (in dB). The output of the function are the original sinewave (without the added noise), the new signal (with the added noise), and the power of the original signal. The function itself is rather short. I started out by creating an array that represents the time length. In order to match it up with the signal contents (for plotting) I started from time =0 and went to the duration of the signal (in seconds). The step interval I used in between was the inverse of the sample frequency , which is given in Hertz, so the inverse of it would be the actual sample interval in seconds. Then I created a sine wave through the use of the sin function. I supplied the function with the frequency in Hertz and converted to radians by multiplying by two pi. The variable t, that corresponds to the time array, was also in the function. I multiplied all of that by the amplitude, which is given through the inputs. In order to find the power of the original sine wave signal, I squared the amplitude and divided by two, because that is the known equation for the power of a sinewave. Next, I used the awgn function to add Gaussian white noise to my signal. I provided the command with the sinewave signal as well as the signal to noise ratio, and the result is the sine wave with the added noise.

For the actual main body of the code, I started by coding in certain constant values, as well as created an array for the various signal to noise ratios used later on. I called the generate signal with Gaussian white noise (GWN). I plotted it, but the plot has way too much data in such a small range so it is virtually impossible to read and looks like a giant blob, until you zoom way in, and then you can actually see the signal form. I also plotted the signal with the Gaussian white noise, just to see.

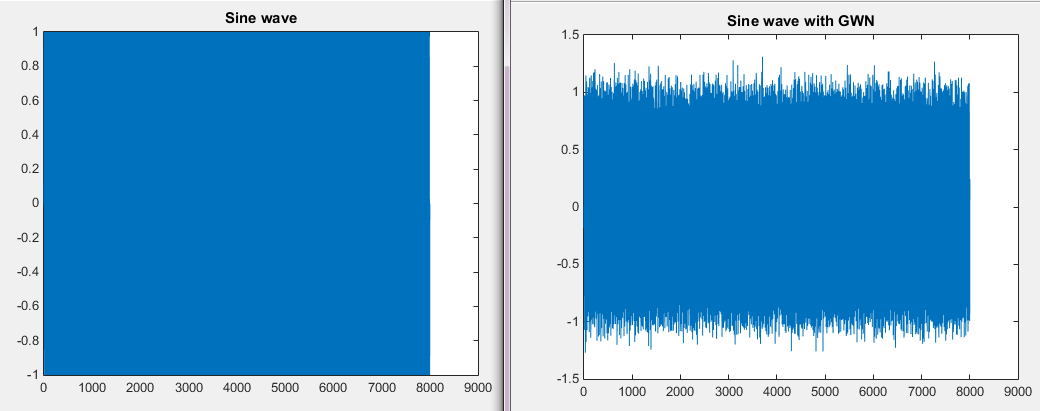


Figure : Generated Sine wave and Sine wave with GWN

Then it asks us to get the autocorrelation and the FFT for various signal to noise ratios. I did this by creating a loop and looping through the different signal to noise ratio values. Within the loop, I called the generate\_sin\_GWN function and kept the parameters the same, except for the signal to noise ratio I substituted the array of values throughout the loop. I used the autocorr function to determine the autocorrelation between the signal through 16 lags. I used the fft function to find the fast Fourier Transform of the data using 8192 points. All of these values, for all of the ratios were stored in corresponding matrices, with each row pertaining to a different signal to noise ratio. I then plotted them all through the use of subplot cut down on the number of figures used.

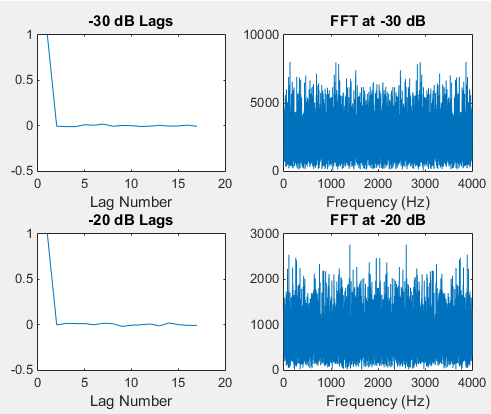


Figure : Autocorrelation and FFT for -30 dB and -20 dB

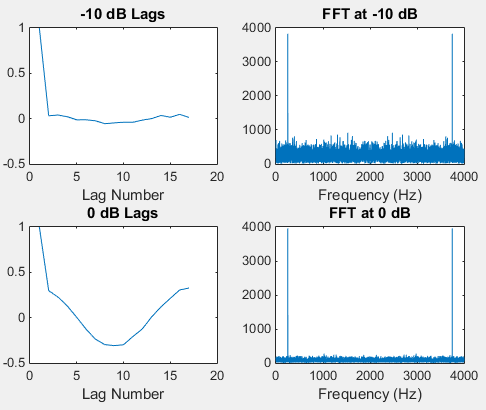


Figure : Autocorrelation and FFT for -10 dB and 0 dB

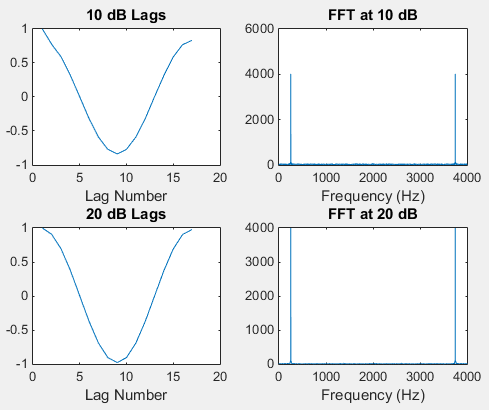


Figure : Autocorrelation and FFT for 10 dB and 20dB

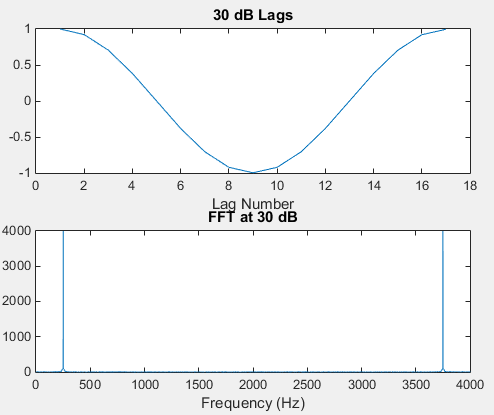


Figure : Autocorrelation and FFT for 30 dB

The next task to accomplish is the digital filtering. I used my function to generate the signal with the Gaussian white noise at 30 dB. I used the filter function and supplied the numerator and denominator of the would- be transfer function corresponding to the specified filter, and then used the filter function to use that filter on the generated signal x. I plotted the digitally filtered signal, and then used the autocorr function to determine the autocorrelation for the 16 lag values. This too was plotted, and then the fast Fourier Transform was calculated for 8192 points and plotted as well.

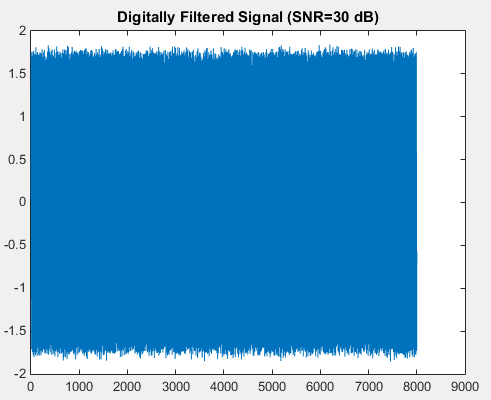


Figure : Digitally Filter Signal at 30 dB

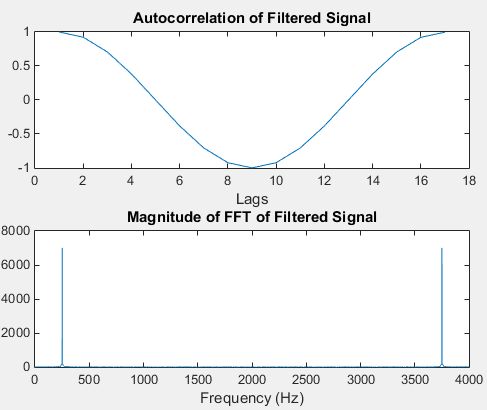


Figure : Autocorrelation and FFT of Digitally Filtered Signal with SNR= 30 dB

# MATLAB Code

Function:

function [ sinewave, sig,origSigPwr] = generate\_sin\_GWN(amplitude, frequency\_Hz, duration\_sec, sample\_freq\_Hz,snr\_db )

%UNTITLED2 Summary of this function goes here

% Detailed explanation goes here

t=0:1/sample\_freq\_Hz: duration\_sec; %set time length wrt sample frequency

sinewave= amplitude\*sin(2\*pi\*frequency\_Hz\*t);

origSigPwr= 0.5\*amplitude^2;

sig= awgn(sinewave, snr\_db);

end

Code:

%Stoch CA 9

amplitude=1;

SNR=[-30 -20 -10 0 10 20 30];

frequency\_Hz=500;

duration\_sec=1;

sample\_freq\_Hz= 8e3;

snr\_db=20;

[ sinewave, sig, origSigPwr]=generate\_sin\_GWN(amplitude, frequency\_Hz, duration\_sec, sample\_freq\_Hz,snr\_db );

l= linspace(1,(sample\_freq\_Hz) +1,(sample\_freq\_Hz +1));

figure(1)

plot(l, sinewave);

title('Sine wave');

figure(2)

plot(l, sig);

title('Sine wave with GWN');

for i= 1:7

[sinwave1, sig1, origSigPwr1]=generate\_sin\_GWN(amplitude, frequency\_Hz, duration\_sec, sample\_freq\_Hz,SNR(i));

a(i,:)= autocorr(sig1 , 16);

F(i, :)= fft( sig1, 8192);

end

lags= 1:1:17;

p= linspace(1, 4000, 8192);

figure(3)

subplot(2,2,1)

plot(lags, a(1,:));

title('-30 dB Lags');

xlabel('Lag Number');

subplot(2,2,2)

plot(p, abs(F(1,:)));

title('FFT at -30 dB');

xlabel('Frequency (Hz)');

subplot(2,2,3)

plot(lags, a(2,:));

title('-20 dB Lags');

xlabel('Lag Number');

subplot(2,2,4)

plot(p, abs(F(2,:)));

title('FFT at -20 dB');

xlabel('Frequency (Hz)');

figure(4)

subplot(2,2,1)

plot(lags, a(3,:));

title('-10 dB Lags');

xlabel('Lag Number');

subplot(2,2,2)

plot(p, abs(F(3,:)));

title('FFT at -10 dB');

xlabel('Frequency (Hz)');

subplot(2,2,3)

plot(lags, a(4,:));

title('0 dB Lags');

xlabel('Lag Number');

subplot(2,2,4)

plot(p, abs(F(4,:)));

title('FFT at 0 dB');

xlabel('Frequency (Hz)');

figure(5)

subplot(2,2,1)

plot(lags, a(5,:));

title('10 dB Lags');

xlabel('Lag Number');

subplot(2,2,2)

plot(p, abs(F(5,:)));

title('FFT at 10 dB');

xlabel('Frequency (Hz)');

subplot(2,2,3)

plot(lags, a(6,:));

title('20 dB Lags');

xlabel('Lag Number');

subplot(2,2,4)

plot(p, abs(F(6,:)));

title('FFT at 20 dB');

xlabel('Frequency (Hz)');

figure(5)

subplot(2,1,1)

plot(lags, a(7,:));

title('30 dB Lags');

xlabel('Lag Number');

subplot(2,1,2)

plot(p, abs(F(7,:)));

title('FFT at 30 dB');

xlabel('Frequency (Hz)');

%Filtering

B= 1;

A= [1 -0.5];

[sinewave2, x, pwr]=generate\_sin\_GWN(amplitude, frequency\_Hz, duration\_sec, sample\_freq\_Hz,30);

y=filter(B,A,x);

figure(6)

plot(l, y)

title('Digitally Filtered Signal (SNR=30 dB)');

figure(7)

b=autocorr(y, 16);

subplot(2,1,1)

plot(lags,b);

title('Autocorrelation of Filtered Signal');

xlabel('Lags');

dfft= fft(y, 8192);

subplot(2,1,2)

plot(p, abs(dfft));

title('Magnitude of FFT of Filtered Signal');

xlabel('Frequency (Hz)');

# Conclusions

The power of a plain and normal sine wave can be easily calculated by squaring the amplitude and then dividing that by two. When the amplitude is plus or minus 1, then the power will be 0.5. It doesn’t matter if the amplitude is positive or negative, because the value gets squared anyway.

The positive signal to noise ratios appear to be fine for the autocorrelation and the FFT. They produced essentially part of a sine wave, which is what should happen due to the nature and properties of the input sinewave. The FFT shows content at specific frequencies at intervals regarding the sampling frequency. We can expect to see spikes in the FFT graph when the sine wave has content. It is not until the signal to noise ratio is 0 dB that the Gaussian white noise begins to show up and start to interfere with the results. As the signal to noise ratio becomes more negative, the Gaussian white noise has a larger and more prominent effect on the resulting plots of the autocorrelation and the fast Fourier Transform. As the ratio value gets more negative, the autocorrelation quickly becomes zero, due to the sporadic nature of the white noise and the zero mean of a Gaussian distribution.

The autocorrelation function and the magnitude of the fast Fourier transform of the digitally filtered signal at 30 dB are almost identical to that of the regular signal at 30 dB. This is because of the periodic nature of the sine wave and the fact that with a signal to noise ratio of 30 dB, the Gaussian white noise is not prominent enough to detract from the autocorrelation or to really show content on the magnitude plot of the fast Fourier Transform.