import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

from scipy.interpolate import CubicSpline

import sounddevice as sd

from pydub import AudioSegment

from pydub.playback import play

from playsound import playsound

from scipy.io import wavfile

sampling\_rate\_1X = 1

sampling\_rate\_2X = 2

sampling\_rate\_10X = 10

sampling\_rate\_100X = 100

sampling\_rate\_1000X = 1000

default\_sampling\_rate = 10

# in Hz

carrier\_frequency = 8000

frequency\_deviation = 1000

import sys

sampling\_rate = default\_sampling\_rate

# choose the sampling rate from script argv parameter

if len(sys.argv) > 1:

    match sys.argv[1]:

        case "1x":

            sampling\_rate = sampling\_rate\_1X

        case "2x":

            sampling\_rate = sampling\_rate\_2X

        case "10x":

            sampling\_rate = sampling\_rate\_10X

        case "100x":

            sampling\_rate = sampling\_rate\_100X

        case "1000x":

            sampling\_rate = sampling\_rate\_1000X

        case \_:

            sampling\_rate = default\_sampling\_rate

file\_csv = open("data\_points.txt","r")

lines = file\_csv.readlines()

# extract the total time the signal was emitted

final\_line = lines[-1]

final\_line = final\_line.strip().split(',')

total\_time = final\_line[0]

# sample values from the csv file according to sampling\_rate

lines = lines[0::sampling\_rate]

lines = [line.strip().split(',') for line in lines]

time = [float(line[0]) for line in lines]

values = [int(line[1]) for line in lines]

data = pd.DataFrame({'values': values, 'time': time})

plt.figure(figsize=(10, 10))

plt.plot(data['time'], data['values'], marker='x')

plt.title('Data Chosen')

plt.xlabel('Time')

plt.ylabel('Values')

plt.grid(True)

plt.show()

### spline interpolation on the sets of data received

# perform cubic spline interpolation

cs = CubicSpline(data['time'], data['values'])

# generate a range of time values for plotting the spline

time\_spline = np.linspace(min(data['time']), max(data['time']), 500)

# Use the spline function to get interpolated values

values\_spline = cs(time\_spline)

# plot the original data points and the interpolated spline

plt.figure(figsize=(12, 10))

plt.plot(data['time'], data['values'], 'x', label='Original Data')

plt.plot(time\_spline, values\_spline, label='Cubic Spline')

plt.xlim(min(data['time']), max(data['time']))

plt.ylim(min(data['values']), max(data['values']))

plt.title('Cubic Spline Interpolation of Data Chosen')

plt.xlabel('Time')

plt.ylabel('Values')

plt.legend()

plt.show()

### amplitude modulation

# generate the carrier wave

carrier\_wave = np.cos(2 \* np.pi \* carrier\_frequency \* time\_spline)

# perform the amplitude modulation

am\_modulated\_signal = values\_spline \* carrier\_wave

# perform the frequency modulation

fm\_modulated\_signal = np.cos(2 \* np.pi \* carrier\_frequency \* time\_spline

                             + frequency\_deviation \* np.cumsum(values\_spline) / len(time\_spline))

# combine the am and fm signals

combined\_signal = am\_modulated\_signal + fm\_modulated\_signal

### show the final result

plt.figure(figsize=(12, 18))

plt.subplot(5, 1, 1)

plt.plot(time\_spline, values\_spline, label="Original Signal")

plt.title("Original Signal")

plt.xlabel("Time")

plt.ylabel("Amplitude")

plt.legend()

plt.subplot(5, 1, 2)

plt.plot(time\_spline, carrier\_wave, label="Carrier Signal")

plt.title("Carrier Signal")

plt.xlabel("Time")

plt.ylabel("Amplitude")

plt.legend()

plt.subplot(5, 1, 3)

plt.plot(time\_spline, am\_modulated\_signal, label="AM Modulated Signal")

plt.title("AM Modulated Signal")

plt.xlabel("Time")

plt.ylabel("Amplitude")

plt.legend()

plt.subplot(5, 1, 4)

plt.plot(time\_spline, fm\_modulated\_signal, label="FM Modulated Signal")

plt.title("FM Modulated Signal")

plt.xlabel("Time")

plt.ylabel("Amplitude")

plt.legend()

plt.subplot(5, 1, 5)

plt.plot(time\_spline, combined\_signal, label="Combined AM and FM Signal")

plt.title("Combined AM and FM Modulated Signal")

plt.xlabel("Time")

plt.ylabel("Amplitude")

plt.legend()

plt.subplots\_adjust(hspace=1)

plt.show()

### generate wav files from the instrument notes

# get instrument notes

# 3000 ms the average length of a .wav file

step = int(float(total\_time))//3000

violin\_notes = []

violin\_notes.append("audio/355893\_\_mtg\_\_violin-a3.wav")

violin\_notes.append("audio/355828\_\_mtg\_\_violin-asharp4.wav")

violin\_notes.append("audio/355777\_\_mtg\_\_violin-b5.wav")

violin\_notes.append("audio/355808\_\_mtg\_\_violin-c4.wav")

violin\_notes.append("audio/356138\_\_mtg\_\_violin-csharp6.wav")

violin\_notes.append("audio/355796\_\_mtg\_\_violin-d4.wav")

violin\_notes.append("audio/355992\_\_mtg\_\_violin-dsharp5.wav")

violin\_notes.append("audio/355976\_\_mtg\_\_violin-e4.wav")

violin\_notes.append("audio/356145\_\_mtg\_\_violin-f4.wav")

violin\_notes.append("audio/356135\_\_mtg\_\_violin-fsharp4.wav")

violin\_notes.append("audio/355979\_\_mtg\_\_violin-g3.wav")

violin\_notes.append("audio/356176\_\_mtg\_\_violin-gsharp4.wav")

# beginning low pitch

sample\_rate, main\_data = wavfile.read("audio/356176\_\_mtg\_\_violin-gsharp4.wav")

# append notes to form the output file

for i in  combined\_signal[::step]:

    match True:

        case \_ if i > 769.0:

            sample\_rate, data\_aux = wavfile.read(violin\_notes[0])

        case \_ if 744.0 < i and i <= 769.0:

            sample\_rate, data\_aux = wavfile.read(violin\_notes[1])

        case \_ if 730.0 < i and i <= 744.0:

            sample\_rate, data\_aux = wavfile.read(violin\_notes[2])

        case \_ if 700.0 < i and i <= 730.0:

            sample\_rate, data\_aux = wavfile.read(violin\_notes[3])

        case \_ if 670.0 < i and i <= 700.0:

            sample\_rate, data\_aux = wavfile.read(violin\_notes[4])

        case \_ if 640.0 < i and i <= 670.0:

            sample\_rate, data\_aux = wavfile.read(violin\_notes[5])

        case \_ if 610.0 < i and i <= 640.0:

            sample\_rate, data\_aux = wavfile.read(violin\_notes[6])

        case \_ if 580.0 < i and i <= 610.0:

            sample\_rate, data\_aux = wavfile.read(violin\_notes[7])

        case \_ if 580.0 >= i:

            sample\_rate, data\_aux = wavfile.read(violin\_notes[8])

    main\_data = np.concatenate((main\_data, data\_aux))

# output the resulted file

wavfile.write("combined\_file.wav", sample\_rate, main\_data)

# play the audio

playsound("combined\_file.wav")