An Ideal Guitar Tuner: Optimizing Consonance by Minimizing Beating

Cassi J L Burton

Physics 492R Capstone Project Report
September 16, 2015
Dr. Dallin S Durfee

Contents

List of Figures

Acknowledgements

I would like to express my profound appreciation to Dr. Dallin Durfee for his contributions. It was his vision that enabled this project to become a reality from its beginning, and his broad spectrum of invaluable talents that brought it to life. His unfailing work ethic, endless creativity, and musical ability were all major factors in the success of this research. Furthermore, his experience and in-depth knowledge of music ensured the musical value of the end result.

Dr. Durfee's dedication to his students and his remarkable efficacy in teaching are exceptional in the academic world. I am privileged to have had him as an advisor and professor throughout my undergraduate experience.

I am grateful to have worked with every one of my professors at BYU. They equipped me over the years with the skills and knowledge that culminated in this project. I would also like to thank the BYU Department of Physics and Astronomy for providing the materials and technology that made this project possible.

Finally, I fondly acknowledge the love and support of my family. My beloved husband has sustained me with his constant patience and encouragement. The uplifting friendship of my mother, the love of my father, and the unwavering support of my grandfather have sustained me in my academic efforts.

It has been my privilege to compile this report and gain the knowledge involved by working on this project.

Abstract

An Ideal Guitar Tuner: Optimizing Intonation by Minimizing Beating

Cassi J L Burton

Department of Physics and Astronomy

Bachelor of Science

In music, consonance is the quality of sound sought after by musicians to create pure-sounding, pleasant music. The governing laws of physics behind consonance dictate that a guitar cannot have perfect consonance across all chords. To make all chords reasonably consonant, the equal temperament scale is accepted as the standard tuning scheme today. Its downfall is that it does not provide the best possible consonance in any one song. Since a song is comprised of only a few chords, creating consonance across all chords is not necessary—only across the chords being used in a song. With physics, we can calculate the frequencies to which each of the strings on a guitar should be tuned in order to optimize consonance across any set of chords. This report discusses the calculation used to optimize consonance and the Android app that I've developed to perform that calculation.

A senior capstone report submitted to the faculty of
Brigham Young University
In partial fulfillment of the requirements for the degree
Bachelor of Science

Department of Physics and Astronomy
Brigham Young University
September 2015

Chapter 1. Introduction

1.1 The Physics of Consonance

Music is both the art and science of sound—it is a phenomena created by combining frequencies of sound waves in an aesthetically pleasing manner. At the foundation of music's pleasant quality is intonation. Intonation describes the interactions between frequencies that are described as either dissonant (out of tune) or consonant (in tune). Throughout history, intonation schemes have been developed by scientists, philosophers, and musicians in an effort to improve the beauty and appeal of music. Musicians today still search for alternate tunings to better suit their instruments or musical pieces by improving consonance. This project seeks to serve guitarists by creating custom tuning frequencies for each guitar string that will optimize consonance across a custom set of chords. Musicians will be able to access this innovative intonation tool as a simple android app.

1.1.1 Beating

In order to understand how to better serve the modern musician by improving consonance, we must first understand the fundamental physics that governs music and consonance. The dominant underlying cause of human perception of intonation—and therefore consonance—is a phenomenon called "beating": the interaction between sound waves that can be perceived by the human ear. The interacting frequencies that create music are the fundamental frequencies, also known as "notes" or "pitches" that are played on an instrument and the harmonics (discussed in Section 1.1.4) of a frequency. If we look at two pure sine waves representing two different frequencies, we can see a simple visual representation of beating in Figure 1.

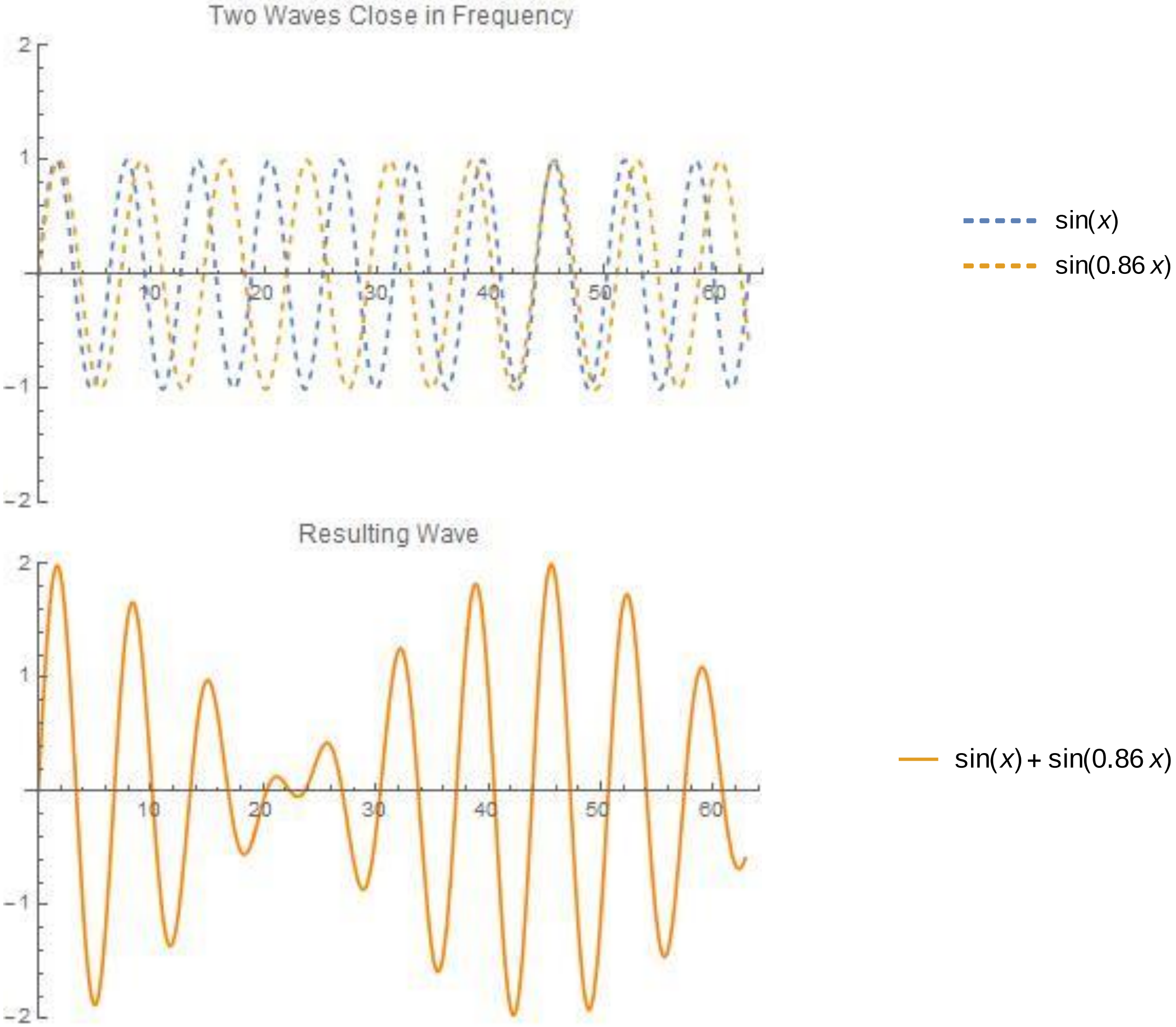


Figure 1. Two waves close in frequency interfering with each other. The interfering waves, shown as the dotted lines, result in a beat wave, shown as the solid line.

When two sound waves interact, constructive and destructive interference occurs. As a peak of one wave lines up with the valley of the other wave, the amplitude of the resulting wave is diminished: the waves interfere destructively. As the peaks of both waves line up, the amplitude of the resulting wave increases: the waves interfere constructively. The resulting wave has beats, which gets louder and softer at the beat frequency. The frequency at which the beats occur in this resultant wave can be calculated as the difference between the two interacting frequencies. For example, if two notes of frequencies 440 Hz and 460 Hz are played together, the resulting beat frequency will be 20 Hz: $460 - 440 = 20$ Hz.

Beating occurs between any two frequencies, but for the purposes of consonance in music we narrow down the beating with which we are concerned to the beats whose frequencies are between 0 and 50 Hz (McMurtney, Fleming, & Steffensen, 2013). This is the range of beat frequencies that are processed by the human brain to result in dissonance. Outside of this range, beating is not consciously heard so it does not have a perceptible effect on intonation. This is because two frequencies that are relatively far from each other (enough that beats between them are not perceived as dissonance) can be heard as separate notes instead of as their blended result.

1.1.2 Intervals

When two frequencies are far enough apart that the beating between their fundamental frequencies is not processed as bad intonation, they are considered together to make up an "interval". Intervals are pleasant when beats caused by the interaction of the involved notes are minimized—in other words, when they are consonant. We define intervals in music as the distance between two notes of the given musical scale that is in use to make it easier to discuss them. Any two notes can be considered an "interval" apart, but it is accepted in music that intervals are predefined as specific ratios of frequencies. Most combinations of notes as defined in various musical scales are related by the ratios of defined intervals.

We can calculate the beating that occurs in an interval just as we calculated beating that occurs between any two random frequencies. Consider the combination of two frequencies whose interval relation is described as a "minor third," or the difference in frequency between the tonic and the third note of a minor scale; the notes concert A at 440 Hz and C at 523.25 Hz make up a minor third (see Figure 7). The beating between these fundamental frequencies would be 83.25 Hz, a rate that wouldn't result in audible dissonance between these two notes. On the other hand, the relatively close frequencies of 440 and 450 Hz—which do not form a predefined musical interval—would cause a beat frequency of 10 Hz. A beat rate of 10 Hz is within the intonation-affecting range, so the two notes played together would cause dissonance.

1.1.3 Chords

As a more complicated example, consider beating across multiple notes spaced at specific, predetermined intervals. In this example, we will look at beating within a "major triad". A major triad is a simple chord comprised of a root note, its third, and its fifth. We can take the major third between 440 Hz (concert A) and 554.37 Hz (C$^\sharp$) to use in this example. Adding a fifth interval in relation to A to build a full major triad will include the frequency 659.25 Hz (E). Considering only the fundamental frequencies that comprise the chord (excluding harmonics), we calculate the beating between each of the note combinations:

Note 1	Note 2	Beating
440 Hz (A)	554.37 Hz (C$^\sharp$)	114.37 Hz
440 Hz (A)	659.25 Hz (E)	219.25 Hz
554.37 Hz (C$^\sharp$)	659.25 Hz (E)	104.88 Hz

Figure 2. Beat rates between notes making up an A minor chord. None of the beats outlined here will be heard as dissonance.

One can see that there is no audible beating between the frequencies with the current considerations: none of the beat frequencies are within the intonation-affecting beat range of 0-50 Hz. However, this situation is an idealized example of calculating beat rates within a chord to familiarize us with the concept of beating across multiple intervals.

1.1.4 Harmonics

More complicated than the examples of beating we just looked at are real musical notes, which are never “pure.” A note is comprised of many more frequencies than just its fundamental frequency; these frequencies are called harmonics. In an ideal, “pure” harmonic series, the harmonics are frequencies which are integer multiples of the fundamental frequency. In linear resonators like the strings of a guitar or the air column of a trumpet, the harmonics are approximately integer multiples of the fundamental frequency. We will consider the dispersion which occurs in the instruments to be negligible in our calculations since we can approximate our guitar strings as ideal strings—ideal strings are non-dispersive (Morin, Drafted 2009).

Every harmonic present in music beats against every other frequency, just as fundamental frequencies do; this is why we made the distinction in the previous example that there is no audible beating within a chord *excluding harmonics*.

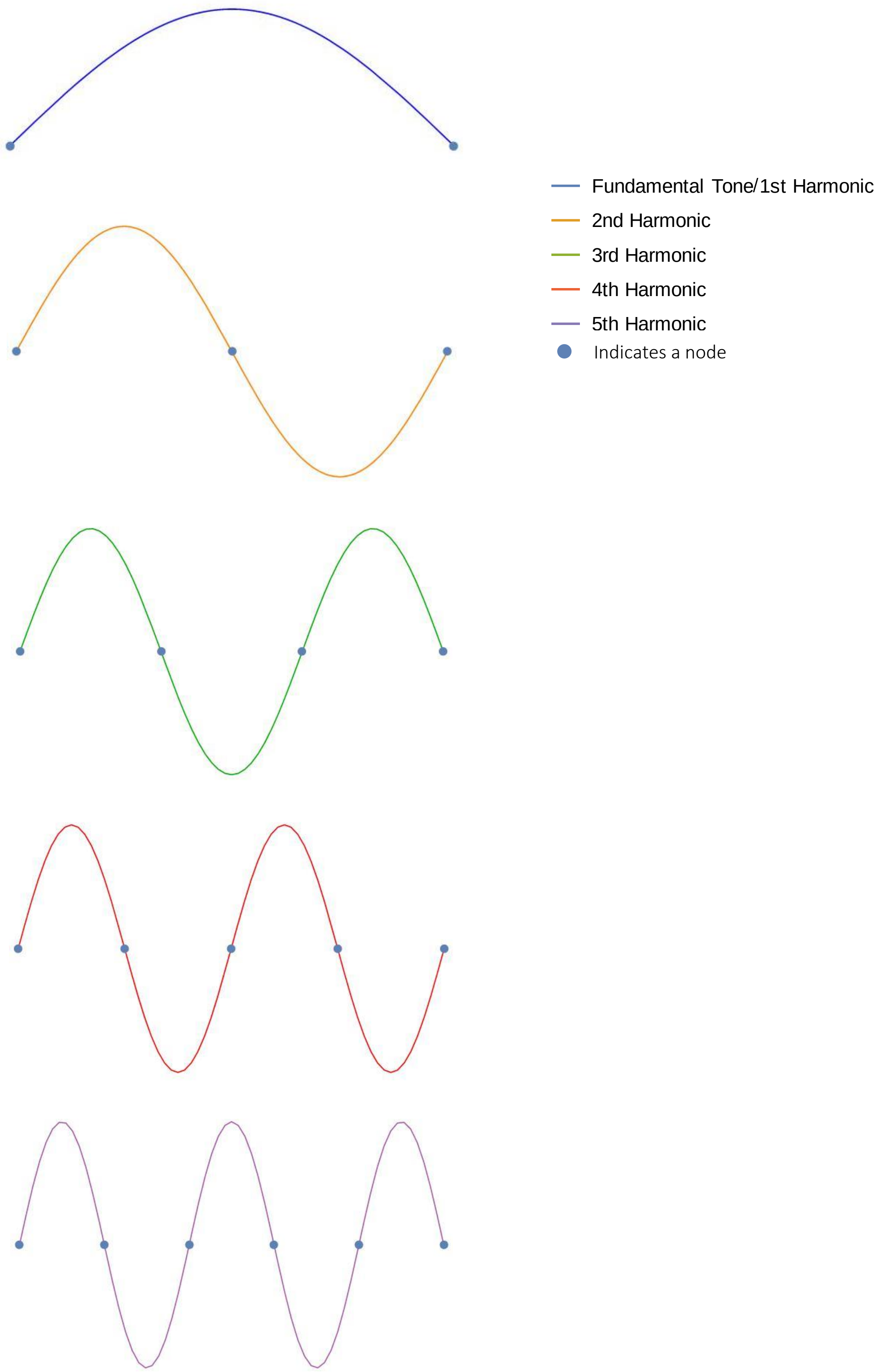


Figure 3. The shapes of the first five harmonics of a string.

1.1.5 Harmonics on the Guitar

Looking at Figure 3, we see the representation of a string’s first five harmonics. A string vibrates as a combination of the harmonic shapes represented, sounding a note. The ends of a string will always be fixed on a musical instrument, dictating that a node on each end of the string will be present in all harmonics. The two nodes present at the end of each guitar string are forced by the nut (or a fret) and the bridge of the guitar. The frequency of the first harmonic of a string—the one that only has the two forced nodes in its vibration—is known as the fundamental frequency of a note; vibrations that occur at this frequency on a string make up the dominant frequency that is heard in a note. One additional node on a vibrating string (as in the 2nd Harmonic) will result in a note two times the fundamental frequency; two extra nodes (as in the 3rd Harmonic) will result in a note three times the fundamental frequency, and so on. Each of these frequencies has the same properties as the fundamental frequencies we have discussed in examples up to this point.

Guitarists—and many other string musicians—change the note of a string by placing a finger on the string to shorten the string. This forces a node on the string, changing the frequencies of its vibration. Any note played on a guitar string will consist of a combination of the string’s harmonics.

1.1.6 Beating across Harmonics

Since harmonics are waves with the same properties as fundamental tones, they can also beat against one another and affect intonation—they do so in the exact way the simpler fundamental frequencies beat against one another. This complicates the beating that occurs within even a single interval; the increased number of frequencies that must be considered between any two random notes greatly increases the likelihood of dissonance between them. With our new knowledge of harmonics, we now consider the beating between the fundamental notes *and* their harmonics to find the beating that occurs. For example, let’s look at the harmonics for the fundamental frequencies 440 Hz (A) and 659.25 Hz (E)—outlined in Figure 4.

Fundamental Frequency (Hz)	2nd Harmonic	3rd Harmonic	4th Harmonic	5th Harmonic
440 (Concert A)	880	1320	1760	2200
659.25 (E)	1318.5	1977.75	2637	3296.25

Figure 4. Harmonics of the equal temperament notes A and E.

Most of the beats between the harmonics of these two notes are negligible since they don’t fall within the range relevant to intonation. However, the frequencies of the third harmonic of A and the second harmonic of E are very close: 1320 and 1318.5 Hz respectively. This will result in a beat frequency of 1.5 Hz and create audible dissonance when the notes are played together.

1.1.7 Adjusting Frequencies to Minimize Beating

To improve intonation between the two notes in our example, we can adjust the fundamental frequencies that are played by a small amount to improve the beating between the harmonics. If we adjust the frequency of E to make the note exactly 3/2 times the frequency of concert A, the second harmonic of E and the third harmonic of A will be at the exact same frequency (outlined in Figure 5):

$$Adjusted\ Frequency = 440\ Hz * \frac{2}{3},$$

$$= 660\ Hz\ .$$

With this slight adjustment to the frequency of E, the beat frequency between the harmonics of concern has gone to zero and all the other beat frequencies remain negligible, seen in Figure 5. This type of interval relationship that improves intonation perfectly is called a “just” interval. Just intervals are relationships that can be represented as ratios of small integers, such as the 3/2 ratio that was used in this example (see Figure 7 for a complete list of just interval relationships).

Fundamental Frequency (Hz)	2nd Harmonic	3rd Harmonic	4th Harmonic	5th Harmonic
440 (Concert A)	880	1320	1760	2200
660 (adjusted E)	1320	1980	2640	3300
659.25 (equal temperament E)	1318.5	1977.75	2637	3296.25

Figure 5. Harmonics of 440, 660, and 670 Hz. The second harmonic of 660 Hz will not beat with the third harmonic of 440 Hz.

From this example, we see that we can adjust frequencies to minimize beating. As one can see from investigating the fundamental physics of consonance, beating can be minimized between two frequencies by manipulating tuning. However, physics also demonstrates that tuning an instrument to create perfect consonance in multiple chords across keys is impossible. Though some intervals can be adjusted to form just intervals, the consequence involves another interval being adjusted in the wrong direction and causes beating to worsen. For example, if you add to the previous example that you'd also like to play a second interval in relation to Concert A (as in a suspended chord), we would have included the frequency 493.88 Hz (B). The fourth harmonic of B is very close to the third harmonic of E (see Figure 6). Before the adjustment of E to optimize consonance with A, the beat rate between E and B would have been 2.23 Hz. After the adjustment of E, the beat rate between E and B is 4.48 Hz—much more pronounced beating than before.

Fundamental Frequency (Hz)	2nd Harmonic	3rd Harmonic	4th Harmonic	5th Harmonic
440 (Concert A)	880	1320	1760	2200
493.88 (B)	987.76	1481.64	1975.52	2469.40
660 (adjusted E)	1320	1980	2640	3300
659.25 (equal temperament E)	1318.5	1977.75	2637	3296.25

Figure 6. Consequences of adjusting a frequency. Adjustment to reduce beating in one interval results in worse beating for another.

This impossibility of perfection in consonance across multiple intervals is what drives the need for musical scales. A musical scale attempts to create conditions such that any pair of notes and their harmonics has the least amount of unpleasant beats as possible. More realistically stated, the goal of a musical scale is to minimize unpleasant beating or to distribute the beating across all intervals within the scale in the most pleasing manner.

Scales have been developed throughout history by musicians, astronomers, and scientists in attempts to optimize consonance and clarity of music (Durfee & Colton, 2012) by distributing beats in various ways. The Ptolemaic scale, meantone scales, and compromise scales each prioritize beating differently: the Ptolemaic scale focuses on creating beat-free intervals in one key, particular meantone scales focus on minimizing beating in the major third intervals, and compromise scales do just as you would think: compromise intonation in one place to improve it in another. The most commonly used compromise scale today—the equal-temperament scale—is meant to provide equal intonation across all keys. This enables musicians to switch keys or play another song without having to retune or switch instruments.

In order to be a perfect compromise of intonation across all keys, the frequencies of the equal temperament scale must be equally separated in the way that we perceive pitch. Since we hear sound logarithmically, frequencies of the equal temperament scale are separated by a factor of $2^{1/12}$. That gives twelve spacing of frequencies—known as "half steps"—between two notes that differ by an octave. For example, between two octave C's, all of the existing Western notes are B$^{\sharp}$/C, C$^{\sharp}$/D$^{\flat}$, D, D$^{\sharp}$/E$^{\flat}$, E/F$^{\flat}$, E$^{\sharp}$/F, F$^{\sharp}$/G$^{\flat}$, G, G$^{\sharp}$/A$^{\flat}$, A, A$^{\sharp}$/B$^{\flat}$, and B/C$^{\flat}$ respectively. Let's take a moment to observe the relationships of notes in this chromatic scale and create a foundation for understanding the standard frequencies that can be played on a guitar. C and G form a fifth interval since G is the fifth note in a proper C major scale: (1) C, (2) D, (3) E, (4) F, and (5) G. The two notes are separated by 7 half steps as defined by the equal-temperament scale, so G's frequency will be $2^{7/12}$ times the frequency of C. Thus, if middle C's frequency is 261.6 Hz (as it is on the standard piano), then the frequency of G—a fifth interval above middle C—will be $2^{7/12} * 261.6 = 391.99$ Hz. The frequency of every note in the equal temperament scale can be calculated based off of one chosen frequency, as seen in this example. Today the standard equal temperament scale is based around the frequency of "Concert A" that has been mentioned before in examples, set at 440 Hz. We will use this standard in our discussion of this project.

Note	Example Scale	Interval	Equal Temperament Frequency Factor	Just Interval Frequency Ratio
1	C	unison	1	1
2	C♯/D♭	minor second	$2^{1/12}$	16/15
3	D	major second	$2^{2/12}$	9/8
4	D♯/E♭	minor third	$2^{3/12}$	6/5
5	E/F♭	major third	$2^{4/12}$	5/4
6	E♯/F	perfect fourth	$2^{5/12}$	4/3
7	F♯/G♭	augmented fourth	$2^{6/12}$	45/32
8	G	perfect fifth	$2^{7/12}$	3/2
9	G♯/A♭	minor sixth	$2^{8/12}$	8/5
10	A	major sixth	$2^{9/12}$	5/3
11	A♯/B♭	minor seventh	$2^{10/12}$	9/5
12	B/C♭	major seventh	$2^{11/12}$	15/8
13	B♯/C	octave	$2^{12/12}$	2

Figure 7. Intervals of the equal temperament chromatic scale (Durfee & Colton, 2012).

1.2 Understanding the Guitar

1.2.1 Common Conventions

We need to be aware of a few common conventions to understand the needs of guitarists and to apply the physics of sound to their instrumental situation. Though we anticipate deviating from the equal temperament scale to improve intonation, we must still use its conventions: the frets on a guitar are set at equal-temperament intervals. This way when a guitarist presses a string down on the fingerboard on the first fret (the fret closest to the nut), the fret creates a node and forces the string's frequency to go up by an equal-temperament half step: the new frequency will be a factor of $2^{1/12}$ times the string's original frequency for each fret up the neck that is utilized. Since the frets are permanently set in place, we will need to alter the fundamental frequency of the string rather than the fret it will be voiced with in order to adjust intonation. It is also important to note that guitarists create chords by placing their fingers on multiple strings at various frets and strumming several strings at the same time.

The guitar has six strings; they are typically tuned to the notes E, A, D, G, B, and E (see Figure 7), which are standardly tuned to the equal temperament frequencies of 82.41, 110.00, 146.83, 196.00, 246.94, and 329.63 Hz respectively. We refer to the frequencies of each string as f_6, f_5, f_4, f_3, f_2, and f_1 respectively. We will discuss frequency relationships between the strings in terms of the string frequencies. For example, f_6 and f_5 are an interval of a fourth apart (five half-steps), so we can say that $f_6 * 2^{\frac{5}{12}} = f_5$. In the modern tuning of the guitar, it is important to note that f_3 and f_2 are unique in their frequency relationship, since they are only a major third apart: $f_3 * 2^{\frac{4}{12}} = f_2$. It is also convenient to refer to the frets numbered sequentially from one—starting at the nut. Since each fret represents a half-step in the equal temperament scale, playing string f_1 with the finger on the first fret would result in the frequency$f_1 * 2^{\frac{1}{12}}$. Playing string f_1 at the fourth fret would result in the frequency $f_1 * 2^{\frac{4}{12}}$. This makes it easy to remember the effect that each fret will have on the string's original frequency: the new frequency of a string with original frequency f, voiced with the finger on fret x, will be $f * 2^{\frac{x}{12}}$.

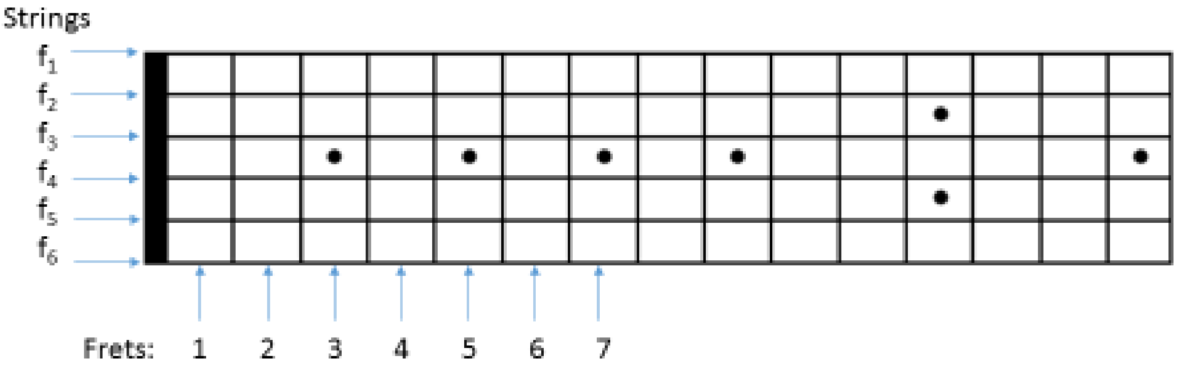


Figure 8. Diagram representing the neck of the guitar. Strings and frets are labeled as referred to in this paper.

1.2.2 Unfulfilled Intonation Needs of Guitarists

Now that we understand the fundamental physics of sound and the standards of the modern guitar, we can discuss the meaning of intonation for guitarists. It is accepted today that perfectly matching the frequencies of the strings to the preset frequencies in the equal temperament scale is the accepted definition of being in tune. But as we discussed earlier, the equal temperament scale still produces beating when notes are played together—its advantage is that it produces equal beating across all scales. Guitarists can hear beating even after perfectly matching their string frequencies with those set in the equal temperament scale. This can be very frustrating to guitarists—especially those who don't understand the physics that governs sound and dictates that intonation can never be perfect across all intervals.

In an effort to relieve this frustration with beating on the guitar, much has been written about tuning guitars. Some methods of tuning involve tweaks to the guitar such as nut, fret, and saddle adjustment (Locke) to better maintain the frequencies that strings are tuned to. These types of fixes "are not for the faint-hearted", Luthier Locke warns, and are not realistic options for the typical guitarist. Other musicians give direction on the methodology of tuning strings (Flatley, 2007), some completely imprecise and vague: "If you want to be really accurate, the fifth should be slightly smoother than the fourth, but there's not a lot in it."

Some tuning schemes in existence can be beneficial for improvement of consonance in very specific situations. For example, guitarists may choose to use "open tuning" schemes, meaning that the six strings' frequencies comprise a chord without having to be fretted. This tuning scheme is great for musicians intending to only play straight-barred chords since the intervals can be tuned by ear to nearly just intervals. This tuning scheme is usually only seen in slide guitar playing in which chords are played by pressing down on all the strings on the same fret. If another chord shape were to be used—altering string-interval relationships that were optimized for consonance—the intervals would likely produce more beating than in equal temperament tuning since the beating was minimized for a single chord shape. Another similar example is overtone tuning (Hanson, 1995) in which the open strings are in the same octave and form just intervals. As in open tuning, any chord shape besides the one optimized will have more beating than equal temperament tuning would cause.

As we can see from these examples, many tuning options are not calculated based on physics. Most are the products of individual opinions and experimentation; they are useful only in very specific situations. To create a more accurate and versatile method of achieving consonance, we consider what we know about the physical laws governing consonance. From our discussion of beats, we know that a tuning which invokes a perfect compromise between all keys is not necessary when only a few chords and keys are being used by a musician. Furthermore, a small tweak in a frequency to make one interval perfectly in tune can make others sound worse by the laws of physics. Therefore, any tuning scheme that deviates from equal temperament without taking into account which chords will be played is inevitably flawed. An ideal tuning scheme will take into account which chords will be used in a given piece of music.

This project is an endeavor to provide the best intonation for guitarists based on physical calculations—given the conditions of specific keys and chords that will be played. A custom tuning that minimizes beating between any given set of chords can be mathematically calculated, thus perfectly optimizing consonance for a specific situation. The end result of such a tuning scheme would be more consonant pieces of music—something musicians and scientists have been searching for throughout history.

1.2.3 Fulfilling Intonation Needs of Guitarists: Currently Available Resources

Other tuning resources that are available include programs, such as MIDI software, that allow for experimentation and better understanding of intonation. For example, Tonalsoft is "a music composition application which allows the user to create any imaginable tuning and compose music using those tunings and a valuable analytical tool which aids in the understanding of tuning theory and the various qualities of different types of musical tunings" (Monzo, 2005). Spectratune is a "musical pitch and spectrum analyzer" (Spier, 2013). Temperament Studio (Durfee D. S., 2013) demonstrates the sound of various historical intonations in pieces of music. Each tuning scheme demonstrates different beat patterns within songs and provides insight into the motivation for our modern-day intonation scheme. Most of these resources are designed around keyboard instruments or instruments where each note's frequency can be customized. This is not always useful for guitarists since only the frequencies of each of the six strings can be readily altered.

1.2.4 This Project as a Resource

In order to provide a straightforward tuning tool for guitarists, this project envisions the ultimate authority on custom tuning schemes for any combination of chords a guitarist would like to play. Out of all the methods to improve intonation on a guitar, it is the most in line with physical laws, using calculations to create a customized tuning scheme based on chord usage—specifically for guitar. Tunings can be calculated quickly for any song. Furthermore, guitarists can directly and quickly apply the custom tunings to their own guitars instead of experimenting with intonation on the internet or with expensive software programs. Musicians won't have to rely on time-consuming research and experimentation to find a desirable tuning scheme. Finally, it will be convenient: guitarists can generate a custom tuning in seconds with a free, simple Android app.

Chapter 2. Methods

2.1 The Algorithm for Minimization of Beating

2.1.1 An Overview

We now describe in overview the algorithm used to calculate the frequencies for optimized intonation. As the string frequencies of a guitar are easily altered, the frequencies of the strings will be considered the variables in our minimization of beating across a custom set of chords. We will choose one string—f_6—to be the constant frequency, giving a reference for the other frequencies to be calculated upon. Given a set of chords and their voicings, we will look at the relationships created in each chord between each of the strings. Each of these relationships can be seen as an equation with the string frequencies as variables. All of the frequency relationships in the set of chords can be written as a system of equations that can be minimized by using least-squares matrix manipulation.

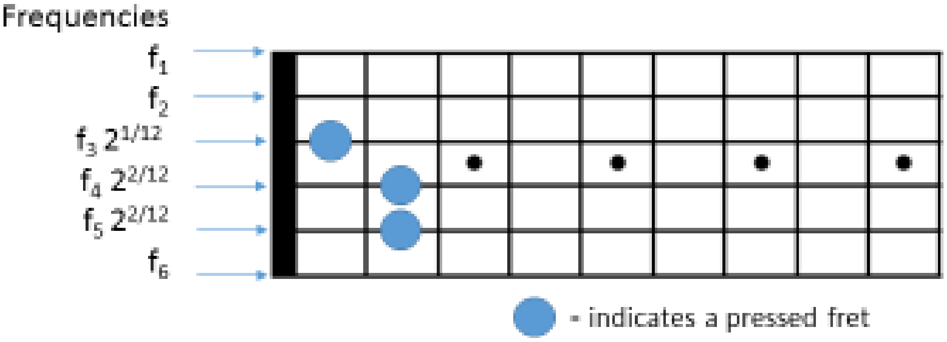


Figure 9. Voicing and frequencies of the open E major chord.

As an example of the algorithm, we will walk through the detailed calculation of frequencies for a single chord's intonation optimization. In this example, we will use the E major chord. E major is voiced on the guitar seen in Figure 9: the blue dots represent where the finger is pressed on a string, giving the location of the fret we will integrate into the calculation. String f_6 will be the constant frequency in our equations, set to the equal-temperament frequency of 82.408 Hz (E). We first look at the interval relationship between f_6 and f_5 with the voicing of the chord in consideration. String f_6 is not voiced with a fret. String f_5 is voiced with the second fret. We represent the notes voiced on the guitar in terms of these variables; since no frets are pressed on

f_6, we represent the voiced note by f_6. Since the second fret is voiced on f_5, the frequency of the voiced note will be the frequency of f_5 plus two equal-temperament half steps, or

$$voiced\ frequency\ on\ f_5 = \ f_5\ 2^{\frac{2}{12}}. \tag{2.1}$$

Now we must express the relationship of these two notes in terms of the strings' fundamental frequencies. We know we would like the frequencies sounded from f_6 and f_5 to form an interval with the least amount of beating as possible—a just interval. Recall that the original relationship between f_6 and f_5 is a fourth, meaning there are five half steps between the frequencies (see Figure 7). Since there are two extra equal-temperament half-steps created between f_6 and f_5 by pressing on the second fret of f_5, seven half steps now exist between the voiced notes, creating a fifth. From Figure 7, a just fifth relates the frequencies by a factor of 3/2. Therefore, we would like the voiced frequencies of f_6 and f_5 to be related by the following equation:

$$f_5\ 2^{\frac{2}{12}} = \frac{3}{2}\ f_6\ . \tag{2.2}$$

But this equation does not directly give any information about our end goal: minimizing the beating that occurs between all the voiced frequencies. If we only used equation 2.2 to set the frequency of f_5, we would eliminate beating across strings f_6 and f_5. We know that doing so could potentially result in worse beating for other intervals involving these strings, so we must choose f_5 in a way that optimizes consonance for all intervals involved in the chord. Thus, we represent the beat rate between f_6 and f_5 as an equation so we can manipulate its value later on. As we recall, this beat rate is calculated by finding the difference in the frequencies:

$$BeatRate = \ -2\ f_5\ 2^{\frac{2}{12}} + 3\ f_6\ . \tag{2.3}$$

It is easier to see here that we have a quantity we would like to be as close to zero as possible: the beat rate. We will refer to this as a *residual* in our calculations. If we calculate the beat rate that occurs between these frequencies using equal-temperament tuning, we find it to be non-zero:

$$residual = -2\ f_5\ 2^{\frac{2}{12}} + 3\ f_6,$$

$$= -2\ (110\ Hz)\ 2^{\frac{2}{12}} + 3\ (82.41\ Hz), \qquad (2.4)$$

$$= 10\ Hz.$$

However, if we only set *f6* to be constant and are allowed to adjust the frequency of *f5*, the beat rate can be set to zero for this interval.

But this is not the only interval in consideration—we will need to compromise the beat rate in this interval to allow for small beating across other intervals, as well.

Now consider the relationship between *f5* and *f4*:

$$residual = -3\ f_4\ 2^{\frac{2}{12}} + 4\ f_5\ 2^{\frac{2}{12}}\ . \qquad (2.5)$$

If we want to minimize the beat rates for both the *f4-f5* relation and the *f5-f6* relation, we must solve the system of equations representing their residuals, calling the residuals *r1* and *r2*:

$$\begin{cases} r_1 = -2\ f_5 2^{\frac{2}{12}} + 3\ f_6\ , \\ r_2 = -3\ f_4\ 2^{\frac{2}{12}} + 4\ f_5 * 2^{\frac{2}{12}}\,. \end{cases} \qquad (2.6)$$

Here we can easily set *r1* and *r2* equal to 0 to completely eliminate beats, implement our known value of *f6,* and solve the resulting system of equations:

$$\begin{cases} 0 = -2\ f_5 2^{\frac{2}{12}} + 3\ (82.41)\ , \\ 0 = -3\ f_4\ 2^{\frac{2}{12}} + 4\ f_5\ 2^{\frac{2}{12}}\,. \end{cases} \qquad (2.7)$$

The first equation immediately gives us

$$f_5 = \frac{3\ (82.41)}{2\ (2^{\frac{2}{12}})}, \qquad (2.8)$$

$$= 110.13\ Hz.$$

The value for *f5* can now be used in the second equation to solve for *f4*:

$$0 = -3\ f_4\ 2^{\frac{2}{12}} + 4\ f_5\ 2^{\frac{2}{12}},$$

$$f_4 = \frac{4\ (110.13)2^{\frac{2}{12}}}{3\ (2^{\frac{2}{12}})}\ , \tag{2.9}$$

$$= 146.84\ Hz\ .$$

These frequencies give us zero beats for *r₁* and *r₂*, but we cannot assume that all of the residuals will be able to be zero; there will have to be compromise in order to minimize beating across all the intervals involved. Furthermore, many residuals still remain to be calculated and we cannot guess what their residuals might be as we did in this case. Therefore, a more reasonable method of solving this system of equations can be used that does not require guessing residual values: a method that uses matrix manipulation known as minimization by least squares.

2.1.2 Using Least Squares Optimization for a Small Matrix

Let us re-solve the system of equations for f_4 and f_5 using minimization by least squares. Beginning exactly as before, we want to minimize the beat rates for both the *f4-f5* relation and the *f5-f6* relation, so we must solve the system of equations representing both their residuals:

$$\begin{cases} r_1 = -2\ f_5 2^{\frac{2}{12}} + 3\ f_6\ , \\ r_2 = -3\ f_4\ 2^{\frac{2}{12}} + 4\ f_5 * 2^{\frac{2}{12}}\ . \end{cases} \tag{2.10}$$

To begin, I can write this system of equations using matrices and vectors:

$$\begin{bmatrix} -2(2^{\frac{2}{12}}) & 0 \\ 4(2^{\frac{2}{12}}) & -3(2^{\frac{2}{12}}) \end{bmatrix} \begin{bmatrix} f_5 \\ f_4 \end{bmatrix} + \begin{bmatrix} 3f_6 \\ 0 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}. \tag{2.11}$$

We can represent this equation as

$$Af\ +\ b\ =\ r\ , \tag{2.12}$$

a common equation seen in dealing with matrices and vectors. With this elegant equation, we can work through the math without having to know the precise values for *A* and b. We can also state our objective simply: we need to minimize the vector of residuals, *r*. We do this by manipulating the matrix and minimizing r^2, which in terms of vectors is

$$r^T r. \tag{2.13}$$

We know that

$$r^T = f^T A^T \ + \ b^T \tag{2.14}$$

from the properties of transposed matrices, so $r^T r$ can be written as

$$r^T r = (f^T A^T + b^T)(Af + b), \tag{2.15}$$

$$= (f^T A^T A f + b^T A f + f^T A^T b + b^T b)\ .$$

On close observation, one can see that each of these terms is a scalar: $r^T r$ and $b^T b$ both consist of a $1x2$ and a $2x1$ matrix, which result in a $1x1$ matrix (a scalar) when multiplied through. We can generalize this statement by noting that a matrix's dimensions are given by its number of columns, *m*, and its number of rows, *n*. Thus we can represent a matrix's dimensions as mxn and the dimensions of its transpose by nxm. Analyzing the rest of equation 2.5 in a similar manner as we did $r^T r$, we demonstrate that each term is a scalar:

$$= (f^T A^T A f + b^T A f + f^T A^T b + b^T b), \tag{2.16}$$

$$= (1xm)(mxn)\ (nxm)(mx1) + (1xn)(nxm)(mx1) + (1xm)(mxn)(nx1) + (1xn)(nx1),$$

$$= ((1x1) + (1x1) + (1x1) + (1x1))\ .$$

Since $A^T f^T b$ is the transpose of $b^T A f$ and the transpose of a scalar is a scalar, we conclude that they are equal and can combine the terms:

$$r^T r = (A^T f^T A f + 2b^T A f + b^T b)\ . \tag{2.17}$$

Now that we have simplified $r^T r$ and know that it is a scalar, we can take the derivative with respect to *f* and set it to zero for minimization:

$$0 = 2A^T A f + 2A^T b\ . \tag{2.18}$$

Solving for the vector *f*—the frequencies which give a minimized beat rate—we get

$$f \ = -(A^T A)^{-1} A^T b\ . \tag{2.19}$$

This is the general solution for any set of chords we wish to optimize for consonance.

Here we return to our specific example and solve equation 2.19 using our values from our system of equations (Eq. 2.11). The matrices *A* and *b* are found in our first representation of our system using matrices (Eq. 2.11):

$$A = \begin{bmatrix} -2(2^{\frac{2}{12}}) & 0 \\ 4(2^{\frac{2}{12}}) & -3(2^{\frac{2}{12}}) \end{bmatrix}, \tag{2.20}$$

$$b = \begin{bmatrix} 3f_6 \\ 0 \end{bmatrix}. \tag{2.21}$$

The transverse of *A* is the reflection of *A* across a diagonal running from its top left to its bottom right:

$$A^T = \begin{bmatrix} -2(2^{\frac{2}{12}}) & 4(2^{\frac{2}{12}}) \\ 0 & -3(2^{\frac{2}{12}}) \end{bmatrix}. \tag{2.22}$$

Plugging these values into equation 2.19, we have an equation in need of simplification:

$$f = -\left(\begin{bmatrix} -2(2^{\frac{2}{12}}) & 4(2^{\frac{2}{12}}) \\ 0 & -3(2^{\frac{2}{12}}) \end{bmatrix}\begin{bmatrix} -2(2^{\frac{2}{12}}) & 0 \\ 4(2^{\frac{2}{12}}) & -3(2^{\frac{2}{12}}) \end{bmatrix}\right)^{-1}\begin{bmatrix} -2(2^{\frac{2}{12}}) & 4(2^{\frac{2}{12}}) \\ 0 & -3(2^{\frac{2}{12}}) \end{bmatrix}\begin{bmatrix} 3f_6 \\ 0 \end{bmatrix}. \tag{2.23}$$

Simplifying and factoring out the $2^{\frac{2}{12}}$ from each matrix gives

$$f = -\left(\begin{bmatrix} 20 & -12 \\ -12 & 9) \end{bmatrix} 2^{\frac{4}{12}}\right)^{-1} \begin{bmatrix} -2 & 4 \\ 0 & -3 \end{bmatrix} 2^{\frac{2}{12}} \begin{bmatrix} 3f_6 \\ 0 \end{bmatrix}. \tag{2.24}$$

We know that the inverse of a 2*x*2 matrix is given by the shortcut

$$(A)^{-1} = \frac{1}{ad-bc}\begin{bmatrix} d & -c \\ -b & a \end{bmatrix}, \tag{2.25}$$

which allows us to further simplify our equation:

$$f = -\left(\frac{1}{36(2^{\frac{8}{12}})}\begin{bmatrix} 9 & 12 \\ 12 & 20 \end{bmatrix} 2^{\frac{4}{12}}\right)\begin{bmatrix} -2 & 4 \\ 0 & -3 \end{bmatrix} 2^{\frac{2}{12}} \begin{bmatrix} 3f_6 \\ 0 \end{bmatrix}, \tag{2.26}$$

$$= -\left(\begin{bmatrix} 1/4 & 1/3 \\ 1/3 & 5/\,9 \end{bmatrix} 2^{\frac{-4}{12}}\right)\begin{bmatrix} -2 & 4 \\ 0 & -3 \end{bmatrix} 2^{\frac{2}{12}} \begin{bmatrix} 3f_6 \\ 0 \end{bmatrix},$$

$$= -\begin{bmatrix} -1/2 & 0 \\ -2/3 & -1/3 \end{bmatrix} 2^{\frac{-2}{12}} \begin{bmatrix} 3f_6 \\ 0 \end{bmatrix},$$

$$= -\begin{bmatrix} -1/2 & 0 \\ -2/3 & -1/3 \end{bmatrix} 2^{\frac{-2}{12}} \begin{bmatrix} 3f_6 \\ 0 \end{bmatrix}.$$

Plugging in our pre-set value for f_6, we get

$$f = \begin{bmatrix} f_5 \\ f_4 \end{bmatrix} = \begin{bmatrix} 110.13 \\ 146.84 \end{bmatrix}, \tag{2.27}$$

the same result as when we simplified our system of equations by setting the residuals to zero (Eq. 2.8 and 2.9). In this example we were able to force the residuals to be zero, but this won't generally be possible when optimizing all strings for multiple chords.

2.1.3 Using Least Squares Optimization for a Large Matrix

We will now walk through the same calculation for an entire chord. We represent the residuals between every relevant string using a large system of equations. If all six strings of the guitar are used in a chord, there will be as many as 15 residuals. For many chords, some of the equations turn out to be linearly dependent on other equations in the system and are therefore unnecessary for completing the calculation. However, the algorithm works even with the inclusion of redundant equations, so we will include them all. In the case of the E major chord, all six strings are used and the 15 residuals are represented as follows:

$$\begin{cases} r_1 = -2\, f_5 2^{\frac{2}{12}} + 3\, f_6\,, \\ r_2 = \ 2f_6 - f_4 2^{\frac{2}{12}}\,, \\ r_3 = 10f_6 - 4f_3 2^{\frac{1}{12}}\,, \\ r_4 = 6f_6 - 2f_2\ \,, \\ r_5 = 4f_6 - f_1\,, \\ r_6 = 4f_5 2^{\frac{2}{12}} - 3f_4 2^{\frac{2}{12}}\,, \\ r_7 = 5f_5 2^{\frac{2}{12}} - 3f_3 2^{\frac{1}{12}}\,, \\ r_8 = 2f_5 2^{\frac{2}{12}} - f_2\,, \\ r_9 = 8f_5 2^{\frac{2}{12}} - 3f_1\,, \\ r_{10} = 5f_4 2^{\frac{2}{12}} - 4f_3 2^{\frac{1}{12}}\,, \\ r_{11} = 3f_4 2^{\frac{2}{12}} - 2f_{2\,,} \\ r_{12} = 2f_4 2^{\frac{2}{12}} - f_1\,, \\ r_{13} = 6f_3 2^{\frac{1}{12}} - 5f_2\,, \\ r_{14} = 8f_3 2^{\frac{1}{12}} - 5f_1\,, \\ r_{15} = 4f_2 - 3f_1\,. \end{cases} \tag{2.28}$$

This large system of equations needs to be minimized in order to minimize the beat rates occurring between all strings in an E major chord. We can minimize the residuals in this large system of equations using the same matrix manipulation we used for solving our small system. To begin, I can write this system of equations using matrices and vectors:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & -2*2^{2/12} \\ 0 & 0 & 0 & -2^{2/12} & 0 \\ 0 & 0 & -4*2^{2/12} & 0 & 0 \\ 0 & -2 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -3*2^{2/12} & 4*2^{2/12} \\ 0 & 0 & -3*2^{1/12} & 0 & 5*2^{2/12} \\ 0 & -1 & 0 & 0 & 2*2^{2/12} \\ -3 & 0 & 0 & 0 & 8*2^{2/12} \\ 0 & 0 & -4*2^{1/12} & 5*2^{2/12} & 0 \\ 0 & -2 & 0 & 3*2^{2/12} & 0 \\ -1 & 0 & 0 & 2*2^{2/12} & 0 \\ 0 & -5 & 6*2^{1/12} & 0 & 0 \\ -5 & 0 & 8*2^{1/12} & 0 & 0 \\ -3 & 4 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{bmatrix} + \begin{bmatrix} 3f_6 \\ 2f_6 \\ 10f_6 \\ 6f_6 \\ 4f_6 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_6 \\ r_7 \\ r_8 \\ r_9 \\ r_{10} \\ r_{11} \\ r_{12} \\ r_{13} \\ r_{14} \\ r_{15} \end{bmatrix}. \quad (2.29)$$

Using the same reasoning as in equations 2.12 through 2.19, we know that the frequencies which give a minimized beat rate is given by (Eq. 2.19):

$$f = -(A^T A)^{-1} A^T b\ . \quad (2.30)$$

Plugging in our values from our equation for E major (Eq. 2.29) and simplifying equation 2.30 will give the solution for vector *f*. The result is as follows:

$$f = \begin{bmatrix} f_5 \\ f_4 \\ f_3 \\ f_2 \\ f_1 \end{bmatrix} = \begin{bmatrix} 110.12 \\ 146.83 \\ 194.45 \\ 247.22 \\ 329.63 \end{bmatrix}. \quad (2.31)$$

Each of the frequencies that were solved for are given in Hz and are the frequencies that will give the best consonance possible in the E major chord on the guitar. Just as in our simpler example, these frequencies cause the value of all the residuals in the system of equations to be zero (discussed in Section 3.1.1). This typically won't be the case when optimizing consonance for multiple chords. However, the calculation will provide frequencies that minimize beating as much as physically possible across the involved chords (see Chapter 3 for specific examples).

For some perspective on the magnitude of frequency changes that optimization causes, see Figure 10 for a comparison of the frequencies of each string before and after optimization. The frequency adjustment for each of the five adjusted strings, on average, is only .258 Hz.

Tuning Scheme	f_6	f_5	f_4	f_3	f_2	f_1
Equal Temperament Frequency	82.41	110.00	146.83	196.0	246.94	329.63
Optimized Frequency	82.41	110.12	146.83	194.45	247.22	329.63

Figure 10. String frequencies before and after optimization of the E major chord.

When we calculate an optimized tuning scheme for more than one chord, we consider the beat rates across all the strings from *both* chords. Even though the system of equations grows much larger for each additional chord (at most 15 equations) included in the conditions for optimization, the most complicated part of the calculation—the inversion of a matrix—will always involve a 5*x*5 matrix:

$$\begin{aligned} A^T A &= (5xn)(nx5), \\ &= 5x5. \end{aligned} \tag{2.32}$$

Inverting the matrix can be done (rather tediously) by using the Gauss-Jordan method or the Adjunct method, but a computer program has no problem solving it in a very short amount of time.

2.2 The Android App

The app that will be used to create custom guitar tunings provides a straightforward user interface for a clean user experience. A custom multiple-choice ListView displays each chord and its voicing. Guitarists can choose specific voicings of chords that they will be playing based on the diagram next to each chord entry in the list; the inclusion of chord diagrams in the ListView is the result of the CustomListAdapter.java file. A button at the bottom of the screen is always visible and can be selected at any time to execute the least squares calculation involving the selected chords. A popup dialog displays once the calculation is complete, listing the frequency of each string that will provide the optimal intonation for the chords selected. When the dialog popup is acknowledged and exited, all chord selections are cleared in preparation to receive a new selection.

The entire calculation for optimization of consonance is carried out in a file called *Optimization.java*, written to follow the algorithm described in the Chapter 2 of this paper. Noteworthy specifics of my code in *Optimization.java* include its features that allow for versatile inclusion of new chords. New chords of any shape and voicing can easily be added to my code. The chord must be given a unique name and must be represented as a single array of integers at the beginning of the class "optimization". Each integer in the chord array represents the voicing of strings f_6 through f_1: the integer indicates the number of the fret that is pressed on each string to voice the chord. A "0" represents no fret pressed, a positive number represents the pressed fret's number, and a "-1" represents an unvoiced string. For example, the open E major chord (seen in Figure 9) is represented in the code as follows:

```
private static final Integer[] EmajorFingering1 = new Integer[]{0,2,2,1,0,0};  .
```

The new chord must also be added into the variable *String[] chords* of MainActivity.java in order to be displayed in the interface's multiple-choice ListView. A corresponding image of the chord must be added to the "drawable" folder under the same name as was entered into *chords* to allow selection of the correct chord voicing by the user.

The app uses standard Android and Java libraries to provide the more general features of the app. For the calculation, the Jama Matrix library (MathWorks & NIST, 2012) is used to aid in several matrix operations found in *Optimization.java* including transposition, multiplication, and matrix inversion. The calculation is executed on action of the "Calculate" button on the Android interface. The optimized frequencies are returned to the Android class, which then displays them to the user.

With the simplicity and versatility of the code, any new chord can be easily implemented into the program. The calculation is designed to dynamically calculate the residuals between any strings based on the integer array representing a chord, so adding chords does not require any manual calculations. The user interface is also very simple, ensuring that the general public will be able to understand and use this tool.

Chapter 3. Results and Discussion

3.1 Quantitative Analysis of Results

In verification of consonance improvement when using these calculations, we performed both quantitative and qualitative analysis. The result of minimizing the residuals in the E major chord gave an overall 100% decrease in beating within the chord from equal temperament tuning. Figure 11 displays a full comparison of beat rates between the strings before and after optimization.

3.1.1 E Major Optimization

String Pair	Equal Temperament Beating (Hz)	Minimized Beating (Hz)
f_6, f_5	0.2790	0
f_6, f_4	0.0005	0
f_6, f_3	6.5501	0
f_6, f_2	0.5574	0
f_6, f_1	0.0004	0
f_5, f_4	0.5567	0
f_5, f_3	5.6102	0
f_5, f_2	0.0003	0
f_5, f_1	1.1174	0
f_4, f_3	6.5523	0
f_4, f_2	0.5560	0
f_4, f_1	0.0013	0
f_3, f_2	11.2186	0
f_3, f_1	13.0981	0
f_2, f_1	1.1160	0
Average Beating within E Major	3.1475	0

Figure 11. Beat rates before and after optimization.

As more chords are added, the amount of compromise necessary to optimize consonance approaches the amount of compromise in the equal temperament scale so there is less reduction of beating. Results for other chords and combinations of chords are outlined in the tables that follow.

3.1.2 E Major and A Major Optimization

String Pair	Equal Temperament Beating (Hz)	Minimized Beating (Hz)
E Major		
f_6, f_5	0.2790	0.1334
f_6, f_4	0.0005	0.0315
f_6, f_3	6.5501	1.2930
f_6, f_2	0.5574	3.1254
f_6, f_1	0.0004	0.3078
f_5, f_4	0.5567	0.1724
f_5, f_3	5.6102	0.6363
f_5, f_2	0.0003	1.6961
f_5, f_1	1.1174	0.3897
f_4, f_3	6.5523	1.1357
f_4, f_2	0.5560	3.2197
f_4, f_1	0.0013	0.2449
f_3, f_2	11.2186	9.7530
f_3, f_1	13.0981	1.0472
f_2, f_1	1.1160	7.1740
A Major		
f_6, f_5	0.3724	0.9236
f_6, f_4	0.00045	0.0315
f_6, f_3	0.7525	3.4248
f_6, f_2	7.4801	3.1564
f_6, f_1	0.0004	0.3078
f_5, f_4	0.3733	0.8607
f_5, f_3	0.0026	1.7573
f_5, f_2	8.7321	1.1300
f_5, f_1	0.7440	1.2316
f_4, f_3	0.7543	3.5506
f_4, f_2	7.4823	2.9992
f_4, f_1	0.0013	0.2449
f_3, f_2	8.7193	9.9165
f_3, f_1	0.7517	4.0403
f_2, f_1	14.9581	4.7740
Average Beating within Chords	3.2779	2.2739

Figure 12. Beats before and after optimization for the use of E major and A major.

The average beating within the chords E major and A major has been reduced by 36% through optimization, demonstrated in Figure 12.

3.1.3 E Major, A Major, and G Major Optimization

String Pair	Equal Temperament Beating (Hz)	Minimized Beating (Hz)
E Major		
f_6, f_5	0.2790	0.1334
f_6, f_4	0.0005	0.0315
f_6, f_3	6.5501	1.2930
f_6, f_2	0.5574	3.1234
f_6, f_1	0.0004	0.3078
f_5, f_4	0.5567	0.1724
f_5, f_3	5.6102	0.6363
f_5, f_2	0.0003	1.6951
f_5, f_1	1.1174	0.3897
f_4, f_3	6.5523	1.1358
f_4, f_2	0.5560	3.2177
f_4, f_1	0.0013	0.2449
f_3, f_2	11.2186	9.7480
f_3, f_1	13.0981	1.0472
f_2, f_1	1.1160	7.1740
A Major		
f_6, f_5	0.3724	0.9236
f_6, f_4	0.00045	0.0315
f_6, f_3	0.7525	3.4248
f_6, f_2	7.4801	3.1598
f_6, f_1	0.0004	0.3078
f_5, f_4	0.3733	0.8607
f_5, f_3	0.0026	1.7573
f_5, f_2	8.7321	1.1345
f_5, f_1	0.7440	1.2316
f_4, f_3	0.7543	3.5506
f_4, f_2	7.4823	3.0025
f_4, f_1	0.0013	0.2449
f_3, f_2	8.7193	9.9209
f_3, f_1	0.7517	4.0403
f_2, f_1	14.9581	4.7807
G Major		
f_6, f_5	3.8889	4.7138
f_6, f_4	0.3326	0.2758
f_6, f_3	0.0023	1.2382

f_6, f_2	7.7793	2.6474
f_6, f_1	0.0005	0.3660
f_5, f_4	6.6650	7.7602
f_5, f_3	7.7666	15.6189
f_5, f_2	0.0003	1.6951
f_5, f_1	15.5534	17.0253
f_4, f_3	0.6720	3.1632
f_4, f_2	6.6660	2.6749
f_4, f_1	1.3319	2.2011
f_3, f_2	7.7680	8.8386
f_3, f_1	0.0040	2.8425
f_2, f_1	15.5562	3.4647
Average Beating within Chords	3.7998	3.1234

Figure 13. Beats before and after optimization for the use of E major, A major, and G major.

The average beating within the chords E major, A major, and G major has been reduced by 20% through optimization, demonstrated in Figure 13.

3.2 Qualitative Analysis of Results

Qualitatively, the improvement was judged both on an actual guitar and on MIDI guitar software. In my opinion, the actual guitar did perform with improved consonance when tuned to the optimized frequencies. The MIDI Guitar software, developed by Dr. Dallin Durfee for the purpose of evaluating tuning schemes on guitar, takes a frequency input for any string and will then simulate chords played on the guitar. The software did provide a more reliable source for qualitative judgment than an actual guitar since the frequencies being heard were guaranteed to be perfectly in tune with the calculations. Overall, the software did demonstrate an improvement in consonance from equal temperament tuning when implementing the optimized frequencies.

These results indicate an improvement in intonation on a guitar for any custom set of chords. The consistent improvement across chord combinations proves that this method of tuning is versatile and has the potential to be used to improve the consonance of any song. The simple app and its availability online makes it accessible and usable to musicians—both hobbyists and professionals.

Chapter 4. Conclusions

4.1 General Conclusions

This project provides the first optimized, custom tuning based off of precise calculations for guitarists. This innovation is significant not only due to its precise calculations and versatility in providing custom tuning schemes, but also because of its accessibility and applicability. Anyone can download the app on their Android phone. A custom tuning can be generated in as little time as it takes to enter the chords being used. The custom tuning can be implemented in the short time it takes to tune the strings. In short, this app provides the simplest and most accurate custom tuning tool for guitarists in existence as of yet and has the potential to be a significant contribution to the musical experience of the general public.

In summary, this project has resulted in an elegant java program that effectively optimizes intonation and provides a custom tuning scheme to guitarists. Intonation can be effectively improved by deviating from equal temperament intonation, as seen by analyzing the beat rates existent in each tuning scheme. The project is packaged into a simple app. It provides a convenient and easily-implemented solution for guitarists searching for greater consonance in their music.

4.2 Suggestions for Future Research

There remain areas of intonation that can be researched to improve the quality of the calculations. This application gives equal weight to all intervals, so beating is overall minimized. However, if it is found that particular intervals are more important to the human ear in achieving consonance, giving greater weight to those intervals during minimization would improve the results.

This type of information has the greatest potential to be gathered from surveys, as it is the opinions and perceptions of potential users that will be of greatest importance in this matter. Various tuning schemes could potentially be generated by using different weights for particular intervals and played for a subject. Subjects can then vote on the tuning scheme they find most pleasing. When enough data is collected, the most important intervals for intonation—if any—can be determined.

Areas for improvement and further research remain concerning the calculations in this project. Calculating and accounting for dispersion in guitar strings and accounting for error in fret placement by measuring pitch changes could potentially improve the effectiveness of the algorithm in optimizing consonance. Further development of the Android app could result in an implemented tuner that analyzes a guitar string frequency and indicates if it is flat or sharp in comparison with the optimized frequency, making it a more approachable tool for musicians.

Development of the project to this point has resulted in a working algorithm that has been written to calculate the frequencies for guitar strings that will optimize consonance. Its effectiveness has been proven by the results outlined in Chapter 3 of this paper. Finally, the culmination of the research: the algorithm has been successfully integrated into software that is running reliably in the form of an Android app. It is anticipated that the Android app will be made available to the general public and will provide guitarists with greater consonance in their music.

References

Durfee, D. S. (2013). *Temperament Studio 1.8.0 - Demonstrating Intonation with MIDI*. Department of Physics and Astronomy: Brigham Young University. Retreived from http://www.physics.byu.edu/faculty/durfee/TemperamentStudio/.

Durfee, D. S., & Colton, J. S. (2012). The Physics of Musical Scales: Theory and Experiment. Department of Physics and Astronomy. Brigham Young University. (preprint).

Flatley, C. (2007). *Using Musical Intervals to Greatly Improve Tuning and Intonation*. Retrieved from http://www.ultimate-guitar.com/columns/the_guide_to/using_musical_intervals_to_greatly_improve_tuning_and_intonation.html.

Gold, J. (2007). *Fender VG Stratocaster*. Retrieved from http://www.guitarplayer.com/miscellaneous/1139/fender-vg-stratocaster/15254.

Hanson, M. (1995). *The Complete Guitar Player Series: The Complete Book of Alternate Tunings*. Accent On Music, LLC.

Locke, J. (n.d.). Intonation and Tuning of the Classical Guitar. *Guitarra Magazine*. Retreived from http://www.guitarramagazine.com/Intonation.

MathWorks, & NIST. (2012). National Institute of Standards and Technology. JAMA: A Java Matrix Package. Accessed from http://math.nist.gov/javanumerics/jama/.

McMurtney, R. J., Fleming, D., & Steffensen, S. (2013). Effects of Harmony and Dissonance with Two-Tone Narrow and Wide Range Frequencies on Auditory Evoked Potentials. *Journal of Undergraduate Research*. Department of Psychology and Neuroscience. Brigham Young University. Retrieved from http://jur.byu.edu/?p=7763.

Monzo, J. (2005). Tonalsoft. Tonalsoft Inc. Retrieved from http://tonalsoft.com/enc/encyclopedia.aspx.

Morin, D. (Drafted 2009). Dispersion, Chapter 6. Cambridge, MA: Harvard University. Retrieved from http://www.people.fas.harvard.edu/~djmorin/waves/dispersion.pdf. (preprint).

Sethares, B. (2009). Alternate Tunings Guide. University of Wisconsin. Department of Electrical Engineering. Retrieved from http://sethares.engr.wisc.edu/alternatetunings/alltunings.pdf.

SetitupBetter. (n.d.). Understanding Guitar Setup. Retrieved from http://www.setitupbetter.com/.

Spier, N. (2013). Spectratune. Northampton, MA: Retrieved from http://www.nastechservices.com/Spectratune.html.

Appendix A: Android App Renderings

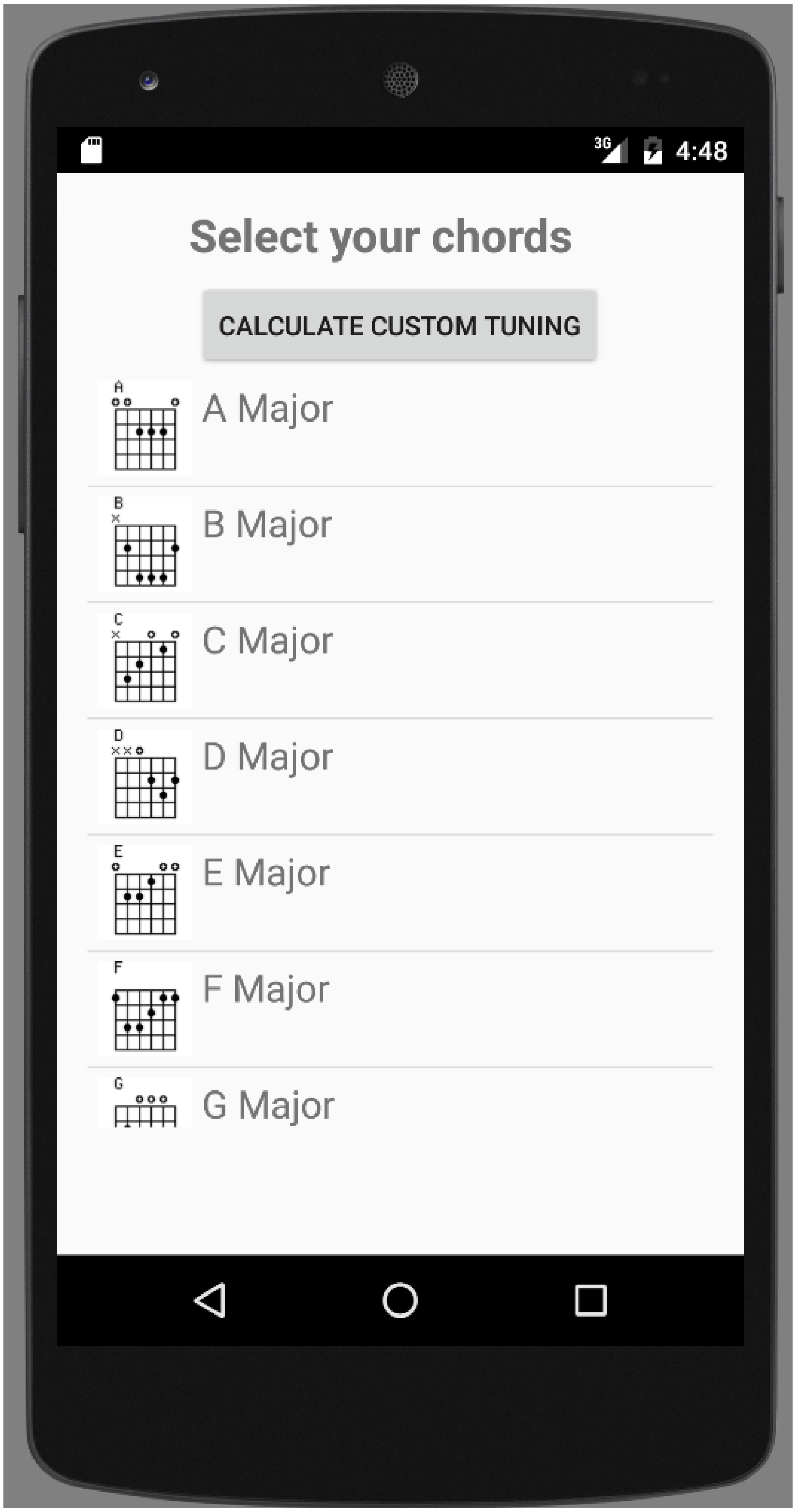


Figure 14. Android app home page rendering. The screen is displaying the list of available chords to choose from.

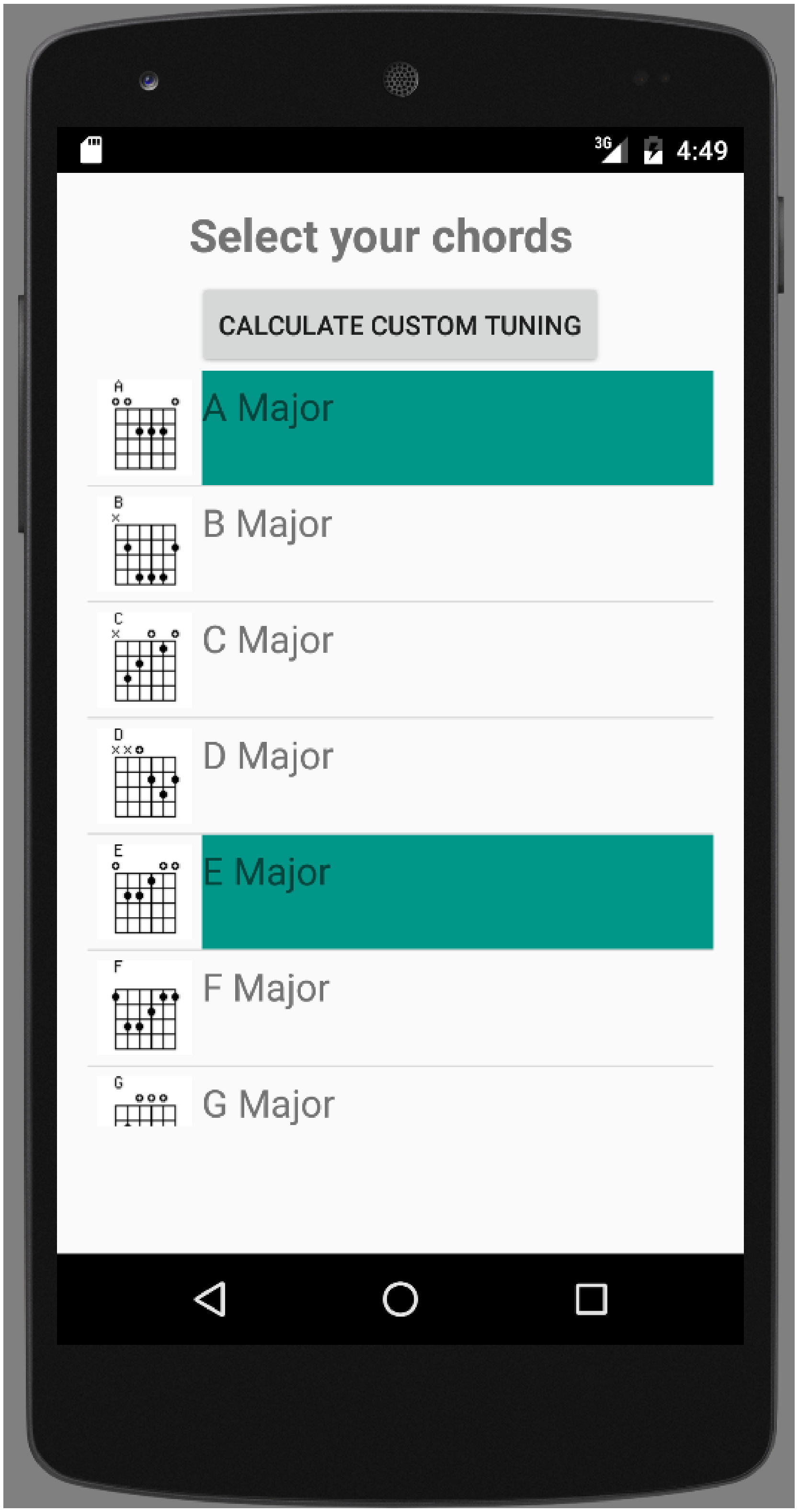
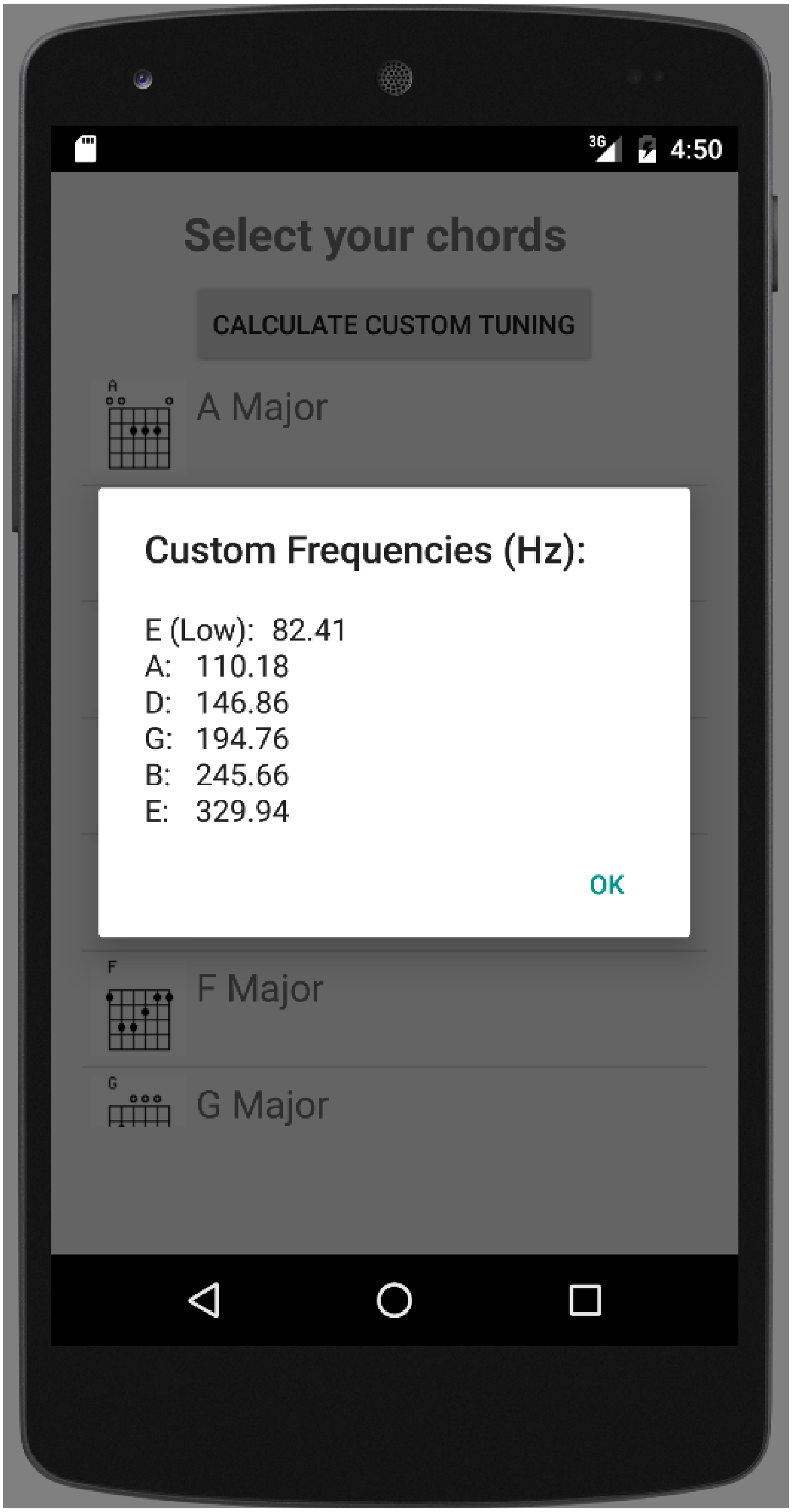


Figure 15. Chord selection within the Android App. "A Major" and "E Major" are selected here in the list.



Custom Frequencies (Hz):

```
E (Low):  82.41
A:   110.18
D:   146.86
G:   194.76
B:   245.66
E:   329.94
```

Figure 16. The calculated frequencies of each string. The "Calculate Custom Tuning" button was selected and the frequencies for each string that will optimize consonance are displayed as a popup dialog.

Appendix B: Android App Code

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.clee.listviewdemo" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">


    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical">

        <TextView android:text="Select your chords"
            android:layout_width="221dp"
            android:layout_height="40dp"
            android:layout_gravity="center_horizontal"
            android:id="@+id/select_chords"
            android:textSize="24dp"
            android:textStyle="bold" />
        <Button
            android:id="@+id/button1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:onClick="calculateFrequencies"
            android:text="Calculate Custom Tuning" />

        <ListView
            android:id="@+id/android:list"
            android:layout_width="match_parent"
            android:layout_height="397dp"
            android:choiceMode="multipleChoice">

        </ListView>
    </LinearLayout>
</RelativeLayout>
```

MainActivity.java

```
package com.example.clee.listviewdemo;

import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.util.Log;
import android.util.SparseBooleanArray;
import android.view.Menu;
import android.view.MenuItem;
import android.app.ListActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Set;

public class MainActivity extends ListActivity {
    private String [] chords = {
            "A Major",
            "B Major",
            "C Major",
            "D Major",
            "E Major",
            "F Major",
            "G Major",
    };
    Integer[] imageId = {
            R.drawable.a_major_1,
            R.drawable.b_major_1,
            R.drawable.c_major_1,
            R.drawable.d_major_1,
            R.drawable.e_major_1,
            R.drawable.f_major_1,
            R.drawable.g_major_1
    };

    //Put items into this array list when selected
    ArrayList selectedItems = new ArrayList();
    Set selectedListItems = new HashSet();

    @Override protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        CustomListAdapter customAdapter = new CustomListAdapter(this, chords,
imageId);
        ListView listView = getListView();
        listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE);
        setListAdapter(customAdapter);
    }

    @Override protected void onListItemClick(ListView l, View v, int position, long
id){
        ListView listView = getListView();
        for(int i = 0; i<listView.getCount(); i++) {
```

CustomListAdapter.java

```java
package com.example.clee.listviewdemo;

import android.app.Activity;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

public class CustomListAdapter extends ArrayAdapter<String> {

    private final Activity context;
    private final String[] itemname;
    private final Integer[] imgid;

    public CustomListAdapter(Activity context, String[] itemname, Integer[] imgid) {
        super(context, R.layout.my_list_images, itemname);

        this.context=context;
        this.itemname=itemname;
        this.imgid=imgid;
    }

    public View getView(int position,View view,ViewGroup parent) {
        LayoutInflater inflater=context.getLayoutInflater();
        View rowView=inflater.inflate(R.layout.my_list_images, null, true);

        TextView txtTitle = (TextView) rowView.findViewById(R.id.item);
        ImageView imageView = (ImageView) rowView.findViewById(R.id.icon);

        txtTitle.setText(itemname[position]);
        imageView.setImageResource(imgid[position]);
        return rowView;

    };
}
```

my_list_images.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal">
    <ImageView
        android:id="@+id/icon"
        android:layout_width="50dp"
        android:layout_height="50dp"
        android:layout_marginBottom="5dp"
        android:layout_marginLeft="5dp"
        android:layout_marginRight="5dp"
        android:layout_marginTop="5dp" />
    <TextView
        android:id="@+id/item"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:textSize="20sp"
        android:background="?android:attr/activatedBackgroundIndicator"
        android:paddingTop="5dp"/>
</LinearLayout>
```

Optimization.java

```
package com.example.clee.listviewdemo;

/**
 * Created by cassi lee on 4/17/2015.
 */

import android.util.Log;

import java.util.ArrayList;
import java.util.List;

import Jama.Matrix;

public class Optimization{
    private
    static final double F6 = 82.406899;
    static double f5;
    static double f4;
    static double f3;
    static double f2;
    static double f1;
    static int m;
    static int n;
    static final int[] stringHalfSteps = {0,5,10,15,19,24};//{f6, f5, f4, f3, f2 f1}
where f6 is the lowest string
    static final int[] defaultStringHalfSteps = {0,5,10,15,19,24};
    static final double justIntervalFactors[][] = {{1.0, 1.0},
            {16.0, 15.0},
            {9.0, 8.0},
            {6.0, 5.0},
            {5.0, 4.0},
            {4.0, 3.0},
            {45.0, 32.0},
            {3.0, 2.0},
            {8.0, 5.0},
            {5.0, 3.0},
            {9.0, 5.0},
            {15.0, 8.0},};
    static ArrayList<Double> customChordBList = new ArrayList<Double>();
    static List<double[]> customChordArrayList = new ArrayList<double[]>();
    static double[] residualLine = new double[]{0.0, 0.0, 0.0, 0.0, 0.0};
    static ArrayList<double[]> matrixOfResiduals = new ArrayList<double[]>();
    static double[] frequenciesToDisplay = new double[6];

    static final Integer[] AmajorFingering1 = new Integer[]{-1,0,2,2,2,0};
    static final Integer[] BmajorFingering1 = new Integer[]{-1,2,4,4,4,2};
    static final Integer[] CmajorFingering1 = new Integer[]{-1,3,2,0,1,0};
    static final Integer[] DmajorFingering1 = new Integer[]{-1,-1,0,2,3,2};
    static final Integer[] EmajorFingering1 = new Integer[]{0,2,2,1,0,0};
    static final Integer[] FmajorFingering1 = new Integer[]{1,3,3,2,1,1};
    static final Integer[] GmajorFingering1 = new Integer[]{3,2,0,0,0,3};

public static double[] getCustomFrequencies(ArrayList<String[]> CustomChords) throws
Exception {
        matrixOfResiduals.clear();
        customChordArrayList.clear();
        customChordBList.clear();
        matrixOfResiduals.clear();

        String[] customChordsArray = CustomChords.toArray(new
String[CustomChords.size()]);

        for(int i =0; i < CustomChords.size(); i++) {
            Log.i("customChordsArray["+i+"]: ", customChordsArray[i]);
            System.out.println("customChordsArray[" + i + "]: "+
customChordsArray[i]);
        }

        //calculate the residuals for each relationship between the strings based on
the fingering of the chord. add each residual calculated to the matrix
        for(int i = 0; i < CustomChords.size(); i++){
            if(customChordsArray[i] == "A Major"){
                calculateMatrix(AmajorFingering1);
            }
            else if(customChordsArray[i] == "B Major"){
                calculateMatrix(BmajorFingering1);
            }
            else if(customChordsArray[i] == "C Major"){
                calculateMatrix(CmajorFingering1);
            }
            else if(customChordsArray[i] == "D Major"){
                calculateMatrix(DmajorFingering1);
            }
            else if(customChordsArray[i] == "E Major"){
                calculateMatrix(EmajorFingering1);
            }
            else if(customChordsArray[i] == "F Major"){
                calculateMatrix(FmajorFingering1);
            }
            else if(customChordsArray[i] == "G Major") {
                calculateMatrix(GmajorFingering1);
            }
        }

/****************Create the desired matrix by putting all the chords together chosen
by user in a huge matrix**************/
        double[][] customChordArray = new double[matrixOfResiduals.size()][];
        for(int i = 0; i < matrixOfResiduals.size(); i++) {
            customChordArray[i] = matrixOfResiduals.get(i);
        }
        Double[] customChordB = customChordBList.toArray(new
Double[customChordBList.size()]);

        double[] customChordB2 = new double[customChordB.length];
        for(int i = 0; i < customChordB.length; i++){//change the customChordArray to
match the type "double" that DenseMatrix requires
            customChordB2[i] = customChordB[i];
        }

/*************************Solve using the jama library*****************************/
        Matrix A = new Matrix(customChordArray/*numberChordsRows*/);//set matrix a
        Matrix b = new Matrix(customChordB2, customChordB2.length);//set matrix b
        Matrix frequenciesSolution = new Matrix(1, 5);
        frequenciesSolution =
((((A.transpose()).times(A)).inverse()).times((A.transpose()).times(b))).times(-1.0);


        double[][] frequenciesForArray = frequenciesSolution.getArray();
        frequenciesToDisplay[0] = F6;
        for(int i = 0; i < 5; i++){
            frequenciesToDisplay[i+1] = frequenciesForArray[i][0];
        }

   return frequenciesToDisplay;
 }
```

```
    private static List<double[]> calculateMatrix(Integer[] fingering){
        int lowNote = 0;
        int lowString = 0;
        int intervalRelation = 0;
        int octaveFactor = 0;//adds factors of 2
        int justIntervalRelation = 0;//how many absolute half steps are between the
notes if they were in the same octave
        //for each fingering, go through and calculate all the relationships between
all of the strings
        for(int i = 0; i < 6; i++){
            if(fingering[i] >= 0){
                lowString = i;
                lowNote = fingering[i];
                break; //ignore the -1's
            }
        }
        for(int h = lowString; h<5; h++){
            if(h == 0){//start on residual 1
                for(int i =1; i < 6; i++){//iterate through the strings to get each
relationship with F6, starting with f5 (i = 1)
                    intervalRelation = (stringHalfSteps[i]+fingering[i]) -
(fingering[0] + stringHalfSteps[0]);//relationship between strings f5-f1 with
f6(stringHalfSteps[0]) takes into consideration equal-temperament tuning
                    octaveFactor = (intervalRelation/12)*2;//get the integer rep
first, then multiply by 2 to get number of octaves between notes.
                    if(octaveFactor == 0){
                        octaveFactor = 1;
                    }
                    justIntervalRelation =
intervalRelation%12;//justIntervalFactors[intervalRelation];//defines the just
interval between the two strings

                    //calculate what factor needs to be put into the residualLine
                    residualLine[i-1]=-
justIntervalFactors[justIntervalRelation][1]*Math.pow(2.0,
fingering[i]/12.0);//(residual for matrix starts with f5)=(interval between
strings)*(equal temperament factor from fingering to solve the open string frequency);
                    Double f6Residual =
octaveFactor*justIntervalFactors[justIntervalRelation][0]*1.0*F6*Math.pow(2.0,
fingering[0]/12.0);
                    customChordBList.add(f6Residual);
                    //add this residualLine[] to the matrix of residual arrays
                    matrixOfResiduals.add(residualLine.clone());

                    //change residualLine back to all zeros for next time
                    for(int j = 0; j<5; j++){
                        residualLine[j] = 0.0;
                    }
                }
            }
            if(h == 1){
                for(int i =2; i < 6; i++){//iterate through the strings to get each
relationship with f5, starting with f4 (i = 2)
                    intervalRelation = (stringHalfSteps[i]+fingering[i]) -
(fingering[1] + stringHalfSteps[1]);//relationship between strings f5-f1 with
f6(stringHalfSteps[0]) takes into consideration equal-temperament tuning
                    octaveFactor = (intervalRelation/12)*2;//get the integer rep
first, then multiply by 2 to get number of octaves between notes.
                    if(octaveFactor == 0){
                        octaveFactor = 1;
                    }
                    justIntervalRelation =
intervalRelation%12;//justIntervalFactors[intervalRelation];//defines the just
```

```
interval between the two strings
                    residualLine[i-1]=-
justIntervalFactors[justIntervalRelation][1]*Math.pow(2.0,
fingering[i]/12.0);//(residual for matrix starts with f5)=(interval between
strings)*(equal temperament factor from fingering to solve the open string frequency);

residualLine[0]=octaveFactor*justIntervalFactors[justIntervalRelation][0]*1.0*Math.pow
(2.0, fingering[1]/12.0);//set f5's residual
                    matrixOfResiduals.add(residualLine.clone());//add this
residualLine[] to the matrix of residual arrays
                    for(int j = 0; j<5; j++){
                        residualLine[j] = 0.0;
                    }
                }
                double[] f5Residuals = {0.0, 0.0, 0.0, 0.0};
                for(int i = 0; i < f5Residuals.length; i++){
                    customChordBList.add(f5Residuals[i]);
                }
            }
            if(h == 2){
                for(int i =3; i < 6; i++){//iterate through the strings to get each
relationship with f4, starting with f3 (i = 3)
                    intervalRelation = (stringHalfSteps[i]+fingering[i]) -
(fingering[2] + stringHalfSteps[2]);//relationship between strings f5-f1 with
f6(stringHalfSteps[0]) takes into consideration equal-temperament tuning
                    octaveFactor = (intervalRelation/12)*2;//get the integer rep
first, then multiply by 2 to get number of octaves between notes.
                    if(octaveFactor == 0){
                        octaveFactor = 1;
                    }
                    justIntervalRelation =
intervalRelation%12;//justIntervalFactors[intervalRelation];//defines the just
interval between the two strings
                    residualLine[i-1]=-
justIntervalFactors[justIntervalRelation][1]*Math.pow(2.0,
fingering[i]/12.0);//(residual for matrix starts with f5)=(interval between
strings)*(equal temperament factor from fingering to solve the open string frequency);

residualLine[1]=octaveFactor*justIntervalFactors[justIntervalRelation][0]*1.0*Math.pow
(2.0, fingering[2]/12.0);//set f4's residual
                    matrixOfResiduals.add(residualLine.clone());//add this
residualLine[] to the matrix of residual arrays
                    for(int j = 0; j<5; j++){
                        residualLine[j] = 0.0;
                    }
                }
                double[] f4Residuals = {0.0, 0.0, 0.0};
                for(int i = 0; i < f4Residuals.length; i++){
                    customChordBList.add(f4Residuals[i]);
                }
            }
            if(h==3){
                for(int i =4; i < 6; i++){//iterate through the strings to get each
relationship with f3, starting with f2 (i = 4)
                    intervalRelation = (stringHalfSteps[i]+fingering[i]) -
(fingering[3] + stringHalfSteps[3]);//relationship between strings f5-f1 with
f6(stringHalfSteps[0]) takes into consideration equal-temperament tuning
                    octaveFactor = (intervalRelation/12)*2;//get the integer rep
first, then multiply by 2 to get number of octaves between notes.
                    if(octaveFactor == 0){
                        octaveFactor = 1;
                    }
                    justIntervalRelation =
intervalRelation%12;//justIntervalFactors[intervalRelation];//defines the just 
interval between the two strings
                    residualLine[i-1]=-
justIntervalFactors[justIntervalRelation][1]*Math.pow(2.0, 
fingering[i]/12.0);//(residual for matrix starts with f5)=(interval between 
strings)*(equal temperament factor from fingering to solve the open string frequency);

residualLine[2]=octaveFactor*justIntervalFactors[justIntervalRelation][0]*1.0*Math.pow
(2.0, fingering[3]/12.0);//set f4's residual
                    matrixOfResiduals.add(residualLine.clone());//add this 
residualLine[] to the matrix of residual arrays
                    for(int j = 0; j<5; j++){
                        residualLine[j] = 0.0;
                    }
                }
                double[] f3Residuals = {0.0, 0.0};
                for(int i = 0; i < f3Residuals.length; i++){
                    customChordBList.add(f3Residuals[i]);
                }
            }
            if(h==4){
                for(int i =5; i < 6; i++){//iterate through the strings to get each 
relationship with f2, starting with f1 (i = 5)
                    intervalRelation = (stringHalfSteps[i]+fingering[i]) - 
(fingering[4] + stringHalfSteps[4]);//relationship between strings f5-f1 with 
f6(stringHalfSteps[0]) takes into consideration equal-temperament tuning
                    octaveFactor = (intervalRelation/12)*2;//get the integer rep 
first, then multiply by 2 to get number of octaves between notes.
                    if(octaveFactor == 0){
                        octaveFactor = 1;
                    }
                    justIntervalRelation = 
intervalRelation%12;//justIntervalFactors[intervalRelation];//defines the just 
interval between the two strings
                    residualLine[i-1]=-
justIntervalFactors[justIntervalRelation][1]*Math.pow(2.0, 
fingering[i]/12.0);//(residual for matrix starts with f5)=(interval between 
strings)*(equal temperament factor from fingering to solve the open string frequency);

residualLine[3]=octaveFactor*justIntervalFactors[justIntervalRelation][0]*1.0*Math.pow
(2.0, fingering[4]/12.0);//set f4's residual
                    matrixOfResiduals.add(residualLine.clone());//add this 
residualLine[] to the matrix of residual arrays
                    for(int j = 0; j<5; j++){
                        residualLine[j] = 0.0;
                    }
                }
                double[] f2Residuals = {0.0};
                for(int i = 0; i < f2Residuals.length; i++){
                    customChordBList.add(f2Residuals[i]);
                }
            }
        }

        return matrixOfResiduals;
    }
}
```