# Automatic Segmentation of Sung Melodies

Norman H Adams

December 15, 2002

**Abstract**

The present work explores several techniques for the automatic segmentation of sung melodies. Most contemporary music information retrieval (MIR) systems require sung queries to be segmented into disjoint regions representing individual notes for database searching. The fundamental philosophy adhered to throughout this work is that a melody segmentation algorithm should rely primarily on fundamental pitch information. Three classes of segmentation algorithms are explored: predictive filtering, LMS detection and curve fitting. Two different predictive filter formulations are presented, Kalman filters and adaptive RLS filters. A single-node neural network, or perceptron, is implemented as a LMS detector. Lastly, a curve fitting algorithm is developed. The curve fitting algorithm makes use of dynamic programming to keep the computational complexity manageable.

# 1 Introduction

The music segmentation problem can be viewed as one component of the transcription problem. We are given a recording of a passage of music and wish to reconstruct the musical score the performer used in creating the recording, assuming the piece is based on a written score.

Sung melodies are monophonic, no more than one note is ever articulated at once. For monophonic music transcription, the audio recording must be partitioned into disjoint regions in time, with each region corresponding to one note. In particular, this work will explore a variety of techniques for segmenting melodies sung by untrained singers.

Three broad techniques are explored: predictive filtering, neural networks and curve fitting. Two different predictive filter formulations are presented in Chapter 3, Kalman filters and adaptive RLS filters. A single-node neural network is implemented as a perceptron in Chapter 4. A perceptron is equivalent to a LMS filter with a threshold decision performed on the filter output. Lastly, a curve fitting algorithm is presented in Chapter 5. The curve fitting algorithm makes use of dynamic programming to keep the computational complexity reasonable.

The next section provides some background and motivation for this work. The fundamental approach is stated in Chapter 2. A pitch tracking algorithm first presented in [5] is used throughout the present work, and is described in section 2.2. The global performance metric used to compare the various techniques is described in section 2.3.

This research is a part of the ongoing MUSART project, a collaboration between the University of Michigan and Carnegie Mellon University to develop a robust music information retrieval (MIR) system for aural queries.

## 1.1 Background & Motivation

The rapidly increasing number of digital audio files available on the Internet motivates the question of how someone might find a particular piece of music when all they know of the piece is the melody, rhythm or timbre. The music information retrieval (MIR) systems currently being developed hope to answer this question.

Most individuals, lacking any specialized music training, remember a piece of music by it's hook, riff, theme, or chorus. That is, they remember some characteristic segment of the melody. They have no knowledge, or interest, in the precise notes that define the tune. As such, it is desirable to be able to search the Internet, or any other digital audio database, with a sung melody as the *only* query input.

The final years of the $20^{\text{th}}$ Century were witness to the first concerted developments in music information retrieval. The MELody inDEX (MELDEX), developed by the New Zealand Digital Library, was one of the first MIR systems, and to this day remains one of the more prominent systems available [20]. MELDEX accepts a sung melody as input and searches its database of 9400 folk tunes, returning the tunes with similar melodic segments. One of the restrictions the system places on the user is that every note must be separately articulated with a 'da' or 'ta'. MELDEX segments the recorded audio file with a constant amplitude threshold. If the user sung a continuous melody or lyrics, the algorithm would be unable to isolate separate notes [20]. It has been shown that even when the user adheres to this restriction, the majority of errors MELDEX makes result from this rudimentary segmentation method [7, 19, 20].

Several other MIR systems have been developed since MELDEX [4, 15, 16, 21], all of which use some sort of transcription of a sung query for searching. The transcription is found by computing a pitch-contour, or pitch-track, from the sung query and segmenting this contour into disjoint rejoins representing individual notes. The segmentation step is performed using either an amplitude threshold (not necessarily constant) or a manually input metronome track.

Before proceeding we should clarify why the above restriction is undesirable. For untrained singers it is often simpler to sing a melody with the lyrics they are familiar with, the lyrics aid the singer in reproducing the correct pitches and rhythms. From a more technical perspective, the lyrics themselves contain information that could be used by the MIR. To date, such phonetic-stream analysis has yielded limited results, but the method is still in its infancy [4].

It would be reasonable to propose circumventing the segmentation problem altogether if it is so troublesome. One alternative that has been explored is the use of the pitch-track itself in performing the database search. Rather than completing the transcription of the query and using the transcribed notes to search the database, use the pitch-track directly. While this method has been shown to be promising in some circumstances, it is computationally intensive and cannot be easily scaled to large databases [4, 18].

As mentioned above, initial attempts to solve the melody segmentation problem use a simple amplitude threshold [7, 20]. This approach, of course, only works when the singer inserts brief pauses between every note. More sophisticated approaches have been explored. [20] also proposes the use of a smoothed discrete derivative of the pitch-track to perform segmentation. The pitch-track is partitioned into $20ms$ regions and if any two consecutive regions differ in pitch by more than 50 cents a new note is instantiated. While this approach does partially loosen the above restriction, performance is mediocre, as shown in section 2.3.

Hidden Markov Models (HMM) were proposed in [23] to aid with segmentation, however the application was somewhat different. The author was exploring music segmentation for use with automatic music accompaniment systems. In this case the musical score is also available to the algorithm, and hence the problem is closer to score following than blind segmentation. The use of HMMs was extended to unsupervised segmentation in [3], but with mixed results. HMMs were more successfully employed in [9] to segment piecewise constant signals corrupted by Gaussian white noise. While the pitch-tracks generated from sung melodies do resemble piecewise constant signals, the corrupting 'noise' is colored and nonstationary. The assumptions made in [9] are too restrictive to apply the method directly to segmenting sung melodies.

A related music segmentation problem was addressed in [25] with Kalman filters. A Kalman filter was used to track every partial detected in a recording of a brass quartet. When the Kalman prediction failed to match the observed data, a note-change was assumed

to have occurred. The method worked well for a brass quartet, however it is unclear whether a Kalman filter is well-suited to predict the more volatile pitch-tracks produced by sung melodies. The Kalman filter is explored in Chapter 3.

Automatic segmentation of tonal languages was explored in [8]. Segmenting tonal languages is similar in some regards to segmenting melodies in that tonal languages associate meaning to different pitches. It was found in [8] however that human segmentation of tonal languages relied heavily on contextual information, and hence could not be reliably automated.

While automatic segmentation of sung melodies is still a relatively new field, there are two related audio segmentation problems that have been more thoroughly explored. Automatic segmentation for continuous speech recognition is a mature research area, numerous techniques have been explored with several notable successes [1, 24, 28]. These methods all make use of a very different set of features than considered here however. Similarly, automatic segmentation of audio content uses features such as spectral shape and zero-crossings (to detect regions of pure dialog vs regions with other material, for example) [6, 14, 22, 30]. As described below, the fundamental approach here is to detect note changes based only on pitch-track information, without considering any phonetic-stream information.

# 2 Approach

A fundamental assumption behind this work was that a sung melody should be segmented using *primarily* pitch information. It is not uncommon for a singer to sing a melodic passage with constant volume and timbre. Only the pitch is changing, all other relevant features are constant. This is not to say amplitude or phonetic-stream information should not be used, but that the segmentation algorithm must be able to work even when these features contain no information useful to segmentation. In this work no phonetic-stream or timbrel information was considered. Amplitude information was incorporated into the perceptron described in Chapter 4, but aside from this only pitch information is utilized.

## 2.1 Recorded data

All of the sung melodies used in this research were recorded in the School of Music at the University of Michigan. Five students without any vocal training sang a collection of ten popular tunes, from the Beatles' "Hey, Jude" to Richard Rodgers' "Sound of Music", four times each. All recordings were mono, 16 bit. The sampling rate was 44.1 kHz. The recordings are not professional however, and do contain some background noise and the occasional student giggle. Of the 200 files in the database, 80 were manually segmented for training and analysis of the algorithms below.

## 2.2 Pitch track

The segmentation methods developed here make use of an autocorrelation-based pitch tracker developed in [2, 5]. The pitch track algorithm takes the input audio file and slices it into frames of length $10ms$. The fundamental frequency is then computed for each frame. The algorithm returns three discrete signals, all of which are indexed by the same $10ms$ interval: the frequency track, the autocorrelation track and the root mean square (RMS) amplitude track. The frequency track is the reciprocal of the location of maximum autocorrelation. The autocorrelation track is the correlation value at this maximum. And the amplitude track is the root mean square value in each $10ms$ frame. The autocorrelation track can be viewed as a confidence measure in the frequency estimate for each frame.

Rather than work with the frequency track directly, the frequency track is converted to MIDI (musical instrument digital interface) pitch number. If $f_k$ is the frequency track

value at time step $k$, then the pitch track value is given by,

$$p_k = 12 \cdot \log_2 \left( \frac{f_k}{440} \right) + 69. \tag{1}$$

For example, the A natural below middle C is assigned the MIDI number 69 ($440Hz$) and the A# below middle C is assigned the MIDI number 70 ($466Hz$). MIDI scaling aids in interpretation of the pitch track. Note that in general $p_k$ is not an integer.

The pitch tracker has the useful property that whenever it cannot confidentially estimate the pitch for a given frame, it outputs a fundamental frequency of zero. Hence pauses and unvoiced phonemes are represented by zero in the pitch track. The practical result of this that the pitch track itself performs a partial segmentation automatically. In fact, this rudimentary segmentation is roughly equivalent to the amplitude threshold segmentation described in the previous chapter. Note that the pitch track *never* pins to zero when a note is being clearly articulated.

In detection terminology, the pitch track performs a segmentation with zero *false-alarm* probability and a rather high *miss* probability. In this context, a *false-alarm* is mistakenly deciding that a note-change occurred when in truth one did not. Similarly, a *miss* is mistakenly deciding that a note-change has not occurred when in truth one has. Hence, the goal of the segmentation algorithm developed here is too reduce the *miss* rate without substantially increasing the *false-alarm* rate.

The output of the pitch tracker is first inspected to find the regions where the pitch track is nonzero. Very short regions, of length less than 6 frames ($60ms$) are assumed to be spurious and discarded. Regions of length less than 20 frames ($200ms$) are assumed to only contain one note and hence not analyzed for note-changes. These values were found experimentally.

A sample pitch track, along with a manual segmentation, is shown in Figure 1. This pitch track was generated from the tune "Rock a bye baby," the lyrics corresponding to this portion of pitch track are "ro 'ka bye ba-by in the." Note that the manual segmentation is represented as a piecewise constant function. The specific values of each constant region of of no interest here, they were computed simply as the weighted average of the pitch track within each region (weighted by the correlation track). The beginning and end of each constant region represent the manual segmentation.

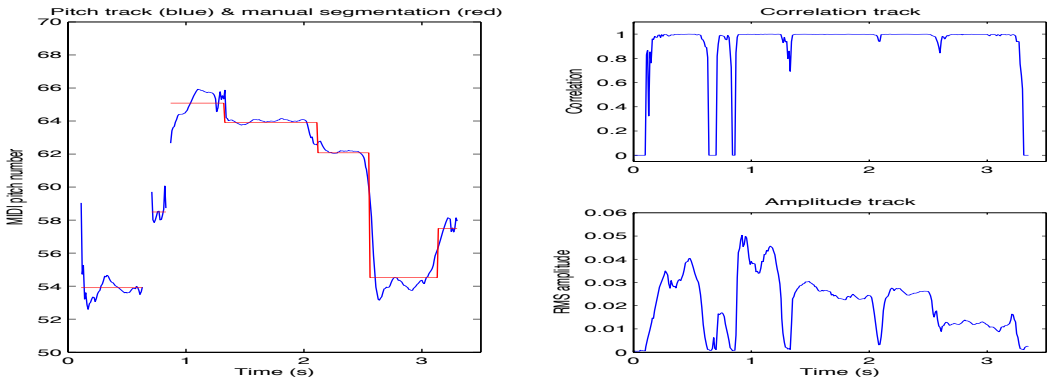


Figure 1: The left plot shows a sample pitch track (blue) along with a manual segmentation (red). This pitch track corresponds to the lyrics "Rock a bye baby in the" from the tune of the same name. The right plots show the corresponding correlation and amplitude tracks.

Figure 1 shows are relatively clean pitch track. Most of the recordings in the database did not produce such clean pitch tracks. Nonetheless, it is apparent that the 'ideal' pitch track is in fact a piecewise constant signal. This would represent a singer that transitions instantly between notes and maintains each pitch exactly, with no fluctuations.

Framed in this way, the melody segmentation problem resembles the edge detection problem in image processing, where edge detection algorithms partition two-dimensional signals the melody segmenter partitions one-dimensional signals. Having made this connection, it is apparent that just as the edge detection problem is ill-defined [17], so is the melody segmentation problem. There exists no general analytic solution. Rather, a collection of different techniques is needed to solve the melody segmentation problem in different contexts, with each technique relying on its own assumptions.

## 2.3 Performance metrics

Two types of errors are possible for automatic segmentation algorithms. Deletion-errors (missed notes) occur if a true note boundary has not been detected by the segmenter. Insertion-errors (false-alarm notes) occur if a detected note boundary does not correspond to a true note boundary. Most segmentation algorithms can trade one type of error for the other. The information retrieval community uses two measurements to quantify the frequency of both deletion and insertion errors, precision (PRC) and recall (RCL) [13, 14],

$$\begin{aligned} \text{RCL} &= \frac{\text{\# of correctly found boundaries}}{\text{total \# of true boundaries}} = 1 - \hat{p}_m = \hat{p}_d \\ \text{PRC} &= \frac{\text{\# of correctly found boundaries}}{\text{total \# of hypothesized boundaries}} = 1 - \hat{p}_{fa} \end{aligned} \tag{2}$$

where $\hat{p}_m$ and $\hat{p}_{fa}$ are the measured miss and false-alarm rates, respectively. $\hat{p}_d$ is the measured detection rate. A plot of $\hat{p}_d$ versus $\hat{p}_{fa}$ is referred to as a receiver operator characteristic (ROC).

While ROC curves are familiar to anyone with experience in detection theory, the MIR community often prefers to measure segmentation performance in terms of the number of missed and inserted notes *per query*. Let MPQ represent the average number of missed notes per query and IPQ represent the average number of inserted notes per query.

Plotting MPQ versus IPQ yields a curve similar to an ROC curve, but the scaling is somewhat different. The manually segmented files in the database used in this work contain an average of 14.55 notes. As such, MPQ will take on values between 0 and 14.55, but IPQ can take on values between 0 and $\infty$. $\hat{p}_d$ and $\hat{p}_{fa}$ both take on values between 0 and 1. Both plots will be provided throughout this work.

It should be noted that the correct position of a note boundary is not well defined. The manually segmented files were segmented by this author. There were numerous instances where the precise boundary locations were arbitrary within $50ms$. Furthermore, it has been found that missed/inserted notes are far more damaging to MIR performance than slight rhythmic errors [4, 19]. With this being the case, a reasonably large radius of $100ms$ was chosen for associating hypothesized and true note boundaries.

To serve as a baseline to compare the methods explored here, two conventional melody segmentation algorithms were implemented. The constant amplitude threshold and smoothed discrete derivative methods described in section 1.1 were tested on the database of 80 manually segmented melodies. The results are shown in Figure 2.

The amplitude threshold method is represented simply as a point. This is because the singers sang continuous lyrics. Modifying the specific threshold for detecting new notes did not significantly reduce the number of missed notes. The performance of the smoothed derivative method could be changed by adjusting the pitch change threshold. The threshold was swept between 0.5 and 4 MIDI pitch values to produce the curves shown. Note that using a $20ms$ bin size (as was used in [20]) results in mediocre performance. Increasing the bin size to $80ms$ (corresponding to more smoothing) improves performance considerably. Both curves intersect the amplitude threshold operating point. The segmentation algorithms can be viewed as searching for additional notes *after* the pitch tracker has performed a partial segmentation.

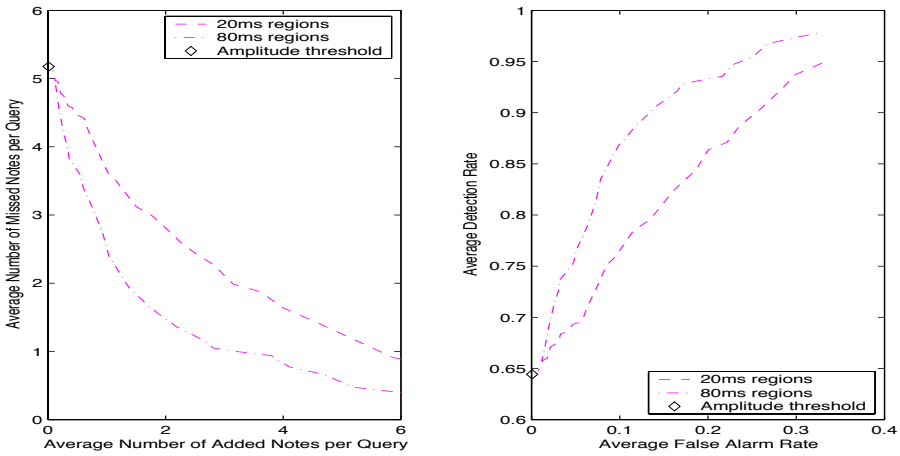


Figure 2: The left plot shows the average number of inserted notes per query vs. the average number of missed notes per query. The right plot shows the corresponding ROC curves.

# 3 Predictive filtering

One of the difficulties encountered when trying to detect a note change was that often the magnitude of the change in pitch near a note boundary was no greater than the pitch fluctuations within a single note. The pitch tracks were very volatile, but not necessarily unpredictable. In particular, many of the pitch tracks contained significant *vibrato*. Vibrato is an oscillation in pitch around some average frequency. Typically vibrato is roughly sinusoidal with frequency between two and five Hz.

The observed patterns in the pitch tracks motivate the question, can the pitch track be modelled in such a way as to distinguish between the fluctuations in pitch within a note (due to vibrato and such) versus those that indicate a note change? Predictive filtering techniques were explored to estimate *future* pitch track values based on a collection of *current* states. Specifically, a time-invariant Kalman filter and a recursive least squares (RLS) filter were considered. Both filters use a model of signal behavior to predict future pitch track values based on current states, such as past and present pitch track values. In the case of the Kalman filter the signal model was specifically designed to account for vibrato, whereas for the RLS filter the signal model automatically adapts itself to pitch track fluctuations. When the filters made an accurate prediction the current signal model was assumed correct and the new pitch track value was added to the current note. Conversely, when the predictive filters did not accurately estimate the next pitch track value a new note was instantiated.

The development and performance of the Kalman filter is given in sections 3.1 through 3.5, and the RLS filter is presented in sections 3.6 through 3.8.

## 3.1 Statistical Analysis

If the pitch track, within any single note, could be modelled as the output of a stationary linear system driven by white noise, a Kalman filter would be a natural choice for tracking the behavior of the pitch track. The first step in designing the Kalman filter was to develop a model of the pitch track behavior within any single note. This was done by first performing a nonparametric spectral estimation on individual notes and then averaging across all notes in search of consistent characteristics. In particular, Welch's averaged periodogram method was used to estimate the power spectral density of the pitch tracks [26]. The statistical analysis performed here was modelled after a similar analysis in [25].

Of the 200 sound files in the database the author selected 20 files that contain a large number of sustained notes. From these files regions of each sustained note that were relatively 'stable' for at least $500ms$ were selected. In this case 'stable' was taken to mean that any transient behavior due to note transitions was negligible. Singers would take up to $100ms$ to slide into new notes, these so-called 'scooping' regions were not considered part of a 'stable' note. Regions of the pitch track that contained any other fluctuation, such as vibrato, were retained. A total of 70 regions were included in the analysis.

The mean pitch was subtracted from each region. The regions were divided into $400ms$ segments (40 samples), with $200ms$ overlap (20 samples). Each segment was windowed with a Blackman window, normalized and zero-padded. The magnitude squared DFT was computed for each segment and averaged over the region to give the *pitch variation* PSD. Two sample pitch track regions and their corresponding PSDs are shown in Figure 3. These 70 PSDs were then averaged in the log-domain, yielding an average pitch variation PSD.

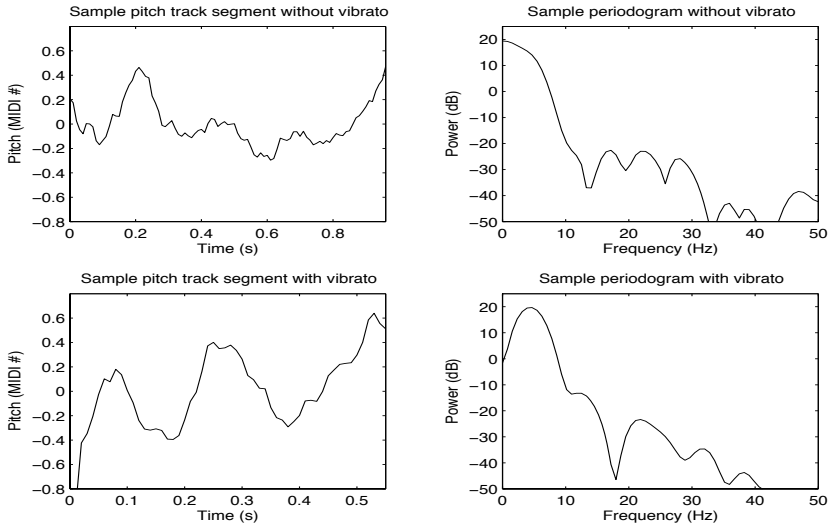


Figure 3: Sample pitch track regions and their corresponding PSDs. The top plots represent a segment without any vibrato and the bottom plots represent a segment with vibrato.

In addition to the spectral analysis, a histogram of the pitch track data was computed as an estimate of the PDF. The histogram was computed using 81 bins uniformly distributed over the interval [-2,2].

The final PSD estimate and PDF estimate are shown in Figure 4. As can be seen from the figure, the pitch variation PSD does exhibit a second-order low-pass characteristic with a slight resonance at approximately 3 Hz. The PDF estimate is shown with the minimum squared-error best fit Gaussian PDF. The match is reasonable, although the pitch variation PDF would perhaps be better described as double exponential. Nonetheless, the match is good enough to justify the use of a Kalman filter.

## 3.2 Pitch Evolution Model

The spectral analysis performed in the previous section implied that the pitch evolved according to a second order autoregressive noise process. In particular,

$$p_k = p_{k-1} + w_k \tag{3}$$

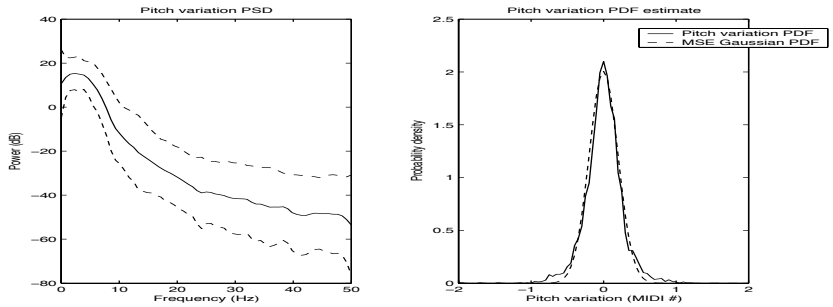


Figure 4: The plot on the left is the average pitch variation PSD. One standard deviation is indicated by the dotted lines. The plot of the right is the pitch variation PDF estimate. The best squared-error fit Gaussian PDF is shown by the dotted line.

where $p_k$ is the pitch at discrete time step $k$ and $w_k$ is the driving noise process. The behavior of the noise process is given by,

$$w_k \;=\; \alpha_1\, w_{k-1} \;+\; \alpha_2\, w_{k-2} \;+\; G\, u_k \tag{4}$$

where $\alpha_1$ and $\alpha_2$ are the autoregressive (AR) model parameters and $u_k$ is the input white Gaussian noise process, scaled by $G$. $\alpha_1$ and $\alpha_2$ are the forward linear predictor coefficients for the AR(2) prediction of $w_k$ based on $w_{k-1}$ and $w_{k-2}$. In assuming an autoregressive model, any of the conventional parametric spectral estimation techniques could be used to compute $\alpha_1$ and $\alpha_2$. The least squares method, using the Levinson-Durbin recursion, was chosen to compute the AR(2) coefficients [26].

Before proceeding, recall that some of the pitch track regions contain substantial vibrato and some do not. Using the data that does contain vibrato versus the data that does not contain vibrato produces two different sets of AR(2) coefficients. The AR(2) coefficients from the data without vibrato place both poles on the real axis, whereas the AR(2) coefficients from the data with vibrato places the poles as a conjugate pair. Averaging these two AR(2) coefficients yields a forward linear predictor that is neither especially good at predicting regions without vibrato nor regions with vibrato. One of the two sets must be chosen.

In Figure 3 the amplitude variation of the pitch track when vibrato was present is larger than without vibrato. This trend was observed throughout the data. This being the case, the AR(2) coefficients were computed using only the pitch tracks *that contain vibrato.* The argument being that if the filter successfully tracks the more volatile regions of the pitch then it should be able to track the less volatile regions reasonably well.

The final AR(2) coefficients are $\alpha_1 = -1.3895$ and $\alpha_2 = 0.4918$, which place poles at angle $\frac{\pi}{23}$ ($2.2Hz$ at $10ms$ sampling period), please see Figure 5.

## 3.3 Kalman formulation

The Kalman filter is an optimal linear MMSE recursive estimator [12, 25]. Given a model of a system's behavior and driving and observation noise processes, the Kalman filter uses successive observations to update the system's state. Specifically, the discrete-time, vector-state, scalar-observation Kalman filter assumes a process of the form,

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}u_k \\ y_k &= \mathbf{C}\mathbf{x}_k + v_k \end{aligned} \tag{5}$$

where $\mathbf{x}_k$ is the system state at time step $k$, $y_k$ is the observation at time step $k$, and $u_k$ and $v_k$ are the driving and observation noise processes, respectively. The $\mathbf{A}$ matrix is the state

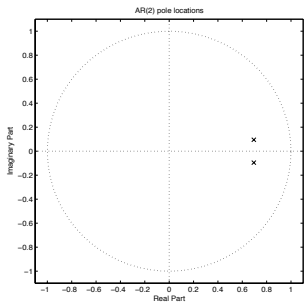


Figure 5: Pole locations for the autoregressive noise process.

transition matrix, which describes now the state vector $x_k$ evolves in the absence of external influence. The $\mathbf{B}$ matrix describes how the driving noise influences the state transitions. The $\mathbf{C}$ matrix describes how the observation $y_k$ is formed from the state vector $x_k$. The noise processes, $u_k$ and $v_k$, are both zero-mean white Gaussian noise processes with unit variance.

From (3) and (4) it is evident that the system state vector contains three elements,

$$\mathbf{x}_k = [\, p_k \quad w_k \quad w_{k-1} \,]^t \tag{6}$$

and the observation vector is simply the pitch track plus observation noise, $v_k$,

$$y_k = \tilde{p}_k \tag{7}$$

where,

$$\tilde{p}_k = p_k + v_k. \tag{8}$$

Combining (3) through (8) we have that,

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & \alpha_1 & \alpha_2 \\ 0 & 1 & 0 \end{bmatrix}, \tag{9}$$

$$\mathbf{B} = \begin{bmatrix} 0 \\ G \\ 0 \end{bmatrix}, \tag{10}$$

$$\mathbf{C} = [\, 1 \quad 0 \quad 0 \,]. \tag{11}$$

## 3.4 Kalman tracking

Having defined the Kalman filter parameters, the Kalman filter was run using the pitch tracks as the observation data $y_k$. Each successive pitch track value was compared to the pitch predicted by the Kalman filter. When the prediction matched the observation data the new sample was taken to be a continuation of an existing note. When the prediction did not match the observed data a new note was instantiated. The distance function for the vector-state, scalar-observation Kalman filter is given by [12, 25],

$$d_k^2 = \frac{e_k^2}{\mathbf{C}\mathbf{P}_k\mathbf{C}^t + R} \tag{12}$$

where $e_k = y_k - \mathbf{C}\hat{\mathbf{x}}_k$ is the error between the observation and predicted observation. The variance of the error $e_k$ is given by $\mathbf{C}\mathbf{P}_k\mathbf{C}^t + R$, where $R$ is the variance of the observation noise $v_k$ (taken to be unity) and $\mathbf{P}_k$ is the expected covariance of the system state $\mathbf{x}_k$ (given by the filter update equations). The distance $d_k$ is then compared to a threshold to determine whether a new note must be instantiated.

### 3.5 Kalman filter performance

A sample output of the Kalman filter is shown in Figure 6. As expected, the distance was typically small, but large near note transitions. Notice that not all note changes produced equal Kalman distances. Furthermore, a note change did not cause a single large distance value, but rather a cluster of large distances in the vicinity of the note change.

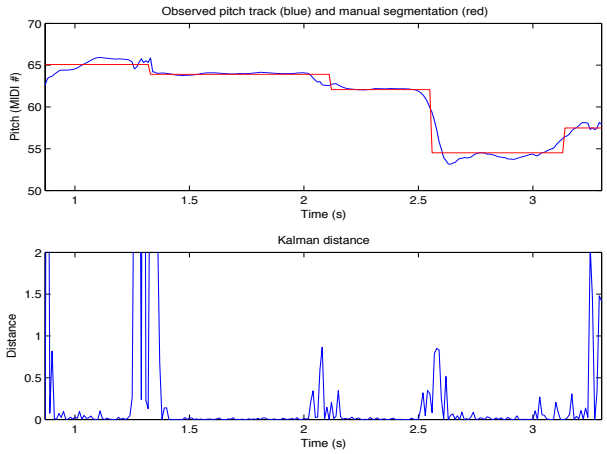


Figure 6: A sample output of the Kalman distance. The top plot shows the measured pitch track along with a manual segmentation. The bottom plot shows the corresponding Kalman distance.

To perform an automatic segmentation of a pitch track, the distance curve was compared to a threshold. A new note was inserted wherever the distance function exceeded the threshold. To prevent clusters of spurious notes a thinning procedure similar to that described later in section 4.4 was employed. Clearly, by adjusting the decision threshold there was a tradeoff between missed and inserted notes. This tradeoff can be seen in Figure 7. Note that the Kalman segmenter produced consistently higher detection rates at the same false alarm rate than either of the conventional segmenters.

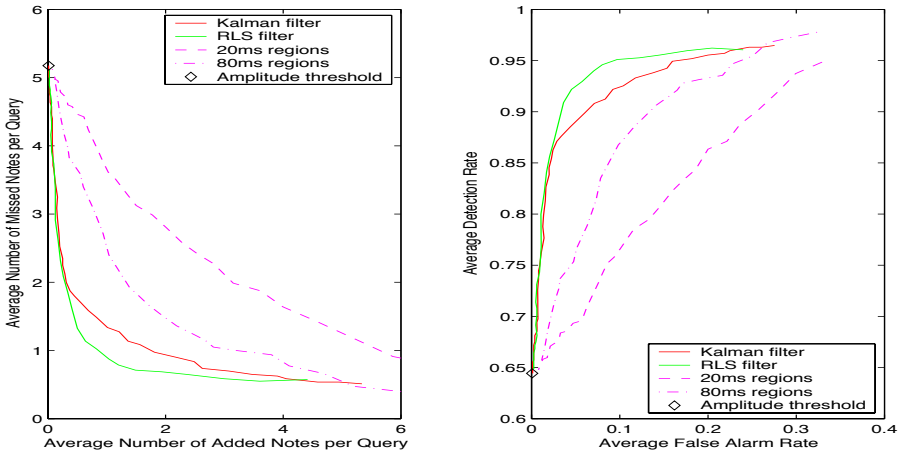


Figure 7: Performance curves demonstrating the tradeoff between missed and inserted notes for the predictive filter segmenters. The left plot gives the average number of missed and inserted notes per query, and the right plot gives the segmenter ROC.

### 3.6 RLS formulation

The Kalman filter formulation presented above is time-invariant. This is arguably not an appropriate formulation given the time-varying nature of the pitch tracks. Of particular concern is the inability of the above formulation to adapt to regions with and without vibrato. Had the pitch tracks not been so volatile, a stationary formulation would have been better justified.

Allowing the **A**, **B** and **C** matrices to be time-varying enables the filter to more accurately predict pitch tracks whose statistical properties change over time. A special case of this general Kalman filter is the recursive least squares (RLS) filter [10, 27].

RLS filters are especially attractive because they do not require precise assumptions about the behavior of the pitch tracks. No statistical analysis is required. The filter recursively updates itself at each time step to minimize the squared error between the predicted and actual pitch tracks.

In this work an RLS filter was implemented as a $P$-step predictor [27]. At time step $k$ the information vector is the most recent $N$ pitch track values,

$$\mathbf{x}_k = [\, p_k \; p_{k-1} \; \cdots \; p_{k-N+1} \,]^{\,t} \tag{13}$$

The desired, or observed, pitch track prediction is,

$$d_k = p_{k+P} \tag{14}$$

and the actual pitch track prediction is,

$$\hat{d}_k = \hat{p}_{k+P} = \mathbf{w}_k^t \, \mathbf{x}_k \tag{15}$$

where $\mathbf{w}_k$ is the adaptive filter coefficients. These coefficients are chosen so as to minimize the squared error through time step $k$,

$$J_k = \sum_{n=N+P-1}^{k-1} |d_n - \hat{d}_n|^2 \; = \; J_{k-1} \; + \; |d_{k-1} - \hat{d}_{k-1}|^2. \tag{16}$$

Note the sum begins at $n = N + P - 1$ instead of $n = 0$, this is due simply to indexing. The first pitch track value is taken to be $p_0$, as such the first predicted value is $\hat{p}_{N-P+1}$. Neglecting for the moment that $\mathbf{x}_k$ has an unknown non-zero-mean, the linear minimum mean square error coefficient vector is given by the well-known normal equation [10, 27],

$$\mathbf{w}_k = \mathbf{R}_k^{-1} \, \mathbf{c}_k \tag{17}$$

where $\mathbf{R}_k$ is the autocorrelation matrix at time step $k$ and $\mathbf{c}_k$ is the cross-correlation vector, numerically calculated by

$$\begin{aligned} \mathbf{R}_k &= \sum_{n=N-1}^{k-1} \mathbf{x}_n \mathbf{x}_n^t \; = \; \mathbf{R}_{k-1} \; + \; \mathbf{x}_{k-1} \, \mathbf{x}_{k-1}^t \\ \mathbf{c}_k &= \sum_{n=N-1}^{k-1} \mathbf{x}_n d_n \; = \; \mathbf{c}_{k-1} \; + \; \mathbf{x}_{k-1} \, d_{k-1} \end{aligned} \tag{18}$$

If unlimited computational power were available, (15) through (18) would be a sufficient description of the RLS algorithm. In practice, of course, the repeated calculation of $\mathbf{R}_k^{-1}$ is an inefficient use of computing power. From (18) it seems reasonable to postulate that just as there is a recursive formula for $\mathbf{R}_k$, there should exist a recursive formula for $\mathbf{R}_k^{-1}$. Such a formula can be found by making use of the matrix inversion lemma [10, 27],

$$(A + BCD)^{-1} = A^{-1} - A^{-1}B(DA^{-1}B + C^{-1})^{-1}DA^{-1}. \tag{19}$$

With the substitutions $A = \mathbf{R}_{k-1}$, $B = \mathbf{X}_k$, $C = 1$, $D = \mathbf{X}_k^t$ and a bit of algebra a recursive formula is found,

$$\mathbf{R}_{k+1}^{-1} \;=\; \mathbf{R}_k^{-1} \;-\; \frac{\mathbf{z}_k\,\mathbf{z}_k^t}{1+q_k} \tag{20}$$

where,

$$\begin{aligned} \mathbf{z}_k &= \mathbf{R}_k^{-1}\,\mathbf{x}_k \\ q_k &= \mathbf{x}_k^t\,\mathbf{z}_k. \end{aligned} \tag{21}$$

Substituting this result into (17) and a bit of algebra give us the filter coefficient update equation,

$$\begin{aligned} \mathbf{w}_{k+1} &= \left(\mathbf{R}_k^{-1} \;-\; \frac{\mathbf{z}_k\,\mathbf{z}_k^t}{1+q_k}\right)\cdot(\mathbf{c}_k \;+\; \mathbf{x}_k d_k) \\ &= \mathbf{w}_k + \frac{(d_k - \hat{d}_k)\cdot\mathbf{z}_k}{1+q_k}. \end{aligned} \tag{22}$$

A 'forgetting' factor, $0 < \rho < 1$, is frequently incorporated into the RLS filter to diminish the influence of 'older' data,

$$\begin{aligned} \mathbf{R}_k &= \sum_{n=N-1}^{k-1} \rho^{k-1-n}\,\mathbf{x}_n\mathbf{x}_n^t \\ \mathbf{c}_k &= \sum_{n=N-1}^{k-1} \rho^{k-1-n}\,\mathbf{x}_n d_n \end{aligned} \tag{23}$$

giving modified update equations,

$$\begin{aligned} \mathbf{R}_{k+1}^{-1} &= \frac{1}{\rho}\left(\mathbf{R}_k^{-1} \;-\; \frac{\mathbf{z}_k\,\mathbf{z}_k^t}{\rho+q_k}\right) \\ \mathbf{w}_{k+1} &= \mathbf{w}_k + \frac{(d_k - \hat{d}_k)\cdot\mathbf{z}_k}{\rho+q_k}. \end{aligned} \tag{24}$$

The only remaining question is how to initialize the filter, specifically $\mathbf{R}_{N-1}$ and $\mathbf{w}_{N-1}$. The autocorrelation matrix is typically initialized as,

$$\mathbf{R}_{N-1} = \eta\, I_N \tag{25}$$

where $\eta$ is a positive parameter. $\eta$ influences how quickly the filter converges to steady-state. Given that a new note is never instantiated within $120ms$ of a note boundary, the precise choice of $\eta$ was not important. The filter coefficients themselves were simply initialized by $\mathbf{w}_{N-1} = [1, 0\cdots 0]^t$.

## 3.7 RLS tracking

The above development leaves three relevant parameters that must be chosen; the filter length $N$, how many time steps to predict $P$ and the forgetting factor $\rho$. This three-dimensional parameter space was found to have numerous local minima and maxima (in terms of segmentation performance). Setting the ratio of $\frac{P}{N} \simeq \frac{2}{3}$ yielded consistently good results. Final parameter choices were found manually, $N = 5$, $P = 3$ and $\rho = 0.90$.

Having set the filter parameters the RLS filter is run using the pitch track as the information signal (13) and the desired signal (14). The squared prediction error $|d_k - \hat{d}_k|^2$ is compared to a threshold to determine whether a new note should be instantiated. The same thinning procedure used with the Kalman filter is again needed to reduce clusters of notes to a single new note.

### 3.8 RLS filter performance

A sample output of the RLS filter is shown in Figure 8. Similar to the Kalman filter, the error was typically small, but large near note transitions. The overall performance of the RLS segmenter is shown in Figure 7. As can be seen from the figure, the RLS and Kalman filters demonstrated equivalent performance in the low false-alarm domain. However for detection rates above 0.87, the RLS filter provided superior performance to the Kalman filter. This was due in part to the RLS filter's ability to adapt to different types of pitch track fluctuations, whereas the Kalman filter was restricted to tracking only one class of fluctuations (vibrato in this case).

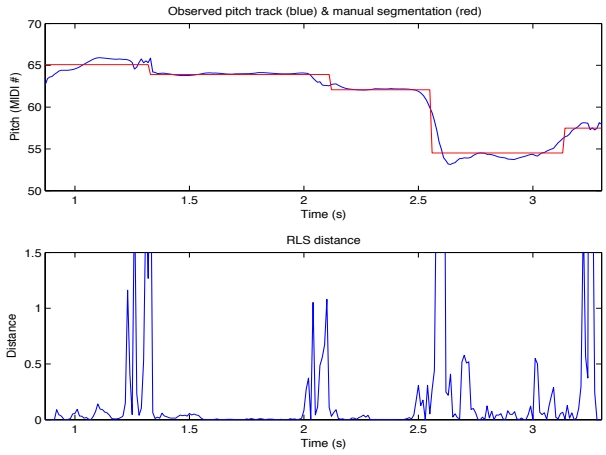


Figure 8: A sample output of the RLS distance. The top plot shows the measured pitch track along with a manual segmentation. The bottom plot shows the corresponding RLS distance.

# 4 Nonlinear LMS filtering

The previous chapter explored the use of two different predictive filters to detect note changes. Both nonparametric and parametric spectral estimation tools were employed to design a simple time-invariant model of the pitch track for use in a Kalman filter. The time-varying characteristics of the pitch track however limited performance. A predictive RLS filter was also implemented. Although the RLS filter yielded considerably better results, it did not make use of any modelling of the pitch track (other than it being relatively constant within a single note). In short, the Kalman filter assumed too much while the RLS filter did not assume enough.

A compact model of vocal production of melodic lines has yet to be developed. In spite of this, it is reasonable to postulate that incorporating as much information about the pitch tracks as possible into the segmentation algorithm should improve performance. The difficulty, of course, is how to incorporate this information when no analytic model is available.

Neural networks are frequently used in situations where no analytic model is available. These systems rely on the availability of training data to capture statistical characteristics of the signal. That is, neural networks allow the data to "speak for itself" [10].

The present work considers the simplest form of neural network, a single layer network or *neuron*. A single neuron consists of a linear combiner, with waits determined via a training routine, and a threshold function. Viewed in this way, a neuron is equivalent to a nonlinear least mean squares (LMS) filter. A LMS filter makes use of a reference signal (training data) to compute a linear weight vector that minimizes the square error between

the reference signal and the output of the filter. The addition of a threshold operator (a nonlinear operator) to the output of the LMS filter yields a NLMS filter [10].

## 4.1 Detection formulation

Rather than formulate the neural network as a filter, as was done in the previous chapter, it was decided to formulate the neuron in a detection framework. The pitch track was initially sliced into regions $80ms$ long (8 pitch track samples). Regions were treated independently and hence the problem was reformulated as a conventional simple-hypothesis detection problem.

The detection hypotheses are,

$$\begin{aligned} \mathcal{H}_0^k &: \text{A note-change has not occurred in region } k \\ \mathcal{H}_1^k &: \text{A note-change has occurred in region } k. \end{aligned} \tag{26}$$

To determine which of the hypotheses is more probable a feature vector is calculated for each region. A weighted sum of these elements is computed and the result is compared to a threshold. If the sum is greater than the threshold hypothesis $\mathcal{H}_1^k$ is chosen, otherwise hypothesis $\mathcal{H}_0^k$ is chosen.

## 4.2 Feature vector

The precise features to include in the feature vector were developed experimentally by exploring various types of discrete derivatives. The assumption was that an 'ideal' pitch track would be a piecewise constant function. That is, the singer transitions between pitches instantaneously and maintains each pitch without any deviation. If this had been the case only a single feature would have been needed, the discrete first-order difference. This simple derivative would have been nonzero if and only if a note change had occurred. Of course, this 'ideal' pitch track is grossly optimistic, transition times are substantial and pitches are never held constant. Clearly, a single first-difference feature is not sufficient. The final features chosen are presented next, with a cursory justification given after.

The pitch track was sliced into region of length 8 ($80ms$). Computing the feature vector for each region involves both the current region and the regions preceding and following the current region.

Let $\mathbf{p}^k$ be the eight-element vector representing the $k^{\text{th}}$ region of the pitch track. Further, let $p_j^k$ be the $j^{\text{th}}$ element of $\mathbf{p}^k$. Similarly, define $\mathbf{c}^k$ and $c_j^k$ for the autocorrelation track and $\mathbf{a}^k$ and $a_j^k$ for the amplitude track. Finally, let $\mathbf{f}^k$ be the feature vector for the $k^{\text{th}}$ region and $f_j^k$ be the $j^{\text{th}}$ element of $\mathbf{f}^k$. Numerous feature vectors were experimented with, some of which were quite long, and the ultimately a five-element feature vector was decided upon. For the $k^{\text{th}}$ region the feature vector is calculated by

$$\begin{aligned} \mathbf{f}_0^k &= \sqrt{|\overline{\mathbf{p}^{k+1}} - \overline{\mathbf{p}^{k-1}}|} \\ \mathbf{f}_1^k &= \max|\Delta\mathbf{p}^k| \\ \mathbf{f}_2^k &= \begin{cases} 1 & \text{if } 0.9 > \min \mathbf{c}^k \\ 0 & \text{if } 0.9 \leq \min \mathbf{c}^k \end{cases} \\ \mathbf{f}_3^k &= \max(0, \Delta\mathbf{c}^k) \\ \mathbf{f}_4^k &= \sqrt{\max(0, \overline{\mathbf{a}^{k+1}} - \overline{\mathbf{a}^{k-1}})} \end{aligned} \tag{27}$$

where $\overline{\mathbf{p}^k}$ and $\overline{\mathbf{a}^k}$ are the mean of $\mathbf{p}^k$ and $\mathbf{a}^k$, respectively. $\Delta\mathbf{p}^k$ and $\Delta\mathbf{c}^k$ are the conventional discrete derivatives of $\mathbf{p}^k$ and $\mathbf{c}^k$. That is,

$$\Delta p_j^k = \begin{cases} p_{j+1}^k - p_j^k & \text{if } 0 \leq j \leq 6 \\ p_0^{k+1} - p_7^k & \text{if } j = 7 \end{cases} \tag{28}$$

The features are arranged roughly in order of decreasing importance. $f_0^k$ is far and away the most important feature. This feature represents a compromise between a derivative (to indicate change) and a low-pass filter (to smooth out noise etc.). Using an averaging length of 8 frames and a gap of 8 frames was found experimentally to be a good value. The square root operation is included in $f_0^k$ because we do not want to favor large note-changes, transitions greater than one semitone are pulled back towards one by the square root operation. In hindsight, this feature can be further justified by considering the impulse response that the first feature represents. The frequency response of this impulse is shown in Figure 9. As can be seen from the figure, the amplitude response is maximum at frequency $2.9Hz$, which happens to be the average note rate within continuous passages.

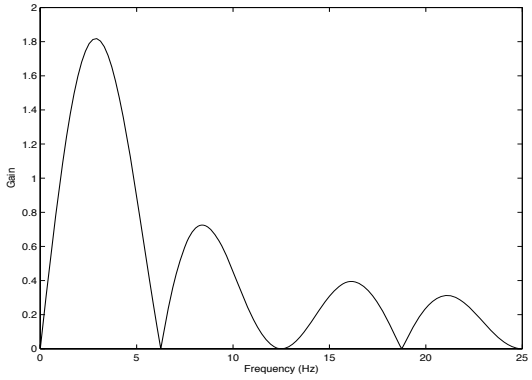


Figure 9: Amplitude response for the first NLMS feature.

Features $f_1^k$ and $f_3^k$ indicate the maximum derivative within each region, the idea being that if the region contains large instantaneous derivatives there is a higher probability that the region contains a note change. It was also observed that during note changes frequently the autocorrelation track would drop momentarily, indicating less confidence in the pitch estimate. This observation is incorporated into feature $f_2^k$, which uses a hard threshold to indicate when the pitch track algorithm has less confidence in its pitch estimate.

Sample pitch and autocorrelation tracks are given in Figure 10. The amplitude track was excluded from the figure, but similar trends are observed, although typically not as distinctly. The plots on the left give an example of an interval when a note change has *not* occurred, and the corresponding feature vector, $\mathbf{f}^0$, is given below. Similarly, the plots on the right give an example of a note change, with corresponding feature vector $\mathbf{f}^1$ given below. In this figure the differences between the two sets of curves are obvious. This is not typical, the differences are usually more subtle.

$$\mathbf{f}^0 = \begin{bmatrix} 0.2799 \\ 0.1752 \\ 0 \\ 0.0082 \\ 0 \end{bmatrix} \qquad \mathbf{f}^1 = \begin{bmatrix} 1.4058 \\ 0.8086 \\ 1 \\ 0.1038 \\ 0.0930 \end{bmatrix} \tag{29}$$

Before proceeding with the scalar decision statistic was computed from this vector, it is useful to consider the correlation between elements of the feature vector. The following correlation matrix was computed numerically.

$$\begin{bmatrix} 1 & 0.471 & 0.400 & 0.329 & 0.268 \\ 0.471 & 1 & 0.567 & 0.627 & 0.195 \\ 0.400 & 0.567 & 1 & 0.722 & 0.183 \\ 0.329 & 0.627 & 0.722 & 1 & 0.237 \\ 0.268 & 0.195 & 0.183 & 0.237 & 1 \end{bmatrix} \tag{30}$$

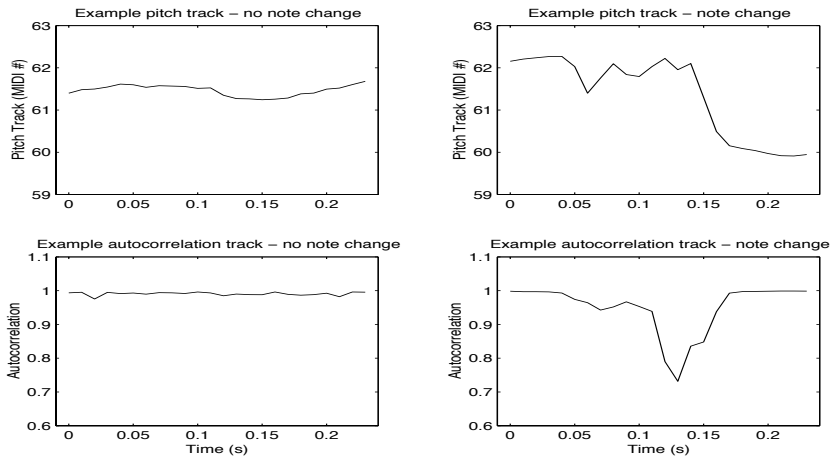


Figure 10: Sample pitch and autocorrelation tracks, the plots on the left give an example of the tracks when a note change has *not* occurred and the tracks on the right give an example of the tracks when a note change has occurred.

As can be seen from the matrix, the elements of the feature vector are somewhat correlated with each other. More importantly, this shows that the elements are not at all independent (had they been independent the correlation matrix would have been an identity matrix).

## 4.3 Decision statistic

A scalar decision statistic, $s^k$, is then calculated from each feature vector by taking a linear combination of the elements,

$$s^k = \mathbf{w}^t \cdot \mathbf{f}^k = \sum_{j=0}^{4} w_j \cdot f_j^k \tag{31}$$

where $\mathbf{w} = [w_0, w_1, w_2, w_3, w_4]^t$ is the feature vector. The decision is then made by simply comparing $s^k$ to a threshold $\delta$,

$$\begin{aligned} s^k < \delta \quad &: \quad \text{decide } \mathcal{H}_0^k \\ s^k \geq \delta \quad &: \quad \text{decide } \mathcal{H}_1^k. \end{aligned} \tag{32}$$

The weights, $\mathbf{w}$, used in the above summation were computed using a *perceptron training rule*. A *perceptron* is a single layer (i.e. linear) neural network whose weights are computed using training data to minimize the the probability of error when choosing between $\mathcal{H}_0$ and $\mathcal{H}_1$ [29].

A perceptron is ideally suited for classifying vectors that are linearly separable, in which case the weights can be computed to result in no errors with a finite number of training cycles, or epochs [29]. The feature vectors in the training data are *not* linearly separable and hence the perceptron training cannot return zero error. Furthermore, there is no guarantee that the training will converge to the globally optimal solution.

Another assumption made by the perceptron training rule is that the elements of the feature vector are statistically independent. As shown in the previous section, the elements are *not* independent here. This too implies that the perceptron may be unable to find the globally optimal weighting. Nonetheless, it is still a pragmatic solution for determining reasonable weights.

If $\mathcal{H}_0$ and $\mathcal{H}_1$ had been equally likely and misses and false-alarms had been equally undesirable, the perceptron training rule would have been given by

$$\mathbf{w} = \mathbf{w} + v \cdot (t - a) \cdot \mathbf{f}^k, \tag{33}$$

where $t$ is the true value the detector should have chosen ($\mathcal{H}_0 : t = 0$ and $\mathcal{H}_1 : t = 1$) and $a$ is the value actually chosen with the current weight vector $\mathbf{w}$. $0 < v \leq 1$ is the training rate and controls the tradeoff between how quickly the weights converge and the variance of the final weights.

The hypotheses $\mathcal{H}_0$ and $\mathcal{H}_1$ are not equally likely however, the probability of $\mathcal{H}_0$ being much greater than that of $\mathcal{H}_1$ (any given $80ms$ region of pitch track will probably not contain a note change). Further, a miss event is considered more undesirable than a false-alarm. Therefore the training rule was modified to be asymmetric,

$$\mathbf{w} = \begin{cases} \mathbf{w} + v_{\mathrm{m}} \cdot \mathbf{f}^k, & \text{if } t - a = 1 \\ \mathbf{w} - v_{\mathrm{fa}} \cdot \mathbf{f}^k, & \text{if } a - t = 1 \\ \mathbf{w}, & \text{if } a = t \end{cases} \tag{34}$$

Reasonable training rates were determined experimentally, $v_{\mathrm{m}} = 0.01$ and $v_{\mathrm{fa}} = 0.0005$. These training rates resulted in relatively slow convergence. For training, 447 feature vectors were selected, of which 67 contained note change events. These feature vectors were taken from 30 of the files that had been manually segmented. The perceptron converged after several dozen epochs to the following weights,

$$\mathbf{w} = \begin{bmatrix} 0.8318 \\ 0.2105 \\ 0.0690 \\ 0.0926 \\ 0.0983 \end{bmatrix} \tag{35}$$

## 4.4 False alarm thinning

The primary objective of the segmentation algorithm presented here is to reduce the miss rate of conventional amplitude-threshold methods. The segmentation algorithm described above does in fact reduce the miss rate substantially, but at the cost of increasing the false-alarm rate.

The large increase in the false alarm rate is a result of an implicit assumption made when reformulating the segmentation problem as a detection problem. The detection scenario assumes that consecutive feature vectors are independent of each other. This is not the case here, there is a high degree of correlation between consecutive feature vectors. In particular, when a note change occurs not only does the decision statistic containing the note change exceed the threshold, but so do some of the neighboring decision statistics. That is, the false alarms typically occur in clusters centered around a single legitimate note change.

This problem is reminiscent of a problem in image detection. Similar to this segmentation problem, the edge detection problem is an ill-defined problem. Conventional edge detection techniques involve discrete derivatives analogous to those used in this work. False alarm edges often occur in clusters centered around a single legitimate edge. Because of this, numerous edge *thinning* or *clustering* procedures have been developed for various scenarios [11, 17].

Two simple thinning, or clustering, procedures were implemented in this work. The first thinning procedure worked as follows. Consecutive note changes were first located. Within these regions, the segment with the highest confidence (the largest value of $s^k$) was retained and one segment to the left and right was discarded. This procedure was repeated until all consecutive note changes were removed. The second clustering procedure searched for notes less than $150ms$ in duration and merged them with the note to the left or right depending upon which was closer in pitch.

These two procedures, while ad hoc, were found to remove many of the false-alarms while only increasing the miss rate slightly.

After a note change has survived the thinning process, the precise location within each $80ms$ region is determined. The location of the maximum derivative of the pitch track defines the precise segment location within each region, $\arg_j \max |\Delta \mathbf{p}^k|$.

### 4.5 Performance

The overall performance of the NLMS segmenter is shown in Figure 11. As with the predictive filters, the NLMS segmenter consistently performed better than the conventional methods.

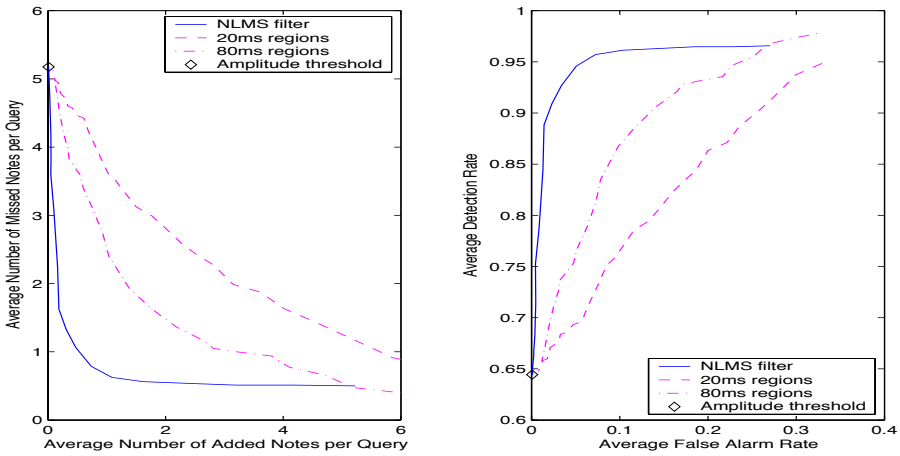


Figure 11: Performance curves for the NLMS segmenter. The left plot is the average number of missed and inserted notes per query, and the right plot gives the segmenter ROC.

Before proceeding with the final segmentation method considered in this work, a comparison of the NLMS with and without thinning is instructive. Such a comparison is shown in Figure 12. Two ROC curves are shown; the blue curve represents the segmenter using both note thinning routines, the green curve represents the segmenter using only the first of the two thinning routines described in the previous section. Overall, the use of both thinning procedures yields better performance. However, the second thinning procedure prevents the segmenter from ever recognizing short notes (of duration less than $150ms$), hence the apparent asymptote in the ROC curve for the NLMS segmenter with both thinning routines. To date, all segmentation algorithms used in MIR systems produce no false alarms, but often many missed notes. This being the case, little work has been done comparing the performance decreases due to missed versus inserted notes. Intuitively one might assume that both errors are equally undesirable. However, should it turn out that missed notes are more detrimental to MIR performance than inserted notes, the second thinning procedure may be counterproductive.

## 5 Curve fitting

Throughout this work the assumption that an 'ideal' pitch track would simply be a piecewise constant function has been critical. This intuitive, if not dubious, assumption motivates the question: why not segment the pitch track by fitting it to a piecewise constant function? The use of curve fitting to perform segmentation is explored in this chapter.

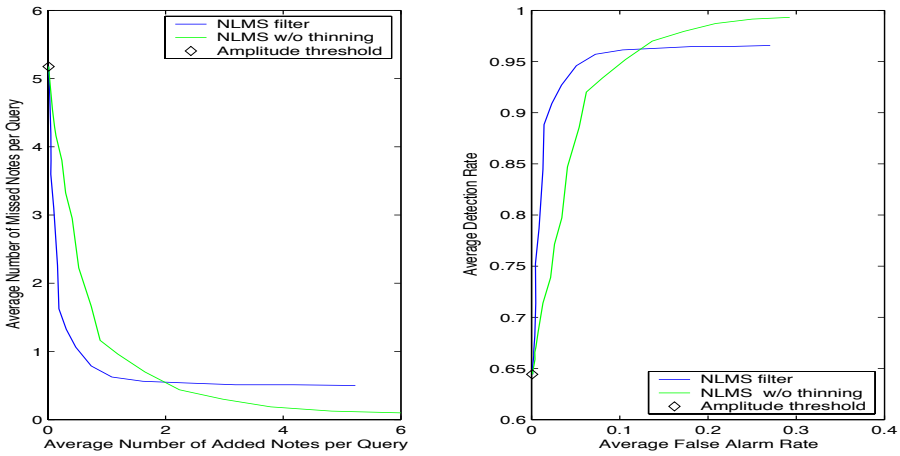


Figure 12: Performance curves for the NLMS segmenter with and without false-alarm thinning.

## 5.1 Problem formulation

Consider an arbitrary pitch track, $\mathbf{p} = [p_0, p_1, \cdots p_{N-1}]^t$ and piecewise constant signal $\mathbf{s} = [s_0, s_1, \cdots s_{N-1}]^t$ containing $M$ constant portions,

$$s_k = \begin{cases} a_0 & k = 0, 1, \cdots, k_1 - 1 \\ a_1 & k = k_1, k_1 + 1, \cdots, k_2 - 1 \\ \vdots & \\ a_{M-1} & k = k_{M-1}, k_{M-1} + 1, \cdots, N - 1. \end{cases} \tag{36}$$

The vector $\mathbf{a} = [a_0, a_1, \cdots a_{M-1}]^t$ defines the amplitude of each constant portion and $\mathbf{k} = [k_0, k_1, cdots\, k_{M-1}, k_M]^t$ define the boundaries between constant portions, where $k_0 = 0$ and $k_M = N - 1$. Taken together, the $2M - 1$ nontrivial parameters contained in the vectors $a$ and $k$ define the piecewise constant signal $\mathbf{s}$.

If the pitch track $p$ is *known* to contain $M$ notes, then the $(2M - 1)$-dimensional parameter space of all possible piecewise constant signals must be searched to find the solution that minimizes the error between $\mathbf{p}$ and $\mathbf{s}$. If the conventional magnitude-squared error metric is used, the error function is given by [13].

$$J(\mathbf{p}, \mathbf{s}) = (\mathbf{p} - \mathbf{s})^t \cdot (\mathbf{p} - \mathbf{s}) = \sum_{m=0}^{M-1} \sum_{k=k_m}^{k_{m+1}-1} (p_k - a_m)^2. \tag{37}$$

Using a squared error metric, for a given $\mathbf{k}$ the (MLE) amplitude estimates of each constant portion are computed as the *mean* of the pitch track in each portion [13],

$$\hat{a}_m = \frac{1}{k_{m+1} - k_m} \cdot \sum_{k=k_m}^{k_{m+1}-1} p_k. \tag{38}$$

It was found experimentally however that an absolute error metric yielded better segmentation accuracy, in which case the error function is given by,

$$J(\mathbf{p}, \mathbf{s}) = \sum_{m=0}^{M-1} \sum_{k=k_m}^{k_{m+1}-1} |p_k - a_m|, \tag{39}$$

and the (MLE) amplitude estimates are given by the *median* of each portion [12],

$$\hat{a}_m = \text{median}(p_k : k_m \leq k < k_{m+1}). \tag{40}$$

## 5.2 Dynamic programming

As was shown in the previous section, if the boundary locations were known, estimating the amplitude values would be trivial. The difficulty was in estimating the boundary locations. If a brute force search was used the computational complexity would be $O(N^M)$ [13] (again, assuming the number of notes $M$ was known) . Although computational complexity was not the primary focus of this work, the brute force search was far too slow for experimentation and simulation. To reduce the computational load to a more manageable level a search algorithm that makes use of a technique known as *dynamic programming* was employed.

Dynamic programming is used frequently in signal processing applications, such as Viterbi decoding, to reduce redundant computations. As applied to the melody segmentation problem, the fundamental idea is as follows. Suppose a note boundary is located at $L+1$ $(0 < L < N-2)$, and that there are $M_1$ notes prior to time step $L+1$ and $M_2$ notes following time step $L+1$ $(M_1+M_2 = M)$. If a $M_1-1$ segmentation that minimizes error for $k = 0, \cdots L$ is found, and a $M_2-1$ segmentation that minimizes error for $k = L+1, \cdots N-1$ is found, then the segmentation that minimizes the total error is simply the concatenation of the two partial segmentations.

Through the use of this technique the computational complexity is reduced to $O(NM)$. The primary components of the algorithm are outlined below. A more complete description can be found in [13].

Define the error for the constant region bounded by $k_m$ and $k_{m+1}$ as,

$$\Delta_m[k_m,\, k_{m+1}-1] = \sum_{k=k_m}^{k_{m+1}-1} |p_k - \hat{a}_m| \tag{41}$$

where $\hat{a}_m$ is computed by (40). The minimization of $J(\mathbf{p},\mathbf{s})$ over $\mathbf{a}$ *and* $\mathbf{k}$ is equivalent to the minimization of,

$$J(\mathbf{p},\mathbf{k}) = \sum_{m=0}^{M-1} \Delta_m[k_m,\, k_{m+1}-1] \tag{42}$$

over *only* $\mathbf{k}$. Suppose the $n^{\text{th}}$ boundary is located at time step $k = L+1$, then define the minimized error through to time step L as,

$$J_n[L] = \min_{k_1,k_2,\cdots k_{n-1}} \sum_{m=0}^{n-1} \Delta_m[k_m,\, k_{m+1}-1] \tag{43}$$

where $k_0 = 0$ and $k_n = L+1$. Note that when the optimal $\mathbf{k}$ has been found, $J_{M-1}[N-1] = J(\mathbf{p},\mathbf{k})$. A recursive formula for $J_n[L]$ can be easily obtained by noting,

$$\begin{aligned} J_n[L] &= \min_{k_{n-1}} \min_{k_1,k_2,\cdots k_{n-2}} \sum_{m=0}^{n-1} \Delta_m[k_m,\, k_{m+1}-1] \\ &= \min_{k_{n-1}} \left[ \left( \min_{k_1,k_2,\cdots k_{n-2}} \sum_{m=0}^{n-2} \Delta_m[k_m,\, k_{m+1}-1] \right) + \Delta_{n-1}[k_{n-1},\, k_n-1] \right] \\ &= \min_{k_{n-1}} \left( J_{n-1}[k_{n-1}-1] + \Delta_{n-1}[k_{n-1},\, k_n-1] \right). \end{aligned} \tag{44}$$

The complete curve fitting algorithm then works as follows. Compute $J_0[L]$ for all $L = 0, 1, \cdots N-M$. Compute $J_1[L]$ for all $L = 1, 2, \cdots N-M+1$, and store the value $k_1[L]$ that minimizes $J_1[L]$ for each $L$. Repeat this process through to $J_{M-2}[L]$, storing the values $k_n[L]$ for all $n$ and $L$. And lastly, compute $J_{M-1}[N-1]$ and store the minimizing value $k_{M-1}[N-1]$. The optimal partition, or segmentation, is then found via the backward

recursion,

$$\begin{aligned}
\hat{k}_{M-1} &= k_{M-1}[N-1] \\
\hat{k}_{M-2} &= k_{M-2}[\hat{k}_{M-1}] \\
&\vdots \\
\hat{k}_2 &= k_2[\hat{k}_3] \\
\hat{k}_1 &= k_1[\hat{k}_2]. \qquad (45)
\end{aligned}$$

It should be noted that while the so-called 'greedy' algorithm described above does keep the computational complexity manageable, this algorithm is still far slower than the other methods explored in this work.

## 5.3 Note number estimation

The development presented in the previous section made one critical assumption, that the number of notes the pitch track represented was known. This assumption is, of course, false. The number of notes the pitch track represents must be estimated, perhaps using a battery of generalized likelihood ratio tests (GLRT) [13]. Unfortunately, such tests require an analytic model of pitch track behavior. No such model has been developed. While numerical measurements could be made to compute the necessary probabilities, time constraints prohibited such explorations.

Clearly, as the number of hypothesized notes in the pitch track increases, the final error of the optimal piecewise constant curve fit will decrease. It is not unreasonable to postulate that as the number of hypothesized notes is increased beyond the true number of notes, the rate of decrease in the error will itself decrease. In other words, a plot of the minimum curve fit error versus the number of hypothesized notes would demonstrate a sharp 'knee'. Unfortunately, this was not found to be the case. An example is shown in Figure 13.

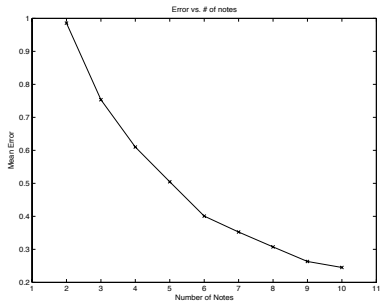


Figure 13: An example plot showing the minimum curve fit error versus the number of hypothesized notes. In this case the pitch track represented six notes.

In this work a more rudimentary method was used to approximate the number of notes in a given pitch track: simply multiply the duration of the pitch track (in seconds) by a scalar number (representing the expected number of notes per second) and round up. Use this number as the hypothesized number of notes. By adjusting the expected number of notes per second inserted notes can be traded for missed notes, hence an ROC curve can still be generated as in previous chapters. A false-alarm thinning routine similar to that described in section 4.4 is then used to remove spurious clusters of notes.

It should be noted that in the development presented in section 5.2, no restrictions regarding minimum note lengths were made. Such a restriction could easily be incorporated into the algorithm. Initially such a restriction was included, with the idea that there is no

point in searching for short notes given that they would be removed by the thinning routine anyway. This was a counterproductive assumption in hindsight. By letting the algorithm include short notes (which would then be removed by the thinning routine) diminished the impact of the mediocre note number estimates.

## 5.4 Performance

The overall performance of the curve fitting segmenter is shown in Figure 14. As will be further discussed in the final chapter, the curve fitting segmenter does not perform as well as the RLS and NLMS segmenters. It should be noted, that the performance of the curve fitting segmenter was greatly improved if the algorithm knew *a priori* the true number of notes (and no thinning was performed).

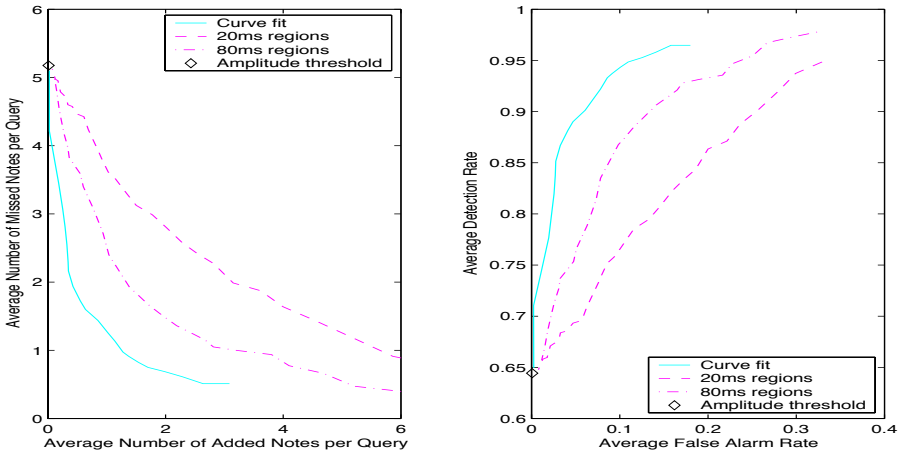


Figure 14: Performance curves for the curve fitting segmenter. The left plot is the average number of missed and inserted notes per query, and the right plot gives the segmenter ROC.

## 5.5 Alternate representations

The uncertainty regarding the number of notes a given pitch track corresponds to presents an intriguing possibility. Perhaps a single estimate of the number of notes should *not* be made, but rather use a family of possible solutions (i.e. transcriptions), each based on a different number of estimated notes, to search the MIR database.

As will be discussed in the final chapter, the most commonly missed notes were 'passing' notes. The most commonly inserted notes were inserted in regions where the singer gradually 'scooped' between two pitches. In fact, there is no clear distinction between passing notes and prolonged scooping regions. Often, the author had difficulty deciding between a passing note and a scooping region when manually segmenting pitch tracks. This implies that it is unreasonable to expect an automatic segmentation algorithm to reliably distinguish between the two.

An interesting property of the curve fitting algorithm was that if the algorithm were first run searching for a two note partition, and then a three note partition, the first boundary found would be retained and second boundary added. In a sense, the first boundary found was more 'reliable' (or 'important') than the second.

An example of this phenomena is shown in Figure 15. In all four plots the blue curve represents the measured pitch track and the red curve represents the manual segmentation. The top plot shows the segmentation that results from assuming the pitch track represents only two notes, the second plot shows the segmentation from three notes. The true number of notes in this pitch track is six, and the segmentation resulting from this number of assumed

notes is second from the bottom. The bottom plot shows the segmentation resulting from ten assumed notes.

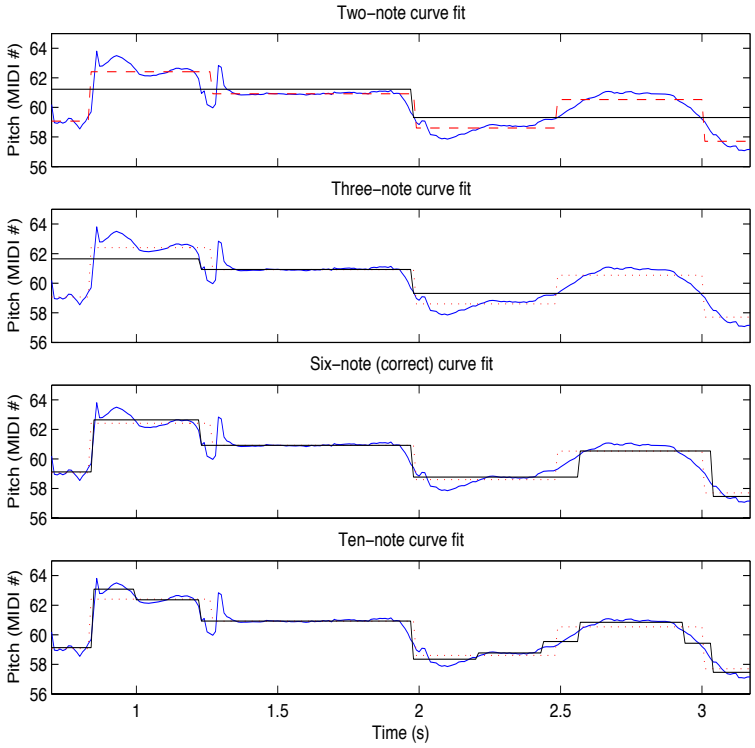


Figure 15: Sample curve fits for a pitch track generated "Rock a bye baby". In all plots, the blue curve is the measured pitch track, the red curve is the manual segmentation and the black curve is the automatic segmentation.

Given that repeatedly running the algorithm for an increasing number of notes never changed the boundaries already found implied that the algorithm could be run once for a large number of notes and then successively remove the boundary the resulted in the smallest increase error. Strictly speaking, this method potentially yields suboptimal curve fits.

We now have a method or ranking note boundaries. This motivates some possible alternate representation of a sung query that may be more 'robust' to sung imperfections. Two possible representations are described below.

A binary tree could be built. Please see Figure 16 for an example tree, generated from the same pitch track as that used in Figure 15. The root node represents the first, most reliable, note boundary. Four numbers are contained in every node: the boundary rank (the root node is the first boundary), the location of this boundary in the pitch track and the pitch values on either side of the boundary.

When a new node is added, the tree must be searched. Suppose the location of the new boundary is time step $L$. Beginning at the root node, check if $L$ is greater or less than the location of the root boundary. If it is less, move to the left child of the node, if it is greater move to the right child of the node. Repeat this process until an open space is found and add the new node. At any point during the tree's construction, the 'current' piecewise constant pitch track can be found by simply traversing the tree from left to right.

Roughly speaking, the more 'reliable' note boundaries are located near the top of the tree. The leaves of the tree represent the least 'reliable' note boundaries. Traversing the tree from the root node to any leaf yields a list of successively less 'reliable' note boundaries.

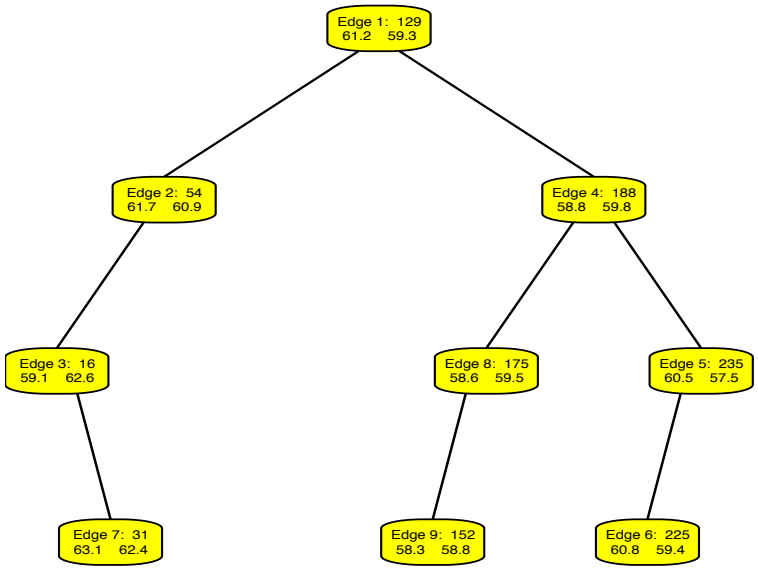


Figure 16: An example tree decomposition of a sung melodic passage from "Rock a bye baby". Each node represents a note boundary. Each node is labelled with the order in which it was created, the location of the boundary it represent and the pitches of the notes to its left and right.

Unfortunately, nodes that do not lie on the same root-leaf path cannot be compared. Referring to Figure 16, the third edge is lower than the fourth edge, even though the third edge was generated before the fourth (and hence is more 'reliable'). Therefore, the tree cannot be truncated below some level and hope the notes remaining were the most 'reliable'.

This difficulty can be overcome by computing a *dendrogram*. Dendrograms are a popular tool in the data clustering community [11]. A dendrogram is formed by clustering data (in this case the pitch track) into successively fewer classes based on some error measure. Figure 17 provides a sample dendrogram generated from the same pitch track as before. It should be noted that Figure 17 is not a typical visualization of a dendrogram. In particular, only the final, or most clustered nine layers of the dendrogram are shown. Each horizontal layer in Figure 17 represents a piecewise constant curve fit to the pitch track shown in Figure 15. Progressing down the dendrogram corresponds to adding more and more notes to the curve fit. In principle the decomposition could be continued until each pitch track sample represented a separate note. Lighter colored regions correspond to higher notes.

While the tree and dendrogram decompositions just described present some intriguing possibilities, comparing two trees or dendrograms is more complicated than comparing two transcriptions. It was found experimentally that the trees or dendrograms generated from two different pitch tracks representing the same melody often had different structures. For example, one tree might be heavily biased to the left and the other might be well-balanced, and yet they both represent the same melody. Resolving such difficulties was beyond the scope of this work.

# 6 Conclusion

## 6.1 Results

A complete set of ROC curves for all the techniques explored in this work is shown in Figure 18. A few points should be noted while comparing the performance of the various techniques. First, *all* of the techniques explored here appear to perform better than the

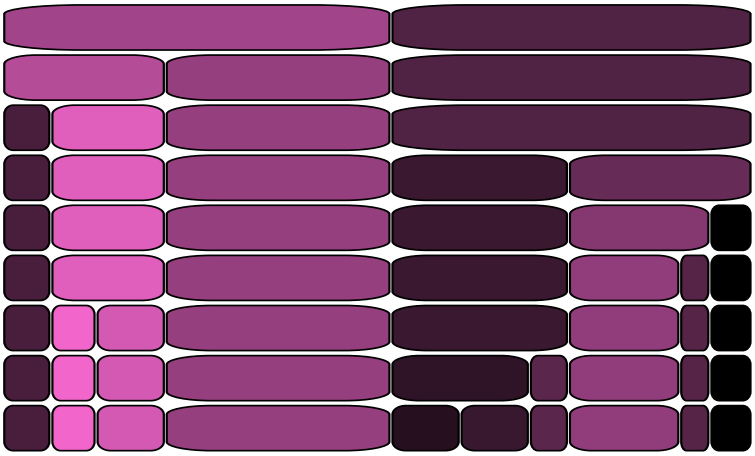

Figure 17: An example dendrogram decomposition of a sung melodic passage from "Rock a bye baby". Each horizontal layer represents the optimal piecewise constant curve fit based on fixed number of notes. Progressing from top to bottom, one note is added at each layer. The lighter regions represent higher notes.

conventional amplitude threshold and pitch derivatives described in section 2.3. While the methods explored here are more sophisticated and do offer some performance improvement to the conventional techniques, the degree of their superiority is potentially overstated. In particular, no post-processing of any kind was performed on the segmentations computed by the pitch derivative methods, including thinning. Had a thinning routine been developed for these methods, their ROC curves would have improved.

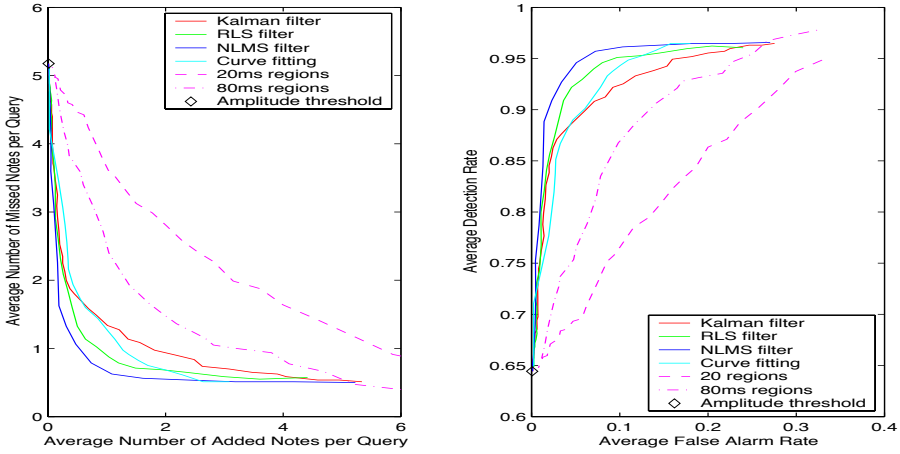


Figure 18: The left plot shows the average number of inserted notes per query vs. the average number of missed notes per query. The right plot shows the corresponding ROC curves.

An initial comparison of the ROC curves seems to imply that the NLMS filter gave the best performance. This is perhaps misleading. The NLMS filter had two advantages over the other methods that may not be present in a legitimate MIR system. First, the weight vector $\mathbf{w}$ described in section 4.3 was trained on a subset of the testing data. That is, 80 files were manually segmented. All 80 files were used in generating the ROC curves. 30 of these same files were also used in training the weight vector. Second, the features themselves were developed manually using the same data set. For example, the square root

operation in the first feature of (27) was included because it was found *experimentally* to improve performance. The NLMS filter was developed and tested on the same, relatively small, data set. The other techniques explored here did not make such direct use of the data set during development. Therefore, it is reasonable to suspect that performance of the NLMS filter will suffer more than the other methods when tested on a different or expanded data set. With this in mind, the RLS filter may offer the most reliable overall performance.

By adjusting the decision threshold, both the RLS and NLMS segmenters could be set so as to give an average of less than one inserted and one deleted note per query. While even a single inserted or deleted note can be enough to yield incorrect MIR results, this is still better than current continuous melody segmenters. All four techniques explored here could be set to produce less than 1.5 inserted and deleted notes per query.

Note that none of the techniques explored here are able to exceed a 0.95 detection rate. This is due to the thinning procedure described in section 4.4, which prevents especially short notes from being detected.

Before proceeding, recall that the ROC curves were computed numerically. The small ripples evident in all curves are a result of the relatively small database used for testing. A larger database would have further reduced these ripples.

For further analysis we resort to subjective evaluation. Figure 19 gives a sample output of the NLMS filter segmenter. This is an example of the tune "Rock a bye baby

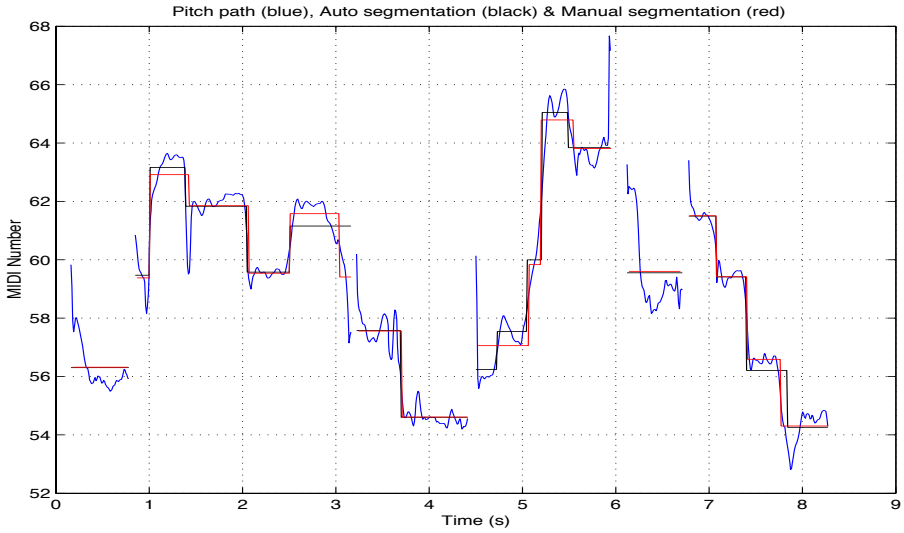


Figure 19: A sample output of the NLMS segmentater using both thinning procedures. The passage sung is "rock a bye baby in the tree top, when the wind blows the cradle will drop". The blue curve is the measured pitch track. The black curve is the automatic segmentation. And the red curve is the manual segmentation.

in the tree top, when the wind blows the cradle will drop." The first multiple-note region corresponds to "a bye baby in the", the second to "tree top", the third to "when the wind blows" and the forth "cradle will drop". Note, the singer has actually sung the "in the" incorrectly, but this is immaterial to the present analysis. This is a particularly good example because it demonstrates both types of errors while still giving a reasonable indication of average performance.

As can be seen in the figure, the automatic segmentation matches the manual segmentation reasonably well. There is a single missed note, "the" in "baby in the" and a single inserted note, "when" in "when the wind".

Listening to the audio file reveals that the missed "the" is sung very briefly, the singer does not clearly articulate the note. This is the most common type of missed note.

In particular, all the segmentation methods had difficulty detecting 'passing' notes. Passing notes are nestled between two important notes. Often a passing note would manifest itself as a continuous slide between two sustained pitches in the pitch track, there would be no plateau at all. Furthermore, the length and shape of this slide was not (visually) any different than when the singer was 'scooping' between two notes. While this does reveal an interesting question in the human perception of notes, for the time being it represents a reliable source of error in the segmentation algorithms. All of the systems developed here can be adjusted so as to usually detect passing notes or ignore scooping regions, but not both.

The inserted note, "when", is also an instance of the singer not clearly articulating the note. The singer lilts the pitch, causing the sporadic output seen in the pitch track. In fact, on this particular recording the singer falls into a short giggle shortly after with "blows the". Rather than viewing this as an error on the part of the singer, it is best to view this as another obstacle to designing a robust segmentation algorithm.

These results may appear reasonably good on the surface, but whether they are robust enough for subsequent database searching remains unclear. Depending upon how the MIR search is implemented, a *single* miss or false alarm could seriously degrade system performance.

## 6.2 Discussion

The recording analyzed in the previous section brings up an important issue. We do not want to design an MIR system that only works for trained singers. An untrained singer will inevitably make numerous small mistakes when singing a tune. Typically these errors are not as bad as actually giggling during the recording, but at the very least some notes will not be clearly articulated.

In particular, passing notes are frequently missed. The algorithms described here have the flexibility to be tuned so as to not miss these notes, however it comes at the cost of substantially increasing the false alarm rate. As it stands, it is better to miss the occasional passing note than insert many extra false alarm notes. This implies that the ultimate search engine used to match the query notes with the database must have some mechanism for assigning lower cost to these passing notes.

Similarly, that the precise location of the note boundaries is often ill-defined, even when performing manual segmentation, implies that the MIR search must not be sensitive to slight timing errors.

But returning to the segmentation problem itself, each of the methods explored here open particular doors for further improvement. All of the techniques explored here make use of a rudimentary 'minimum duration' note thinning procedure. Data clustering is a vast field of research with many elegant solutions for various applications. More sophisticated thinning or clustering algorithms could yield performance improvements for all the techniques explored in this work.

The NLMS segmenter, being relatively *ad hoc*, presents the fewest possibilities for further improvement. As with any system based on ad hoc fiddling, further fiddling will eventually reach a point of diminishing returns. While further epsilon improvements probably do exist, they will not be of particular interest to the research community. Although it is worth noting that the feature vectors explored here only consider a variety of discrete first derivatives. Higher order derivatives were not considered. Second derivatives could provide useful statistics not employed by the current system.

Both of the predictive filter techniques subtract a running sample mean from the pitch track before processing the pitch track values. Such a step is necessary because the Kalman and RLS filters considered here were developed assuming zero-mean processes. Simply subtracting a running sample mean potentially distorts important characteristics of the pitch track. A less distorting method of forcing the tracks to zero-mean could improve performance considerably.

The Kalman filter developed in Chapter 3 assumed a pitch track modulated by vibrato. There are, of course, many other types of 'distortion' an otherwise constant pitch track may experience. Designing the filter under a vibrato-corrupted assumption obviously limits the filters performance when no vibrato is present. This implies that perhaps two, or more, Kalman filters should be developed and an initial detection step performed first to determine which filter should be used.

As was mentioned in the previous chapter, the curve fitting algorithm used a crude estimate of the number of notes to fit the pitch track to. It was found experimentally that when the algorithm knew *a priori* what the correct *number* of notes was that performance was very good. A more sophisticated method of estimating the number of notes to fit would certainly improve segmenter performance. However, a more sophisticated approach would most likely involve computing the optimal curve fit for multiple note number hypotheses and then selecting the hypothesis with the least overall cost, or risk [13]. Given that the algorithm was quite slow for even a single note number hypothesis, computing multiple hypotheses may be computationally unreasonable.

But this lack of a satisfying method to estimate the number of notes opened the door to a new possibility. Rather than use a transcription of a sung melody as the database query, why not use a decomposition from the most 'reliable' to the least 'reliable' notes as the database query? Such an investigation was beyond the scope of this work.

In spite of the above recommendations, a substantial improvement in performance will only be had by taking a fundamentally different approach. In particular, a more analytic model of the pitch track needs to be developed in order make the subtle passing notes more obvious. Visually inspecting the pitch tracks of various recordings reveals that often the passing notes are completely hidden in the pitch changes between major notes. It is an interesting issue in the perception of sound, we can hear a passing note even when the pitch track shows no plateau whatsoever. In some cases this may be due to other information, that is the listener is familiar with the tune and 'inserts' the passing note. This disparity between what pitches we can perceive aurally versus those visually apparent in the pitch track must be resolved. It is unclear however whether any amount of modelling will be able to unravel this complexity.

One particularly promising direction for future work is incorporating speech recognition ideas into melody segmentation. Often a passing note is recognized by the listener not because of its pitch, but because it is articulated as a separate syllable. The speech recognition community has explored many techniques for segmenting continuous speech into isolated syllables, although to date segmenting sung speech into syllables has not been considered.

Hence the question of circumventing the segmentation problem again presents itself. To date, most aural MIR systems have all been based on some sort of transcription of recorded audio. The choice of using a note sequence for searching may be an intuitive, but there is no particular evidence to imply it is optimal. Given that a robust transcription algorithm for sung melodies has yet to be developed, perhaps other characteristics of the sung queries should be explored. There are any number of other characteristics that may be more analytically tractable, such as note renewal times or time and frequency differences between the high and low notes of a passage.

## 6.3 Conclusion

Several techniques for the automatic segmentation of sung melodies have been explored. The fundamental philosophy adhered to throughout this work was that a melody segmentation algorithm should rely *only* on pitch change information. A multiple note passage will often be sung with constant amplitude and timbre, hence a segmentation algorithm should not rely on amplitude and timbre. Three classes of segmentation algorithms were explored: predictive filtering, neural networks and curve fitting. Of the two predicative filters implemented, the RLS filter outperformed the Kalman filter. This was due primarily

to the Kalman filters inability to track the time-varying characteristics of the pitch track. Numerically the neural network yielded the best performance, although this superior performance is due in part to the training data used to design the algorithm. Applying the NLMS algorithm to a different or expanded database will likely reveal weaknesses not seen in this work. Lastly, while the curve fitting algorithm did not yield especially good segmentation results, the exploration did propose alternate melody representations that may be useful for future MIR systems.

It was concluded that any substantial further improvement will require incorporating volume and timbre information. In particular, the ambiguity regrading passing notes and scooping regions cannot be resolved with only pitch information. Nonetheless, the methods explored here do present improved performance over conventional methods for segmenting sung melodies.

# References


[1] R. Andre-Obrecht, *A Statistical Approach for the Automatic Segmentation of Continuous Speech Signals*, IEEE Trans. on Acoustics, Speech and Signal Processing **36** (1988), no. 1, 29–110.

[2] M. A. Bartsch, *Automatic Assessment of the Spasmodic Voice*, Qual II Report, University of Michigan, 2002.

[3] E. Batlle and P. Cano, *Automatic Segmentation for Music Classification Using Competitive Hidden Markov Models*, Proceedings of ISMIR 2000 (2000), Plymouth, MA.

[4] W. P. Birmingham, R. B. Dannenberg, G. H. Wakefield, M. Bartsch D. Bykowski, D. Mazzoni, C. Meek, M. Mellody, and W. Rand, *Musart: Music Retrieval Via Aural Queries*, Proceedings of ISMIR 2001 (2001), Bloomington, IN.

[5] P. Boersma, *Accurate Short-Term Analysis of the Fundamental Frequency and the Harmonics-to-Noise Ratio of a Sampled Sound*.

[6] W. Chou and L. Gu, *Robust Singing Detection in Speech/Music Discriminator Design*, IEEE International Conference on Acoustics, Speech and Signal Processing **2** (2001), 865–868.

[7] R. B. Dannenberg, *Music Representation Issues, Techniques, and Systems*, Computer Music Journal **17** (1992), no. 3, 20–30.

[8] S. Duanmu and G. H. Wakefield et al, *Taiwanese Putonghua Corpus (TWPTH) Speech and Transcripts*, Linguistic Data Consortium (1998).

[9] J. K. Fwu and P. M. Djurie, *Segmentation of Piecewise Constant Signal by Hidden Markov Models*, IEEE Signal Processing Workshop on Statistical Signal and Array Processing (1996), 283–286.

[10] S. Haykin, *Adaptive Filter Theory*, Prentice Hall Ptr, Upper Saddle River, NJ, 1996.

[11] A. K. Jane and R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall Ptr, Englewood Cliffs, NJ, 1988.

[12] S. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*, Prentice Hall Ptr, Upper Saddle River, NJ, 1993.

[13] ———, *Fundamentals of Statistical Signal Processing: Detection Theory*, Prentice Hall Ptr, Upper Saddle River, NJ, 1998.

[14] T. Kemp, M. Schmidt, M. Westphal, and A. Waibel, *Strategies for automatic segmentation of audio data*, IEEE International Conference on Acoustics, Speech and Signal Processing **3** (2000), 1423–1426.

[15] Y. E. Kim, W. Chai, R. Garcia, and B. Vercoe, *Analysis of a contour-based representation for melody*, 2000.

[16] N. Kosugi, Y. Nishihara, and T. Sakata et al, *A PracticalL Query-By-Humming System for a Large Music Database*, ACM Multimedia (2000), 333–342, Los Angelos, CA.

[17] J. S. Lim, *Two-Dimensional Signal and Image processing*, first ed., Prentice Hall Ptr, Upper Saddle River, NJ, 1990.

[18] D. Mazzoni and R. B. Dannenberg, *Melody Matching Directly from Audio*, Proceedings of ISMIR 2001 (2001), Bloomington, IN.

[19] R. J. McNab and L. A. Smith, *Evaluation of a Melody Transcription System*, IEEE International Conference on Multimedia and Expo, 2000 **2** (2000), 819–822.

[20] R. J. McNab, L. A. Smith, D. Bainbridge, and I. H. Witten, *The New Zealand Digital Library MELody inDEX*, D-Lib Magazine (1997), http://www.dlib.org/dlib/may97/meldex/05witten.html.

[21] M. Melucci and N. Orio, *Musical information retrieval using melodic surface*, Proceedings of ACM Digital Libraries Conference (1999), 152–160, Berkeley, CA.

[22] M. Pandit, J. Kittler, Y. Li, and E. H. S. Chilton, *A Comparative Study of Different Segmentation Approaches for Audio Track Indexing*, International Conference on Pattern Recognition **2** (2000), 467–470.

[23] C. Raphael, *Automatic Segmentation of Musical Signals Using Hidden Markov Models*, IEEE Trans on PAMI **21** (1999), no. 4, 360–370.

[24] A. K. V. SaiJayram, V. Ramasubramanian, and T. V. Sreenivas, *Robust parameters for automatic segmentation of speech*, IEEE International Conference on Acoustics, Speech and Signal Processing **1** (2002), 513–516.

[25] A. Sterian, *Model-Based Segmentation of Time-Frequency Images for Musical Transcription*, Ph.D. thesis, University of Michigan, Department of Electrical Engineering and Computer Science, 1999.

[26] P. Stoica and R. Moses, *Introduction to Spectral Analysis*, Prentice Hall Ptr, Upper Saddle River, NJ, 1997.

[27] J. R. Treichler, C. R. Johnson, and M. G. Larimore, *Theory and Design of Adaptive Filters*, Prentice Hall Ptr, Upper Saddle River, NJ, 2001.

[28] J. P. van Hemert, *Automatic Segmentation of Speech*, IEEE Trans on Signal Processing **39** (1991), no. 4, 1008–1012.

[29] O. Weisman and Z. Pollack, *The Perceptron*, http://www.cs.bgu.ac.il/ omri/Perceptron, 1995.

[30] T. Zhang and C. C. J. Kuo, *Audio Content Analysis for Online Audiovisual Data Segmentation and Classification*, IEEE Trans. on Speech and Audio Processing **9** (2001), no. 4, 441–457.