CHAPTER 3. DSP WORK

We investigated several DSP techniques for this project. The most practical we found is the Short Time Fourier Transform (STFT). STFT performs a sequence of windowed FFTs on a data set, with the next FFT starting a fixed time delta after the current FFT. Generally we used a Hanning window (a variation of a Gaussian bell curve, see figure 3.3), slightly shorter than the length of the FFT. A few experiments using other windowing strategies suggest that this is an area that could be substantially optimized, but these refinements are not within the scope of this thesis.

The time duration of most audio samples we look at is less than twenty seconds, which means there are several hundred thousand individual data points representing the audio signal. For simplicity we look at mono signals, because the rhythmic patterns we study can be considered as a single stream of note events. More subtle analysis of music should process all available sources of information: both stereo channels, comparison of phase information for correlated harmonics representing a single instrument, or changes of power level and frequencies in a sound that indicates features like tremolo or vibrato.

Wavelet processing is an interesting modern technique (1990's) which blends time and frequency processing into a single framework, decomposing the signal into a metric space that uses the set of wavelets as basis functions. We have done lengthy work with wavelets, motivated by the appealing concept of performing time and frequency processing together, plus the potentially high performance that wavelets deliver in some cases. Our investigation did not lead us to a good unified approach using wavelets, and so we returned to using the simpler and more straightforward FFT as our primary DSP tool.

3.1 Spectra and Time Series

Figure 3.1 shows a spectral analysis of an audio sample that is typical of all samples passed through our algorithm. We chose it to introduce our work with spectra and STFTs because it clearly portrays both simple and advanced musical events that we want to identify. This figure shows Natalie Cole singing a chorus of ***Fever***. Elapsed sample time is displayed in seconds along the lower (***X)*** axis (turn the page sideways) where the lyrics are also shown. The total time for the ***Fever*** sample is about 14.3 seconds. We compute several thousand FFTs on the sample in order to get a representation of how the spectrum changes in time. An FFT is computed at the beginning of the sample, then the FFT window is shifted forward in time by a small delta and another FFT is computed, and so on until the end of the sample. All FFTs use the same window size. In this example the window size is 2048 data points of the original music sample (2 Ksamples). The time shift between FFTs is much smaller than the FFT window size. In this example we use 441 data points of the audio sample as our time delta. At 44,100 Ks/sec, an FFT shift of 441 points means 10 milliseconds resolution in the specgram. A shift of 132 points equals 3 milliseconds, and so on. More detailed analysis is presented in the next section.

A 2 Ks audio sample gives an FFT spectrum of 1025 evenly spaced frequencies. The number of frequencies is computed as (N/2) + 1 where N is the size of the FFT window (2048). The frequencies found by the FFT are displayed on the ***Y*** axis, from 20 Hz to 22,500 Hz which is the Nyquist frequency of the CD sampling rate. Dividing 22,500 by 1025 gives a frequency resolution of about 22 Hz for the 2 Ks FFT. A 1 Ks FFT gives frequency resolution of about 44 Hz, and a 4 Ks FFT yields 11 Hz resolution. There is a tradeoff between resolution in the time and frequency dimensions. While this frequency resolution is sufficient for the current work, it is quite limiting. There are some important efficiency and optimization issues that we explore in the section on future work.

The somewhat regularly spaced sharp vertical red lines are primarily finger snap and conga events, and also include some portions of other drum sounds. The yellow tips

extending into the blue "sky" that are *exactly* evenly spaced are Ray Charles snapping his fingers on the back beat. Specgram colors are determined by the audio power of each frequency at each time point, normalized to [0,1]. Blue is low and red is high power.

In the lower third of the diagram are patterns of wavy red lines, regularly spaced in the vertical axis. This is the spectrum of Natalie Cole's voice. You can visually correlate these frequency features with the lyrics written below. The concentration of red at the bottom are the frequencies generated mostly by bass drum and bass guitar. These signals are less well defined than a human singer or melodic instrument like a trumpet. This is partly a limitation of the FFT frequency resolution, partly typical of features that can be extracted at low frequencies and partly spectral structure of the sounds of these instruments *per se*. Both the physics of sound waves and the physiology of the human ear limit the information available at these low frequencies. These physical limits are probably part of why the human voice uses a higher range of frequencies, 200 Hz to several thousand Hz, to encode the majority of speech information.

The details of the correlated frequencies of Natalie Cole's voice are the data that give rise to our perception of *timbre* which is the quality of sound that we associate with identifiable audio events such as words or phonemes (speech), and musical instrument identification. As noted in (Dowling & Harwood, 1986) the higher frequencies (200 Hz to 5000 Hz) encode most of the information about timbre as heard by humans. These frequencies are directly perceived in the human ear by stimulation of frequency sensing "hair" cells in the cochlea. The frequency responses of the hairs is determined by their location in the cochlea, with high frequencies being sensed near the eardrum, and lower frequencies sensed at the far end of the cochlea. Pitch (or tone) is encoded more in the lower frequencies (20 Hz to 2000 Hz) which are sensed both by the frequency sensitive hairs as well as the beat phenomenon (see Appendix E). This is a reason why most melodic information is in the middle and lower frequencies: the precision of our pitch perception diminishes as the frequency goes above about 5000 Hz. Our sensitivity to higher

pitched sounds is not diminished -- hearing the snap of a twig makes the difference between the tiger having you for lunch or not. At high frequencies however, direction and distance are more important information than precise identification of pitch. (Dowling & Harwood, 1986) discuss this aspect of human hearing in some detail. They also mention, as does (Buser, et al. 1992), that human tone perception is ***not*** perfectly correlated with the measured frequency.

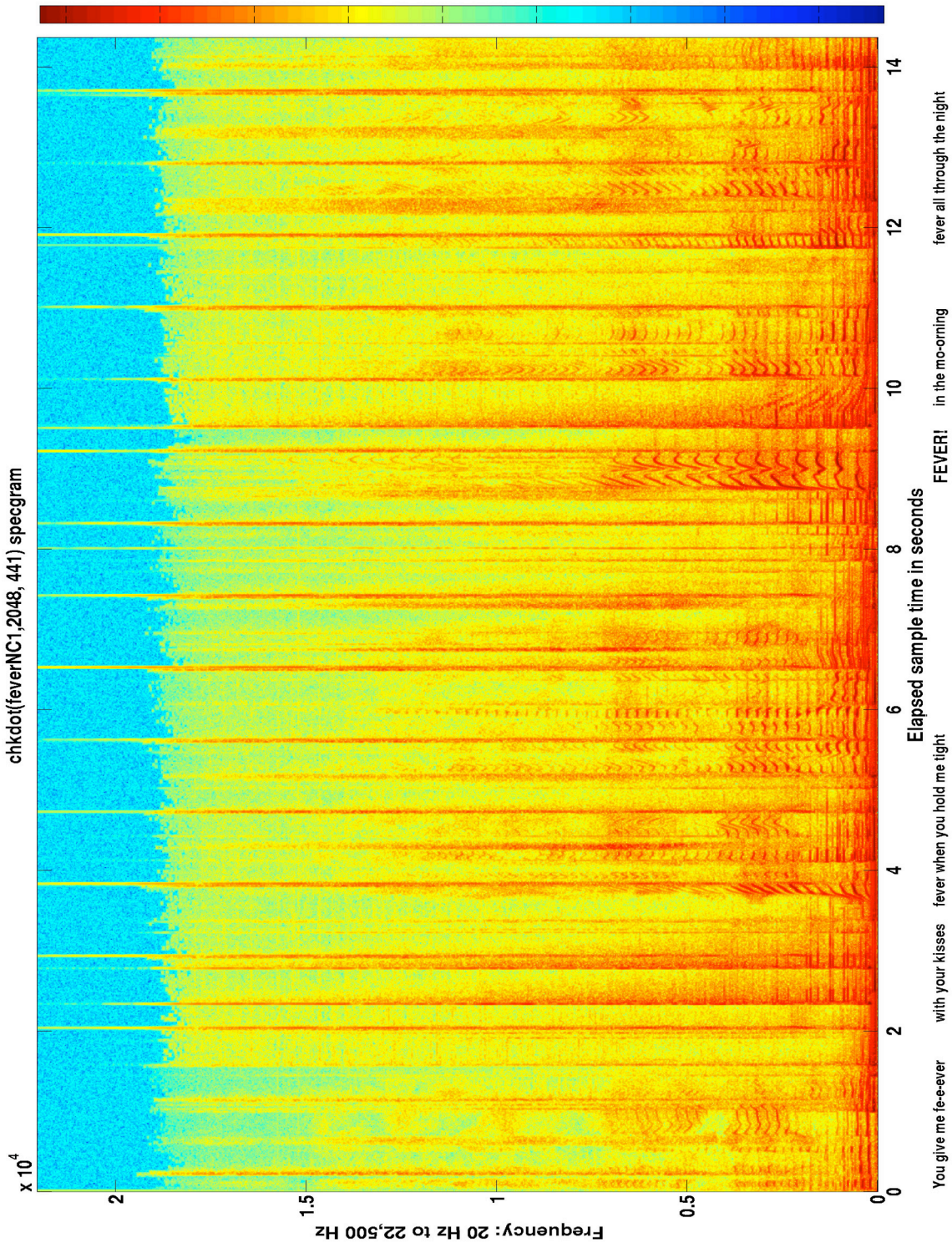

Figure 3.1 Spectrogram for Natalie Cole singing the chorus of ***Fever***

3.2 FFT and STFT

An FFT returns an array of the frequencies contained in a musical sample during a particular short time slice of data. The frequencies change quickly and incessantly throughout any "interesting" sound. Of course there is music with very slow moving changes of frequency content which is also interesting, e.g. Pink Floyd, or classical Pastoral music. Our techniques could be applied successfully to this sort of music, but in this work we only investigate music with quick events. In order to generate a picture of how the frequency spectrum of a music sample changes in time, we apply the FFT repeatedly with a slight time change between successive FFTs. This is commonly referred to as Short Time Fourier Transform (STFT), yielding a spectrogram or, as Matlab calls it, specgram. The time/frequency tradeoff is very important for making an STFT useful. A short FFT gives coarser frequency resolution and finer time resolution. Conversely, a long FFT give more frequency detail, but for a sample whose frequencies are averaged together over a longer time, yielding a less precise view of the temporal changes. Depending on the frequency content of a piece of music, adjusting the frequency and time granularities brings the resultant specgram into clearer "focus," in the sense of rendering particular details more or less clearly. These adjustments are very much like focusing a camera lens, but rather than focusing a spatial image, we change between looking more precisely at time or looking more precisely at frequency. There is a fundamental limit to the total precision, governed by the equivalent of the Heisenburg uncertainty principal.

The STFT approach we use performs a sequence of overlapping FFTs on the musical audio data. The FFT window size (time slice) for any particular run of the algorithm on a music sample is constant, e.g. a time slice of 1024, 2048 or 4096 samples of input data (1024 samples == 1 Kilo sample or 1 Ks). These three window sizes correspond to time intervals of 23, 46 and 93 milliseconds respectively, at the CD sample rate of 44,100 samples per second. The FFTs yield a frequency resolution of 44, 22 or 11 hertz respectively for 1 Ks, 2 Ks and 4 Ks lengths. We tested a few samples with 8192 length FFTs,

but in these cases the time span of the FFT significantly blurs the frequency details, and were generally not very useful. Similarly, using a 512 sample window for the FFT produces a very choppy picture of how the music sample changes in time, which makes detecting note events difficult due to the numerous small spikes in the temporal waveforms that we generate from the STFT.

The time slice and frequency information for a particular set of STFT parameters can be computed with simple algebra. For a time slice (i.e. FFT window size) of N_{ts} data points, and sampling frequency F_s samples/sec, the time occupied by the FFT window T_{fft} is given by $T_{fft} = N_{ts} / F_s$. For convenience and efficiency we use powers of 2 for the size of N_{ts} , e.g. $1024 = 2^{10}$, $2048 = 2^{11}$, $4096 = 2^{12}$. In some cases we use an FFT window which is the sum of numbers that are powers of 2, e.g. $3072 = 2^{11} + 2^{10}$. After performing the FFT, we obtain a vector of frequencies contained in the audio sample. The number of frequencies N_F is determined by the FFT window size, counted in audio data points. The formula is $N_F = (N_{ts} / 2) + 1$, so a 2048 point FFT yields 1025 distinct frequencies. The specific frequencies contained in the frequency vector are integer multiples of the "fundamental" frequency of this particular FFT, based on the formula $F_{fund} = 1 / T_{fft}$, so the frequency set is given by $F_{\{\}} = \{ n * F_{fund} : n = 1, 2, 3, ... N_F \}$.

The time shift between FFTs is the same for each run of the algorithm. Depending on what temporal resolution we want, we may run a sample several times with different time shifts in order to find an optimal resolution for specific musical features. In some cases we will present results with several different time/frequency settings for the same music sample. Typically we used time shifts between 3 and 10 milliseconds, although we tested some music samples using as short as 0.5 milliseconds time granularity.

The frequencies in the spectrum of the FFT are equally or *linearly* spaced: the difference between adjacent frequencies in cycles per second (Hz) is the same whether they are low frequencies or high frequencies. This is inherent to the design of Fourier Analy-

sis, and we consider it a disadvantage, which we explore in chapter 6. The frequency spacing in the human perceptual apparatus, and also in the spacing of note pitches in music is *exponential*. This means that the "distance" between adjacent notes as measured in Hz increases for higher frequencies and decreases for lower frequencies. Given two pairs of different adjacent notes on the keyboard, there are no two pairs that have the same frequency distance in Hz between note N-1 and N-2 compared to N-3 and N-4. For example, the number of notes between middle C and the C above or below is the same: seven white keys and five black keys on a piano, i.e., twelve half steps. The frequency in Hz of the tone corresponding to these C notes is doubled if you go up the scale or divided in half if you go down. This means that the frequency spacing of the notes within each octave is also doubled or divided in half compared to the corresponding note one octave below or above the note being currently examined. In practical terms, there are too many frequencies measured by the FFT in the upper part of the spectrum, and not enough different frequencies delivered by the FFT in the lower part of the spectrum.

3.3 Windows and Filters

Windows are scaling functions used in conjunction with the FFT algorithm for the purpose of improving results and decreasing false artifacts in the transformed data. A window is typically a simple function such as a gaussian curve that modifies the current slice of audio data, prior to the FFT. This modification is simply a sample by sample multiplication of the audio data and the window data. This yields a data slice of the same length as the original data that is smoothly reduced to zero at the beginning and end of the slice. The main effect of this pre-processing is to reduce or remove ***aliasing*** artifacts. These are the result of wraparound effects in the Fourier transform when it converts the audio data from the time domain to the frequency domain. For a detailed technical analysis, see (Brigham, 1974), (Press et al., 2002), and (Hamming, 1983). Figure 3.3 shows a sample of audio data (green), gaussian window function (blue) and the composite (red),

ready for passing into the FFT. Hundreds or thousands of such slices are processed for every musical sample processed by the STFT.

Filters are functions applied to incoming data which change the frequency content of a data sample by reducing or amplifying some range of frequencies. This may simplify subsequent processing of the data sample, or the filtering step can produce useful results directly such as measuring the power of the signal in the range of the filter. Our analysis of the compute costs of filter processing compared to FFT processing led us to prefer the FFT for the current work. The FFT approach was similar in compute cost and substantially simpler in system design, saving development time.

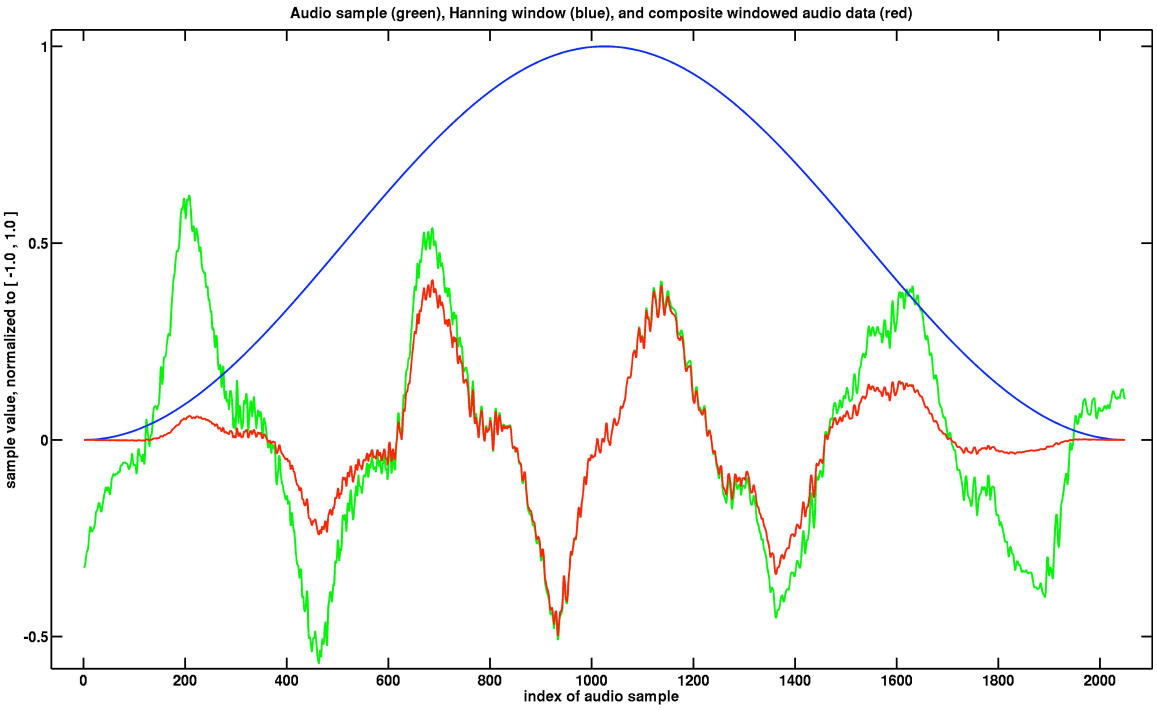

Figure 3.3 Audio Data, Window Function and Composite Result for FFT

3.4 ICA (Independent Components Analysis)

Independent components analysis is a recently developed technique useful for separating several sources of data that are mixed together in one or more data streams

(blind source separation or BSS). We briefly investigated this approach during work on note identification. Figure 3.4 shows how ICA might be used for identification of an instrument's note events by compositing a short data sample of the desired instrument sound with a longer data stream. In essence, this is a form of autocorrelation, but one that matches statistical patterns rather than exact waveforms. The bold, clear vertical line of correlated data points and the elliptical "galaxy" indicate that the data stream included the sound of the pandeiro in this case. This is a promising area for future research.

(Anemüller & Gramss, 1999) used artificial neural networks in preference to ICA for the task of source separation, claiming fast learning (about one second) for their algorithm to be able to distinguish two mixed sounds recorded in an anechoic chamber. Their network topology was a variation of feed forward multi-layer perception.

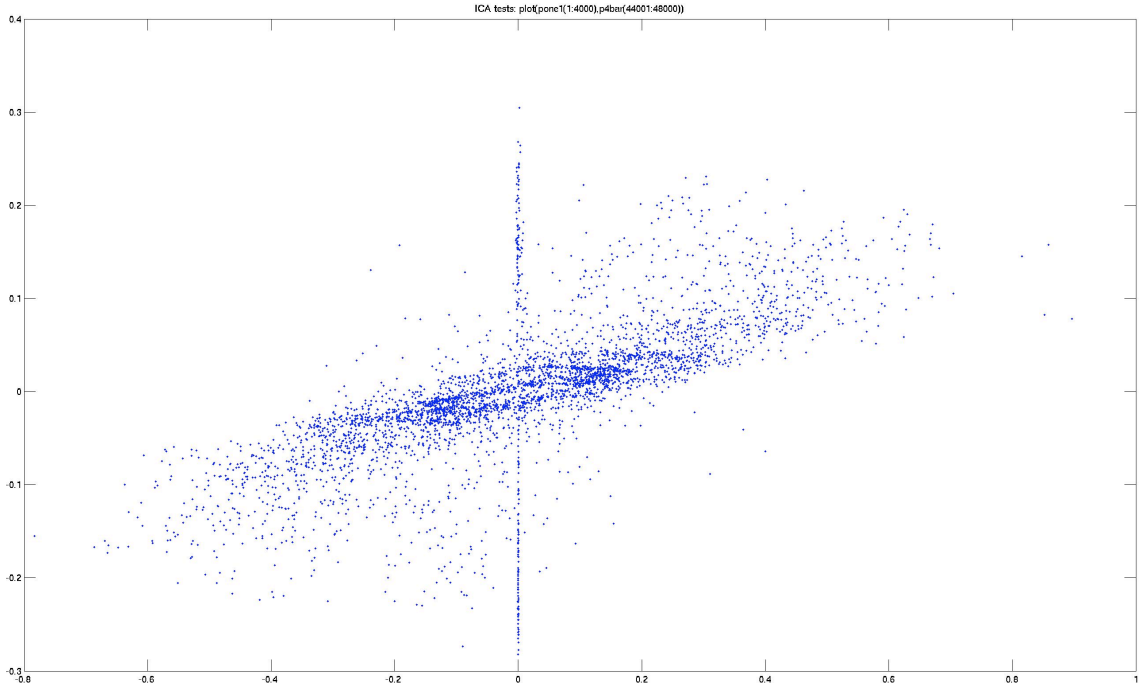

Figure 3.4 ICA Autocorrelation Plot Showing Identification of Pandeiro

3.5 Wavelets

Wavelet analysis decomposes a single large data set into several smaller data sets. These are determined, analogously to Fourier analysis, in terms of basis functions span-

ning a function space (Hilbert or Banach space). The function space for our work is simply the waveforms of the audio samples being analyzed. Fourier analysis creates a set of sine and cosine waveforms and their amplitude coefficients which, when added together, recreate the original waveform. Similarly, wavelet analysis uses a mother wavelet and copies of the mother wavelet which are scaled and translated so that the resultant set of waveforms and their coefficients will accurately represent the original waveform.

We investigated wavelets hoping to use them as a source of features for identification of note events contained in the data set. Wavelets could be used to identify note events, while also identifying the temporal location of these events in the audio stream. Our preliminary investigation and several papers in the computer music research field indicate that wavelets could be a useful analysis framework for musical note events and audio streams. We chose to abandon this line of research due to its technical difficulty and because of efficiency and scalability concerns. While an individual musical note event is both tractable and practical to analyze using wavelets, the extension to analyzing a complex audio stream with multiple instruments would entail a compute cost that we think scales at least as $\boldsymbol{O}(N^2)$ or worse for a number N of different note types. In contrast, Fourier analysis using the FFT is an $\boldsymbol{O}(N \log(N))$ operation. These numbers are merely indicative, and a complete analysis would involve considering both the number of steps in either the wavelet or FFT process, as well as the true compute cost of each step. We believe that wavelets could be used in an effective and efficient manner, but would require a deep knowledge of mathematics (Hilbert space etc) that is beyond our expertise.

3.6 Zero Crossings

Zero crossings are time points where the input audio signal power level (voltage or sound pressure level) goes from positive to negative or vice versa. The data points themselves may not equal zero exactly, in which case we noted the time points of sign changes and plotted the time of the first data point after the sign change as a zero crossing

event. While this technique is commonly cited in the literature, its utility was not immediately obvious for our work and after a short investigation moved on to other techniques.

Figure 3.6 shows a short sample of a pandeiro *pee* note event, with zero crossings marked as sets of blue dots along the two horizontal lines $\boldsymbol{Y} = \pm 0.8$. The audio sample is plotted in cyan. We count the zero crossings and show this count by the red, black, blue and magenta dotted lines. The blue line shows the count in a moving 40 data point window, red uses an 80 point window, black uses a 160 point window and magenta uses a 320 point window. The lines for the counts are normalized to fit into the same vertical scale as the audio waveform. Thus the lines show the relative count rather than the absolute count of zero crossings in their respective windows.

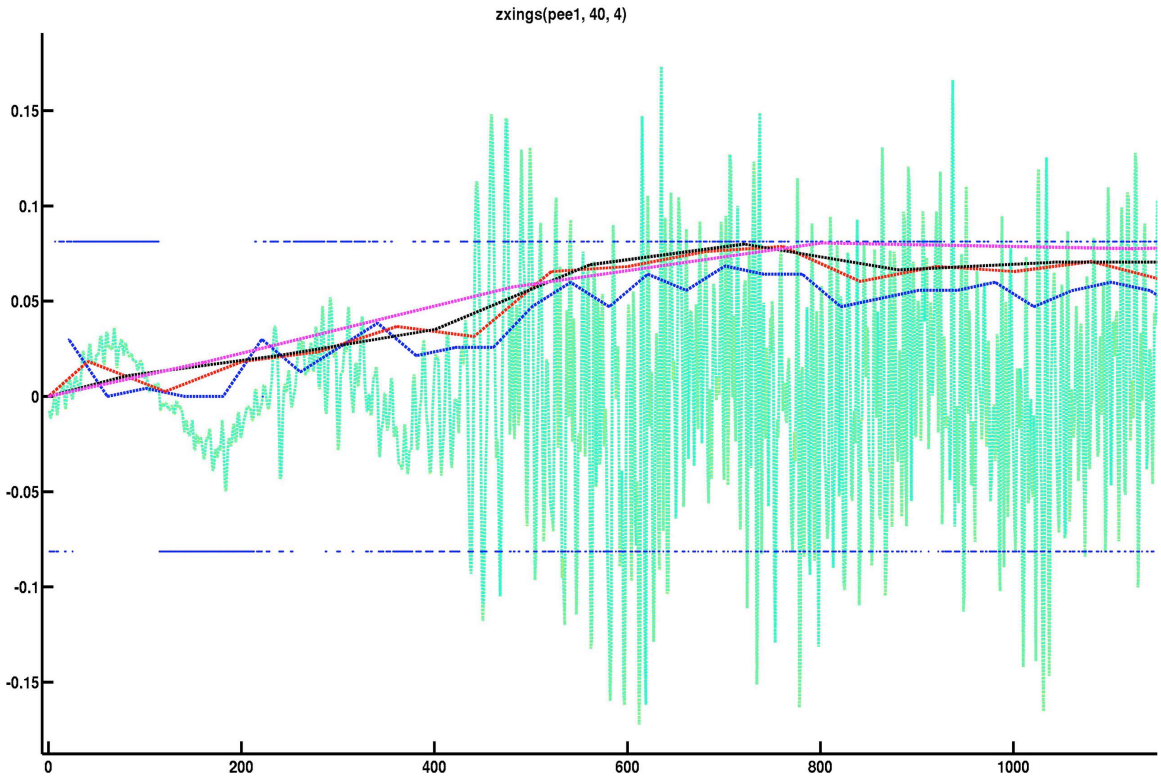

Figure 3.6 Zero Crossings in a Pandeiro Note Event (close-up)

3.7 Signal and Noise

For the most part, the music recognition literature differentiates percussion sounds from melodic and other instruments. This is a reasonable distinction because harmonically correlated sounds such as pitched or melodic instruments are fundamentally different from most percussion note events, which tend to have strong non harmonic features and characteristics. The jargon used to describe these distinctions, however, is sometimes misleading and should be amended in favor of more accurate language.

While there is some discussion about the pitch qualities of percussive sounds, percussion and drum sounds are commonly referred to as "noise". We believe this is essentially an ignorant viewpoint. Noise is merely information that is not properly understood. The canonical form of noise, white noise, is an idealized gaussian distribution of all frequencies with very useful properties. Noise as a name for some more generalized category of information takes on a wide variety of characteristics. In audio production, a low level noise signal, e.g. water or wind sounds, is commonly used as "bed" or foundation for the mix of a soundtrack. This provides subliminal shaping of the listener's perception of the meaning of the soundtrack and if done skillfully, greatly enhances the believability of the soundtrack. Usually this is *very* skillfully mixed to the point where most listeners are not explicitly aware of this shaping of perception. Another common use of noise is in the visual effects industry for film, where libraries of "film grain" are always used to help blend the computer generated graphics with real world scenes. Early productions in the 1980's and 1990's (e.g. ***The Abyss*** or ***Babylon 5***) would sometimes omit the film grain and the special effects from these early works have a very "clean" quality, whereas modern computer effects are blended much more skillfully with real world footage. Again, most people who are not professionals in the field are rarely aware of these subtleties, but the presences of this noise work greatly enhances the immersive believability of the audio or visual piece.

While percussion sounds are complex statistical entities having far less harmonic correlation than melodic instruments, there is little difficulty for our human perceptual system to distinguish between most types of individual percussion note events, even if they are very similar to each other, or mixed with several other instruments' sounds. Thus these sounds are not noise in the sense of being random or unpredictable.

In our work we easily distinguish different drum note events by using a simple frequency based approach, without resorting to any statistics. We have also seen examples where the simple frequency summing technique fails to separate two somewhat similar sounds, such as caixa and shaker. From our survey of the research literature, we expect that refining our note identification process by using simple statistics will produce useful improvements with moderate effort. We discuss this a bit further in chapter 6.

3.8 Description of Our DSP Algorithm

Our DSP algorithm (`chkdot.m`) performs an STFT on musical audio data, followed by note event identification logic, and marking of timing patterns. This is implemented as a Matlab script, listed in the appendix. For input data, the code takes a vector of digital audio data, the FFT length, and time delta for shifting to the next FFT. This stage produces the specgram which we then inspect to determine what frequency ranges to use for identifying note events. Optimizing tradeoffs between time and frequency resolution often requires testing several different sets of parameters to get a truly useful specgram. The transformed spectral data, lists of frequencies in the spectrum and time points of the shifted and overlapped FFTs are retained by Matlab as script internal data and used in subsequent passes through the algorithm for analyzing the specgram. Since computing the specgram can be as much as several hundred times more compute cost than running a time/frequency analysis, this allows us to perform multiple analyses of a single useful specgram, in order to get the best results possible in minimum time. This design can be easily adapted for use in a GUI, which we have not yet implemented.

Subsequent passes through `chkdot` use several vectors and matrices for guiding the DSP and note ID logic. These include a vector that specifies which frequency ranges to use for identifying note events (event tracks), a vector specifying how note events should be counted in the pulse track, a vector specifying in which secondary event tracks to mark events, a vector indicating how to subdivide the sample time based on the primary events detected in the pulse track, and a matrix of threshold values to use on the waveform in each event track. Thresholds can be specified for the waveform itself and its first and second derivatives. For each time slice, in each frequency range, we sum the values of each FFT for the frequency ranges specified in the frequency vector. This gives a sequence of points that represents, for each time frame, the signal's audio power in the current frequency range. These points are plotted as time series that show the changing power levels of the audio signal in the several frequency ranges specified.

The primary pattern recognition logic, after the STFT, uses thresholds of the amplitude changes between the time frame data points. We use the power level of the signal, and the first and second derivatives implemented as first and second order difference equations. Figure 3.8 shows a composite of the waveforms for the standard pandeiro batida, along with the first and second derivatives of the waveform.

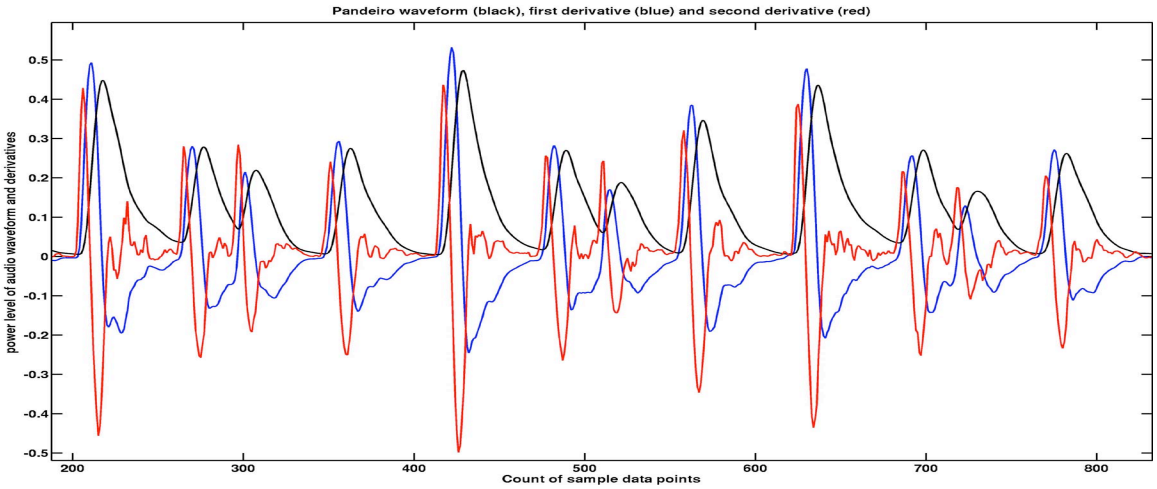

Figure 3.8 Pandeiro Waveform, First and Second Derivatives