Hearing Trig Functions

Michael Lloyd, Ph.D.
Department of Mathematics and Computer Science

Abstract

In this paper, it will be explained how to use the computer algebra system Maple to create sound files from the graphs of periodic functions.

Introduction

When I have taught trigonometry, I have pointed to a graph of a function and said that it could be a snapshot of a sound wave. In addition to helping students understand trigonometric functions better, the ideas in this paper could be used to communicate concept of a graph of a function to blind students. In this paper, I will describe a tool to create wav or mp3 files that can be played for students. Note that you can click on many of the graphs in the web version of this paper to hear the corresponding sound. At the end of this paper, the inverse problem is considered: Sound files are turned into graphs and their frequencies are analyzed.

Setting Up Maple's Signal Processing Packages

An explanation of how to create and use Maple packages is found at http://www.mapleapps.com/packages/acpackages.html. I found it useful, but also sketchy and confusing, so I will explain the process here. Download the Maple worksheets SigGen Functions.mws and WAV.mws from the Signal Analysis and Synthesis website http://www.mapleapps.com/categories/engineering/electrical_electronic/html/signal.htm. The first worksheet contains signal-generating functions, and the second contains functions for reading and writing wav files. We will create packages, which are similar to Maple libraries, from these worksheets. This only has to be done once.

The maple commands in this paper will be prefixed with the symbol >, and the resulting output will be displayed centered on the page or column in italics. I used Maple version 6.01, and I know that these packages will not work with Maple version 5.

libname is a Maple system variable that contains a list of the directories on your computer that contain Maple libraries. For example, libname originally contained "C:\\Program Files\\Maple 6/lib" on my PC. I decided to put the new sound processing packages in a folder called c:\MyLib that I created. Note that Maple is not case sensitive regarding directory and file names. The following command will add the directory to the list libname.

```
> libname:="c:/MyLib",libname;
```

Execute the block of code (a module) at the top if the SigGen Functions.mws file. To create the package, do

> savelib(SigGen);

The signal generating function package now exists in memory, but it is not permanently stored on your hard drive until you do

> march('create',libname[1],100);

Now, repeat the process for the other Maple worksheet, WAV.mws. That is, execute the module block at the top of the WAV.mws file, and then do

> savelib(WAV); march('create',libname[1],100);

The second command will appear to generate the error:

Error, (in march) there is already an archive in "C:/MyLib"

However, Maple **added** the WAV package to the signal-generating package that you made earlier. The process I just described only needs to be completed once. However, when you come back another day and you want to access the functions in the two packages, you will need to tell Maple the directory you put them in:

> libname:="c:/MyLib",libname;

Then load the packages the same way you load commonly used libraries such as plots and linalg:

> with(SigGen);

[ASD, Attack, Decay, FM_Signal, FM_Sine, Integer2Bytes, LFO, Limiter, Normalize, Quantize, Sawtooth, Signal, Sine, ViewEnvelope, WriteSound, word2byte, writeintegers]

> with(WAV);

[ReadWAV, WriteWAV]

A complete description of these signal-generating and wav functions will be found at the Maple Signal Analysis and Synthesis website referred to earlier.

Simple Sine Wave

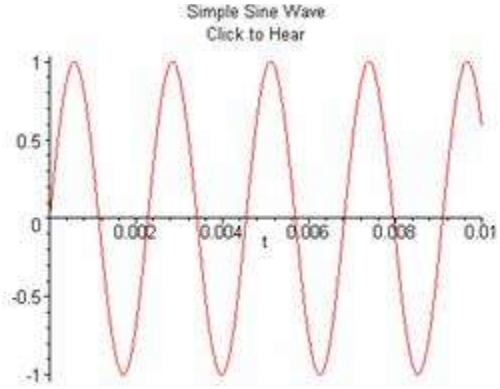
Suppose we want to hear what a simple sine wave $y = \sin t$ sounds like. The frequency obviously must be in the range of human hearing, so I choose 440 Hertz, which is the standard frequency set for the A above middle C.

Of course, the actual formula for a simple sine wave with this frequency is $y = \sin(2\pi \cdot 440t)$, but the shape will be the same. Also, we will not be concerned about the amplitude as long as we can hear the sound. The following Maple variables will remain the same for all the wav files we make:

```
> Duration:=5:                          # Duration in seconds
> SampleFreq:=44100:                    # full CD sample rate, number of samples per secor
> Resolution:=16:                       # 2^16 levels is CD quality, 2^8 is telephone
> deltaT:=1/SampleFreq:                 # time between adjacent sample data
> Npoints:=round(Duration*SampleFreq);  # number of points in sample
```

The number of points required to generate 5 seconds worth of sound at CD quality is $44100 \times 5 = 220{,}500$. Consequently, each Maple command often takes several minutes to execute, even on my 650 MHz computer. The signal information is inputted as lists. You can add more sine waves to the signal if you like by extending these lists.

```
> amps:=[1]:          # list of amplitudes
> freqs:=[440]:       # list of frequencies
```



Here is a plot of the first 0.01 seconds of the sound:

> plot('Signal(amps,freqs,t)',t=0..0.01, title="Simple Sine Wave\nClick to Hear");

The vertical axis is understood to be pressure, and the horizontal axis is time in seconds.

I decided to use the function Signal from the SigGen package to keep in the spirit of signal processing instead of writing out the function in mathematical form. The next command will generate the data that will be converted to a wav:

< p>>SigData:=Array([seq(Signal(amps,freqs,i*deltaT),i=1..Npoints)],datatype=float):

The Array data type in Maple is similar to a list, except that by specifying the type of its elements to be float, Maple supposedly will process it faster. Here is the command for writing a wav file for this sound:

> WriteWAV("c:/~temp/exams/sine440.wav", SigData, SampleFreq, Resolution);

1 channel(s) found.
220500 points per channel found.

"Quantizing to 16 bits..."

"Saving to c:/~temp/exams/sine440.wav"
"Done"

You will get an IO error if a file with that name already exists. I stored these files in the directory C:/~temp/exams/, but you could store them in any directory that you like. The size of the wav file "sin.wav" was 430 kilobytes. I compressed it to an mp3 using the freeware, dBPowerAmp Music Convertor. The corresponding mp3 was 78.7 kilobytes which makes it more suitable for putting on a web page. In a previous talk, Math in Music, I used midi files exclusively. Midi files are much smaller than mp3s and wavs, and they are much faster to generate, but I was restricted to using a limited palette of sounds, and in fact, they may sound differently depending on the computer they are played on. By using wavs, you hear exactly what the sound looks like.

Square, Saw Tooth, and Triangular Waves

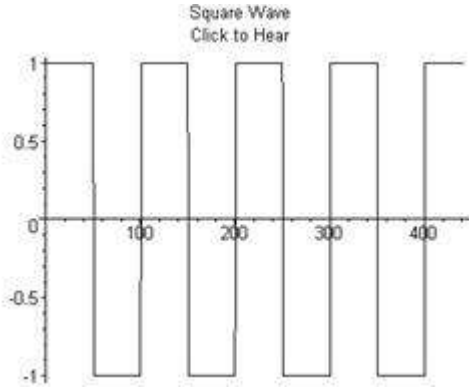
The simple waves which follow, are all the same note that was created above, only the timbre will be different. To save time and memory, I reused the same SigData variable name to create the data for a square wave:

```
> SigData:=
Array([seq(signum(SigData[i]),i=1..Npoints)],datatype=flo
at):
```

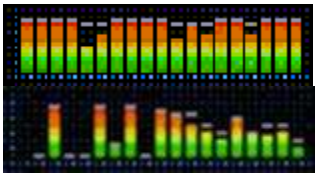
Here is the graph:

```
> with(plots):

> listplot([seq(SigData[i],i=1..441)],
title="Square Wave\nClick to Hear");
```



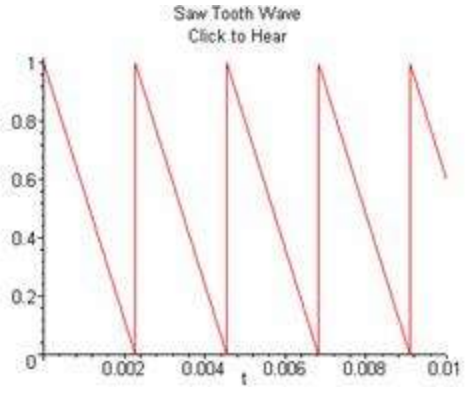
Note that the square corners require many frequencies, so that the frequency distribution appears fairly flat (Click to hear):



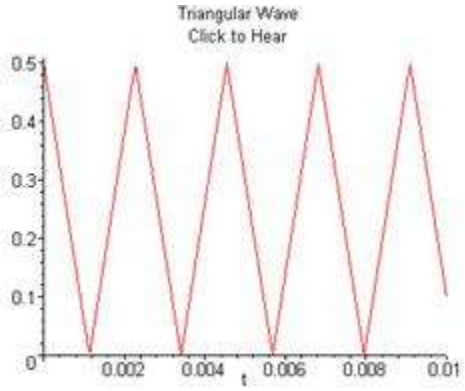
I could hear the mp3 was not quite as good as the wav, and this is illustrated in its frequency distribution (Click to hear):

Here are the commands for generating plots for the saw-tooth and triangular waves:

> plot(Sawtooth(440,0,t),t=0..0.01, title="Saw Tooth Wave\nClick to Hear");



> plot(abs(Sawtooth(440,0,t)-.5),t=0..0.01, title="Triangular Wave\nClick to Hear");



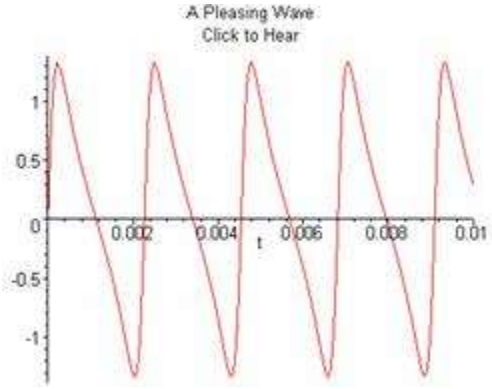
An Almost Perfect Wave

Most people find the simple waves shown above to be annoying. In the paper "Designing a Pleasing Sound Mathematically", Erich Neuwirth said that the function

$$y = \sum_{n=1}^{\infty} \frac{1}{2^{n-1}} \sin(2\pi 440t) = \frac{\sin(2\pi t)}{5/4 - \cos(2\pi t)}$$ is simple, yet sounds pleasing.

> PleasingSound:=t->sin(2*Pi*440*t)/(1.25-cos(2*Pi*440*t));

> plot(PleasingSound(t),t=0..0.01, title="A Pleasing Wave\nClick to Hear");

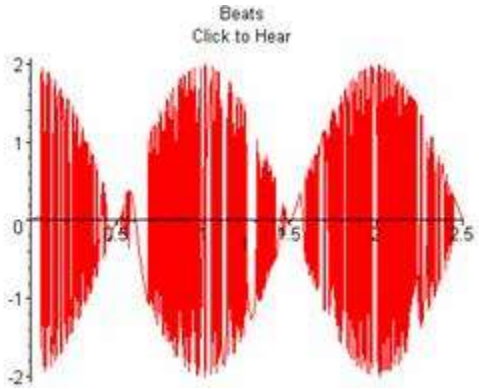


Beats

The musical phenomenon of beats occurs when two sounds with approximately the same frequencies combine. The volume of the sound can be heard to oscillate, and the frequency of this oscillation is called the beat frequency. When I teach trigonometry, I work an example involving beats where a sum-to-product identity is applied. The beat frequency will be the difference of the two component frequencies, and the perceived pitch of the combined notes will be the average of the two pitches. String players often tune their instruments using the concept of

beats: They simply adjust the string tension until the beats disappear.

I decided to use the frequencies 440 Hz and 441 Hz:



> amps:=[1,1]: freqs:=[440,441]:

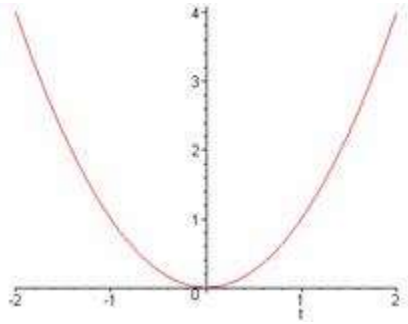
The beat frequency is $441 - 440 = 1\,\text{Hz}$ and the perceived frequency is $(440 + 441)/2 = 440.5\,\text{Hz}$. Here is a plot of the first 2.5 seconds of the sound.

> plot('Signal(amps,freqs,t)',t=0..2.5, title="Beats\nClick to Hear");

Note that there are about 440 oscillations squished into each lobe, so a complete graph would appear solid.

Envelopes

The boundary around the graph of the beats above is called an envelope. This suggests a method for "hearing" non-trigonometric curves. For example, suppose you wanted to "hear" the parabola $y = t^2$:

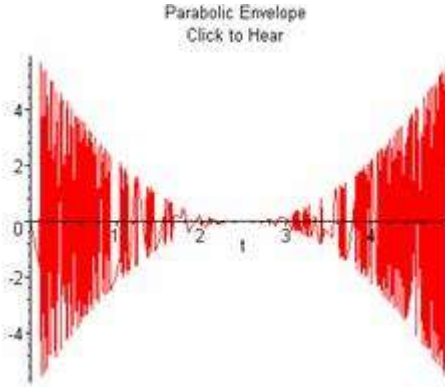


The essence of the shape stays the same if we translate the parabola horizontally, so I decided to put the vertex in the middle of a 5 second 440 Hz sine wave.

> amps:=[1]: freqs:=[440]:

> SigData:= Array([seq((i*deltaT-2.5)^2*Signal(amps,freqs,i*deltaT),i=1..Npoints)],datatype=float):

> plot('(t-2.5)^2*Signal(amps,freqs,t)',t=0..5, title="Parabolic Envelope\nClick to Hear);



Unfortunately, you cannot tell from this sound whether the function is above or below the x-axis. The Maple signal-processing commands do allow the creation of stereo wave files, so one

solution to this dilemma would be to put the sound through the left channel if the function were negative, and put it through the right, if the function were positive. For example, suppose we wanted to hear the graph of the cubic $y = t^3$. The commands for creating the stereo sound appear below. Creating the array took 1 hour and writing the 5-second stereo wav file took 8.5 hours on my 450 MHz PC! Click here or on the far left graph of the cubic to hear the stereo sound.

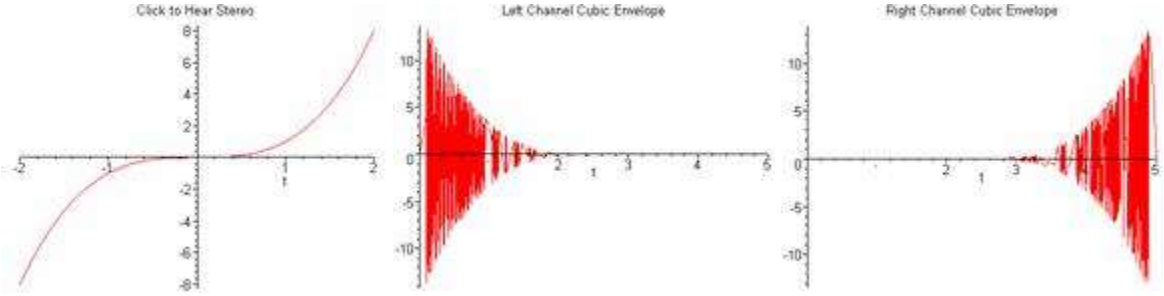
```
> stereo:=t->[piecewise(t<2.5,(t-2.5)^3*Signal(amps,freqs,t),0),piecewise(t>2.5,(t-
2.5)^3*Signal(amps,freqs,t),0)]:

>SigData:=Array([seq(stereo(i*deltaT),i=1..Npoints))],datatype=float):

> WriteWAV("c:/~temp/exams/cubic.wav", SigData2, SampleFreq, Resolution);2 channel(s) found.

                                220500 points per channel found.

                                "Interweaving sound data..."
```

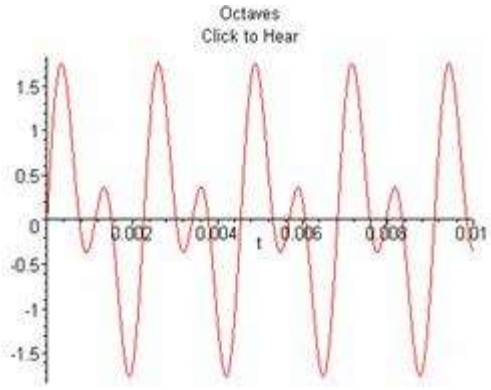


Combining Trigonometric Functions

For the first example below, I made the amplitudes the same, and used a Maple function called chord. This function generates a list of frequencies if you tell it the names of the notes and octaves to which these notes belong. A complete description of this function and its listing are given in the appendix. The two notes in this example are an octave apart, that is, the higher frequency is twice the other. The shape of the graph is the same as the graph of the function $y = \sin t + \sin 2t$.

> amps:=[1,1]: freqs:=chord(["A","A"],[4,5]);

freqs := [440.00, 880.00]

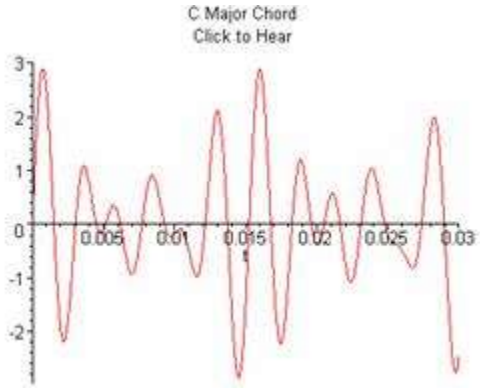


A major chord is generated for this second example. This graph is close to the graph of $y = \sin 4t + \sin 5t + \sin 6t$.

> amps:=[1,1,1]:

> freqs:=chord(["C","E","G"],[4,4,4]);

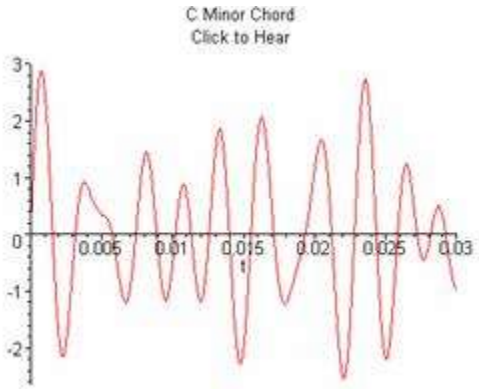
freqs := [261.6255653, 329.6275568, 391.9954358]



For the third example, the middle note is dropped half of a step to form a C minor chord. This graph is close to the graph of $y = \sin 10t + \sin 12t + \sin 15t$.

> freqs:=chord(["C","Eb","G"],[4,4,4]);

freqs := [261.6255653, 311.1269838, 391.9954358]



It is not clear by looking at the graphs of these pressure-versus-time plots why a major chord is more consonant than a minor chord. I believe that the right way to visualize a complex sound is to look at a frequency distribution. According to the resonance theory of hearing, our ears acts like a Fourier transform by separating the sound into its component frequencies before sending the signals to our brain. The major chord is more consonant because the frequencies that make up that sound are roughly proportional to smaller integers.

Analyzing CBL Data

The following Maple commands create a buffer for reading in pressure data. This data was obtained by playing a guitar string into a microphone attached to a Calculator Based Laboratory (CBL) like the one pictured here.

```
> fd := fopen(`C:/~temp/exams/FFTpress.txt`,READ,TEXT):

> InData:=readdata(fd, float):

> fclose(fd):

> InDataSize:=nops(InData);
```

InDataSize := 32

We will do a Fast Fourier Transform (FFT) on this data to compute the component frequencies. The original data contained 99 time-pressure pairs of data. (99 is the largest dimension for a list on the TI-82 graphing calculator.) Since the FFT requires that the data size be a power of 2, I applied linear interpolation to the original data to get exactly 32 points for one cycle of the sound. For thousands of points, it is unnecessary to do this interpolation or get an integral number of cycles, just take the largest power of 2 points. Here is a graph of the cycle that I chose:

```
> listplot([seq(InData[k][1],k=1..InDataSize)], title="D String");
```

The following Maple commands are used to set up the FFT:

```
> m:=5: SigSize:=2^m;
```

SigSize := 32

```
> RealPart:=hfarray([seq(InData[i][1],

i=1..SigSize)]):

>ImagPart:=hfarray([seq(0,i=1..SigSize)]):
```

Now to perform the FFT:

```
> evalhf(FFT(m,RealPart,ImagPart)):
```

The hardware float arrays RealPart and ImagPart now contain the complex frequencies. We will use the following Maple function to find the real magnitude for each frequency:

> mag:=proc(RealPart, ImagPart): sqrt(RealPart^2+ImagPart^2): end:

The sample period 000210266 in units of seconds was manually obtained from the time text file.

> SampleFreq:=1/.000210266;

SampleFreq := 4755.880646

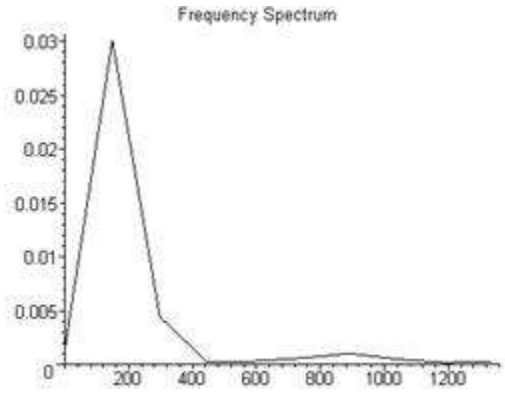
The variable df (delta frequency) is a scaling factor that relates the indices of the RealPart and ImagPart arrays to the actual frequency.

> df:=evalf(SampleFreq/SigSize):

> FreqSpectrum:= [seq([(i-1)*df,(2*mag(RealPart[i],ImagPart[i])/SigSize)],i=1..floor((SigSize/2)))]:

Here is a plot of the frequency spectrum:

> listplot([seq(FreqSpectrum[i],i=1..10)], title="Frequency Spectrum");



The two peaks are

> FreqSpectrum[2];FreqSpectrum[7];

[148.6212702, .03009241976]

[891.7276212, .001030763241]

A useful feature in Maple is that you can left click on a graph and read the coordinates. Doing that and then looking at the list, we see that the peaks are at approximately 149 Hz and 892 Hz. If the data set had been much larger, then these peaks would be very narrow. 149 Hz is close to the note D (146.8 Hz), and 892 Hz is close to the note A (880 Hz), the harmonic which is about 6 times higher. I generated the sound by plucking the D string in the middle, which explains the lack of the harmonics 2 through 5. If I had used a pick near the bridge and the data set had been much larger, these other harmonics should have been visible.

Analyzing WAV Data

In this section, we will analyze a wav file that I created by connecting a tape player to my sound card. I trimmed the file using the software GoldWave. This wav is the sung word, “wonderful” (click to hear) in a song called “Cornerstone”. At this point in the song, the pianist at our church and I disagreed on the chord accompaniment. She believed the chord to be minor while I

believed the chord should be major. We will do a spectral analysis on this wav to determine who was correct.

The first step is to make sure that the variable SampleFreq is free.

> SampleFreq:='SampleFreq':

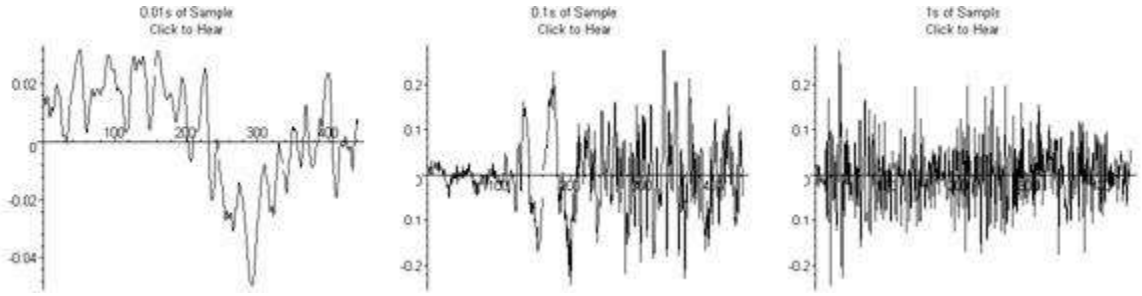
The function ReadWAV will pass the sample frequency to its second argument.

> SigData:=ReadWAV("c:/~temp/exams/wavs/wonderful.wav",SampleFreq):

> SampleFreq;

44100

Here are three graphs of the sampled sound to demonstrate its appearance at various time scales:



The Maple command ArrayNumElems(SigData) gives 80970 for the sample size, so we will use the first $2^{16} = 65536$ points for the FFT. A graph of the frequency spectrum appears here.

The frequencies for the two highest peaks in Hertz are 125 Hz and 254 Hz, clearly an octave apart. These notes are close to a B, which I will take to be the root of the appropriate chord for the sampled portion of the song.

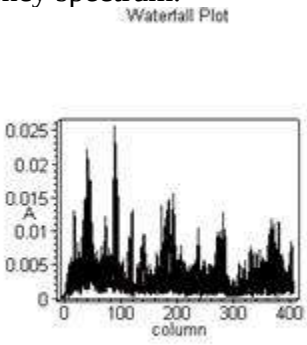
I decided not to include the Maple commands which were used to generate the remaining plots for this paper because they were a bit tedious. Many of the commands were tedious and some are at the website. Each graph shows the original frequency spectrum on the bottom. The downward pointing spikes for the first graph locate the powers of 2 times the root frequency 254 Hz. The next two graphs compare the original frequency spectrum with the powers of 2 multiples of the major and minor 3rd, respectively. (A major 3rd above 254 Hz is computed ; a minor 3rd above 254 Hz is computed .)

The major 3^{rds} (the middle graph) matches better than the minor 3^{rds} (the last graph), so a B major chord is a better fit than B minor.

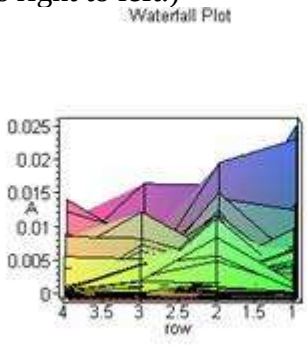
I made an animation of the frequency spectrum evolving over time, but the sample size was only large enough to make 4 time frames. Each frame pictured here is 0.37 seconds apart.

The waterfall plot is another way to visualize the frequencies evolving over time. The row axis is time (slice 1, 2, 3, and 4); the column axis is frequency (Hz).

If the waterfall plot is turned on end, then the graph is almost the same as the original frequency spectrum.



Looking at the other end, you can see how the wav is getting quieter over time. (Increasing time is right to left.)



Appendix: Maple Function for Generating Chord Frequencies

Author: Michael Lloyd Date: August 2001

Description

This function takes a sequence of notes and corresponding octaves and returns a frequency list to be used by the functions in the SigGen Maple package. The 0th octave corresponds to the lowest notes on a piano, and the 4th octave is middle C up to and including the B on the treble clef. (This is the octave identification system recommended by the International Acoustical Society and also used in Braille music notation.) For example, a command for generating the frequencies for a C chord is chord(["C","E","G"],[4,4,4]); This will return the list [261.6255653, 329.6275568, 391.9954358] where each frequency is in Hertz.

This function can be improved by adding enharmonic notes that require double sharps or double flats such as “FX” or “Ebb”. This would accommodate musically correct spellings of the chords in unusual keys or chords such as Ab diminished which should be spelled as Ab Cb Ebb.

Program Listing

```
chord := proc(notes, octaves)
```

```
local freqs, k, scale;
  freqs := NULL;
  for k to nops(notes)
  do
     if notes[k] = "C"  or notes[k] = "B#" then scale := 0 end if;
     if notes[k] = "C#" or notes[k] = "Db" then scale := 1 end if;
     if notes[k] = "D" then scale := 2 end if;
     if notes[k] = "D#" or notes[k] = "Eb" then scale := 3 end if;
     if notes[k] = "E"  or notes[k] = "Fb" then scale := 4 end if;
     if notes[k] = "F"  or notes[k] = "E#" then scale := 5 end if;
     if notes[k] = "F#" or notes[k] = "Gb" then scale := 6 end if;
     if notes[k] = "G" then scale := 7 end if;
     if notes[k] = "G#" or notes[k] = "Ab" then scale := 8 end if;
     if notes[k] = "A" then scale := 9 end if;
     if notes[k] = "A#" or notes[k] = "Bb" then scale := 10 end if;
     if notes[k] = "B"  or notes[k] = "Cb" then scale := 11 end if;
     freqs := freqs, 13.75*2^(octaves[k] + 1/12*scale + .25)
  end do;
  freqs := [freqs]
end proc
```

References

Creating Maple 6 Library Packages, http://www.mapleapps.com/packages/acpackages.html.

“Designing a Pleasing Sound Mathematically” by Erich Neuwirth, Mathematics Magazine, Vol. 74, No. 2, April 2001, pp. 91-98.

"Music in Theory and Practice" by Bruce Benward and Gary White ©1989 WmC Brown Publishing Co.

Signal Analysis and Synthesis, http://www.mapleapps.com/categories/engineering/electrical_electronic/html/signal.htm.

Software Used in This Paper

dBPowerAmp (music convertor), http://admin.dbpoweramp.com/

GoldWave (digital audio editor), http://www.goldwave.com/

Maple version 6.01 (computer algebra system), http://www.maplesoft.com

Microsoft Photo Editor

Noteworthy Composer (music composer), http://www.NoteworthyComposer.com

NullSoft WinAmp (mp3 player), http://www.winamp.com/

Biographical Sketch

Michael Lloyd received his B.S in Chemical Engineering in 1984 and his Ph.D. in Mathematics from Kansas State University in Manhattan. He has presented papers at meetings of the Mathematical Association of America, the Arkansas Conference on Teaching, and the American Mathematical Society. He has been at Henderson State University since August 1993.