Pitch Analysis using Cumulative Mean Normalized Difference Method with ACF and Deep Learning

Atharva Saney (23PGAI0069), Rohan Mehta (23PGAI0001)

Abstract

Pitch analysis is an essential task in many audio-related applications, such as music transcription, speech recognition, and sound processing. With the rise of machine learning techniques in the field of audio signal processing, there has been a growing interest in using machine learning algorithms for pitch analysis. This paper presents a comprehensive review of pitch analysis techniques using Python, machine learning and deep learning. We discuss the fundamental concepts of pitch analysis, including the types of pitch estimation methods and its fundamentals. We use the mathematical foundation of pitch and employ it to do pitch analysis. We will also deep dive into deep learning models which can do the same task in an efficient manner.

Introduction

Pitch analysis is the process of analyzing the fundamental frequency or pitch of a sound signal. It involves using digital signal processing techniques to measure and quantify the frequency of a sound wave.

The pitch of a sound is determined by the frequency of the sound wave, which refers to the number of oscillations or cycles of the sound wave that occur per second, measured in Hertz (Hz). The human ear is capable of detecting frequencies between 20 Hz and 20,000 Hz, and pitch analysis is used to measure and analyze the pitch of sounds within this range.

Pitch analysis is used in a variety of fields, including music, speech analysis, and audio engineering. In music, pitch analysis is used to analyze and transcribe melodies and harmonies, while in speech analysis, it is used to study prosody and intonation. Audio engineers use pitch analysis to adjust the pitch of recorded audio signals or to create pitch-based effects such as autotune.

Motivation

Pitch analysis has a wide range of applications in various domains. Some of the common applications of pitch analysis include:

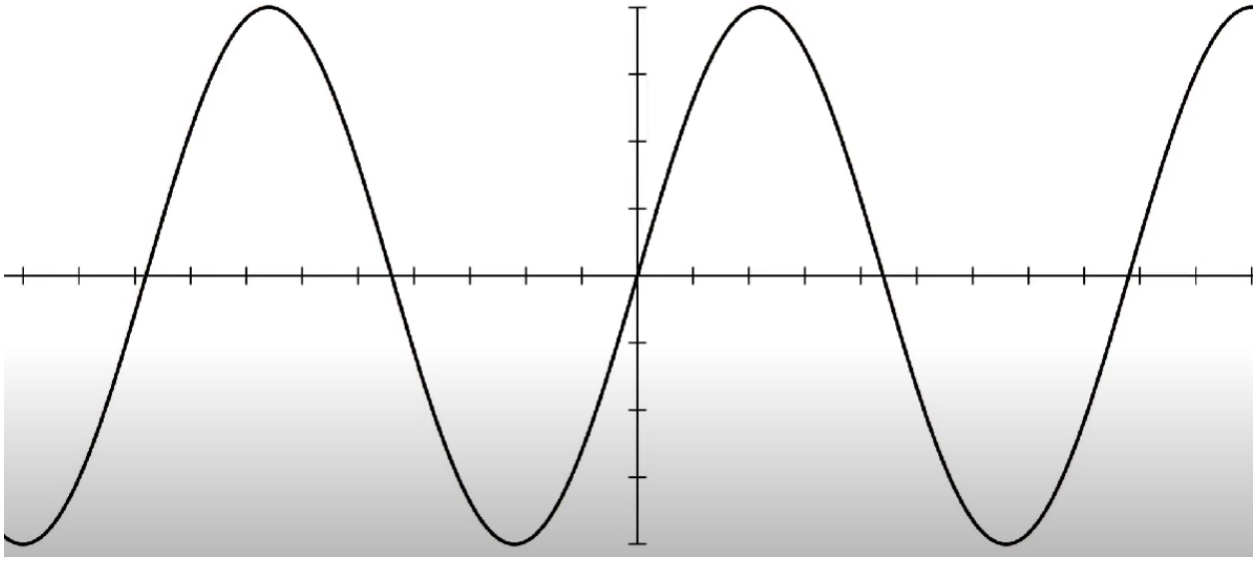
1. Music transcription: Pitch analysis is used to transcribe musical notes from an audio recording. This application is particularly useful for creating sheet music from a recording of a performance or for analyzing the melodic and harmonic structures of a musical piece.
2. Instrument tuning (guitar, violin)
3. Speech recognition: Pitch analysis is used in speech recognition systems to extract the fundamental frequency of a speaker's voice. This information can be used to identify the speaker, recognize speech patterns, and convert speech to text.
4. Emotion detection: Pitch analysis can be used to detect changes in the pitch and tone of a person's voice, which can indicate changes in emotion. This application is particularly useful in fields such as psychology and mental health, where detecting changes in emotional states can be important for diagnosis and treatment. High pitch sounds are often associated with emotions such as excitement, happiness, and fear, while low pitch sounds are associated with emotions such as sadness, calmness, and seriousness.
5. Sound processing: Pitch analysis can be used in sound processing applications to filter out unwanted frequencies or to modify the pitch of a sound. This application is particularly useful in fields such as music production and audio engineering.
6. Auto-tune: This can be used to manipulate the pitch to create music
7. Medical diagnosis: Pitch analysis has potential applications in medical diagnosis, particularly in the field of cardiology, where changes in the pitch and frequency of heart sounds can be used to diagnose various cardiac conditions.
8. Industrial monitoring: This can be used in industrial monitoring to detect and diagnose faults in machines and equipment based on the frequency of the sounds they produce. For eg, cavitation in pumps.

Overall, pitch analysis is a crucial component of many audio-related applications and has potential applications in several other fields as well.

Methodology

For this project, we have followed various methods. Initially we used cumulative mean normalized difference method. This involves finding the squared difference in between two instances of an audio and trying to minimize this difference. Best case scenario is when we find the minimum time lag after which the signal repeats itself, but this is rarely true for any signals in real life. For simplicity, we have converted the squared difference into ACF (autocorrelation function). We will first test this out on a test function, and then apply it on a real audio file. Our output will print a sequential list of all the pitches in the audio. Lets try to understand this process further from the basics

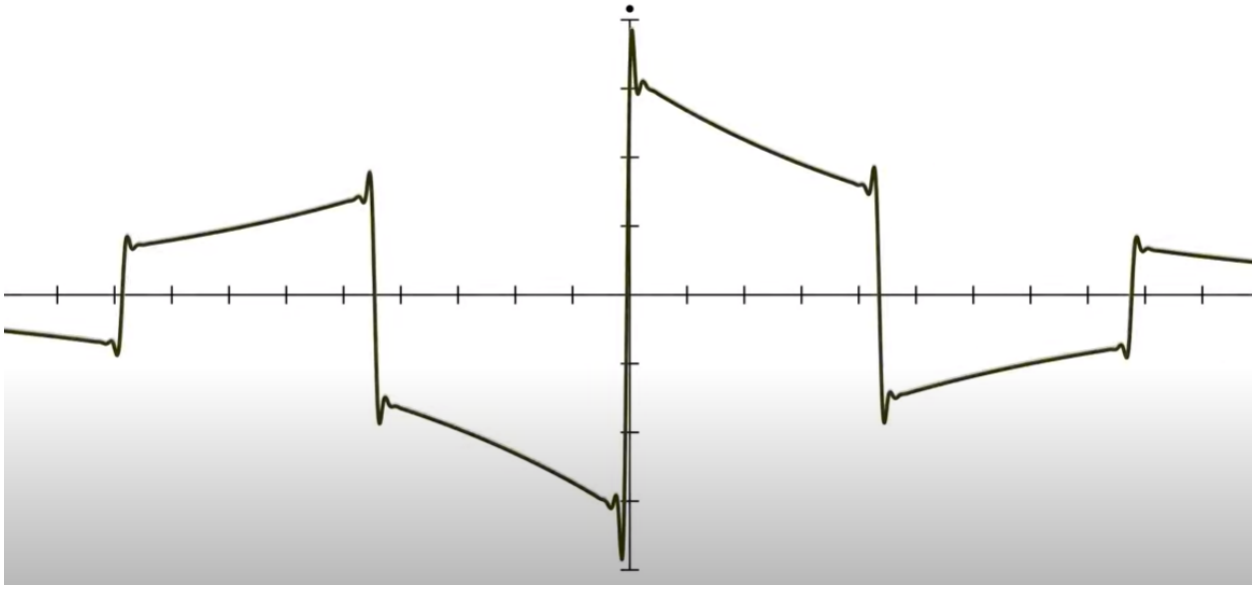
Lets consider a sine wave with unknown frequency:



We can easily find its frequency by finding the time period after which it repeats itself, and taking the inverse of it. We can find the time period by playing around with it, and we find that the time period comes out to be 1/220 seconds. Hence, we can conclude that the frequency or pitch of this sine wave is 220 Hz.

To clarify why we found the frequency and set it as the pitch, recollect that any periodic signal can be expressed as a sum of sine waves at different frequencies and the component with lowest frequency is called fundamental frequency of that signal. This becomes the pitch of the signal. So the pursuit of finding the pitch of a signal is nothing but finding the fundamental frequency of the signal.

Moreover, this method quickly breaks down as we try to find the pitch of the signals for non periodic or modulated signals. Consider the signal below



No matter how much we shift the signal, it will never be identical. Hence, playing around with it will not give us the pitch. So we need to look at better and more generalized methods to find the same.

To deal with this, we can use the similarity technique. We shift the signal by 1 time lag, and find the similarity of the original signal with it.

$$\text{similarity}(\tau) = \sum_i f(x_i) \cdot f(x_i + \tau)$$

This similarity can also be thought of as the autocorrelation function.

$$\text{ACF}(\tau) = \sum_{i=t}^{t+W} f(x_i) \cdot f(x_i + \tau)$$

Where W is the window at timestep t at which we want to determine the pitch.

In order to make the method more robust, we will actually consider the squared difference with one time lag over the same window and try to minimize it.

$$\mathrm{DF}(\tau) = \sum_{i=t}^{t+W} (f(x_i) - f(x_i + \tau))^2$$

We can expand this squared difference expression, and express it in terms of ACF, since we have defined it earlier and it's easy to understand

$$\mathrm{ACF}(\tau, t) = \sum_{i=t}^{t+W} f(x_i) \cdot f(x_i + \tau)$$

$$\mathrm{DF}(\tau) = \sum_{i=t}^{t+W} (f(x_i) - f(x_i + \tau))^2$$

$$= \sum_{i=t}^{t+W} \left(f(x_i)^2 + f(x_i + \tau)^2 - 2 \cdot f(x_i) \cdot f(x_i + \tau)\right)$$

$$= \mathrm{ACF}(0, t) + \mathrm{ACF}(0, t + \tau) - 2 \cdot \mathrm{ACF}(\tau, t)$$

Since in real life, we will be dealing with signals of varied amplitudes, we can normalize the difference too, before implementing our technique.

$$\mathrm{CMNDF}(\tau) = \begin{cases} 1 & \tau = 0 \\ \tau \cdot \dfrac{\mathrm{DF}(\tau)}{\sum_{j=1}^{\tau} \mathrm{DF}(j)} & \tau > 0 \end{cases}$$

What this means is that the function will only drop below 1 when the difference function is lower than the average of what we have guessed so far.

Now we are ready to implement this technique. You can refer to the code to see how it works. First we will create a test signal and try to find its pitch using our technique.

Next, we will use the same technique to detect pitch live. Users can play the audio to the code and it will detect pitches live. Two approaches are followed to accomplish this.

Approach 1

We will calculate and print the frequencies (pitches), just like we did before with the test audio. After you are done, press ENTER to view the historical graph of all previous pitches in that session.

Approach 2

The main intention to predict or calculate pitch, is to check the octave of specific notes. All musical notes have a specific frequency range in that specific octave. We have included a master table for the same, and we want to display the user those specific notes. Hence we have followed approach 2, which is again another method to calculate frequency, and convert them to specific notes, not only do we need to display notes, but also the octave.

To evaluate the working of our model, we will compare the same with the Guitar tuner app which is one of the most accurate pitch measurement devices. We have included the comparison in video as well

The third and last approach that we will use is deep learning, for this we have used pre-trained models from the library “basic-pitch”. It will detect pitches and print them in a similar fashion

Implementation

Major libraries used:
1. Pyaudio

2. Time

3. Librosa

4. Matplotlib

5. Numpy

1. Implementation on test audio

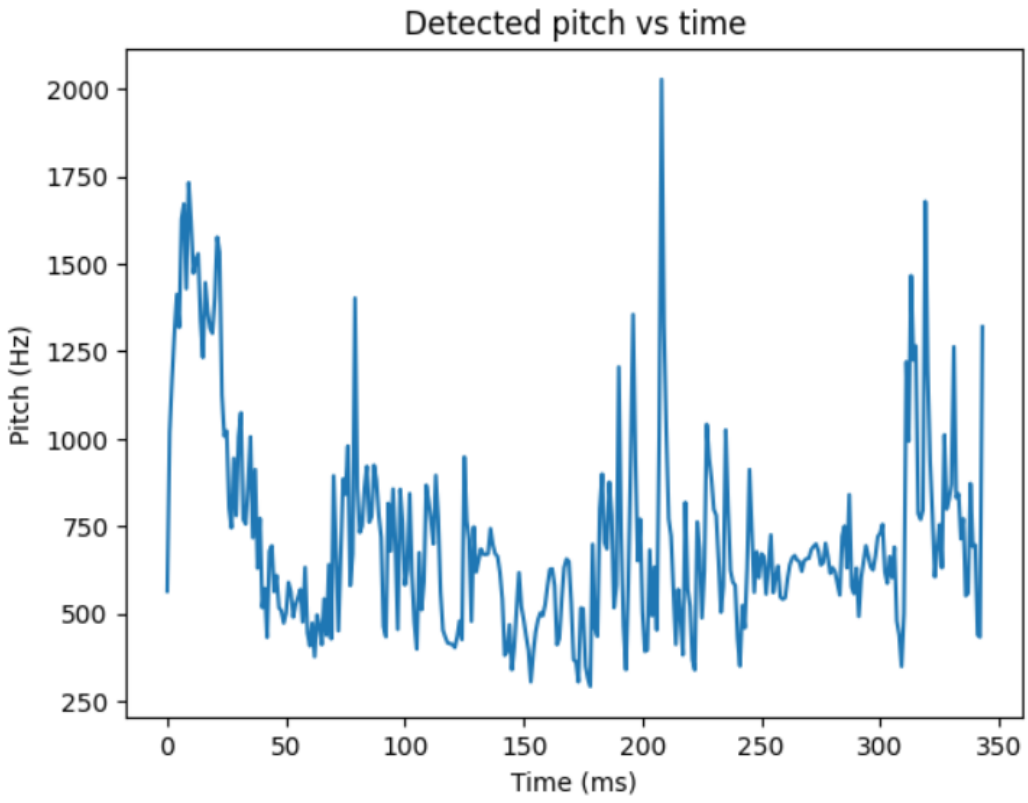
We created a test audio with known pitch of 1 Hz, these are the results on that:

```
In [2]: %%time

        ## first we will feed in a sample input, to check if the code is working

        test()

        detected pitch: 1.002004008016032 Hz
        CPU times: total: 8.42 s
        Wall time: 8.53 s
```

2. Implementation on live data

Approach 1 output:

```
Detected Pitch: 433 Hz
Detected Pitch: 1320 Hz
```



Approach 2 output:

```
Closest note: F#4, max frequency: 380.0,  closest pitch: 370.0
Closest note: E5, max frequency: 661.0,  closest pitch: 659.3
Closest note: E5, max frequency: 661.0,  closest pitch: 659.3
Closest note: E4, max frequency: 324.0,  closest pitch: 329.6
Closest note: E4, max frequency: 324.0,  closest pitch: 329.6
```

Master table for approach 2:

Table of Musical Frequencies

Note	Frequency
C	130.82
C#	138.59
D	146.83
D#	155.56
E	164.81
F	174.61
F#	185
G	196
G#	207.65
A	220
A#	233.08
B	246.94

Note	Frequency
C	261.63
C#	277.18
D	293.66
D#	311.13
E	329.63
F	349.23
F#	369.99
G	392
G#	415.3
A	440
A#	466.16
B	493.88

Note	Frequency
C	523.25
C#	554.37
D	587.33
D#	622.25
E	659.26
F	698.46
F#	739.99
G	783.99
G#	830.61
A	880
A#	932.33
B	987.77

Note	Frequency
C	1046.5
C#	1108.73
D	1174.66
D#	1244.51
E	1318.51
F	1396.91
F#	1479.98
G	1567.98
G#	1661.22
A	1760
A#	1864.66
B	1975.53
C	2093.00

Source: https://www.intmath.com/trigonometric-graphs/music.php

3. Detection using standard deep learning libraries

```
Predicting MIDI for CantinaBand3.wav...
```

```
Out[36]: {'contour': array([[0.12747292, 0.09957641, 0.09814113, ..., 0.10240234, 0.0837875 ,
                0.1070776 ],
               [0.12726292, 0.10001253, 0.09772835, ..., 0.10539532, 0.08761113,
                0.10866801],
               [0.12722479, 0.10015807, 0.09714406, ..., 0.10287426, 0.08785017,
                0.10715754],
               ...,
               [0.09274735, 0.08341444, 0.09868293, ..., 0.10437474, 0.08582141,
                0.15232265],
               [0.09283369, 0.08504324, 0.09860569, ..., 0.10540222, 0.08795976,
                0.16192298],
               [0.09284732, 0.08695046, 0.09818973, ..., 0.10062294, 0.08828031,
                0.13847543]], dtype=float32),
 'note': array([[0.11603396, 0.09931272, 0.10048861, ..., 0.10310158, 0.10474327,
                0.10662778],
               [0.11603969, 0.09959069, 0.10020691, ..., 0.10300893, 0.10441104,
                0.10696567],
               [0.11603568, 0.09996271, 0.09984739, ..., 0.10290199, 0.10409098,
                0.10727966],
               ...,
               [0.11072019, 0.09746922, 0.09699234, ..., 0.09445094, 0.10007014,
                0.10700075],
               [0.11109252, 0.09738567, 0.09733161, ..., 0.09379276, 0.09930755,
                0.10675437],
               [0.11145017, 0.09726664, 0.09768471, ..., 0.09345133, 0.09892678,
                0.10665303]], dtype=float32),
 'onset': array([[0.12822069, 0.08546641, 0.08397669, ..., 0.08331963, 0.07684267,
                0.09781824],
               [0.12926114, 0.08620655, 0.08626656, ..., 0.09549867, 0.0820595 ,
                0.10590827],
               [0.12981355, 0.09115729, 0.08751168, ..., 0.09932519, 0.0850867 ,
                0.10066921],
               ...,
```

Conclusion

We have demonstrated the implementation of a pitch detection algorithm using Python and the Librosa library. The results showed that the implemented algorithm was able to accurately detect the pitch of audio signals, thus confirming the effectiveness of our approaches.