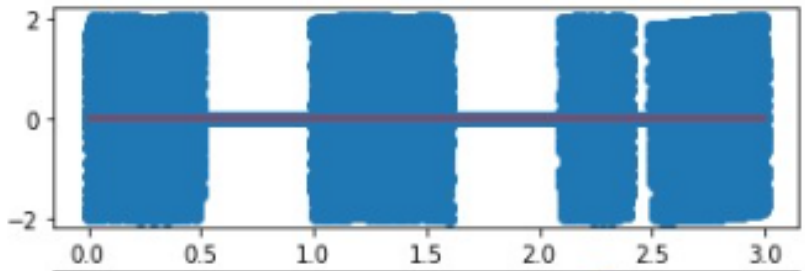
Name: Arwa Hussein ID: 55-0406
Name: Zeina Yasser ID: 55-28086

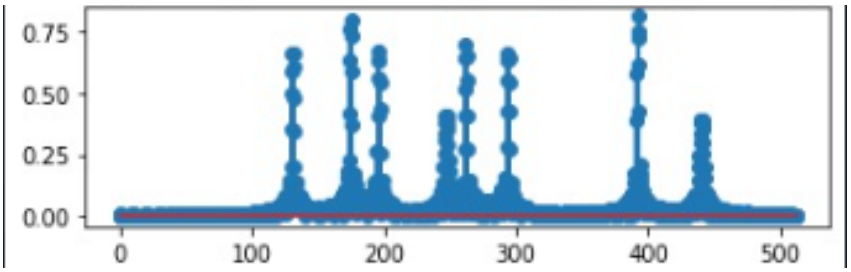
Signals Project

The frequencies that were chosen from the 3rd octave section were : FI= 130.81, F2 = 174.61 F3 =246.93 and F4 = 196. The frequencies that were chosen from the 4th octave to be paired with the 3rd octave frequencies respectively were : fl = 293.66, f2 = 392, f3 = 440 and f4 = 261.63. The number of pairs therefore will be N=4. For each pair of frequencies, one from the 3rd octave and one from the 4th octave, we generate the signal or song with the following equation: x (t) = (sigma from i=1 to N [sin 2Fi t + sin 27 fi t] - 46) - wit - # - TD], where ti is the starting time of the each note and Ti is the duration of each note. For the first note: the starting time (t1) = 0, and its duration (T2) =0.5. For the second note: (t2) = 1, T2 = 0.6. For the third note: (t3)=2.1, T3= 0.3. For the fourth note: t4 = 2.5, T4 = 0.7. For the plotting of the following graph, we use plt.plo(t, x), where the independent variable is t and the dependent variable is x(t). To be able to plot the graph the matplotlib.pyplot library was imported beforehand. To be able to play the song created from the following graph, we imported the sounddevice library. We played the song using : sd.play(x, 3 • 1024). Here is the graph for song in Time Domain:

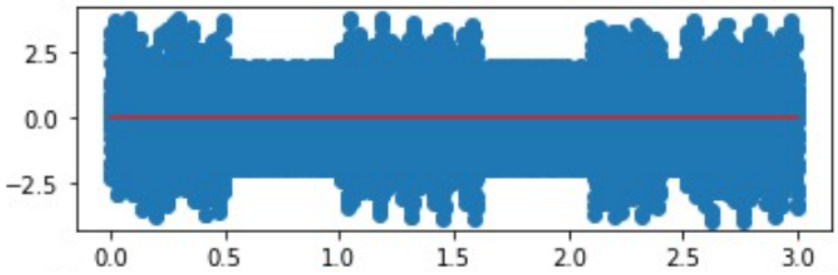


After generating sound , the goal is Noise Cancellation and filtration of frequencies and this was achieved by first, setting the number of sample to the song duration times 1024. The frequency width range is adjusted by using linspace : f = np.linspace(0 , 512 , (int)(N/2)).

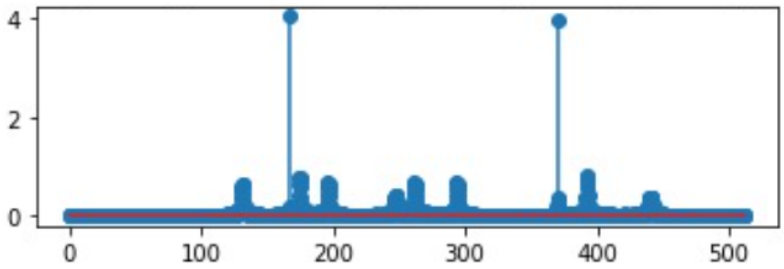
The song is then changed from a time domain signal to a frequency domain signals by using these converting equations: x_f = fft(x) , x_f = 2/N * np.abs(x_f [0:int(N/2)]), and while converting we have to import (from scipy.fftpack import fft) for the conversion to work. Here is the graph for song before additional noise in frequency Domain:



Next is generating additional noise by creating 2 random frequencies by using the equation (np.random.randint(0, 512)) and using them as input to generate noise using the equation (n_t = np.sin(2*fn1*np.pi*t)+np.sin(2*fn2*np.pi*t)) then adding the additional noise to the signal x (xn = x + n_t). Here is the graph for song after adding noise in Time Domain:



The noise signal in time domain will be converted to frequency domain by using the same converting equation: (x_f = fft(x) , x_f = 2/N * np.abs(x_f [0:int(N/2)])). After plotting the signal as you shall see the signal has 2 peaks at extremely high frequencies. Here is the graph for song after adding noise in Frequency Domain:



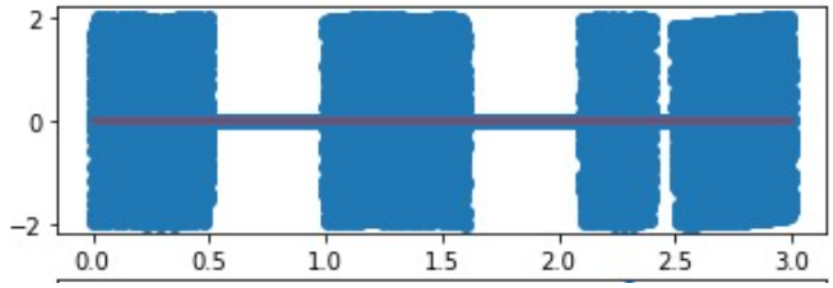
To achieve Noise cancellation, First by searching the indices of the 2 peaks with high frequencies by writing a condition that will output the indices of the frequencies only if the frequency after adding additional noise is bigger than the maximum frequency before adding additional noise. And this is done by using the code: f_array=f[np.where(xn_f>math.ceil(no.max(x_f)))

f1 = int (f_array[0])

f2 = int(f_array[1])

After finding the 2 frequencies we're going to subtract them with the 2 tones to filter the noise using the equation : x_filtered = xn - (np.sin(2*f1*np.pi*t)+np.sin(2*f2*np.pi*t)). To be able to play the song created from the following graph, we imported the sounddeviss library. We played the song using : sd.play(x_filtered, 3* 1024).

Here is the graph of song after filtration in Time Domain:



Here is the graph of song after filtration in Frequency Domain: