EENG 341 Signals and Systems

Term Project

Fall 2021

Report



Written By:

Rasheed Martin

Emilio Santana-Ferro

Mram Shalabi

# Objective:

In this project, the group had to compose a 5-10 second clip of our choice using different frequencies from The American Standard Pitch as reference. Given these frequencies, the group had to use MATLAB in order to compose the music. After the music is composed, the group has to be able to plot the signal in the time and frequency domain. Additionally, using MATLAB, the group has to create a spectrogram for the input signal and conduct a thorough analysis. The group, as extra credit, can use MATLAB to add white gaussian noise to the input and design a filter to filter out the white noise. Use MATLAB to plot the time domain, frequency domain and the spectrogram of both the input signal with noise and the filtered input signal.

# Procedure:

To commence the project, the group composed a song using different frequencies using the American Standard Pitch as reference. Starting with t = 0s and ending at t =.5s, each vector that was created generates a sine wave of a given frequency, with each sample separated 0.0000625s apart. The total number of samples is.5s/0.0000625 s + 1= 8001. This corresponds to an 8-kilohertz sampling frequency, which is common for voice-grade audio channels.

By merging specific vectors, the group was able to construct the song "Mary Had a Little Lamb." When you compile the code, you'll get a wav file called "mary.wav", which is the song without the input noise.

Then we declared the variables f1, f2, and f3 and initialized them in order to plot our overview charts in three separate plot windows. The overview signal f1 is for the input signal, the overview signal f2 is for the input signal with noise, and the overview signal f3 is for the overview signal with the noise filtered.

The group then used the linspace function which formed a time vector ranging from 0 to 10.0013 (the song's length) with 80010 equally spaced values in between. This was used to plot the input signal in the time domain. We plotted this on the f1 overview and on an individual plot.

We produced a frequency vector using linspace from 0 to 8000 (the sampling frequency) with 1024 even intervals between those ranges. After that, we used the fast fourier transform function to get the fourier transform of the input signal, as well as the absolute value function to get the absolute value of the signal’s fourier transform. We then plotted this on the f1 overview and on an individual plot.

The spectrogram of the input signal was returned using the spectrogram function.We decided to use the hanning window length of 2000 because it provided us with the best frequency to time resolution. We then plotted this on the f1 overview and on an individual plot.

**Extra credit:**

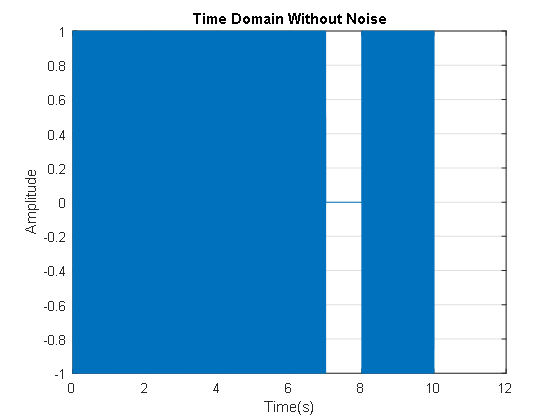
We needed to determine a signal to noise ratio in order to add white noise to the input signal. We decided to use a signal to noise ratio of 10db because it presented a good challenge on filtering it out. After that, we added white gaussian noise to the input signal using the Add White Gaussian Noise function. We utilized the audiowrite function to save the input signal with noise to the user's PC.

Then, in the same manner as before, we had to plot the input signal with noise in the time domain, frequency domain, and spectrogram. We then plotted these on the f2 overview and on their own individual plots.

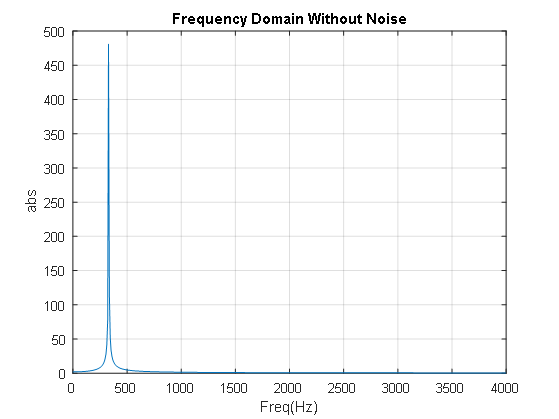
We wanted to create a bandpass filter to filter out low and high frequencies in order to filter out the gaussian noise. As a result, we used the bandpass feature to filter out frequencies outside of the 275Hz-350Hz range.

The input signal with filtered noise is then shown in the time domain, frequency domain, and spectrogram. We then plotted these on the f3 overview and on their own individual plots.

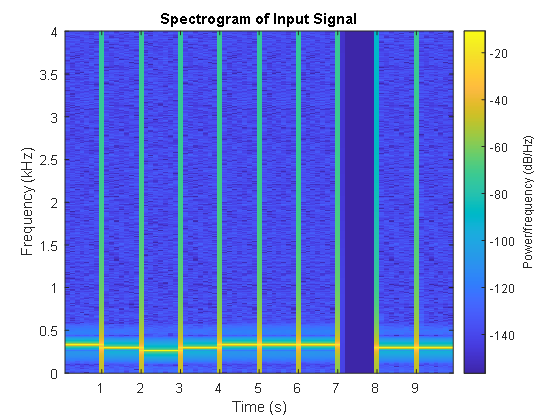
# Graph Representation of Signal:



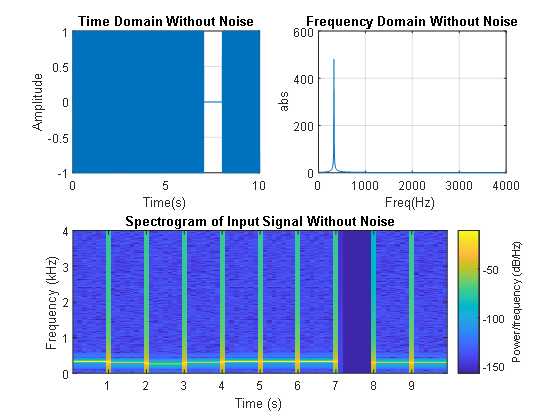
***Fig 1****. Time Domain of Input Signal Without Noise*



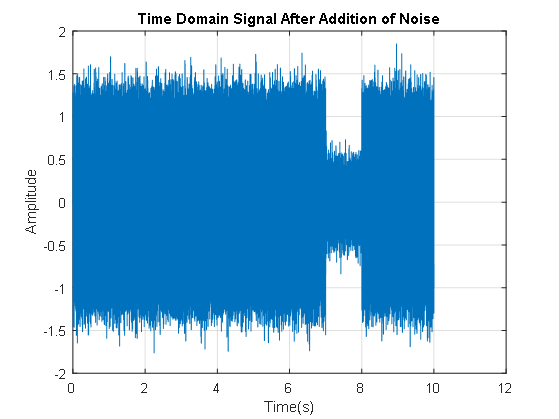
***Fig 2****. Frequency Domain of Input Signal Without Noise*

**

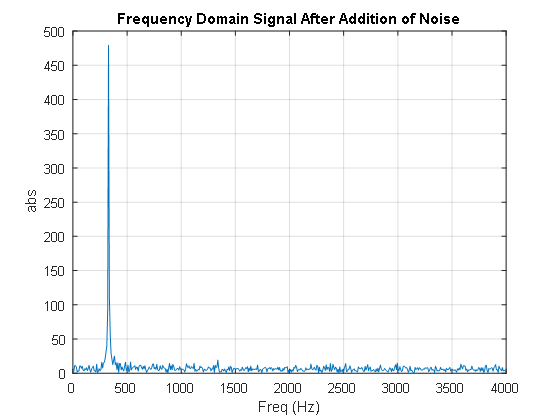
***Fig 3.****. Spectrogram of Input Signal Without Noise*

**

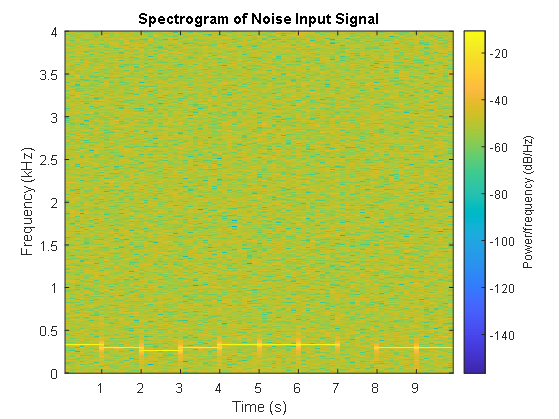
***Fig 4.*** *Overview of Input Signal Without Noise in Time Domain, Frequency Domain and Spectrogram*



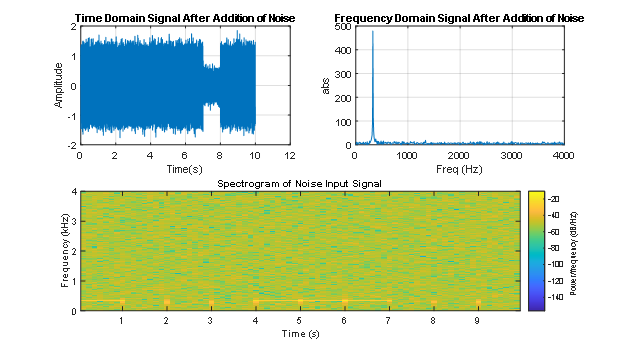
***Fig 5****. Time Domain of Input Signal With Noise*



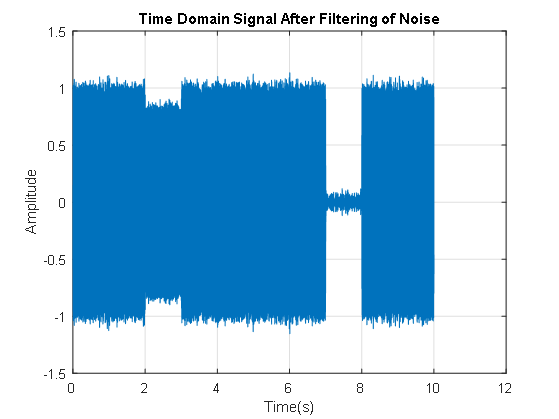
***Fig 6****. Frequency Domain of Input Signal With Noise*



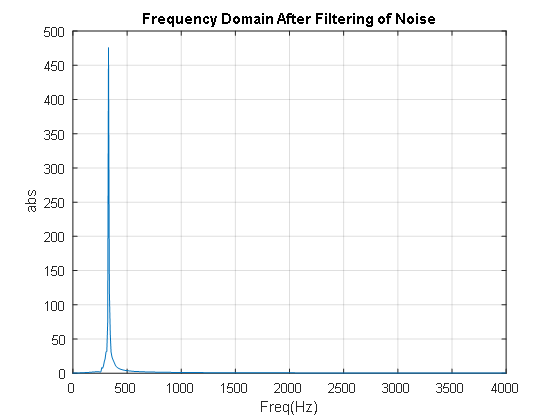
***Fig 7.*** *Spectrogram of Input Signal With Noise*



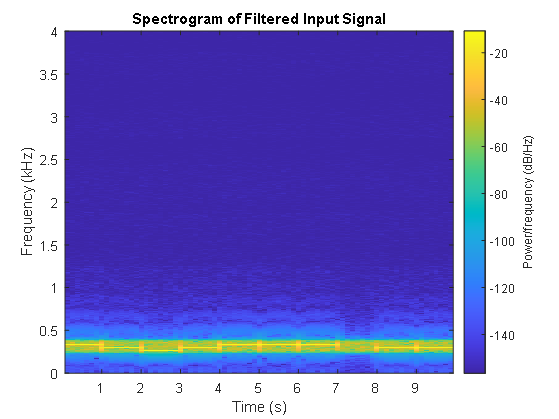
***Fig 8.*** *Overview of Input Signal With Noise in Time Domain, Frequency Domain and Spectrogram*

**

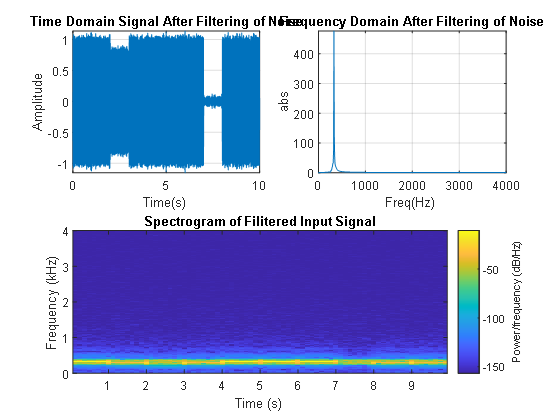
***Fig 9****. Time Domain of Input Signal After Filtering Noise*



***Fig 10****. Frequency Domain of Input Signal After Filtering Noise*



***Fig 11.*** *Spectrogram of Input Signal After Filtering Noise*



***Fig 12.*** *Overview of Input Signal After Filtering Noise in Time Domain, Frequency Domain and Spectrogram*

# Analysis:

In Fig 1, the plot represents the time domain of the input signal. The reason the plot of the input signal is represented as a block is because the input signal is the sum of sine waves of different frequencies. Given the MATLAB figure, when zoomed in, multiple sinusoidal waves are shown. The plot on the x axis is the length of the song and the y axis is the amplitude of the sinusoidal wave. In Fig 2, the graph represents the frequency domain of the input signal. Shown on the plot is a delta function at about 350 Hz. This plot is the fourier transform of the input signal which is sinusoidal. However, the reason there is no negative component is because this plot represents the absolute value of the fourier transform of our input signal. The x axis is the frequency in hertz and the y axis is the magnitude. In Fig 3, the plot represents the spectrogram of the input signal. The spectrogram of the input signal is a very powerful tool as it is able to approximately show the frequency and the order in which each frequency occurred. This is shown in Fig 3 as it is very similar to how the song was composed using different frequencies in a certain order to compose a song. There is a blank section in between 7-8s because at that moment in time, the frequency is zero. The x axis is the time in seconds and the y axis is the frequency in hertz. There is a colorbar on the side of the plot to show the strength of the frequency achieved.

**Extra Credit Analysis**:

In Fig 5, this plot represents the time domain of the input signal when noise is added to the signal. As we can see, the amplitude of the input signal increases and the rate of change within the signal also increases. In between 7 and 8s where there is supposed to be a 0 amplitude, the amplitude has increased drastically. In Fig 6, this plot shows the frequency domain of the input signal after the addition of noise. Analyzing the plot, we can see that there is an increase in the magnitude of the higher and lower frequency of the input signal in the frequency domain. Additionally, the spectrogram, Fig 7, which represents the input signal after noise has been added. In this plot, we are able to see that the strength in the other frequencies has increased due to the color changing from blue to green according to the colorbar. The spectrogram now looks more complicated and harder to read compared to the original spectrogram, Fig 3. On the other hand, analyzing Fig 9, which represents the time domain of the signal when the noise is filtered. After applying the bandpass filter to the input signal with noise, we can see a decrease in the amplitude of the signal. The decrease is significant, but not perfect, therefore there is still some noise in the signal. However, ideally, noise in the signal is fine as long as the signal to noise ratio is high enough so that the noise is not heard. Analyzing the filtered input signal in the frequency domain, Fig 10, we can see that the bandpass filter performed well. The bandpass filter was able to filter out the frequencies higher than 350 Hz and less than 275 Hz. This is shown in the plot as the range of frequencies is significantly smoother than the plot of the input signal with noise, Fig 6. The bandpass filter is more visible in the spectrogram of the filtered input signal, Fig 11 , which shows a low power at any frequency higher than 350 Hz and any frequency lower than 275 Hz. This shows the success in the bandpass filter used and it is even more evident when the audio is played.

# Team Member Contribution:

* MATLAB Coding
  + [Rasheed Martin](mailto:rmarti29@nyit.edu)
* Report
  + [Rasheed Martin](mailto:rmarti29@nyit.edu)
  + Mram Shalabi
  + [Emilio Santana-Ferro](mailto:esanta02@nyit.edu)
* Video
  + Rasheed Martin
* Read Me
  + Emilio Santana Ferro

# Appendix:

**Explanation of the code of the project**

Note in Matlab, a statement ended with “;”means the command will be executed on the

background without being displayed in the command window. Anything after a “%” is

considered comments of the program and will not be executed.

**Listed below is the description of the project and what is required of the group for this project.**

%{

1. Plot the time and frequency domain components of the music you created;

2. Explain the output of your graphs according to your design.

3. Plot the spectrogram of the music you created and explain your findings from the

generated spectrogram.

Extra credit challenges (15 points):

a. Add white Gaussian noise to your music and analyze and plot the output music in both time

and frequency domains. Explain the signal to noise ratio of your output music.

b. Design a filter to remove noise

c. Analyze and plot the music in both time and frequency domain.

%}

close all; %closes any previous windows of matlab (such as plots or figures)

**Listed below is the description of how the music was created.**

**Some content of the music generation in this instruction is from the following source:**

[**http://users.rowan.edu/~shreek/networks1/music.html**](http://users.rowan.edu/~shreek/networks1/music.html)

**Sound waves are created when a waveform is used to vibrate molecules in a material medium at audio frequencies (audible frequency range for humans is 300 Hz ≤ f ≤ 3 kHz). To process signals using digital computers or digital devices (ipod, smartphones, mp3 players), the continuous-time signals have to be digitized (continuous time signals -> discrete time signals). Signals are sampled at discrete times to create waveform. To be able to reconstruct the continuous time signal from the discrete sampled signal, the**

**sampling frequency (fs=1/Ts) has to be at least twice the maximum frequency of the signal being sampled. This frequency is also called the Nyquist frequency.**

**A little background on music notes - The American Standard Pitch for each of these notes is:**

**C5 = 523.25 Hz**

**D5 = 587.33 Hz**

**E5 = 659.25 Hz**

**ED = 0 Hz (arbitrary frequency)**

%% Creating Music

FS=8000; % sampling frequency

T = 1/FS; % Sampling period

**Listed below are the vectors created which each contain a sample of a sine wave. Each vector generates a sine wave of a certain frequency, starting at t = 0s, to t = .5s, with each sample spaced 0.0000625s apart. In Matlab pi = 𝜋, [0:0.0000625:0.5] represents a vector of values from 0 to 0.5 with each element incremented by a step of 0.0000625. The total number of samples from time 0s to .5s is .5s/0.0000625 s + 1= 8001. This corresponds to a sampling frequency of 8 KHz, which is standard for voice grade audio channel.**

% listed below are the notes used to make the song

c=sin(2\*pi\*523.25\*(0:0.0000625:0.5));

d=sin(2\*pi\*587.33\*(0:0.0000625:0.5));

e=sin(2\*pi\*659.25\*(0:0.0000625:0.5));

ed =sin(2\*pi\*0\*(0:0.0000625:0.5));

**Using the following code below the group was able to compose the song “Mary Had a Little Lamb” by combining the notes above. The code below when compiled will save a wav file titled, “mary.wav”, which is the song without an input noise added.**

%generate 3 lines of music by composing the notes together

line1 = [e, d, c, d, e, e, e, ed];

line2 = [d, d];

song=[line1, line2]; % generate the song

%sound(song, FS); %uncomment to listen to the song on runtime.

audiowrite('mary.wav', song, FS); % output the final music notes as a wave form

**The declaration and initialization of the variables f1, f2, f3 is in order to set up 3 different plot windows to plot our overview plots. f1 is for the overview of the input signal, f2 is for the overview of the input signal with noise and f3 is for the overview signal with the noise filtered.**

f1 = figure;

f2 = figure;

f3 = figure;

**The Matlab code listed below is how the group plotted the composed song in the time domain. The group utilized the audioread function (unnecessarily as we create the signal in the same program) which reads the audio file and returns the input signal and the sampling frequency. Then in order to plot the input signal, the group used the linspace function which created a time vector from 0 to 10.0013(which is the length of the song) with 80010 evenly spaced numbers in between. We plotted this on the f1 overview and on its own individual plot for viewing purposes. Additionally, we made the figure save to the user’s computer so it will constantly update the file when compiled.**

%% time domain

[y, fs] = audioread('mary.wav'); %this will read the music and save the input signal and sampling frequency

t = linspace(0, length(y)/fs, length(y)); % time vector ( from 0 to 10.0013 which is the length of the song and linspace creates 80010 evenly spaced numbers in between; hence a time vector)

figure(f1);

subplot(2,2,1);

plot(t,y);

title("Time Domain Without Noise");

xlabel('Time(s)');

ylabel('Amplitude');

grid on;

% Additional individual plot

figure;

plot(t,y);

title("Time Domain Without Noise");

xlabel('Time(s)');

ylabel('Amplitude');

grid on;

savefig("TimeDomain\_noNoise.fig");

**Listed below is the MATLAB code used to plot the input signal in the frequency domain. In order to do this, we had to establish how many samples we wanted to take. We decided to take 1024 samples because we believed it helped to create an accurate representation of the fourier transform. Then using this number of samples, we created a frequency vector using linspace from 0 to 8000(which is the sampling frequency) with 1024 even spaces between that range. After the creation of the frequency vector, we took the fourier transform of the input signal using the *fast fourier transform* function along with the *absolute value* function which returned the absolute value of the fourier transform of the signal. Then we proceeded to plot this on the f1 overview and on its own individual graph for viewing purposes. Note: The reason the number of samples is in half when plotting is because the human ear cannot hear that high in frequency. Additionally, we made the figure save to the user’s computer so it will constantly update the file when compiled.**

%% freq domain

nfft = 1024; % number of samples

f = linspace(0, fs, nfft); % freq vector(0 to 8000 which is the range of the sampling freq . it will create 1024 even spaces between that range)

Y = abs(fft(y, nfft));

figure(f1);

subplot(2,2,2);

plot(f(1:nfft/2),Y(1:nfft/2)); % The reasons the number of samples is in half because the human ear cannot hear that high in frequency.

title("Frequency Domain Without Noise");

xlabel('Freq(Hz)');

ylabel('abs');

grid on;

% Additional individual plot

figure;

plot(f(1:nfft/2),Y(1:nfft/2)); % The reasons the number of samples is in half because the human ear cannot hear that high in frequency.

title("Frequency Domain Without Noise");

xlabel('Freq(Hz)');

ylabel('abs');

grid on;

savefig("FreqDomain\_noNoise.fig");

**Listed below is the MATLAB code used to create the spectrogram of the input signal. The team used the *spectrogram* function to return the spectrogram of the input signal. We decided to use a hanning window of 2000 in order to get a good balance between the frequency and time resolution. We plotted the spectrogram on the f1 overview and on its own individual graph for viewing purposes.Additionally, we made the figure save to the user’s computer so it will constantly update the file when compiled.**

%% spectrogram

figure(f1);

subplot(2,2,[3:4]);

spectrogram(y, hanning(2000), [], [], fs, 'yaxis'); % plotting of the spectrogram. (Used hanning window 2000 to get a good balance between freq and time resolution)

title('Spectrogram of Input Signal Without Noise');

savefig("OverviewofInputSignal.fig");

% Additional individual plot

figure;

spectrogram(y, hanning(2000), [], [], fs, 'yaxis'); % plotting of the spectrogram.

title('Spectrogram of Input Signal');

savefig("SpectrogramOfNoNoise.fig");

**Extra Credit:**

**Below is the MATLAB code used to add white noise to the input signal. In order to add white noise to the input signal, we had to establish a signal to noise ratio. We decided to go with 10db because it had more noise than signal and presented a good challenge on filtering it out. Therefore, we used the Add *White Gaussian Noise*  function in order to add white gaussian noise to the input signal. We additionally added a *sound*  function in case the user wanted to play the song directly from compile time. We used the *audiowrite* function in order to save the input signal with noise to the user’s computer so they can hear what it sounds like.**

%%

% adding white noise

SNR = 10; % signal to noise ratio ( 10db)

st\_nn = awgn(y, SNR, 'measured'); % used the function "Add white gaussian noise" Look up on help

%sound(st\_nn, fs); % uncomment to hear the input with noise

audiowrite('mary-white-noise.wav', st\_nn, FS);

**Below is the MATLAB Code used to plot the input signal with noise to the time domain, frequency domain and the spectrogram. Please refer to the notes above if there is any confusion on how we plotted the signal in each domain. We plotted this on the f2 overview and on its own individual plot for viewing purposes. Additionally, we made the figure save to the user’s computer so it will constantly update the file when compiled.**

% Plotting in time domain

figure(f2)

subplot(2,2,1);

plot(t, st\_nn)

xlabel('Time(s)');

ylabel('Amplitude');

title('Time Domain Signal After Addition of Noise');

grid on;

% Additional individual plot

figure;

plot(t, st\_nn)

xlabel('Time(s)');

ylabel('Amplitude');

title('Time Domain Signal After Addition of Noise');

grid on;

savefig("TimeDomain\_WithNoise.fig");

% plotting in freq domain

Yx = abs(fft(st\_nn, nfft));

figure(f2);

subplot(2,2,2);

plot(f(1:nfft/2),Yx(1:nfft/2)); % The reasons the number of samples is in half because the human ear cannot hear that high in frequency.

title("Frequency Domain Signal After Addition of Noise");

xlabel('Freq (Hz)');

ylabel('abs');

grid on;

%spectrogram

figure(f2);

subplot(2,2,[3:4]);

spectrogram(st\_nn, hanning(2000), [], [], fs, 'yaxis'); % plotting of the spectrogram. (Used hanning window 2000 to get a good balance between freq and time resolution)

title('Spectrogram of Noise Input Signal');

savefig("OverviewofNoiseInputSignal.fig");

%Additional individual plot

figure;

plot(f(1:nfft/2),Yx(1:nfft/2)); % The reasons the number of samples is in half because the human ear cannot hear that high in frequency.

title("Frequency Domain Signal After Addition of Noise");

xlabel('Freq (Hz)');

ylabel('abs');

grid on;

savefig("FreqDomain\_WithNoise.fig");

% Additional individual plot of spectrogram

figure;

spectrogram(st\_nn, hanning(2000), [], [], fs, 'yaxis'); % plotting of the spectrogram.

title('Spectrogram of Noise Input Signal');

savefig("SpectrogramOfNoise.fig");

**Below is the MATLAB code used to filter out the white gaussian noise in the nosy signal. In order to go about this, we needed to make a bandpass filter to filter out low and high frequencies. Therefore we used the *bandpass* function in order to filter out frequencies that are not between 450Hz-600Hz. For this function we passed through the nosy signal, with the band of frequencies and the sampling frequency.**

%%

% filtering out the white noise

yu = bandpass(st\_nn, [275 350], fs);

**Below is the MATLAB Code used to plot the input signal with filtered noise to the time domain, frequency domain and the spectrogram. Please refer to the notes above if there is any confusion on how we plotted the signal in each domain. We plotted this on the f3 overview and on its own individual plot for viewing purposes. Additionally, we made the figure save to the user’s computer so it will constantly update the file when compiled.**

% plotting in time domain

figure(f3);

subplot(2,2,1);

plot(t, yu);

xlabel('Time(s)');

ylabel('Amplitude');

title('Time Domain Signal After Filtering of Noise');

grid on;

%Additional Individual Plot

figure;

plot(t, yu);

xlabel('Time(s)');

ylabel('Amplitude');

title('Time Domain Signal After Filtering of Noise');

grid on;

savefig("TimeDomain\_WithFilteredNoise.fig");

%sound(yu, fs); %uncomment to play the filtered version of the signal with noise

audiowrite('mary-filitered-white-noise.wav', song, FS);

% plotting in freq domain

yc = abs(fft(yu, nfft));

figure(f3);

subplot(2,2,2);

plot(f(1:nfft/2),yc(1:nfft/2));

title("Frequency Domain After Filtering of Noise");

xlabel('Freq(Hz)');

ylabel('abs');

grid on;

%Additional Individual Plot

figure;

plot(f(1:nfft/2),yc(1:nfft/2));

title("Frequency Domain After Filtering of Noise");

xlabel('Freq(Hz)');

ylabel('abs');

grid on;

savefig("FreqDomain\_WithFilteredNoise.fig");

% spectrogram

figure(f3);

subplot(2,2,[3:4]);

spectrogram(yu, hanning(2000), [], [], fs, 'yaxis'); % plotting of the spectrogram. (Used hanning window 2000 to get a good balance between freq and time resolution)

title('Spectrogram of Filitered Input Signal');

savefig("OverviewofFiliteredInputSignal.fig");

% Additional individual plot of spectrogram

figure;

spectrogram(yu, hanning(2000), [], [], fs, 'yaxis'); % plotting of the spectrogram.

title('Spectrogram of Filtered Input Signal');

savefig("SpectrogramOfFiliteredNoise.fig");