SIGNALS AND SYSTEMS THEORY (COMM 401)

SPRING SEMESTER 2022

REPORT OF LAB (PIANO) PROJECT

TUTORIAL NUMBER 16

This Report Contains the description of the implemented code of mailstone 1 and mailstone 2 which generates time and frequency domain signals without, with noise and after noise cancellation , in addition screenshots of the output figure obtained.

Kareem Ahmed Eladl, T-16, 52-5934

Mariam Mohamed Fawzy, T-16, 52-6892

THE IMPLEMENTATION OF THE PROJECT

The imported libraries

numpy, matplotlib.pyplot and sounddevice

The total song playing time to 3 seconds starting from 0 for 12 × 1024 samples.

t = np. linspace (0, 3, 12 ∗ 1024)

The number of pairs of notes *NoOfFreqs*. In our implementation we choose $NoOfFreqs = 4$

For each pair number i,

- Left hand frequency chosen from 3rd octave Fi.

In our implementation we choose $Fi = [246.93, 220, 146.83, 164.81]$ which corresponds to the notes *B3, A3, D3, E3.*

- Right hand frequency chosen from 4th octave fi.

In our implementation we choose $fi = [293.66, 349.23, 493.88, 440]$ which corresponds to the notes *D4, F4, B4, A4.*

- Number of samples N

In our implementation we choose N= 3$(song_duration)$ x 1024

- Frequency axis range f

In our implementation we choose f= np.linspace(0, 512, int(N/2))

- The pressing starting time ti.

In our implementation we choose ti = [0, 0.7, 1.7, 2.3]

- How long you will press both keys Ti.

In our implementation we choose $Ti = [0.7, 1, 0.6, 0.7]$

Chosen from the 3rd octave

Note	Frequency
C3	130.81
D3	146.83
E3	164.81
F3	174.61
G3	196
A3	220
B3	246.93

Chosen from the 4th octave

Note	Frequency
C4	261.63
D4	293.66
E4	329.63
F4	349.23
G4	392
A4	440
B4	493.88

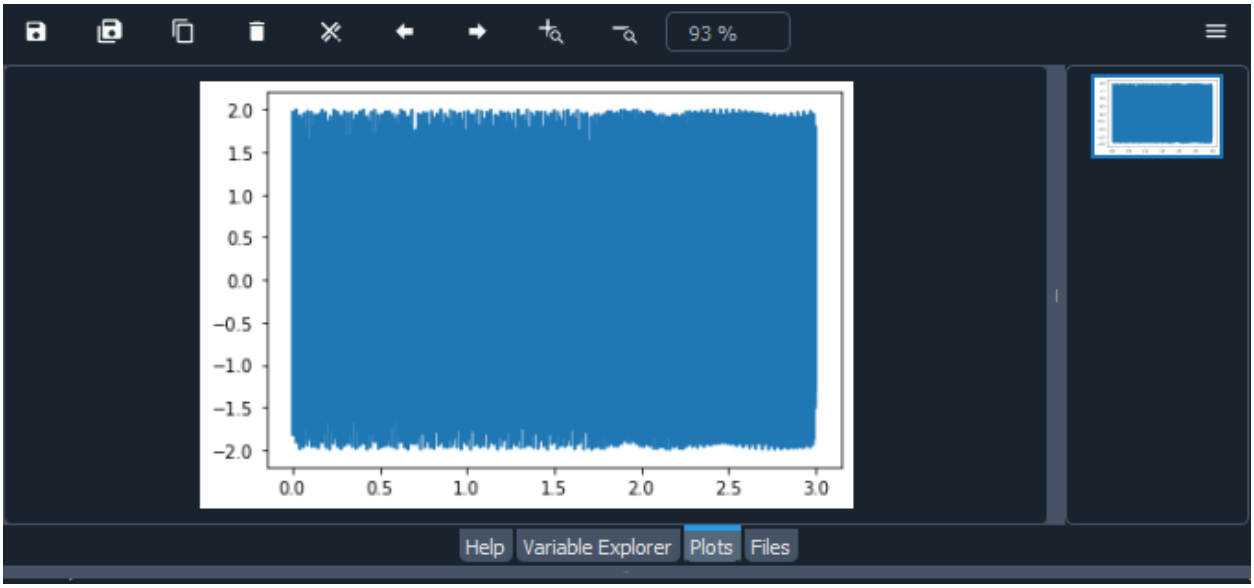
To achieve the previous points for each pair i, we used for loop with range(), we can repeat an action a specific number of times. The range() is a built-in function that returns a range object that consists series of integer numbers, which we can iterate using a for loop.

The signal x(t) is the summation of two parts multiplied by each other *NoOfFreqs* times , the 1st part is two sinusoidal waves added together, angular frequencies of the two sinusoidal waves are calculated using the frequencies *Fi* and *fi.*
2nd part is the shifted unit step function where the unit step functions defines the playing interval.

$$x(t) = \sum_{i=1}^{N} [\sin(2\pi F_i t) + \sin(2\pi f_i t)]\,[u(t - t_i) - u(t - t_i - T_i)]$$

Screenshot of the output figure obtained

The time separation between the notes varies according to our song. We also can't notice the variations in the sinusoidal tones due to the high frequencies.



Implementation of the signal indicating song with noise

- Converting the time signal $x(t)$ to the frequency signal x(f)
- $fn1$ and $fn2$ are two frequencies selected randomly from the 3rd and 4th octaves frequencies list respectively. In our implementation we donated $fn1$ to be an array list that has a size of 2 contacting two random values (frequencies) from 0 to 512
- Noise signal is generated by a child who kept pressing the same 2 random piano keys during the song playing interval

$$n(t) = \sin(2f_{n_1}\pi t) + \sin(2f_{n_2}\pi t)$$

- We defined X$n(t)$ to be the final song after adding the noise signals

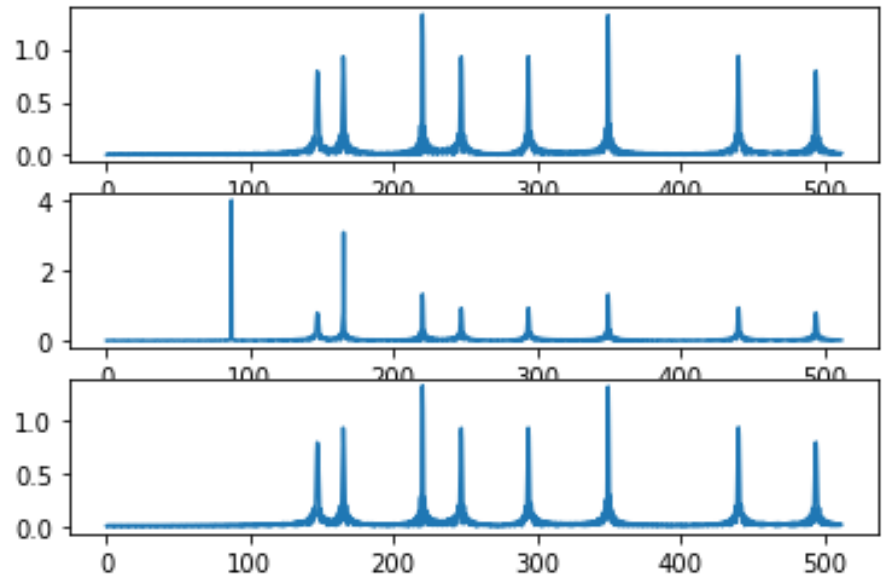
$$x_n(t) = x(t) + n(t)$$

- Converting the time signal X$n(t)$ to the frequency signal X$n(f)$
- Search the frequency signal x(f) for the maximum peak by iteration through the array storing the amplitudes of the signal x(f), when the highest amplitude of the signal x(f) is found, store it in a variable named *"maxSignal"*
- *"maxSignal"* is rounded to the next integer to avoid including the maximum value due to double precision errors
- Fetch n(f) for the peaks that are higher than *"maxSignal"*
- Store the indices at which this occurs in an array called "*indices*" then get the corresponding frequency values of these indices in f
- Round the frequency values at that indices in f
- Filter the noise by subtracting two tones with the two found frequencies

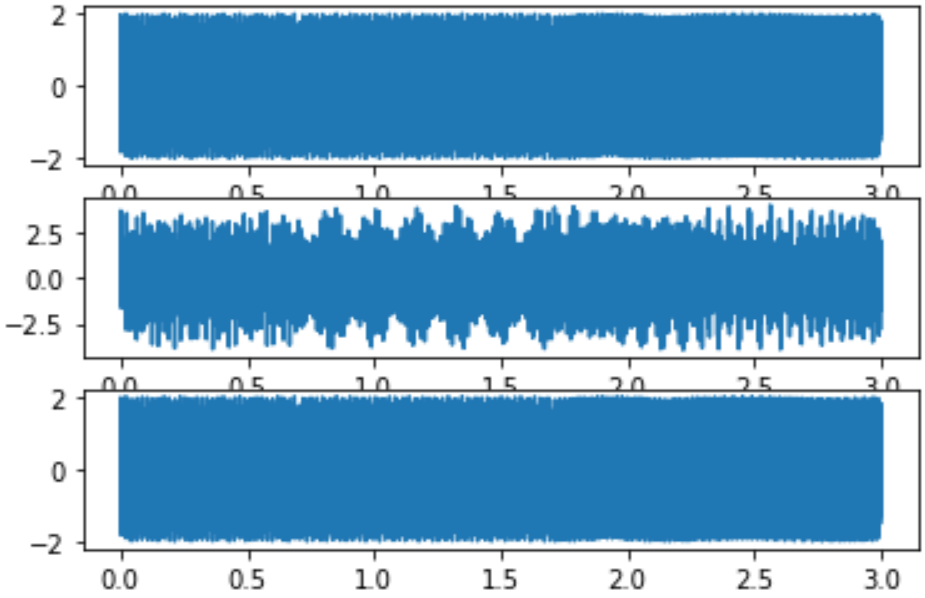
$$x_{filtered}(t) = x_n(t) - [\sin(2f_{n_1}\pi t) + \sin(2f_{n_2}\pi t)]$$

- Converting the time signal X**filtered**(t) to the frequency signal X**filtered** (f)

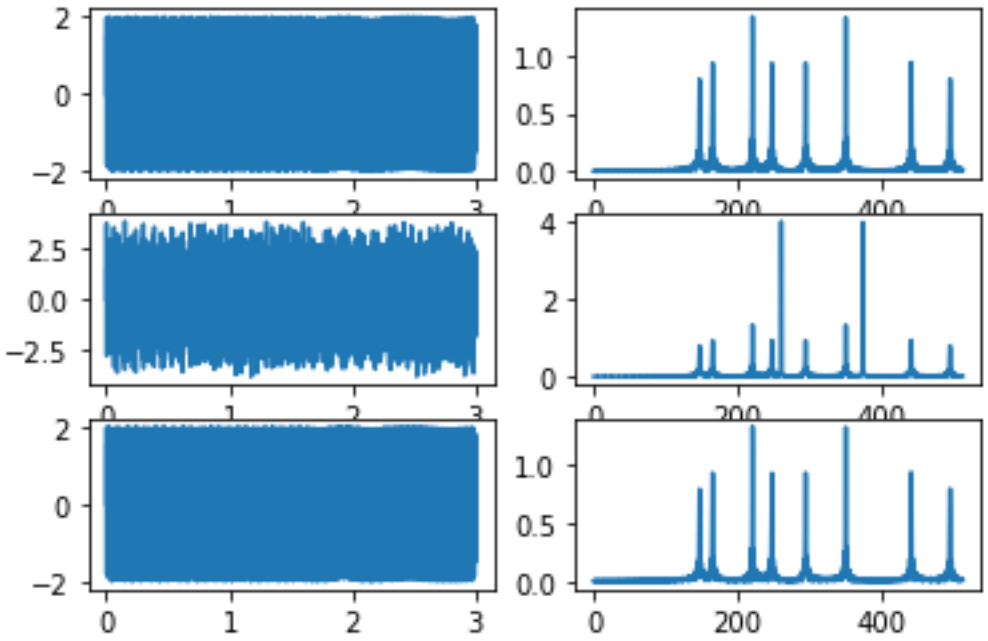
Screenshot of the output figure obtained frequency domain



Screenshot of the output figure obtained time domain



Screenshot of the output figure obtained in our implementation



Six plots obtained in our project divided into two figures

- The first figure contains the time domain signal without, with noise and after noise cancellation.
- The second figure contains the frequency domain signal without, with noise and after noise cancellation.