**Signals Project Report**

***Milestone 1***

A graph with blue rectangles

Description automatically generated

import numpy as np

import matplotlib.pyplot as plt

import sounddevice as sd

import numpy as np

import matplotlib.pyplot as plt

import sounddevice as sd

t = np.linspace(0, 5, 12\*1024)

x1 = np.sin(2 \* np.pi \* 261.63 \* t) \* ((t >= 0) \* (t <= 0.5))

x2 = np.sin(2 \* np.pi \* 261.63 \* t) \* ((t >= 1.0) \* (t <= 1.5))

x3 = np.sin(2 \* np.pi \* 392.00 \* t) \* ((t >= 2.0) \* (t <= 2.5))

x4 = np.sin(2 \* np.pi \* 392.00 \* t) \* ((t >= 3.0) \* (t <= 3.5))

x5 = np.sin(2 \* np.pi \* 440.00 \* t) \* ((t >= 4.0) \* (t <= 4.5))

x = x1+x2+x3+x4+x5

sd.play(x, 3\*1024)

plt.plot(t,x)

1. **Signal Generation**

* Generate a time array `t` spanning 5 seconds with a sampling rate of 12 kHz using `np.linspace(0, 5, 12\*1024)`.
* Generate sinusoidal waves for each musical note using `np.sin(2\*np.pi\*f\*t)`, where `f` represents the frequency of the note.
* Create binary masks for each note's time interval `[ti, ti + Ti]` using `(t >= ti) \* (t <= (ti + Ti))` to generate notes within specified time intervals.

**2.** **Combining Notes**

* Combine the individual note signals `x1`, `x2`, `x3`, `x4`, and `x5` into a single composite signal `x` by adding them together.

**3.** **Audio Playback**

* + Play the composite signal `x` using SoundDevice's `play()` function, specifying a duration of 3 seconds.
  + SoundDevice plays the audio with a sample rate of 3\*1024 samples per second, resulting in a duration of 3 seconds.

**4. Visualization**

* Plot the composite signal `x` against time `t` using Matplotlib's `plot()` function.
* The plot provides a visual representation of the waveform of the composite signal, showing how the individual notes combine to form the overall melody.

***Milestone 2***

**1. Signal Generation**

* + Generate a time array `t` spanning 3 seconds with a sampling rate of 12 kHz using `np.linspace(0, 3, 12\*1024)`.
  + Generate sinusoidal waves for each musical note using `np.sin(2\*np.pi\*f\*t)`, where `f` represents the frequency of the note.
  + Create binary masks for each note's time interval `[ti, ti + Ti]` using `(t >= ti) \* (t <= (ti + Ti))` to generate notes within specified time intervals.

**2. Frequency Domain Analysis**

* Apply the Fast Fourier Transform (FFT) to the signal `x` using `fft(x)` to analyze its frequency components.
* Compute the magnitude spectrum of the FFT result using `np.abs()` and normalize it by dividing by the length of the signal `n` and scaling by 2/n.
* Generate the frequency array `f` corresponding to the FFT result using `np.linspace(0, 512, int(n/2))`.

**3. Noise Injection**

* + Generate two random frequencies `fn1` and `fn2` within the range [0, 512) using `np.random.randint(0, 512, 2)`.
  + Generate sinusoidal noise signals with frequencies `fn1` and `fn2` over the time array `t` using `np.sin(2\*np.pi\*fn\*t)`.
  + Add the noise signals `x\_noise` to the original signal `x` to create a noisy version `x\_noise\_song`.

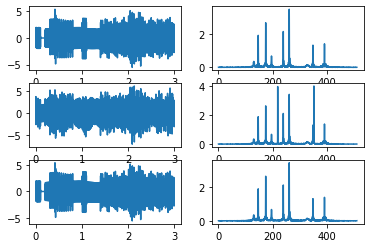
**4. Noise Filtering**

* Subtract the noise frequencies `fn1` and `fn2` from the noisy signal `x\_noise\_song` to filter out the injected noise.

**5. Audio Playback and Visualization**

* Play the filtered signal `xFiltered` using SoundDevice with a duration of 3 seconds using `sd.play(xFiltered, 3\*1024)`.
* Plot the original signal `x`, noisy signal `x\_noise\_song`, and filtered signal `xFiltered` against time `t`.
* Plot the magnitude spectrum of the original signal `x`, noisy signal `x\_noise\_song`, and filtered signal `xFiltered` against frequency `f`.

**Graphs**



**Code**

import numpy as np

import matplotlib.pyplot as plt

import sounddevice as sd

from scipy.fftpack import fft

t = np.linspace(0, 3, 12\*1024)

N = 10

f1 = 392

f2 = 329.63

ti = 0

Ti = 0.1

x = 0

x1 = np.sin(2\*np.pi\*f1\*t)

x2 = np.sin(2\*np.pi\*f2\*t)

x += (x1+x2)\*((t>=ti)\*(t<=(ti+Ti)))

f1 = 130.81

f2 = 392

ti = 0.2

Ti = 0.25

x1 = np.sin(2\*np.pi\*f1\*t)

x2 = np.sin(2\*np.pi\*f2\*t)

x += (x1+x2)\*((t>=ti)\*(t<=(ti+Ti)))

f1 = 196

f2 = 261.3

ti = 0.3

Ti = 0.5

x1 = np.sin(2\*np.pi\*f1\*t)

x2 = np.sin(2\*np.pi\*f2\*t)

x += (x1+x2)\*((t>=ti)\*(t<=(ti+Ti)))

f1 = 174.61

f2 = 392

ti = 0.4

Ti = 0.7

x1 = np.sin(2\*np.pi\*f1\*t)

x2 = np.sin(2\*np.pi\*f2\*t)

x += (x1+x2)\*((t>=ti)\*(t<=(ti+Ti)))

f1 = 174.61

f2 = 261.63

ti = 1.0

Ti = 1.3

x1 = np.sin(2\*np.pi\*f1\*t)

x2 = np.sin(2\*np.pi\*f2\*t)

x += (x1+x2)\*((t>=ti)\*(t<=(ti+Ti)))

f1 = 146.83

f2 = 239.63

ti = 1.4

Ti = 1.7

x1 = np.sin(2\*np.pi\*f1\*t)

x2 = np.sin(2\*np.pi\*f2\*t)

x += (x1+x2)\*((t>=ti)\*(t<=(ti+Ti)))

f1 = 261.63

f2 = 349.23

ti = 2.0

Ti = 2.3

x1 = np.sin(2\*np.pi\*f1\*t)

x2 = np.sin(2\*np.pi\*f2\*t)

x += (x1+x2)\*((t>=ti)\*(t<=(ti+Ti)))

n = 3\*1024

f = np.linspace(0, 512, int(n/2))

x\_f = fft(x)

x\_f = 2/n\*np.abs(x\_f[0:int(n/2)])

fn1, fn2 = np.random.randint(0, 512, 2)

x\_noise = np.sin(2\*np.pi\*fn1\*t) + np.sin(2\*np.pi\*fn2\*t)

x\_noise\_song = x + x\_noise

x\_noise\_songf = fft(x\_noise\_song)

x\_noise\_songf = 2/n\*np.abs(x\_noise\_songf[0:int(n/2)] )

xFiltered = x\_noise\_song - (np.sin(2\*np.pi\*fn1\*t) + np.sin(2\*np.pi\*fn2\*t))

sd.play(xFiltered, 3\*1024)

xFilteredf = fft(xFiltered)

xFilteredf = 2/n\*np.abs(xFilteredf[0:int(n/2)])

plt.figure()

plt.subplot(3,2,1)

plt.plot(t,x)

plt.subplot(3,2,5)

plt.plot(t,xFiltered)

plt.subplot(3,2,3)

plt.plot(t,x\_noise\_song)

plt.subplot(3,2,2)

plt.plot(f, x\_f)

plt.subplot(3,2,4)

plt.plot(f,x\_noise\_songf)

plt.subplot(3,2,6)

plt.plot(f, xFilteredf)

plt.show()

*Done by: Zain Omar 58-0726*

*Menna Hamed 58-14518*