

Digital Analysis and Synthesis of Musical Recordings

New Mexico

Supercomputing Challenge

Final Report

April 4th, 2011

Team 43

La Cueva High School

Eldorado High School

Team Members:

Alexandra Porter

Evan Roche

Sponsor:

Jennifer Coughlin

Digital Analysis and Synthesis of Music Recordings

Table of Contents

Project Summary	5
Introduction.....	6
Purpose	7
Background and Theory	8
Fourier Transform.....	8
Wavelets.....	9
Harmonics and Overtones.....	10
Procedural Overview.....	12
Fourier Transform.....	13
Haar Wavelet Transform.....	10
Analysis Program Outline.....	16
Results and Conclusions.....	17
Fourier.....	17
Wavelets.....	23
Analysis Program.....	26
Synthesis.....	29
Future Work.....	36
Acknowledgements.....	38
Bibliography.....	37
Appendix A - Data.....	39
Appendix B - Code.....	43

Tables

1. Instrument Characteristics.....	9
2. Dynamic Ranges.....	11
3. Wavelet Step Size Ratios.....	11
4. Waveform Analysis Results.....	27

Equations

1. Continuous Fourier Transform.....	4
2. Euler's Formula.....	4
3. Euler's Formula applied to the Fourier Transform.....	4

Figures

1. The Haar Mother Wavelet.....	6
2. The Wavelet Transform.....	10
3. Flute A4 Waveform.....	18
4. Clarinet A4 Waveform.....	18
5. Trumpet A4 Waveform.....	19
6. Oboe A4 Waveform.....	19
7. Flute A4=440 Hz.....	20
8. Clarinet A4=440 Hz.....	20
9. Trumpet A4=440 Hz.....	21
10. Oboe A4=440 Hz.....	21
11. Oboe B4=493.883 Hz.....	22
12. Oboe A5=880 Hz.....	22
13. Oboe A4 Dynamic: High Volume.....	23
14. Oboe A4 Dynamic: Low Volume.....	24
15. Oboe A4: High Pass Wavelet Coefficients.....	24
16. Oboe A4: Intermediate Pass Wavelet Coefficients.....	25
17. Oboe A4: Low Pass Wavelet Coefficients.....	25
18. Oboe A4: Original Waveform.....	30
19. Oboe A4: 1 Peak.....	30
20. Oboe A4: 2 Peaks.....	31
21. Oboe A4: 4 Peaks.....	31
22. Trumpet C5: Original Waveform.....	32

23. Trumpet C5: 1 Peak.....	32
24. Trumpet C5: 3 Peaks.....	33
25. Trumpet C5: 6 Peaks.....	33
26. Oboe A4 Trumpet C5: Original Waveform.....	34
27. Oboe A4 and Trumpet C5: 1 Peak.....	34
28. Oboe A4 and Trumpet C5: 3 Peaks.....	35
29. Oboe A4: 4 Peaks; Trumpet C5: 6 Peaks.....	35

Project Summary

The goal of the project was to develop a C++ analysis program that can analyze recordings of musical instruments using Fourier and wavelet transform algorithms. The combined use of wavelets and the Fourier transform is a unique way of modeling the physical characteristics of sound for accurate analysis.

First, instrumental recordings were produced from various instruments playing specific pitches and series of pitches. C++ programs were developed to read the recorded .wav format files into numerical arrays that were then used in a Fast Fourier Transform program. The Fourier transform converts the discretely sampled data into the frequency domain, showing which frequencies are present in the sound. Peaks in the resulting frequency domain data show the fundamental frequencies and overtones present in a recording. A C++ program was developed to identify these peaks and measure their relative emphasis and position. This information determines timbre, or the distinct sound of the instrument. Knowing the frequency composition of a particular instrument at a certain pitch, a range of pitches can be synthesized with the timbre of the instrument.

Wavelets were also used to analyze the instrumental recordings. Wavelets allow for time domain analysis of discretely sampled data, which means a recording containing multiple pitches or a decaying pitch can be more accurately analyzed. Wavelets are a more versatile tool than Fourier Transforms, due to the fact that they allow the detection and analysis of multiple notes in sequence and musical phrases. Wavelets were also used in instrument identification.

A java program was developed to visually display analysis results in sheet music notation, a practical format for potential users.

Introduction

Sound, and especially music, is an essential part of the world that we live in. Sound is purely a mechanical wave of oscillating pressure with a variety of physical characteristics and properties that we record in our brains and on computers as electrical impulses [4]. In recent years, technology has allowed us to accurately detect and measure these characteristics. Our understanding of the physics and math governing sound has increased greatly, mainly through research in the scientific fields of acoustics and signal processing.

This increased understanding has led to the creation of many new sound and music technologies such as sound editing programs, sound recognition technologies, and artificial sound production.

There are still many things to learn about the physics of sound and the processes used for analysis and synthesis. Polyphonic sound analysis is currently being researched, but no defined methods have become widely accepted.

Purpose

The purpose of this project is to develop an algorithm to more accurately analyze the waveforms of sound files through modeling with wavelets and Fourier transforms. This algorithm will be used identify the instrument, frequency or frequencies, volume, and timbre of waveforms. The algorithm will also be used to analyze polyphonic recordings for these same characteristics.

Background and Theory

All the tests in this project were done on the waveforms of sound recordings sampled at 44100 Hz. A waveform is a series of discretely sampled pressures over time.

The Fourier Transform

The Fourier transform is an algorithm that determines the frequency components of a wave from either a function or discretely sampled amplitudes in the time domain. The Fourier transform results are in the form of amplitude over frequency. It finds the frequency by breaking a wave down into a summation of cosine and sine waves [9]. The Fourier transform is a useful algorithm applicable to any form of periodic data.

The Fourier transform of the continuous function $f(x)$ is shown as:

$$F(u) = \int_{-\infty}^{\infty} f(x)e^{-i2\pi ux} dx \quad \text{Equation 1}$$

Applying Euler's formula:

$$e^{i\theta} = \cos\theta + i\sin\theta \quad \text{Equation 2}$$

$$F(u) = \int_{-\infty}^{\infty} f(x)(\cos 2\pi ux - i\sin 2\pi ux) dx \quad \text{Equation 3}$$

The results of the Fourier transform are divided into real and imaginary parts. For most practical applications, only the real parts are of use. The data in this project is entirely composed of real numbers so all imaginary components are set to zero.

The Discrete Fourier Transform is a discrete adaptation of the Fourier Transform, that takes in a sampled set of finite values. This project made use of the FFT (Fast Fourier Transform), which is simply a more efficient way to compute the Discrete Fourier Transform. The FFT exploits useful patterns found in the Fourier transform to save both memory and time [9].

Wavelets

Wavelets are functions used in signal processing to analyze data that is not necessarily periodic. They are defined by two functions, the mother wavelet function, and the scaling function (also referred to as the father function) [2]. In wavelet transforms, the mother function is shifted, scaled and duplicated so that it can be used to model portions of the function being analyzed. By finding the scaling and wavelet coefficients, information can be obtained and the desired spectra produced.

Wavelets are a more recent approach to the problem of extracting both time and frequency information from a transformed function. The main weaknesses of the Fourier transform is that it requires periodic and continuous data [2]. The Discrete and Fast Fourier Transforms allow for non-continuous inputs, but data must still be periodic. The Short-Time Fourier Transform is an adaptation that has properties similar to wavelets. It is localized in both time and frequency, but it still cannot capture detail as precisely as wavelets. Wavelets can give a more accurate signal representation using multiresolutional analysis.

Another significant advantage of wavelets is their computational efficiency. They are even faster than the FFT. Wavelet transforms are calculated in $O(n)$, which is a linear growth

directly proportional to the size of the input. The FFT on the other hand, functions in the linearithmic $O(N \log n)$ time, which is slower than linear but faster than quadratic.

In this project, we used the Haar wavelet as a mother function. The Haar wavelet is a basic mother wavelet, as shown below. The Haar wavelet is not continuous so it can easily be used to scale inputted functions with rapid growths and decays [6]. This makes it a good candidate for analyzing audio files, as they often have sharp peaks or shifts in amplitude.

The Haar Mother Wavelet:

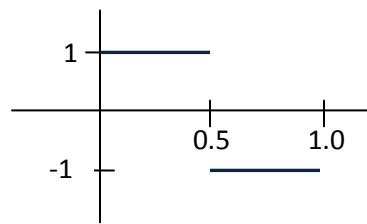


Figure 1

Harmonics and Overtones

The fundamental frequency of a signal is often accompanied by higher frequencies called overtones. Harmonics are ideal overtones, with frequencies that are integer multiples of the fundamental frequency. The overtones of instruments are usually close to the harmonics, but not necessarily exact multiples. Overtones that don't correlate to harmonics create small amounts of dissonance, which can contribute to the unique sounds of different instruments. During the construction of most instruments efforts are taken to minimize dissonance, such as flaring the bell on a trumpet.

Sound in instruments is generated by oscillating strings or columns of air. Strings and columns of air in an open tube vibrate at a certain frequency, but their length also vibrates in halves, thirds, fourths, etc. This produces overtones with doubled, tripled, and quadrupled frequencies. The first harmonic is the fundamental frequency, the first overtone is the frequency twice the fundamental frequency. If a tube is closed on one end, as some instruments are, only

odd harmonics are generated, due to the node created at the closed end. Overtones and harmonics are higher in frequency than the fundamental, and generally decrease in amplitude as they get higher [4].

Overtones are a very important part of sound, as they are the physical components that give each sound its unique timbre. Timbre is a term used to describe sound quality, sometimes it is also referred to as the “color” or “flavor” of sound. Timbre is determined by how many overtones can be heard, which of those are emphasized, and how close those overtone are to their respective harmonic series [4].

The relative ratios of amplitude and distance between overtones is a key factor, as well as the rate at which different overtones decay. Overall, overtones are an essential element in the process of determining the sound characteristics of a note or phrase.

Procedural Overview

First, a set of .wav recordings were created and obtained from the Internet [6]. A program was developed in C to read .wav files into .txt format.

A Fourier transform program was then written in C [7.]. The Fourier transform program performs a Fast Fourier transform on data points read from a text file. In order to get the most accurate results in the Fourier transform the articulation and decay at the end of each note were removed. The peaks resulting from the Fourier transform were then analyzed for location and emphasis, factors which determine the note's fundamental frequency and timbre.

The second program developed was a C program that performs the Haar wavelet transform. This program loops through the data and calculates the wavelet coefficients. These coefficients were then analyzed in order to reveal characteristics of the waveform in the time domain.

The main program used both of the transforms. It calculated each of them separately and used the results to analyze a sound file, identify the instrument, find the frequencies of multiple notes, and determine the volume levels.

The Inverse Fourier transform was used to recreate a waveform from a modified frequency spectrum. The Fourier transform of a waveform was calculated and noise was removed. The inverse transform resulted in a cleaner, more accurate pitch that still preserved timbre.

Fourier Transform Program

The Fourier transform was used to identify the fundamental frequency and the instrument being played. First, the highest peak was calculated and points in the bottom fourth of that peak were removed. The peak in each cluster of remaining data was then found. These remaining peaks were the most prominent and formed the basis for our overtone data.

The general trend in emphasis was calculated by counting how many peaks were higher than the previous peak and how many were lower. This was particularly useful in identifying the oboe, which has a fairly unique upward trend in its overtones.

Lastly, the relative emphasis of the first three peaks was calculated. This was accomplished for each of the three peaks by calculating the proportion of the amplitude of the peak to the average of the other two. For example, the ratio of emphasis for the first peak is $\text{amplitude}_1 / ((\text{amplitude}_2 + \text{amplitude}_3) / 2)$, which is equivalent to $2 * \text{amplitude}_1 / (\text{amplitude}_2 + \text{amplitude}_3)$. These relative emphases are generally the same for all notes on a given instrument. Data was calculated based on a range of notes (see Appendix A) and is show below. The program determines the instrument of the waveform, assuming it is the one that the ratios and upward or downward trend resemble most.

Instrument Characteristics

	Flute	Oboe	Trumpet	Clarinet
Ratio of 1st Peak	1.75	0.30	3.80	2.90
Ratio of 2nd Peak	1.25	1.55	.050	0.40
Ratio of 3rd Peak	0.80	2.00	0.40	1.00
Increasing or Decreasing	decreasing	increasing	decreasing	decreasing

Table 1

Haar Wavelet Transform Program

The Haar Wavelet transform operates by calculating the average and the step size between each pair of adjacent points in the waveform. The step sizes and averages are stored in an array, and the new averages become the data used to calculate the next set of step sizes and averages. This is repeated until a single average is achieved, therefore the number of passes is $\log_2(n)$ for $n = \text{number of original data points}$.

$2^x = n$ Points

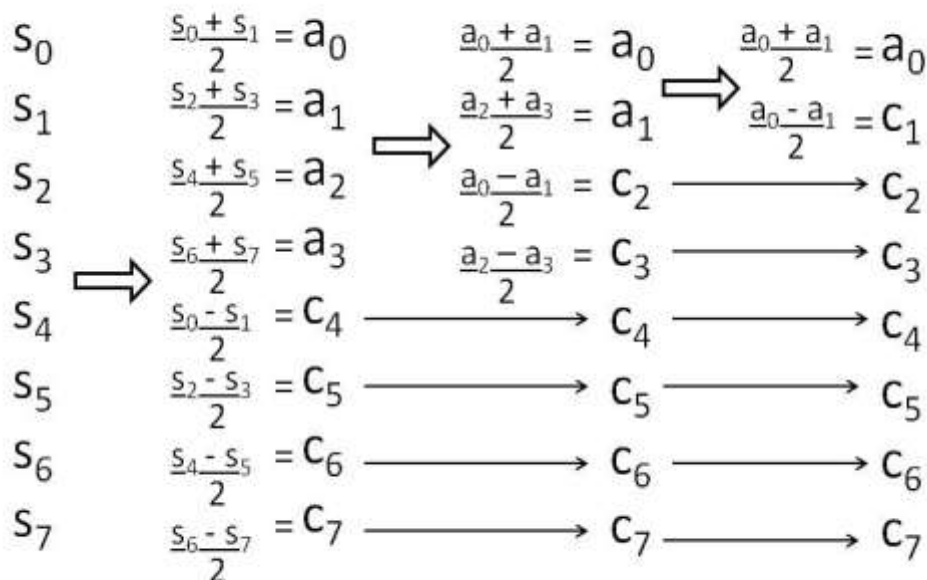


Figure 2

The Haar wavelet transform was calculated for the waveform and used to determine characteristics in the time domain, specifically note length and dynamic (volume) level.

Note lengths were calculated relative to each other. Because the tempo is unknown the actual musical length of the note (eighth note, quarter note, etc) cannot be determined. Instead, the notes are scaled to each other and the relative length is outputted.

The average dynamic level was determined by comparing the average step size of the wavelet transform to given dynamic ranges specific to the instrument. As shown below, the dynamics from softest to loudest are pianissimo, piano, mezzo-piano, mezzo-forte, forte, and fortissimo.

Dynamic trends upward (crescendo) or downward (diminuendo) were also calculated by looking at the trend in step size for a very low pass level filter on the wavelet transform. If this trend was significant, a starting and ending dynamic were determined and outputted.

Dynamic Ranges

	Piano	Mezzo-piano	Mezzo-forte	Forte	Fortissimo
Flute	0.0007	0.00125	0.0018	0.00238	0.0029
Oboe	0.000225	0.0002863	0.0003475	0.000409	0.00047
Trumpet	0.0008	0.00165	0.0025	0.00335	0.0042
Clarinet	0.0037	0.00385	0.004	0.00415	0.0043

Table 2

Wavelets were also used as a factor in determining the instrument. The ratio of magnitude of positive to negative step sizes was found to be effected by instrument. As shown, the positive and negative step sizes on the oboe are very close in magnitude.

	Flute	Oboe	Trumpet	Clarinet
Step Size Ratio	1.4	1.0	1.7	0.7

Table 3

Analysis Program Outline

1. Locate gaps in input; store each section/ note separately
2. Compute Fourier transform on each note
 - > Sends middle portion with length 2^x ; cleanest periodic sample
 - > Locate peaks in Fourier transform: points over fraction of maximum
 - > First peak is fundamental frequency, others are overtones
 - > If more than 2 peaks correspond to multiples of a different fundamental, identifies recording as polyphonic, separates out peaks by their corresponding fundamental
3. Compute relative size of each peak returned by Fourier transform
 - > Average up or down trend and relative sizes compared to instrument data
4. Compute wavelet transform
 - > ratio of positive to negative step sizes correlates to instrument
5. Record best instrument; instrument with most matches characteristics
6. Compute average amplitude from wavelet transform, match to instrument to identify dynamic range, trend
7. Display information in musical notation

Results and Conclusions

Fourier Transform

On the Fourier transforms shown the peaks over approximately 0.25 (a fourth of the highest peak) are of significance. The first of these peaks is the fundamental frequency and is within about 20 Hz of the frequency listed for each graph. The waveforms (figures 3-6) shown are the actual samples contained in the original .wav file. The differences between instruments in the shape of their cycle correspond to differences in the relative peak sizes in the Fourier transform (figures 7-10). Figures 10, 11, and 12 show the Fourier transforms of different notes being played on the same instrument, the oboe. These Fourier transforms all have similar characteristics, particularly the upward trend in emphasis of the first three peaks. Each of these Fourier transforms also has the fundamental frequency represented by the first significant peak. Due to the fact that oboe's fundamental frequency is relatively small, a slightly lower threshold is necessary to preserve the fundamental frequency and the first few overtones.

Flute A4

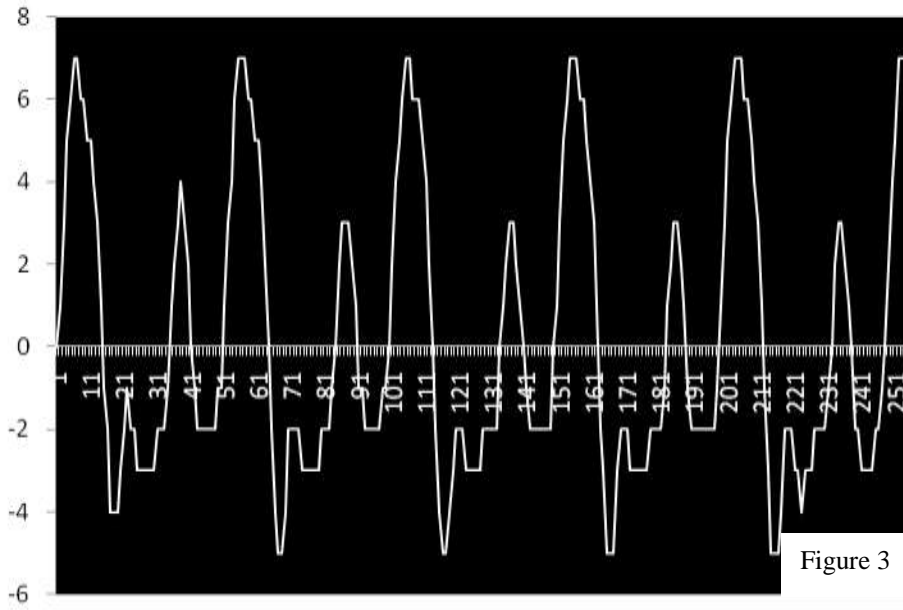


Figure 3

Clarinet A4

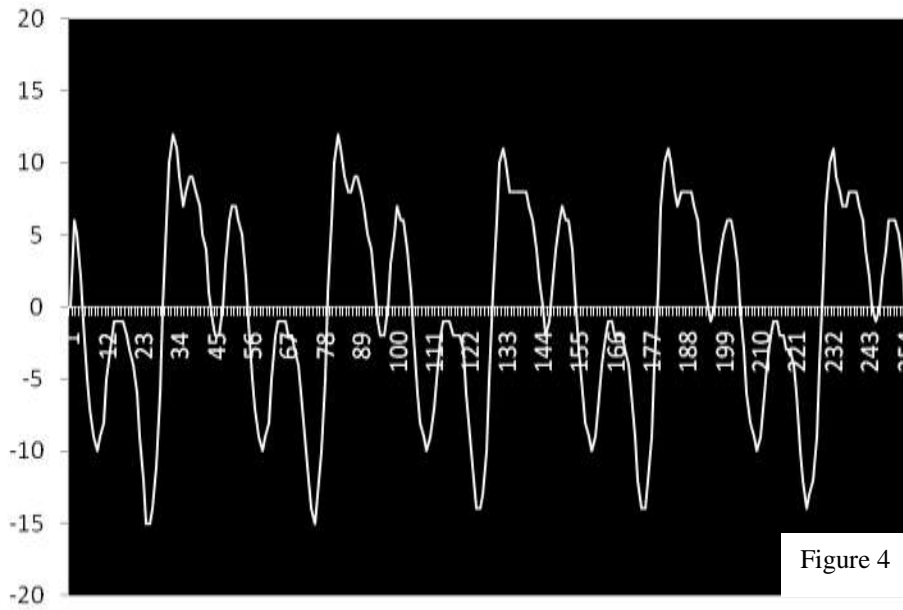


Figure 4

Trumpet A4

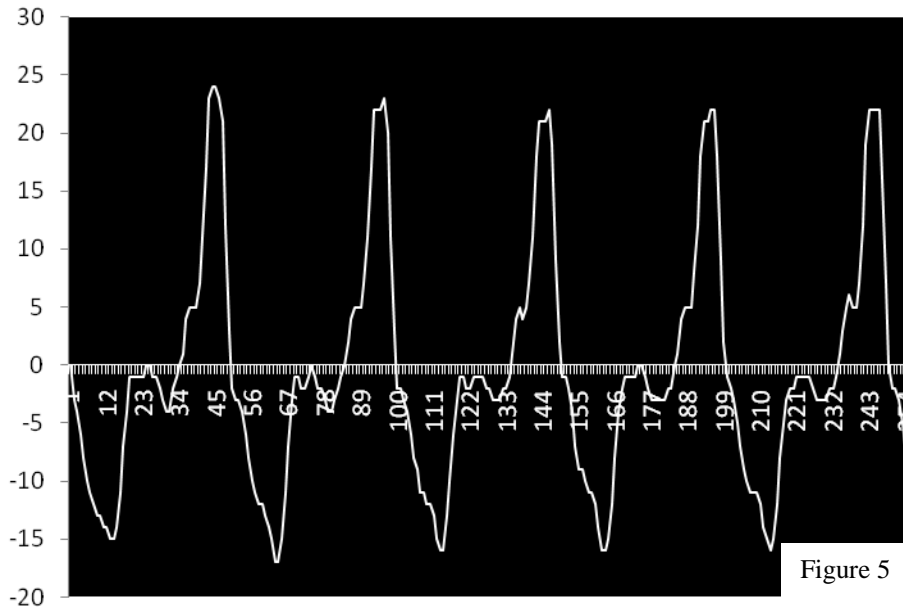


Figure 5

Oboe A4

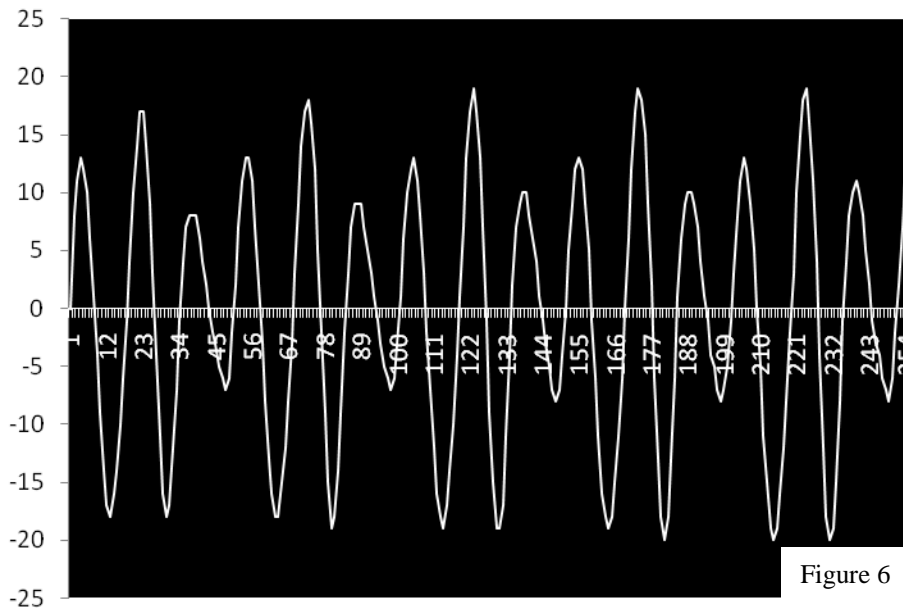


Figure 6

Flute A4= 440 Hz

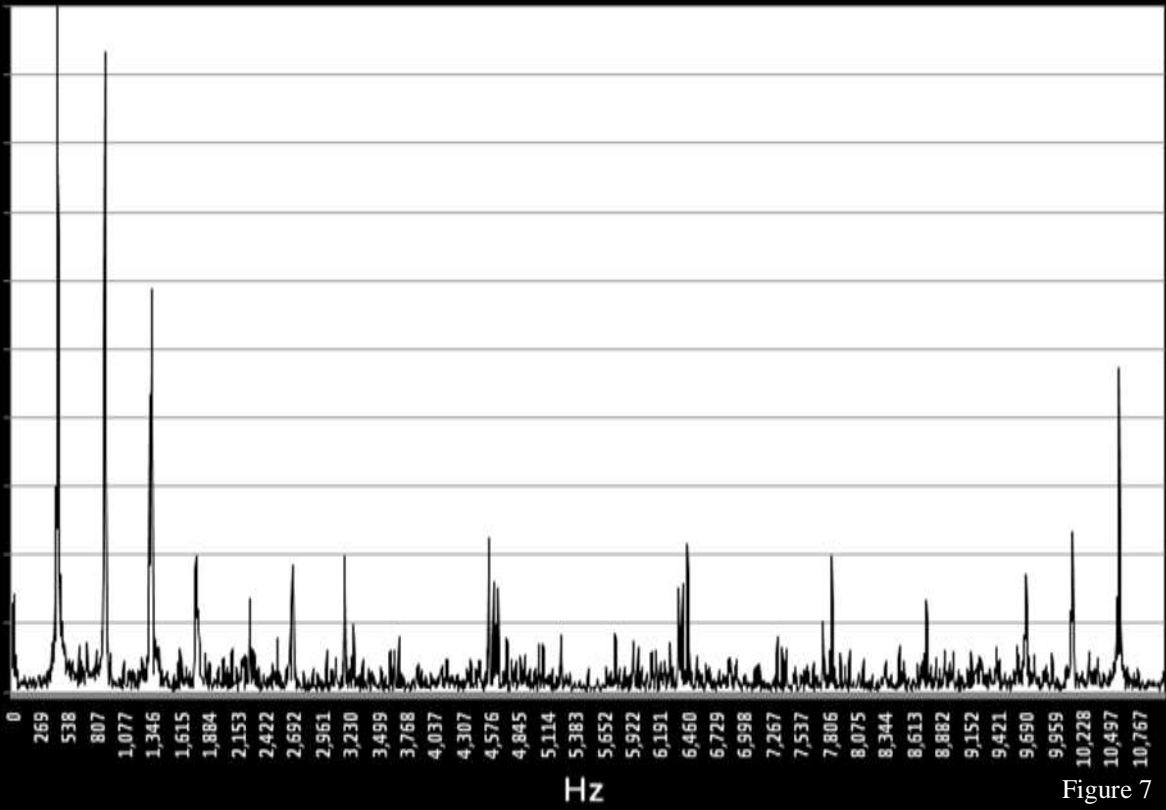


Figure 7

Clarinet A4=440Hz

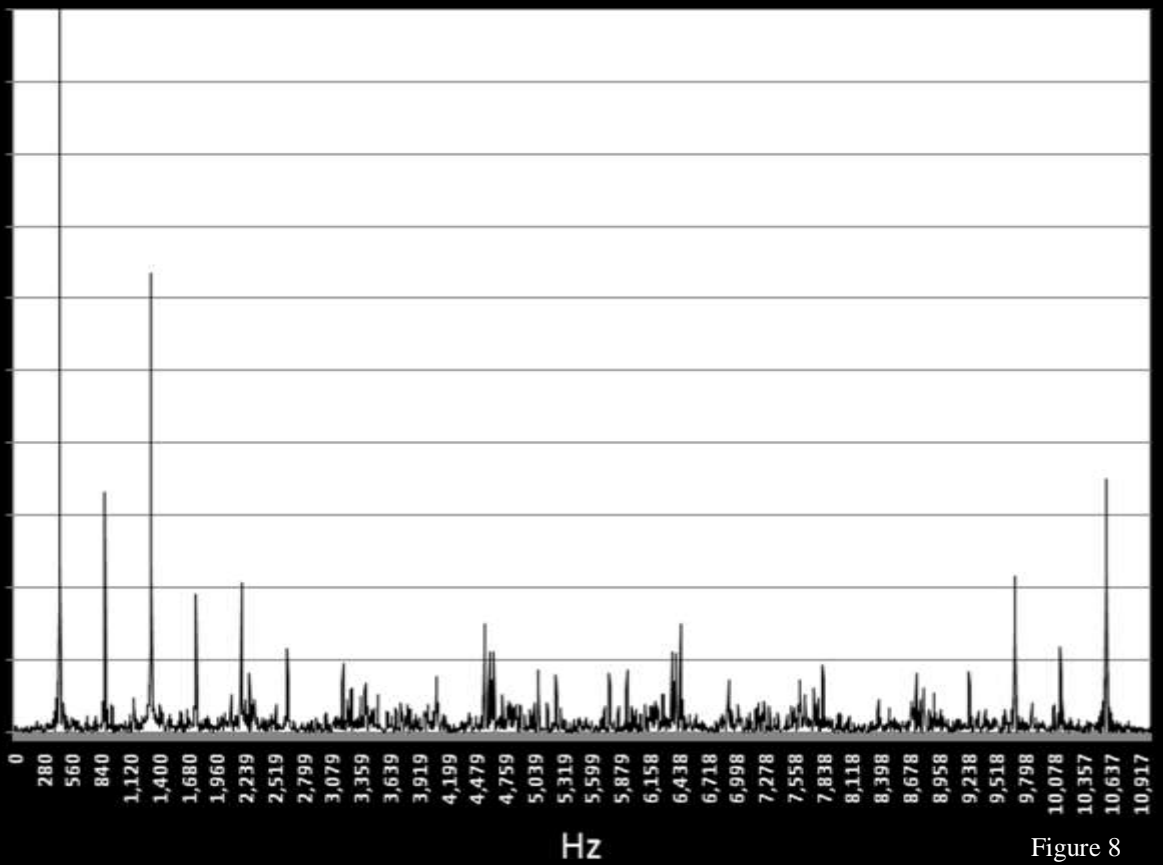


Figure 8

Trumpet A4= 440 Hz

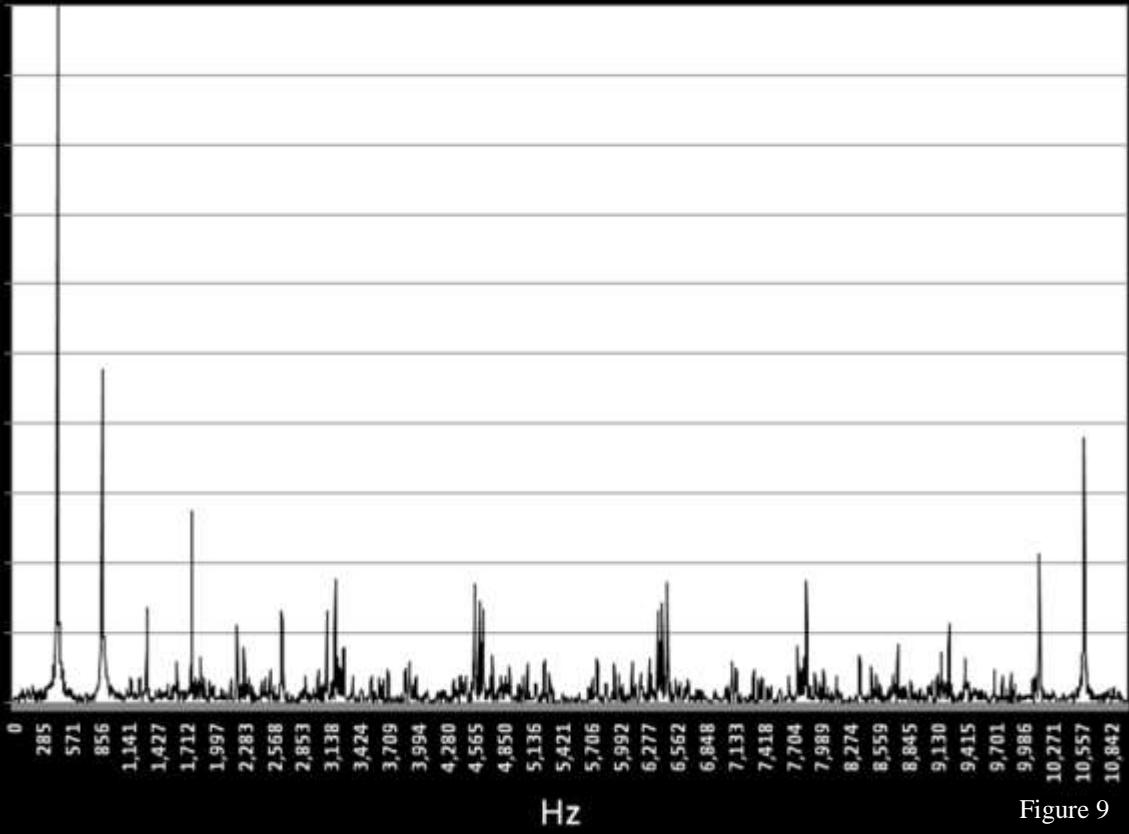


Figure 9

Oboe A4=440 Hz

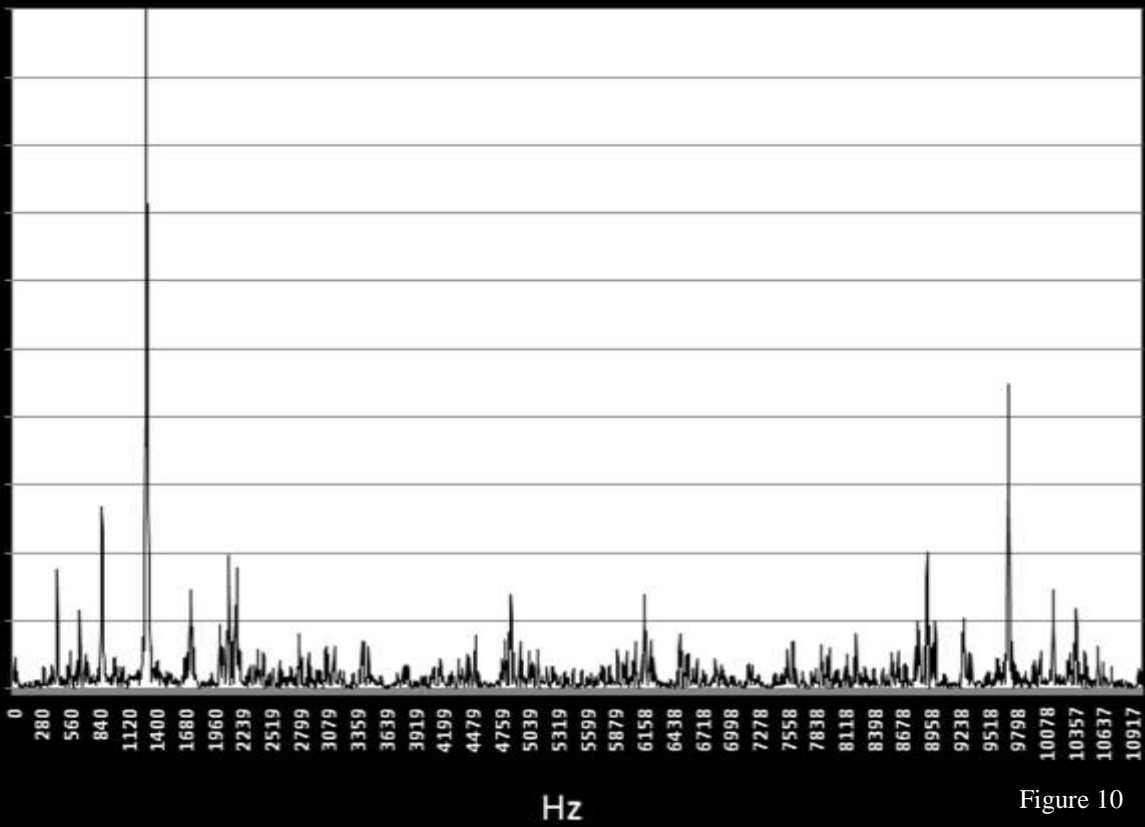


Figure 10

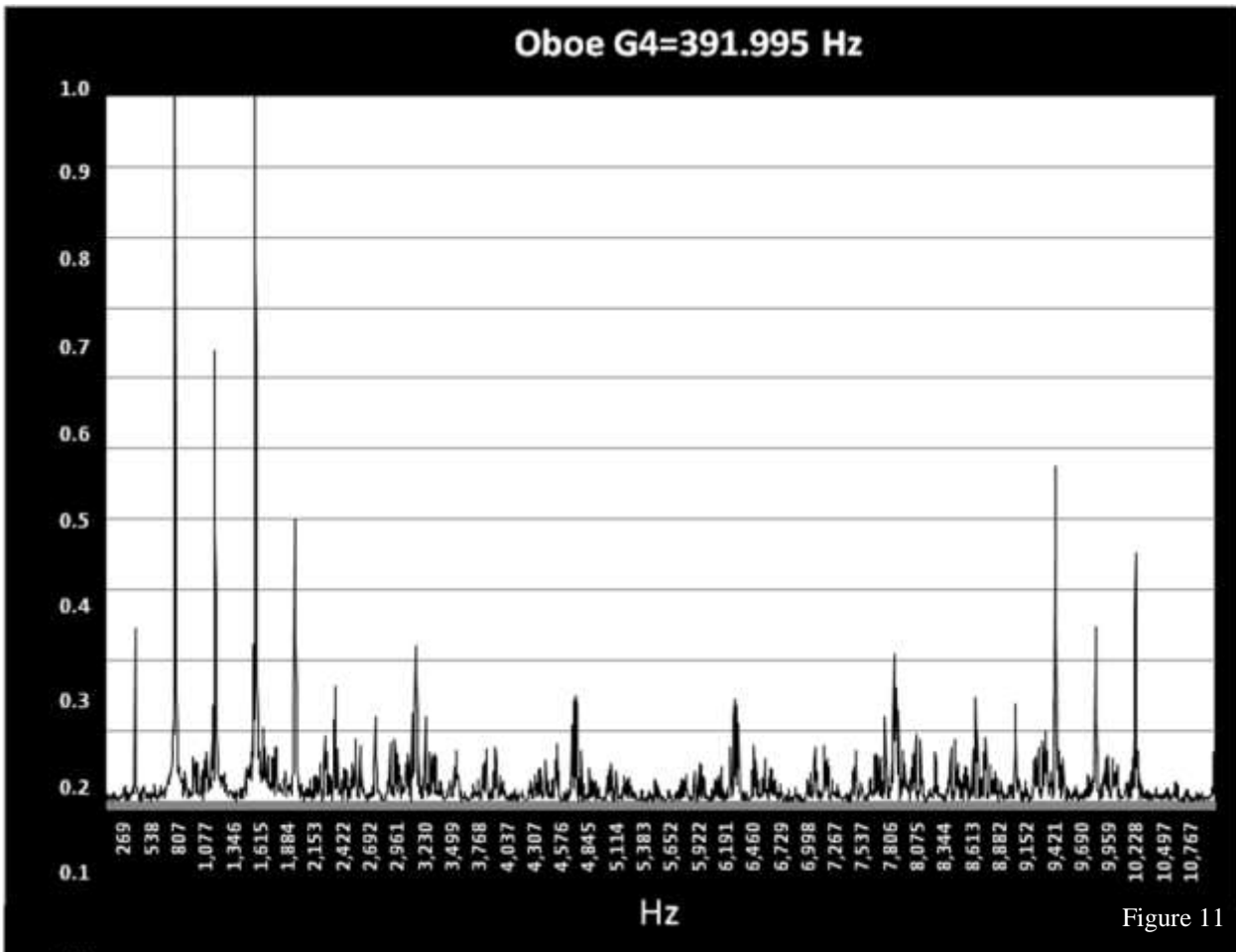


Figure 11

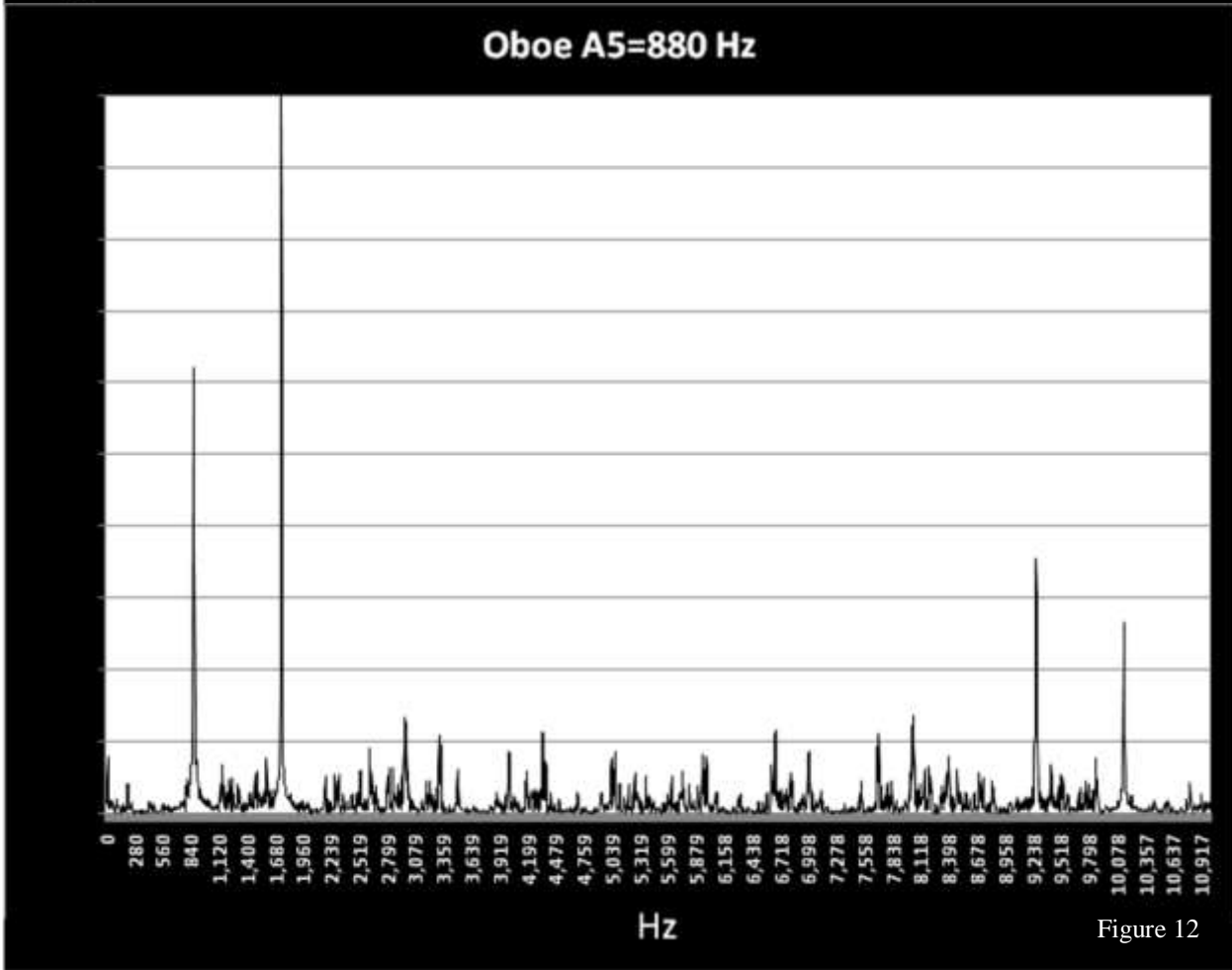
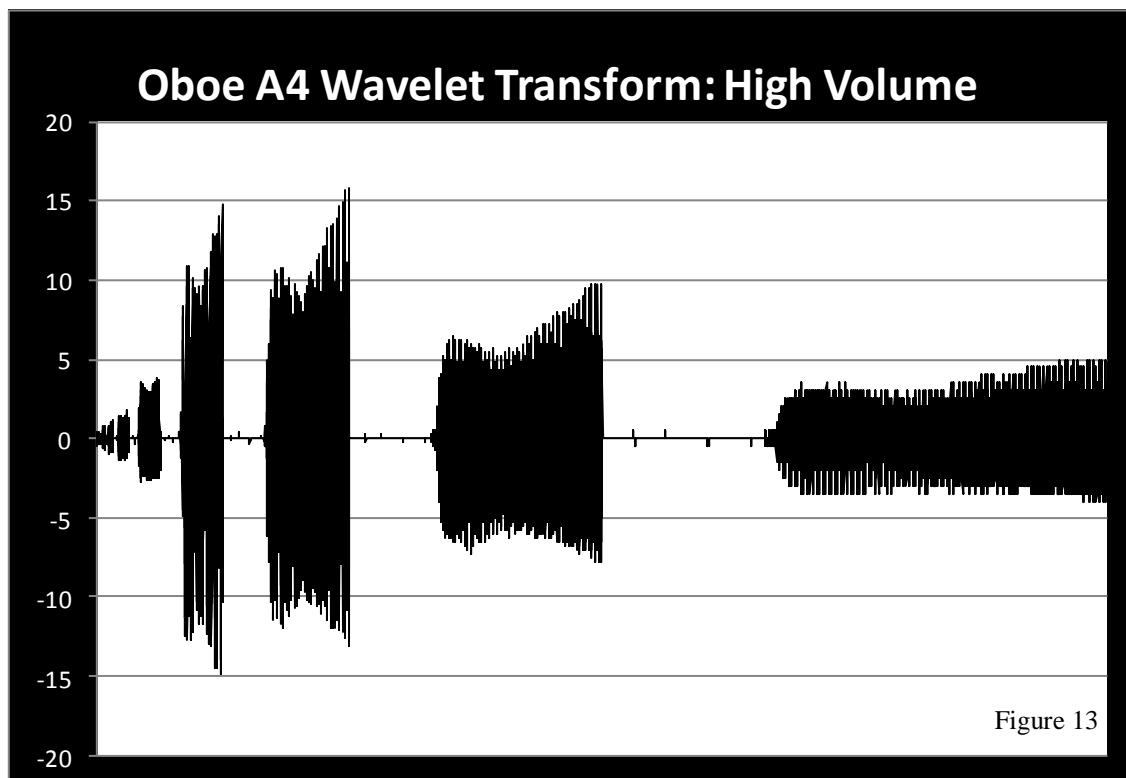


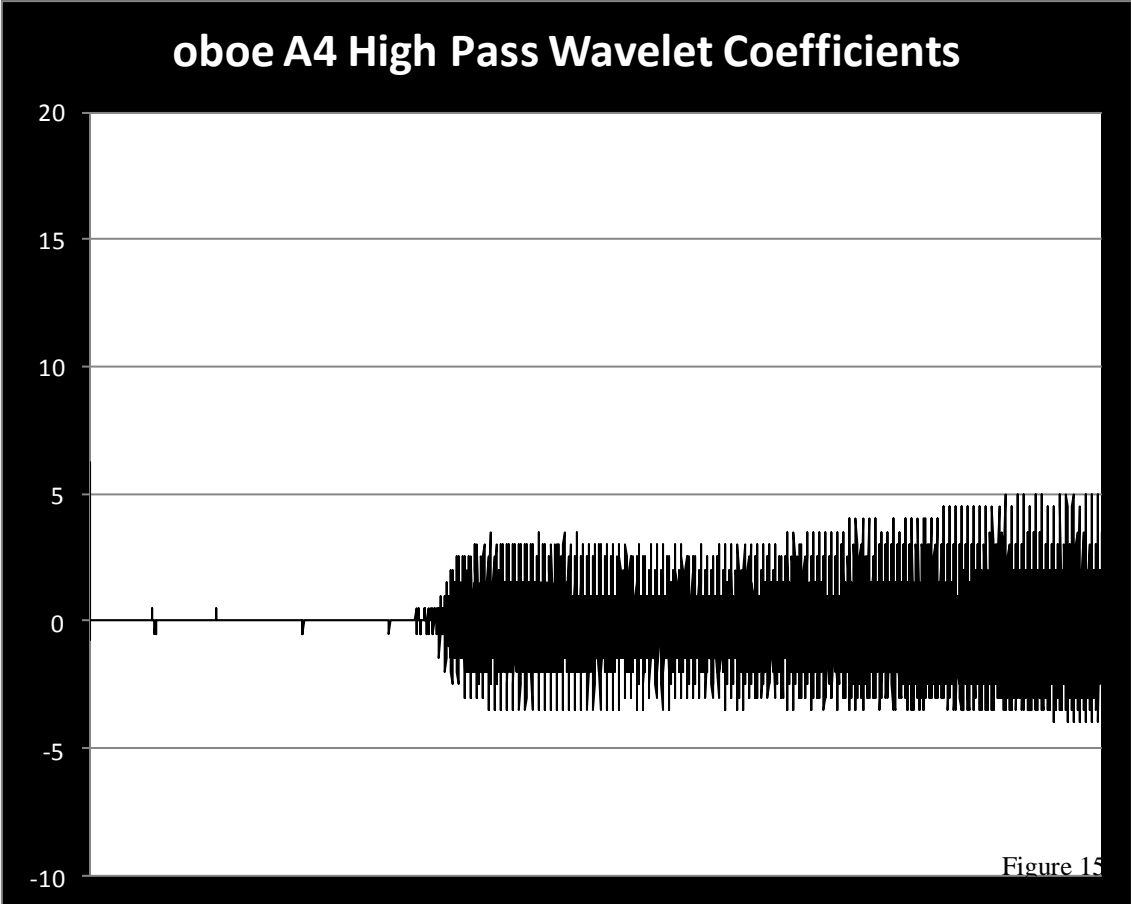
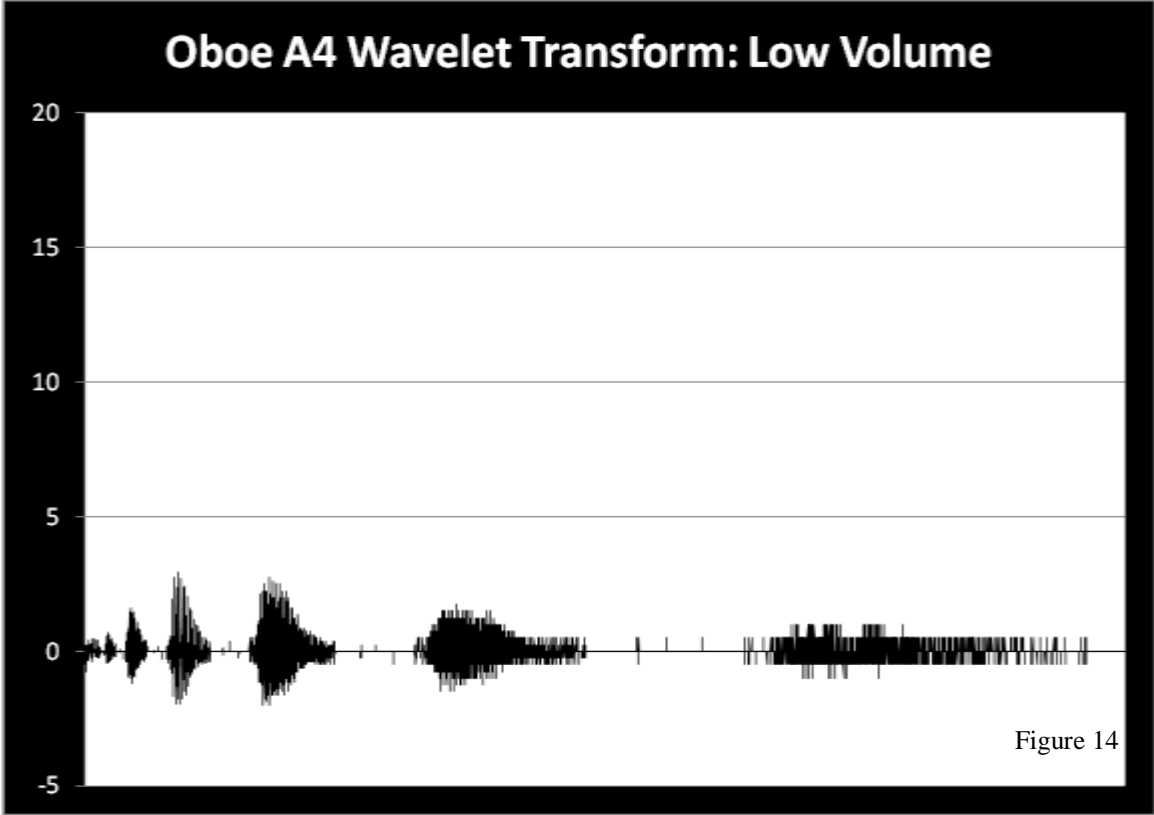
Figure 12

Wavelet Transform

The first two wavelet transforms shown represent the same note, A4=440 Hz on the same instrument, an oboe. The difference is that in figure 14, where the dynamic is piano (the second lowest dynamic used in this project), the transform has much smaller step sizes than in figure 14, where the dynamic is fortissimo (the highest in this project).

Figures 15-17 show parts of the final wavelet transform. Figure 15, the high pass filter results, contains the most detail because it is the step sizes between groups of four points. The intermediate and low pass wavelet transform results are more general and contain fewer discrete points, each of which is the average of more points for lower passes.





oboe A4 Intermediate Coefficients

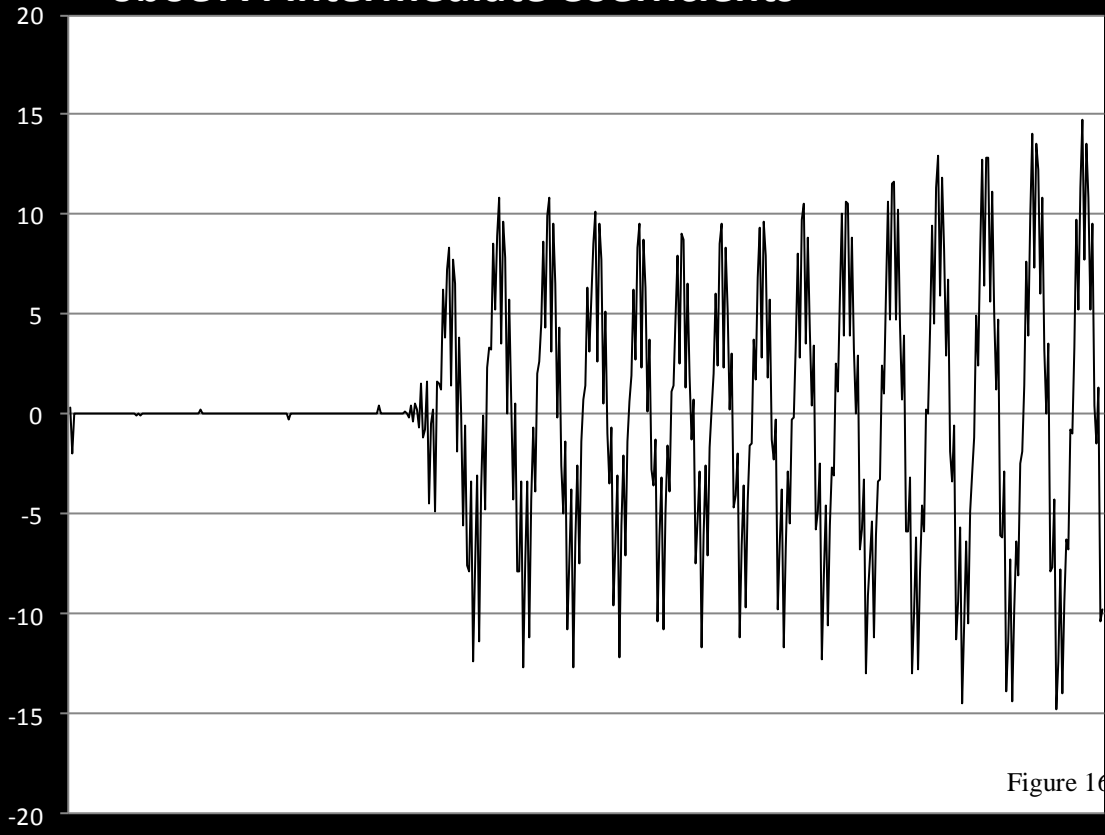


Figure 16

oboe A4 Low Pass Wavelet Coefficients

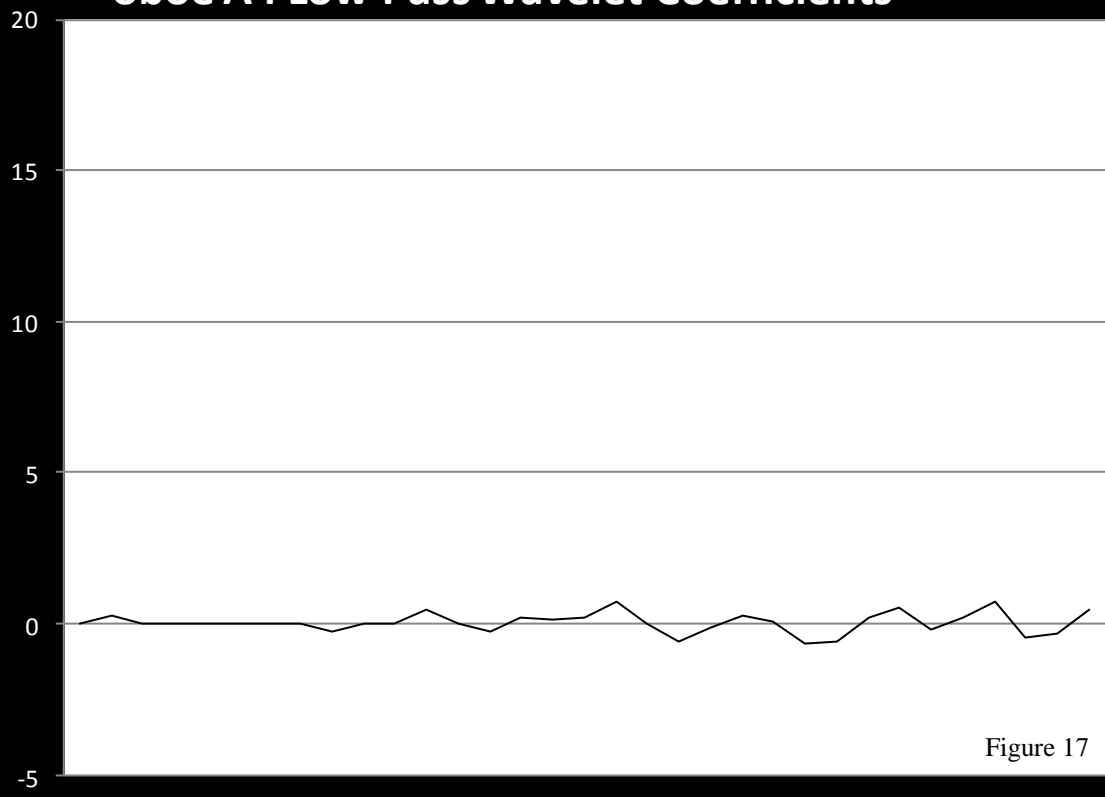


Figure 17

Analysis Program

The analysis program analyzed waveforms by using both the Fourier transform and the wavelet transform. First, gaps in the waveform were identified and if they were above a set minimum length the gaps were considered spaces between notes. Next, a Fourier transform was performed on each note. The Fourier transform of each note was analyzed and matched with the best instrument, based on the criteria in table 1. The instrument with the highest number of criteria matches for all the notes was identified as the instrument used to produce the waveform. The first peak in the Fourier transform was also identified as the fundamental frequency and the index of this peak was scaled to the sampling rate of 44100 Hz.

Next, the wavelet transform was performed on the waveform. Due to lack of computing power, the waveform had to be segmented to perform the wavelet transform, but the results are the same. The average step sizes were calculated over a single pass of the waveform. Step sizes vary in each pass, but the one used in calculations was the same pass used to determine the known volume ranges. Average step sizes of the waveform were compared to the volume ranges for the identified instrument. A very low pass filter was also used to identify any significant volume trends in the waveform.

As shown in tables 4 and 5, the analysis program was able to identify each note and its dynamic characteristics. The instrument was also correctly identified in all trials shown. The program was also able to distinguish between two notes and identify their frequencies. However, due to the computer's capacity, a longer waveform could not be analyzed. The most significant result obtained was that the algorithm had the capacity to sort peaks in the Fourier transform based on which fundamental frequency the peaks

correspond to, which allows for polyphonic waveform analysis. A java program was developed to adapt analysis results into a more practical visual format (see appendix).

Waveform Analysis Results

Instrument	Flute	Oboe	Clarinet	Trumpet
(all correctly identified)				
Ideal Frequency	Flute, A4= 440 Hz	A4=440 Hz	A4=440 Hz	A4=440 Hz
Number of Notes	1	1	1	1
Calculated Frequency	400 Hz	420 Hz	420 Hz	440 Hz
Overall Dynamic Level	Fortissimo	Fortissimo	Fortissimo	Fortissimo
Volume Trend	None	None	None	None

Table 4

Waveform Analysis Results

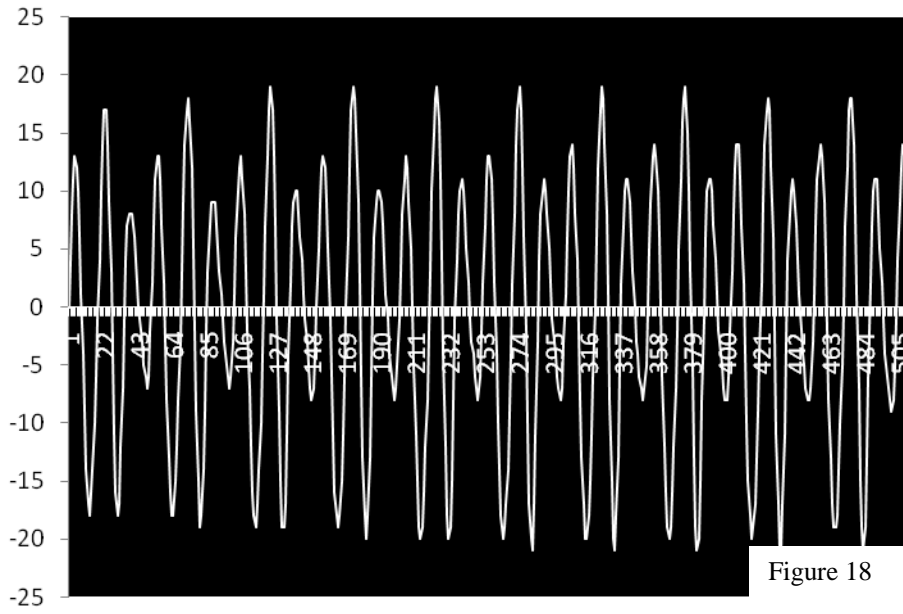
Instrument (all correctly identified)	Flute	Trumpet	Flute	Oboe and Trumpet (simultaneous)
Ideal Frequency	G5= 783.991 Hz	A5=880 Hz	G6=1567.98 Hz, A6=1760 Hz	A4=440 Hz, C5=523.251 Hz
Number of Sequential Notes	1	1	2	1
Calculated Frequency	820 Hz	850 Hz	1533 Hz, 1750 Hz	440 Hz, 523 Hz
Overall Dynamic Level	Pianissimo	Mezzo- forte	Fortissimo	Fortissimo
Volume Trend	None	Crescendo	None	None

Table 4 Continued

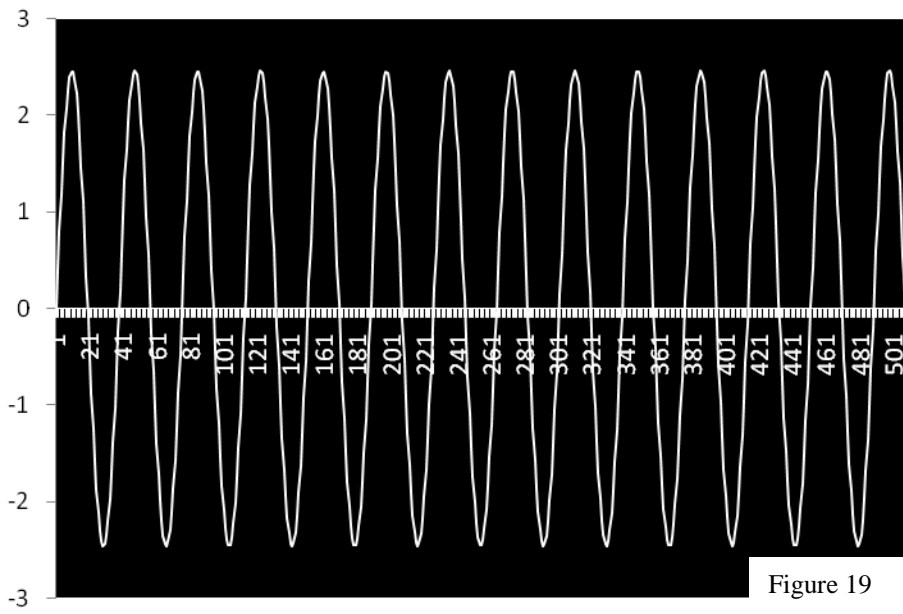
Synthesis

After determining the primary frequencies and overtones present in the sound files, the waveforms were recreated from the most prominent overtones. These new waveforms resemble the originals but are more consistent. By adding together sine waves representing each significant peak in the Fourier transform, the Inverse Fourier transform is essentially being computed on a cleaned version of the Fourier transform. The waveform of an oboe playing A4=440 Hz was re-created using one, two, three, and four frequency components. The waveform of a trumpet playing C5=523.251 Hz was created using one, two, three, four, five, and six frequency components. More frequency components were necessary for the trumpet waveform to resemble the original. This is data that corresponds to the physical complexity of the instrument's sound. The waveforms of a trumpet playing C5 and oboe playing A4 at the same time was also synthesized by combining one, two and three frequencies from each, along with four from the oboe and six from the trumpet.

Oboe A4: Original Waveform



Oboe A4: 1 Peak



Oboe A4: 2 Peaks

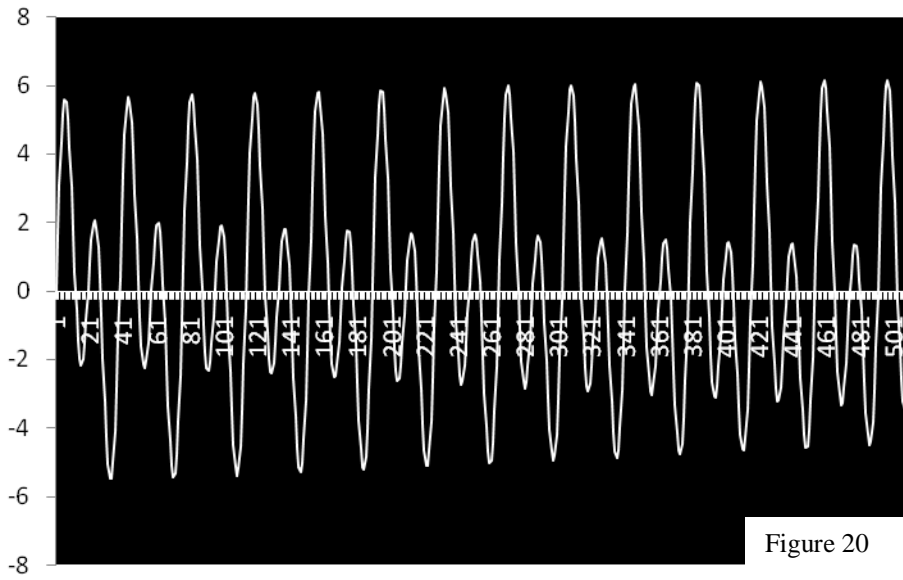


Figure 20

Oboe A4: 4 Peaks

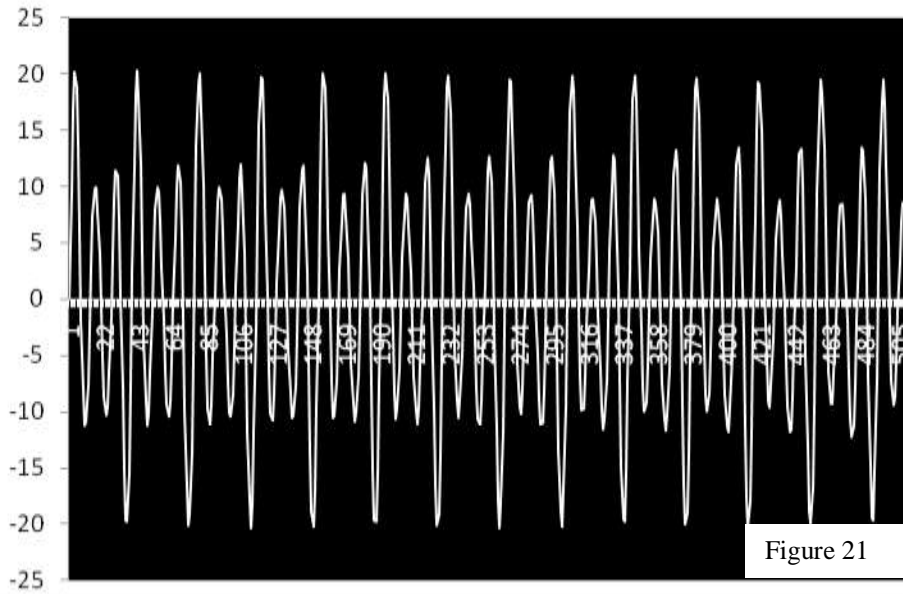


Figure 21

Trumpet C5: Original Waveform

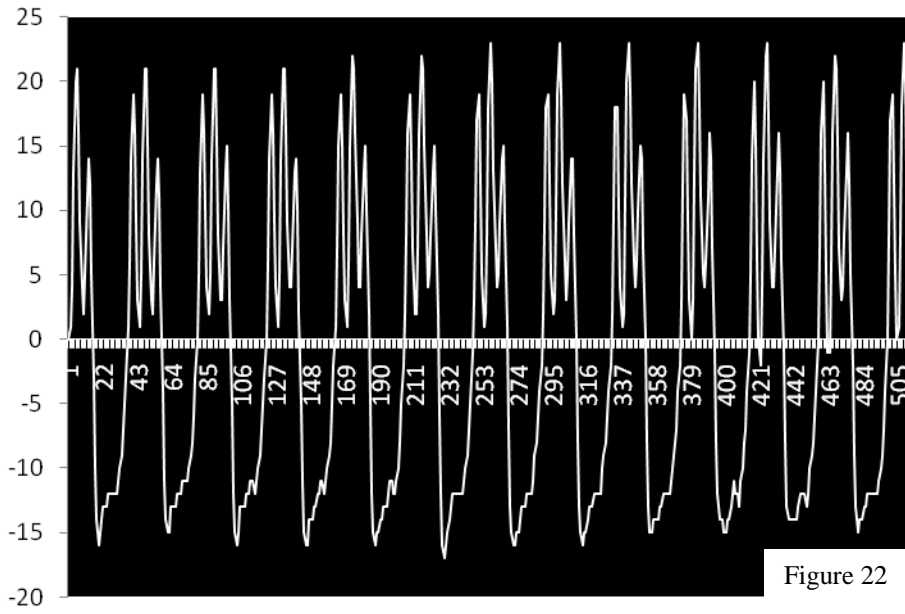


Figure 22

Trumpet C5: 1 Peak

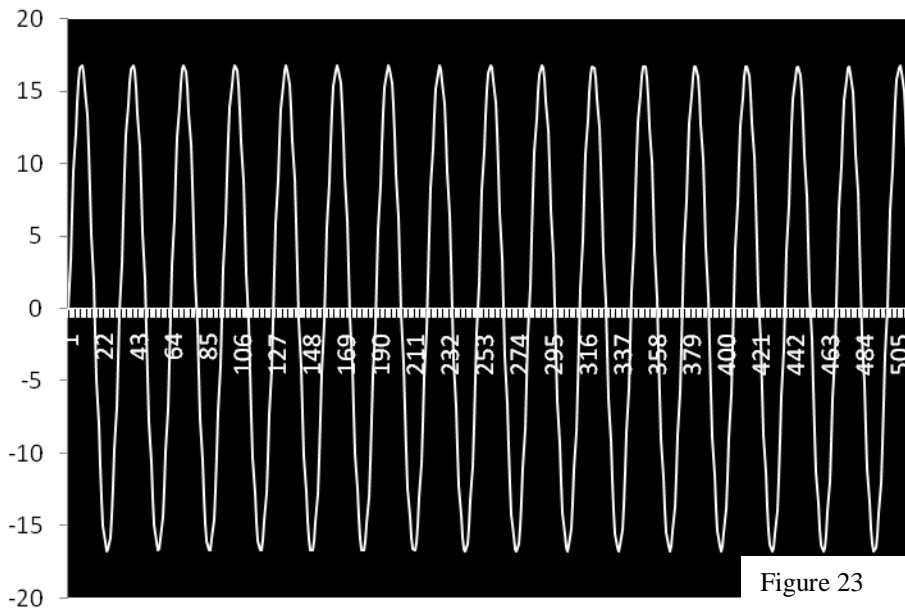


Figure 23

Trumpet C5: 3 Peaks

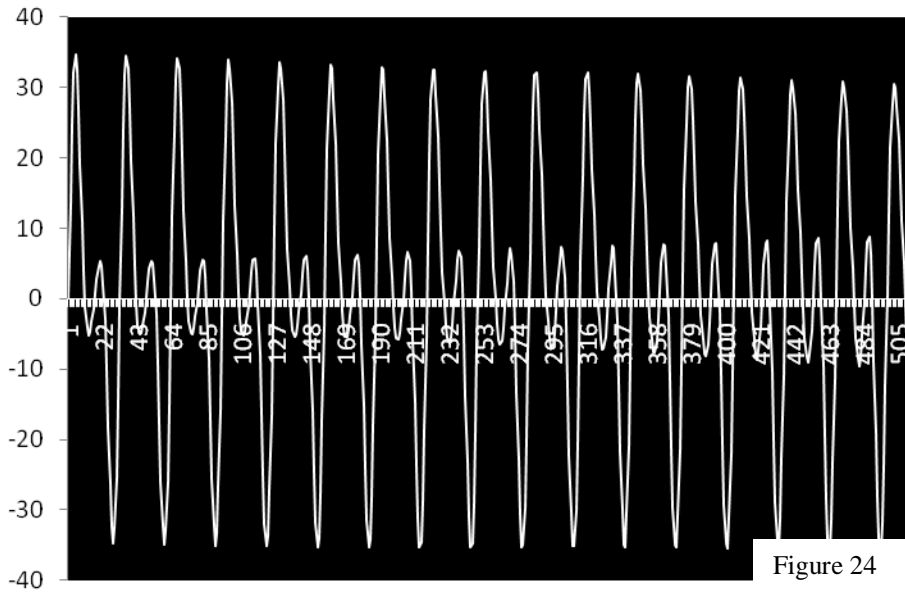


Figure 24

Trumpet C5: 6 Peaks

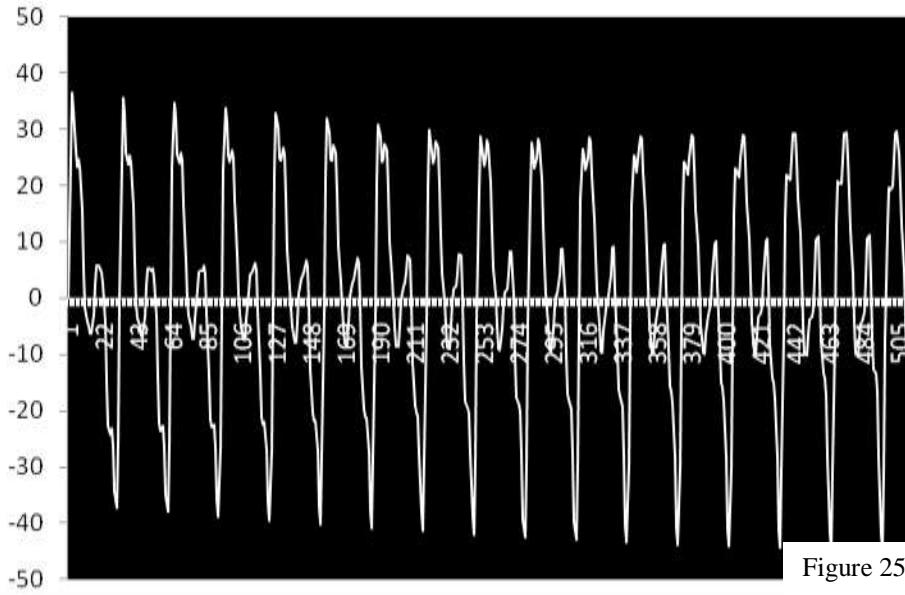


Figure 25

Oboe A4 Trumpet C5 Original Waveform

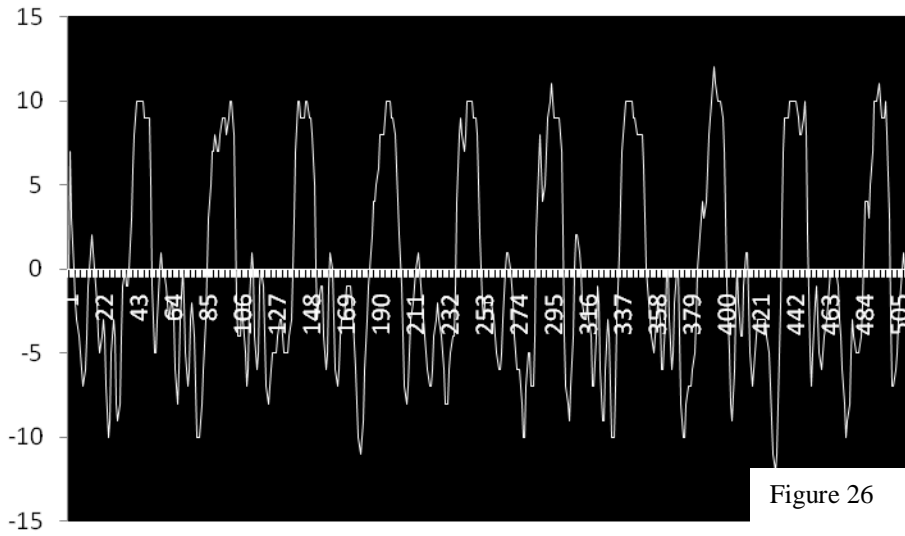


Figure 26

Oboe A4 and Trumpet C5: 1 Peak (from each)

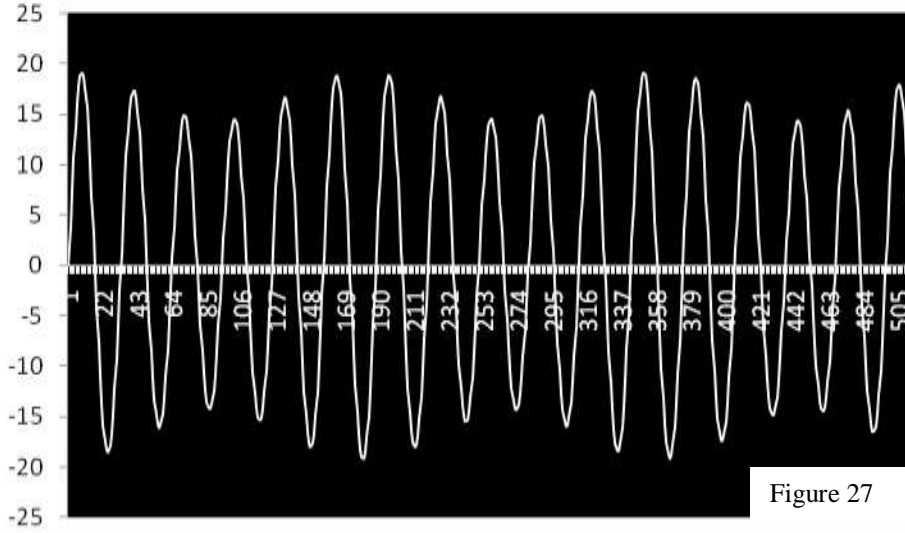
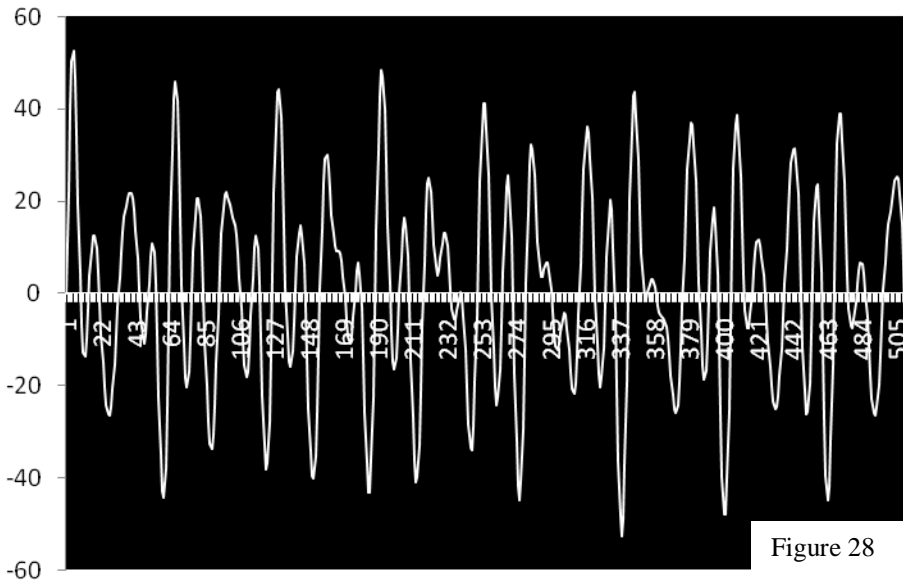
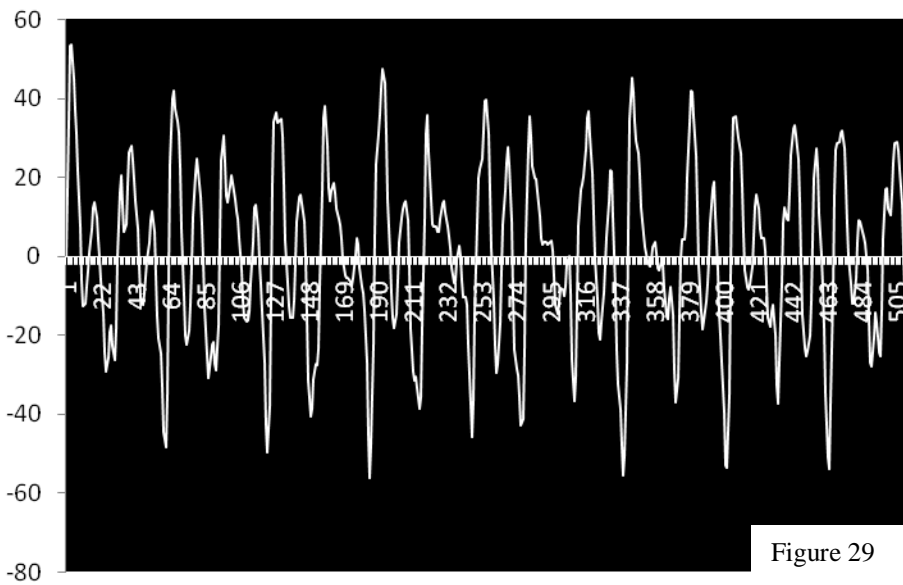


Figure 27

Oboe A4 and Trumpet C5: 3 Peaks (from each)



Oboe A4: 4 Peaks Trumpet C5: 6 Peaks



Future Work

In the future we plan on using parallelization to increase the capacity of our programs. The algorithms can handle sequential and polyphonic notes, so being able to analyze a longer waveform would mean being able to analyze an entire piece of music.

Acknowledgements

We would like to thank the following people for their helpful suggestions and feedback.

Bob Chesebrough

Nick Bennett

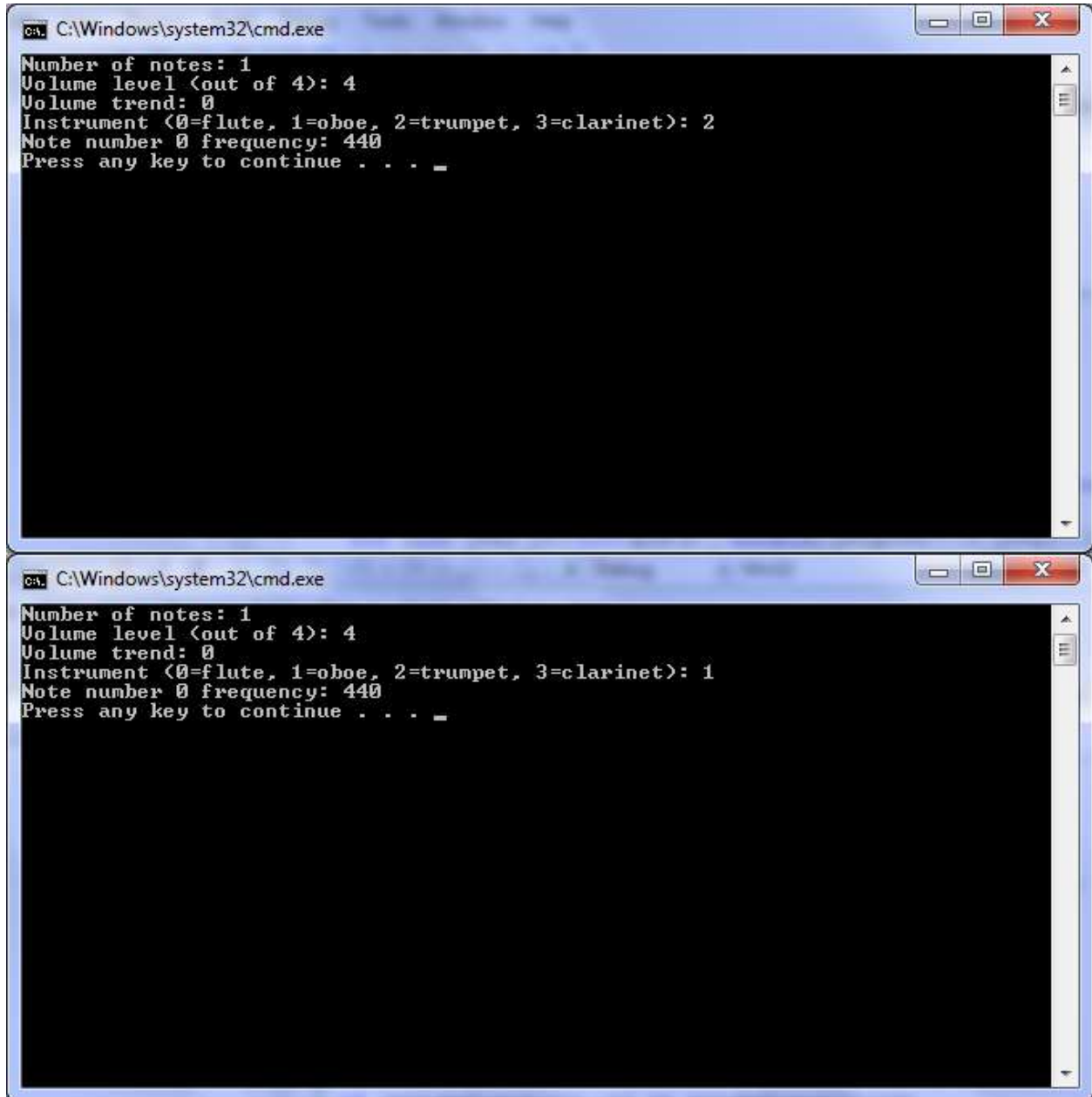
Jeffry Clymer

Bibliography

1. Clymer, Jeffrey W., 2010 High Brass Overtones
2. Graps, Amara., 1995, An Introduction to Wavelets, Institute of Electrical and Electronics Engineers, Inc. 18 p.
3. Hubbard, B. B., 1998, The World According to Wavelets, Second Edition, A K Peters, Ltd. 330 p.
4. Howe, Hubert S., 1975, Electronic Music Synthesis: Concepts, Facilities, Techniques, W. W. Norton & Company, Inc. 272 p.
5. Lynch, John T., 2001, The Natural Resonances of a Trumpet.
6. Ogden, R. T., 1997, Essential Wavelets for Statistical Applications and Data Analysis, Birkhauser Boston, 206 p.
7. Philharmonia Orchestra, The Sound Exchange. Sample Library
8. Press, W. H., Teukolosky, S. A., Vetterling, W. T., Flannery, B. P., 2007, Numerical Recipes: The Art of Scientific Computing, Third Edition, Cambridge University Press, 1235 p.
9. Yoo, Y., 2001, Tutorial on Fourier Theory, 18 p.

Appendix A: Data

Screen shots of Analysis Output



The image displays two screenshots of a Windows command prompt window, each showing the output of a program. The window title is "C:\Windows\system32\cmd.exe".

The top screenshot shows the following output:

```
Number of notes: 1
Volume level (out of 4): 4
Volume trend: 0
Instrument (0=flute, 1=oboe, 2=trumpet, 3=clarinet): 2
Note number 0 frequency: 440
Press any key to continue . . . _
```

The bottom screenshot shows the following output:

```
Number of notes: 1
Volume level (out of 4): 4
Volume trend: 0
Instrument (0=flute, 1=oboe, 2=trumpet, 3=clarinet): 1
Note number 0 frequency: 440
Press any key to continue . . . _
```



```
C:\Windows\system32\cmd.exe
Number of notes: 1
Volume level (out of 4): 4
Volume trend: 0
Instrument (0=flute, 1=oboe, 2=trumpet, 3=clarinet): 3
Note number 0 frequency: 440
Press any key to continue . . .
```

```
C:\Windows\system32\cmd.exe
Number of notes: 1
Volume level (out of 4): 4
Volume trend: -40
Instrument (0=flute, 1=oboe, 2=trumpet, 3=clarinet): 0
Note number 0 frequency: 440
Press any key to continue . . .
```

```
C:\Windows\system32\cmd.exe
Number of notes: 1
Volume level (out of 4): 2
Volume trend: 14
Instrument (0=flute, 1=oboe, 2=trumpet, 3=clarinet): 1 2
Frequencies: 440.001 523.252
note length: 1
Press any key to continue . . .
```

Java Musical Notation Output

Graph of .wav File

oboe




f

This block shows a window titled "Graph of .wav File" containing a musical staff. The staff has five lines. A single note is placed on the second line from the bottom. Below the staff, the dynamic marking "f" is written.

Graph of .wav File

trumpet



f

This block shows a window titled "Graph of .wav File" containing a musical staff. The staff has five lines. A single note is placed on the second line from the bottom. Below the staff, the dynamic marking "f" is written.

flute



f

clarinet



f

Appendix B: Code

Analysis Program

```
#include <iostream>
#include "waveAnalysis.h"
#include "fourier.h"
#include "linearWavelets.h"
#include <fstream>
#include <string>
#include <math.h>
#include <stdio.h>

using namespace std;
int main(int argc, char** argv)
{
    ifstream infile;
    infile.open("infile.txt");

    ofstream outfile("transform_analysis.txt");

    string line;

    const int length=8192;

    const int segNum=3;
    double data[2*length];
    double segs[segNum][length];
    //segments divide where notes start and end
    double transform[length];
    int seglength[segNum];
    double dataTemp[length];
    double transformData[segNum][5];
    int ifouriermax;
    int tempLength=0;
    int threePeaks=0;
    int segpow2=0;
    int singleCounter=0;
    int countHoles=0;
    int segmentcount=0;
    int j=0;
    int lengthtemp=0;

    double peak1=0;
    double peak2=0;
    double peak3=0;
    double peakSum1=0;
    double peakSum2=0;
    double peakSum3=0;
    int peakCount=0;
    double peakPrev=0;
    int upDown=0;
    double fundamentalFreq[segNum];
```

```

double fundamentalFreqAvg=0;
double inputTemp;
double scaleK=0;
double outCount=0;
double peaks[30];
const int notesinRange=100;
double notes[notesinRange];
int totalPeakprev=0;
int totalPeaks=0;
double peakSeparate[2][15];
//will contain all peaks, sorted by corresponding fundamental frequency
int instrument1count=0;
int instrument2count=0;
int roundedFreq=0;
int roundedFreqPrev=0;

//define note ranges
int octaveCount=0;
int octave23=0;
int noteCountersingle=0;

for (int i=0; i<notesinRange; i+=2)
{
    notes[noteCountersingle]=261.6262*pow(1.059463094, (double) (i));
    octaveCount+=1;
    noteCountersingle+=1;
    if (octave23==0 && octaveCount==3)
    {
        octave23=1;
        octaveCount=0;
        i=i-1;
    }
    if (octave23==1 && octaveCount==4)
    {
        octave23=0;
        octaveCount=0;
        i=i-1;
    }
}

for (int i=0; i<length; i++)
//thresholding- holes show where notes end
{

    getline(infile, line);
    inputTemp=atoi(line.c_str());
    data[i]=inputTemp;

    if(data[i]<2 && data[i]>-2)
    //excludes preset zeros
    {
        dataTemp[i]=0;
    }
    if(data[i]>2 || data[i]<-2)

```

```

        {
            dataTemp[i]=data[i];
        }
        dataTemp[i]=data[i];
        if (dataTemp[i]==0 && (dataTemp[i-1]==0 || dataTemp[i-1]==0) &&
(dataTemp[i-2]==0 || dataTemp[i-2]==0))
        {

            countHoles+=1;

        }

        if (dataTemp[i]!=0)
        {
            if (countHoles>500)
            {

                seglength[segmentcount]=j;
                segmentcount+=1;
                j=0;
                countHoles=0;

            }

        }
        if (i==(length-1))
        {
            seglength[segmentcount]=j;
        }
        segs[segmentcount][j]=data[i];
        if (data[i]!=0)
            j+=1;
    }

segmentcount+=1;
if (segmentcount==1)
    seglength[0]=length;
//if all one segment first segment is total length

//fourier segments
for (int i=0; i<segmentcount; i++)
{

    segpow2=(int) (log((long double) seglength[i])/log((long double)2));
    singleCounter=seglength[i]/4;
    cout<<"segpow2: "<<segpow2<<endl;
    for (int j=0; j<((int)pow(2.,(double) (segpow2+1))); j+=2)
        //length thats a power of two, double so that it alternates with 0s.
    {
        dataTemp[j]=segs[i][singleCounter];
        //takes middle half, cleanest periodic part of note
        dataTemp[j+1]=0;
        singleCounter+=1;
    }
}

```

```

}

ifouriermax=fourier(dataTemp,
transform, ((int)pow(2., (double) (segpow2))));
//transform each piece-return into transform

for (int k=0; k<((int)pow(2., (double) (segpow2))); k++)
//through all of fourier transform
{
    outfile<<transform[k]<<endl;
    if (transform[k]<ifouriermax/4)
    {
        transform[k]=0;
        //weed out data below a fraction of the max
    }

    if (transform[k]>transform[k-1] && transform[k]>transform[k+1])
    {

        scaleK=k*(44100/((int)pow(2., (double) (segpow2))/2));

        for (int m=0; m<notesinRange; m++)
        {

            if(scaleK<notes[m+1] && scaleK>notes[m])
            {
                if (abs(scaleK-notes[m])<abs(scaleK-
notes[m+1]))
                {
                    peaks[totalPeaks]=notes[m];
                }
                else
                {
                    peaks[totalPeaks]=notes[m+1];
                    cout<<"peak true."<<endl;
                }

                if (peaks[totalPeaks]==peaks[totalPeaks-1])
                //if to close, accounts for peak missed in threshold
                    peaks[totalPeaks]=0;

            }

        }

        if (peaks[totalPeaks]!=0)
        {

            totalPeaks+=1;
        }

        if (peakCount==0)
        {

```

```

        peak1=transform[k];
        peakSum2+=peak1;
        peakSum3+=peak1;
        //calculate frequency
fundamentalFreq[i]=k*(44100/((int)pow(2.,(double)(segpow2)/2)));
        cout<<"fundamental peak: "<<k<<endl;
        fundamentalFreqAvg+=fundamentalFreq[i];
        cout<<"fundamentalFreq: "<<fundamentalFreq[i]<<endl;

        peakPrev=peak1;
    }
    if (peakCount==1)
    {

        peak2=transform[k];
        cout<<"peak2: "<<peak2<<endl;
        peakSum1+=peak2;
        peakSum3+=peak2;

        if (peak2>peakPrev)
        {
            upDown=1;
        }
        if (peak2<peakPrev)
        {
            upDown=-1;
        }

        peakPrev=peak2;
    }
    if (peakCount==2)
    {

        peak3=transform[k];
        cout<<"peak3: "<<peak3<<endl;
        peakSum1+=peak3;
        peakSum2+=peak3;
        if (peak3>peakPrev)
            upDown+=1;
        if (peak3<peakPrev)
            upDown-=1;
        peakPrev=peak3;
    }

    peakCount+=1;
}
else
{
transform[k]=0;
//cut out to peaks only above threshold
}
}
}

```



```

//end of loop through fourier data

for (int j=0; j<totalPeaks; j++)
{
    if ((int) (peaks [j]/peaks [0]) != (int) ((peaks [j]/peaks [0])+0.9))
        roundedFreq=(int) ((peaks [j]/peaks [0])+0.9);
    else
        roundedFreq=(int) (peaks [j]/peaks [0]);

    if (roundedFreq!=roundedFreqPrev)
    {
        peakSeparate [0] [instrument1count]=peaks [j];
        instrument1count+=1;
        cout<<"peakSep [0]:"<<peakSeparate [0] [instrument1count]<<" ";

    }

    else
    {
        outCount+=1;
        if (abs (peaks [j]/peaks [0]-roundedFreq)>abs (peaks [j-
        1]/peaks [0]-roundedFreq))
            //if previous is closer to ratio than current peak
            {
                peakSeparate [1] [instrument2count]=peaks [j];
                cout<<"peakSep [0]:"
                "<<peakSeparate [0] [instrument1count]<<" ";
                cout<<"peakSep [1]:"
                "<<peakSeparate [1] [instrument1count]<<endl;

                instrument2count+=1;
            }
            else
            {
                peakSeparate [0] [instrument1count-1]=peaks [j];
                peakSeparate [1] [instrument2count]=peaks [j-1];
                cout<<"peakSep [0]:"
                "<<peakSeparate [0] [instrument1count]<<" ";
                cout<<"peakSep [1]:"
                "<<peakSeparate [1] [instrument1count]<<endl;
                instrument1count+=1;
            }
        }
        roundedFreqPrev=roundedFreq;
    }

}

if (peakCount>2)
{
    transformData [i] [0]=peak1/(peakSum1/(peakCount-1));
    //ratio of first peak to average of rest
    transformData [i] [2]=peak2/(peakSum2/(peakCount-1));
}

```

```

        transformData[i][3]=peak3/(peakSum3/(peakCount-1));
    }
    else
    {
        transformData[i][0]=peak1/(peakSum1);
        transformData[i][2]=peak2/(peakSum2);
        transformData[i][3]=peak3/(peakSum3);
    }

    if (upDown>0)
    //general up or down trend
        transformData[i][1]=1;
    if (upDown<0)
        transformData[i][1]=-1;
    if (upDown==0)
        transformData[i][1]=0;

    peak1=0;
    peak2=0;
    peak3=0;
    peakSum1=0;
    peakSum2=0;
    peakSum3=0;
    peakCount=0;
    peakPrev=0;
    upDown=0;
}
//end of loop through all notes

//combine information from each note
const int instrumentCount=4;
const int factors=5;
double instruments[instrumentCount][factors];
const int chunklength=2048;
const int waveseqs=length/chunklength;
const int sweeps=(int) (log((long double) chunklength)/log((long double) 2));
//flute data
instruments[0][0]=1.75;
instruments[0][1]=0;
instruments[0][2]=1.25;
instruments[0][3]=0.8;
instruments[0][4]=1.4;

//oboe data
instruments[1][0]=0.30;
instruments[1][1]=1;
instruments[1][2]=1.55;
instruments[1][3]=2.0;
instruments[1][4]=1.0;

//trumpet data
instruments[2][0]=3.80;
instruments[2][1]=-1;
instruments[2][2]=0.5;

```

```

instruments[2][3]=0.3;
instruments[2][4]=1.7;

//clarinet data

instruments[3][0]=2.9;
instruments[3][1]=-1;
instruments[3][2]=0.4;
instruments[3][3]=1.0;
instruments[3][4]=0.7;

double ratioAvg1=0;
double ratioAvg2=0;
double ratioAvg3=0;
double upDownTotal=0;

for (int i=0; i< segmentcount; i++)
{
    ratioAvg1+=transformData[i][0];
    ratioAvg2+=transformData[i][2];
    ratioAvg3+=transformData[i][3];
    upDownTotal+=transformData[i][1];
}

upDownTotal=0;
if (upDownTotal>0)
    upDownTotal=1;
if (upDownTotal<0)
    upDownTotal=-1;
if (upDownTotal==0)
    upDownTotal=0;

    ratioAvg1=ratioAvg1/(segmentcount);
ratioAvg2=ratioAvg2/(segmentcount);
ratioAvg3=ratioAvg3/(segmentcount);

//make comparisons to instruments
int instrumentPoints[instrumentCount];
double diffSums[instrumentCount];
double diffSumMin=1000;
int diffSumMinindex=0;
int closestRatioIndex1=0;
int closestRatioIndex2=0;
int closestRatioIndex3=0;
for (int i=0; i<instrumentCount; i++)
{
    diffSums[i]=0;
    instrumentPoints[i]=0;

    if (upDownTotal==instruments[i][1])
//give "point" for same trend
        instrumentPoints[i]+=8;

```

```

diffSums[i]+=abs(ratioAvg1-instruments[i][0]);
diffSums[i]+=abs(ratioAvg1-instruments[i][2]);
diffSums[i]+=abs(ratioAvg1-instruments[i][3]);
if (diffSums[i]<diffSumMin)
{
    diffSumMinindex=i;
    diffSumMin=diffSums[i];
}
}

instrumentPoints[diffSumMinindex]+=5;

double positiveMax=-100;
double negativeMin=1000;
double plusMinusRatio=0;
double waveletClosest=100;
int waveletIndex=0;

for (int i=0; i<wavesegs; i++)
{
    for (int j=0; j<chunklength; j++)
    {
        dataTemp[j]=data[j+(chunklength*i)];
    }
    linearWavelets(dataTemp, transform, chunklength,sweeps);
//use of wavelet characteristics to identify instrument

    for (int g=(chunklength/32); g<(chunklength/16); g++)
    {
        dataTemp[g-(chunklength/32)]=transform[g];
        if (transform[g]>positiveMax)
            positiveMax=transform[g];
        if (transform[g]<negativeMin)
            negativeMin=transform[g];
    }
}

plusMinusRatio=abs(positiveMax)/abs(negativeMin);
for (int i=0; i<instrumentCount; i++)
{
    if (abs(plusMinusRatio-instruments[i][4])<waveletClosest)
    {
        waveletClosest=abs(plusMinusRatio-instruments[i][4]);
        waveletIndex=i;
    }
}
instrumentPoints[waveletIndex]+=2;

int bestInstrument=0;
//set to closest instrument by finding instrument with most points
for (int i=0; i<instrumentCount; i++)

```

```

    {
        if (instrumentPoints[i]>instrumentPoints[bestInstrument])
        {
            bestInstrument=i;
        }
    }

    const int numVolumeLevels=6;
    double instrumentAmps[instrumentCount][numVolumeLevels];
    double amplitudeAverage[wavesegs];
    int amplitudeAverageCount[wavesegs];
    int volumeLevel[wavesegs];
    fundamentalFreqAvg=fundamentalFreqAvg/segmentcount;
    int volumeTrend[wavesegs];
    int volumeLevelAvg=0;

//flute dynamics
instrumentAmps[0][0]=0.0007; //pp
instrumentAmps[0][1]=0.00125; //mp
instrumentAmps[0][2]=0.0018; //mf
instrumentAmps[0][3]=0.00238; //f
instrumentAmps[0][4]=0.0029; //ff
instrumentAmps[0][5]=1; //top bound

//oboe dynamics
instrumentAmps[1][0]=0.000225; // p
instrumentAmps[1][1]=0.0002863; //mp
instrumentAmps[1][2]=0.0003475; //mf
instrumentAmps[1][3]=0.00040875; //f
instrumentAmps[1][4]=0.00047; //ff
instrumentAmps[1][5]=1; //topbound

//trumpet dynamics
instrumentAmps[2][0]=0.0008; // p
instrumentAmps[2][1]=0.00165; //mp
instrumentAmps[2][2]=0.0025; //mf
instrumentAmps[2][3]=0.00335; //f
instrumentAmps[2][4]=0.0042; //ff
instrumentAmps[2][5]=0.012; //topbound

//clarinet dynamics
instrumentAmps[3][0]=0.0037; // p
instrumentAmps[3][1]=0.00385; //mp
instrumentAmps[3][2]=0.004; //mf
instrumentAmps[3][3]=0.00415; //f
instrumentAmps[3][4]=0.0043; //ff
instrumentAmps[3][5]=1; //topbound

for (int i=0; i<wavesegs; i++)
{
    amplitudeAverage[i]=0;
    amplitudeAverageCount[i]=0;
    volumeTrend[i]=0;
    for (int j=0; j<chunklength; j++)

```

```

    {
        dataTemp[j]=data[j+(chunklength*i)];
    }
    linearWavelets(dataTemp, transform, chunklength,sweeps);

    for (int k=(chunklength/8); k<(chunklength/4); k++)
//a level of the wavelet transform
    {
        dataTemp[k-(chunklength/8)]=abs(transform[k]);

        if (dataTemp[k-(chunklength/8)]!=0 && dataTemp[k-
(chunklength/8)]!=-1)
//over boundary- so doesn't include between notes
        {
            amplitudeAverage[i]+=dataTemp[k-(chunklength/8)];

            amplitudeAverageCount[i]+=1;

        }

    }

    amplitudeAverage[i]=(amplitudeAverage[i]/amplitudeAverageCount[i])/fund
amentalFreqAvg;

    volumeLevel[i]=0;
    for (int m=0; m<(numVolumeLevels-1); m++)
    {
        if (amplitudeAverage[i]>instrumentAmps[bestInstrument][m] &&
            amplitudeAverage[i]<instrumentAmps[bestInstrument][m+1])
            volumeLevel[i]=m;
//best range-range is space above set mark
        volumeLevelAvg+=volumeLevel[i];
    }

    for (int g=(chunklength/32); g<(chunklength/16); g++)
    {
        dataTemp[g-(chunklength/32)]=abs(transform[g]);

        if (dataTemp[g-(chunklength/32)]>dataTemp[g-
(chunklength/32)-1])
        {
            volumeTrend[i]+=1;
        }
        if (dataTemp[g-(chunklength/32)]<dataTemp[g-
(chunklength/32)-1])
        {
            volumeTrend[i]-=1;
        }

    }
}

```

```

    }

    if (volumeLevel[0]==volumeLevel[wavesegs-1])
    {
        int x=1;
    }
    else
    {
        for (int i=1; i<wavesegs; i++)
        {
            volumeTrend[0]+=volumeTrend[i];
        }
    }

    //write data to file
    outfile<<segmentcount<<endl;
    cout<<"Number of notes: "<<segmentcount<<endl;
    volumeLevelAvg=volumeLevelAvg/wavesegs;
    outfile<<volumeLevelAvg<<endl;
    cout<<"Volume level (out of 4): "<<volumeLevelAvg<<endl;
    if (volumeTrend[0]> -10 && volumeTrend[0]<10)
        volumeTrend[0]=0;
    outfile<<volumeTrend[0]<<endl;
    cout<<"Volume trend: "<<volumeTrend[0]<<endl;
    outfile<<bestInstrument<<endl;
    cout<<"Instrument (0=flute, 1=oboe, 2=trumpet, 3=clarinet):
"<<bestInstrument<<endl;
    int smallestNote=100000000;
    for (int i=0; i<segmentcount; i++)
    {
        if (seglength[i]<smallestNote)
            smallestNote=seglength[i];
    }
    for (int i=0; i<segmentcount; i++)
    {
        outfile<<(int) fundamentalFreq[i]<<endl;
        cout<<"Note number "<<(i+1)<<"
frequency:"<<(int) fundamentalFreq[i]<<endl;
        outfile<<1<<endl;
        cout<<"note length: "<<seglength[i]<<
(int) (seglength[i]/smallestNote)<<endl;
    }

    return 0;
}

```

Fourier Transform Program

```
#include <cmath>
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <fstream>
#include <string>
#define SWAP(a,b) tempr=(a); (a)=(b); (b)=tempr
#define _USE_MATH_DEFINES
#include <math.h>
using namespace std;
int fourier (double* dataComplex, double* transform, int n)
{
    const int isign=-1;
    //const int n=256;
    unsigned long mmax,m,j,istep,i;
    double wtemp,wr,wpr,wpi,wi,theta,tempr,tempi;
    double swapTemp;
    n=n/2;
    int number_of_complex_samples=n/2;
    j=0;

    for (int i=0;i<n/2;i+=2)
    {
        if (j > i)
        {
            //swap the real part
            SWAP(dataComplex[j],dataComplex[i]);
            //swap the complex part
            SWAP(dataComplex[j+1],dataComplex[i+1]);
        }

        m=n/2;
        while (m >= 2 && j >= m)
        {
            j -= m;
            m = m/2;
        }
        j += m;

        mmax=2;
        //external loop
        while (n > mmax)
        {
            istep = mmax<< 1;
            theta=isign*(2*3.14159265358979323/mmax);
            wtemp=sin(0.5*theta);
            wpr = -2.0*wtemp*wtemp;
            wpi=sin(theta);
            wr=1.0;
            wi=0.0;
            //internal loops
            for (m=1;m<mmax;m+=2)
```



```

    {
    for (i= m;i<=n;i+=istep)
    {
        j=i+mmax;
        tempr=wr*dataComplex[j-1]-wi*dataComplex[j];
        tempi=wr*dataComplex[j]+wi*dataComplex[j-1];
        dataComplex[j-1]=dataComplex[i-1]-tempr;
        dataComplex[j]=dataComplex[i]-tempi;
        dataComplex[i-1] += tempr;
        dataComplex[i] += tempi;
    }
    wtemp=wr;
    wr=wtemp*wpr-wi*wpi+wr;
    wi=wi*wpr+wtemp*wpi+wi;
}
mmax=istep;
}

int datamax=0;
int imax=0;
int r=0;
for (int i=0; i<(n/2); i+=2)
{
    transform[r]=sqrt((dataComplex[i]*dataComplex[i])+(dataComplex[i+1]*dataComplex[i+1]));
    transform[0]=0;
    if (transform[r]>datamax)
    {
        datamax=transform[r];
        imax=datamax;
    }
    r+=1;
}
return imax;
}

```

Wavelet Transform Program

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <vector>
#include <cstdlib>
#include <cmath>
using namespace std;

void linearWavelets(double* data, double* transform, int n, int sweeps)
{
    const int size=19;
    const int length=2048;
    double cof[size][length];
    int k=0;
    for (int i=0; i<length; i++)
        cof[0][i]=data[i];
    for (int j=1; j<=sweeps; j++)
    {
        k=0;
        for (int i=0; i<(length*2); i+=2)
        {
            if (i<((length/(int)pow(2.,(double)(j-1))))))
            {
                cof[j][k]=(cof[j-1][i]+cof[j-1][i+1])/2;
                cof[j][k+(length/(int)pow(2.,(double)j))]=(cof[j-1][i]-
                    cof[j-1][i+1])/2;
            }
            else
            {
                cof[j][k+(length/(int)pow(2.,(double)j))]=cof[j-1][
                    k+(length/(int)pow(2.,(double)j))];
            }

            k+=1;
        }
    }
    for (int i=0; i<length; i++)
        transform[i]=cof[sweeps][i];
}
```

Java Sheet Music Notation Program

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;
import java.util.*;
import java.lang.Math.*;
import java.io.*;
class main extends JFrame
{

    public static void main( String[] args )
    {
        main graphicsObject = new main();
        graphicsObject.setBackground(Color.white);
        graphicsObject.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    }
    main()
    {
        super("Graph of .wav File");
        setSize(1300,1200);
        setVisible(true);
    }
    public void paint (Graphics g)
    {
        Graphics2D g2d = (Graphics2D) g;
        g2d.translate (0.0,500.0); //make 0 where graph is 0,0
        g2d.scale (1.0, -1.0); //make positive up, to the right
        g.setColor(Color.black);
        g.drawLine(50, 0, 1225, 0); //horizontal lines
        g.drawLine(50, 25,1225, 25);
        g.drawLine(50, 50,1225, 50);
        g.drawLine(50, 75,1225, 75);
        g.drawLine(50, 100,1225, 100);
        //***** importing
        double max=0;
        double min=100000;
        int dataCount=0;
        int oldX;
        int oldY=0;
        int size;
        int secondRow=0;

        try{
            // Open the file that is the first
            FileInputStream fstream = new
            FileInputStream("waveform_analysis.txt");
            // Get the object of DataInputStream
            DataInputStream in = new DataInputStream(fstream);
            BufferedReader br = new BufferedReader(new
            InputStreamReader(in));

            //Read File Line By Line
            String strLine;
            size=15;
```

```

    int numberNotes;
    int trend=0;
    int startVolume=10;
    int endVolume=10;
    int averageVolume;
    int bestInstrument;
    int instrument;
    int smallestIndex;
    int[] relativeLengths=new int[15];
    int[] frequencies=new int[15];

    strLine=br.readLine();
    numberNotes=Integer.parseInt(strLine);

    strLine=br.readLine();
    averageVolume=Integer.parseInt(strLine);

    strLine=br.readLine();
    if (Integer.parseInt(strLine)>0) //trend up
    {
        trend=1;
        strLine=br.readLine();
        startVolume=Integer.parseInt(strLine);
        strLine=br.readLine();
        endVolume=Integer.parseInt(strLine);
    }
    if (Integer.parseInt(strLine)<0) //trend down
    {
        trend=-1;
        strLine=br.readLine();
        startVolume=Integer.parseInt(strLine);
        strLine=br.readLine();
        endVolume=Integer.parseInt(strLine);
    }
    if (Integer.parseInt(strLine)==0) //trend down
    {
        trend=0;
        startVolume=averageVolume;
        endVolume=averageVolume;
    }
    strLine=br.readLine();
    instrument=Integer.parseInt(strLine);
    g2d.scale (1.0, -1.0); //make positive up, to the right
    Font font = new Font("Arial", Font.PLAIN, 75);
    g2d.setFont(font);

    if (trend==0)
    {
        if (averageVolume==0)
            g.drawString("pp", 30, 80);
        if (averageVolume==1)
            g.drawString("p", 30, 80);
        if (averageVolume==2)

```

```

        g.drawString("mp", 30, 80);
    if (averageVolume==3)
        g.drawString("mf", 30, 80);
    if (averageVolume==4)
        g.drawString("f", 30, 80);
    if (averageVolume==5)
        g.drawString("ff", 30, 80);
}
else
{
}
if (instrument==0)
    g.drawString("flute", 50, -200);
if (instrument==1)
    g.drawString("oboe", 50, -400);
if (instrument==2)
    g.drawString("trumpet", 50, -400);
if (instrument==3)
    g.drawString("clarinet", 50, -400);
g2d.scale (1.0, -1.0); //make positive up, to the right

g2d.translate (200.0,0.0); //make 0 where graph is 0,0

for (int i=0; i<numberNotes; i++)
{
    strLine=br.readLine();
    frequencies[i]=Integer.parseInt(strLine);
    strLine=br.readLine();
    relativeLengths[i]=Integer.parseInt(strLine);
    if (relativeLengths[i]==1)
        smallestIndex=i;
}
int xPosition=0;
int halfSteps=0;
int octave=0;
double yPosition=0;

for (int i=0; i<numberNotes; i++)
{
    System.out.println("freq:" +frequencies[i]);

    halfSteps=(int) (Math.log((double) (frequencies[i]/329
628))Math.log((double)1.059463)); //steps from e
    if (halfSteps<13)
        octave=1;
    if (halfSteps>12 && halfSteps<25)
        octave=2;
    if (halfSteps>24 )
        octave=3;
    if ((double) ((int) (halfSteps/2))< halfSteps/2)
        yPosition=25*((int) (halfSteps/2)+octave)\
        12;
    else
        yPosition=25*(halfSteps/2+octave)-12;
}

```

```

System.out.println("ypos: "+yPosition);
System.out.println("half steps: "+halfSteps);
xPosition+=75;
// g.fillRect(xPosition, (int)yPosition, 50, 50);
if (relativeLengths[i]==1)
{
    System.out.println("quarter note");
    g.fillOval(xPosition, (int)yPosition, 30, 25);
    g.fillRect(xPosition+26, (int)yPosition+12,2,100 );
}
if (relativeLengths[i]>1 && relativeLengths[i]<2.2)//twice
smallest
{
    System.out.println("half note");
    g.drawOval(xPosition, (int)yPosition, 30, 25);
}
if (relativeLengths[i]>=2.2 && relativeLengths[i]<3.2)
{
    System.out.println("dotted quarter");
    g.fillOval(xPosition, (int)yPosition, 30, 25);
    g.fillOval(xPosition+28, (int)yPosition, 8, 8);
}
if (relativeLengths[i]>=3.2)
{
    g.drawOval(xPosition, (int)yPosition, 30, 25);
    System.out.println("whole note");
}

}

//Close the input stream
in.close();
}catch (Exception e){//Catch exception if any
    System.err.println("Error: " + e.getMessage());
}

}
}

```