

**DIGITAL SIGNAL PROCESSING TECHNIQUES FOR
THE AUTOMATED RECOGNITION OF MUSICAL TONES**

by

STEPHEN CLARK DAVIES

B.S., Rice University, 1992

A thesis submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirements for the degree of
Master of Science
Department of Electrical and Computer Engineering

1996

This thesis for the Master of Science degree by

Stephen Clark Davies


has been approved for the

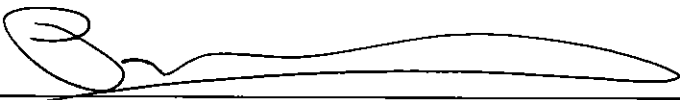
Department of

Electrical and Computer Engineering

by


Delores Etter


Michael Lightner


Peter Mathys

ACKNOWLEDGMENTS

I dedicate this thesis to the Lord God Almighty, without Whose mercy and guidance finishing on time would have been impossible, but through Whose strength all things are possible.

Also, to Dr. Delores Etter for her seemingly endless patience through many illnesses and other trials I experienced, and to my loving wife Rae, whose help on all aspects of this project was invaluable, and whose constant devotion and confidence is always a great source of encouragement to me.

CONTENTS

| CHAPTER | | |
|---------|--|----|
| I. | INTRODUCTION | 1 |
| | Motivation | 1 |
| | Scope of the Study | 3 |
| | Physical Setup | 4 |
| | Arrangement of the Thesis | 7 |
| II. | CHARACTERISTICS OF MUSICAL INSTRUMENTS | 8 |
| | Volume Envelope | 8 |
| | Harmonic Signature | 14 |
| | A Model for Musical Instruments | 21 |
| III. | OVERALL STRATEGY FOR RECOGNITION | 23 |
| | Event Detection | 24 |
| | Pitch Extraction - Compiling the Frequency Domain Information | 38 |
| | Pitch Extraction - Analyzing the Frequency Domain Information | 41 |
| | <i>Score's</i> Implementation of the Pitch Extraction Algorithm | 58 |
| | A Formal Description of the Pitch Extraction Algorithm | 59 |
| IV. | RECOGNITION OF SINUSOIDAL INSTRUMENTS WITH SQUARE ENVELOPES | 63 |
| | Choosing a Simplified Model | 63 |
| | Results | 67 |
| | Considerations | 73 |

| | | |
|------|--|-----|
| V. | RECOGNITION OF "REAL" INSTRUMENTS | 78 |
| | Choosing a set of "real" instruments | 79 |
| | Solo instruments | 82 |
| | The "Soprano-Alto problem" | 87 |
| | Recognition of Songs Containing Harmonic Ambiguity | 95 |
| VI. | CONCLUSION | 110 |
| VII. | FUTURE AREAS OF INVESTIGATION | 113 |
| | END NOTES | 116 |
| | BIBLIOGRAPHY | 118 |
| | APPENDIX A: MATLAB Implementation of the <i>Score</i> Program | 119 |
| | APPENDIX B: Volume envelopes and harmonic signatures for various synthesized instruments..... | 158 |

FIGURES

FIGURE

| | | |
|-----|--|----|
| 1. | Studio and laboratory equipment used to test the <i>Score</i> program..... | 5 |
| 2. | Volume envelope for the plucking of a harp string..... | 9 |
| 3. | Volume envelopes used to describe ADSR parameterization | 10 |
| 4. | Volume envelopes for striking and continuous instruments..... | 11 |
| 5. | Discrete standing waves of a stringed instrument | 14 |
| 6. | Standing waves for wind instruments | 16 |
| 7. | Effects of resonator frequency response on audible signal | 18 |
| 8. | Illustration of STA/LTA power ratio threshold technique | 29 |
| 9. | The insufficiency of the STA/LTA algorithm in isolation | 30 |
| 10. | Combination of linear predictor error with STA/LTA | 36 |
| 11. | False alarm elimination heuristic | 37 |
| 12. | The expected value of the joint amplitude of two sinusoids | 57 |
| 13. | Effects of <i>staccato</i> and <i>legato</i> styles on <i>Score</i> output | 65 |
| 14. | Characteristics of the simplified model used in initial testing of <i>Score</i> | 67 |
| 15. | Effect of the LTA window size in event detection | 74 |
| 16. | The ambiguity inherent in multiple part attacks | 76 |

TABLES

TABLE

| | | |
|----|---|----|
| 1. | Terminology used in the discussion of the pitch extraction algorithm | 43 |
| 2. | Summary of notation for the pitch extraction algorithm | 59 |

CHAPTER I

INTRODUCTION

Motivation

As low-cost computing solutions have become more widely available to them, studio musicians have discovered many applications of concise, digital representations of their music. Already, keyboardists can capture their performances in digital MIDI format and play them back later at a different tempo, with a different volume mix between parts, or even with different timbres (synthesized instruments) than were originally used in the recording. The songs themselves are simply saved in a "musician's format," with information about which notes were played at what instant, and with what volume, but with no explicit audio information such as an actual waveform. At playback time, a digital sound module plays these notes in time from selected voices (a voice could be the sound of a flute, or bagpipe, or piano, for instance.) The composer can experiment with different voices, levels, and effects to produce a wide variety of different sounding songs, all from the same original recording. Such flexibility is a powerful tool in the hands of a competent musician.

What is lacking, though, is the ability to "go the other way." In other words, it is a simple matter to take a MIDI file with information about pitches and timing, and produce an audible song from it using a digital sound module. All that is necessary is to play the proper tones at the proper times using whatever voices the musician wishes. However, the ability to take raw audio information from, say, a band in a concert hall, and analyze it to produce a written score of the piece (or at least a MIDI file from which a score can be retrieved) is a much more difficult matter. A sample of

a single-voice, pure-tone melody could probably be analyzed and the corresponding notes of the melody constructed. But the digital signal processing required to determine the score of a piece being performed by a twelve-part orchestra, each part with a rich audio spectrum and performing out of time with the other parts, staggers the mind.

Many parallels exist between this problem and other analysis-synthesis problems, such as text recognition. It is a relatively straightforward task, for instance, to take a file of ASCII text, together with a set of bitmaps (or mathematical descriptions) depicting those characters in a particular font, and display the document in that font. This is because there is no guesswork involved. The computer does not have to analyze anything complicated or ambiguous, but just to spit out the image associated with each of the characters in a repetitive way. Consider the difficulties, however, when the problem is reversed: instead of starting with the list of characters and producing the graphics, to be given a graphical image and attempt to derive the characters from it. Now, instead of monotonously outputting images that are known *a priori*, the program must analyze its inherently "fuzzy" input and apply a set of rather vague guidelines to try and deduce what information is present. This is the situation with automated musical recognition, or automatic transcription, as it is also called. We are given not a rigid written score from which to play a melody, but a recording of instrumentalists playing the tune with feeling and interpretation, in the presence of noise, reverberations, echoes, and other acoustical effects, and asked to derive the notes.

The capability to analyze musical performances in an automated fashion - either to produce a written score, or to segment the separate voices of the audio track so they can be treated independently - would make possible a wealth of applications. Rock musicians who like to "jam" in the studio to come up with new songs could

circumvent the painful process of trying to put down on paper what felt natural on the guitar. Computer systems could be developed for beginning music students to provide automated feedback about notes incorrectly played. Erroneous notes in an *a cappella* recording could be identified and removed without even having to do another take. The musical expressions of primitive cultures without technology or even a precise notation could be recorded locally on cassette tape and quickly analyzed for rhythm and harmony. Expensive multi-track recording could be eliminated, since keeping different parts isolated on different tracks would become unnecessary. And alterations of existing musical pieces could be created that were only dreamed of before. Imagine being able to reduce the Cleveland Orchestra's performance of Mozart's 40th Symphony down to the score, and thereby play it back with woodwinds instead of strings. Or capturing a jazz saxophonist's improvisation in MIDI format so it could be analyzed and elaborated upon. Or combining these techniques with sophisticated speech processing to change John Lennon's voice into Dolly Parton's in the Beatles' original recording of "Hey Bulldog." Granted, these applications represent "grand challenges" and would require considerable research effort. Nevertheless, their utility and power provide a great motivation for learning more.

Scope of the Study

As the title of this work implies, my research concentrates on the development of *software* techniques that might make automated musical recognition possible. I have not designed a special-purpose digital signal processing (DSP) chip or microprocessor for the task, but rather have considered the problem of taking mono-channel audio, recorded and digitally sampled elsewhere, and employing software algorithms to attempt to derive the notes from the sample.

The centerpiece around which this thesis revolves, then, is a software system which implements and combines a variety of DSP procedures. The title of this program is "*Score*" - so named because its purpose is to furnish information from which a written musical score of its input could be constructed. *Score* was developed entirely in MATLAB, and comprises several modules which analyze digital audio in different ways. The input it receives, in addition to a considerable number of parameters through which its internal algorithms can be "tweaked," is a single data vector of samples from a recorded song. After processing the vector, it attempts to produce textual information equivalent to a written musical score of its input, including the durations (half note, dotted 16th note, etc.) and pitches (C4, F#6, etc.) of notes contained therein. A complete program listing of the *Score* prototype is provided in Appendix A.

The principal focus of my study was to investigate the DSP-related problems involved in identifying musical information, and not to build a comprehensive product which musicians would find immediately useful. Therefore, a few things that would be necessary in a fully-featured tool, but that deal strictly with musical issues and not with DSP, have not been addressed as part of this thesis. For example, I have not written procedures to make a reasonable guess as to the key signature or time signature of the piece, given the series of pitches. A user-friendly, graphical display of notes on staves, too, is outside the bounds of this project. I have focused only on the problem of extracting the pitch and timing information of notes embedded in a digitized signal.

Physical Setup

Most of the songs I have used as input for my experiments in the laboratory were generated by a Korg 03R/W sound synthesis module. (See Figure 1.) The 03R/W is a powerful piece of equipment which accepts as input MIDI controller

information about what notes to play (for instance, from a keyboard capable of producing MIDI output, or from a software sequencer) and produces an analog signal simulating the waveforms of live instruments playing those notes. Its advanced synthesis techniques allow it to generate sounds which are quite similar to those of an actual violin, trumpet, oboe, etc., and in many cases they are indistinguishable to the human ear. Effects such as echoes or reverberations occurring in live conditions can also be generated by the 03R/W.

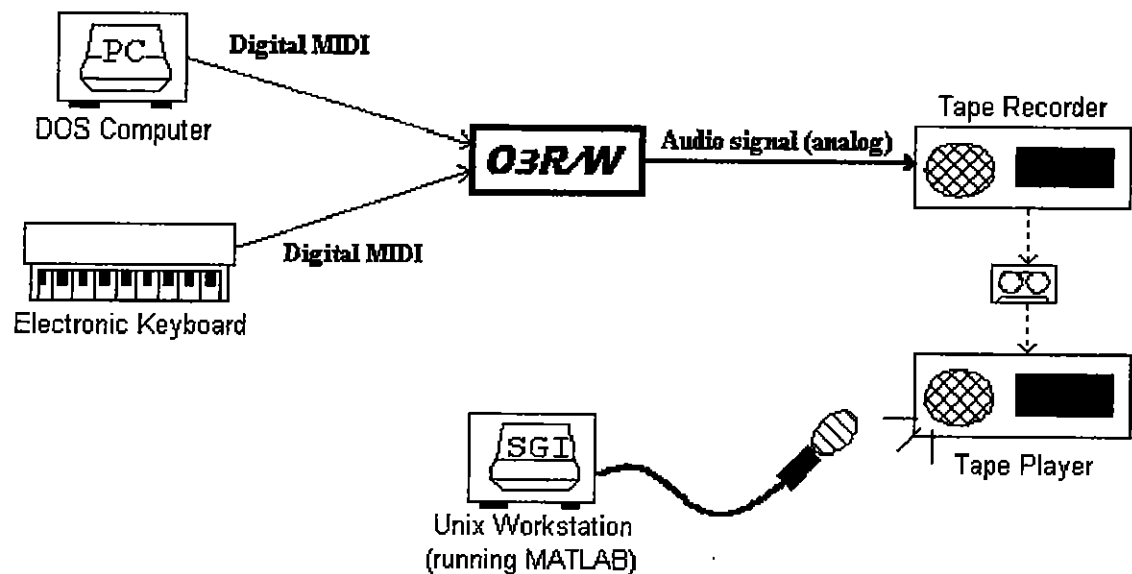


Figure 1. Studio and Laboratory equipment used to test the *Score* software. Notes are generated from either an electronic keyboard played by an instrumentalist, or from sequencer software on a PC playing an existing piece of music. Pitch, volume, and timing information about these notes are transmitted over a MIDI cable (asynchronously, at extremely low data rates) to the Korg 03R/W sound module, which in turn synthesizes an audio signal that can be input to the AUX jack of an ordinary tape recorder. This tape is then taken from the studio to the laboratory, recorded into the microphone of the workstation, digitally sampled, and prepared for software analysis.

To test the *Score* program, I have used the *Midisoft Studio for Windows* software sequencer to create simple melodies and harmonies. (This sequencer is essentially a musical editor, through which information about the times and pitches of notes can be entered and modified.) After a melody has been entered, the sequencer "plays" the song by sending MIDI information over a serial cable to the 03R/W, instructing it to generate the pitches of the song at the specified times. The 03R/W, selecting from a bank of available instruments, then produces the sounds required, such that the output is an audio signal which can be amplified and heard by a listener.

In order to capture this "performance" in a form that MATLAB can input and analyze, I connect the output of the 03R/W to an ordinary cassette recorder via RC audio jacks, and record the sound onto a blank cassette tape. Then, in the laboratory, I play the tape back through the speakers of a tape player, and use the built-in microphone of a Silicon Graphics Unix workstation to sample and digitize this audio signal. The workstation's "Sound Editor" utility program saves the digital data into an ordinary Unix file which the MATLAB program can read and store as a single vector of samples. *Score* then analyzes this data vector using the techniques described in this thesis, and prints lines of text to the screen describing the durations and pitches of the notes it finds. Such text is not in the same form as, but is virtually equivalent to, a written score in musical notation.

One liability of the above arrangement is in the frequency response of the tape player and recorder. The equipment I used was both old and inexpensive, and as a result had a very poor bass response. (Upon testing, frequencies at middle C (261.63 Hz) came through about 10 dB higher than frequencies at an octave below middle C (130.81 Hz.)) Therefore, for much of my testing, I have used only input songs in the treble clef; that is, songs with notes that have fundamental frequencies of 260 Hz or

higher. It should be understood that this is purely due to a weakness in the physical equipment I had available for testing, and not due to any inherent flaw in the algorithms used. The *Score* program does not treat notes below middle C any differently than it does those above, and if machinery capable of a better bass response were available, *Score* should have exactly the same success rate in detecting lower notes as it does the higher ones.

Arrangement of the Thesis

The structure of any kind of systemized analysis depends greatly on the character of what is being analyzed. In Chapter 2, therefore, I will provide an overview of the characteristics of sounds produced by musical instruments, and point out which particular features factor heavily into the automated musical recognition problem. Chapter 3 contains a discussion of the "big picture" algorithm employed by *Score* in light of this information.

From that point on, the thesis - as well as my underlying research - falls into two parts. First, in Chapter 4, I will introduce my approach regarding a simplified model for musical instruments: a sound generator with a sinusoidal wave form and square volume envelope. I will present the results of the *Score* program when given such simplistic data, and demonstrate that the automated recognition of notes produced by this model is not only theoretically possible, but realistically achievable with a very high rate of success. Then, Chapter 5 explores the program modifications necessary when the eccentricities of "real world" instruments are introduced. Finally, Chapters 6 and 7 put my research into perspective, and discuss what the next step might be towards the realization of a practical product suitable for assisting musicians in a substantial, everyday way.

CHAPTER II

CHARACTERISTICS OF MUSICAL INSTRUMENTS

The differing qualities of the sounds produced by common musical instruments are complex and widely varied. A study of audio characterizations could easily fill volumes; an extensive treatment is given in [4]. For purposes of the *Score* program, however, such detailed analysis is unnecessary. The problem of automated musical recognition is concerned neither with how to duplicate the tonal qualities produced by live performers, nor even with the identification of what *kinds* of instruments are present in a given recording. My algorithms seek only to recognize the pitch and timing information associated with each instrument in a song, and hence, the only tonal features we need to explore are those which factor into this problem. This chapter presents some practical considerations regarding what distinguishing attributes of musical notes can be exploited to uncover this information.

Volume Envelope

If we were to plot the audio waveform produced by the plucking of a harp string, we would discover that the magnitude of the signal varied significantly with time. At the instant the note is played, we would see a sharp rise in intensity, followed by something very much like an exponential decay. (See Figure 2.) This agrees with our intuition in hearing the instrument, for a "plucking" sound has a sudden, crisp burst of tone followed by rapidly decreasing volume, and in just a few moments, we hear nothing. The signal generated when a piano key is struck has a somewhat similar shape, though the peak is not as intense, and the sound does not die away as quickly.

Other instruments - such as violins or flutes - have completely different characteristics, and may even present increasing intensity as the note is played. We can think of this phenomenon as an envelope which is multiplied by the underlying periodic components that produce the sound. It turns out that this so-called "volume envelope" is one of the most important factors in determining how the instrument sounds to the listener. The differences in the periodic waveforms produced by a piano and a flute, for instance, may not be very great; yet their sounds could never be confused precisely because they have such remarkably dissimilar volume envelopes.

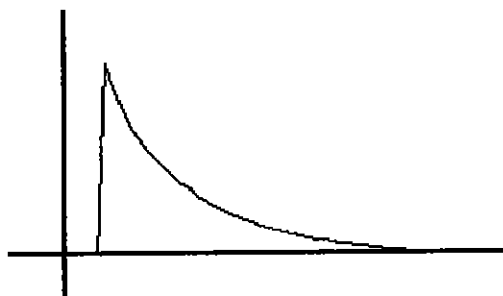


Figure 2. The approximate shape of the volume envelope for the plucking of a harp string. The amplitude of the signal produced is not uniform throughout the note, but rises very fast initially and then experiences a near exponential decay.

Theoretically, there is no limit to the amount of "detail" the volume envelope of a sound might exhibit; we might even envision a hypothetical instrument whose intensity varied with time as pictured in Figure 3a. For convenience, however, we will adopt a simplified model of the volume envelope that is familiar to musicians who attempt to synthesize sounds: the "Attack-Decay-Sustain-Release," or ADSR representation, depicted in Figure 3b. In this simplified model, the volume envelope of an instrument is represented with just four parameters. The "Attack time" is the time, in seconds, from which the note is struck until it reaches its maximum intensity. The "Decay time" represents how long it takes for the sound to drop from its peak intensity

to its "Sustain level," or the volume (in percentage of the maximum) held by a long whole note. Lastly, the "Release time" characterizes how long the instrument continues to sound (above some arbitrary minimum level) after the performer stops playing the note. We might parameterize the ADSR envelope of any instrument, then, as a four-tuple of values. For example, the sound in Figure 3c has an ADSR of (1,0,100%,1), and the sound in Figure 3d has an ADSR of (0.1,1,20%,2). Note that the precise mathematical description of how the envelope behaves during the four intervals (eg., whether the attack is exponential, linear, or sub-linear) is not specified as part of the ADSR. The model is only used to provide a rough classification of the way the sound's intensity varies with time, and will be suitable for our purposes.

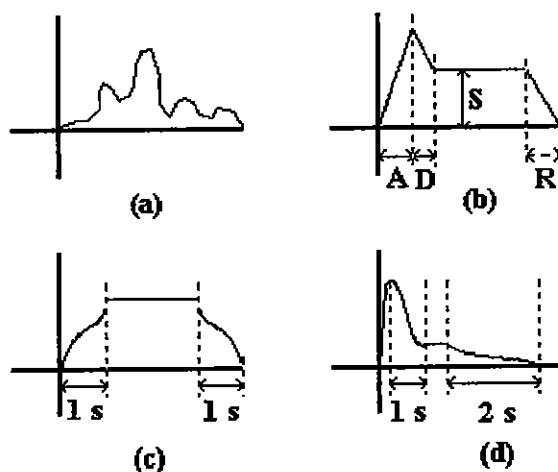


Figure 3. (a) Volume envelope for a hypothetical instrument. (b) The definitions for the ADSR volume envelope parameterization. (c), (d) Sample volume envelopes.

Each of the instrumentalists that make up an orchestra or a rock band produces sound with a widely different ADSR characterization. To get some idea of the nature of these envelopes, however, we can divide the vast array of musical instruments into

two broad classes. I will call "striking" instruments those devices where a note is sounded by some initial sharp contact, such as when an acoustic guitar string is plucked or a xylophone is hit with a mallet. "Continuous" instruments, on the other hand, produce sound only when provided with a continuous amplitude source, such as the constant bowing action of a cello or the blowing of a trombone. These two classes of instruments tend to have very different ADSR characterizations, as depicted in Figure 4.

Figure 4a shows an approximation of the volume envelope for a piano (which is a striking instrument, since notes are produced when hammers strike strings inside the body of the mechanism.) We notice several important features from the diagram, which is roughly typical of all striking instruments. First of all, such instruments have an extremely fast attack, which corresponds to $A \approx 0$ in the ADSR model. The moment the key is pressed (or the string is plucked, or the drum is struck), there is an explosion of intensity up to the maximum level. We also see that there is no real notion of a sustain level for such instruments, since after the initial event which triggered the sound, there is no more energy input to the system by the performer. Hence, we may consider $S \approx 0$ as well.

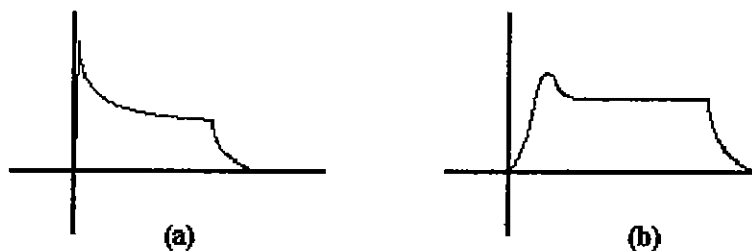


Figure 4. Typical volume envelope for (a) instruments of the "striking" family, (b) instruments of the "continuous" family.

The rate of decay of intensity is highly dependent upon the nature of the striking instrument, and it is difficult to establish generalizations. For stringed instruments, the more rigidly the string is mounted, the more energy is transformed into heat instead of sound, leading to a less intense peak but a slow decay. A piano, on the other hand, is constructed such that the endpoints of the string can move slightly, releasing energy to a resonator (the piano's sound board). This allows for a greater overall signal intensity, since the energy is converted with greater efficiency into sound, but also gives rise to a faster rate of decay. Striking instruments involving other vibrating elements (eg., the percussion family) have rates of decay that are dependent on completely different parameters. In general, we may say that the D value of the ADSR characterization varies even among instruments within the "striking" categorization. Similarly, we find striking instruments with varying release times, since this is also governed by a broad range of physical parameters. The rates at which a piano's damper cushions its string or a timpanist's hands quiet the drum's membrane are difficult quantities for which to establish broad guidelines. So in the end, we are left with the principle that striking instruments tend to have an ADSR parameterization of nearly (0,D,0,R), where the values of D and R are dependent on physical measurements peculiar to the instrument and which may vary widely from instrument to instrument.

When we come to analyze instruments in the "continuous" class, we find a very different sort of volume envelope than we did with striking instruments. As can be seen in the Figure 4b (an approximation of the envelope for a bowed violin), the attack is no longer instantaneous. It takes the sound some time to reach its maximum intensity, which has a significant effect on the ability of the *Score* program to detect notes from these instruments, as we will see later. The actual value of the A parameter is dependent upon a number of factors specific to the instrument and performer, such

as the time it takes for the bow to fully "grip" the string, or how long the flautist takes to reach maximum air pressure. Another feature we glean from this diagram is that although the continuous instrument may have some decay from its peak intensity, it often very quickly establishes a nearly constant amplitude which lasts for the duration of the note. This is because the amplitude loss seen in the striking instrument is now compensated by the supply of extra energy to the vibrating system. Instead of a piano, which is struck once and then left to decay, we have a clarinet, which is continually given more energy by the breath of the performer. At the very beginning of the attack, the supplied power far exceeds the rate of energy loss, and the signal's amplitude gradually builds up. Then, while the power supply remains constant, the power dissipation increases as the amplitude grows, and a steady-state condition is reached where the dissipated power is equal to the supplied power. This is the situation all the time the note is sounding (subject to intentional or unintentional variations on the part of the instrumentalist, of course), until the release point. The R parameter for continuous instruments varies widely across the different families as it did for striking instruments, since once the breath or bow has stopped the rate of decay of power is subject to all kinds of physical properties of the particular mechanism in question. Even among different violins we will find bodies that resonate for longer periods, so we dare not make any simplifications for the R parameter. In summary, then, the distinguishing marks of a continuous instrument's volume envelope can be expressed in an ADSR which is (nonzero, short, high, R), remarkably different than striking instruments in three of the four categories.

We will defer until Chapter 4 the discussion regarding which ADSR characteristics might facilitate the correct identification of notes in an analog waveform, and which might hinder it. For now, it is sufficient to note that different musical instruments - particularly those from different categories - have noticeably

diverse volume envelopes, which not only lend a distinctive quality to the sounds they produce, but which appear as conspicuously dissimilar attributes in the signal processing domain.

Harmonic Signature

A second characteristic of musical tones which has a great bearing on the observed quality of the sound is the frequency content of the instrument's waveform. A basic property of virtually all musical instruments (percussion excepted) is that the chief frequencies they produce are almost all harmonically related to one another.¹ The reason for this can be seen in Figure 5a, which depicts a string about to be plucked with both of its ends rigidly fixed. When such a string is struck, hammered, or bowed, it begins to vibrate; and elementary physics tells us that the only possible stable form of vibration are standing waves, which by necessity have frequencies such that an integer number of wavelengths will "fit" on the string. Because of the principle of linear superposition, these standing waves can all co-exist on the same string without interfering with each other.

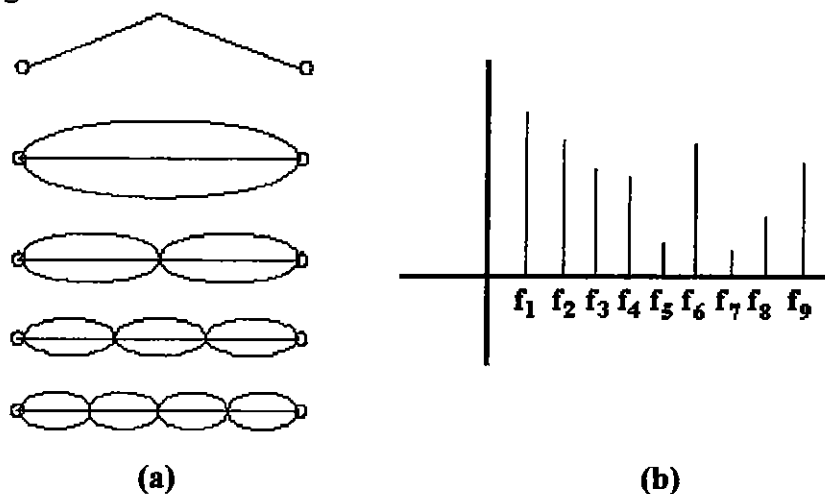


Figure 5. (a) Standing waves produced by the plucking or hammering of a string. (b) The discrete nature of the frequency spectrum for a musical instrument.

The standing wave with the longest wavelength - and lowest frequency - is the top one shown in the diagram. It vibrates with frequency f_1 , which can be described to a good approximation as

$$f_1 = \frac{1}{2L} \sqrt{\frac{T}{d}}$$

where L is the length of the string, T is the tension in units of force, and d is the mass per unit length of the string. We refer to this frequency f_1 as the fundamental frequency of the vibration, and use it as the best measure of the pitch of the note². The other frequencies represented in the diagram are all integer multiples of the fundamental frequency, and are denoted the harmonics of the fundamental. f_1 itself is the first harmonic, f_2 the second harmonic, and so on. There are an infinite number of these harmonics for a given fundamental frequency, though only a certain number fall within the range of human hearing. The result is that if we took a Fourier Transform of the waveform produced by the vibrating string, we would see that virtually all of the energy was concentrated in these harmonic frequencies, in various proportions (see Figure 5b).

This situation is not unique to the family of stringed instruments; in fact, all orchestral and acoustical instruments exhibit this property. Figure 6a depicts a simple model for a flute, with the mouthpiece and the first open hole on the body serving as the open ends of an otherwise closed pipe. When longitudinal waves driven by the flautist's breath propagate through the cylinder, the pressure at any point along the body can experience temporary increases or decreases, since the rigid cylinder walls hold the necessary forces in balance with respect to the atmospheric pressure outside. At the two holes, however, there can be no substantial difference between the pressure inside and outside of the flute, because the inside is exposed to the open air. These points therefore play the same role with respect to air pressure that the fixed ends of

the string do to vibration, and the only stable frequencies at which air can vibrate in a standing mode are those which satisfy the integer number of wavelengths property. Virtually all the instruments that are driven by breath work in roughly this way, where the pitch of the note played can be altered by changing the distance between the holes (via fingers, valves, or a trombone's slide). The result is that all common instruments almost exclusively produce frequencies that appear at discrete, regular intervals. It is really this property which gives musical instruments their recognizable tone that is pleasing to the ear. When frequencies are produced which are *not* harmonically related, the result is more like a sound than a note. The notion of the "pitch" of such a sound becomes blurry, and these kinds of audio events become useless for carrying their weight as a distinctive tone in a melody. Most of the instruments in the percussion family behave in this way - for instance, a cymbal crash, which appears in the frequency spectrum as a concentration of high frequency noise. The prototype of the *Score* program will not attempt to analyze such instruments.

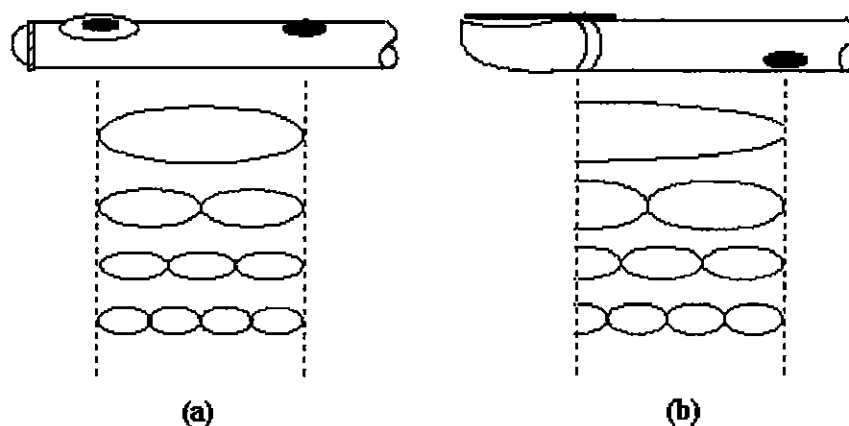


Figure 6. Standing waves formed in the chambers of (a) a flute, (b) a clarinet.

We have established, then, that for ordinary, non-percussive instruments, only frequencies which are harmonically related appear with any significant magnitude. The next question becomes: what are the relative *strengths* of each of the harmonics for particular instruments? And what physical properties of the instruments give rise to these strengths? In practice, the question proves very difficult to answer because of the extremely complex physical situations. We can, however, provide some idea of the factors involved. In some cases, the construction of the body of the instrument itself naturally reinforces certain harmonics while suppressing others. A good example is the clarinet, shown in Figure 6b. The mouthed end of the instrument is a vibrating reed which serves a purpose opposite to that of the first open finger hole, or the bell at the end of the body. It essentially functions as a closed end, making the clarinet itself very much like a stopped cylindrical pipe. The result is that the clarinet's fundamental frequency is only half of that for a pipe of the same length with two open ends, and the stable modes of vibration are only at the odd numbered harmonic frequencies with respect to this lower fundamental. The distinctive tone of the clarinet arises in part from this extreme emphasis of the odd harmonics and absence of even harmonics.

The harmonics produced by a stringed instrument are highly dependent on the physical location along the string where it is plucked or struck. If a string is grasped at the midpoint between its two fixed ends, as was the case in Figure 5a, we find that odd harmonics are emphasized and evens are suppressed. The reason is simply that the point on the string with the greatest initial displacement - and hence, greatest kinetic energy when the string is released - is in the middle, which corresponds to a peak on all standing waves at an odd integer multiple of the fundamental, and to a node for all even multiples. In general, it can be shown that all harmonics which have a peak at the point where the string is excited will be strong and all that have a node will be weak. Plucking near the ends of the string, then, will naturally augment the upper harmonics

and give a "richer" tone, while plucking in the center subdues them and yields a "purer" tone. This phenomenon allows harpists great flexibility to modify the quality of the sounds they produce throughout a single song.

Other effects are more subtle. The fingerholes of a flute, whether open or closed, tend to attenuate the high harmonics (those above 1.5 kHz) and give the instrument a very simple but beautiful spectral content. We find that partly because of this effect, throughout most of its range the flute exhibits a strong first three harmonics, and then almost nothing higher. This quality is what gives the instrument its pure, clean tone quite different from many of the other woodwinds. Another important physical factor which influences the harmonic content of instruments is the resonator employed to efficiently transform vibrational energy into sound. Two good examples are the body of a cello and the piano's soundboard. Both instruments are initially driven by vibrating strings, but the relative magnitudes of the harmonics at which the strings vibrate may be quite different from the frequencies the resonator will naturally reinforce. We may crudely say that each resonator can be characterized by a "resonance curve" as shown in Figure 7, and that whatever spectrum the vibration component of the instrument produces is filtered through that curve to arrive at the set of frequencies really heard by the listener.

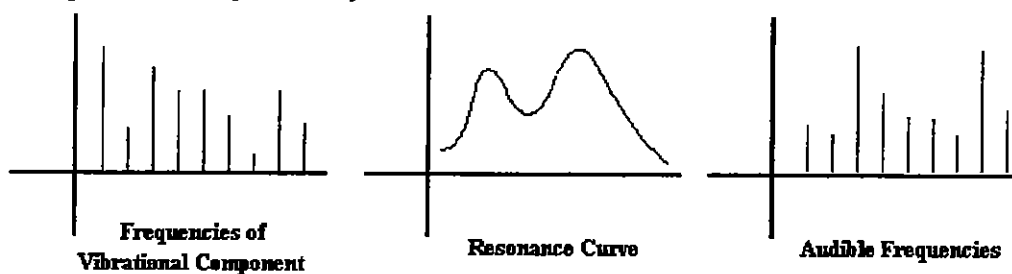


Figure 7. Effects of resonator frequency response on audible signal. The frequencies originally present in the vibrational component of the instrument are filtered through the resonance curve (specific to the type and particular model of the instrument) to produce what is actually heard by the listener.

Given that different kinds of instruments have such diverse sets of harmonic components, one might be tempted to attach a unique "harmonic signature" (or "spectral envelope," as it is often called in the literature) to each instrument and use that information to assist in automated musical recognition. Unfortunately, this idea proves overly simplistic in practice. It is certainly true that there are specific frequency domain characteristics that remain constant in many circumstances, such as the clarinet's weak even harmonics or the flute's bold fundamental. But the view that the harmonic signature forms a stable and compact representation of the frequency content produced by an instrument, and can be used to reliably and completely characterize its behavior, is flawed for a number of reasons.

First of all, even instruments of the same kind display vastly different harmonic spectra. It has been determined by experiment that two different makes of grand piano, each playing the same note, can easily produce signatures that are dissimilar in many respects.³ The aforementioned resonance curve is another example: comparing the frequency response of violin bodies of different makes or even different instruments among the same make yields widely different behaviors. For these reasons, it seems hopeless to attempt to construct "the piano signature" or "the oboe signature" for use in recognizing any particular piano or oboe with accuracy.

To make matters even worse, we find that the harmonic signature varies *even on the same instrument during the same performance*. We have already seen how the audio frequencies produced by a stringed instrument vary with the physical location of string excitation. Even more universal, the frequency content changes with the *intensity* of the notes played. This is particularly evident in reed instruments, where the n th harmonic is perceived to grow proportionally to the $2n$ th power of the fundamental, leading to a "brighter" sound for louder notes. Therefore, any musical piece or even individual phrase where an instrument is likely to vary its dynamics from

note to note will involve a part with a dynamically changing harmonic signature. The signature can also be seen to fluctuate with the pitch of the note played. This much is apparent just from the fact that a resonator has fixed filter characteristics, while notes of different pitch will input different sets of frequencies to that filter. But at an even more basic level, the waveforms of the vibrational components themselves can change dramatically with the fundamental frequency. The four strings of a violin, for instance, have harmonic signatures so completely different from one another that upon analyzing them one would never guess they came from the same instrument.

Finally, and even more detrimental to the prospect of characterizing instruments by a static harmonic signature, is the fact that the signature can vary with time even in the duration of a single note. As a rule, for example, frequencies with greater initial energy decay more rapidly than those with less in striking instruments (though this is more or less constant with continuous instruments.) Therefore, as the note plays out its decay interval, each frequency component will be decaying at a different rate, producing an ever-changing signature. Other instruments have bursts of high harmonics present during the attack, and which are immediately quenched thereafter. For wind instruments, the excitation mechanism (such as the reed or air stream) is in turn affected by the sound waves in the air column, a form of considerably complex non-linear feedback. During the initial part of the sound, the tone buildup process often amplifies the upper harmonics more quickly than even the fundamental (since the reed itself tends to vibrate at a much higher frequency until driven by the feedback from the instrument body.) This, too, leads to a decidedly different signature depending on which instant in time it is measured.

The practical result of all these findings is that any attempt to extract the components of a single part in a song from the complete waveform must be very cautious in making assumptions about the frequency content of that instrument. It

may be possible to apply very sophisticated techniques and get a handle on all of the above factors for a given instrument. But any algorithm that simply identifies the harmonic signature of a particular part and proceeds on the presupposition that it will remain constant is doomed to failure.

A Model for Musical Instruments

For purposes of this thesis, then, I have adopted a very flexible model for instruments that makes few presumptions about harmonic content. I have assumed that the volume envelope of a particular instrument, which can be roughly characterized by the ADSR model, is to a great extent determined by whether the instrument is of the striking or continuous variety. The envelope associated with a particular performer should be fairly stationary throughout the song. In regards to frequency content, however, I make few such assumptions about stasis. My model is that the notes sounded by any instrument will have energy present at a frequency corresponding to the pitch of the note played, and at integer multiples of that fundamental, with all non-harmonically related frequencies essentially zero. The relative strengths of these peaks is unknown *a priori*, even if the type of instrument is known, and may change without notice as the song progresses and even as a single note is played. However, I *do* assume that the "harmonic signature" of a particular instrument will remain "relatively stable" throughout the song. This does not mean that the strengths of the harmonics will march in lock-step with precisely the same ratios note after note. We will be under no delusions that we can accurately "subtract out" one instrument's waveform from the whole. In order to track the movement of each part from note to note, though, we will use the harmonic signatures as a rough guideline for instrument identification. These signatures will be computed dynamically, as explained in Chapter 3, not assumed to be known before the data is

analyzed. Such an approach prohibits us from taking advantage of too many unique features of instruments, but allows more hope for a general solution to the automated musical recognition problem.

CHAPTER III

OVERALL STRATEGY FOR RECOGNITION

Presented with a sampled input of an audio signal, there are many possible schemes one might employ to come up with the rudimentary musical information. The approach taken by the *Score* program is a combination of time domain and frequency domain analysis designed to mimic the response of the human ear and mind. What it lacks in theoretical optimacy it makes up for in simplicity and (hopefully) universal applicability.

When one listens to a piece of music, one is conscious of at least two general phenomena. The first is that the character of the tones is changing with time, such that a melody or rhythm is conveyed to the audience. There are identifiable musical phrases which begin and end and lead to other phrases in some sort of progression. A "song" that did *not* exhibit this property - one which had a static character throughout - would be decidedly uninteresting and would probably not be classified as music at all. The second quality is that at each moment in time - an infinitesimal "snapshot" of the song, as it were - there are possibly several instruments each playing a particular note. And from what we learned in chapter 2 regarding the nature of the harmonic spectra of musical tones, each instrument's contribution to the total song is isolated to a set of frequencies related by integer multiples. From the principle of superposition, then, we can say that each of a given song's snapshots is characterized in the frequency domain by a series of discrete peaks of various amplitudes.⁴

Moreover, we observe that in most cases, the gross changes of the music with time are not continuous, but discrete. Except in the case of an instrument "sliding"

through a series of notes (a trombone's *glissando*, for example), we typically hear each instrument begin a note, hold the pitch constant throughout some period, and then move to the next note. We find that the lengths of these periods are usually related in some simple mathematical way, so that the "beat" of the song is easily established by the listener. The audience will then mentally impose a timing framework on the sounds that reach them, so that what they perceive is a group of simultaneous tones which change pitch at fairly regular intervals.

The most natural approach to the automated musical recognition problem, then, is to devise an algorithm which "sees" music in this way. First, the program must ascertain the beat of the song similar to the way a human listener would. Then, once this has been established, it faces the task of analyzing each of the intervals to decide what notes are being played by what instruments in that interval. As will be seen, the first of these operations is best analyzed in the time domain, while the second lends itself principally to the frequency domain. This two-step plan for arriving at musical information forms the heart of the *Score* program, and will be outlined below.

Event Detection

A present-day computer can do many things well; unfortunately, enjoying a musical performance and "digging the beat" is not one of them. When the bare digitized signal is first received by our program, it appears not as a series of well-defined musical intervals, but as a long string of indistinguishable samples of varying amplitudes. Our first task, then, is to find some way to carve up this sequence meaningfully, so that we can discover where the notes it contains begin and end. Mathematically, if we let $\mathbf{X}=\{x_i\}$ for $i=1,2,\dots,N$ be our input data vector of N samples, we seek to find $\mathbf{V}=\{v_i\}$ for $i=1,2,\dots,M$, where $\{v_i\}$ is the set of indices into \mathbf{X} (ie., sample numbers) that correspond to the beginnings of the M notes in the song. (For

purposes of notation, we will designate our algorithm's guess at the event samples as $V' = \{v'_i\}$ for $i=1,2,\dots,M'$.)

A Naive Approach

One simple attempt to solve this problem would be to ask the user of the program to provide information about the tempo of the song (ie., the number of notes that were played per second), and to divide the data vector into uniform segments of the length of one note. Each of these segments could then be analyzed to determine the pitches present. Assuming that the input the program receives has some "dead space" before and after the actual music, it would be necessary to employ some elementary power threshold techniques to determine the precise beginning and ending of the song. We let $X_{\text{trim}} = \{x_\alpha, x_{\alpha+1}, \dots, x_\beta\}$, where x_α and x_β are the first and last samples, respectively, which are greater than a trim threshold th_t . Then we would compute the number of notes in the song as

$$M' := \left\lfloor \frac{\beta - \alpha + 1}{N_{\min}} \right\rfloor .$$

The quantity N_{\min} represents the minimum number of samples a note will contain, and is computed from the initial user input as follows:

$$N_{\min} = f_s \cdot \frac{60}{T} \cdot (b_{\min} \cdot 4)$$

where f_s is the rate at which the digitized signal was sampled, T is the tempo in bpm, and b_{\min} is the shortest note in the song, expressed as a musical fraction - for instance, $b_{\min} = 1/2$ would indicate that the shortest note was a half note, $b_{\min} = 1/16$ a sixteenth note, etc. Our approximation to the true event indices $\{v_i\}$, then, is merely

$$v'_i := \alpha + (i - 1) \cdot N_{\min} .$$

This solution is unsatisfactory for a number of reasons. In the first place, even presuming that all songs our program would wish to analyze would have unvarying tempos, the character of real music is not such that one can employ such mechanistic rules. A soloist - and indeed, an entire orchestra - will naturally speed up and slow down during various parts of a melody, which would skew the positions of the actual notes relative to the estimates this "equal segmentation algorithm" provided. Also, consider the problem if the beginning or ending of the song were not computed accurately. This is certainly possible if one is establishing the endpoints based on a crude power threshold, because musical pieces vary widely in volume, and hence their signals vary widely in amplitude. In trimming the input vector, we might easily mistake a modest noise spike for the first note in a song, or even unwittingly trespass on a pianissimo ending. In this case, not only the *positions* of the crudely determined intervals, but their *widths* become suspect. There is every possibility that each of the segments we analyze will actually contain parts of two or more of the real notes in the song.⁵ Additionally, we have the problems that realistic songs have dynamically changing tempos, that the user often will not know the tempo accurately, and that with this approach we are really asking the user to do some of the work that the program ought to be doing. And even more troublesome from a conceptual viewpoint is the fact that this sort of division into uniform segments does not begin to imitate what human beings do when they listen to music. No one even hears a whole song in an instant, let alone mathematically partitions it into artificial segments regardless of their actual content. For all these reasons, it is clear that a better solution is needed to determine the positions of the notes.

Power Threshold Detection

What we would like is a method of traversing through the song, sample by sample, searching for the beginnings of notes. This is doubtless the way the human being analyzes music: by simply waiting and listening until he or she notices that tones are present. We will refer to these "beginnings of notes" $\{v_i\}$ as *events*, and to the process of searching for them as *event detection*. This approach looks much more promising than the aforementioned equal segmentation algorithm, because it has none of the drawbacks. Since we will move linearly through the song, analyzing it sample by sample, missing one event should not impact those before or after it. We will be able to naturally adjust to changes in tempo, because we are not artificially imposing a lattice of intervals over the entire song. And this solution requires no great degree of accuracy in determining the first and last sample in the actual song. The question now simply becomes: as we progress through the sound samples, what sorts of things should we look for? What criteria should we adopt for judging where the true events are?

Perhaps the most natural inclination would be to zero in on the note attacks we saw in chapter 2. Each note had some sort of rapid rise in amplitude at its beginning (though the rate of this ascent was greatly dependent on the type of instrument), so it would seem that merely monitoring the power of the signal as the song progressed would give us a very good idea of where the true events were. Whenever we saw a sudden increase in the magnitude of the samples, we would assume that we had discovered the beginning of a note.

Implementing this idea is a bit more complicated than that, because the graphs of the instrument volume we saw in the previous chapter were really volume *envelopes*. The actual signal, of course, is oscillating with a frequency characteristic of its pitch, and only the *peaks* of this oscillation are described by these envelopes. If we

merely watched for an increase in raw magnitude, we would detect numerous false events in the middle of every note, because of the strongly periodic character of a tone. Such an obstacle can be easily overcome, however, by adopting a scheme similar to that described in [2] where one compares the *average* signal power over some small interval with the power of the recent past to determine if an event is present. Figure 8 depicts the algorithm graphically. For each sample k of the signal, we compute the average signal power both over the most recent S_{avg} samples and the most recent L_{avg} samples. The first of these averages we will call the STA (for "Short Term Average" power), and the second the LTA (for "Long Term Average" power). If we use only these averages for event detection rather than the magnitudes of individual samples, we should succeed in filtering out the periodic nature of notes and getting a good estimate of when the attacks of notes occur. The procedure is simply to take the ratio of the two power averages, and compare it to some threshold th_p :

$$\frac{STA}{LTA} > th_p$$

An appropriate value for the threshold must be determined by experiment. If it is exceeded, then that should indicate that the power of the signal in the proximity of sample k is significantly higher than the power for the time immediately preceding it, and we include k in V' .

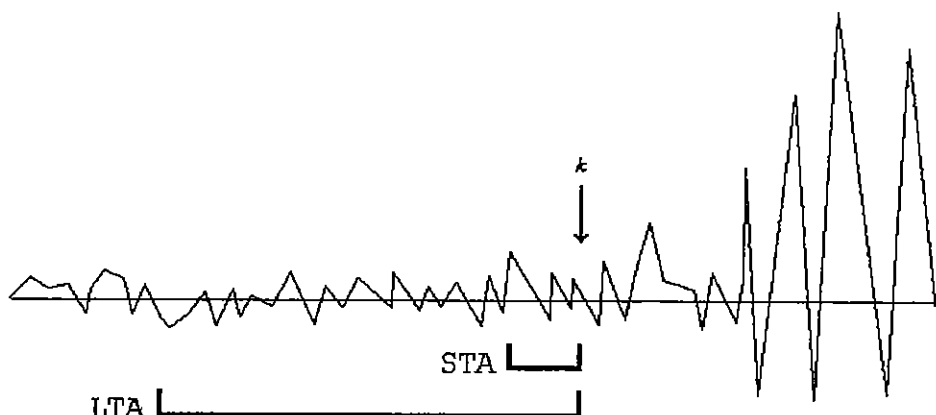


Figure 8. STA/LTA power ratio threshold technique. For each data point in the signal k , the average power over the smaller interval is compared to the average power over the longer interval. If the STA/LTA ratio is above a certain threshold, an event at sample k is predicted.

Unhappily, this method is still not sufficient by itself in practice. Its downfall comes from the fact that in real music, individual notes are not always accompanied by a significant amplitude peak. This is because once a musical phrase is begun by a performer, the attacks of notes may be hidden beneath the volume level sustained by the phrase as a whole. Especially in passages performed in a *legato* style, a change in pitch may be the only noticeable effect when a new note is played. Figure 9 contains the waveform for such a passage. As is easily seen, the beginning of the second note in the phrase does not correspond to any significant rise in signal amplitude; in fact, it is barely visible in the plot. The bottom half of the figure shows the value of the STA/LTA ratio for each of the samples in the song, and it is clear that any peak resulting from the second note lies buried far below the point where any threshold would be safely crossed. When the song represented is played for a human listener, it is clear that a new note has been played because one detects a change in pitch; but the

STA/LTA ratio technique is powerless to track such changes. One could envision tweaking the M , N , and th_p parameters just to the point where the peak would be singled out and no false events would be detected; however, the general-purpose utility of an algorithm that required such fine tuning for every input is doubtful, and in any case this calls into question whether the method is really the best suited for the problem. We are facing a bigger issue here: in reality, a volume attack is not the only feature which may signal a new note, and therefore, this technique will only find a subset of the events if used in isolation.

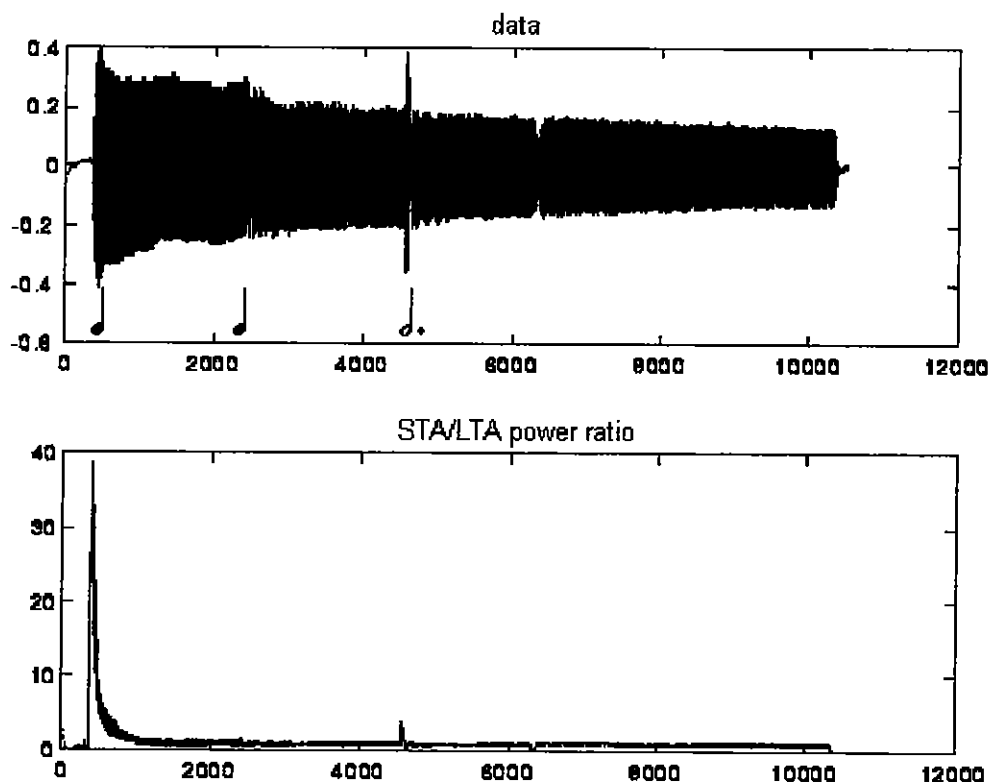


Figure 9. The insufficiency of the STA/LTA power ratio algorithm to properly identify events by itself. On the top graph is an audio signal of a three-note melody played by a synthesized instrument. The three notes of the melody are shown immediately below the points at which they were played in the recording. Notice that the second of the three has no significant amplitude peak at its beginning; the only thing signalling the listener to the fact that a new note has begun is a different

pitch. On the bottom graph is the corresponding STA/LTA power ratio taken at each point of the sample. It is clear that although peaks are seen for the first and third notes, and could be detected by a threshold test on this ratio, the event associated with the beginning of the second note lies buried, and could never be properly identified using this technique.

Tracking the Error of an Adaptive Linear Predictor

It appears, then, that we will need the STA/LTA ratio to find volume attacks, another algorithm to detect changes in pitch, yet another to find variations in quality, and so on until we have exhausted all of the possibilities for what might constitute a "new note" audially. However, a simpler approach can be found if one considers the general notion of a note as it strikes the listener. More than a change in one particular characteristic such as volume, pitch, or quality, a new note really constitutes *any* kind of change. Speaking loosely, when we listen to a song and perceive that the music is "holding steady," we would naturally write down a group of long notes if asked to provide the score. It is when changes or "surprises" occur in the music that we view them as signalling distinct musical events, and hence, new notes. This suggests that we might use an algorithm which didn't hone in on one particular attribute of the music, but which simply informed us when the character of the piece was seen to have "changed" in a significant manner.

This concept leads us to the method of event detection utilized by the *Score* program: tracking the error of an adaptive predictor. Adaptive predictors are used in a wide variety of applications to perform event detection: for instance, tracking inconspicuous but meaningful seismic activity, monitoring the sound waves in a secured area for faint footsteps, or extracting data transmissions in the presence of background noise. The idea is that the predictor, which is really an adaptive digital filter, analyzes a signal and "learns" what to expect of its input so that it is sufficiently

"surprised" when the character of the input changes. It accomplishes this by maintaining a series of weights which it multiplies by its input samples, and then adjusts to minimize the error of its prediction for the value of the next sample. After only a small amount of data, the predictor will have adjusted to the input so that its error is very small. When an unexpected change in the data occurs, such as a new note sounding that a human being would notice, the error becomes very large for a short period of time while the predictor adapts to it. Monitoring the predictor's error while allowing it to perform its function, then, would seem to be an appropriate way to uncover notes in the midst of an audio sample.

The particular algorithm implemented by the *Score* program is the well-known LMS (Least Mean Square) adaptive linear predictor described in [13]. Briefly, it operates as follows. The adaptive filter consists of a set of L modifiable weights⁶ which, at each iteration of the algorithm, are multiplied by the L most recent samples of the data, and added together to yield a prediction of the next data sample. Symbolically, for the k th data point, we specify the predictor's output y_k as

$$y_k := \vec{\mathbf{X}}_k^T \cdot \vec{\mathbf{W}}_k$$

where \mathbf{X}_k is a column vector of the most recent L data points $\{x_{k-L}, x_{k-L+1}, \dots, x_{k-1}\}$, and \mathbf{W}_k is the column vector of weights. The error of this prediction ε_k is simply the difference between it and the actual value of the data point, x_k :

$$\varepsilon_k := x_k - \vec{\mathbf{X}}_k^T \cdot \vec{\mathbf{W}}_k$$

The adaptive predictor problem is to compute the set of weights for successive data points such that the error in the prediction becomes as small as possible. Many methods are available for achieving this goal; most of them involve forming some estimate of the gradient of the predictor's performance surface. (This gradient is simply a vector containing the partial derivative of $E[\varepsilon_k^2]$ with respect to each of the

weights.) The LMS algorithm represents the simplest approach to estimating this gradient: just take the square of the predictor error *itself*, ε_k^2 , as an approximation to $E[\varepsilon_k^2]$. The result is an algorithm which is somewhat less than optimal in minimizing the error (since it is based on rather crude estimates of the true gradient), but which is elegant and straightforward to compute.

Specifically, for each point k , we compute the set of weights for the next data point by adjusting our current set of weights as follows:

$$\vec{W}_{k+1} := \vec{W}_k - \mu \cdot \hat{\nabla}_k$$

where μ , the coefficient of the gradient estimate, is a gain constant which must be carefully chosen. By using ε_k^2 as our approximation to $E[\varepsilon_k^2]$, our expression for the gradient reduces to

$$\hat{\nabla}_k := \begin{bmatrix} \frac{\delta \varepsilon_k^2}{\delta w_{k,0}} \\ \vdots \\ \frac{\delta \varepsilon_k^2}{\delta w_{k,L}} \end{bmatrix} = 2 \cdot \varepsilon_k \cdot \begin{bmatrix} \frac{\delta \varepsilon_k}{\delta w_{k,0}} \\ \vdots \\ \frac{\delta \varepsilon_k}{\delta w_{k,L}} \end{bmatrix} = -2 \cdot \varepsilon_k \cdot \vec{X}_k$$

and the set of weights for each progressive iteration of the algorithm becomes simply

$$\vec{W}_{k+1} := \vec{W}_k + 2 \cdot \mu \cdot \varepsilon_k \cdot \vec{X}_k$$

In other words, our adjustment to the weights is simply a scalar multiple of the data window. Conceptually, the greater the error ε_k , the faster the weights will change to reduce it, and when the error grows small, the predictor "settles down" and makes only small changes in response to the input. The gain parameter μ governs the whole process by dictating at what rate the predictor will shift its weight values. Obviously,

the factor of 2 may be absorbed into the arbitrarily defined gain constant, so that an extra multiplication is not necessary at each iteration.⁷

The purpose of implementing the adaptive linear predictor is not to get an accurate prediction of the next sample and make use of it; after all, we are given the entire data vector at the start of the problem. Rather, we are interested merely in the value of the squared error ε_k^2 at each data point k . Therefore, the only output from the linear predictor algorithm that *Score* uses is a data vector \mathbf{E} composed of the squared errors for each data point prediction. Our hope is that $\mathbf{E}(k)$, the squared error of the predictor at point k , is small for $k \neq v_i$, but that it rises sharply for $k = v_i$.

A Combination of the Two Methods

In practice, the error vector produced by the linear predictor does in fact rise coincident with note beginnings. After testing the algorithm on sample music data, though, it appeared that \mathbf{E} itself is not the best entity to perform a threshold test on. The trouble is that depending on the parameters of the linear predictor, the error even near events can be quite small. In many cases, the error peaks are so short that it is extremely difficult to establish the proper threshold such that true events will exceed it and false alarms will not. Extracting small peaks from a signal, however, is precisely what the previously described STA/LTA power ratio technique excels at. By using the predictor's error vector \mathbf{E} rather than the input signal \mathbf{X} as data for the STA/LTA algorithm, excellent results for event detection are possible. (See Figure 10.) During laboratory experiments, it was very rare for *Score* to fail to identify the beginning of a note once appropriate settings for the predictor and the STA/LTA windows were selected; and false alarms, though more frequent, were to a great degree eliminated.

It is useful, though, to combine this signal processing wizardry with some musical know-how in order to improve performance even further. *Score* allows the

user to input the approximate tempo of the song (in beats per minute, bpm) so that it may apply additional heuristics after the above event detection techniques have been completed.⁸ In particular, false alarms occurring immediately after true events are suppressed by enforcing a minimum length of time between events, subject to a certain relaxation factor. Basically, a procedure scans through the vector V' of supposed events and eliminates any that occur too close in time to their preceding neighbor. (See Figure 11.) Mathematically, for a relaxation factor $0 < r \leq 1$, the i th predicted event which corresponds to index v'_i is rejected if

$$v'_i - v'_{i-1} < N_{\min} \cdot r, \quad 2 \leq i \leq M'.$$

The relaxation factor r is necessary in case the user's guess at the tempo was inaccurate, or if the song contains an actual tempo change. In the lab, a relaxation factor of 65% of the user's tempo input was found to be satisfactory, and successfully reduced the small number of spurious events left over from the adaptive predictor / STALTA scheme. The only downside to this particular technique is that if a false alarm occurring just *before* a true event were recognized, the algorithm would view the succeeding true note as a false event that needed to be suppressed. If enough false alarms were left over from the predictor, then the entire song could get off-track and only false alarms would be recognized. (See illustration in Figure 11b.) However, the first note of the song by definition has no musical events before it, and therefore is recognized accurately nearly all the time. And in practice, once the first note has been established, this heuristic inevitably masks out false events instead of true, so the pitfall does not often materialize itself. Still, it is a consideration, and could be eliminated at the cost of a few more false alarms.

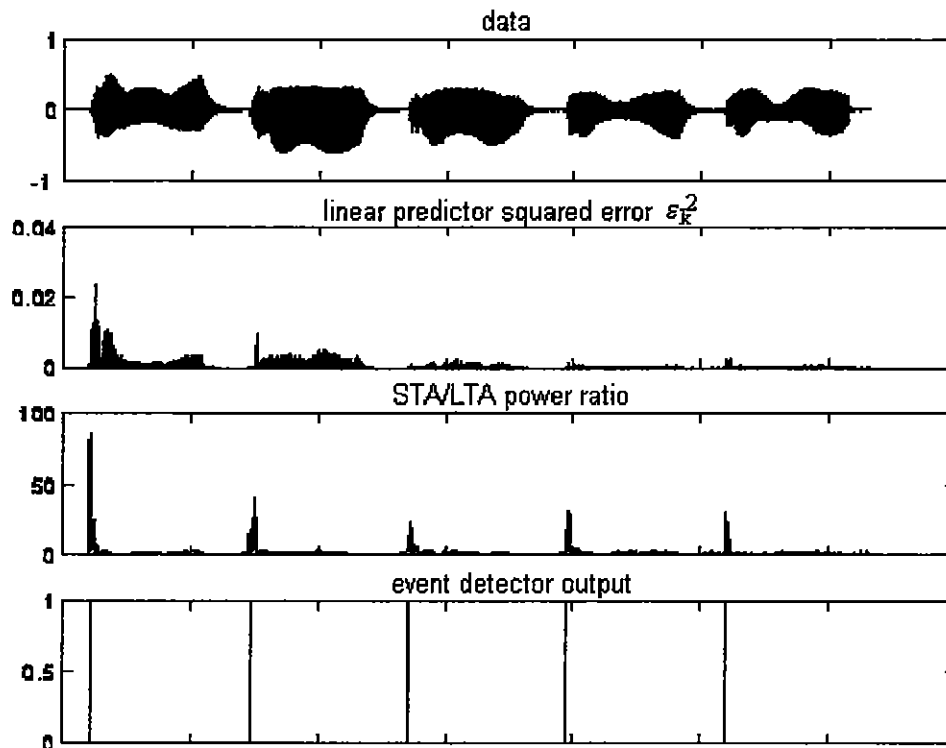


Figure 10. The combination of the linear predictor error with the STA/LTA power ratio test is sufficient to detect events. On the top graph is the data sampled from a five-note melody performed by a french horn. The second graph shows the linear predictor's squared error at each data point. Note that although there is a rise in ϵ_k^2 accompanying each note, this rise is not restricted to the *beginning* of each note; in fact, the squared error is sometimes higher in the middle of the note than it was at the beginning. Therefore, a threshold test on the squared error alone is insufficient to detect the true events properly. The third graph, however, shows the results of using the squared error vector \mathbf{E} as input to the STA/LTA power ratio algorithm. This filter succeeds in picking out the starting points of the notes and suppressing the magnitude during the middle of the notes. Therefore, a threshold test can be applied to this output to yield a good \mathbf{V} vector. The results of the threshold test are shown in the bottom graph; peaks indicate guesses at the true events.

Event Detection - Summary

The overall strategy for event detection, then, is as follows. First, we feed the input vector to an adaptive linear predictor and compute the squared error of the predictor at each sample. Then, turning our attention to this error vector rather than the input data, we compute the STA and LTA powers for each point, and use the results to build another vector. We then apply a threshold to this vector, and identify all indices whose STA/LTA ratio exceeds it. Finally, we eliminate some of these indices which are likely to be false by enforcing a minimum time interval in between each pair of events; any sample with a sufficient STA/LTA ratio that occurs too soon after another sample with such a ratio is removed from the vector of events. The final result is a set of M' indices $v'_1, v'_2, \dots, v'_{M'}$ into the original data vector \mathbf{X} that forms a very good approximation as to when the notes in the music began.

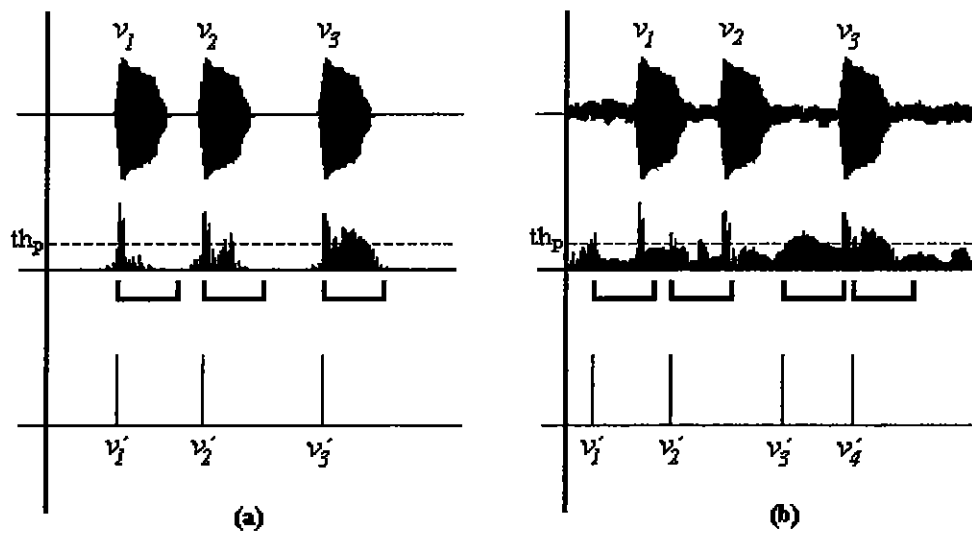


Figure 11. False alarm elimination heuristic. On the top graph in (a) is the digitized data for three hypothetical musical notes. The second graph contains the STA/LTA power ratio of ϵ_k^2 for each point. The dotted horizontal line represents the threshold th_p applied to the power ratio to determine events. False alarm elimination is performed on this output, however, by masking out any predicted events that occur too

close to their previous neighbor. The thick brackets immediately below the second graph indicate the area in which events are suppressed; their length in samples is equal to $N_{\min} * r$, where r is an experimentally chosen relaxation factor. The result is that the false events where the ratio exceeded th_p during the middle of the note are masked out, and the resulting $\{v'_i\}$ estimates are extremely close to the actual $\{v_i\}$ events. (b) The possible danger of using this algorithm. If the STA/LTA power ratio is often above the threshold (due to an extremely noisy or poorly-behaved signal, for example) then true events might be masked out instead of false ones. In practice, however, this hazard rarely materializes.

Pitch Extraction - Compiling the Frequency Domain Information

Once the events (ie., note beginnings) in a song have been properly identified, the next phase in the *Score* algorithm is to analyze the frequency content of the established intervals to determine what tones, if any, are present. The first step in accomplishing this goal is to compute the Fourier Transform of the data, so that the power of the various frequency components of the signal can be readily identified and dissected.

Clearly, taking the overall transform of the entire music signal will not yield any useful information, because the Fourier Transform makes sense only on signals that are at least locally *periodic*. A given song is composed of a long sequence of connected notes, each of which carries unique frequency information related to its pitch. What we need, then, is an incremental transform performed locally within each of the intervals identified in the event detection phase, so that we may isolate and analyze only the frequencies contained in each of the individual notes. The first question to be asked is, where during the interval should the transform be taken? Should all of the samples comprising the given note be used, or only a portion, and if so, which portion?

A purely performance-related issue constrains us somewhat. If we compute the Fourier Transform of a number of points which is an even power of two, then we can employ the Fast Fourier Transform (FFT) algorithm, and significantly reduce the total computation time.⁹ Another consideration is that the larger the number of data points we use, the more frequency bins will be available from the transform, and consequently, the better our resolution in the frequency domain will be.¹⁰ Hence, we desire to use the greatest even power of two that is less than the allotted number of samples for the note. In general, though, the notes in a song will be of varying lengths: some will be whole notes, some half-notes, quarter-notes, eighth-notes, etc. The next issue, then, is whether to change the size of the transform depending on the length of each interval, or whether to consistently use the size that would fit in the smallest interval. The effect of lengthening the transform in the larger intervals would be to increase the FFT resolution for only *some* of the notes. It is not clear whether this would be advantageous, especially in light of the global nature of *Score*'s subsequent pitch extraction algorithm. After all the FFT's have been taken, the algorithm will compare the magnitudes of peaks from interval to interval, and so it seems natural to maintain a uniform size so we can compare "like to like." Hence, the program as currently implemented computes the largest even power of two that is less than N_{\min} , and uses this for T_{SZ} , the transform size.¹¹

We then face the question of *which* samples within the interval should be used to compute the FFT. Recalling our discussion in the previous chapter about the characteristics of striking and of continuous instruments, it seems that we might want to do something different in each case. For continuous instruments, a significant power level is typically maintained throughout the duration of the note, so we may desire to take the transform of the samples centered around the midpoint of the interval. This way, any noisy or atonal effects produced by the note's attack (eg., the

violin bow catching, the flautist's breath before a pure tone is achieved) can be avoided and the FFT can be taken over the heart of the richest harmonic portion of the note. For striking instruments, though, the intensity of the sound decays as the note is played, so we might rather take the FFT of the samples near the *beginning* of the interval, in order to maximize the frequency domain power.

After experimentation on songs which contain only one *kind* of instrument (eg., three violins, or two harps), it turned out that centering the FFT within the interval yields excellent results, even for striking instruments. The decay of the latter is rarely so steep that the resulting harmonics cannot be easily located in the frequency domain. Mathematically, then, to extract the pitches from event i , we would take the FFT of the vector \mathbf{X}_i , where

$$\vec{\mathbf{X}}_i := \left\{ \mathbf{x} \left(v_i + \frac{v_{i+1} - v_i}{2} - \frac{T_{SZ}}{2} \right) \cdots \mathbf{x} \left(v_i + \frac{v_{i+1} - v_i}{2} + \frac{T_{SZ}}{2} - 1 \right) \right\}.$$

Score used this approach for all the results in Chapter 4 (songs containing only the simplified model), and for the solo instrument results in Chapter 5. In all these cases, centering the FFT within the interval proved to be a prudent method. The strategy was found to be lacking, however, when different *kinds* of instruments were present in a song. More specifically, when a song contains *both* striking and continuous instruments, the FFT of the vector above will often be dominated by the frequencies of the continuous instruments. This is simply because continuous instruments are still more or less at their maximum intensity throughout the interval, while the striking instruments have decayed significantly. The notes played by the latter can be simply lost, especially during the longer intervals, as we will find in Chapter 5. For this reason, a modified strategy is often more useful when analyzing such "mixed-instrument" recordings. For the shortest interval(s) in the song, we center the FFT, as before. For the longer intervals, however, we place it *the same*

number of samples after the starting point of the interval as we did for the shortest interval(s). The result is that we take the FFT of the vector \mathbf{X}_i , where now

$$\vec{\mathbf{X}}_i := \left\{ \mathbf{x} \left(v_i + \frac{N_{\min}}{2} - \frac{T_{SZ}}{2} \right) \cdots \mathbf{x} \left(v_i + \frac{N_{\min}}{2} + \frac{T_{SZ}}{2} - 1 \right) \right\}.$$

For all intervals I_i , then, we take an FFT of size T_{SZ} starting the same number of samples *after* the beginning of the interval, v_i , regardless of the size of the interval. As we will see, this proves to be a fair compromise between the desire to avoid the noisy qualities of the attack, and the necessity to capture the frequency information of striking instruments before too much of it decays away. This also will tend to make the harmonic signatures we retrieve from the frequency domain more stable. Recall from Chapter 2 that the signature of a particular instrument can vary with time as the note resonates. By taking the FFT the same amount of time after the beginning of each note, we will avoid some of the difficulties presented by this phenomenon.

Pitch Extraction - Analyzing the Frequency Domain Information

After the FFT of an interval has been taken comes the much more difficult problem of extracting the pitches of notes from the sea of harmonic peaks. As we discovered in Chapter 2, each note of each instrument may contribute a half-dozen or more strong frequency components to the total picture we see in the FFT. The basic problem we are presented with is identifying which peaks represent fundamental frequencies of notes, and which are simply harmonic frequencies of some lower fundamental. Once we have recognized the true set of fundamental frequencies in each of the intervals, we must associate each one with a particular part in the song as a whole. (In other words, the trumpet notes from each interval must be grouped into one part, and the clarinet notes into another part, etc.) We refer to this total process

as *pitch extraction*, and as we will see, it requires a very complicated algorithm to attempt to resolve the inherent ambiguities.

We have been using terms like "voice" and "instrument" rather loosely up to now, and before we consider the pitch extraction problem in detail, it is sensible to define our terminology more precisely. (The exact meanings of these and other terms is summarized in Table 1.) We will use the word "instrument" to refer to a real-world object capable of producing one or more simultaneous tones. This is likely to be the way a user of the *Score* program would think about the music signal: as composed of a number of instruments. (The relevant questions are, "What are the clarinets playing? What is the harpsichord playing?" etc.) From the viewpoint of a pitch extraction algorithm, however, we have to consider the problem space a bit differently. Each instrument may produce one *or more* tones in a given interval, and in general it will not be obvious to the algorithm which sets of tones came from a common instrument. For example, if during the first beat of the third measure of a song there is a piano playing four notes, and a flute soloist playing one note, all the program will be able to "see" is five distinct notes. It has no idea without a good deal of extra processing whether they were produced by five different instruments, two, or only one. Hence, the term "instrument" to refer to the producer of a single note in an interval is misleading. Instead, we will use the term "voice" to indicate the producer of an individual note. (In the previous example, then, there are five *voices* present in the given interval, not five *instruments*.) We will also find it necessary to refer to the *sequence* of notes throughout a song that are all associated with a given voice. This is called a *part*. For instance, in a song featuring a trumpet, bassoon, and oboe, there are three *parts*: one for each instrument. In the case of a polyphonic instrument like a piano, the notion of a "part" is somewhat artificial; however, we can still satisfy our definition as follows. If J is the maximum number of notes the polyphonic instrument

plays anywhere in the song, then we will simply say that the instrument is composed of J parts, some of which have rests during the intervals in which it plays fewer than J simultaneous notes. To illustrate, if we have a piano which plays at most five notes simultaneously in a given song, then we will assign five *parts* to the piano; perhaps calling them "the lowest piano part," "the second lowest piano part," etc. During an interval in which the piano is playing only three notes, we will say that the parts called "the fourth lowest piano part" and "the highest piano part" have rests. Note that if the output of the *Score* program were put into musical notation, all the notes that make up a single part would be combined into one staff; though a given staff may contain the notes from *more* than one part, as would be the case with our J piano parts. Hence, the terms "instrument," "voice," and "part" all refer to related, but distinct concepts. One of the questions that will come up repeatedly in the pitch extraction algorithm is, "This song has J parts in it, and a certain interval has J_i voices present. Which of the voices correspond to which of the song's parts?"

Table 1. Terminology used in the discussion of the pitch extraction algorithm.

instrument - A real-world object capable of producing one or more simultaneous tones.

voice - The producer of an individual note that appears in the song. Observe that in any interval, a given instrument may be responsible for only one voice (eg., a trumpet), or many voices (eg., a piano.)

part - The sequence of notes throughout a song which all come from the same voice. For instance, we may refer to "the trumpet part" as the series of notes (and possibly rests) that correspond to the trumpet voice.

harmonic independence - Two sets X and Y containing frequencies are harmonically independent iff no frequency in X is an integer multiple of a frequency in Y .

harmonic irreducibility - A set X containing frequencies is harmonically irreducible iff every frequency in X is an integer multiple of the lowest frequency in X .

harmonic decomposition - The process of taking a general spectrum S and representing it as a group of harmonically independent, harmonically irreducible sets Q .

harmonic ambiguity - A condition arising where the *fundamental* frequency of a particular voice is very close to the harmonic frequency of another voice. In this case, the first voice will be hidden in the Q set after a straightforward harmonic decomposition of the spectrum.

upper harmonic ambiguity - A condition arising where *any harmonic* frequency of a particular voice is very close to the harmonic frequency of another voice. In this case, the harmonic signatures of the two voices will "overlap." The two fundamental frequencies will be included in their own distinct Q sets, but neither Q set will properly represent the harmonic signature of either voice.

interval - a region in time which begins at the starting point of some note in the song and ends at the starting point of the next note. These intervals are identified by the event detection algorithm, which is completed before any pitch extraction takes place.

well-behaved interval - An interval in which all of the parts present in the song appear unambiguously. Therefore, (a) every voice is represented in the interval, and (b) no harmonic ambiguity exists in the interval.

poorly-behaved interval - An interval in which *not all* of the parts present in the song appear unambiguously. This means that either (a) one or more of the voices is resting during the interval, or (b) the interval contains harmonic ambiguity.

Basic Concepts

We are now ready to consider the problems involved in pitch extraction. First, let us develop some notation so that we can formulate our ideas precisely. (This notation is summarized in Table 2, at the end of this chapter.) Let I_i , $i=1,2,\dots,M$ denote the i th interval that was identified in the event detection phase; namely, that which contains data values

$$\left\{ \mathbf{x}_{\left(\begin{smallmatrix} v_i \\ i \end{smallmatrix}\right)} \cdots \mathbf{x}_{\left(\begin{smallmatrix} v_{i+1} - 1 \\ i+1 \end{smallmatrix}\right)} \right\}.$$

For simplicity, we will assume that our event detection was error-free ($\mathbf{V}' = \mathbf{V}$), and that we will manually insert an event $v_{M+1} = x_N$ at the end of the data so that the last note of the song will have an interval which satisfies this definition. We examine the

frequency content of each of these M intervals as previously described, in order to acquire a set of peak frequencies and their corresponding amplitudes. In actuality, this is somewhat complicated, for two reasons. First, the song may contain changes in volume, so the threshold for what constitutes a "peak" frequency must be dynamic. The *Score* algorithm handles this by simply computing the threshold for each interval as a function of the power in that interval, subject to a certain minimum which allows for the proper identification of rests. Secondly, the actual peaks corresponding to fundamental and harmonic frequencies have non-zero widths, which means that extra processing is required to isolate the true peak frequency from each contiguous series of FFT bins that exceed the threshold, and to combine the power from the true peak and any leakage bins to yield an accurate amplitude. After this processing has been completed, we have for each I_i an ordered set S_i of ordered pairs of peak frequencies and amplitudes:

$$S_i = \{ \{ f_1, a_1 \}, \{ f_2, a_2 \}, \dots, \{ f_K, a_K \} \}.$$

where $f_1 < f_2 < \dots < f_K$. We refer to this ordered set as the *spectrum* for interval I_i .

Suppose that in interval I_i , there are J_i different voices present. This S_i , then, contains peak frequencies from J_i voices, each voice with its own fundamental frequency and series of harmonics. To uncover the pitches present in interval I_i , our task is to decompose S_i into sets $Q_{i,j}$, where $Q_{i,j}$ has the same structure as S_i except that it contains only the peak frequencies that were contributed by part j of the song. Then, the pitch corresponding to the lowest frequency in each $Q_{i,j}$ represents a note that is present in interval I_i . All that remains is to associate each of these notes with one of the parts in the song as a whole. Note that the $Q_{i,j}$'s are not *subsets* of S_i in the usual set theory sense, because two voices may contribute amplitude to the same peak frequency f_k , in which case their a_k 's will add to form the amplitude in S_i .

Consider a melody played by a solo instrument. For each of the intervals I_i , no more than one voice will be present. (Zero voices will be present if the instrument has a rest during the interval.) Since $J \leq 1$, we have at most one set Q with $Q_{i,1} = S_i$. We can safely deduce, then, that the lowest frequency in the set S_i (denoted $S_i|f_1$) is the fundamental frequency of the note present, if any. We output the pitch corresponding to that fundamental frequency. This is straightforward and involves no ambiguities.

Now suppose there are two or more voices playing in a given interval, but that none of them "overlap" in their harmonic spectra. In other words, none contribute harmonic frequencies which are present in any of the others, and we have $Q_{i,1} \cap Q_{i,2} \cap \dots \cap Q_{i,j} = \emptyset$, if we take the intersection with respect to the frequencies only (*ie.*, we ignore the amplitude part of each ordered pair.) Again, this situation presents no ambiguities, because it is an easy matter to extract each of the $Q_{i,j}$'s. The procedure is to take the ordered pair with the lowest frequency f_1 in S_i and include it in $Q_{i,1}$. We can now "strike it out" of S_i because we know that it is not present in the spectra of any other part j . We then examine each other ordered pair in S_i and include it in $Q_{i,1}$ iff it is harmonically related to f_1 ; namely, only if its $f_k = n \cdot f_1$, for some integer n . After we finish this for all ordered pairs in S_i (striking out the pairs which we include in $Q_{i,1}$), we have effectively extracted voice 1 from I_i , and we repeat the procedure with the remaining ordered pairs in S_i . When $S_i = \emptyset$, we have successfully derived our $Q_{i,j}$'s, and we can determine the pitches as we did in the solo instrument case. We refer to this process of forming the $Q_{i,j}$'s as the *harmonic decomposition* of S_i , and we say that the resulting $Q_{i,j}$ sets are *harmonically independent*; that is, no frequency in any Q set of an interval is an integer multiple of a frequency in any *other* Q set of that interval. These Q sets are also *harmonically irreducible*, which means that each frequency in $Q_{i,j}$ is an integer multiple of the lowest frequency in $Q_{i,j}$.

Let us now consider more carefully our requirement that $Q_{i,1} \cap Q_{i,2} \cap \dots \cap Q_{i,J} = \emptyset$. If we want each of the Q sets to contain only the frequency contributions from one instrument, and no others, we do in fact need to impose that requirement. But if our goal is simply to uncover the *pitch*s of each of the notes, then it is actually overly strict. In order to unambiguously derive the pitches present in S_i , it is necessary only that the *fundamental* frequency of each $Q_{i,j}$ set is not "covered" by a frequency component in another $Q_{i,j}$. We can see this by examining the above algorithm carefully. Suppose $J=2$ and that in interval I_1 we have one voice contributing frequencies at 100, 200 and 300 Hz, while the other voice contributes frequencies at 150 and 300 Hz. We assume for simplicity that all frequency contributions from all voices are of equal magnitude. This results in

$$S_1 = \{ \{100,1\}, \{150,1\}, \{200,1\}, \{300,2\} \}.$$

It is clear that the information about which voice contributed how much amplitude to the 300 Hz bin has been lost. However, in order to properly identify the pitches of the two notes, we need only guarantee that neither *fundamental* frequency has been thus obscured. We see that applying the extraction algorithm yields

$$Q_{1,1} = \{ \{100,1\}, \{200,1\}, \{300,2\} \} \quad \text{and} \quad Q_{1,2} = \{ \{150,1\} \},$$

and we correctly deduce that the two pitches present are at 100 and 150 Hz. Observe carefully that neither $Q_{1,1}$ nor $Q_{1,2}$ actually contain the frequency contributions from a particular voice any longer. In particular, $Q_{1,1}$ contains not only the frequencies from voice 1, but also the 300 Hz component from voice 2. And $Q_{1,2}$ has only the *fundamental* frequency of voice 2, while its second harmonic has been "stolen" by $Q_{1,1}$. However, it is easy to see that this does not prevent us from correctly determining the *pitch*s of the two notes. Hence, we can say that as long as no voice's fundamental frequency "lands on" another voice's harmonic, pitch extraction can be performed without any ambiguities. Since the second harmonic of a note is musically

an octave higher, situations such as these arise whenever two voices are closer than an octave in pitch. An example would be most piano chords that are played by one hand - all the notes played simultaneously are typically within an octave of each other. An analysis of the *Score* program's behavior on such musical pieces is given in the section on the "Soprano-Alto problem" in Chapter 5.

The Problem of Harmonic Ambiguity

Unfortunately, pitch extraction is not always this straightforward. Problems arise, of course, when the fundamental frequencies of certain voices *are* obscured by the harmonics of other voices. Consider the previous example once more, only let the second voice contribute frequencies of 200, 400, and 600 Hz. Now, we have

$$S_1 = \{ \{100,1\}, \{200,2\}, \{300,1\}, \{400,1\}, \{600,1\} \}.$$

The second voice's fundamental of 200 Hz has been "hidden" by a harmonic of the first voice, and so our harmonic decomposition algorithm yields

$$Q_{1,1} = S_1 = \{ \{100,1\}, \{200,2\}, \{300,1\}, \{400,1\}, \{600,1\} \}.$$

The algorithm "sees" only one voice. We are faced with an inherent ambiguity: this simplified algorithm has no choice but to assume that all frequencies which are harmonically related to 100 Hz were in fact produced by the voice at that fundamental. One might argue that the magnitude of the 200 Hz component should be enough to alert the program that a voice has been hidden, but this presupposes fairly precise knowledge about the harmonic signatures of the instruments involved. It is certainly possible that a single voice might well have a second harmonic with twice the amplitude of the first, and therefore, it is unlikely that this sort of indication could ever be used reliably in practice.

We say that intervals such as the one above contain *harmonic ambiguity*. To provide a concrete definition, harmonic ambiguity is a condition arising where the

fundamental frequency of a particular voice is very close to the harmonic frequency of another voice. In this case the first voice will "hide" behind the second voice in the S_i set, and a straightforward decomposition into $Q_{i,j}$ sets will result in both voices being included in the same set. Note that our definition requires the *fundamental* frequency of a voice to be thus obscured. There are also situations where two voices have fundamental frequencies that are not harmonically related to one another, but where each of them have *upper* harmonics that overlap. For instance, in our previous example, the third harmonic of 100 Hz was equal to the second harmonic of 150 Hz, even though the pitches of the two simultaneous notes were unambiguously resolved. To distinguish this situation (which presents its own problems) from the former, we will refer to it as *upper harmonic ambiguity*.

(Before we continue, let us address a possible point of confusion regarding our notation. Earlier, we spoke of the $Q_{i,j}$ sets as each containing the significant frequency components of a particular voice. Now, however, we will use this notation to refer to the harmonically independent subsets of the spectrum S_i , as obtained by a straightforward harmonic decomposition. *If there is no harmonic ambiguity or upper harmonic ambiguity in a particular interval*, then these $Q_{i,j}$ sets will indeed each contain the peaks from one and only one voice. In general, though, some components of some voices will have been "stolen" by Q sets that do not correspond to those voices. We could have used a notation such as $Q'_{i,j}$ to refer to these sets which may or may not each represent one voice, but to simplify the subsequent discussion, it seemed prudent to drop the "prime" and refer to them simply as $Q_{i,j}$.)

An Approach for Resolving Harmonic Ambiguity

Score's approach to resolving harmonic ambiguity is to make use of its knowledge about the song as a whole. The idea is that even though in a particular

interval I_i we might have a part obscured by other parts, it is likely that over the course of an entire song, there will be other "well-behaved" intervals in which we will be able to see all parts. For purposes of this thesis, I have made two simplifying assumptions in order to facilitate this approach, each of which could be analyzed to adapt the algorithm to other situations. First, I make the basic assumption that every song will have at least one well-behaved interval in which all voices appear unambiguously. Second, I assume that the same set of instruments is present in the song throughout the period analyzed. This does not mean that a particular part cannot rest for one or more intervals, but it does mean that we will not have, for instance, a flute in the beginning of the song which is replaced by a trumpet towards the end. Such an assumption is essential for the harmonic signature averaging portion of the algorithm.

An Intuitive Overview of the Pitch Extraction Algorithm

Construct and decompose the spectrum for each interval

A formal description of the complete pitch extraction procedure is provided at the end of this chapter. In order to provide a more intuitive understanding of it, however, I include the following informal description. First, the ordered sets S_i are computed for each interval $i=1,2,\dots,M$, and split up into the harmonically independent $Q_{i,j}$ components according to the algorithm above. At this point, it is almost certain that some of the $Q_{i,j}$ sets contain "hidden" notes. Yet we assume that it is likely that for at least *some* intervals I_i , none of the fundamental frequencies were obscured, and hence the $Q_{i,j}$'s are "correct" for those intervals. Let us refer to the number of resulting $Q_{i,j}$ sets in I_i as J'_i . We now make the assumption that J , the true number of parts present in the song as a whole, is equal to the maximum of the J'_i 's (we call this maximum J'_{MAX} .) It is clear that this presupposition breaks down in the event that in

every interval one or more notes were obscured. However, as long as $\exists i \mid J'_i = J$, the assumption will be reliable and we will have deduced the correct number of parts.

Identify well-behaved and poorly-behaved intervals

We now conceptually separate the intervals into two groups: those that we know are well-behaved, and have $J'_i = J'_{MAX}$, and those which may contain hidden notes, where $J'_i < J'_{MAX}$. Call the first group the "well-behaved intervals" W_i , $i=1,2,\dots,\omega$, and the second group the "poorly-behaved intervals" P_i , $i=1,2,\dots,\pi$. (Note that the P_i 's fall into two different classes: intervals in which there is harmonic ambiguity, and intervals in which one of more of the parts has a "rest." In either case, $J'_i < J'_{MAX}$.) Because we will treat the Q sets in these two kinds of intervals completely differently, it is appropriate to introduce a different notation for each of them: let $\Omega_{i,j}$ represent the $Q_{i,j}$ sets for well-behaved intervals, and let $\Pi_{i,j}$ represent the $Q_{i,j}$ sets for the poorly-behaved intervals. Now we have seen that for each of the ω intervals W_i , we can extract the pitches without ambiguity: we would simply take the lowest frequency in each of the $\Omega_{i,j}$ sets. Our plan, however, is to use the information about the harmonic spectra derived from the W_i 's to help us identify hidden notes in the P_i 's.

Compute the mean harmonic signatures

In this vein, we begin by computing the *mean harmonic signatures* Σ_j , which will be our best approximation to the signatures for each of the J parts in the song. In order to even form these averages, however, it is necessary to identify which voices in each well-behaved interval correspond to which parts. (It would obviously be counterproductive to form the mean harmonic signature for a tuba by averaging together seven signatures from tuba notes and six from piccolo notes!) Therefore, we must determine what to average with what by comparing each signature in an interval

to a "master list" of signatures. We decide which instrument's signature in the master list a given voice looks "closest" to (in practice, by computing the norm of a subtraction between the two vectors), and average it into the Σ_j for that instrument. One question that arises is, how do we get the master list in the first place, since we are assuming nothing *a priori* about the values of the signatures? The answer is that we must choose one of the intervals W_i with which to bias the signature computation. We take the $\Omega_{i,j}$'s from one interval W_i , and use them for comparison with all other intervals W_i , $i \neq j$. Intuitively, if we choose poorly (ie., if the $\Omega_{i,j}$'s for our choice of W_i are "atypical"), then we will not have a good initial model with which to compare the signatures from the other intervals. The consequence is that we will end up averaging in a few piccolo notes with our tuba notes, for instance. The brute force solution to this problem is to try *every* well-behaved interval W_i as a possible bias interval, and to find the one that yields the least total error - where we define the total error to be the sum of all the norms of differences as we compare $\Omega_{i,j}$'s with the $\Omega_{i,j}$'s. (This is essentially step 5A of the formal description at the end of the chapter.) We form the Σ_j 's by choosing the interval which gave the least error as the bias interval, and performing the averaging described above.

Note that in a well-behaved interval, we have no harmonic ambiguity, but we might still have *upper* harmonic ambiguity. In other words, even though no voice's *fundamental* frequency has been "hidden" by another voice's harmonic signature, one or more *upper harmonic* frequencies may have been hidden. In this case, the $\Omega_{i,j}$ sets will each have a fundamental frequency which corresponds to one and only one voice, but the rest of the harmonics will not necessarily accurately represent the voice's signature. (Such was the case with our two voices at 100 and 150 Hz, above.) In effect, when we compute the Σ_j 's, we will not actually be averaging in perfectly accurate signatures for these kinds of intervals. The result of this will be decreased

resolution in the Σ_j computations, and it is unknown how much of a problem this might pose in general. For the inputs used as part of this thesis, we still achieved fairly adequate results, as will be seen in chapter 5. In general, however, we must admit at this time that we are tacitly making the assumption that the harmonics which are "stolen" from various signatures in these intervals will not have too severe an effect on the computations of the Σ_j 's. Our hope is that they should roughly "average out" and still leave us with reasonably accurate signature approximations.

Derive the pitches in the well-behaved intervals

Once the Σ_j 's have been computed, then, we know which $\Omega_{i,j}$ sets in each interval correspond to which parts of the song, and we can derive the notes for the W_i 's in a straightforward way. (Since by definition there is no harmonic ambiguity in a W_i interval, we just take the lowest frequency in each $\Omega_{i,j}$ as the fundamental frequency for a voice.) Now, we can use this knowledge about what the signatures of the J parts look like to analyze the poorly-behaved intervals P_i .

Derive the pitches in the poorly-behaved intervals

Consider the J_i harmonically independent sets $\Pi_{i,j}$ for a given poorly-behaved interval P_i . By definition, $J_i < J_{MAX}$. This could either be because one or more of the J parts has a rest during the interval, or because one or more of the voices present has been obscured by harmonic ambiguity, or both. In general, a given set $\Pi_{i,j}$ with fundamental frequency $\Pi_{i,j}|f_1$ may be composed of one or more voices. At least one of the voices must be playing a note at pitch $\Pi_{i,j}|f_1$, while each of the others has a fundamental frequency equal to an integer multiple of $\Pi_{i,j}|f_1$. We also have the constraints that each of the parts in the song may be present in no more than one Π set, and that every Π set must contain the frequencies for at least one voice.

Our approach to determining which voices are present in the Π sets and with which fundamental frequencies is to use a sort of brute force approximation algorithm. In every interval, *some* combination of voices at particular fundamental frequencies was responsible for making up the peaks in the spectrum. And for every group of Π sets in a particular poorly-behaved interval, there is a finite (albeit large) set of these combinations that might possibly represent what "really" was being played in the song. As an example, suppose that we have a flute, trumpet, and oboe as the parts in a three-part song, and that in a certain poorly-behaved interval of this song we have two Π sets: one with lowest frequency 400 Hz, and another with lowest frequency 500 Hz. There are many possible ways these two Π sets could have been generated: perhaps the flute was playing at 400 Hz, the trumpet at 500 Hz, and the oboe was resting. Or the oboe could have been playing at 500 Hz, while the trumpet was the one resting. Or we could have had the flute at 400 Hz, the trumpet at 500 Hz, and the oboe at 800 Hz, which was therefore harmonically obscured by the 400 Hz flute fundamental. Or the flute and oboe might have been in unison, *both* with fundamental 400 Hz. Or the oboe could have been at 1500 Hz, obscured by the trumpet's fundamental, etc., etc. The number of possible combinations will quickly become large as we add more and more voices, but it should be clear that as long as we place a limit on how many multiples one voice's frequency can be above another, there are only a finite number of possibilities.

Our approach is therefore to "try" every such combination (under the assumption that each voice's harmonic signature will be reasonably close to its mean harmonic signature Σ_j) and compare these results to the actual representations from the Π sets. We will choose whichever combination gives us the least total error, and report that the voices and pitches corresponding to that combination were what was

actually present in the interval. This method is described formally in step 7 of the pitch extraction algorithm at the end of the chapter.

An issue concerning the formation of guesses

One remark is in order concerning how to form these "guesses" for each combination that we will compare to the actual Π sets. It may appear at first glance that we could directly add harmonic signatures (appropriately shifted to account for differences in pitch) to obtain a guess at the Π sets for a particular combination, but this is not so. To illustrate: if we have two sinusoids, each at the same frequency and with amplitude 1, we would not necessarily expect that the addition of the two would yield a sinusoid with amplitude 2. That could only occur if the two sinusoids happened to be at exactly the same phase, which will not be generally true. We know only that the two signals will add to form another sinusoid with an amplitude somewhere between 0 (perfectly destructive interference) and 2 (perfectly constructive interference.) How, then, do we form our guesses? Suppose we want to guess that the oboe was at 400 Hz and the flute at 800 Hz - how do we determine what the signature for this combination "would have" looked like so we can compare it to what was actually found in the spectrum?

One strategy would be to keep track of not only amplitude information, but phase information in the computation of the Σ_j 's. Then, the phase differences between the frequency components of a *single* signature would be fixed, and we could simply add as another degree of freedom the phase difference between each pair of voices in a given Π set. For simplicity, however, the *Score* program does not attempt this level of sophistication. Rather, it forms the "guess" for a particular combination of voices and frequencies by using a sort of "expected value addition." That is, when it adds the

components of two signatures to arrive at an amplitude, it takes the expected value with respect to the possible phase differences between the components.

To determine what this expected value is, we apply a bit of probability theory. Suppose we have two sinusoids (represented by phasors) of equal amplitude but possibly unequal phase: $a e^{j0}$ and $a e^{j\theta}$. If we add them, the magnitude of the result is:

$$\begin{aligned} & \sqrt{(a + (a \cdot \cos\theta))^2 + (a \cdot \sin\theta)^2} \\ &= a \cdot \sqrt{2 + 2 \cdot \cos\theta} \end{aligned}$$

We see that by allowing θ to range from 0 to π , the amplitude indeed ranges from 0 to $2a$, as expected. Assuming that θ , the phase difference between the two signals, is uniformly distributed on $(0, 2\pi)$, we can find the expected value of this expression:

$$\begin{aligned} E[\text{amplitude}] &= \int_0^{2 \cdot \pi} \frac{1}{2 \cdot \pi} \cdot (a \cdot \sqrt{2 + 2 \cdot \cos\theta}) \, d\theta \\ &= \frac{4}{\pi} \cdot a \end{aligned}$$

In other words, if we add two sinusoids of equal amplitude with a uniformly distributed random phase shift, we will on average get a signal with an amplitude of $4/\pi$ (or about 1.27) times the amplitude of the original signals.

Now in general our sinusoids will not have the same amplitude. Let us extend this argument to the two phasors $a e^{j0}$ and $b e^{j\theta}$. In this case, we have as the amplitude:

$$\sqrt{a^2 + 2ab \cdot \cos\theta + b^2}$$

and the expected value is given by

$$E[\text{amplitude}] = \int_0^{2 \cdot \pi} \frac{1}{2 \cdot \pi} \cdot \sqrt{a^2 + 2ab \cdot \cos\theta + b^2} \, d\theta$$

Unfortunately, this integral has no known closed-form solution. We can normalize it, however, by dividing through by a and scaling our answer accordingly; this gives us the plot in Figure 12, which was obtained through a number of numerical integrations. *Score* evaluates the integral to approximate the expected value of the joint amplitude, and from the result computes the "guesses" for each combination of voices and frequencies.

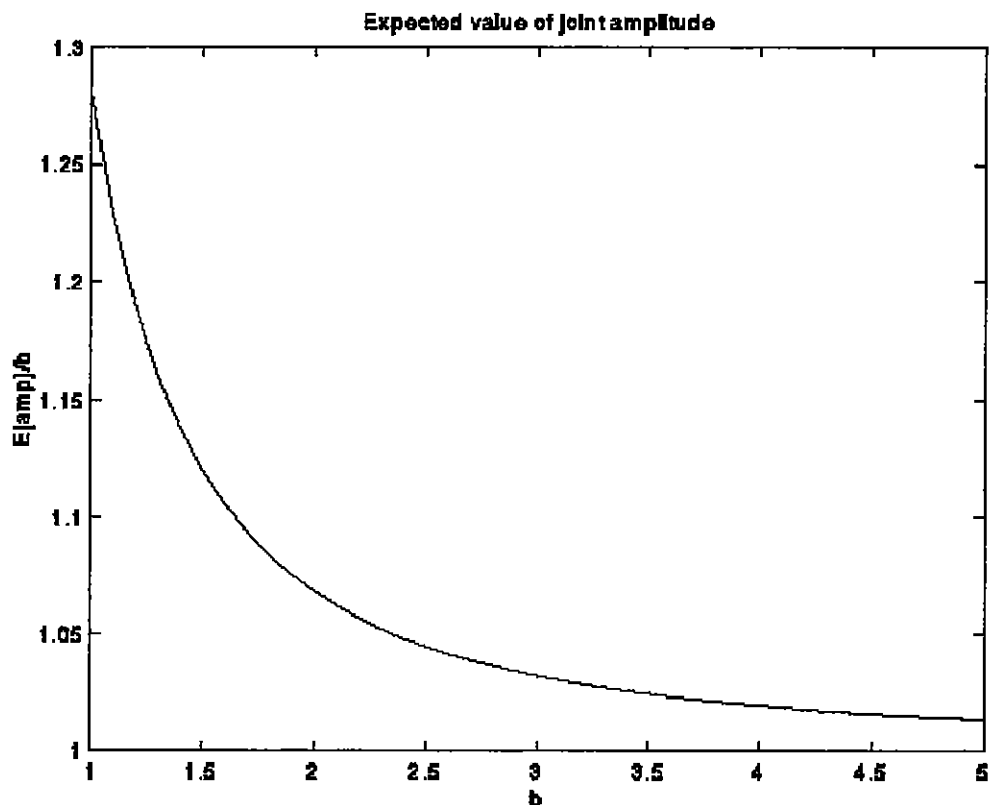


Figure 12. The expected value of the joint amplitude of two sinusoids. Suppose we have two sinusoids at the same frequency but with different magnitudes and phases. Without loss of generality, we may say that one of them has an amplitude of 1, as long as we scale our result accordingly. In the figure, the scaled magnitude of the second sinusoid (called b) is plotted versus the ratio of the expected value of the joint amplitude to the amplitude of this second sinusoid. Note that as b increases, the ratio approaches 1; this agrees with our intuition, since if one of the signals has a magnitude that far exceeds that of the other, we would expect the joint amplitude to be approximately that of the greater sinusoid.

***Score's* Implementation of the Pitch Extraction Algorithm**

Because of time constraints and practical limitations, I was not able to implement the pitch extraction algorithm in its most general form. For one thing, the version of MATLAB I used did not allow multiple indexing of matrices, which meant that it was literally impossible to have, say, M different Q matrices, each one containing the H harmonic frequencies for each of J different voices, where M , H , and J were all parameters that could grow without bound. The most feasible option was to "hardcode" the solution for a particular number of voices, which is what I elected to do. The version of *Score* included in Appendix A is hardcoded for no more than three voices, and is unable to even recognize any others. It should be understood, though, that this is due to no inherent weakness of the algorithm, but only of the implementation tool selected. The program could be redesigned in a fairly straightforward manner to accommodate a greater number of voices.

Also, it should be noted that with only three voices, the number of voice/frequency combinations that must be checked for each Π set is not great enough to cause a noticeable performance slowdown. In fact, the event detection procedure which precedes any pitch extraction still dominates the total computation time by far. If the program were extended, though, the time required by step 7 of the pitch extraction algorithm would explode as $J!$, where J is the number of voices, and the straightforward approach I used for checking each combination individually would produce unacceptable delays. It is likely that a more intelligent method of analyzing the possible combinations could be developed, such that those that are clearly improbable could be eliminated at the outset, before the computation-intensive procedures of shifting and adding signatures are performed. The well-known Viterbi algorithm, described in [7], could probably be adapted for this purpose. It provides a

more efficient means of arriving at the "most likely" solution to a set of observation data.

In summary, then, the pitch extraction procedure used by the *Score* program involves taking FFTs in each of the intervals found by the event detection procedure. The resulting frequency domain peaks from each interval are then analyzed with respect to the entire song. Those intervals which appear to contain pitches that do not harmonically conceal each other are treated in a straightforward manner. Additionally, information from them is compiled to form an approximation to the harmonic signature for each part present; these approximations can be used in analyzing the other "poorly-behaved" intervals. Assuming that the signature for each part is relatively consistent throughout the song, and that there are enough well-behaved intervals from which to form an approximation, we should expect good results for pitch extraction.

A Formal Description of the Pitch Extraction Algorithm

Table 2. Summary of notation for the pitch extraction algorithm.

- M - The total number of intervals identified by the event detection algorithm.
- $I_i, i=1, \dots, M$ - The M intervals in the song.
- Y_i - The total number of peak frequencies found in the FFT for interval I_i .
- $S_i = \{ \{f_1, a_1\}, \{f_2, a_2\}, \dots, \{f_{Y_i}, a_{Y_i}\} \}$ - A set representing the full spectrum of peaks for interval I_i . Each $\{f_y, a_y\}$ pair represents a frequency and an amplitude. (The a_y 's are found by summing the amplitude corresponding to f_y and any leakage bins immediately surrounding f_y .)
- $Z_{i,j}$ - The number of (harmonically related) peak frequencies in set $Q_{i,j}$.
- $Q_{i,j} = \{ \{f_1, a_1\}, \{f_2, a_2\}, \dots, \{f_{Z_{i,j}}, a_{Z_{i,j}}\} \}$ - A subset of S_i where all the f_z 's are harmonically related.
- J - The actual number of individual voices present in the recording.
- J'_i - The number of $Q_{i,j}$ sets extracted from spectrum S_i .
- J'_{MAX} - The estimated number of individual voices present in the recording; this is calculated as the maximum of the J'_i 's over all intervals I_i .

- ω - The number of well-behaved intervals in I_i ; viz., the number of intervals where $J'_i = J'_{MAX}$.
- W_i - Those ω intervals which are well-behaved.
- $\Omega_{i,j}$ for $i=1,\dots,\omega, j=1,\dots,J'_{MAX}$ - These sets are merely the $Q_{i,j}$'s for all the well-behaved intervals. (A different symbol is appropriate here because the treatment of the well-behaved intervals is fundamentally different from the poorly-behaved intervals.)
- π - The number of poorly-behaved intervals in I_i .
- P_i - Those π intervals which are poorly-behaved.
- $\Pi_{i,j}$ for $i=1,\dots,\pi, j=1,\dots,J'_i$ - These sets are the $Q_{i,j}$'s for all the poorly-behaved intervals.
- H - The maximum number of harmonics the system will keep track of for each voice. (An adjustable system parameter.)
- Σ_j for $j=1,\dots,J'_{MAX} = \{ a_1, a_2, \dots, a_H \}$ - The mean harmonic signature for voice j . It represents the relative magnitudes of each of the harmonics in voice j , independent of any particular fundamental frequency.
- $\Sigma_j(\phi)$ - This notation describes the harmonic signature for voice j "projected" onto fundamental frequency ϕ . It is equivalent to a set with the same structure as the $Q_{i,j}$'s and with value $\{ \{ \phi, a_1 \}, \{ 2 \cdot \phi, a_2 \}, \dots, \{ H \cdot \phi, a_H \} \}$.
- $\Sigma_j|a_n$ - This notation describes the magnitude of the n th harmonic component of the mean harmonic signature for voice j .
- $\Pi_{i,j}|a_n$ - This notation describes the magnitude of the n th harmonic component of the set $\Pi_{i,j}$.
- $\Pi_{i,j}|f_n$ - This notation describes the frequency of the n th harmonic component of the set $\Pi_{i,j}$.
- W_1 - The interval which is chosen to bias the mean harmonic signatures.
- β_j for $j=1,\dots,J'_{MAX}$ - The signatures of the voices taken unaltered from interval W_1 . These are the "biased" harmonic signatures which we will use to compare to the $\Omega_{i,j}$'s for $i \neq 1$ in order to compute the mean harmonic signatures.
- Δ_j for $j=1,\dots,J'_{MAX}$ - A "dummy" harmonic signature "variable" used to describe step 5A of the algorithm below; used instead of Σ_j to avoid confusion.
- $\psi_{i,j}$, for $j=1,\dots,J'_{MAX}$ - A variable to indicate where voice j appears in poorly-behaved interval P_i . for example, if $\psi_{9,2}=3$, then voice 2 is present in $\Pi_{9,3}$; if $\psi_{5,1}=0$, then voice 1 is resting during interval P_5 .
- n_j - A variable indicating how far voice j is "shifted up" in its $\Pi_{i,j}$ set. For example: if $n_j=1$, then the fundamental frequency of voice j is simply the

lowest frequency present in $\Pi_{i,j}$. If $n_j=2$, then voice j is an octave above some other note *also* present in $\Pi_{i,j}$.

n_{\max} - The maximum number of multiples of a low note's fundamental a high note can occupy. (An adjustable system parameter.)

The Algorithm:

1. Take the incremental FFT within each interval. Perform heuristics to determine the "true" peaks, and form the sets S_i , $i=1, \dots, M$.
2. Decompose each S_i into J'_i subsets $Q_{i,j}$.
3. Compute J'_{\max} as the maximum of the J'_i 's over all intervals I_i . We assume that $J'_{\max}=J$, the true number of parts present in the song.
4. Separate all intervals I_i into W_i 's (if $J'_i = J'_{\max}$) and P_i 's (if $J'_i < J'_{\max}$).
5. We now compute the mean harmonic signatures Σ_j for each voice j . Note that we are only dealing with the intervals W_i throughout step 5.

A. Find the starting interval with which to bias the Σ_j computation. (Note that we are not actually computing the real Σ_j 's in step 5A, but only determining which interval W_i is best to bias the real computation with.)

For each $i=1, \dots, \omega$,

i. Set $\Delta_j = \Omega_{i,j}$ for $j=1, \dots, J'_{\max}$.

ii. Set $\text{error}_i = 0$.

iii. For each $k=1, \dots, \omega$, $k \neq i$,

a. Find set of (j, ζ) pairs over all combinations of (j, ζ) , with $j, \zeta=1, 2, \dots, J'_{\max}$, such that $\sum \|\Delta_j - \Omega_{k, \zeta}\|$ is minimized.

b. Add this minimum value of the summation to error_i .

Find the interval i for which error_i is the smallest. We will bias the mean harmonic signatures with this interval W_i , by setting $\beta_j = \Omega_{i,j}$ for $j=1, 2, \dots, J'_{\max}$.

B. Actually compute the mean harmonic signatures with this bias.

Initially set each $\Sigma_j = \beta_j$ for $j=1, 2, \dots, J'_{\max}$. Then, for each $k=1, \dots, \omega$, $k \neq i$,

i. Find set of (j, ζ) pairs over all combinations of (j, ζ) , with $j, \zeta=1, 2, \dots, J'_{\max}$, such that $\sum \|\beta_j - \Omega_{k, \zeta}\|$ is minimized.

ii. Average $\Omega_{k, \zeta}$ into Σ_j for each (j, ζ) pair.

6. We now extract the pitches from each well-behaved interval W_i in a straightforward fashion:

For each $i=1, \dots, \omega$,

i. Find set of (j, ζ) pairs over all combinations of (j, ζ) , with $j, \zeta=1, 2, \dots, J'_{MAX}$,

such that $\sum \| \beta_j - \Omega_{i, \zeta} \|$ is minimized.

ii. For each (j, ζ) pair, assign voice j a note with fundamental frequency $\Omega_{i, \zeta} | f_1$ for interval W_i .

(In actuality, steps 5B and 6 can be combined, because both involve minimizing the same expression for all well-behaved intervals. They are listed separately here because conceptually, they are two different steps.)

7. We now attempt to extract the pitches from each poorly-behaved interval P_i . We analyze each set $\Pi_{i, j}$, $i=1, \dots, \pi$, to see if it actually contains the "sum" of two or more voices. If we determine it does, then we will attempt to divide the $\Pi_{i, j}$ into new Π sets containing only one voice each, and include them in our set of $\Pi_{i, j}$'s for this interval. Note that we are only dealing with the intervals P_i throughout step 7.

For each $i=1, \dots, \pi$,

A. Each voice j may appear in one of the $\Pi_{i, j}$ sets, or in none of them. Let $\psi_{i, j}$, for $j=1, \dots, J'_{MAX}$, be a variable to indicate in which Π set voice j appears in interval P_i , as explained in the definitions, above. We constrain the values of the $\psi_{i, j}$'s in the following manner: $\forall \zeta, \zeta=1, \dots, J'_i, \exists j, j=1, \dots, J'_{MAX} | \psi_{i, j} = \zeta$.

Find the set of $\psi_{i, j}$, and n_j , with each $\psi_{i, j}$ ranging over $0, 1, 2, \dots, J'_i$ and with each n_j ranging over $0, 1, 2, \dots, n_{max}$, subject to the above constraint, such that the sum over all Π sets of

$$\| (\Pi_{i, j} - \sum (\sum_q (n_q \cdot \Pi_{i, j} | f_1))) \|, \text{ for all } q \text{ where } \psi_{i, j} = j$$

is minimized. It is understood that the above summation notation represents not an ordinary arithmetic sum, but the expected value of the amplitude of the sums of frequency components, where the phase difference between any two frequency components is assumed to be uniformly distributed on $(0, 2\pi)$.

B. When the $\psi_{i, j}$ and n_j which minimize the above expression have been found, the pitches can be extracted from P_i as follows. For each nonzero $\psi_{i, j}$, assign voice j a note with fundamental frequency

$$n_j \cdot \Pi_{i, \psi_{i, j}} | f_1$$

for interval P_i . For each $\psi_{i, j}=0$, assign voice j a rest for interval P_i .

8. All voices have now been assigned either a rest or a pitch for all intervals. The score may then be printed to the screen in the usual way.

CHAPTER IV

RECOGNITION OF SINUSOIDAL INSTRUMENTS WITH SQUARE ENVELOPES

Before testing the *Score* program on music obtained from "real" instruments, I wanted to establish the validity of the algorithm by using inputs produced by a simplified model. This way, it would be possible to postpone dealing with some of the idiosyncrasies involved in real-world sound and instead concentrate on the meat of the problem: designing an implementation of the general procedures described in the last chapter to perform automated event detection and pitch extraction.

Choosing a Simplified Model

Recall the discussion in Chapter 2 regarding the general characteristics of sound produced by musical instruments. The question before us is, what should the features of our simplified model be such that the program can recognize the music most easily? We want this pseudo-instrument to be at least a gross approximation of real instruments, so that what it produces sounds like "music" to the human ear. This way, our algorithm can use the same techniques on the model that it will later use on more realistic sounds. Also, it will be convenient to describe our model using the same framework by which we will later describe real instruments, so that we can pinpoint the trouble spots when we turn to them. At the same time, however, we want the model to be as simplistic and manageable as possible, so that the testing of our algorithm doesn't get bogged down in the details of one particular input.

Let us first consider the volume envelope; or more specifically, the four ADSR parameters by which we have conveniently parametrized it. What should the values of

A, D, S, and R be so that musical recognition is simplified? As we mentioned earlier, it is a basic property of the adaptive predictor that its error is large when it encounters significant "surprises" in the input. A sudden burst of magnitude within a sea of background noise will cause the predictor's error to jump much more dramatically than will a slow crescendo up to the maximum intensity. And sudden rises in predictor error are precisely what our event detection algorithm is keying on in order to find note beginnings. Clearly, then, our program will best perform if its input has $A=0$, to correspond to an instantaneous attack.

Since once the beginning of the note has been identified the only thing left to do is to analyze the frequency domain, we cannot help our algorithm by allowing the instrument to decay from its maximum intensity. A significant decay time would lead to only two things: first, the linear predictor error would increase somewhat, having been "surprised" by the decay as it was by the attack. This is undesirable, because we are considering the beginning of the attack as the true starting point of the note; and once that has been detected, we want the predictor error to be suppressed as much as possible until the next "true" event. The second effect of a decay would be simply to lessen the energy in the signal during the interval in which frequency domain information is to be collected. This leads to a reduced signal-to-noise ratio, and smaller frequency domain peaks that must be identified. In the ideal case, we would rather have the spectral peaks stand out as much as possible from the surrounding noise, so that they are more easily analyzed. For both of these reasons, then, we would like this idealistic model to maintain its peak intensity all the way through the notes it plays. Therefore, we set $D=0$ (no decay time) and $S=100\%$ (sustained level of intensity is equal to the peak.)

We are faced with a dilemma in choosing the R parameter. On the one hand, if R is too small, and the intensity of the note drops off too quickly, the linear predictor

will be surprised by the sudden decrease in magnitude and will likely signal an event at the *end* of the note in addition to the beginning. Therefore, it might be appropriate to have the note's intensity dwindle to nothingness as slowly as possible, so that no false alarms are registered. However, there is a limit to how slow this fade can be, because if some intensity remains when the next note begins, we will see some residual frequency components from the *previous* note in the melody when we do the spectral analysis on the current note. Hence, we have two conflicting goals when we consider the value of R.

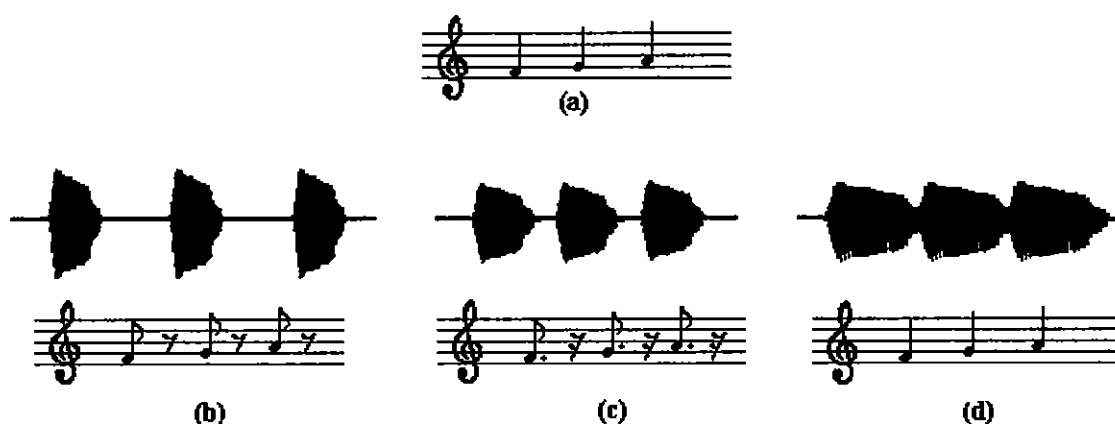


Figure 13. (a) A simple melody written by a composer. The figures below show the waveforms for this piece if the performer played it in a (b) *staccato*, (c) normal, and (d) *legato* style. The staves below the waveforms show what the *Score* program might produce as it output for this song; they are different for the three styles, but in each case a perfectly accurate representation of the piece.

For simplicity, when I chose the values for the simplified model in my experiments, I ruled entirely in favor of the second consideration and against the first. The reasons for this are shown in Figure 13. When presented with a piece of music as in Figure 13a, a performer is in some sense still free to choose the lengths of each of

the notes. The beat of the song dictates that the *beginning* of each note in the sequence fall in a certain place, of course, but there are still differences in how long the intensity of a particular note will be sustained within the given interval. Figure 13b shows what the waveform would look like if the musician played the piece in a staccato style, and what the resulting score built from the output of the *Score* program would be. The program's output is not identical to the original score: this is because the program has detected "events" at the ends of the *staccato* notes and inserted rests appropriately. The reason I chose $R=0$ for my simplified model, however, is because I consider the music in Figure 13b to be a perfectly acceptable score of the piece. A composer who wrote a *staccato* passage might place dots above the notes to indicate that it should be played this way, but an alternative and equally valid way of expressing the same thing would be to write each note as an eighth note followed by an eighth rest instead of a lone quarter note. In some respects, then, the scores at the bottom of Figure 13 are really *more* accurate than the original score. And in any case, it is a simple matter to run the output through another procedure which will lengthen notes and eliminate rests as the user desires.¹² For these reasons, I do not consider an output which detected "events" at the ends of notes, and inserted rests accordingly, to be an invalid one. Hence, for simplicity, the model I used had a zero release time.

When we turn to the harmonic content of our model and ask what characteristics would make it simplest to recognize correctly, we get a straightforward answer. The only ambiguity in the frequency domain analysis of an interval comes when some notes have a fundamental which coincides with the harmonic of another note. Clearly, then, these problems are alleviated if we choose a model with *no harmonic frequencies*. The simplified model I used in my initial testing of *Score* was a hypothetical instrument with only a fundamental frequency, and which appears in the frequency domain as a single peak at the pitch of the note. The time domain

waveform corresponding to a single frequency domain peak is a pure sinusoid, and the resulting sound is much like that of a tuning fork - just an unwavering pure tone. Figure 14, below, summarizes these characteristics of the simplified model: a sinusoidal waveform with a rectangular volume envelope.

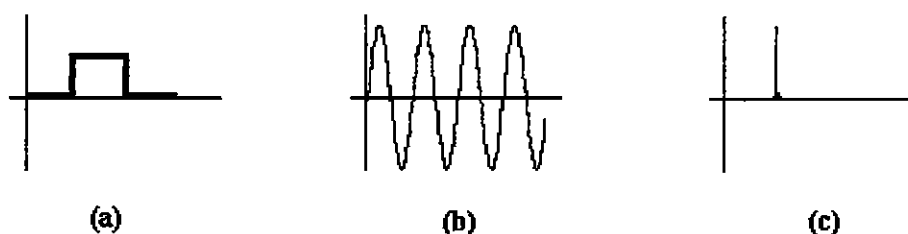


Figure 14. Characteristics of the simplified model used in initial testing of *Score*. (a) The volume envelope (rectangular.) (b) Waveform (pure sinusoid.) (c) Frequency domain representation (single fundamental peak, no harmonics.)

Results

After numerous laboratory trials, the success of the *Score* program in correctly recognizing music played by this simplified instrument is impeccable, both with single-part and multi-part songs.¹ Included on pages 68-72 are some example musical phrases that were performed by the square-envelope sinusoid, recorded, and processed by the program. As you can see, the resulting output is in nearly all cases identical to the true score that was input.

Melody #1 from "My Heart Cried" - Sine voice, rect. envelope.



```

*****
*****
**                               **
**           S C O R E !           **
**           Stephen Davies - version 1.0           **
*****
**                               **
**   Current system parameters:           **
**                               **
**   Filename of sound file to use: MUSIC.DAT           **
**   Sampling frequency of sound file: 8000 Hz           **
**   Tempo of song: 110 beats per minute           **
**   Minimum length of a note: 0.125 whole notes           **
**   Incremental FFT size: 1024 samples           **
**   Trim threshold: 0.5 standard deviations           **
**   Gain of linear predictor: 10% of maximum           **
**   Window size of linear predictor: 10           **
**   Event detector STA window size: 10           **
**   Event detector LTA window size: 1000           **
**   Event detector power ratio threshold: 10           **
**   Incremental FFT size: 2048 samples           **
**   FFT size tolerance: 80% of shortest note           **
**   Pitch extraction cut-off factor: 1000           **
**   Pitch extraction dynamic range: 0.4           **
**   Reporting accidentals in: flats           **
**   Instrument type: sine           **
**   Score mode: brief           **
**                               **
*****
*****

```

Preparing song for analysis...

Loading sound file...
 Removing DC bias...
 Cancelling noise...
 Trimming song...

Song now has 59766 samples.

Analyzing song to locate notes...

Compiling linear predictor error...
 Analyzing linear predictor error for events...
 Performing additional note detection heuristics...

Determining score...

```

-----
 1      8th-note:  Eb4 ( 320.3 Hz.)
-----
 2      8th-note:  F4  ( 359.4 Hz.)
-----
 3 dotted-quarter: G4  ( 398.4 Hz.)
-----
 4      8th-note:  G4  ( 398.4 Hz.)
-----
 5      8th-note:  G4  ( 398.4 Hz.)
-----
 6      8th-note:  F4  ( 359.4 Hz.)
-----
 7      8th-note:  G4  ( 398.4 Hz.)
-----
 8 half-tied-8th:  Ab4 ( 421.9 Hz.)
-----
 9      8th-note:  Ab4 ( 421.9 Hz.)
-----
10      8th-note:  Ab4 ( 421.9 Hz.)
-----
11      8th-note:  G4  ( 398.4 Hz.)
-----
12      8th-note:  Ab4 ( 421.9 Hz.)
-----
13      8th-note:  Bb4 ( 476.6 Hz.)
-----
14 quarter-note:  Bb4 ( 476.6 Hz.)
-----
15 quarter-note:  Bb4 ( 476.6 Hz.)
-----
16 quarter-note:  C5  ( 531.2 Hz.)
-----
17 dotted-quarter: Ab4 ( 421.9 Hz.)
-----

```

Results: 17 notes correct
 0 notes incorrect

(Note: The output above was what was actually printed by the *Score* program. In order to allow the reader to more easily compare inputs and outputs, however, in all subsequent examples I will instead include musical notation representing the output obtained. Observe that the *Score* program does not produce such a graphical representation, but that it is a straightforward task to transform an output such as the one above into a musical score.)

Melody #2 from "My Heart Cried" - Sine voice, rect. envelope.

A musical score consisting of three staves. The top staff is a treble clef with a key signature of two flats (Bb, Eb) and a 4/4 time signature. It contains a melody of 24 notes. The middle and bottom staves are also treble clef and contain accompaniment notes.

Output from Score version 1.0:

A musical score consisting of three staves, identical in notation to the first score. It shows the output from Score version 1.0, which includes some additional markings like a 'z' (accidental) and a 'b' (flat) on the top staff.

Results: 24 notes correct
1 note incorrect due to false event
1 note incorrect due to wrong pitch

"How Great Thou Art" 3-part - Sine voice, rect. envelope.



Output from Score version 1.0:



Results: 33 notes correct
0 notes incorrect

"A Mighty Fortress is Our God" 3-part - Sine voice, rect. envelope.



Output from Score version 1.0:



Results: 28 notes correct
1 note incorrect due to missed event

Considerations

During the process of testing inputs such as these, I uncovered a few issues that are worth noting at this stage before we proceed to "real" instruments. First, some of the program's parameters are fairly sensitive, and need to be fine-tuned for the algorithm to work successfully. These parameters include the number of weights and the gain of the linear predictor, the threshold and the sizes of the windows for the STA/LTA algorithm, the threshold for frequency domain peak detection, and others. The linear predictor gain μ , particularly, has an enormous impact on the ability to detect events faithfully, especially when it is set too high. In this case, the predictor adjusts so quickly to any change in input that its squared error ϵ_k^2 never rises to a significant level where it can be monitored. When too low, it does not adjust very quickly and the high error level remains even for hundreds of samples after the "true" event. However, the additional filter which enforces a minimum amount of time between two successive events is very capable of eliminating false alarms in the period immediately after a true note beginning, so that in practice it is much safer to err on the low side when choosing the predictor's gain.

The success of event detection is sensitive to the sizes of the windows in the STA/LTA algorithm, too, particularly the length of the long-term window. In general, best results are achieved when the window is quite long, so that much of the "low error" time in between events is factored into the long-term average. However, the window should not be so long that it contains the peak in error from the *previous* event when it reaches the next event. (See Figure 15). The best approach seemed to be to base the length of the long-term window on the tempo information the user specifies for the song, making it equal to a significant fraction of N_{\min} .

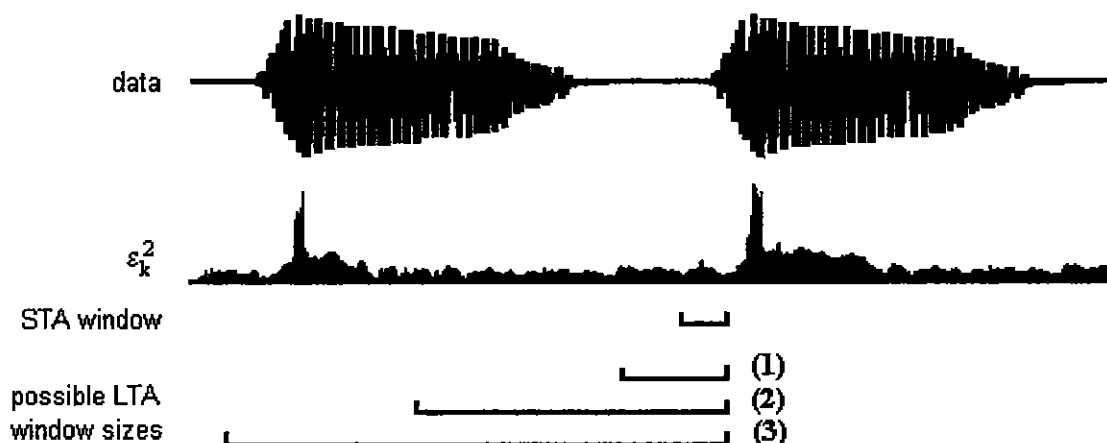


Figure 15. Effect of the LTA window size in event detection. The top graph shows the data signal from two successive notes; we assume that these are two of the shortest notes in the song, such that the distance between them is $\sim N_{\min}$. Below these notes is the value of the linear predictor's squared error at each data point. When choosing the length of the LTA window for the power ratio algorithm, it is preferable to make it substantially longer than the STA window so that sudden rises in squared error are easily detected. (As the LTA window shrinks, the LTA power becomes closer and closer to the STA power, such that nothing can be distinguished.) Hence, (1) is a poor choice for the LTA window. On the other hand, the window should not be so long that the peak in error from the previous note is included in its average, as (3); such an effect blurs the ability to recognize sharp peaks at the current note. The best compromise is to make the window large but not too large - a significant fraction of N_{\min} , as is (2).

Another interesting phenomenon I discovered while testing these simplified inputs was the effect of noise cancellation. In the beginning, I assumed that applying a simple low-pass filter to the input in order to remove high frequencies that weren't of interest would be a desirable step in preparing the data for analysis. I soon discovered, however, that my order-100 FIR filter was hampering the capability of the program to perform accurate event detection. The reason is really rather simple: the linear predictor's error, upon which event predictions are ultimately based, is greatest when

the input experiences sudden, unexpected changes in amplitude. And these abrupt changes are represented in the frequency domain as very high frequency components. The use of a low-pass filter, consequently, will naturally suppress such changes and smooth out the transitions from note to note. Applying such a filter is therefore particularly undesirable for the *Score* program, and hence, all input data is analyzed unfiltered.

Finally, it is worth commenting on a peculiarity of the outputs for multi-part songs, as illustrated in the result on page 72. Observe that there are times during the song when two of the parts hold a longer note than the third part; an example would be the third beat of the second measure, where the melody plays two eighth notes while both the harmony parts hold out a quarter note. Upon examination of the *Score* program's output (intervals 4-5), we find that it has instead recorded that *all three* of the parts played two eighth notes, the harmonies simply repeating their notes at the same pitch. The reason for this idiosyncrasy is that the event detection algorithm only finds places in the song where *some* new note has begun; it is impossible to tell *how many* or *which* of the parts actually had a new attack at that time. One solution would be to run the output through another procedure which would find repeated notes and merge them into longer ones. However, this approach doesn't really solve the problem, as it is perfectly possible for a piece of music to specify that repeated notes be played, in which case the previous output would be correct and this new output incorrect. For the time being, it must be conceded that this particular case illustrates an ambiguity which the *Score* program is unable to resolve completely. In a fully-featured product, the program would have to ask the user for assistance, giving them the choice, for instance, of merging all repeated notes in a particular range of measures, pitches, or parts.

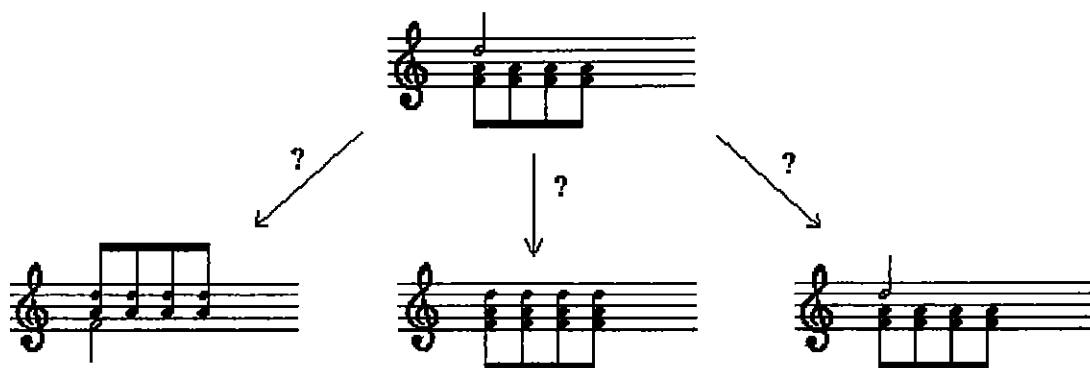


Figure 16. The ambiguity inherent in multiple part attacks. The top staff contains a piece of music which was actually performed and used as input to the *Score* program. After the event detection algorithm, *Score* knows that four eighth-notes were played by *some* part, but is unable to determine which note(s) had attacks and were responsible for the event detected. From the program's point of view, all three parts might have played eighth notes, or only one of them; or perhaps the top instrument played two eighth-notes followed by a quarter, and the bottom instruments played a quarter followed by two eighths, etc. The only solution to this problem at the present time is to run the *Score* output through another routine which will merge repeated notes at the discretion of the user.

In conclusion, the *Score* program performs very encouragingly when the instruments represented in the input are of this simplified, square-envelope sinusoid variety. Once its parameters are adjusted, the algorithm can effectively recognize notes with a high degree of success, whether its input is a single-part or multi-part song. The only weakness seems to be the peculiar phenomenon of moving parts and unmoving parts occurring simultaneously, in which case *Score* records the pitches accurately but faces an event detection ambiguity whereby it has to guess which notes were held through the event. Having established that the basic algorithm is competent

to analyze simplified music, then, the next step is to move on to the domain of real-life instruments.

CHAPTER V

RECOGNITION OF "REAL" INSTRUMENTS

There are two principal differences between songs produced from our simplified model and songs containing synthesized waveforms which approximate real-world instruments. The first is that we no longer have a perfectly sharp attack and cut-off for the notes: each waveform contributed by a particular instrument will now have a magnitude dictated by its (non-rectangular) volume envelope. Secondly, the instruments will no longer merely produce a single peak in the frequency domain. Instead, we will find energy at an array of harmonically-related frequencies for each note that is being played.

In practice, the first consideration is not too difficult to overcome. For the instruments I chose to examine (two of which were striking, and one continuous) the event detection algorithm given in Chapter 3 was able to identify the beginnings of notes as it was for the simplified model, with only a slight decline in accuracy. Interestingly, the biggest event detection problem comes not from exchanging a perfectly rectangular envelope for a more realistic one, but from adding more voices to the song. In other words, the program's performance did not degrade noticeably when I moved from a single instrument with a rectangular envelope to a single violin. However, when I added two or three *other* violins playing harmony with the first, the true events became a bit harder to find. This, too, was surmountable, however, and in most cases merely required a little more time in finding the right parameters for the predictor gain, STA/LTA window sizes, etc.

The complicated frequency spectra produced by real-world instruments, however, give rise to quite formidable challenges in pitch extraction. For the simplified model presented in Chapter 4, no real pitch extraction was necessary except to identify peaks in the FFT's for each interval. We will also discover that in certain, very restricted circumstances (namely, those without harmonic ambiguity) we will be able to find similar easy ways to identify pitches of "real" songs. The assumptions required in order to use these straightforward techniques, however, are far too limiting to be of general use in the problem of automated musical recognition. Our only choice when examining the vast majority of real musical performances will be to implement the complexities of the pitch extraction algorithm detailed at the end of Chapter 3.

Choosing a Set of "Real" Instruments

In order to restrict the scope of this project to a manageable level, I decided to select only a few types of real-world instruments to analyze. This way, I could concentrate on a limited amount of data and hopefully achieve consistent results.

All of the "real" instruments analyzed were actually synthesized waveforms from the Korg 03R/W sound module. Appendix B contains samples of the volume envelopes and frequency content produced by a number of voices on the 03R/W. I had the following criteria in selecting a set of them to examine:

1. The set should include both continuous and striking instruments, because these two general classes are representative of (non-percussive) musical instruments as a whole. Each of the two classes poses unique problems, and to ignore either of them would be to leave a large portion of the problem space untouched.
2. The frequency spectra of the instruments in the set should not have "too many" upper harmonics present. This is primarily because the n_{\max} parameter

of the pitch extraction algorithm must be set high enough to include all the relevant harmonics of the instruments in the song, and the processing time required by the algorithm increases dramatically with n_{\max} .

3. The spectra of the instruments in the set should be sufficiently "different" from one other to facilitate identification. The algorithm depends on the ability to distinguish which voice is playing which note in a given interval, and it does so based on the relative magnitudes of the harmonic peaks. We will likely achieve better performance if the harmonic signatures of the voices present are so unique that it is easier to tell them apart.

With these considerations in mind, I selected three instruments from those represented in Appendix B to be used as input for the *Score* program. The first is the *harp*, which features an unusually strong fundamental compared to its other harmonics. In general, as we noted in Chapter 2, a harpist can control the quality of the instrument's sound by plucking strings closer to the middle or to one of the ends of the string. This yields a variety of standing wave patterns, and hence, a variety of possible harmonic signatures. The Korg 03R/W, however, synthesizes a harp sound using only one of these possible signatures; namely, the simplest, with a very strong fundamental and virtually non-existent upper harmonics (see Appendix B). This makes the synthesized harp appear in the frequency domain much like the simplified model we analyzed in Chapter 3; however, the time domain characteristics are vastly different, as the harp's notes exhibit the rapid attack and exponential decay typical of striking instruments.

The second instrument I analyzed was the *piano*, though instead of looking at the large number of simultaneous notes usually produced by a pianist, I examined only a single voice. Also a striking instrument, the piano features time-domain qualities similar to those of the harp: rapid attack and exponential decay. Two important

characteristics make it significantly different, however. First, the piano has a much smaller R parameter in its ADSR characterization; this is because once the player releases the key, a damper quickly muffles the tone from that string. This is not the case with the synthesized harp, where the tone continues to resonate for some time after the interval in which it was played was supposed to have "ended." (In reality, of course, a live harpist could quiet the instrument's strings by stopping the vibration with his/her hand, but the Korg does not simulate this, essentially playing everything *legato*.) This feature makes the piano somewhat more "well-behaved" than the harp from the viewpoint of automated musical recognition, because in interval I_i there is very little sound that continues to resonate from interval I_{i-1} . In the frequency domain, however, the synthesized piano exhibits a slightly more complex harmonic signature, with its first *two* harmonics quite strong and some other residual components at high frequencies.

Finally, the continuous instrument I chose to include in my set was the *violin*, both because of its widespread use in music and because its waveform characteristics appeared challenging but not intractable. As is typical of continuous instruments, it has a much slower attack than would a harp or piano, and instead of exponential decay, it holds its magnitude relatively constant throughout the interval in which it plays. Harmonically, it is significantly more complex than either of the other instruments. As seen in Appendix B, the Korg's synthesized violin boasts substantial peaks for at least the first eight harmonics, though the first three or four are usually the strongest. Intuitively, however, this complexity ought to help rather than hinder the pitch extraction algorithm if the other voices present are a harp and violin, because the large number of harmonics should make the violin more easily distinguishable from the others.

Solo Instruments

Before we approach the complex topic of pitch extraction in the presence of harmonic ambiguity, it is worthwhile to note that there exist a limited number of situations where no complicated pitch extraction techniques are necessary. Specifically, *any song in which no harmonic ambiguity exists* may be analyzed in much the same way we analyzed songs that contained the simplified model. The only difference is that instead of labelling each FFT peak as the fundamental frequency of a note, we label *the lowest frequency in each harmonically related set* as a fundamental.

We can describe this quite simple procedure using the terminology of the more advanced pitch extraction algorithm given in Chapter 3. For each interval I_i , we form the spectrum S_i of "real" frequency domain peaks and amplitudes. Then we form the $Q_{i,j}$ sets for $j=1, \dots, J'_{MAX}$. If there is no harmonic ambiguity present in the song, it follows that the lowest frequencies in each of these $Q_{i,j}$ sets - namely, $Q_{i,j}|f_1$ for $j=1, \dots, J'_{MAX}$ - represent the notes present in the interval I_i . This is essentially a degenerate case of the advanced pitch extraction algorithm, where all intervals are well-behaved: $\{ I_i \} \equiv \{ W_i \}$. Any song in which no harmonic ambiguity exists could be correctly analyzed in this way; though only the pitches of the notes in each interval have been identified, not which of the J'_{MAX} voices each of them corresponds to. (This latter information could be derived by using the same kind of harmonic signature comparison that the advanced algorithm makes use of; however, this step has been omitted below.)

The simplest case of a song in which no harmonic ambiguity exists is that of a solo instrument. Only one voice is present in such a song, and therefore, no ambiguity could possibly occur. I used the straightforward algorithm described above to analyze songs containing only one of the three instruments at a time. The results are shown on pages 84-86. It is clear from the program's accuracy that solo melodies can be

easily analyzed using this elementary procedure, even if the instruments are "real" and not as simplistic as the rectangular-envelope, sinusoidal model.

"O Worship the King" - Harp solo



Three staves of musical notation for a harp solo. The key signature is two sharps (F# and C#) and the time signature is 3/4. The first staff contains the first four notes of the melody. The second staff contains the next four notes. The third staff contains the final four notes, including a whole rest at the end.

Output from Score version 1.0:



Three staves of musical notation, identical to the original score above. The key signature is two sharps (F# and C#) and the time signature is 3/4. The first staff contains the first four notes of the melody. The second staff contains the next four notes. The third staff contains the final four notes, including a whole rest at the end.

Results: 23 notes correct
1 note incorrect due to wrong pitch (long decay)

"O Worship the King" - Piano solo



Musical score for "O Worship the King" - Piano solo. The score is written in treble clef, key signature of two sharps (F# and C#), and 4/4 time. It consists of three staves. The first staff begins with a treble clef, a key signature of two sharps, and a 4/4 time signature. The melody consists of quarter notes: G4, A4, B4, C5, B4, A4, G4. The second staff continues the melody with quarter notes: F#4, G4, A4, B4, A4, G4, F#4. The third staff continues with quarter notes: E4, F#4, G4, A4, G4, F#4, E4, followed by a whole rest.

Output from Score version 1.0:



Musical score for "O Worship the King" - Piano solo. The score is written in treble clef, key signature of two sharps (F# and C#), and 4/4 time. It consists of three staves. The first staff begins with a treble clef, a key signature of two sharps, and a 4/4 time signature. The melody consists of quarter notes: G4, A4, B4, C5, B4, A4, G4. The second staff continues the melody with quarter notes: F#4, G4, A4, B4, A4, G4, F#4. The third staff continues with quarter notes: E4, F#4, G4, A4, G4, F#4, E4, followed by a whole rest.

Results: 24 notes correct
0 notes incorrect

"O Worship the King" - Violin solo



Three staves of musical notation for a violin solo. The key signature is one sharp (F#) and the time signature is 3/4. The first staff contains the first three measures. The second staff contains the next three measures. The third staff contains the final three measures, ending with a double bar line.

Output from Score version 1.0:



Three staves of musical notation for a violin solo, identical to the first section. The key signature is one sharp (F#) and the time signature is 3/4. The first staff contains the first three measures. The second staff contains the next three measures. The third staff contains the final three measures, ending with a double bar line.

Results: 24 notes correct
2 notes incorrect due to false events

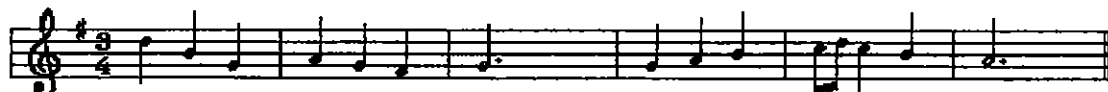
The "Soprano-Alto Problem"

Another class of songs which contain no harmonic ambiguity are those in which two or more instruments play simultaneously, but are never more than an octave apart in pitch. This is common in songs featuring a melody with a closely associated harmony or harmonies, such as a soprano-alto duet in choral music. The simple procedure for pitch extraction outlined above can also be used to correctly derive the score from such pieces, without having to employ the full pitch extraction algorithm from Chapter 3.

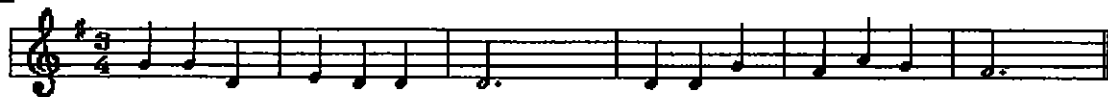
Recall that the harmonics of a note are at integer multiples of its fundamental frequency. This implies that the second harmonic of a given note is at twice the fundamental, or a full octave above the note's true pitch. Therefore, the closest two notes can be and still result in harmonic ambiguity is one octave apart. It follows that any song in which all the parts are within an octave of each other will be free from ambiguity, and can be analyzed using the simplified procedure. A number of these "soprano-alto" performances were analyzed in this way, with the results on the following pages.

"Come Thou Almighty King" - Two piano duet

Piano



Piano



Output from Score version 1.0:



Results: 30 notes correct
1 notes incorrect due to wrong pitch (long decay)

(Observe that the *Score version 1.0* output simply reports what notes were present in which intervals, and makes no attempt to identify a sequence of notes with a particular part. This is because this version of the program is not performing the pitch extraction algorithm. Later on in this chapter, we will see the output for version 2.0, which does perform this algorithm.)

The *Score* analysis of a "duet" with two piano voices is shown on page 88. Observe that the performance of the algorithm on this input is comparable to that with the simplified model from Chapter 3. When we shift to a harp on the soprano voice and a violin on the alto voice (results on page 91), we encounter a difficulty which may or may not prove to be serious in the long haul. We saw before that the event detection algorithm is very successful in identifying the true events when all parts are playing the same rhythm - in other words, when all voices begin a new note at the same time. It is more difficult, however, to identify *one* voice changing notes when the *other* voices are all holding steady. Notice that on page 91 the program missed the harp's D in measure 5, incorrectly merging the two eighth notes into a single quarter note. The reason for this mistake is that the violin was holding an F# throughout the interval, which made the harp's second eighth note harder for the linear predictor to pick up. Overall, however, we see that the program's performance for this duet of real instruments is quite accurate.

Pages 92-94 contain the results when a third voice is added. A comparison of the two program outputs illustrates the principle described in Chapter 3 regarding the position of the FFT within the interval. The first time the program was run (output on page 92) the FFT's were centered within each interval; therefore, for longer intervals, the samples fed to the FFT were further from the start of the note than for shorter intervals. A result of this decision is that the spectra of continuous instruments (the violin) overwhelm the spectra of striking instruments (harp and piano) in the longer intervals. (For instance, notice the dotted half-notes in intervals 8 and 16. Only the violin's note was picked up by the pitch extraction algorithm.) This phenomenon is due to the fact that as the interval progresses, the continuous instruments are still at full strength, while the amplitudes of the striking instruments have fallen off dramatically. We can mostly fix this problem by using the approach given in Chapter

3: place the first sample of the FFT the same distance away from the start of the interval regardless of the interval size. The outcome is shown on page 93, where all three voices come through even in the longer intervals.

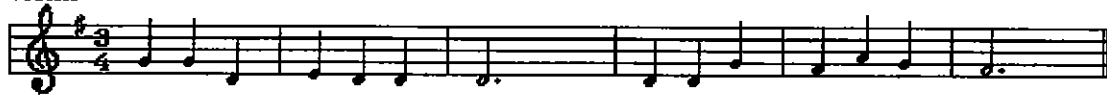
A final example of "soprano-alto" analysis is shown on page 94. In this piece, the instruments were placed in a different orientation than in the piece we just discussed (now the piano plays the descant, the violin the soprano melody, and the harp the alto harmony.) This example was included to demonstrate that the algorithm works regardless of which instrument is playing the highest and lowest parts. One last observation we might make about these three-part songs is that in general, the performance of the *Score* program seems to have degraded somewhat. It still tracks the correct notes with reasonable accuracy, but we find that it makes a few more slip-ups than in the two-instrument or solo cases. This is probably merely because there is more going on in the song, and this larger amount of data to track yields a higher probability of these minor errors.

"Come Thou Almighty King" - Harp-violin duet

Harp



Violin



Output from Score version 1.0:



Results: 29 notes correct
2 notes incorrect due to missed event

"Come Thou Almighty King" - Harp-piano-violin trio
FFT's taken in the center of each interval

Harp



Piano



Violin



The image shows three staves of musical notation for the piece "Come Thou Almighty King". The top staff is for Harp, the middle for Piano, and the bottom for Violin. All three parts are in G major (one sharp) and 3/4 time. The Harp part features a melodic line with some chords. The Piano part provides harmonic support with chords and some melodic fragments. The Violin part plays a steady, rhythmic accompaniment.

Output from Score version 1.0:



The image shows a single staff of musical notation representing the output from Score version 1.0. It is in G major and 3/4 time. The notation consists of vertical stems and horizontal lines representing notes, with some chords indicated by multiple stems at the same time.

Results: 38 notes correct
6 notes incorrect (missed) due to pitch extraction failure
2 notes incorrect due to false event

"Come Thou Almighty King" - Harp-piano-violin trio
 FFT's taken the same distance from the start of each interval,
 regardless of interval size

Harp

Piano

Violin

The image shows three staves of musical notation for the piece "Come Thou Almighty King". The top staff is for Harp, the middle for Piano, and the bottom for Violin. All three parts are in G major (one sharp) and 3/4 time. The Harp part features a melodic line with some grace notes. The Piano part provides harmonic support with chords and moving lines. The Violin part plays a steady, rhythmic accompaniment.

Output from Score version 1.0:

The image shows a single staff of musical notation representing the output from Score version 1.0. It displays a sequence of notes and chords, likely representing the detected notes from the original score. The notation includes stems, beams, and chord symbols, showing the results of the pitch extraction process.

Results: 42 notes correct
 2 notes incorrect (missed) due to pitch extraction failure
 1 note incorrect (falsely identified) due to pitch
 extraction failure
 2 notes incorrect due to false event

"Great is Thy Faithfulness" - Piano-violin-harp trio

Piano



Violin



Harp



The image shows three staves of musical notation for the piece "Great is Thy Faithfulness". The top staff is for Piano, the middle for Violin, and the bottom for Harp. All three parts are in G major (one sharp) and 3/4 time. The Piano part features a melodic line with eighth and quarter notes. The Violin part plays a similar melodic line. The Harp part provides a harmonic accompaniment with chords and single notes.

Output from Score version 1.0:



The image shows a single staff of musical notation representing the output from Score version 1.0. It is in G major and 3/4 time, matching the original score. The notation consists of a series of chords and single notes, representing the extracted notes from the original score.

Results: 35 notes correct
1 note incorrect (falsely identified) due to pitch
extraction failure
1 note incorrect due to wrong pitch

Recognition of Songs Containing Harmonic Ambiguity

We were able to avoid having to use the complicated pitch extraction algorithm detailed in Chapter 3 when we analyzed songs in the two categories above. Now, however, we have exhausted these possibilities, and we must face up to the fact that most practical songs will contain harmonic ambiguity, and potentially lots of it. A number of short songs that had intervals with this property were used as input to the *Score* program, with results as shown below.

(Note: All of the preceding examples of *Score* output were from "*Score version 1.0*," which implemented a subset of the entire automated musical recognition algorithm. The examples below are obtained from "*Score version 2.0*." The former differs from the latter in two important respects: (1) Version 1.0 treated each interval individually, without concern for any of the others. Hence, it did not attempt to associate a sequence of individual notes throughout the song with a particular voice, but rather simply reported the notes that were found in each interval, without regard to which voice produced them. (2) Version 1.0 did not treat the problem of harmonic ambiguity. This actually follows directly from (1), because no mean harmonic signatures can be formed with which to resolve the ambiguities if no attempt is made to identify individual notes with voices.)

First, we will demonstrate the basic ability of the algorithm to determine which notes belong to which voices. In the following sample output, the three parts of the song "cross over" each other in pitch - each voice is the highest in some intervals, the lowest in others, and the middle in others. From the resulting output we see that *Score* has correctly determined which notes were associated with the same voice.



```

*****
*****
**                S C O R E !                **
**                Stephen Davies - version 2.0          **
*****
**
** Current system parameters:                      **
**   Filename of sound file to use:  MUSIC.DAT        **
**   Sampling frequency of sound file:  8000 Hz      **
**   Tempo of song: 120 beats per minute             **
**   Minimum length of a note: 1/ 4 whole notes    **
**   Trim threshold: 0.5 standard deviations         **
**   Gain of linear predictor: 10% of maximum       **
**   Window size of linear predictor: 10            **
**   Event detector STA window size: 6             **
**   Event detector LTA window size: 500           **
**   Event detector power ratio threshold: 4        **
**   Incremental FFT size: 2048 samples             **
**   FFT size tolerance: 80% of shortest note      **
**   Pitch extraction cut-off factor: 500           **
**   Pitch extraction dynamic range: 0.05           **
**   Reporting accidentals in: sharps              **
**   Instrument type: real                          **
**   Score mode: verbose                            **
**
*****
*****

```

Preparing song for analysis...

Loading sound file...

Removing DC bias...

Trimming song...

Song now has 17811 samples.

Analyzing song to locate notes...

Compiling linear predictor error...

Analyzing linear predictor error for events...

Performing additional note detection heuristics...

Performing pitch extraction...

Computing FFT spectra...

Decomposing spectra into harmonically independent components...

Identifying well-behaved and poorly-behaved intervals...
 Computing mean harmonic signatures...
 Analyzing poorly-behaved intervals...

The score is:

=====

Voice 1:

1 quarter-note: G#4 (418 Hz.)

2 quarter-note: E5 (664.1 Hz.)

3 quarter-note: G4 (395.3 Hz.)

4 dotted-quarter: D4 (296.1 Hz.)

=====

Voice 2:

1 quarter-note: F#4 (373.4 Hz.)

2 quarter-note: D5 (598.4 Hz.)

3 quarter-note: A4 (443.7 Hz.)

4 dotted-quarter: F4 (351.6 Hz.)

=====

Voice 3:

1 quarter-note: E4 (331.2 Hz.)

2 quarter-note: C5 (526.6 Hz.)

3 quarter-note: C#5 (557.8 Hz.)

4 dotted-quarter: E4 (332 Hz.)

=====

Results: 12 notes correct

(As before, I will include a representation of *Score*'s output in musical notation for all of the following version 2.0 examples, rather than the text output as seen above.)

Recall that our definition of a poorly-behaved interval was one in which not all parts of the song appear unambiguously. This can occur if the interval contains harmonic ambiguity *or* simply if one of the parts has a rest during the interval. In the following output, *Score* is seen to have correctly identified when the violin has a rest.

The image shows two staves of music in 4/4 time. The top staff is labeled 'Violin' and the bottom staff is labeled 'Piano'. Both staves have a treble clef. The Violin staff contains four notes: G4, A4, a whole rest, and B4. A 'P' is written above the whole rest. The Piano staff contains four notes: G4, A4, B4, and C5.

(Poorly behaved interval marked with a "P".)

Output from *Score* version 2.0:

The image shows two staves of music in 4/4 time. The top staff is labeled 'Part 1' and the bottom staff is labeled 'Part 2'. Both staves have a treble clef. Both parts play the same sequence of notes: G4, A4, B4, and C5.

Results: 8 notes correct

The simplest example of an interval with harmonic ambiguity is one in which two voices are playing in unison. The following output demonstrates the program's ability to correctly detect voices playing the same pitch.

Harp P
 Violin
 Piano

(Poorly behaved interval marked with a "P".)

Output from Score version 2.0:

Part 1
 Part 2
 Part 3

Results: 12 notes correct

The closest two notes can be in pitch and still have harmonic ambiguity is one octave. *Score* correctly identifies two notes an octave apart in the following output.

Part 1
 Part 2
 Part 3

(Poorly behaved interval marked with a "P".)

Output from Score version 2.0:

Part1



Part2



Part3



Results: 12 notes correct

Now that we have demonstrated the basic capabilities of the pitch extraction algorithm to resolve resting voices and harmonic ambiguity, we proceed to test it on a set of more "realistic" songs with several well-behaved and poorly-behaved intervals. As is seen from the resulting outputs, the performance of the program is encouraging, but less than perfect. It appears that the longer and more complex the song, the more chances there are for something to go wrong. Nevertheless, with a modest amount of "parameter tweaking," the following fairly acceptable results can be achieved.

"Immortal, Invisible, God only Wise"

Part 1



Part 2



Part 3



The image shows three staves of musical notation for the hymn "Immortal, Invisible, God only Wise". Each staff is labeled "Part 1", "Part 2", and "Part 3" respectively. The music is written in treble clef, with a key signature of one sharp (F#) and a time signature of 3/4. The notes are: Part 1: G4, A4, B4, C5, B4, A4, G4, F#4, E4, D4, C4; Part 2: G4, A4, B4, C5, B4, A4, G4, F#4, E4, D4, C4; Part 3: G4, A4, B4, C5, B4, A4, G4, F#4, E4, D4, C4.

(Poorly behaved intervals marked with a "P".)

Output from Score version 2.0:

Part 1



Part 2



Part 3



This image is identical to the one above, showing the musical notation for three parts of the hymn. The notation is consistent across both images.

Results: 30 notes correct
3 notes incorrect due to wrong pitch

"And the Glory of the Lord" - (from Handel's Messiah)

The image shows two systems of musical notation for three instruments: Harp, Violin, and Piano. The key signature is two sharps (F# and C#) and the time signature is 3/4. The first system has dynamic markings **P**, **P P P**, and **P** above the Harp staff. The second system has dynamic markings **PP**, **P**, and **P** above the Harp staff. In both systems, the Harp and Violin parts have several intervals marked with a "P" above them, indicating poorly behaved intervals. The Piano part is written in the bass clef.

(Poorly behaved intervals marked with a "P".)

Output from Score version 2.0:

The image shows two systems of musical notation for three parts: Part 1, Part 2, and Part 3. The key signature is two sharps (F# and C#) and the time signature is 3/4. The first system has dynamic markings **P**, **P P P**, and **P** above the Part 1 staff. The second system has dynamic markings **PP**, **P**, and **P** above the Part 1 staff. In both systems, the Part 1 and Part 2 parts have several intervals marked with a "P" above them, indicating poorly behaved intervals. The Part 3 part is written in the bass clef. The word "Sva..." is written above the Part 1 staff in the second system.

Results: 59 notes correct
 1 note incorrect due to false event
 7 notes incorrect due to wrong pitch
 2 notes incorrect due to missed events

For completeness, here is the actual text output from the *Score* program for this example:

```
*****
*****
**                               S C O R E !                               **
**                               Stephen Davies - version 2.0                       **
*****
**                               **
** Current system parameters: **
**   Filename of sound file to use: MUSIC.DAT **
**   Sampling frequency of sound file: 8000 Hz **
**   Tempo of song: 120 beats per minute **
**   Minimum length of a note: 1/ 8 whole notes **
**   Trim threshold: 0.5 standard deviations **
**   Gain of linear predictor: 11% of maximum **
**   Window size of linear predictor: 10 **
**   Event detector STA window size: 6 **
**   Event detector LTA window size: 500 **
**   Event detector power ratio threshold: 5 **
**   Incremental FFT size: 2048 samples **
**   FFT size tolerance: 80% of shortest note **
**   Pitch extraction cut-off factor: 1100 **
**   Pitch extraction dynamic range: 0.05 **
**   Reporting accidentals in: sharps **
**   Instrument type: real **
**   Score mode: brief **
**                               **
*****
*****
```

Preparing song for analysis...

Loading sound file...

Removing DC bias...

Trimming song...

Note: specified FFT size 2048 is greater than 80% of the minimum number of samples per note, 2000. Resampling original signal appropriately.

Song now has 129998 samples.

Analyzing song to locate notes...

Performing pitch extraction...

Computing FFT spectra...

Decomposing spectra into harmonically independent components...

Identifying well-behaved and poorly-behaved intervals...

Intervals 2, 3, 4, 5, 6, 10, 13, 15, 16, 18, 19, 20, 21, 22, 24, are well-behaved.

Intervals 1, 7, 8, 9, 11, 12, 14, 17, 23, are poorly-behaved.

Computing mean harmonic signatures...
 Biasing signatures with interval 18...
 Analyzing poorly-behaved intervals...

The score is:

=====

Voice 1:

| | | | |
|----|-----------------|-----|--------------|
| 1 | quarter-note: | B3 | (248.4 Hz.) |
| 2 | quarter-note: | D#4 | (312.9 Hz.) |
| 3 | dotted-quarter: | C#4 | (278.9 Hz.) |
| 4 | 8th-note: | C#4 | (278.9 Hz.) |
| 5 | 8th-note: | D#4 | (314.1 Hz.) |
| 6 | 8th-note: | D#4 | (312.9 Hz.) |
| 7 | quarter-note: | E4 | (331.6 Hz.) |
| 8 | quarter-note: | B3 | (248.4 Hz.) |
| 9 | quarter-note: | C#4 | (278.9 Hz.) |
| 10 | quarter-note: | B3 | (248.4 Hz.) |
| 11 | quarter-note: | B3 | (248.4 Hz.) |
| 12 | quarter-note: | D#4 | (314.1 Hz.) |
| 13 | 8th-note: | C#4 | (278.9 Hz.) |
| 14 | 8th-note: | B3 | (248.4 Hz.) |
| 15 | quarter-note: | A#3 | (234.4 Hz.) |
| 16 | quarter-note: | B3 | (248.4 Hz.) |
| 17 | quarter-note: | C#4 | (278.9 Hz.) |
| 18 | quarter-note: | C#4 | (278.9 Hz.) |
| 19 | quarter-note: | D#4 | (314.1 Hz.) |
| 20 | quarter-note: | C#4 | (278.9 Hz.) |
| 21 | quarter-note: | C#4 | (278.9 Hz.) |
| 22 | quarter-note: | D#4 | (314.1 Hz.) |
| 23 | quarter-note: | C#4 | (278.9 Hz.) |
| 24 | quarter-note: | A#3 | (234.4 Hz.) |

=====

Voice 2:

=====

| | | | |
|----|-----------------|---------|--------------|
| 1 | quarter-note: | B3 | (248.4 Hz.) |
| 2 | quarter-note: | F#4 | (371.5 Hz.) |
| 3 | dotted-quarter: | F#4 | (371.5 Hz.) |
| 4 | 8th-note: | F#4 | (371.5 Hz.) |
| 5 | 8th-note: | B3 | (248.4 Hz.) |
| 6 | 8th-note: | B3 | (248.4 Hz.) |
| 7 | quarter-note: | (Rest.) | |
| 8 | quarter-note: | (Rest.) | |
| 9 | quarter-note: | F#4 | (371.5 Hz.) |
| 10 | quarter-note: | F#4 | (371.5 Hz.) |
| 11 | quarter-note: | F#4 | (372.7 Hz.) |
| 12 | quarter-note: | B4 | (495.7 Hz.) |
| 13 | 8th-note: | A#4 | (466.4 Hz.) |
| 14 | 8th-note: | G#4 | (418.4 Hz.) |
| 15 | quarter-note: | F#4 | (372.7 Hz.) |
| 16 | quarter-note: | F#4 | (371.5 Hz.) |
| 17 | quarter-note: | F#4 | (371.5 Hz.) |
| 18 | quarter-note: | F#4 | (372.7 Hz.) |
| 19 | quarter-note: | F#4 | (371.5 Hz.) |
| 20 | quarter-note: | F#4 | (371.5 Hz.) |
| 21 | quarter-note: | F#4 | (371.5 Hz.) |
| 22 | quarter-note: | F#4 | (372.7 Hz.) |
| 23 | quarter-note: | F#4 | (372.7 Hz.) |
| 24 | quarter-note: | F#4 | (372.7 Hz.) |

Voice 3:

| | | | |
|---|-----------------|-----|--------------|
| 1 | quarter-note: | F#4 | (371.5 Hz.) |
| 2 | quarter-note: | B4 | (495.7 Hz.) |
| 3 | dotted-quarter: | A#4 | (466.4 Hz.) |
| 4 | 8th-note: | G#4 | (416 Hz.) |

| | | | |
|----|---------------|---------|--------------|
| 5 | 8th-note: | F#4 | (371.5 Hz.) |
| 6 | 8th-note: | F#4 | (371.5 Hz.) |
| 7 | quarter-note: | B4 | (494.5 Hz.) |
| 8 | quarter-note: | E4 | (331.6 Hz.) |
| 9 | quarter-note: | (Rest.) | |
| 10 | quarter-note: | D#5 | (624.6 Hz.) |
| 11 | quarter-note: | B3 | (248.4 Hz.) |
| 12 | quarter-note: | (Rest.) | |
| 13 | 8th-note: | D#7 | (2549 Hz.) |
| 14 | 8th-note: | B3 | (248.4 Hz.) |
| 15 | quarter-note: | C#5 | (556.6 Hz.) |
| 16 | quarter-note: | D#5 | (623.4 Hz.) |
| 17 | quarter-note: | C#5 | (557.8 Hz.) |
| 18 | quarter-note: | A#4 | (468.7 Hz.) |
| 19 | quarter-note: | B4 | (495.7 Hz.) |
| 20 | quarter-note: | D#7 | (2549 Hz.) |
| 21 | quarter-note: | A#4 | (466.4 Hz.) |
| 22 | quarter-note: | B4 | (495.7 Hz.) |
| 23 | quarter-note: | C#5 | (557.8 Hz.) |
| 24 | quarter-note: | C#5 | (556.6 Hz.) |

As we might have expected, the program's performance is at its worst during the poorly-behaved intervals. *Score* seems to have very little trouble in identifying the pitches and properly associating notes with voices in the well-behaved intervals, but it simply cannot maintain a very high level of consistency in the others. Most likely, this is due principally to three factors: (1) the mean harmonic signatures may not perfectly reflect the "real" signatures for the voices involved, due to upper harmonic ambiguity in the well-behaved intervals. (2) The additions of signatures to form "guesses" in the poorly-behaved intervals may not be accurate. This could be because we are eliminating the deterministic phase shifts between peaks in the same signature and just taking the expected value with respect to all phases, which we assume are uniformly distributed. (3) The harmonic signatures of instruments from note to note may simply be too unstable to track with a high enough degree of accuracy.

It remains to be seen which of the three factors above is most responsible for the performance degradation in the poorly-behaved intervals: this could be a topic of future investigation. If (1) and/or (2) are the chief causes, then we might be able to devise a strategy for counteracting them. For instance, suppose it turns out that upper harmonic ambiguity in the well-behaved intervals causes the mean harmonic signatures to lose too much resolution. We could develop a method by which the well-behaved intervals that "poorly fit" the others (ie., those intervals that had significant error with respect to the mean harmonic signatures before they were averaged in) would be assumed to contain significant upper harmonic ambiguity. Then, a methodology similar to the one described in chapter 3 for the poorly-behaved intervals could be used for these well-behaved intervals, with the only difference being that we know no *fundamental* frequency has been obscured by a harmonic.

Or, suppose that (2) is the main problem: by taking the expected value of the amplitude over all possible phase differences when we add two sinusoids, we obtain an

unsatisfactory approximation to the true amplitude. The next logical step would be to use the phase information as well as the magnitude information of the harmonic signatures. In other words, when a particular instrument plays a sequence of notes, we would not only expect that the ratios of the *magnitudes* of the harmonic peaks to be roughly constant from note to note, but that the *phase differences* between peaks would be roughly constant. This is because the phases of the various frequency domain peaks have a huge impact on what the resulting time-domain waveform looks like, and it is really this time-domain waveform which we presume will remain reasonably constant. By ignoring this dependable characteristic of each instrument's frequency spectrum, we are really throwing away information which could be put to use. A more sophisticated approach would be to maintain complex-valued harmonic signatures, using the absolute phase difference between two simultaneous voices as another degree of freedom in our series of pitch extraction "guesses."

The only circumstance in which we will have run into a genuine dead end is if (3) turns out to be the foremost difficulty. If the harmonic signatures of instruments produced by an electronic synthesizer - let alone those produced by live acoustical instruments played with "feeling" and "interpretation" - are not sufficiently stable to track from note to note, then the entire approach to pitch extraction outlined in Chapter 3 is doomed to failure. Intuitively, we would not expect this to be the case. A human being can accurately discern between a trumpet and an oboe playing a duet on a radio station, and the only information he/she receives in this case is the same waveform the program receives. This tells us that conceptually, there *is* enough information present in the signal to make this determination reliably. No doubt a listener makes use of the time domain characteristics of instruments discussed in Chapter 2 as well as frequency domain information, but this author would hope - and

suspect - that frequency qualities alone would be sufficient to distinguish two voices, if only they were analyzed in the proper manner.

CHAPTER VI

CONCLUSION

Whatever else we may have learned from this study, it is clear that the human brain is a marvelously powerful organ. Given only a sound waveform and a few other audio clues which the stereo nature of the ears can make use of, a musically competent human can identify and describe the musical information present therein with a tremendous degree of accuracy. This appears to be true of a wide spectrum of different kinds of instruments, regardless of the genre of music being played, the acoustical characteristics of the performance environment, or the presence of noise and other audio distractions. A computer, on the other hand, given an identical waveform and programmed to employ a large number of quite sophisticated signal processing algorithms, can only match the human's performance under the best of circumstances. As an engineer, it is truly humbling to realize that despite one's most earnest attempts to mimic the activities of the human brain, one seems to consistently fall short of what simply "comes naturally" to multitudes of people ignorant of signal processing basics.

Nonetheless, we have gained some ground. The techniques for automated musical recognition described in this thesis appear to be somewhat successful in analyzing a small set of simple songs, even when the most daunting challenge to the endeavor - the presence of harmonic ambiguity - is present. At the very least, we have succeeded in precisely characterizing the nature of musical waveforms to the point where even more sophisticated approaches might be attempted. At most, we have laid a basic framework which might be combined with additional heuristics to provide a realizeable solution to the problem of automated musical recognition.

Whether the outputs given at the end of Chapter 5 would be considered "satisfactory" depend on the nature of one's application. Certainly, if they were played for a human listener, he/she could identify the song, even though a few notes might be incorrect. If the goal was to produce a written score which could be used as a performance guideline, an experienced musician would surely be able to recognize the errors and add their own improvisation. And if an instrumentalist, who was not particularly adept at transcribing to musical notation, were to use the program on a "jam session" recording, he/she would be able to avoid much of the tedium of putting things down on paper, and instead merely fix a few erroneous notes. So clearly, there exist a number of applications where even the performance level achieved by the *Score* prototype would be acceptable and helpful.

To faithfully and dependably produce musical scores for publishing purposes, however, we must have loftier goals. The results produced by *Score* do not appear to be consistently accurate enough to simply accept without some amount of editing. The most realistic solution seems to be to fall back on a joint computer/human approach, whereby the program cranks out an "approximation" to the true score, and the human then analyzes this and corrects errors. One could even envision this correction being done aurally, with the computer playing the original song along with the notes of the approximation so that differences might easily stand out to the listener. It might also be possible to add yet another layer of automated processing to the program's output, such that "musically unlikely" outcomes could be rejected and others tried. For instance, on the output on page 101, a routine might notice voice 3's tremendous jumps down to C#4 and D4 in measures 3 and 4 and reject them on the grounds of musical implausibility. It could then send feedback to the original program, which would re-examine these intervals and weigh the more likely results more heavily.

For now, we simply remark that the language of music is extremely intricate, diverse, and not even fully understood. The distinction between different kinds of musical events (quarter-note, or eighth-note/eighth-rest? F#, or F? Piccolo voice, or flute voice?) can easily become blurry to the point where the decision between them is as much a product of subjective opinion as of anything else. Given such an inherently "fuzzy" input, then, we should not be surprised that any rigorously concrete algorithm will produce some number of errors. Our best hope is that we can nail down the problem space to something we can get handles on, and devise a general solution to approximate what happens in the listener's mind under ideal circumstances. Perhaps as more is understood about musical phenomena and human perception of them, we will be able to apply additional methods to these basic techniques and yield a system with true human-like performance.

CHAPTER VII

FUTURE AREAS OF INVESTIGATION

Most future research topics in this area would center around finding ways to improve the system's accuracy. Under what circumstances does the program perform poorly, and what can be done to correct the resulting errors?

Since the least dependable part of the program is clearly the pitch extraction in poorly-behaved intervals, we would do best to begin there. It would certainly be prudent to investigate the possibilities of maintaining complex harmonic signatures for the various voices, since intuitively this is information which can and should be exploited. It is unknown at this time whether this would result in a substantial performance increase. Also, a study into the effects of upper harmonic ambiguity on the mean harmonic signature computations is in order. If it turns out that the reason for performance degradation is unreliable Σ_j 's, techniques as described at the end of Chapter 5 might be employed to improve this reliability.

It is also probably worth searching for a more sophisticated parameter selection process. Most of the outputs presented in this thesis were obtained only after a few minutes of adjusting the linear predictor gain, frequency peak detection threshold, FFT window sizes, and other parameters. This was done with *a priori* knowledge about what the output should be, which calls into question the program's utility for the general user who will usually not, of course, have previous knowledge of the score. It is possible that an adaptive mechanism could be added to provide feedback to the program about parameter adjustment. For instance, if only a small number of events are detected, then the linear predictor might need adjustment; if all the mean harmonic signatures look alike, the peak detection threshold needs to be

lowered, etc. Hopefully, this process could be fully automated so that these details of the algorithm's operation can be hidden from the user.

If the program is given additional information about what it is likely to see, it is possible that performance might be dramatically improved. The situations under which this is likely to occur would be an appropriate topic for further research. For example, suppose that *Score* is given a library of harmonic signatures for certain instruments under particular conditions, and informed which instruments will be present in the recording. This would eliminate the need to acquire mean harmonic signatures, since perfectly accurate ones would be provided at the outset. As we noted in Chapter 2, every individual violin, piano, etc. has its own unique characteristics, but we might even eliminate this variance by including a "training" session before the main algorithm. Each instrument to be included in the recording could play a scale through its entire pitch range so that the program could "lock on" to that *particular* instrument rather than to some global notion of "the violin signature for all violins." Similarly, we might see how the program performed if it were first trained on a different song, but with the same instruments playing. If successful, this would be useful in situations where scores were periodically desired from a particular orchestra or band, using the same instruments. After "warming up" on a couple of symphonies, the program might know enough to accurately identify the notes of future performances.

Finally, for the algorithm as described in Chapter 3 to work "as is," we presumed that every song would have at least one well-behaved interval. This is virtually the only assumption we made, but for completeness we should explore under what situations it is true. It would be worthwhile to conduct an investigation into the annals of music history to determine what genres of music tend to fulfill this requirement, and which, if any, do not. For those pieces which featured only poorly-behaved intervals, the kind of mean harmonic signature computations described earlier

would be impossible. We would have to modify our approach once again for these inputs, perhaps using the kinds of training techniques described above to avoid having to compute the signatures from the piece itself.

¹Exceptions to this rule occur during short periods when some instruments are just entering their pure tones; for instance, the "breathy" sound a flautist makes during the attack of a note, or the somewhat noisy component of a violin bow "catching" the string on initial contact. These effects are typically brief and have little impact on the overall spectral content of the instrument.

² The subtle distinction between the fundamental frequency of vibration and the perceived "pitch" of a note has given rise to quite a debate in the literature. For purposes of this thesis, however, I will ignore such nuances and use the fundamental to measure pitches directly.

³ *Musical Acoustics*, Charles A. Culver, McGraw-Hill, 1956, p.172.

⁴ Disregarding percussive and noisy instruments, of course.

⁵ These predictions were borne out clearly in the laboratory, where the division algorithm performed very poorly. For extremely short songs - say, eight or fewer notes - it can function fairly well provided the threshold for song trimming is set adequately. But for anything longer, the relationship between the segments and the song's actual notes becomes blurry and nonexistent.

⁶ The vector of weights is defined on the real numbers, provided that the input data is also real. The digitized music data analyzed by the *Score* program are all real.

⁷ As an aside, though the focus of this thesis was not to optimize the program's calculations to produce its output most efficiently, it should be noted that the most time-consuming part of the entire *Score* algorithm by far is the process of computing the linear predictor's error vector. Such a consideration speaks even more strongly in favor of using the LMS algorithm, a relatively speedy procedure.

⁸ Earlier, we noted that forcing the user to accurately specify the tempo of the song was requiring them "to do some of the work that the program ought to be doing." However, it should be clear that the use of the tempo T described here is not the same, because no great accuracy is required. In order to suppress false events that occur immediately after true ones, we require only a very rough approximation of the tempo so that we have an idea of how long after each accepted event we should disallow others.

⁹ The straightforward Discrete Fourier Transform of N data points requires approximately N^2 complex operations, while the FFT requires only $N \log_2 N$.

¹⁰ The *Score* program requires a minimum resolution in the frequency domain in order to distinguish adjacent pitches. If not enough samples are available in the shortest note's interval to provide this resolution, it automatically uses interpolation to increase the sampling rate of the signal so that the criteria is met.

¹¹ In practice, the N_{\min} is again subject to a relaxation factor to relieve the user of having to specify the song's tempo precisely. This relaxation factor is, in general,

different from the relaxation factor described in the event detection section of this chapter.

¹² The Midisoft Studio sequencer has a built in feature which will do exactly this. The underlying sound of the music remains the same while the user can request that certain aspects of the displayed score be modified. Straightforward techniques are available for implementing this.

¹³ I use the term "single-part" to refer to a song in which only one instrument is playing at any given time. "Multi-part" songs are those in which more than one instrument is playing simultaneously; ie., those with harmony.

BIBLIOGRAPHY

- [1] A. H. Benade. *Horns, Strings, and Harmony*. Anchor Books, New York, 1960.
- [2] J. Bibbo, D. Etter, and D. Breeding. A Software Tool for Processing Seismic Data. *Computers & Geosciences*, Vol. 17, No. 2, pp. 301-305, 1991.
- [3] C. A. Culver. *Musical Acoustics*. McGraw-Hill, 1956.
- [4] N. H. Fletcher and T. D. Rossing. *The Physics of Acoustical Instruments*. Springer-Verlag, New York, 1991.
- [5] S. Fletcher, S. D. Stearns, and P. Thompson. Some Considerations in Using an Adaptive Filter as an Event Detector. *Proc. 24th Midwest Symposium on Circuits and Systems*, June, 1981
- [6] S. Fletcher. *A Comparison of Seismic Event Detection Algorithms*. Sandia National Laboratories, Albuquerque, New Mexico, 1982.
- [7] H. Lou. "Implementing the Viterbi Algorithm." *IEEE Signal Processing*, Sept. 1995, Vol. 12, No. 5, pp.42-51. ISSN 1053-5888.
- [8] R. C. Maher. "An Approach for the Separation of Voices in Composite Musical Signals." University of Illinois at Urbana-Champaign, Dept. of Electrical Engineering, 1989.
- [9] J. Moorer. "On the Segmentation and Analysis of Continuous Musical Sound by Digital Computer." Stanford University, Dept. of Music, 1975.
- [10] J. G. Roederer. *Introduction to the Physics and Psychophysics of Music*. Springer-Verlag, New York, 1973.
- [11] S. D. Stearns and J. Vortman. Seismic Event Detection Using Adaptive Predictors. *Proc. ICASSP-81*, Atlanta, Georgia, April 1981.
- [12] W. H. Tranter and R. E. Ziemer. *Principles of Communications*. Houghton Mifflin Co., Boston, Massachusetts, 1995.
- [13] B. Widrow and S. D. Stearns. *Adaptive Signal Processing*. Prentice-Hall, New Jersey, 1985.