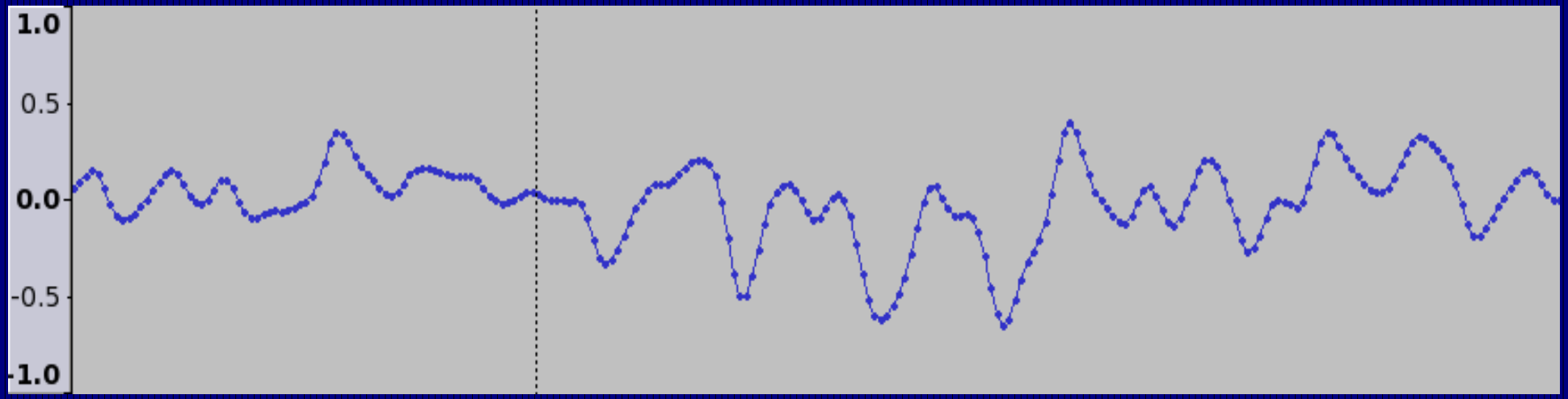
Automatic Classification of Digital Audio Signals Based on Chord Movement

Progress Report by Dan Church, Project Leader
Project Name: Audicon

Review: Digital Audio Storage

Digital audio data is composed of **samples**.
Samples are floating-point values from -1 to +1.
Standard audio CD samples are composed of 16 bits and are played at 14,400 Hz (samples per second).



Time →

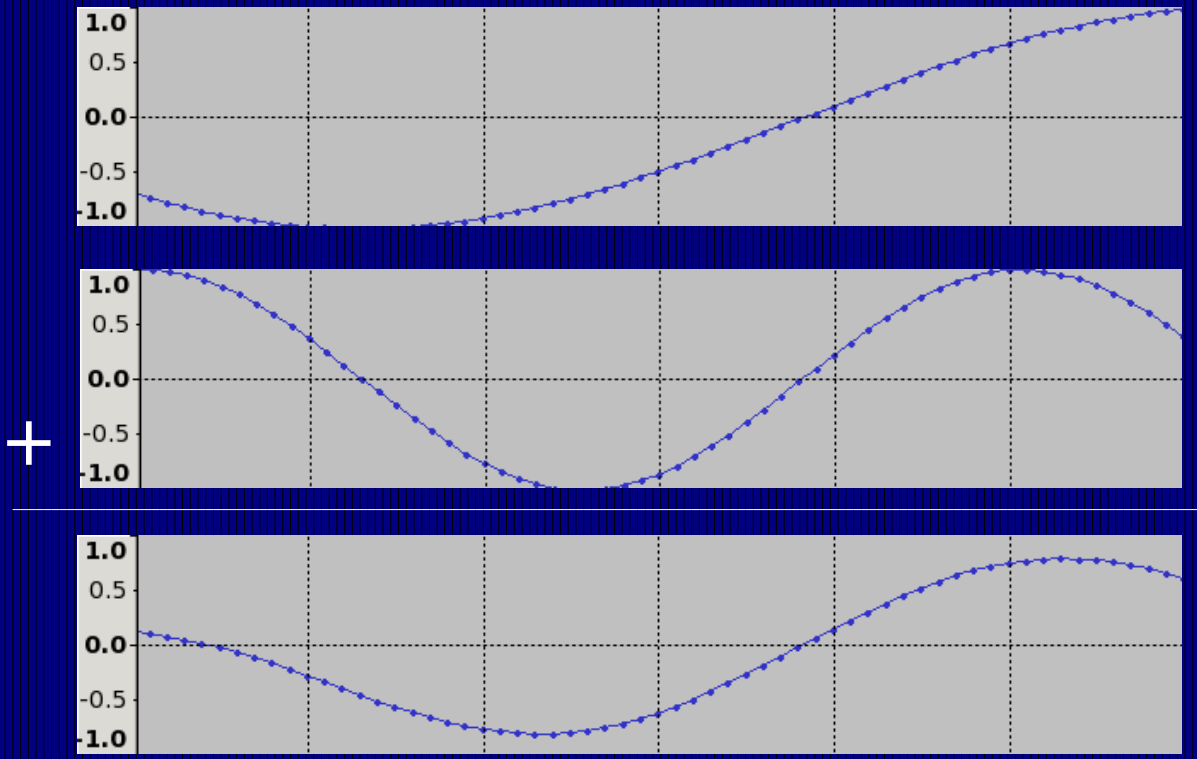
Plot of samples in Ogg-compressed song “Rock the 40 Oz.” by Leftöver Crack (2004) positioned from 0:01.948420 to 0:01.953955

Review: The Fast Fourier Transform (FFT)

Named for Joseph Fourier, a French physicist who discovered in 1822 that complex waveforms are formed by composing (averaging the amplitudes of) smaller, less complex waves.

He also discovered that the inverse of a wave, when added to another wave would effectively “subtract” that wave from the other.

Thus, it is possible to decompose complex waves into simple, single-frequency waves.



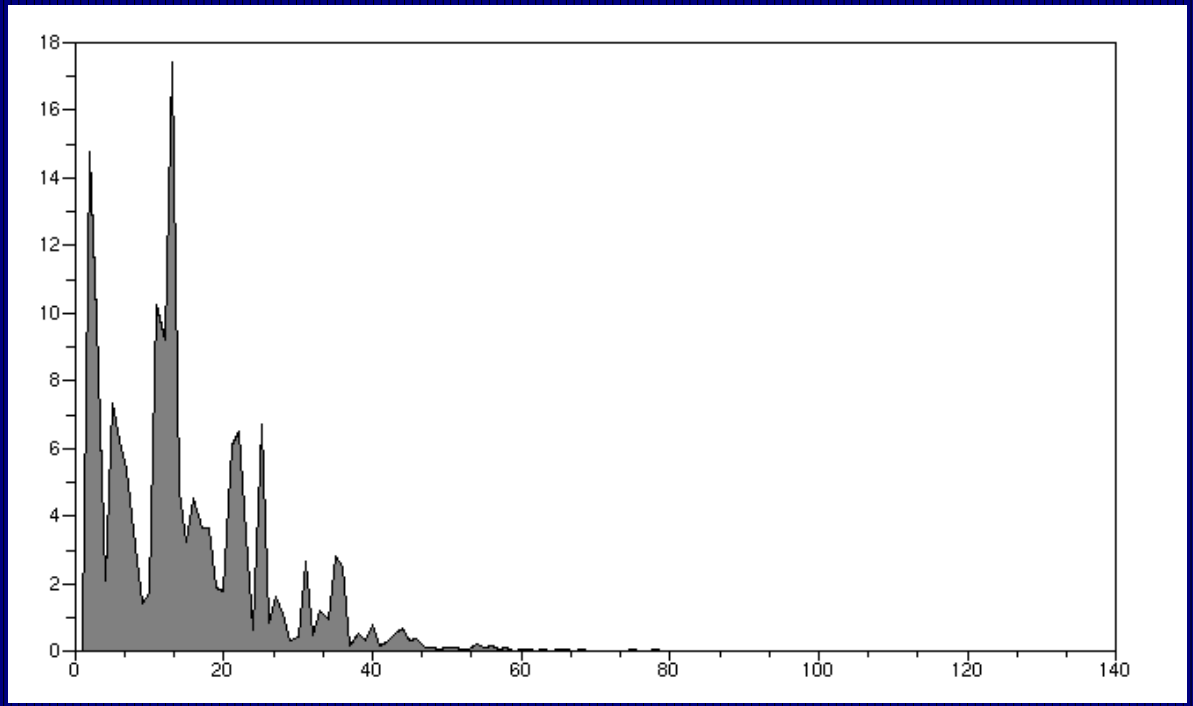
Review: The Fast Fourier Transform (FFT)

$$\text{fft}(k) = F[f(x)](k) = \int_{-\infty}^{\infty} f(x) \cdot e^{-2\pi i k x} dx$$

Analyzes a signal by determining what frequencies compose a signal and in what strength (amplitude).

Can be used to draw more meaningful information out of audio and signal streams.

Used by SETI@Home to find patterns in radio telemetry data.



An FFT plot of the sample data in the previous slide

Inherent Problems with the Fast Fourier Transform

Let **A** be a vector of values of size 1x***s*** such that:

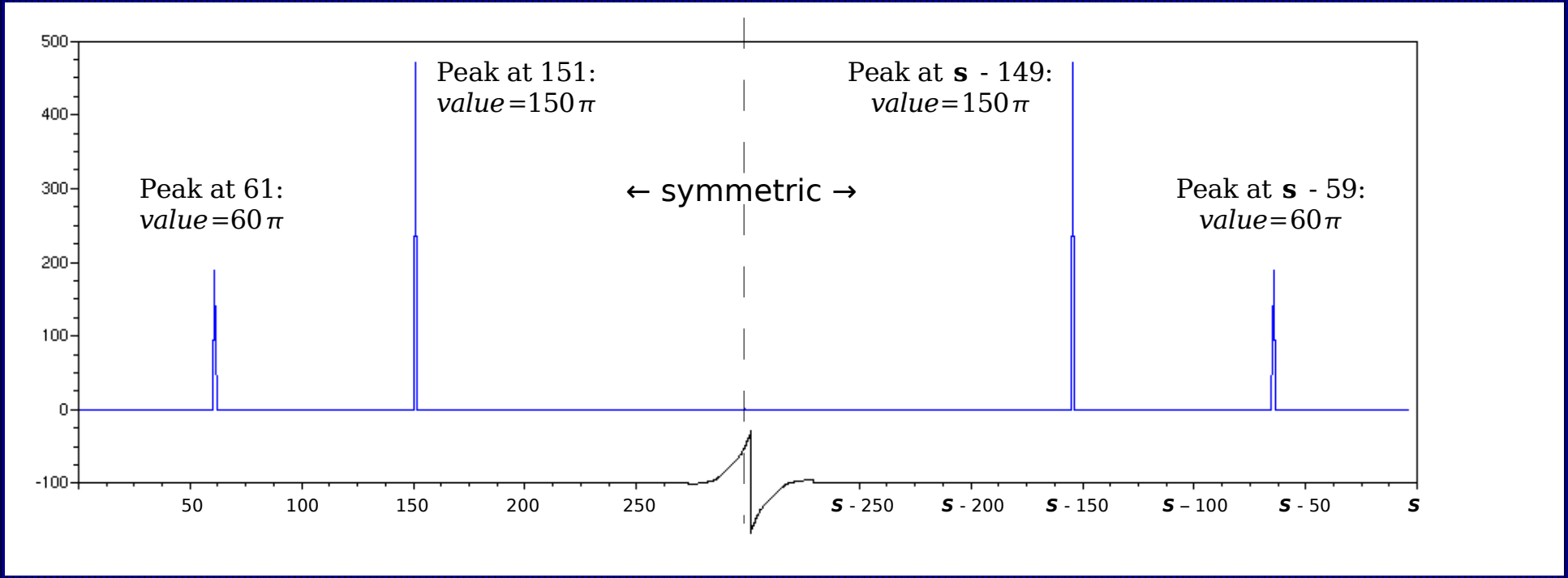
$$A[n] = \boldsymbol{c} * \sin\left(\frac{n}{s} * 2\pi * \mathbf{60}\right) + \boldsymbol{c} * \sin\left(\frac{n}{s} * 2\pi * \mathbf{150}\right)$$

That is, if one thinks of **A** as a vector of wave sample values played at ***s*** samples per second, then **A** can be thought of as a combination of a **60**-Hz wave and a **150**-Hz wave, each at amplitude ***c***.

The FFT-decomposition of **A** would then yield the following:

Inherent Problems with the Fast Fourier Transform

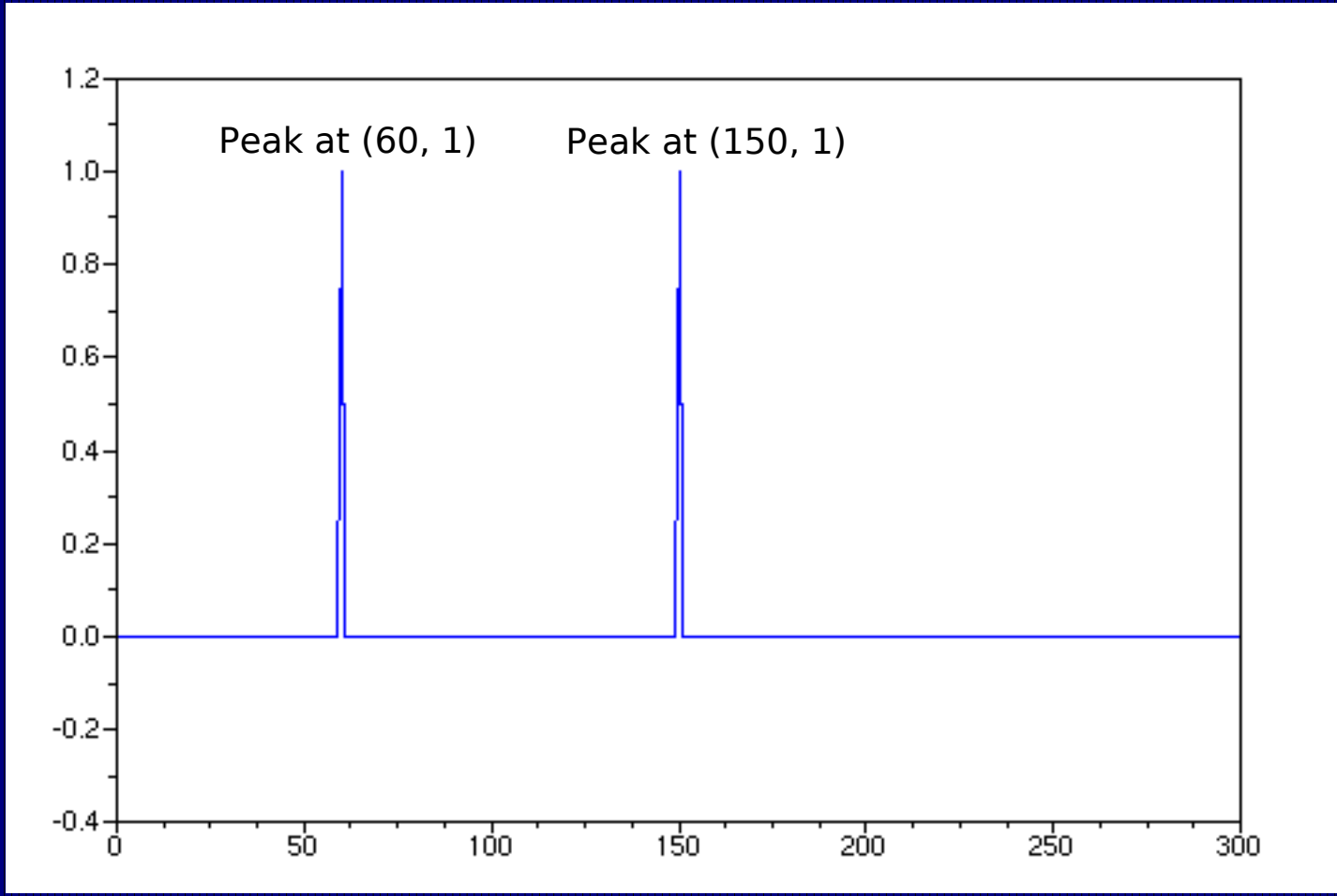
Plotted values of ***abs(real(fft(A)))***



Even though the two component waves were of the same amplitude, they were given different amplitude values when passed through **fft()**. I wrote a function that would give an accurate representation of the amplitudes of component waves.

Inherent Problems with the Fast Fourier Transform

Plot of (freqs, amps) where ***[amps freqs] = fft_unweighted(A)***



This plot shows the output of my unweighted FFT function.

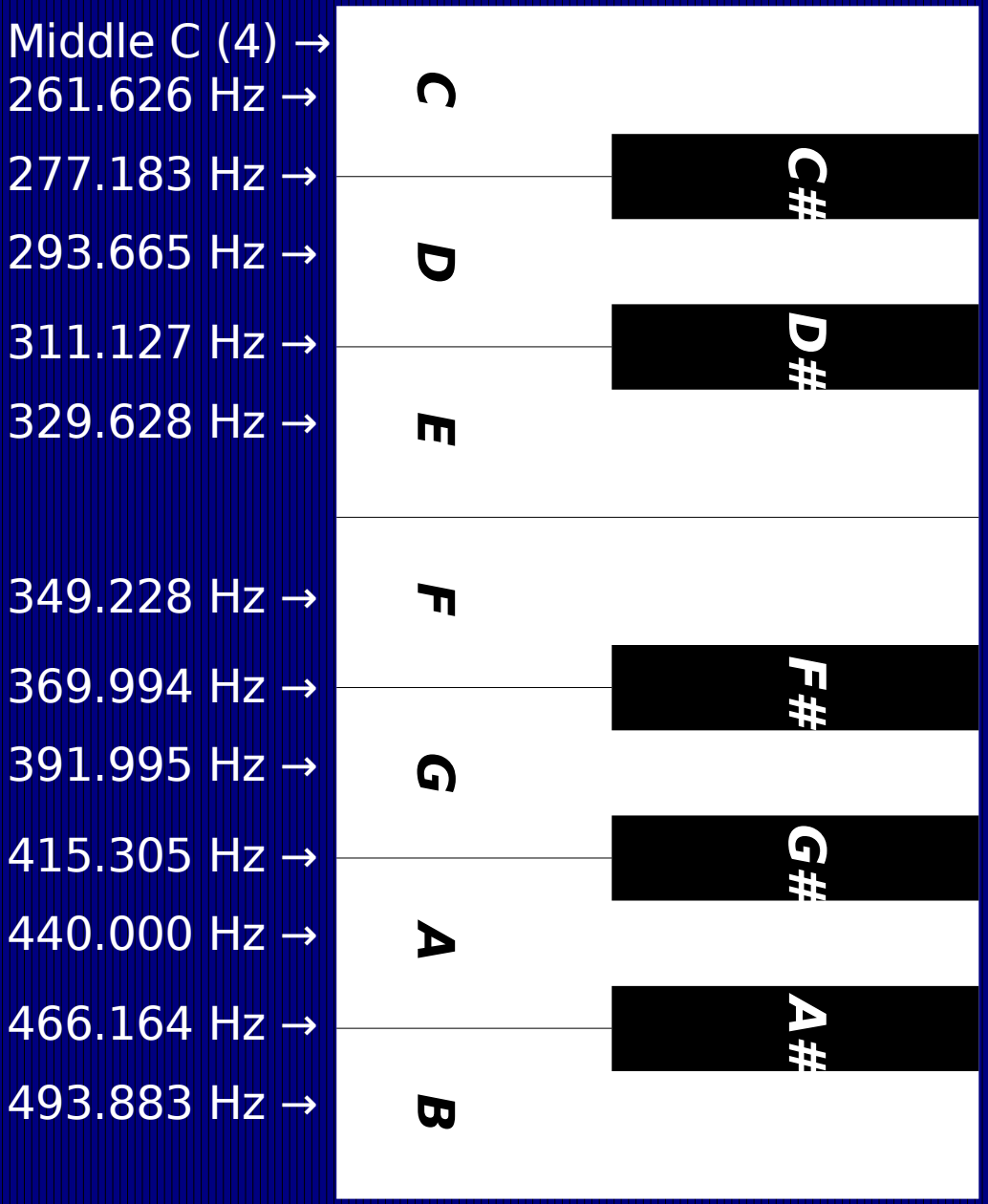
Note Extraction

Piano Scales

The Piano is the standard instrument for tunings.

Since the piano scale is a logarithmic (base 2) scale, frequencies for other octaves can be found by found by multiplying or dividing by two for each octave away.

For instance, the note after B on the diagram on the right would be 2*261.626 = 523.252



Source: http://en.wikipedia.com/Piano_key_frequencies (30 Oct 2006)

Note Extraction

Function ***amps_by_key***

•Relies on function ***fft_unweighted*** to deliver amplitude and frequency values for the given set of sound data.

•Takes into account **frequencies** defined as corresponding to **musical notes** in an arbitrary octave

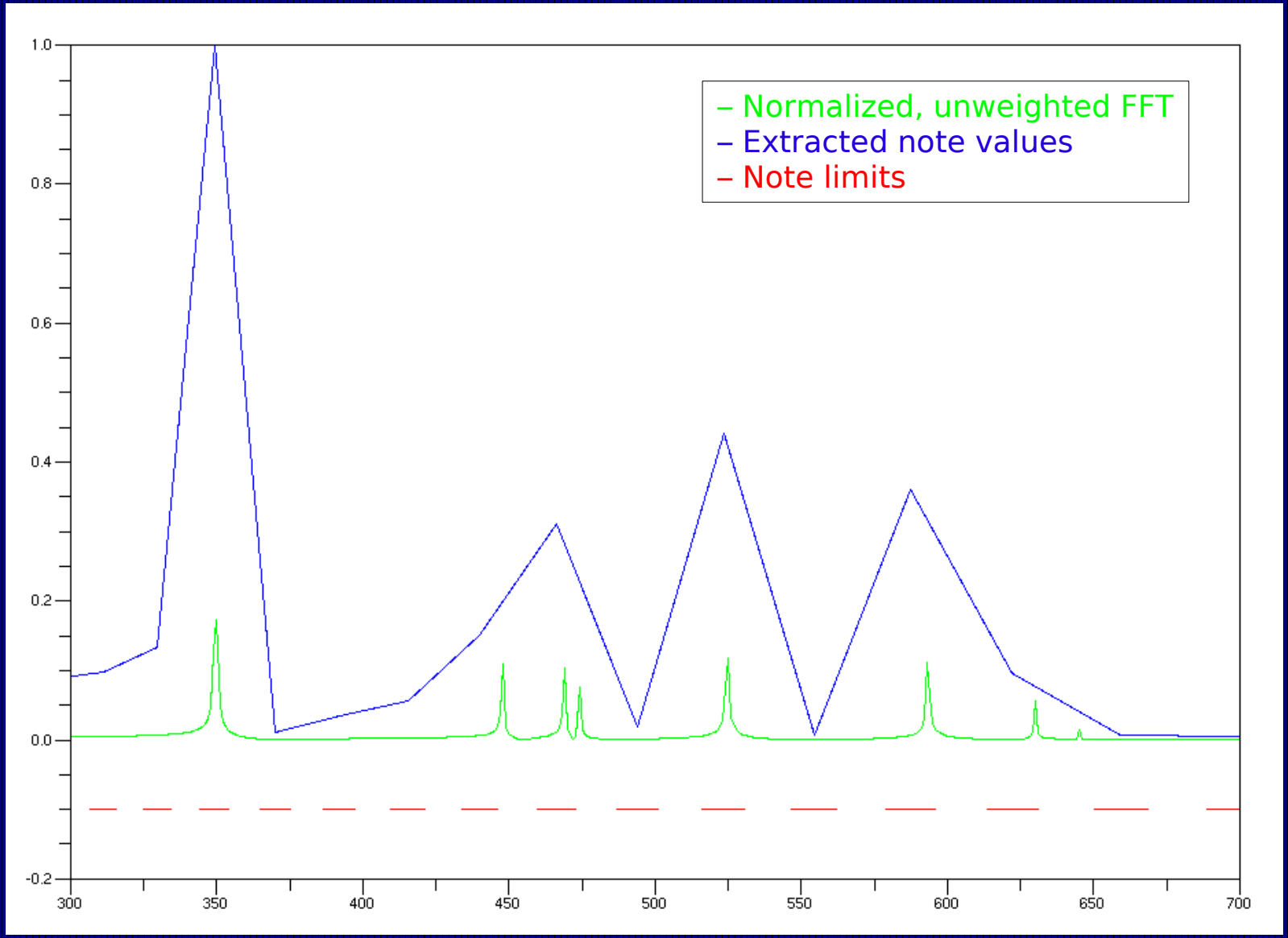
- Default frequencies for piano-based middle-C octave are pre-defined.

•Takes into account an **octave span** in which to detect notes

- Frequency values for an octave one higher than the current has frequencies multiplied by 2
- Inversely, one octave lower is ½ the frequency of the current

•Makes use of a **note span**, which is taken to be where to set the upper and lower frequency limits, as a fraction of the distance between the note and the halfway-point between adjacent notes, affecting the “width” of the “bins” to which the amplitudes are tallied.

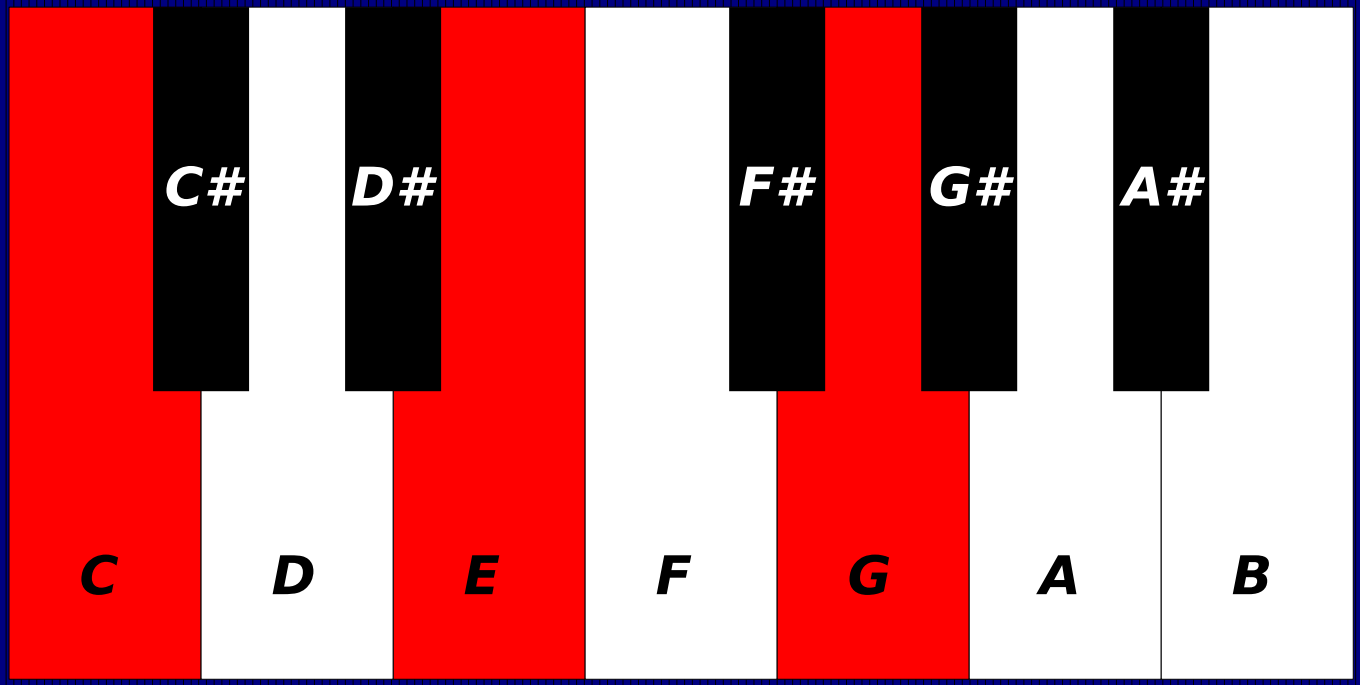
Note Extraction



Plotted test of note extraction from a 1-second wave plot with 300 sine waves of random frequency 1-1000

Review: Chord Detection

Chords are when three or four harmonious notes are played simultaneously



Example of a chord:
C Major

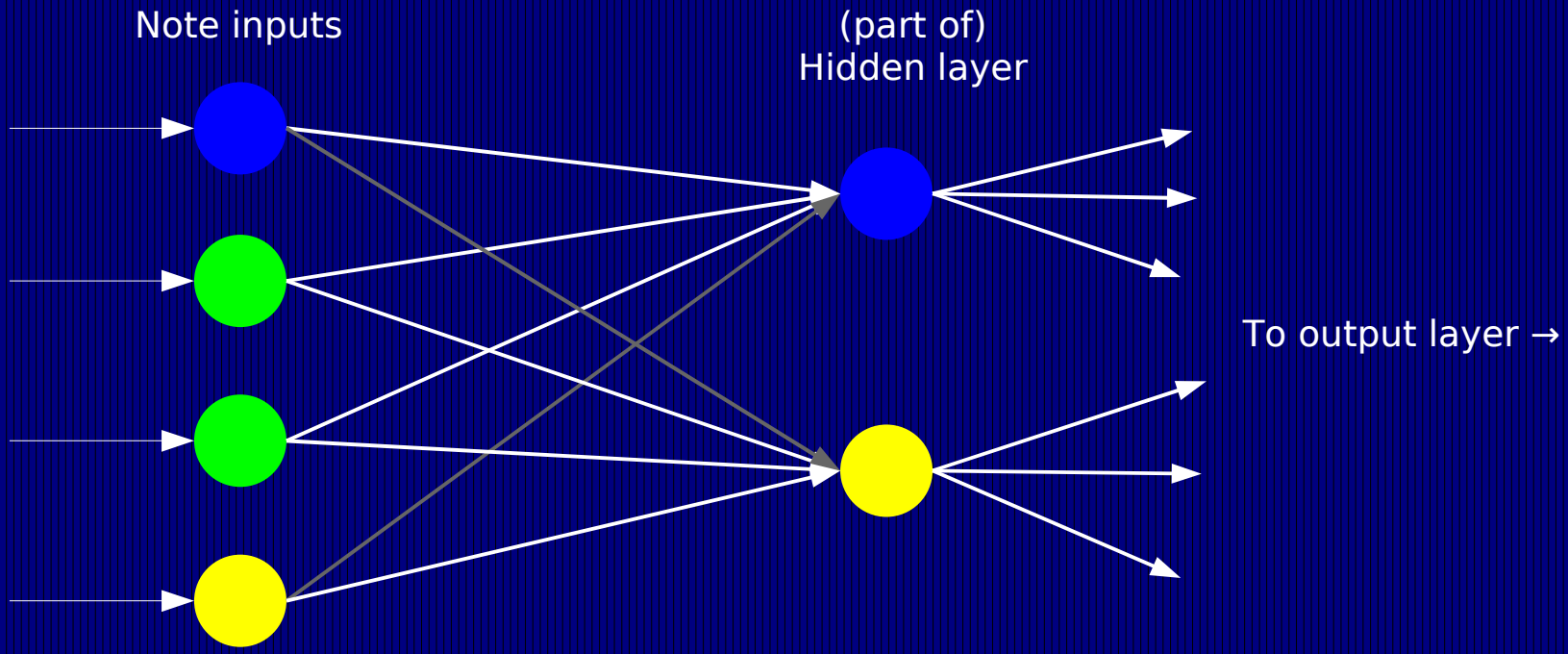
Chord Extraction

Bypassing the need for chord detection

If one were to use the amplitude value of each of the notes detected as part of the input pattern to the network, it would then be plausible to assume that the network would have enough information to be able to (in theory) train itself to recognize any chords in its hidden layer. For example:

Chord Extraction

Assume that this network has 4 inputs, each the amplitude of a note. Notes are color-coded: blue and green notes are part of the same chord, as are yellow and green notes.
I'm not saying that the network will definitely arrange its weights to detect chords in its hidden layer. It may take more than 1 hidden node to report a detected chord to the output layer, or it may not detect chords at all. It all depends on how effective the practice is at minimizing errors.



Statistical Audio Features

Almost every paper I read used four of the same audio **features** for generating useful audio characteristics:

1 Centroid Frequency
- 100% implemented, 80% tested
- A measure of where in the spectrum the signal is brightest
- Computed from weighted FFT (frequency * amplitude)

$$c = \frac{\sum_{1}^{N} f\,M[f]}{\sum_{1}^{N} M[f]}$$

2 Spectral Rolloff
- 50% implemented, 0% tested
- A measure of where the spectrum contains 85% of its total brightness
- Also computed from weighted FFT

$$\sum_{n=1}^{r} M[n] = 0.85 * \sum_{n=1}^{N} M[n]$$

3 Spectral Flux
- 100% implemented, 50% tested
- A measure of the total spectral change from previous audio frame
- Computed from unweighted FFT

$$f = \sum_{n=1}^{N} \left(N_t[n] - N_{t-1}[n]\right)^2$$

4 Zerocrossings Per Second
- 100% implemented, 100% tested
- A measure of the amount of noise in the signal
- Computed from raw audio sample data.

$$z = \frac{1}{2} \sum_{n=1}^{N} \left| sign(x[n]) - sign(x[n-1]) \right|$$

Network Design

Input vector will be made up of both detected note amplitudes and the four statistical audio features.

Default octave span is 3 octaves below middle c to 4 octaves above it & default note span is 0.5 (half the space between notes is covered).

There are 12 default notes defined as belonging to the middle octave, so, coincidentally enough the number of input vectors is

$12 * (3 + 1 + 4) + 4 = 100$

Output vector size varies directly to how many genres are defined. Each value in the target matrix (0 or 1) corresponds to whether its corresponding input song fit that genre.

Current State of Project

Aspect	*Implementation*	*Testing*
Note Detection	100%	70%
Feature Extraction	75%	60%
Network Set-up	60%	0%
Network Training	10%	0%
Input Filters	10%	0%
Memory Optimization	70%	30%
Totals	***54%***	***27%***

Ideas for expansion:

- Usage of multicomputing API to train network
- Server-client architecture-based training data collection

Code Statistics (for Your Amusement)

File name	Code lines	Documentation lines	Comment lines	Total
adjust_stack.sci	16	15	7	37
amps_by_key.sci	73	55	86	214
audicon_defaults.sci	37	15	38	86
centroid.sci	14	36	2	52
fft_unweighted.sci	17	42	13	72
foreach_split.sci	37	70	42	144
load_genres.sci	15	12	5	31
load_training_data.sci	13	16	5	34
normalize.sci	14	12	6	32
process_genre.sci	17	15	7	39
process_wav.sci	14	54	3	70
save_genres.sci	17	16	14	47
save_training_data.sci	19	17	14	50
spectral_flux.sci	19	37	4	60
train_net.sci	23	31	7	61
zerocrossings.sci	21	25	9	54
Totals	***366***	***468***	***262***	***1083***

Arnold's Laws of Documentation:

(1) If it should exist, it doesn't.

(2) If it does exist, it's out of date.

(3) Only documentation for useless programs transcends the first two laws.