June 2014

Listening to Optical Spectra

Luke Cullen Goodman
Worcester Polytechnic Institute

Listening to Optical Spectra

Luke Goodman

May 2014

A Major Qualifying Project Report:
submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science
by

Luke Goodman

Date: May 2014

Approved:

Professor Richard Quimby, Advisor

Contents

List of Figures

Abstract

Industrial mass production of goods requires individual inspection of each produced item to ensure adherence to quality standards. The processes of inspection can vary from fully automated systems to hiring workers to manually inspect each item. While these approaches are entirely valid, a hybrid system could be designed to capitalize upon the strengths of each. The process of evolution has optimized human senses to interact with their environment, while technology has provided increasingly powerful tools for observation. This project developed a system by which the reflected light off of an object can be used to create an aural representation of the properties of the object. By listening to the created sounds, discrepancies between the expected and observed reflectance are discerned as intuitively noticeable dissonance. The system developed thus allows observation of subquality items with minimal dedicated attention.

1. Introduction

The goal of this project is to convert an optical spectrum into an audio signal. The core motivation of this transformation is the appropriate utilization of basic human senses. Visual processing, while adept at many things, is inferior to aural processing at recognizing minor differences. While a graphical or other visual representation of spectral information may elucidate a significant amount of information, an audio representation of the same information could provide a faster, more intuitive understanding of the composite nature of a given signal.

Through the naturally self-optimizing course of evolution, life has developed keen abilities governing interaction with and observation of its surroundings. These senses are individually capable of assessing specific aspects of the environment and cooperate to provide a comprehensive assessment. In humanity, the dominant sense is visual. As a result, reliance upon optically conveyed information is often assumed to be the optimal method of intuitive observation; for comparisons such as that of *Figure 1.1*, visual appraisal can yield significant information about the circumstances being observed.

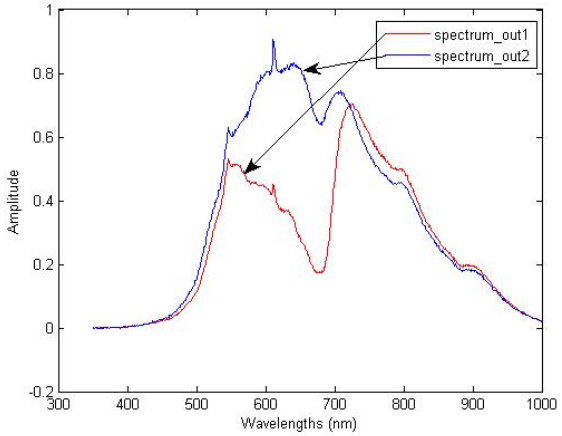


Figure 1.1: A graphical comparison of light reflection off of two leaves. The red line is a healthy leaf, the blue an unhealthy leaf.

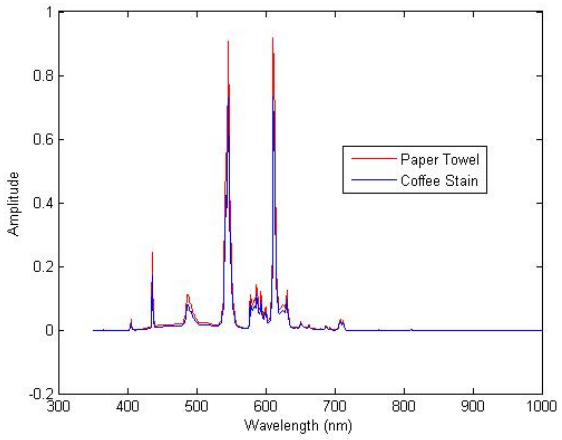


Figure 1.2: A graphical comparison of light reflection off of two paper towels. The red line is a clean paper towel, the blue a stained paper towel.

As demonstrated in *Figure 1.2*, not all comparisons are visually compelling. While there are differences in the spectra of light reflected from the compared objects, they are difficult to percieve simply through graphing. Here, the visual approach is of marginal efficacy at best. However, subtle differences such as those present in this graph become readily apparent when presented aurally.

From the global existence of music, it can be assumed that the difference between notes is readily apparent to the human ear. The modern vocal and inner ear apparatuses were first exhibited 1.5 million years ago in *Homo ergaster* and evidence of musical instruments dates back at least 40 thousand years to the Upper Paleolithic [1]. In this time, humanity has gained the ability to distinguish between notes of different frequency and to interpret chords comprised of individual notes of precisely varied frequencies relative to each other. Varied among cultures is the structure of these notes which creates a desireable chord, but within a given musical paradigm, the presence of notes not within the desired chord structure gives way to noticeable discordance.

In Western music, a D Major chord (comprised of notes D, F#, and A) will sound pleasant, but with the addition of an A#, the chord becomes dissonant. An even more drastic effect is observed with the addition of a note with a frequency which does not exist within the framework provided by the musical scale. This same D Major chord (comprised of frequencies *294Hz, 370Hz, and 440Hz*) will be greatly disrupted by the addition of a tone of *475Hz*, which lies between A# and B.

Unaided perception of notes at relatively similar frequencies is easily made possible from beat frequencies, a resultant tone generated by interference between two sound waves of different frequencies. This is described by the following equation:

$$f_{beat} = |f_1 - f_2|$$

Musicians can tune instruments using only their ears by listening to the beat frequency generated by the approach of their instrument to a desired reference tone. This implies an inherent ability to distinguish between any pair of frequencies, limited by the following restrictions:

$$20Hz < [f_1, f_2] < 20kHz$$

These frequency restrictions specify that the frequencies being compared are within the audible range of the unaided ear. It is important to note that beat frequencies below $20Hz$ may be detected; beat frequencies are an envelope function about the component frequencies. The applicable utility of this lies with the ability to superimpose two audio spectra, one corresponding to the expected optical input frequency and the other conforming to a changing input. As an example, in a scenario involving the inspection of objects intended to be uniform, a baseline audio spectrum could be created to correspond to an ideal object. This constant signal could then be superimposed with a time-dependent signal from a progressing source of produced objects. Ideally, a listener would hear only the baseline audio spectrum if the object being inspected is within acceptable parameters, while an unacceptable result would produce a noticeable beat frequency. Notably, the human brain does not require physical interaction between soundwaves to observe beat frequencies. Binaural beats are envelope functions identical to beat frequencies which are observed when one ear hears sound of frequency f_1, while the other ear hears sound of frequency f_2. In this case, the superposition of the waves occurs entirely within the brain.

This project creates a chord utilizing a frequency shift dependent upon the observed items. If an observer's attention must be drawn to the observed items, the frequencies comprising the chord are shifted by a degree determined by the optical nature of the items. If the frequency shift is small, the observer may detect it by noticing the beat frequencies generated by the simultaneous presence of similar notes. With a larger shift, discordance becomes noticeable, guaranteeing that any frequency shift will be intuitively observeable.

Automated optical inspection is commonly utilized in quality control for printed circuit boards. Similar to this project, these inspection systems make use of a light source, optical

sensors (image cameras), and assessment programming to verify the accuracy of the circuitry [2]. However, these systems are designed for a more specialized use, and thus use imaging for comparison. This project seeks to compare the spectral reflection from objects and provide the comparison as sound for the user to interpret, rather than making a boolean comparison based on established criteria.

2. Objectives

2.1 Goals

The primary focus of this project is to provide a new interface to compare two objects. Specifically, this will be accomplished by two fundamental elements. First, the ability must be established to observe the objects being compared. In this case, this will be accomplished by using a USB650 Red Tide Spectrometer[3] to measure the reflected light off of the objects. This will be performed using fixed conditions; among all measurements being compared, identical program settings and physical setups will be used.

The second component to be implemented will be the comparison. This includes the mathematical manipulations necessary to create soundwaves from optical spectra. The amplitudes of individual notes in the soundwaves will be defined by the intensity of the corresponding optical spectra; the frequencies of the notes will be harmonics defining an arbitrary chord, with the addition of a frequency shift defined by the difference between the optical spectra. By way of this frequency shift, spectral discrepancies will establish either dissonance or beat frequencies, calling attention to variances in the reflectance and enabling an intuitive aural monitoring system. If no discrepancy exists, the created soundwave will form the original unaltered chord.

By way of these operations, the pursued goal of this project may be realized: to design and implement an aural interface for intuitive industrial inspection. The inherent human ability to distinguish subtle differences in sounds is a crucial factor in designing an aural system to exhibit comparisons. With such a system, an inspector could simultaneously listen to the sound output of this sytem and primarily focus on an entirely separate task, yet retain the ability to quickly respond to a discrepancy in the sound output.

Upon inception of the project, several goals were put forth; using sound to indicate optical observations can be applied to a wide range of opportunities. One such goal, which

was considered for implementation, was using different tones to represent different colors observed in an individual's surroundings. This was not implemented due to time and scope constraints, but may be considered for future development. A plausible application would be tonal differentiation between objects and locations for the blind.

2.2 Essential Program Capabilities

In order to meet the goals established for this project, several key aspects must be properly enacted. The project can be broken down into two distinct components, one focusing on observation of the objects being compared and the other focusing on manipulation of the data thus acquired.

2.2.1 Observation

The physical setup for this project, as shown in *Figure 2.1*, is to be held constant within comparisons; any two objects being compared must be illuminated by identical light sources, held at identical distances from the spectrometer, and compared along the same wavelength range. While comparisons between variable orientations may yield further information, they also add mathematical complexity to data analysis and may change the origin of reflectance discrepancies. In the interest of focusing analysis on the reflection from the object, and not on the object's environment, the physical orientation is to be held as a constant.

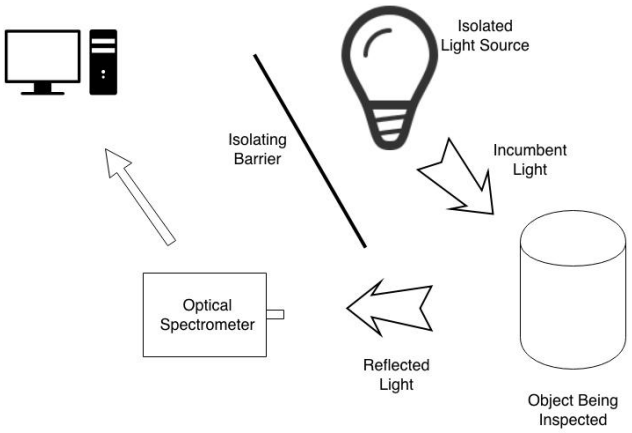


Figure 2.1: The physical layout used to observe objects. A light source isolated from directly illuminating the spectrometer radiates light onto an object. The reflected light is measured by a spectrometer, which sends measurements of the reflectance to a computer.

The computer must then store the measurements made by the spectrometer such that the data detailing the reflectance can be recalled and utilized. For the purpose of implementing this system in an industrial setting, the process of storing the data would be fully automated to take measurements at a precisely defined rate. For prototyping, however, automation is unnecessary.

2.2.2 Manipulation

Once the reflectance data has been saved to the computer, it will be retrieved by a program which will create the desired sound waveforms. The process, shown in *Figure 2.2*, will build soundwaves from the supplied data.

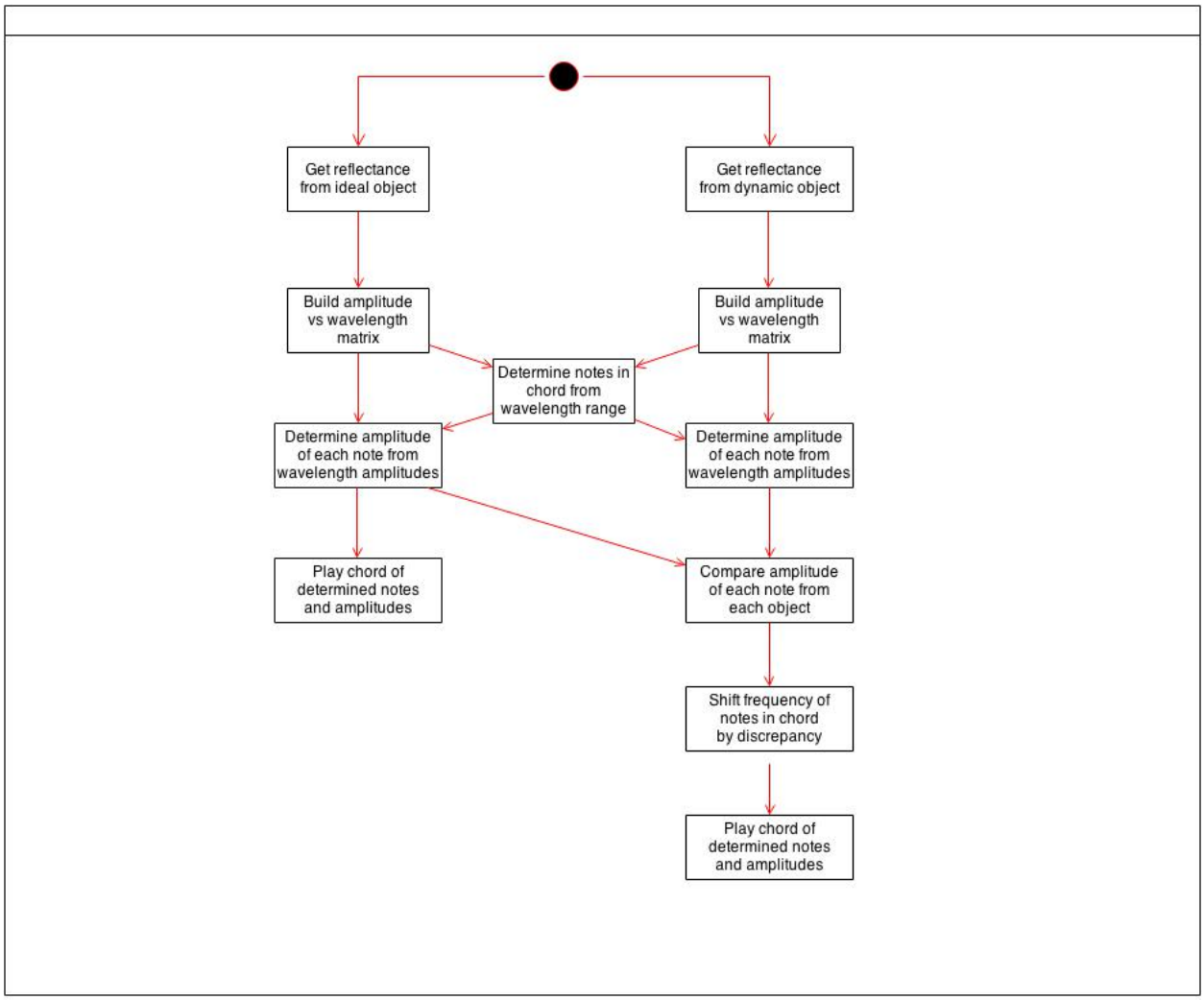


Figure 2.2: The order of actions to be taken by the program from data input through sound output.

First, the program will have to create a matrix (of the form defined in *Figure 2.3*) of the reflectance spectrum from the benchmark file; the benchmark object is the standard against which all inspected objects will be compared. Next, the reflectance spectrum of the item under test (IUT) file will be used to create a matrix. This file is the reflectance data from the object being inspected; in an industrial setting, the file would automatically be utilized,

logged, and replaced with the next IUT file.

$\lambda(nm)$	A
350	a_0
351	a_1
352	a_2
353	a_3
354	a_4
355	a_5
356	a_6
357	a_7
358	a_8
359	a_9
360	a_{10}
...	

Figure 2.3: The form of each matrix containing the measured amplitude at each observed wavelength.

After the matrices are created, they will be used to create musical chords. The length of each matrix, containing the reflectance amplitude at each wavelength in the measured spectrum, will be used to determine the number of notes in the chords to be created. For each note to be created, a bin (binning is described in detail in *Section 3.6*) will be generated corresponding to a section of the spectrum. The average value of the reflectance amplitudes in each bin will become the amplitude of the corresponding note in the chord. The benchmark chord will be played as such, but the IUT chord will first be altered to exhibit any discrepancies between the two reflected spectra. The notes in the chord will, for any bins which do not conform to the spectrum, be frequency shifted to induce a beat frequency. The chord will then be played.

3. Theory

Mathematics is heavily utilized in computer programming. The majority of this project involved matrix manipulation and algorithm development in MATLAB, a programming language meant for mathematical operations. For a full understanding of the details of the operation of the code, the following concepts are crucial.

3.1 Matrices and Arrays in MATLAB

In MATLAB, an m by n matrix A is an array of values arranged in m columns and n rows. An array can be manually written between a pair of brackets using commas to separate rows. All rows in an array must be of the same magnitude.

$$[a_1\ a_2\ a_3\ a_4\ ...\ a_n,\ b_1\ b_2\ b_3\ b_4\ ...\ b_n,\ c_1\ c_2\ c_3\ c_4\ ...\ c_n,\ ...]$$

creates the matrix:

$$\begin{bmatrix} a_1 & a_2 & a_3 & a_4 & ... & a_n \\ b_1 & b_2 & b_3 & b_4 & ... & b_n \\ c_1 & c_2 & c_3 & c_4 & ... & c_n \\ ... & & & & & \end{bmatrix}$$

Any individual value can be accessed using the form $A(i, j)$.

3.1.1 Array Operators

Colon Operator

To access a specific set of values within the matrix, the form

$$B = A(i_a : i_b\ ,\ j_c : j_d)$$

will create a matrix B containing all values which are contained in rows i_a through i_b and

also contained in columns j_c through j_d. Here, the colon operator is used to define a range of values. This can be used in two ways.

$$1 : 10 = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10]$$

. To specify an array with nonunit spacing, a second colon operator will define the amount by which to increment.

$$0 : 10 : 100 = [0\ 10\ 20\ 30\ 40\ 50\ 60\ 70\ 80\ 90\ 100]$$

Element-by-Element Operators

An important notation in MATLAB is the element-by-element operator. By appending a period to the front of an arithmetic operator, in the form $A .* B$, each element $A(i,\ j)$ will be multiplied by the corresponding element at $B(i,\ j)$. This allows arithmetic operations to be performed on one matrix by another matrix of identical dimensions while retaining the original dimensions m by n of each matrix.

3.1.2 Bit Depth

Several MATLAB commands utilized in this project incorporate bit depth. This value specifies the precision of each value in an array by defining the number of bits used to store the number. A value of bit depth 8 has $2^8 = 256$ possible values, while a value of bit depth 16 has $2^{16} = 65,536$ possible values. High quality audio can have a bit depth as high as 24, which has $2^{24} = 16,777,216$ possible values. By increasing the bit depth of values at a fixed decimal place, the precision increases exponentially.

3.2 Vector Scaling

In the context of this project, the commands **sound(y)** and **soundsc(y)** operate on an m by 1 column vector **y**. Both write this as a waveform to the soundcard. The **sound(y)** command sound creates a sound which clips all elements of magnitude greater than one. The **soundsc(y)** command linearly scales the column vector **y** to fit between -1 and 1 before writing the waveform to the soundcard.

This linear scale can be replicated using the following algorithm in MATLAB:

$$B = (2 .* \ A \ - \ min(A(:)) \ - \ max(A(:))) \ ./ \ (max(A(:)) \ - \ min(A(:))))$$

The **min(M)** and **max(M)** commands find the maximum and minimum values, respectively, within the matrix **M** upon which they operate. This algorithm writes a new column vector B where each element B_i is determined by the following expression:

$$B_i = \frac{A_i - A_{min} - A_{max}}{A_{max} - A_{min}}$$

3.3 Wave Superposition

Summing waves in MATLAB is a trivial operation; MATLAB assumes matrix addition is element-by-element, thus the summation of two matrices of identical dimensions is written as $A = B + C$. Matrix addition is used in this project to superimpose multiple sine waves of varying frequencies, creating a composite wave.

3.4 Optical Reflectance Spectrum

The input data for this project was the spectrum of the light reflected off of various objects. These spectra were saved as 2 by n arrays; the first column, $data\_array(:, 1)$, is a list of the wavelengths at which reflectance was measured and the second column, $data\_array(:, 2)$, is a list of the measured amplitude corresponding to each wavelength.

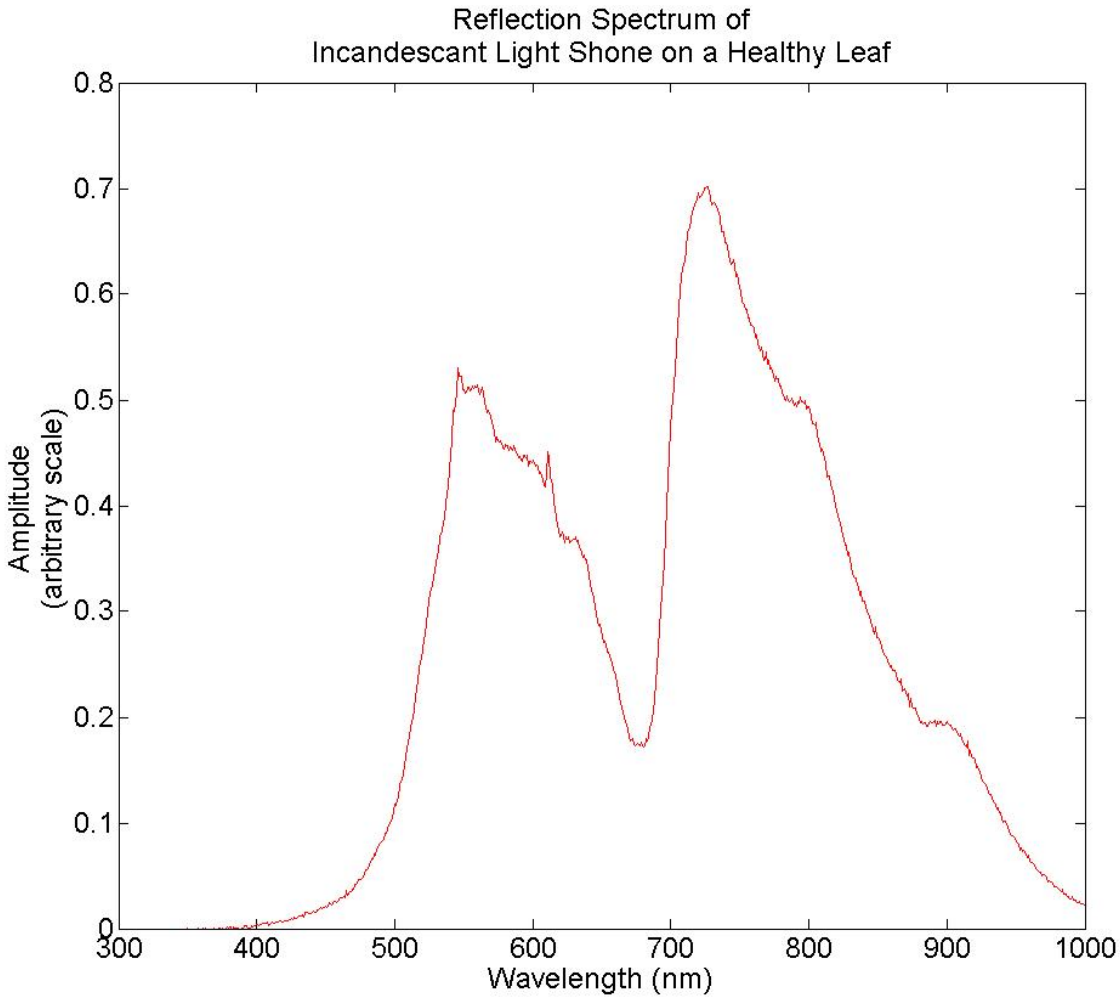


Figure 3.1: A plot of measured amplitude versus corresponding wavelength.

3.5 Sound Waves

The output data of this program is a soundwave created using the reflectance spectrum in the input data. The soundwave is an arbitrarily chosen chord, which contains harmonic frequencies of each note in the chord. The number of harmonics included in the chord is

determined by the number of wavelengths measured in the reflectance spectrum; for a range of less than $400nm$, twelve notes are selected, while for a range of $400nm$ or greater, twenty-four notes are selected. The amplitudes are determined by binning the optical spectrum.

3.6 Binning

One of the most crucial aspects of this project is the creation of a soundwave from an optical spectrum. However, as seen in *Figure 3.1*, the number of discrete wavelengths measured will most likely be far larger than the number of notes selected for the produced chord. Binning is the process by which the values stored in a large array are used to create a smaller array. The larger array is divided into segments, or *bins*; each bin is then averaged, and the average value is written to the smaller array. The larger array in this application is the optical spectrum and the smaller array is a matrix containing the respective amplitudes of each note to be contained in the soundwave. This array is an intermediate stage before the soundwave is created.

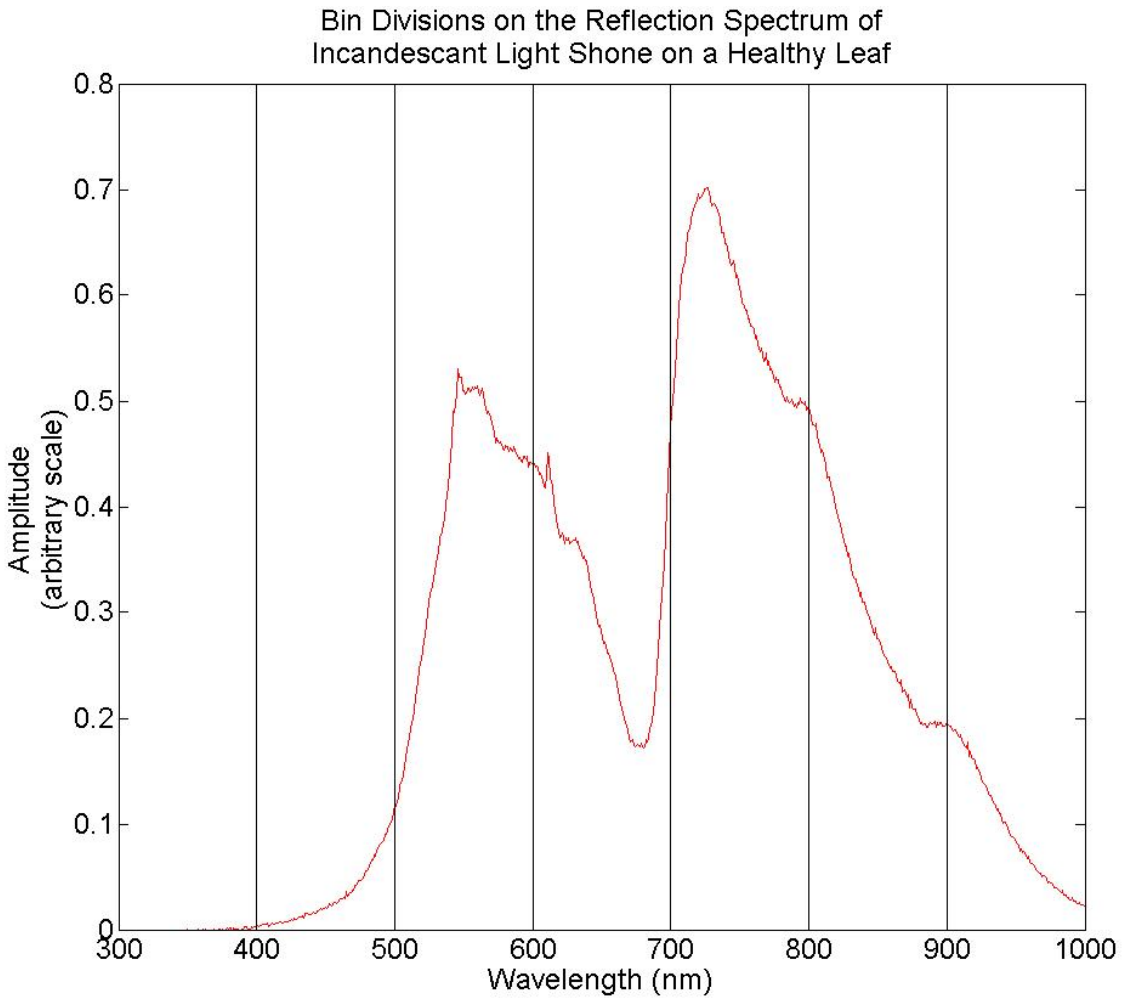


Figure 3.2: A binned plot of measured amplitude versus corresponding wavelength.

3.7 Frequency Shift Algorithm

In order for the generated soundwave to announce the presence of discrepancies between the benchmark item and the IUT, an algorithm was written to shift the frequencies of the notes in the chord based on the discrepancies. In the chord corresponding to the benchmark item, the notes in the chosen chord are unaltered. In the IUT chord, each note frequency is shifted by the following operation:

$$\text{shiftednote}_i = \text{note}_i * \frac{\langle \text{ratio} \rangle}{\text{ratio}(i)}$$

where *ratio* is the resulting matrix of dot division between the binned arrays of the compared

spectra and *ratio(i)* represents an individual value in the array

$$\langle \text{ratio} \rangle = \left\langle \frac{\begin{bmatrix} b_1 & b_2 & b_3 & b_4 & b_5 & b_6 & b_7 \end{bmatrix}}{\begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 \end{bmatrix}} \right\rangle = \frac{b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7}{a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7}$$

$$\text{ratio}(i) = \frac{b_i}{a_i}$$

Here the IUT reflectance is given by the array $\begin{bmatrix} b_1 & ... & b_n \end{bmatrix}$ and the benchmark reflectance is given by the array $\begin{bmatrix} a_1 & ... & a_n \end{bmatrix}$. The choice to divide the IUT reflectance by the benchmark reflectance is arbitrary, as is the choice to divide $\langle \text{ratio} \rangle$ by ratio(i). The frequency shift would continue to function if either or both were inverted.

For each bin, this multiplies the corresponding note frequency by the average ratio of the IUT reflectance to the benchmark item reflectance. This is then divided by the specific bin's ratio of the IUT reflectance to the benchmark reflectance. If the ratios are not the same, this is indicative of a discrepancy in the reflected spectra and the corresponding notes in the IUT chord are subjected to a frequency shift. Thus, frequency shifting is induced in the case

$$\frac{b_i}{a_i} \neq \frac{b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7}{a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7}$$

and is proportional to

$$\frac{a_i(b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7)}{b_i(a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7)}$$

This is a sensible result, as any multiplicative factor must be unitless. If the spectra are identical in value, the notes remain unchanged. Additionally, if the spectra vary by a constant factor, the notes remain unchanged. This ensures that a frequency shift will only be induced if the shape of the spectra differ.

4. Methodology and Results

4.1 Physical Apparatus

The physical setup for this project, as shown again in *Figure 4.1*, used a light source isolated from directly illuminating the spectrometer to shine upon the object being inspected. The light reflected off of the object entered a nearby fiber-optic cable which delivered the light to the spectrometer. The position of the object being inspected, the ambient environment, and the proximity and orientation of each piece of equipment was kept identical among all measurements being compared. Additionally, the light source was kept the same among all comparisons. The spectrometer was observed to have non-zero amplitudes for the observed wavelengths even when there was no light input; this background was corrected for in the program written by subtracting the amplitude intensity measured when the spectrometer input was blocked with a cap.

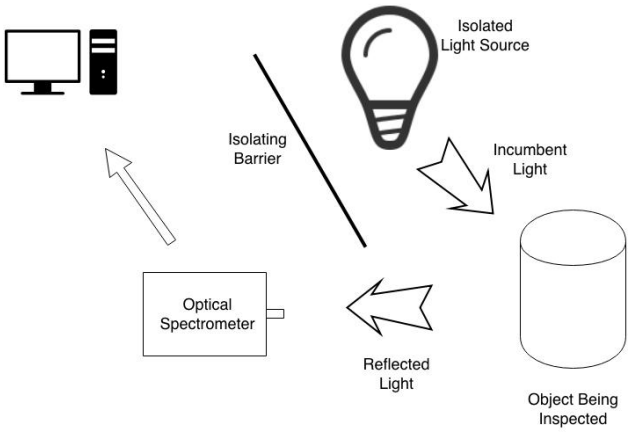


Figure 4.1: The physical layout used to observe objects. A light source isolated from directly illuminating the spectrometer radiates light onto an object. The reflected light is measured by a spectrometer, which sends measurements of the reflectance to a computer.

The reflection spectra were measured using a USB650 Red Tide Spectrometer [3]. This

spectrometer functions in the range [350nm - 1000nm]. The measured reflection spectra were interpreted and saved to the computer using Logger Pro [4]. This computer data logger saved spectral data based on the parameters given. Time averaging of data, the spectral range observed, and sensitivity to light amplitude could all be adjusted using Logger Pro.

4.2 MATLAB

The program outlined previously was written in MATLAB over the course of the project. Initial work was not intended to provide a usable product, but rather to establish familiarity with the basic concepts governing the structure, notation, and vocabulary of MATLAB. From the outset, all work was intended to gradually move towards the completion of the final program. Over time, as familiarity was built and advanced concepts became more accessible, the programs written were able to achieve this. A particularly noteworthy discovery was *tonec.m* [5], a function written by Professor Kevin Donohue of the University of Kentucky; this function, shown in Appendix A, greatly facilitated the development of the project by streamlining the process of creating soundwaves. Additionally, the documentation provided by Mathworks [6] provided examples and explanations which further accelerated the development of the programs.

4.2.1 Development of the Program

The individual test programs written, fully and sequentially shown in Appendix B, advanced from the most basic concepts up to the concepts driving the final iteration of the program. All programs were fully commented such that the purpose of the test was apparent.

The first program of twenty-five tested manual creation of soundwaves as well as the utility of the **sound(y, Fs, bits)** command, which sends an audio signal **y** to the soundcard of the computer at sample rate **Fs** with bit depth **bits**; bit depth indicates the precision of the values in signal **y**. The program was successful, generating a sound corresponding to the manually written waveform **y**.

```
y = repmat([1 0 -1 0], 1, 10000);
sound(y,2000,16)
```

The second program used a template program found in the Mathworks documentation [6] to generate a square wave. The number of cycles of the wave, the length of each cycle, and the percentage of each cycle in which the wave was set to a high value were all manually

chosen. This wave was then sent to the soundcard using the **sound(y)** command; in this case, where the function is not supplied with either **Fs** or **bits**, they are set to default values of *8192Hz* and 8-bit depth, respectively.

```
RESOLUTION = 20; %whatever is appropriate
DUTYCYCLE = .73; %e.g. 73% on, 27% off
NUMBEROFCYCLES = 180; %as appropriate
basepulse = ones(1,RESOLUTION);
squarepulse = basepulse;
squarepulse(floor(DUTYCYCLE * RESOLUTION) + 1 : end) = 0;
wavetrain = repmat(squarepulse, 1, NUMBEROFCYCLES);
y = wavetrain;
sound(y);
```

The third and fourth programs focused on writing and reading data to and from comma separated value (*.csv*) files using **csvwrite(filename, M)** and **csvread(filename)**, where **M** is the matrix to be written to the file.

```
csvwrite('csvtest1.dat',m);
type csvtest1.dat;
```

Upon discovering that the data saved by Logger Pro was not saved as *.csv* files, the fifth program was written to replicate the results of the third using the proper *.txt* files. The command used was **dlmread(filename, delimiter, range)**; specifying the delimiter **'\t'** allowed the program to correctly parse the spectral data.

```
M = dlmread('dec3a.txt', '\t', 1,1);
```

Test programs six and seven were designed to normalize a matrix before using **sound(y, Fs, bits** to write the matrix to the soundcard. While intended to normalize the wavefunction between negative one and one, the algorithm used was incorrect and did not accomplish its goal.

```
normM = M./max(M);
finalnormM = normM.*2.-1;
syntaxtestM = M./max(M).*2-1;
```

finalM = repmat(finalnormM,1,100);

Program eight used **sin(X)** to generate a sine wave of the elements of **x**, which was then written to the soundcard using **sound(y, Fs)**.

```
x = 0:0.01:100*pi;
A = repmat(sin(x),1,20);
sound(A,200000);
```

The ninth program used **soundsc(y, Fs)** to normalize and write to the soundcard a sine wave created at a chosen frequency and samplerate. Where **sound(y)** clipped values of magnitude greater than one, **soundsc(y)** normalizes the values to fit between negative one and one before writing to the soundcard.

```
testFreq = 440;
samplerate=8192;
t = [0:1: 4*samplerate];
soundsc(sin(testFreq/samplerate*t*2*pi),samplerate);
```

Program ten created sine waves of chosen frequencies and added them before writing the resultant chord to the soundcard using **soundsc(y)**.

```
Freq1 = 554;
freq1 = 547;
Freq2 = 659;
Freq3 = 831;
samplerate=8192;
t = [0:1: samplerate];
Db = sin(Freq1/samplerate*t*2*pi);
Db1 = sin(freq1/samplerate*t*2*pi);
E = sin(Freq2/samplerate*t*2*pi);
Ab = sin(Freq3/samplerate*t*2*pi);
chord = Db + E + Ab;
chord2 = Db1 + E + Ab;
soundsc(chord);
```

soundsc(chord2);

Program eleven created sine waves at each note in an octave, wrote them to the soundcard using **soundsc(y)**, and then saved each sine wave to a data file using **csvwrite(filename, M)**. Program twelve accessed these files. These programs were intended to test the concept of saving sound data to permanent files. By using these files as a reference, rather than repeatedly generating the sine waves for each use, later programs would save processing time. Also saved were the maximum and minimum values of each sine wave for later use in normalizing.

Nmax = max(A&Bb&B&C&Db&D&Eb&E&F&Gb&G&Ab)
Nmin = min(A&Bb&B&C&Db&D&Eb&E&F&Gb&G&Ab)

Program thirteen was an attempt to normalize the sine waves before saving them to further reduce processing time. However, the normalization algorithm was unsuccessful, caused the program to fail, and was removed.

The fourteenth program created eight sine waves at harmonic frequencies, summed them together, and both plotted them with **plot(M)** and wrote them to the soundcard using **soundsc(y)**.

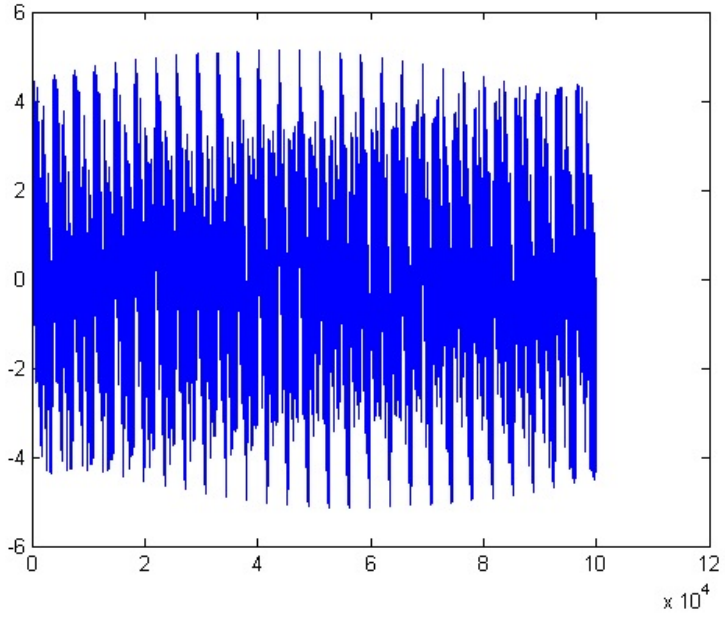


Figure 4.2: The sum of eight harmonic sine waves.

The fifteenth program was written to test the *tonec.m* function using **csvwrite(filename, tonec(frequency, time, samplerate))** to save the waveform and **soundsc(tonec([**$frequency_1$**, ... ,** $frequency_n$**], time, samplerate))** to generate a chord of chosen frequencies.

```
sound(tonec(440,1,10000));
soundsc(tonec([28,55,110,220,440,880,1760,3520],1,10000));
```

The sixteenth program used a noise reduction algorithm to minimize the effect of non-zero background measurements. This method of background reduction was removed in the twentieth program in favor of a preferable method; this algorithm introduced an undesireable nonlinear distortion to the array.

$$B_i = \frac{A_i * |A_i|}{A_{max}}$$

```
spectrum_in = dlmread('dec3a.txt', '\t', 1,1);
all = dlmread(''dec3a.txt', '\t'', 1, 0);
t = all(1:end,1);
noiseless = spectrum_in.*abs(spectrum_in)/max(abs(spectrum_in));
plot(t,noiseless,'r-',t,spectrum_in,'b-');
axis tight;
```

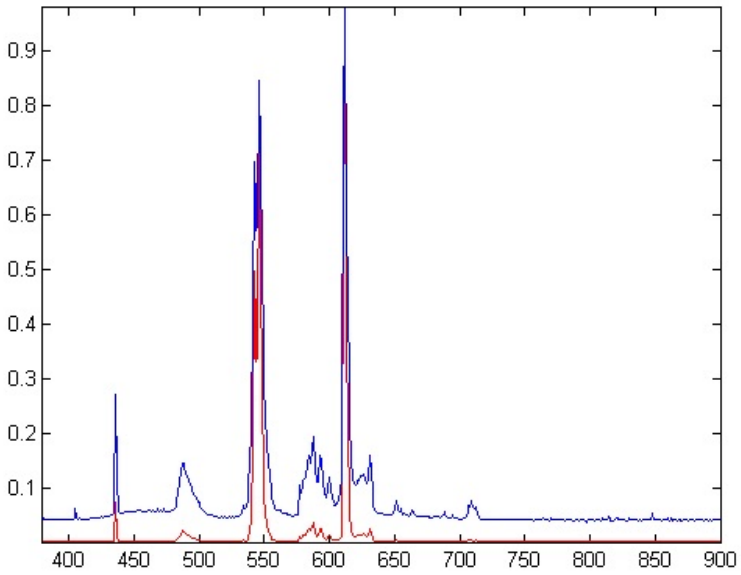


Figure 4.3: A nonlinear reduction designed to emphasize spectral peaks and minimize the effects of background signals.

The seventeenth and eighteenth programs investigated methods for binning the input spectrum data. First attempted was the **hist(Y, nbins)** command; the resulting plot failed to carry any of the desired information and created a histogram graph rather than an array. This approach was abandoned in favor of the binning algorithm detailed in section 3.6. The binning algorithm successfully created an array storing the averaged values of the spectrum data array and determined the number of bins using the size of the spectrum data array.

bin_locations = [450,500,550,600,650,700,750,800,850];

hist(noiseless,bin_locations);

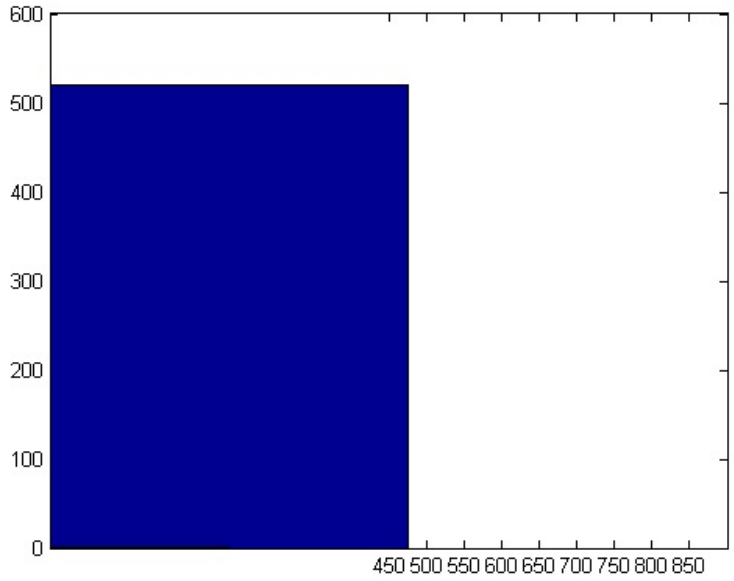


Figure 4.4: A histogram made to bin the spectral data.

```
[M,N] = size(t)
if M < 700
    bin_num = 7;
else
    bin_num = 14;
end
bin = round(M./bin_num)
k = 0;
for i = 1:bin_num
    finale_array(i,1) = i;
    index=i-1;
    k = index*bin+1;
    finale_array(i,2) = mean(noiseless(k:(k+bin)));
end
noiseless_finale(:,1) = 1:bin_num;
noiseless_finale(:,2) = finale_array(:,2).*abs(finale_array(:,2))...
    /max(abs(finale_array(:,2)));
```

plot(noiseless_finale(1:7,1),noiseless_finale(1:7,2))

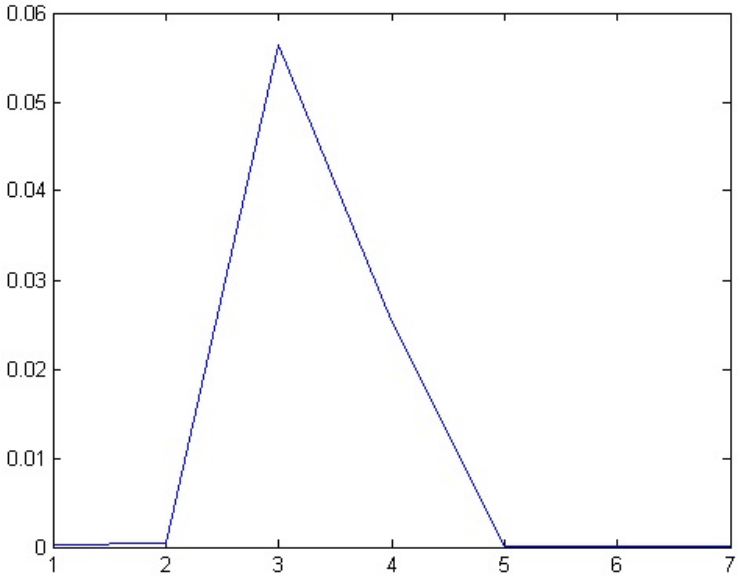


Figure 4.5: A graph of the binned values from the binning algorithm.

Program nineteen was a proof-of-concept test to create a soundwave using spectrum data saved in a file using the **dlmread(filename, delimiter, range)** command in program five, the noise reduction algorithm in program sixteen, the binning algorithm from program eighteen, the **tonec(**[$frequency_1$**, ... ,** $frequency_n$]**, time, samplerate))** command from program fifteen, and the **soundsc(y)** command from program nine. This program successfully created a chord.

Program twenty replicated the efforts of program nineteen, but removed the noise reduction algorithm from the sixteenth program to remove nonlinear distortion.

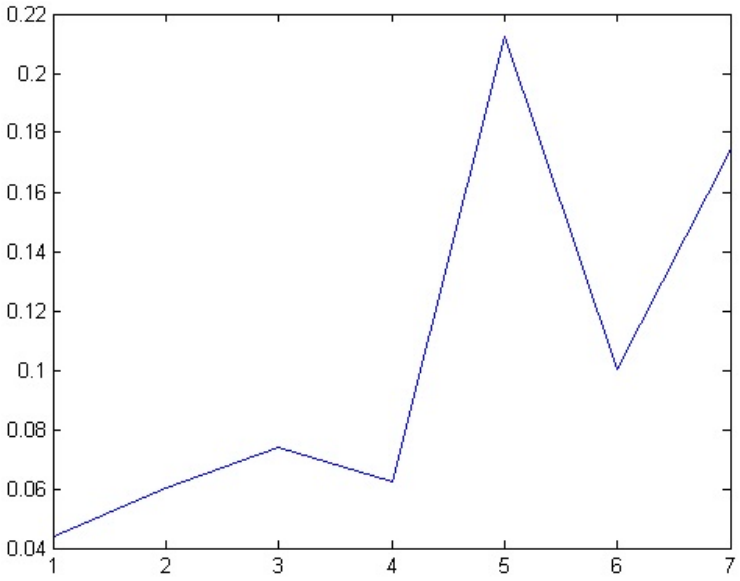


Figure 4.6: A graph of the binned amplitudes for the notes output by the *tonec.m* command.

The twenty-first program replaced the previous noise reduction algorithm with a linear background reduction. New data was used, including a measurement of the background readings from the spectrometer when all light input was blocked. This background was linearly subtracted from the spectrum data.

```
spectrum_in = dlmread('Spectrometer Data\jan24e.txt', '\t', 1,1);
all = dlmread('Spectrometer Data\jan24e.txt', '\t', 1, 0);
t = all(1:end,1);
background = dlmread('Spectrometer Data\jan24 background.txt', '\t', 1, 1);
a=find(background, t(1));
b=find(background, t(end));
background_adj = background(a:b);
spectrum_out = spectrum_in - background_adj;
```

The twenty-second program created a chord using the benchmark spectrum data and a second chord using the IUT data. The notes in the IUT chord were frequency shifted using the algorithm

$$\text{shiftednote}_i = \text{note}_i * (1 + \text{constant} * (bin_{IUT} - bin_b))$$

This algorithm was later found to be inferior to the method described in *Section 3.7*. By comparing the absolute value of reflected intensities, this algorithm produced a false negative where the reflection spectra differ in overall intensity, but are proportional.

```
weight = 1 - 10*final_array1(:,2) + 10*final_array2(:,2);
weighted = weight .* notes
plot(final_array1(:,1),final_array1(:,2), 'r');
hold;
plot(final_array2(:,1),final_array2(:,2), 'g');
```

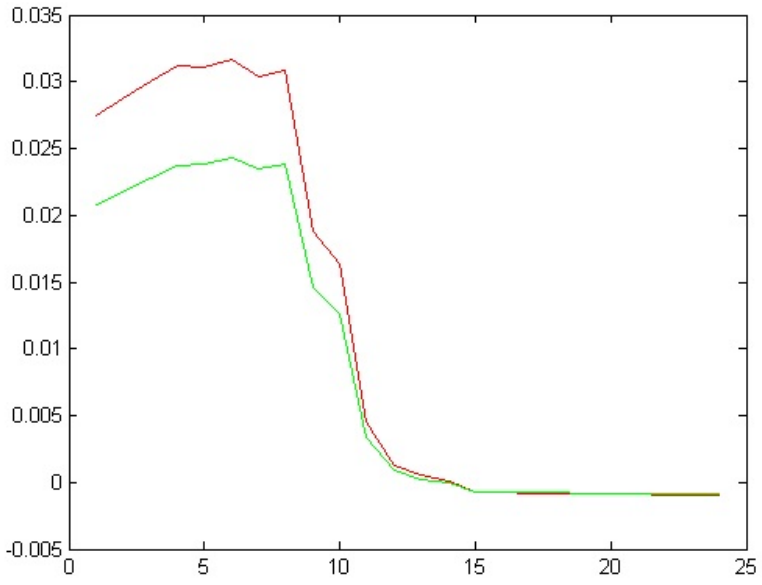


Figure 4.7: A comparison of the binned values from the benchmark reflectance(red) and the IUT reflectance (green).

Twenty-third and twenty-fourth programs continued to investigate algorithms to induce frequency shift. The twenty-third was an incremental step towards the twenty-fourth, in which the final frequency shift algorithm was introduced.

$$\text{shiftednote}_i = \text{note}_i * \frac{\langle \text{ratio} \rangle}{\text{ratio}(i)} = \text{note}_i * \frac{a_i(b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7)}{b_i(a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7)}$$

from test23_weighting.m:

```
ratio = final_array2(:,2) ./ final_array1(:,2)
weight = 1 + ratio(:) - mean(ratio(:));
weighted = weight .* notes
testing = spectrum_out2(:,1) ./ spectrum_out1(:,1);
plot(t(:,1),testing(:,1), 'r');
```

from test24_newweight.m:

```
ratio = final_array2(:,2) ./ final_array1(:,2)
weight = mean(ratio(:)) ./ ratio(:);
weighted = weight .* notes
testing = spectrum_out2(:,1) ./ spectrum_out1(:,1);
plot(t(:,1),testing(:,1), 'r');
```

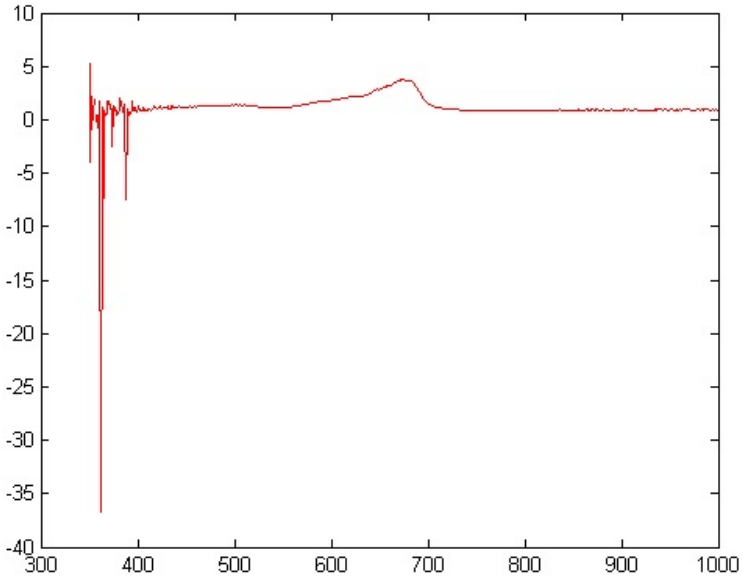


Figure 4.8: The ratio of the IUT reflectance divided by the benchmark reflectance.

The twenty-fifth program manually wrote two arrays to be compared; these arrays were manipulated to test if the frequency shift was functioning in all cases. When one array was a multiple of the other

$$A = c. * B$$

where c was a constant, no frequency shift was observed. If one array could not be represented as a multiple of the other, a frequency shift was observed. This demonstrated the success of the frequency shift algorithm.

```
spectrum_in1 = [5,9,11,25,9,5];
background_adj = [1,1,1,1,1,1];
spectrum_out1 = spectrum_in1 - background_adj;
bin_num = 6;
notes = [294, 370, 440, 587, 740, 880]';
spectrum_in2 = [3,5,6,13.5,5,3];
spectrum_out2 = spectrum_in2 - background_adj;
ratio = spectrum_out2(:) ./ spectrum_out1(:)
weight = mean(ratio(:)) ./ ratio(:)
weightednotes = weight .* notes
soundsc(tonec([weightednotes,spectrum_out2(:)], 1, 10000));
testing = spectrum_out1(:) ./ spectrum_out2(:)
plot(testing(:), 'g');
```

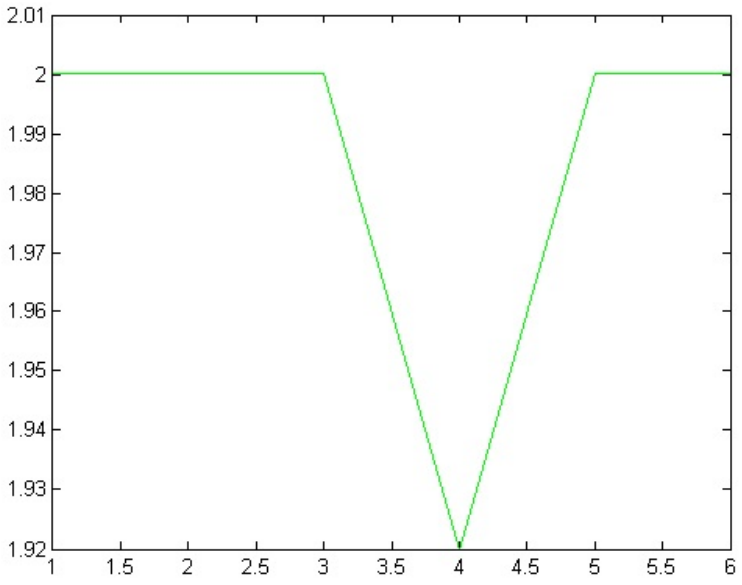


Figure 4.9: The frequency shift induced by the diference between the IUT reflectance and the benchmark reflectance.

The first attempt at writing a final program, *final1.m*, used the frequency shift algorithm in program twenty-two, and was later replaced with *final2.m*, which uses the successful frequency shift algorithm.

weight = 1-sensitivity*final_array1(:,2)+sensitivity*final_array2(:,2)

weighted = weight .* notes

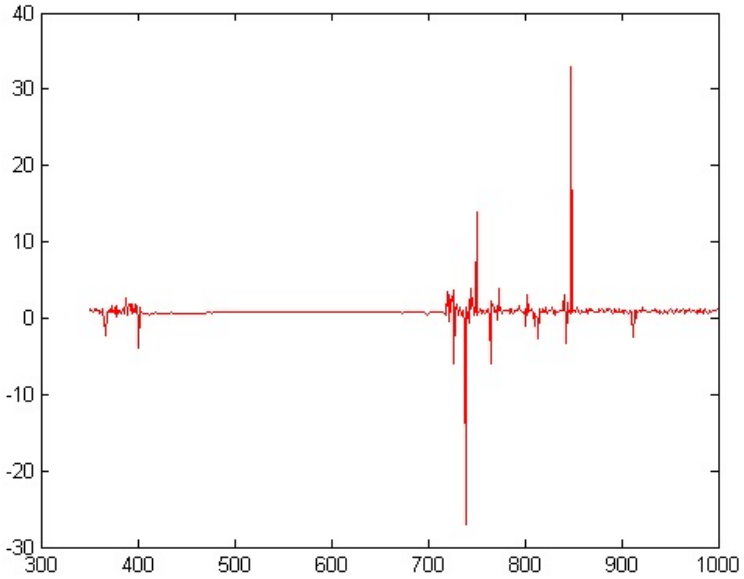


Figure 4.10: The ratio of the IUT reflectance divided by the benchmark reflectance.

4.2.2 Final Program

After successfully testing multiple approaches and programs, two final programs were written to accomplish all goals set for the project. *final1.m*, while functional, was replaced by *final2.m*, which contains a better algorithm of determining the frequency shift, shown in section 3.7, in the IUT chord as well as a more flexible system for determining the wavelength range; the minimum wavelength measured by the spectrometer is saved as a variable, rather than as a constant. The algorithm determining the frequency shift was altered such that it would compare the relative spectra rather than comparing the absolute spectra. By comparing the absolute spectra, two identical objects could be interpreted as different if one reflects a greater quantity of light than the other, even if the relative amplitudes by wavelength are the same. The newer algorithm compares the relative amplitudes among one object to the relative amplitudes among another, revealing discrepancy only if the objects

reflect wavelengths in differing proportions.

The program *final2.m* incorporates results directly from the developmental programs. The spectral data was imported using the **dlmread(filename, delimiter, range)** command from *test5*. The **soundsc(y, Fs)** came from *test9* and the **tonec(f, INT, Fs)** command was first tested in *test15*. The **if/else** statement used to choose the number of bins and the **for** loop used to bin the spectral data both came from *test18*. The background reduction was developed in *test21* and the frequency shift algorithm was created in *test24*.

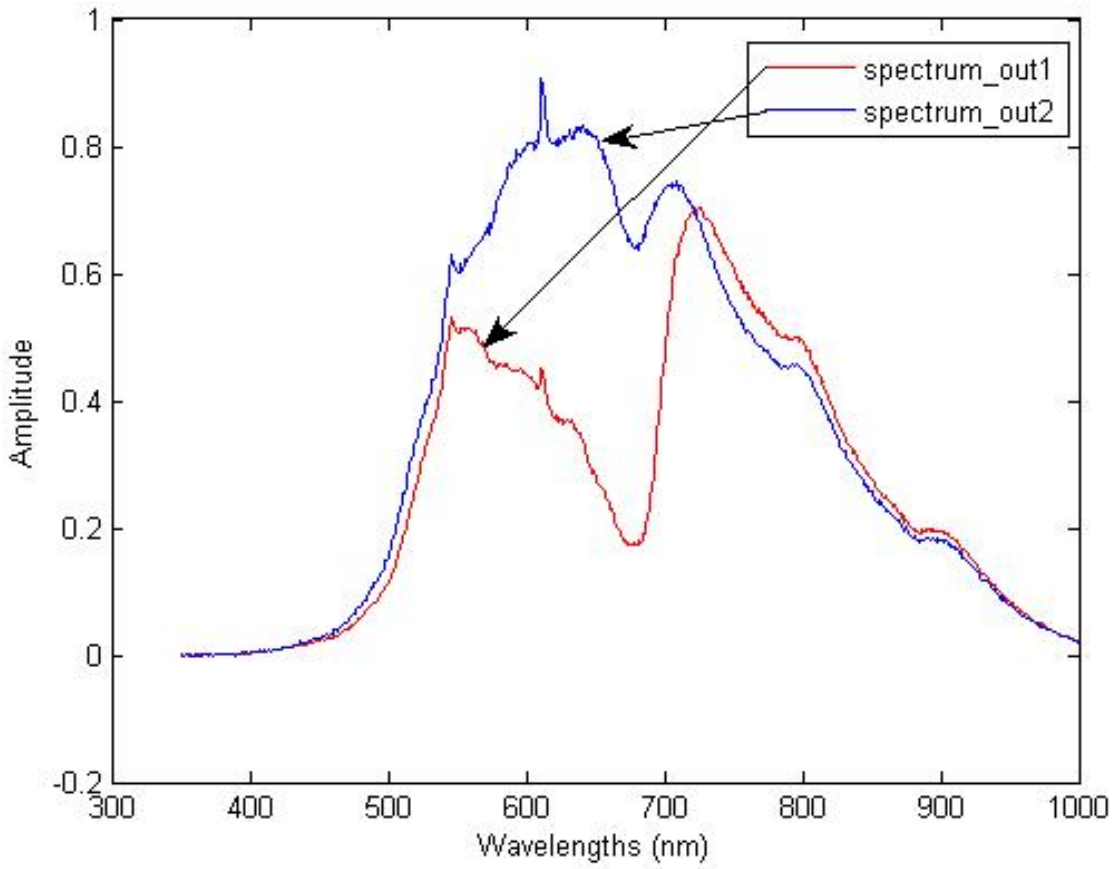


Figure 4.11: A graphical comparison of light reflection off of two leaves..

Examples of the soundwaves created by the final program are provided as supplemental files and correspond to Figure 4.11. The red line is the spectrum data used to create *healthyleaf.wav* and the blue line is the spectrum data used to create *unhealthyleaf.wav*.

5. Conclusion

The overarching goal of this project was to create an aural interface for intuitive industrial inspection. In order for this interface to be created, a physical system was devised to acquire reflection spectra corresponding to the items being inspected and a MATLAB program was written to create sounds communicating the nature of the reflections.

By announcing the presence of spectral discrepancies as audible beat frequencies, the system developed in this project relegates the inspection process from a conscious act to an intuitive act. With this interface in place, a worker could focus on an entirely unrelated task, yet retain the ability to observe items of substandard quality. By removing the need for dedicated attention to inspection and instead relying upon the inherent ability to detect beat frequencies and dissonance, this system can greatly increase inspection efficiency and facilitate multitasking among workers.

An aural inspection system derived from optical reflectance would be of great utility in any inspection process in which a variance in reflectance corresponds to decreased value of the item. Particularly, consumable products provide a vast market for application. Expired or otherwise tainted food would be easily discerned due to the decay process and different foods would be easily separable. Other applications are equally viable; this system can be applied to wide variety of scenarios.

Going forward with this project, future work should be considered to test the inspection system in an industrial setting. The spectrum input should be automated in further development and a graphical user interface should be implemented to increase market appeal. While this project has built a framework for a radical new inspection system, aesthetic and experimental development should be pursued to bring it to market.

Appendix A

tonec.m

```
function [v, t] = tonec(f,int,fs)
%  This function will create a series of samples at sampling rate
%  FS for a duration of INT seconds at frequencies in the first column
%  of vector F in Hertz.  The second column of F will be the weight for
%  each tone.  If a second column is not given then an equal scaling
%  will be given to each tone.
%
%              [v, t] = tonec(f,int,fs)
%
%  The output is a row vector V containing the sampled points
%   T is an optional output and is the time axis assoicated with V.
%
%   Written by Kevin D. Donohue (donohue@engr.uky.edu) 6/2003


[r, c] = size(f);  %  Check dimension of f
%  Ensure frequency values are in different rows
if c > 2  %  if not transpose
    f = f';
    [r, c] = size(f);
end

%  Check to see if coefficients for frequenies are provided
if c == 1   %  If not set them = to one
    f(:,2) = ones(length(f),1);
end

t = [0:fix(fs*int)-1]/fs;   %  Create time axis
v = zeros(size(t));    %  initialize vector to accumulate multiple tones
for k=1:r
    v = v + f(k,2)*sin(2*pi*t*f(k,1));     %  Create sampled tone signal
end
```

```
Error using tonec (line 16)
Not enough input arguments.
```

Published with MATLAB® 7.14

Appendix B

Test Code Written Throughout the Project

```
% test1_sound.m
% goal: basic test of sound
% y = load \\filepath
y = repmat([1 0 -1 0], 1, 10000);
sound(y,2000,16)
```

```
% test2_square_wave.m
% goal: create square wave

%-----------------------------------------------------
% found on mathworks.com
RESOLUTION = 20;   %whatever is appropriate
DUTYCYCLE = .73;    %e.g. 73% on, 27% off
NUMBEROFCYCLES = 180;  %as appropriate
basepulse = ones(1,RESOLUTION);
squarepulse = basepulse;
squarepulse(floor(DUTYCYCLE * RESOLUTION) + 1 : end) = 0;
wavetrain = repmat(squarepulse, 1, NUMBEROFCYCLES);
%-----------------------------------------------------

y = wavetrain;
sound(y);
```

Published with MATLAB® 7.14

```
% test3_csv.m
% goal: import and/or write .csv file

% to write: csvwrite(filename,M)
% to write: csvwrite(filename,M,row,col)

m = [3 6 9 12 15; 5 10 15 20 25; ...
     7 14 21 28 35; 11 22 33 44 55];

csvwrite('csvtest1.dat',m);
type csvtest1.dat;

% to import: M = csvread(filename);
% to import: M = csvread(filename,row,col);
% to import: M = csvread(filename,row,col,csvRange);

A = csvread('csvtest1.dat');
```

```
3,6,9,12,15
5,10,15,20,25
7,14,21,28,35
11,22,33,44,55
```

Published with MATLAB® 7.14

```
% test4_csv.m
% goal: write m by 2 matrix to a .csv file

% to write: csvwrite(filename,M)
% to write: csvwrite(filename,M,row,col)

m = [3 6; 9 12; 15 5; 10 15; 20 25; ...
     7 14; 21 28; 35 11; 22 33; 44 55];

csvwrite('csvtest2.csv',m);
type csvtest2.csv;

% success? compare vs spectral output
```

```
3,6
9,12
15,5
10,15
20,25
7,14
21,28
35,11
22,33
44,55
```

Published with MATLAB® 7.14

```
% test5_txt.m
% goal: input .txt from spectrometer
% goal: save only desired numbers in a matrix

% M = dlmread(filename, delimiter, range)

M = dlmread('dec3a.txt', '\t', 1,1);
```

Published with MATLAB® 7.14

```
% test6_txt.m
% goal: normalize and transpose matrix imported from .txt

M = transpose( dlmread('dec3a.txt', '\t', 1,1));

normM = M./max(M);

finalnormM = normM.*2.-1;

syntaxtestM = M./max(M).*2-1;

tf = isequal(finalnormM,syntaxtestM);
```

Published with MATLAB® 7.14

```
%test7_sound.m
% play .txt direct using sound

M = transpose( dlmread('dec3a.txt', '\t', 1,1));

normM = M./max(M);

finalnormM = normM.*2.-1;

finalM = repmat(finalnormM,1,100);

sound(finalM, 10000);
```

Published with MATLAB® 7.14

```
% test8_sound.m
% goal: generate and play a sinewave

x = 0:0.01:100*pi;

A = repmat(sin(x),1,20);

sound(A,200000);
```

Published with MATLAB® 7.14

```
% test9_sine.m
% goal: import desired frequency
% play corresponding sine wave at default sample rate

testFreq = 440;

% discovered soundsc() scales sound to fit -1 to 1 range
% no need to manually normalize

% remember: default sample rate is 8192 Hz
samplerate=8192;

t = [0:1: 4*samplerate];

soundsc(sin(testFreq/samplerate*t*2*pi),samplerate);
```

Published with MATLAB® 7.14

```
% test10_sum_sine.m
% goal: import desired frequencies
% play sum of corresponding sine waves at default sample rate

Freq1 = 554;
freq1 = 547;
Freq2 = 659;
Freq3 = 831;

% discovered soundsc() scales sound to fit -1 to 1 range
% no need to manually normalize

% remember: default sample rate is 8192 Hz
samplerate=8192;

t = [0:1: samplerate];

Db = sin(Freq1/samplerate*t*2*pi);
Db1 = sin(freq1/samplerate*t*2*pi);
E = sin(Freq2/samplerate*t*2*pi);
Ab = sin(Freq3/samplerate*t*2*pi);

chord = Db + E + Ab;
chord2 = Db1 + E + Ab;

soundsc(chord);
soundsc(chord2);
```

Published with MATLAB® 7.14

```
% test11_reference_list.m
% create a reference list of all notes A -> G
clc;
clear all;
samplerate = 8192;
nBits = 8;
t = [0:1: samplerate];

FreqA = 440;
FreqBb = 466;
FreqB = 494;
FreqC = 523;
FreqDb = 554;
FreqD = 587;
FreqEb = 622;
FreqE = 659;
FreqF = 699;
FreqGb = 740;
FreqG = 784;
FreqAb = 831;

A = sin(FreqA/samplerate*t*2*pi);
Bb = sin(FreqBb/samplerate*t*2*pi);
B = sin(FreqB/samplerate*t*2*pi);
C = sin(FreqC/samplerate*t*2*pi);
Db = sin(FreqDb/samplerate*t*2*pi);
D = sin(FreqD/samplerate*t*2*pi);
Eb = sin(FreqEb/samplerate*t*2*pi);
E = sin(FreqE/samplerate*t*2*pi);
F = sin(FreqF/samplerate*t*2*pi);
Gb = sin(FreqGb/samplerate*t*2*pi);
G = sin(FreqG/samplerate*t*2*pi);
Ab = sin(FreqAb/samplerate*t*2*pi);

Nmax = max(A&Bb&B&C&Db&D&Eb&E&F&Gb&G&Ab)
Nmin = min(A&Bb&B&C&Db&D&Eb&E&F&Gb&G&Ab)


chord = Db + E + Ab;
soundsc(chord, samplerate,nBits);
soundsc(A,samplerate,nBits);
soundsc(Bb,samplerate,nBits);
soundsc(B,samplerate,nBits);
soundsc(C,samplerate,nBits);
soundsc(Db,samplerate,nBits);
soundsc(D,samplerate,nBits);
soundsc(Eb,samplerate,nBits);
soundsc(E,samplerate,nBits);
soundsc(F,samplerate,nBits);
soundsc(Gb,samplerate,nBits);
soundsc(G,samplerate,nBits);
soundsc(Ab,samplerate,nBits);
```

```
csvwrite('note_A.dat',A);
csvwrite('note_Bb.dat',Bb);
csvwrite('note_B.dat',B);
csvwrite('note_C.dat',C);
csvwrite('note_Db.dat',Db);
csvwrite('note_D.dat',D);
csvwrite('note_Eb.dat',Eb);
csvwrite('note_E.dat',E);
csvwrite('note_F.dat',F);
csvwrite('note_Gb.dat',Gb);
csvwrite('note_G.dat',G);
csvwrite('note_Ab.dat',Ab);

csvwrite('samplerate.dat',samplerate);
csvwrite('nBits.dat',nBits);
```

```
Nmax =

     1


Nmin =

     0
```

Published with MATLAB® 7.14

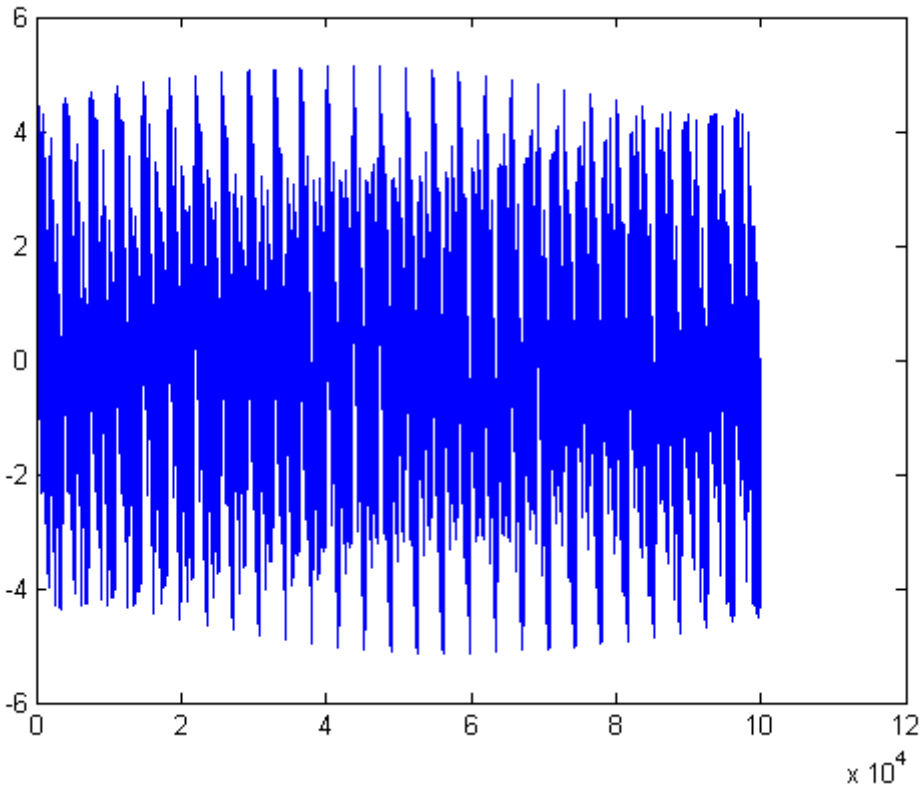
```
% test12_reference_list.m
% use reference list
clc;
clear all;

samplerate = csvread('samplerate.dat');
nBits = csvread('nBits.dat');

A = csvread('note_A.dat');
Bb = csvread('note_Bb.dat');
B = csvread('note_B.dat');
C = csvread('note_C.dat');
Db = csvread('note_Db.dat');
D = csvread('note_D.dat');
Eb = csvread('note_Eb.dat');
E = csvread('note_E.dat');
F = csvread('note_F.dat');
Gb = csvread('note_Gb.dat');
G = csvread('note_G.dat');
Ab = csvread('note_Ab.dat');

chord = Db + E + Ab;

soundsc(chord,samplerate,nBits);
```

Published with MATLAB® 7.14

```
% test13_prenormalize.m
%goal: repeat reference list with normalized initial sine waves
clc;
clear all;
samplerate = 8192;
nBits = 8;
t = [0:1: samplerate];

FreqA = 440;
FreqBb = 466;
FreqB = 494;
FreqC = 523;
FreqDb = 554;
FreqD = 587;
FreqEb = 622;
FreqE = 659;
FreqF = 699;
FreqGb = 740;
FreqG = 784;
FreqAb = 831;

A = sin(FreqA/samplerate*t*2*pi);
Bb = sin(FreqBb/samplerate*t*2*pi);
B = sin(FreqB/samplerate*t*2*pi);
C = sin(FreqC/samplerate*t*2*pi);
Db = sin(FreqDb/samplerate*t*2*pi);
D = sin(FreqD/samplerate*t*2*pi);
Eb = sin(FreqEb/samplerate*t*2*pi);
E = sin(FreqE/samplerate*t*2*pi);
F = sin(FreqF/samplerate*t*2*pi);
Gb = sin(FreqGb/samplerate*t*2*pi);
G = sin(FreqG/samplerate*t*2*pi);
Ab = sin(FreqAb/samplerate*t*2*pi);

csvwrite('note_A.dat',A);
csvwrite('note_Bb.dat',Bb);
csvwrite('note_B.dat',B);
csvwrite('note_C.dat',C);
csvwrite('note_Db.dat',Db);
csvwrite('note_D.dat',D);
csvwrite('note_Eb.dat',Eb);
csvwrite('note_E.dat',E);
csvwrite('note_F.dat',F);
csvwrite('note_Gb.dat',Gb);
csvwrite('note_G.dat',G);
csvwrite('note_Ab.dat',Ab);

csvwrite('samplerate.dat',samplerate);
csvwrite('nBits.dat',nBits);
```

Published with MATLAB® 7.14

```
% test14_harmonic.m
% goal: make a harmonic and test frequency shift

clc;
clear all;

samplerate = 100000;
nBits = 8;
t = [0:1: samplerate];
A1 = sin(28/samplerate*t*2*pi);
A2 = sin(55/samplerate*t*2*pi);
A3 = sin(110/samplerate*t*2*pi);
A4 = sin(220/samplerate*t*2*pi);
A5 = sin(440/samplerate*t*2*pi);
A6 = sin(880/samplerate*t*2*pi);
A7 = sin(1760/samplerate*t*2*pi);
A8 = sin(3520/samplerate*t*2*pi);

harmonic = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8;
%soundsc(harmonic, samplerate, nBits);
plot(harmonic);
```

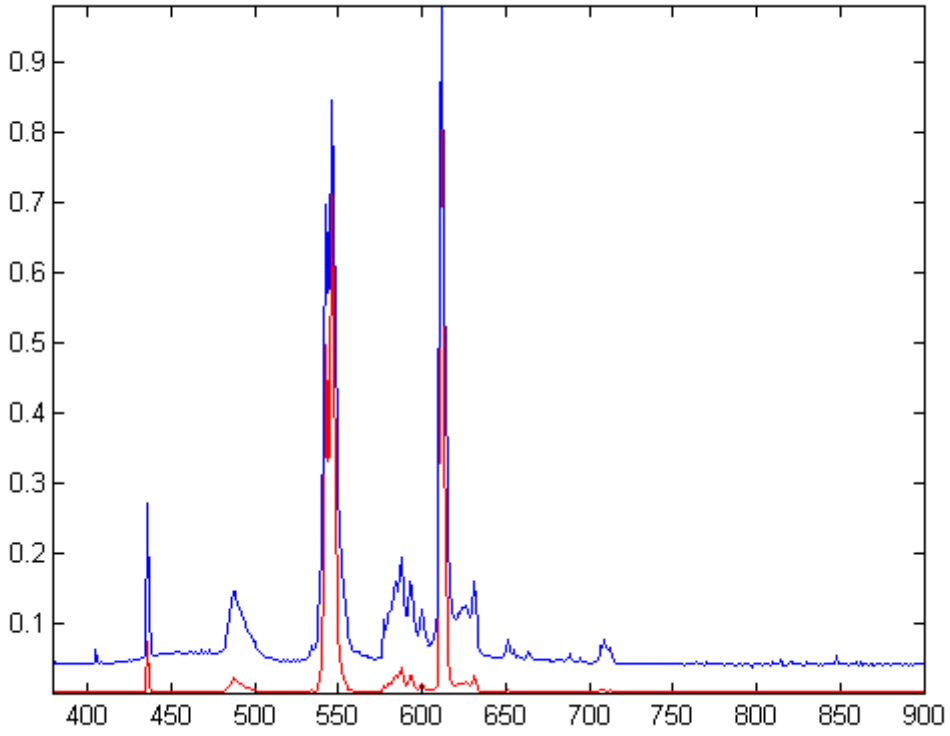


Published with MATLAB® 7.14

```
% test15_tonec.m
% goal: test tonec.m

csvwrite('tonectest.dat', tonec(440,1,10000));
sound(tonec(440,1,10000));
soundsc(tonec([28,55,110,220,440,880,1760,3520],1,10000));
```

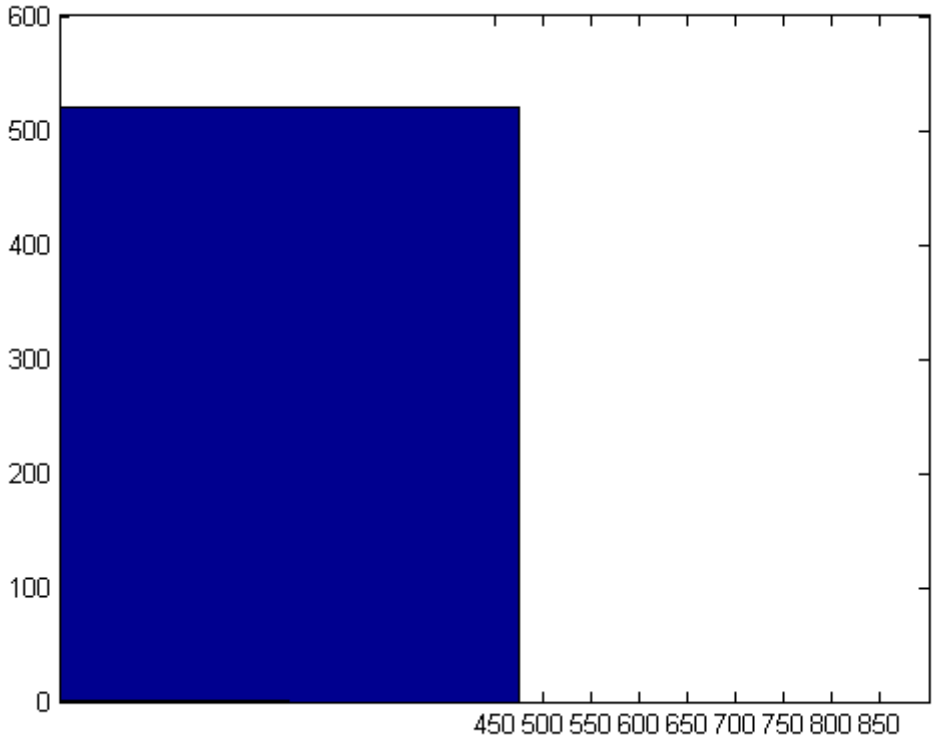
Published with MATLAB® 7.14

```
% test16_threshold.m
% main goal: begin to develop intelligent binning system for input data
% goal: determine signal/noise threshold in spectral data

spectrum_in = dlmread('dec3a.txt', '\t', 1,1);
all = dlmread('dec3a.txt', '\t', 1, 0);
t = all(1:end,1);
noiseless = spectrum_in.*abs(spectrum_in)/max(abs(spectrum_in));
plot(t,noiseless,'r-',t,spectrum_in,'b-');
axis tight;
```



Published with MATLAB® 7.14

```
% test17_hist.m
% main goal: begin to develop intelligent binning system for input data
% goal: investigate binning of data using hist command

spectrum_in = dlmread('dec3a.txt', '\t', 1,1);
all = dlmread('dec3a.txt', '\t', 1, 0);
t = all(1:end,1);
noiseless = spectrum_in.*abs(spectrum_in)/max(abs(spectrum_in));

bin_locations = [450,500,550,600,650,700,750,800,850];


hist(noiseless,bin_locations);
```

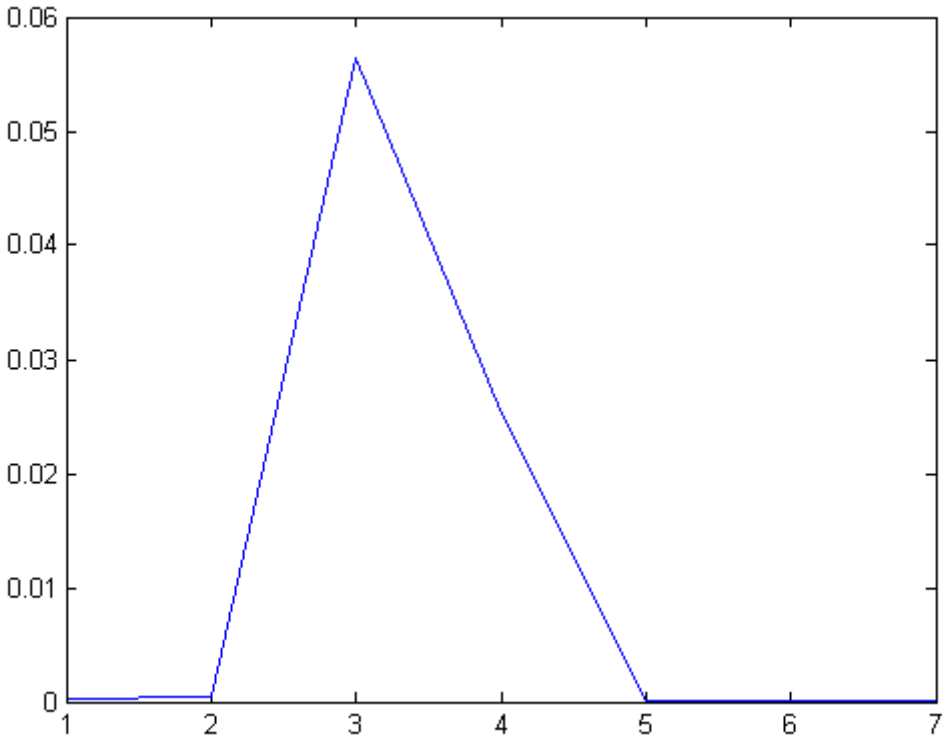


Published with MATLAB® 7.14

```
% test18_for.m
% main goal: begin to develop intelligent binning system for input data
% goal: investigate binning of data using iterative for loop

clc;
clear all;

spectrum_in = dlmread('dec3a.txt', '\t', 1,1);
all = dlmread('dec3a.txt', '\t', 1, 0);
t = all(1:end,1);
noiseless = spectrum_in.*abs(spectrum_in)/max(abs(spectrum_in));

[M,N] = size(t)

if M < 700

    bin_num = 7;

else

    bin_num = 14;

end

bin = round(M./bin_num)
k = 0;
for i = 1:bin_num
    finale_array(i,1) = i;
    index=i-1;
    k = index*bin+1;
    finale_array(i,2) = mean(noiseless(k:(k+bin)));
end

noiseless_finale(:,1) = 1:bin_num;
noiseless_finale(:,2) = finale_array(:,2).*abs(finale_array(:,2))...
    /max(abs(finale_array(:,2)));

plot(noiseless_finale(1:7,1),noiseless_finale(1:7,2))
```

```
M =

   521


N =

     1


bin =
```

74



Published with MATLAB® 7.14

```
% test19_datatosound.m
% goal: create sound using tonec from input data

clc;
clear all;

spectrum_in = dlmread('dec3a.txt', '\t', 1,1);
all = dlmread('dec3a.txt', '\t', 1, 0);
t = all(1:end,1);
noiseless = spectrum_in.*abs(spectrum_in)/max(abs(spectrum_in));

[M,N] = size(t)

if M < 700

    bin_num = 7;
    notes = [294, 330, 349, 392, 440, 466, 523]'

else

    bin_num = 14;
    notes = [294, 330, 349, 392, 440, 466, 523, 587, 659,...
        699, 784, 880, 932, 1047]'


end

bin = round(M./bin_num)
k = 0;
for i = 1:bin_num
    final_array(i,1) = i;
    index=i-1;
    k = index*bin+1;
    final_array(i,2) = mean(noiseless(k:(k+bin)));
end

noiseless_final(:,1) = 1:bin_num;
noiseless_final(:,2) = final_array(:,2).*abs(final_array(:,2))...
    /max(abs(final_array(:,2)));

soundsc(tonec([notes,final_array(:,2)], 1, 10000));
soundsc(tonec([notes,noiseless_final(:,2)], 1, 10000));
```

```
M =

   521


N =

     1
```

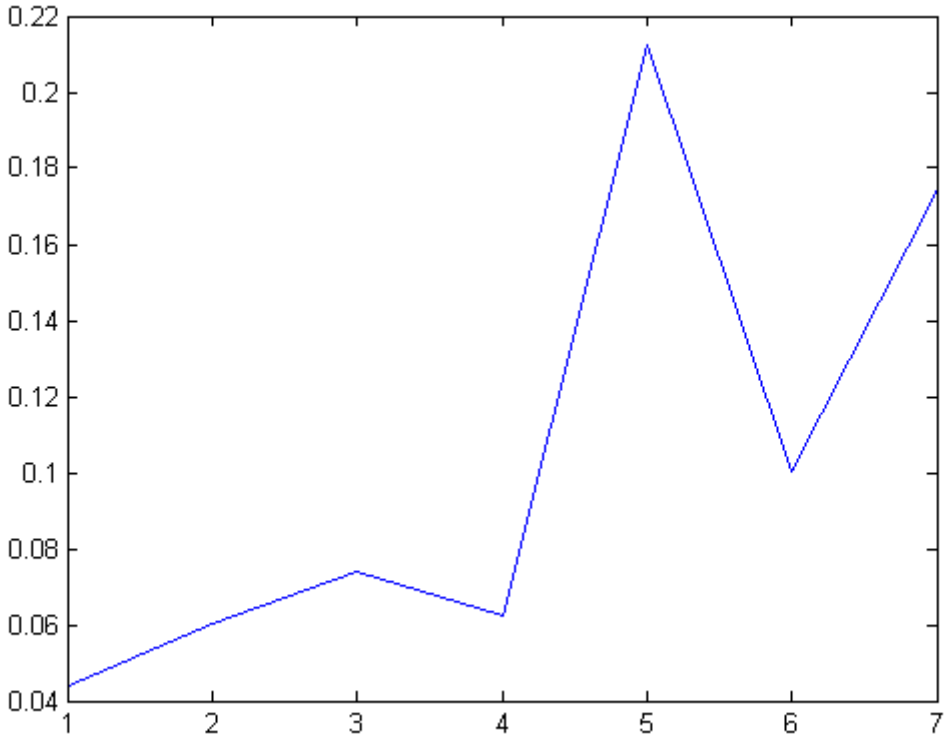
```
notes =

   294
   330
   349
   392
   440
   466
   523


bin =

    74
```

Published with MATLAB® 7.14

```
% test20_nobackgroundreduction.m
% goal: create sound using tonec from input data

clc;
clear all;

spectrum_in = dlmread('dec3a.txt', '\t', 1,1);
all = dlmread('dec3a.txt', '\t', 1, 0);
t = all(1:end,1);

[M,N] = size(t)

if M < 200

    bin_num = 7;
    notes = [294, 330, 349, 392, 440, 466, 523]'

else

    bin_num = 14;
    notes = [294, 330, 349, 392, 440, 466, 523, 587, 659,...
        699, 784, 880, 932, 1047]'


end

bin = round(M./bin_num)
k = 0;
for i = 1:bin_num
    final_array(i,1) = i;
    index=i-1;
    k = index*bin+1;
    final_array(i,2) = mean(spectrum_in(k:(k+bin)));
end

soundsc(tonec([notes,final_array(:,2)], 1, 10000));
plot(final_array(1:7,1), final_array(1:7,2));
```

```
M =

   521


N =

     1


notes =

         294
```

```
   330

   349
   392
   440
   466
   523
   587
   659
   699
   784
   880
   932
  1047

bin =

    37
```



Published with MATLAB® 7.14

```
% test21_properbackgroundreduction.m
% goal: create sound using tonec from input data

clc;
clear all;

spectrum_in = dlmread('Spectrometer Data\jan24e.txt', '\t', 1,1);
all = dlmread('Spectrometer Data\jan24e.txt', '\t', 1, 0);
t = all(1:end,1);

background = dlmread('Spectrometer Data\jan24 background.txt', '\t', 1, 1);

a=find(background, t(1));
b=find(background, t(end));

background_adj = background(a:b);

spectrum_out = spectrum_in - background_adj;

[M,N] = size(t)

if M < 200

    bin_num = 7;
    notes = [294, 330, 349, 392, 440, 466, 523]'

else

    bin_num = 14;
    notes = [294, 330, 349, 392, 440, 466, 523, 587, 659,...
        699, 784, 880, 932, 1047]'


end

bin = round(M./bin_num)
k = 0;
for i = 1:bin_num
    final_array(i,1) = i;
    index=i-1;
    k = index*bin+1;
    if  k+bin < size(spectrum_out)
        final_array(i,2) = mean(spectrum_out(k:(k+bin)));
    else
        final_array(i,2) = mean(spectrum_out(k:end));
    end
end

soundsc(tonec([notes,final_array(:,2)], 3, 10000));
%plot(final_array(1:bin_num,1), final_array(1:bin_num,2));

%goal: create sound using tonec from input data
```

```
clc;
clear all;

spectrum_in = dlmread('Spectrometer Data\jan24f.txt', '\t', 1,1);
all = dlmread('Spectrometer Data\jan24f.txt', '\t', 1, 0);
t = all(1:end,1);

background = dlmread('Spectrometer Data\jan24 background.txt', '\t', 1, 1);

a=find(background, t(1));
b=find(background, t(end));

background_adj = background(a:b);

spectrum_out = spectrum_in - background_adj;

[M,N] = size(t)

if M < 200

    bin_num = 7;
    notes = [294, 330, 349, 392, 440, 466, 523]'

else

    bin_num = 14;
    notes = [294, 330, 349, 392, 440, 466, 523, 587, 659,...
        699, 784, 880, 932, 1047]'


end

bin = round(M./bin_num)
k = 0;
for i = 1:bin_num
    final_array(i,1) = i;
    index=i-1;
    k = index*bin+1;
    if  k+bin < size(spectrum_out)
        final_array(i,2) = mean(spectrum_out(k:(k+bin)));
    else
        final_array(i,2) = mean(spectrum_out(k:end));
    end
end

soundsc(tonec([notes,final_array(:,2)], 3, 10000));
%plot(final_array(1:bin_num,1), final_array(1:bin_num,2));

%goal: create sound using tonec from input data

clc;
clear all;
```

```
spectrum_in = dlmread('Spectrometer Data\jan24h.txt', '\t', 1,1);
all = dlmread('Spectrometer Data\jan24h.txt', '\t', 1, 0);
t = all(1:end,1);

background = dlmread('Spectrometer Data\jan24 background.txt', '\t', 1, 1);

a=find(background, t(1));
b=find(background, t(end));

background_adj = background(a:b);

spectrum_out = spectrum_in - background_adj;

[M,N] = size(t)

if M < 200

    bin_num = 7;
    notes = [294, 330, 349, 392, 440, 466, 523]'

else

    bin_num = 14;
    notes = [294, 330, 349, 392, 440, 466, 523, 587, 659,...
        699, 784, 880, 932, 1047]'


end

bin = round(M./bin_num)
k = 0;
for i = 1:bin_num
    final_array(i,1) = i;
    index=i-1;
    k = index*bin+1;
    if  k+bin < size(spectrum_out)
        final_array(i,2) = mean(spectrum_out(k:(k+bin)));
    else
        final_array(i,2) = mean(spectrum_out(k:end));
    end
end

soundsc(tonec([notes,final_array(:,2)], 3, 10000));
%plot(final_array(1:bin_num,1), final_array(1:bin_num,2));


        M =

           651


        N =

             1
```

```
notes =

         294
         330
         349
         392
         440
         466
         523
         587
         659
         699
         784
         880
         932
        1047


bin =

    47


M =

   651


N =

     1


notes =

         294
         330
         349
         392
         440
         466
         523
         587
         659
         699
         784
         880
         932
        1047


bin =
```

```
    47

M =

   651

N =

     1

notes =

         294
         330
         349
         392
         440
         466
         523
         587
         659
         699
         784
         880
         932
        1047

bin =

    47
```

Published with MATLAB® 7.14

```
% test22_staincomparison.m
% goal: create sound using tonec from input data

clc;
clear all;

sensitivity = 5;

spectrum_in = dlmread('Spectrometer Data\jan31f.txt', '\t', 1,1);
all = dlmread('Spectrometer Data\jan31f.txt', '\t', 1, 0);
t = all(1:end,1);

background = dlmread('Spectrometer Data\jan24 background.txt', '\t', 1, 1);

a=find(background, t(1));
b=find(background, t(end));

background_adj = background(a:b);

spectrum_out = spectrum_in - background_adj;

[M,N] = size(t)

if M < 400

    bin_num = 12;
    notes = [73, 87, 110, 147, 175, 220, 294, 349, 440, 587, 699, 880]'

else

    bin_num = 24;
    notes = [37, 44, 55, 73, 87, 110, 147, 175, 220, 294, 349, 440, 587,...
        699, 880, 1175, 1397, 1760, 2349, 2794, 3520, 4699, 5588, 7040]'


end

bin = round(M./bin_num)
k = 0;
final_array1 = zeros(2,bin_num);

for i = 1:bin_num
    final_array1(i,1) = i;
    index=i-1;
    k = index*bin+1;
    if  k+bin < size(spectrum_out)
        final_array1(i,2) = mean(spectrum_out(k:(k+bin)));
    else
        final_array1(i,2) = mean(spectrum_out(k:end));
    end
end
```

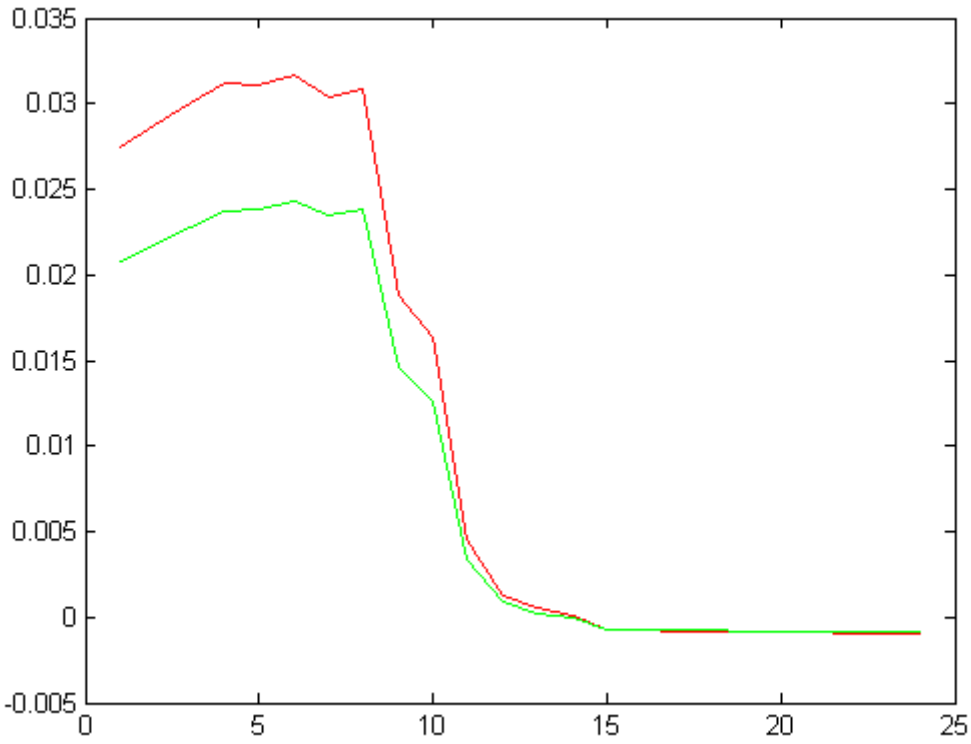
```
clc;
clear all;

spectrum_in = dlmread('Spectrometer Data\jan31g.txt', '\t', 1,1);
all = dlmread('Spectrometer Data\jan31g.txt', '\t', 1, 0);
t = all(1:end,1);

background = dlmread('Spectrometer Data\jan24 background.txt', '\t', 1, 1);

a=find(background, t(1));
b=find(background, t(end));

background_adj = background(a:b);

spectrum_out = spectrum_in - background_adj;

[M,N] = size(t)

if M < 200

    bin_num = 7;
    notes = [294, 330, 349, 392, 440, 466, 523]'

else

    bin_num = 24;
    notes = [37, 44, 55, 73, 87, 110, 147, 175, 220, 294, 349, 440, 587,...
        699, 880, 1175, 1397, 1760, 2349, 2794, 3520, 4699, 5588, 7040]'


end

bin = round(M./bin_num)
k = 0;
final_array2 = zeros(2,bin_num);
for i = 1:bin_num
    final_array2(i,1) = i;
    index=i-1;
    k = index*bin+1;
    if  k+bin < size(spectrum_out)
        final_array2(i,2) = mean(spectrum_out(k:(k+bin)));
    else
        final_array2(i,2) = mean(spectrum_out(k:end));
    end
end

weight = 1 - 10*final_array1(:,2) + 10*final_array2(:,2);
weighted = weight .* notes


soundsc(tonec([notes,final_array1(:,2)], 1, 10000));
soundsc(tonec([weighted,final_array2(:,2)], 1, 10000));
plot(final_array1(:,1),final_array1(:,2), 'r');
hold;
```

```
plot(final_array2(:,1),final_array2(:,2), 'g');
```

```
M =

   651


N =

     1


notes =

          37
          44
          55
          73
          87
         110
         147
         175
         220
         294
         349
         440
         587
         699
         880
        1175
        1397
        1760
        2349
        2794
        3520
        4699
        5588
        7040


bin =

    27


M =

   651


N =

     1
```

```
notes =

          37
          44
          55
          73
          87
         110
         147
         175
         220
         294
         349
         440
         587
         699
         880
        1175
        1397
        1760
        2349
        2794
        3520
        4699
        5588
        7040


bin =

    27


weighted =

   1.0e+03 *

    0.0345
    0.0410
    0.0510
    0.0675
    0.0806
    0.1019
    0.1368
    0.1627
    0.2107
    0.2832
    0.3452
    0.4381
    0.5855
    0.6978
    0.8800
```

```
    1.1751

    1.3973
    1.7605
    2.3501
    2.7956
    3.5221
    4.7022
    5.5913
    7.0453

Current plot held
```



Published with MATLAB® 7.14

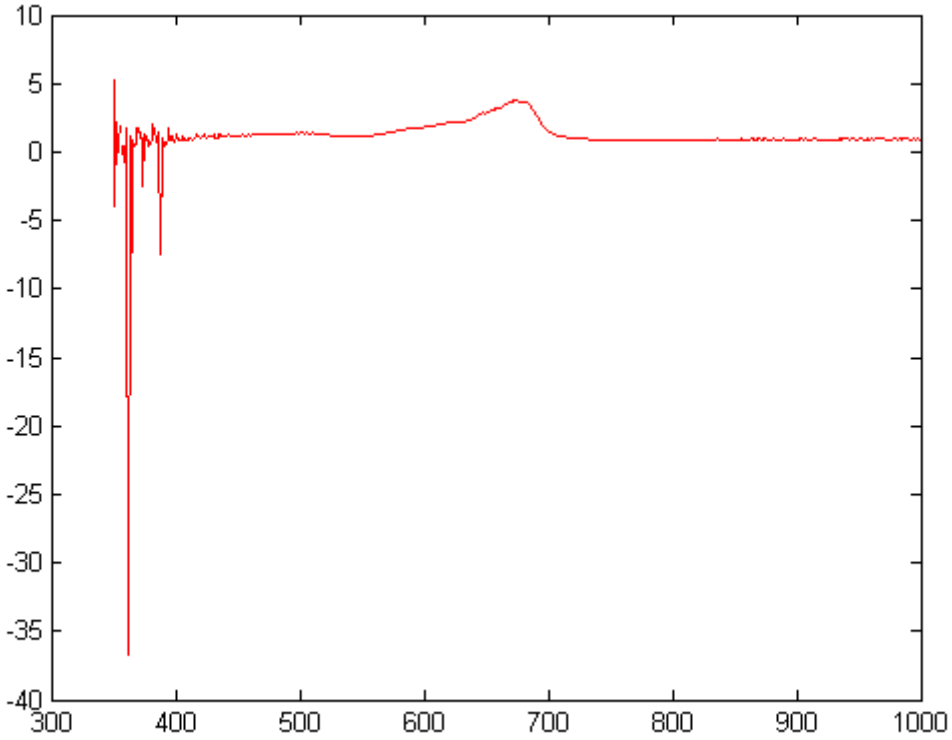
```
% test23_weighting.m
% goal: devise proper weighting system

% clear ALL the things
clc;
clear all;

% manually set sensitivity to differences between input spectrum and
% expected spectrum
% sensitivity = 5;

% calls file chosen as expected value for comparison with dynamic
% readings
% creates individual arrays for each column
spectrum_in1 = dlmread('Spectrometer Data\feb18a.txt', '\t', 1,1);
all = dlmread('Spectrometer Data\feb18a.txt', '\t', 1, 0);
t = all(1:end,1);

% this block establishes the dark readings of the spectrometer and removes
% the background from the spectrum
background = dlmread('Spectrometer Data\feb18_background.txt', '\t', 1, 1);

a=find(background, t(1));
b=find(background, t(end));
background_adj = background(a:b);

spectrum_out1 = spectrum_in1 - background_adj;

% determines size of spectrum and sets size of audio spectrum
[M,N] = size(t)

if M < 400
    bin_num = 12;
    notes = [73, 93, 110, 147, 185, 220, 294, 370, 440, 587, 740, 880]'

else
    bin_num = 24;
    notes = [37, 46, 55, 73, 93, 110, 147, 185, 220, 294, 370, 440, 587,...
        740, 880, 1175, 1475, 1760, 2349, 2960, 3520, 4699, 5920, 7040]'
end

% sets bin size, initializes vectors
bin = round(M./bin_num)
k = 0;
final_array1 = zeros(2,bin_num);

% creates binned array with average values of input spectrum
% this array corresponds to the ideal input spectrum
for i = 1:bin_num
    final_array1(i,1) = i;
    index=i-1;
    k = index*bin+1;
```

```
    if  k+bin < size(spectrum_out1)
        final_array1(i,2) = mean(spectrum_out1(k:(k+bin)));
    else
        final_array1(i,2) = mean(spectrum_out1(k:end));
    end
end

q = 1;
while q;

    % calls file given once per second by the spectrometer
    % spectrometer needs to always output same file name
    spectrum_in2 = dlmread('Spectrometer Data\feb18b.txt', '\t', 1,1);
    spectrum_out2 = spectrum_in2 - background_adj;

    % creates binned array with average values of input spectrum
    % this array corresponds to the dynamic input spectrum
    k = 0;
    final_array2 = zeros(2,bin_num);
    for i = 1:bin_num
        final_array2(i,1) = i;
        index=i-1;
        k = index*bin+1;
        if  k+bin < size(spectrum_out2)
            final_array2(i,2) = mean(spectrum_out2(k:(k+bin)));
        else
            final_array2(i,2) = mean(spectrum_out2(k:end));
        end
    end

    % introduces frequency shift where the dynamic spectrum differs from
    % the ideal spectrum
    %weight = zeros(size(notes));


    ratio = final_array2(:,2) ./ final_array1(:,2)

    weight = 1 + ratio(:) - mean(ratio(:));

    weighted = weight .* notes

    testing = spectrum_out2(:,1) ./ spectrum_out1(:,1);

    plot(t(:,1),testing(:,1), 'r');
    %hold;
    %plot(1:bin_num, weight, 'b');

    %plot(t(:,1),spectrum_out1(:,1),'r');
    %hold;
    %plot(t(:,1),spectrum_out2(:,1),'b');
```

```
    % stops while loop so I don't crash matlab again
    q = q - 1;
end
```

```
M =

   651


N =

     1


notes =

          37
          46
          55
          73
          93
         110
         147
         185
         220
         294
         370
         440
         587
         740
         880
        1175
        1475
        1760
        2349
        2960
        3520
        4699
        5920
        7040


bin =

    27


ratio =

    1.3295
    1.3295
    1.3296
    1.3300
```

```
    1.3304
    1.3306
    1.3301
    1.3356
    1.3479
    1.3303
    1.2756
    1.1870
    1.0657
    0.9540
    0.9187
    0.9191
    0.9241
    0.9282
    0.9313
    0.9335
    0.9354
    0.9372
    0.9485
    0.9455


weighted =

   1.0e+03 *

    0.0444
    0.0552
    0.0660
    0.0876
    0.1116
    0.1320
    0.1764
    0.2230
    0.2679
    0.3528
    0.4238
    0.4649
    0.5491
    0.6096
    0.6938
    0.9268
    1.1708
    1.4042
    1.8815
    2.3773
    2.8339
    3.7917
    4.8437
    5.7391
```



Published with MATLAB® 7.14

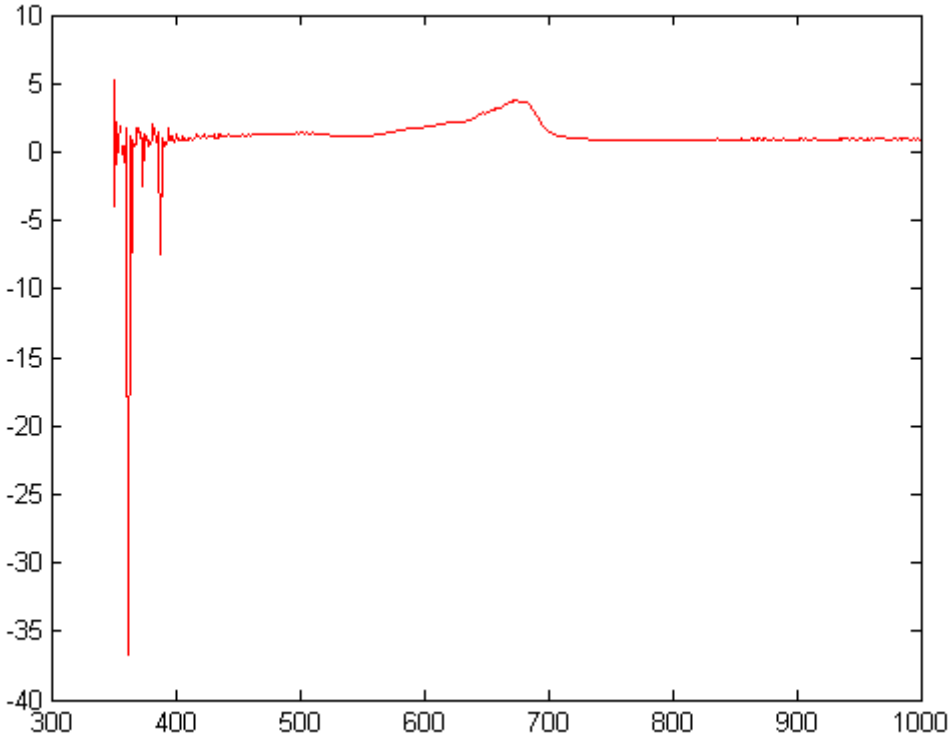
```
% test24_newweight.m
% goal: devise proper weighting system

% clear ALL the things
clc;
clear all;

% manually set sensitivity to differences between input spectrum and
% expected spectrum
% sensitivity = 5;

% calls file chosen as expected value for comparison with dynamic
% readings
% creates individual arrays for each column
spectrum_in1 = dlmread('Spectrometer Data\feb18a.txt', '\t', 1,1);
all = dlmread('Spectrometer Data\feb18a.txt', '\t', 1, 0);
t = all(1:end,1);

% this block establishes the dark readings of the spectrometer and removes
% the background from the spectrum
background = dlmread('Spectrometer Data\feb18_background.txt', '\t', 1, 1);

a=find(background, t(1));
b=find(background, t(end));
background_adj = background(a:b);

spectrum_out1 = spectrum_in1 - background_adj;

% determines size of spectrum and sets size of audio spectrum
[M,N] = size(t)

if M < 400
    bin_num = 12;
    notes = [73, 93, 110, 147, 185, 220, 294, 370, 440, 587, 740, 880]'

else
    bin_num = 24;
    notes = [37, 46, 55, 73, 93, 110, 147, 185, 220, 294, 370, 440, 587,...
        740, 880, 1175, 1475, 1760, 2349, 2960, 3520, 4699, 5920, 7040]'
end

% sets bin size, initializes vectors
bin = round(M./bin_num)
k = 0;
final_array1 = zeros(2,bin_num);

% creates binned array with average values of input spectrum
% this array corresponds to the ideal input spectrum
for i = 1:bin_num
    final_array1(i,1) = i;
    index=i-1;
    k = index*bin+1;
```

```
    if  k+bin < size(spectrum_out1)
        final_array1(i,2) = mean(spectrum_out1(k:(k+bin)));
    else
        final_array1(i,2) = mean(spectrum_out1(k:end));
    end
end

q = 1;
while q;

    % calls file given once per second by the spectrometer
    % spectrometer needs to always output same file name
    spectrum_in2 = dlmread('Spectrometer Data\feb18b.txt', '\t', 1,1);
    spectrum_out2 = spectrum_in2 - background_adj;

    % creates binned array with average values of input spectrum
    % this array corresponds to the dynamic input spectrum
    k = 0;
    final_array2 = zeros(2,bin_num);
    for i = 1:bin_num
        final_array2(i,1) = i;
        index=i-1;
        k = index*bin+1;
        if  k+bin < size(spectrum_out2)
            final_array2(i,2) = mean(spectrum_out2(k:(k+bin)));
        else
            final_array2(i,2) = mean(spectrum_out2(k:end));
        end
    end

    % introduces frequency shift where the dynamic spectrum differs from
    % the ideal spectrum
    %weight = zeros(size(notes));


    ratio = final_array2(:,2) ./ final_array1(:,2)

    weight = mean(ratio(:)) ./ ratio(:);

    weighted = weight .* notes

    testing = spectrum_out2(:,1) ./ spectrum_out1(:,1);

    plot(t(:,1),testing(:,1), 'r');
    %hold;
    %plot(1:bin_num, weight, 'b');

    %plot(t(:,1),spectrum_out1(:,1),'r');
    %hold;
    %plot(t(:,1),spectrum_out2(:,1),'b');
```

```
    % stops while loop so I don't crash matlab again
    q = q - 1;
end
```

```
M =

   651


N =

     1


notes =

          37
          46
          55
          73
          93
         110
         147
         185
         220
         294
         370
         440
         587
         740
         880
        1175
        1475
        1760
        2349
        2960
        3520
        4699
        5920
        7040


bin =

    27


ratio =

    1.3295
    1.3295
    1.3296
    1.3300
```

```
    1.3304
    1.3306
    1.3301
    1.3356
    1.3479
    1.3303
    1.2756
    1.1870
    1.0657
    0.9540
    0.9187
    0.9191
    0.9241
    0.9282
    0.9313
    0.9335
    0.9354
    0.9372
    0.9485
    0.9455


weighted =

   1.0e+03 *

    0.0315
    0.0391
    0.0468
    0.0620
    0.0790
    0.0934
    0.1249
    0.1566
    0.1845
    0.2498
    0.3278
    0.4190
    0.6226
    0.8767
    1.0827
    1.4451
    1.8042
    2.1433
    2.8509
    3.5842
    4.2535
    5.6671
    7.0547
    8.4159
```



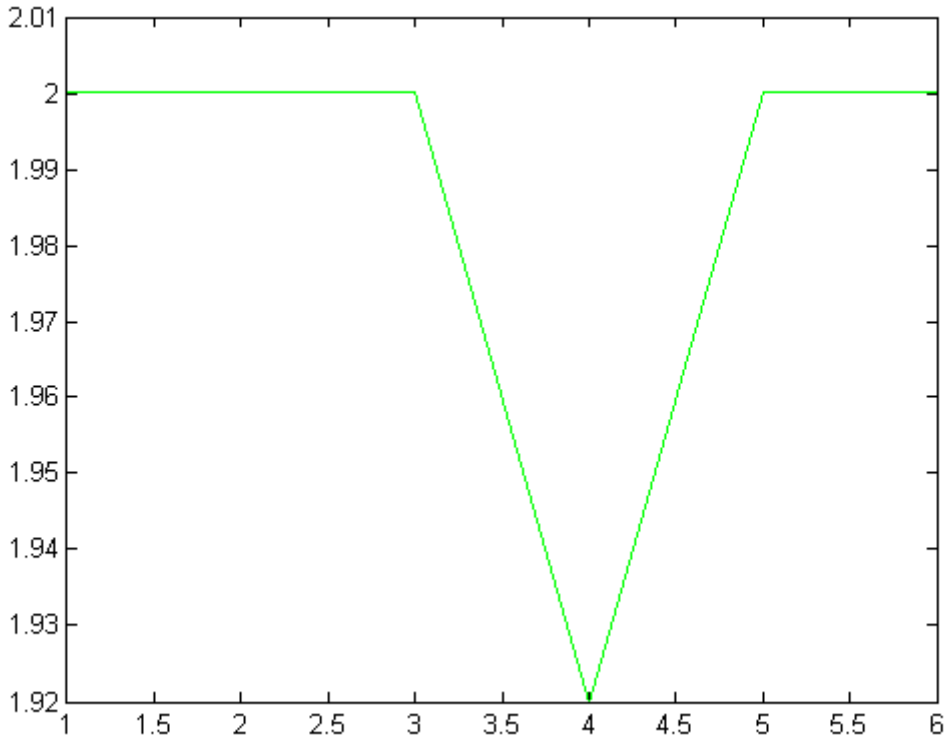
Published with MATLAB® 7.14

```
% test25_testarrays.m
% goal: usable final product!!!!

% clear ALL the things
clc;
clear all;

% manually set sensitivity to differences between input spectrum and
% expected spectrum
% sensitivity = 5;

% calls file chosen as expected value for comparison with dynamic
% readings
% creates individual arrays for each column
% spectrum_in1 = dlmread('Spectrometer Data\feb18a.txt', '\t', 1,1);
% all = dlmread('Spectrometer Data\feb18a.txt', '\t', 1, 0);
% t = all(1:end,1);

spectrum_in1 = [5,9,11,25,9,5];

% this block establishes the dark readings of the spectrometer and removes
% the background from the spectrum
% background = dlmread('Spectrometer Data\feb18_background.txt',...
% '\t', 1, 1);
%
% a=find(background, t(1));
% b=find(background, t(end));
% background_adj = background(a:b);

background_adj = [1,1,1,1,1,1];

spectrum_out1 = spectrum_in1 - background_adj;

% determines size of spectrum and sets size of audio spectrum
% [M,N] = size(t)
%
% if M < 400
%     bin_num = 12;
%     notes = [73, 93, 110, 147, 185, 220, 294, 370, 440, 587, 740, 880]'
%
% else
%     bin_num = 24;
%     notes = [37, 46, 55, 73, 93, 110, 147, 185, 220, 294, 370, 440,...
% 587, 740, 880, 1175, 1475, 1760, 2349, 2960, 3520, 4699, 5920, 7040]'
% end
bin_num = 6;
notes = [294, 370, 440, 587, 740, 880]';

% sets bin size, initializes vectors
% bin = round(M./bin_num)
% k = 0;
% final_array1 = zeros(2,bin_num);
```

```
% creates binned array with average values of input spectrum
% this array corresponds to the ideal input spectrum
% for i = 1:bin_num
%     final_array1(i,1) = i;
%     index=i-1;
%     k = index*bin+1;
%     if  k+bin < size(spectrum_out1)
%         final_array1(i,2) = mean(spectrum_out1(k:(k+bin)));
%     else
%         final_array1(i,2) = mean(spectrum_out1(k:end));
%     end
% end

% plays D major chord with harmonics varying based on signal
soundsc(tonec([notes,spectrum_out1(:)], 1, 10000));

q = 1;
while q;

    % calls file given once per second by the spectrometer
    % spectrometer needs to always output same file name
    % spectrum_in2 = dlmread('Spectrometer Data\feb18b.txt', '\t', 1,1);

    spectrum_in2 = [3,5,6,13.5,5,3];

    spectrum_out2 = spectrum_in2 - background_adj;

    % creates binned array with average values of input spectrum
    % this array corresponds to the dynamic input spectrum
%     k = 0;
%     final_array2 = zeros(2,bin_num);
%     for i = 1:bin_num
%         final_array2(i,1) = i;
%         index=i-1;
%         k = index*bin+1;
%         if  k+bin < size(spectrum_out2)
%             final_array2(i,2) = mean(spectrum_out2(k:(k+bin)));
%         else
%             final_array2(i,2) = mean(spectrum_out2(k:end));
%         end
%     end

    % introduces frequency shift where the dynamic spectrum differs from
    % the ideal spectrum

    ratio = spectrum_out2(:) ./ spectrum_out1(:)

    weight = mean(ratio(:)) ./ ratio(:)

    weightednotes = weight .* notes

    % plays D major chord with harmonics varying based on signal and with
    % frequency shift introduced to provide audible warning of differences
```

```
        % between the dynamic and ideal spectra
        soundsc(tonec([weightednotes,spectrum_out2(:)], 1, 10000));

        % plots graphs of ideal vs dynamic spectrum so I know what the heck's
        % going on
        testing = spectrum_out1(:) ./ spectrum_out2(:)

        plot(testing(:), 'g');

        % stops while loop so I don't crash matlab again
        q = q - 1;
    end
```

```
    ratio =

        0.5000
        0.5000
        0.5000
        0.5208
        0.5000
        0.5000


    weight =

        1.0069
        1.0069
        1.0069
        0.9667
        1.0069
        1.0069


    weightednotes =

      296.0417
      372.5694
      443.0556
      567.4333
      745.1389
      886.1111


    testing =

        2.0000
        2.0000
        2.0000
        1.9200
        2.0000
        2.0000
```



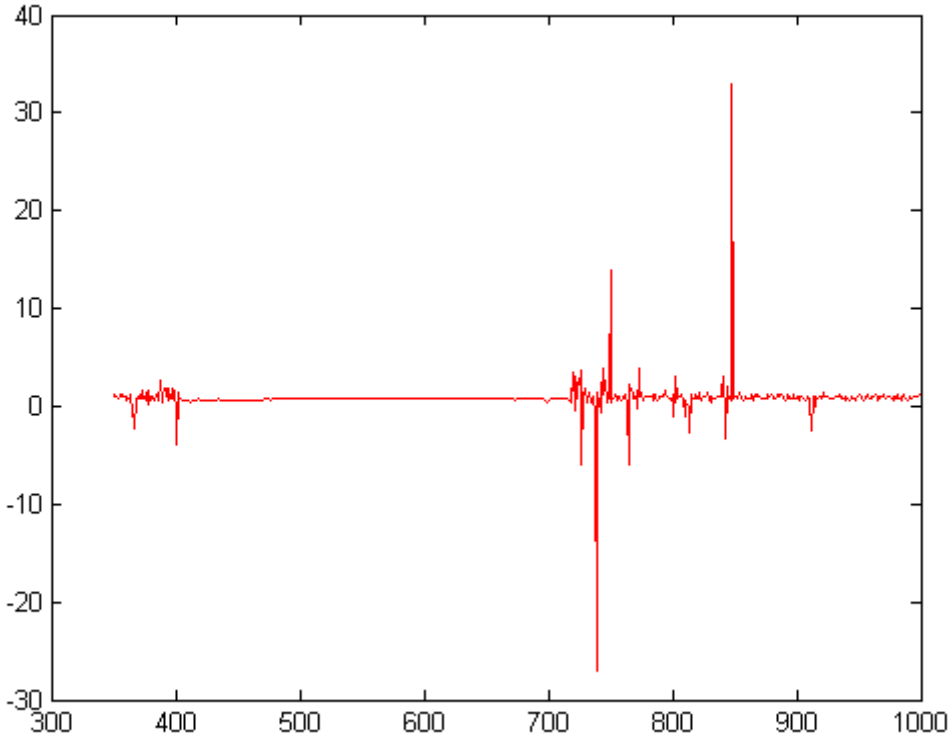
Published with MATLAB® 7.14

```
% final1.m
% goal: usable final product!!!!

% clear ALL the things
clc;
clear all;

% manually set sensitivity to differences between input spectrum and
% expected spectrum
sensitivity = 5;

% calls file chosen as expected value for comparison with dynamic
% readings
% creates individual arrays for each column
spectrum_in1 = dlmread('Spectrometer Data\jan31f.txt', '\t', 1,1);
all = dlmread('Spectrometer Data\jan31f.txt', '\t', 1, 0);
t = all(1:end,1);

% this block establishes the dark readings of the spectrometer and removes
% the background from the spectrum
background = dlmread('Spectrometer Data\jan24 background.txt', '\t', 1, 1);

a=find(background, t(1));
b=find(background, t(end));
background_adj = background(a:b);

spectrum_out1 = spectrum_in1 - background_adj;

% determines size of spectrum and sets size of audio spectrum
[M,N] = size(t)

if M < 400
    bin_num = 12;
    notes = [73, 93, 110, 147, 185, 220, 294, 370, 440, 587, 740, 880]'

else
    bin_num = 24;
    notes = [37, 46, 55, 73, 93, 110, 147, 185, 220, 294, 370, 440, 587,...
        740, 880, 1175, 1475, 1760, 2349, 2960, 3520, 4699, 5920, 7040]'
end

% sets bin size, initializes vectors
bin = round(M./bin_num)
k = 0;
final_array1 = zeros(2,bin_num);

% creates binned array with average values of input spectrum
% this array corresponds to the ideal input spectrum
for i = 1:bin_num
    final_array1(i,1) = i;
    index=i-1;
    k = index*bin+1;
```

```
    if  k+bin < size(spectrum_out1)
        final_array1(i,2) = mean(spectrum_out1(k:(k+bin)));
    else
        final_array1(i,2) = mean(spectrum_out1(k:end));
    end
end

% plays D major chord with harmonics varying based on signal
soundsc(tonec([notes,final_array1(:,2)], 1, 10000));

q = 1;
while q;

    % calls file given once per second by the spectrometer
    % spectrometer needs to always output same file name
    spectrum_in2 = dlmread('Spectrometer Data\jan31g.txt', '\t', 1,1);
    spectrum_out2 = spectrum_in2 - background_adj;

    % creates binned array with average values of input spectrum
    % this array corresponds to the dynamic input spectrum
    k = 0;
    final_array2 = zeros(2,bin_num);
    for i = 1:bin_num
        final_array2(i,1) = i;
        index=i-1;
        k = index*bin+1;
        if  k+bin < size(spectrum_out2)
            final_array2(i,2) = mean(spectrum_out2(k:(k+bin)));
        else
            final_array2(i,2) = mean(spectrum_out2(k:end));
        end
    end

    % introduces frequency shift where the dynamic spectrum differs from
    % the ideal spectrum
    weight = 1-sensitivity*final_array1(:,2)+sensitivity*final_array2(:,2)
    weighted = weight .* notes

    % plays D major chord with harmonics varying based on signal and with
    % frequency shift introduced to provide audible warning of differences
    % between the dynamic and ideal spectra
    soundsc(tonec([weighted,final_array2(:,2)], 1, 10000));

    % plots graphs of ideal vs dynamic spectrum so I know what's
    % going on
    testing = spectrum_out2(:,1) ./ spectrum_out1(:,1);

    plot(t(:,1),testing(:,1), 'r');

    % stops while loop so I don't crash matlab again
    q = q - 1;
end
```

```
M =

   651


N =

     1


notes =

          37
          46
          55
          73
          93
         110
         147
         185
         220
         294
         370
         440
         587
         740
         880
        1175
        1475
        1760
        2349
        2960
        3520
        4699
        5920
        7040


bin =

    27


weight =

    0.9668
    0.9654
    0.9638
    0.9626
    0.9634
    0.9632
    0.9652
    0.9649
    0.9789
```

```
    0.9817
    0.9945
    0.9979
    0.9987
    0.9991
    1.0000
    1.0000
    1.0001
    1.0002
    1.0002
    1.0003
    1.0003
    1.0003
    1.0003
    1.0004


weighted =

   1.0e+03 *

    0.0358
    0.0444
    0.0530
    0.0703
    0.0896
    0.1060
    0.1419
    0.1785
    0.2154
    0.2886
    0.3680
    0.4391
    0.5863
    0.7394
    0.8800
    1.1751
    1.4752
    1.7603
    2.3496
    2.9608
    3.5211
    4.7006
    5.9217
    7.0427
```



Published with MATLAB® 7.14

Appendix C

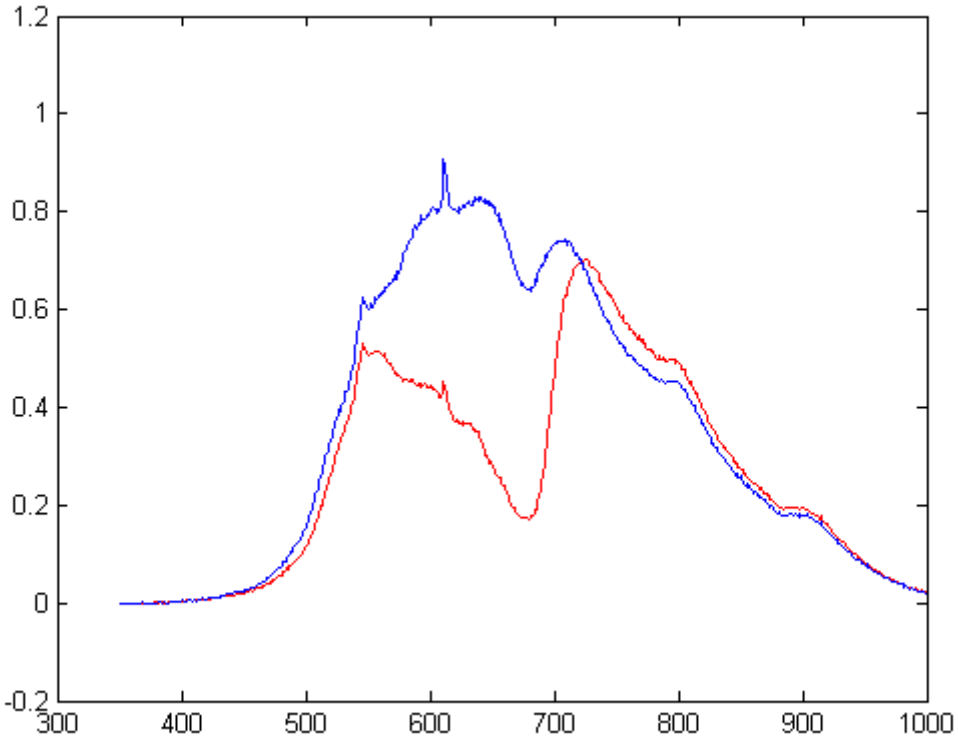
Final Code

```
% goal: usable final product!!!!

% clear ALL the things
clc;
clear all;

% manually set sensitivity to differences between input spectrum and
% expected spectrum - inactive
% sensitivity = 1;

% sets the minimum wavelength measurable by the spectrometer
min_wavelength = 350;

% calls file chosen as expected value for comparison with dynamic
% readings
% creates individual arrays for each column
spectrum_in1 = dlmread('Spectrometer Data\feb18a.txt', '\t', 1,1);
all = dlmread('Spectrometer Data\feb18a.txt', '\t', 1, 0);
t = all(1:end,1);

% this block establishes the dark readings of the spectrometer and removes
% the background from the spectrum
background = dlmread('Spectrometer Data\BG600_1-2.txt', '\t', 1, 1);

a=t(1) - (min_wavelength - 1);
b=t(end) - (min_wavelength - 1);
background_adj = background(a:b);

spectrum_out1 = spectrum_in1 - background_adj;

% determines size of spectrum and sets size of audio spectrum
[M,N] = size(t);

if M < 400;
    bin_num = 12;
    notes = [73, 93, 110, 147, 185, 220, 294, 370, 440, 587, 740, 880]';

else
    bin_num = 24;
    notes = [37, 46, 55, 73, 93, 110, 147, 185, 220, 294, 370, 440, 587,...
        740, 880, 1175, 1475, 1760, 2349, 2960, 3520, 4699, 5920, 7040]';
end

% sets bin size, initializes vectors
bin = round(M./bin_num);
k = 0;
final_array1 = zeros(2,bin_num);

% creates binned array with average values of input spectrum
% this array corresponds to the ideal input spectrum
for i = 1:bin_num
    final_array1(i,1) = i;
```

```
    index=i-1;
    k = index*bin+1;
    if  k+bin < size(spectrum_out1)
        final_array1(i,2) = mean(spectrum_out1(k:(k+bin)));
    else
        final_array1(i,2) = mean(spectrum_out1(k:end));
    end
end

% plays D major chord with harmonics varying based on signal
soundsc(tonec([notes,final_array1(:,2)], 1, 10000));

q = 1;
while q;

    % calls file given once per second by the spectrometer
    % spectrometer needs to always output same file name
    spectrum_in2 = dlmread('Spectrometer Data\feb18b.txt', '\t', 1,1);
    spectrum_out2 = spectrum_in2 - background_adj;

    % creates binned array with average values of input spectrum
    % this array corresponds to the dynamic input spectrum
    k = 0;
    final_array2 = zeros(2,bin_num);
    for i = 1:bin_num
        final_array2(i,1) = i;
        index=i-1;
        k = index*bin+1;
        if  k+bin < size(spectrum_out2)
            final_array2(i,2) = mean(spectrum_out2(k:(k+bin)));
        else
            final_array2(i,2) = mean(spectrum_out2(k:end));
        end
    end

    % introduces frequency shift where the dynamic spectrum differs from
    % the ideal spectrum

    ratio = final_array2(:,2) ./ final_array1(:,2);

    weight = mean(ratio(:)) ./ ratio(:);

    weighted = weight .* notes;

    % plays D major chord with harmonics varying based on signal and with
    % frequency shift introduced to provide audible warning of differences
    % between the dynamic and ideal spectra
    soundsc(tonec([weighted,final_array2(:,2)], 1, 10000));


    %testing:
    plot(t(:), spectrum_out1(:), 'r');
    hold;
    plot(t(:), spectrum_out2(:), 'b');
```

```
    % stops while loop so I don't crash matlab again
    q = q - 1;
end
```

Current plot held



Published with MATLAB® 7.14

Bibliography

[1] Iain Morley. The Evolutionary Origins and Archaeology of Music. PhD thesis, Cambridge University, 2003.

[2] Inc. ACI Technologies. Tech tips... automated optical inspection, December 2003.

[3] Ocean Optics Incorporated. Usb650 red tide spectrometer, 2012.

[4] Vernier Software & Technology. Logger pro 3: Quick reference guide, 2014.

[5] Kevin D. Donohue. tonec.m, June 2003.

[6] The Mathworks Incorporated. R2012a documentation, 2014.