CA1 - Voice signal processing

Mahdy Mokhtari
810101515

فاز اول

مقدمه

در این بخش ما میخواهیم نوت موسیقی harrypotter را درست کنیم و عملا فایل صوتی harrypotter_me.wav را به وجود اوریم. دلیل نام گذاری به صورت _me به آن دلیل است که یک فایل با آن نام از قبل به ما داده شده و ما باید نسخه خودمان را درست کنیم و در نهایت برای صحت سنجی با نسخه داده شده مقایسه کنیم. کل کد ما به زبان برنامه نویسی پایتون زده شده است.
در انتهای گزارش **لینک** هایی است که از ChatGPT برای پروژه کمک گرفتم.

آماده سازی محیط

notes.m / notes.py

از آنجایی که به همه چیز باید به زبان پایتون باشد در ابتدا فایل از پیش داده شده notes.m که از پسوند ان هم مشخص است به زبان متلب نوشته شده و درواقع یک لیست است که که note های موزیک هری پاتر در آن نگهداری میشود که هر رشته شامل سه جزء است, "Note name" + "octave" + "Duration in seconds".
"Note name" : اسم نوت است در پیانو که شامل حرف انگلیسی است مانند A, B, C#, G, …
"Octave" : فرکانس پایه برای اکتاو 0 استو بقیه اکتاو ها دو به توان انها میرسد و در فرکانس پایه ضرب شده و فرکانش آن نوت را تشکیل میدهد. $\text{Freq} = \text{base\_freq} \times 2^{\text{octave}}$
"Duration" : مقدار زمانی که آن نوت باید اجرا شود.
آز انجایی که به صورت لیست است به همان ترتیب ما از آن در برنامه میخوانیم نوت ها را فقط برای آماده سازی این لیست را به زبان پایتون تبدیل کردم و در فایلی به نام notes.py که مشخص کننده آن است که فایل پایتون است ذخیره کردم.

```
noteHarryPotter = ['B 4 0.3', 'E 5 0.6', 'G 5 0.2', 'F# 5 0.3', 'E 5 0.6',
                   'B 5 0.4', 'A 5 0.8', 'F# 5 0.8', 'E 5 0.6', 'G 5 0.2',
                   'F# 5 0.3', 'D# 5 0.7', 'F 5 0.4', 'B 4 1.6', 'B 4 0.3',
                   'E 5 0.6', 'G 5 0.2', 'F# 5 0.3', 'E 5 0.6', 'B 5 0.4',
                   'D 6 0.6', 'C# 6 0.3', 'C 6 0.6', 'G# 5 0.3', 'C 5 0.5',
                   'B 5 0.2', 'A# 5 0.3', 'A# 4 0.6', 'G 5 0.3', 'E 5 1.6',
                   'G 5 0.3', 'B 5 0.6', 'G 5 0.3', 'B 5 0.6', 'G 5 0.3',
                   'C 6 0.6', 'B 5 0.3', 'A# 5 0.6', 'F# 5 0.3', 'G 5 0.5',
                   'B 5 0.2', 'A# 5 0.3', 'A# 4 0.6', 'B 4 0.4', 'B 5 1.6',
                   'G 5 0.3', 'B 5 0.7', 'G 5 0.3', 'B 5 0.7', 'G 5 0.3',
                   'D 6 0.7', 'C# 6 0.3', 'C 6 0.8', 'G# 5 0.3', 'C 6 0.6',
                   'B 5 0.2', 'A# 5 0.3', 'A# 4 0.6', 'G 5 0.4', 'E 5 1', 'E 5 1.6']

```

fig 1. notes.py

note_frequencies.py

یک جدول به ما داده شده در صورت پروه که برای مقدار فرکانس هر نوت با اکتاو آن را مشخص میکند. اما همانطور که گفته شده hard code کردن همچین حدولی با همچین ابعادی لازم نیست. (درسته با chatgpt میتوان آن را سریعا به وجود آورد اما باز هم clean code نیست) پس من آمدم فایلی به نام note_frequencies.py درست کردم که درواقع فرکانش های پایه که در اکتاو 0 هستند را نگهداری میکند و بقیه اکتاو ها فرکانس هایشان را با یک حلقه با توجه به رابطه رو به رو به دست می اورد.

$$\text{Freq} = \text{base\_freq} \times 2^{\text{octave}}$$

```python
# Define the notes in octave 0
base_octave_frequencies = {
    "C": 16.352, "B#": 16.352, "C#": 17.324,
    "Db": 17.324, "D": 18.354, "D#": 19.445,
    "Eb": 19.445, "E": 20.602, "Fb": 20.602,
    "F": 21.827, "E#": 21.827, "F#": 23.125,
    "Gb": 23.125, "G": 24.500, "G#": 25.957,
    "Ab": 25.957, "A": 27.500, "A#": 29.135,
    "Bb": 29.135, "B": 30.868, "Cb": 30.868
}

# Calculate the frequencies for other octaves
note_freqs = {}

# Populate the dictionary with frequencies for all octaves (from 0 to 10)
for octave in range(11):
    for note, base_freq in base_octave_frequencies.items():
        note_freqs[f'{note} {octave}'] = base_freq * (2 ** octave)

```

fig 2. note_frequancies.py

مقدار دهی متغیر های اولیه در فایل اصلی main.py

در ابتدا کتابخانه های مورد نیاز و همینطور فایل های نوشته خود را در بالا کد وارد (import) کردیم.

Sample rate : همانطور که خود صورت پروژه گفته ما هر ثانیه 44100 نمونه از سیکنال صوتی خود که دراینجا به صورت سینوسی است باید برداریم و عملا این همان f_s است.

Silence duration : این متغیر مقدار پانیه ای که بین هردو نوت باید سکوت مطلق داشته باشیم (مقدار 0) را مشخص میکند که چون مقدار ان 0.025 ثانیه است به گوس طبیعی انسان زیاد مشخص نمیشود اما اگر مقدار آن را بر فرض 1 ثانیه بگذاریم به وضوح سکوت را و عدم صدا در 1 ثانیه بین دو نوت را متوجه میشویم.

Silence samples : در واقع با ضرب دو متغیر پیشن درهم عملا میگوییم در مدت زمان سکوتمان چند نمونه باید از صفر نمونه برداری کنیم در و در sequance فرکانس ها برای پخش نهایی صدای هری پاتر بگذاریم.

Scale factor : این برای نرمال کردن اندازه (magnitude) هر اندازه صداست که درنهایت از تابع سینوس خارج میشود است تا صدای تولید شده در فایل wav را بتوان شنید.

توضیح منطق اجرا برنامه

یک حلقه میزنیم بروی تمام نوت های موزیک هری پاتر و بعد هر نوت را پردازش میکنیم.
هر نوت در ابتدا به بخش های جزعی تر اسم و مدت زمان در ابتدا تقسیم میشود. با استفاده از جدول در پایتون دیکشنری که در مراحل اماده سازی تهیه کردیم فرکانس نوت مورد نظر را درمی اوریم. سپس با استفاده از تابع سینوسی و تعداد نمونه هایی که باید برداریم باتوجه به مدت زمان ضربدر sample_rate عملا تعدادی نمونه برای سینوس ان فرکانس خاص را در ارایه ای میریزیم و به لیست sequance یی که میخواهیم درست کنیم تا درنهایت به فایل موزیک .wav تبدیل شود ان را اضافه میکنیم. پس از آن نوت عملا تعدادی صفر سکوت که قبلا بدست آمده را در انتهای sequance میگذاریم. بعد ازینکه این کار برای تمامی نوت های هری پاتر انجام شد کل اندازه ها و عملا مقادیرسیگنال های تک تن در یک scale ضرب شده تا مقدار (اندازه) آن قابل شنیدن و استاندارد شود.
در نهایت در فایل 'noteHarryPotter_me.wav' اندازه ها را به کمک کتابخانه scipy نوشته و فایل ایجاد میشود.

```
26  def generate_sine_wave(frequency, duration, sample_rate, amplitude=1.0):
27      t = np.linspace(0, duration, int(sample_rate * duration), endpoint=False)
28      wave = amplitude * np.sin(2 * np.pi * frequency * t)
29      return wave
```

fig 3. sine_wave function

```
36  # Loop through noteHarryPotter
37  for note in noteHarryPotter:
38      note_info = note.split()
39      note_name = note_info[0] + " " + note_info[1]
40      note_names.append(note_name)
41      duration = float(note_info[2])
42
43      frequency = note_freqs.get(note_name)
44
45      if frequency:
46          note_wave = generate_sine_wave(frequency, duration, sample_rate)
47
48          sound_sequence.extend(note_wave)
49
50          # Add silence after the note (25ms of zeros)
51          sound_sequence.extend(silence_samples)
```

fig 4. loop on all harryPotterNotes

```
53  sound_sequence = np.array(sound_sequence) * scale_factor  # scale
54  sound_sequence = sound_sequence.astype(np.int16)
55
56  print("note names: ", note_names)
57
58  # saving .wav file
59  write('noteHarryPotter_me.wav', sample_rate, sound_sequence)
60
```

fig 5. writing & scaling

فاز دوم

مقدمه

در این فاز میخواهیم ابتدا صدای نوت های C5 تا B5 را ذخیره تبدیل فوریه آن را محاسبه و در نهایت ضرایب هارمونی های آن را بدست اوریم. تمام این کار ها به هدف این است که چند نوت به دلخواه خودمان بنویسیم و صدای آن را صاف تر و ملایم تر ذخیره کنیم در فایل noteOptimized.wav .

آماده سازی محیط

در ابتدا یک ترکیب نت های رندوم از بین نوت های C5 تا B5 به صورت ترتیبی در یک لیست در فایلی به نام my_optional_notes میگذاریم. این نوت ها در نهایت باید پردازش هایی روی آنها اتفاق افتد و نتایج آن به صورت ترتیبی از اعداد که اندازه سیگنال در هرلحظه است در فایل نهایی noteOptimized.wav ذخیره شود.
مانند بقیه فایل ها این فایل را نیز در پروژه import کردیم.

```
optional_notes = [
    'E 5 0.4', 'G 5 0.5', 'C 5 0.6', 'F 5 0.7', 'A# 5 0.6', 'C 5 0.8',
    'B 5 0.5', 'A 5 0.4', 'D# 5 0.6', 'C# 5 0.3', 'F 5 0.5', 'G 5 0.6',
    'B 5 0.4', 'A 5 0.7', 'D 5 0.6', 'F# 5 0.5', 'G# 5 0.3', 'C 5 0.6',
    'D 5 0.8', 'E 5 0.5', 'F 5 0.4', 'G 5 0.5', 'A 5 0.6', 'B 5 0.7',
    'C# 5 0.3', 'D 5 0.6', 'F 5 0.5', 'A# 5 0.4', 'G 5 0.6', 'E 5 0.8',
    'D# 5 0.5', 'C 5 0.7', 'G# 5 0.6', 'F# 5 0.4', 'B 5 0.6', 'A 5 0.5'
]

```

fig 6. My_optional_notes.py

سپس آمدم با استفاده از برنامه voice memos, نوت های C5 تا B5 را ذخیره کردم و رکورد کردم که خود این نوت ها را با کمک از سایت Virtual Piano شبیه سازی کردم و بعد آنها را رکورد کردم.

سپس ویس ها را از فرمت m4a به فرمت .wav با کمک از سایت M4A to WAV تبدیل کردم. پس از آن در یک فولدری به نام piano notes ذخیره کردم و مسیر آن را در فایل پایتون اصلی تعریف کردم.

Part 2.1

در ابتدا نوت فایل ها را اسامی شان را ذخیره کردیم در یک لیست و همچنین فولدری که در آن انها را گذاشتیم را نیز تعیین کردیم.

```
# List of note names (you can change them based on your actual files)
note_files = [
    'A#5.wav', 'A5.wav', 'D5.wav', 'D#5.wav', 'E5.wav', 'F5.wav',
    'F#5.wav', 'G#5.wav', 'G5.wav', 'C5.wav', 'C#5.wav', 'B5.wav'
]

# Dictionary to store Fourier Transforms for each note
fft_dict = {}
FOLDER_PATH = './piano notes/'
```

fig 7. Note files names

حال آمدم یک حلقه گذاشتم تا برروی تمامی ویس های ذخیره شده پردازشی انجام دهم.

- ابتدا فایل ویس مورد نظر را میخوانم با کمک از کتابخانه wavfile و با متد read آن . نتیجه این یک آرایه از اندازه سیگنال برحسب زمان نمونه برداری است.
- تبدیل فوریه آن را به کمک تابع آماده کتابخانه numpy بدست می آورم. (FFT: fast fourier transform)
- فرکانس هایمان را در هر نمونه بدست می آورم.

- اندازه (magnitude) هرسیگنال که عدد مختلط است را بدست می اورم.
- در یک دیکشنری با کلید نام فایل آن اطلاعاتی را که از تبدیل فوریه بدست آمده برای آن نوت را ذخیره میکنم.
- سپس نمودار فرکانس-اندازه آن را می کشم برای آن نوت که اطلاعات تبدیل فوریه و فرکانس های آن را در آوردیم.

```
# Process each note file
for note_file in note_files:

    sr, audio = wavfile.read(FOLDER_PATH+note_file)

    if len(audio.shape) == 2:
        audio = audio.mean(axis=1)

    fft_result = np.fft.fft(audio)

    freqs = np.fft.fftfreq(len(fft_result), 1/sr)

    magnitude = np.abs(fft_result)

    fft_dict[note_file] = {
        'fft_result': fft_result,
        'freqs': freqs,
        'magnitude': magnitude
    }

    plt.figure(figsize=(10, 6))
    plt.plot(freqs[:len(freqs)//10], magnitude[:len(freqs)//10])
    plt.title(f'Frequency Spectrum of {note_file}')
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Magnitude')
    plt.show()
```

fig 8. loop to calculate Fourier Transform

پس از اتمام حلقه و پردازش روی تمامی نوت ها, همه نوت ها را از دیکشنری که اطلاعات سیگنال ها را در آن ذخیره کردیم بروی صفحه ترمینال چاپ میکنیم.

Part 2.2

در این بخش در ابتدا آمدم و برای نوت هایی که قرار است ضرایب هارمونیک های آن را در بیاورم فرکانس های پایه آن را نوشتم.

```python
fundamental_frequencies = {
    'A#5.wav': 932.328, 'A5.wav': 880.000, 'D5.wav': 587.330, 'D#5.wav': 622.254,
    'E5.wav': 659.255, 'F5.wav': 698.456, 'F#5.wav': 739.989, 'G#5.wav': 830.609,
    'G5.wav': 783.991, 'B5.wav': 987.767, 'C5.wav': 523.251, 'C#5.wav': 554.365
}

harmonic_coefficients_dict = {}
```

fig 9. fundumental frequencies

سپس یک حلقه زدم روی نوت هایی که از ویس آن ها اطلاعات در اورده بودیم.

- فرکانس پایه نوت مورد نظر را با توجه به دیکشنری تعریفی خودم در مرحله قبل بدست می اورم.
- از آنجایی که 6 هارمونی اول آن را میخواهیم پس ضریب 1 تا 6 آن فرکانس را محاسبه کردم.
- از انجایی که فرکانس ها پیوسته نیستند و گسسته ذخیره شده اند (با rate بالا) نزدیک ترین فرکانس به هرکدام از 6 فرکانس را برای نوت مورد نظر بدست اوردم.
- با توجه به value کلید دیکشنری که اطلاعات آن نوت را ذخیره دارد, اندازه آن (magnitude) آن اندیسی که عملا نزدیک ترین به فرکانس ما بود را میدهم تا اندازه یا همان ضریب هارمونیک آن را بدست اورم و بعد نرمالایز نیز میکنم. برای بقیه هارمونی ها هم نیز به همین صورت حساب میشود.
- آرایه ضرایب هارمونی ها (6 ضریب) را در دیکشنریی که کلید آن اسم نوت فایل است ذخیره میکنم.
- در آخر حلقه ضرایب را چاپ میکنم.

```python
for note_file, data in fft_dict.items():
    # Get the fundamental frequency for the current note
    fundamental_freq = fundamental_frequencies.get(note_file)

    # Calculate the first 6 harmonics
    harmonic_freqs = [fundamental_freq * i for i in range(1, 7)]

    harmonic_coefficients = []
    for h_freq in harmonic_freqs:
        # Find the index of the closest frequency to h_freq in the FFT result
        idx = np.argmin(np.abs(data['freqs'] - h_freq))

        # Calculate the harmonic coefficient (normalized by the fundamental)
        harmonic_coefficients.append(data['magnitude'][idx] / data['magnitude'][0])

    harmonic_coefficients_dict[note_file] = harmonic_coefficients

    # Print the harmonic coefficients for the current note
    print(f"Harmonic Coefficients for {note_file}:")
    print(f"Fundamental: {fundamental_freq} Hz")
    print(f"Harmonics: {harmonic_coefficients}")
    print("-" * 50)
```

fig 10. calculating coefficient of harmonics

Part 2.3

در این بخش میخواهیم ضرایب بدست آمده در مرحله قبل را در فایل excel به نام
'harmonic_coefficients.xlsx' ذخیره کنیم.
برای اینکار ابتدا به لیست ضرایب به اولش اسمم هر نوت را اضافه میکنیم.
سپس دیتا فریم مربوطه را که هر ردیف مربوط به یک نوت ویس و ستون ها نشان دهنده هارمونیک های 1, 2 , ... , 6 هستند را تشکیل میدهیم.
در انتها این دیتا فریم را با کمک متد .to_excel کتابخانه pandas مینویسم تا در پروژه فولدر, این فایل ساخته شود.

```
153  harmonics_data = []
154
155  for note_file, harmonic_coeffs in harmonic_coefficients_dict.items():
156      harmonics_data.append([note_file] + harmonic_coeffs)
157
158  df = pd.DataFrame(harmonics_data, columns=['Note', 'Harmonic 1', 'Harmonic 2', 'Harmonic 3', 'Harmonic
159
160  df.to_excel('harmonic_coefficients.xlsx', index=False)
161
162  ########################
163  ##### part 2.4
164  # Final step: Optimize sound sequence with harmonics and damping
```

fig 11. save excel file

در فایل اکسل تولید شده خواهید که هرچه هارمونیک آن کمتر مثلا هارمونیک اول یا دوم , مقدار ضریب آن بیشتر و این نشان میدهد فرکانس پایه آن نوت بیشتر در صدای نهایی تاثیر میگذارد و نقش مهم تری را ایفا میکند چون مقدار ضریب آن بیشتر است.

Part 2.4

در این بخش میخواهیم یک لیست از نوت ها را اجرا و بخوانیم اما به طوری که این سیگنال های سینوسی تک تن damp شوند تا صدای آن نرم تر و به پیانو شبیه تر بشود.
یک رابطه خیلی مهم داریم که مبنای این بخش است که ابتدا باید این را توضیح دهم و بعد به سراغ توضیح کد می روم.

$$y_{\text{final}}(t) = \left(\sum_{n=1}^{6} A_n \cdot \sin(2\pi n f_1 t) \right) \cdot e^{-\alpha t}$$

هر نوتی که به ما داده میشود و در لیست اجرا است یک فرکانس پایه دارد که با هارمونیک اول آن سیگنال تک تن میشناسیم. پس یعنی برای هر نوت ما این جمع را محاسبه میکنیم و بعد مقدار آن را بعد از نمونه برداری در sequance مقدار ها در زمان برای نوشتن بر روی فایل صوتی نهایی انجام میدهیم.

f1: این همان فرکانس پایه آن نوت است که میخواهیم روی آن این sum را انجام دهیم.
n: این عدد نشان گر هارمونیک nام است و درواقع فرکانس پایه باید در n ضرب سود تا فرکانس آن هارمونیک را به وجود آورد.
A_n: در واقع amplitude یا همان ضریب هارمونیک nام ماست که باید در عبارت سینوسی ضرب شود.
sum: همانطور که در صورت پروژه هم گفته شده باید جمع این سیگنال های سینوسی را در نظر گرفت.
e^(-alpha*t): عبارت دمپ کننده است. با ضرب جمع سینوس ها در این عبارت سیگنال نرم تر شده و اندازه اش در زمان بیشتر کم می شود. (این واقعه به علت شکل نمودار توان منفی e رخ می دهد.)
Alpha : ضریب تابع دمپ کننده.

```
optional_notes = [
    'E 5 0.4', 'G 5 0.5', 'C 5 0.6', 'F 5 0.7', 'A# 5 0.6', 'C 5 0.8',
    'B 5 0.5', 'A 5 0.4', 'D# 5 0.6', 'C# 5 0.3', 'F 5 0.5', 'G 5 0.6',
    'B 5 0.4', 'A 5 0.7', 'D 5 0.6', 'F# 5 0.5', 'G# 5 0.3', 'C 5 0.6',
    'D 5 0.8', 'E 5 0.5', 'F 5 0.4', 'G 5 0.5', 'A 5 0.6', 'B 5 0.7',
    'C# 5 0.3', 'D 5 0.6', 'F 5 0.5', 'A# 5 0.4', 'G 5 0.6', 'E 5 0.8',
    'D# 5 0.5', 'C 5 0.7', 'G# 5 0.6', 'F# 5 0.4', 'B 5 0.6', 'A 5 0.5'
]

```

fig 12. random notes for optimizedNote.wav

برای انجام عملیات برروی هر نوت از لیست بالا یک حلقه زدم روی تمامی نوت های لیست.

- بعد ابتدا مدت زمان و اسم آن را در آوردم .
- بعد از نوت مورد نظر را فایلش را خواندم.
- تبدیل فوریه و ضرایب هارمونیک آن را بدست آوردم مانند بخش های پیشین.

- از اینجا به بعد بخش اصلی است که عملا آمدم موج سینوسی آن را تولید کردم.
- ضرایبب هر هارمونیک را در عبارت متناظر آن ضرب نمودم
- 6 هارمونیک اول آن را با هم جمع کردم.
- در نهایت در عبارت دمپ کننده (e^(-aplha*t ضرب کردم تا صدا نرم شود سیکنالش و به پیانو نزدیک تر شود.
- سپس با سری سکوت که به اندازه 0.025 ثانیه ضربدر فرکانس نمونه برداری (44100) اندازه صفر مطلق اضافه کردم.

پس از پایان حلقه این عبارت را بعد از scale و به فرمت int16 درآوردن آن را در فایل noteOptimized.wav نوشتم.

```
172  final_optimized_sequence = []
173  alpha_damp = 6
174
175  for note in optional_notes:
176
177      note_name = note.split()[0] + note.split()[1]
178      duration = float(note.split()[2])
179
180      fs, audio = wavfile.read(FOLDER_PATH+note_name+'.wav')
181
182      # If the audio is stereo, convert it to mono by averaging the channels
183      if len(audio.shape) > 1:
184          audio = audio.mean(axis=1)
185
186      # Normalizing
187      audio = audio / np.max(np.abs(audio))
188
189      # Perform FFT to extract the frequencies and magnitudes
190      n = len(audio)
191      fft_spectrum = np.fft.fft(audio)
192      frequencies = np.fft.fftfreq(n, 1/fs)
```

fig 13. First part of loop

```
    # Find peaks in the FFT spectrum (harmonics)
    peaks, _ = find_peaks(magnitude, height=np.max(magnitude)*0.1, distance=100)
    harmonic_frequencies = positive_frequencies[peaks[:6]]  # First 6 harmonics
    harmonic_amplitudes = magnitude[peaks[:6]]

    t = np.linspace(0, duration, int(fs * duration), endpoint=False)

    # Generate the sound using the detected harmonics
    synthesized_signal = np.zeros_like(t)
    for i in range(len(harmonic_frequencies)):  # Summing the sine signals
        synthesized_signal += harmonic_amplitudes[i] * np.sin(2 * np.pi * harmonic_frequencies[i] * t)

    # Apply a damping factor to the signal
    damping_factor = np.exp(-alpha_damp * t)
    synthesized_signal *= damping_factor

    # Normalize the signal
    synthesized_signal /= np.max(np.abs(synthesized_signal))

    # Append the synthesized signal to the final sequence
    final_optimized_sequence.extend(synthesized_signal)

    final_optimized_sequence.extend(silence_samples)
```

fig 14. Second part of loop

فاز امتیازی

در این بخش می خواهیم با استفاده و کمک از روش گرفتن کورلیشن بین یک نوت و یک سگمنت از داده موسیقی و میزان ارتباطشان به لیست نوتی که آن موسیقی از آن تولید شده برسیم .

یک تابع به اسم predict_notes_from_wav نوشتیم. کاری که میکند آن است که در ابتدا با کمک توابع آماده فایل صوتی را میخواند و بعد به اندازه فرکانس ضربدر مقدار 0.025 که مقدار ثانیه سکوت است با این ریت پیمایش میکنیم و سپس هر بخش را کورلیشن (برای این بخش از توابع آماده کتابخانه scipy استفاده کردیم) آن را با تمام نوت هایی که در دیکشنری note_freqs که از قبل داشتیم میگیریم و آن که بیشترین هماهنگی را دارد انتخاب میکنیم . و عملا آن میشود نوت آن بخش ما.
راجب خود 0.025 ثانیه سکوت هم چون گفته شده بود که حتما از نوت ها انتخاب شود تنها آنجا پیشبینی های ما درست در نمی آید وگرنه میتوانستم یک نوتی به نوت ها اضافه کنم به اسم سکوت و در آن زمان ها چون نوت سکوت فرکانس اندازه صفر دارد لذا بیشترین کورلیشن را میتوانست داشته باشد و انتخاب شود اما به علت محدودیت سوال از این کار صرف نظر کردم.
در آخر هم لیستی از نوت ها دارم که نوت هایی که مشابه بودند پشت هم صرفا زمانشان را جمع کردم و در نهایت در فایلی به اسم predictedNotes.txt ذخیره کردم.
یک نکته ای هم که باید دقت کرد جون به segment های خیلی کوچ تقسیم میکنم داده موسیقی را **حدود یک دقیقه** اجرا شدن کد آن بخش زمان میبرد.

```python
def predict_notes_from_wav(wav_file, note_freqs, sample_rate, silence_duration=0.025):
    rate, audio_data = read(wav_file)

    if len(audio_data.shape) > 1:
        audio_data = audio_data.mean(axis=1)

    predicted_notes = []
    note_duration = silence_duration
    previous_note = None
    accumulated_duration = 0

    for start in range(0, len(audio_data), int(sample_rate * note_duration)):
        end = start + int(sample_rate * note_duration)
        segment = audio_data[start:end]

        correlations = {}
        for note_name, freq in note_freqs.items():
            ref_wave = generate_reference_wave(note_name, note_duration, sample_rate, note_freqs)
            correlation = correlate(segment, ref_wave, mode=
            correlations[note_name] = np.max(correlation)

        predicted_note = max(correlations, key=correlations.get)

        if predicted_note == previous_note:
```

Parameter **note_freqs** of CA1.main.predict_notes
note_freqs: {items, get}

fig 15. predict notes function

```python
            if previous_note is not None:
                predicted_notes.append((previous_note, accumulated_duration))
            previous_note = predicted_note
            accumulated_duration = note_duration

    if previous_note is not None:
        predicted_notes.append((previous_note, accumulated_duration))

    return predicted_notes


1 usage
def write_predicted_notes_to_file(predicted_notes, file_name="predictedNotes.txt"):
    with open(file_name, "w") as f:
        for note, duration in predicted_notes:
            f.write(f"{note} {duration}\n")


wav_file = 'noteHarryPoter.wav'
predicted_notes = predict_notes_from_wav(wav_file, note_freqs, sample_rate)

write_predicted_notes_to_file(predicted_notes)

print(f"Predicted notes with durations have been written to 'predictedNotes.txt'.")
```

fig 16. Write notes to text file

نمایش 3 نمونه از نمودار های فوریه

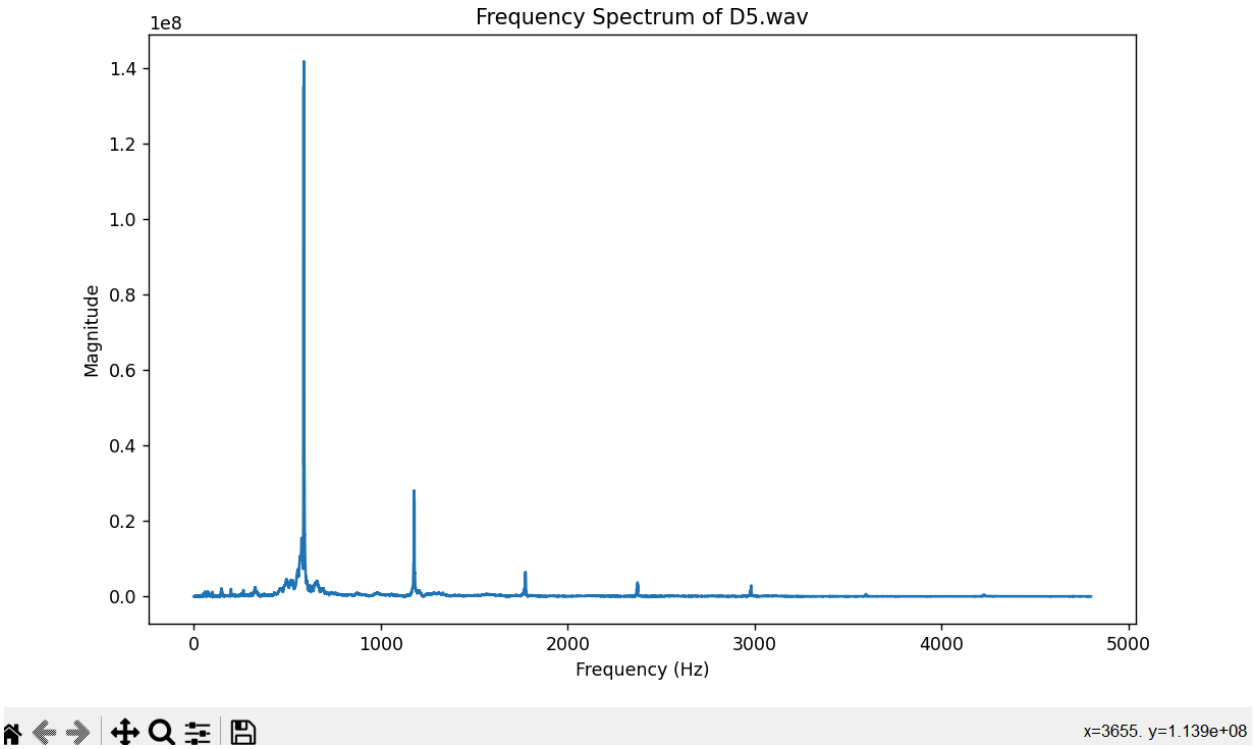

fig 17. D5

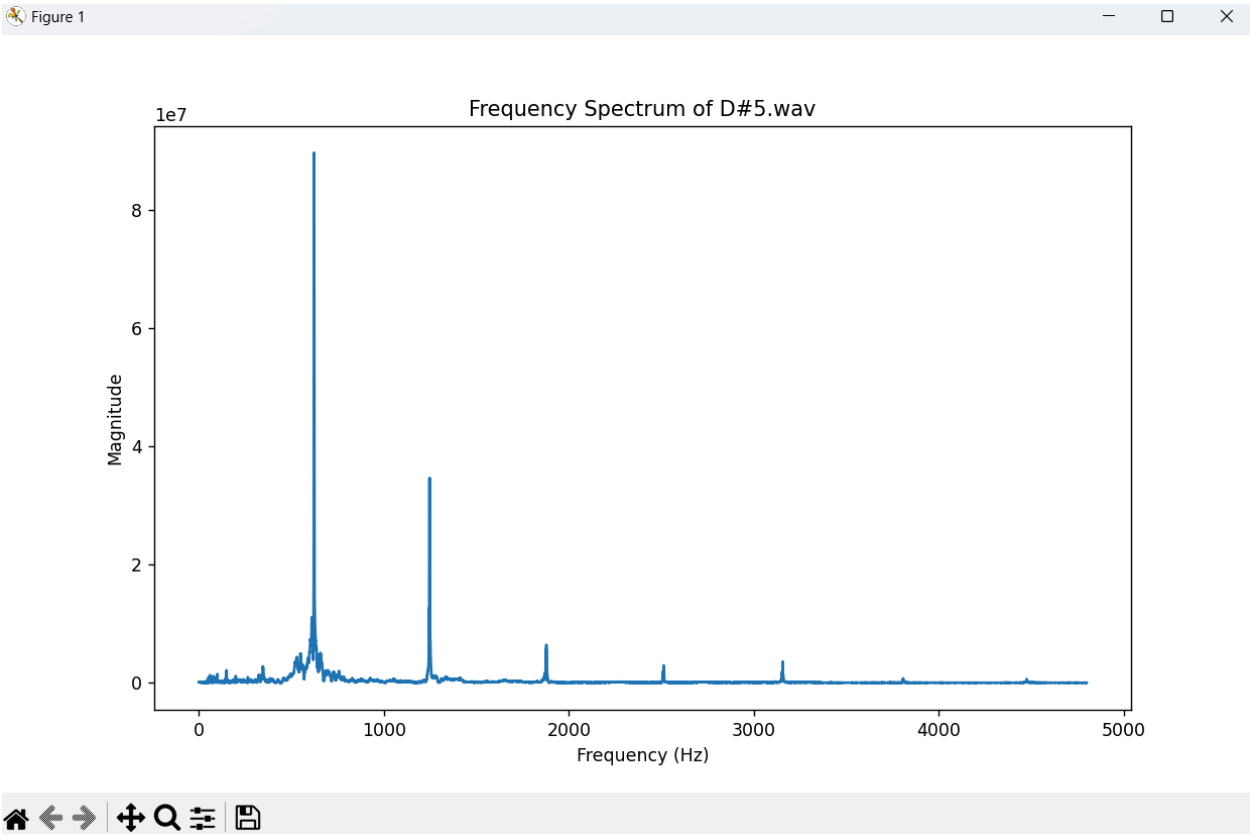
Figure 1
Frequency Spectrum of D#5.wav
1e7
8
6
4
2
0
Magnitude
0
1000
2000
3000
4000
5000
Frequency (Hz)

fig 18.D#5

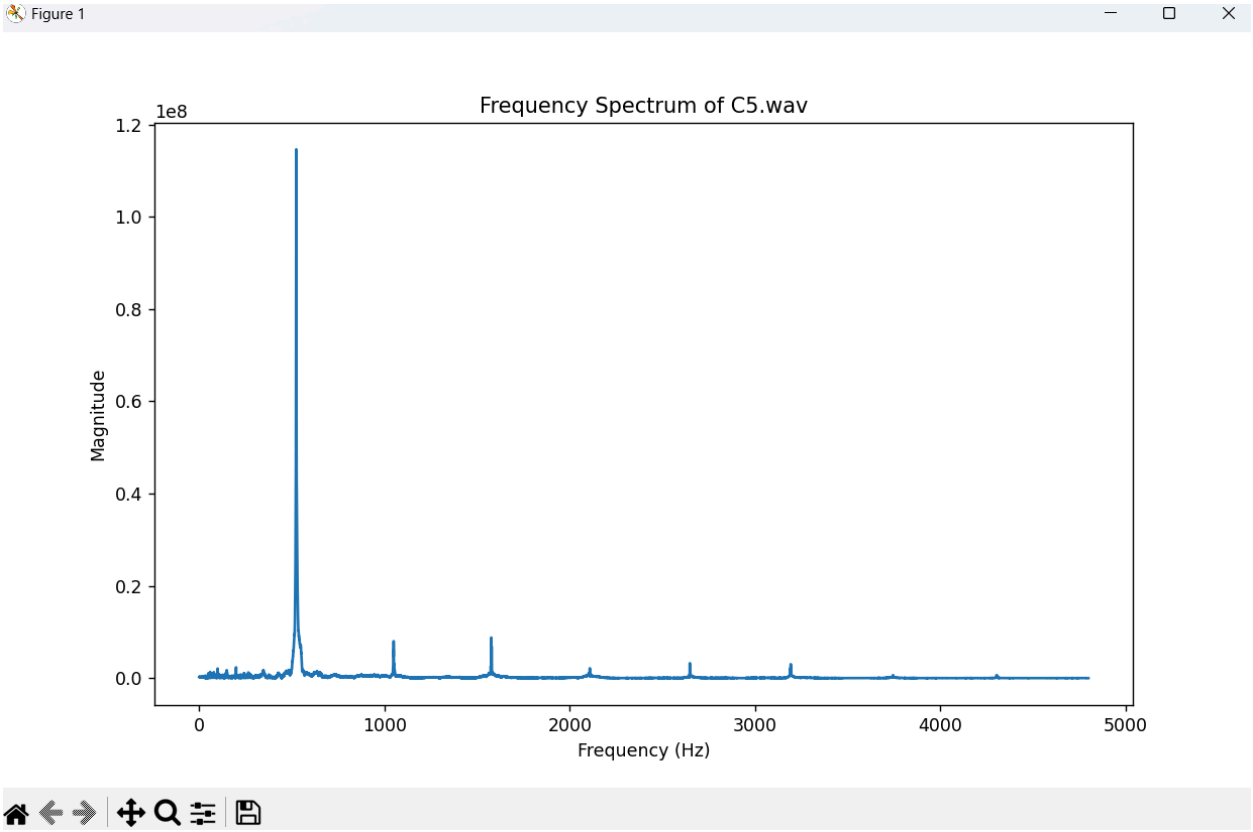

fig 19. C5

لینک های چت ها با ChatGPT :

https://chatgpt.com/share/67964750-bee0-8000-8ab4-a0404d9b45b4

https://chatgpt.com/share/679647d0-2c18-8000-9754-60947718553f

https://chatgpt.com/share/679647eb-2f14-8000-87ee-0cdc82f45ecb

https://chatgpt.com/share/67964808-f0e4-8000-becb-64061eec3ee4

https://chatgpt.com/share/6796483c-6274-8000-bb88-dd9a1833456c

https://chatgpt.com/share/6796487b-b0dc-8000-a5c3-1e85d488d8de

با تشکر!