Extracting Music Notes from Chords by Applying and Improving Fourier Transform

Research Question: To what extent can the Fourier Transform be used and improved to be more helpful in chord musical notes extraction by eliminating the effect of overtones?

Word Count: 3878

Contents

1. Introduction

Given a random music clip featuring an instrument playing a melody, individuals with perfect pitch can often replicate the melody directly on their own instruments. Similarly, music sheets are traditionally composed by those with perfect or relative pitch, who painstakingly transcribe the notes they perceive. However, this process is labor-intensive, requiring numerous listens to accurately capture the nuances of the music.

The Fourier Transform, a mathematical tool with profound implications in signal processing, offers a method to simplify and expedite this process. By converting the time-domain signal of the music into its frequency components, the Fourier Transform allows us to identify the individual frequencies that make up the sound. This makes it a potentially powerful tool for extracting music notes from a given clip.

In this extended essay, I will explore the extent to which the Fourier Transform can be employed to analyze frequencies within a music clip, with the aim of accurately identifying the music notes being played. Additionally, I will investigate methods to refine this process, ensuring that overtones produced by musical instruments are filtered out, leaving only the fundamental frequencies that correspond to the actual notes of the melody. This study will not only highlight the strengths of the Fourier Transform in music analysis but also suggest potential improvements to enhance its effectiveness in practical applications.

2. Fourier Analysis

Suppose we have a continuous function $f(t)$ with the period T, then expanding $f(t)$ as sum of sine and cosine functions, as $f(t)$'s period is T, the sine and cosine components must have period $\frac{T}{n}$ for positive integer n, we have

$$f(t) = a_0 + \sum_{n=1}^{\infty} \left(a_n \cos\left(\frac{2n\pi}{T}t\right) + b_n \sin\left(\frac{2n\pi}{T}t\right)\right).$$

We aim to derive the constants a_0, and the coefficients of sine and cosine functions, a_n, b_n for positive integer n. This would rely on trigonometric identities.

2.1. Proving Integration Trigonometric Identities

For the constant a_0, integrate $f(t)$ on $t \in [0, T]$, since

$$\int_0^T \sin\left(\frac{2n\pi}{T}t\right) dt = 0, \tag{2.1}$$

$$\int_0^T \cos\left(\frac{2n\pi}{T}t\right) dt = 0, \tag{2.2}$$

the result of integration would be

$$\int_0^T f(t)dt = [a_0 x]_0^T = a_0 T,$$

the constant term

$$a_0 = \frac{1}{T}\int_0^T f(t)dt$$

To derive a_n and b_n coefficients, we would need to prove the following lemmas which will be useful for the calculation required to derive the coefficients for the Fourier series [1].

Lemma 2.1. *For all $m, n \in \mathbb{R}, x \in [0, 2\pi]$,*

$$\int_0^{2\pi} \sin(nx)\cos(mx)dx = 0.$$

Proof. Consider $\cos(mx), \sin(nx)$,

$$\int_0^{2\pi} \sin(nx)\cos(mx)dx = \int_0^{2\pi} \frac{\sin[(m+n)x] + \sin[(m-n)x]}{2}dx = 0$$

for all m and n. □

Lemma 2.2. *For $m, n \in \mathbb{R}$, $x \in [0, 2\pi]$*

$$\int_0^{2\pi} \cos(mx)\cos(nx) = \begin{cases} 0 & m \neq n \\ \pi & m = n \end{cases}$$

Proof. Consider $\cos(mx), \cos(nx)$

$$\int_0^{2\pi} \cos(mx) \cdot \cos(nx)dx = \int_0^{2\pi} \frac{\cos((m+n)x) + \cos((m-n)x)}{2}dx.$$

When $m \neq n$, $\int_0^{2\pi} \cos(mx) \cdot \cos(nx) = 0$. When $m = n$,

$$\begin{aligned} \int_0^{2\pi} \cos(mx) \cdot \cos(nx) &= \int_0^{2\pi} \frac{\cos((m+n)x) + \cos((m-n)x)}{2}dx \\ &= \int_0^{2\pi} \frac{\cos(2nx) + 1}{2}dx \\ &= 0 + \left[\frac{x}{2}\right]_0^{2\pi} = \pi \end{aligned}$$

□

Lemma 2.3. *For $m, n \in \mathbb{R}$, $x \in [0, 2\pi]$*

$$\int_0^{2\pi} \sin(mx)\sin(nx) = \begin{cases} 0 & m \neq n \\ \pi & m = n \end{cases}$$

Proof. Consider $\sin(mx), \sin(nx)$

$$\int_0^{2\pi} \sin(mx) \cdot \sin(nx)dx = \int_0^{2\pi} \frac{\cos((m-n)x) - \cos((m+n)x)}{2}dx.$$

When $m \neq n$, $\int_0^{2\pi} \sin(mx) \cdot \sin(nx) = 0$. When $m = n$,

$$\begin{aligned} \int_0^{2\pi} \sin(mx) \cdot \sin(nx) &= \int_0^{2\pi} \frac{\cos((m-n)x) - \cos((m+n)x)}{2}dx \\ &= \int_0^{2\pi} \frac{1 - \cos(2nx)}{2}dx \\ &= \left[\frac{x}{2}\right]_0^{2\pi} - 0 = \pi \end{aligned}$$

□

To derive the expression of a_n, we multiply $f(x)$ by $\cos(\frac{2m\pi}{T})t$ for $m \in \mathbb{Z}^+$,

$$\begin{aligned}
&\int_0^T f(t) \cdot \cos\left(m\frac{2\pi}{T}t\right) dt \\
=&\int_0^T \left(\frac{a_0}{2} \cdot \cos\left(\frac{2m\pi}{T}t\right) + \sum_{n=1}^{\infty} a_n \cos\left(\frac{2n\pi}{T}t\right)\cos\left(\frac{2m\pi}{T}t\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{2n\pi}{T}t\right)\cos\left(\frac{2m\pi}{T}t\right)\right) dt \\
=&\int_0^T \frac{a_0}{2} \cdot \cos\left(\frac{2m\pi}{T}t\right) dt + \int_0^T \left(\sum_{n=1}^{\infty} a_n \cos\left(\frac{2n\pi}{T}t\right)\cos\left(\frac{2m\pi}{T}t\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{2n\pi}{T}t\right)\cos\left(\frac{2m\pi}{T}t\right)\right) dt
\end{aligned}$$

In this equation, as a_0 is constant, and according to Equation (2.2), the first term $\int_0^T a_0 \cdot \cos(\frac{2m\pi}{T}t)dt$ is 0.

Then by Lemma 2.1, all the $b_n \sin(\frac{2n\pi}{T}t)\cos(\frac{2m\pi}{T}t)$ terms can be counted as 0. Also, according to Lemma 2.2, we know that all the $a_n \cos(\frac{2n\pi}{T}t)\cos(\frac{2m\pi}{T}t)$ terms with $n \neq m$ can be eliminated, and the only term left with is when n, which ranges from 0 to infinity, equals to m.

Therefore, what we are left with is

$$\begin{aligned}
\int_0^T f(t) \cdot \cos\left(\frac{2m\pi}{T}t\right) dt =& \int_0^T a_m \cos\left(\frac{2m\pi}{T}t\right) \cdot \cos\left(\frac{2m\pi}{T}t\right) dt \\
=&a_m \int_0^T \frac{\cos\left(\frac{2m\pi \cdot 2}{T}t\right) + \cos(0)}{2} dt \\
=&a_m \left[\frac{t}{2}\right]_0^T = a_m \cdot \frac{T}{2}.
\end{aligned}$$

Therefore, we have

$$a_n = \frac{2}{T}\int_0^T f(t)\cos\left(\frac{2n\pi}{T}t\right) dt.$$

Similarly, to find b_n, we try to eliminate a_n by finding the orthogonal function of $\cos(\frac{2n\pi}{T}t)$, so multiply $f(t)$ by $\sin(\frac{2m\pi}{T}t)$ for $m \in \mathbb{Z}^+$,

$$\begin{aligned}
&\int_0^T f(t) \cdot \sin\left(m\frac{2\pi}{T}t\right) dt \\
=&\int_0^T \left(a_0 \cdot \sin\left(\frac{2m\pi}{T}t\right) + \sum_{n=1}^{\infty} a_n \cos\left(\frac{2n\pi}{T}t\right)\sin + \sum_{n=1}^{\infty} b_n \sin\left(\frac{2n\pi}{T}t\right)\sin\left(\frac{2m\pi}{T}t\right)\right) dt \\
=&\int_0^T a_0 \cdot \sin\left(\frac{2m\pi}{T}t\right) dt + \int_0^T \left(\sum_{n=1}^{\infty} a_n \cos\left(\frac{2n\pi}{T}t\right)\sin\left(\frac{2m\pi}{T}t\right) + \sum_{n=1}^{\infty} b_n \sin\left(\frac{2n\pi}{T}t\right)\sin\left(\frac{2m\pi}{T}t\right)\right) dt.
\end{aligned}$$

Similarly, Equation (2.1) implies that the first term $\int_0^T a_0 \cdot \sin(\frac{2m\pi}{T}t)dt = 0$, and by Lemma 2.1 and 2.3, we can cancel most terms here except the $b_m \sin(\frac{2m\pi}{T}t)\sin(\frac{2m\pi}{T}t)$ terms,

$$\begin{aligned}
\int_0^T f(t) \cdot \sin\left(\frac{2m\pi}{T}t\right) dt =& \int_0^T b_m \sin\left(\frac{2m\pi}{T}t\right) \cdot \sin\left(\frac{2m\pi}{T}t\right) dt \\
=& b_m \cdot \frac{T}{2},
\end{aligned}$$

resulting in

$$b_n = \frac{2}{T}\int_0^T f(t)\sin\left(\frac{2n\pi}{T}t\right)dt.$$

Therefore, the coefficients a_0, a_n, b_n have been derived

$$a_0 = \frac{1}{T}\int_0^T f(t)dt, \tag{2.3}$$

$$a_n = \frac{2}{T}\int_0^T f(t)\cos\left(\frac{2n\pi}{T}t\right)dt, \tag{2.4}$$

$$b_n = \frac{2}{T}\int_0^T f(t)\sin\left(\frac{2n\pi}{T}t\right)dt, \tag{2.5}$$

for $n \geq 1$, [1].

2.2. Fourier Series in Complex Form

Applying Euler's Formula [1], we have

$$e^{it} = \cos t + i\sin t$$
$$e^{-it} = \cos t - i\sin t,$$

So we can express $\cos t$ and $\sin t$ as

$$\cos t = \frac{e^{it}+e^{-it}}{2}, \qquad \sin t = (-i)\cdot\frac{e^{it}-e^{-it}}{2}.$$

Therefore,

$$\begin{aligned} f(t) &= a_0 + \sum_{n=0}^{\infty}\left[a_n\cos\left(\frac{2n\pi}{T}t\right) + b_n\sin\left(\frac{2n\pi}{T}t\right)\right] \\ &= a_0 + \sum_{n=0}^{\infty}\left[a_n\cdot\frac{e^{i\frac{2n\pi}{T}t}+e^{-i\frac{2n\pi}{T}t}}{2} - i\cdot b_n\cdot\frac{e^{i\frac{2n\pi}{T}t}-e^{-i\frac{2n\pi}{T}t}}{2}\right] \\ &= \sum_{n=0}^{\infty}\left[\frac{a_n - ib_n}{2}e^{i\frac{2n\pi}{T}t} + \frac{a_n+ib_n}{2}e^{-i\frac{2n\pi}{T}t}\right]. \end{aligned} \tag{2.6}$$

In the equation (2.6), the part $\frac{a_n+ib_n}{2}$ can be expressed as

$$\begin{aligned} \frac{a_n+ib_n}{2} &= \frac{1}{2}\left(\frac{2}{T}\int_0^T f(t)\cos(\frac{2n\pi}{T}t)dt + i\cdot\frac{2}{T}\int_0^T f(t)\sin(\frac{2n\pi}{T}t)dt\right) && \text{plugging in (2.4) and (2.5)} \\ &= \frac{1}{T}\int_0^T f(t)\left[\cos\left(\frac{2n\pi}{T}t\right) + i\sin\left(\frac{2n\pi}{T}t\right)\right]dt && \text{combining the integrals} \\ &= \frac{1}{T}\int_0^T f(t)\cdot e^{i\frac{2n\pi}{T}t}dt && \text{applying Euler's Formula} \end{aligned} \tag{2.7}$$

Similarly, for $\frac{a_n - ib_n}{2}$, it can be expanded to

$$\begin{aligned}\frac{a_n - ib_n}{2} &= \frac{1}{T}\int_0^T f(t)\left[\cos\frac{2n\pi}{T}t - i\sin\frac{2n\pi}{T}t\right]dt \\ &= \frac{1}{T}\int_0^T f(t)\cdot e^{-i\frac{2n\pi}{T}t}dt \end{aligned} \tag{2.8}$$

For convenience, we define $c_n(t)$ to be

$$c_n = \frac{a_n - ib_n}{2} = \frac{1}{T}\int_0^T f(t)\cdot e^{-\frac{2n\pi}{T}t}dt$$

and notice that the new expression for $\frac{a_n + ib_n}{2}$ is just by switching the sign of t, we have

$$c_{-n} = \frac{a_n + ib_n}{2} = \frac{1}{T}\int_0^T f(t)\cdot e^{\frac{2n\pi}{T}t}dt$$

Notice that we can transform (2.6) to

$$\begin{aligned} f(t) &= \sum_{n=0}^{\infty}\left(c_n\cdot e^{i\frac{2n\pi}{T}t} + c_{-n}\cdot e^{-i\frac{2n\pi}{T}t}\right) \\ &= \sum_{n=0}^{\infty} c_n\cdot e^{i\frac{2n\pi}{T}t} + \sum_{n=0}^{\infty} c_{-n}\cdot e^{-i\frac{2n\pi}{T}t} && \text{expanding the sum} \\ &= \sum_{n=0}^{\infty} c_n\cdot e^{i\frac{2n\pi}{T}t} + \sum_{n=0}^{-\infty} c_{-(-n)}\cdot e^{-i\frac{2(-n)\pi}{T}t} && \text{replace all } n \text{ in the second sum to } -n \\ &= \sum_{n=-\infty}^{\infty} c_n\cdot e^{i\frac{2n\pi}{T}t} && \text{combining the range of the 2 identical sums} \end{aligned}$$

so we can generalize $f(t)$ to

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{i\frac{2n\pi}{T}t}, \qquad c_n = \frac{1}{T}\int_0^T f(t)\cdot e^{-i\frac{2n\pi}{T}t}dt.$$

3. Fourier Transform

The Fourier series, with the angular frequency $\omega = \frac{2\pi}{T}$, we have

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{in\omega t}, \qquad c_n = \frac{1}{T}\int_0^T f(t)\cdot e^{-in\omega t}dt$$

as we mentioned before, can represent periodic function $f(t)$ with the period T.

In real world applications, signals are frequently represented as non-periodic , where $T \to \infty$. Also we want to get all finite frequency ω, $n \to \infty$, therefore resulting in $\omega = \frac{2\pi}{T} \to d\omega, n\omega \to \omega, c_n = c(\omega)d\omega$, [2],

therefore we have

$$\begin{aligned} c_n &= \lim_{T\to\infty} \frac{1}{T}\int_0^T f(t)\cdot e^{-in\omega t}dt \\ \implies C(\omega)d\omega &= \lim_{T\to\infty} \frac{1}{T}\int_0^T f(t)\cdot e^{-i\omega t}dt \\ &= \frac{d\omega}{2\pi}\int_0^\infty f(t)\cdot e^{-i\omega t}dt \\ \implies C(\omega) &= \frac{1}{2\pi}\int_0^T f(t)\cdot e^{-i\omega t} \end{aligned}$$

And thus the expression of $f(t)$ also changes to

$$f(t) = \int_0^\infty C(\omega)d\omega e^{i\omega t} = \int_0^\infty C(\omega)e^{i\omega t}d\omega$$

This is the method of going from the non-periodic time domain signal $f(t)$ to a frequency domain spectrum $C(\omega)$. And by modern convention, we instead define $F(\omega) = 2\pi C(\omega)$ for convenience. Hence we got the expression of Fourier Transform and the Inverse Fourier Transform,

$$\text{Fourier Transform: } F(\omega) = \int_0^\infty f(t)e^{-i\omega t}dt \tag{3.1}$$

$$\text{Inverse Fourier Transform: } f(t) = \frac{1}{2\pi}\int_0^\infty F(\omega)e^{i\omega t}dt \tag{3.2}$$

3.1. Discrete Fourier Transform

In practical applications (with the help of computers), the input signals cannot be continuous, but instead discrete. So we instead use the discrete Fourier transform with only slightly different from the continuous Fourier transform [1]:

$$F(\omega) = \int_0^\infty f(t)e^{-i\omega t}dt \implies F_k = \sum_{n=0}^{N-1} f_n \cdot e^{-i\frac{2\pi kn}{N}}.$$

3.1.1. An Example of Discrete Fourier Transform

We try to demonstrate an easy case to show the discrete Fourier transform procedure to analyze a wave's frequency.

With the sampling frequency of 8Hz, assume we sampled $N = 8$ discrete signals of $f(t) = \sin(\frac{t}{2\pi})$ in 1 second, with the numerical results below

$$\begin{aligned} x_0 &= 0 & x_1 &= 0.707 \\ x_2 &= 1 & x_3 &= 0.707 \\ x_4 &= 0 & x_5 &= -0.707 \\ x_6 &= -1 & x_7 &= -0.707. \end{aligned}$$

Then we can calculate the frequency bins by plugging those values in the discrete Fourier transform formula

$F_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i\frac{2\pi kn}{N}}$, we have

$$\begin{aligned}
F_1 &= 0 \cdot e^{-i\frac{2\pi(1)(0)}{8}} + 0.707 \cdot e^{-i\frac{2\pi(1)(1)}{8}} + 1 \cdot e^{-i\frac{2\pi(1)(2)}{8}} + \cdots + (-0.707) \cdot e^{-i\frac{2\pi(1)(7)}{8}} \\
&= 0 + 0.707[\cos(-\frac{\pi}{4} + i\sin(-\frac{\pi}{4})] + 1[\cos(-\frac{\pi}{2}) + i\sin(-\frac{\pi}{2})] + \cdots + (-0.707)[\cos(-\frac{7\pi}{4}) + i\sin(-\frac{7\pi}{4})] \\
&= 0 + (0.5 - 0.5i) + (-i) + (-0.5 - 0.5i) + (0.5 - 0.5i) + (-i) + (-0.5 - 0.5i) \\
&= -4i
\end{aligned}$$

Similarly,

$$\begin{aligned}
F_2 &= 0 \cdot e^{-i\frac{2\pi(2)(0)}{8}} + 0.707 \cdot e^{-i\frac{2\pi(2)(1)}{8}} + 1 \cdot e^{-i\frac{2\pi(2)(2)}{8}} + \cdots + (-0.707) \cdot e^{-i\frac{2\pi(2)(7)}{8}} \\
&= 0 + (-0.707i) + (-1) + (0.707i) + (0.707i) + (1) + (-0.707i) \\
&= 0
\end{aligned}$$

By the same method, we can calculate all the Fourier coefficients:

$$\begin{aligned}
F_0 &= 0 \qquad F_1 = -4i \\
F_2 &= 0 \qquad F_3 = 0 \\
F_4 &= 0 \qquad F_5 = 0 \\
F_6 &= 0 \qquad F_7 = 4i,
\end{aligned}$$

noticing that only F_1 and F_7 have nonzero Fourier coefficients. Since the samples are contributed by a 1Hz sine wave $f(t) = \sin\left(\frac{t}{2\pi}\right)$, $F_1 \neq 0$ makes sense. For the nonzero Fourier coefficient F_7, it is due to the impossibility to measure the frequency larger than half of the sampling frequency, known as the Nyquist limit [3]. For this situation, what we do instead is get rid of all the frequencies above the Nyquist limit and double all the Fourier coefficients below it, $F_1' = -8i$.

Since $F_1' = -8i$, plotting in the complex plane, it can be visualized as a vector with the amplitude 8 and phase $\theta = \frac{3\pi}{2}$. The magnitude $|F_1'| = \sqrt{0^2 + (-8)^2} = 8$. Because we use 8 samples to operate the discrete Fourier transform, we need to average it out to get the amplitude $A = \frac{8}{8} = 1$. The $\theta = \frac{3\pi}{2}$ marks the phase difference between the cosine wave and our sampling wave, which is the sine wave, this can be corresponds the magnitude/phase form of two-sided series [4].

3.2. Speeding Up – Fast Fourier Transform

Consider the discrete Fourier transform, with N samples in total, we end up with calculating N frequency bins, and with each frequency bin we have the summation of N multiplication of distinct N samples and the exponential, we end up with in total N^2 operations to run in the computer, so the complexity would be $O(n^2)$ and will take a long time to compute with a reasonable large sampling frequency.

The fast Fourier transform (FFT) algorithm is thus applied to speed up the computation of the Fourier transform, taking advantage from the periodic nature of the trigonometric waves. The computation can be

simplified by splitting the indices to even and odd indices [5]:

$$
\begin{aligned}
F_K &= \sum_{n=0}^{N-1} x_n \cdot e^{-i\frac{2\pi kn}{N}} \\
&= \underbrace{\sum_{m=0}^{N/2-1} x_{2m} \cdot e^{-i\frac{2\pi k(2m)}{N}}}_{\text{Even Index}} + \underbrace{\sum_{m=0}^{N/2-1} x_{2m+1} \cdot e^{-i\frac{2\pi k(2m+1)}{N}}}_{\text{Odd Index}},
\end{aligned}
$$

and this splitting trick can be applied repeated, until we only have one sample to compute, then combine all the values until we arrive at the frequency bins.

4. Related Music Theory

4.1. Fundamental Tone and Its Overtone

For many musical instruments, string instruments (i.e. piano, guitar, etc) and wind instruments (i.e. oboe, flute, etc) in specific, unlike tuning fork, which produces a simple sinusoidal wave with only one single frequency, these instruments have their timbres enriched by producing not only just the fundamental frequency but also the harmonics. Different musical instruments can provide fundamental frequency and harmonics with distinct combinations of the intensity of frequencies, resulting in their unique timbres.

For string instruments, waves traveling in opposite directions are commonly produced when a wave is reflected from a fixed boundary. The original wave and the reflected wave interfere to produce the standing waves. More complicated, with small energy dissipated, the waves can travel back and forth between boundary points, creating different standing waves. However, the possible standing waves must meet one requirement – the two ends must be nodes of the standing wave (where the amplitude of the point remains to be 0) as the two ends are fixed points on the string.

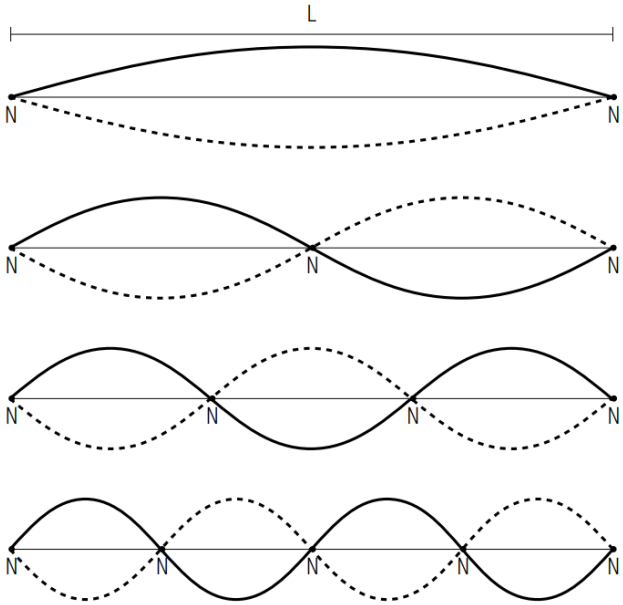


Figure 1: The 1st, 2nd, 3rd, 4th (from top to bottom) harmonics of the standing wave in string instruments formed, [6].

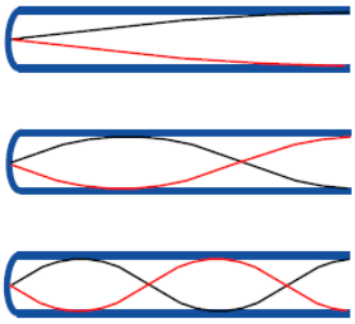


Figure 2: The 1st, 3rd, 5th (from top to bottom) harmonics of the standing wave in wind instruments formed, [7].

Therefore, to calculate the harmonics, we have that for the string instruments, when the string length L is fixed, the fundamental frequency, the smallest frequency, provided by the first harmonic in the standing wave, where the relation between the wave $\lambda = 2L$ and the fundamental frequency can be expressed as

$$f_0 = \frac{v}{\lambda} = \frac{v}{2L}.$$

For the harmonics, consider the nth harmonics, the wavelength of the harmonics $\lambda' = \frac{2}{n}L$ and the harmonics frequency can be expressed as

$$f_n = \frac{v}{\lambda'} = \frac{v}{\frac{2}{n}L} = n\frac{v}{2L} = nf_0.$$

Therefore, the harmonics are in the frequencies $nf, n \in \mathbb{Z}^+$.

A little bit differently, for the wind instruments, with the functioning part simplified as a tube, it has an opening end and a closed end. The air moves "back and forth", creating longitudinal waves in the tube. Since the air cannot oscillate back and forth at the closed end of the tube, it is limited to be a node, while air at the open end of the tube is free to move, it acts as the antinode. The first three possible harmonics are shown in the Figure 2. Therefore, for the wind instruments, while the length of tube L is fixed, the fundamental frequency $f_0 = \frac{v}{\lambda} = \frac{v}{4L}$. And consider the nth harmonics, since the length of tube is $L = (\frac{n-1}{2} + \frac{1}{4})\lambda_n = \frac{2n-1}{4}\lambda_n \implies \lambda_n = \frac{4L}{2n-1}$

$$f_n = \frac{v}{\lambda_n} = \frac{v}{\frac{4L}{2n-1}} = (2n-1)\frac{v}{4L} = (2n-1)f_0,$$

the harmonics frequency is also of integer multiples of the fundamental frequency, $(2n-1)f_0$ for $n \in \mathbb{Z}^+$.

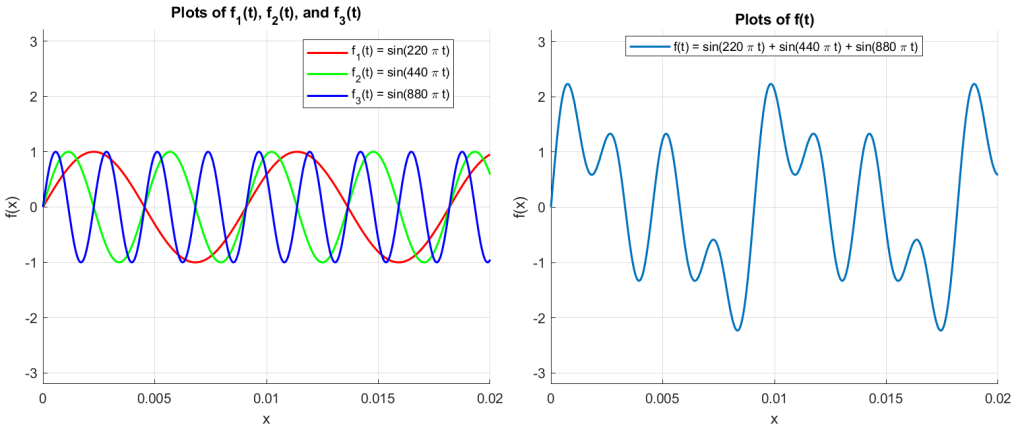
4.2. Chord and Single Notes

A chord is a group of 2 or more notes played simultaneously, building blocks of harmony and also enrich the music piece. The waveform of the chord follows the superposition rule of the separate waves of single notes, [8]. If we have tuning forks producing standard sinusoidal waves (without overtones) [9] in A2, A3, A4, they will produce sine waves in frequency $f = 110, 220, 440$Hz, its wave is the simple superposition of separate

A2, A3, A4 waves, in the functions of

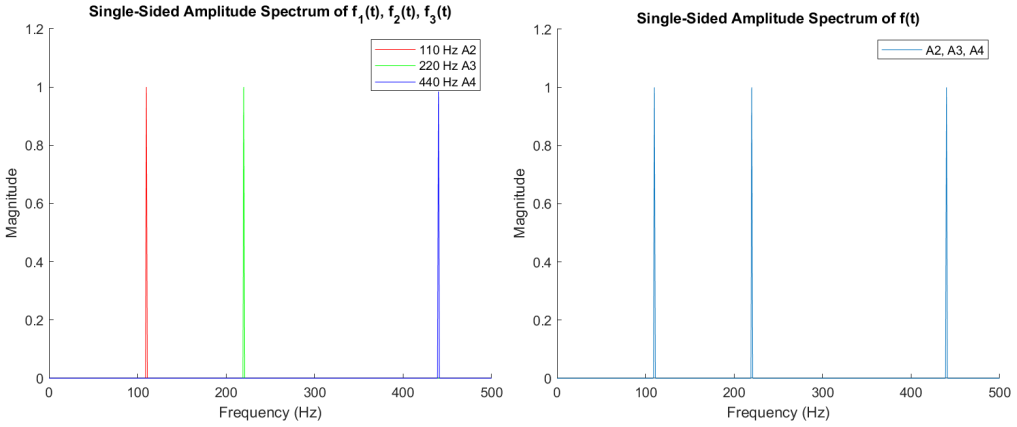
$$\begin{aligned} f_1(t) &= \sin(220\pi t) \\ f_2(t) &= \sin(440\pi t) \\ f_3(t) &= \sin(880\pi t) \\ \implies f(t) &= f_1(t) + f_2(t) + f_3(t) \\ &= \sin(220\pi t) + \sin(440\pi t) + \sin(880\pi t), \end{aligned}$$

and the superposed results are shown in Figure 3 (a) and (b). By Fourier Transform, the corresponding



(a) Original f_1, f_2, f_3 functions

(b) Superposed f function



(c) Original Fourier transform results of f_1, f_2, f_3

(d) Superposed Fourier transform results of f

Figure 3: (a), (b): Plots of $f_1(t), f_2(t), f_3(t)$ and $f(t) = f_1(t) + f_2(t) + f_3(t)$ as their superposed results; (c), (d): Plots of the frequency spectrum of $f_1(t), f_2(t), f_3(t)$ and $f(t)$.

frequency domain would also be the simple superposition of the frequency domains of the simple functions, as shown in Figure 3(c) and (d).

5. Approaches for Overtone Disturbance Elimination

General FFT analysis can only show result of frequencies and their magnitudes. With disturbance by overtones, we cannot directly see the notes forming the chords. So we aim to eliminate the impact of

overtones. Despite their contribution to the richness of timbre, their existence, instead, counts as the disturbance in our extracting music notes process. Overtones do not represent the actual note being played, and their varying intensities across different fundamental frequencies can result in their magnitude surpassing that of the fundamental frequency, complicating the note extraction process.

For example, here we have a chord with notes numbered 1, 2, 3 played, their fundamental frequencies are noted as f_1, f_2, f_3 respectively, and they will have overtones in frequencies in multiples of their fundamental frequencies. Suppose that they have their frequency spectrums analyzed results of the frequencies and their magnitudes listed in Figure 4. Suppose that the fundamental frequency of note 2 is two times the

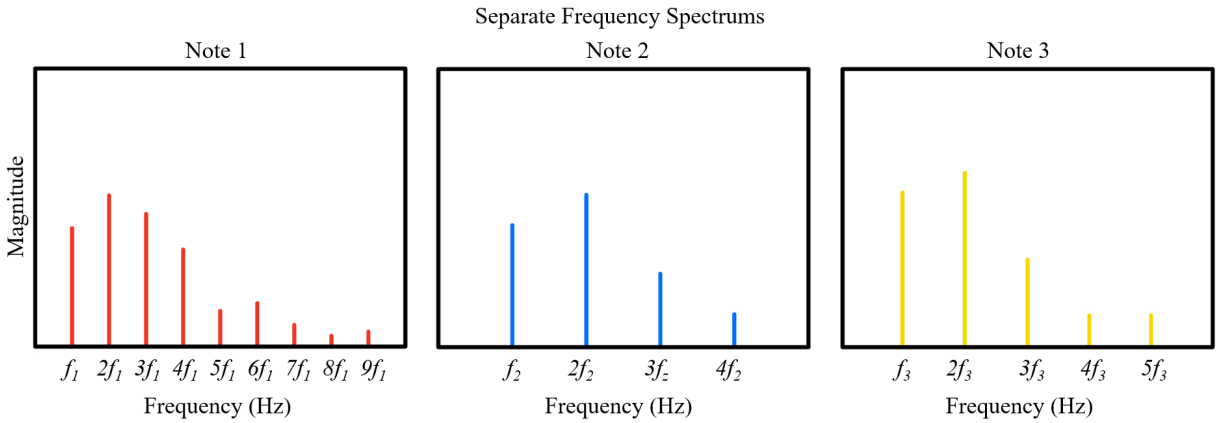


Figure 4: Separate theoretical frequency spectrums of note 1, 2, and 3, with fundamental frequencies f_1, f_2, f_3.

fundamental frequency of note 1, $f_2 = 2f_1$ (implying that they are a pair of octave notes), then note 1 and 2 have frequencies in common. So if we analyze the chord played, the frequency spectrum should be a combined spectrum of the three notes, with the intensity of repeating frequencies in notes 1 and 2 superposed to a higher magnitude. The frequency spectrum is shown in Figure 5.

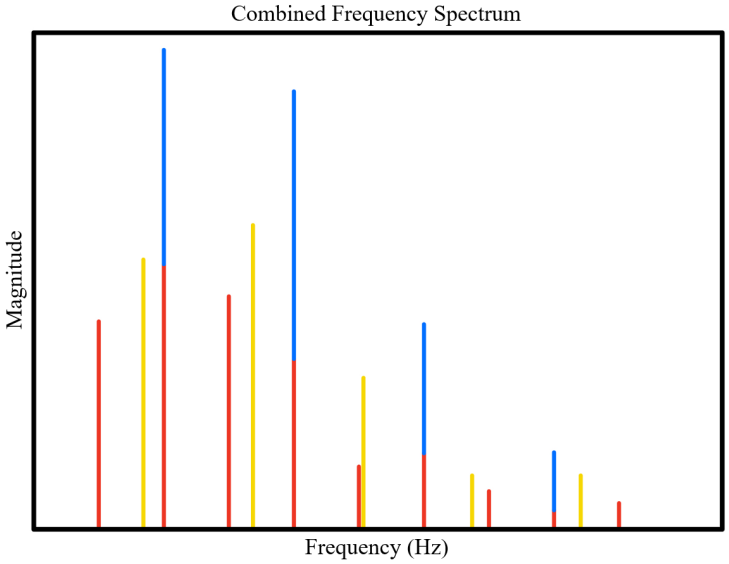


Figure 5: The theoretical frequency spectrum of the chord, with colored peaks labeling the note contributed to such intensity (red, blue, yellow for notes 1, 2, 3 respectively). On the $2nf_1 = nf_2, n \in \mathbb{Z}^+$ positions the intensities of the frequencies are contributed by both notes 1 and 2.

With the frequency of analyzing results in Figure 5, it would be hard to determine the actual chords played. Our aim would be to eliminate the overtones' impact on the frequency analysis, with the following three approaches taken into consideration.

Approach A: Eliminate the Frequency Multiples A straightforward approach represented in Figure 6 involves detecting the lowest frequency (Figure 6 (a)). Because there are no lower frequencies, it can only be identified as a fundamental tone. Therefore, it is the base tone of the note played and what we are seeking for. To eliminate the influence of its overtones, we simply disregard all frequencies that are integer multiples of this lowest frequency (Figure 6 (b)); Then iterate to find the second lowest frequency, and repeat this procedure.

This operation will continue until there are no higher frequencies, and all the frequencies presented are not integer multiples of one another.

However, this method is prone to errors when multiple notes are played simultaneously. When the fundamental frequency of one note overlapped with the other's overtone (note 1 and 2 in the example). By discarding frequencies as multiples of the fundamental frequency, there exists a highly possible, substantial risk of omitting essential chord notes (note 2), as shown in Figure 6.

Consider the example we introduced in Figure 5, according to this approach, the processing steps and result are shown in Figure 6.

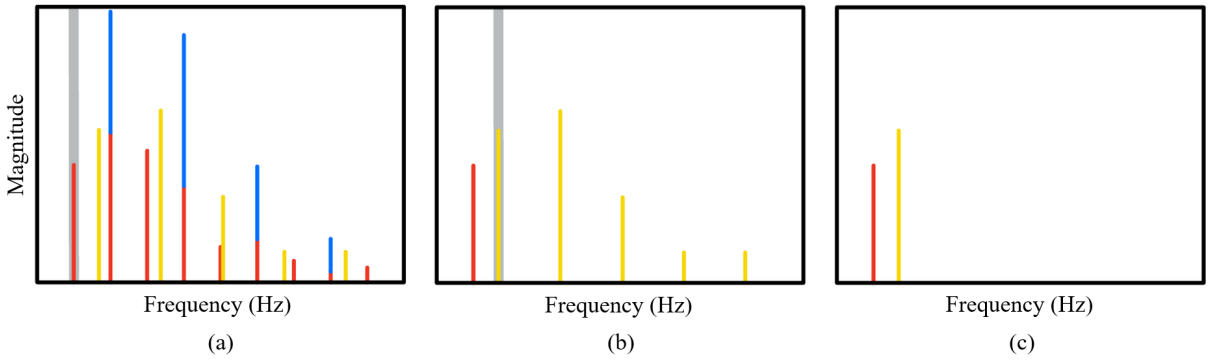


Figure 6: (a) Pick the lowest frequency from the frequency analysis, in this example, f_1; (b) Eliminate all frequencies in the form nf_1, choose the second lowest frequency, f_3; (c) Eliminate all frequencies in the form nf_3.

Approach B: Recognition Boundary Set-Up A more refined method involves analyzing the intensity of the frequencies present and selecting those with intensities above a specified threshold for representation, also used by Jeff Heaton for musical note extraction [10].

However, the intensities of overtones may exceed the intensity of its fundamental tone. Figure 7 lists frequency spectrum of C1, C4, C6, and C8, the fundamental frequencies (first obvious peaks) of C1, C4, C8 have lower intensity than their overtones.

Additionally, this method also encounters difficulties when dealing with chords. In the example demonstrated in Figure 8, intensity of shared overtones of note 1 and 2 may combine, resulting in an overtone with a higher intensity than either of the fundamental frequencies, exceeding the intensity boundary we set up to determine whether it is counted as the fundamental frequency of a note or not (Figure 8 (a)). Consequently, overtones may be incorrectly identified as a distinct note in the output, despite not representing actual

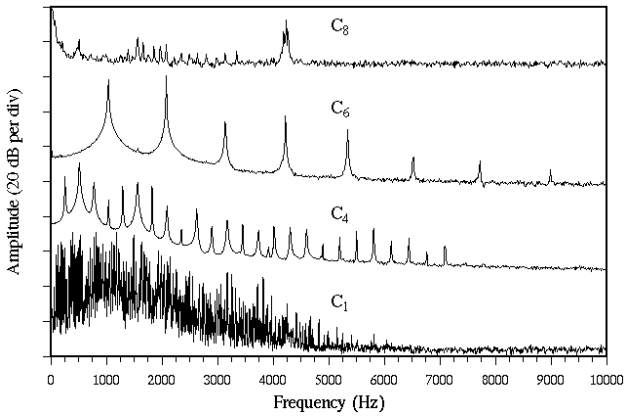


Figure 7: Frequency spectrum for piano notes C1, C4, C6, and C8 [11].

note played – the result shown in Figure 8 (b) contains overtones $2f_2 = 4f_1$ and $2f_3$, but the fundamental frequency of note 1 is excluded.

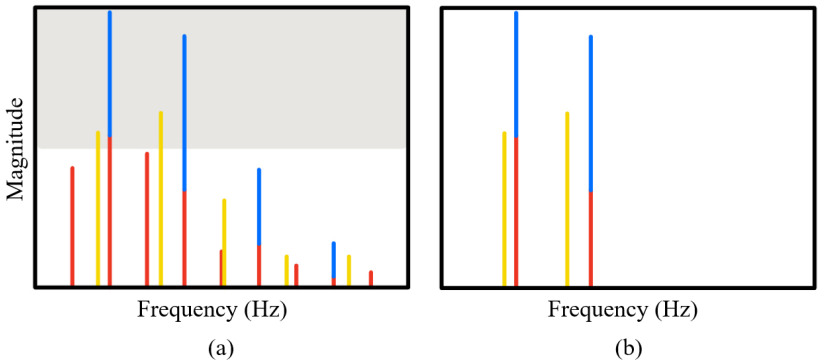


Figure 8: (a) Check all frequencies by the boundary; (b) Eliminate all frequencies with magnitudes lower than the boundary

Approach C: Combined Approach To address the limitations of the previous methods, we propose a more sophisticated approach that combines the ideas from both Approaches A and B.

With a predetermined boundary of the intensity that filters all the frequencies with obvious intensities, by reviewing all of them we can find the lowest frequency $f_{\min}$, the shaded red spike (fundamental frequency of note 1) in the frequency spectrum in Figure 9 (b).

Since the lowest frequency $f_{\min}$ cannot be the integer multiple of any other frequencies $nf \geq f$ for $n \in \mathbb{Z}^+$, it must be the fundamental frequency of a note played. Then for the note that has fundamental frequency $f_{\min}$, it also produces other harmonics with certain intensities – other red spikes.

Checking all the frequencies with intensity above the boundary, if we find a multiple of the lowest frequency, $nf_{\min}$, we can reduce the known note (with the fundamental frequency $f_{\min}$)'s influence to the frequencies that are its overtones, $nf_{\min}$, by subtracting the intensity part contributed as the overtone, while we don't know whether part of their intensities are contributed by other notes played/overtones of other notes played in the audio, resulting in Figure 9 (c). The subtraction would require an existing database of the intensity proportion of the overtone with respect to their fundamental frequencies, which can be determined

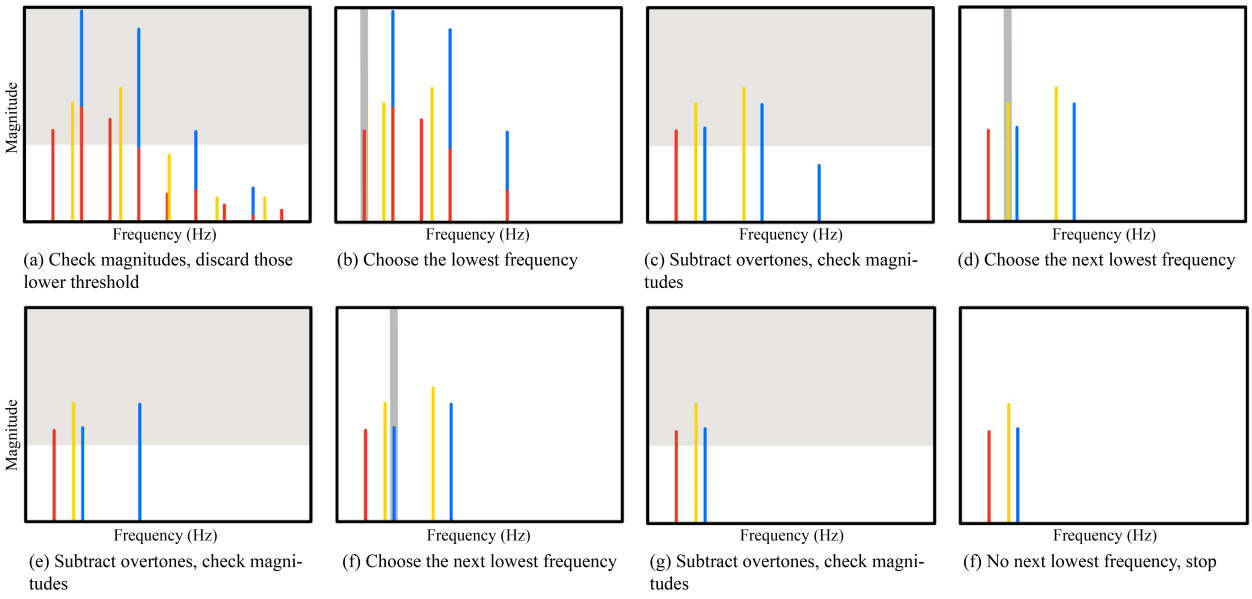


Figure 9: For Approach C, we combine the Approach A and B by doing the following: first check magnitudes and discard those with lower threshold, choose the lowest frequency, subtract overtones' magnitudes contributed by the fundamental frequency, and repeat until there has no next lowest frequency.

by the algorithm demonstrated in §6 Data Collection.

After this subtraction, we review through the remaining frequencies against the predetermined intensity boundary again (Figure 9 (c)). The next step involves detecting the second lowest frequency that exceeds this threshold (Figure 9 (d), the shaded yellow spike representing the fundamental frequency of note 2), which should also be a fundamental frequency of another higher note played, and applying the same intensity subtraction process to all the frequencies with intensity above the boundary (Figure 9 (d)). This procedure is iteratively repeated until no new frequencies above the threshold remain.

This combined approach offers a more reliable method for accurately identifying and preserving the fundamental frequencies of notes, while effectively mitigating the interference caused by overtones.

6. Data Collection

By evaluating the approaches, **Approach C** seems to be the most helpful one for notes recognition.

For the subtracting operation of the magnitudes, we need matrices for notes containing their frequencies and corresponding magnitudes, and it needs the data collection progress from single notes.

The normal FFT result would present a long list of frequencies and their corresponding magnitudes, and the plot will contain small nonzero magnitude values for frequencies that are not their fundamental frequencies nor overtone frequencies, as shown in Figure 10, the result of FFT for piano C4 note.

This will not help us with using **Approach C** to analyze the notes representing in the audio. Instead, we would like to have a neat matrix containing the frequencies and magnitudes of the peaks generated by the fundamental frequency and its overtones.

To achieve that, we can first divide the frequency domain into intervals[1] Then, find the local maximum in

[1]For data collection of a single note's fundamental frequency and overtones, the interval length can be 20Hz because the

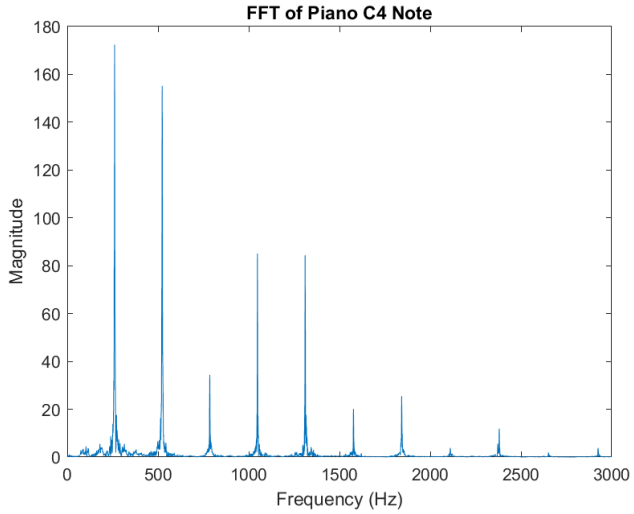
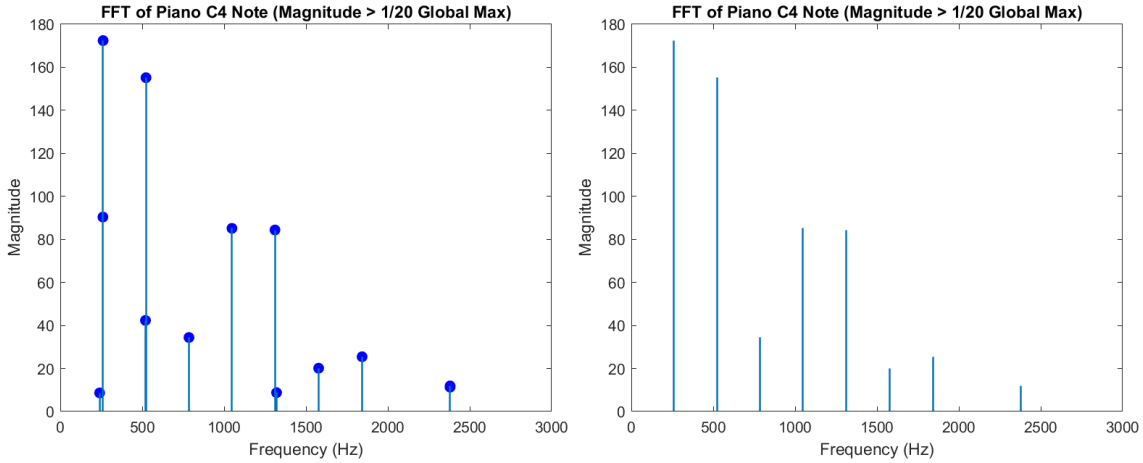


Figure 10: The raw FFT frequency spectrum derived from the wave of piano C4 note.

each interval, and compare them with the magnitude of the global maximum. A boundary (1/20 of the global maximum) is used to examine all local minima with magnitude higher than 1/20 of the global maximum and cancel out others. and the corresponding matrix representing C4's frequency and magnitude is



(a) (b)

Figure 11: (a): Spikes of the maxima in every 20Hz intervals; (b): Spikes after eliminating the lower spikes in nearby intervals.

$$\texttt{freq\_mag\_C4} = \begin{bmatrix} 239.8 & 259.8 & 261.0 & \cdots & 1577.2 & 1842.6 & 2379.8 & 2380.2 \\ 8.6 & 90.4 & 172.4 & \cdots & 20.1 & 25.5 & 11.3 & 11.9 \end{bmatrix}.$$

However, here we see that there exist multiple spikes representing one frequency on the plot, for example, $239.8, 259.8, 261$ and $2379.8, 2380.2$ in the `freq_mag_C4` matrix. It forms because the increasing trend caused by the peak frequency spans over one interval. For this situation, we first measure the lowest frequency through the `freq_mag` matrix, if its value is greater than 40Hz, implying that there do not exist actual

lowest fundamental frequency on piano is 27.5Hz. However, for chords if close notes are played together (i.e. A3 and A#3 with fundamental frequencies 220Hz and 233Hz, smaller interval should be applied.

different peaks that we want to appear in adjacent intervals; we choose the one with higher magnitude and eliminate the other.

The updated matrix `freq_mag_C4` now is

$$\texttt{freq\_mag\_C4} = \begin{bmatrix} 261.0 & 523.0 & 785.4 & 1048.0 & 1310.6 & 1577.2 & 1842.6 & 2380.2 \\ 172.4 & 155.2 & 34.4 & 85.1 & 84.4 & 20.1 & 25.5 & 11.9 \end{bmatrix}.$$

For convenience of future calculation, the matrix is normalized to

$$\texttt{freq\_mag\_C4} = \begin{bmatrix} 261.0 & 523.0 & 785.4 & 1048.0 & 1310.6 & 1577.2 & 1842.6 & 2380.2 \\ 1 & 0.900 & 0.199 & 0.494 & 0.489 & 0.117 & 0.148 & 0.069 \end{bmatrix}$$

by taking the magnitude of the lowest fundamental frequency as 1 and adjusting others proportional.

Similarly, we can use this program to analyze other notes' frequencies and magnitudes and derive their `freq_mag` matrices.

7. Output Enhancement

Because we would like to present the output as notes instead of frequencies, we need the process of transforming frequencies to notations.

A very commonly used standard note, A440 (sometimes called A4), is the 440 Hz tone that serves as the internationally recognized standard for musical pitch, as the reference when tuning. A440 is the musical note A above middle C.

According to the equal-tempered chromatic scale, which states that an octave is divided into twelve semitones based on a logarithmic frequency axis, each note has a frequency that is $2^{1/12}$ times that of the one below it.

Also for standardization, applying the MIDI (Musical Instrument Digital Interface) note numbers, we can associate to each pitch $p \in [0, 127]$ with a centered frequency f measured in Hz. In the MIDI system, the pitch A4 (440 Hz) is noted as $p = 69$, serving as a reference. Therefore, we can derive the following relation between the centered frequency and MIDI note number for all note numbers,

$$f = 2^{(p-69)/12} \cdot 440$$

and reversely, to calculate the MIDI note from the frequency, we have

$$p = 69 + 12 \log_2(f/440)$$

Also, in piano, its 88 distinct notes from A0 to C8 can be converted to MIDI pitch in the range $p \in [21, 108]$.

We can convert the MIDI note number p to the scientific pitch notation by defining the chroma list as `chroma = ['A ', 'A#', 'B ', 'C ', 'C#', 'D ', 'D#', 'E ', 'F ', 'F#', 'G ', 'G#']`, and the scientific pitch notation for the musical note with the MIDI note number p should be `'chroma[(p-69) % 12]'+'p//12-1'` [12, 13].

For example, when $p = 101$, `(p-69)% 12` $\equiv 101 - 69 \equiv 8 \pmod{12}$, `chroma[(p-69)% 12]=chroma[8]=F`, `p//12-1` $= \lfloor \frac{p}{12} \rfloor - 1 = 8 - 1 = 7$. So the scientific pitch notation for the musical note with MIDI note number $p = 101$ is F7.

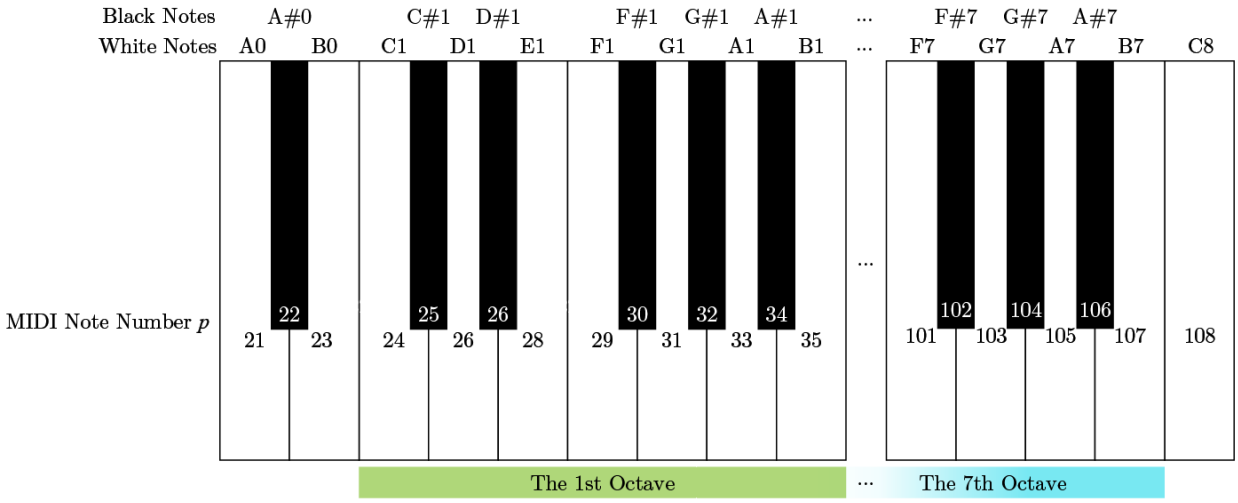
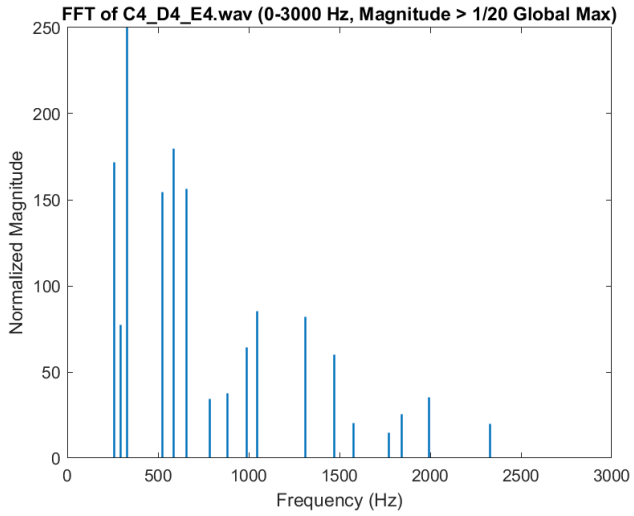


Figure 12: The music notes on a piano, starting from A0 to C8, the corresponding MIDI note has the range $p \in [21, 108]$. The notations (for example, A0) at the top are the scientific pitch notations. All the black notes are noted in sharp notations #, raising the lower pitch by one semitone.

8. Algorithm Presenting

8.1. Example 1 – C4, D4, E4

Suppose that we are playing the three keys C4, D4, E4 together, the first step is the same as data collection, deriving such a matrix for the chord.



`freq_magnitude` =

$$\begin{bmatrix} 261.0 & 294.4 & 329.4 & 523.0 & 587.4 & 659.2 & 785.4 & 881.6 & \cdots & 1771.0 & 1842.6 & 1992.0 & 2331.0 \\ 171.8 & 77.1 & 276.6 & 154.4 & 179.8 & 156.2 & 34.2 & 37.7 & \cdots & 14.8 & 25.6 & 35.5 & 19.8 \end{bmatrix}$$

Obeying the procedure in **Approach C**, we select the lowest frequency and its magnitude $[261.0; 181.9]$,

examine which note it is presenting (C4), then call `freq_mag_C4` that's already in the derived variable list,

$$\texttt{freq\_mag\_C4} = \begin{bmatrix} 261.0 & 523.0 & 785.4 & 1048.0 & 1310.6 & 1577.2 & 1842.6 & 2380.2 \\ 1 & 0.900 & 0.199 & 0.494 & 0.489 & 0.117 & 0.148 & 0.069 \end{bmatrix},$$

compare the first row of `freq_magnitude` and `freq_mag_C4`, find their frequencies in common (due to the measuring error, find the frequencies within difference smaller than 3Hz), we have

$$\begin{bmatrix} \texttt{matching\_freq} \\ \texttt{matching\_mag\_C4} \\ \texttt{matching\_mag\_chord} \end{bmatrix} = \begin{bmatrix} 261.0 & 523.0 & 785.4 & 1048.0 & 1310.6 & 1577.2 & 1842.6 \\ 1 & 0.900 & 0.199 & 0.494 & 0.489 & 0.117 & 0.148 \\ 171.8 & 154.4 & 34.2 & 85.2 & 81.8 & 20.1 & 25.6 \end{bmatrix}.$$

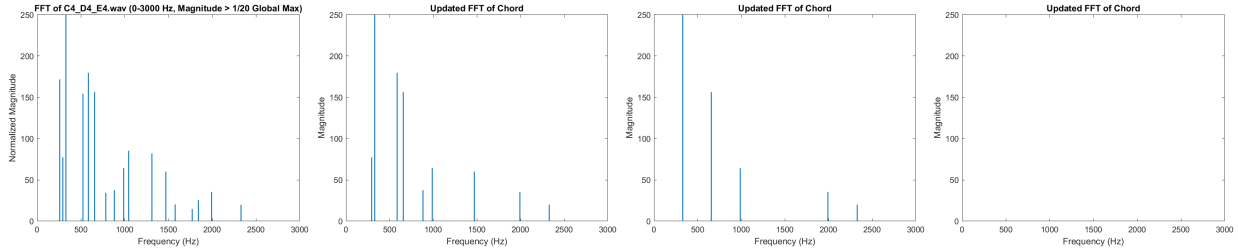
Since the first column, the lowest frequency can be assured that is fully contributed by C4, we update `matching_mag_chord` by

$$\begin{aligned} \texttt{matching\_mag\_chord} &= \texttt{matching\_mag\_chord} - \texttt{matching\_mag\_chord}[0] \cdot \texttt{matching\_mag\_C4}/\texttt{matching\_mag\_C4}[0] \\ &= \begin{bmatrix} 0.0 & -0.2200 & 0.0118 & 0.3308 & -2.2102 & -0.0006 & 0.1736 \end{bmatrix} \\ \text{(take 1 decimal place)} \implies &\approx \begin{bmatrix} 0.0 & -0.2 & 0.0 & 0.3 & -2.2 & 0.0 & 0.2 \end{bmatrix} \end{aligned} \tag{8.1}$$

Then it was updated back to the `freq_mag_chord`,

$$\texttt{freq\_mag\_chord} = \begin{bmatrix} 261.0 & 294.4 & 329.4 & 523.0 & 587.4 & 659.2 & 785.4 & 881.6 & \cdots & 1771.0 & 1842.6 & 1992.0 & 2331.0 \\ 0.0 & 77.1 & 276.6 & -0.2 & 179.8 & 156.2 & 0.0 & 37.7 & \cdots & 14.8 & 0.2 & 35.5 & 19.8 \end{bmatrix}$$

Applying the boundary, frequencies with low magnitudes are removed if their intensities are lower than the boundary (here the threshold is set as `magnitude >= threshold = 10` of the highest magnitude). Then we can get the updated FFT cancelling the first note C4 and its overtones impact, also some frequencies with low magnitudes.



(a) Original Chord C4, D4, E4 (b) Impact of note C4 eliminated (c) Impact of note D4 eliminated (d) Impact of note E4 eliminated

Figure 13: The results shown of every notes and their overtones eliminated by certain magnitudes until there is no frequencies with magnitude larger than 10.

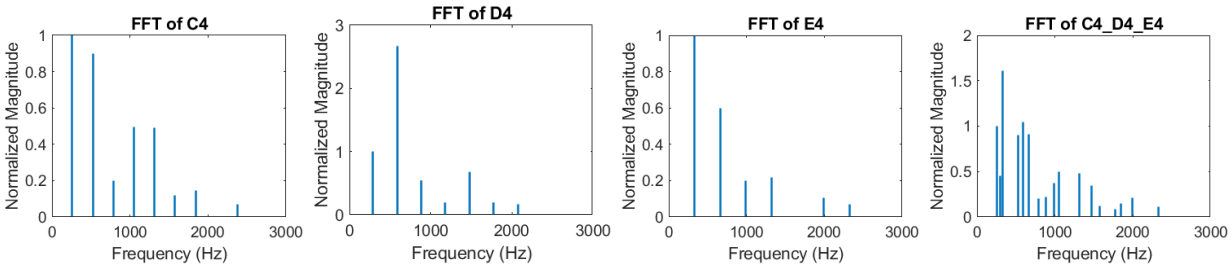
By applying the same algorithm repeatedly until all entries in the matrix `freq_mag_chord` are deleted due to low magnitudes, we can derive all the existing notes played in the chord:

```
pitch: C4, frequency: 261, magnitude: 171.7776
pitch: D4, frequency: 294.4, magnitude: 77.1226
pitch: E4, frequency: 329.4, magnitude: 276.5716
```

The result is highly accurate.

The varying difference in the magnitudes of the fundamental frequencies of different notes is likely due to

the distribution of the strength of the keys. For example, in Figure 14(b), D4's fundamental frequency does not have the highest magnitude, so the result also shows a lower magnitude (magnitude: 77.1) compared to C4 (magnitude: 171.8) and E4 (magnitude: 276.6).

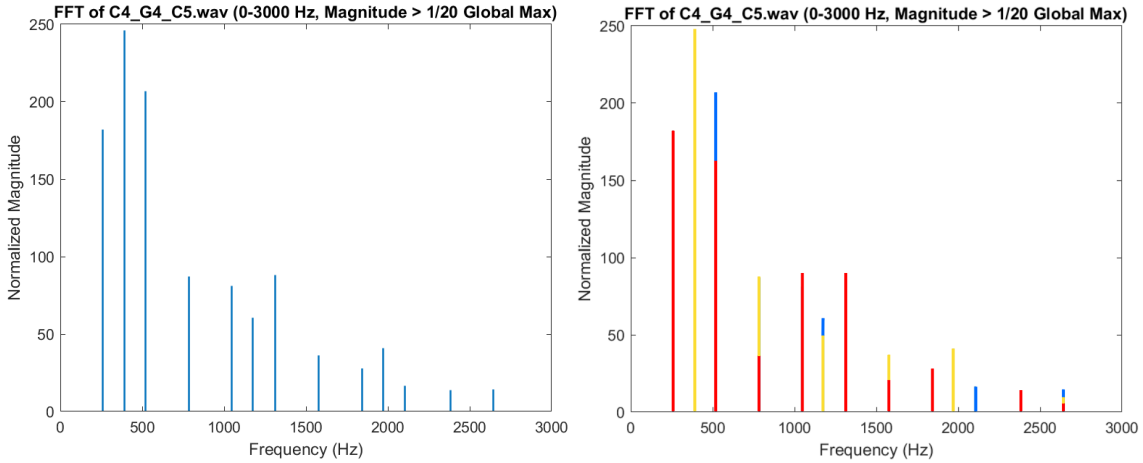


(a) C4 FFT Result (b) D4 FFT Result (c) E4 FFT Result (d) Chord FFT Result

Figure 14: The FFT results of C4, D4, E4, and the chord combination of C4, D4, E4, respectively.

8.2. Example 2 – C4, G4, C5

Another example is the chord C4, G4, C5. This chord is special compared to the first example: the fundamental frequency of C4, f_{C4}, is half of the fundamental frequency of C5 and $3f_{C4} = 2f_{G4}$, which discusses to the complex situation shown in Figure 5 where there are overtones overlapping.



(a) Original FFT Result for the C4, G4, C5 Chord (b) Magnitudes contributed by different music notes captioned (red: C4, yellow: G4, blue: C5)

By applying the same algorithm repeatedly until all entries in the matrix `freq_mag_chord` are deleted due to low magnitudes, we can derive all the existing notes played in the chord:

```
pitch: C4, frequency: 261, magnitude: 181.8912
pitch: G4, frequency: 392.2, magnitude: 246.2257
pitch: C5, frequency: 522, magnitude: 42.9171
pitch: G6, frequency: 1576.8, magnitude: 12.0278
```

The G6 note does not exist but presented with a low magnitude of 12.02.

One assumption I made is that all the wave functions simply superpose in phase (constructive interference), resulting in simple superposition of the frequency spectrum. But the magnitudes contributed by a

note in the chord may be different from that in a single vote due to phase difference in superpose process [14].

This might also be the reason of the slightly negative entry in Equation (8.1).

9. Conclusion

Our study demonstrates that the improvements made to the Fourier Transform algorithm significantly enhance its capability in accurately extracting music notes from a given clip while maintaining a high level of precision. By addressing the challenge posed by overtones and optimizing the computational process, we have achieved a balance between accuracy and efficiency in frequency analysis.

The key improvement of subtracting overtone intensities from the detected frequencies proved effective in minimizing the impact of overtones, leading to more accurate identification of the fundamental frequencies that correspond to the actual music notes. This approach, coupled with the transition from the Discrete Fourier Transform (DFT) to the Fast Fourier Transform (FFT), has not only enhanced the precision of the extracted notes but also significantly reduced the computational time required for the analysis.

To eliminate the impact of the overtones existing in the music instruments, we improve the result of frequency analysis by subtracting the intensities of the frequencies which are gained from the overtone.

To speed up the calculation process, the common Discrete Fourier Transform (DFT) with the time complexity in $O(n^2)$ is replaced by the Fast Fourier Transform (FFT) with the time complexity of $O(n \log_2 n)$ [5].

In §6 Data Collection, we improve the simplicity of the frequency and magnitude matrix by eliminating indices with small magnitudes, defining "peak" for programming.

Looking ahead, future research could focus on refining the overtone elimination process to reduce its dependency on extensive databases. Additionally, exploring alternative algorithms or hybrid approaches that balance the need for accuracy with computational efficiency could further enhance the practicality of this method for music transcription and analysis.

As examined in §8.2, the interference nature still need examine for the chord so that a result with a higher accuracy on fundamental frequency's recognition can be yielded when the notes have overtones overlapped.

Further visualization can be improved by adding a slider to the audio and the algorithm can be operated on each piece to perform a continuous musical note recognition inspired by Jeff Heaton's video Extract Musical Notes from Audio in Python with FFT [10].

In conclusion, while our improvements to the Fourier Transform algorithm have shown promise in extracting music notes with greater accuracy, ongoing development and refinement will be necessary to overcome the identified limitations and to fully realize the potential of this approach in various musical and signal processing applications.

References

[1] V. Serov *et al.*, *Fourier series, Fourier transform and their applications to mathematical physics*, vol. 197. Springer, 2017.

[2] W. K. Jenkins, "Fourier series, Fourier transforms and the DFT," in *Mathematics for Circuits and Filters*, pp. 83–111, CRC Press, 2022.

[3] H. Landau, "Sampling, data transmission, and the nyquist rate," *Proceedings of the IEEE*, vol. 55, no. 10, pp. 1701–1706, 1967.

[4] "Fourier Series and Fourier Transforms, EECS2 (6.082), MIT, Fall 2006, Lectures 2 and 3." https://web.mit.edu/6.02/www/f2006/handouts/Fourier.pdf, 2006.

[5] P. Duhamel and M. Vetterli, "Fast Fourier transforms: a tutorial review and a state of the art," *Signal processing*, vol. 19, no. 4, pp. 259–299, 1990.

[6] F. Reif and J. Larkin, "Standing Waves." https://physnet.org/modules/pdf_modules/m433.pdf.

[7] "Standing Waves in Wind Instruments — opentextbooks.org.hk." https://www.opentextbooks.org.hk/ditatopic/2286.

[8] "Maths of Chords — physics.stackexchange.com." https://physics.stackexchange.com/questions/372760/maths-of-chords.

[9] S. Leccia, A. Colantonio, E. Puddu, S. Galano, and I. Testa, "Teaching about mechanical waves and sound with a tuning fork and the sun," *Physics Education*, vol. 50, no. 6, p. 677, 2015.

[10] J. Heaton, "Extract Musical Notes from Audio in Python with FFT — youtube.com." https://www.youtube.com/watch?v=rj9NOiFLxWA.

[11] E. J. D. O. Maia Gibbs, Gabrielle Curry, "Piano Overtones, Comparative Study." https://courses.physics.illinois.edu/phys398dlp/sp2022/documents/group1_final.pdf, 2022.

[12] M. Müller, *Fundamentals of music processing: Using Python and Jupyter notebooks*, vol. 2. Springer, 2021.

[13] "Frequency and Pitch, Chapter 1, Section 3 — audiolabs-erlangen.de." https://www.audiolabs-erlangen.de/resources/MIR/FMP/C1/C1S3_FrequencyPitch.html.

[14] agate_, "When multiple players in an orchestra play the same note, on the same instrument, why is it consistently louder? — r/askphysics, reddit." https://www.reddit.com/r/AskPhysics/comments/1bdxf6g/when_multiple_players_in_an_orchestra_play_the/.

A. Appendix

A.1. Single Note Matrix Data Collection

MATLAB code for single note frequencies and corresponding magnitudes collection, with the magnitudes normalized scaled by fixing their lowest frequency's magnitude as the unit 1.

```
% File name for the single .wav file (C4.wav)
fileName = 'C4.wav';

% Initialize storage for audio data, sampling rate, frequency, and magnitude
audioData = []; % Store the audio data
Fs = 0;         % Store the sampling rate
freq_magnitude = []; % Store frequency and magnitude matrix for the file

% Define the maximum frequency range for the plot (0–3000 Hz)
maxFreq = 3000;

% Read the .wav file
[audioData, Fs] = audioread(fileName);

% Plot settings
figure;

% Apply FFT and reduce Nyquist effect (aliasing)
n = length(audioData);          % Number of samples
y_fft = fft(audioData);         % Compute the FFT
f = (0:n-1)*(Fs/n);             % Frequency vector

% Nyquist effect reduction: limit the FFT to the positive frequencies
y_fft = y_fft(1:floor(n/2));    % Take only the first half (positive frequencies)
f = f(1:floor(n/2));            % Corresponding frequency range

% Filter to only include frequencies within 0–3000 Hz
validFreqIdx = f <= maxFreq;    % Find frequencies <= 3000 Hz
f = f(validFreqIdx);            % Filter frequencies
y_fft = y_fft(validFreqIdx);    % Filter FFT results

% Compute magnitude and normalize by sampling rate
mag = abs(y_fft);               % Magnitudes of the FFT
globalMax = max(mag);           % Find the global maximum magnitude

% Split frequencies into 20 Hz intervals
freqBins = 0:20:maxFreq;        % Create frequency bins of 20 Hz intervals
new_f = [];                     % Initialize new frequency array
new_mag = [];                   % Initialize new magnitude array

% Storage for intermediate results
temp_f = [];  % Store frequencies
```

```
temp_mag = []; % Store magnitudes

% Loop over each 20 Hz interval
for j = 1:length(freqBins)-1
    % Find the indices of frequencies within the current 20 Hz bin
    binIdx = (f >= freqBins(j)) & (f < freqBins(j+1));
    bin_frequencies = f(binIdx);  % Frequencies in the current bin
    bin_magnitudes = mag(binIdx); % Magnitudes in the current bin

    % If there are any frequencies in this bin, find the maximum magnitude
    if ~isempty(bin_magnitudes)
        [maxMag, maxIdx] = max(bin_magnitudes); % Find max magnitude in the bin

        % Check if the max magnitude is greater than 1/20th of the global maximum
        if maxMag >= globalMax / 20
            % Add the maximum frequency and its magnitude to the new arrays
            temp_f = [temp_f, bin_frequencies(maxIdx)];  % Maximum frequency in this bin
            temp_mag = [temp_mag, maxMag];   % Normalized maximum magnitude in this bin
        end
    end
end

% Check minimum frequency and cancel spikes based on neighboring ranges
minFreq = min(temp_f); % Find the minimum frequency in the current results
if minFreq > 40
    % Create a copy of the temporary arrays for modification
    final_f = temp_f;
    final_mag = temp_mag;

    % Identify and remove lower magnitude spikes if frequency difference < 40 Hz
    for j = 1:length(final_f)-1
        if abs(final_f(j+1) - final_f(j)) < 40
            % Compare magnitudes and remove the lower one
            if final_mag(j) < final_mag(j+1)
                final_f(j) = NaN;  % Mark the lower frequency to be removed
                final_mag(j) = NaN; % Mark the lower magnitude to be removed
            else
                final_f(j+1) = NaN; % Mark the lower frequency to be removed
                final_mag(j+1) = NaN; % Mark the lower magnitude to be removed
            end
        end
    end

    % Remove NaN values (lower spikes)
    final_f = final_f(~isnan(final_f));
    final_mag = final_mag(~isnan(final_mag));

    % Update the results
```

```
    temp_f = final_f;
    temp_mag = final_mag;
end

% Create frequency and magnitude matrix with frequencies in first row, magnitudes in second row
freq_magnitude = [temp_f; temp_mag];  % Create a 2-row matrix: 1st row frequencies, 2nd row magnitudes

% Plot the FFT magnitude spectrum as spikes (stem plot without circles)
stem(temp_f, temp_mag, 'Marker', 'none', 'LineWidth', 1.2); % Remove markers on spikes
title('FFT of Piano C4 Note (Magnitude > 1/20 Global Max)');
xlabel('Frequency (Hz)');
ylabel('Magnitude');
xlim([0 3000]);  % Set frequency range from 0 to 3000 Hz

% Output frequency and magnitude matrix
freq_mag_C4 = freq_magnitude;  % Frequencies and magnitudes for C4.wav

% Display the frequency and magnitude matrix in the command window (optional)
disp('Frequency and Magnitude matrix for C4.wav:');
disp(freq_mag_C4);
```

A.2. Chord Note Processing and Recognition

MATLAB code for determining the notes in the chord, outputting processed data.

```
% File name of the .wav file
fileName = 'C4_G4_C5.wav';



% Read the .wav file
[audioData, Fs] = audioread(fileName);

% Plot settings
figure;

% Apply FFT and reduce Nyquist effect (aliasing)
n = length(audioData);         % Number of samples
y_fft = fft(audioData);        % Compute the FFT
f = (0:n-1)*(Fs/n);            % Frequency vector

% Nyquist effect reduction: limit the FFT to the positive frequencies
y_fft = y_fft(1:floor(n/2));   % Take only the first half (positive frequencies)
f = f(1:floor(n/2));           % Corresponding frequency range

% Define the maximum frequency range for the plot (0-3000 Hz)
maxFreq = 3000;

% Filter to only include frequencies within 0-3000 Hz
```

```
validFreqIdx = f <= maxFreq;   % Find frequencies <= 3000 Hz
f = f(validFreqIdx);           % Filter frequencies
y_fft = y_fft(validFreqIdx);   % Filter FFT results

% Compute magnitude and normalize by sampling rate
mag = abs(y_fft);         % Magnitudes of the FFT divided by sampling rate
globalMax = max(mag);          % Find the global maximum magnitude

% Split frequencies into 20 Hz intervals
freqBins = 0:20:maxFreq;       % Create frequency bins of 20 Hz intervals
temp_f = [];                   % Initialize new frequency array
temp_mag = [];                 % Initialize new magnitude array

% Loop over each 20 Hz interval
for j = 1:length(freqBins)-1
    % Find the indices of frequencies within the current 20 Hz bin
    binIdx = (f >= freqBins(j)) & (f < freqBins(j+1));
    bin_frequencies = f(binIdx);  % Frequencies in the current bin
    bin_magnitudes = mag(binIdx); % Magnitudes in the current bin

    % If there are any frequencies in this bin, find the maximum magnitude
    if ~isempty(bin_magnitudes)
        [maxMag, maxIdx] = max(bin_magnitudes); % Find max magnitude in the bin

        % Check if the max magnitude is greater than 1/10th of the global maximum
        if maxMag >= globalMax / 20
            % Add the maximum frequency and its magnitude to the new arrays
            temp_f = [temp_f, bin_frequencies(maxIdx)];  % Maximum frequency in this bin
            temp_mag = [temp_mag, maxMag / 1];   % Normalized maximum magnitude in this bin
        end
    end
end

% Check minimum frequency and cancel spikes based on neighboring ranges
minFreq = min(temp_f); % Find the minimum frequency in the current results

if minFreq > 40
    % Create a copy of the temporary arrays for modification
    final_f = temp_f;
    final_mag = temp_mag;

    % Identify and remove lower magnitude spikes if frequency difference < 40 Hz
    for j = 1:length(final_f)-1
        if abs(final_f(j+1) - final_f(j)) < 40
            % Compare magnitudes and remove the lower one
            if final_mag(j) < final_mag(j+1)
                final_f(j) = NaN;  % Mark the lower frequency to be removed
                final_mag(j) = NaN; % Mark the lower magnitude to be removed
```

```
            else
                final_f(j+1) = NaN; % Mark the lower frequency to be removed
                final_mag(j+1) = NaN; % Mark the lower magnitude to be removed
            end
        end
    end

    % Remove NaN values (lower spikes)
    final_f = final_f(~isnan(final_f));
    final_mag = final_mag(~isnan(final_mag));

    % Update the results
    temp_f = final_f;
    temp_mag = final_mag;
end

% Create frequency and magnitude matrix with frequencies in the first row, magnitudes in the second
    row
freq_magnitude = [temp_f; temp_mag];  % Create a 2-row matrix: 1st row frequencies, 2nd row magnitudes
freq_magnitude_initial = freq_magnitude;
% Plot the FFT magnitude spectrum as spikes (stem plot without circles)
stem(temp_f, temp_mag, 'Marker', 'none', 'LineWidth', 1.2); % Remove markers on spikes
title(['FFT of ' fileName ' (0-3000 Hz, Magnitude > 1/20 Global Max)']);
xlabel('Frequency (Hz)');
ylabel('Normalized Magnitude');
xlim([0 3000]);  % Set frequency range from 0 to 3000 Hz
ylim([0 250]);
% Output frequency and magnitude matrix
disp('Frequency and Magnitude matrix for test.wav:');
disp(freq_magnitude);

while ~isempty(temp_f)

% Find the first note by the lowest frequency
lowestFreq = min(temp_f);
[~, minIndex] = min(temp_f);
A4 = 440;  % Frequency of A4
semitones = round(12 * log2(lowestFreq / A4));  % Semitone difference from A4
noteNames = {'C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#', 'A', 'A#', 'B'};  % Musical notes

% Find the pitch corresponding to the semitone difference
octave = 4 + floor((semitones + 9) / 12);  % Calculate the octave number
noteIdx = mod(semitones + 9, 12) + 1;      % Find the note within the octave
pitch = [noteNames{noteIdx}, num2str(octave)];  % Form the pitch string (e.g., A4)



% Retrieve the magnitude corresponding to the lowest frequency
```

```
lowestFreqMagnitude = temp_mag(minIndex);

% Display the result
disp(['pitch: ', pitch, ', magnitude: ', num2str(lowestFreqMagnitude)]);

matrixName = ['freq_mag_' pitch]; % Construct the matrix name
freq_mag_funda = eval(matrixName); % Evaluate the matrix name

% Compare the first row (frequencies) of freq_mag_funda and freq_magnitude matrices

% Get frequencies from both matrices
frequencies_funda = freq_mag_funda(1, :);  % First row of freq_mag_C3
frequencies_mag = freq_magnitude(1, :);  % First row of freq_magnitude

% Ensure that matching frequencies are found and magnitudes are updated properly

% Initialize arrays to store results
matching_freq = [];
matching_mag_funda = [];
matching_mag_chord = [];
mag_after_subtract = [];  % Array to store the updated magnitudes after subtraction

% Loop through each frequency in freq_mag_funda
for i = 1:length(frequencies_funda)
    % Find frequencies in freq_magnitude that are within 10 Hz of the current frequency in
        freq_mag_funda
    freq_diff = abs(frequencies_mag - frequencies_funda(i));

    % If any frequency matches within 10 Hz
    if any(freq_diff <= 3)
        % Get the index of the closest matching frequency
        [~, matchIdx] = min(freq_diff);

        % Store the matching frequency and its corresponding magnitudes
        matching_freq = [matching_freq, frequencies_funda(i)];
        matching_mag_funda = [matching_mag_funda, freq_mag_funda(2, i)];  % Magnitude from
            freq_mag_funda
        matching_mag_chord = [matching_mag_chord, freq_magnitude(2, matchIdx)];  % Magnitude from
            freq_magnitude

        % Subtract the magnitudes and store the result
        correspondingMagnitude = matching_mag_chord(end); % Ensure this value is set correctly
        subtracted_mag = correspondingMagnitude - lowestFreqMagnitude*matching_mag_funda(end);
        mag_after_subtract = [mag_after_subtract, subtracted_mag];

        % Update temp_mag with the new subtracted magnitude
        temp_mag(matchIdx) = subtracted_mag; % Ensure matchIdx is valid
    end
```

```
end


% Remove frequencies with magnitudes less than 0.01
threshold = 15; % Magnitude threshold
validIndices = temp_mag >= threshold; % Find indices where magnitude is above the threshold

% Update temp_f and temp_mag by keeping only valid indices
temp_f = temp_f(validIndices);
temp_mag = temp_mag(validIndices);

% Plot the updated FFT magnitude spectrum
figure;
stem(temp_f, temp_mag, 'Marker', 'none', 'LineWidth', 1.2); % Remove markers on spikes
title(['Updated FFT of Chord']);
xlabel('Frequency (Hz)');
ylabel('Magnitude');
xlim([0 3000]);  % Set frequency range from 0 to 3000 Hz
ylim([0 250]);

% Output the updated frequency and magnitude matrix
freq_magnitude = [temp_f; temp_mag]; % Create a 2-row matrix: 1st row frequencies, 2nd row magnitudes
end
```