Math 104C Final Project Report

Jinze Li, 7327737; Jiajia Wang, 3140357; Simon Wu, 6990410

June 2, 2021

1 Abstract

Digital signal processing utilizes numerous numerical algorithms in computers to analyze, modify, and synthesize various signals. Due to its linear nature, sound signal processing is a straightforward branch of signal processing that is common in everyday life. Since pitch of the sound is one of the most essential properties of sound – especially musical sound – it is essential to have an algorithm that can reliably identify the pitch or *perceived frequency of sound by human ears*. However, due to the existence of noise and harmonic frequencies in recordings of most instruments, finding the pitch is not as straightforward as finding the frequency. In this project, we will apply numerical manipulation in both the time domain and frequency domain to reliably identify the pitch of a single pitch instrument or human vocals both in musical settings.

2 Introduction

Sound is the recurring vibration of an object. Other than a few instruments such as the tuning forks, most musical instruments and natural objects – including human vocals generated through the vibration of the vocal cord – do not create a single frequency. Rather, following Newton's law of motion, most instruments produce a fundamental frequency f_1 and many corresponding harmonic frequencies f_n. The relative amplitude between these frequencies governs the timber of sound. The wavelength of the harmonic frequencies follows the harmonic series; thus in theory, fundamental frequency should be the the greatest common divisor of the harmonic frequencies. Take a vibrating string of a guitar, for example, the fundamental frequency is generated by the vibration of the entire string; the second harmonic is generated by each half of the string; the third is generated by each third of the string, and so on. However, it is noteworthy to point out that the fundamental frequency f_0 does no always have the largest magnitude.

Due to the upper limit of human hearing around 20,000Hz and computer's efficiency of handling data of length that is an integer power of two, most lossless sound files record sound at a discrete bitrate of 44,100Hz. In other words, the sound file captures the amplitude at $44,100$ points every second. These amplitude data points are in the *time domain*. We will apply Fast Fourier Transformation(FFT) to find the time domain data and get the amplitude at or near each discrete frequency bin. This is the *Frequency Domain*. Many numerical manipulations are applied in both domains to accurately and reliably identify the fundamental frequency despite the existence of discretization error, noise, and harmonic. The algorithm will sacrifice computational efficiency for the sake of accuracy.

3 Problem statement

In our project, we want to develop an algorithm to estimate the pitch of a piece of music accurately. use n to denote the time index, $f(n)$ denote the fundamental frequency of the music at time n, N denote the largest time index, $e(n)$ denote the frequency we estimate. We want to develop the algorithm to minimize

$$E = \sum_{n=0}^{N} |f(n) - e(n)|$$

It should be noticed that, besides fundamental frequency, a sound signal may also has many harmonic frequencies. In theory, the fundamental frequency should be the greatest common divisor of the harmonic frequencies, so we can find the fundamental frequency by computing the greatest common divisor of the peaks in frequency spectrum. However, in practice it is hard to do so due to the rounding error and the error caused by data processing and fft process. Therefore we apply several methods to distinguish between fundamental frequency and harmonic frequency.

4 Method

The pitch detection algorithm outline is as follows:

1. Extract file and identify the bitrate
2. Split-data into overlapping frames
3. Apply Hamming window function
4. Apply time domain zeros padding
5. Determine how many partials to perform in Harmonic Product Spectrum(HPS)
6. Apply FFT
7. Get partials of HPS
8. Apply high-pass and low-pass filter
9. Data stabilization
10. Find the fundamental frequency of each window using the HPS algorithm
11. Compare phase difference to estimate true frequency

4.1 Extract file and identify the bitrate

We mainly use the package: *scipy.io.wavfile* to read the wav file. By extracting the data from the wav file, we get the sample rate (bitrate) of the data and an array of data points which represent the amplitude of the sound. Normally, the sampling rate is 44.1 kHz and one second recording will produce 44,100 data-points.

4.2 Split-data into overlapping frames

Since the pitch of the music may vary quickly, we choose to split the data into windows of size 8192, and we increase accuracy of estimating the pitch at the edge of the windows by adding 50% overlapping between two adjacent windows. It is important to ensure value of window size is the power of 2 since we need to apply FFT to each window.

4.3 Applying Hamming window function

Denote the value of data point at index n by $x(n)$, the range of index of a window by $[a, b]$, one simple window function is defined by

$$w(n) = \begin{cases} 1, n \in [a, b] \\ 0, o.w. \end{cases}$$

However, this kind of window function will introduce new frequency components to the

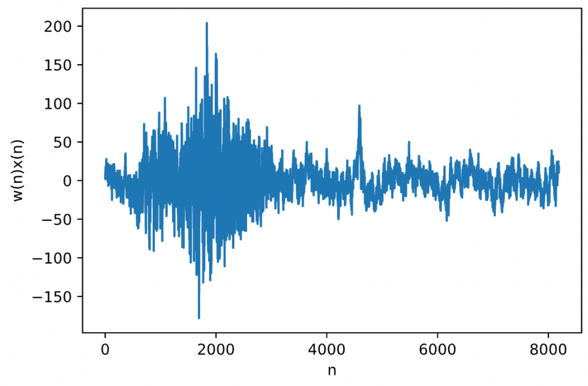

Figure 1: data before applying hamming window

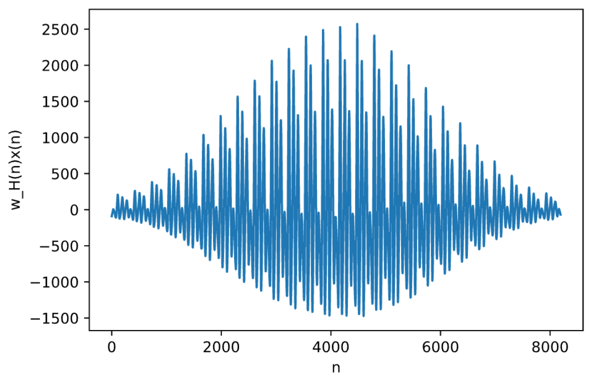

Figure 2: data after applying hamming window

signal. In order to minimize this effect, we apply a common window function called Hamming window, which is defined by

$$w_H(n) = 0.54 + 0.46cos(\frac{2\pi n}{N-1})$$

where N is the number of data points in the window. Finally, we use the formula $x_{new}(n) = w_H(n)x(n)$ to update the data in the window.

From the construction of hamming window, it is clear that $w_H(n)$ approach to 1 when n approach $\frac{1}{2(N-1)}$, and

$$0 < w_H(n) = 0.54 + 0.46cos(\frac{2\pi n}{N-1}) \leq 1.$$

Figure 1 and 2 also demonstrate the effect of hamming window on the data: it adds more weight to data close to the center of the window, and adds less weight to data on the edge.

In this process, we input the original data and output a list of weighted data ready for zero padding.

4.4 Time Domain Zero Padding

In earlier steps of the algorithm, the time domain data are first spitted into windowed frames. The default length of each frame is 2^{13} which equates to 0.18575 seconds. In this case, the density of *resolution* is fairly limited. For example, the true frequency of C_3 key on a piano (C small octave "Do") is 130.8128Hz; however, the nearest bins are 123.82Hz, 129.20Hz, and 134.58Hz. Thus, without zero padding, we can only identify the true frequency within 5.2Hz. In contrast, after applying zero padding such that the padded frame length is 4 times that of the original, the nearest bins are 129.120Hz, 130.55Hz, and 131.89Hz; the theoretical accuracy is within 1.4Hz, a acceptable amount.

The issue of a limited resolution poses a more serious issue for lower frequencies. This is because humans perceive sound in "octaves" or on the logarithmic scale. In other words, humans are more sensitive to the difference in frequency for lower frequencies than that of higher frequencies, and the difference in sensitivity follows an approximately logarithmic function of the frequency. A fact correlating to this is that the fundamental frequency of neighboring keys on a piano is closer together for lower notes. Since our algorithm is mainly for musical settings, the frequency we want to identify is usually between around 80Hz to around 1050Hz, the common frequency range of human singers. At this range, some nearby notes can be less than 5Hz apart, so the resolution of FFT will not be enough.

One way to address this is to use a larger frame size. However, a larger frame size usually does not work well with songs when the pitch can change many times within a few seconds. A solution for this is time domain zero padding. The process of time domain zero paddings is simple. After partitioning the original sound signal into frames and apply the hamming window weight function to each window, we simply add the desired amount of zeros at the end of each windowed frame. Note, in order to apply FFT efficiently, the total number of data points, including the zeros, should still be an integer power of two.

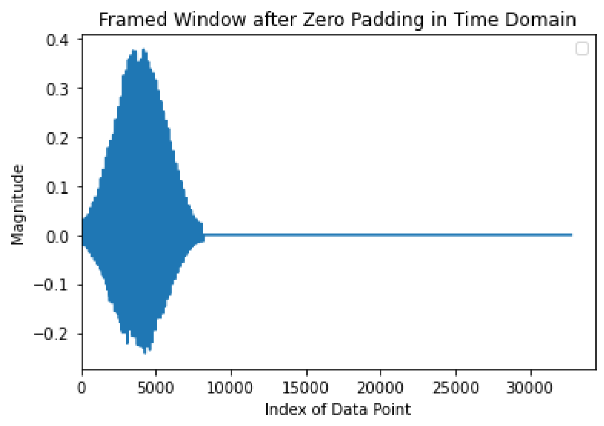

Figure 3: Frame after windowing and zero padding

I will now discuss the motivation for using zero paddings. Since the musical frequencies we are interested in are much lower than the highest frequency a 44,1000 bitrate signal can capture, the loss of accuracy when directly applying FFT is not because of the energy splitting between bins sharing the common denominator of the target frequency. In other words, the loss of accuracy is not because of the finite bitrate of the data. Rather, it comes from the gap between the discrete bins, as discussed above. By zero padding, we are essentially giving the FFT algorithm "dummy" data points such that the first bin can have a lower frequency. Because the gap between each bin is constant, the gap between each bin is decreased just as the first bin becomes close to the zeroth bin. In addition, other than some edge cases, adding zeros at the end of each frame does not change the existence of the frequencies in the original frame. These frequencies still "require" the coefficient of each frequency bin to be the right value in order to represent the data in the time domain. Thus, other than an increase in resolution, the properties of the frequency domain should remain the same. Lastly, since the window function compresses the edge value closer to zero, the start of the windowed frame naturally aligns with the zeros at the end, which reduces the artifact caused by using periodic functions to represent the data.

However, zero padding makes each frame a few multiples longer. Thus, this will introduce significant computation costs to the steps after. The algorithm, however, allows the user to disable or customize zero paddings if computational cost is a huge concern.

4.5 Determining Number of Partials in HPS and High-Pass Low-Pass Filters

The HPS algorithm requires the multiplication of a few downsampled frequency space data vectors. I will refer to the number of multiplication as a number of partials P. Note, $P = 1$ means we apply HPS multiplication one time. However, applying this multiplication an arbitrary amount of times may be undesirable. For one, applying too much multiplication introduce an unnecessary computational cost. Furthermore, as shown in Figure 5, harmonics of instruments tend to diminish, so only the first few harmonics are within the magnitude of the fundamental frequency. Therefore, having too many numbers of partials may be destructive to the fundamental frequency, allowing noise to overshadow it. On the other hand, if the partial number is too low, the HPS algorithm may not identify the fundamental frequency from the harmonics.

Thus the required number of partials is determined by the following algorithm.

$$P = min(floor(\frac{f_{lowpass}}{f_{highpass}}), 7)$$

Here, $f_{lowpass}, f_{highpass}$ refers to the low pass threshold and high pass threshold set by the user. Since the algorithm is only looking between the two thresholds, any fundamental frequency can at most have $floor(f_{lowpass}/f_{highpass})$ many harmonics within the thresholds; thus, HPS only requires this many partials. The maximum partials number 7 is obtained through experimentation. Due to the diminishing property of harmonics, 7 is usually enough to identify the harmonics and not be affected by noise. Note, the algorithm also allows users to input a specific number of partials.

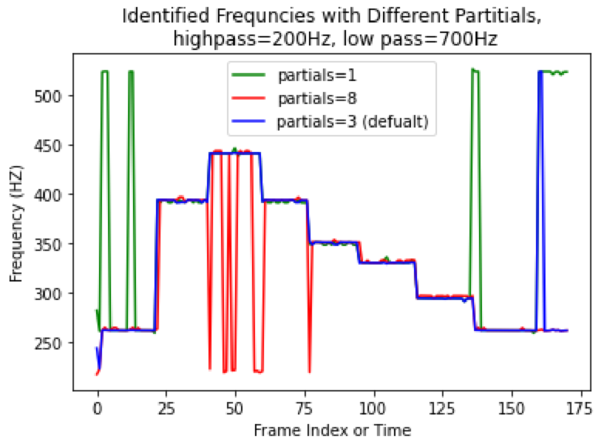

Figure 4: Number of partial's effect on accuracy

From the Figure 4, the algorithm used a partial that generate a much more accurate result. When using too few partials, HPS cannot separate the fundamental frequency from the harmonics; when too many, the HPS misidentify low frequency noise as the fundamental frequency.

Low-pass and high-pass filters are used within the HPS algorithm as well. In short, after sub sampling, the length of the frequency domain frame shortens. Usually, we add zeros to match the length to the original window; however, due to the frequency range we usually work with, we just apply Low-pass and high-pass filters to each partial. This reduces calculation complexity and focuses the search in the desired range.

4.6 Fast Fourier Transform

The Fast Fourier Transform (FFT) is an efficient way to implement the Discrete Fourier Transform (DFT) by dividing the data points into many smaller chunks. The FFT decomposes the time domain data into a summation of complex trigonometric functions (bins) of different periods; note, the periods follow the harmonic distribution. DFT takes in discrete values of signal data pints and outputs the coefficient or energy $(X[k])$of each bin. The definition of the DFT process of finite signal data points $x[n]$ is as follows:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-i2\pi nk/N}$$

The input signal and its DFT has the same length N. After FFT, the data under time frame are successfully transfer into the frequency frame. Moreover, the coefficient $X[k]$ of the DFT relates to a complex sinusoid whose normalized frequency is k/N. The precess of DFT can be perfectly invertible by inverse DFT so that we can transfer the signal from frequency frame back to the time frame.

A few problems still persist when directly applying FFT. First, as indicated by the

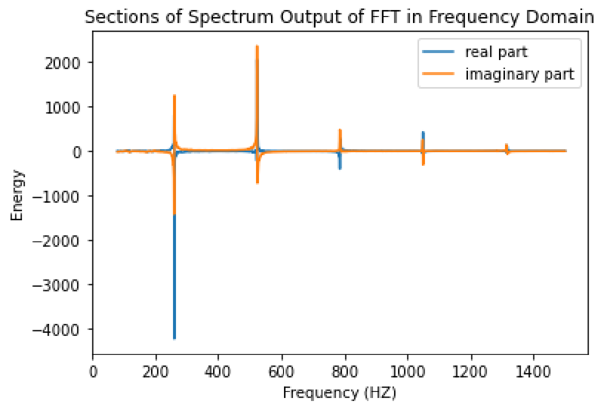

Figure 5: Sections of spectrum output of FFT of piano key C_4

graph, some harmonics have a similar magnitude as the fundamental frequency; second, often, the true frequency is between two bins, which overflow into nearby bins; third, there tends to be noise in the lower frequency; last, the frequency bins are not continuous and are relatively less denser for lower frequencies. To address these issues we apply the following manipulation respectively: use HPS to isolate the fundamental frequency, compare phase difference to estimate true frequency near each bin, apply low-pass and high-pass filter to isolate desired frequency, and use time domain zero padding to increase density of bins.

4.7 Harmonic Product Spectrum

After we transform the signal from the time domain into the frequency domain, we can see its spectrum should consist of a series of peaks corresponding to fundamental frequency and harmonic frequency. Since the harmonic frequencies are multiples of the fundamental frequency, we will compress the spectrum several times and compare it with the original spectrum. In general, the first peak of the original spectrum will correspond to the first peak of the second spectrum after downsampling by a factor of two. Similarly, the first peak of the original spectrum will correspond to the first peak of the third spectrum after downsampling by a factor of three. Hence, when the various spectrums are multiplied together, the result will form a clear peak at the fundamental frequency.

The following algorithm will apply to achieve HPS.

Algorithm 1: Harmonic Product Spectrum (HPS)

```
Input: data, sample rate, window
       length, hop length, partials: n
Result: Getting the fundamental
        frequency using HPS
1  Divide input signal by windows and
     apply Hamming window function;
2  Apply FFT denote as $F_{freq}$ and get
     original spectrum denote as $S_0$;
3  for $i \leq n+1$ do
4      Downsampling ($S_0$, len($S_0$)//$n$);
5      filter high pass and low pass;
6      if $len(S_i) \neq len(S_0)$ then
7          extent $S_i$ to equal length with
             zeros;
8      end
9  end
10 Spectrum: $prodS = \prod_{i=0}^{n} |S_i|$;
11 Fundamental frequency index: $x =$
     $\max\{prodS\}$;
12 Fundamental frequency: $F_{freq}[x]$;
```

4.7.1 Limitation of HPS

As we change the frequency we want to detect, the partial of the spectrum will also be altered. In this case, we will revise the number of compressed spectra to use, which may cause the algorithm to become computationally expensive and less accurate than before.

Another serious flaw of the HPS method is relative to calculate the same short FFT length. If we perform a fast FFT, the number of discrete frequencies we can consider is limited. To obtain higher resolution in our output, we need to perform more FFTs, which takes more time.

4.8 Data Stabilization

As Figure 5 suggest, the energy of the fundamental and harmonic frequencies are usually split between the real and imaginary part of two neighbouring bins. This can cause issue if the energy in the fundamental are evenly split while the harmonics are focused on one bin. Thus, we apply a stabilization algorithm on each HPS partials to combine the energy of neighbouring bins. Note, the frame are already reduced in length by applying low-pass and high-pass filter to reduce unnecessary calculations. The algorithm on the original data and each partials are given as follows:

$$\begin{aligned}\tilde{h}_k &= [|real(h_{k-1})| + |imag(h_{k-1})|] \\ &\quad + 4[|real(h_k)| + |imag(h_k)|] \\ &\quad + [|real(h_{k+1})| + |imag(h_{k+1})|]\end{aligned}$$

After applying stabilization, the stabilized partials are multiplied in the HPS algorithm. We used absolute value on the real and imaginary parts respectively instead of finding the norm to reduce computational complexity.

4.9 Estimating True Frequency through Phase Based Approach

The last step of our algorithm is estimating the true frequency by comparing phase difference. In short, after determining which bin represent the fundamental frequency, the algorithm compares that bin's phase – or angle in the complex plane – to that of the same bin in the previous frame to make small adjustments to the bin's frequency. It is a way to further improve the resolution of the FFT algorithm.

However, this step is optional as it only occasionally and marginally improve the accuracy. We created two version of the algorithm, one with this step and another without. Note, all data generated are using algorithm with this step. The reason for the unreliability of phase-based estimation is because it does not consider the energy split between neighbouring bins. Accounting for this split are much out of the scope of Math 104 series, and only marginally improves the accuracy since we already implemented time domain zero padding. Thus, we will not implement it in our algorithm for now.

5 Experiment Results

5.1 Results of HPS

After windowing and FFT process, we apply the HPS algorithm to the sample names "gingervoicelow.wav", we get the following result.

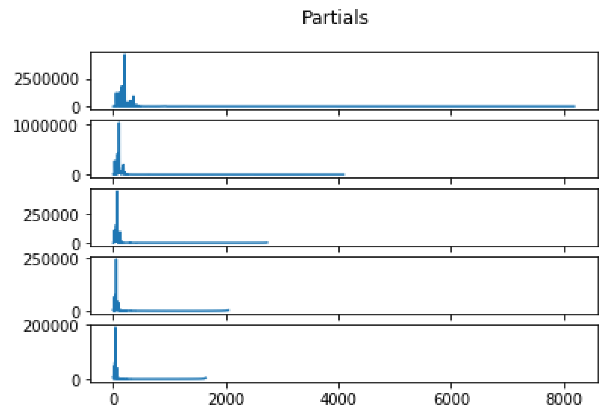

Figure 6: Frequency vs. magnitude for different partials in HPS

In this case, we can clearly see that the HPS peak for "gingervoicelow.wav" is about 25 index. After plug into the frequency we have the fundamental frequency for this clip is about 142 HZ.

5.2 Results of Overall Project

We tested our algorithm on a piano recording of C major and a simple song, "Twinkle Twinkle Little Star'. Bello

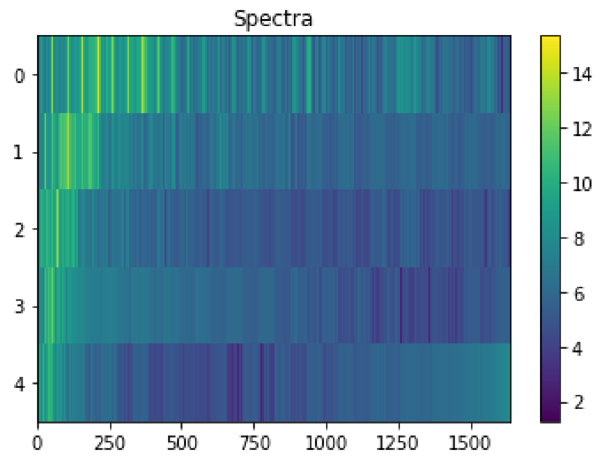

Figure 7: Combined spectrum

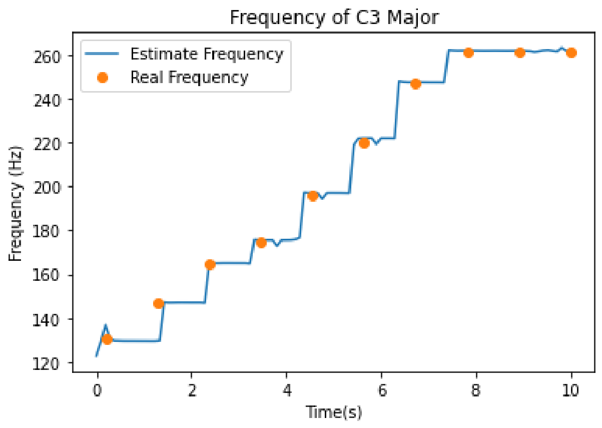

Figure 9: C3 major frequency compare with the real frequency

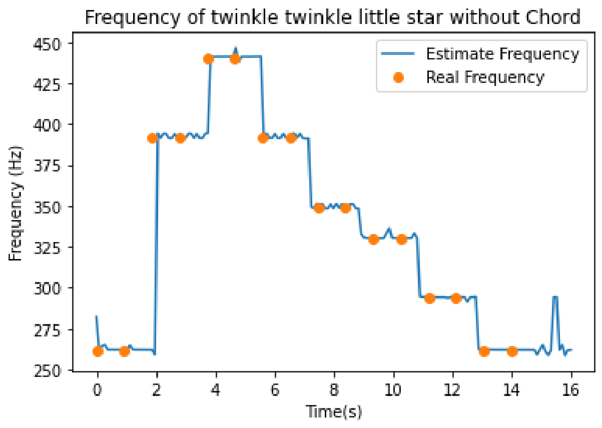

Figure 10: "Twinkle Twinkle Little Star" frequency compare with the real frequency

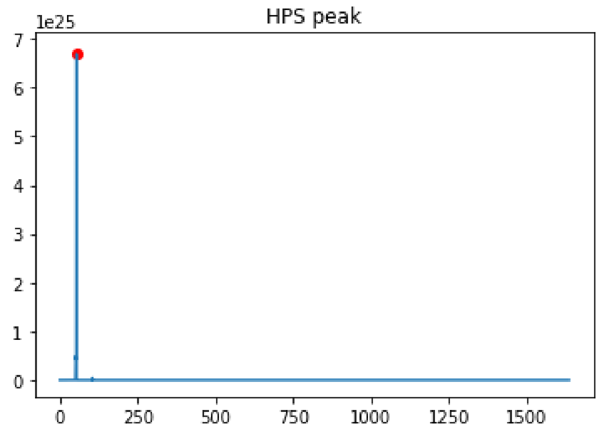

Figure 8: The plot of HPS peak

The Figure 9 and 10 suggest that our algorithm can accurately and reliably identify the pitch of a piece of simple music with no cords. What is more the low-pass and high-pass filter successfully prevented the noise from significantly distorting the results. Apart from some minor insignificant instability when the volume is low, the overall precision is very high. For example, the algorithm identifies the C_3 key as 129.4997Hz while the theoretical real frequency is 130.8128Hz. Overall, the algorithm can identify the pitch to within 1-2Hz of accuracy; considering the differences of frequency between piano keys are much grater, the result is very reliable.

The algorithm are also used to distinguish and identify the pitch of human singing. Figure 11 are the results. Although the output are not as stable, the accuracy is sufficient to identify its actually notes. The stability gets worse when the sound's volume are lower at the beginning or end of the signal. However, when distinguishable sound is present, the algorithm gives relatively consistent results.

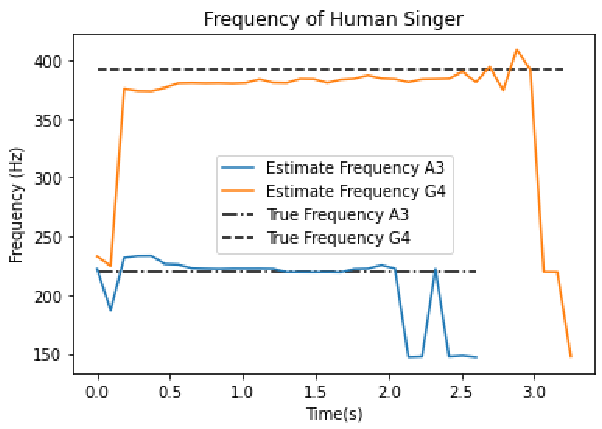

Figure 11: "Human singing frequency compare with the intended "real" frequency

6 Discussion

After we run the pitch detection algorithm, we can see that the fundamental frequency obtained is very similar to the natural frequency, which shows that our algorithm has successfully realized the tone detection of a single tone. But for chords, our algorithm is not that perfect. When we run the algorithm on chords, we can draw the following conclusions.

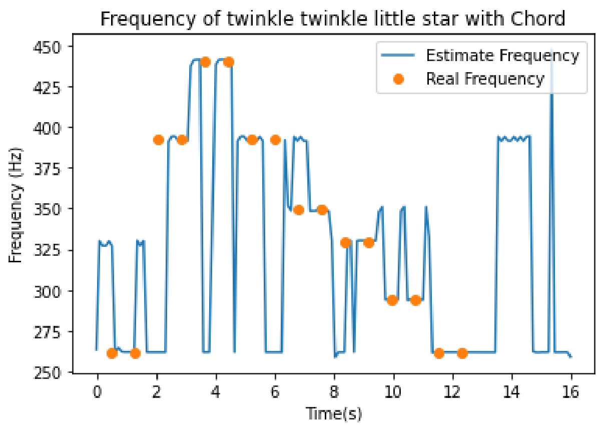

Figure 12: "Music with chord frequency compare with the real frequency; partials=1 (default)

When loud chords of a lower octave is present, the harmonic frequencies may be present in the search range, which are hard for the algorithm to distinguish. A way to partially counter this is to manually input an higher partial number and limiting the difference between the low-pass and high-pass filters.

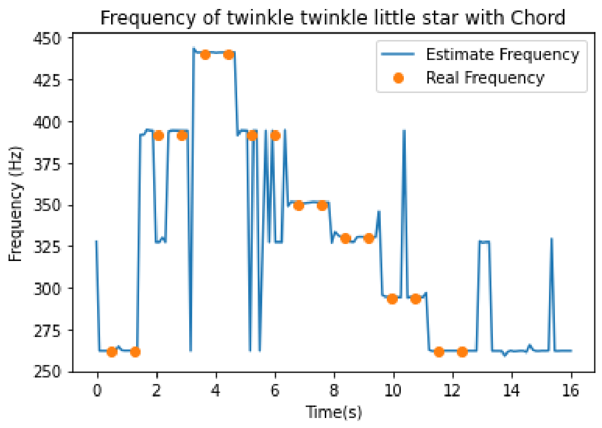

Figure 13: "Music with chord frequency compare with the real frequency; partials=4

The Figure above demonstrates a clear improvements by using a higher partials. Still, the algorithm are not designed to use with inputs with multiple tracks and cords.

From the above figure, we can see that our algorithm will become wildly inaccurate if we run the algorithm on the chord. This is because the frequencies of the two notes of a chord will affect each other, and for our algorithm, we are currently targeting a single sine curve. For multiple sine curves, we may need to separate them properly, then perform pitch detection on separate curves so that we can get better results.

We can try to further improve the accuracy in the case by manually input an higher partial number and decrease the range of low-pass and high-pass filters. However, this method still cannot get the accurate answer and also become computation expensive.

7 Summary

Our algorithm can reliably recognize the pitch or fundamental frequency of a sound signal, given some constrain on the complexity of the signal. Since we cannot know the exact pitch of an signal – due to imperfect tuning of instrument and imperfection of rhythm of the musician – it is hard to deduce the numerical summary of the error.

8 Future research

Our project could be improved in several ways:

1. we can optimize the algorithm further since currently, we sacrifice computation cost for accuracy.

2. we could try to find the exact frequency of the entire piece music to get the accuracy of our estimation numerically.

3. we could explore more about how to decrease the inaccuracy caused by chords and harmony.

4. we can consider the split of energy between two neighboring bins to further improve accuracy.Third, we can compare the output in log scale to the frequency of western music notes in log scale and output the notes directly.

5. we can develop other algorithms based on the output of the current one, such as auto tune or correction.

6. figure out some other better ways to achieve pitch shifting in the frequency frame

9 Acknowledgement

Sincere respect for all faculties of UCSB Math Department, and appreciation for the help from my Professor Paul's project instructions and materials and help on this report! Special thanks to Peisong Li at University of Hong Kong and Ann Li at University of California, Irvine for providing the music files.

10 References

References

[1] Tamara Smyth. *Music 270a: Signal Analysis*. Department of Music, University of California, San Diego..

[2] Mark Husband, Adan Galvan, Charlet Reedstrom, Richard Baraniuk. *ECE 301 Projects Fall 2003*. Dec 18, 2003.

[3] Jingjie Zhang, and Ziheng Chen. *A Pitch Shifting Reverse Echo Audio Effect*. Center for Computer Research in Music and Acoustics (CCRMA), Stanford University

[4] Shannon Hilbert. *FFT Zero Padding.* Bitweenie

[5] Peimani, Michael A. *Pitch correction for the human voice*. Diss. PhD. Thesis, University of California, Santa Cruz, 2009.