NYIT
ELECTRICAL AND COMPUTER ENGINEERING

Spring 2019 Semester

EENG 341 – M01
Signals and Systems
Instructor: Dr. Dong

Time and Frequency Analysis of Signals
Term Project
Report

Group: Jens Daci and Jonathan Coskuner

Due Date: April 28th, 2019
Date of Submission: April 28th, 2019

Table of Contents

Jens Daci and Jonathan Coskuner 4/28/2019

Instructor: Dr. Dong

EENG 341 – Signals and Systems

Time and Frequency Analysis of Signals - Term Project

I. Objective

- To create a music (sound) using MATLAB software by combining different frequencies.
- To plot the time domain of the created sound.
- To plot the frequency domain of the created sound.
- To be able to understand and explain the output of both graphs and relate it to the created sound.
- To plot and understand the spectrogram of the created sound.

Extra:

- To add White Gaussian Noise to the original music.
- To analyze and plot time and frequency domain graphs of the noisy signal.
- To understand and explain the signal-to-noise ratio of the noisy signal.
- To design a filter that removes the noise.
- To analyze and plot time and frequency domain graphs of the filtered signal.

II. Procedure

Essential Requirements

Before starting the assignment and programming, we downloaded the MATLAB software (MATLAB R2017a). A step-by-step installation of the MATLAB software can be found in the Readme.txt file that will be attached to the folder. While the computer is doing the installation, a pop-up window will appear asking if all the toolboxes available are needed to be installed.

For this assignment a toolbox called “Communications Systems Toolbox” is needed. Make sure that all toolboxes are installed to be on the safe side. This toolbox will aid in adding noise to the signal without using the random function.

Even though the following procedure steps work with all MATLAB versions, the syntax of some of the commands might not be the same for older versions. MATLAB’s function catalog can help in identifying how the functions have changed over time.

Part 1. Generating a Signal (Sound)

The following steps are needed to generate a sound using different frequencies (musical notes). All musical notes have their characteristic frequencies that can be found in Appendix A.
An example of a sound creation using MATLAB can be found in “SongGeneration.m” file.

Steps:

1. Open the MATLAB software by clicking on the icon on your Desktop.
2. Click on “New Script” in order to start programming
    a. Make sure that the coding is not done on the “Command Window”.
3. Clear all the previous data that was stored by using the following line:

    clear;

4. Create two variables, one for the the sampling frequency and one for the amplitude of all the signals that will be generated using the following lines:

    FS=8000;

    A=0.5;

    a. Sampling Frequency, or sample rate, is the number of samples per second in a sound. For this assignment, FS=8000 [Hz] was chosen.
    b. Amplitude is the maximum value of a wave. For this assignment, the amplitude was chosen to be 0.5.
5. Create a new variable that will hold one of the waves using the following line:

    *g=A*sin(2*pi*392*(0:0.000125:0.3));*

a. This wave is used as an example. The line produces a sine wave (note "g") with:
    i. Amplitude A = 0.5,
    ii. Frequency *f* = 2*pi*392 [Hz]
    iii. Time t = 0 -- 0.3 [s] → steps: 0.000125 [s]

6. For this assignment, the "Spiderman Theme Song" was composed and the code can be found in Appendix B of this report. In order to produce a music, many of these signals are needed to be combined together using the following lines:

```
line1=[c,DS,glong,f,DS,clong];
line2=[c,DS,g,g,g,f,DS,clong];
line3=[fshort,GS,c_five_octet,AS];
line4=[GS,flong,c,DS,g,g,g,DSshort,c];
```

    a. This template code produces the "Spiderman Theme Song" by putting together all the notes created as waves and storing them in 4 variables (line 1, 2, 3, 4).

7. Combine all these lines together, compose the sound, and save it in your computer, using the following lines:

```
song=[line1,line2,line3,line4];
audiowrite('song.wav', song, FS);
```

    a. The combination of the lines is stored in a variable called "song".
    b. Then the audiowrite() function saves the sound using the sampling frequency set at the beginning of the script.

After following all the steps and writing the code to compose any music, move the mouse cursor to the top of MATLAB's toolbar and click on "Run".

As a result, the MATLAB will run your program and produce the sound. If no errors can be seen in the "Command Window", it means that the program was executed successfully.

Therefore, a file named "song.wav" must be saved in your computer in the folder where the MATLAB file is located. Play the sound and see if it matches to the desired music.

Part 2. Importing and Graphing the Signal (Time and Frequency Domain, and Spectrogram)

The following steps are needed to import the signal created, graph the time and frequency domain as well as the spectrogram.

An example of this analysis using MATLAB can be found in "SongAnalysis.m" file.

Steps:

1. Open a new script by clicking on the "New Script" button on MATLAB's toolbar.
2. Import the sound created in the first part of the procedure by using the following lines:

```
filename = 'song.wav';
[original, Fs] = audioread(filename);
N = length(original);
```

   a. The filename should correspond with the one used in the first part of the procedure
   b. The audioread() function is used to read the signal. For this assignment, "original" is the name of the variable that holds the signal and "Fs" holds the sampling frequency.
   c. N is created to hold the length of the imported signal. Will be needed later.
3. Plot the full time domain of the signal using the following lines:

```
figure(1);
subplot(3,1,1);
t = linspace(0, N/Fs, N);
plot(t, original);
title("Time Domain Graph (Full)");
ylim([-1 1]);
xlabel("Time [s]");
ylabel("Amplitude");
```

   a. The first line creates a new figure for plotting. Second line divides the window into sections (subplots). Lines 3 and 4 are used to plot the signal. Time ranges from 0→ N. Lines 5, 6, 7, and 8 are used to make visual changes to the plot.

4. To plot just a section of the sound in time domain the following line can be added to the code found above:

```
xlim([0.45 0.55]);
```

   a. This line limits the x-values to a certain range. This way a clear and visible wave will be seen in MATLAB's figure window.

5. Plot the frequency domain of the imported sound using the following lines:

```
subplot(3,1,3);
fftSignal = fft(original);
fftSignal = fftshift(fftSignal);
dF = Fs/N;
f = -Fs/2 : dF : Fs/2-dF;
plot(f, abs(fftSignal)/N);
title("Frequency Domain Graph");
xlim([-550 550]);
xlabel("Frequency [Hz]");
ylabel("Magnitude");
```

   a. The first line creates another section for the subplot right under the first two plots from the code before
   b. The fft() function takes the Fourier Transform of the signal.
   c. The fftshift() function shifts the zero-frequency components to the center of the spectrum.
   d. The variable dF stores the small pieces of frequencies (step size).
   e. The fifth line creates a matrix *f* that holds the frequency values for plotting.
   f. The plot() function is used to plot *f* as the x-component and absolute value of fft Signal (magnitude must be plotted for frequency domain) as the y-component.
   g. Lines 7, 8, 9, and 10 are used to make visual changes to the plot.
   h. The x-limit is put on the plot in order to clearly see the range of frequencies used for the sound. The limits can be changed depending on which section needs to be shown on the plot.

6. Plot the spectrogram of the original sound using the following template code:

```
figure(2)
spectrogram(original,256,120,128,1e3,'yaxis');
title("Spectrogram of the Original Sound");
```

   a. A new figure is created in order to distinguish the spectrogram with the other plots.
   b. The spectrogram() function in MATLAB accepts the following parameters:
      i. The first parameter is the signal that needs to be plotted (“original”).
      ii. The second parameter is the sampling size (“256”).
      iii. The third parameter is the number of overlap samples (“120”).
      iv. The fourth parameter is the number of DFT points (“128”).
      v. The fifth parameter is the sample rate (“1000”).
      vi. The sixth parameter puts the frequency in the desired axis (“y-axis”).
   c. The title() function sets the title of the spectrogram to “Spectrogram of the Original Sound”.

After following all the steps and writing the code to analyze the music created, move the mouse cursor to the top of MATLAB’s toolbar and click on “Run”.

As a result of the execution, two windows must show. One window should contain three plots and the other window should contain the spectrogram. On the first window, a plot of the signal in time domain is shown in full and zoomed versions. Also, the first window must contain the plot of the signal in frequency domain.

Now that the graphs of the original sound have been plotted, it is time to add noise to the signal, design a filter, and apply the filter to the noisy signal. All these are done in the next part (Part 3) of the procedure.

Part 3. Adding Noise, Designing a Filter, and Applying the Filter

The following steps are needed to add noise to the signal, graph the time and frequency domain of the noisy signal, and apply the filter to the noisy signal.

An example of the signal filtering using MATLAB can be found in "SongFiltering..m" file.

Steps:

1. Open a new script by clicking on the "New Script" button on MATLAB's toolbar.
2. Import the sound created in the first part of the procedure by using the following lines:

```
filename = 'song.wav';
[original, Fs] = audioread(filename);
N = length(original);
t = linspace(0, N/Fs, N);
```

   a. The explanation to these lines of code are the same as in the ones found in Part 2 of the procedure. Reading the sound file is the same.

3. Add noise to the signal by using the following lines:

```
song_noise = awgn(original, 10, 'measured');
audiowrite('song_noise.wav', song_noise, Fs);
```

   a. The awgn() function is part of the signal toolbox for MATLAB that was needed to be downloaded before starting the program. It adds White Gaussian Noise to the signal and it needs three parameters:
      i. The first parameter is the signal that we wish to add noise on ("original").
      ii. The second parameter is the signal-to-noise ratio ("10 dB"). The signal-to-noise ratio is the ratio of the strength of an electrical signal carrying information to that of interference.
      iii. The third parameter tells the awgn() function to measure the power of the input signal before adding noise.
   b. The audiowrite() function is used to save the file in the computer with a predetermined sampling frequency (signal with noise). In this case the file name is "song_noise.wav".

4. Plot the original and noisy signal in the time domain using the following lines:

```
subplot(4,1,1)
plot(t, [original song_noise]);
ylim([-1 1]);
xlim([5.49 5.52]);
title("Time Domain Graph (Signal with Noise)")
xlabel("Time [s]");
ylabel("Amplitude");
legend("Original Signal", "Noisy Signal");
```

   a. The explanation of the functions used are the same as the ones found in Part 2 of the procedure.
   b. The only difference is that two signals are plotted in order to show the difference between the original and noisy signal.
   c. A legend is used to identify the plotted signals.

5. Plot the noisy signal in the frequency domain using the following lines:

```
subplot(4,1,2);
X_mags = abs(fft(song_noise))/7500;
num_bins = length(X_mags);
plot([0:1/(num_bins/2-1):1], X_mags(1:num_bins/2));
title("Frequency Domain Graph (Signal with Noise)");
xlabel("Normalized Frequency [\pi*rads/sample]");
ylabel("Magnitude");
ylim([0 1.1]);
```

   a. These lines of code are different from the ones that were used in Part 2 of the procedure. In this case, the frequency is normalized before it is plotted with the help of lines 2 and 3 (X_mags and num_bins). The normalization of the frequency helps in determining the right cut-off frequency for the filter.
   b. The other lines are used to make visual changes to the plot.

6. Design a Butterworth Low Pass Filter using the following lines:

```
[B,A] = butter(20, 0.2, 'low');
song_filtered = filter(B, A, song_noise);
audiowrite('song_filtered.wav', song_filtered, Fs);
```

   a. The butter() function in MATLAB designs a Butterworth Filter with the following specifications:
      i. The first parameter is the order of the filter (“20th order”).
      ii. The second parameter is the cut-off frequency (normalized) of the filter (“0.2”).
      iii. The third parameter is the type of the filter (“Low Pass Filter”).
      iv. The outputs of the butter() function are saved in the variables B and A.
   b. The filter() function applies the already designed filter to a noisy signal. In this case, the Butterworth Low Pass Filter is being applied to the noisy signal.
   c. The audiowrite() function is used to save the file in the computer with a predetermined sampling frequency (filtered signal). In this case the file name is “song_filtered.wav”.

7. Plot the frequency response of the designed filter using the following lines:

```
H = freqz(B, A, floor(N/2));
hold on
plot([0:1/(N/2 - 1):1], abs(H));
legend("FT of Signal", "Response of Filter")
```

   a. The freqz() function in MATLAB produces the frequency response of a digital signal and in this case it is stored in a variable H.
   b. The *hold on* command is used to plot the frequency response in the previous plot without making a new one.
   c. A legend is used to distinguish the plots.

8. Plot the time-domain and frequency domain graphs of the filtered signal.
   a. The code for plotting these two graphs is the same as the one found in Part 2 of the procedure.

After following all the steps and writing the code to analyze the music created, move the mouse cursor to the top of MATLAB's toolbar and click on "Run".

As a result of the execution, one window must show with four graphs. The first graph should show the time-domain graph of both the original and noisy signals. The second graph should contain the frequency domain plot of the noisy signal as well as the frequency response of the filter. The third graph should contain the time domain graph of the filtered signal. The final and fourth graph should show the frequency domain plot of the filtered signal.

III. Implementation and Analysis

Generation of the Sound ("Spiderman Theme Song")

The song that we chose for this project is the Spiderman Theme Song. Before writing the program in MATLAB, we researched the frequencies of all musical notes that can be found in Appendix A. Then, we searched for the piano version of the theme song in order to implement it using waves in MATLAB. The following musical notes (all fourth octet except for last one), found in Table 1 below, were needed for the creation of the song:

Musical Notes	Frequencies (in Hz)
Note C	261.63
Note D#	311.13
Note F	349.23
Note G	392.00
Note G#	415.30
Note A#	466.16
Note C (fifth octet)	523.25

Table 1. Musical notes used for the Spiderman Theme Song and their frequencies

Analysis of the Sound

After generating the sound using the template code found in Part 1 of the procedure, the sound was then imported to MATLAB for analysis.

The imported sound is plotted in the time domain as shown in Figure 1 below. Two graphs are included in the Figure since the sound was plotted in its entirety (14 seconds) and also a portion of it for better visibility of the wave.

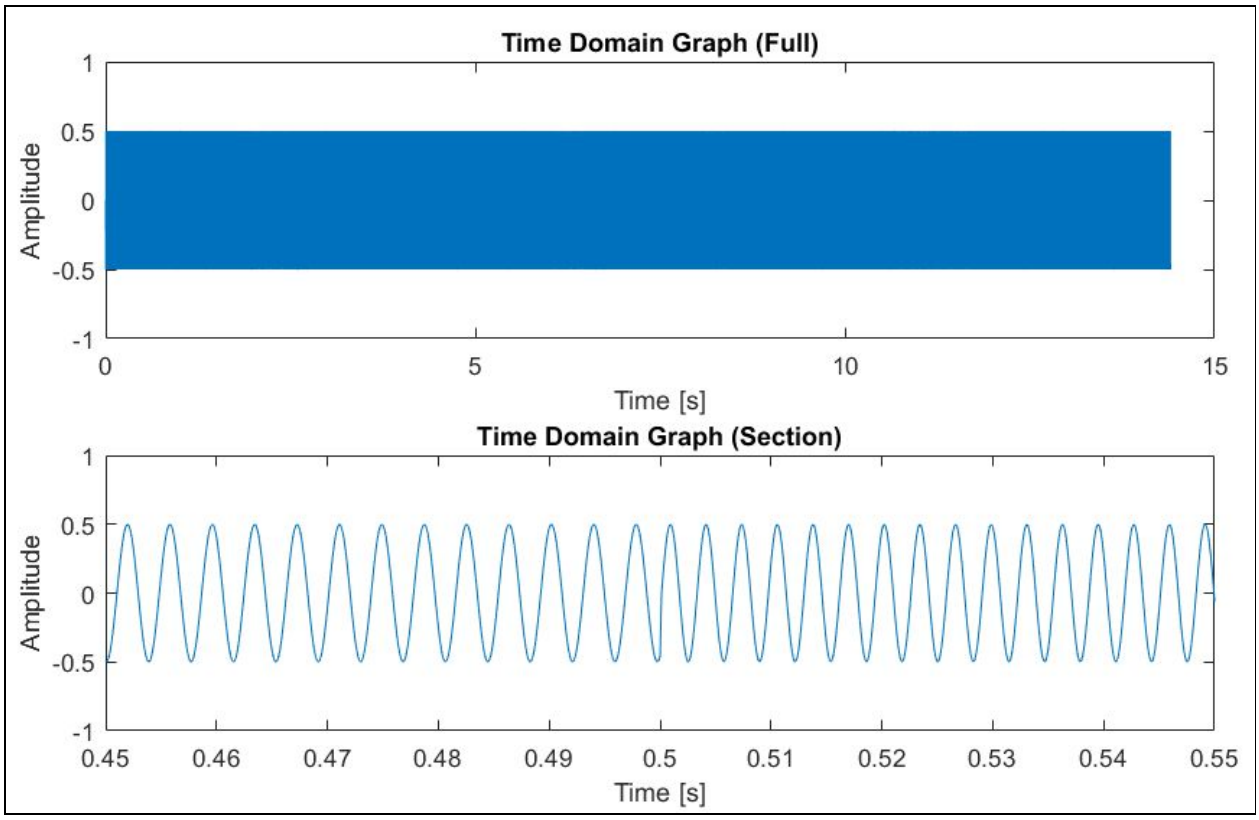

Figure 1. Plot of Signal in the Time-Domain (Full and Section)

As seen in Figure 1, the full plot of the sound does not clearly distinguish between the different frequencies used in the generation of the song. However, in the second plot (section), one can see that from 0.45s up until 0.5s, musical note C (261.63 Hz) is graphed. On the other hand, from 0.5s up until 0.55s, musical note D# (311.13 Hz) is graphed. Unlike in the first plot, the difference in the frequencies between these two musical notes can be clearly seen in the second plot.

Next, the imported sound is then plotted in the frequency domain as shown in Figure 2 below. The plot includes both the positive and negative frequencies available in the imported sound. In theory the negative frequencies must be mirrors of the positive ones. As seen in Figure 2 down below that is the case for the imported sound as well.

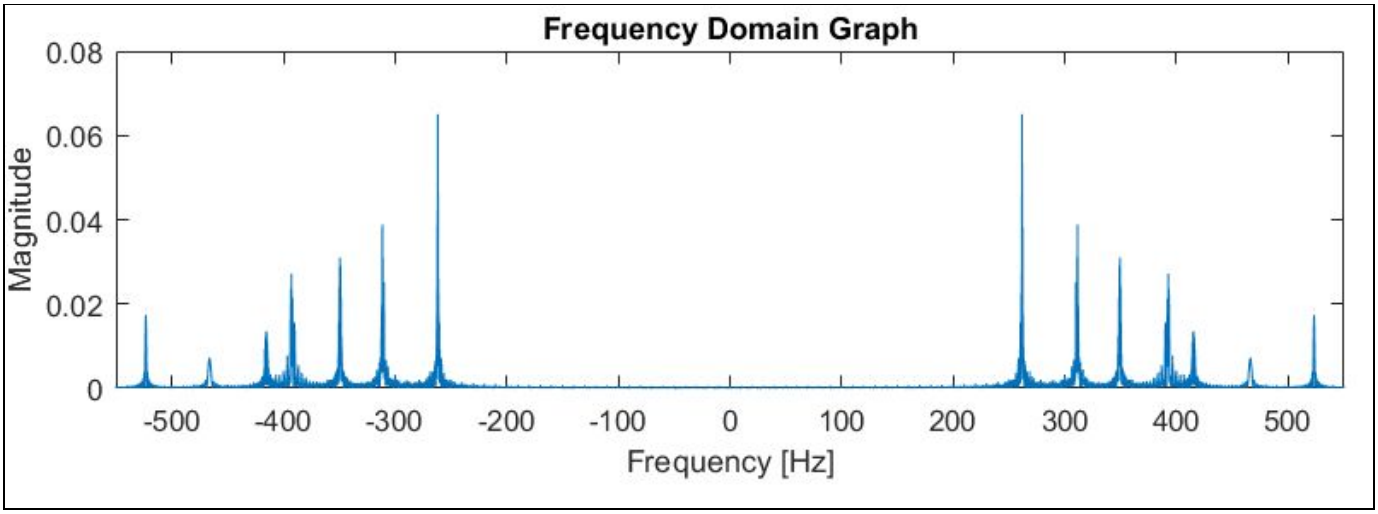

Figure 2. Plot of the Signal in the Frequency Domain

As seen in Figure 2, there are seven frequencies shown (disregarding the mirrored negative ones). The number of frequencies in the plot matches the one that were used for generating the sound.

Looking back at Table 1 above, the order of the frequencies of the musical notes is the same as the order in which they are shown in Figure 2. First we have musical note C, then in turn, musical notes D#, F, G, G#, A#, and finally note C (fifth octet). The frequency values that were picked in the generation of the song MATLAB file perfectly match the ones shown in Figure 2 above.

In addition, the spectrogram of the sound was also plotted in MATLAB. The spectrogram of the song can be seen in Figure 3 down below.

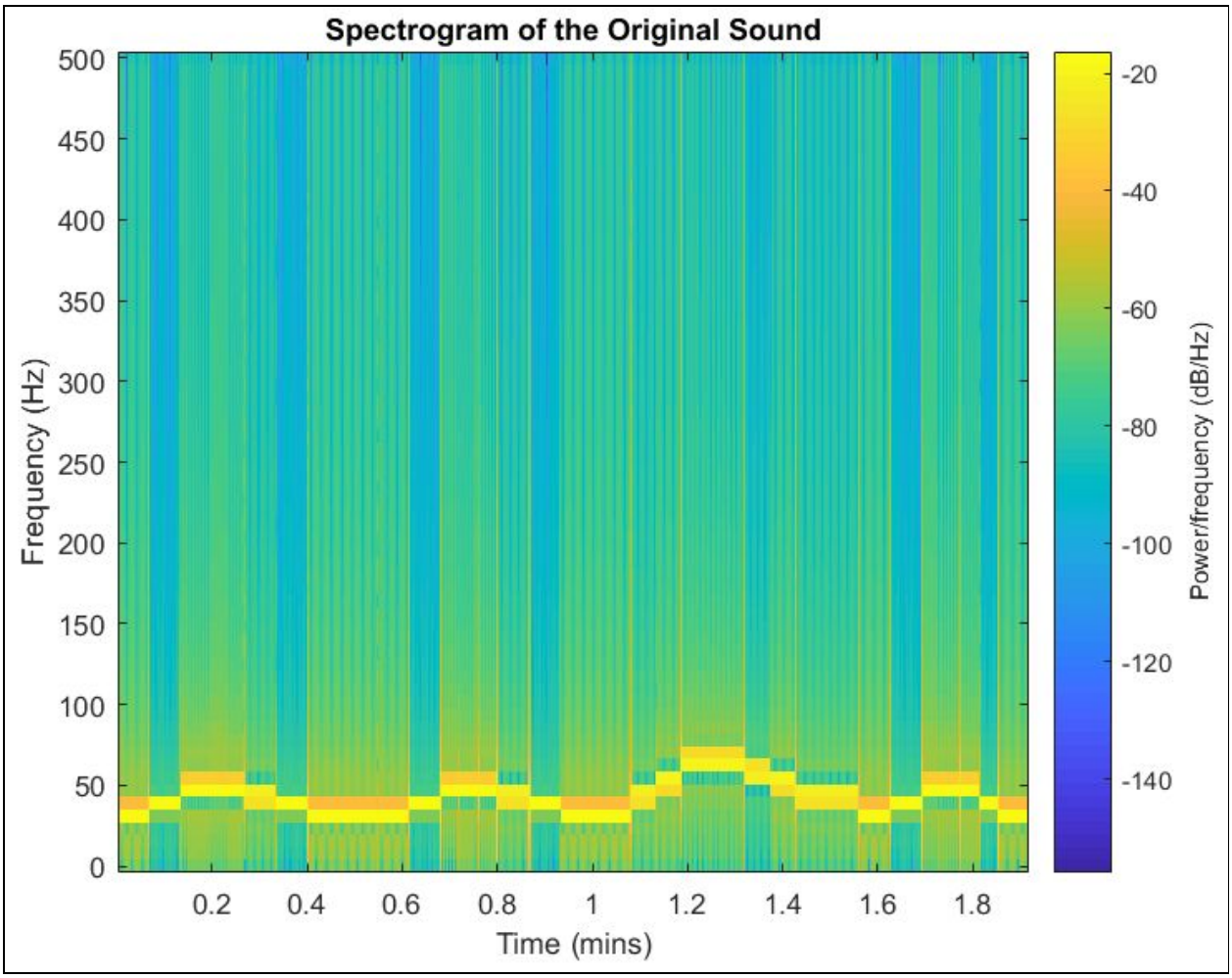

Figure 3. Spectrogram of the Generated Sound

A spectrogram is a visual representation of the spectrum of frequencies in a sound or other signal as they vary with time. Spectrograms map out sound in a similar way to a musical score, only mapping frequency rather than musical notes. Musicians can also use spectral editing to compose and generate sounds that could not be made any other way.

As seen in the spectrogram in Figure 3, yellow color shows the frequencies with the highest intensities. On the other hand, the darker colors (i.e. blue) show the frequencies with the lowest intensities.

Filtering of the Sound

After analyzing the sound both in the time and frequency domain, it is time to add White Gaussian Noise to the signal.

After adding White Gaussian Noise to the signal in MATLAB, both the original signal and the noisy signal were plotted as seen in Figure 4 down below. This was done in order to compare and find the differences in both of these plots.

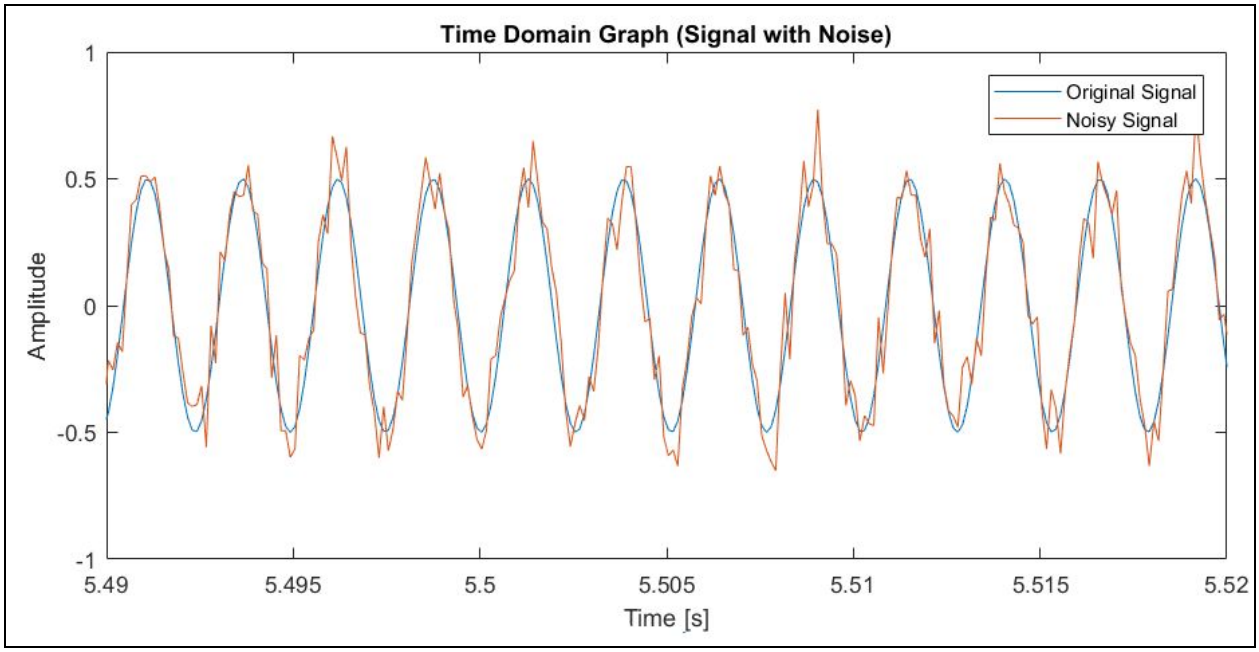

Figure 4. Signal with Noise Plotted in the Time Domain

As seen in Figure 4 above, only a portion of the sound is being shown. This was done in order to show clearly the differences between the two signals. The noisy signal is quite different from the original one. Even though the frequency stays the same, the amplitude of the noisy signal changes randomly as time progresses.

The signal-to-noise ratio is the ratio of the strength of an electrical signal carrying information to that of interference. The signal-to-noise ratio was chosen to be 10 dB. The higher the signal-to-noise ratio is, the clearer the signal is.

In addition to the time-domain plot, the noisy signal was then plotted in the frequency domain. Besides plotting the noisy signal in the frequency domain, a filter was designed in order to filter out the noise.

The frequency domain plot of the signal and the frequency response of the Butterworth Low Pass Filter can be found in Figure 5 below.

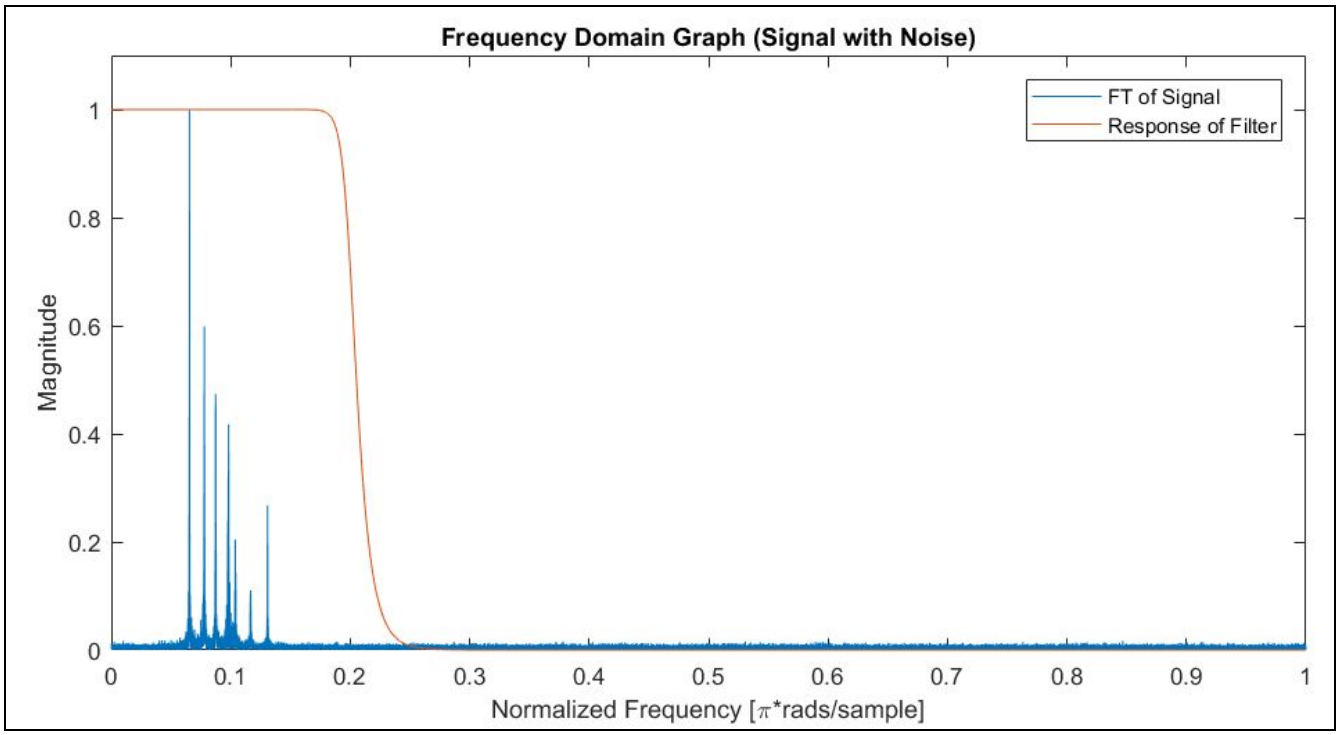

Figure 5. Frequency Domain of Signal and Frequency Response of the Low Pass Filter

As seen in Figure 5 above, the cut-off frequency (normalized) for the Low Pass Filter was chosen to be 0.2. All the frequencies not needed to be filtered out are located below 0.2. Every frequency that is greater than that has been filtered out by the Low Pass Filter.

Even though the signal after being filtered is not exactly the same as the original sound, still some of the noise that was added has been filtered out. All the audio files can be found in the project folder that will be attached with this report.

Finally, the time and frequency domain of the filtered signal was plotted using MATLAB. These plots can be found in Figure 6 down below.

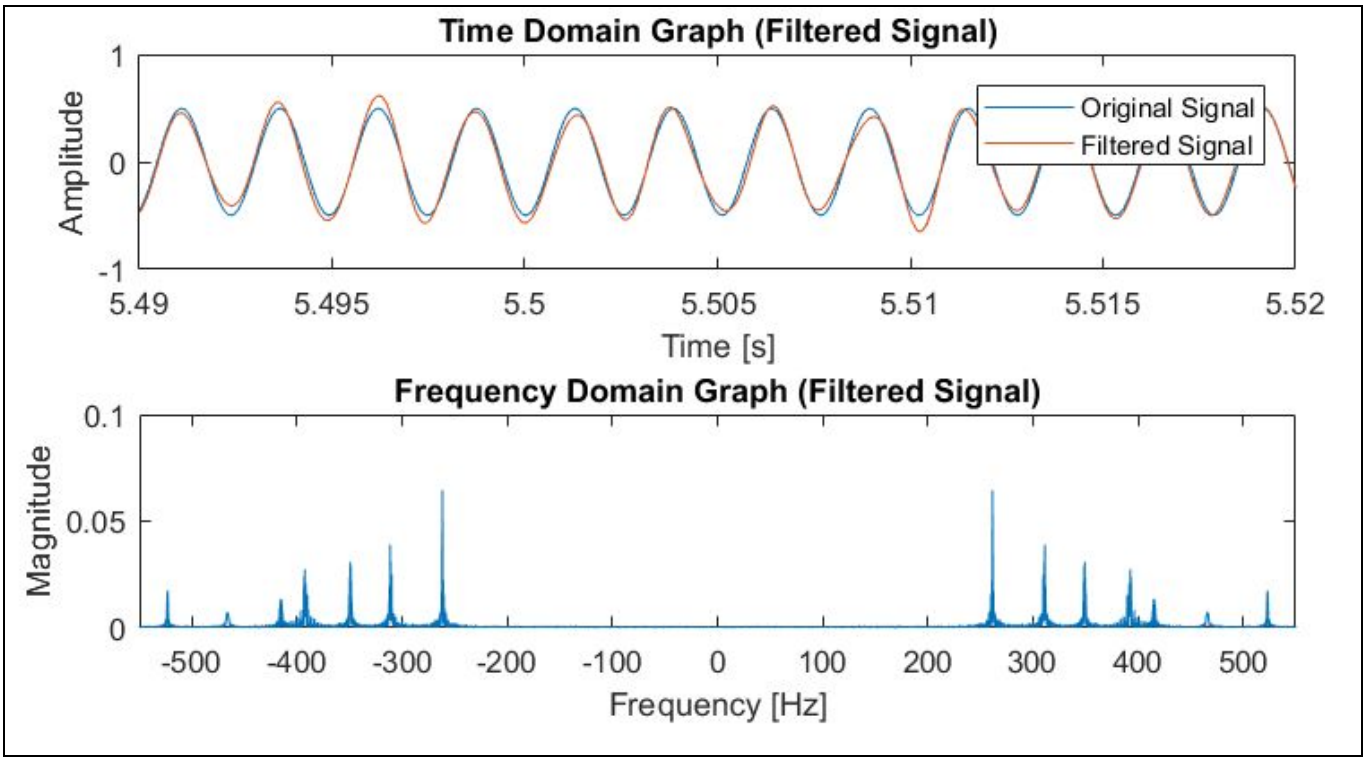

Figure 6. Time and Frequency Domain of the Filtered Signal

As seen in both of the plots in Figure 6 above, the original and the filtered sound are really close in the time and frequency domain.

Even though in some cases the amplitudes do not perfectly match in the time domain, still the filter has filtered out the noise. If the filtered signal were to be listened, it would sound almost the same as the original one.

On the other hand, the frequency domain graph perfectly matches that of the original signal, just as expected. This is because the addition of the White Gaussian Noise did not have an effect on the frequencies of the original sound.

Appendix A

Before making the sound in the MATLAB program found in Appendix A, one first must determine what notes to use for the sound. Below, Table A can be found showing the frequencies of every note available in music. This table is attached in the project folder as an Excel file.

Musical Notes and the Corresponding Frequencies									
Note	Frequency [Hz]		Note	Frequency [Hz]		Note	Frequency [Hz]		Note	Frequency[Hz]
C0	16.35		G2	98		D5	587.33		A7	3520
C#0/Db0	17.32		G#2/Ab2	103.83		D#5/Eb5	622.25		A#7/Bb7	3729.31
D0	18.35		A2	110		E5	659.25		B7	3951.07
D#0/Eb0	19.45		A#2/Bb2	116.54		F5	698.46		C8	4186.01
E0	20.6		B2	123.47		F#5/Gb5	739.99		C#8/Db8	4434.92
F0	21.83		C3	130.81		G5	783.99		D8	4698.63
F#0/Gb0	23.12		C#3/Db3	138.59		G#5/Ab5	830.61		D#8/Eb8	4978.03
G0	24.5		D3	146.83		A5	880		E8	5274.04
G#0/Ab0	25.96		D#3/Eb3	155.56		A#5/Bb5	932.33		F8	5587.65
A0	27.5		E3	164.81		B5	987.77		F#8/Gb8	5919.91
A#0/Bb0	29.14		F3	174.61		C6	1046.5		G8	6271.93
B0	30.87		F#3/Gb3	185		C#6/Db6	1108.73		G#8/Ab8	6644.88
C1	32.7		G3	196		D6	1174.66		A8	7040
C#1/Db1	34.65		G#3/Ab3	207.65		D#6/Eb6	1244.51		A#8/Bb8	7458.62
D1	36.71		A3	220		E6	1318.51		B8	7902.13
D#1/Eb1	38.89		A#3/Bb3	233.08		F6	1396.91			
E1	41.2		B3	246.94		F#6/Gb6	1479.98			
F1	43.65		C4	261.63		G6	1567.98			
F#1/Gb1	46.25		C#4/Db4	277.18		G#6/Ab6	1661.22			
G1	49		D4	293.66		A6	1760			
G#1/Ab1	51.91		D#4/Eb4	311.13		A#6/Bb6	1864.66			
A1	55		E4	329.63		B6	1975.53			
A#1/Bb1	58.27		F4	349.23		C7	2093			
B1	61.74		F#4/Gb4	369.99		C#7/Db7	2217.46			
C2	65.41		G4	392		D7	2349.32			
C#2/Db2	69.3		G#4/Ab4	415.3		D#7/Eb7	2489.02			
D2	73.42		A4	440		E7	2637.02			
D#2/Eb2	77.78		A#4/Bb4	466.16		F7	2793.83			
E2	82.41		B4	493.88		F#7/Gb7	2959.96			
F2	87.31		C5	523.25		G7	3135.96			
F#2/Gb2	92.5		C#5/Db5	554.37		G#7/Ab7	3322.44			

Table A. Musical Notes and their corresponding frequencies
(Source: MTU, https://bit.ly/2hm5oN8)

Appendix B

SongGeneration.m → MATLAB File (Attached in the Project Folder)

```
% Names:       Jens Daci and Jonathan Coskuner
% Course:      Signals and Systems
% Description: This code generates a .wav file (song)

clear;
FS=8000; % sampling frequency
A=0.5; % Amplitude of each wave

% All the notes used in making the song
% Some of the notes have different lengths
g=A*sin(2*pi*392*(0:0.000125:0.3));
glong=A*sin(2*pi*392*(0:0.000125:1.0));
c=A*sin(2*pi*261.63*(0:0.000125:0.5));
clong=A*sin(2*pi*261.63*(0:0.000125:1.1));
c_five_octet=A*sin(2*pi*523.25*(0:0.000125:1.0));
DS=A*sin(2*pi*311.13*(0:0.000125:0.5));
DSshort=A*sin(2*pi*311.13*(0:0.000125:0.3));
AS=A*sin(2*pi*466.16*(0:0.000125:0.4));
f=A*sin(2*pi*349.23*(0:0.000125:0.5));
fshort=A*sin(2*pi*349.23*(0:0.000125:0.4));
flong=A*sin(2*pi*349.23*(0:0.000125:1));
GS=A*sin(2*pi*415.30*(0:0.000125:0.4));

% Generating the lines of music using the notes above
line1=[c,DS,glong,f,DS,clong];
line2=[c,DS,g,g,g,f,DS,clong];
line3=[fshort,GS,c_five_octet,AS];
line4=[GS,flong,c,DS,g,g,g,DSshort,c];

% Generating the song using the lines created
song=[line1,line2,line3,line4];
% Saving the song waveform as .wav file
audiowrite('song.wav', song, FS);
```

Appendix C

SongAnalysis.m → MATLAB File (Attached in the Project Folder)

```
% Names:        Jens Daci and Jonathan Coskuner
% Course:       Signals and Systems
% Description: This code reads the .wav file and plots:
%                  1. Time Domain Graph
%                  2. Frequency Domain Graph
%                  3. Spectrogram

% Reading the file
filename = 'song.wav';
[original, Fs] = audioread(filename);
N = length(original);

figure(1);
set (gcf, 'color', 'w')

% Plotting the Time Domain Graph (full)
subplot(3,1,1);
t = linspace(0, N/Fs, N);
plot(t, original);
title("Time Domain Graph (Full)");
ylim([-1 1]);
xlabel("Time [s]");
ylabel("Amplitude");

% Plotting the Time Domain Graph (section of it)
subplot(3,1,2);
t = linspace(0, N/Fs, N);
plot(t, original);
title("Time Domain Graph (Section)");
ylim([-1 1]);
xlim([0.45 0.55]);
xlabel("Time [s]");
ylabel("Amplitude");
```

```
% Plotting the Frequency Domain Graph
subplot(3,1,3);
fftSignal = fft(original);
fftSignal = fftshift(fftSignal);
dF = Fs/N;
f = -Fs/2 : dF : Fs/2-dF;
plot(f, abs(fftSignal)/N);
title("Frequency Domain Graph");
xlim([-550 550]);
xlabel("Frequency [Hz]");
ylabel("Magnitude");

figure(2)
spectrogram(original,256,120,128,1e3,'yaxis');
title("Spectrogram of the Original Sound");
```

Appendix D

SongFiltering.m → MATLAB File (Attached in the Project Folder)

```
% Names:        Jens Daci and Jonathan Coskuner
% Course:       Signals and Systems
% Description: This code reads the .wav file and does the
               following:
%                 1. Adds noise to the original signal
                     (plots and saves file)
%                 2. Designs filter to cancel the noise
                     (frequency response)
%                 3. Filter is added to the noisy signal
                     (saves file)


% Reading the file
filename = 'song.wav';
[original, Fs] = audioread(filename);
N = length(original);
t = linspace(0, N/Fs, N);
set (gcf, 'color', 'w')

% Adding noise to the original signal
song_noise = awgn(original, 10, 'measured');
audiowrite('song_noise.wav', song_noise, Fs);

% Plotting the Time Domain Graph
% Original and Noisy Graphs
subplot(4,1,1)
plot(t, [original song_noise]);
ylim([-1 1]);
xlim([5.49 5.52]);
title("Time Domain Graph (Signal with Noise)")
xlabel("Time [s]");
ylabel("Amplitude");
legend("Original Signal", "Noisy Signal");
```

```
% Plotting the Frequency Domain Graph (with Noise)
subplot(4,1,2);
X_mags = abs(fft(song_noise))/7500;
num_bins = length(X_mags);
plot([0:1/(num_bins/2-1):1], X_mags(1:num_bins/2));
title("Frequency Domain Graph (Signal with Noise)");
xlabel("Normalized Frequency [\pi*rads/sample]");
ylabel("Magnitude");
ylim([0 1.1]);

% Designing a filter to filter out the noise
[B,A] = butter(20, 0.2, 'low');
song_filtered = filter(B, A, song_noise);
audiowrite('song_filtered.wav', song_filtered, Fs);

% Plotting the frequency response of the filter
H = freqz(B, A, floor(N/2));
hold on
plot([0:1/(N/2 - 1):1], abs(H));
legend("FT of Signal", "Response of Filter")

% Plotting the Time Domain Graph
% Original and Filtered Graphs
subplot(4,1,3)
plot(t, [original song_filtered]);
ylim([-1 1]);
xlim([5.49 5.52]);
title("Time Domain Graph (Filtered Signal)")
xlabel("Time [s]");
ylabel("Amplitude");
legend("Original Signal", "Filtered Signal");
```

```
% Plotting the Frequency Domain Graph
% Filtered Signal
subplot(4,1,4);
fftSignal = fft(song_filtered);
fftSignal = fftshift(fftSignal);
dF = Fs/N;
f = -Fs/2 : dF : Fs/2-dF;
plot(f, abs(fftSignal)/N);
title("Frequency Domain Graph (Filtered Signal)");
xlim([-550 550]);
xlabel("Frequency [Hz]");
ylabel("Magnitude")
```

Appendix E

The following links were used throughout the project and they explain the different functions used in MATLAB in good details.

The awgn() Function:
https://www.mathworks.com/help/comm/ref/awgn.html?s_tid=doc_ta

The filter() Function:
https://www.mathworks.com/help/matlab/ref/filter.html?s_tid=doc_ta

The butter() Function:
https://www.mathworks.com/help/signal/ref/butter.html?s_tid=doc_ta

The spectrogram() Function:
https://www.mathworks.com/help/signal/ref/spectrogram.html?s_tid=doc_ta
https://www.mathworks.com/matlabcentral/answers/265969-how-to-understand-spectrogram-function

The audiowrite() Function:
https://www.mathworks.com/help/matlab/ref/audiowrite.html?s_tid=doc_ta

The audioread() Function:
https://www.mathworks.com/help/matlab/ref/audioread.html?s_tid=doc_ta

The fft() Function:
https://www.mathworks.com/help/matlab/ref/fft.html?s_tid=doc_ta
https://www.mathworks.com/matlabcentral/answers/36430-plotting-the-frequency-spectrum

The fftshift() Function:
https://www.mathworks.com/help/matlab/ref/fftshift.html?s_tid=doc_ta

The freqz() function:
https://www.mathworks.com/help/signal/ref/freqz.html?s_tid=doc_ta
https://www.mathworks.com/help/signal/ug/frequency-response.html

Team Member Contributions

Contributions of each team member can be found below:

Jens Daci:

- MATLAB file that generates the song (SongGeneration.m)
- MATLAB file that filters the noisy signal (SongFiltering.m)
- Procedure part of the report
- Analysis part of the report
- README.txt file

Jonathan Coskuner:

- Music Choice
- MATLAB file that analyzes the generated sound (SongAnalysis.m)
- Objective part of the report
- Implementation part of the report
- Appendices A, B, C, D, E

Group Effort:

- General research on the musical notes and their corresponding frequencies.
- Packaging all files of the project into one zipped folder.