MATLAB Signal Processing Analysis in Time/Frequency Domain, Including White Noise Interference Filter

Group A: Andreas Vlahos, Andres Gene
May 1, 2023
EENG 341 (Signals and Systems)
Professor Dong

Due Date: May 1st, 2023

1) **Objective:**

The objective of this project was to create a 10 second snippet of the ZeddXValorant theme song, aptly named ValorantXSignals&System, using different signals of varying frequencies. Analysis of the audio waveform that was generated was done using MATLAB software, which helped plot the signal in the Time/Frequency domain in addition to creating the spectrogram. White Gaussian Noise was applied, and filtered using the toolbox provided by MATLAB, which aided in the illustration of how the output can be altered as a result of various filters and external noise.

Procedure:

The first step to creating our song was to write code that generates sound. The code found in Appendix A served as the basis for the project. In summary, a variety of frequencies were used to create different musical notes, and the duration of each affected how long the audio signal was played for. A graphical representation of the resulting audio signal can be displayed in the Time and Frequency Domain in order to understand the output.

A sampling frequency was determined in the initial stages in composing this remix. This aided in determining the pitch that was required for each note, and for how long each note needed to be played for. This project required a sampling frequency of 11kHz, which resulted in an end product that was much closer to the original piece in regard to the tune and speed.

Different notes were created using different frequencies, which corresponded to the name of each variable. The formula provided in Appendix A used the following sine expression to represent the general form of each note: **v = sin(2*pi*n*[0:0.000125:1.0])**, where **v** is an arbitrary note, and **n** is the corresponding frequency of that note. The interior matrix is used to determine the duration of the note [0:1] , and how quickly it needed to be incremented [0.000125] by to achieve the desired rhythm.

After compiling the different notes and formatting the basic structure of the song by line, for example: line1 = [a, b, a, c, d], they can be compiled into the song as follows:
song = [line1, line2, line3]. Note that for this project, the song exceeds the 10 second threshold by about 9 seconds. However, only the first 9 seconds are necessary for the completion of the assignment.

The audio signal was then converted into an audio signal utilizing the code below:

```
audiowrite('ValorantXSignals&Systems.wav', song, FS);
[audioIn, FS] = audioread('ValorantXSignals&Systems.wav');
audioOut = stretchAudio(audioIn,9);
```

The audiowrite function will create the audio wave, audioread will read the wave's input, and stretchAudio will stretch the audio wave by a factor of 9. By stretching the audio, the sound produced is much clearer.

A vector was then used to plot the Time domain of the audio signal, which was defined as follows: **t = (0:size(audioOut,1)-1)/FS;**. This creates the time vector required in order to plot the stretch audio output as mentioned before.

Using the **fft** function of Matlab, we were able to convert from time domain to frequency domain signal. This is demonstrated with the following lines of code:

```
[y, FS] = audioread('ValorantXSignals&Systems.wav');
Y = fft(y); n = numel(y);
frequency = linspace(-FS/2, FS/2, n+1);
frequency(end) = [];
shiftSpectrum = fftshift(Y);
```

This process finishes the graphical illustration of the audio file. For context, **fft** is defined as the Fast Fourier Transform of the audio signal, which was obtained using the audioread function. Note that **numel(y)** refers to the number of elements in the specified array. The frequency is then created using the **linspace()** function to generate a frequency vector. In order to shift the audio file's spectrum in the frequency domain, we use shiftSpectrum. Finally, **freq(end)** will represent the end frequency within the array.

Since we had both the time and frequency domain signal the only thing left was to plot them and also generate a spectrogram plot. The audio output was then plotted using the subplot function, which implements the **plot(y)** function. This created and plotted the signal in the time domain on the graph that we see as Figure 1 in the Graphical Analysis section. The magnitude of the audio signal was created next in the frequency domain. This is done similarly by using the subplot function to create the graph desired. Since this is to be plotted in the frequency domain, the following line was implemented to do so: **plot(frequency,abs(shiftSpectrum));**. This allowed the audio signal to be plotted in the frequency domain using the **fft** function. The spectrogram plot was created by utilizing the following lines of code:

```
figure;
window=hamming(1024);
noverlap=512;
nfft=2048;
[S,F,T,P]=spectrogram(y,window,noverlap,nfft,FS,'yaxis');
surf(T,F,10*log10(P),'edgecolor','none');
axis tight;view(0,90);
colormap(hot);
Colorbar; set(gca,'clim',[-95 0]);
 title('Spectrogram');
```

```
xlabel('seconds');
ylabel('kHz');
```

The code above divided the signal into 1024 segments which determined the number of overlapping samples in the window, which came out to be 512. This will create the frequency resolution. After those steps are completed, the spectrogram will be plotted and adjusted depending on the size of the window. The design and color of the spectrogram are arbitrary to the result/output.

For the extra credit we used a toolbox to generate the white Gaussian noise for us and put in our signal. In order to add in the white noise to the audio, the following line was implemented: **Gaussian = awgn(song, 10);**. The signal-to-noise ratio was defined as 20 in this case. Additionally, the line **audiowrite('WNoise.wav', Gaussian,FS);** serves to allow the white noise to be heard as the signal was sampled, which creates a distorted version of the audio output by adding the white noise to the background. It was then plotted similarly to how all of the other time/frequency and spectrum domain plots by use of the subplot functions.

After generating the white noise we coded a bandpass filter to filter out the noise so that only the desired music was playing.This was done by using the following MATLAB function: **Filter = bandpass(song,[1, 11000],FS);,** which filtered out the white noise from the audio signal. In this case, 11000 specifies the frequency that is permitted to pass through the filter. This is the same as the sampling frequency that was utilized to create the audio wave in the beginning. The filtered noise was then plotted in the time and frequency domain, which allowed for a better understanding of the impact of the filter. A spectrogram of this was also created as well.

To compare our original audio signal to the signal affected White Gaussian Noise and the altered signal after applying the bandpass filter, the results from each were plotted on the same time/frequency domain plots. The spectrogram was created and plotted in the same fashion and are showcased below.

2) Graph Representations / 3) Explanation and Analysis:

a. Figure 1: Time/Frequency Domain Plots and Analysis

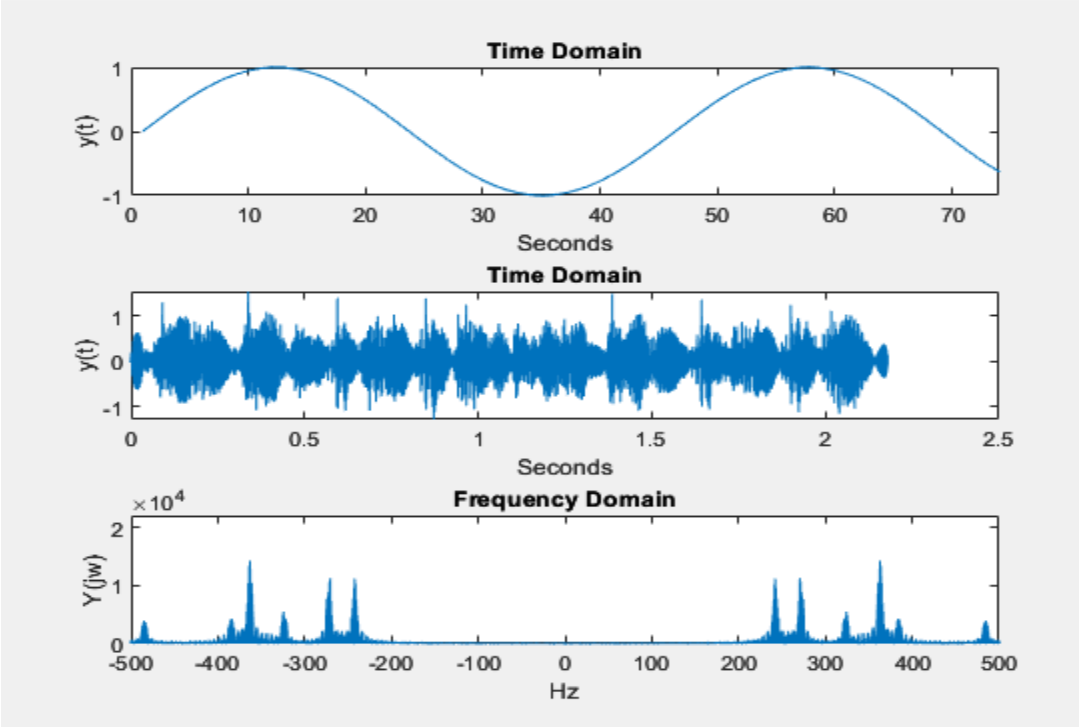

Figure 1 displays the signal in both the time and frequency domain. In the time domain we see perfect sin waves combined together with zero distortion. The middle graph is the same time domain representation of the signal, however it is stretched by a factor of 9 to best visualize the output of the signal. In the frequency domain we see peaks at positive and negative 250, 350, 500 HZ. These results agree with the American standard pitch values.

b. Figure 2: Spectrogram Plot and Analysis

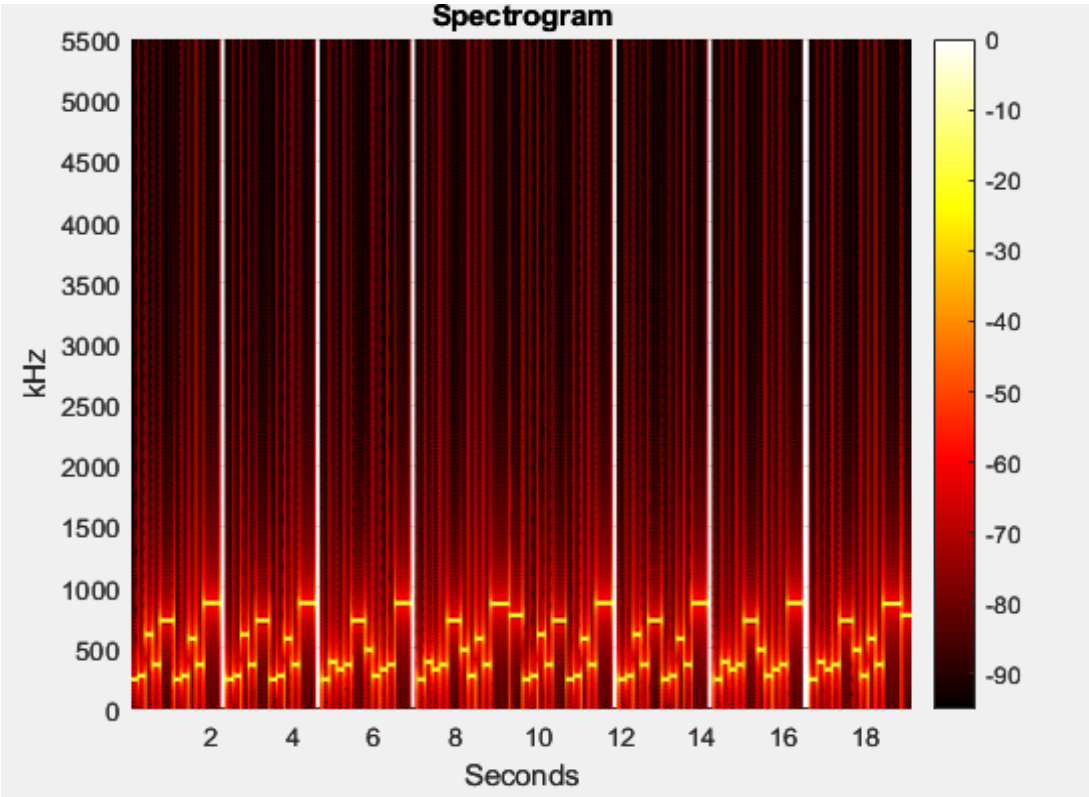

Figure 2 displays the spectrogram of the original signal. It shows the combination of frequencies at different points in time and for how long each note lasts. The white streaks are slight pauses[0 kHz] in the music, and the depth of each color on this plot references the pitch at that given time; where the darkest color indicates a strong signal at that frequency.

c. Figure 3: Plot and Analysis of Signal After Applying White Gaussian Noise

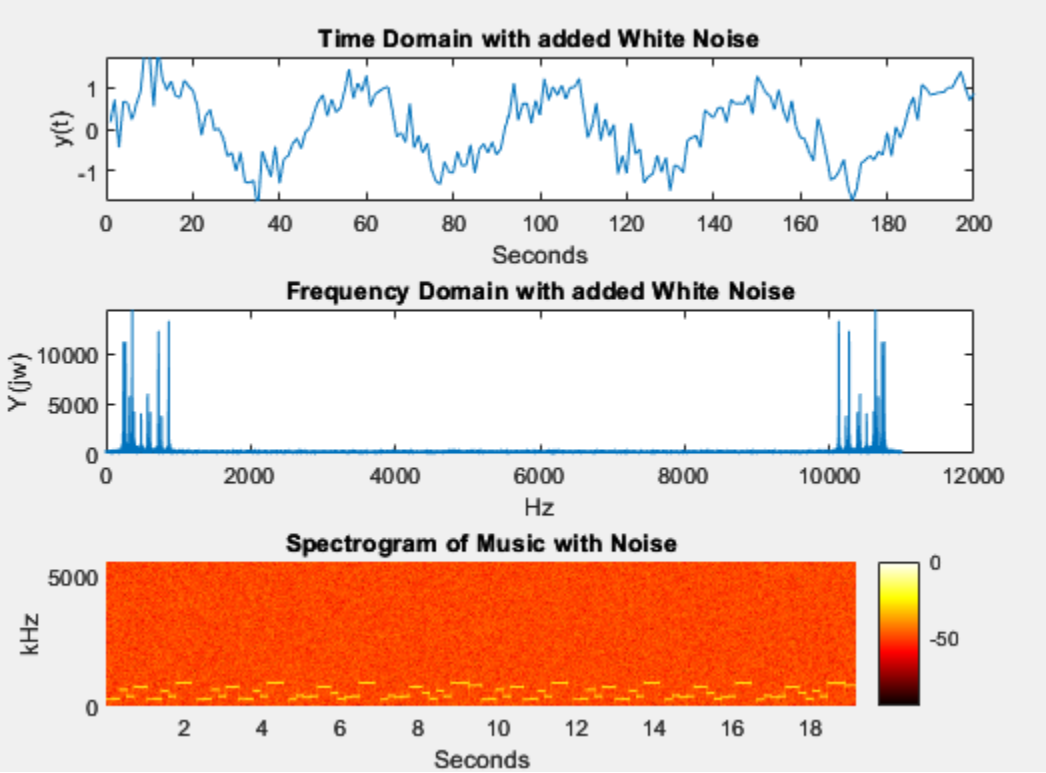

Figure 3 White Gaussian Noise (WGN) is introduced and now the time signal has become a bit distorted. Also the frequency domain has shifted greatly with the addition of noise. This is due to the Nyquist frequency. Furthermore, the spectrogram for this signal is all over the place because the white noise has a bunch of different frequencies. The gaps in the original audio file have now been filled with white noise.

d. Figure 4: Plot and Analysis of WGN Signal After Use of Bandpass Filter

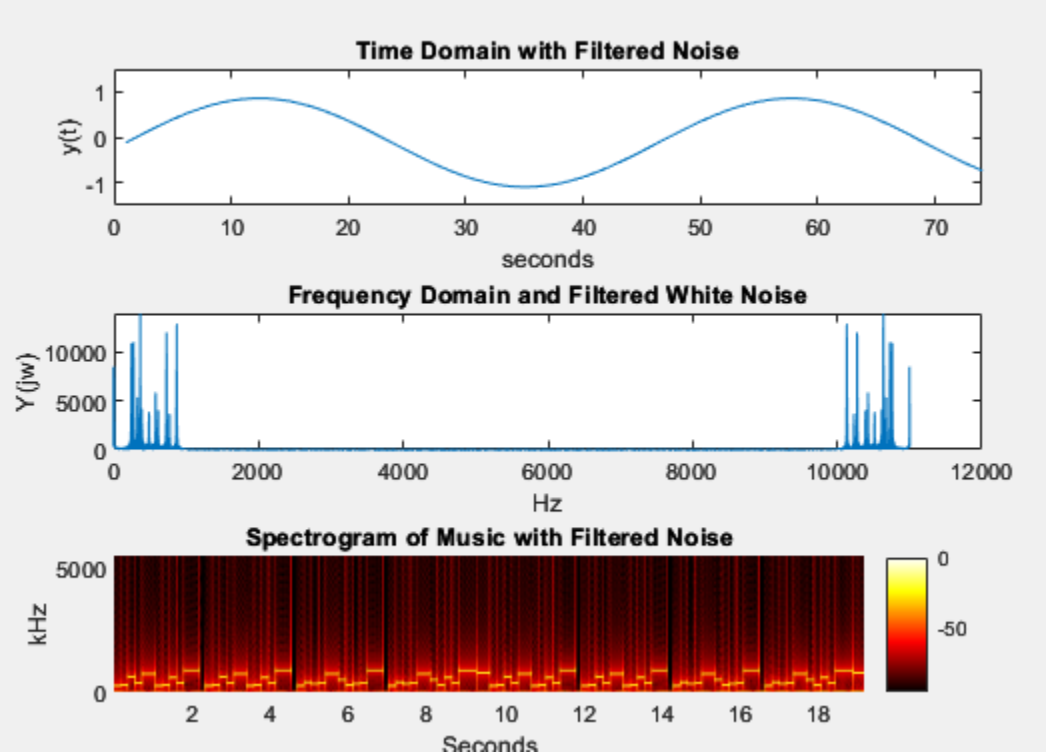

Figure 4 represents the signal with noise being put through a filter. It is important to observe that the once distorted signal has now become smooth again. The only difference from the original signal lies in the frequency domain where the frequency is shifted to the positive axis. The spectrogram for the filtered signal is looking close to the original spectrogram.

e. **Figure 5: Plot and Analysis of Overlapping Filtered and Unfiltered Signals in Time and Frequency Domain**

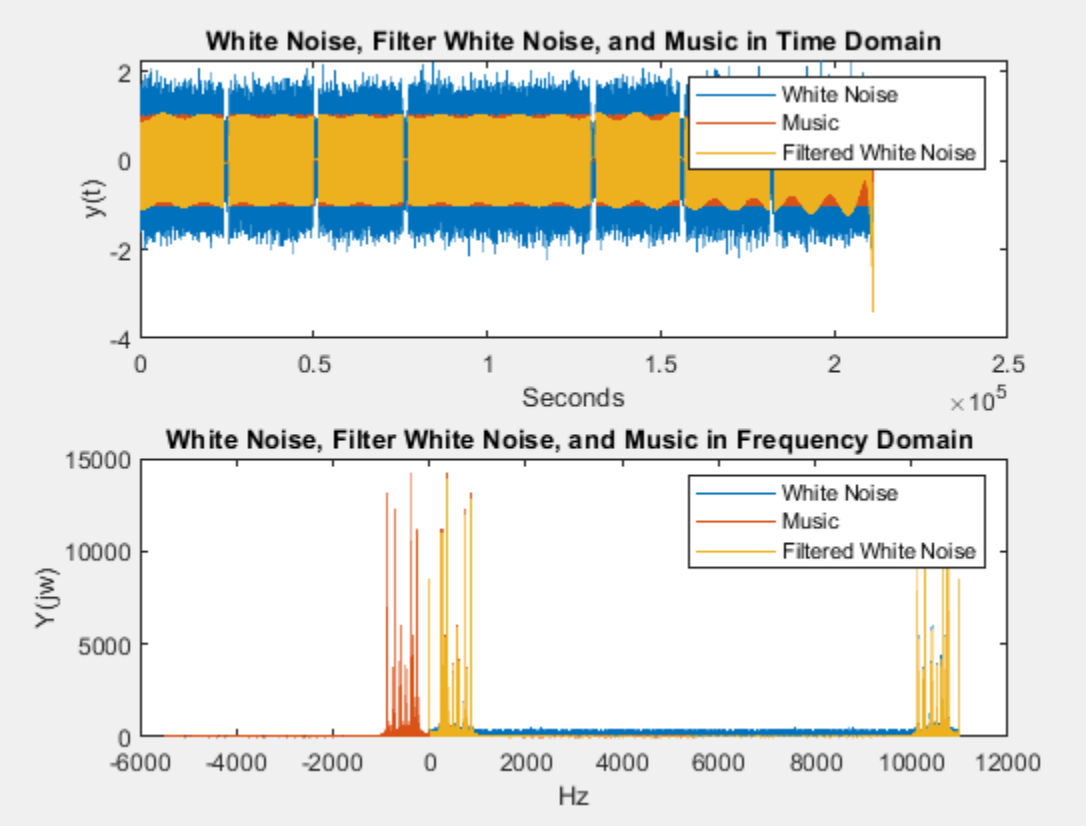

Figure 5 Make note of the breakdown of the signal with white noise. In the time domain we have the white noise peak to peak signal shown in blue, the music wave shown in red, and the filtered wave shown in yellow. By looking at the graph we observe the yellow wave travels in between the red wave, meaning the filter is working properly and onlys filters what we want.

4) **Members Contribution:**

MATLAB Code

- Song Composer: Andres and Andreas
- Graph Titles/General Syntax: Andreas
- Comments: Andres
- Extra Credit Section: Andreas

Report

- Formatting: Andres
- Procedure: Andreas
- Analysis/Explanations: Andreas
- Appendix: Andres and Andreas
- ReadMeFile: Andres

Video Recording

- Recording: Andreas

Video Editing

- Clip Merging: Andres

5) **Appendix and Comments:**

a) **Audio Notes and Generation**

A: 220.00 Hz, 440.00 Hz
B Flat: 466.1638 Hz
C: 523.2511
D: 146.83 Hz, 293.66 Hz, 587.33 Hz
E: 164.81 Hz
E Flat: 233.0819
F: 349.23 Hz
F#: 369.9944 Hz
G: 195.9977 Hz
Space: 0 Hz

```
clear %removes all variables from workspace

%Sampling Frequency(Hz)
FS = 11000
```

Referring to the Appendix, sine wave sample vectors are created using v = sin(2*pi*n*[0:0.000125:1.0])

To create each music note respectively will be shown as follows:

```
a_low = sin(2*pi*220.00*(0:0.00015000:0.55));              %a_low is in the 4th
octave
a_high = sin(2*pi*440.00*(0:0.00015000:0.55));
a_low_q = sin(2*pi*220.00*(0:0.00015000:0.3));             %a_low_q is A in the
4th octave but the note is played quicker
b_fla = sin(2*pi*466.1638*(0:0.00015000:0.55));
c_5 = sin(2*pi*523.2511       *(0:0.00015000:0.75));
d_q = sin(2*pi*146.83*(0:0.00015000:0.3));
d_high = sin(2*pi*293.66*(0:0.00015000:0.3));              %d_high is in the 4th
octave
d_higher = sin(2.15*pi*587.33*(0:0.000125000:0.55));       %d_higher is in the
5th octave
e_q = sin(2*pi*164.81*(0:0.00015000:0.3));
e_q_thr = sin(2*pi*164.81*(0:0.00015000:0.3));             %e_q_thr is E in the
3rd octave played quicker
e_q_fla = sin(2*pi*233.0819*(0:0.00015000:0.3));           %e_q_fla is a E Flat
note played in the 3rd octave
f_sharp_q = sin(2*pi*369.9944*(0:0.00015000:0.3));
f_four = sin(2*pi*349.23*(0:0.00015000:0.3));
g_q = sin(2*pi*195.9977*(0:0.00015000:0.3));
space_16 = sin(2*pi*000.00*(0:0.000150:0.25));


line1 = [d_q, e_q, f_sharp_q, a_low_q, a_high, d_q, e_q, f_four, a_low_q, d_higher,
space_16];
line2 = [d_q, e_q, f_sharp_q, a_low_q, a_high, d_q, e_q, f_four, a_low_q, d_higher,
space_16];
line3 = [d_q, e_q_fla, g_q, a_low_q, a_high, d_high, e_q_thr, g_q, a_low_q, d_higher,
space_16];
line4 = [d_q, e_q_fla, g_q, a_low_q, a_high, d_high, e_q_thr, f_four, a_low_q, c_5,
b_fla];
song = [line1, line2, line3, line4];                  %this section of the code converts the
array of lines into an audio file

sound(song,FS);                                       %remove "%" to play the music
audiowrite('ValorantxSignals.wav', song, FS);         %generates the audio wave
[audioIn, FS] = audioread('ValorantxSignals.wav'); %audio wave input is read
audioOut = stretchAudio(audioIn,9);           %stretches the audio wave
```

In the above section, the song is generated by listing the notes defined above in an array. The array is then converted into a wav file which is named 'ValorantxSignals'.

```
%Fourier transform of audio signal code
[y, FS] = audioread('ValorantXSignals.wav')  %reads y as the audio wave input
Y = fft(y);                                  %this is the fast fourier transform of y
n = numel(y);                                %number of elements
frequency = linspace(-FS/2, FS/2, n+1);      %creates frequency vector
frequency(end) = [];
shiftSpectrum = fftshift(Y);                 %this centers the spectrum
```

For the above code, the Fourier Transform of the audio signal must be defined before making the frequency domain graph. To do this, the audio wave was made to be a frequency vector. During this time, there is a shift in the spectrum as the frequency vector is ending. The zero frequency will be centered in the array.

```
%Declaring and initializing variables for subplots
f1 = figure;
f2 = figure;
f3 = figure;
f4 = figure;
```

The above creates the variables that allow for grouping of the subplots to their respective figures for the sake of organization and to prevent cases of overwriting itself.

```
%Plotting and Labeling the audio signal in time domain code
figure(f1);
subplot(3,1,1);                  %This formats the 3 plots into one window in the
first slot
plot(y);                         %this line plots the song in time domain
axis([0,74,-1,1]);               %the following sets x and y axes of the plot
title('Time Domain');
xlabel('Seconds');
ylabel('y(t)');
```

The above code presents the original audio signal, y, which is the combination of all the sine waves for the music notes and is presented in the graph with the specified axes. This will be located in figure(f1)

```
%Plotting and Labeling the stretched signal in time domain code
figure(f1);                          %configuring subplot window for the original signal
subplot(3,1,2);                      %formats 3 plots into one window in the second slot
t = (0:size(audioOut,1)-1)/FS;  %the time vector created to plot the stretch audio output
plot(t, audioOut);             %plot the stretched audio output in time domain
title('Time Domain');
xlabel('Seconds');
ylabel('y(t)');
```

This section of the code applies to the stretched time domain of the original signal, which enables each separate sine wave to be visible. The x-axis was limited to a range of 0-9 which is the duration of the audio signal and since f1 was defined as the function for the figure, a specific window was generated for the original audio signal in order to prevent it from being overwritten by another plot.

```
%Plotting and Labeling the magnitude in the frequency domain code
figure(f1);
subplot(3,1,3);                          %formats 3 plots into one window in the third slot
plot(frequency,abs(shiftSpectrum));     %this line plots the song in frequency domain
axis([-500,500,0,22000]);                %the following sets x and y axes of the plot
title('Frequency Domain');
xlabel ('Hz');
ylabel('Y(jw)');
```

The code above applies to the creation of frequency and spectrum shift. This was done using the Fourier Transform portion of the code. As indicated, it was plotted into f1

```
%Plotting and Labeling the spectrogram code of the music created
figure;
window=hamming(1024);                          %signal is divided into 1024 segments
noverlap=512;                    %setting the number of overlap samples in the window
nfft=2048;              %generating frequency resolution from comparing different signals
[S,F,T,P]=spectrogram(y,window,noverlap,nfft,FS,'yaxis');          %spectrogram plot
surf(T,F,10*log10(P),'edgecolor','none'); axis tight;view(0,90);       %surface plot
colormap(hot);
colorbar;
set(gca,'clim',[-95 0]);                                    %setting color bar
title('Spectrogram');
xlabel('Seconds')
```

```
ylabel('kHz');
```

The final portion of the initial code plots the spectrogram. This was done by setting up a window that was 1024 segments long allowing for the signal to be divided and overlapped twice to stretch it to fit the window. The variable 'nfft' compared different signals by creating a matching frequency resolution. The colorbar indicates the frequency of each note.

b. Extra Credit Notes

```
%EXTRA CREDIT: Adding White Gaussian Noise (WGN) to music
Gaussian = awgn(song, 20);          %adds WGN to song with a signal-to-noise ratio of 20
audiowrite('WNoise.wav', Gaussian,FS);
%sound(Gaussian,FS);                    %remove the "%" symbol to play WGN with code

%EXTRA CREDIT: Plotting WGN in the Time Domain
figure(f2);                                                 %arranging subplot
window for WGN
subplot(3,1,1);                                 %formatting 3 plots into one window in the
new first slot
plot(Gaussian);
axis([0,200,-1.75,1.75]);                               %the following sets x and y
axes of the plot
title('Time Domain with added White Noise');
xlabel('Seconds');
ylabel('y(t)');
```

White Gaussian noise was added into the music created by a ratio of 10. This worked well for the pitch of the audio before it was filtered. This was used to plot the noise in both the time and frequency domains. The spectrogram was also altered to represent the new signal with the added white noise. Since the f2 was declared it prevents it from being overwritten.

```
%EXTRA CREDIT: WGN in the Frequency Domain
figure(f2);
subplot(3,1,2);                                 %formatting 3 plots into one window in
the new second slot
White_Noise_Freq = (0:length(Gaussian)-1)*FS/(length(Gaussian));        %generating
frequency vector
White_Noise_Signal = abs(fft(Gaussian));                                %shifting the
spectrum
```

```
plot(White_Noise_Freq,White_Noise_Signal);
title('Frequency Domain with added White Noise');
xlabel('Hz');
ylabel('Y(jw)');
%White Noise for Spectrogram
figure(f2);
subplot(3,1,3);                                    %formatting 3 plots into one window
window=hamming(512);                               %signal is divided into 1024 segments
noverlap=256;                                      %setting number of overlap samples in the window
nfft=1024;                                         %generating frequency
resolution to compare different signals
[S,F,T,P]=spectrogram(Gaussian,window,noverlap,nfft,FS,'yaxis');     %spectrogram plot
surf(T,F,10*log10(P),'edgecolor','none'); axis tight;view(0,90);     %surface plot
colormap(hot);
colorbar;
set(gca,'clim',[-95 0]);
title('Spectrogram of Music with Noise');
xlabel('Seconds')
ylabel('kHz');
```

This above code indicates that the white noise's frequency domain was plotted which was done by creating a frequency vector and then shifting the spectrum of the signal. Note that the graph looks similar to the original, with additional distortion in the center of the array which indicates that additional noise had been generated.

```
%Bandpass Filter Code
Filter = bandpass(song,[1, 11000],FS);                    %filtering the white
noise frequency amount that is allowed to pass through
audiowrite('Filter.wav', Filter,FS);
%sound(Filter, FS);                                %remove the "%", to play the song
with the filter

%Filtering Noise in the Time Domain Code
figure(f3);                                        %formatting the subplot window for filtered WGN
subplot(3,1,1);
plot(Filter)
axis([0,74,-1.5,1.5]);
title('Time Domain with Filtered Noise')
xlabel ('seconds')
ylabel('y(t)');
```

```
%Filter for Noise in the Frequency Domain Code
figure(f3);
subplot(3,1,2);
FWhite_Noise_Freq = (0:length(Filter)-1)*FS/(length(Filter));        %creates frequency
vector
FWhite_Noise_Signal = abs(fft(Filter));
plot(FWhite_Noise_Freq,FWhite_Noise_Signal);
title('Frequency Domain and Filtered White Noise');
xlabel('Hz');                    %displays the label for the x axis for the frequency domain
ylabel('Y(jw)');                 %displays the label for the y axis for the frequency domain

%Filter of Noise Spectrogram Code
figure(f3);
subplot(3,1,3);
window=hamming(512);                                    %signal will be divided in to
into 1024 sections
noverlap=256;                                          %how many of overlaps window
nfft=1024;                                                  %creates frequency
resolution to compare different signals
[S,F,T,P]=spectrogram(Filter,window,noverlap,nfft,FS,'yaxis')   %spectrogram plot
surf(T,F,10*log10(P),'edgecolor','none'); axis tight;view(0,90);       %surface plot
colormap(hot);
colorbar;
set(gca,'clim',[-95 0]);
title('Spectrogram of Music with Filtered Noise');
xlabel('Seconds')                                       %displays the label for the x axis for
the filter spectrum
ylabel('kHz');                                         %displays the label for the y axis for the
filter spectrum
```

The above code is focused on filtering the altered audio signal. The filtering was done using a bandpass filter which declared which frequencies were allowed to pass through into the music, which lowered the amount of white noise that made its way into the audio. The filter created both the time and frequency domain plots as well as the spectrogram plot for the filtered white noise and was put into f3.

```
%Code for Graphs of Filtered and Unfiltered WGN, and the Music Time Domain
figure(f4);
subplot(2,1,1);
```

```
plot(Gaussian);
hold on;
plot(song);
hold on;
plot(Filter);
title('White Noise, Filter White Noise, and Music in Time Domain');
legend('White Noise', 'Music', 'Filtered White Noise');
xlabel('Seconds');                    %displays the label for the x axis for the time domain
ylabel('y(t)');                       %displays the label for the y axis for the time domain

%Code For Graphs of Filtered and Unfiltered WGN, and Frequency domain
figure(f4);
subplot(2,1,2);
plot(White_Noise_Freq,White_Noise_Signal);
hold on;
plot(frequency, abs(shiftSpectrum));
hold on;
plot(FWhite_Noise_Freq,FWhite_Noise_Signal);
title('White Noise, Filter White Noise, and Music in Frequency Domain ');  %title of
graph
legend('White Noise', 'Music', 'Filtered White Noise');
xlabel('Hz');                         %displays the label for the x axis for the frequency domain
ylabel('Y(jw)');                      %displays the label for the y axis for the frequency domain
```

The final part of the extra credit uses previously used lines of code, which was used by the coming time and frequency domain plots. Each of the individual signals was plotted and held onto with the hold on function so that the two would not overlap each other. Finally, a legend was added to distinguish between the three signals presented in the final plot.