Milestone Two Report

In this milestone, we have implemented a part of the twinkle, twinkle little star song using the NumPy, matplotlib, and soundDevice libraries using Anaconda's Spyder.

Firstly, we started by importing the three libraries as follows: import numpy as np, import matplotlib.pyplot as plt, and import sounddevice as sd. Then, we set the time t to be from 0 to 3 seconds with a sample space of 12 * 1024 as follows: t = np.linspace(0, 3, 12 * 1024).

Thirdly, we set the 3rd Octave (Left hand) frequencies Fi and the 4th Octave (Right Hand) frequencies fi with the frequencies of the 1st 20 notes of the song as follows Fi = np.array([164.81, 164.81, …]) and fi = np.array([261.63, 261.63, …]).

After that, we initial time to zero like that ti = 0.

Then we set the duration of each press of each note as an array of 20 elements as follows: Ti = np.array([0.2, 0.2, …]).

Then we initialize the signal with and initial value zero: signal = 0; After that, we loop through fi, Fi, and Ti using a for loop starting from 0 to 20 exclusive. With each loop we setup the unit step function that we will use to cut the signal for each frequency. Then we add the sinusoidal signals with each new frequency and multiply it by the unit step function, then we increment the initial time ti by the duration Ti plus 0.05. All of this is done as follows:

```
for i in range (0, 20):
u = [t >= ti] * [t <= ti + Ti]
signal = (np.sin(2*np.pi*Fi*t) + np.sin(2*np.pi*fi*t)) * u
ti += Ti + 0.05
```

After the summation process is done, we put the graph in a subplot using the plt.subplot(3, 2, 1) method. Then we plot the graph using the plt.plot(t, signal) methods.

Then we set the number of samples N to be the duration of the song, which is 3, multiplied by 1024.

Following that we set the frequency array f with a range from 0 to 512 with a sample size of the integer division of N / 2 using the np.linspace(0, 512, int(N / 2)) method.

Then we taking the Fourier transform of the original signal and assigning it to x_f using the fft(signal) method, which gives the Fourier Transform of a signal.

Then we take the first N / 2 elements then we take their absolute value to allow for plotting.

Then we multiply it by 2 because the spectrum returned by fft is symmetric about the DC component (f=0).

Then we divide by N because each point of the FFT transform is the result of a sum over a time interval of N samples.

After that we place x_f in the subplot by typing plt.subplot(3, 2, 2) and then we plot the graph using the plt.plot(f, x_f). Note that we are plotting the graph of x_f in the frequency domain which is why we use f instead of t. We also plot it in the second columns which is used to display the frequency domain.

After plotting x_f we get its maximum amplitude by using the np.max(x_f) method and assigning it to max.

Then we create an array, fn, of size two with two random frequencies to be used in generating the noise using the np.random.randint(0, 512, 2) method.

Then we create the noise function to be nt = np.sin(2 * np.pi * fn[0] * t) + np.sin(2 * np.pi * fn[1] * t).

Then we create the signal signaln(t) and assigning it the value of the original signal with the noise function n(t) added to it.

Then we add it to the subplot using the plt.subplot(3, 2, 3) method. After that we plot the graph in the time domain using the plt.plot(t, signalnt) method.

After that we get the Fourier Transform of the signal with noise signalnt using the same way we got the Fourier Transform for the original signal and assigning the value of the transform to x_f2. Then we add it to the subplot using the plt.subplot(3, 2, 4) method then plotting the graph in the frequency domain using the plt.plot(f, x_f2).

Then we get a signal in the frequency domain that contains the noise frequencies only by subtracting x_f from x_f2 and assigning it to toBeCancelled.

Then we instantiate an array called frequencies of size 2 and set its initial values to 0. Then we instantiate two counters; j for the frequencies array and i for the while loop that will traverse the toBeCancelled signal array.

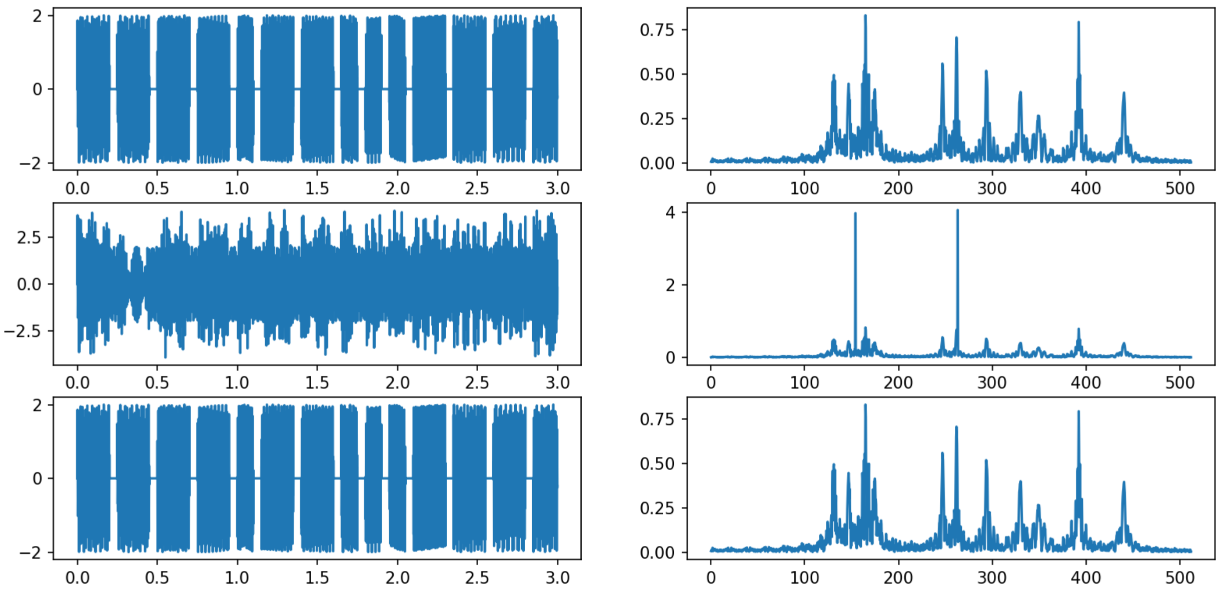
After that we traverse the toBeCancelled and check to see if any of the signals has an amplitude of more than the max, which means that this is a noise frequency; if true we take the frequency f[i], which is the x axis, of the corresponding signal toBeCancelled[i] and store it in frequencies[j]. then we increment j and i by one. i is naturally incremented since it is the loop counter.

After that we create a final signal which is the result of subtracting the noise frequencies from the signal with noise signalnt as follows: signalnt – (np.sin(2 * np.pi * np.round(frequencies[0]) * t) + np.sin(2 * np.pi * np.round(frequencies[1]) * t)) and assigning it to a variable finalSignal.

After that we add it to the subplot using the plt.subplot(3, 2, 5) method and then plotting the graph in the time domain using the plt.plot(t, finalSignal).

Then we get the Fourier Transform of the finalSignal using the same way we got the Fourier Transform of the original Signal and the signal with noise signalnt and assigning it to x_f1. Then we add it to the subplot using the plt.subplot(3, 2, 6) method. Then we plot the graph in the frequency domain using the plt.plot(f, x_f1). Then we show the plotting using the plt.show() method. Then we play the finalSignal, which is the filtered signal, using the sd.play(finalSignal, 3*1024).

Bellow you will find the graphs of the plotting. The right side is the time domain and the left side is the signal's equivalent in the frequency domain:



Report Prepared by Ziad Elsabbagh.