Instrument Timbres and Pitch Estimation in Polyphonic Music

A Thesis
Presented to
The Academic Faculty

by

Beatus Dominik Loeffler

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Electrical and Computer Engineering

School of Electrical and Computer Engineering
Georgia Institute of Technology
May 2006

Instrument Timbres and Pitch Estimation in Polyphonic Music

Approved by:

Professor Chin-Hui Lee
School of Electrical and Computer Engineering
Georgia Institute of Technology, Advisor

Professor Aaron D. Lanterman
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor David V. Anderson
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: April 10, 2006

To

Michael and Waltraud Löffler

ACKNOWLEDGEMENTS

I want to thank my advisor Dr. Chin-Hui Lee for his guidance and patience, Dr. Lanterman and Dr. Anderson for additional input, Dr. Rüdiger Naumann-Etienne and his foundation for generously sponsoring my studies throughout the two past years, and Carlton Parker of the World Student Fund for his administrative and organizational work. Without their support, this work would not have been possible.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

In the past decade, the availability of digitally encoded, downloadable music has increased dramatically, pushed mainly by the release of the now famous MP3 compression format (Fraunhofer-Gesellschaft, 1994). Online sales of music in the US doubled in 2005, according to a recent news article [2], while the number of files exchanged on P2P platforms is much higher, but hard to estimate.

The existing and coming informational flood in digital music prompts the need for sophisticated content-based information retrieval. Query-by-Humming is a prototypical technique aimed at locating pieces of music by melody; automatic annotation algorithms seek to enable finer search criteria, such as instruments, genre, or meter. Score transcription systems strive for an abstract, compressed form of a piece of music understandable by composers and musicians. Much research still has to be performed to achieve these goals.

This thesis connects essential knowledge about music and human auditory perception with signal processing algorithms to solve the specific problem of pitch estimation. The designed algorithm obtains an estimate of the magnitude spectrum via STFT and models the harmonic structure of each pitch contained in the magnitude spectrum with Gaussian density mixtures, whose parameters are subsequently estimated via an Expectation-Maximization (EM) algorithm. Heuristics for EM initialization are formulated mathematically.

The system is implemented in MATLAB, featuring a GUI that provides for visual (spectrogram) and numerical (console) verification of results. The algorithm is tested using an array of data ranging from single to triple superposed instrument recordings. Its advantages and limitations are discussed, and a brief outlook over potential future research is given.

CHAPTER I

INTRODUCTION

1.1 Organization and Goals of this Project

This introductory chapter explains some fundamental knowledge and frequently used terms about pitch, timbre and human perception of pitch. Some special cases of instrument timbres and pitch perceptional peculiarities are treated in more detail to demonstrate the breadth of the field of study.

Chapter 2, Overview of Pitch Estimation Approaches, gives a brief overview of some previous approaches to pitch estimation in the simpler monophonic context. Also, a general algorithm scheme for developing polyphonic pitch estimators is enumerated.

Chapter 3, A GMM Approach to Pitch Estimation, introduces key mathematical concepts used in a subsequent implementation of a pitch estimation system, first for a monophonic case, then for the more general polyphonic case. A Gaussian Mixture Model (GMM) is used to model the harmonics in the spectral magnitude. Model parameters are estimated via an Expectation Maximization algorithm.

The subsequent chapter, Development of Heuristics for Initial Estimates, discusses ways our EM algorithm can be initialized in the case of pitch estimation, both for single and multiple pitch cases. Issues related to running the algorithms on real data are discussed, and further heuristics are developed.

Finally, Chapters 5 and 6 discuss problems with and limitations of our current implementation and propose various future enhancements.

1.2 Pitch

1.2.1 Definition

The term *pitch* describes a psychoacoustic sensation of the auditory system and is therefore subjective. It should not be confused with *frequency* or *note*, although it has intimate

relationships with those terms.

A particular note on the keyboard of a piano can be matched to a pure sinusoidal single frequency by a listener, but the frequency spectrum of the signal produced by the piano will carry more than that sinusoidal frequency. Instead, it will show a more complex mixture of different frequencies with different weights, enabling the listener to clearly discern between instruments (Chapter 1.3). Psychoacoustic studies try to discover how the human hearing system matches these two quite different signals.

Frequency denotes a precisely defined analytic term, such as the number of vibrations of a string per second or the resulting vibration of atmospheric pressure carrying the signal to the listener's ear. The term will often be used in the context of the DFT or spectrograms, which contain DFT analyses.

For the purpose of this thesis, a *note* is simply the standardized name of a pitch, such as A#, or the graphical representation of it on the staff. It can thus be considered a synonym for pitch, if the definition relating the two is known. This leads to the topic of *pitch standards*, and there is a long history of such. Today, the most commonly used pitch standard is ISO 16 [14] (Appendix).

In ISO 16, the A above middle C (A4) must be tuned to $440Hz$. This note is the center of attention in pitch standards, and all other pitches are typically tuned with respect to it.

1.2.2 Harmonics, Overtones, and Partials

As mentioned before, a pitch may be considered more complex than a single frequency; it is a mixture of sinusoids of different frequencies with different weights that are perceived to be equivalent to a single frequency. These "other components" are the so-called *harmonics*, which make the tone richer and more interesting to the listener. In fact, in music, we almost never deal with pure tones, that is, pitches of single frequency. These sound bland and boring, and the possibility of electronically producing a pure tone has not been around for long, considering the entire history of music. This mixture of frequencies is a kind of "signature" or "fingerprint" called *timbre*, which is dependent on the instrument's physical specifications (length, diameter, width, girth, specific shape, density of the materials used,

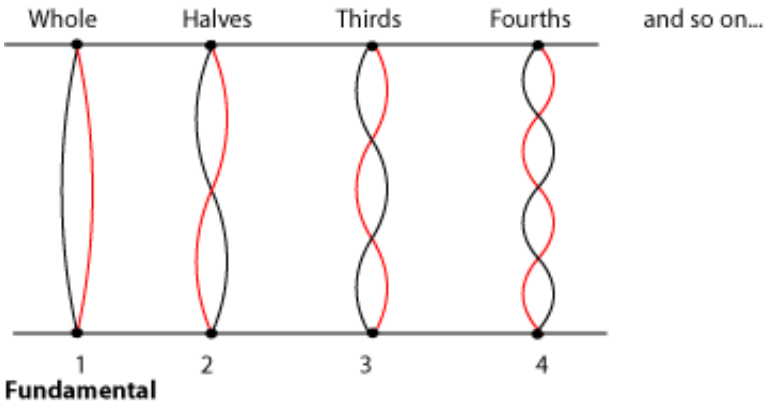

Figure 1: Harmonics of strings [26]

etc). We can think of timbre as the "color" or "quality" of sound.

For an example of timbre, when a string vibrates, it does not vibrate like only a single sinusoid. It will vibrate with a basic frequency ω_0, but it will also vibrate at integer multiples of that basic frequency.

Expressed in the frequency domain,

$$\omega_m = m \cdot \omega_0 \,. \tag{1}$$

The same principle can be observed with the air pressure in a trumpet or other winds, or the human voice. The above description is a simplification; the partials of real instruments have amplitudes and frequencies that change with time. In particular, they may not line up at exact integer multiples, and even the fundamental may change frequency over time due to a vibrato.

Figure 2 shows a Discrete Fourier Transform of a short (64ms) piece of an oboe's C4 pitch. Recall that the Fourier transform of a single sinusoid is

$$F\left[\sin(\omega_c t)\right] = \frac{1}{2} j \left(\delta(\omega + \omega_c) - \delta(\omega - \omega_c)\right). \tag{2}$$

This makes it clear that if we show only the positive frequency range and if we take the magnitude, we will see a single delta peak in the spectrum. We should discover something similar for the DFT magnitude of a musical pitch. Each harmonic should yield a spike located near an integer multiple of the fundamental (typically the spike with the lowest frequency).

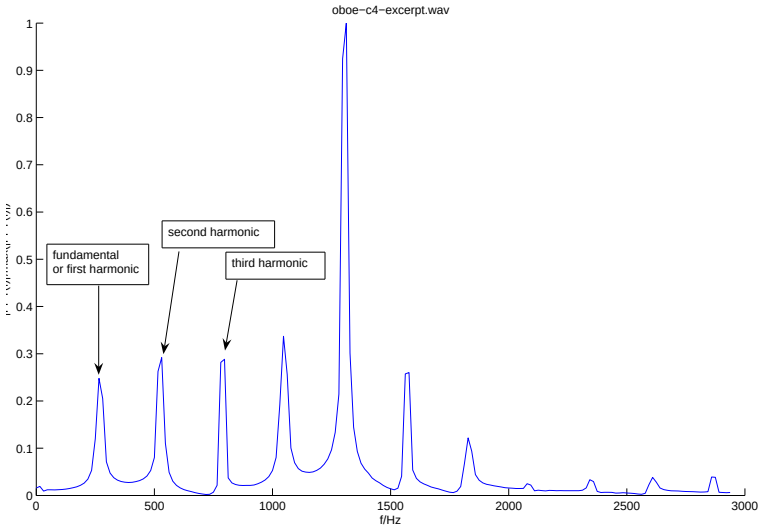

Figure 2: Harmonics of an oboe C4 tone, DFT magnitude

Figure 2 does show significant peaks. Of course, they are not true delta peaks, since the analysis is over a finite window of data and real instruments cannot be perfectly periodic. We also see that the peaks of the harmonics are located at or near integer multiples of the fundamental.

The term *overtone* is a general term for referring to all the frequency components besides the fundamental. Overtones will be categorized into *harmonics* that are integer multiples of the fundamental, and *partials*, which consist of the rest.

This nomenclature is not standard; most authors consider *all* harmonics to be partials (but not all partials to be harmonics). Our convention allows us to avoid the cumbersome expression "nonharmonic partials." (see Figure 2)

1.2.3 Pitch Spacing in Western Music

The following table shows the pitches of a chromatic scale from C4 to C5 in western music.

This scheme of musical tuning is called a *12-tone equal temperament* system, in which an octave is subdivided into a series of *equal pitch ratios*. The pitch is not linearly spaced across the octave (the last row shows the numbers for a hypothetical linear spacing) since humans perceive frequencies logarithmically. Mathematically speaking, the pitches of the scale follow a *geometric sequence*.

Table 1: Common pitch spacing

Note	C4	C#4	D4	D#4	E4	F4	F#4	G4
Pitch/Hz	261.6	277.2	293.7	311.1	329.6	349.2	370	392
Lin.	261.6	283.4	305.2	327	348.8	370.1	392.5	414.3
Note	G#4	A4	A#4	B4	C5			
Pitch/Hz	415.3	440	466.2	493.9	523.3			
Lin.	436.1	457.9	479.7	501.5	523.3			

1.2.4 Musical Scales and the Weber-Fechner-Law of Perception

Human perception, in general, is not associated with a linear scale. A general law describing human response to physical stimulus was discovered by Ernst Heinrich Weber and Gustav Theodor Fechner in 1860 [6]. A later extension was made by Stanley Smith Stevens in 1957 [27].

Stimulus (S) and perception (p) can be related by the differential equation

$$dp = k \cdot \frac{dS}{S}\,, \tag{3}$$

which is solved by

$$p = k \cdot \ln\left(\frac{S}{S_0}\right) = k \cdot \ln(2) \cdot \log_2\left(\frac{S}{S_0}\right) = \tilde{k} \cdot \log_2\left(\frac{S}{S_0}\right), \tag{4}$$

where $\tilde{k}$ is an experimentally determined constant, and S_0 denotes a stimulus threshold below which nothing is perceived. Vice versa, this means that if we scale the perception linearly, such as with the musical scale, the stimulus will be a certain exponential function. In the case of pitch and the table above, it is

$$S = S_0 \cdot 2^{p/\tilde{k}} = S_0 \cdot 2^{p/12}\,. \tag{5}$$

To verify this, set $S_0 = 261.6$ and let $p = 1, 2, \ldots 13$. The spacing of the pitches are important in note transcription systems, in which a nearest-neighbor classification based on this result might be used.

A measure frequently appearing in the literature is the *cent*, which equals one hundredth of an equal tempered semitone. This is equivalent to one twelve-hundredth of an octave, since there are twelve semitones to the octave. Thus, if the frequency f_2 is said to be a cent

higher than f_1, then

$$\frac{f_1}{f_2} = 2^{1/1200} \approx 1.00057779\,. \tag{6}$$

Humans can notice a difference in pitch of about 5 to 6 cents, which is an astoundingly high degree of resolution.

Despite this result for the pitch scale, a deviation can be observed for very high frequencies such as the 7^{th} and 8^{th} octaves and higher. Pitches perceived and performed by absolute pitch possessors and flutists, respectively, exceed the theoretical value, and do so increasingly the higher the note. An experimental study and discussion of physiological origins can be found in [22]. For the purpose of this project, this phenomenon shall not be taken into account.

1.2.5 Motivation for Pitch Estimation

Some applications of pitch estimation include

- Melody note detection and query-by-humming (information retrieval)
- Score transcription systems / conversion to MIDI
- Music visualization
- Polyphonic sound separation, e.g. for instrument identification

1.3 Timbre of Instruments

The general ideas of pitch, timbre and the relationship of the harmonics have been presented previously. This section makes some more general remarks and reviews specific characteristics of instrument timbres.

In particular, timbres differ mostly in the weight and number of the harmonics, and differ to a much lesser extent in the basic shape of the peaks (this fact will be exploited later in our mathematical modeling). The spectral magnitude of most acoustic instruments approximately follows a $1/f$ curve above a certain frequency, so most of the significant timbral parameters are found in the lower frequency bands. This fact will also be exploited later.

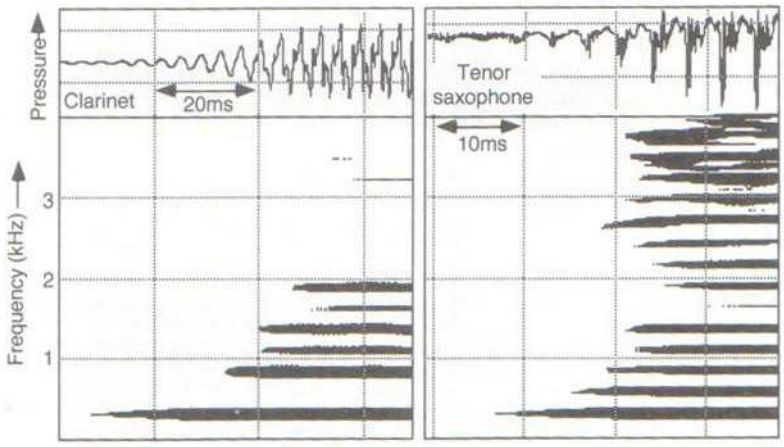

Figure 3: Onset of clarinet and sax, from [12] p.205
fig:peakCutoffs

1.3.1 The Three Phases of a Tone

So far, we have assumed that timbre is constant over time. The three temporal phases of a tone (onset/attack, steady state, offset/release/decay) each help define the overall timbre of an instrument. The attack particularly enriches the musical listening experience and seems to reveal important physical parameters not contained in the other phases. The decay, on the other hand, is far less important as far as timbre recognition is concerned, since the timbral characteristics can be clearly heard during the (often much longer) steady-state phase, which inherently always precedes the offset phase.

A typical spectral magnitude of a $64ms$ frame of a monophonic piece in the steady state is shown in the introduction, but for better analysis *of this phase*, it is useful to look at time-averaged spectra to be certain of observing general tendencies (often called long-term average spectra, or LTAS). Any noise present in the signal will then approximately average to zero.

For the *onset phase* on the other hand, LTAS would be detrimental, since we have an explicit time dependency of the spectral features. This can be easily demonstrated by some spectrograms, as in Figure 3.

The two instruments in Figure 3 not only differ in number of harmonics and their weighting (the latter is not visible here), but especially in time onset structure of harmonics.

These two instruments, clarinet and saxophone, could easily be differentiated from an organ. As also shown in [12], a principal 8' open flue organ stop has a second harmonic that begins much earlier than the fundamental (about 35ms).

1.3.2 Pitch Dependency of Timbre

Generally, the timbral parameters in all phases are pitch-dependent. This is one reason why a reliable pitch estimation system is a prerequisite to instrument identification. Figure 4 shows the notes A4 and F4 performed by the same viola in the same environment, on different strings.

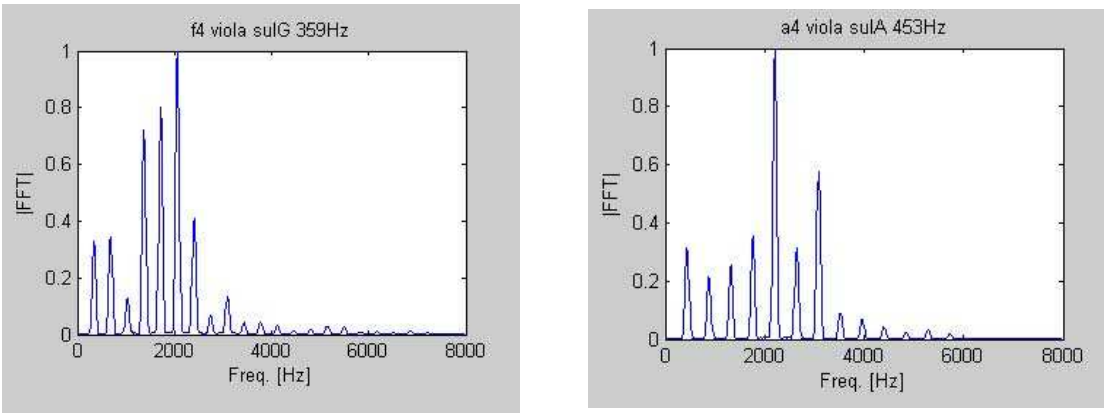

Figure 4: LTAS of the viola's F4 and A4 tones

Although the pitches are quite close, the weights of the harmonics vary considerably between these two tones. This is not a special case; it seems impossible to create a single weighting profile for an instrument across all its pitches.

Figure 5 shows the LTAS of an F4 from a cello. Of course, telling instruments apart within a family such as the strings (violin, viola, cello, bass) is challenging. This is further complicated by the range overlap between different instruments, as shown for the string family in Figure 6.

Furthermore, there is not just an overlap between instruments but also strings on the same instrument (see also Figure 6). This would have to be considered when training a model for an instrument, dramatically increasing the number of samples needed.

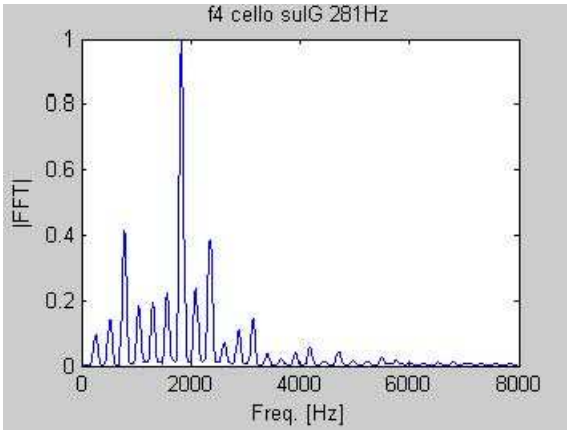

Figure 5: LTAS of the cello's F4 tone

1.3.3 Characteristics of some Specific Instruments

Many instruments have distinctive characteristics. For instance

- Figure 7 show that the peak "feet" have a distinctive shape. One side typically has a sharp upward or downward turn, but on the other side of the valley the change is slight.
- Plucked instruments (such as guitar, strings, harp) and the piano have a particular short-time energy or amplitude envelope. Figure 8 shows the time samples of a piano note, thus indicating the energy envelope.
- The trumpet has strong and regular harmonics up through high frequencies.
- Overblowing of wind instruments or organ pipes allows another octave (or octave and perfect fifth for stopped pipes) to be reached, that otherwise would not be available to the instrument. The timbre changes audibly.
- Stopped organ pipes sound an octave lower than open pipes at the same length, but only the odd-numbered harmonics are present. Conversely, an open pipe and a closed pipe at half the length produce the same pitch, but their timbres are quite different because of the lack of even-numbered harmonics in the signal of the closed pipe ([12] p.176).

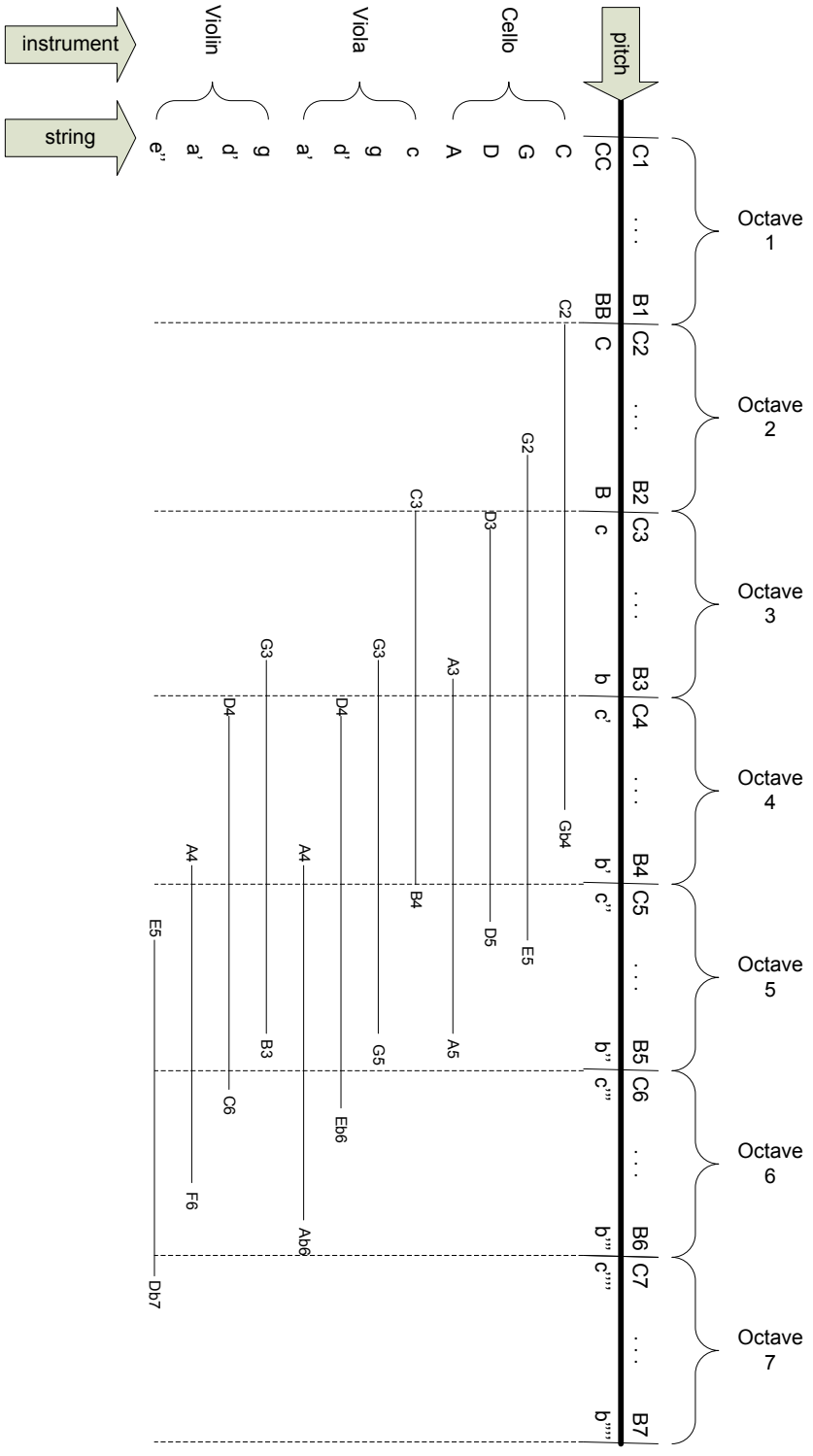

Figure 6: Pitch overlap in the string instrument family

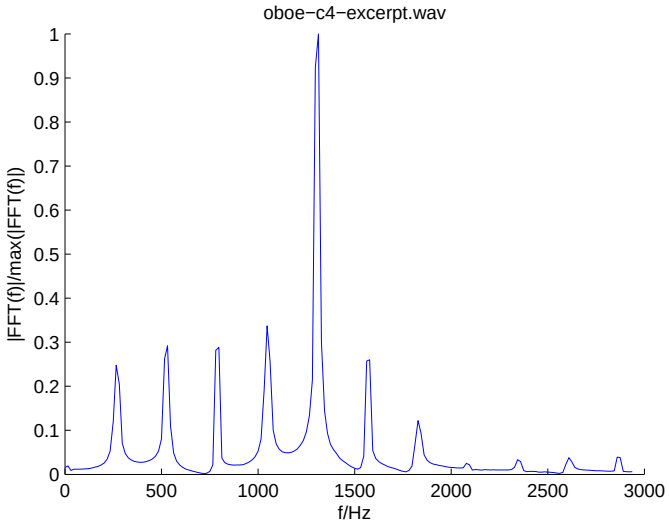

Figure 7: Spectrum of the oboe's C4 tone (no LTAS)

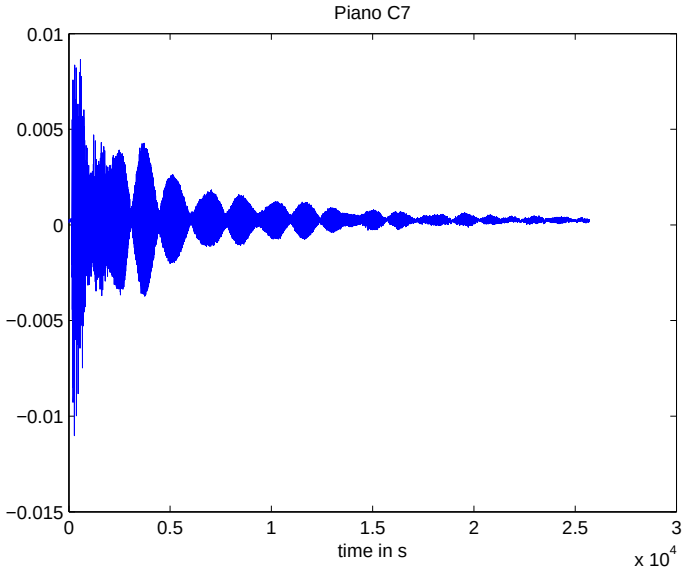

Figure 8: The piano's amplitude over time

1.4 Human Auditory Perception of Pitch

Perceptional studies show that the auditory system is not straightforward. It is still not clear exactly how the auditory cortex, a part of the brain, processes information or how training is performed. However, indications point towards a tonotopical organization of the auditory cortex, that is, certain cells are sensitive to specific frequencies ([23, 30]). Spectrally rich pitches are easier to discriminate than those of sinusoidal tones [28]. Further, it appears that the brain separates encoding of spectrally complex information from temporally complex information [28]. In a similar way it may separate phonetic content from musical content [29].

However the brain processes auditory information internally, some important results may be derived using a black-box approach in which the listener is asked to give subjective perceptual judgement on specially prepared sound sources. The first experiments of this kind were performed by Helmholtz [32], who also formulated the *place theory of pitch perception*, which will be discussed in more detail here. The counterpart, the *temporal theory of pitch perception*, will be skipped because it does not directly relate to our spectrum-based pitch estimation algorithm. Note that neither of the theories alone can sufficiently model actual perception in all cases.

1.4.1 F_0 Extraction Only

The most straightforward pitch determining method extracts only the fundamental, i.e. the first harmonic. The problem is that some sounds may be missing the fundamental, yet the perceived pitch is still the same as if it were present. Examples of sounds of this category are overblown flutes and organs. This fact is also used in small loudspeakers that cannot reproduce lower frequencies well.

1.4.2 Minimum Difference of Adjacent Harmonics

The pitch perception of a sound with a missing harmonic suggests that the brain looks at the spacing between adjacent peaks for pitch determination, so the same idea is used here.

$$pitch = \min_{j \in 1..M-1} f_{j+1} - f_j \ . \tag{7}$$

While this method works for navigating around the simple case where only the fundamental is missing, different mixtures can be created for which this method does not work either. An example is presented in Table 2, in which there are only three harmonics at rather high frequencies relative to their distances. exist. The pitch extracted by the minimum-difference method would clearly be $200Hz$, but this does *not* match interpretation of a human subject.

1.4.3 Highest Common Factor

Suppose we have M harmonics present. To overcome the shortcomings of the preceding methods, the highest common factor approach divides each center frequency of a harmonic by a sequence of integers $1..N$. N should be sufficiently larger tham M. Among this set of numbers, we find the M that are closest to each other so that we will pick one number uniquely from each set associated with a certain harmonic. The pitch is then computed by simply taking the arithmetic mean of these M numbers. Let $\pi_1 \dots \pi_M$ denote the M closest values. The pitch estimate is then

$$pitch = \frac{1}{M} \sum_{i=1}^{M} \pi_i \ . \tag{8}$$

In Table 2, there are three harmonics, so $M = 3$. The three bold numbers in the table correspond to the three closest ones, prohibiting two or more from the same row.

Table 2: HCF example

div by → harmonic ↓	1	2	3	4	5	6	7	8
1	1040	520	346.7	260	**208**	173.3	148.6	130
2	1240	620	413.3	310	248	**206.7**	177.1	155
3	1440	720	480	360	288	240.0	**205.7**	180

The pitch is then determined to be

$$pitch = \frac{208 + 206.7 + 205.7}{3} = 206.8 \approx 207 \ , \tag{9}$$

which corresponds exactly to a human subject's matching of the test tone with a pure frequency.

The reason for looking at these special cases and discussing human perception is that we would like our pitch estimator to work similarly to our brain, in the hope that its results would match what we hear. The heuristics in Chapter 4 do not take all the above discussed cases into account. However, the reader should keep in mind that estimating a pitch is generally a complex task, and that for a more sophisticated system developed in the future, perceptional aspects might assume a higher priority.

CHAPTER II

OVERVIEW OF PITCH ESTIMATION APPROACHES

To our knowledge any approach to the problem of pitch estimation can be categorized into one of two general categories: short-time frame based, or spectrogram based. The former looks at isolated signal slices and the latter uses the whole spectrogram of the entire piece. While the short-time slice approach could be viewed as just a special case of the spectrogram approach, the distinction makes sense if we look at the mathematical formulations. Furthermore, the different approaches are often uses in two application contexts, namely audio streaming and entire-file-based analysis.

The first four sections present four approaches for the time-slice technique. The last section summarizes some spectrogram-based techniques. This overview is not meant to be exhaustive, but it should show some breadth and highlight important general concepts.

2.1 Spectral Smoothing Using "Specmurt Anasylis" (sic)

In principle, the "specmurt anasylis" approach [24] finds the pitch by reducing the set of harmonics to a single peak by a signal filtering technique. It is meant to produce piano-roll visualizations, which are similar to spectrograms. The piano-roll output has a linear pitch frequency axis and is obtained using a wavelet transform (might also use a CQT). The pitch is modeled as a single delta peak in the frequency domain, and the resulting spectrum $v(x)$ is obtained by the convolution of this delta distribution $u(x)$ (or in the polyphonic case, superposed deltas) with a "common harmonic structure" $h(x)$:

$$v(x) = h(x) * u(x) \ . \tag{10}$$

The left side is a known (signal spectrum). We must make an educated guess about the common harmonic structure $h(x)$ (basically, the timbre information), which is crucial to this technique's performance. All that remains is a "simple" deconvolution of the above equation to obtain the pitch estimate(s), which must be picked from the remaining peak(s).

The authors of [24] do not go this far, though, and are satisfied with the visual output and a comparison to the MIDI equivalent. To convert the piano-rolls of their described system to a musical score, a pitch-spelling algorithm would need to be attached (a comparison of such algorithms is given in [20]).

The follow-up paper [25] discusses how to optimize the shape of the common harmonic structure.

2.2 An Expectation Maximization for a Constrained GMM Using an Information Criterion

This approach, given in [15], [16], models each "harmonic structure" (set of harmonics for a particular pitch) with a constrained Gaussian Mixture Model, and is thus similar to the formulation in Chapter 3. The difference is that [15] uses a log-frequency transform and an alternate maximization part which does not only constrain the means, as we do here, but the weights as well. The Akaike information criterion (AIC) is used for estimating the number of concurrent pitches, in this paper denoted as K. A high parameter (in the range of $K = 12$) is initially assumed, and then iteratively reduced. The minimizer of the AIC function across these values will determine the best guess of the true value of K. Thus, this implementation requires the reestimation of the parameters via EM as many as 12 times. The authors of [15] have not included any analytical or experimental remarks about computational complexity.

2.3 A General Iterative Multipitch Estimation Algorithm

The paper titled "Robust Multipitch Estimation for the Analysis and Manipulation of Polyphonic Musical Signals" [17], introduces a general scheme for polyphonic pitch estimation and describes a particular approach to pitch estimation based on Short-Time Fourier Transform (STFT). Bandwise processing of DFT frames is used to come up with bandwise pitch estimates that are then combined into a single score.

The first part of the general scheme, predominant pitch estimation, finds the pitch of the most prominent sound standing out from the background interference of other harmonics and noisy sounds. In the second part, the spectrum of the detected sound is estimated and

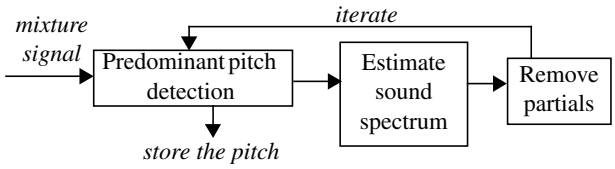

Figure 9: General multipich estimation principle by [17]

subtracted from the mixture. The estimation and subtraction steps are then repeated for the residual signal, until a predefined maximum number of pitches has been reached, or no new pitch estimates can be found confidently.

The follow-up paper [31] takes the outputs of the multipitch estimator described in [17] and improves on their accuracy in the least-squares sense while retaining the structure of the sounds. Smooth linear models are constructed for all significant sounds; the parameters of the models are obtained with a least-squares solution.

2.4 Partial Searching Algorithm with a Tree Search and Dynamic Programming

The algorithm presented in [34] assumes that the fundamentals for each pitch are given. It recognizes that a particular significant peak can be "shared" by the harmonics of two different pitches, and comes up with the best guess for a solution. It tries to find these respective harmonics via a "relaxed harmonics condition," using a particularly defined tree search algorithm and dynamic programming. A peak-picking heuristic, or some other method of coming up with the prospective harmonic locations on the frequency axis, can be used prior to running the algorithm. The results don't look overly promising at present when the number of pitches is not known in advance. However, to our present knowledge, this approach should be investigated further.

2.5 Independent Subspace Analysis, Prior Subspace Analysis, Generalized Prior Subspace Analysis

Independent Subspace Analysis (ISA; [4]) considerably extends Independent Component Analysis (ICA; [13]). First, the one-dimensional signal is projected onto a two-dimensional space using the STFT or Constant-Q Transform [3]. Then, the ICA constraint of requiring

as many mixture observation signals as there are sources is relaxed. Further, dynamic independent components are allowed to account for the nonstationarity of the signal. Sources are tracked by the similarities of dynamic components over small time steps.

Prior Subspace Analysis (PSA; [10], [8]) requires an individual prior subspace for each note of an instrument. Since this is a large collection for most instruments, the first application of PSA to music analysis was an application to drum sounds.

The magnitude spectrogram of the signal is modeled as the superposition of l unknown spectrograms Y_j, and each of the spectrograms can be represented as the outer product of an invariant frequency basis function and an invariant amplitude basis function:

$$\mathbf{Y} = \sum_{j=1}^{l} Y_j = \sum_{j=1}^{l} a_j s_j \ . \tag{11}$$

PSA assumes that there are known frequency basis functions or prior subspaces a_{pr} that are good approximations to the actual subspaces (these must be found using some domain knowledge). Substituting for the a_j with these prior subspaces yields

$$\mathbf{Y} = \sum_{j=1}^{l} a_{pr} s_j = A_{pr} s \ , \tag{12}$$

which is solved in PSA by premultiplying the overall spectrogram with the pseudo-inverse of the frequency basis functions, to get an initial estimate for s. ICA is used to improve these estimates.

In Generalized Prior Subspace Analysis, given in [9], Equation 11 is essentially reinterpreted in such a way that the prior subspaces are viewed as an undercomplete signal dictionary and that the data is sparse in nature, which results in a new version of the PSA algorithm [9].

CHAPTER III

A GMM APPROACH TO PITCH ESTIMATION

This chapter provides the necessary theoretical tools for the pitch estimation approach presented in this document. Our algorithm merely takes spectral information into account, via STFT.

In general, the DFT serves as a good basis for frequency analysis, since its properties are well-explored and implementations using the Fast Fourier Transform (FFT) are efficient. However, other transformations such as the Modulation Spectrogram [11] (originally thought of speech), the Constant-Q-Transform (CQT; [3]), or wavelet transformations make sense [24]. The CQT is mainly motivated by the way it scales the frequency axis: it is linear in the pitches of the chromatic scale discussed in 1.2.3. Generally, psychoacoustic results of pitch perception question the efficiency of the DFT as the basis of the further analysis, but a more detailed study of the suitability of other transforms is outside of the scope of this thesis.

The following sections discuss in detail the mathematical modeling of the harmonics via Gaussian Mixture Models and the Expectation Maximization algorithm designed to estimate the parameters of the model, eventually leading to a set of frequencies (ideally) corresponding to the true center frequencies of the harmonics.

3.1 General Fundamentals of Expectation Maximization Algorithms

The Expectation Maximization (EM) technique is a general method for solving maximum likelihood (ML) estimation problems with incomplete data. It was first formally formulated by Dempster, Laird, and Rubin [18] and extended for superimposed signals by Feder and Weinstein [7].

Assume the probability density function of an observable random variable $\mathbf{Y}$ is $f_{\mathbf{Y}}(y;\Theta)$.

In the EM approach, we hypothesize a "complete data" random variable $\mathbf{X}$, which is nonobservable. They two kinds of data are related by a noninvertible and unknown function H,

$$H(\mathbf{X}) = \mathbf{Y} \ . \tag{13}$$

Therefore, the density for $\mathbf{X}$ can be expressed by

$$f_{\mathbf{X}}(x;\Theta) = f_{\mathbf{X}|\mathbf{Y}=y}(x;\Theta) \cdot f_{\mathbf{Y}}(y;\Theta) \tag{14}$$

over the complete support of H. Rearranging and taking the logarithm of both sides yields

$$\log(f_{\mathbf{Y}}(y;\Theta)) = \log(f_{\mathbf{X}}(x;\Theta)) - \log(f_{\mathbf{X}|\mathbf{Y}=y}(x;\Theta)) \ . \tag{15}$$

If we now take the conditional expectation with respect to X given $\mathbf{Y} = y$ for a parameter value Θ' on both sides, we are left with

$$\begin{aligned} \log(f_{\mathbf{Y}}(y;\Theta)) = \quad & E\left[\log(f_{\mathbf{X}}(x;\Theta))|\mathbf{Y} = y;\Theta'\right] - \\ & E\left[\log(f_{\mathbf{X}|\mathbf{Y}=y}(x;\Theta))|\mathbf{Y} = y;\Theta'\right] \ . \end{aligned} \tag{16}$$

This looks cumbersome, so for convenience we define

$$\begin{aligned} L(\Theta) &= \log(f_{\mathbf{Y}}(y;\Theta)) \\ U(\Theta,\Theta') &= E\left[\log(f_{\mathbf{X}}(x;\Theta))|\mathbf{Y} = y;\Theta'\right] \\ V(\Theta,\Theta') &= E\left[\log(f_{\mathbf{X}|\mathbf{Y}=y}(x;\Theta))|\mathbf{Y} = y;\Theta'\right] \ , \end{aligned} \tag{17}$$

to get the more elegant expression

$$L(\Theta) = U(\Theta,\Theta') - V(\Theta,\Theta') \ . \tag{18}$$

This is a special way of writing the original loglikelihood that we want to maximize. Using Jensen's inequality on the V-term yields

$$V(\Theta,\Theta') \leq V(\Theta',\Theta') \ . \tag{19}$$

We infer that if $U(\Theta,\Theta') > U(\Theta',\Theta')$, then

$$L(\Theta) > L(\Theta') \, . \tag{20}$$

An EM algorithm starts from an arbitrary initialization point $\hat{\Theta}^{(0)}$ at time $t = 0$. Subsequent parameter estimates are computed in the following manner:

- Expectation-step: Compute

$$U(\Theta, \hat{\Theta}^{(t)}) \ . \tag{21}$$

- Maximization-step:

$$\hat{\Theta}^{(t+1)} = \max_{\Theta}(U(\Theta, \hat{\Theta}^{(t)})) \ . \tag{22}$$

- The iteration is repeated, for instance, until

$$\left\| \hat{\Theta}^{(T)} - \hat{\Theta}^{(T+1)} \right\| \leq \varepsilon \tag{23}$$

for some iteration step T (other conditions such as a maximum number of iterations are also possible). Smaller epsilons result in more precise final estimates at the cost of more iteration steps needed to converge.

The formulation follows the scheme of a steepest-descent algorithm. Hence, EM algorithms will converge to a local maximum, and in general we do not know whether or not a particular found local maximum coincides with the global maximum we want. The final results are thus heavily dependent on the starting points, but the EM procedure itself does not define those.

3.2 Fundamentals of Gaussian Mixture Models (GMM)

Gaussian mixture models are intended to express a more general, multimodal probability density function (i.e., multiple peaks) as a superposition of single Gaussian distributions (called "mixtures"). This is convenient because the ML estimates of the parameters of Gaussians are analytically tractable and well-known. In most cases, the GMM formulation can be used as a good approximation to a real distribution, even if mixture components are not really Gaussian. The mathematical formulation is simply

$$f_{\mathbf{X}}(x; \Theta) = \sum\nolimits_{m=1}^{M} w_m \cdot N(x; \Theta_m) \ , \tag{24}$$

where the w_m are mixture-weights that must sum to unity, i.e. $\sum_{m=1}^{M} w_m = 1$, and $\Theta_m = [\sigma_m^2, \mu_m]$ denote the means and variances, as found in the Gaussian density function

$$N(x; \sigma^2, \mu) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2} \right) \ . \tag{25}$$

Further,

$$\Theta = \bigcup_{m=1}^{M} \Theta_m \ , \tag{26}$$

and M, the number of mixture components, is assumed to be known for now.

With knowledge of the observable data x_i, the (unbiased) ML parameter estimates of a single Gaussian are [5]

$$\begin{aligned} \hat{\mu} &= \tfrac{1}{N} \cdot \textstyle\sum_{i=1}^{N} x_i \\ \hat{\sigma}^2 &= \tfrac{1}{N-1} \cdot \textstyle\sum_{i=1}^{N} (x_i - \mu)^2 \ . \end{aligned} \tag{27}$$

3.3 An EM Algorithm for GMM Parameter Estimation

The general EM approach can now be adapted to work on GMM problems. The "missing data" are the means, variances, and weights (or prior member probabilities) of the mixture components, which must be estimated from a limited number of samples.

- The typical expectation step for a GMM [21] yields

$$\hat{P}(m|x_i)^t = \frac{\hat{w}_m^t \cdot N(x_i; \hat{\sigma}_m^{2\,t}, \hat{\mu}_m^t)}{\sum_{k=1}^{M} \hat{w}_k^t \cdot N(x_i; \hat{\sigma}_k^{2\,t}, \hat{\mu}_k^t)} \ , \tag{28}$$

  which is the estimated probability (at iteration step t) that a sample x_i belongs to Gaussian component m.

- The maximization part gives

$$\begin{aligned} \hat{w}_m^{t+1} &= \tfrac{1}{N} \textstyle\sum_{i=1}^{N} \hat{P}(m|x_i)^t \\ \hat{\mu}_m^{t+1} &= \frac{1}{\sum_{i=1}^{N} \hat{P}(m|x_i)^t} \cdot \textstyle\sum_{i=1}^{N} x_i \hat{P}(m|x_i)^t \\ \hat{\sigma}_m^{2\,t+1} &= \frac{1}{\sum_{i=1}^{N} \hat{P}(m|x_i)^t} \cdot \textstyle\sum_{i=1}^{N} (x_i - \mu_m^t)^2 \hat{P}(m|x_i)^t \ . \end{aligned} \tag{29}$$

- An instance of a termination condition for the iterative scheme is given above (general EM), and is also used in our implementation.

3.4 Missing Data in EM/GMM Analysis of Monophonic Music (Number of Components)

Now that we have an iterative scheme for estimating the mixture parameters, let us take a look back at the pitch estimation problem. The fundamental and the harmonics are spread

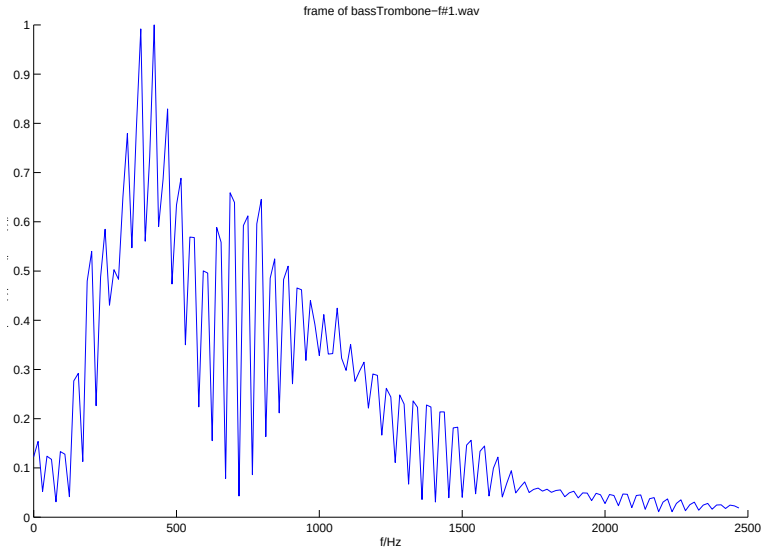

Figure 10: Many components arising from low pitch (bass trombone, F#1)

in approximately equal distance from each other across the magnitude-spectrum. A lower pitch results in these peaks being closer to each other, and higher pitches force them farther apart.

Given the limited frequency interval of typically $0..8kHz$ (for a $16kHz$ sampling frequency), we will see a different number of mixture components if we look at different pitches. Figures 10 and 11 demonstrate this. A low pitch has many components (bass trombone F#1 in Figure 10), whereas a high pitch has few components (flute B6 in Figure 11).

Naturally, if the assumed number of components M does not coincide with the actual number, our estimation is not optimal, even with the best initial parameter guess. There are a number of reasonably well-explored algorithms to determine M (see [1]), but they are not low in complexity. In testing of the EM at this single pitch stage we experimented with different choices for M and found that for many cases, $15 \leq M \leq 20$ works best (although still far from the desired quality). More discussion about this is given in Chapter 5.1.

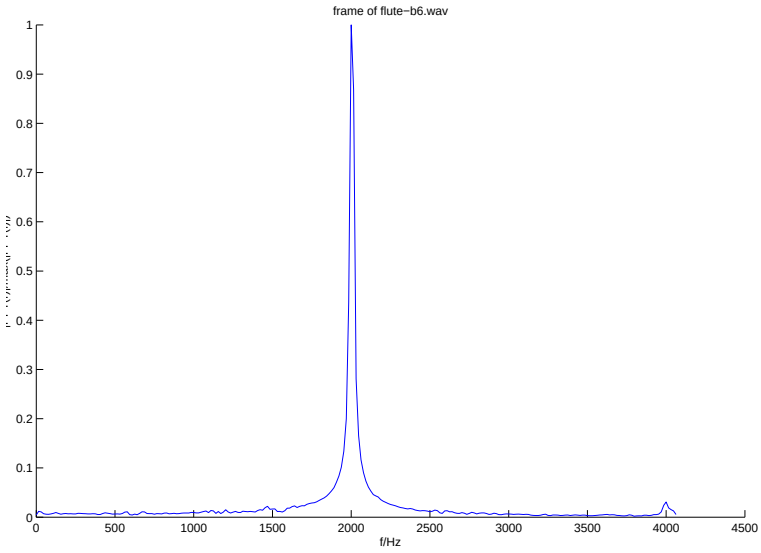

Figure 11: Only one component arising from high pitch (flute, B6)

3.5 Reinterpretation of the Magnitude Spectrum as a Spectral Density

So far, an EM algorithm has been formulated to work on monophonic cases using a GMM formulation. However, we assume that we have samples x_i from the unknown distribution. In our problem of pitch estimation, we are not supplied with these samples. The definition of the (discrete) Fourier Transform does not provide any meaning for some "sample" to form a frequency density, simply because a single sample does not carry an inherent frequency. Indeed, the DFT is deterministic in nature.

The EM presented above can be reformulated to merely work with the input of the *density* (strictly speaking, it is a *probability mass function*, or *pmf*, the discrete equivalent to the probability density) of the samples. In the implementation, this density is the magnitude-spectrum for a single frame of the sound file. Note that we are using a probabilistic artifice to solve what is basically a deterministic function fitting problem.

3.6 Modification of the EM Algorithm to Accept the "Frequency Density" whose Parameters Are to Be Estimated

Let the "frequency density" input to the EM algorithm be denoted by $\phi(\omega_i)$, where the support points ω_i are the frequencies at which the DFT for a sound frame was computed (it is mandatory that $\sum_{i=1}^{\Omega} \omega_i = 1$ in order to be a valid density).

- Expectation-step

$$\Gamma(m, \omega_k)^t = \frac{w_m^t \cdot N(\omega_i; \hat{\sigma}_m^{2\,t}, \hat{\mu}_m^t)}{\sum_{k=1}^{M} w_k^t \cdot N(\omega_i; \hat{\sigma}_k^{2\,t}, \hat{\mu}_k^t)} \ . \tag{30}$$

  This is the curve of a single Gaussian component at its current parameters, normalized over the sum over all mixtures. Comparing this to the former equation for P, this expression gives a likelihood measure for each abscissa value of the density graph, whereas before, probabilities were computed for each such value, even if it occurred multiple times. The "number of occurrences" is accounted for by the multiplication with $\phi(\omega_k)$ in the maximization step, where now real positive values can and will also be assumed.

- Maximization-step:

$$\begin{aligned}
\hat{w}_m^{t+1} &= \sum_{k=1}^{\Omega} \phi(\omega_k) \cdot \Gamma(m, \omega_k)^t \ , \\
\hat{\mu}_m^{t+1} &= \tfrac{1}{\hat{w}_m^{t+1}} \cdot \sum_{k=1}^{\Omega} \omega_k \cdot \phi(\omega_k) \cdot \Gamma(m, \omega_k)^t \ , \\
\hat{\sigma}_m^{2\,t+1} &= \tfrac{1}{\hat{w}_m^{t+1}} \cdot \sum_{k=1}^{\Omega} (\omega_k - \hat{\mu}_m^t)^2 \cdot \phi(\omega_k) \cdot \Gamma(m, \omega_k)^t \ .
\end{aligned} \tag{31}$$

This formulation enables a better understanding of the nature of this EM algorithm: multiplication of each mixture with the observed data will move it towards the next peak and eventually converge, yet depending on the initial estimates, some peaks may be "out of reach," especially if the amplitudes between the peaks are very close to zero. Figure 12 demonstrates this visually.

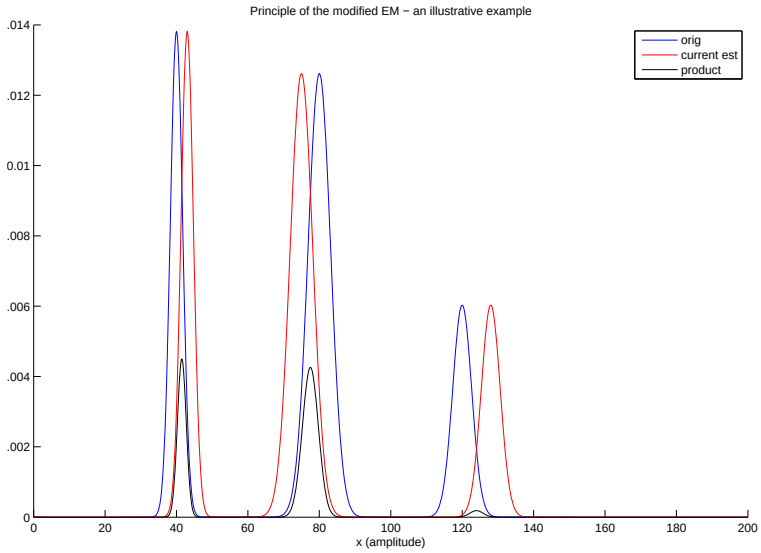

Figure 12: Illustration of the behavior of the EM algorithm for Gaussian Mixture Models

3.7 Missing Data in EM/GMM Analysis of Polyphonic Music (Number of Tones, Number of Components)

Progressing now to the polyphonic case, the formula for GMM introduced above would still be able to describe any multitone mix, yet the membership of each Gaussian component to a particular tone is not expressed. This necessitates a GMM formulation for the polyphonic case:

$$f_X(x;\Theta) = \sum_{p=1}^{P} \sum_{m=1}^{M(p)} w_m^p \cdot N(x;\Theta_m^p) \ . \tag{32}$$

The number of components M is now dependent on the pitch they belong to, and here denoted $M(p)$. P is the number of overlaid pitches. The parameters $\Theta_m^p = [\sigma_m^{2\,p}, \mu_m^p]$ are now clearly separated according to pitch membership, and the means for a particular p should be thought of as, at least ideally, equally spaced from each other in the frequency domain. The problem of finding a good guess for P will be addressed in Chapter 4.

CHAPTER IV

HEURISTICS FOR INITIAL ESTIMATES

As previously mentioned in Chapter 3.1, the performance of an EM algorithm will heavily depend on its initialization because of its nature as a steepest-descent algorithm. This chapter tackles this problem for the specific domain of pitch estimation in polyphonic music, and also features other heuristics that are necessary for the current system to work well on real data. Chapters 4.1 through 4.3 discuss heuristics for single-pitch cases, and 4.4 leaves everything untouched except for the core heuristic, which is modified to support polyphony.

4.1 Limiting the Frequency Range ("Cut-Off")

The optimization described in this section may seem insignificant, but it can have a tremendous effect on the number of iterations needed for the (unconstrained) EM algorithm to converge. The idea is that we can disregard the highest frequency portion of the magnitude spectrum of a frame. In many cases, the significant peaks are located in the lower regions; sometimes even up to $60 - 70\%$ of the spectrum does not provide much discrimination information. Hence, reducing the data becomes appealing, as illustrated by the magnitude spectrum of a real piece of music (sampled at 16kHz) shown in Figure 13.

The cutoff frequency should be determined for each frame independently. It was experimentally determined to be well chosen by

$$\omega_{cutoff} = \arg\max_{\omega_k} \left(\{ \omega_k | \phi(\omega_k) \geq 0.025 \cdot \max(\phi(\omega_k)) \} \right) \ , \tag{33}$$

in other words, the largest frequency for which the spectral density value is larger than or equal to 0.025 of the global maximum. In Figure 13, only the spectrum inside the red box will be passed on to *any* further processing. The rest is discarded since it does not carry useful information for our purposes.

Figures 14 and 15 show spectrogram results for the (unconstrained) EM algorithm but with and without the spectral cutoff heuristic (and without further heuristics). This simple

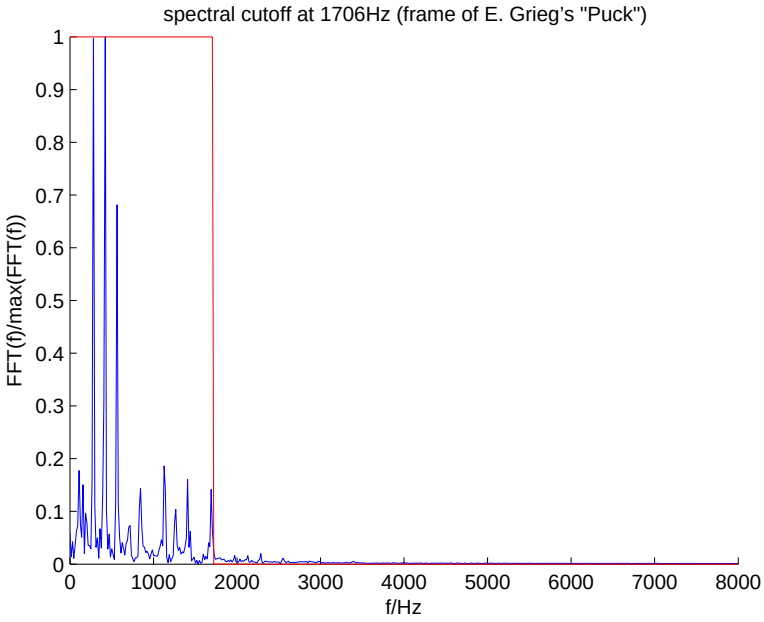

Figure 13: Reducibility of the spectrum

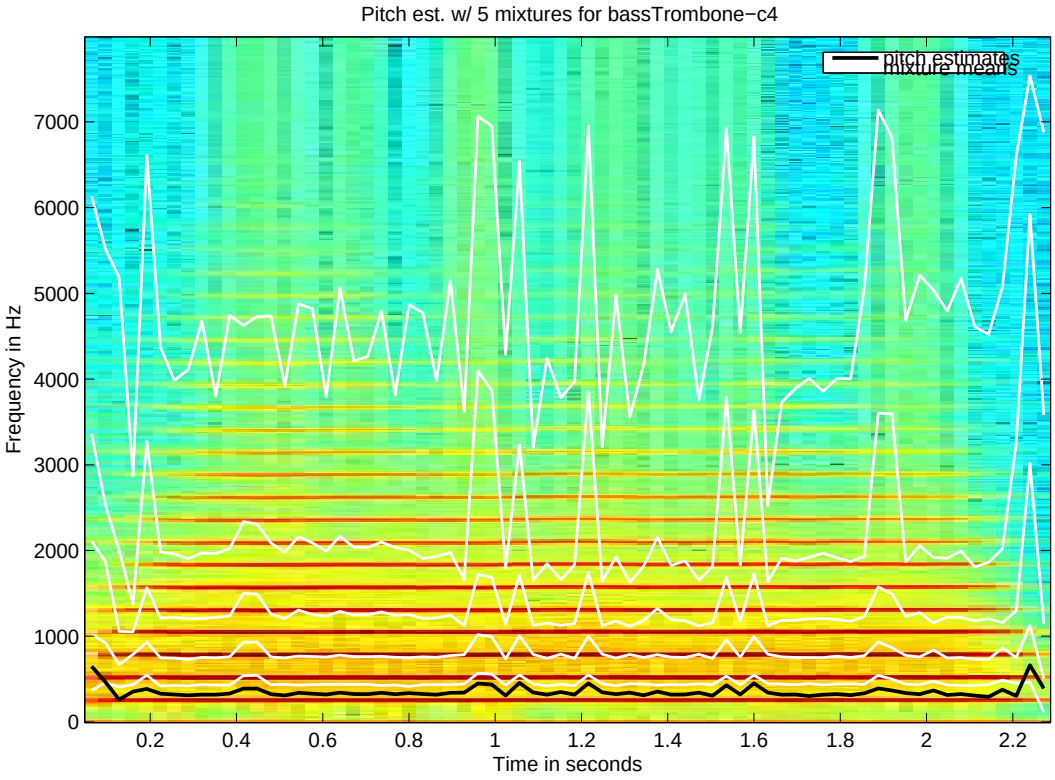

Figure 14: Spectrogram result without spectral cutoff

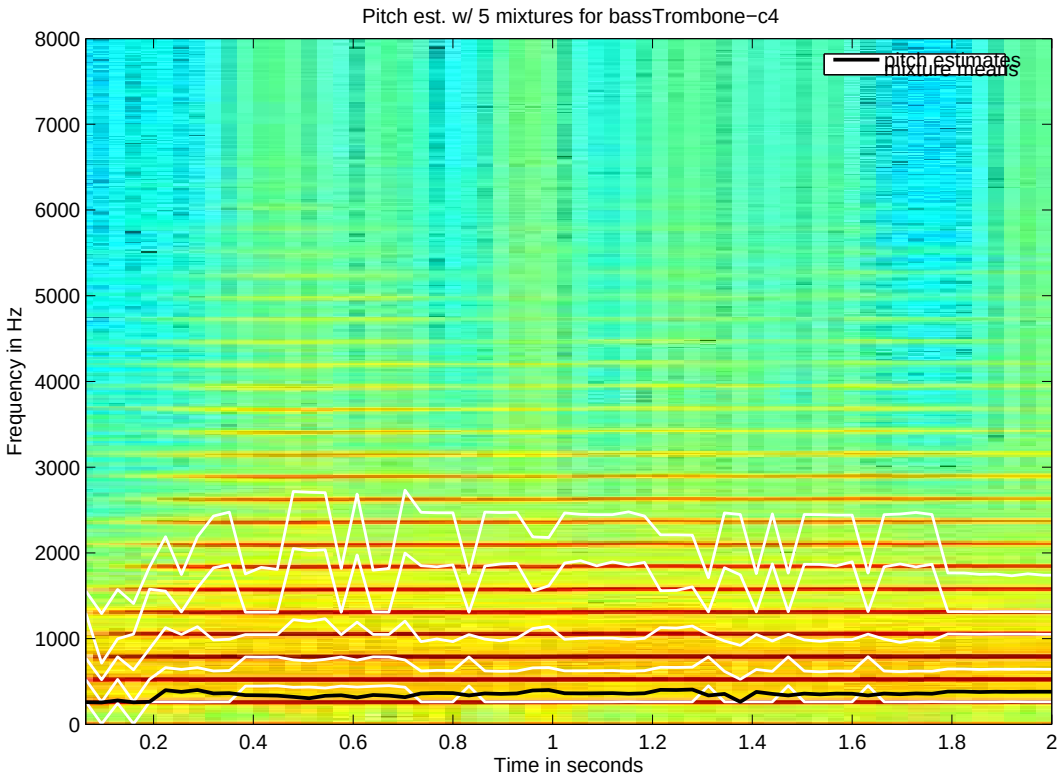

Figure 15: Spectrogram result with spectral cutoff

heuristic yields a 47% reduction of EM algorithm iterations for the given spectrogram. The saving will generally depend on musical timbres and pitches contained in the data.

4.2 Leveraging the Spacing of the Harmonics

As discussed in the introduction, we know that the locations of the peaks belonging to a particular pitch should adhere closely to frequencies that are integer multiples of the fundamental (1). Assuming we are given a recording with reasonable SNR, we can also assume that the peaks we seek generally have significantly higher amplitudes than any noise-related peaks. Supposing we can find a set of such peaks that follow these principles with some tolerance, we can use this set as a good initial estimate for the EM algorithm.

4.2.1 Peak-Picking Function

A code module that finds all local maxima of a curve had to be added (low complexity). This module finds all the local maxima, no matter where they are located, and no matter how

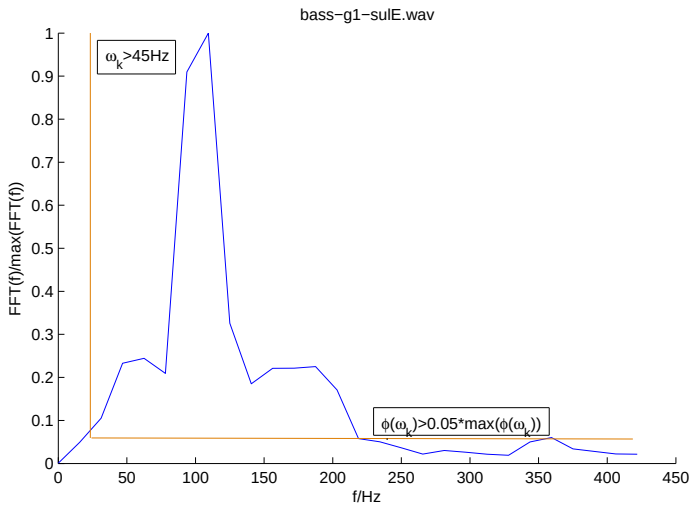

Figure 16: Summary of cutoff ranges

flat the peaks might be. Further filtering of these peaks had to be applied; the discussion follows.

4.2.2 Peak Filtering Heuristic

Examination of a lot of data has shown that there may be a low-frequency peak with a high amplitude that does not fit into the harmonic structure. Although it is not entirely clear where such peaks come from, we suspect it may be a recording issue. It is necessary to exclude them ($f_k \leq 45Hz$). Furthermore, we exclude extremely low amplitude peaks from consideration as components (those $\leq 0.05 \cdot max\left(\phi(\omega_k)\right)$). This results in the following peak exclusion window (after peak-picking):

This means that the set of peaks will be initialized according to:

$$\begin{aligned} \Lambda &:= \max_{local}(\phi(\omega_k)); \\ \Lambda &:= \Lambda \backslash \left\{\lambda_k | \lambda_k < 45Hz \vee \phi(\lambda_k) < 0.05 \cdot \max(\Lambda)\right\}; \end{aligned} \tag{34}$$

4.2.3 Core Heuristic: Basic Idea

Here, the core heuristic will be explained for the single pitch case, and in Chapter 4.4 extended to support polyphony.

Suppose the peak-picking function returns a set of frequency locations for the peaks,

denoted by $\Lambda = \{\lambda_k\}$, where $k = 1 \ldots K$, and K is the total number of peaks found, which is not predictable because of noise. The basic idea of the heuristic described in this section can be expressed in pseudocode as (let the initial guesses we are seeking be denoted by $\hat{\mu}_m^0$):

$$
\begin{aligned}
& m = 1: \ \hat{\mu}_1^0 := \lambda_1; \\
& \text{for} \ \ (m := 2; \ m := m + 1; \ m < \lfloor \tfrac{\lambda_K}{\hat{\mu}_1^0} \rfloor) \\
& \qquad \text{if} \ \exists \, \lambda_k \in \Lambda : \left| m\hat{\mu}_1^0 - \lambda_k \right| \leq \delta_f \\
& \qquad\qquad \text{then} \\
& \qquad\qquad\qquad \hat{\mu}_m^0 := \arg\min_{\lambda_k} \left| m\hat{\mu}_1^0 - \lambda_k \right|; \qquad (35) \\
& \qquad\qquad \text{else} \\
& \qquad\qquad\qquad \hat{\mu}_m^0 := \infty \\
& \qquad \text{endif} \\
& \text{endfor}
\end{aligned}
$$

In other words, we assume the peak at the lowest frequency to be the fundamental. Then we proceed to look for integer multiples of it (within some tolerance) across the entire frequency range. The tolerance with which the peaks are allowed to deviate from their ideal locations (+ or −) is defined as δ_f. This value has to be determined experimentally. We also have to consider the pitch range we want to investigate. The difficulty is that for low pitches, we would want δ_f to be low because otherwise we might consider a tolerance corridor that extends over adjacent peaks. For higher pitches, peaks are far apart, and then it would make sense to increase δ_f, to allow for a little more variation. From experience, we currently recommend using

$$\delta_f = min(50Hz, \hat{\mu}_1^0/4) \ , \qquad (36)$$

which prevents the tolerance from increasing unboundedly. This yields reasonable values for a pitch range from the second to the ninth octave (approx. $65 - 15800Hz$). The lowest pitches are hard to estimate, because the fluctuation of the peak locations over the frequency range of the many peaks is quite significant. Chapter 5 discusses this problem further.

4.2.4 Interpolation for "Missing Peaks"

Suppose we have a low-amplitude harmonic in the mid-frequency range that is so weak it would be excluded from the set of peaks Λ after the peak-filtering heuristic discussed in 4.2.2. Instead of only recognizing the first consecutive set of found harmonics, counting from lowest frequency on, and discarding the rest, it makes sense to fill any "holes" or "missing peaks" of "not found" harmonics up to the highest valid found harmonic.

For example, if we have a set of found harmonics

```
50 100 150 200 Inf 300 350 Inf Inf ,
```

where the `Inf` entries represent harmonics that are not found in the set of filtered peaks, we interpolate the `Inf` entry between 200 and 300, but the two others are deemed uncertain and/or imprecise. However, this is not done by filling in the mean of the two adjacent values, instead, the heuristic looks at the set of peaks that was returned *before* peak-filtering occurred. If there is nothing found within that frequency range, only the entries of 50 to 200 are considered to belong to this pitch (this more significance for the polyphonic case treated later). This modification is not given here analytically because it is straightforward. It can be followed in the code.

4.3 Dynamic Determination of Number of Mixtures

If the peak-picking heuristic works well, the problem of the user having to enter the correct number of mixtures is also solved. It is clear that the above formulation (Chapter 4.2.3) can terminate at any step, and hopefully, we will receive an M, equaling the number of peaks found, that will be equal to the true number of components for that pitch.

4.4 Extension to Polyphonic Cases

4.4.1 Maximum Number of Polyphonic Tones

In the implementation of the GUI, a field was added that lets the user specify the maximum number of simultaneous tones played in the data (the user/developer should ideally know this for testing purposes). This value serves as upper termination condition for the polyphonic-capable heuristic developed next.

4.4.2 Iterative Scheme

Suppose we are given Λ, the set of peaks, as outlined earlier in this chapter. For notational simplicity, we drop the superscript zero indicating the initial estimate, since this should be understood by now. We define $\hat{\mu}_m^p$ as the initial estimate of the m^{th} mixture mean of the p^{th} pitch, and P as the user-defined maximum of simultaneous pitches available in the signal. The extension to the polyphonic case is realized with this iterative scheme:

$$
\begin{array}{l}
p := 1; \\
\text{while } \ (p < P \ \& \ |\Lambda| = K > 0) \\
\quad m = 1: \ \hat{\mu}_1^p := \lambda_1; \\
\quad \text{for } \ (m := 2; \ m := m + 1; \ m < \lfloor \frac{\lambda_K}{\hat{\mu}_1^p} \rfloor) \\
\quad\quad \text{if } \ \exists \lambda_k \in \Lambda : |m\hat{\mu}_1^p - \lambda_k| \leq \delta_f \\
\quad\quad\quad \text{then} \\
\quad\quad\quad\quad \hat{\mu}_m^p := \arg\min_{\lambda_k} |m\hat{\mu}_1^p - \lambda_k| \, ; \\
\quad\quad\quad\quad \Lambda := \Lambda \setminus \hat{\mu}_m^p; \\
\quad\quad\quad \text{else} \\
\quad\quad\quad\quad \hat{\mu}_m^p := \infty \\
\quad\quad \text{endif} \\
\quad \text{endfor} \\
\quad p := p + 1; \\
\text{endwhile}
\end{array}
\tag{37}
$$

It is thus possible for the algorithm to terminate before $p = P$ has been reached, and this will be the case if all significant peaks have been used up in prior set(s) of peaks. This is useful because P is entered as a global parameter for a piece of music, in which the number of simultaneous tones can vary, and the above iterative scheme merely works on a single frame of typically $64ms$.

All non-initialized means for a certain value of p and its successors can be subsequently regarded as a pitch (or pitches) which is (or are) not available at this time frame. Consequently, no EM algorithm will be executed for those means.

4.4.3 Constrained EM Algorithm

In this subsection, the EM algorithm working on GMM will be modified in such a manner as to only allow component means that adhere to a harmonic structure. The motivation for this is that we expect both the initial estimates by the heuristic and the final estimates of the EM algorithm to be in that range. Constraining the EM algorithm also saves us precious computation time, since the constraint should cut down on the number of iterations necessary for convergence.

Let the initial estimates, the outputs of the heuristic, be denoted by $\hat{\mu}_m^p(0)$ and $\hat{\sigma}_m^p(0)$. These are the estimates of the means and variances for each pitch $p = 1..P$ and each component $m = 1..M(p)$ as previously defined (see Chapter 3.7 on polyphonic GMM), at $t = 0$ (init). While there is only a loosely enforced harmonic spacing condition in these initial estimates, the subsequent EM algorithm iterations will place strict requirements on the component means.

- Auxiliary EM Initialization: $\hat{\mu}_m^p(0) = m \cdot \hat{\mu}_1^p(0)$, or any other pitch computation scheme as discussed in Section 5.1 (here, we use the "fundamental only" scheme)

- Expectation-step:

$$\Gamma(p, m, \omega_k)^t = \frac{w_m^p(t) \cdot N(\omega_i; \hat{\sigma}_m^{2\,p}(t), \hat{\mu}_m^p(t))}{\sum_{k=1}^{M} w_k^p(t) \cdot N(\omega_i; \hat{\sigma}_k^{2\,p}(t), \hat{\mu}_k^p(t))} \ . \tag{38}$$

- Maximization-step:

$$\begin{aligned} \hat{w}_m^p(t+1) &= \sum_{k=1}^{\Omega} \phi(\omega_k) \cdot \Gamma(m, \omega_k)^t \ , \\ \hat{\mu}_m^p(t+1) &= \tfrac{1}{\hat{w}_m^p(t+1)} \cdot \sum_{k=1}^{\Omega} \omega_k \cdot \phi(\omega_k) \cdot \Gamma(p, m, \omega_k)^t \ , \\ \hat{\sigma}_m^{2\,p}(t+1) &= \tfrac{1}{\hat{w}_m^p(t+1)} \cdot \sum_{k=1}^{\Omega} (\omega_k - \hat{\mu}_m^p(t))^2 \cdot \phi(\omega_k) \cdot \Gamma(p, m, \omega_k)^t \ . \end{aligned} \tag{39}$$

- Enforcing strict harmonic spacing for next iteration:

$$\begin{aligned} \Delta_f(t+1) &= \tfrac{1}{M(p)} \cdot \sum_{m=1}^{M(p)} \tfrac{1}{m} \left(\hat{\mu}_m^p(t+1) - \hat{\mu}_m^p(t)\right) \ , \\ \hat{\mu}_m^p(t+1) &= m \cdot (\hat{\mu}_1^p(t) + \Delta_f(t+1)) \ . \end{aligned} \tag{40}$$

  This deserves some explanation. In the previous two steps, the same unconstrained EM formulation given before has been used. The parameters can flow freely. After

that, we look at how the means changed in comparison to the previous means. The formula for Δ_f is the mean of these changes. It will be positive if the sum of all components moving up in frequency is larger than those moving downward, thus following the overall dominant trend. This will be accounted for in a correction to the component means, which are in this step strictly located according to the harmonics condition.

- Modified termination condition. Iterate until

$$\begin{aligned} &|\Delta_f| \leq 0.01 \text{ or} \\ &|\Delta_f(t) + \Delta_f(t+1)| = 0 \text{ or} \\ &\hat{\mu}_m^p(t+1) \geq \hat{\mu}_m^p(0) \cdot 2^{1/24} \text{ or} \\ &\hat{\mu}_m^p(t+1) \leq \hat{\mu}_m^p(0) \cdot 2^{-1/24} \ . \end{aligned} \tag{41}$$

  The first line is defined in analogy with the previously used termination condition. The second line is necessary because, in some cases, subsequent Δ_f have the same magnitude but alternate in sign, and this is regardless of the first condition. Conditions three and four are simple sanity-checks for the moving of the first mixtures – we do not want them to move more than half a semitone (see the introduction for details).

4.5 Chapter Review

This chapter discussed how initial estimates for polyphonic music can be found using straightforward heuristics, and how the EM algorithm can be constrained and terminated using knowledge of the application. At this stage, the EM algorithm is executed serially for each harmonic structure reflecting one pitch, assuming at each time that it were the only pitch present. Some problems with this assumption will be discussed in Chapter 5.2.

CHAPTER V

PRELIMINARY RESULTS AND ANALYSIS

5.1 Implementation Remarks: Single-Pitch Prototype

Figure 17 shows a screenshot of a MATLAB-based visualizer that plots the spectrogram and computes pitch estimates dependent on some user input parameters. The input parameters include:

- Frame length (defaulting $64ms$)
- Number of components (typically 15 to 20), constant over all frames
- Method selector: how to determine pitch value from the sets of mixture mean estimates

The last bullet has not been discussed yet; and several different approaches may come to mind. We implemented the following:

- Fundamental only: $p = \hat{\mu}_1^T$ (T is not a matrix transpose, it denotes the final value of the estimation process)
- Second harmonic minus fundamental: $p = \hat{\mu}_2^T - \hat{\mu}_1^T$ (if $M > 1$)
- Mean: $p = \frac{1}{M} \sum_{m=1}^{M} \frac{\hat{\mu}_m^T}{m}$ (if $M > 1$)
- "Sum and Divide": $p = \frac{1}{M(M+1)/2} \sum_{m=1}^{M} \mu_m^T$ (if $M > 1$)

The idea for the latter is that if, in the ideal case, $\hat{\mu}_m^T = m\hat{\mu}_1^T$ for all m, then

$$\sum_{m=1}^{M} \hat{\mu}_m^T = \sum_{m=1}^{M} m\hat{\mu}_1^T = \hat{\mu}_1^T \sum_{m=1}^{M} m = \hat{\mu}_1^T \frac{M(M+1)}{2} \ , \tag{42}$$

where $\hat{\mu}_1^T$ is the pitch coinciding with the fundamental that we would like to discover. Other parameters include:

- Frame overlap set to 50%

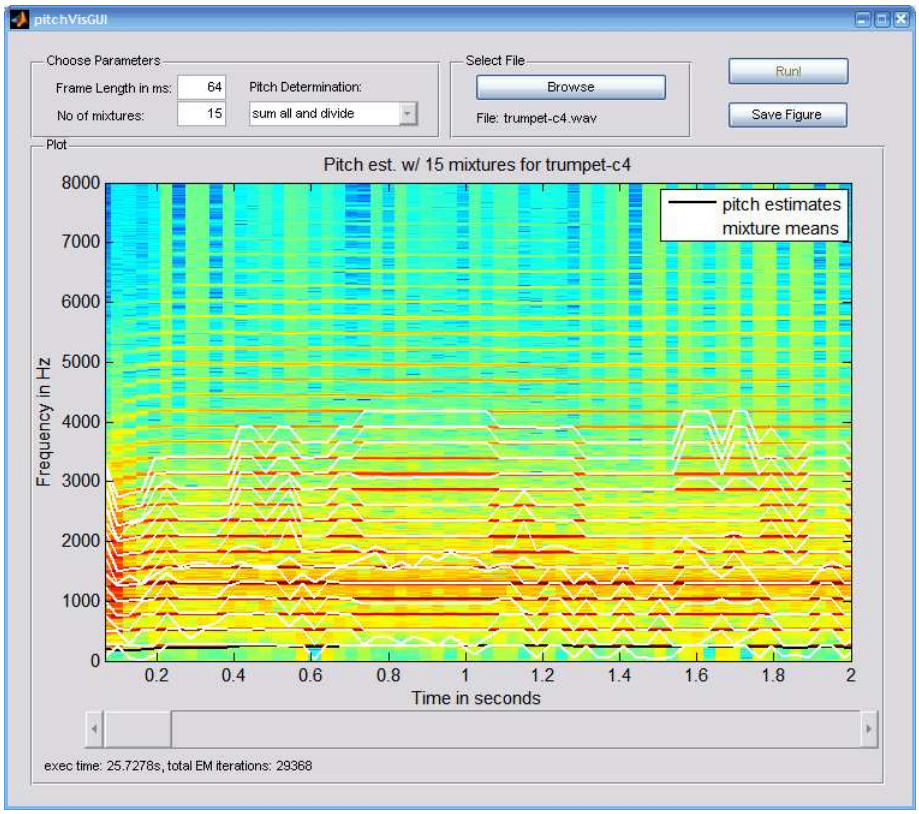

Figure 17: GUI screenshot of single-pitch prototype

- Hamming-windowing of the time frames
- Sample rate usually $16kHz$ (but works with all of course)

The EM algorithm is initialized by the following procedure:

- Spread mixture means (μ's) equally across the entire frequency range
- Set all variances equal to a fraction of the global variance
- Initialize all mixture weights with $1/M$ (note the weights sum to unity)

This GUI system running in MATLAB was used in all the later parts of the pitch estimation project for visual verification purposes. During the course of the research, it underwent some changes that will be discussed later.

The graph in Figure 17 should be interpreted as follows. The spectrogram shows the development of the signal's spectral magnitude over time (the time axis is scrollable for

better resolution of long files). The harmonics are apparent in the graph as thin red horizontal lines, with red indicating larger amplitude. These are the peaks of each frame. Blue indicates low amplitude. Due to the $1/f$ falloff typical of acoustic instruments, higher frequency regions tend towards blue.

The white lines show where the EM estimated the harmonics to be. Ideally, we would want them to lay exactly on top of the red lines of the data. Finally, the only black line in the graph shows the actual pitch estimate. It should ideally be close (but not necessarily identical) to the fundamental, which is the lowest of the red lines.

5.1.1 Results for Single Pitch Cases

We tested with labeled data [19], so we know what the true numerical pitch values of the recorded notes should be. We discovered that for many of the samples, the pitch was not identical to the scientific pitch notation [33]. We verified the operation of the EM algorithm by looking at how well the means matched the peaks of the given data. After all, if we have an A4 recording and the first peak is at $449Hz$ and the second at $893Hz$, instead of $440Hz$ and $880Hz$, for instance, it does not make sense to try to adapt an implementation to the latter. Our observations do not mean that the basic truths (labeling) are wrong[1], just that the perceived pitch is a bit more involved, as laid out earlier in the introduction. Hence, we currently have no sophisticated numerical statistics summarizing our tests.

5.1.2 Potential Difficulties in Adopting GMM For Pitch Estimation

- The Gaussian function will inevitably return numerical values so small that they are considered zero. This happens in frames whose peaks are widely spread across the whole spectral range. The normalization of the E-step would divide by zero, which clearly has to be prevented. This might seem unlikely to happen, but we found it occurred quite frequently.

- Peaks are sometimes extremely narrow, so variances that become numerically close to zero are possible. Additional precautions have to be taken when implementing to

[1] Unfortunately, the authors of [19] did not return feedback concerning the question of how tuning was arranged and which pitch standard was used.

prevent this from causing trouble. In particular, the Gaussian function, which takes in the frequency range, as well as the parameter estimates of mean and variance, has to be modified also to work with zero variance inputs if we do not prohibit this by defaulting to a minimum nonzero variance. In such cases, we returned a vector that evaluates to zero everywhere but at the specified mean, at which some arbitrary large value is chosen (we chose 1000).

- Some noise peaks may confuse the EM algorithm. Smoothing may resolve this issue and make the peaks more similar to Gaussians. We tried a little bit of running-average smoothing with a window of two and experimented with other values as well. If we smooth using this technique, the window should not be chosen large, because this could merge larger peaks into one. This is generally not desired, unless the heuristics explicitly account for such cases.

5.2 Discussion of the Final Implementation

The main limitations of the system (see Figure 18 for a screenshot) with all the described heuristics (GUI screenshot in Figure) for polyphonic operation is that it is dependent on a user-specified maximum number of tones P. If this maximum is correct, we enjoy relatively robust number of tone detection. However, for certain cases, the algorithm will find excess pitches not present in the data if P is chosen too high.

Figure 19 shows pitch estimation results superimposed over the spectrogram of a high-pitched triad (C diminished in octave 6, ideally having pitches of 1047, 1319, 1480Hz). The results are quite good. We can see that at about $t = 2s$, the C6 pitch fades out, one of the pitch lines discontinues (blue), and the two remaining lines take over the remaining two pitches. Also, the precision of the pitch estimates seems to visually follow the spectrogram lines of the fundamentals. Note that the rapid decline of the last remaining pitch at the end is due to offset artifacts.

The verbose MATLAB console printout for the experiment shown in Figure 19 looks like:

```
runEM: frame 22 of 73
```

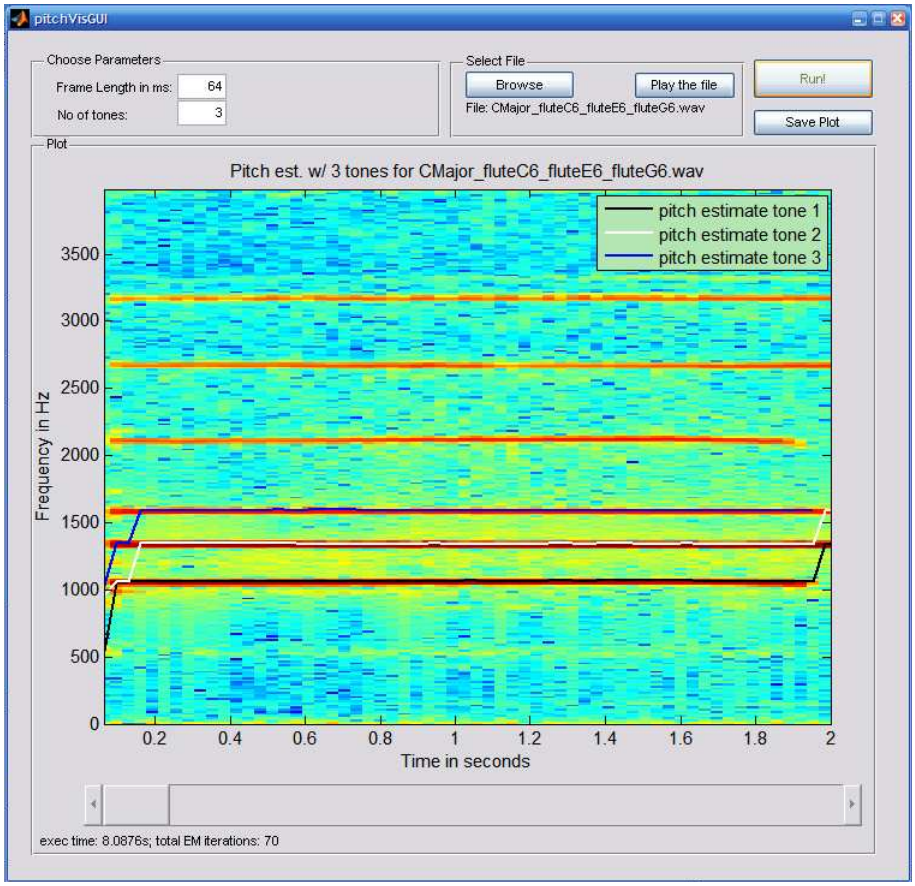

Figure 18: GUI screenshot of the final implementation

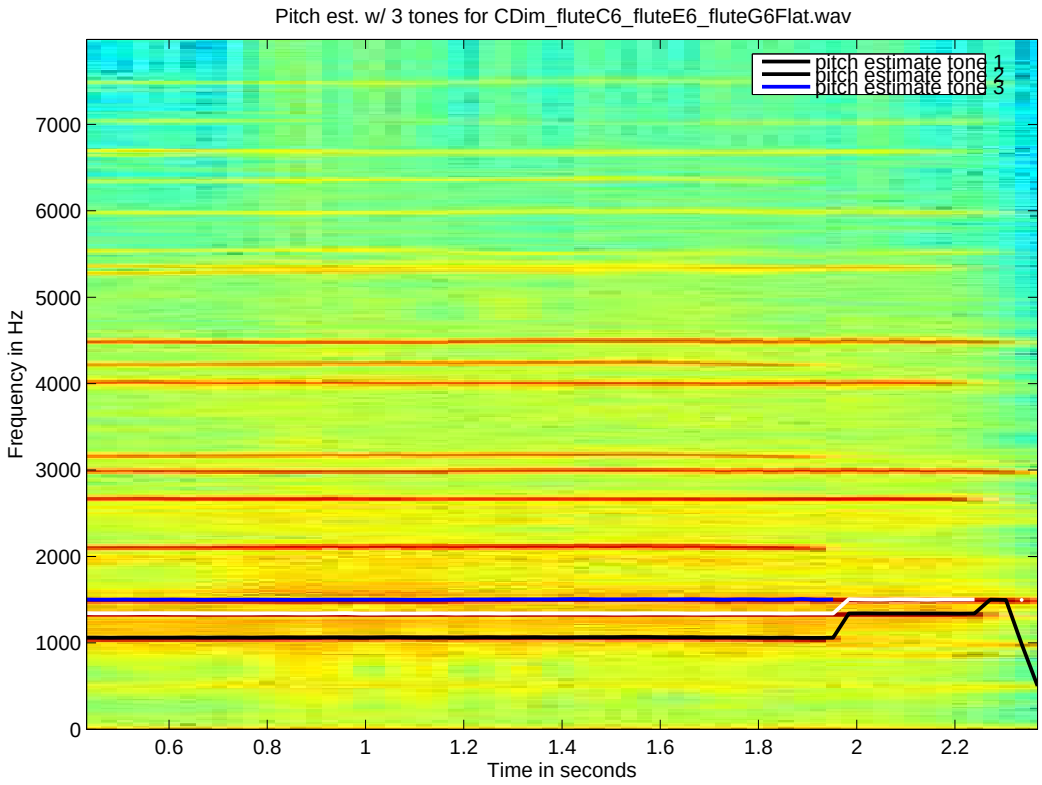

Figure 19: Heuristic working well in a high-pitch example

```
myEM (tone 1): init(heur.) means: 1062.5 2125
myEM (tone 1): init(pitch) means: 1062.5 2125
myEM (tone 1): after EM: 1063.8381 2127.6762
myEM (tone 2): init(heur.) means: 1343.75 2687.5
myEM (tone 2): init(pitch) means: 1343.75 2687.5
myEM (tone 2): after EM: 1341.3972 2682.7945
myEM (tone 3): init(heur.) means: 1500 3000
myEM (tone 3): init(pitch) means: 1500 3000
myEM (tone 3): after EM: 1500.8629 3001.7259
```

In other octaves, more problems occur. In Figure 20, an undesired effect occurs at $t = 1.8s$, where the estimated number of pitches does not change, but in reality, a tone fades out at this point. The blue (top) line is not desired here. There are several reasons for this. The fundamentals of the tones E4 and G4 lie close together, and in this case appear

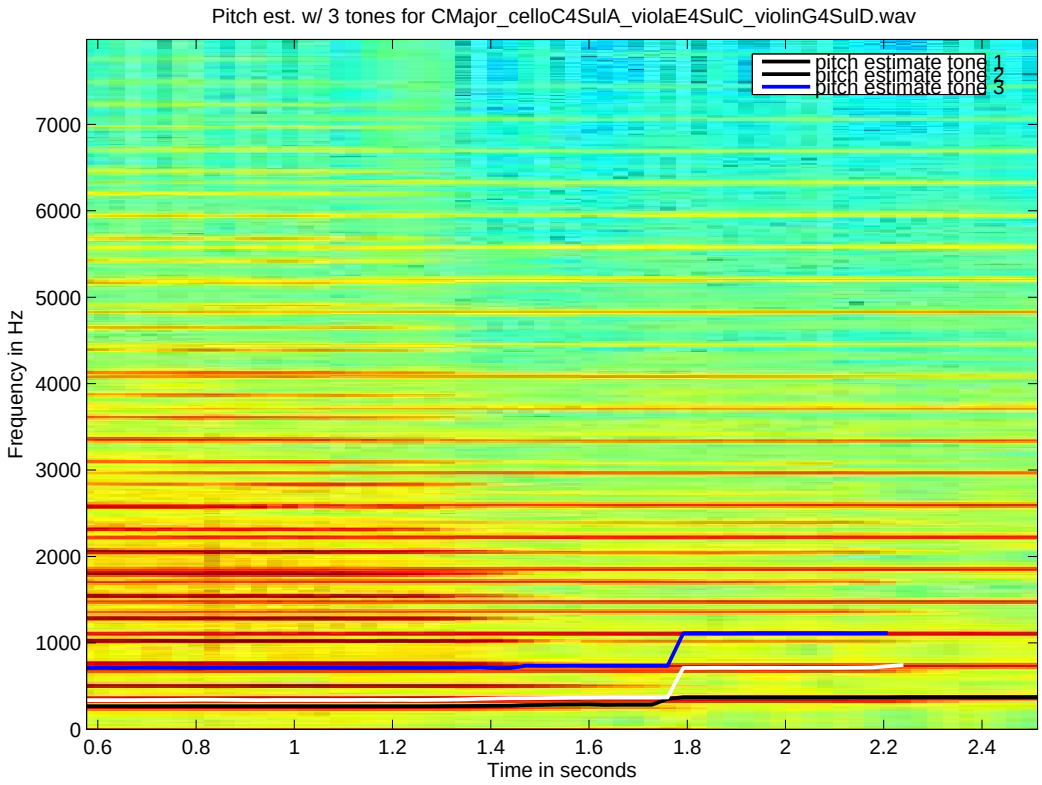

Figure 20: Excess pitches in medium-pitch example

to be wider than usual. Furthermore, there is some apparent imprecision in the recordings (pitches should be more distant from each other, E4=329.6Hz, G4=392Hz), and the moving average process also contributes to the merging of the two fundamentals into one. Thus, the peak-picking heuristic will come up with only one value here. The following heuristic picks one fundamental and then looks for integer multiples of it, however, without assuming that certain peaks can be "shared." It then follows that there will be a higher spurious peak detected within the range of the second harmonics, which of course separate out better. In reality this is not an actual new fundamental of another pitch, but just the harmonic of one of the pitches that unfortunately merged together as just described.

Figure 21 shows the behavior of the algorithm when the spectral magnitude is not smoothed prior to executing the heuristics and the EM algorithm. It becomes clear that the E4 and G4 fundamentals now separate better, considering all the places where the blue and white pitch lines lie close together, as they should.

Another effect that potentially occurs when two harmonics of two different pitches lie

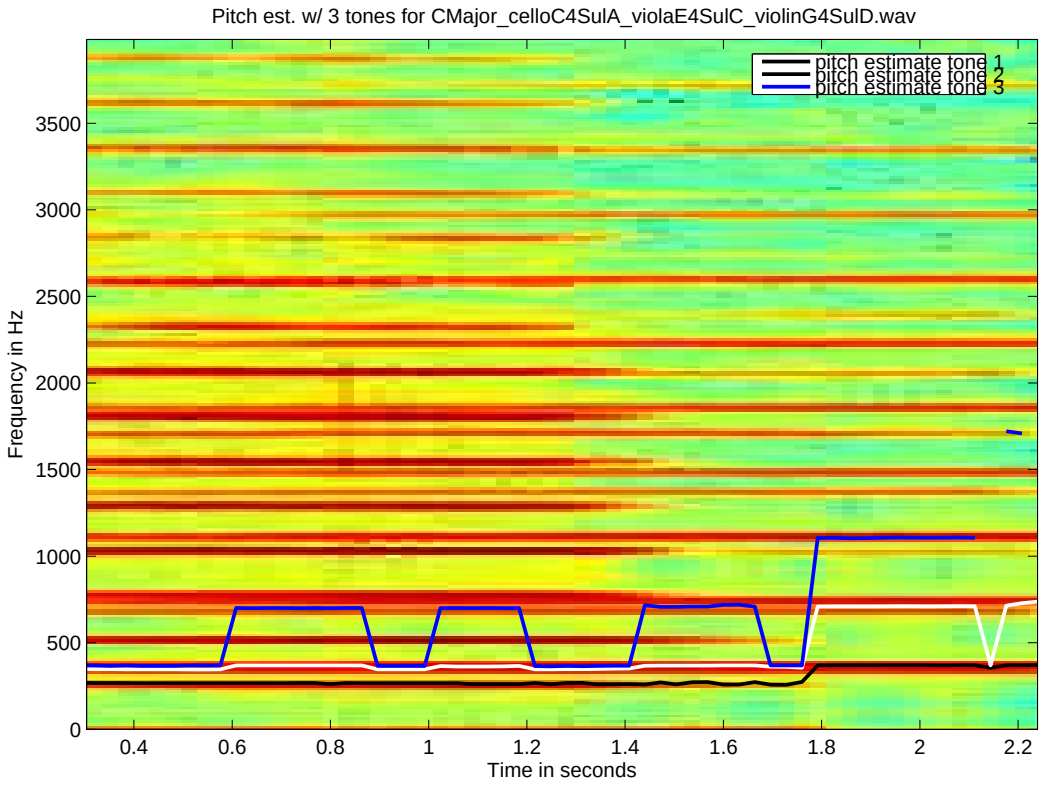

Figure 21: Excess pitches in a medium-pitch example - without spectral smoothing

close together is that the EM algorithm, in its current implemented form of being serially executed for each pitch, will tend to try combining the two harmonics into a larger one, thus shifting the initial estimate towards the middle of the two adjacent peaks. Precautions have been taken, however, to limit this behavior (namely small component variance and a maximum deviation from the initial pitch estimate as one of the termination conditions). This effect may be leveled out by the opposite tendencies of higher harmonics. In our tests, we generally did not find this issue to cause much concern.

The third issue we mention is that despite allowing for some variation from the ideal harmonic spacing, the highest harmonics of a complex tone (i.e. one with a high number of harmonics) may escape the heuristic and thus lead to spurious high pitch estimates that do not exist in the data. This is demonstrated in Figure 22.

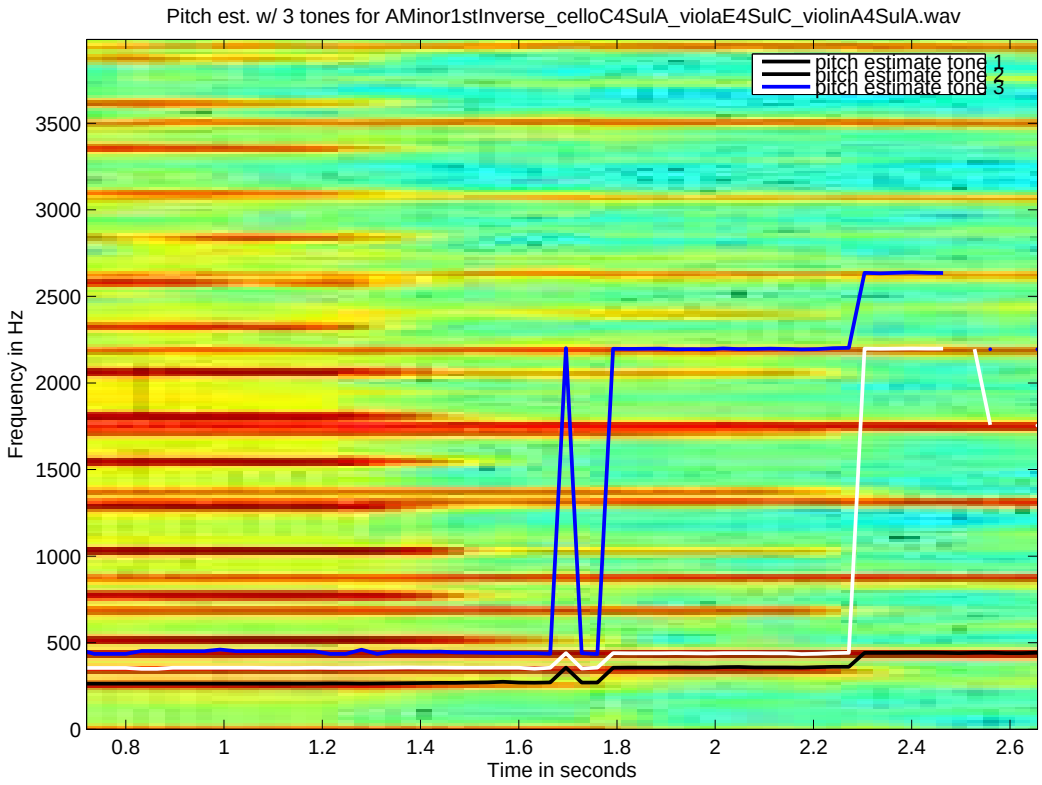

Figure 22: Excess pitches in medium-pitch example - escaping the heuristic

5.2.1 Summary of Issues

Clearly, this approach is highly dependent on the quality of the heuristics. The EM algorithm is a great mathematical tool for making precise estimates of the needed parameters, but its nature as a steepest descent algorithm demands that great effort go into finding good initialization. Our heuristic techniques are not claimed to be highly sophisticated, but they should encourage further investigations into this approach. In particular, the assumption that one peak always reflects only a single harmonic is too simplistic and quickly leads to problems when the degree of polyphony increases.

Some concepts for that may be useful in this application can be found in [34].

The main issues found mey be summarized as follows:

- We need to set a small constant variance for the Gaussian mixtures, so that the EM algorithm will not gravitate too much toward an adjacent peak that belongs to a different pitch.

- Limited peak resolution, the smearing of two peaks close to each other but belonging to different pitches, is aggravated by spectral smoothing.
- The number of pitches is not estimated very reliably in all octaves.
- This approach is good if peaks have some minimum frequency distance between each other.
- If there are many harmonic belonging to a pitch, the highest harmonics seem prone to escape the heuristic, and spurious high pitches are detected.

CHAPTER VI

CONCLUSIONS AND DIRECTIONS FOR FUTURE RESEARCH

6.1 Conclusions

The final system we developed allows for good estimates of pitches in the middle to high octaves. It uses a low number of EM steps, especially compared to our earlier unconstrained version without the heuristics. The total number of iterations (for a piece of music of length $2-3s$) has been reduced from the order of $10,000$ to the order of 100, while also usually coming up with better estimates. The implementation requires entering the maximum number of pitches (P) played simultaneously, which is a much more practical parameter than the number of components found in an early version. Many instruments, such as winds and bowed strings (excluding special cases), are only capable of producing a single pitch at a time, so for some ensembles in which the instrumentation should be known, this could already be a usable application. For other instruments such as the piano and the harp, P might not be obvious at all.

6.2 Directions for Future Work

6.2.1 Account for Harmonic Overlapping of Pitches

The possibility of one peak being "shared" by two different harmonic structures should be incorporated into our heuristics. Due to the harmonic composition principles of western music, the "sharing" of harmonics is quite likely to happen.

6.2.2 Follow Klapuri Closely

The paper by Klapuri [17] proposes that after a decision about the membership of a set of mixtures to a specific pitch has been made, the spectrum should be recomputed. This computation simply takes the previous spectrum and subtracts the set of estimated mixtures

belonging to the current pitch. The next steps would then involve re-picking the peaks, running the heuristic, and subtracting again, until the maximum number of pitches is reached.

Klapuri's proposal makes sense, because neighboring peaks have a mutual influence on their weights: the closer they are together, the higher they will appear, when in reality, they are lower. This is not important for pitch estimation itself because we are only interested in the mixture means, but suppose we were interested in identifying what instruments played the pitches; then, the weights are the crucial parameters (and to some minor extent, maybe also the variances). Some problems discussed in the previous chapter could also be resolved.

6.2.3 Confidence Score for Missing Harmonic Interpolation

The decision of what to do with "missing harmonics" has to consider two notions:

- The harmonics found after the gap could also be the first harmonics of other pitches,
- The pitch's harmonics may actually have ended at the missing harmonic (such timbres exist; we only need to mix the "right" instruments)

To better account for this, some kind of "confidence score" could allow us to decide whether or not to interpolate the missing harmonic.

6.2.4 Tune Some of the EM Numerical Parameters such as Termination

More testing with the termination condition described in Section 4.4.3 and possibly other termination conditions, should be done.

6.2.5 Set Up an Instrument Identifier on the Separated Sources

If we receive a good separation of pitches, we can use our work to identify different instruments. Some further literature review would need to be done prior to this.

APPENDIX A

PITCH TABLE

Table of note frequencies - Key: Frequency in hertz (semitones from middle C										
Octave Note ↓	0	1	2	3	4	5	6	7	8	9
C	16.35 (-48)	32.70 (-36)	65.41 (-24)	130.8 (-12)	**261.6 (0)**	523.3 (+12)	1047 (+24)	2093 (+36)	4186 (+48)	8372 (+60)
C#	17.32 (-47)	34.65 (-35)	69.30 (-23)	138.6 (-11)	277.2 (+1)	554.4 (+13)	1109 (+25)	2217 (+37)	4435 (+49)	8870 (+61)
D	18.35 (-46)	36.71 (-34)	73.42 (-22)	146.8 (-10)	293.7 (+2)	587.3 (+14)	1175 (+26)	2349 (+38)	4699 (+50)	9397 (+62)
D#	19.45 (-45)	38.89 (-33)	77.78 (-21)	155.6 (-9)	311.1 (+3)	622.3 (+15)	1245 (+27)	2489 (+39)	4978 (+51)	9956 (+63)
E	20.60 (-44)	41.20 (-32)	82.41 (-20)	164.8 (-8)	329.6 (+4)	659.3 (+16)	1319 (+28)	2637 (+40)	5274 (+52)	10548 (+64)
F	21.83 (-43)	43.65 (-31)	87.31 (-19)	174.6 (-7)	349.2 (+5)	698.5 (+17)	1397 (+29)	2794 (+41)	5588 (+53)	11175 (+65)
F#	23.12 (-42)	46.25 (-30)	92.50 (-18)	185.0 (-6)	370.0 (+6)	740.0 (+18)	1480 (+30)	2960 (+42)	5920 (+54)	11840 (+66)
G	24.50 (-41)	49.00 (-29)	98.00 (-17)	196.0 (-5)	392.0 (+7)	784.0 (+19)	1568 (+31)	3136 (+43)	6272 (+55)	12544 (+67)
G#	25.96 (-40)	51.91 (-28)	103.80 (-16)	207.7 (-4)	415.3 (+8)	830.6 (+20)	1661 (+32)	3322 (+44)	6645 (+56)	13290 (+68)
A	27.50 (-39)	55.00 (-27)	110.00 (-15)	220.0 (-3)	**440.0 (+9)**	880.0 (+21)	1760 (+33)	3520 (+45)	7040 (+57)	14080 (+69)
A#	29.14 (-38)	58.27 (-26)	116.50 (-14)	233.1 (-2)	466.2 (+10)	932.3 (+22)	1865 (+34)	3729 (+46)	7459 (+58)	14917 (+70)
B	30.87 (-37)	61.74 (-25)	123.50 (-13)	246.9 (-1)	493.9 (-1)	987.8 (+23)	1976 (+35)	3951 (+47)	7902 (+59)	15804 (+71)

APPENDIX B

CODE LISTINGS

B.1 GUI Code

```
1  function varargout= pitchVisGUI(varargin)
2  % varargout= pitchVisGUI(varargin)
3  % GUI for pitch estimation and replay of sound files
4  % Author: Dominik Loeffler. (c) 2005/2006
5  gui_Singleton= 1;
6  gui_State = struct('gui_Name',       mfilename, ...
7                     'gui_Singleton',  gui_Singleton, ...
8                     'gui_OpeningFcn', @pitchVisGUI_OpeningFcn, ...
9                     'gui_OutputFcn',  @pitchVisGUI_OutputFcn, ...
10                    'gui_LayoutFcn',  [] , ...
11                    'gui_Callback',   []);
12 if nargin && ischar(varargin{1})
13     gui_State.gui_Callback = str2func(varargin{1});
14 end
15 if nargout
16     [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
17 else
18     gui_mainfcn(gui_State, varargin{:});
19 end
20
21 % --- Executes just before pitchVisGUI is made visible.
```

```
22 function pitchVisGUI_OpeningFcn(hObject, eventdata, handles,
       varargin)
23 handles.pitchPlotHandle= [];
24 handles.sliderHandle= [];
25 handles.EMOut= [];
26 handles.fileLocCurrentPlot= '';
27 % Update handles structure
28 handles.output = hObject;
29 set(gcf, 'DefaultAxesColorOrder', [0 0 0; 1 1 1; 0 0 1; 0 1 0; 1 0
       0]);
30 guidata(hObject, handles);
31 % UIWAIT makes pitchVisGUI wait for user response (see UIRESUME)
32 % uiwait(handles.figure1);
33
34 %--- Outputs from this function are returned to the command line.
35 function varargout = pitchVisGUI_OutputFcn(hObject, eventdata,
       handles)
36 % Get default command line output from handles structure
37 varargout{1} = handles.output;
38
39 function frameLengthField_Callback(hObject, eventdata, handles)
40 % Hints: get(hObject,'String') returns contents of frameLengthField
        as text
41 %        str2double(get(hObject,'String')) returns contents of
       frameLengthField as a double
42 % frameLength is expected to come in in milliseconds
43 set(hObject, 'UserData', 10^(-3)*str2double(get(hObject,'String')))
       ;
44 guidata(hObject, handles);
```

```
45
46 % --- Executes during object creation, after setting all properties
       .
47 function frameLengthField_CreateFcn(hObject, eventdata, handles)
48 % Hint: edit controls usually have a white background on Windows.
49 %       See ISPC and COMPUTER.
50 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
       defaultUicontrolBackgroundColor'))
51     set(hObject,'BackgroundColor','white');
52 end
53
54 function noTonesField_Callback(hObject, eventdata, handles)
55 % Hints: get(hObject,'String') returns contents of noTonesField as
       text
56 %        str2double(get(hObject,'String')) returns contents of
       noTonesField as a double
57 set(hObject, 'UserData', str2double(get(hObject,'String')));
58 guidata(hObject, handles);
59
60 % --- Executes during object creation, after setting all properties
       .
61 function noTonesField_CreateFcn(hObject, eventdata, handles)
62 % Hint: edit controls usually have a white background on Windows.
63 %       See ISPC and COMPUTER.
64 if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'
       defaultUicontrolBackgroundColor'))
65     set(hObject,'BackgroundColor','white');
66 end
67
```

```
68 % --- Executes on button press in browseButton.
69 function browseButton_Callback(hObject, eventdata, handles)
70 [fileName,filePath] = uigetfile('*.wav','Select the wave-file to be
       analyzed');
71 if isstr(fileName) % user made a valid selection
72     s= [filePath fileName]; % the current selection
73     if ~strcmp(get(hObject, 'UserData'), s) % inequality with
           previous selection
74         handles.meansMatrix= [];
75         handles.timeAxis= [];
76     end
77     set(handles.runButton, 'Enable', 'On');
78     set(handles.playButton, 'Enable', 'On');
79     set(handles.FileNameDisplayText, 'String', fileName);
80 else % this happens if dialog gets cancelled
81     s= 'none';
82     set(handles.runButton, 'Enable', 'Off');
83     set(handles.playButton, 'Enable', 'Off');
84     set(handles.FileNameDisplayText, 'String', s);
85 end
86 set(hObject, 'UserData', s);
87 guidata(hObject, handles);
88
89 % --- Executes on button press in saveFigureButton.
90 function saveFigureButton_Callback(hObject, eventdata, handles)
91 [fileName, pathName] = uiputfile('*.*', 'Select file to write,
       include extension');
92 if ischar(fileName)
93     plotCp= findobj(gcf, 'Type', 'axes'); % 1) legend 2) axes
```

```
94      p= get(plotCp(2), 'Position');
95      shift= [-.00 -.1 0.02 0.09]; % if you enlarge the copy in the
            new window,
96      p= p+shift; % matlab will erase everything (bug?)
97      set(plotCp(2), 'Position', p);
98      newFig= figure('Visible', 'off');
99      newAxes= copyobj(plotCp, newFig);
100     set(plotCp(2), 'Position', p-shift);
101     saveas(newFig, [pathName fileName]); % only for whole figures!
102     delete(newFig);
103 end
104
105 function runButton_Callback(hObject, eventdata, handles)
106 fileLocation= get(handles.browseButton, 'UserData'); %get: [path
        filename]
107 tonesNo= get(handles.noTonesField, 'UserData');
108 windowLength= get(handles.frameLengthField, 'UserData');
109 tic;
110 clc;
111 handles.EMOut= runEM(fileLocation, windowLength, tonesNo);
112 set(handles.statisticsText, 'String', ['exec time: ' num2str(toc) '
        s; '...
113     'total EM iterations: ' num2str(handles.EMOut.iterEM)]);
114 % display specgram. w/ time slider if necessary.
115 imagesc([handles.EMOut.timeAxis(1) handles.EMOut.timeAxis(end)],...
116     [handles.EMOut.freqAxis(1) handles.EMOut.freqAxis(round(end/2))
            ],...
117     log(handles.EMOut.spectralMatrix(1:round(end/2),:)));
```

```
118 set(gca,'YDir','normal'); % b/c image-function behaves abnormally
    ...
119 [pathstr, fileName, ext, versn] = fileparts(fileLocation);
120 title(['Pitch est. w/ ' num2str(tonesNo) ' tones for ' fileName ext
    ],'Interpreter','none');
121 xlabel('Time in seconds');
122 ylabel('Frequency in Hz');
123 colormap(jet);
124 % ################## CREATING SCROLL BAR FOR TIME AXIS
    ####################
125 shiftPlot= [0 0.02 0 -0.02];
126 pos= get(gca, 'position');
127 if ~isempty(handles.sliderHandle) % clear old stuff
128     delete(handles.sliderHandle);
129     handles.sliderHandle= [];
130     set(gca, 'position', [pos(1) pos(2) pos(3) pos(4)]-shiftPlot);
131 end
132 if handles.EMOut.timeAxis(end) > 2;
133     dx= 2; % dx is the width of the axis 'window'
134     % Set appropriate axis limits and settings
135     set(gcf, 'doublebuffer', 'on');
136     % This avoids flickering when updating the axis
137     set(gca, 'xlim', [windowLength dx], 'ylim', [0 handles.EMOut.
        freqAxis(round(end/2))]);
138     pos= get(gca, 'position');
139     set(gca, 'position', [pos(1) pos(2) pos(3) pos(4)]+shiftPlot);
140     % Generate constants for use in uicontrol initialization
141     scrollBarPos= [pos(1) pos(2)-0.1 pos(3) 0.05];
142     % This will create a slider which is just underneath the axis
```

```
143       % but still leaves room for the axis labels above the slider
144       xmax= handles.EMOut.timeAxis(end);
145       S= ['set(gca,''xlim'',get(gcbo,''value'')+[' num2str(
              windowLength) ' ' num2str(dx) '])'];
146       % Setting up callback string to modify XLim of axis (gca)
147       %   based on the position of the slider (gcbo)
148       % Creating Uicontrol
149       handles.sliderHandle= uicontrol('style','slider',...
150           'units','normalized','position',scrollBarPos,...
151           'callback',S,'min',0,'max',xmax-dx);
152   end
153   handles.fileLocCurrentPlot= fileLocation; % update fileLocation to
          current
154   hold on; % add means traces and pitch to specgram
155   % add pitch estimation plot to specgram
156   handles.pitchPlotHandle= plot(handles.EMOut.timeAxis, handles.EMOut
          .pitches, 'LineWidth', 2);
157   hold off;
158   for t=1:tonesNo
159       legendStrings{t}= ['pitch estimate tone ' num2str(t)];
160   end
161   legHandle= legend(handles.pitchPlotHandle, legendStrings);
162   set(legHandle, 'Color', [.7 .9 .7]);
163   set(handles.axes1, 'Visible', 'On');
164   set(handles.saveFigureButton, 'Enable', 'On');
165   guidata(hObject, handles);
166
167
168   % --- Executes on button press in playButton.
```

```
169 function playButton_Callback(hObject, eventdata, handles)
170 % hObject    handle to playButton (see GCBO)
171 % eventdata  reserved - to be defined in a future version of MATLAB
172 % handles    structure with handles and user data (see GUIDATA)
173 fileLocation= get(handles.browseButton, 'UserData');
174 [wave,Fs,bits]= wavread(fileLocation);
175 wavplay(wave, Fs);
```

B.2 STFT Analysis

```
1  function out= runEM(fileLocation, windowLength, tonesNo)
2  % out= runEM(fileLocation, windowLength, tonesNo)
3  % STFT analyis of file (50% overlap), invokes EM for pitch
       estimation
4  % - fileLocation: path and filename, concatenated
5  % - windowLength: unit shall be seconds.
6  %       common definition: windowLength==64msec
7  % - tonesNo: number of tones assumed to be hidden in file
8  % - out.spectralMatrix: for spectrogram plotting of
9  %       the input file, unfiltered
10 % - out.freq: frequency points for which DFT was computed
11 % - out.timeAxis: vector of frame analysis start points
12 %       (in sec), for plotting the out.pitches and specgram
13 % - out.pitches: columns correspond to frames, rows to
14 %       (maximally tonesNo) pitches. Inf entry/entries indicate
15 %        that for this frame: no of actual pitches < tonesNo
16 % - out.iterEM: total number of EM iterations for the file
17 %
18 % Author: Dominik Loeffler     Date: Mar 15, 2006
19 % Version: 7
20
21 if nargin~=3 || nargout~=1
22     error('runEM: must have exactly 3 input and 1 output parameters
           ');
23 end
24 if windowLength <= 2*eps
25     error('runEM: windowLength input parameter too small');
```

```
26 end
27 [path, fileName, ext, versn]= fileparts(fileLocation);
28 try
29     [wave,Fs,bits]= wavread(fileLocation);
30 catch
31     errordlg('Could not open file.');
32     error('pitchEstEM: could not open file.');
33 end
34 % windowSize equals to 1024 samples at 16kHz and 0.064s
35 windowSize= round(Fs*windowLength);
36 if mod(windowSize,2)==1
37     windowSize= windowSize+1; % ensure it's always even
38 end
39 disp(['Analyzing file' fileName]);
40 fftSize= max(windowSize, 256);
41 % freqs for which DFT will be computed
42 out.freqAxis= 0:Fs/windowSize:Fs/2-Fs/windowSize;
43
44 % advancing window in windowSize/2 steps always (-> 50% overlap)
45 indices= windowSize:windowSize/2:length(wave);
46 % standard hamming window; size== windowSize x 1
47 hamm= hamming(windowSize);
48 out.spectralMatrix= zeros(windowSize/2, length(indices));
49 specSmoothTemp= zeros(windowSize/2, 1);
50 nAvg= 2; % smoothing parameter for running average
51
52 % for each frame, save all the obtained means for each mixture
53 out.pitches= zeros(tonesNo, length(indices));
54 spec= zeros(windowSize, 1);
```

```
55 iterFrame= 0;
56 out.iterEM= 0;
57 for k= indices
58     iterFrame= iterFrame+1;
59     disp(['runEM: frame ' num2str(iterFrame) ' of '...
60         num2str(length(indices))]);
61     spec= hamm.*wave(k-windowSize+1:k);
62     if sum(realpow(spec,2))/windowSize <= 7*10^-8 % silence
           detector
63         spec= abs(fft(spec, fftSize));
64         out.spectralMatrix(:,iterFrame)= spec(1:windowSize/2);
65         out.pitches(:, iterFrame)= Inf*ones(tonesNo, 1);
66         disp('runEM: silence detected; EM skipped for this frame');
67         continue;
68     end
69     spec= abs(fft(spec, fftSize));
70     out.spectralMatrix(:,iterFrame)= spec(1:windowSize/2);
71     specSmoothTemp= filter(ones(1,nAvg)./nAvg, 1, ...
72         out.spectralMatrix(:,iterFrame));
73     %specSmoothTemp= out.spectralMatrix(:,iterFrame);
74     % find hi-freq cutoff frequency
75     ind= find(specSmoothTemp>=0.1*max(specSmoothTemp));
76     cutoffInd= ind(end)+round(windowSize/2/100);% add 1% of total
           length
77     [pitches, priorProbs, variances, iterations]= ...
78         myEM(out.freqAxis(1:cutoffInd), ...
79             specSmoothTemp(1:cutoffInd)', tonesNo);
80     out.iterEM= out.iterEM + iterations;
81     out.pitches(:, iterFrame)= pitches;
```

```
82 end
83
84 out.timeAxis= indices./Fs;
85 disp('** function runEM terminated successfully **');
86 end % function runEM
```

B.3 Core Heuristic

```
1  function [harmonicsFreq, hInd]= pitchTrace(freq, spec, tonesNo,
       verbose)
2  % [harmonicsFreq, hInd]= pitchTrace(freq, spec, tonesNo, verbose)
3  % Core heuristic for EM algorithm initialization
4  % - freq: row vector (only 1-D data!) frequency pts for which
       spectrum def.
5  % - spec: fft magnitude value, abs(fft)
6  % - tonesNo: number of polyphonic tones we assume to be hidden in
       source
7  % - harmonicsFreq: freq points at which heuristic supposes
       harmonics of a
8  %         pitch. cell array of (tonesNo,?) dimensions, 2nd dim
       expanded
9  %         dynamically for each tone
10 % - hInd: indices in freq corresponding to harmonics in
       harmonicsFreq
11 %
12 % Author: Dominik Loeffler     Date: Mar 30, 2006
13 % Version: 7 (+ weak harmonics interpolation)
14 if ~isequal(size(freq), size(spec))
15     error('pitchTrace: size of freq and spec arguments must match')
           ;
16 end
17 Nfreq= length(freq);
18 if tonesNo > Nfreq
19     error(['pitchTrace: not enough frequency points to start '...
20           'estimating ' num2str(tonesNo) ' tones']);
```

```
21 end
22
23 [peakVal, peakInd]= lmax(spec, 0, false);
24 % filter out very small peaks which are unlikely real harmonics;
25 % cut off some low-frequency peaks that cause trouble when we
       assume them
26 % to be the fundamental below. mostly occurs at very low pitches.
27 cutoffPeakAmpl= 0.05;
28 sigIndices= find(peakVal >= cutoffPeakAmpl*max(peakVal) &...
29     freq(peakInd) > 45);
30 sigPeak= peakVal(sigIndices);
31 sigPeakFreq= freq(peakInd(sigIndices));
32
33 harmonicsFreq= num2cell(Inf*ones(tonesNo, 1));
34 hInd= num2cell(zeros(tonesNo, 1));
35 sigPeakFreqTemp= sigPeakFreq;
36 for t= 1:tonesNo
37     % need to have _at least_ as many significant peaks as tones
38     if isempty(sigPeakFreqTemp) % no more peaks available -> quit
39         disp(['pitchTrace: not enough significant peaks in spectrum
              for '...
40             'estimating ' num2str(tonesNo) ' tones. ' ...
41             'I did ' num2str(t-1) ' tone(s).']);
42         return;
43     end
44     currentFundamental= sigPeakFreqTemp(1);
45     harmonicsFreq{t}= Inf*ones(1, round(sigPeakFreqTemp(end)/
           currentFundamental));
46     hInd{t}= zeros(size(harmonicsFreq{t}));
```

```
47      % tolerance in Hz hat harmonics can deviate at the max from
48      % their ideal frequency locations
49      tol= min(50, currentFundamental/4);
50      harmonicsFreq{t}(1)= currentFundamental;
51      % remove so in next iteration choose different fundamental
52      sigPeakFreqTemp(1)= [];
53      hInd{t}(1)= find(freq==currentFundamental);
54      for k= 2:length(harmonicsFreq{t})
55          binIndices= find(abs(sigPeakFreqTemp-k*currentFundamental)
              <= tol);
56          if ~isempty(binIndices)
57              indClosest= findapprox(sigPeakFreqTemp(binIndices), k*
                  currentFundamental);
58              harmonicsFreq{t}(k)= sigPeakFreqTemp(binIndices(
                  indClosest));
59              %harmonicsFreq{t}(k)= sigPeakFreqTemp(binIndices(round(
                  length(binIndices)/2)));
60              hInd{t}(k)= find(freq==harmonicsFreq{t}(k));
61              % remove the peak just found
62              sigPeakFreqTemp(binIndices(indClosest))= [];
63          end % else: leave the preinitialized Inf entries there
64      end
65
66      % *** V7: optimize this here by interpolating missing entries
            ***
67      % remove any trailing consecutive Inf entries
68      indNotInf= find(harmonicsFreq{t}~=Inf);
69      harmonicsFreq{t}= harmonicsFreq{t}(1:indNotInf(end));
70      hInd{t}= hInd{t}(1:indNotInf(end));
```

```
71      % now interpolate remaining Inf entries, if any
72      infInd= find(harmonicsFreq{t}==Inf);
73      if ~isempty(infInd) % && some other condition?
74          for k= infInd
75              binIndices= find(abs(freq(peakInd)-k*currentFundamental)
                    <= tol);
76              if ~isempty(binIndices) % attempt to interpolate with
                    low-amp peak
77                  indClosest= findapprox(freq(peakInd(binIndices)), k*
                        currentFundamental);
78                  harmonicsFreq{t}(k)= freq(peakInd(binIndices(
                        indClosest)));
79                  hInd{t}(k)= find(freq==harmonicsFreq{t}(k));
80                  %disp('pitchTrace: interpolated missing harmonic');
81              else % interpolation failed - stop here
82                  harmonicsFreq{t}= harmonicsFreq{t}(1:k-1);
83                  hInd{t}= hInd{t}(1:k-1);
84                  break;
85              end
86          end
87      end
88  end % for
89
90
91  %---- only if verbose is selected ----------------------
92  if ~verbose
93      return;
94  end
95  % graphical output (optional)
```

```
96  figure(2);
97  cla;
98  hold on;
99  plot(freq, spec); % spectrum
100 %stem(sigPeakFreq,sigPeak,'gd','LineWidth', 1.6); %significant
        peaks
101 plotProp= {'ro', 'k*', 'r^', 'y'};
102 legendString{1}= 'spectrum';
103 for t= 1:tonesNo
104     stem(harmonicsFreq{t}, ...
105         (1.1-t*0.2)*max(peakVal)*ones(1,length(harmonicsFreq{t})),
                ...
106         plotProp{mod(t,4)}); % pitch trace
107     legendString{t+1}= ['tone ' num2str(t)];
108     %stem(freq(hInd{t}), spec(hInd{t}), 'g^');
109 end
110 hold off;
111 legend(legendString);
112 title('results for pitchTrace (initial estimates for EM invocation)
        ');
113 xlabel('f/Hz');
114 ylabel('FFT(f)/max(FFT(f))');
115
116 disp(['pitchTrace: sign. peaks: ' num2str(freq(peakInd))]);
117 for t= 1:tonesNo
118     disp(['pitchTrace (tone ' num2str(t) '): found trace: '...
119         num2str(harmonicsFreq{t})]);
120     %disp(['pitchTrace (tone ' num2str(t) '): we found ' ...
121     %  num2str(100*length(find(harmonicsFreq{t}~=Inf))...
```

```
122         %  /length(harmonicsFreq{t})) ...
123         %  ' per cent significant peaks of the trace']);
124 end
125 disp(['pitchTrace: tolerance is: ' num2str(tol) 'Hz']);
126
127 end % function pitchTrace
```

B.4 Expectation Maximization Algorithm

```
1  function [pitches, priorProbs, variances, iterations]= myEM(freq,
       spec, tonesNo)
2  % [pitches, priorProbs, variances, iterations]= myEM(freq, spec,
       tonesNo)
3  % Expectation Maximization Algorithm, constrained, working on GMM
4  % - freq: row vector (only 1-D data!) frequency pts
5  %            for which spectrum defined
6  % - spec: fft magnitude value, abs(fft)
7  % - tonesNo: number of tones assumed to be hidden in file
8  % - pitches: pitch estimates for each tone
9  % - priorProbs: best estimate of prior prob for each mixture (P(X|
       mix_i))
10 % - variances: analogous...
11 % - iterations: iterations needed for EM alg to converge
12 %
13 % Author: Dominik Loeffler     Date: Mar 15, 2006
14 % Version: 7
15
16 Nfreq= length(freq);
17 if ~isequal(size(freq), size(spec))
18     error('myEM: size of freq and spec arguments must match
           precisely');
19 end
20 sumSpec= sum(spec);
21 if sumSpec/Nfreq < eps % should be caught by silence detector on
       caller side
```

```
22      error('myEM: exited prematurely (sum of spectrum near 0 -
            silence!)');
23  end
24  spec= spec./sumSpec;
25
26  % use this heuristic to find good initial estimates
27  % meansOld: a cell array of tonesNo elements, each element with
        variable
28  %   components as determined by the heuristic "pitchTrace"
29  [meansOld, meansInd]= pitchTrace(freq, spec, tonesNo, false);
30  % 2nd dim will be expanded in the following loop
31  means= num2cell(Inf*ones(tonesNo, 1));
32  variances= means; % stays constant in this version
33  priorProbsOld= means;     priorProbs= means;
34  mixtures= zeros(1, tonesNo); % no of peaks for each tone
35  pitches= Inf*ones(tonesNo, 1);
36  iterations= 1;
37  for t= 1:tonesNo
38      % heuristic couldn't come up with anything sensible
39      if meansOld{t}(1)==Inf
40          disp(['myEM (tone ' num2str(t) ...
41              '): heuristic output: not enough data for this tone']);
42          return;
43      end
44      mixtures(t)= length(meansOld{t});
45      variances{t}= 10*ones(1, mixtures(t));
46      priorProbsOld{t}= spec(meansInd{t}); % then normalize
47      priorProbsOld{t}= priorProbsOld{t}./sum(priorProbsOld{t});
48      means{t}= zeros(1, mixtures(t));
```

```
49      priorProbs{t}= means{t};
50
51      disp(['myEM (tone ' num2str(t) ...
52          '): init(heur.) means: ' num2str(meansOld{t})]);
53      %disp(['myEM: initial priorProbs: ' num2str(priorProbsOld{t})])
            ;
54      %disp(['myEM: initial variance: ' num2str(variances{t})]);
55
56      y= zeros(mixtures(t), Nfreq);
57      %meansOld{t}= ...
58      %     [1:mixtures(t)].*(sum((1:mixtures(t)).^-1.*meansOld{t})...
59      %     /mixtures(t)); % resetting with good initial pitch
            estimate
60      firstPeak= meansOld{t}(1);
61      meansOld{t}= [1:mixtures(t)] .* meansOld{t}(1);
62      disp(['myEM (tone ' num2str(t) '): init(pitch) means: ' num2str
            (meansOld{t})]);
63      deltaF= 0; deltaFOld= 0;
64
65      while (1) % loop for the EM of _one_ pitch
66          sumY= zeros(1, Nfreq); % E-step
67          for k= 1:mixtures(t)
68              y(k,:)= priorProbsOld{t}(k)*...
69                  myGauss(freq, [sqrt(variances{t}(k)), meansOld{t}(k
                        )], true);
70              sumY= sumY + y(k,:);
71          end
72          sumYZeroInd= find(sumY==0);
73          if ~isempty(sumYZeroInd)
```

```
74              sumY(sumYZeroInd)= eps;
75              %disp(['myEM (tone ' num2str(t) ...
76              %       '): zero sum of likelihoods occurred, fixing it
                    now']);
77          end
78
79          for k= 1:mixtures(t)
80              y(k,:)= y(k,:)./sumY;
81              specMultY= spec.*y(k,:);
82              sumSpecMultY= sum(specMultY);
83              if sumSpecMultY==0
84                  % seems to happen when unsmoothed or few distro
                        values
85                  means{t}= zeros(1, mixtures(t));
86                  priorProbs{t}= zeros(1, mixtures(t));
87                  variances{t}= zeros(1, mixtures(t));
88                  disp('myEM: exited prematurely (sum(spec.*y(k,:))''
                        ==0)');
89                  return;
90              end
91              priorProbs{t}(k)= sumSpecMultY;   % M-step
92              means{t}(k)= sum(freq.*specMultY)/sumSpecMultY;
93          end
94
95          %deltaF= sum(means{t}-meansOld{t})/sum(abs(means{t}-
                meansOld{t}));
96          % if pos: increases in frequency higher than decreases
97          deltaF= sum((1:mixtures(t)).^-1.*(means{t}-meansOld{t}))/
                mixtures(t);
```

```
98          means{t}= [1:mixtures(t)] .* (meansOld{t}(1)+deltaF);
99          %disp(['deltaF ' num2str(deltaF)]);
100         if abs(deltaF) <= 0.01   || abs(deltaF+deltaFOld) <= 2*eps
                || ...
101                 means{t}(1) > firstPeak*(2^1/24) || ...
102                 means{t}(1) < firstPeak*(2^-1/24)
103             break;
104         end
105
106         priorProbsOld{t}= priorProbs{t};
107         meansOld{t}= means{t};
108         deltaFOld= deltaF;
109         iterations= iterations+1;
110     end % while
111
112     % sort the outputs in ascending order of the means
113     % (sometimes they get flipped around by EM)
114     [means{t}, ind]= sort(means{t});
115     priorProbs{t}= priorProbs{t}(ind);
116     pitches(t)= means{t}(1);
117     disp(['myEM (tone ' num2str(t) ...
118         '):       after EM: ' num2str(means{t})]);
119 end % for
120
121 end % function myEM
```

B.5 Gaussian Function

```
1  function g= myGauss(input, params, dirtyFix)
2  % g= gauss(input, [sigma mean], dirtyFix)
3  % like gaussmf, but gaussmf is not good enough:
4  % need cases with sigma==0 and the scaling factor
5  % 1/(sqrt(2*pi)*sigma) for most general uses.
6  % note that sigma denotes standard deviation==sqrt(variance)
7  % - input: 1-D vector with real values only
8  % - params: 1x2 vector:
9  %   params(1)=standard deviation, params(2)=mean
10 % - dirtyFix: logical, indicates whether or not a case of
11 %   variance=0 should be fixed by returning 1000 for the input
12 %   value closest (not necessarily equal!) to the mean specified
13 %   in "params". This guarantees that the output vector be nonzero.
14 %
15 % Author: Dominik Loeffler  Date: Oct 20, 2005  Last update: Nov 06
16 % Version: 5 (+ real values optimization w/ realpow,
17 %             + dirtyFix)
18
19 if nargin~=3
20     error('myGauss: wrong number of input arguments');
21 end
22 if (~isequal(size(params), [1 2]) || params(1) < 0)
23     error('myGauss: wrong input parameters "params"');
24 end
25 if ~isvector(input)
26     error('myGauss: input is not a vector');
27 end
```

```
28 if isempty(input)
29     error('myGauss: input vector is empty');
30 end
31 if ~islogical(dirtyFix)
32     error('myGauss: third input parameter is not a logical');
33 end
34
35 sigma= params(1);
36 mean= params(2);
37 if sigma==0 % -> gauss goes to infinity in continuous case
38     g= zeros(size(input));
39     if dirtyFix
40         % choose the next two lines if your "input" vector can
41         %  have some arbitrary values, i.e. its elements are
42         %  not unique
43         %closest= find(abs(input-mean)==min(abs(input-mean)));
44         %g(closest(round(length(closest)/2)))= 1000;
45         [m mIndex]= min(abs(input-mean));
46         g(mIndex)= 1000;
47     else
48         equal= find(input==mean);
49         if isempty(equal)
50             warning(['myGauss: returning zero output ' ...
51                     'vector due to zero variance']);
52         end
53         g(equal)= 1000;
54     end
55 else
56     g= (1/(sqrt(2*pi)*sigma))*...
```

```
57              exp(-realpow(input-mean,2)/(2*realpow(sigma,2)));
58  end
59
60  end % function myGauss
```

B.6 Local Maxima

```
1  function [lmval,indd]=lmax(xx, filt , strict)
2  %LMAX  [lmval, indd]=lmax(xx,filt). Find local maxima
3  %   in vector XX, where LMVAL is the output vector with
4  %   maxima values, INDD  are the corresponding indexes,
5  %   FILT is the number of passes of the small running
6  %   average filter in order to get rid of small peaks.
7  %   Default value FILT =0 (no filtering). FILT in the
8  %   range from 1 to 3 is usually sufficient to remove
9  %   most of a small peaks
10
11 %*************************************************|
12 %  Serge Koptenko, Guigne International Ltd., |
13 % phone (709)895-3819, fax (709)895-3822      |
14 %---------------06/03/97--------------------------------|
15
16 x= xx;
17 len_x= length(x);
18 fltr=[1 1 1]/3;
19 if nargin < 2
20     filt= 0;
21 else
22     x1= x(1);
23     x2= x(len_x);
24     for jj= 1:filt
25         c= conv(fltr ,x);
26         x= c(2:len_x+1);
27         x(1)= x1;
```

```
28          x(len_x)= x2;
29      end
30  end
31  lmval=[]; indd=[];
32  i=2;    % start at second data point in time series
33  while i < len_x-1
34      if x(i) > x(i-1)
35          if x(i) > x(i+1) % definite max
36              lmval= [lmval x(i)];
37              indd= [indd i];
38          elseif (x(i) == x(i+1)) && (x(i) == x(i+2)) % 'long' flat
                spot
39              if ~strict
40                  lmval= [lmval x(i)];
41                  indd= [indd i];
42              end
43              i= i+2; % skip 2 points
44          elseif x(i) == x(i+1) % 'short' flat spot
45              if ~strict
46                  lmval= [lmval x(i)];
47                  indd= [indd i];
48              end
49              i= i+1; % skip one point
50      end
51   end
52   i= i+1;
53  end
54  if filt>0 && ~isempty(indd)
55   if (indd(1) <= 3) || (indd(length(indd))+2 > length(xx))
```

```
56            rng= 1; % check if index too close to the edge
57        else
58            rng= 2;
59    end
60    for ii=1:length(indd)    % Find the real maximum value
61            [val(ii) iind(ii)]= max(xx(indd(ii)-rng:indd(ii)+rng));
62            iind(ii)= indd(ii)+iind(ii)-rng-1;
63        end
64        indd= iind;
65        lmval= val;
66    end
```

B.7 Findapprox

```
1  function i= findapprox(vector, scalar)
2  % i= findapprox(vector, scalar)
3  % Find closest element in vector, akin to find.
4  % - scalar: value we're trying to match against vector
5  % - vector: of same datatype as scalar
6  % if there is no element in vector equal to scalar, the index
7  % of the element being closest to it will be returned (thus
8  % the name findAPPROX). If more that could be more than one index,
9  % the first (leftmost) is taken only.
10 %
11 % USE WITH CAUTION. it doesn't check for input errors.
12 % Author: Dominik Loeffler   Last update: Nov 26, 2005
13
14 diff = abs(vector-scalar);
15 i = find(min(diff)==diff, 1, 'first');
16
17 end % function findapprox
```

REFERENCES

[1] Akaike, H., "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, 1974.

[2] AssociatedPress, "Online and wireless music sales tripled in 2005." WWW, January 2006. Accessed: March 06, 2006.

[3] Brown, J. C., "Calculation of a constant q spectral transform," *The Journal of the Acoustical Society of America*, vol. 89, no. 1, pp. 425–434, 1991.

[4] Casey, M. A. and Westner, A., "Separation of mixed audio sources by independent subspace analysis," in *Proceedings of ICMC*, pp. 154–161, 2000.

[5] Duda, R. O., Hart, P. E., and Stork, D. G., *Pattern Classification*. Hoboken, NJ: John Wiley & Sons, second ed., 2001.

[6] Fechner, G. T., *Elemente der Psychophysik*. Bristol, UK: Thoemmes Press, second ed., 1889. First ed. 1860.

[7] Feder, M. and Weinstein, E., "Parameter estimation of superimposed signals using the em algorithm," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 36, no. 4, pp. 477–489, 1988.

[8] Fitzgerald, D., *Automatic Drum Transcription and Source Separation*. PhD thesis, Dublin Institute of Technology, 2004.

[9] Fitzgerald, D., "Generalised prior subspace analysis for polyphonic pitch transcription," in *Proceedings of DAFX*, 2005.

[10] Fitzgerald, D., Lawlor, R., and Coyle, E., "Prior subspace analysis for drum transcription," in *Proceedings of AES 114th Convention*, 2003.

[11] Greenberg, S. and Kingsbury, B. E. D., "The modulation spectrogram: in pursuit of an invariant representation of speech," in *Proceedings of ICASSP*, 1997.

[12] Howard, D. M. and Angus, J., *Acoustics and Psychoacoustics*. Burlington, MA: Focal Press/Elsevier, 1996.

[13] Hyvarinen, A., Karhunen, J., and Oja, E., *Independent Component Analysis*. Hoboken, NJ: John Wiley & Sons, 2001. GT library call number: QA278 .H98 2001.

[14] ISO, "Acoustics - standard tuning frequency (standard musical pitch): Iso 16." WWW, 1975.

[15] Kameoka, H., Nishimoto, T., and Sagayama, S., "Accurate f0 detection algorithm for concurrent sounds based on em algorithm and information criterion," in *Proceedings of Special Workshop in Maui (SWIM)*, 2004.

[16] Kameoka, H., Nishimoto, T., and Sagayama, S., "Separation of harmonic structures based on tied gaussian mixture model and information criterion for concurrent sounds," in *Proceedings of ICASSP*, 2004.

[17] Klapuri, A., Virtanen, T., and Holm, J.-M., "Robust multipitch estimation for the analysis and manipulation of polyphonic musical signals," in *Proceedings of DAFX*, 2000.

[18] Laird, N. M., Dempster, A. P., and Rubin, D. B., "Maximum likelihood from incomplete data via the em algorithm," *Ann. Royal Statistical Society*, pp. 1–38, 1977.

[19] University of Iowa, Electronic Music Studios, "Musical instrument samples." WWW. Accessed: March 2006.

[20] Meredith, D. and Wiggins, G., "Comparing pitch spelling algorithms," in *Proceedings of ISMIR*, pp. 280–287, 2005.

[21] Mitchell, T. M., *Machine Learning*. New York, NY: McGraw-Hill, 1997.

[22] Ohgushi, K. and Ano, Y., "The relationship between musical pitch and temporal responses of the auditory nerve fibers," in *Integrated Human Brain Science: Theory, Method, Application (Music)* (Nakada, T., ed.), pp. 357–364, Elsevier Science B. V., 2000.

[23] Pantev, C., Eulitz, C., Verkindt, C., Hampson, S., Schruierer, G., and Elbert, T., "Specific tonotopic organizations of different areas of the human auditory cortex revealed by simultaneous magnetic and electric recordings," *Electroencephalography and clinical Neurophysiology*, vol. 94, pp. 26–40, 1995.

[24] Sagayama, S., Takahashi, K., Kameoka, H., and Nishimoto, T., "Specmurt anasylis: A piano-roll-visualization of polyphonic music signal by deconvolution of log-frequency spectrum," in *Proceedings of ISCA. SAPA*, 2004.

[25] Saito, S., Kameoka, H., Nishimoto, T., and Sagayama, S., "Specmurt anasylis of multi-pitch music signals with adaptive estimation of common harmonic structure," in *Proceedings of ISMIR*, pp. 84–91, 2005.

[26] Schmidt-Jones, C., "Connexions project (cnx): Harmonic series." WWW. Accessed: March 2006.

[27] Stevens, S. S., "On the psychophysical law," *Psychological Review*, vol. 64, no. 3, pp. 153–181, 1957.

[28] Tervaniemi, M., "Automatic processing of musical information evidenced by eeg and meg recordings," in *Integrated Human Brain Science: Theory, Method, Application (Music)* (Nakada, T., ed.), pp. 325–335, Elsevier Science B. V., 2000.

[29] Tervaniemi, M., Kujala, A., Alho, K., Virtanen, J., Ilmoniemi, R., and Näätänen, R., "Functional specialization of the human auditory cortex in processing phonetic and musical sounds: A magnetoencephalographic study," *NeuroImage*, no. 9, pp. 330–336, 1999.

[30] Tiitinen, H., Alho, K., Huotilainen, M., Ilmoniemi, R., Simola, J., and Näätänen, R., "Tonotopic auditory cortex and the magneto encephalographic (meg) equivalent of the mismatch negativity," *Psychophysiology*, vol. 30, pp. 537–540, 1993.

[31] Virtanen, T. and Klapuri, A., "Separation of harmonic sounds using linear models for the overtone series," in *Proceedings of ICASSP*, vol. 2, pp. 1757–1760, 2002.

[32] von Helmholtz, H. L. F., *On the Sensations of Tone as a Physiological Basis for the Theory of Music*. New York, NY: Dover Publications, 1954. First ed. (German) 1863. GT library call number: ML3820.H42 1957.

[33] Wikipedia, "Scientific pitch notation." WWW.

[34] Xue, W. and Sandler, M., "A partial searching algorithm and its application for polyphonic music transcription," in *Proceedings of ISMIR*, pp. 690–695, 2005.