Extraction of Musical Pitches from Recorded Music

Mark Palenik

ABSTRACT
Methods of determining the musical pitches heard by the human ear hears when recorded music is played were investigated. The ultimate goal was to find a good method for accurately converting recordings into midi, thereby gaining a complete set of notation information. For this project, only methods for determining pitch information, as opposed to rhythm or key signature information, were investigated. Various methods were investigated with the hopes of getting the best response from complex polyphony.

I. Introduction

When objects vibrate, they produce a mechanical disturbance in the surrounding medium called sound. This disturbance propagates through the medium as a wave whose properties are determined by the motion of the object creating the disturbance and properties of the medium itself. The human ear is the product of millions of years of evolution in an environment filled with a medium, air, that contains many sound waves produced by various sources. As such, it became very advantageous to be able to discriminate between different sound sources and interpret their meanings.

Although it isn't all that difficult to a computer with a proper microphone to produce an accurate recording of a sound wave, software, at present, isn't even close to being capable of replicating the type of interpretation of sound that the human ear and brain are capable of together. There has been a lot of research over the years into voice and recognition, and despite the fact that the technology has its limitations, there have been some impressive results. There has been considerably less research, however, into musical recognition, most likely because the technology is considered less important and useful.

First, various methods of decomposing waves into frequency components and extracting phase information were explored. This stage of the program was meant to emulate the function of the cochlea in the ear, which performs a similar decomposition of sound waves. From there, the frequency data was sent through a peak search, to find the location of the dominant frequencies. These frequencies were then fed through an algorithm which fit them to a tunable musical scale. Algorithms for analyzing phase information and spectral comparisons were written to help determine audible pitches from overtones and noise, but have not as of yet been implemented.

II. Background

Looking at a simplified model of human hearing, it could be said that sound recognition occurs in two stages. First, there's the input of sound into the ear, which decomposes sound waves into their constituent harmonics. Second of all, there is the interpretation of this sound by the brain, which filters out extraneous sounds, and matches the relevant ones to some kind of meaning.

This project emulates both stages, relying on computational, rather than mechanical methods for many of the aspects in the first step. The cochlea in the ear is a curled piece of material whose thickness varies along its length, giving it a different resonant frequency at each point. Thus, the location of resonance on the cochlea can be used by the brain to determine the frequency of incoming sound waves.

Attempts have been made to emulate the function of the cochlea by using a series of overlapping bandpass filters, while other, more common methods of frequency decomposition involve using the Fourier transform, or one of its variants. The first method I explored for frequency decomposition is similar to the bandpass filter method. However, this method was not based on previous research, but rather, something derived independently, from researching cochlear function. It makes use of a Green's function to simulate harmonic oscillator response to a sound wave.

Part III: Program Methods

The program written for this project decomposed waves into frequency components, did some further signal analysis, found the peak frequencies, and matched them to corresponding notes. The method for doing each of these things is described below.

Frequency Decomposition - Green's Function

Green's functions are used for solving differential equations that include a term for a time dependent force. The exact solutions for an arbitrary force function are found by taking the limiting case of finite step impulse functions as they approach infinitesimal delta functions. However, if instead of taking the limit as the width of the step size goes to zero, the solution is left as a discrete sum over small, but finite steps, the resulting function is perfect for use with a recorded digital signal. The Green's function used finds the response of a damped harmonic oscillator with a natural frequency w_0. The resulting function is of the form:

$$\sum a[t]\frac{T}{w}e^{B*[t-tn]}\sin w[t-tn]$$

where T is the duration of each impulse, a[t] is the acceleration as a function of time, t is the current time, w is the frequency of the oscillator after dampening, and B is a dampening constant. The function is summed over tn.

A set of linearly spaced oscillators were created and their output amplitude measured after responding to a portion of the recorded sound. The amplitudes were then plotted to create a spectrum. Amplitudes were calculated by the phasor addition of each term in the sum. This required doing a second sum over cos(w[t-tn]) and calculating the vector length of the two sums. Only around 250 oscillators are necessary to get well resolved peaks, which is good since large numbers of oscillators cause significant slowdown calculation time.

Since the acceleration, rather than force, appears in the function, an appropriate method of determining the mass of the oscillator must be determined, since the amplitude of the recorded sound is proportional to its force, and thus will have a different effect on objects of different masses.

The boundary conditions for an actual cochlea are somewhat difficult to emulate. The resonant frequency changes smoothly as a function of position, making it difficult to really treat the object as a collection of discrete oscillators. Most likely, some form of Cauchy boundary conditions on a collection of connected strings would be the best way to model its behavior. For the purposes of this project, however, the boundary conditions were simplified to assume that each oscillator was a string of the same tension, obeying Dirichlet boundary conditions with the ends fixed at one wavelength.

These conditions were plugged into the wave equation to produce effective masses for each oscillator, which ended up being proportional to $1/w_0$. Testing the equation with a constant mass for each oscillator produced a very large response bias at low frequencies. Inserting a mass term proportional to $1/w_0$ corrected the bias and produced similar results the Fourier transform.

The function was also tested with and without a dampening term. Inserting a dampening term, B, effectively creates a windowing function on the input signal, causing nearby frequencies to blend together, while removing most of the low frequency noise. The best results occurred with a low dampening constant, around 10 or 15, which removed a small amount of the more difficult to compensate for noise, while retaining good peak separation for nearby frequencies. The latter part is especially important, since for pitches near the bottom of the audible range, the frequencies of neighboring notes can be very close together.

Fast Fourier Transform (FFT)

Another method explored was the fast Fourier transform, or FFT. The main advantage of the FFT is that it performs very quickly. A regular Fourier transform is calculated by:

$$F[k] = \int f[x] e^{-2pi*ikx} dx$$

A fast Fourier transform speeds up this calculation for discrete functions so that it can be done in time proportional to 2Nln(N), rather than $2N^2$, where N is the number of data points to be analyzed. It also makes use of properties of signals, like the Nyquist frequency, which determines the highest frequency that needs to be used in the calculation.

The FFT, however, produces too many extraneous peaks, and does not give good enough peak resolution to be used for this project. Since a musical piece will in theory, change with time, the recorded sample must be broken up into small pieces when analyzed. The FFT does not perform well enough on sample sizes on the order of 1024 samples to be used. Even on larger sample sizes, the Green's function method produced better results, while taking considerably longer to calculate.

To improve the frequency response of a FFT, a windowing function can be used. I experimented with the Hanning window, which is of the form:

$$\frac{1}{2}\left[1 + \cos\frac{pi * x}{a}\right]$$

In many cases, this does improve results somewhat, but it has the same negative effects as the dampening term in the Green's function. It causes nearby peaks to be blended together, which can mask notes that should be distinguishable.

The Continuous Wavelet Transform (CWT)

To help resolve some of the problems with the FFT, a continuous wavelet transform was used. Although created mainly to retrieve good time dependent information from a sample, the continuous wavelet transform also provided better frequency information at a fixed time than the FFT. This was due to the fact that the frequencies and window sizes tested were not chosen to have orthogonality, but rather, for the best possible frequency response. While this removed any possibility of signal reconstruction, that was not important for the analysis being done.

The continuous wavelet transform decomposes a sample into wavelets, rather than sine and cosine functions. The Fourier transform can be thought of as the projection of a function onto a set of orthogonal sine and cosine functions. The continuous wavelet transform uses the same projection operation, but uses a series of wavelets as its basis.

A wavelet is a function with some sort of frequency, like a sine or cosine function, but the bulk of the function is confined to a narrow window. The frequency of the wavelet is proportional to its size, so larger wavelets are used to measure lower frequency components of a signal, and smaller wavelets measure higher frequency components. The wavelet can be slid forward by a small increment to change the point in time where the signal is being analyzed. The uncertainty in the time at which the various frequency components appear is equal to the wavelet size of each frequency. Because of this, when processing a digital signal, it is only necessary to slide the wavelet forward in increments proportional to its size.

For this project, I used wavelets made up of simple sine and cosine functions cut off by a rectangular window. However, I found that making the wavelength of the sine and cosine functions equal to the size of the wavelet did not produce the best frequency response. Rather, I found that, for a wavelet size L, a wavelength varying linearly from L to 2L as L increased from its minimum to its maximum size, provided very good peak resolution, while still allowing relatively small wavelet sizes for low frequencies.

I experimented a with a Hanning window again, using a CWT. The Hanning window created the same problems with frequency resolution as before, except magnified to a greater extent. This is most likely because the wavelets I used did not go to zero at both ends of the window like they do in a Fourier series. Multiplying by a Hanning window, which forces both ends of the window to zero would have introduced additional distortion. Low frequency response with the Hanning window did not even resemble the actual frequency components of the signal.

All in all, the CWT gave the best frequency resolution, and better time resolution than any other method. I ended up using a maximum window size of around 2000 sample, but because of the nature of the transform, most frequencies are analyzed at a smaller window size. The CWT gave marginally better frequency response than the Green's function method on windows of a similar size. The CWT, however, performed significantly slower.

Extracting Phase Information – Green's Function

Since the code for calculating harmonic oscillator response was already written, it was a simple matter to modify it for use in determining phase information. As cited in Joe Yasi's paper on extracting phase harmonics from a digital signal, a Fourier series does not provide accurate phase information over a small sample size. I confirmed this fact experimentally, and decided to go with a method similar to his bandpass filter method for determining phase. Instead of a bandpass filter, I used a harmonic oscillator with a natural frequency at the frequency being analyzed.

Like before, a summation was done over sine and cosine functions. An additional step was then used, which took the arc tangent of the sine summation divided by the cosine summation, to give a result for the phase of the oscillator response in terms of radians. The output varied from -pi to pi. With a few additional calculations, the phase between 0 and pi was found. To determine the phase offset of the signal, rather than the phase at a particular time, the following code was used:

```
If XofTOut(n) < 2 * 3.1415927 * (Freq * n * Tau - Int(Freq * n * Tau)) Then
      XofTOut(n) = 2 * 3.14159 + XofTOut(n) - 2 * 3.1415927 * (Freq * n * Tau - Int(Freq * n * Tau))
    Else
      XofTOut(n) = XofTOut(n) - 2 * 3.1415927 * (Freq * n * Tau - Int(Freq * n * Tau))
End If
```

where XofTOut is an array that stores the phase information at a point in time, n. The result is the phase offset of the frequency component from the start of the window.

There was one additional hurdle to overcome in this section of the program. At every point in time at which the phase was calculated, the Green's functions would need to be re-summed, which was impossible to do in any realistic time frame. However, looking at the sum once more:

$$\sum a[t]\frac{T}{w}e^{B*[t-tn]}\sin w[t-tn]$$

It can be seen that each term in the sum varies by a constant phase factor, T. *tn* increases by T with each consecutive term in the sum. The current time, *t*, also increases by T whenever the phase calculation is moved forward in time. This means that given a phase at a particular point in time, P(t), P(t + T) is just the same sum with a phase offset, plug one additional term. If the phase of every term in the sum at t + T can be increased by T, it would only take one additional calculation to determine the phase at time t + T.

This type of "phase promotion" is impossible to do with only a sine function. However, given the exponential function $e^{ikx} = \cos(x) + i\sin(x)$, increasing the phase is a simple matter of multiplying by e^{iT}. This corresponds to a raising operator in phase space, or a rotation operator in coordinate space. The phase, then is just the arc tangent of the imaginary part of the function over the real part, with the same additional calculations as described before.

Peak Search

The next step after determining frequency and phase information of the signal was to run a peak search over the frequency spectrum. I used the method described on the Thermogalactic website (a company that makes spectral analysis software and spectroscopy equipment) with a few minor modifications. The average difference between data points is computed, and stored as a value called “noise”. Then, the program searches for any slopes that pop up above the “noise” value, and calls it the start of a peak. The end of the peak is marked at the point after the slope goes down by an amount greater than the “noise” value, then levels off to a smaller value. The actual peak is calculated by taking the center of mass of the three highest points between the start and endpoints. From there, baselines, and other pieces of information are calculated.

Spectrum Comparisons

I also wrote a piece of code that does a spectral peak search to match similar spectra in order of decreasing similarity and return a percent match value. Since the human ear is bad at distinguishing between frequencies close to each other, I thought the brain might use a method similar to this in music recognition to recognize chords when the intervals between the notes were small. The idea was to test the spectrum calculated against a list of possible spectra, computed from the CWT of a combination of pure tones, however this process was never implemented, due to time constraints.

Matching Peaks to Notes

After the peak search takes place, the peaks need to be matched to notes on the musical scale, and not just frequencies. To do this, the program calculates a musical scale, based on an adjustable input for middle A (440 hz). The notes in the scale calculated are A, A#, B, C, C#, D, D#, E, F, F#, G, and G# for as many octaves above and below middle A as specified by an input parameter.

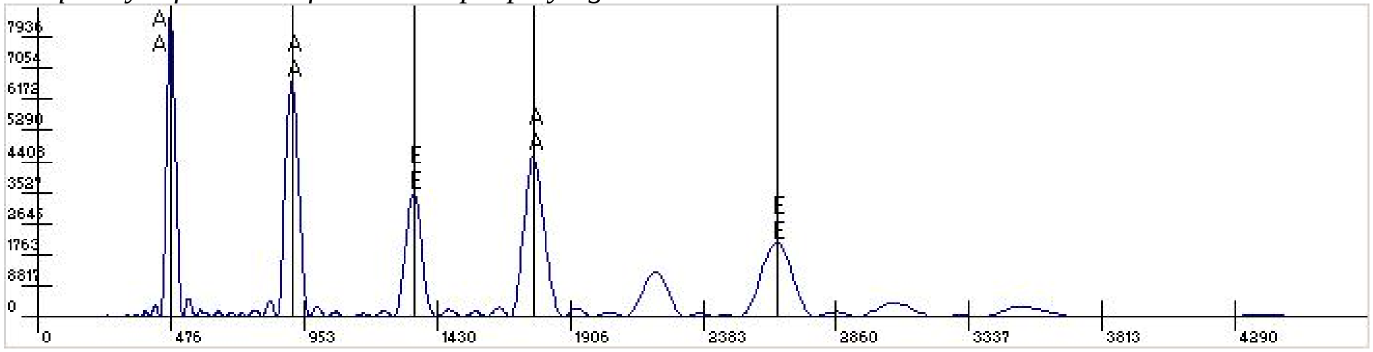
No allowances are made in the program for spelling a note with a flat instead of a sharp, or for using double sharps, double flats, etc. The determination of the spelling of a piece of music is a complex task for a computer that would be an entire project in itself.

The peak frequencies are matched to the nearest note, and for especially close matches, a “true” value is returned to indicate that the peak is more likely a real note, and not noise, although it may still be a harmonic.

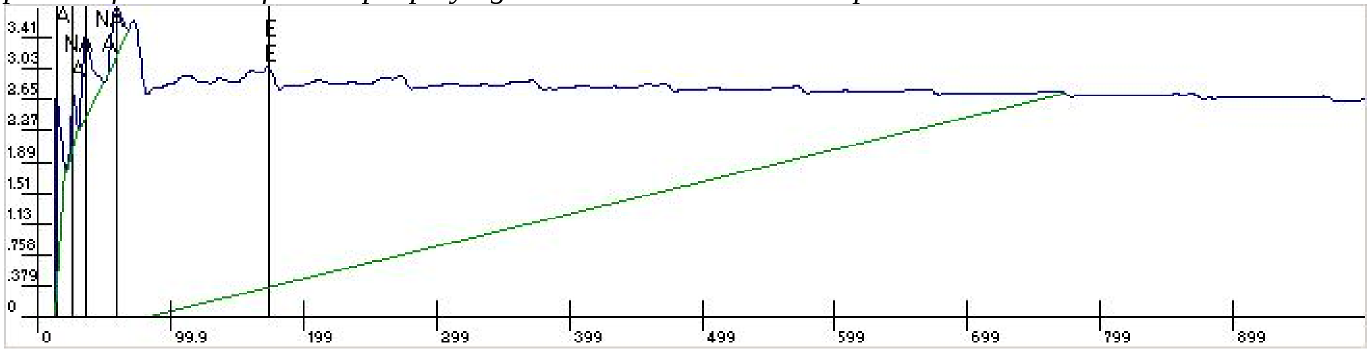
IV. More about the program

Also included in the program is a function for inverting matrices, a function for doing a polynomial fit of n degrees, based on Euler's method (as described in the textbook Linear Algera by Leon), Direct Sound code for loading and playing .wav files, and a network which I had written earlier and thought I might use in this project, but was unable to get to perform in a useful manner. Below are some screenshots from the program:

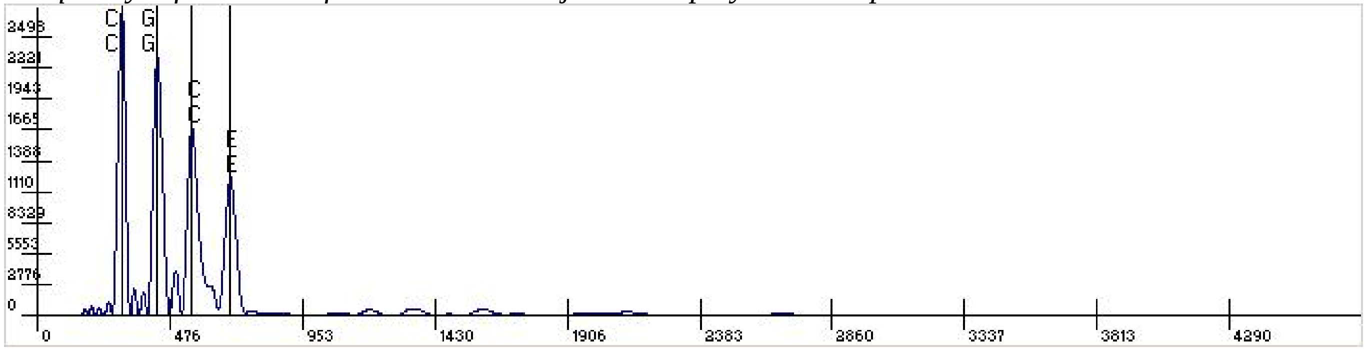
Frequency information from a trumpet playing a 440 hz A



phase information of a trumpet playing a 440 hz A at the 1760 Hz peak



Frequency information of an inverted C major chord played on the piano.



(the note information obtained by the program is accurate in this image)

V. Conclusion

So far, the program doesn't perform a whole lot of useful functions. It has a decent algorithm for determining frequency composition of a wave in discrete time intervals, and a good base of functions for a more advanced program. It still needs a lot of work to put it into a more useable form, which I was not able to do, due to time constraints. Ideally, I would like the program to be able to convert a full recorded song into a midi file, with minimal errors. Implementation of a sliding wavelet in time, to gain time as well as frequency information, and a method for determining rhythms need to be added. I will most likely work on adding these features on my own at a later date.

Acknowledgements

I would like to thank Robert Bristow-Johnson for providing a simulation filter for the Fletcher-Munson curve at 0 dB, despite the fact that I never used it. I would also like to thank Professor Errede for running this class and answering a few questions I had while doing my project.

References

Classical Dynamics of Particles and Systems: Fifth Edition, Thorton&Marion.

Linear Algebra with Applications, Leon

http://users.rowan.edu/~polikar/WAVELETS/WTtutorial.html "The Engineers Guide to Wavelet Analysis", Robi Polikar

http://oto.wustl.edu/cochlea/ "Cochlear Fluids Laborotory", Alec N. Salt, PhD

"An Algorithm for Extracting the Relative Phase Harmonics from a Periodic Digital Signal", Joe Yasi.