Andy Lew
DSP Project 2 Report

Abstract:

The objective of this project was to learn more about FIR filters and how to use them to get rid of noise in signals. The main piece of software used was MATLAB, more specifically the fdatool. The fdatool is used to create a variety of filters by using the inputs given. The main three types of filters used in this project were high pass, low pass and band pass filters. Noise signals were first used to contaminate a signal, so it could be analyzed to increase familiarity with their spectrums and waveforms. Different speech signals that contained some noise were then read into MATLAB and analyzed in similar fashion using their spectrums and frequency responses. Based off the "shapes" of their spectrums, different filters were created to remove the noise that was present. Scripts were created to filter the signals, play the signals and plot their spectrums and frequency responses. Most the speech signals have been "cleared" up and the messages have been deciphered, however some of the signals were influenced by white noise that were unable to be completely removed.

Introduction and Objectives:

There are many type of filters in the world, but this project focused on digital filters. The digital filters mainly consisted of FIR filters that were either low pass, high pass or band pass. Each of these filters being specialized for different scenarios. For low pass filters, they are used to block out higher frequencies. For high pass filters, they block out lower frequencies and for band pass filters, they are adjusted to let a certain frequency range be heard. With these three types of filters, one can get rid of unwanted frequencies in a distorted speech signal.

The objectives of this project are as follows: to learn more about how to use spectrums and frequency responses to filter signals, to become more familiar with software tools for filtering, and to experiment with several different filters to enhance signals corrupted by "noise." The software used to create the filters was MATLAB with its fdatool which allows users to pick from a variety of different types of filters and input their specifications to what they desire. MATLAB is used all around the world and is quite versatile in all that it can do. With the fdatool built into MATLAB, it is easy to look at the spectrum of a signal and design the necessary filter to block the unwanted frequencies.

Methods:

There were a few different methods used in the designing process of the filters used. The first thing done for every signal was to listen to the sound of the signal through the MATLAB function soundsc. Sometimes the "noise" was very easy to identify while in other situations it would be quite difficult to pinpoint the specific "noise" that would need to be blocked. After listening to the signals a couple of times, the spectrum would then be plotted using the function pwelch. With the spectrum plotted, it could now be analyzed to look for key areas or points on

them. These key locations would be based off how the speech signal sounded. For example, if the “noise” sounded like it was a higher frequency, then we would look for spikes in the spectrum at higher frequencies. If the frequencies disrupting the signals sounded like a lower tone, then the spectrum would be analyzed for a spike in the lower frequencies.

There were two main ways of attempting to clarify the messages in the signals. The first method was to increase the difference between the wstop and wpass values for low pass filters. Instead of filtering out a certain frequency range completely with low pass filters, it was found that messages could be heard more clear sometimes when the frequencies allowed were not completely filtered. This is due to speech signals having a larger range of frequencies rather than a set frequency. The other method used to enhance the signals was to modify the sampling frequency when listening to them. When the sampling frequency is decreased, the speech signals appear to have a lower tone and are played at a slower speed. If a signal was being sampled too quickly and the message was too fast to decipher than lowering the sampling frequency would make it easier to hear the message. In the same way, if a signal was being sampled too slow, then it would appear to have a very low tone and play at a slower rate. Increasing the sampling frequency in this case could clarify the message.

Results:

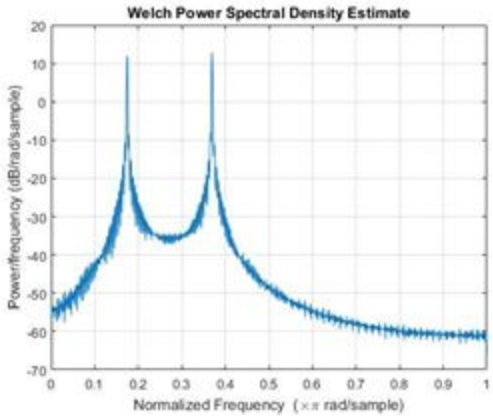

Figure 1. Spectrum of Wave1

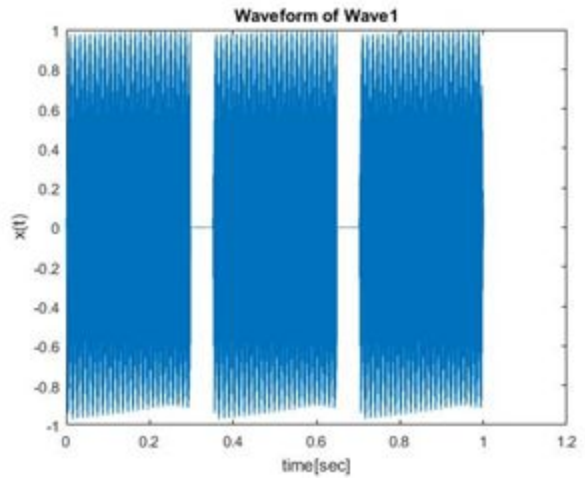

Figure 2. Portion of Wave1

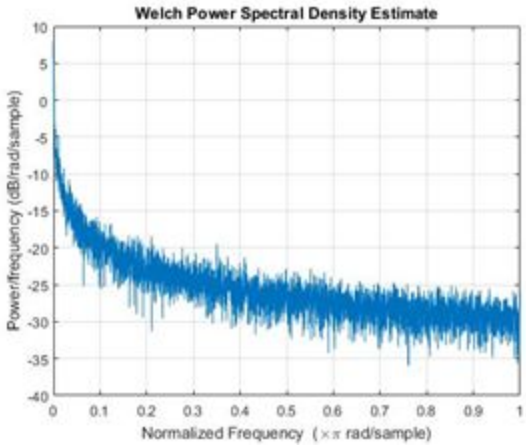

Figure 3. Spectrum of Pink Noise

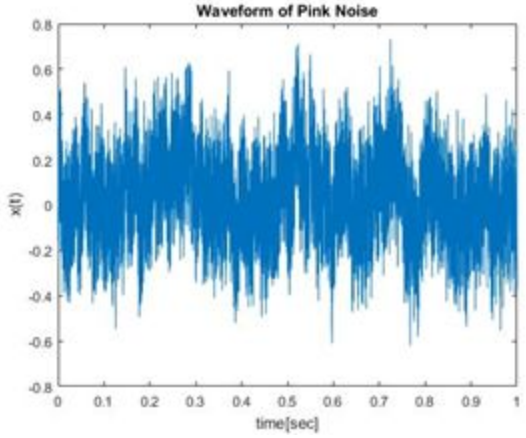

Figure 4. Waveform of Pink Noise

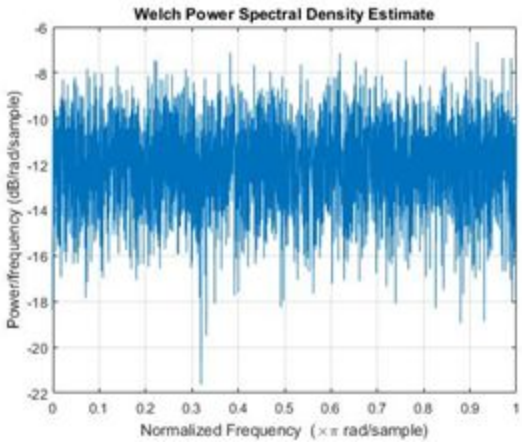

Figure 5. Spectrum of White Noise

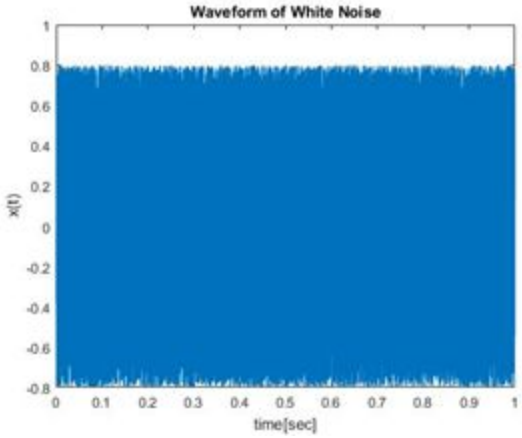

Figure 6. Waveform of White Noise

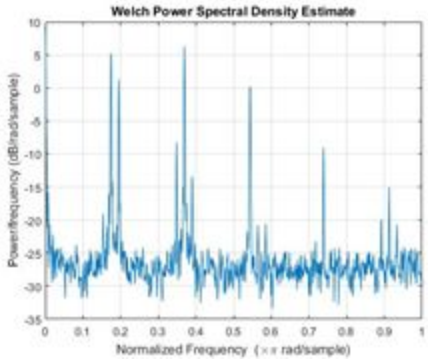

Figure 7. Spectrum of Wave1 with Gaussian noise inserted

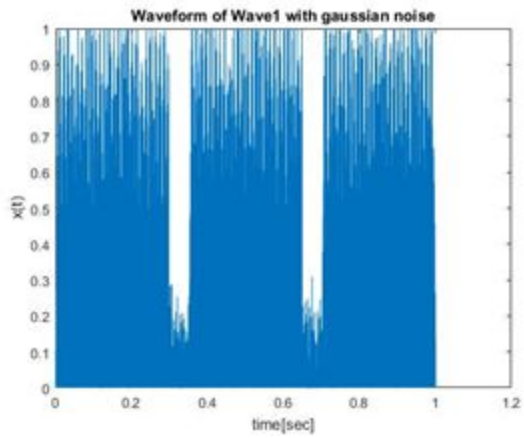

Figure 8. Portion of Wave1 with Gaussian noise inserted

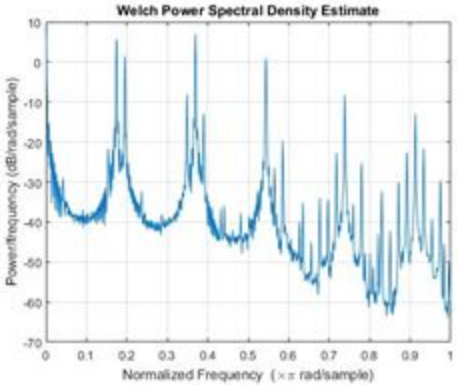

Figure 9. Spectrum of Wave1 with Poisson noise inserted

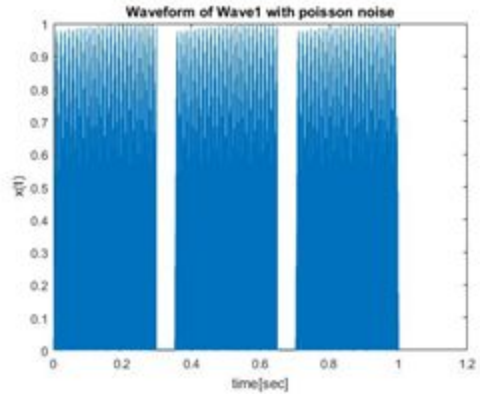

Figure 10. Portion of Wave1 with Poisson noise inserted

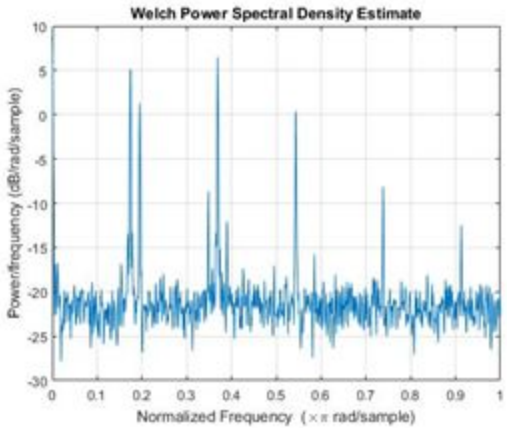

Figure 11. Spectrum of Wave1 with salt and pepper noise inserted

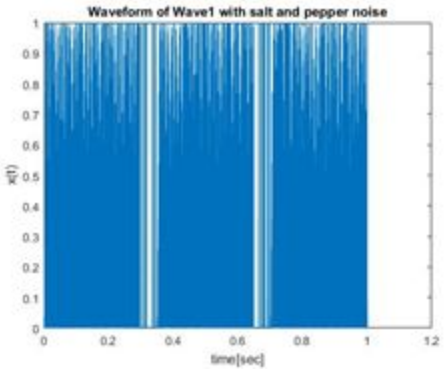

Figure 12. Portion of Wave1 with salt and pepper noise inserted

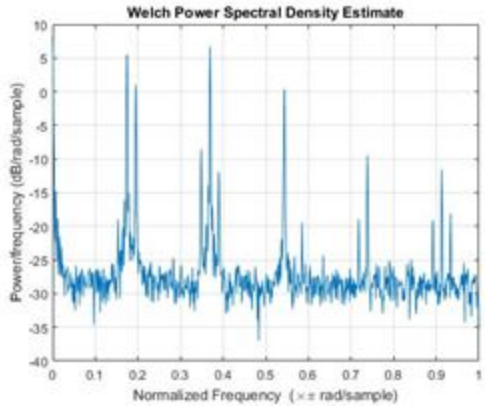

Figure 13. Spectrum of Wave1 with speckle noise inserted

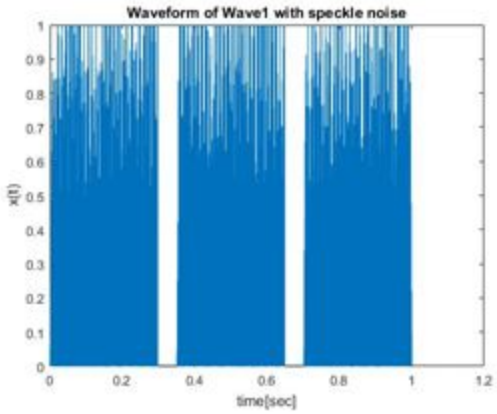

Figure 14. Portion of Wave1 with speckle noise inserted

Table 1. Generalized Analysis of Signals

Signal	Type of Noise
Nsp0	Higher Frequency Tone
Nsp1	Higher Frequency Tone
Nsp2	White Noise
Nsp3	Pink Noise

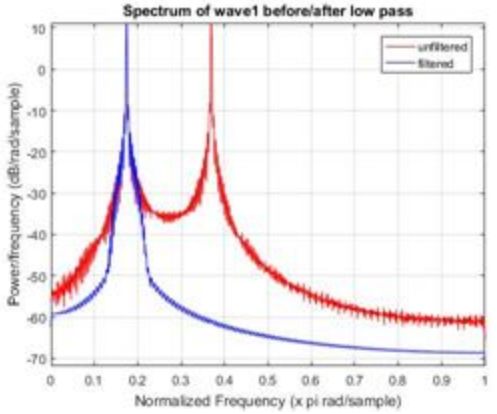

Figure 15. Spectrum of Wave1 before/after low pass filter applied

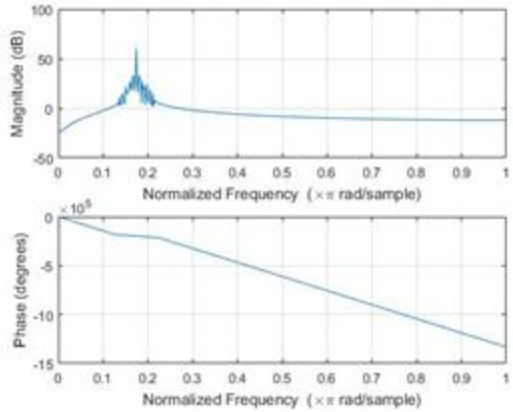

Figure 16. Frequency response of Wave1 low pass filter

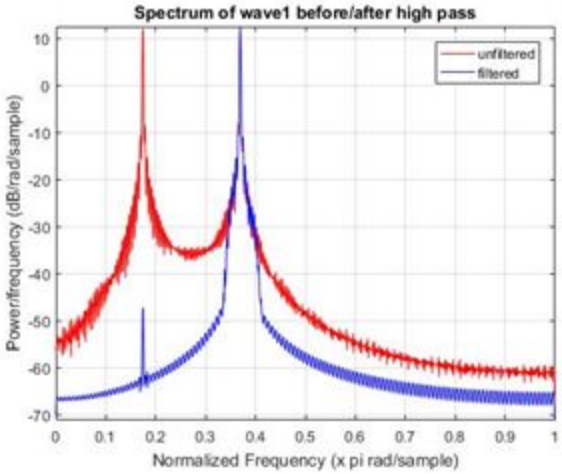

Figure 17. Spectrum of Wave1 before/after high pass filter applied

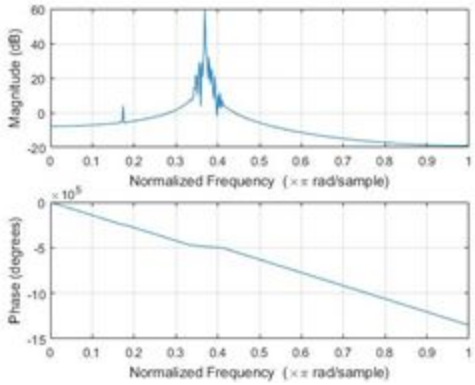

Figure 18. Frequency response of Wave1 high pass filter

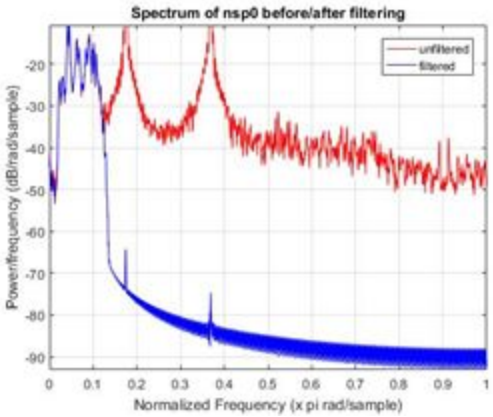

Figure 19. Spectrum of nsp0 before/after filtering

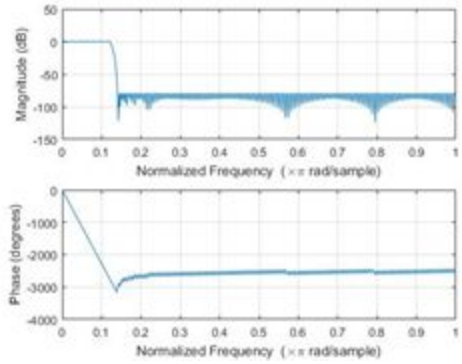

Figure 20. Frequency response of nsp0 filter

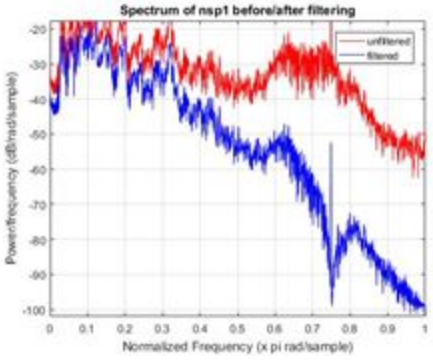

Figure 21. Spectrum of nsp1 before/after filtering

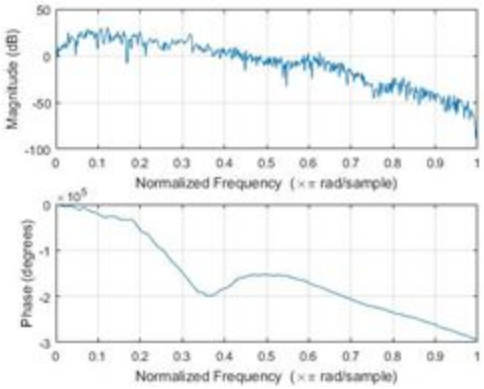

Figure 22. Frequency response of nsp1 filter

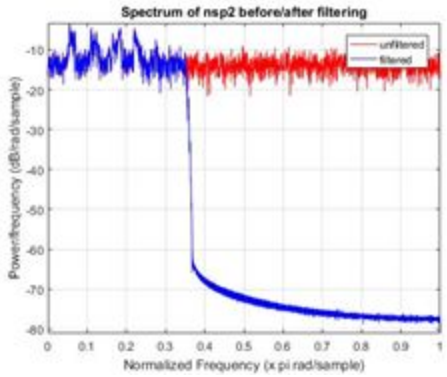

Figure 23. Spectrum of nsp2 before/after filtering

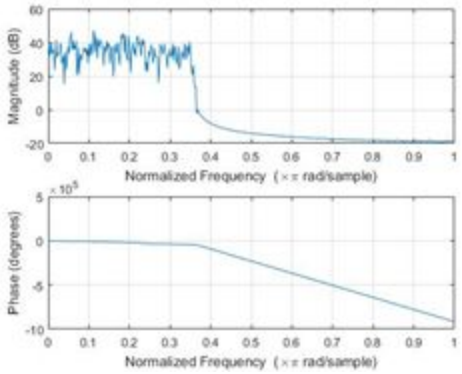

Figure 24. Frequency response of nsp2 filter

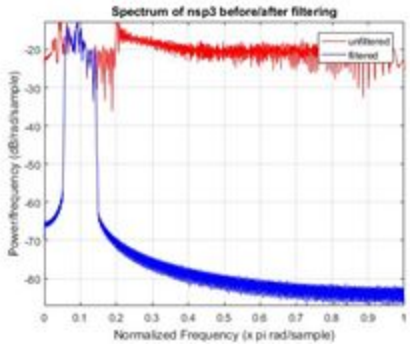

Figure 25. Spectrum of nsp3 before/after filtering

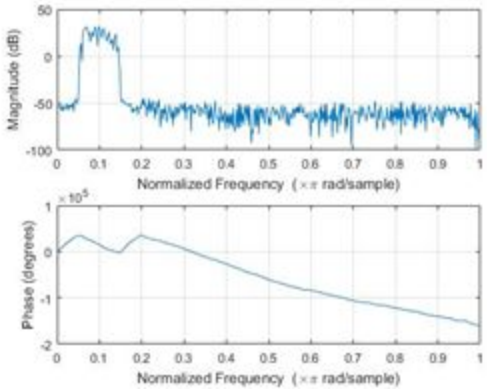

Figure 26. Frequency response of nsp3 filter

Table 2. Messages deciphered from contaminated speech signals

Signal	Message
Nsp0	Should we chase?
Nsp1	Thieves who rob friends deserve jail.
Nsp2	Add the sum to the product of these three.
Nsp3	Should we chase?

Discussion:

Many signals were analyzed in this project starting with wave1. Figures 1 and 2 show the spectrum and waveform of wave1 respectively. With the two spikes in the spectrum, wave1 was seen to be formed from two tones at different frequencies. Wave1 was then contaminated by four different types of noise: Gaussian, Poisson, Salt & Pepper, and Speckle. The spectrums and waveforms of the contaminated signals can be seen from figures 7-14. Each type of noise seems

to have its own “signature,” but by looking at the spectrums alone, they seem very similar. For example, the spectrums of the Gaussian and salt & pepper contaminated signals are almost identical, while the Poisson spectrum has many more peaks at different frequencies. Next, we examined white noise and pink noise. The spectrums and waveforms for these can be seen in figures 3-6. Pink noise has a very distinct looking spectrum where the noise occurs at almost all frequencies, but is more prevalent in lower frequencies. White noise however can contaminate a signal at all frequencies at about the same strength. This makes white noise very difficult to remove from signals. With some knowledge of how different types of “noise” work, we then examined the four speech signals and tried to determine what the “noise” that would need to be filtered out was. This can be seen in table 1. The classifications listed are based off the sound of the signals alone, the spectrums were later plotted and analyzed. From the sound of the signals, nsp0 and nsp1 seemed to be affected by a higher frequency “noise” and nsp2 and nsp3 seemed to be affected by either white or pink noise. Through the plotting of the spectrums, it could be seen that certain ranges of frequencies were affected by “noise.” To get rid of the “noise” in each signal, filters needed to be designed to certain specifications.

We started off by looking back at wave1’s spectrum and seeing that there were two distinct peaks around .18 and .38 (normalized frequency units). Since the “noise” was clear to see, a band pass filter could be designed to separate these two frequencies from each other. The low pass filter’s frequency response can be seen in figure 16, while the high pass filter’s frequency response can be seen in figure 18. The before and after spectrum plots of the two filters are referenced in figures 15 and 17. It is important to note that the magnitude response is in the same “shape” as the “after filtered” spectrums as they should be. The phase response for the filters are also all linear since we are using FIR filters. If we did not have linear phase, then the filters would add distortion to the signals and the messages would become very difficult to understand. The nsp0 and nsp1 signals were both filtered similarly. Their respective spectrums and frequency responses are in figures 19-22. Both were low pass filters, but with different wpass and wstop values. Nsp0’s message was easier to understand with a small difference of wpass and wstop values, while nsp1’s message needed a larger difference. This was caused primarily because of the ranging frequencies in the speech signals. Nsp2 was unable to be completely “noise” free since white noise was contaminating the signal. With white noise interfering with the speech signal at all frequencies, it became very difficult to filter out the “noise” and make the message comprehendible. Nsp2’s spectrum and frequency response are shown in figures 23 and 24. The spectrum shows that even the speech signal was filtered to the output, it is still affected by the “noise” by a substantial amount. Nsp3’s spectrum and frequency response can be seen in figures 25 and 26. By looking at the “unfiltered” signal, the spectrum seems to have a familiar appearance to the pink noise spectrum from figure 3. With pink noise being like white noise, it was difficult to completely get rid of the “noise,” but a band pass filter was designed to let the frequency range of the speech signal appear in the output. After all the

filtering, most of the messages in the signals could be made out and the results are listed in table 2.

Conclusion

Overall the project was a success. MATLAB has proven in many cases to be a very versatile tool. It's fdatool filter designer was very helpful to learn more about how filters are created with certain parameters and how they can affect signals. The use of spectrums to design filters was also determined to be helpful. There are many filters aside from the ones used in this project, but learning the uses of these three will prove to be useful in the future. Understanding how frequency responses and spectrums relate to filters is an important concept in digital signal processing and the knowledge obtained through this project can be applied to many other fields of DSP.

Appendix:

```
function signals(choice)
% Contaminates a signal with different types of noise and then play the
% output through the speakers
%
% usage: signals(choice)
%
% choice = which type of noise to contaminate with (0 for no noise
%          1 for gaussian, 2 for poisson, 3 for salt and pepper, and
%          4 for speckle
%

if (nargin == 0)
        choice = 0;
end
[x,fs] = audioread('wave1.wav');              % uncomment for wave1
% [x,fs] = audioread('pink_noise.wav');       % uncomment for pink_noise
% [x,fs] = audioread('white_noise.wav');      % uncomment for white_noise

a = x;
if (choice == 1)
        a = imnoise(2*(mat2gray(x)-0.5),'gaussian');
elseif (choice == 2)
        a = imnoise(2*(mat2gray(x)-0.5),'poisson');
elseif (choice == 3)
        a = imnoise(2*(mat2gray(x)-0.5),'salt & pepper');
elseif (choice == 4)
        a = imnoise(2*(mat2gray(x)-0.5),'speckle');
end
soundsc(a);
% tt = (1/fs)*(1:length(a));
```

```
% plot(tt,a)
% xlabel('time[sec]');
% ylabel('x(t)');
% title('Waveform of Pink Noise');
% pwelch(a);
```

Listing 1. Signals.m Code

```
function project2_filtering(choice, wave1)
% Filter out the noise in a signal and play the output through
% the speakers
%
% usage: project2_filtering(choice, wave1)
%
% choice = which signal to play depending if wave1 is selected
%          (0-3 correspond to nsp0-nsp3 if 1 argument is detected,
%          1-2 correspond to high or low filter of wave1 if 2 arguments
%          are detected)
% wave1 = decides if wave1 is selected (0 for no, 1 for yes)
%

if (nargin == 0)
        choice = 1;
        wave1 = 1;
elseif (nargin == 1)
        wave1 = 0;
end

%% wave1
if (wave1 == 1)
        [x,fs] = audioread('wave1.wav');
        if (choice == 1)
        wave1highcoef = load('HighFilterWave1.mat');
        wave1high = wave1highcoef.Num;
        wave1filteredhigh = filter(wave1high,1,x);
        soundsc(wave1filteredhigh,fs);
        else
        wave1lowcoef = load('LowFilterWave1.mat');
        wave1low = wave1lowcoef.Num;
        wave1filteredlow = filter(wave1low,1,x);
        soundsc(wave1filteredlow,fs);
        end
end
% wave1filtered = wave1filteredhigh + wave1filteredlow;
% soundsc(wave1filtered, fs);
% normal_wav = wave1filteredhigh/max(abs(wave1filteredhigh));
% audiowrite('wave1_high.wav', normal_wav, fs);
% [Pxx1 F1] = pwelch(x);
```

```
% Pxx1dB = 10*log10(Pxx1);
% plot(F1/pi,Pxx1dB,'-r')
% hold on
% [Pxx2 F2] = pwelch(wave1filteredlow);
% Pxx2dB = 10*log10(Pxx2);
% plot(F2/pi,Pxx2dB,'-b')
% axis([0 1 min(Pxx2dB) max(Pxx2dB)])
% hold off
% grid on
% legend('unfiltered', 'filtered','location','northeast');
% xlabel('Normalized Frequency (x pi rad/sample)');
% ylabel('Power/frequency (dB/rad/sample)');
% title('Spectrum of wave1 before/after low pass');
% freqz(wave1filteredlow);
% soundsc(x,fs);


%% nsp0
if (wave1 == 0)
        if (choice == 0)
        [x,fs] = audioread('nsp0.wav');
        nsp0coef = load('nsp0');
        nsp0 = nsp0coef.Num;
        nsp0filtered = filter(nsp0,1,x);
        soundsc(nsp0filtered, fs*1.1);
        end
end
% freqz(nsp0);
% normal_nsp0 = nsp0filtered/max(abs(nsp0filtered));
% audiowrite('fsp0.wav', normal_nsp0, fs*1.1);
% [Pxx1 F1] = pwelch(x);
% Pxx1dB = 10*log10(Pxx1);
% plot(F1/pi,Pxx1dB,'-r')
% hold on
% [Pxx2 F2] = pwelch(nsp0filtered);
% Pxx2dB = 10*log10(Pxx2);
% plot(F2/pi,Pxx2dB,'-b')
% axis([0 1 min(Pxx2dB) max(Pxx2dB)])
% hold off
% grid on
% legend('unfiltered', 'filtered','location','northeast');
% xlabel('Normalized Frequency (x pi rad/sample)');
% ylabel('Power/frequency (dB/rad/sample)');
% title('Spectrum of nsp0 before/after filtering');
% soundsc(x,fs);

%% nsp1
if (wave1 == 0)
```

```
        if (choice == 1)
        [x,fs] = audioread('nsp1.wav');
        nsp1coef = load('nsp1');
        nsp1 = nsp1coef.Num;
        nsp1filtered = filter(nsp1,1,x);
        soundsc(nsp1filtered, fs*2);
        end
end
% freqz(nsp1filtered);
% normal_nsp1 = nsp1filtered/max(abs(nsp1filtered));
% audiowrite('fsp1.wav', normal_nsp1, fs*2);
% [Pxx1 F1] = pwelch(x);
% Pxx1dB = 10*log10(Pxx1);
% plot(F1/pi,Pxx1dB,'-r')
% hold on
% [Pxx2 F2] = pwelch(nsp1filtered);
% Pxx2dB = 10*log10(Pxx2);
% plot(F2/pi,Pxx2dB,'-b')
% axis([0 1 min(Pxx2dB) max(Pxx2dB)])
% hold off
% grid on
% legend('unfiltered', 'filtered','location','northeast');
% xlabel('Normalized Frequency (x pi rad/sample)');
% ylabel('Power/frequency (dB/rad/sample)');
% title('Spectrum of nsp1 before/after filtering');
% soundsc(x,fs);

%% nsp2
if (wave1 == 0)
        if (choice == 2)
        [x,fs] = audioread('nsp2.wav');
        nsp2coef = load('nsp2');
        nsp2 = nsp2coef.Num;
        nsp2filtered = filter(nsp2,1,x);
        soundsc(nsp2filtered, fs);
        end
end
% normal_nsp2 = nsp2filtered/max(abs(nsp2filtered));
% audiowrite('fsp2.wav', normal_nsp2, fs);
% freqz(nsp2filtered);
% [Pxx1 F1] = pwelch(x);
% Pxx1dB = 10*log10(Pxx1);
% plot(F1/pi,Pxx1dB,'-r')
% hold on
% [Pxx2 F2] = pwelch(nsp2filtered);
% Pxx2dB = 10*log10(Pxx2);
% plot(F2/pi,Pxx2dB,'-b')
% axis([0 1 min(Pxx2dB) max(Pxx2dB)])
```

```
% hold off
% grid on
% legend('unfiltered', 'filtered','location','northeast');
% xlabel('Normalized Frequency (x pi rad/sample)');
% ylabel('Power/frequency (dB/rad/sample)');
% title('Spectrum of nsp2 before/after filtering');
% soundsc(x,fs);

%% nsp3
if (wave1 == 0)
        if (choice == 3)
        [x,fs] = audioread('nsp3.wav');
        nsp3coef = load('nsp3');
        nsp3 = nsp3coef.Num;
        nsp3filtered = filter(nsp3,1,x);
        soundsc(nsp3filtered, fs);
        end
end
% normal_nsp3 = nsp3filtered/max(abs(nsp3filtered));
% audiowrite('fsp3.wav', normal_nsp3, fs);
% freqz(nsp3filtered);
% [Pxx1 F1] = pwelch(x);
% Pxx1dB = 10*log10(Pxx1);
% plot(F1/pi,Pxx1dB,'-r')
% hold on
% [Pxx2 F2] = pwelch(nsp3filtered);
% Pxx2dB = 10*log10(Pxx2);
% plot(F2/pi,Pxx2dB,'-b')
% axis([0 1 min(Pxx2dB) max(Pxx2dB)])
% hold off
% grid on
% legend('unfiltered', 'filtered','location','northeast');
% xlabel('Normalized Frequency (x pi rad/sample)');
% ylabel('Power/frequency (dB/rad/sample)');
% title('Spectrum of nsp3 before/after filtering');
% soundsc(x,fs);
```

Listing 2. Project2_filtering.m Code