An Introduction to Chord Recognition through FFT

Mathematics and Music

Kenneth Ng

Abstract

We have developed a workflow to identify musical chords through the processing of raw waveform-based audio signals via FFT. We considered a simple case where chords to be identified are within an octave, 3-note, with octave number (0-7), random and generated through a music library. We used certain peak-picking techniques for improving chord recognition. Experimental results with 800 random piano chords in total showed that the recognition rate was 61.875%.

Author's Declaration

I certify that this project report has been written by me, is a record of work carried out by me, and is essentially different from work undertaken for any other purpose or assessment.

Signed: .....

Date: .....

Contents

Chapter 1

Introduction

1.1. Objectives

The main objectives are to understand how FFT as a tool helps in chord recognition and explore how different types of musical signals influence the way we utilize the tool.

1.2. Intended Readership

This project is intended to be read by anyone interested in music without a particular background in Mathematics.

1.3. Pre-requisite definitions

There are a few definitions in music we need to know:

- *chords* are musical constructs that typically consist of three or more note;
- the *Scientific Pitch Notation* tells us that *Chroma*(or Pitch Class) and *Octave Number* together - in this order - form note names (eg A_4 and C_4); and
- *12-TET* represents the twelve-tone equal-tempered scale, dividing an octave in 12 by

$$f_p = 2^{(p-69)/12} * f_{69} \tag{1.1}$$

[1] , where $p \in [0 : 127]$ is the MIDI number of a note and f_{69} is the frequency of A_4.

Figure 1.1 is a concise summary of the definitions introduced above in the context of a piano keyboard:

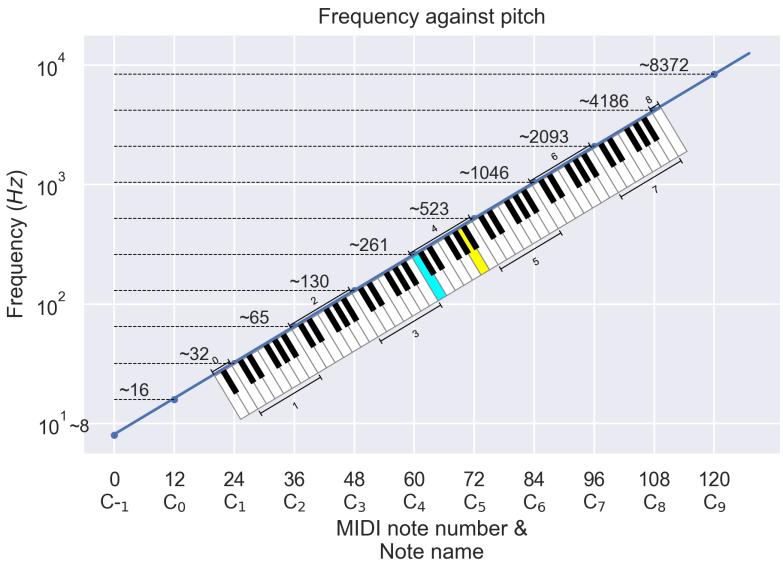


Figure 1.1

The standard 88-key piano in the figure is tuned in 12-TET according to A440, with its keys distributed in an exponential scale following Equation 1.1. A4 and C4 represent the notes in yellow and cyan respectively according to the Scientific Pitch Notation.

1.4. Assumptions

For the purpose of this project, a few assumptions are made to help reduce the complexities of the task without loss of (meaningful) generality:

- the notes to be recognized are recorded digitally to produce audio samples (eg MP3, WAV and FLAC);
- the audio samples are produced by instrument tuned in 12-TET[1], which is the current Western standard tuning method; and

[1]Other equal temperaments (n-TET) also exist, as well as different tuning systems like Pythagorean Tuning, just intonation and mean-tone temperament. However, the principles behind chord recognition are not affected by their differences.

- 12-TET is tuned relative to A440, where

$$f_{69} = A_4 = 440\,\text{Hz}$$

1.5. Motivations

Chord recognition is task that falls under audio signal processing, with a particular focus on harmonic analysis. It also belongs to the multidisciplinary domain of music information retrieval (MIR), which covers areas like optical music recognition (OMR), query by humming (QbH) and music feature (eg key, genre, melody and emotion) identification. It is generally most useful in music indexing/labelling, which lays the groundwork for higher-level applications in MIR. In particular, chord recognition is important in the following ways:

- musicologists or music enthusiasts involved in areas such as music analysis and transcription where the identification of chords is indispensable. Although many could preform harmonic analysis by ear, it could quickly become tedious or even too difficult, especially when a complex piece of music is involved. One can automate the process with a chord recognition program;
- it helps automate the process of music indexing, broadening the database for certain search engines built for music; and
- the software or devices musicians and tuners use to tune their instruments rely on accurate frequency measurements, which is a sub-task of chord recognition — pitch recognition.

1.6. Typical workflow

Below shows the typical series of procedures involved in the making of the examples presented in the project.

1. Pre-processing (audio files)

- Trim the audio file to separate the chords out
- Normalize
- Denoise

2. Performing FFT, which spits out frequencies and their corresponding amplitudes
3. Processing FFT results
   - Analyzing and accounting for the influence overtones
   - Picking peaks, namely the notes identified through FFT
4. Post-processing (peaks)
   - Template matching using a pitch class profile to find the most likely chord present

1.7. Other methods

FFT could be understood as the 'classical' approach as FFT on its own only separates out individual sinusoids that compose the original audio signal. In this project, FFT is understandably the main focus. However, there are many other approaches to chord recognition such as the use of Hidden Markov Models and Neural Network Models. Notably, the latter allowed Google's Now Playing feature to be run entirely on-device by way of a robust audio fingerprinting system.[2]

Chapter 2

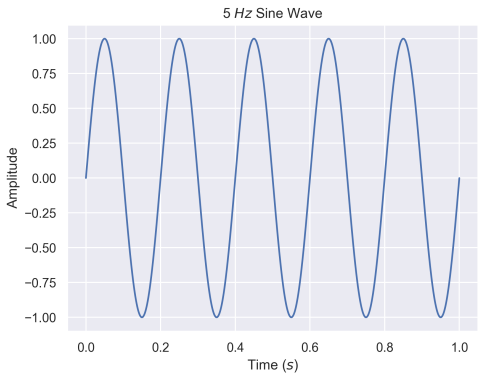
Pitch Recognition

Before tackling chords, we first look at how we can identify a note or a pitch.

2.1. Synthetic Note

In order to generate a synthetic note, we need to define the wave function, sampling rate and duration. Using these variables, we obtain two arrays of values consisting of time and amplitudes. In the example, we have $y = \sin(2\pi ft)$, sampling rate 44 100 Hz and duration 1 s.

We start by looking at a pure 5Hz sine wave. A



(a) Score

(b) Audio

Figure 2.1: 2 representations of a 5Hz sine wave

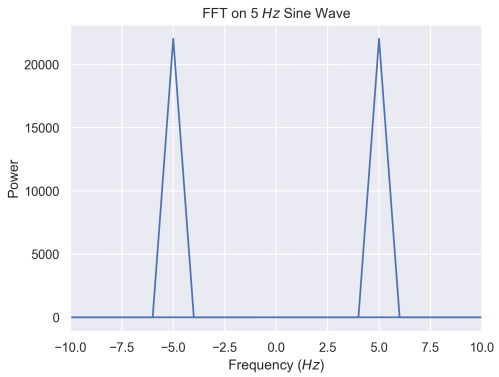


Figure 2.2

We see two spikes in our FFT graph. This is because only the norm of the transform is used, which we will see later in chapter.

2.2. Real Note

When we have a real note, we immediately and counter a few important issues. First, there is inevitably noise in our audio sample. This means that our audio signal would not behave like a pure sign wave. Second, there is the issue of overtones where we do not just see the frequency of the note, but a series of frequencies instead in our FFT graph.

To deal with the first complexity, there are many well developed the noising algorithms available such as.. In this project, however, we will not attempt to handle this complexity. Instead we will choose clean audio samples, meaning that they are coming from a well-controlled studio recording environment.

2.2.1. Definitions

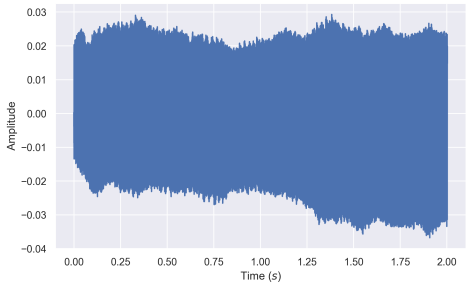
In order to better understand these important complexities, a few terms (equivalent to that given by Müller[1]) are defined below:

- *partial* is any of the sinusoids by which a musical tone is described (integer multiples of the fundamental);
- *overtone* is any partial except the lowest;

- The *fundamental frequency* is the the frequency of the lowest partial present ;

2.2.2. Example

In this example, the oboe A_4 is picked specifically as it is used to tune the orchestra, suggesting perhaps the pitch of the note is easily recognizable and relatively stable.



(a) Waveform

(b) Audio [3]

Figure 2.3: 2 representations of an oboe's A_4

Figure 2.4

In Figure 2.4, we clearly see the behaviour of overtones. Firstly, we have the fundamental frequency at 436 Hz. Then we see spikes at integer factors of the fundamental frequencies. The spike at E_6 is particularly interesting.

Since we picked the tuning A that we might expect A_4 to be the highest peak, but it is E_6 that is the highest. From this, we know that the human ear does not perceive a pitch by decomposing the audio signals in two contributions of sinusoids and then picking that with the highest contribution. Upon comparing different recordings of oboes A_4, it is observed that each of them produce spikes at similar frequencies, however their magnitudes vary. The variations in these peaks explains how different oboes playing the same note could sound different from each other. In other words, one might pick up some information about the timbre of the notes from Figure 2.4.

Another curious concern about the interpretation of Figure 2.4 also arises. Just from looking at the graph, it is not clear at all whether we are looking at a single note or chord. One could, for instance, argue that both A_4 and A_5 are played.

The final issue concerns the interpretation of peaks in Figure 2.4. First, if we look closely we see these tiny peaks near zero that are not labelled. This is because we imposed an arbitrary threshold for our peaks. In this case, anything lower than 0.00015 would not be considered as a peak. It is not entirely obvious how the threshold should be determined as different durations of recordings produces FFT graphs with peaks of different magnitudes. Second, we see a lot of crosses gathering on seemingly isolated peaks. Each of these in fact represent a different peak at its unique position. Later in Chapter 3, we will device certain criteria in order to only pick out peaks that are most useful.

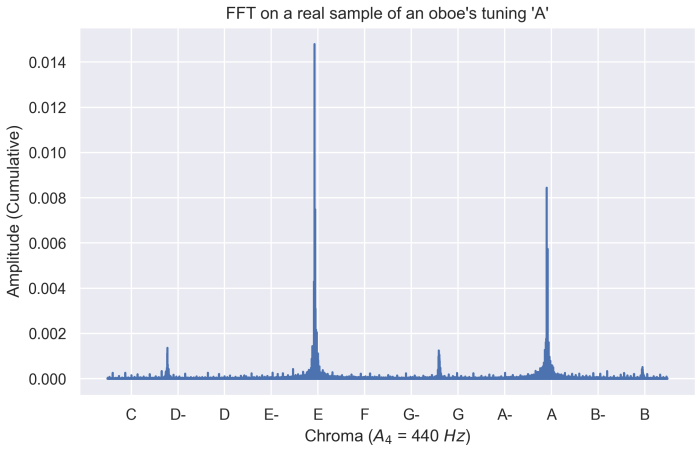


Figure 2.5

One can only consider the chroma feature of the spectrum by adding up contributions from different octaves, as reflected in Figure 2.5. Using the

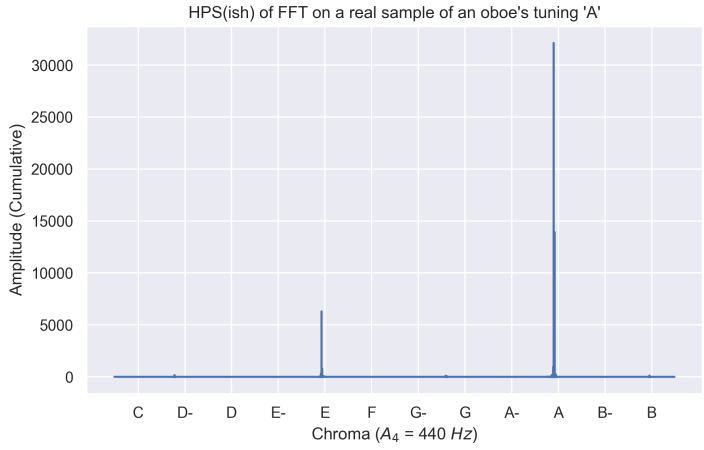


Figure 2.6

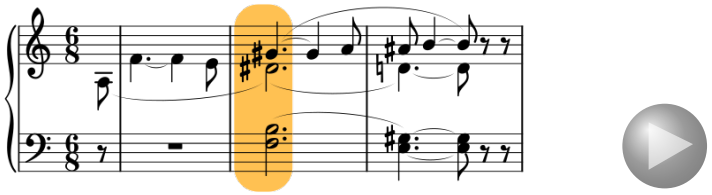
Chapter 3

Chord Recognition

3.1. Synthetic Chord

We begin by looking at a synthetically generated half-diminished seventh chord taken from the opening phrase of Wagner's *Tristan und Isolde*.

3.1.1. Example



(a) Score (b) Audio

Figure 3.1: 2 music representations of the Tristan chord

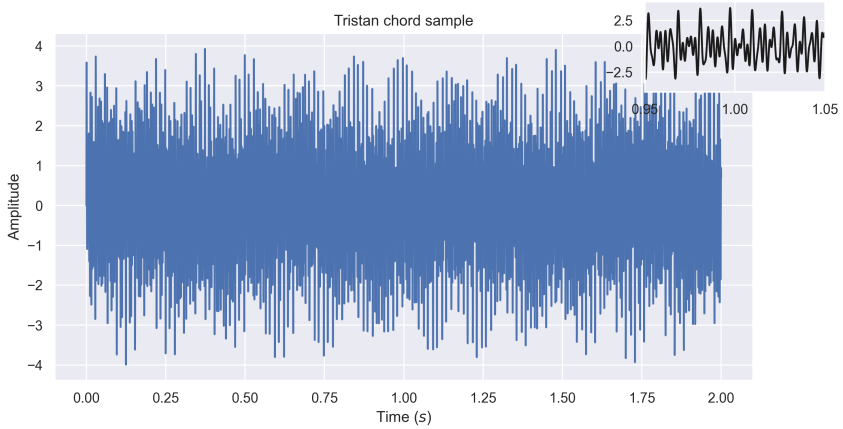


Figure 3.2

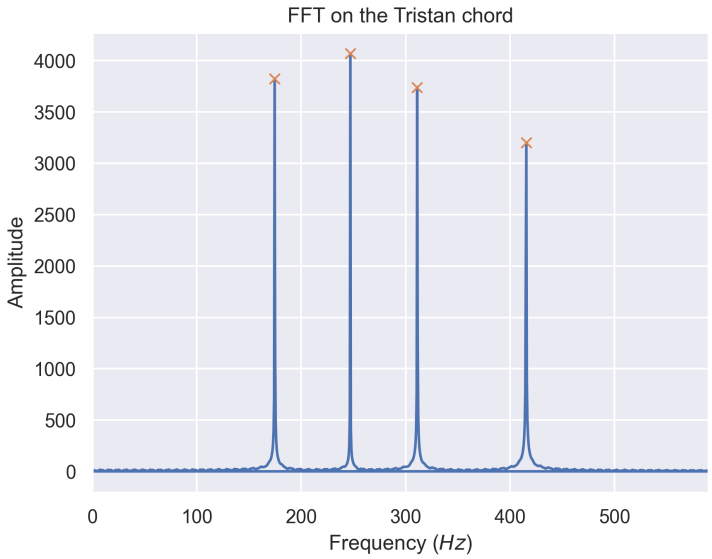


Figure 3.3

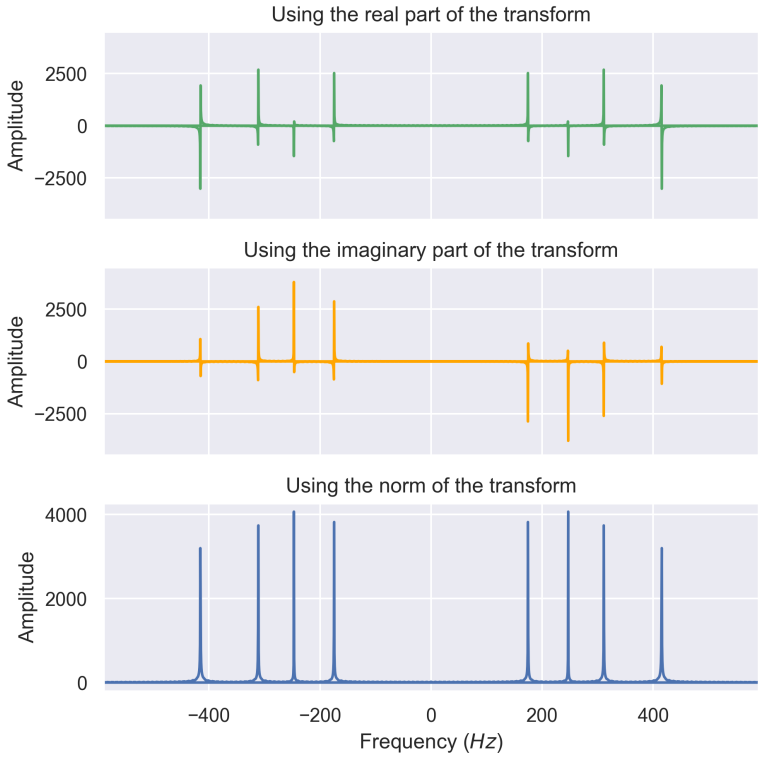


Figure 3.4

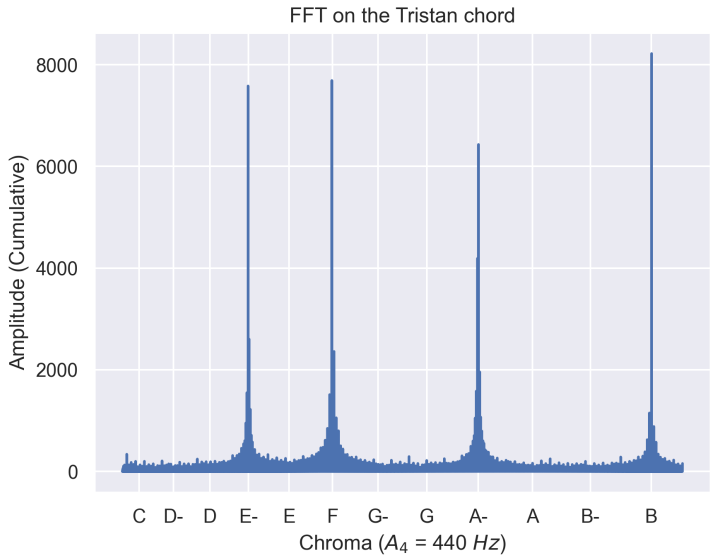
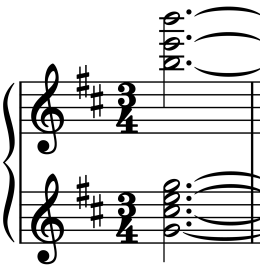


Figure 3.5

3.2. Real Chord

Next, we look at another half-diminished seventh chord using the first chord of Chopin Scherzo No.1.

3.2.1. Example



(a) Score

(b) Audio

[4]

Figure 3.6: 2 music representations of the 1st chord of Chopin's Scherzo No.1

3.2.2. Results

The results are surprisingly promising. Although no de-noising algorithms have been deployed, the FFT is able to pick out all 4 pitch classes (B, G, C#

and E) present according to Figure 3.8. We can also roughly identify 7 spikes present in 3.7.

3.2.3. Discussion

It is however inconclusive what exactly has contributed to the success in this example. The FFT is working as expected but we would expect to see more messy partials. It appears, after some surface research, that this is partly the result of the fact that the partials do not concentrate at integer multiples of the fundamental frequencies[5]. This perhaps causes the spikes to not line up as much as what is seen in Figure 2.5. Instead, we see clear thick spikes with a more noisy base floor of spikes in Figure 3.8.

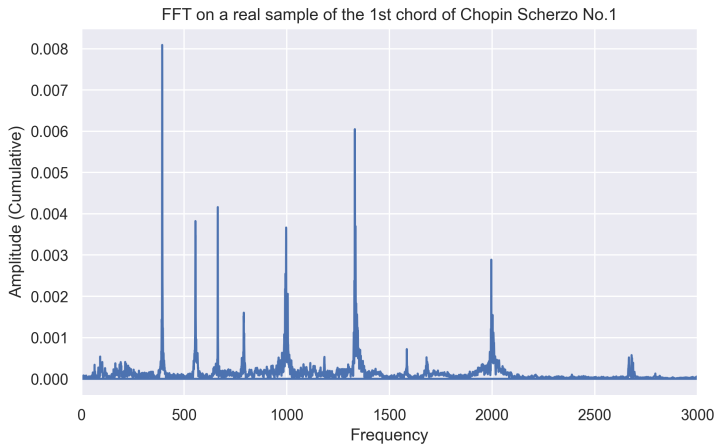


Figure 3.7

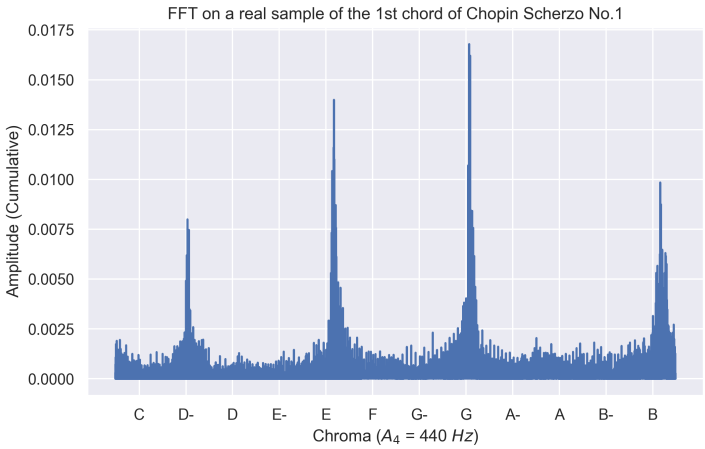


Figure 3.8

3.3. More 'Real' Chords

Finally, we will look at how well our workflow performs in a larger scale. In an ideal case, we would find or create labelled real audio samples for testing. However, we have decided against using real chords weighing the gains against the complications involved and considering our time constraints.

There are three main benefits of using synthetically generated 'real' chords. Firstly, we could do tests by simply changing the parameters in the code. For example, we could experiment with different instruments, volumes, dynamics, octaves and specific ranges or combinations of notes. Secondly, the notes we identify through FFT can be verified easily without labelling as we know the correct answers. Finally, individual notes from the chords are still taken from a real instrument so the chords are still semi-real.

We now consider a simple case with the following conditions:

- we consider chords within an octave, meaning no overtone analysis as only fundamental frequencies exist; and
- we use 100 random 3-note piano chords per octave number ranging from 0 to 7 (inclusive).

A series of operations similar to those introduced in the workflow given in

Section 1.6 is then underwent. Here, we generate a MIDI file with random chords using **Python**, use the music library provided by **MuseScore 4** to generate the audio file, split the audio file using **ffmpeg**, perform FFT and assess the results.

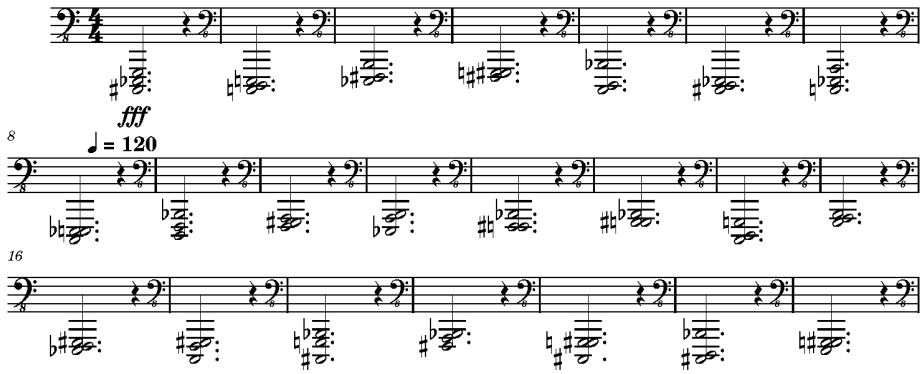


Figure 3.9: First 22 (out of 800) bars from the MIDI file

3.3.1. Peak-picking

Gathering experience from all previous examples, we can device a couple of criteria for isolating the desired peaks. In the code, two thresholds are defined to improve the determination of peaks:

- `peak_threshold`: the minimum height for a peak

- `cent_threshold`: allowance in cents (hundredth part of the smallest interval in a 12-TET scale) for deviance from the A440 standard (for each chroma)

3.3.2. Results

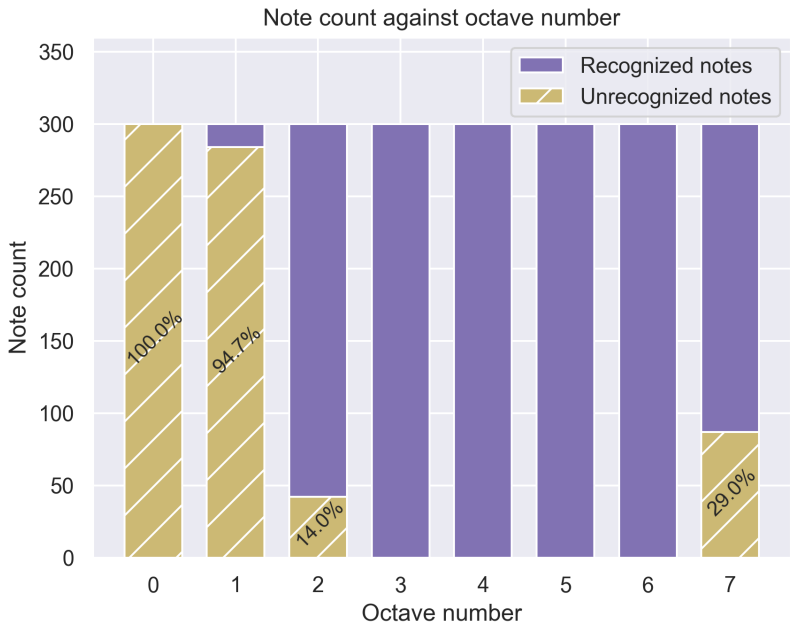


Figure 3.10

In the end, $495/800 = 61.875\%$ of the chords are successfully identified. From Figure 3.10, we see that notes from octave number 3 to 6 are actually identified with 100% accuracy. Our FFT could not pick up most notes with octave number 0 or 1, which are low-frequency notes in music terms.

3.3.3. Discussion

From Section 2.1, we know that the FFT is capable of picking up a low-frequency signal. According the some of the failed examples in Figure 3.11, we see that in some cases, the chroma can in fact be correctly identified. If we look at the FFT of the chord highlighted in blue in Figure 3.11 given by 3.12, we see that first 4 peaks are from pitch classes D, G, A and $B-$. A is incorrectly identified as it turns out to be a partial of D_1. If we zoom in, we see from Figure **??** that the reason is that the peaks at the fundamental frequencies are too low in power to be identified. A potential fix would be to increase the duration of our audio samples, though more noise might be

introduced to confuse the peaks.

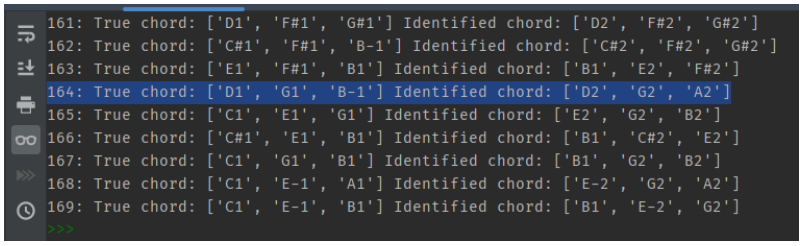


Figure 3.11: a nice plot

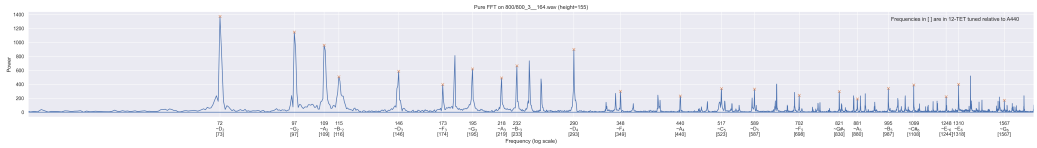


Figure 3.12

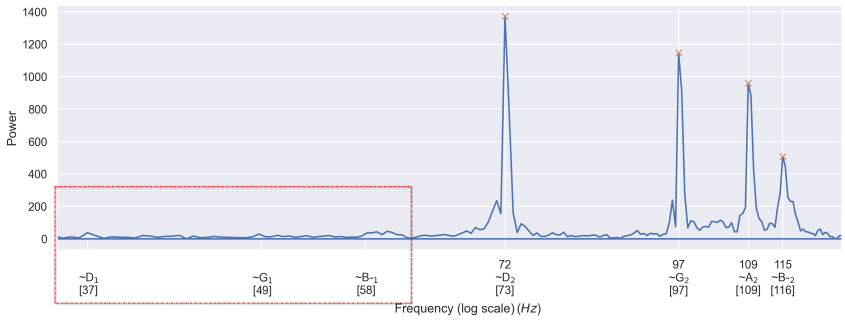


Figure 3.13

References

[1] Meinard Müller. *Fundamentals of Music Processing: Using Python and Jupyter Notebooks*. Jan. 2021. ISBN: 978-3-030-69807-2. DOI: 10.1007/978-3-030-69808-9.

[2] *Google's Next Generation Music Recognition*. Sept. 14, 2018. URL: https://ai.googleblog.com/2018/09/googles-next-generation-music.html.

[3] Humbert Lucarelli. *Oboe Concerto: Tuning Game*. 1990. URL: https://www.youtube.com/watch?v=t18EgI2Jsmk.

[4] Ivo Pogorelich. *Chopin: Scherzo No. 1 in B Minor, Op. 20*. Jan. 1, 1998. URL: https://www.youtube.com/watch?v=9VG2a37A64g.

[5] Simon Hendry. "Inharmonicity of Piano Strings". The University of Edinburgh, School of Physics, 2008. URL: http://www.simonhendry.co.uk/wp/wp-content/uploads/2012/08/inharmonicity.pdf.

[6] Purves D et al. *Neuroscience*. 2001.