Fourier Analysis and Musical Signal Processing

Matthew Lad
Department of Math and Computer Science
Samford University
Birmingham, AL 35229, USA

Abstract

There is an undeniable connection between music and mathematics, and with modern computer technology, these connections can be better explored and understood. As such, this paper presents research into the mathematics behind audio and musical signal analysis from both a mathematical and computer science viewpoint. The mathematics is centered around Fourier series and Fourier transforms and the computer science aspects focus on using python for algorithms and visualizations. There is less emphasis on detailed computer algorithms and rigorous mathematical proofs as a general understanding of the material is emphasized instead.

1 Introduction

Throughout history, music has always been shared and greatly enjoyed. Alongside this there has always been a drive to preserve and better understand the music created. The ancient Chinese developed a musical system in which notes were given names, much like our ABCDEFG system, with each holding deep connections to religious and cosmological practices. Other ancient civilizations in Egypt, Palestine, Phoenicia, Babylonia, Syria, Greece, and southern Europe developed similar organized systems connected to mythological or cosmological practices. A large change came in the understanding of music with the ancient Greeks. A group called the Pythagoreans began to associate musical systems with geometry while still clinging to mythological and philosophical connections. Specifically, they discovered that the most beautiful sounds were created by string length ratios of whole numbers. The two most important to them were 2:1 and 3:2, which represent in today's notation the octave and perfect fifth intervals [1][2].

As understanding of physics and acoustics grew, the inner workings of sound moved beyond simple ratios. With the invention of the telegraph and telephone, a much deeper dive into understanding waves, signals, sounds, and music began. These two technologies created an entire field of mathematics called signal processing, of which Fourier analysis is a part [1][3]. In more recent times, investigations into the psychological

nature of sound and music uses mathematics to help explain or convey ideas. The phenomena synethsia; where the different senses are "mixed" in the brain, causing a sound to be perceived as an image or color, is one area where mathematics can be used to help demonstrate what synethsia might be like [4]. This paper serves as an exploration of various bits of this large history with focus on Fourier analysis of musical signals. Additionally featured in the paper are experiments done by the author into the various topics covered.

The paper begins in section 2 by introducing musical preliminaries of physics, theory, and mathematics. The subsection 2.1 covers some background information on sound and introduces terms used throughout the rest of the paper. In 2.2, music theory is introduced, and 2.3 introduces the methods for graphing signals. The information on music theory comes from a collection of sources listed in the bibliography, but most notably *Introduction to the Theory of Music* by Howard Boatwright. This book provided a great introduction to acoustics and gave inspiration for many diagrams in this paper [5].

Section 3 begins the mathematical portion of the paper and quickly moves into a rigorous covering of the mathematics. The sources that most influenced section 3 were *Introduction to Fourier Analysis* by Norman Morrison, *Elementary Linear Algebra* by Paul C. Shields, and the YouTube channel *3Blue1Brown* by Grant Sanderson [6][7][8]. While these three sources provided a majority of the mathematical foundations in section 3, the overall order and layout of the section came largely from [9]. This paper by Nathan Lenssen and Deanna Needell provided a backbone for the paper, having a similar layout and topic of study.

Moving away from the mathematics, section 4 introduces the visualization side of the research, concerning how music can be visualized and how phenomena like synethsia can be simulated. In section 5 the experiments and implementations for the research are introduced. Lastly, a discussion portion serves as section 6, with the last sections being a conclusion, acknowledgements, and appendix for any programming code discussed within the paper.

2 Musical Preliminaries

2.1 Sound

In physics, **sound** is a series of oscillating low and high air pressures. The oscillations are created when a material vibrates. A **cycle**, or **vibration**, is one high-pressure and one low-pressure area together. The **frequency** or **pitch** is the number of cycles per second, measured in **Hz** [10]. The higher the frequency of the sound, the higher the pitch of the sound heard. The **volume**, "**loudness**", or **amplitude** of a sound is defined by the difference in air pressure between a low area and high area of a cycle. A quiet sound can be described as a transition between two densities of similar value. Likewise, a loud sound is a transition between a very high density and a very low density [5] [9]. These terms oscillating, cycles, frequencies, etc. are all terms used

when dealing with certain functions in mathematics. A diagram of this relation is shown in figure 1. The elementary **sine** function graphs the air density over time. As the density increases, the sine wave increases.

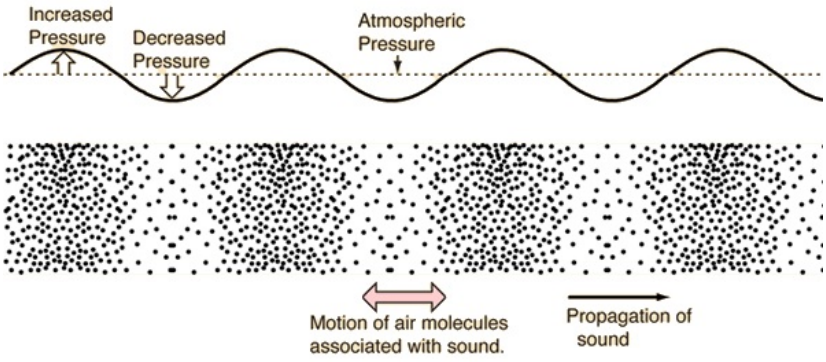


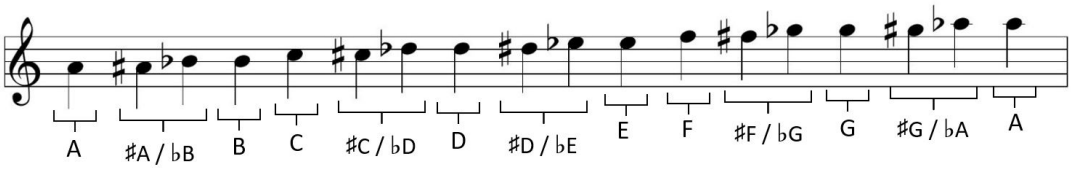
Figure 1: The high and low air pressures of sound can be represented by a wave function [11].

2.2 Music Theory

Different notes are determined by specific frequencies. The lower sounding a note is, the lower the frequency of the sound wave. Similarly with high-pitched sounds, their frequencies are much higher. The most common system in western music for determining the frequency values of musical notes is a 12-tone system of ratios called **equal temperament**. Given a reference note with frequency x, there are 11 notes between x and $2x$. Each of these notes are separated by a ratio equal to the 12th root of 2, ($\sqrt[12]{2} \approx 1.05946$). See figure 2 for a 12-tone scale using equal temperament [12] [5].

Today the note "A", denoted "Unison" or A4, is commonly set to 440Hz and is used as a reference point for calculating the other frequencies. This standard has changed throughout history and has its origins in medieval Europe. See [15] for further reading.

When a note is played by a computer, the frequency is perfectly calculated and a very artificial sound is heard. With instruments and the human voice however, perfect frequencies are not generated and there are qualities which cause notes to sound unique and beautiful. For example, when a clarinet and a trumpet both play A4, even though they play the same frequency, they sound very different. This is due to differences in air flow and materials, wood vs. metal, lip buzzing vs. reed vibrations, etc. Analyzing the frequencies of computer generated notes and instrument generated notes reveals something amazing behind the physics of musical sound.



12-Tone Equal Temperament			
Note	Interval	Ratio	Frequency(Hz)
A	Unison	1 : 1	440.00
♯A / ♭B	Minor Second	1 : 1.05946	466.16
B	Major Second	1 : 1.12246	493.88
C	Minor Third	1 : 1.18921	523.25
♯C / ♭D	Major Third	1 : 1.25992	554.37
D	Fourth	1 : 1.33483	587.33
♯D / ♭E	Diminished Fifth	1 : 1.41421	622.25
E	Fifth	1 : 1.49831	659.25
F	Minor Sixth	1 : 1.58740	698.46
♯F / ♭G	Major Sixth	1 : 1.68179	739.99
G	Minor Seventh	1 : 1.78180	783.99
♯G / ♭A	Major Seventh	1 : 1.88775	830.61
A	Octave	1 : 2	880.00

Figure 2: The following shows the 12 divisions within the equal temperament system. Each ratio is a $\sqrt[12]{2} \approx 1.05946$ apart [5] [13] [14]. Although considered the same in the equal temperament system, the paired accented notes (♯A / ♭B, etc.) are actually not the same. See [12] for more information on this beyond the scope of this paper.

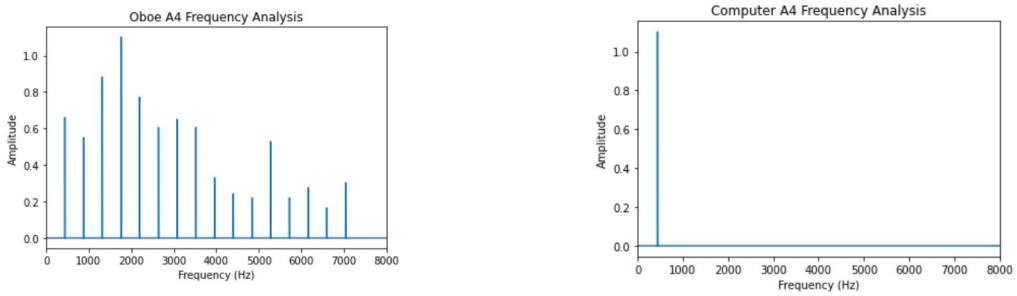


Figure 3: The graph on the left shows the frequency analysis of an oboe playing an A4. The right graph shows a computer playing the same note.

Figure 3 shows the frequency analysis of a computer generated A4 and an A4 played by a oboe. As you can see on the computer graph, the large spike is centered around 440 Hz, which matches the table in figure 2 for the note A4. Notice how the oboe graph contains multiple other spikes decreasing in size from the largest spike at

440 Hz. This phenomena is known as the **overtone** or **harmonic** series and consists of regular intervals from the **fundamental**, or largest, frequency. These successive spikes are created by properties of the instrument used such as material, size, air column, etc., meaning every note played is slightly unique. Because a computer will always generate the exact same frequency, its notes are never unique. This is one reason for the unique beauty in singing and instrumental music.

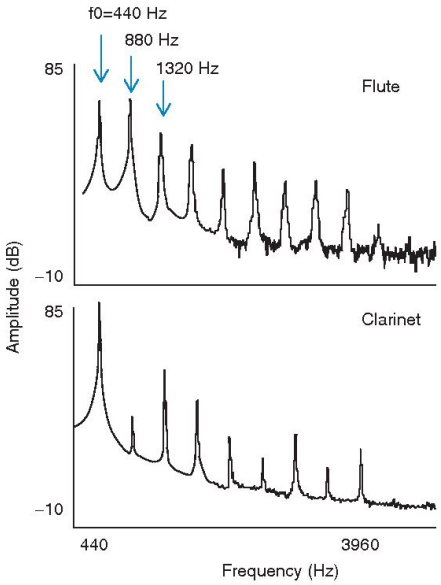


Figure 4: Graphs of a clarinet and a flute both playing the note A4.

Figure 4 shows the frequency analysis of a clarinet and flute both playing the note A4. Notice how although both are playing the same note, the differences in spikes leads to the different sound between the two instruments. The frequencies within the harmonic series are determined by whole number ratios. No matter what the fundamental frequency is, the harmonic series will always follow the ratios; 1:1, 1:2, 1:3, 1:4, ..., 1:n for n harmonics in the series.

2.3 Time and Frequency Domains

There are two common ways of graphing sound. It can be graphed as amplitude over time, which shows the volume of the sound, or as amplitude over frequency, which shows the frequency of the sound. Mathematically, these are referred to as the **time domain representation** and **frequency domain representation**. The time domain shows how the sound physically behaves over time (the x-axis is time and the y-axis is amplitude). The frequency domain shows the frequencies of the sound (the x-axis is frequency and the y-axis is the amplitude, or strength of the frequencies). Time domain data can be generated through a microphone as it is a literal representation of the sound waves. The frequency domain data is much more complicated and requires a

bit of math to extract from the time domain data. The rest of this paper will focus on understanding this process of moving from the time domain to the frequency domain.

3 Mathematical Preliminaries

The following sections will serve both as an introduction to the mathematics of music as well as an exploration of various concepts through computer programs. Additionally, the process of moving between the time and frequency domains will be described and explored.

3.1 Periodic Functions

A **periodic function** is any function for which $f(t) = f(t+T)$ for all t. The smallest constant T which satisfies it is called the **period** of the function (it 'repeats' every T) [16]. The sine wave that represented the sound in figure 1 is an example of a periodic function where $f(t) = A\sin(\omega t + \varphi)$. Here A is the amplitude, or the maximum deviation of the function from zero. $\omega = 2\pi f$ where f is the frequency, also denoted $\frac{1}{T}$, multiplied by 2π to convert it to an angular frequency (a measure of radians per second instead of cycles per second). Lastly, φ specifies the phase of the wave (a measure, in radians, of where the oscillation is in within its cycle at $t = 0$) [17] [18].

Showing an example of a sine function that generates a musical note is figure 5.

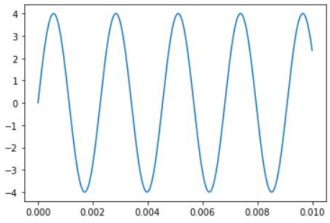


Figure 5: The note A4 created using the sine function: $f(t) = A\sin(\omega t + \varphi)$, where $A = 4$, $\omega = 2\pi 440$, and $\varphi = 0$.

By changing the amplitude $A = 20$ and making the x-axis go from 0 to 10000, the sine function can be calculated and stored into a wav file. This file can then be played on a computer and the note A4 will actually be heard. By adding successive sine functions to the original, a more natural sounding A4 can be generated. These added sine functions represent the additional harmonics.

Notice in figure 6 that the amplitudes (4, 3.95, 3.9, and 3.85) are decreasing and the frequencies (440, 880, 1320, and 1760) are increasing by the before mentioned whole number ratios 1:1, 1:2, 1:3, and 1:4.

Moving now away from sound and music, we will dive into the Fourier analysis, which will allow us to move from the time domain into the frequency domain and vice versa. This exploration begins with Orthogonality.

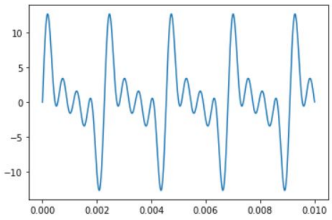


Figure 6: The note A4 created using the sine functions: $f(t) = 4\sin(2\pi 440t + \varphi) + 3.95\sin(2\pi 880t + \varphi) + 3.9\sin(2\pi 1320t + \varphi) + 3.85\sin(2\pi 1760 + \varphi)$.

3.2 Orthogonality of Vectors and Functions

A **vector space** is any set that satisfies the defined axioms of addition and scalar multiplication. Details not listed here. An **inner product** is a function on a vector space which satisfies the following axioms: (u and v and w are vectors in the vector space and c is a scalar) [8].

1. $\langle u, v\rangle = \langle v, u\rangle$
2. $\langle u, v + w\rangle = \langle u, v\rangle + \langle u, w\rangle$
3. $c\langle u, v\rangle = \langle cu, v\rangle$
4. $\langle v, v\rangle \geq 0$, and $\langle v, v\rangle = 0$ if and only if $v = 0$.

An **inner product space** is a vector space with a defined inner product which satisfies the above axioms for every vector within it. Properties & Definitions related to Inner Product Spaces: [8].

1. $\langle 0, v\rangle = \langle v, 0\rangle = 0$
2. $\langle u + v, w\rangle = \langle u, w\rangle + \langle v, w\rangle$
3. $c\langle u, v\rangle = \langle u, cv\rangle$
4. the **norm** (or **length**) of u is $||u|| = \sqrt{\langle u, u\rangle}$
5. the **distance** between u and v is $d(u, v) = ||u - v||$
6. the **angle** between two nonzero vectors u and v is:
   $\cos(\theta) = \frac{\langle u,v\rangle}{\|u\|\|v\|}$, $0 \leq \theta \leq \pi$
7. vectors u and v are **orthogonal** if $\langle u, v\rangle = 0$.

When a set of vectors $S = \{u_1, u_2, u_3, ...\}$ satisfies $(u_n, u_m) = 0$ when $n \neq m$, we say that the elements of S are **mutually orthogonal** and that they form an **orthogonal basis** for the space spanned by S. Any vector in that space can be represented as a linear combination of those basis vectors [8].

The set of functions $\{f_1(t),\ f_2(t),\ f_3(t),\ ...,\ f_k(t),\ ...\}$ is orthogonal on an interval $a < t < b$ if for any two functions $f_n(t)$ and $f_m(t)$ in the set, [8].

$$\langle f_n, f_m \rangle = \int_a^b f_n(t) f_m(t)\, dt = \begin{cases} 0 & \text{for } n \neq m \\ r_n & \text{for } n = m \end{cases} \tag{1}$$

This property of sets of functions will provide usefully when working with complex exponentials in series.

3.3 Complex Exponentials

Let z be a complex number where $z = x + jy$ and $j = \sqrt{-1}$. An additional complex number to note is the complex conjugate $x - jy$, denoted z^*.

The **complex exponential** is a formula of sorts which represents the distance traveled around a circle of radius 1. Denoted e^{jx}, this exponential gives us the location on a unit circle after walking x units in a clockwise direction. The exponential e^{-jx} will calculate the same location but after traveling in a counter clockwise direction. By adding $2\pi f$ to the exponential and changing the x for a t we get $e^{2\pi f j t}$. By adding the $2\pi f t$, where f is the frequency and t is time, we now discover that the formula makes f full rotations of the unit circle every second. The counterclockwise rotation of our earlier formula holds here as well. By setting $\omega_0 = 2\pi f$, we have $e^{j\omega_0 t}$.

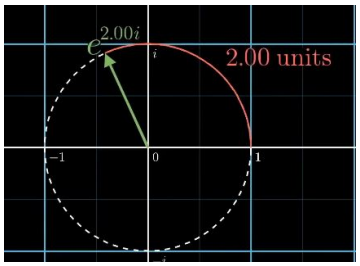


Figure 7: Euler's formula is an incredible connection between complex analysis and trigonometry. For more information, see [19].

The complex exponentials $e^{jn\omega_0 t}$ (by adding n the value is magnified) together form the set [7].

$$S = \{..., e^{-j3\omega_0 t}, e^{-j2\omega_0 t}, e^{-j\omega_0 t}, 1, e^{j\omega_0 t}, e^{j2\omega_0 t}, ..., e^{jn\omega_0 t}, ...\}$$

Theorem 1 (Orthogonality of the Complex Exponentials)**.** *The complex exponentials* $e^{jn\omega_0 t}$ *of the above set satisfy the orthogonality condition*

$$\frac{1}{T_0}\int_{-T_0/2}^{T_0/2} e^{jn\omega_0 t}e^{jm\omega_0 t^*}\,dt = \begin{cases} 0 & \text{if } n \neq m \\ 1 & \text{if } n = m \end{cases}$$

where $T_0 = 2\pi/\omega_0$ *and* $e^{jm\omega_0 t^*}$ *is the complex conjugate* $e^{-jm\omega_0 t}$*[7].*

Here is a proof showing this.

Proof.

$$\begin{aligned} &\frac{1}{T_0}\int_{-T_0/2}^{T_0/2} e^{jn\omega_0 t}e^{jm\omega_0 t^*}\,dt \\ &\qquad = \frac{1}{T_0}\int_{-T_0/2}^{T_0/2} e^{jn\omega_0 t}e^{-jm\omega_0 t}\,dt \\ &\qquad = \frac{1}{T_0}\int_{-T_0/2}^{T_0/2} e^{jn\omega_0 t - jm\omega_0 t}\,dt \\ &\qquad = \frac{1}{T_0}\int_{-T_0/2}^{T_0/2} e^{jw_0 t(n-m)}\,dt \end{aligned} \tag{2}$$

When $n \neq m$, then $n - m$ is a nonzero integer which we shall call p, and so (2) continues as

$$\begin{aligned} \frac{1}{T_0}\int_{-T_0/2}^{T_0/2} e^{jw_0 t(n-m)}\,dt &= \frac{1}{T_0}\int_{-T_0/2}^{T_0/2} e^{jp\omega_0 t}\,dt \\ &= \frac{1}{T_0}\Big[\frac{1}{j\omega_0 p}e^{j\omega_0 pt}\Big]_{-T_0/2}^{T_0/2} \\ &= \frac{1}{T_0}\Big(\frac{e^{j\omega_0 p(T_0/2)}}{j\omega_0 p} - \frac{e^{j\omega_0 p(-T_0/2)}}{j\omega_0 p}\Big) \\ &= \frac{1}{T_0}\Big(\frac{e^{(j\omega_0 pT_0)/2} - e^{(-j\omega_0 pT_0)/2}}{j\omega_0 p}\Big) \\ &= \frac{1}{T_0}\Big(\frac{e^{jp\pi} - e^{-jp\pi}}{j\omega_0 p}\Big) \\ &= \frac{e^{jp\pi} - e^{-jp\pi}}{T_0 j\omega_0 p} \\ &= \frac{e^{jp\pi} - e^{-jp\pi}}{2\pi jp} \\ &= \frac{(\cos p\pi + j\sin p\pi) - (\cos p\pi - j\sin p\pi)}{2\pi jp} \\ &= \frac{\cos p\pi + j\sin p\pi - \cos p\pi + j\sin p\pi}{2\pi jp} \end{aligned}$$

$$
\begin{aligned}
&= \frac{2j \sin p\pi}{2\pi jp} \\
&= \frac{\sin p\pi}{p\pi} \\
&= 0
\end{aligned}
$$

On the other hand, when $n = m$ then (2) continues as

$$
\begin{aligned}
\frac{1}{T_0}\int_{-T_0/2}^{T_0/2} e^{jw_0t(n-m)}\,dt &= \frac{1}{T_0}\int_{-T_0/2}^{T_0/2} e^{j\omega_0 t(0)}\,dt \\
&= \frac{1}{T_0}\int_{-T_0/2}^{T_0/2} e^{0}\,dt \\
&= \frac{1}{T_0}\int_{-T_0/2}^{T_0/2} dt \\
&= \frac{1}{T_0}[t]\Big|_{-T_0/2}^{T_0/2} \\
&= \frac{1}{T_0}\Big(\frac{T_0}{2} + \frac{T_0}{2}\Big) \\
&= \frac{T_0}{T_0} \\
&= 1
\end{aligned}
$$

And the proof is complete [7]. □

3.4 Fourier Series

Using this property of orthogonality, let $f_p(t)$ be a periodic function with period T_0, and assume it can be expressed as an infinite sum of complex exponentials, that is, that

$$
f_p(t) = \sum_{n=-\infty}^{\infty} F(n)e^{jn\omega_0 t} \tag{3}
$$

The complex exponentials on the right-hand side all repeat at least once whenever t is increased by T_0, so both sides are periodic with period T_0. To obtain the constants $F(n)$, we first multiply both sides by $e^{-jm\omega_0 t}$ and integrate, obtaining

$$
\begin{aligned}
f_p(t) &= \sum_{n=-\infty}^{\infty} F(n)e^{jn\omega_0 t} \\
\frac{1}{T_0}\int_{-T_0/2}^{T_0/2} f_p(t)e^{-jm\omega_0 t}\,dt &= \frac{1}{T_0}\int_{-T_0/2}^{T_0/2} \sum_{n=-\infty}^{\infty} F(n)e^{jn\omega_0 t}e^{-jm\omega_0 t}\,dt
\end{aligned} \tag{4}
$$

Interchanging the order of summation and integration on the right we get

$$\frac{1}{T_0}\int_{-T_0/2}^{T_0/2} f_p(t)e^{-jm\omega_0 t}\,dt \;=\; \sum_{n=-\infty}^{\infty} F(n)\Big[\frac{1}{T_0}\int_{-T_0/2}^{T_0/2} e^{jn\omega_0 t}e^{-jm\omega_0 t}\,dt\Big] \tag{5}$$

By theorem 2.1, when $n = m$, the quantity in square brackets becomes 1 and so we are left with

$$\frac{1}{T_0}\int_{-T_0/2}^{T_0/2} e^{jn\omega_0 t}e^{-jm\omega_0 t}\,dt = F(m) \tag{6}$$

Then replacing m with n we get

$$F(n) \;=\; \frac{1}{T_0}\int_{-T_0/2}^{T_0/2} f_p(t)e^{-jn\omega_0 t}\,dt \quad [7]. \tag{7}$$

This formula essentially describes the inner product of $f_p(t)$ and $e^{-jn\omega_0 t}$. A way to visualize this process is that the formula "projects" the function $f_p(t)$ onto a unit circle in the complex plane. In figure 8 we see this more literally. As mentioned previously, the complex exponential provides rotational movement while the function $g(t)$, substituted for our $f_p(t)$, provides the amplitude of the vector (the green arrow). By integrating from $-T_0/2$ to $T_0/2$ and dividing by T_0, $F(n)$ is essentially the average value along the projection, or a sort of center-of-mass for the unit circle. This is shown in figure 9. Lastly, by graphing only the real number portion of $F(n)$, shown in figure 10 as $\hat{g}(f)$, we have the frequency domain graph.

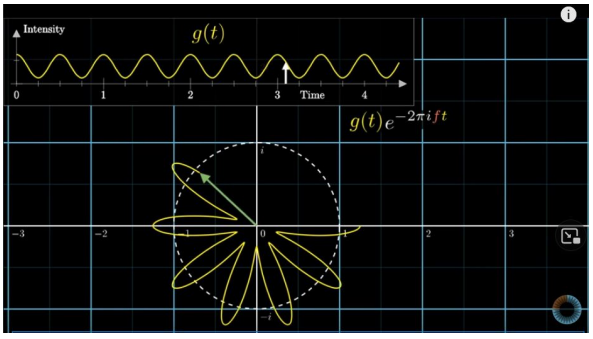


Figure 8: The projection of $g(t)$ into the complex plane. This visualization was done by Grant Sanderson on his YouTube channel 3Blue1Brown. The channel contains tons of incredible videos on mathematics topics, see [6] for more information.

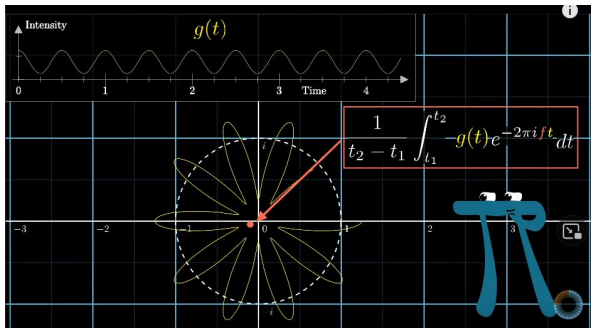


Figure 9: Another visualization showing how the integral calculates a "center-of-mass" for the function $g(t)$ [6].

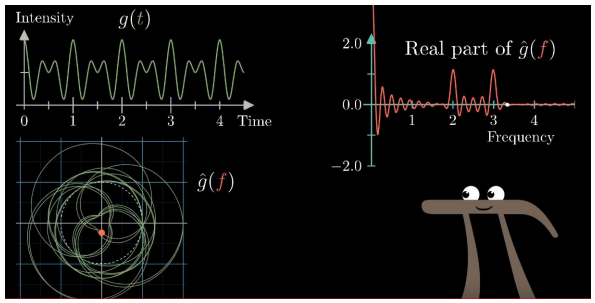


Figure 10: Visualization showing how the real number portion of the integral's calculated values can be graphed to create a frequency-domain graph [6].

This can all be summarized as

Theorem 2 (Complex Fourier Series for Periodic Functions)**.** *Let $f_p(t)$ be periodic with period T_0, defined analytically. Then it can also be represented by the infinite series of complex exponentials*

$$f_p(t) = \sum_{n=-\infty}^{\infty} F(n) e^{jn\omega_0 t} \tag{8}$$

where the coefficients $F(n)$ can be found from the analytical definition of $f_p(t)$ as follows:

$$F(n) = \frac{1}{T_0} \int_{-T_0/2}^{T_0/2} f_p(t) e^{-jn\omega_0 t}\, dt \quad (\forall n) \tag{9}$$

The previous sections will serve as a basis for what the rest of the paper will discuss. To summarize parts of theorem 2.2

- Equation (8) is called the **synthesis equation** because it generates a periodic function from coefficients and complex exponentials.

- Equation (9) is called the **analysis equation** because it extracts frequency coefficients from a periodic function.
- The constants $F(n)$ are called the **complex Fourier coefficients** [7].

3.5 Fourier Transform

Until now, we have looked exclusively at periodic functions of which we can create a Fourier series representation. The question arises if single pulse functions which do not repeat over a period can be represented by a Fourier series. The side of Fourier analysis dealing with single pulses uses the **Fourier transform**.

Given a single pulse, we can think of it as being embedded in a single period of a periodic function. We can then have the period T_0 tend towards infinity. Even though we have a periodic function, because the period tends toward infinity we essentially eliminate all other pulses but our original single pulse [7]. There is a great deal of complexity beyond this, but for the purposes of this paper, we will move straight into theorem 3.1.

Theorem 3 (Fourier Transform for a Single Pulse)**.** *(provided that the integrals exist:)*
Synthesis:

$$f(t) \;=\; \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{j\omega t}\,d\omega \tag{10}$$

Analysis:

$$F(\omega) \;=\; \int_{-\infty}^{\infty} f(t)e^{-j\omega t}\,d\omega \tag{11}$$

Equation (11) shows that a time domain function $f(t)$ can be **analyzed** to give an associated frequency domain function $F(\omega)$ that contains all the information in $f(t)$. $F(\omega)$ is known as the **Fourier transform** of $f(t)$. Likewise, in equation (10) we can see that we can **synthesize** a pulse given it's Fourier transform function. $f(t)$ then is called the **inverse Fourier transform** of $F(\omega)$ [7].

An amazing feature of the Fourier transform/ inverse Fourier transform is they can be used to operate on all pulse cases. One-time pulses of finite duration, one-time pulses of infinite duration, and periodic functions that inherently have infinite duration can all be analyzed/ synthesized.

3.6 Spectrograms

Section 3.5 ended with the introduction to the Fourier transform and as previous sections described, the transform can be used on an audio signal in order to "extract" the frequencies in the sound. With this ability to graph either the time or frequency domains, an issue arrises. The time graph focuses only on amplitude over time and the frequency graph focuses only on the the frequencies present at a single period of

time. What if there was a need to graph what frequencies are present over time? The spectrogram answers this question by showing the frequencies present across a range of times. The graph uses time for it's x-axis and frequency for it's y-axis. The spectrum of colors within the graph indicate the amplitude of the different frequencies. Figure 11 shows an example of a spectrogram of an audio signal.

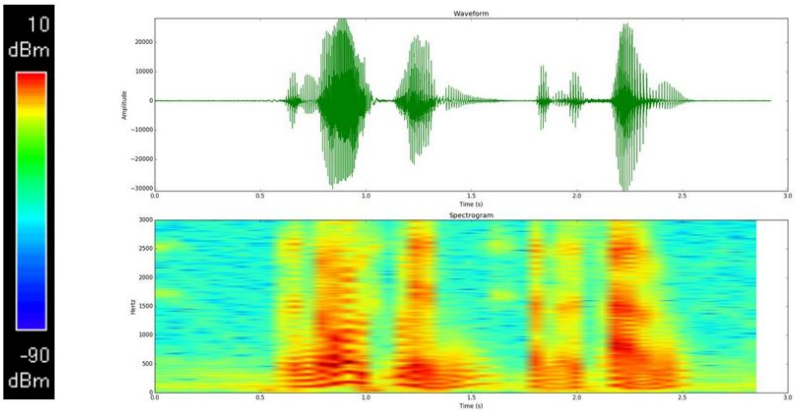


Figure 11: A spectrogram of an audio signal. The top graph in green is a time domain graph with time vs. amplitude. The bottom graph is a spectrogram with time vs. frequency. The color shows frequency amplitude.

Although the spectrogram serves to combine the time and frequency domain graphs, some accuracy is lost in using color. The colors do not accurately display numerical values and differences in how people view colors or in how screens or printers show the colors can cause errors. Regardless, the spectrogram provides a useful and creative method for visualizing and understanding an audio signal.

4 Experiments and Implementations

With the mathematics involved in Fourier analysis presented, the usefulness of computers can now be shown in full. Microphones and modern computer technologies allow signals to be analyzed using the fourier transform. Real world examples are feasible and studying the mathematics of fourier analysis is no longer theoretical. Starting in python, there are various libraries which facilitate real-time Fourier analysis. The library Scipy contains multiple Fast Fourier Transform functions which allow a computer's microphone input to be analyzed in real-time. Additionally, as has already been mentioned, python has support for creating and graphing sine functions and then using those functions to create audio files.

4.1 Time Domain of a Real Time Signal

In order to visualize music, an audio signal must be broken down so that various aspects of the sound can be analyzed. The most sensible first steps would be to look at the musical signal in the time and frequency domains. This would then display volume and pitch of the signal. The time domain can be easily obtained without much math with the help of a python library. Using the library PyAudio, the microphone can be accessed and sound input can be read and manipulated. The library takes in the microphone data as samples and graphs the samples in real time. The x and y axes are time and amplitude.

Figure 12 shows work done so far in real-time sound analysis. The code starts off by opening a PyAudio "stream." This object takes in microphone data and organizes it into chunks. The chunks are loaded into a buffer and the graphing library, matplotlib.pyplot, takes the buffer and graphs it. There is also some data manipulation to get the microphone raw data into a graph-able format but details are unimportant for the scope of this paper. The Python, PyAudio, and PyPlot versions used here and throughout the rest of the paper are 3.7.7, 0.2.11, and 3.5.1 respectively [20] [21] [22]. The code for this work is presented in Appendix 1. Figure 13 shows another such graph.

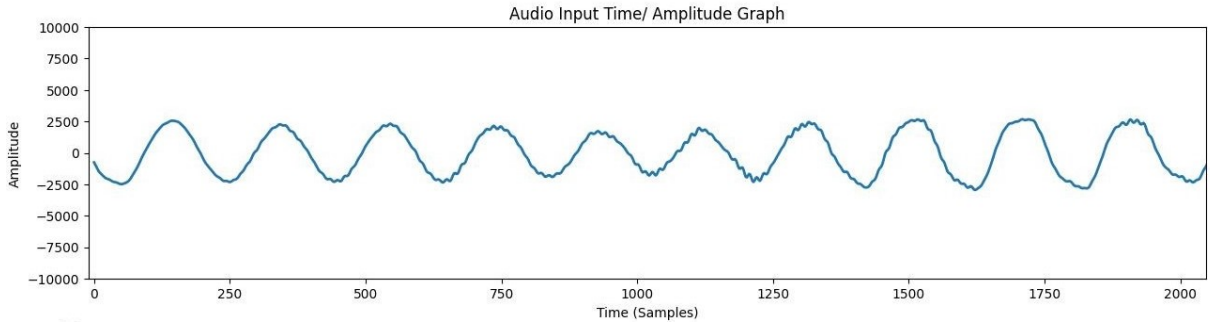


Figure 12: A graph of the amplitude over time of the author singing a note somewhere between G4 and A4 [14]

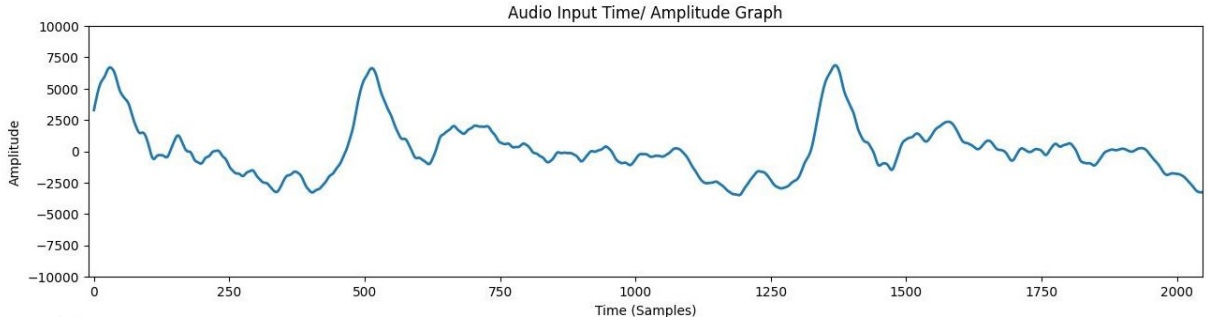


Figure 13: A graph of the amplitude over time of the author and the author's friend trying to harmonize.

4.2 Frequency Domain of a Real Time Signal

The next step in this direction is to look at the frequency domain. Taking the same exact signal data as shown in figures 12 and 13, a library called SciPy can be used to go from the time domain to the frequency domain. With the fast Fourier trasform (FFT) build into SciPy, the frequencies that make up the signals of figures 12 and 13 can be extracted. This was a bit more complicated than the previous analysis but luckily all of the mathematical intricacies are handled by SciPy. [23].

Figures 14 and 15 show the same note and harmony done by the author but instead of plotting time on the x-axis, the frequency is plotted instead. This is how the author approximated what note was being sung.

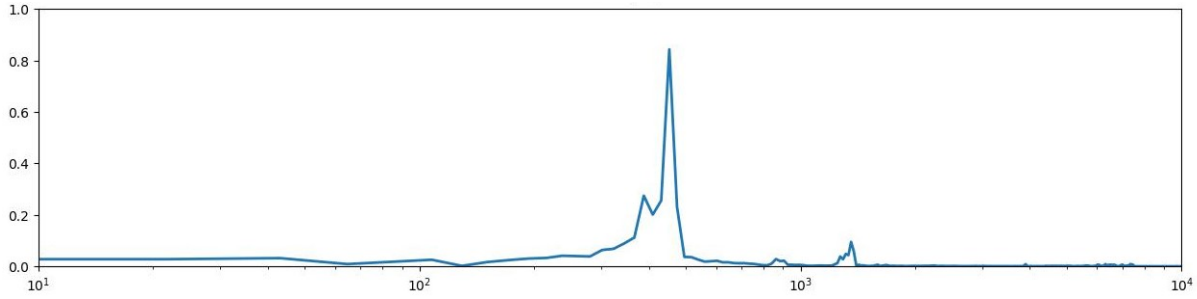


Figure 14: A graph of the frequencies sung by the author in figure 12

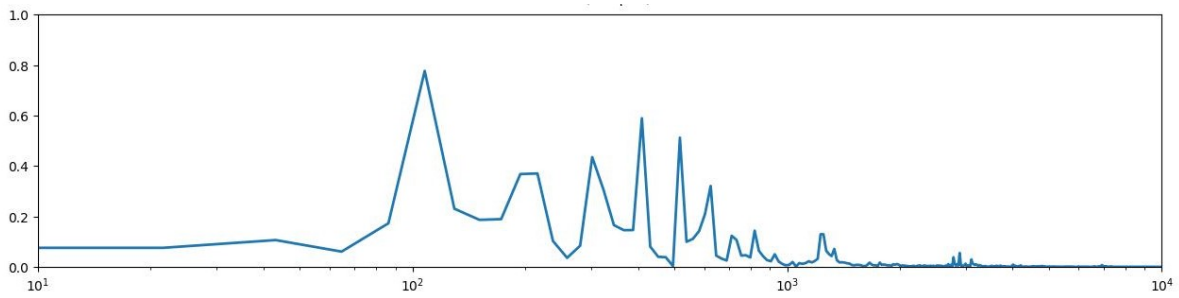


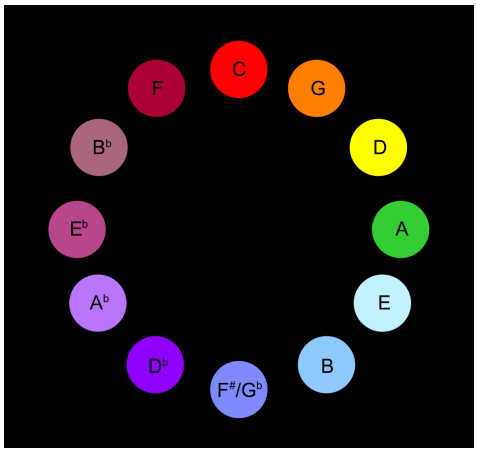
Figure 15: A graph of the frequencies sung by the author and author's friend's attempted harmony in figure 13

The program used to generate these graphs can certainly be expanded with better graphing libraries, cleaner fonts and colors, etc. It would also be interesting to add spectrogram creation to this section and maybe expand into other visualization methods.

5 Synesthesia & Extra Information

Although this paper's main focus is on the mathematical/computer aspects of music, it is important to mention the incredible visualization side of music and how different listeners can perceive it. Donny Snyder, cello performance major at Samford University, has graciously written this section giving a window into such visualizations.

> People will commonly say metaphorical phrases such as "I like what I hear," or "this food is sweet," but there is a minority of people who actually perceive these metaphorical phrases as literal. Synesthesia is the union of parallel sensations that occur inside a person's mind [24]. There are various forms of synesthesia, some synesthetes link auditory sensations with visual sensations, olfactory to visual, and some even link senses to emotions, such as numbers having distinct and different personalities. Chromesthesia is the type of synesthesia where color corresponds to a non visual stimulus. The stimulus in concern is sound. Often called color-hearing, a person perceives the non visual sound vibrations as color.
>
> Alexander Scriabin was a composer in the late romantic era of music. Born with a significant amount of chromesthesia, Scriabin would often compose his music to paint a picture with the colors he perceived. In his symphonic work Prometheus: Poems of Fire, Scriabin based the sounds off of his color wheel that he perceived when notes were played.



While chromesthesia is a rare condition, many people have been able to replicate its effect. Artists will often work alongside musicians and create a picture based off of the sounds produced. Computers are also able to take sound and convert it into color.

If specific frequencies or amplitudes could be assigned specific colors like in scriabin's color wheel, the resultant graph might serve as a more accurate or easier to read visualization for audio signals than the spectrogram. If not for scientific purposes, such an experiment could create unique and beautiful visualizations for different songs. These visualizations could serve to aid in understanding chromesthesia or aid artists in their creative processes.

6 Discussion & Conclusion

The expected outcome of this project was to research the mathematics behind music and in doing so develop a computer program which can take in microphone input, analyze it using Fourier analysis, and then display some form of output to represent the sound input. This paper has served as the culmination of that research detailing the mathematics in section 3 and the program in section 4 and appendix 1. Because this paper focuses on expository information instead of novel results, it can certainly be expanded to give more details into both the sound, music, and algorithms. One avenue of work could be in saving the real-time audio data to files for future analysis. Specific to synesthesia, it would be interesting to see what visualization techniques can be developed from a combination of spectrograms and Scriabin's wheel. Lastly, this paper serves as part of the requirements for bachelor's degrees in Computer Science and Mathematics.

Appendix 1: Code

This section introduces the python code behind some of the graphs used throughout the paper. Starting off with figure 3, the two graphs were created in Google Collaboratory with use of the libraries numpy, pyplot from matplotlib, and scipy [25][20]. Numpy was used for generating a sine wave and pyplot was used to create the actual graphs.

```
# Analyzing the frequencies of different instruments.

# This function generates a list containing the y-data of a sine wave.
#  - freq: the frequency of the sine wave.
#  - mult: number to multiply the frequency by (freq = freq * mult).
#  - xVals: the x-data of the sine wave, used in calculating the y-data.
#  - amp: the amplitude multiplier of the sine wave.
def ySine(freq, mult, xVals, amp):
  return amp * np.sin(2 * np.pi * (freq * mult) * xVals)

rate = 44100
duration = 5
freq1 = 440

# Computer
comp_x_vals = np.linspace(0, duration, duration * rate, endpoint=False)
comp_y_vals = ySine(freq1, 1, comp_x_vals, 1)

# Oboe
oboe_x_vals = np.linspace(0, duration, duration * rate, endpoint=False)
oboe_y_vals = ySine(freq1, 1, oboe_x_vals, 0.6) \
       + ySine(freq1, 2, oboe_x_vals, 0.5) \
       + ySine(freq1, 3, oboe_x_vals, 0.8) \
       + ySine(freq1, 4, oboe_x_vals, 1) \
       + ySine(freq1, 5, oboe_x_vals, 0.7) \
       + ySine(freq1, 6, oboe_x_vals, 0.55) \
       + ySine(freq1, 7, oboe_x_vals, 0.59) \
       + ySine(freq1, 8, oboe_x_vals, 0.55) \
       + ySine(freq1, 9, oboe_x_vals, 0.3) \
       + ySine(freq1, 10, oboe_x_vals, 0.22) \
       + ySine(freq1, 11, oboe_x_vals, 0.2) \
       + ySine(freq1, 12, oboe_x_vals, 0.48) \
       + ySine(freq1, 13, oboe_x_vals, 0.2) \
       + ySine(freq1, 14, oboe_x_vals, 0.25) \
       + ySine(freq1, 15, oboe_x_vals, 0.15) \
       + ySine(freq1, 16, oboe_x_vals, 0.275)

# Plots the two instruments.
comp_y_f = rfft(comp_y_vals)
comp_x_f = rfftfreq(duration * rate, 1 / rate)
```

```
41
42 oboe_y_f = rfft(oboe_y_vals)
43 oboe_x_f = rfftfreq(duration * rate, 1 / rate)
44
45 plt.plot(comp_x_f, np.abs(comp_y_f) / 100000)
46 # plt.plot(oboe_x_f, (np.abs(oboe_y_f) / 100000))
47 plt.xlim(0, 2000)
48 plt.ylabel("Amplitude")
49 plt.xlabel("Frequency (Hz)")
50 # plt.title("Oboe A4 Frequency Analysis")
51 # plt.title("Computer A4 Frequency Analysis")
52 plt.title("440 Hz Wave Frequency Analysis")
53 plt.show()
```

The next code section comes from section 3.1, Periodic Functions. The two figures 5 and 6 show different sine waves and use similar code. The first wave is a single sine wave of the note A (440Hz). First an array of x values is generated from 0 to 0.01 with a step size of 0.00001. These values are then used as input into a sine function which produces an array of y values. The x and y value arrays are then plotted onto a cartesian plane. The sine function used follows the one introduced in section 3.1, $f(t) = A\sin(\omega t + \varphi)$.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 q = 0
5
6 # Code for generating a single sine wave.
7 # The plotting data for the note 'A'
8 A_x_data = np.linspace(0, 0.01, 1000)
9 A_y_data = 4 * np.sin((2 * np.pi * 440) * A_x_data + q)
10
11 plt.plot(A_x_data, A_y_data)
12 plt.show()
13
14 # Code for generating four sine waves added together.
15 # The plotting data
16 A_x_data = np.linspace(0, 0.01, 1000)
17 A_y_data = (4 * np.sin((2 * np.pi * 440) * A_x_data + q)) + (3.95 *
       np.sin((2 * np.pi * 880) * A_x_data + q)) + (3.9 * np.sin((2 * np.
       pi * 1320) * A_x_data + q)) + (3.85 * np.sin((2 * np.pi * 1760) *
       A_x_data + q))
18
19 plt.plot(A_x_data, A_y_data)
20 plt.show()
```

This last portion of code comes from section 4, Experiments and Implementations. Both section Section 4 and comments within the code provide basic explanations of how it works and so no more details will be provided here. The author does not take credit for this last section of code and [26] was used under Fair Use as an educational

source for this paper. This section concludes the appendix for python code used in the paper.

```
# Necessary Imports
import pyaudio
import struct
import numpy as np
import matplotlib.pyplot as plt
import time
from scipy.fftpack import fft

"""
This file creates a window which displays two graphs. After reading input from the microphone, the first graph
displays the time/amplitude of the input audio signal. The second graph shows the time/frequency of the signal.
"""

# Variables
SAMPLES_PER_FRAME = 2048        # How many audio samples per frame to display
AUDIO_FORMAT = pyaudio.paInt16  # bytes per sample
CHANNELS = 1    # monosound
RATE = 44100    # samples per second (Hz)
Y_MAX = 10000
Y_MIN = -10000


# Creates an instance of PyAudio
pyAudio = pyaudio.PyAudio()


# Creates and starts a stream for reading input from the microphone and creating buffers for output.
stream = pyAudio.open(
    format= AUDIO_FORMAT,
    channels= CHANNELS,
    rate= RATE,
    input= True,
    output= True,
    frames_per_buffer= SAMPLES_PER_FRAME
)


# Creates a graph to plot the microphone input onto.
fig, (axAmp, axFreq) = plt.subplots(2, figsize= (15, 8))


# Creates a range of x values for the time/amplitude graph.
# The values are every even number from 0 to two times the samples per frame.
```

```
# Ex. [0, 2, 4, 6, 8, ..., 4094] when SamplesPerFrame=2048
x_vals = np.arange(0, SAMPLES_PER_FRAME * 2, 2)
x_fft = np.linspace(0, RATE, SAMPLES_PER_FRAME)


# This graphs the x_vals and for the y values it takes a random number between 0 and 1.
# We will change the y values later on in the code.
# np.random.rand() creates a list of the same size as SAMPLES_PER_FRAME where each element
# is a random number between 0 and 1.
line, = axAmp.plot(x_vals, np.random.rand(SAMPLES_PER_FRAME), '-', lw = 2)
line_fft, = axFreq.semilogx(x_fft, np.random.rand(SAMPLES_PER_FRAME), '-', lw= 2)


axAmp.set_title("Audio Input Time/ Amplitude Graph")
axAmp.set_xlabel("Time (Samples)")
axAmp.set_ylabel("Amplitude")
axAmp.set_ylim(Y_MIN, Y_MAX)
axAmp.set_xlim(-10, SAMPLES_PER_FRAME)

axFreq.set_ylim(0, 1)
axFreq.set_xlim(10, 10000)

plt.show(block=False)


frame_count = 0
start_time = time.time()
while True:

    mic_in_data = stream.read(SAMPLES_PER_FRAME)

    formatted_data = np.frombuffer(mic_in_data, dtype= np.int16)

    # Updates the y values of our graph with the newly formatted data
    # Draws the graph to the screen.
    line.set_ydata(formatted_data)

    y_fft = fft(formatted_data)
    line_fft.set_ydata(np.abs(y_fft[0:SAMPLES_PER_FRAME]) * 2 / (2000 * SAMPLES_PER_FRAME))

    try:
        fig.canvas.draw()
        fig.canvas.flush_events()
        frame_count += 1
    except TclError:
        frame_rate = frame_count / (time.time() - start_time)
```

```
        print("average frame rate = {:.0f} FPS".format(frame_rate))
        break


# Ends the stream.
stream.stop_stream()
stream.close()
pyAudio.terminate()
print("Exited Successfully")
```


Acknowledgements

Dr. Kwadwo Antwi-Fordjour
Dr. Brian Toone
Professor Kevin Kozak
Donny Snyder
Samford University
Mr. and Mrs. Paul and Eileen Lad
Alan Crisologo for being the "author's friend," (See Figure 4)
✝ TKC (Thy Kingdom Come)

References

[1] Karl H. Wö rner. *History of Music, 5th Edition, A Book for Study and Reference*. The Free Press, A Division of Macmillan Publishing Co., Inc., 866 Third Avenue, New York, N.Y. 10022, 1973. ISBN 72-90547. Translated and Supplemented by Willis Wager.

[2] Otto Bettmann Paul Henry Lang. *A Pictorial History of Music*. W. W. Norton and Company, Inc., New York, 1960. ISBN 60-6822.

[3] Richard E. Berg. Acoustics, Invalid Date. URL `https://www.britannica.com/science/acoustics`. Accessed 3 April 2022.

[4] Lynn C Robertson and Noam Sagiv. *Synesthesia: Perspectives from Cognitive Neuroscience*. Oxford University Press, 2004.

[5] Howard Boatwright. *Introduction to the Theory of Music*. W. W. Norton and Company, Inc., 1956.

[6] Grant Sanderson. But what is the fourier transform? a visual introduction., 2018. URL `https://www.youtube.com/watch?v=spUNpyF58BY`.

[7] Norman Morrison. *Introduction to Fourier Analysis*. John Wiley and Sons, Inc., 1994.

[8] P. C. Shields. *Elementary Linear Algebra*. Worth Publishers, New York, 1968.

[9] Nathan Lenssen and Deanna Needell. An introduction to fourier analysis with applications to music. *Journal of Humanistic Mathematics*, 4(1):72–91, 2014.

[10] Merriam-Webster. Hertz., 2022. URL https://www.merriam-webster.com/dictionary/hertz. In Merriam-Webster.com dictionary.

[11] Carl R. (Rod) Nave. Sound waves in air, hyperphysics. Digital image from HyperPhysics website, 2001. Hosted by Georgia State University.

[12] Nicholas Temperley. Tuning and temperament. 2007. URL https://www.britannica.com/art/tuning-and-temperament.

[13] Larry Jacobsen. Just vs equal temperament chart, 2012. URL https://www.flickr.com/photos/ljguitar/7983224694/in/photostream/.

[14] B. H. Suits. Physics of music - notes, tuning, 2022. URL https://pages.mtu.edu/~suits/notefreqs.html. Website hosted by the Physics Department of Michigan Technological University.

[15] Wikipedia contributors. Concert pitch from wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/w/index.php?title=Concert_pitch&oldid=1072127260. Online; accessed 3-March-2022.

[16] H. P. Hsu. *Fourier Analysis*. Simon and Schuster, New York, 1970.

[17] Wikipedia contributors. Sine wave — Wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/w/index.php?title=Sine_wave&oldid=1071914615. [Online; accessed 3-March-2022].

[18] Wikipedia contributors. Angular frequency — Wikipedia, the free encyclopedia, 2021. URL https://en.wikipedia.org/w/index.php?title=Angular_frequency&oldid=1058712937. [Online; accessed 3-March-2022].

[19] Wikipedia contributors. Euler's formula — Wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/w/index.php?title=Euler%27s_formula&oldid=1074655318. [Online; accessed 29-March-2022].

[20] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[21] People.csail.mit.edu. Pyaudio documentation - pyaudio 0.2.11, 2016. URL https://people.csail.mit.edu/hubert/pyaudio/docs/.

[22] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

[23] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, St'efan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, Ilhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antonio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. Scipy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261–272, 2020. URL `https://rdcu.be/b08Wh`.

[24] Richard E. Cytowic. Synesthesia: A union of the senses, 2002. URL `https://search-ebscohost-com.ezproxy.samford.edu/login.aspx?direct=true&db=nlebk&AN=74998&site=eds-live&scope=site`. EBSCOhost.

[25] Harris C.R., Millman K.J., and van der Walt S.J. Array programming with numpy. *Nature*, 585(7825):357–362, September 2020. URL `https://doi.org/10.1038/s41586-020-2649-2`.

[26] Mark Jay. Let's build an audio spectrum analyzer in python! (pt. 1) the waveform viewer., 2017. URL `https://youtu.be/AShHJdSIxkY`.