Polyphonic Pitch Detection for Classical Guitar Using the Julia Programming Language

Mustafa Gunalp

Abstract: A method is proposed to determine the fundamental frequencies of a given polyphonic classical guitar chord. The well-established Discrete Fast Fourier Transform is used as the primary feature extractor to represent time domain signals in the frequency domain. The resulting values are then put through several modifications , with statistical analysis applied afterwards to eliminate false frequency candidates such as overtones and harmonics. The method relies on the selected audio time frame to be noiseless and mostly flat i.e. the musical notes should not have a relatively large difference between their amplitudes. Should these conditions be satisfied, the accuracy of the method is on par with state of the art algorithms. The main advantage of the method is its concise definition, such that any implementation of the method or modifications done on it would be much more simpler and efficient when compared with other methods and algorithms. Lastly, the method sheds light on a type of analysis that can be applied to other types of audio to improve the precision and reduce the complexity of their respective pitch detection algorithms.

Keywords: Polyphonic pitch detection, Music transcription, Classical guitar, Music information retrieval, The Julia Programming Language, Fourier analysis

1. Introduction

Software powered pitch detection for music transcription opens up the possibility for various use cases such as technology based musical learning applications, automatic transcription for musical performances and the restoration of musical pieces from various recordings. The latter is the main motivation behind this project, as it is usually impossible to find sheet music for lesser known classical guitar pieces, making it challenging to perform said pieces considering the only chance to reproduce the notes would be to either play them by ear, or manually transcribe them. As both approaches require one to be able to recognize musical notes by ear, software based pitch detection would allow everyone to perform the same task with the aid of technology. Some projects have been launched to achieve the same result for various instruments, namely the piano [1] [2] [3] [4], albeit none has really focused on the classical guitar.

Historically, most studies on pitch detection [5] [6] focused on general cases, in which the algorithms are not tailored to a specific group of instruments or a particular instrument. The famous YIN pitch detection algorithm [7], proposed in 2002, is accurate only for monophonic audio using a modified version of the Autocorrelation Function (ACF). Accordingly, concentrating on the classical guitar as a target instrument opens the gate for the possibility of being able to perform pitch detection with maximal accuracy [8].

The way the classical guitar is played introduces some natural constraints to possible frequency outputs. The most important constraint is the fact that only a maximum of 6 notes can be played at the same time, effectively eliminating the need of further analyzing a given time window if 6 frequency matches are found. A more advanced concept would be to also eliminate some frequencies based on their "impossibility" of being played, essentially utilizing the fact that each string is restricted to be able to output only one musical note at any given time. Our method does not implement this concept, although it could be the focus point of a future study. Another constraint is the frequency range of the classical guitar, which is approximately between 65Hz and 1330Hz. Lastly, limiting the analysis domain to only the classical guitar makes eliminating harmonics and overtones in the frequency domain representation much simpler. These constraints can help an algorithm to determine the confidence of an output with way more rigor and accuracy.

2. Materials and Methods

Typical steps for pitch detection tasks for music transcription include creating fundamental frequency candidates, eliminating overtones and harmonics and determining the duration of a musical note. The procedure of applying these steps makes way to the detection of note events, in which note events are an end product that get represented in the final presentation of the algorithm i.e. sheet music, midi file or tablature.

Concentrating on the domain of classical guitar introduces the aforementioned advantages for the task of music transcription. Thanks to the possibility of eliminating some difficulties, our algorithm can be constructed in a simpler way which involves pre-set parameters and well-defined mathematical operations, which would not only help reduce the computational complexity greatly, but also allow for future modifications for similar tasks e.g. pitch detection for another instrument.

All computations and graphics in this paper have been made and created with the Julia Programming Language [9] version 1.8.5. The reason for the choice of the Julia language is its performance, functional programming paradigm and rich library of scientific computing packages. Our method is implemented using the Plots [10], DSP [11] and Wav [12] packages. The DSP package uses the Radix-2 variant of the Cooley-Tukey FFT algorithm [13], which is developed to perform a Discrete Fourier Transform (DFT) with a computational complexity of O(nlog2n) instead of O(n^2), as is the case with the standard implementation of the DFT algorithm. One caveat of this algorithm is the fact that the input size has to be of a positive integer power of 2 e.g. 2048 or 4096. Thankfully, this can be circumvented with virtually no side effects.

For the purposes of music transcription, pitch detection using a Fourier Transform is performed by applying the transform to a small window of samples in the time domain representation of the audio signal. The length of this window of samples can be arbitrary, therefore defining a size complying with the restriction of the Radix-2 algorithm is trivial. For the last window in an audio signal, the desired window size can be achieved by zero-padding the signal i.e. adding samples of value 0 to the end of the signal to complete the length of the last window to the defined size. Such addition of zeroes do not cause any modification in the end result for the frequency domain representation of the audio signal as visually demonstrated by Nipun Ramakrishnan in a YouTube video [14]. However, it should be noted that excessive zero padding can cause inconsistencies in the relationships between different frequencies' relative powers.

Using the Discrete Fast Fourier Transform, time domain sample data is converted to its frequency domain representation. This has been an idea that has been used since the start of pitch detection algorithms. Nevertheless, the way that this data is processed and interpreted has been the main factor which decides the use case and effectiveness of a given algorithm. In our case, an approach similar to that of paper [5] can be used. A slight modification is made in the way that instead of matching ACF matches with the spectral peaks in the frequency domain representation, the relative dominance of the spectral peaks is used to determine whether a given frequency candidate is an intrinsic frequency or an overtone/harmonic. We have also decided to eliminate the need to automatically detect the window of samples that would be the most ideal to perform the analysis on. In other words, the algorithm relies on an external entity, either a human or another program, to select the time interval of the audio signal to be selected for analysis.

A simple algorithm, constructed by inspection and observation, can be applied to any frequency candidate in order to determine whether it should be counted as a note event or not. The algorithm has the following steps:

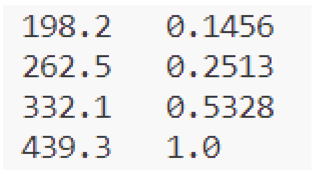
1. Begin with cleaning the data and setting parameters. The optimal signal length for this algorithm is determined to be 8196. Eliminate any frequencies that are not between 65 Hz and 1330 Hz from the DFFT results. Normalize the frequency power values such that the maximum value is mapped to 1 and all the other values are matched with the same coefficient. Apply an adjusting function on the found DFFT results such that the first 4 entries are multiplied by 50 and the remaining entries are multiplied by e^(x/80+0.3) where x is the index number of the frequency in the DFFT list. Note: Julia is 1-indexed by default.

2. Construct a table of frequencies that consist of frequencies available in the DFFT frequency buckets, in which the chosen frequencies from the buckets must correspond to being the next one to a musical frequency. For example, if the bucket contains the 437.3 Hz and 433.1 Hz frequencies, the list must contain 433.1 Hz since it is next to the musical note of A4 with frequency 440

Hz [15]. An exception is made when a DFFT frequency output is within 0.75 Hz of a musical note or if the DFFT frequency output is one of the first 5 outputs on the outputs list. For the case of the former, map the DFFT output to the closest musical note. For the case of the latter, map the DFFT output to the nearest musical note with a lower frequency.

3. Construct a list of 12 boolean values representing each fundamental musical note e.g. A, A#, B etc. These values correspond to whether or not the fundamental musical note has been found within the candidates or not.
4. From the DFFT results, any frequency that is in the musical notes list with a value above 0.1 should be considered as a frequency candidate and the boolean value representing its fundamental musical note should be set to true.
5. From the frequency candidates, if a frequency candidate is not the first candidate with the same fundamental musical note, an harmonics check is applied.
6. If 0.9 times of the previous frequency power on the general DFFT results is greater than the frequency candidate, the harmonics check deems to be failed and the candidate is no longer considered a match.
7. If the last two frequencies in the candidates list are of the same fundamental musical note, the candidate with the higher frequency is dropped as a candidate. Note: this may falsely eliminate a musical note performed in the audio, however, it is observed to eliminate more false positives than it does obscure real notes.
8. Display the remaining frequency candidates and their powers to the user as a final result.

   Such as in Figure 1:



```
198.2   0.1456
262.5   0.2513
332.1   0.5328
439.3   1.0
```

Figure 1: The left column represents frequencies and the right column represents their relative power.

3. Results and Discussion

The results of the method are of generally high accuracy and consistency. It is important to note that errors in the lower frequencies are due to the standard implementation of the Radix-2 algorithm yielding DFFT frequencies of same intervals whereas musical notes have frequency separation that grows larger when the note goes higher. It is equally as important to note that it is significantly easier to identify, predict and correct misidentified notes in the lower frequencies than it is in the high frequencies. For starters, some inconsistencies in the lower frequencies can easily be eliminated by determining that some notes would be impossible to play together on a standard 6-string guitar. If the end user of the program sees that the method has determined both G2 and G#2 as a musical note, they will know that only one of them has a possibility of being played. The method will not eliminate them automatically because it is believed that the end user will have more information to evaluate which note would be false, with a greater scope of knowledge, compared to the machine, about music theory and perhaps the musical piece where the audio was extracted from.

High frequencies rarely lead to inconsistent results, however, sometimes if 2 notes of the same fundamental musical frequency have been played simultaneously, an overtone might incorrectly be identified as a musical note. Nonetheless, it has been identified that the root cause of this issue is the misidentified note being actually audible in the selected audio segment. This leaves the end user with a couple of options: to play the note if possible, to ignore the misidentified note and not play it or evaluate it as being an "impossible" note i.e. not conventionally playable on the guitar together with the correctly identified notes.

Mid frequencies are where the method shines. It makes virtually no mistakes in identifying notes between A2 and E4. If a less than ideal audio window is selected, some mid frequency notes can be "lost" in the DFFT application. This is due to the nature of sound, mid frequencies are not perceived as powerful as they are when they are played with equal amplitude compared to low and high frequencies. It is possible to prevent this issue by carefully selecting the right audio window to capture all sounds with roughly equal amplitude. Another technique is to modify the threshold parameter, such as decreasing it to 0.08 from 0.1 (default). This can cause some false positives, but the end user might easily identify them if they notice their absence when the method is run with a higher threshold parameter.

Lastly, being a method developed for polyphonic sound, the method is not as accurate when used on single notes. It does identify the played note, although alongside one or two false positives. The end user can still apply the method on a single note audio window if they want to have an idea of the base musical note and figure out its frequency amongst them by themselves. For instance, if the method detects A2 and A3 as musical notes, the end user can listen to the audio window to decide between the two. A basic level of pitch perception is deemed enough to perform this task by convention.

Below is a table of chords played on the classical guitar, with their real musical notes and the musical notes identified by the method. Any difference in playing styles, recording equipment or selected audio window may yield varying results.

Tests made with other notes as the bases for the same chord structures yielded similar results. One outlier in these results is the first entry, ways to tackle this problem are described in the discussion section above.

Table 1: Some chords and their identified notes by the method

Chords	Identified Musical Notes
A Minor - A2 E3 A3 C4	G#2, A2, E3, A3, C4, A4
A Minor - C4, E4, A4, C5	C4, E4, A4, C5
A Minor 7 - G3, C4, E4, A4	G3, C4, E4, A4
E Major - E2, G#3, B3, E4	E2, F2, G#3, B3, E4
E 6/9 - E3, G#3, C#4, F#4	E3, G#3, C#4, F#4

4. Conclusions and Future Research

Thanks to its easy implementation, this method can be a very user-friendly and quick way to automatically determine musical notes from a given classical guitar audio. Known problems of the method are identified and possible ideas to overcome them were discussed in this paper. The proposed method also opens the gate for a lot of areas of future research. Firstly, a similar approach for other instruments can be applied to generate specialized pitch detection algorithms for them. Furthermore, machine learning analysis can be made on the sound signature of different instruments and generate the coefficients described in this paper automatically. Machine learning analysis can also be used to determine the ideal audio window to use for pitch detection, increase the precision of identifying harmonics and overtones and possibly offer the end user a way to train their own iteration of a pitch detection model, as in allowing them to interact with the results of the analysis by correcting the machine. Lastly, the implementation of the DFFT can be modified such that the frequency results overlap better with real musical frequencies. This could especially help with the aforementioned inconsistencies in the lower frequencies.

The future of music is going to be governed by how well technology can harness the abilities of the performer, and how the performer and the machine collaborate to take music to the next level. Either be it practicing, performing or composing, computers will play an even larger role in these domains as time elapses. Future analysis of musical instruments will also help computers to mimic instruments in the most natural and pure way. In the foreseeable future, machines will not take the place of humans for the driving seat of musical endeavours, but it is highly likely that they will be our co-pilot very soon.

5. References

[1] Miragaia, Rolando, et al. "Multi pitch estimation of piano music using cartesian genetic programming with spectral harmonic mask." 2020 IEEE Symposium Series on Computational Intelligence (SSCI). IEEE, 2020.

[2] Deng, Kai, Gang Liu, and Yuzhi Huang. "An efficient approach combined with harmonic and shift invariance for piano music multi-pitch detection." Fourth International Workshop on Pattern Recognition. Vol. 11198. SPIE, 2019.

[3] Liu, Shuchang, Li Guo, and Geraint A. Wiggins. "A parallel fusion approach to piano music transcription based on convolutional neural network." 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2018.

[4] Wang, Qi, Ruohua Zhou, and Yonghong Yan. "A two-stage approach to note-level transcription of a specific piano." Applied Sciences 7.9 (2017): 901.

[5] Kraft, Sebastian, and Udo Zölzer. "Polyphonic pitch detection by matching spectral and autocorrelation peaks." 2015 23rd European Signal Processing Conference (EUSIPCO). IEEE, 2015.

[6] Zhou, Ruohua, et al. "A computationally efficient method for polyphonic pitch estimation." EURASIP Journal on Advances in Signal Processing 2009 (2009): 1-11.

[7] De Cheveigné, Alain, and Hideki Kawahara. "YIN, a fundamental frequency estimator for speech and music." The Journal of the Acoustical Society of America 111.4 (2002): 1917-1930.

[8] Benetos, Emmanouil, et al. "Automatic music transcription: challenges and future directions." Journal of Intelligent Information Systems 41 (2013): 407-434.

[9] https://julialang.org/downloads/oldreleases/ accessed on August 20, 2023

[10] https://docs.juliaplots.org/stable/ accessed on August 20, 2023

[11] https://docs.juliadsp.org/latest/contents/ accessed on August 20, 2023

[12] https://docs.juliahub.com/WAV/bHvvS/1.1.0/ accessed on August 20, 2023

[13] Cooley, James W., and John W. Tukey. "An algorithm for the machine calculation of complex Fourier series." Mathematics of computation 19.90 (1965): 297-301.

[14] https://www.youtube.com/watch?v=yYEMxqreA10 accessed on August 17, 2023

[15] https://pages.mtu.edu/~suits/notefreqs.html accessed on September 2, 2023