Library Declaration Form

University of Otago Library

Author's full name and year of birth: Philip Grant McLeod,
(for cataloguing purposes) 22 April 1980

Title of thesis: Fast, Accurate Pitch Detection Tools for Music Analysis

Degree: Doctor of Philosophy

Department: Department of Computer Science

Permanent Address: 2 Royston Street, Dunedin, NZ

I agree that this thesis may be consulted for research and study purposes and that reasonable quotation may be made from it, provided that proper acknowledgement of its use is made.

I consent to this thesis being copied in part or in whole for

i) a library

ii) an individual

at the discretion of the University of Otago.

Signature:

Date:

Fast, Accurate Pitch Detection Tools for Music Analysis

Philip McLeod

a thesis submitted for the degree of

Doctor of Philosophy

at the University of Otago, Dunedin,

New Zealand.

30 May 2008

Abstract

Precise pitch is important to musicians. We created algorithms for real-time pitch detection that generalise well over a range of single 'voiced' musical instruments. A high pitch detection accuracy is achieved whilst maintaining a fast response using a *special normalisation of the autocorrelation* (SNAC) function and its windowed version, WSNAC. Incremental versions of these functions provide pitch values updated at every input sample. A robust octave detection is achieved through a *modified cepstrum*, utilising properties of human pitch perception and putting the pitch of the current frame within the context of its full note duration. The algorithms have been tested thoroughly both with synthetic waveforms and sounds from real instruments. A method for detecting note changes using only pitch is also presented.

Furthermore, we describe a real-time method to determine vibrato parameters - higher level information of pitch variations, including the envelopes of vibrato speed, height, phase and centre offset. Some novel ways of visualising the pitch and vibrato information are presented.

Our project 'Tartini' provides music students, teachers, performers and researchers with new visual tools to help them learn their art, refine their technique and advance their fields.

Acknowledgements

I would like to thank the following people:

- Geoff Wyvill for creating an environment for knowledge to thrive.
- Don Warrington for your advice and encouragement.
- Professional musicians Kevin Lefohn (violin) and Judy Bellingham (voice) for numerous discussions and providing us with samples of good sound.
- Stuart Miller, Rob Ebbers, Maarten van Sambeek for the contributions in creating some of Tartini's widgets.
- Damon Simpson, Robert Visser, Mike Phillips, Ignas Kukenys, Damon Smith, JP, Yaoyao Wang, Arthur Melissen, Natalie Zhao, and all the people at the Graphics Lab for all the advice, fun, laughter and great times shared.
- Alexis Angelidis for the C++ tips and the inspiration.
- Brendan McCane for the words of wisdom, the encouragement and always leaving the smell of coffee in the air.
- Sui-Ling Ming-Wong for keeping the lab in order.
- Nathan Rountree for the Latex thesis writing template, good general advice and sharing some musical insights.
- The University of Otago Music Department String class of 2006 for working together for a workshop.
- The Duck (lab canteen), and Countdown for being open 24/7.
- My mum and dad.

Thank you all.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 2
- 1.2 Limit of scope . . . 3
- 1.3 Contributions . . . 3
- 1.4 Thesis overview . . . 4

2 Background **6**
- 2.1 What is Pitch? . . . 6
  - 2.1.1 MIDI note numbering . . . 7
  - 2.1.2 Seebeck's Siren . . . 7
  - 2.1.3 Virtual Pitch . . . 8
- 2.2 Pitch Detection History . . . 9
- 2.3 Time Domain Pitch Algorithms . . . 11
  - 2.3.1 Simple Feature-based Methods . . . 11
  - 2.3.2 Autocorrelation . . . 12
  - 2.3.3 Square Difference Function (SDF) . . . 17
  - 2.3.4 Average Magnitude Difference Function (AMDF) . . . 18
- 2.4 Frequency Domain Pitch Algorithms . . . 19
  - 2.4.1 Spectrum Peak Methods . . . 23
  - 2.4.2 Phase Vocoder . . . 24
  - 2.4.3 Harmonic Product Spectrum . . . 24
  - 2.4.4 Subharmonic-to-Harmonic Ratio . . . 25
  - 2.4.5 Autocorrelation via FFT . . . 26
- 2.5 Other Pitch algorithms . . . 27
  - 2.5.1 Cepstrum . . . 27
  - 2.5.2 Wavelets . . . 31
  - 2.5.3 Linear Predictive Coding (LPC) . . . 33

3 Investigation **34**
- 3.1 Goals and Constraints . . . 34
  - 3.1.1 Pitch Accuracy . . . 35
  - 3.1.2 Pitch range . . . 36
- 3.2 Responsiveness . . . 37
- 3.3 Investigation of some Existing Techniques . . . 38
  - 3.3.1 SDF vs Autocorrelation . . . 38
  - 3.3.2 Calculation of the Square Difference Function . . . 41

3.3.3 Square Difference Function via Successive Approximation 41
3.3.4 Square Difference Function via Autocorrelation 42
3.3.5 Summary of ACF and SDF properties 43

4 A New Approach **45**
4.1 Special Normalisation of the Autocorrelation (SNAC) Function 45
4.2 Windowed SNAC Function 48
4.2.1 Crosscorrelation via FFT 49
4.2.2 Combined Windowing Functions 49
4.3 Parabolic Peak Interpolation 51
4.4 Clarity Measure 52

5 Experiments - The Autocorrelation Family **53**
5.1 Stationary Signal 54
5.2 Frequency Changes 58
5.2.1 Linear Frequency Ramp 58
5.2.2 Frequency Modulation, or Vibrato 64
5.3 Amplitude Changes 74
5.3.1 Linear Amplitude Ramp 74
5.3.2 Amplitude Step 78
5.4 Additive Noise 80
5.5 Summary 81

6 Choosing the Octave **85**
6.1 Measuring Accuracy of Peak Picking 85
6.2 Can the Fundamental Frequency and the Pitch Frequency be Different? 88
6.2.1 Pitch Perception 89
6.2.2 Outer/Middle Ear Filtering 90
6.3 Peak Picking Algorithm 93
6.3.1 Investigation 93
6.3.2 The Algorithm 95
6.3.3 Results 98
6.4 Using the Cepstrum to choose the Periodic Peak 99
6.4.1 The Modified Cepstrum 101

7 Putting the Pitch in Context **105**
7.1 Median Smoothing 105
7.2 Combining Lag Domain Peaks 107
7.2.1 Aggregate Lag Domain (ALD) 108
7.2.2 Warped Aggregate Lag Domain (WALD) 110
7.2.3 Real-time Use 113
7.2.4 Future Work 114
7.3 Note Onset Detection 114
7.3.1 Detecting Note Changes using Pitch 116
7.3.2 Back-Tracking 117
7.3.3 Forward-Tracking 118

8 Further Optimisations **119**
8.1 Choosing the Window Size . . . 119
8.2 Incremental SNAC Function Calculation . . . 121
8.3 Complex Moving-Average (CMA) Filter . . . 122
8.4 Incremental WSNAC Calculation . . . 127

9 Vibrato Analysis **129**
9.1 Background . . . 129
9.2 Parameters . . . 130
9.3 Prony Spectral Line Estimation . . . 131
9.3.1 Single Sine Wave Case . . . 132
9.3.2 Estimation Errors . . . 133
9.3.3 Allowing a Vertical Offset . . . 134
9.4 Pitch Smoothing . . . 135

10 Implementation **136**
10.1 Tartini's Algorithm Outline . . . 137
10.1.1 Finding the Frequency and Amplitude of Harmonics . . . 139
10.2 Scales and Tuning . . . 140
10.3 User Interface Design . . . 143
10.3.1 File List Widget . . . 143
10.3.2 Pitch Contour Widget . . . 144
10.3.3 Chromatic Tuner Widget . . . 147
10.3.4 Vibrato Widget . . . 147
10.3.5 Pitch Compass Widget . . . 149
10.3.6 Harmonic Track Widget . . . 150
10.3.7 Musical Score Widget . . . 151
10.3.8 Other Widgets . . . 152

11 Conclusion **154**

A Pitch Conversion Table **163**

B Equal-Loudness Filter Coefficients **167**

C Detailed Results Tables **169**

D Glossary **178**

List of Tables

6.1 Pitch octave errors 88
6.2 With middle/outer ear filtering 92
6.3 Peak Picking Results 98
6.4 Cepstrum octave estimate results 100
6.5 Modified Cepstrum results with changing constant 103
6.6 Modified cepstrum results with changing scalar 104

7.1 Periodic errors for modified cepstrum with median smoothing 107
7.2 Octave estimate errors using combined context 109
7.3 Octave estimate errors using warped context 112
7.4 Octave estimate errors using warped context, with reverberation 114

10.1 Summary of scales 140
10.2 Summary of tuning systems 143

A.1 Pitch conversion table 163

C.1 Experiment 9a results 170
C.2 Experiment 9b results 170
C.3 Experiment 10 results 171
C.4 Experiment 11 results 172
C.5 Experiment 12 results 173
C.6 Experiment 13 results 174
C.7 Experiment 14 results 175
C.8 Experiment 15 & 16 results 176
C.9 Experiment 17 results 177

List of Figures

2.1 Virtual pitch example 8
2.2 Helmholtz resonator 10
2.3 An impulse train diagram 12
2.4 Type-I vs type-II autocorrelation 14
2.5 Autocorrelation example 15
2.6 Hamming window 16
2.7 Centre clipping example 17
2.8 Rectangle window 21
2.9 Sinc function 21
2.10 Frequency plot of common windowing functions 22
2.11 Hanning function 23
2.12 Harmonic Product Spectrum example 25
2.13 Basic model for voiced speech sounds 28
2.14 An example of a male speaker saying the vowel 'A' 28
2.15 Spectrum analysis of a male speaker saying the vowel 'A' 29
2.16 Cepstrum analysis of a male speaker saying the vowel 'A' 30
2.17 Wavelet transform diagram 32

3.1 Sinusoids with different phase 39
3.2 Autocorrelation plot of the sinusoids 39
3.3 Square difference function plot of the sinusoids 40

4.1 Combining two parts of the Hann function 50
4.2 Net result of combined windows 51

5.1 Comparison of autocorrelation-type methods on a sine wave 55
5.2 Comparison of autocorrelation-type methods on a complicated waveform 57
5.3 Constant sine wave vs sine wave frequency ramp 59
5.4 The SNAC function of a changing sine wave 60
5.5 Accuracy of sine wave's during a linear frequency ramp 61
5.6 A 110 Hz constant waveform vs ramp 62
5.7 A 440 Hz constant waveform vs ramp 63
5.8 A 1760 Hz constant waveform vs ramp 63
5.9 Accuracy of complicated waveform's during a linear frequency ramp . . 64
5.10 Accuracy of autocorrelation-type functions during vibrato 66
5.11 Accuracy of autocorrelation-type functions at finding pitch on different vibrato widths 68

5.12 Accuracy of autocorrelation-type functions on vibrato at different window sizes . . . 73
5.13 Amplitude ramp function example . . . 75
5.14 Testing sine waves with linear amplitude ramps . . . 76
5.15 Testing complicated waveforms with linear amplitude ramps . . . 77
5.16 Amplitude step function example . . . 78
5.17 Testing sine waves with an amplitude step function . . . 79
5.18 Testing complicated waveforms with an amplitude step function . . . 81
5.19 Testing accuracy with added white noise . . . 82

6.1 Fundamental frequency vs pitch frequency example . . . 89
6.2 Equal-Loudness Curves . . . 91
6.3 Equal-Loudness Attenuation Filter . . . 92
6.4 SNAC function of a violin segment . . . 94
6.5 SNAC function from strong 2^{nd} harmonic . . . 96
6.6 SNAC function showing primary-peaks . . . 97
6.7 SNAC function example . . . 98
6.8 Log power spectrum of a flute . . . 101
6.9 $\log(1 + sx)$ comparison . . . 104

7.1 Warped vs non-warped Aggregate Lag Domain example . . . 112
7.2 Vibrato drift . . . 116

9.1 Sine wave fitting errors . . . 134

10.1 File List widget . . . 144
10.2 Pitch Contour widget . . . 145
10.3 Chromatic Tuner widget . . . 147
10.4 Vibrato widget . . . 148
10.5 Pitch Compass widget . . . 149
10.6 Harmonic Track widget . . . 150
10.7 Harmonic Track widget with vibrato . . . 151
10.8 Musical Score widget . . . 152

Chapter 1

Introduction

Pitch detection is a fundamental problem in a number of fields, such as speech recognition, Music Information Retrieval (MIR) and automated score writing and has become useful in new areas recently, such as computer games like "SingStar" [Sony Computer Entertainment Europe], and singing tools, such as "Sing and See" [CantOvation Ltd], "Melodyne" [Celemony Software] and Antares "Auto-Tune 5" [Ant, 2007]. It is also used in wave to MIDI converters, such as "Digital Ear" [Epinoisis Software].

Research suggests that real-time visual feedback applied to singing enhances cognitive development and skills learning [Callaghan, Thorpe, and van Doorn, 2004]. The findings of Wilson, Lee, Callaghan, and Thorpe [2007] indicate that a learner singer whose training is a hybrid of traditional teaching methods and real-time visual feedback should make better progress in pitch accuracy than those taught by traditional means only.

This project attempts to extend existing methods of pitch detection for the purpose of accuracy, robustness and responsiveness across a range of musical instruments. It then endeavours to use these pitch methods to make visual feedback tools for musicians. These tools are intended to provide objective information about the aspects of one's playing in order to aid a teacher. That is the system is designed to show people *what* they are playing, and *not how* to play. It is likely that the learning of other musical instruments will benefit in a similar way to that of the singing voice. These tools will form a basis for more general musical pedagogical research in the future.

1.1 Motivation

These days a lot of computer music research is dedicated to making music, or manipulating music to make new music. However, one of the goals of this research is instead of using a computer to try to replace the musician's instruments, let us turn the computer into a tool to help people playing 'real' instruments. These tools should provide instant feedback to aid a musician's learning, and refinement of sound production. There is a kind of beauty in the live performance of real instruments, that people will always be drawn to.

Musicians use primarily what they hear as direct feedback to help them adjust and correct what they are playing. Often what the musicians hear is not always the same as what the audience hears. Singers for example, can hear their own voice through internal vibrations, and violinists who hold their violin close to their ear can hear other close-proximity sounds.

Having an objective 'listener' who can show you important parameters about the sound you produce, is like a navigation system to a pilot; for example, even though a pilot may be able to see the ground, an altimeter is still useful. Moreover, a musician has numerous things to concentrate on at the same time, such as pitch, volume and timing, making it possible for unsatisfactory aspects of their sound to slip by without their noticing.

Often certain parameters can be judged better by one sense than another. When listening to music one can often lose track of the overall volume level of the sound. By looking at a volume meter one can quickly get an accurate reading of this parameter. A similar thing can happen with pitch. For example, a reference pitch is often kept in the musician's head which can sometimes drift. Even though the musical intervals at any moment throughout the song may be correct, the absolute pitch of the notes has changed. This error may only become obvious when playing with others, or with a teacher; however, a pitch tool can enable pupil's to detect pitch drift when practicing at home.

The idea of a visual feedback system is not to stop the user from listening to the sound - as this is still primarily the best source of feedback, but to simply add another sensory channel to aid learning and understanding. A fast visual feedback tool can allow for a new kind of experimentation, where a user can see how variations in playing affect the pitch.

This thesis looks at how to find certain parameters of a sound, and how to display

them in a useful way. This thesis is primarily concerned with the parameter of musical pitch. Several different ways to present the information are investigated. Some existing visual feedback tools, such as guitar tuners, have room for improvement. These devices can often take a significant part of a second to respond, and only show pitch information in a limited way.

1.2 Limit of scope

Music often contains sounds from a collection of instruments, with notes played together at the same time. Although there is research in the area of polyphonic analysis, this work deals with a single note being played at a given time. That is, the sound has only one 'voice'. This problem alone is sufficiently complex and we see plenty of room for the improvement of existing techniques.

We also see single voice analysis as being the most useful in practice, as it directs both computer and user to the point of interest in the sound, thus removing any ambiguities that can arise from multiple voices. However, we are concerned with finding the parameters of a voice with efficiency, precision and certainty, so it can be used in real world environments such as teaching. The focus is primarily on string, woodwind and brass instruments, but other instruments such as the human voice are also handled.

A good tool not only does something well, but does something useful. Depending on experience, people can have different expectations for a tool. It is difficult to prove that a visualisation tool is helpful in learning without a thorough psychological study. Proving the usefulness of these tools is beyond the scope of this thesis. We rely on our own judgement, and discussions with numerous other musicians.

1.3 Contributions

This section summaries the main contributions of this thesis into nine points.

- We have developed a *special normalisation of the autocorrelation* (SNAC) function which improves the accuracy of measuring the short-time periodicity of a signal over existing autocorrelation methods.
- We have extended the SNAC function for use with certain windowing functions. This *windowed SNAC* (WSNAC) function improves the accuracy of measur-

ing short-time periodicity on non-stationary signals over existing autocorrelation methods.

- We have developed a *modified cepstrum* and *peak picking* method which has a high rate of detecting peaks that correspond to the perceived pitch.
- We have developed a method which improves the peak error rate further by utilising the context of notes as a whole. This is called the *warped aggregate lag domain* (WALD) method.
- We have developed a simple technique for using pitch to detect note changes. This is intended to be used in conjunction with other methods.
- We have developed an incremental algorithm for both the SNAC and WSNAC functions, allowing them to efficiently calculate a series of consecutive pitch estimates.
- We have developed an efficient algorithm for a smoothed moving-average filter, called the *complex moving-average* (CMA) filter. This technique allows a large smoothing window to be applied very quickly to almost anything.
- We have developed a method for estimating the musical parameters of vibrato over a short-time, including the vibrato's speed, height, phase and centre offset. Using this method the shape of a vibrato's envelope can be found as it changes throughout the duration of a note.
- We have developed an application called 'Tartini' that implements these techniques into a tool for musicians and made it freely available.

1.4 Thesis overview

Chapter 2 defines what we mean by pitch, and reviews existing algorithms that can be used to detect it. An investigation into the properties of two of the algorithms, the autocorrelation and the square difference function, is detailed in Chapter 3. Chapter 4 develops new variations of the autocorrelation function, called the SNAC and WSNAC functions, and a series of experiments are performed on these in Chapter 5 to test their pitch accuracy. Chapter 6 investigates how to determine, from a small sound segment, the correct musical octave which corresponds to what a person would hear. From this a peak picking algorithm is developed. This peak picking is used in the

SNAC function as well as a newly described modified cepstrum method. The peak picking algorithm is developed further in Chapter 7 by using the context of a whole musical note in its decisions. Chapter 8 describes some optimisations of the algorithms, including a technique for incrementally calculating the SNAC and WSNAC functions, and the more general CMA filter. Chapter 9 shows how to use the pitch information to calculate vibrato parameters. Chapter 10 discusses the implementation of Tartini, the application created during this project, and how the information obtained in previous chapters can be displayed. Numerous widgets are shown which present the data in various ways. Finally, a conclusion is drawn in Chapter 11 which summarises the goals and achievements of this work.

A glossary is provided in Appendix D.

Chapter 2

Background

Pitch is an important parameter of a musical note that a musician has control over. This chapter describes what pitch is, from the discovery of its relationship to frequency, and the first attempts to detect pitch using scientific means. Section 2.2 introduces modern signal processing techniques for pitch detection, covering a range of different pitch algorithms including time domain, frequency domain and other techniques.

2.1 What is Pitch?

Pitch is a perceptive quality that describes the highness or lowness of a sound. It is related to the frequencies contained in the signal. Increasing the frequency causes an increase in perceived pitch.

In this thesis the pitch frequency, F_p, is defined as the frequency of a pure sine wave which has the same perceived pitch as the sound of interest. In comparison, the fundamental frequency, F_0, is defined as the inverse of the pitch period length, P_0, where the pitch period is the smallest repeating unit of a signal. For a harmonic signal this is the lowest frequency in the harmonic series.

The pitch frequency and the fundamental frequency often coincide and are assumed to be the same for most purposes. However, from Chapter 6 these assumptions are lifted, and the differences investigated in order to improve the pitch detection rate.

Today there is a fairly universal standard for choosing a reference pitch when tuning an instrument. The A above middle C is tuned to 440 Hz. Tuning forks are often tuned to this frequency, allowing a musician to listen and tune to it. Moreover, since the introduction of the quartz crystal, electronic oscillators can be made to great accuracy and used for tuning as well. However, using an A of 440 Hz has not always been the

standard. Pipe organs throughout the ages were made with A's tuned from 374 to 567 Hz [Helmholtz, 1912]. Handel's tuning fork from the early 1700's was reported to vibrate at 422.5 Hz and was more or less the standard for two centuries. [Rossing, 1990]. This is the standard for which Haydn, Mozart, Bach and Beethoven composed, meaning that their masterpieces are often played nearly a semitone higher than intended. The pitch has risen over the years with musicians wanting to increase the brightness in the sound.

2.1.1 MIDI note numbering

The Musical Instrument Digital Interface, or MIDI, is an industry-standard protocol which defines a system for numbering the regular notes. In MIDI each semitone on the equal-tempered scale is considered a step of 1 unit. A MIDI note number can be calculated using:

$$n = 69 + 12\log_2(f/440) \tag{2.1}$$

where f is the frequency and n is the resulting MIDI note number. The MIDI note number 69 represents a middle 'A' at 440 Hz. This commonly used system is adopted throughout this thesis. Note that n can take on any real number to represent a pitch between the regular note pitches. A table of conversions between common pitches and their MIDI note numbers is given in Appendix A as a useful reference. Note that other tuning systems also exist and a discusson of these is given in Section 10.2.

2.1.2 Seebeck's Siren

Seebeck first showed that the pitch of a note is related to the frequency of sound when he constructed a siren which controlled the number of puffs of air per second [Helmholtz, 1912]. This was done using a disc that rotated at a known speed, with a number of holes in it evenly spaced just inside its rim. As the disc rotated the holes passed over a pipe blowing air, in a systematic fashion. It was found that the pitch of the sound produced was related only to the number of holes that passed over the pipe in a given time, and not to the size of the holes, or the amount of air pushed through the pipe.

Seebeck's siren was used to show that doubling of the frequency of the sound, created an increase in pitch by one octave. He also found relationships between other intervals on the musical scale. Musical intervals are discussed in detail in Section 10.2 with a range of different tuning methods covered.

Another interesting discovery was that even when the holes had unequal spacing the same pitch resulted. This indicates that some kind of frequency averaging was happening somewhere.

2.1.3 Virtual Pitch

A signal need not have any energy components at the fundamental frequency, F_0. The fundamental frequency can be deduced from the other frequency components provided that they are integer multiples of F_0. This phenomanon is referred to as 'virtual pitch' or 'pitch of the missing fundamental'.

Figure 2.1 shows an example of a virtual pitch. Although there is no energy at 200 Hz, it is still the fundamental frequency because its frequency is deduced from the spacing between the harmonics - as this defines the overall frequency of the repeating waveform. Harmonics are defined as frequencies which are of an integer multiple of the fundamental frequency. Therefore, in this example the fundamental frequency is 200 Hz.

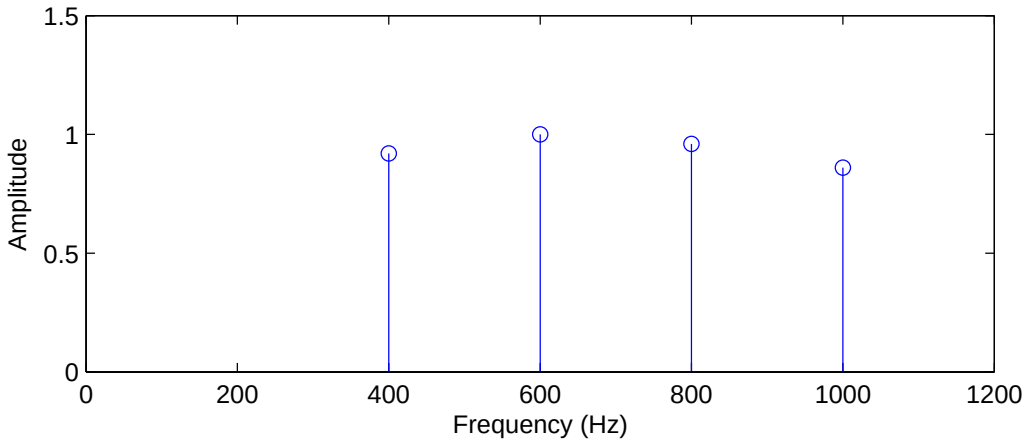


Figure 2.1: An example of a sound with a virtual pitch. The harmonics are all a multiple of 200 Hz, thus giving a fundamental frequency of 200 Hz even though the amplitude of this component is zero.

Plomp [1967] has shown that for complex tones with a fundamental frequency, F_0, below 200 Hz the pitch is mainly determined by the fourth and fifth harmonics. As the fundamental frequency increases, there is a decrease in the number of harmonics that dominate the pitch determination. When F_0 reaches 2500 Hz or above, only the fundamental frequency is used in determining the pitch [Rossing, 1990].

A model using "all-order interspike-interval distribution for neural spike trains" accounts for a wide variety of psychological pitch phenomena [Cedolin and Delgutte, 2005]. These include pitch of the missing fundamental, the pitch shift of in-harmonic tones, pitch ambiguity, the pitch equivalence of stimuli with similar periodicity, the relative phase invariance of pitch, and, to some extent, the dominance of low-frequency harmonics in pitch. These periodicity cues, which are reflected in neural phase locking, *i.e.* the firing of neurons preferentially at a certain phase of an amplitude-modulated stimulus, can be extracted by an autocorrelation-type mechanism, which is mathematically equivalent to an all-order interspike-interval distribution for neural spike trains [Cedolin and Delgutte, 2005]. Autocorrelation is discussed in Section 2.3.2.

2.2 Pitch Detection History

The first attempt actively to detect pitch appears to be that of Helmholtz with his resonators. Helmholtz discovered in the 1860s a way of detecting frequency using a resonator [Helmholtz, 1912]. A resonator is typically made of glass or brass, and spherical in shape, and consists of a narrow neck opening, such as the example in Figure 2.2. This device, like a mass on a spring, has a particular frequency to which it resonates, called the resonant frequency. When a tone consisting of that frequency is played nearby the resonator, it resonates producing a sound reinforcing that frequency. This reinforcement frequency can be heard when the resonator is held near one's ear, allowing even an untrained ear to detect if the given frequency is present or not. The resonant frequency of a resonator, is based on the following formula:

$$f = \frac{v}{2\pi}\sqrt{\frac{a}{Vl}}, \tag{2.2}$$

where a is the area of the neck cross-section, l is the length of the neck, V is the inner volume of the resonator, and v is the speed of sound.

A whole series of resonators could be made, each of a different size, one for each note of the scale. Helmholtz used the resonators to confirm that a complex tone consists of a series of distinct frequencies. These component frequencies are call partial tones, or partials. He was able to show by experiment that some instruments, such as strings produce partial tones which are harmonic, whereas other instruments such as bells and rods, produce in-harmonic partial tones.

On a stringed instrument, such as violin, there are three main parameters that affect pitch [Serway, 1996].

Figure 2.2: A brass Helmholtz resonator, from Max Kohl made around 1890-1900. Photograph by user brian0918, www.wikipedia.org.

1. The length of string, L.
2. The tension force of the string, T.
3. The density, or mass per unit length, of the string, μ.

The fundamental frequency, F_0 is approximately given by:

$$F_0 = \frac{1}{2L}\sqrt{\frac{T}{\mu}} \tag{2.3}$$

Guiseppe Tartini reported that in 1714 he discovered 'terzi suoni', Italian for 'third sounds' [Wood, 1944]. He found that if two notes are played together on a violin with fundamental frequencies of a simple ratio, such as 3:2, then a third note may be heard. The fundamental frequency of this third tone is equal to the difference between the fundamental frequencies of the first two tones. Tartini used this third tone to recognise correct pitch intervals.

To help us understand this, let us consider two notes with fundamental frequencies that are almost the same; for example 440 Hz and 442 Hz. Only a single tone will be heard that beats (in amplitude) at 2 Hz, the difference frequency. As this difference in frequency is increased to about 15 Hz the beating sound turns into a roughness, after which the two distinct tones are heard. Note that this is because the ear can now resolve each of the frequency components separately. After the difference becomes well over 20 Hz what was beating now becomes a third tone, although it may be very weak

relative to the first two tones. The Tartni-tone becomes stronger and even audible when the frequencies of the two notes form a low ratio. For example, a 440 Hz and 660 Hz tone have a ratio of 2:3, giving a Tartini-tone of 220 Hz. Tartini used this scientific principle as a tool to help musicians play correct intervals. Moreover, our software is named after him, as we too strive to make tools for musicians using science.

2.3 Time Domain Pitch Algorithms

Time domain algorithms process the data in its raw form as it is usually read from a sound card - a series of uniformly spaced samples representing the movement of a waveform over time. For example 44100 samples per second is a common recording speed. In this thesis each input sample, x_t, is assumed to be a real number in the range -1 to 1 inclusive, with its value representing the height of the waveform at time t.

This section sumarises time domain pitch algorithms, from simple feature-based methods to autocorrelation, the square difference function and the *average magnitude difference function* (AMDF).

2.3.1 Simple Feature-based Methods

Let us start off with some of the simplest ways of finding pitch, such as the zero-crossing method. In this method, the times at which the signal crosses from negative to positive are stored. Then the difference between consecutive crossings times is used as the period. This simple technique starts to fail once the signal contains harmonics other than the fundamental, as they can cause multiple zero-crossing per cycle. However, if the time between zero-crossings is stored for a number of occurrences, a simple pattern matching approach can be used to find similar groups of crossings. The time between the groups is used as the period. Cooper and Ng [1994] extend this idea to involve other landmark points between the zero-crossings which also need to match well.

Another method is the *parallel processing technique* [Gold and Rabiner, 1969]. Firstly the signal is filtered with a lowpass filter, and then a series of impuse trains are generated in parallel, *i.e.* functions containing a series of spikes which are zero elsewhere. Each impulse train is created using a distinct peak-difference operator. They define six such operators such as 'an impulse equal to the peak amplitude at the location of each peak', and 'an impulse equal to the difference between the peak amplitude and the preceding valley amplitude that occurs at each peak', and others. When an impuse of sufficient amplitude is detected in an impuse train, a constant amplitude level is

held for a short blanking time interval; which then decays exponentially. When a new impulse reaches over this level it is considered as a detection and the process repeats. Note that during the blanking interval no detections are allowed. The time between each detection gives a pitch period estimate for each impuse train. The estimates from all impulse trains are combined to give an overall pitch period estimate. Note that the rate of decay and blanking interval are dependent upon the most recent estimates of pitch period. Figure 2.3 shows an example impulse train and the changing detection level.

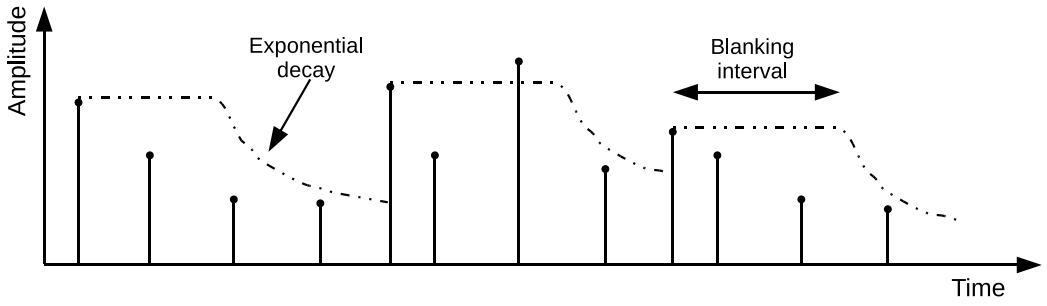


Figure 2.3: An impulse train showing the blanking interval and exponential decay of the detection level.

2.3.2 Autocorrelation

The autocorrelation function (ACF), or just *autocorrelation*, takes an input function, x_t, and cross-correlates it with itself; that is each element is multiplied by a shifted version of x_t, and the results summed to get a single autocorrelation value. The general discrete-time autocorrelation, $r(\tau)$, can be written as

$$r(\tau) = \sum_{j=-\infty}^{\infty} x_j x_{j+\tau}. \tag{2.4}$$

If a signal is periodic with period p, then $x_j = x_{j+p}$, and the autocorrelation will have maxima at multiples of p where the function matches itself *i.e.* at $\tau = kp$, where k is an integer. Note: $r(\tau)$ always has a maximum at $\tau = 0$.

In practice, the short-time autocorrelation function is used when applying it to actual data, as the frequency of a musical note is not typically held steady forever. The short-time autocorrelation function acts only on a short sequence from within the data, called a 'window'. The size of the window is usually kept small so the frequencies

within it are approximately stationary. However, for fast changing frequencies this approximation becomes less accurate. In contrast, the window should be large enough to contain at least two periods of the waveform in order correlate the waveform fully.

There are a two main ways of defining autocorrelation. We will refer to them as type-I and type-II. When not specified we are referring to type-II.

We define the ACF type-I of a discrete window sequence x_t as:

$$r(\tau) = \sum_{j=0}^{W/2-1} x_j x_{j+\tau}, \qquad 0 \leq \tau \leq W/2 \tag{2.5}$$

where $r(\tau)$ is the autocorrelation function of lag τ calculated using the current window of size W. Here the the summation always contains the same number of terms, $W/2$, and has a maximum delay of $\tau = W/2$. The "effective centre", t_c, of the type-I autocorrelation moves with τ as can be seen in Figure 2.4(a). Details of the effective centre are discussed more in Section 3.3.5. The type-I autocorrelation can be used effectively on a section of stationary signal, that is a signal that is not changing over time, but keeps repeating within the window.

We define the ACF type-II as:

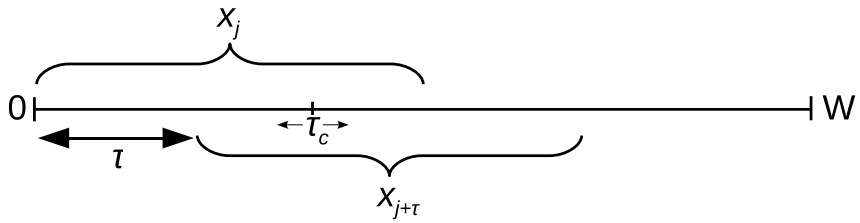
$$r'(\tau) = \sum_{j=0}^{W-1-\tau} x_j x_{j+\tau}, \qquad 0 \leq \tau < W \tag{2.6}$$

In this definition the number of terms in the summation decreases with increasing values of τ. This has a tapering effect on the autocorrelation result, *i.e.* the values tend linearly down toward zero at $\tau = W$. The effective centre, t'_c, of the type-II autocorrelation is stationary, making it more effective to use on non-stationary signals. Figure 2.4(b) shows a diagram to help explain the type-II autocorrelation.

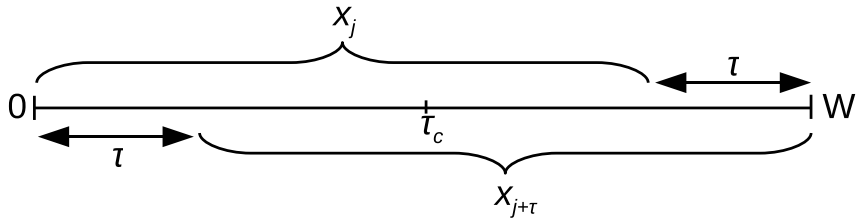
Figure 2.5 shows an example output of the type-I and type-II autocorrelations from a window of data taken from a violin. Note that if an autocorrelation type-II is carried out after first zero-padding the data to double the window size, the result is the same as that from a type-I, for $0 \leq \tau \leq W/2$.

To account for the tapering effect of the type-II autocorrelation, a common solution is to introduce a scaling term. This is called the *unbiased autocorrelation*, and is given by

$$r'_U(\tau) = \frac{W}{W-\tau} \sum_{j=0}^{W-1-\tau} x_j x_{j+\tau}, \qquad 0 \leq \tau < W \tag{2.7}$$



(a) Type-I



(b) Type-II

Figure 2.4: A digram showing the difference between the type-I and type-II autocorrelation.

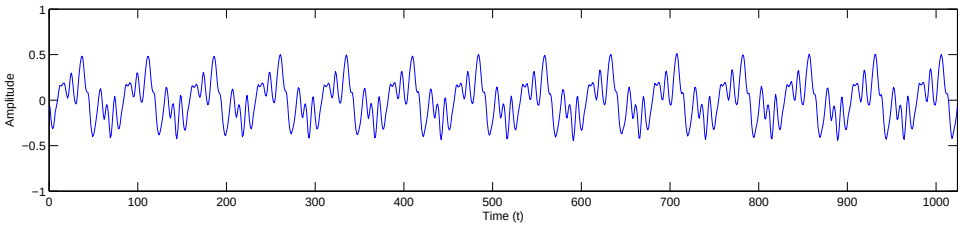
Large instabilities can appear as τ approaches W, so it is common to only use the values of τ up to $W/2$. The function $r'(\tau)$ attains a maximum at $\tau = 0$ which is proportional to the total energy contained in the data window. This zero-lag value can be used to normalise the autocorrelation *i.e.* divide all values by $r'(0)$, so they are in the range -1 to 1.

Windowing functions can be applied to the data before autocorrelation, concentrating stronger weightings at the centre of the window, reducing the "edge effects" caused by sudden starting and stopping of correlation at either end. This allows for a smoother transition as the window is moved. The Hamming window function is often used. The Hamming window coefficients are computed using

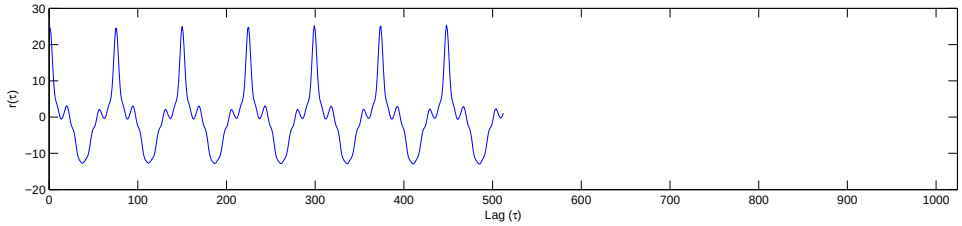
$$w_n = 0.53836 - 0.46164 \cos(\frac{2\pi n}{W-1}), \qquad 0 \leq n < W, \tag{2.8}$$

with the function being zero outside the range. Figure 2.6 shows an example Hamming window with $W = 1000$. To apply a windowing function, it is multiplied element-wise by the data window, shaping the data's ends closer toward zero. The *windowed autocorrelation*, r'_{Win}, is therefore defined as:

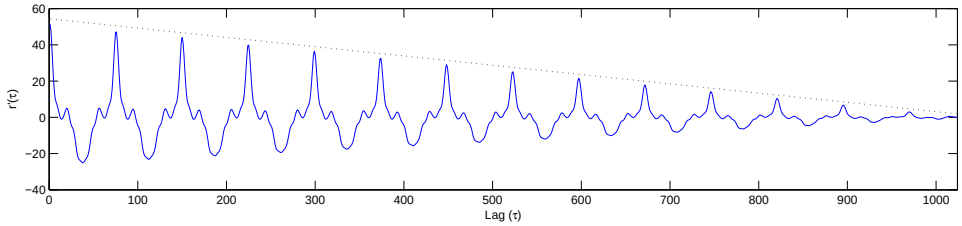
$$r'_{\text{Win}}(\tau) = \sum_{j=0}^{W-1-\tau} w_j x_j w_{j+\tau} x_{j+\tau}. \tag{2.9}$$



(a) A window of size $W = 1024$ taken from a violin recording



(b) An autocorrelation of type-I



(c) An autocorrelation of type-II

Figure 2.5: A piece of violin data, (a), and two types of autocorrelation performed on it, (b) and (c). Type-II in (c) has a linear tapering effect.

Note that there should be the same number of windowing function coefficients, w, as the window size, W. The equation is generally more useful in the unbiased form. The *windowed unbiased autocorrelation*, $r'_{\text{WinUnbiased}}$, is defined as:

$$r'_{\text{WinUnbiased}}(\tau) = \frac{W}{(W-\tau)} \sum_{j=0}^{W-1-\tau} w_j x_j w_{j+\tau} x_{j+\tau}. \tag{2.10}$$

This can be also be normalised by dividing the result by $r'_{Win}(0)$.

Once the autocorrelation has been performed the index of the maximum is found and used as the fundamental period estimate (in samples). This is then divided into the sampling rate to get a fundamental frequency value of the pitch. In the lag domain there are often lots of maxima due to the good correlation at two periods, three periods and so on. These maxima can be very similar in height, making it possible for an incorrect maxima to be selected. The unbiasing of the type-II autocorrelation is often left out

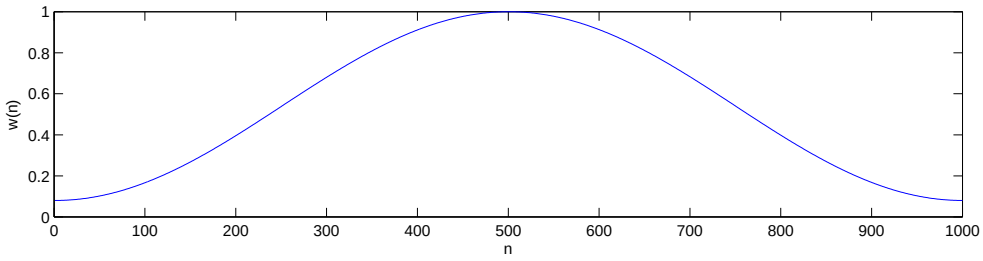


Figure 2.6: A Hamming window of size 1000, often used as a windowing function applied during autocorrelation.

when choosing the maximum, to allow the natural linear decay to give preference to the earlier periods, as with simple sounds, it is often the first maxima that is wanted. However, choosing the correct peak is considered a bit of a black art, and is discussed in more detail in Section 6.3.

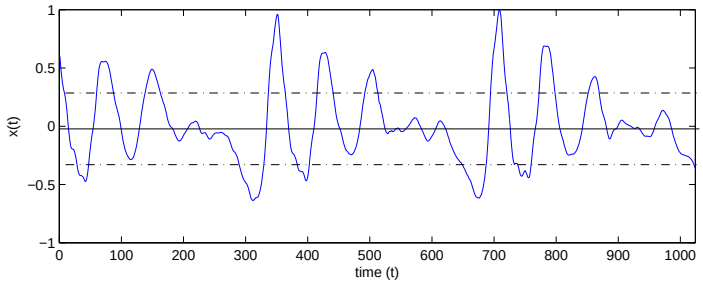
Autocorrelation is good for detecting perfectly periodic segments within a signal, however, real instruments and voices do not create perfectly periodic signals. There are usually fluctuations of some sort, such as frequency or amplitude variations; for example, during vibrato or tremolo. Section 3.3 investigates how these variations affect the autocorrelation. For more information on autocorrelation see Rabiner [1977].

Centre clipping

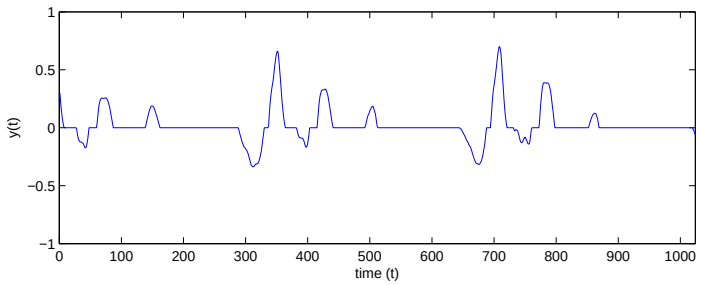
Consistently picking the correct peak from an autocorrelation can be difficult. In speech recognition it is common to pre-process the signal by "flattening" the spectrum. The objective here is to minimise the effects of the vocal tract transfer function, bringing each harmonic closer to the same level, and hence enhancing the vocal source. One method of flattening is *centre clipping* [Sondhi, 1968]. This involves a non-linear transformation to remove the centre of the signal, leaving the major peaks. First a clipping level, C_L, is defined as a fixed percentage of the maximum amplitude of the signal window. Then the following function is applied:

$$y(t) = \begin{cases} 0 & \text{for } |x(t)| \leq C_L \\ x(t) - C_L & \text{for } x(t) > C_L \\ x(t) + C_L & \text{for } x(t) < -C_L. \end{cases} \tag{2.11}$$

Figure 2.7 gives an example of centre clipping using a fixed percentage of 30%, the value used by Sondhi [1968]. The resulting autocorrelation contains considerably fewer extraneous peaks, reducing the confusion when choosing the peak of interest.



(a) The original segment of speech



(b) The centre clipped speech

Figure 2.7: An example of centre clipping on a speech waveform.

Another variation on this theme is the 3 level centre clipper, for use on more restricted hardware, in which the following function is used:

$$y(t) = \begin{cases} 0 & \text{for } |x(t)| \leq C_L \\ 1 & \text{for } x(t) > C_L \\ -1 & \text{for } x(t) < -C_L. \end{cases} \tag{2.12}$$

Here the output is -1, 0, or 1, allowing for a greatly simplified autocorrelation calculation in hardware. This method is purely for computational efficiency and can degrade the result a little from the above method.

2.3.3 Square Difference Function (SDF)

The idea for using the *square difference function*[1] (SDF) is that if a signal is pseudo-periodic then any two adjacent periods of the waveform are similar in shape. So if the waveform is shifted by one period and compared to its original self, then most of the peaks and troughs will line up well. If one simply takes the differences from one waveform to the other and then sums them up, the result is not useful, as some values

[1] also called the *average square difference function* (ASDF) when divided by the number of terms

are positive and some negative, tending to cancel each other out. This could be dealt with by using the absolute value of the difference, as discussed in Section 2.3.4, however it is more common to sum the square of the differences, where each term contributes a non negative amount to the total. When the waveform is shifted by an amount, τ, that is not the period the differences will become greater, and cause an increased sum. Whereas, when τ equals the period it will tend to a minimum.

Analogous to the autocorrelation in section 2.3.2, we define two types of discrete-signal square difference functions. The SDF of type-I is defined as:

$$d(\tau) = \sum_{j=0}^{W/2-1} (x_j - x_{j+\tau})^2, \qquad 0 \leq \tau \leq W/2, \tag{2.13}$$

and the SDF type-II is defined as:

$$d'(\tau) = \sum_{j=0}^{W-1-\tau} (x_j - x_{j+\tau})^2, \qquad 0 \leq \tau < W. \tag{2.14}$$

Like the type-II ACF, the type-II SDF has a decreasing number of summation terms as τ increases. In both types of SDF, minima occur when τ is a multiple of the period, whereas in the ACFs maxima occurred. However, the exact position of these maxima and minima do not always coincide. These differences are discussed in detail in Section 3.3.

When using the type-II SDF it is common to divide $d'(\tau)$ by the number of terms, $W - \tau$, as a method of counteracting the tapering effect. However, this can introduce artifacts, such as sudden jumps when large changes in the waveform pass out the edge of the window. A new normalisation method is introduced in section 4.1 that provides a more stable correlation function which works well even with a window containing just two periods of a waveform.

For more information on the SDF see de Cheveign and Kawahara [2002] and Jacovitti and Scarano [1993].

2.3.4 Average Magnitude Difference Function (AMDF)

The *average magnitude difference function* (AMDF) is similar to both the autocorrelation and the square difference function. The AMDF is defined as

$$\gamma(\tau) = \frac{1}{W} \sum_{j=0}^{W/2-1} |x_j - x_{j+\tau}|, \quad 0 \leq \tau \leq W/2, \tag{2.15}$$

although often the $\frac{1}{W}$ is left out if all you are interested in is the index of the minimum. The motivation here is that no multiplications are used, so this measure of the degree to which data is periodic is well suited to special purpose hardware. However, it still acts similarly to the SDF, producing zeros at multiples of the period for a periodic signal, and non-zero otherwise. Here the larger differences are not penalised as harshly as in the SDF. For more information on AMDF see Ross, Shaffer, Cohen, Freudberg, and Manley [1974].

2.4 Frequency Domain Pitch Algorithms

Frequency domain algorithms do not investigate properties of the raw signal directly, but instead first pre-process the raw, time domain data, transforming it into the frequency space. This is done using the Fourier transform. The following starts with a mathematical definition of the Fourier transform, and develops it into a form useful for signal processing.

The continuous Fourier transform is defined as

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-2\pi i f t} dt, \qquad f, t \in \mathbb{R}, \tag{2.16}$$

where f is the frequency. The capital X denotes the Fourier transform of x. The Fourier transform breaks a function, $x(t)$, up into its sinusoidal components, *i.e.* the amplitude and phase of sine waves on a continuous frequency axis that contribute to the function.

In practice, we do not always have a continuous function, we have a sequence of measurements from this function called samples. These samples are denoted x_t. A discrete version of the Fourier transform (DFT) can be used in this case. This can be written as

$$X_f = \sum_{t=-\infty}^{\infty} x_t e^{-2\pi i f t}, \qquad f, t \in \mathbb{Z}. \tag{2.17}$$

Here the function need only be sampled at discrete regular steps. For example, the input from a sound card/microphone which takes measurements of air pressure every $1/44100^{th}$ of a second is sufficient.

In practice, a discrete Fourier transform cannot be done on an infinite sequence of samples. However, if a sequence is periodic, such that it repeats after fixed number of samples, W, *i.e.*

$$x_t = x_{t+W}, \qquad -\infty < t < \infty \tag{2.18}$$

then the whole infinite sequence can be reproduced from a single period. A DFT can be performed on this single period, with a window of size W to produce an exact representation of the periodic sequence. This is done as follows:

$$X_f = \sum_{t=0}^{W-1} x_t e^{-2\pi i \frac{ft}{W}}. \tag{2.19}$$

X_f are referred to as the Fourier coefficients, and are in general complex for real sequences. An efficient calculation of the DFT called the *fast Fourier transform*, or FFT [Cooley and Tukey, 1965], made the use of the DFT popular for signal processing, as it reduced the computation complexity from $O(W^2)$ to $O(W \log(W))$.

The original sequence can be reproduced from the Fourier coefficients using the *inverse DFT*:

$$x_t = \frac{1}{W} \sum_{f=0}^{W-1} X_f e^{2\pi i \frac{ft}{W}}. \tag{2.20}$$

Note that the DFT and the inverse DFT are almost the same, a useful property.

Let us instead consider a sequence, x_t, of finite length that is zero outside the interval $0 \leq t < W$. This is done by multiplying the sequence by a rectangular windowing function

$$\text{rectangle}(x) = \begin{cases} 1 & \text{for } 0 \leq x < W \\ 0 & \text{otherwise,} \end{cases} \tag{2.21}$$

as shown in figure 2.8 with window size $W = 1000$. This sequence now has the property of being localised in time. This small contiguous section of the data is called a window, and has a given size, W, which is the number of samples it contains in its sequence. The window refers to data being used in a given computation. The window is a view of a given segment of data and in practice will be moved across the data with a given *hop size*[2]. The hop size is the number of samples a window is moved to the right between consecutive frames. Frames are indexes that refer to window positions, and usually follow something of the form $window\_position = frame * hop\_size$.

If a window is taken and interpreted as if it is a periodic sequence, as in Equation 2.18, then Equation 2.19 can be applied to determine the Fourier coefficients. The coefficients contain enough information to reproduce the periodic sequence. Taking only the interval $0 \leq t < W$ and treating the rest as zeros, we have recovered our finite sequence back. This shows that a sequence of length W can be exactly represented by

[2]Also called a jump or step size

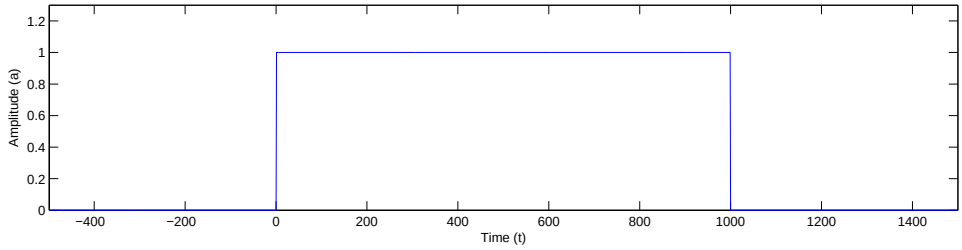


Figure 2.8: A rectangle window of size 1000. This is multiplied by a sequence to zero out everything outside the window.

its Fourier coefficients. But it cannot be used to describe the original signal outside this window.

The Fourier coefficients tell us the amplitude and phase of each sinusoid to add together to reproduce the finite sequence. However, these coefficients are not a true representation of the spectrum because of the rectangular window applied to the sequence. The coefficients represent the spectrum convolved with a *sinc function*. Note that

$$\text{sinc}(x) = \begin{cases} 1 & \text{for } x = 0 \\ \sin(\pi x)/x & \text{otherwise.} \end{cases} \tag{2.22}$$

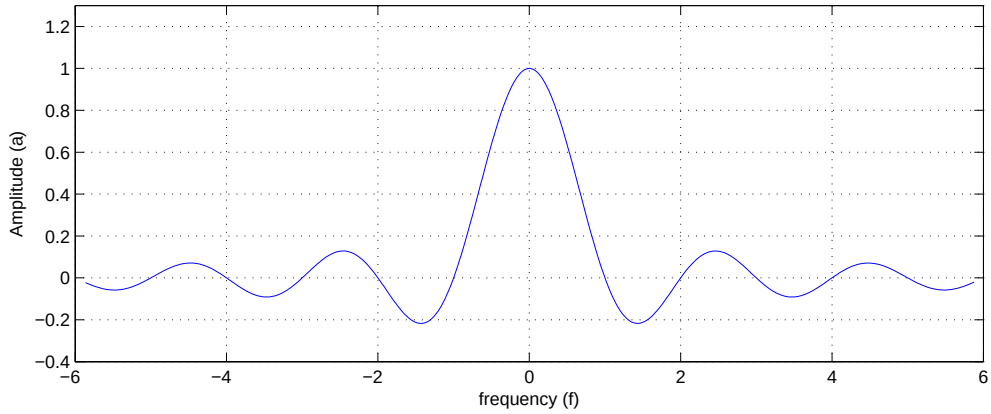


Figure 2.9: The sinc function.

Figure 2.9 shows the sinc function, which is the Fourier transform of a rectangular window. Notice that all integer positions of the sinc function have a value of 0, except at $f = 0$ where it has a value of 1. This means that if the sequence contains a sine wave that has a whole number of periods in the window, it is represented by only one Fourier coefficient. Sine waves without a whole number of periods get scattered or

leaked across a range of coefficients, for example, sinc positions ... -2.5, -1.5, -0.5, 0.5, 1.5, 2.5 ... for a sine wave with 6.5 periods in the window. This leaking of spectral coefficients into their neighbours is often referred to as "edge effects", as it is caused by the hard cutoff shape of the rectangle window function. Moreover, the Fourier coefficients are also referred to as frequency 'bins', as any nearby frequencies also fall into the coefficient's value.

Many different windowing functions have been devised to help smooth this hard edge, with the aim of producing a Fourier transform with less spectral leakage. The spectral leakage is often measured in terms of the main-lobe width and side-lobe attenuation. Harris [1978] discusses the properties of many different windowing functions in some depth, and their effect on spectral leakage. Figure 2.10 shows a normalised frequency plot of the Fourier transforms of some common windowing functions. Note that the sinc function becomes the blue line in the logarithmic plot. It can be seen that the rectangular window has a narrow main-lobe which is good, but poor side-lobe attenuation which makes it difficult to separate real peaks from side-lobe peaks in the spectral domain. The other two spectral curves shown, the Hamming and Hanning window functions, have wider main-lobes but better side-lobe attenuation.

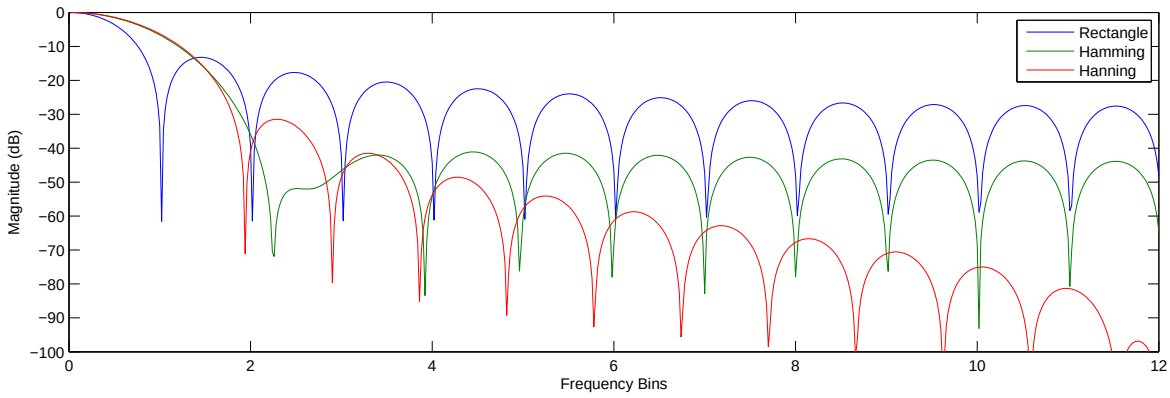


Figure 2.10: A normalised frequency plot of some common windowing functions.

The Hanning window function, shown in Figure 2.11, is common in analysis-synthesis methods, as it reduces the height of the side lobes significantly while still allowing for easy reconstruction of the original signal. Reconstruction is done by using a number of 50% overlapping analysis windows, *i.e.* a W/2 hop size. Note that applying a windowing function to the sequence and then performing a DFT is also referred to as a *short-time Fourier transform* (STFT).

The STFT generates a spectrum estimate, but, in order to determine exactly what

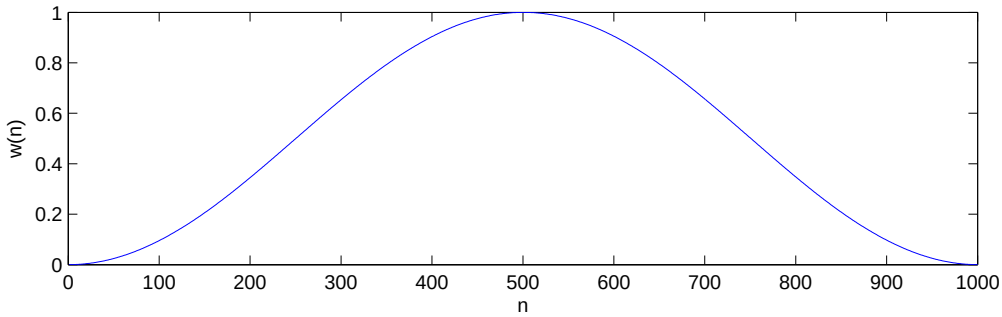


Figure 2.11: The Hanning function with a window size of W = 1000.

the frequency components of interest are, further processing is required. Rodet [1997] contains a good summary of techniques which can be used to obtain spectral components of interest. The following discusses some spectrum peak methods and how a pitch frequency can be drawn from this.

2.4.1 Spectrum Peak Methods

Because the DFT produces coefficients for integer multiples of a given base frequency, any frequencies which fall in between these require further processing to find. The following discusses a variety of methods that have been used to estimate the spectrum peaks and combine them together to determine the pitch frequency. Note that a single spectral component is referred to as a partial.

McLeod and Wyvill [2003] use the property that the Fourier transform of a Gaussian-like window gives a Gaussian-like spectrum, and that the logarithm of a Gaussian is a parabola. Thus, if a STFT is performed using a Gaussian-like window and the logarithm of the coefficients is taken, a parabola can be fitted using 3 values about a local maximum coefficient to yield its centre. This centre represents the frequency and amplitude of the partial in the original signal. However, if multiple partials are close together they can interfere to reduce the validity of the assumptions made. Once the frequency and amplitude of all the peaks are found, the frequency, f, with the highest peak amplitude is assumed to be a multiple of the pitch frequency, and a voting scheme is used to determine which harmonic, h, it is. Each possible choice of harmonic gains votes based on the distances between peaks. If the distance from one peak to another is close to f/h the votes for h being the correct harmonic increase by the square of the constituent amplitudes (in decibels, dB). Note that all combinations of peaks are

compared.

Parabolic fitting techniques are also discussed in Keiler and Marchand [2002] along with the related *triangle algorithm* [Althoff, Keiler, and Zölzer, 1999], in which the windowing function is chosen so that a triangular shape is produced in the Fourier domain. Thus making the interpolation simpler.

Dziubiński and Kostek [2004] employ an *artificial neural network* (ANN) to find the frequency of the highest peak, with the same 3 neighbourhood coefficients used as input. The system was trained with synthetic data. A complicated weighting of energy components is used to determine which harmonic the peak represents.

A similar method by Mitre, Queiroz, and Faria [2006] uses what they call a state-of-the-art sinusoidal estimator. This includes a fundamental frequency refinement stage which considers the frequency estimates of all partials in the harmonic series. All these partials are used to generate a weighted average of each local fundamental estimate.

2.4.2 Phase Vocoder

The basic idea of *phase vocoder* methods, discussed in Keiler and Marchand [2002] and Götzen, Bernardini, and Arfib [2000], is to use the change in phase of frequency bins between two successive STFT's to find the partials accurately. Only the local maxima in the STFT coefficients are of interest. The expected phase advance over the given hop size is known for a given frequency bin. The measured phase difference from this expected value can be used to determine the offset from the frequency bin, and hence the frequency of the partial.

Signal derivatives are used to improve the precision in Marchand [1998, 2001]. This takes advantage of the property that a sine wave's derivative is another sine wave of the same frequency, although its phase and amplitude may differ.

2.4.3 Harmonic Product Spectrum

The *harmonic product spectrum* (HPS) is a method for choosing which peak in the frequency domain represents the fundamental frequency. The basic idea is that if the input signal contains harmonic components then it should form peaks in the frequency domain positioned along integer multiples of the fundamental frequency. Hence if the signal is compressed by an integer factor i, then the i^{th} harmonic will align with the fundamental frequency of the original signal.

The HPS involves three steps: calculating the spectrum, downsampling and mul-

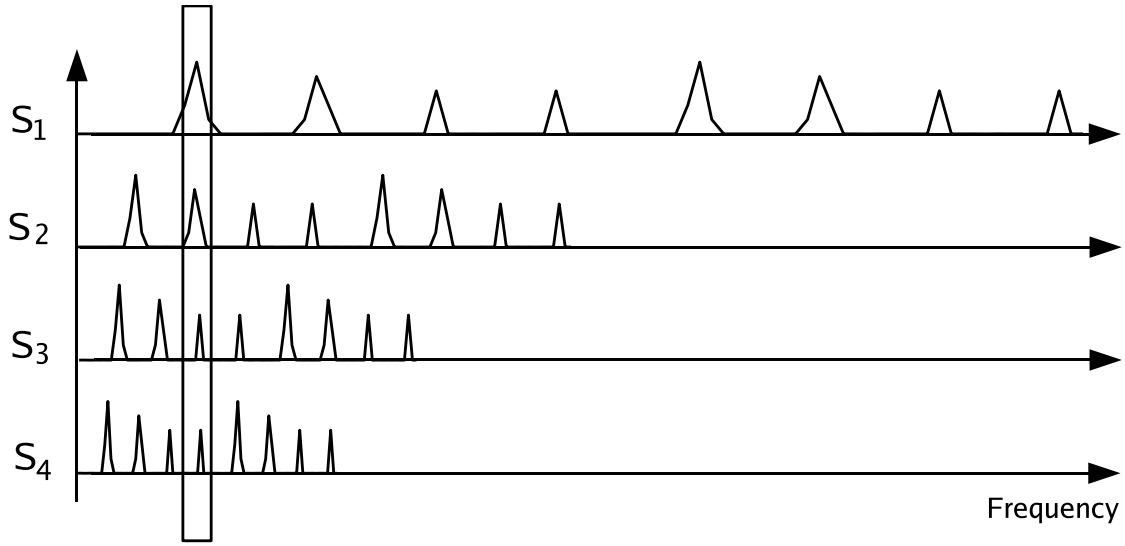


Figure 2.12: An example frequency spectrum, S_1, being downsampled by a factor 2, 3 and 4 for HPS. A rectangle highlights the peaks aligned with the fundamental.

tiplication. The frequency spectrum, S_1, is calculated using the STFT. S_1 is then downsampled by a factor of two using re-sampling to give S_2, *i.e.* resulting in a frequency domain that is compressed to half its length. The second harmonic peak in S_2 should now align with the first harmonic peak in S_1. Similarly, S_3 is created by downsampling S_1 by a factor of three, in which the third harmonic peak should align with the first harmonic peak in S_1. This patten continues with S_i being equal to S_1 downsampled by a factor of i, with i ranging up to the number of desired harmonics to compare. Figure 2.12 shows an example of a function and its compressed versions. The resulting spectra are multiplied together and should result in a maximum peak which corresponds to the fundamental frequency.

One of the limitations of HPS is that it does not perform well with small input windows, *i.e.* a window containing only two or three periods. Hence, it is limited by the resolution in the frequency domain because peaks can get lost in the graininess of the discrete frequency bins. Increasing the length of STFT, so that the peaks can be kept separated improves the result, at the cost of losing time resolution.

2.4.4 Subharmonic-to-Harmonic Ratio

Sun [2000, 2002] discuss a method for choosing the pitch's correct octave, in speech, based on a ratio between subharmonics and harmonics. The *sum of harmonic amplitude*

is defined as:

$$SH = \sum_{n=1}^{N} X(nF_0) \tag{2.23}$$

where N is the maximum number of harmonics to be considered, and $X(f)$ denotes the interpolated STFT coefficients, and F_0 denotes a proposed fundamental frequency. The *sum of subharmonic amplitude* is defined as:

$$SS = \sum_{n=1}^{N} X((n - \frac{1}{2})F_0). \tag{2.24}$$

The *subharmonic-to-harmonic ratio* (SHR) is obtained using:

$$SHR = \frac{SS}{SH}. \tag{2.25}$$

As the ratio increases above a certain threshold, the resultant pitch is chosen to be one octave lower than the frequency defined by F_0. However, the algorithm is modified to use a logarithmic frequency axis, as this causes higher harmonics to become closer and eventually merge together. This effect is similar to the way critical bands within the human auditory system have a larger bandwidth at higher frequencies, and hence, the human inability to resolve individual harmonics at higher frequencies. The linear to logarithmic transformation of the frequency axis, described in Hermes [1988], is performed using cubic-spline interpolation of the spectrum. They found that using 48 points per octave was sufficient to prevent any undersampling of peaks (on a 256-point FFT) for most speech-processing purposes.

2.4.5 Autocorrelation via FFT

Although the autocorrelation method is considered a time domain method (Section 2.3.2), frequency domain transformations can be used for computational efficiency. Using properties from the Wiener-Khinchin theorem [Weisstein, 2006b], the computation of large autocorrelations can be reduced significantly using the FFT method [Rabiner and Schafer, 1978]. An FFT is first carried out on the window to obtain the power spectral density and then the autocorrelation is computed as the inverse FFT of the power spectral density. To avoid aliasing caused by the terms wrapping around one end and into the other, the data is zero-padded by the number of output terms desired, p. In our case p is usually $W/2$. The basic algorithm given a window size, W, is as follows:

- Zero-pad the data with p zeros, *i.e.* put p zero's on the end of the array.

- Compute a FFT of size $W + p$.
- Calculate the squared magnitude of each complex term (giving the power spectral density).
- Compute an inverse FFT to obtain the autocorrelation (type-II).

Note that the result is the same as that from Equation 2.6.

2.5 Other Pitch algorithms

This section discusses other pitch algorithms which are not easily categorised into time or frequency methods. These include the cepstrum, wavelet, and *linear predictive coding* (LPC) methods.

2.5.1 Cepstrum

The 'cepstrum' is defined as the power spectrum of the logarithm of the power spectrum of a signal. Originally the idea came from Bogert, Healy, and Tukey [1963] when analysing banding in spectrograms of seismic signals. The concept was quickly taken up by the speech processing community where Noll [1967] first used the cepstrum for pitch determination. They found short-term cepstrum analysis was required, *i.e.* repeatedly doing cepstrum analysis on small segments of data, typically around 40ms, to cope with the changes in the speech signal.

The basic idea of the cepstrum is that a voiced signal, $x(t)$, is produced from a source signal, $s(t)$, that has been convolved with a filter with an impulse response $h(t)$. This can be shown as:

$$x(t) = s(t) * h(t), \tag{2.26}$$

where $*$ denotes convolution. Figure 2.13 shows this basic model for how voiced sounds are produced in speech. Here the source signal, $s(t)$, is generated by the vocal chords. The vocal tract acts as a filter, $h(t)$, on this signal, and the measured signal, $x(t)$, is the recorded sound. Given only $x(t)$, the original $h(t)$ and $s(t)$ can be found, given that their log spectrum has a curve which varies at different known rates. The separation is achieved through linear filtering of the inverse Fourier transform of the log spectrum of the signal, *i.e.* the cepstrum of the signal. This idea is shown in the following.

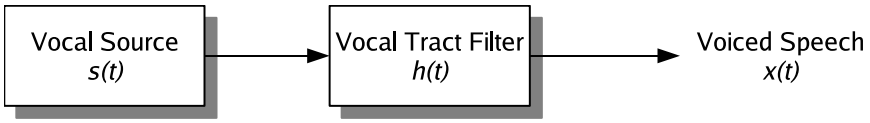
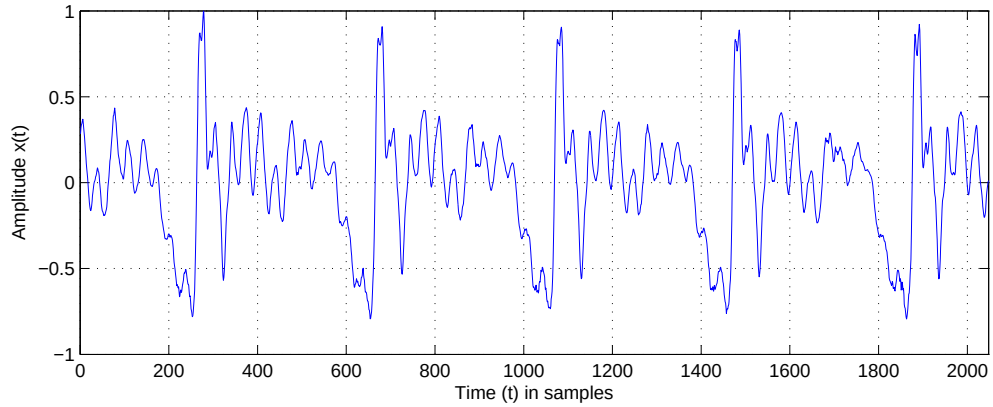
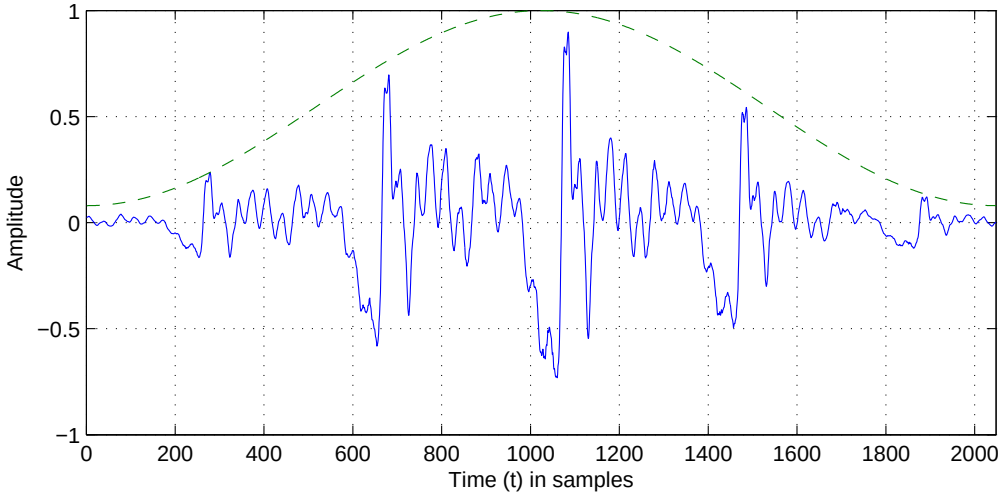


Figure 2.13: Basic model for how voiced speech sounds are produced.

Figure 2.14(a) shows part of the waveform from a male speaker saying the vowel 'A'. A Hamming window is applied in order to reduce spectral leakage: Figure 2.14(b). Then the FFT of $x(t)$ is calculated, giving $X(f)$.



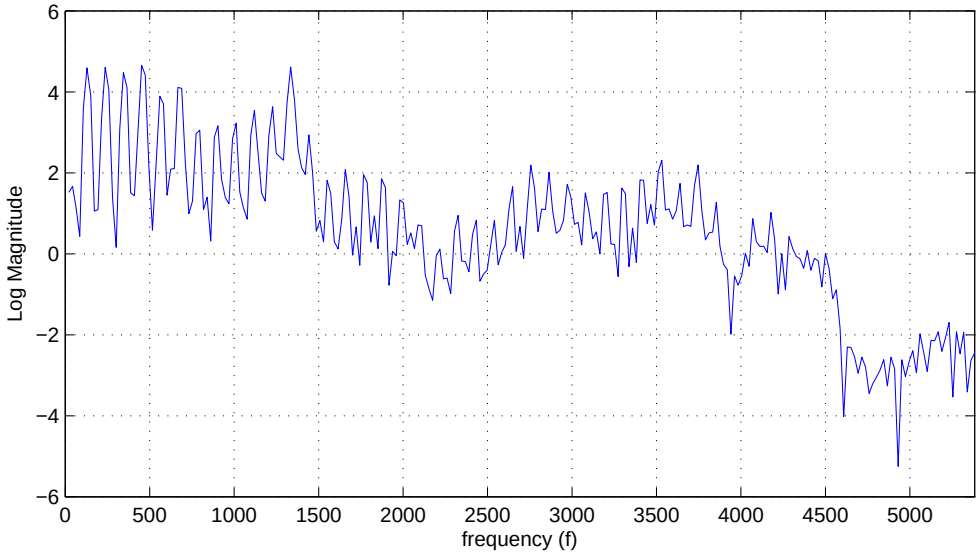
(a) The original waveform.



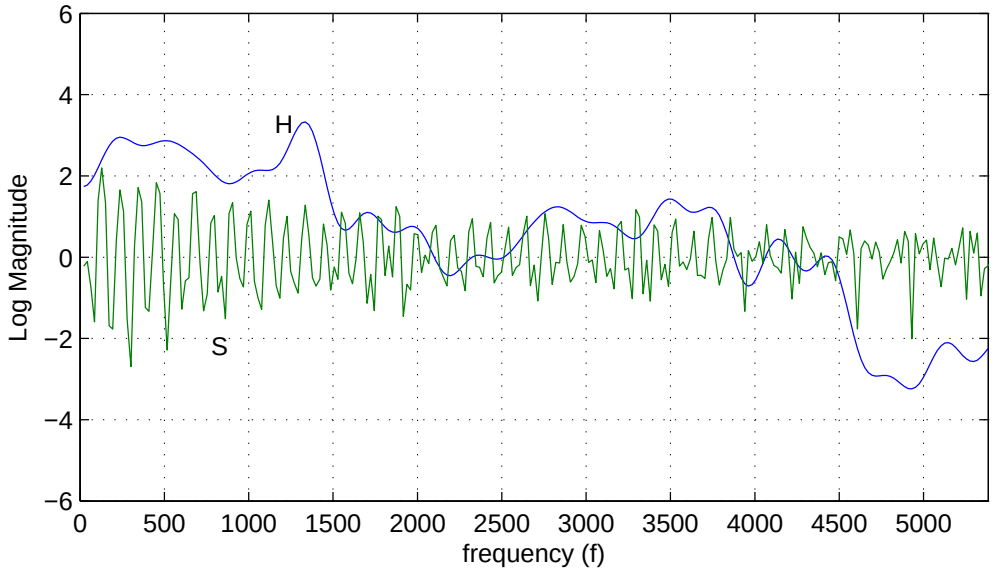
(b) With a Hamming window applied, as indicated by the dotted lines.

Figure 2.14: An example of a male speaker saying the vowel 'A'

Using the properties of the *convolution theorem* [Weisstein, 2006a], a convolution



(a) The spectrogram of the signal, *i.e.* the log magnitude of $X(f)$.



(b) The components of vocal source, S, and vocal tract filter, H, which add to gether to give the spectrogram above.

Figure 2.15: Spectrum analysis of a male speaker saying the vowel 'A'

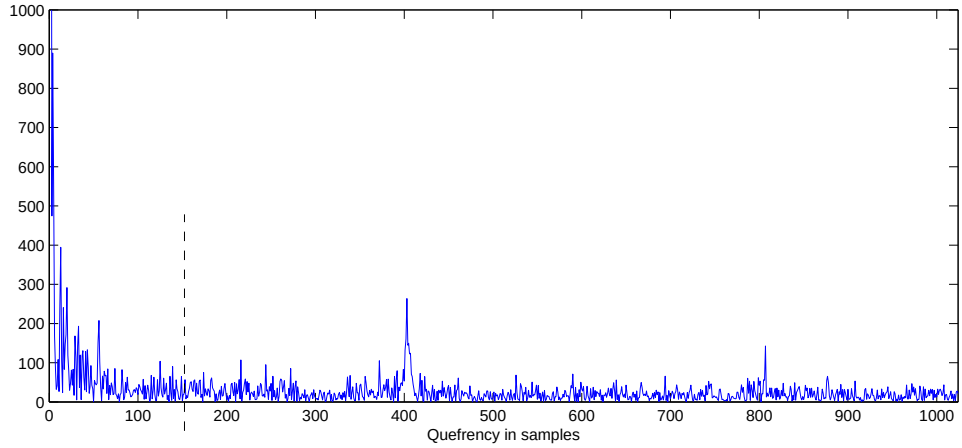


Figure 2.16: The cepstrum, *i.e.* an FFT of the spectrogram, of a male speaker saying the vowel 'A'. The peak around 400 indicates the fundamental period, and the dotted line at 150 indicates the split value used to generate S and H in Figure 2.15(b).

in the time domain is equivalent to a multiplication in the frequency domain, so

$$X(f) = S(f)H(f), \tag{2.27}$$

where $S(f)$ and $H(f)$ indicate the Fourier transforms of $s(t)$ and $h(t)$ respectively. Taking the logarithm of both sides of Equation 2.27 gives

$$\begin{aligned} \log(X(f)) &= \log(S(f)H(f)) & (2.28) \\ &= \log(S(f)) + \log(H(f)). & (2.29) \end{aligned}$$

Note that the operation which combines the terms, has been reduced from a convolution, in Equation 2.26, to an addition. Figure 2.15(a) shows the spectrum, $\log(X(f))$, of our example. It can be seen that there are a lot of small peaks closely spaced along the spectrum. These frequencies are the harmonics that occur at regular intervals of the fundamental frequency. These peaks are produced primarily from the vocal source, $s(t)$. However, the overall shape of the spectrogram is governed by the vocal tract filter, $h(t)$. These spectrogram curves vary at different rates, so we can can take an FFT of the spectrogram to analyse this. The FFT of the spectrogram is referred to as the cepstrum, and has the units of 'quefrency'. The low quefrency values, or slow varying components of the spectrogram's shape are assumed to correspond to $H(f)$, and high quefrency values to $S(f)$. Figure 2.16 shows our example in the cepstrum domain, in which a quefrency-split value can be set, at which our high/low pass filters

cross. This can be done by zeroing out the values to the left and right of the split respectively. Figure 2.15(b) used a split value of 150, and shows the spectrograms of the two components, each generated using an inverse FFT.

The fundamental period of the original waveform corresponds to the index of the maximum in the cepstrum domain that lies after the split value. In our example the maximum has a quefrency of 403 samples. At the sample rate of 44100 Hz, this gives a fundamental frequency of 109.4 Hz.

One of the problems with using the cepstrum is deciding where the split value should be placed. It should always be before the fundamental period, but if you do not know where the fundamental period is this becomes one of those chicken and egg games - to choose the split point you need the period, but to choose the period you need the split point.

In general the cepstrum requires a complex-logarithm and a complex-FFT. However, for pitch detection purposes we are not interested in the phase information, so only the magnitude is required. Often the power spectrum, $|X(f)|^2$, is used directly saving a square root operation as follows,

$$\log(|X(f)|^2) = 2\log(|X(f)|) = 2(\log(|S(f)|) + \log(|H(f)|)), \tag{2.30}$$

as this just produces a scaling effect, which does not affect the positions of maxima.

2.5.2 Wavelets

Unlike the Fourier transform which has a fixed bandwith and time resolution, the wavelet transform changes its time resolution at different frequencies, having an increasing filter bandwidth at higher frequencies. The result is series of *constant 'Q'* filters, in which the bandwidth of each filter is proportional to its centre frequency.

A 'mother wavelet' is a like a template function from which scaled and translated 'daughter wavelets' are made. A wavelet basis that maintains the orthogonal property allows for perfect reconstruction of the original signal from data the same length. A detailed introduction to wavelets can be found in Vetterli and Herley [1992].

Wavelet transforms have some advantages over the Fourier transform in such things as audio and image compression [Joint Photographic Experts Group, 2000], as the multi-resolution nature of it maps well to certain perceptual properties.

The discrete wavelet transform can be calculated by repeatedly applying an M-channel filter bank on its lowpass output [Nguyen, 1995]. Decimation occurs after each

step to remove redundant information. Often a two-channel filter is used with a 50% lowpass/highpass combination.

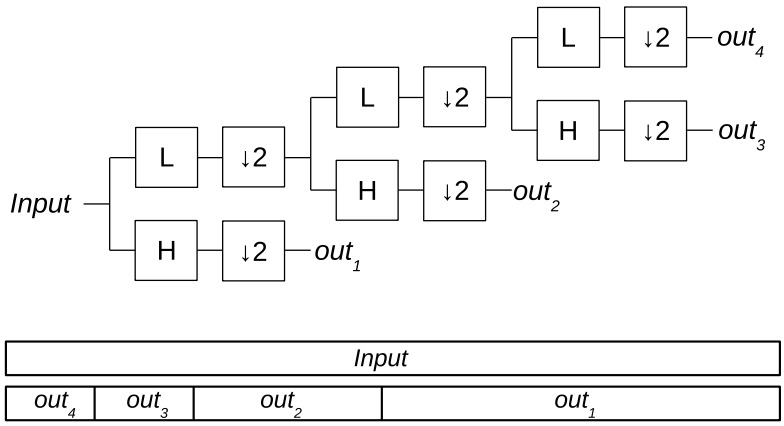


Figure 2.17: A discrete wavelet transform which recursively uses a two-channel filter bank. The relative sizes of the output blocks compared with the input data block are shown at the bottom.

Figure 2.17 shows a diagram of how a two-channel filter bank can be used recursively to perform a wavelet transform. L and H represent a low and high pass filter respectively, with $\downarrow 2$ indicating a decimation by a factor of 2, *i.e.* the removal of every second output value. Note that the highpass filter is typically a *quadrature mirror filter* (QMF) of the lowpass filter. Notice that out_1 contains frequencies from the top half of the frequency spectrum and is half the size of the input, whereas out_2 contains only a quarter of the frequency spectrum and is a quarter of the size of the input. Hence, the time resolution is inversely proportional to the frequency resolution and adheres to the properties of the uncertainty principle, *i.e.* the product of the uncertainties in time and frequency are constrained by a lower bound.

The largest wavelet coefficients correspond to the points of excitation in a voiced signal, and are the most coherent over multiple filter channels, *i.e.* patterns of high peaks that are similar between channels, although scaled to different sampling rates. The distance between consecutive excitations within a filter channel can be used to give a pitch period estimate. Wavelet-based pitch detectors are discused in Fitch and Shabana [1999] and Griebel [1999].

2.5.3 Linear Predictive Coding (LPC)

Linear predictive coding (LPC), is a common method used in the speech recognition community, and is described well in Rabiner and Schafer [1978]. The basic idea is that a new speech sample can be approximated using a linear combination of the past p speech samples from a small finite time interval, *i.e.*,

$$\widetilde{x}_t = \sum_{k=1}^{p} \alpha_k x_{t-k}, \tag{2.31}$$

where α_k are the prediction coefficients and $\widetilde{x}_t$ is an estimate of the function at time t.

The input from voiced speech signals, x, is considered as an impulse train, *i.e.* a bunch of spikes spaced by the pitch period, which is convolved with a time-varying digital filter. This filter describes the parameters of the vocal tract transfer function. These filter parameters are assumed to vary slowly with time.

A by-product of the LPC analysis is the generation of an error signal, e_t, describing the difference between the estimated value and the actual value, shown by

$$e_t = x_t - \widetilde{x}_t. \tag{2.32}$$

This error signal typically contains large values at the beginning of each pitch period. Simply finding the distance between these peaks can yield the pitch period. However, the LPC error signal is often used in conjunction with autocorrelation analysis to find these more reliably.

The *maximum likelihood pitch estimation* method is related to LPC, estimating the parameters of a speech signal in white, Gaussian noise. A nice summary of this can be found in Griebel [1999].

Chapter 3

Investigation

Pitch detection algorithms (PDAs) have been around for a long time, and as can be seen in the literature review, there is a large number of them. There is no one algorithm that works best in all situations. Results will vary depending on the type of input, the amount of data that can be used at any given time, and the speed requirements.

When choosing to develop an algorithm, it is important to keep in mind the motivation and identify the goals and constraints which the algorithm will be expected to work under.

This chapter describes the goals and constraints specific to the objectives of this thesis in Section 3.1, before delving into an investigation and comparison of two existing techniques, the square difference function and autocorrelation, to see where they can be improved.

3.1 Goals and Constraints

We start by defining more precisely the goals of the project, which are discussed and used to guide the algorithm design thoughout the chapter. These are as follows:

- Firstly, the algorithm should give accurate pitch results for a variety of typical user input; preferably as accurately as a person can hear or distinguish in playing (Section 3.1.1).
- Secondly, the algorithm should cover the pitch range of common musical instruments, such as string, brass and woodwind instruments (Section 3.1.2). This also affects the typical sample rate that will be required.

- Thirdly, the system should be responsive. That is, the delay between the input to and the output of the algorithm should be kept to a minimum.

- Lastly, the algorithm should be real-time and capable of on-line processing. That is, it should be computationally efficient enough that the processing speed on a typical machine can keep up with the incoming data as the person is playing or singing.

The first three goals are discussed in Sections 3.1.1, 3.1.2 and 3.2 respectively. The last goal is discussed within the algorithm investigation in Section 3.3

One of the main concerns of a PDA for our purpose, is the range of pitch that it can detect. Section 3.1.2 discusses the typical musical instruments that are expected for the system, and hence the pitch range the algorithm will be required to cover.

Section 3.1.2 also looks at the use of filtering and how it can influence the sample rate required in the input. The other main concern is the accuracy of a PDA. Section 3.1.1 looks at how accurately humans can hear, and what accuracy we hope to expect from a machine.

3.1.1 Pitch Accuracy

The goal in this thesis is to make a tool that gives accurate feedback to the user on how they are varying the pitch within or between notes. Ultimately, any small change that can be heard by a good musician should be picked up by the algorithm and shown in an objective manner to the user. The following discusses more quantitatively what that entails.

The *just noticeable difference* (JND) is a measurement of the smallest change in a sensation that humans can perceive. For pitch perception the JND varies slightly depending on the frequency, sound level and duration of a tone, and suddenness of the change. However, from 1000 to 4000 Hz, the JND is approximately 0.5 percent of the pure tone frequency [Rossing, 1990]. That is to say humans can hear a pitch change of 8 cents[1] in a pure tone, although for complex tones it is possible for smaller differences to be perceived.

When it comes to playing notes together with other notes, for example in chords, or harmonising with others, small differences in pitch can be perceived, often through beating effects of harmonics, or increased roughness to the sound. For example, the difference between the *just scale* major-3rd and an *even tempered* major-3rd is 13.7

[1] 1 cent = $1/100^{th}$ of a semitone

cents and the difference between the just scale 5^{th} and an equal tempered 5^{th} is 2.0 cents. Note that scales and tuning are discussed in Section 10.2. Experienced musicians who play instruments in multiple temperaments can learn to play these pitches distinctively differently, and these differences can be perceived, often by how well they fit or do not fit to another instrument, such as the piano. It is therefore important that these small differences in pitch can be detected by the system as accurately as possible, preferably down to a couple of cents wherever possible.

3.1.2 Pitch range

The goal of the thesis is to make a system that handles a large range of musical instruments, including string, brass and woodwind. Musical instruments can cover a large range of pitch. For example, the lowest note on a double bass is an E1, at 41.2 Hz, with notes commonly ranging up to 3 octaves. This contrasts with the the piccolo which ranges 3 octaves from C5, taking it as high as C8, at 4186 Hz, with of course many instruments in between. This defines a frequency range that spans 100 fold, or about 7 octaves. However, the singing voice and most string, wind and brass instruments themselves span at most 3 or 4 octaves. Therefore, an algorithm that can cover a pitch range of 3 or 4 octaves at any given time is entirely satisfactory, provided this range can be moved for different instruments, for example, letting the user select the position of the pitch range, within the total 7-8 octave range.

It is worth noting that this large frequency range used in music makes for a slightly different problem than that found in speech recognition. Speech is mostly spoken in the chest register, a small range of the singing pitch range. Because of this, speech recognition algorithms often assume only a small range of pitch, and may apply filters to the signal to simplify the detection. However, the goal in this thesis is to cover the larger frequency range of musical instruments, so algorithms from this field cannot always be applied in the same fashion.

Sample Rate

Unlike speech signals, lowpass filtering and down-sampling cannot typically be used on musical signals. As a result the computational cost is higher, making efficiency very important.

In normal speech the range of fundamental frequencies between speakers is fairly small, compared to the range of fundamental frequencies of musical instruments. Over

a large collection of individual speakers, the range can vary from as low as 40 Hz from a low pitch male, to about 500 Hz in a high pitch female or child [Rabiner and Schafer, 1978]. However, during normal speech, a single person's pitch range is less that one octave [Rossing, 1990]. A lot of speech recognition methods filter out the high frequencies in a signal using a lowpass filter prior to analysis, on the basis that the pitch is within this low range. However, for musical signals in general, a much larger range of pitch can occur, so the luxury of lowpass filtering and down-sampling cannot be applied. One must consider the full range of frequencies when looking for musical pitch.

The range of frequencies from 20 Hz to 20000 Hz is considered to cover most people's hearing capability. Musical instruments often contain harmonics which can extend up to the high end of this range. It is known that even high harmonics can influence the pitch of a note. It is therefore important to keep a sample rate sufficient to retain all the sound information thoughout the pitch analysis process. From using the *sampling theorem* [Rabiner and Schafer, 1978], a sample rate of more that twice the highest frequency is required to maintain all the frequency data. So thoughout this project, a sampling rate of 44100 Hz was used for exactly these reasons. This is a higher sampling rate than that typically used in speech recognition.

The consequences of maintaining a high sample rate are an increase the computation cost of certain tasks, making the efficiency of the algorithms an important consideration. So, the efficiency of different algorithms will be discussed throughout this chapter.

Although filtering for down-sampling purposes will not be used, there are some types of filtering that can be very useful. Section 6.2.2 discusses some filters that are used to approximate the sound attenuation throughout the outer and middle ear.

3.2 Responsiveness

One of the goals of this thesis is to be able to follow fast variations in pitch that can happen within a note, such as that during vibrato. Ideally, an algorithm would give an instantaneous pitch of a waveform from a single sample. However, in practice a certain window size is needed.

Typically there is a time/frequency trade off, with larger window sizes giving a higher pitch accuracy, but less time resolution; that is to say any changes in pitch within the window are lost. In contrast a small window gives higher time resolution,

but a less accurate frequency measure. Frequency domain techniques can work on arbitrary frequency components, generally requiring 3-5 periods of the waveform to accurately separate out the frequency components. However, because this thesis is only concerned with harmonically related waveforms, certain time domain techniques such as autocorrelation and SDF prove more useful, with an ability to get a pitch estimate with as few as 2 periods of a waveform.

Vibrato can commonly vary at speeds of up to 12 cycles per second, and reach a full semitone in width. In order to trace out this changing pitch, particularly at lower pitches, being able to find the pitch in the smallest window size possible becomes of primary concern.

3.3 Investigation of some Existing Techniques

This section compares two fairly successful techniques, that cover some of the desired traits for our algorithm. The differences are investigated between the SDF and autocorrelation, followed by examining some computational complexities involved in the algorithms, and ways to speed up the computation.

The SDF (described in section 2.3.3) has some nice properties. For each lag value, τ, the waveform is compared to shifted version of itself and a squared error calculated, that is the sum of the squared differences. When the waveform is repeating it causes minima to occur in the lag domain at multiples of the repeat size. Therefore, finding minima in the lag domain gives us information about where the waveform repeats itself, and hence the period. However, in real sounds the waveforms usually do not repeat exactly, but tend to change shape over time, with different harmonics in the signal changing in amplitude at different rates. Real sounds are often referred to as quasi-periodic because over a short time they are approximately periodic. Even with a quasi-periodic signal, the lag domain will produce minima at the most periodic-like values, although, because they do not match exactly will have a non-zero error.

3.3.1 SDF vs Autocorrelation

There are subtle differences between the SDF and the autocorrelation. Apart from one giving minima where the other gives maxima, there are differences between exactly where these turning points occur. These differences are small when a window contains a large number of periods; however, if the the window contains only a few periods of a

waveform, the differences between the output of the SDF and autocorrelation become significant.

Let us take a look at a simple example to help us understand why. Figure 3.1 shows a window that contains 2.25 periods of a sinusoid, and a series of phase shifted versions. Comparing the results of the autocorrelation in Figure 3.2 and the SDF in Figure 3.3, it can be seen that the SDF minima align at a constant value, $\tau = 910$, across all phases of the input, however, the autocorrelation maxima range between $\tau = 909$ and 925. This inconsistency is due to the varying amounts contributed by partial waveforms at different phases.

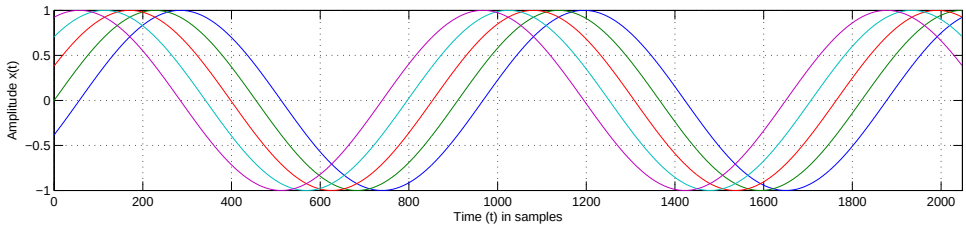


Figure 3.1: A window containing sinusoids with different starting phases $(0, \frac{\pi}{8}, \frac{2\pi}{8}, \frac{3\pi}{8}, \frac{4\pi}{8})$. Each sinusoid has 2.25 periods within the window.

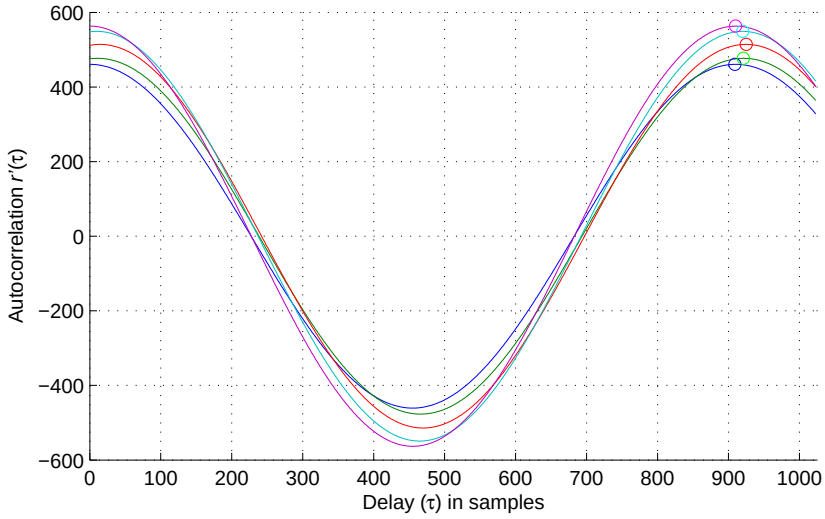


Figure 3.2: An autocorrelation plot of the sinusoids from Figure 3.1. Here the circled maxima do not line up, and give a period estimate which varies with the phase.

This is the nature of the autocorrelation, which requires a whole number of periods in a window to balance out the different phase related summations. Autocorrelation

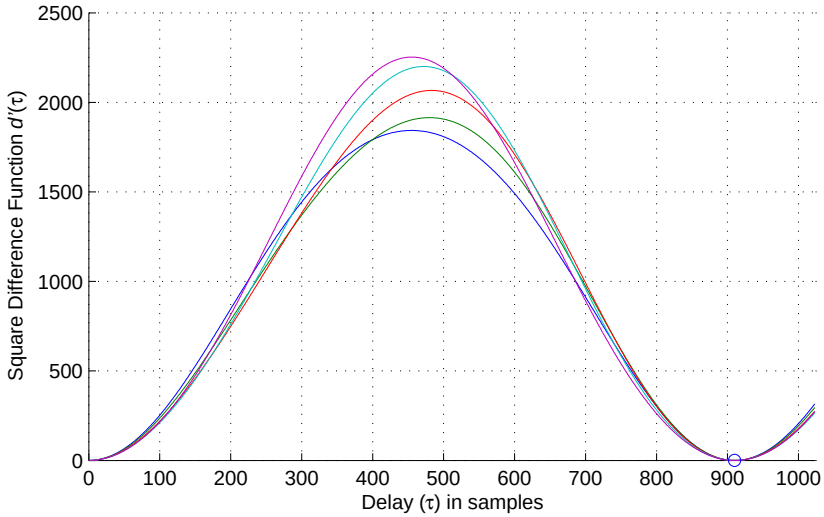


Figure 3.3: A square difference function plot of the sinusoids from Figure 3.1. Here all the minima lie at the same value of τ, giving a stable estimate accurate to the nearest whole number.

without a windowing function can lead to an inaccurate period estimation when there is not a whole number of periods in a window.

Autocorrelation can be used to determine the period accurately only if there are a whole number of periods in the window. However, since the period is initially unknown, choosing the correct window size based on the period size becomes a chicken and egg problem, requiring the period in order to choose a window size in order to find the period.

A windowing function, such as a Hanning window, can be applied in order to reduce the edge effects. However, this generally requires a larger window size to achieve the same resolution, adding extra computational cost. The windowed and non-windowed autocorrelation are compared experimentally in Chapter 5.

In contrast to the autocorrelation, the SDF outputs stable minima, given any fractional number of periods in the window, making it suitable for using in a pitch detection scheme using small windows, and without windowing. One can, without any prior knowledge of phase, take a window from anywhere within a signal and use the SDF to get a good idea of the period.

3.3.2 Calculation of the Square Difference Function

The straight calculation of the SDF on a window is (from Equation 2.14),

$$d'(\tau) = \sum_{j=0}^{W-1-\tau} (x_j - x_{j+\tau})^2, \qquad 0 \leq \tau < W. \tag{3.1}$$

It has a time complexity of $O(N^2)$, where N is the size of the initial window. With a 44.1 kHz sampling rate with window sizes of say 1024 or 2048, and hop size of half a window it is almost feasible for today's average processor (say 3 GHz) to keep up. However this would leave little processor time for anything else. In this chapter we discuss ways of reducing the time complexity of this calculation.

3.3.3 Square Difference Function via Successive Approximation

For pitch detection purposes, only the indices of the smallest minima from the SDF are of interest. The computation required can be reduced by removing some of the more redundant calculations from SDF. The calculations of $d'(\tau)$ which are not near minima become redundant so have no need to be calculated fully. The question becomes how to tell which values of $d'(\tau)$ to calculate fully. A strategy was developed as follows:

1: **repeat**
2: Estimate values of $d'(\tau)$, using only a subset of indices j in Equation 3.1
3: Keep about half the remaining τ, with values most likely to be at global minima - for example the τ where $d'(\tau)$ is less than the mean or median of d'
4: **until** there remains few enough τ values to fully calculate all $d'(\tau)$ quickly
5: Calculate $d'(\tau)$ fully for these remaining τ

How do we choose the subset of x_j values to approximate $d'(\tau)$ in line 2 above? One simple method is to use every k^{th} value of x, where k starts out being quite large, and is reduced during further iterations of the first two steps. However, this causes any frequencies above k/2 to be aliased, bringing false minima, but still retaining original minima. False positives are acceptable, as keeping non-minima is fine. False negatives are unacceptable though: the τ value of the global minimum must not be allowed to be removed from the set.

Another possibility is to choose x_j values at random to try to remove any systematic sampling effects. However, the initially chosen x_j values do not guarantee the sum of

square differences of τ with the smallest global minima are below the chosen cut-off. It will get most of them most of the time, but it is not always reliable.

It appears that the global minima cannot be guaranteed; by trying hard enough a case can typically be constructed that will cause the true global minima to be lost.

After futher investigation, the discovery of a computationally efficient method for calculating the full SDF came about, removing the need to approximate the important τ. The SDF can be calculated via an autocorrelation in Fourier space, explained in section 3.3.4. This allows all the SDF coefficients to be calculated efficiently and allows the smallest minima to be chosen reliably - leaving this *successive approximation* method quite redundant.

3.3.4 Square Difference Function via Autocorrelation

The square difference function can be calculated more efficiently than the direct $O(W^2)$ calculation, by utilising the autocorrelation via FFT method (section 2.4.5). The following shows how to take the SDF from Equation 3.1, and break it up into parts, which can be calculated more efficiently on their own, and then combined together. At the same time, the following equations give us a direct relationship between the SDF and autocorrelation. This relationship is:

$$\begin{aligned} d'(\tau) &= \sum_{j=0}^{W-\tau-1} (x_j - x_{j+\tau})^2 && (3.2) \\ &= \sum_{j=0}^{W-\tau-1} (x_j^2 + x_{j+\tau}^2) - 2 \sum_{j=0}^{W-\tau-1} x_j x_{j+\tau} && (3.3) \\ &= m'(\tau) - 2r'(\tau), && (3.4) \end{aligned}$$

where m' describes the lefthand summation term and r' describes the autocorrelation term (see equation 2.6). This is a very useful relationship. From here it can be seen that m' can be calculated incrementally as follows:

1. Calculate $m'(0)$ by doing the summation.

2. Repeatedly calculate the rest or the terms using

$$m'(\tau) = m'(\tau - 1) - x_{\tau-1}^2 - x_{W-\tau}^2 \ . \qquad (3.5)$$

Calculating, all the m' terms is therefore linear in W. The calculation of r' term is $O(W \log(W))$ using the autocorrelation via FFT technique from Section 2.4.5.

This method is reliable because it is calculating the full SDF, and does not have the possibility of missing the global minimum like the successive approximation method. Some small rounding errors can be introduced from the FFT, especially with low precision numbers, however using 32 bit floating point appears sufficient for our purpose, where the sound data is originally 8 or 16 bit.

3.3.5 Summary of ACF and SDF properties

The following compares properties from the different forms of the ACF and SDF, keeping in mind the possibility of designing a new transformation function that combines the good properties from both.

In the type-I ACF from Equation 2.5 there are a fixed number of terms, in which the shifted part moves to the right each step. We define the 'effective centre', $t_c(\tau)$, as the average in time of all the terms' positions in the summation. That is

$$t_c(\tau) = \frac{1}{W/2} \sum_{j=0}^{W/2-1} \frac{1}{2}\big[j + (j+\tau)\big] \tag{3.6}$$

$$= \frac{1}{4}W + \frac{\tau - 1}{2} \tag{3.7}$$

The effective centre of the type-I ACF moves with different values of τ. In order to compare these ACF values together sensibly, one needs to make the assumption that the period is stationary across the window.

In contrast, the type-II ACF has an effective centre, $t'_c(\tau)$, that is constant, for a given W, across all values of τ, giving a kind of symmetry to the transformation. That is

$$t'_c(\tau) = \frac{1}{(W-\tau)} \sum_{j=0}^{W-1-\tau} \frac{1}{2}\big[j + (j+\tau)\big] \tag{3.8}$$

$$= \frac{W-1}{2} \tag{3.9}$$

This symmetry property maximises cancellations caused by frequency deviations on opposite sides of t'_c within the window, making a direct comparison of ACF values more robust, without having to consider their position in time. Also, with symmetry, it makes more sense to be able to interpolate between the results of different τ, to estimate fractional periods, as the results will be at the same centre point in time. It would be nice if any new transformation function also featured this property.

One advantage of SDF over the ACF is that it does not suffer so strongly from edge effects, that is to say it does not require a whole number of periods in the window in

order to operate effectively, whereas the ACF does. This allows smaller window sizes to be used, while maintaining accuracy, resulting in a higher time resolution. It would be desireable if a new transformation function didn't suffer too much from edge effects.

A problem with SDF is making a useful normalisation. Even if it is divided by the sum of squares of the window, there is no sense of a centre. Whereas the ACF has a concept of positive matches and negative matches, with a zero line indicating no correlation, which can be of great help when choosing which peak corresponds to the fundamental period.

Section 4.1 introduces a transformation function that mantains most of these nice properties.

Chapter 4

A New Approach

From the previous chapter a conflict in features has emerged between two of the most well suited algorithms. The SDF has stable minima, as it can work well in a window containing a fractional number of periods. But the autocorrelation, has the advantage of a zero centre-reference, which is very beneficial when deciding which maxima are significant. Is it possible to combine the stability from one algorithm, with the zero reference property of the other, and at the same time provide a sensible normalisation? This chapter develops a new algorithm in Section 4.1 based on these two algorithms, that combines these nice properties. The algorithm is then extended to allow windowing in Section 4.2, before Section 4.3 discusses a simple interpolation technique to find the position of the output's peak to less than a sample. Another useful property of the SNAC function, called the clarity measure, is mentioned in Section 4.4. Note that Sections 5.2 - 5.4 describe thorough testing on these new algorithms, comparing them to the existing algorithms, through a series of systematic experiments.

4.1 Special Normalisation of the Autocorrelation (SNAC) Function

The autocorrelation described in Equation 2.6 has a tapering effect. Section 2.3.2 discusses the unbiased autocorrelation as a method for removing the tapering effect, and also a method for normalising values between -1 and 1, in which the result is divided by $r'(0)$. Combining those two ideas gives

$$r'_{\text{Unbiased}}(\tau) = \frac{W}{(W-\tau)r'(0)} \sum_{j=0}^{W-1-\tau} x_j x_{j+\tau}. \tag{4.1}$$

In this equation the dividing factor is a certain proportion, $0 < \frac{W-\tau}{W} \leq 1$, of the total energy in the window, $r'(0)$. This may seem reasonable; however, for a given value of τ this does not accurately reflect the highest possible correlation. It is assuming that the total energy is uniformly distributed across the window, which is by no means the case.

We propose a new method of normalisation in which the energy of the terms involved in the summation, at a given τ are used directly to give the denominator. The *specially-normalised autocorrelation* function, or SNAC[1] function, is defined as follows:

$$n'(\tau) = \frac{2\sum_{j=0}^{W-1-\tau} x_j x_{j+\tau}}{\sum_{j=0}^{W-1-\tau} (x_j^2 + x_{j+\tau}^2)} \tag{4.2}$$

$$= \frac{2r'(\tau)}{m'(\tau)}. \tag{4.3}$$

In this equation, the special-normalisation term, $m'(\tau)/2$, takes on the maximum correlation that can occur in $r'(\tau)$ for any given value of τ. Here is a simple proof of this. If x_a and x_b are real numbers, then

$$(x_a - x_b)^2 \geq 0, \quad \text{so} \tag{4.4}$$

$$x_a^2 - 2x_a x_b + x_b^2 \geq 0, \quad \text{and} \tag{4.5}$$

$$x_a^2 + x_b^2 \geq 2x_a x_b, \quad \text{and therefore} \tag{4.6}$$

$$\sum_{j=0}^{W-1-\tau} (x_j^2 + x_{j+\tau}^2) \geq 2\sum_{j=0}^{W-1-\tau} x_j x_{j+\tau}, \tag{4.7}$$

because if it is true for each term in the sum, then it is also true for the whole sum. And in a similar fashion the negative is proved,

$$(x_a + x_b)^2 \geq 0, \quad \text{so} \tag{4.8}$$

$$x_a^2 + 2x_a x_b + x_b^2 \geq 0, \quad \text{and} \tag{4.9}$$

$$x_a^2 + x_b^2 \geq -2x_a x_b, \quad \text{and therefore} \tag{4.10}$$

$$\sum_{j=0}^{W-1-\tau} (x_j^2 + x_{j+\tau}^2) \geq -2\sum_{j=0}^{W-1-\tau} x_j x_{j+\tau}. \tag{4.11}$$

Combining Equation 4.7 and Equation 4.11 it is true that

$$\sum_{j=0}^{W-1-\tau} (x_j^2 + x_{j+\tau}^2) \geq |2\sum_{j=0}^{W-1-\tau} x_j x_{j+\tau}|. \tag{4.12}$$

[1] pronounced 'snack'

This guarantees Equation 4.2 to be in the range $-1 \leq n'(\tau) \leq 1$, a desired property.

If there is no correlation between the signal and itself shifted by τ, the autocorrelation part on the top of the Equation 4.2 becomes zero, making the whole equation zero also. It follows then that the original zero centre reference still remains from the autocorrelation property. In contrast, if there is perfect correlation between the signal and itself shifted by τ, meaning all values of $x_j = x_{j+\tau}$, then the summations on the top and bottom of Equation 4.2 both become

$$2 \sum_{j=0}^{W-1-\tau} x_J^2, \tag{4.13}$$

making the result a perfect 1. This is a great improvement over the *normalised unbiased autocorrelation*, in Equation 4.1, which does not guarantee this property to be true.

Next, the SNAC function is shown to be similar to SDF, and hence certain properties of stability are carried over. If a minimum is found to be zero in the SDF, then a maximum will occur at the same lag term, τ, in the SNAC function. Using the relationship between the autocorrelation and SDF found in Section 3.3.4, an equivalence between the SNAC function and an SDF that is normalised in a similar fashion is made. Firstly, rearranging Equation 3.4 gives

$$2r'(\tau) = m'(\tau) - d'(\tau). \tag{4.14}$$

Then substituting in Equation 4.3 gives

$$n'(\tau) = \frac{m'(\tau) - d'(\tau)}{m'(\tau)} \tag{4.15}$$

$$= 1 - \frac{d'(\tau)}{m'(\tau)}. \tag{4.16}$$

This result shows the SNAC function is equivalent to one minus an SDF that is divided by $m'(\tau)$. Equation 4.16 has also been dubbed the *normalised SDF*, or NSDF [McLeod and Wyvill, 2005], as it is a way of normalising the SDF to make it more like an autocorrelation. However, in this thesis we will refer to $n'(\tau)$ as the SNAC function, as it is easier to pronounce.

So far, we have shown that the SNAC function has taken on certain properties from both the autocorrelation and the SDF. Firstly, the zero centre reference from the autocorrelation remains, which will prove to be very useful in the next stage of choosing the desired peak. Secondly, certain peaks are guaranteed to be aligned with those from the SDF, providing peak stability. Thirdly, the normalisation of the autocorrelation has improved, allowing each evaluation of τ to utilise the range -1 to 1, providing a well defined, rock solid vertical scale allowing thresholds to be confidently set.

4.2 Windowed SNAC Function

Although the SNAC function works well without the need of a windowing function, maybe there is still some benefit in windowing anyway. This section discusses the question of "how do we window the SNAC function?" and "how do we do it efficiently?". Whereas, the question of whether it is beneficial to window or not, is left until Chapter 5, where experiments are performed to compare the different methods.

Trying to window the SNAC function complicates the algorithm, breaking the current speed optimisation. To calculate the *windowed SNAC* (WSNAC) function by summation is an order $O(W^2)$ process, where W is the window size. However, the following introduces a fast method for calculating the WSNAC function, in a time complexity of order $O(W \log W)$, which can be used for real-time analysis.

To explain how this comes about, let us first look at windowing the square difference function. We cannot simply multiply x by the windowing function and take the SDF using Equation 3.2, as is the case for autocorrelation. This would cause the repeating peaks and troughs to change in height with respect to each other, resulting in large square difference error terms. However, one can multiply the result of a square difference term by the constituent window parts, giving a stronger weighting to the central difference values. The *windowed square difference function*, $d'(\tau)$, is defined as:

$$\widehat{d}'(\tau) = \sum_{j=0}^{W-\tau-1} \Big[w_j w_{j+\tau} (x_j - x_{j+\tau})^2 \Big]. \tag{4.17}$$

Note that the hat above the d indicates the windowed version.

This has a two part windowing function, which is discussed in Section 4.2.2.

Now let us expand Equation 4.17 out in a similar fashion as Equation 3.3, giving us:

$$\begin{aligned} \widehat{d}'(\tau) &= \sum_{j=0}^{W-\tau-1} \Big[w_j w_{j+\tau} (x_j^2 + x_{j+\tau}^2) \Big] - 2 \sum_{j=0}^{W-\tau-1} (w_j w_{j+\tau} x_j x_{j+\tau}) && (4.18) \\ &= \widehat{m}'(\tau) - 2\widehat{r}'(\tau) && (4.19) \end{aligned}$$

The right-hand summation becomes $\widehat{r}'(\tau)$, a windowed autocorrelation which can be solved in order $O(W \log(W))$ using the Fourier domain technique described in Section 2.4.5. The $\widehat{m}$ is a little bit trickier, but if we break $\widehat{m}'$ up as follows:

$$\widehat{m}'(\tau) = \sum_{j=0}^{W-\tau-1} (x_j^2 w_j \cdot w_{j+\tau}) + \sum_{j=0}^{W-\tau-1} (x_{j+\tau}^2 w_{j+\tau} \cdot w_j), \tag{4.20}$$

then each summation can be expressed as a crosscorrelation (Section 4.2.1). Note that the $\cdot$ just indicates the grouping of multiplication. The terms in j and $j+\tau$ can be grouped separately within each summation. It turns out that both the first and second summation can be calculated as different terms in a single crosscorrelation of v_a with v_b, where $v_a \in x_j^2 w_j$, and $v_b \in w_j$.

The WSNAC function, $\widehat{n}'(\tau)$, can therefore be defined in a similar way to Equation 4.3, in terms of $\widehat{r}'(\tau)$ and $\widehat{m}'(\tau)$, giving

$$\widehat{n}'(\tau) \;=\; \frac{2\widehat{r}'(\tau)}{\widehat{m}'(\tau)}. \tag{4.21}$$

The whole range of τ values can be calculated efficiently using the method described above. However, the result is equivalent to the full form of the WSNAC function shown as follows:

$$\widehat{n}'(\tau) \;=\; \frac{2\sum_{j=0}^{W-\tau-1}(w_j w_{j+\tau} x_j x_{j+\tau})}{\sum_{j=0}^{W-\tau-1}\Big[w_j w_{j+\tau}(x_j^2 + x_{j+\tau}^2)\Big]} \tag{4.22}$$

4.2.1 Crosscorrelation via FFT

The crosscorrelation can be calculated in the Fourier domain in a similar way to the autocorrelation described in Section 2.4.5. The algorithm differs, giving the following:

- Zero-pad both v_a and v_b with p zeros.
- Compute the FFT, V_a of v_a, and V_b of v_b, using size $W+p$.
- Multiply together the magnitudes of each complex term. *i.e.* $V_{ab}(k) = |V_a| \cdot |V_b|$.
- Compute an inverse FFT of V_{ab} to obtain the crosscorrelation (type-II) of v_a and v_b.

Note that the autocorrelation is just a special case of crosscorrelation, when $v_a = v_b$.

4.2.2 Combined Windowing Functions

Taking a look back at the WSNAC function in Equation 4.17, the windowing function is made up of two parts, w_j and $w_{j+\tau}$. Interaction between the two parts of the windowing function is interesting, as the net result is a windowing function that changes shape

between different values of τ, but keeps the same category of shape. For this, w is chosen to be a Hann window,

$$w_j = 0.5(1 - \cos(\frac{2\pi j}{W-1})), \qquad 0 \leq j < W \tag{4.23}$$

$$= \sin^2(\frac{\pi j}{W-1}). \tag{4.24}$$

This second equation can be found using the *double angle* trigonometric property and is more useful for descriptive purposes. Let us investigate the result of this function as used in Equation 4.17. The net windowing result of $w_j w_{j+\tau}$ when $\tau = 0$ is the Hann function squared. At $\tau = W/2$ we get the left half of the Hann function multiplied by the right half of the Hann function, which is just the Hann function divided by four. Moreover, when $0 < \tau < W/2$, the resulting shape is somewhere between a Hann function and a Hann function squared. Figure 4.1 shows an example of the combined windowing effect using a mid-range τ value.

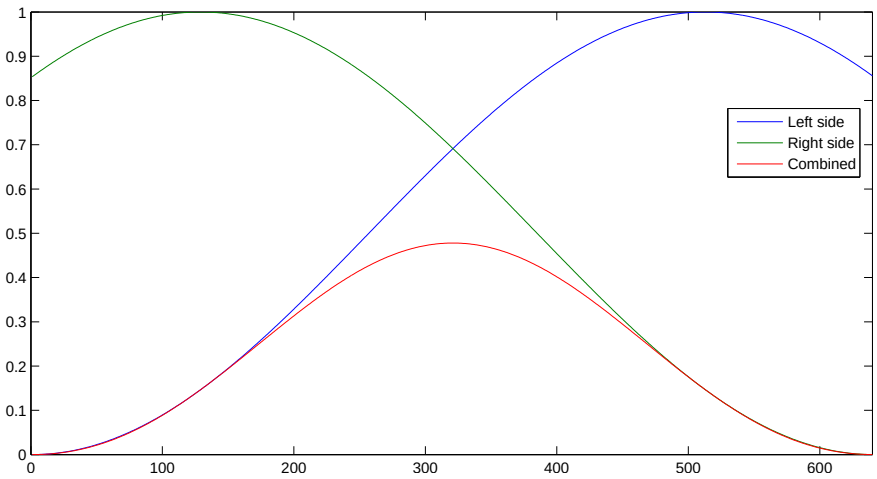
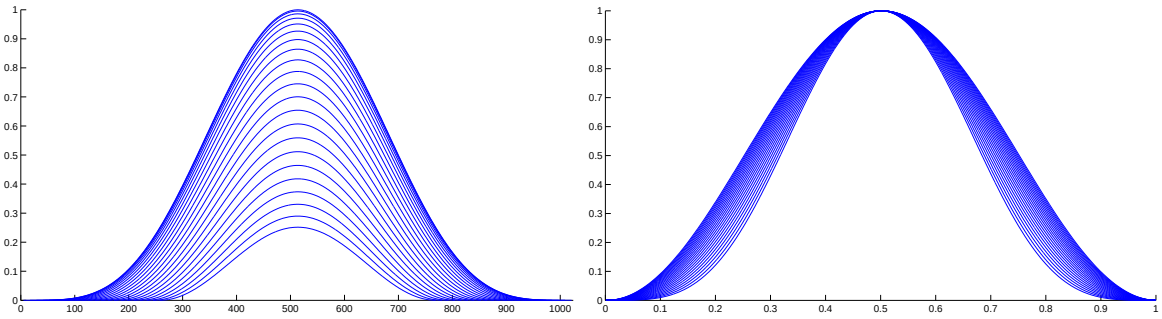


Figure 4.1: The combined windowing effect of the left part of the Hann window multiplied by the right part, with $W = 1024$, and $\tau = 384$.

Figure 4.2 shows us how the combined windowing function changes in shape as τ varies between 0 and $W/2$. It varies between a $\sin^4(\theta)$ shape and a $\sin^2(\theta)$ shape.

The resulting effect of this, is that when detecting larger period sizes a slightly stronger weight will be applied to the central values than when detecting smaller periods. This may not appear an ideal effect, but does not seem to have any negative effects on the period detection. The benefits of the speed increase gained by using this technique make it worthwhile.



(a) As calculated (b) With normalised width and height

Figure 4.2: The combined windowing effect over changing τ, with $W = 1024$.

In comparison to the Hann window, a Sine window can also be used with a similar effect. Here the sine window is defined as

$$w_j = \sin(\frac{\pi j}{W - 1}). \tag{4.25}$$

Using the sine window, the combined window results range between a $\sin^2(\theta)$ shape and a $\sin(\theta)$ shape. This allows for a larger portion of the data to be utilised in the window, while maintaining a smooth edge drop off, so no sharp discontinuities occur at edges.

The accuracy of the WSNAC function for pitch detection is compared to the SNAC function and other autocorrelation-type function in Chapter 5, through a number of experiments.

4.3 Parabolic Peak Interpolation

Currently, τ values in the autocorrelation-type functions are restricted to integers, corresponding to the time steps of the original sound sample. However, a higher degree of accuracy in the positions of a peak is desirable. Therefore, a parabola is fitted to a peak, because it makes a good approximation to a localised curve at the top and gives a fast, stable and accurate way to find the real-valued position of a peak.

A parabola is constructed for a peak using the highest local value of $n'(\tau)$ and its two neighbours. The 2D-coordinate of the parabola maximum is found, with the x coordinate giving a real τ value for a primary-peak's lag, and the y coordinate giving a real value for the function $n'(\tau)$ at that point. It should be noted that a $n'(\tau)$ value slightly greater than one is actually possible for the real-valued peak using the parabola technique.

4.4 Clarity Measure

We have found that not only is the x-coordinate of a SNAC or WSNAC peak useful for finding the pitch, but the y-coordinate of a peak is also a useful property which is used later in Section 10.3.2. We call this value the clarity measure as it describes how coherent the sound of a note is. If a signal contains a more accurately repeating waveform, then it is clearer. This is similar to the term 'voiced' used in speech recognition. Clarity is independent of the amplitude or harmonic structure of the signal. As a signal becomes more noise-like, its clarity decreases toward zero. The value of the clarity measure is simply equal to the height of the chosen peak.

Chapter 5

Experiments - The Autocorrelation Family

In this chapter, the two new methods introduced, the SNAC function, and the WSNAC function, are put to the test along with other standard functions from the autocorrelation family. Because these functions are all similar in nature, direct comparison can be performed to measure their accuracy, across a range of different inputs.

The goal is to measure how accurate each of the methods is at detecting pitch in a fair and consistent manner. Therefore, all methods have the same parabolic peak interpolation algorithm used to measure the period of the pitch as described in Section 4.3. For now, it is assumed that exactly which peak to measure is known. The experiments in this chapter contain signals which are generated mathematically. This allows for all the parameters to be fully controlled, and provides certainty in the accuracy measure, as the desired output is always known. Note that this chapter is concerned only with finding the frequencies of the peaks. The problem of how and when these relate to the pitch frequency is discussed in Chapters 6 and 7.

The experiments were designed to reflect the types of changes in sound that are most common when playing a musical instrument. Section 5.1 discusses stationary signals, *i.e.* that of a constant musical tone, while Section 5.2 discusses changing the input's frequency, such as that during vibrato or glissando. Section 5.3 discusses changing the input's amplitude, such as that during note onset and note release, and tremolo. Some basic tests of the input signal buried in noise are discussed in Section 5.4, with a summary of all the experiments in Section 5.5.

5.1 Stationary Signal

Experiments were carried out to compare the accuracy of some conventional functions in the autocorrelation-type family. The functions compared are the autocorrelation as described in Equation 2.6, the unbiased autocorrelation as described in Equation 4.1, and the windowed unbiased autocorrelation as described in 2.10. These were all used with standard normalisation. Using standard normalisation does not change the position of the peaks at all, and hence the same result will be given with or without it. Also note that a Hamming window was used for the windowed unbiased autocorrelation. Moreover, the two new functions, the SNAC function, and the WSNAC, are also compared to the others.

Experiment 1a: Sine wave.

Firstly, a small segment of a sine wave was sampled at known frequency. Then each of the autocorrelation-type functions were used to transform the data into lag space, and the top of appropriate peak was found using the technique described in Section 4.3. The error was taken as the absolute difference from the measured MIDI number and the original sine wave's MIDI number. For each given frequency, the sine wave was started at a number of different phases, and the maximum error and average error graphed. Figure 5.1 shows this measured error of sine waves at different frequencies, ranging from MIDI numbers 41.5 to 101.5 at intervals of 0.1, covering a 5 octave range[1]. Appendix A contains a useful reference for converting between pitch names, MIDI numbers and frequencies. You should be aware that a linear increase in the MIDI number, corresponds with an exponential increase in frequency; that is an increase of 12 semitones is a doubling of the fundamental frequency.

To generate Figure 5.1, window sizes of 1024 samples were used and a sample rate of 44100 Hz, and hence MIDI number 41.5 is chosen as the bottom of the range, as this produces just over 2 periods in the window. The period can be found with less than 2 periods - using autocorrelation-type methods, however, the reliability of finding the correct period in the waveform drops off quickly, as there can be no guarantee that just because part of the waveform correlates, that the rest of it does too. However using fewer than 2 periods can be useful at certain times, if other information is known. Note that the vertical axis is in cents and is not proportional to the frequency error, but is relative to input value, *i.e.* a 1 cent error at MIDI note number 80 is greater in frequency than a 1 cent error at MIDI note number 50; however, it is the error in cents

[1]Refer to Equation 2.1 to convert MIDI numbers to frequency

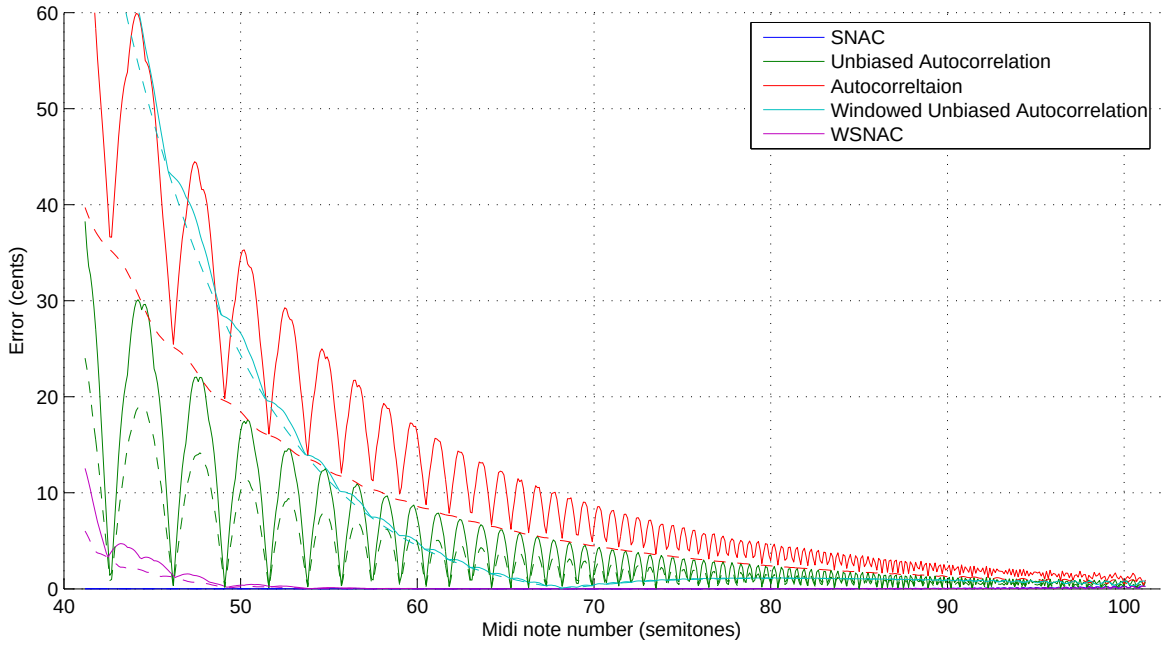


Figure 5.1: A comparison of autocorrelation-type methods for pitch accuracy on a sine wave input at different frequencies. The vertical axis indicates the measured error from original MIDI number. A solid line indicates the maximum error across different phases of the given frequency, whereas a dotted line indicates the average error.

that is important to pitch perception.

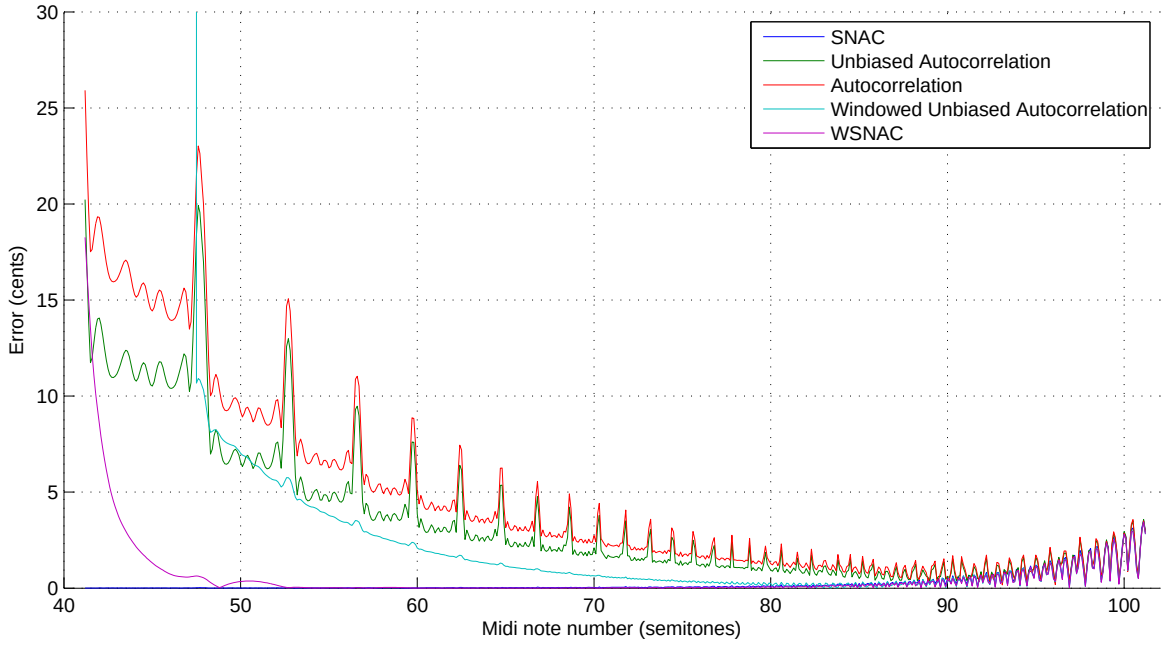
Looking at the results, the autocorrelation, unbiased autocorrelation and windowed unbiased autocorrelation, all have large errors, especially to the left of the graph which corresponds to a smaller number of periods of the sine wave in the input window. The windowed unbiased autocorrelation starts off with the highest error, but by MIDI number 72, has improved over the autocorrelation and unbiased autocorrelation; this is around 12 periods within the input window. Also, the autocorrelation and unbiased autocorrelation have sharp dips along them. These dips correspond to MIDI numbers that produce a window containing a whole number of periods. The unbiased autocorrelation shows great results in the dips, but suffers elsewhere. However, the error from the SNAC function barely even registers on the graph, with its error being less that 0.04 cents up until MIDI number 90, and creeps up to 0.25 cents around MIDI number 100. Thus, the SNAC function is far superior for use on static sine waves. Moreover, the WSNAC function performs the second best, with a small error at the lower MIDI numbers making its results promising.

Experiment 1b: Complicated waveform

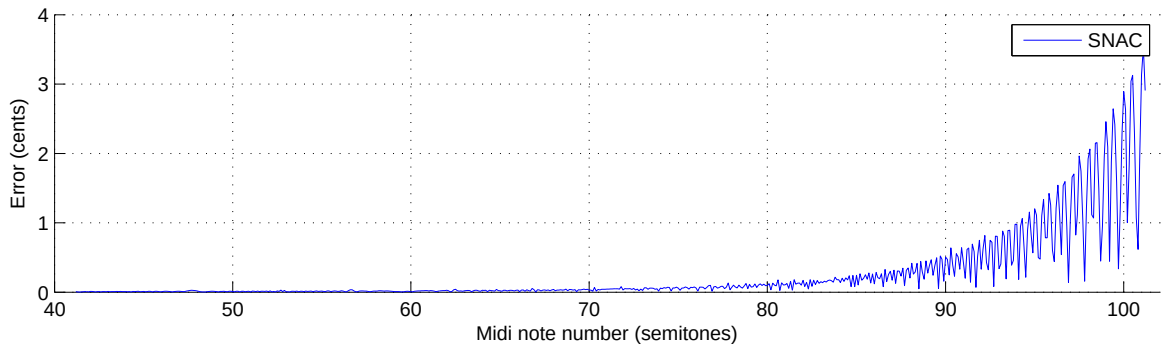
This experiment is conducted in a similar fashion to Experiment 1a. However, this time a more complicated waveform is used instead of a single sine wave. The waveform contains a series of sine waves added together; each having a different frequency and amplitude. The frequency of these sine waves, are $F_0, 2F_0, 3F_0$ and $4F_0$, forming the first four harmonics of the fundamental frequency. The amplitudes of these frequencies are 0.4, 0.7, 0.3 and 0.5. Note that this function is used repeatedly in later experiments and its shape is shown later in Figure 5.6a. The results of the autocorrelation-type methods on this more complicated waveform are shown in Figure 5.2a. A magnified view of the results of the SNAC function are shown in 5.2b for clarity.

The windowed unbiased autocorrelation struggles to find any sensible result before MIDI note 49, but after that has a stable error curve compared to autocorrelation and Unbiased Autocorrelation which are affected significantly by the shape of the input waveform. Again, the SNAC function performs much better than the other three methods, with almost no error up until about MIDI number 85, where the error acquired from calculating the peak in the lag domain takes over. Again the WSNAC function performs second best, with significant errors only kicking in at very low MIDI numbers.

We should not draw conclusions too quickly, since in real music, the signals are often changing. This section has only looked at stationary periodic sequences, *i.e.* where the



(a) The results from all autocorrelation types.



(b) A magnified view of the SNAC function results.

Figure 5.2: A comparison of autocorrelation-type methods on finding the pitch of a complicated waveform at different frequencies. Only the maximum error across the different phases at each given frequency are shown.

frequency and waveform shape have remained constant. However, stationary signals do make a good basic test case, because if the algorithm does not work well on stationary signals, then it is not much use to us. The two new functions have achieved well so far, but the next few sections will put them to tougher tests.

5.2 Frequency Changes

So far we have assumed the frequency of the waveform has not been changing within the window. The following looks at what happens to the autocorrelation-type functions on an input with changing frequency, investigating with both a simple sine wave, and a more complicated waveform shape. Firstly, Section 5.2.1 looks at how the algorithms deal with a linear frequency change, such as that during glissando. Secondly, Section 5.2.2 looks at frequency modulation, such as what happens when a musician plays vibrato.

Putting things in perspective here for a moment, if we were only concerned with finding the frequency of a varying sine waves, there are other methods that could be used to greater accuracy, such as Prony's method discussed later in Section 9.3. However, the reason for all this effort is because the autocorrelation-type methods can be used for the more general problem of any waveform shape. These methods can help us find the fundamental frequency within any data window, and hence the pitch.

5.2.1 Linear Frequency Ramp

Firstly, Figure 5.3 compares the shape of a constant frequency sine wave of 440 Hz, with a sine wave changing linearly in frequency from 427.3 Hz to 452.7 Hz. In this example the frequency ramp takes place within a window size of 1024 samples. Notice how the blue line starts with a larger period than the red, but ends with a smaller period. Moreover, the frequency and hence the period are the same at the centre.

To put this into a musical context, this frequency ramp constitutes a one semitone change in $1/40^{th}$ of a second. The goal here is to represent the upper bound of a typical fast pitch change, as a tough test to get an idea of how the algorithms respond. Moreover, when the algorithm is applied to real sounds the behaviour can be expected to fall within these bounds.

Figure 5.4(b) shows the SNAC function of the constant 440 Hz sine wave in red dashed lines, and the SNAC function of the changing frequency in blue solid line. It can be seen that the 5^{th} peak, at $\tau = 502$, of the blue curve has dropped to 0.96.

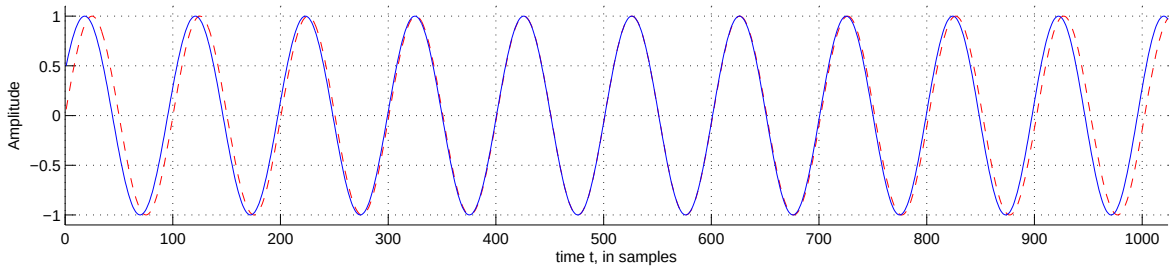


Figure 5.3: A constant sine wave, in the red dashed line, compared to a linear frequency ramped sine wave, in the blue solid line.

However, even under phase variation all the peaks remain well aligned. That is to say, a linear frequency change has SNAC function peaks at the same lag as the constant frequency of that at the centre of the window, with only a small error. These errors are discussed in Experiment 2. To summarise, if the input has a linear frequency ramp, the maxima in the SNAC function appear at multiples of the centre frequency.

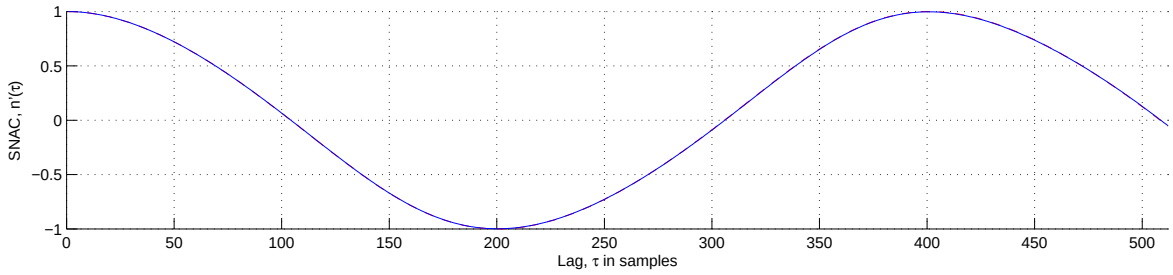
Figures 5.4(a) and (c) show what happens to the SNAC function of sine-wave-ramps with different centre frequencies. Figure 5.4(c) goes 2 octaves higher to 1760 Hz, while maintaining a one semitone range. At this higher frequency the peak drop off becomes more accentuated with the 5^{th} peak, at $\tau = 125$, dropping to 0.89. However, the first peak, at $\tau = 25$, the one of interest is still 0.99, and mantains correct alignment. In contrast Figure 5.4(a) goes 2 octaves lower to 110 Hz, while maintaining a one semitone range. Only one peak can be seen at this lower frequency, which is hardly affected by the linear frequency change. Note that a peak at $\tau = 0$ is always present with a value of 1, and is of no use - the peaks at $\tau > 0$ are the ones that matter.

In music, the size of a semitone, in hertz, is related to where the interval occurs, as decribed in Equation 2.1. In general to move from a given note, with frequency f_s, to the note one semitone higher, with frequency f_{s+1}, the following formula can be used[2]

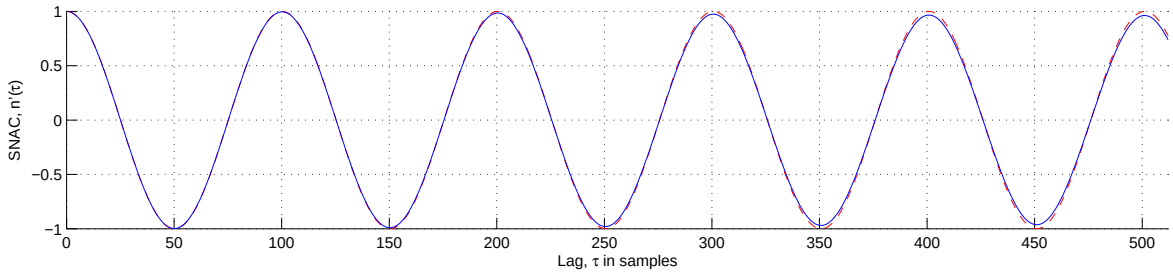
$$f_{s+1} = f_s * \sqrt[12]{2}. \tag{5.1}$$

It can be seen that no matter what the initial frequency, changing it by a given number of semitones only scales the frequency by a fixed ratio. Moreover, the period also scales by a fixed ratio, *i.e.* one over the frequency ratio, and hence the SNAC function is scaled (horizontally) by a fixed ratio. Therefore, at higher input frequencies, more peaks are seen in its SNAC function making the decay caused by phase cancellation more visible. This can be seen in Figure 5.4(c). The phase cancellation is caused because as the $n'(\tau)$

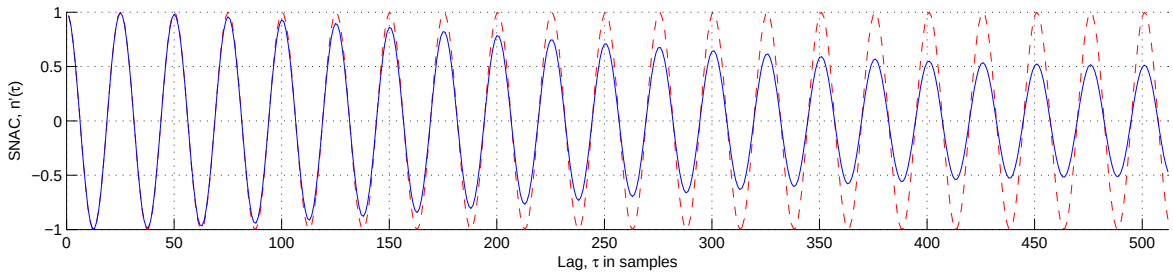
[2]This is on the even tempered scale. For other scales refer to Section 10.2.



(a) Centre frequency, 110 Hz



(b) Centre frequency, 440 Hz



(c) Centre frequency, 1760 Hz

Figure 5.4: The resulting SNAC function, from a sine wave changing linearly in frequency across one semitone in less than $1/40^{th}$ of a second (solid blue line), compared with the SNAC function from a sine wave of constant centre frequency (red dotted line). In each graph the input to the SNAC function has a different sine wave centre frequency.

values of higher peaks are calculated, the phase alignment at the edges of the window become further offset (although they will eventually loop around). This misalignment decreases the amplitude of the peaks with higher τ positions. Moreover, the positions of the peaks can become slightly misaligned, because the left-hand side of the correlation does not balance perfectly with the right-hand side when there is a frequency ramp. Because the phase misalignment is most prominent at the edges of the window, the use of a windowing function can have the effect of reducing the contribution from the edges. Experiment 2 shows us how significant an effect this has on the different algorithms.

Experiment 2a: Sine wave with linear frequency ramp.

In this experiment our attention is turned to comparing error rates of the different autocorrelation-type methods on linear frequency ramps. Note that the error is the difference between the frequency found and the centre-frequency of the ramp, as this is the frequency at the effective centre, t'_c, of the window, and is of interest to us. In the same way Figure 5.1 compared constant frequencies, Figure 5.5 shows a comparison between the SNAC function, unbiased autocorrelation and autocorrelation, windowed unbiased autocorrelation and WSNAC function, using inputs that have a linear frequency ramp. Note that a linear frequency ramp is not linear in the MIDI note number, however the ramp is set such that the total frequency change across the duration of the window is equal to one semitone. Also note that the fundamental period was found using the same peak picking method across all algorithms, described in Section 6.3.

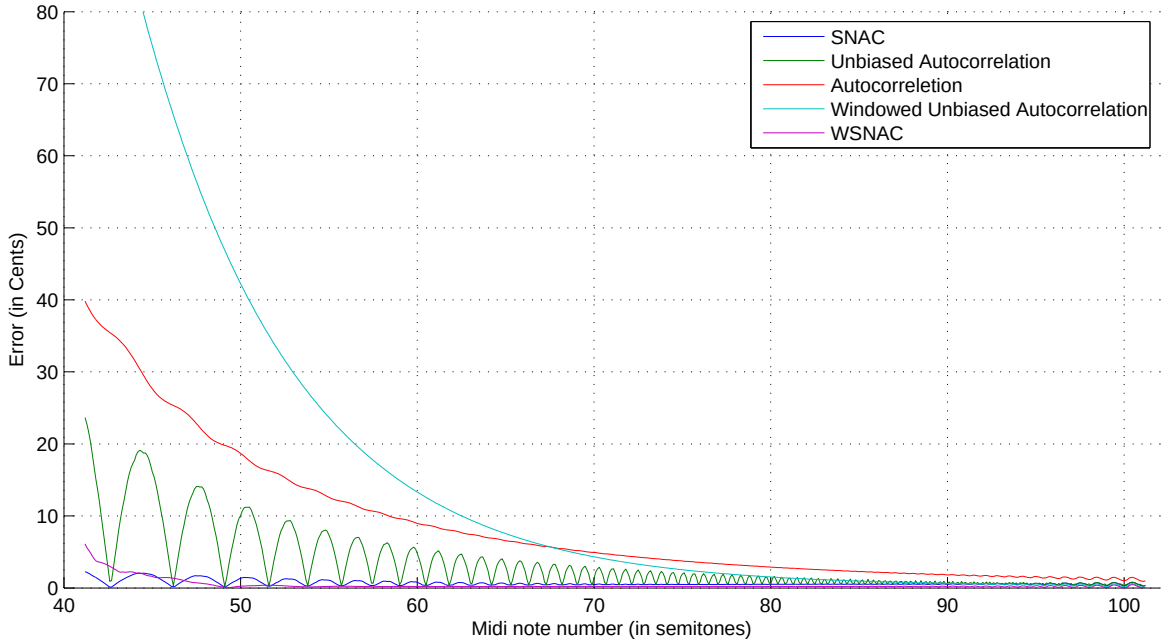


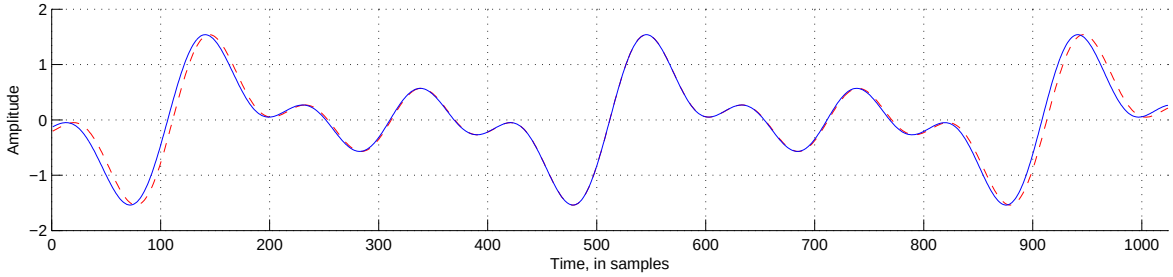
Figure 5.5: The accuracy of different autocorrelation-type algorithms in finding pitch on sine waves during a linear frequency ramp. The error is measured from the centre-frequency of the ramp. The freqency has been ramped one semitone about the centre frequency of each input window.

Figure 5.5 shows the SNAC and WSNAC functions to have significantly smaller errors than the other three, with the SNAC function keeping an error below 2.5 cents. However, it can be seen that the SNAC function has taken on a ripple effect like that of the unbiased autocorrelation, as it too is affected by whether there is a whole number

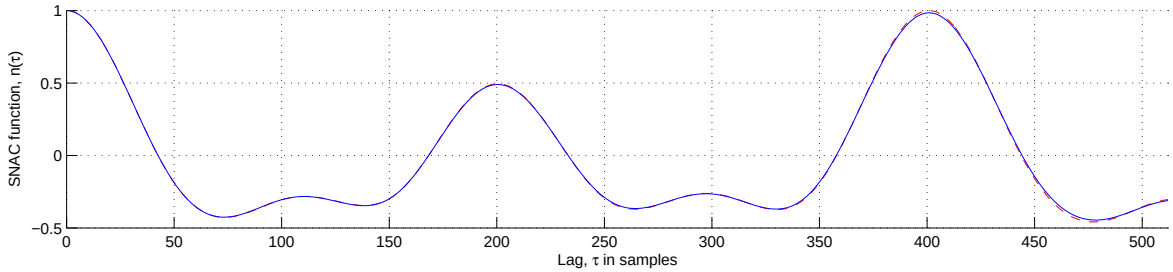
of periods or not when there is a frequency ramp. On the other hand the WSNAC function is more consistent than the SNAC function, at the expense of not having as small an error at the worst cases.

Experiment 2b: Complicated waveform with linear frequency ramp.

Let us now look at how a more complicated waveform is affected by a frequency ramp. Figures 5.6 through 5.8 each consist of 2 parts. Part (a) of each figure shows a waveform at constant frequency (red dotted line), and a ramped frequency (blue solid line). The frequency ramp covers a one semitone change over the 1024 sample window. Part (b) of each figure shows the corresponding SNAC function. Notice how the peaks drop off faster in Figure 5.8(b) with the higher frequency being ramped. However, the first peaks are still strong here.



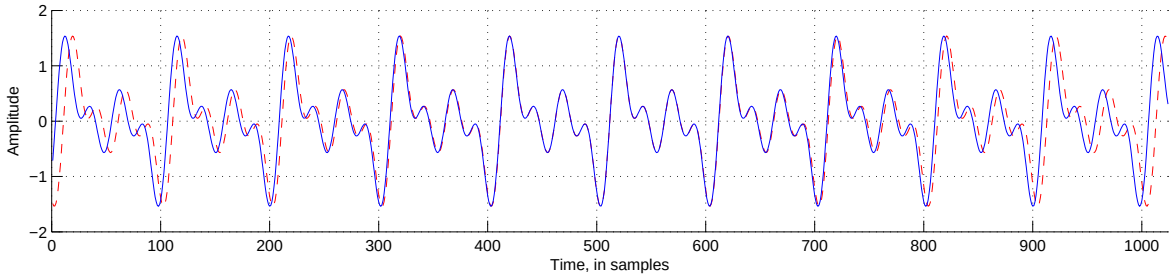
(a) A waveform at constant frequency (red dotted line) vs a waveform with a frequency ramp



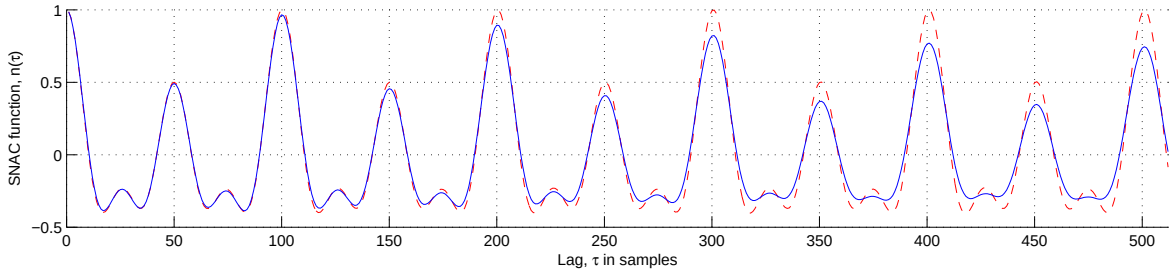
(b) The SNAC function of (a)

Figure 5.6: A 110 Hz constant waveform vs ramp

This experiment is the same as Experiment 2a, but differs only that the complicated waveform shape from Experiment 1b is used instead of a sine wave as input. Figure 5.9 shows the results with one such waveform shape. The exact shape of input waveform has a large effect on the output from all the functions. However, in general the SNAC function performs around a factor of two better than the others at lower MIDI numbers. The WSNAC function, besides exhibiting some acceptable errors at

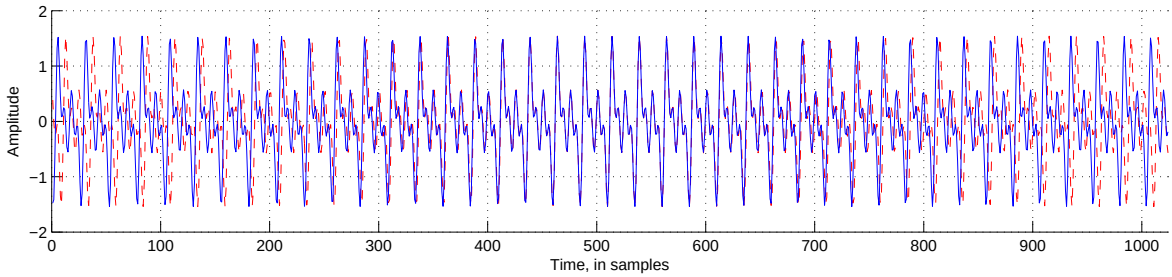


(a) A waveform at constant frequency (red dotted line) vs a waveform with a frequency ramp

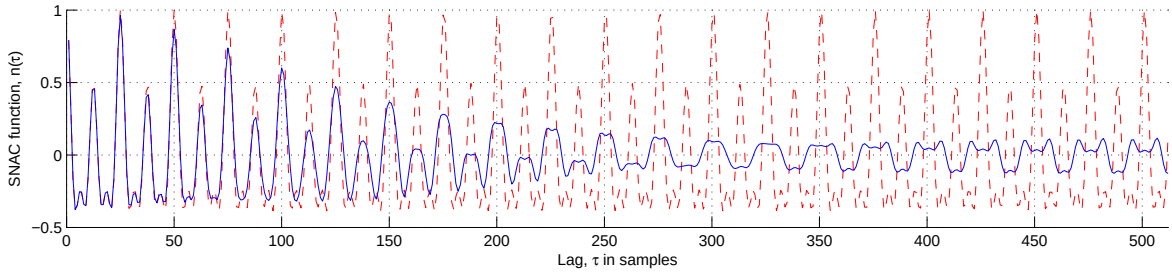


(b) The SNAC function of (a)

Figure 5.7: A 440 Hz constant waveform vs ramp



(a) A waveform at constant frequency (red dotted line) vs a waveform with a frequency ramp



(b) The SNAC function of (a)

Figure 5.8: A 1760 Hz constant waveform vs ramp

low MIDI notes, performs exceedingly well from MIDI note 48 onwards. Moreover, the windowed unbiased autocorrelation does not even form a peak to measure, for MIDI notes below 48.

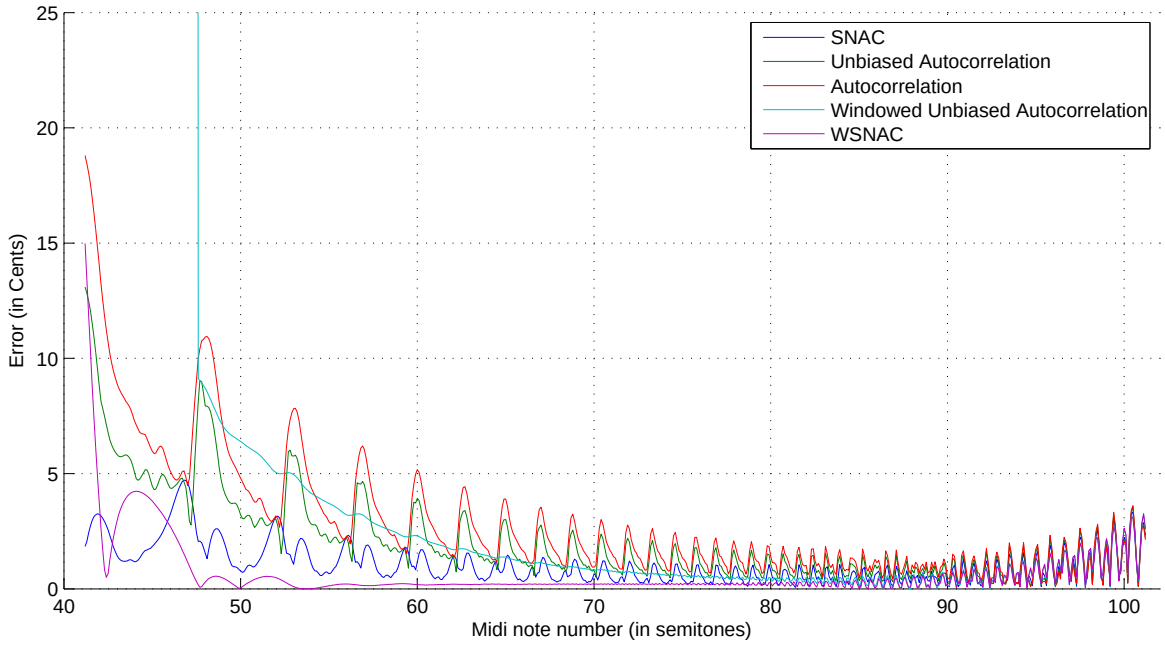


Figure 5.9: The accuracy of different autocorrelation-type algorithms in finding pitch on complicated waveforms during a linear frequency ramp. The error is measured from the centre-frequency of the ramp. The freqency has been ramped one semitone about the centre frequency of each input window.

5.2.2 Frequency Modulation, or Vibrato

A linear frequency ramp can be a good approximation to sliding pitch changes such as that during glissando. However, this next experiment looks at how well the different algorithms cope with rapid 'curved' changes in pitch, such as that during vibrato. A vibrato typically has a sinusoidal like pitch variation, which to a good approximation can be considered a sinusoidal like frequency modulation with variations of less than a couple of semitones.

Experiment 3: Vibrato

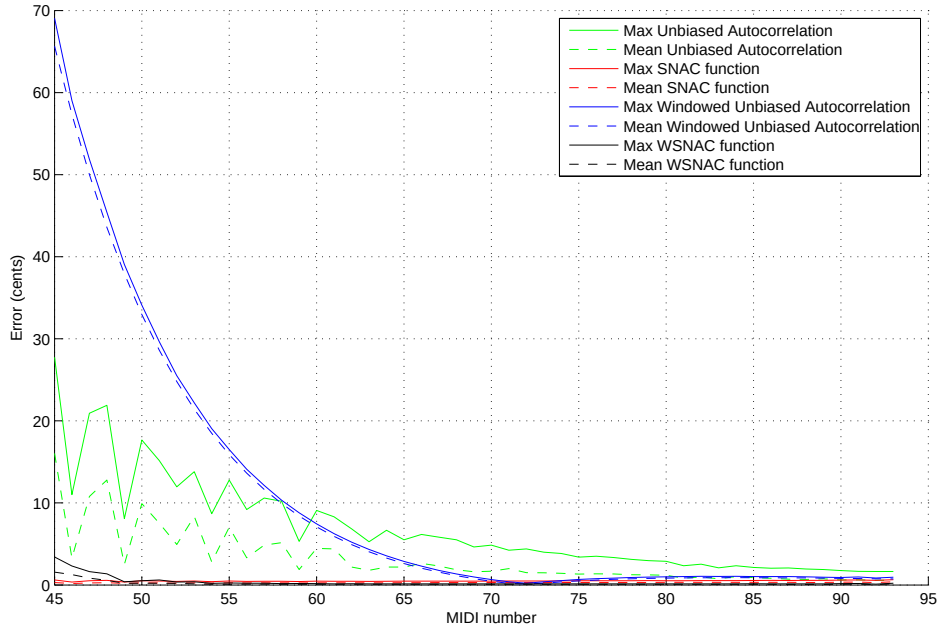
To test the error rates of the different autocorrelation-type algorithms, an experiment was set up in which the input was made with a one-semitone-wide vibrato. Again, there are two parts to this experiment: Part A, in which a sine wave is used, and Part

B, in which the more complicated waveform is used. The vibrato speed for both parts of the experiment is 5 Hz, a typical vibrato speed. A range of base frequencies were tested, ranging from MIDI number 45 to 93; a four octave range. To find the error at each base frequency, a 0.5 second sound stream was generated using the specified parameters at a 44100 Hz sample rate. Then for a given algorithm, a sliding window of size 1024 samples was used across the whole stream, with a step size of one. Note that the stream was made large enough for the window to fall on all possible positions along the vibrato. The average and maximum error from the stream is kept. The results of Experiment 3, Part A are shown in Figure 5.10(a), and Part B in (b). The autocorrelations are not shown to remove clutter.

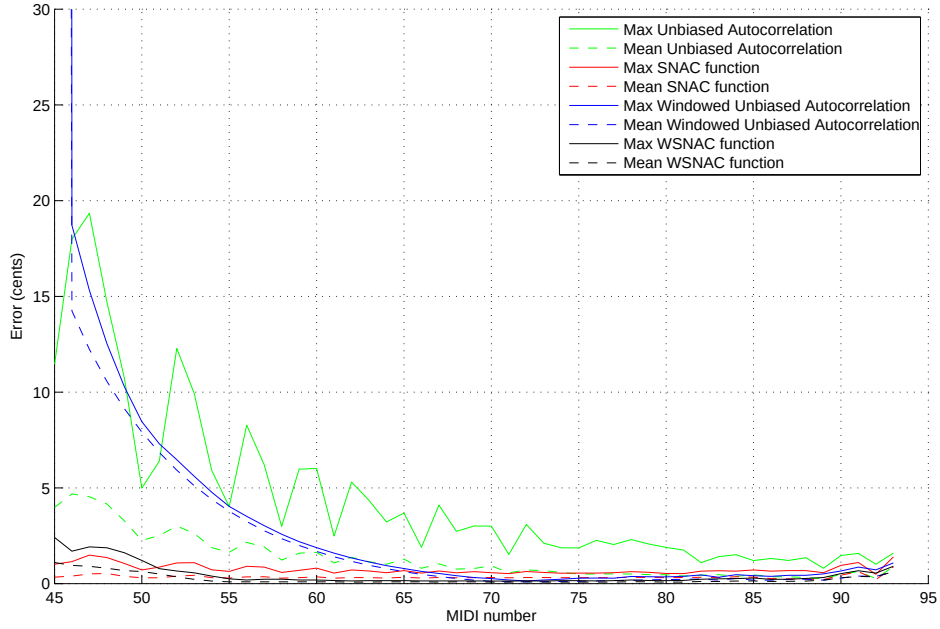
Figure 5.10(a) and (b) are very similar in nature. Interestingly the more complicated waveform resulted in smaller errors (note the different scales on the vertical axis of the two plots). This is due the higher harmonics causing narrower peaks in the lag domain, making it harder for the maxima to be dragged off to the side by other component influences, in all techniques.

Looking across both graphs, the windowed unbiased autocorrelation, performs poorly at lower MIDI numbers, but becomes very good by around MIDI number 65 or 70, corresponding to about 8 to 12 periods in the window. The unbiased autocorrelation and the SNAC function, both have maximum errors which are about twice the mean errors, indicating a bit of variation in the error measurements between different steps of the vibrato analysis. In comparison, the windowed unbiased autocorrelation is relatively stable. The SNAC function, even though it is subject to variation depending on the waveform shape, performs better than existing algorithms at the lower MIDI numbers (less than about 65), and does not perform much worse at MIDI numbers above that. Moreover, the WSNAC function performs exceptionally well overall, with the smallest error everywhere, except where it is surpassed by the SNAC function at very low MIDI numbers.

Recall, the goals of this thesis include trying to maximise the time resolution as well as pitch accuracy of the pitch algorithm. It can be seen from these results that the SNAC and WSNAC functions can give a higher pitch accuracy than the other autocorrelation-type methods with fewer periods of input signal. This will prove valuable indeed.



(a) Using a sine wave input with vibrato



(b) Using a complicated waveform input with vibrato

Figure 5.10: A comparison between autocorrelation-type methods at finding pitch with vibrato input. Here the input has a 5 Hz vibrato one semitone wide.

Changing vibrato width

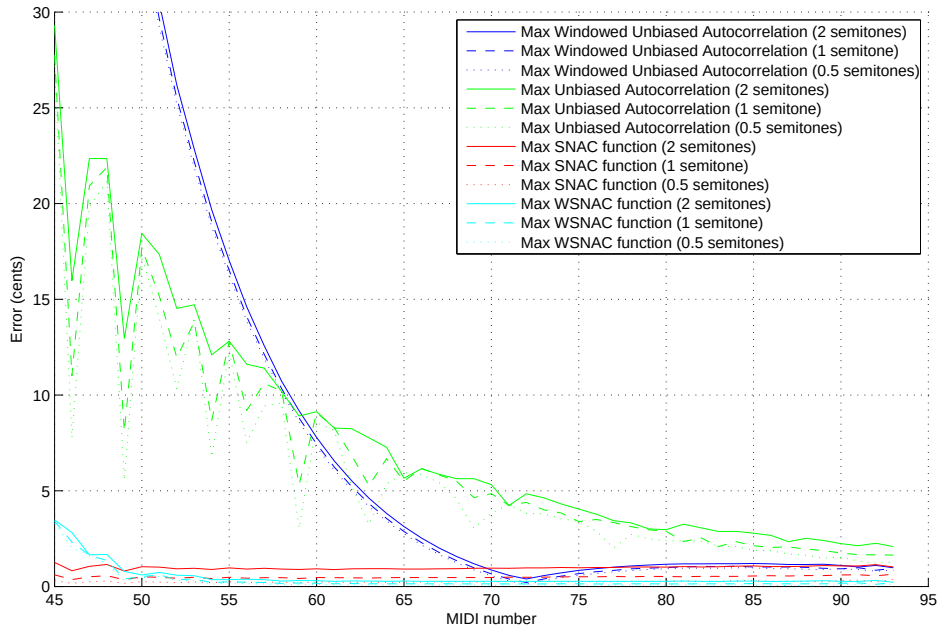
In Experiment 3, a vibrato width of one semitone was used, which was pretty reasonable. However, this next experiment is to see how the errors vary over different vibrato widths. We would expect a wider vibrato to cause greater errors on all methods, with the error curve scaling approximately linearly with the vibrato width.

Experiment 4: Changing vibrato width.

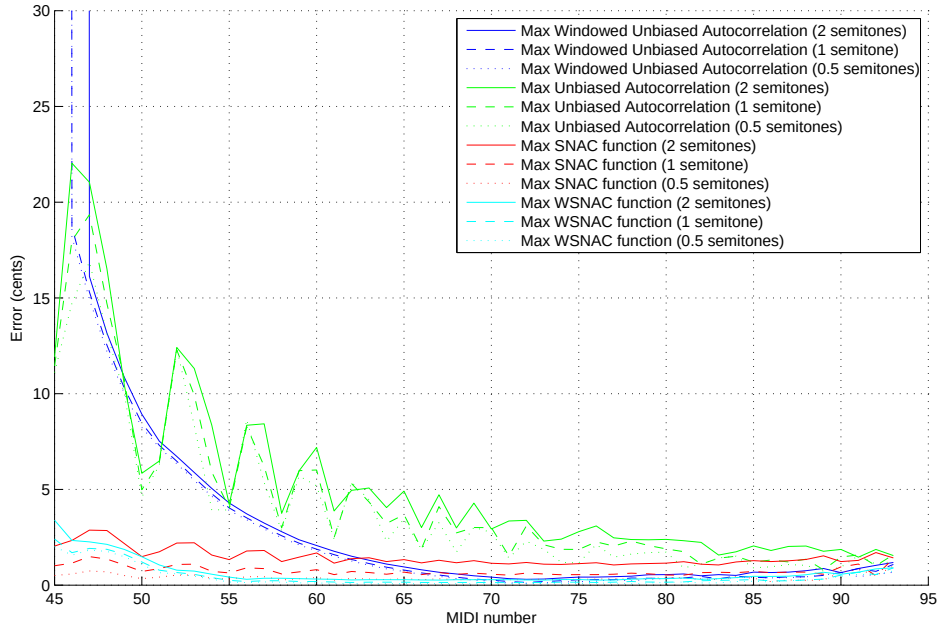
This experiment was performed in the same way as Experiment 3, except three different vibrato widths were used: 2 semitones, 1 semitone and 0.5 semitones. To minimise clutter, Figure 5.11 shows only the maximum error, at a given MIDI number, *i.e.* the error from the worst case position of the window in the vibrato sound stream. Figure 5.11 shows the result of the changing vibrato width, with (a) using a sine wave waveform input, and (b) the complicated waveform input.

Investigating Figure 5.11 it can be seen that windowed unbiased autocorrelation is very consistent between the different vibrato widths, although its average increase in error is only a few times smaller than the SNAC function, it looks insignificant because the error rate is already high. Also, this error curve does not scale linearly, however, the small increase from the 'no vibrato error' curve is approximately linear. The unbiased autocorrelation has quite a lot of variation, but does have a general trend of increasing its error as the vibrato width is increased. The SNAC function increases approximately linearly as the vibrato width is increased, as expected, and tends toward zero as the vibrato width tends to zero. Moreover, the WSNAC function, like the window unbiased autocorrelation, remains very consistent between the different vibrato widths, with only a very small, approximately linear, increase from the 'no vibrato error' curve.

Comparing Figure 5.11(a) and (b), the windowed unbiased autocorrelation decreases its error with the increased waveform complexity. This seems to be because the steepness of the waveform shape overpowers the windowing function curvature more in the lag domain, thus dragging the first peak closer to the correct place. Any fine scale errors that may exist are buried within the larger error. In contrast, the SNAC function increases its error a little, which seems to be because the narrower waveform peaks do not align as well to the desired period at this fine scale, and because the frequency change is not linear, the misalignments on either side of the window do not cancel very well. Hence the first peak tends to get dragged away from the correct place. The exact shape of the SNAC function curve can vary somewhat with different waveform shapes. However, in general it still has lower error rates that the other autocorrelation-type



(a) Using a sine wave input with vibrato



(b) Using the complicated waveform input with vibrato

Figure 5.11: A comparison between autocorrelation-type methods at finding pitch on input containing different vibrato widths. Here all the inputs had a 5 Hz vibrato, and a window size of 1024 samples was used with a 44100 kHz sample rate.

methods at lower MIDI numbers. However, as the vibrato width extends far beyond two semitones, the point at which the error curves cross over will tend further left.

In music a vibrato wider than two semitones is uncommon, so use of the SNAC function on general music is still desirable. However, use of the SNAC function for other frequency analysis tasks that involve fast 'relative frequency' changes may not be suitable. Nevertheless, the WSNAC function proves the best function so far, coping well with any width of vibrato.

Where are the greatest vibrato errors?

From investigating which step locations were causing the greatest errors, it appears that the greatest errors occur at the frequency maxima and minima of the vibrato, with this error pulling the result toward the centre pitch of the vibrato. This agrees with what could be expected, and is consistent with the idea that a linear frequency approximation does not describe the frequency curve very well.

Changing the window size

The next experiment tries to test our theory that using a shorter window size will improve the linear approximation to the vibrato curve, and hence reduce the errors on these signals that vary in pitch.

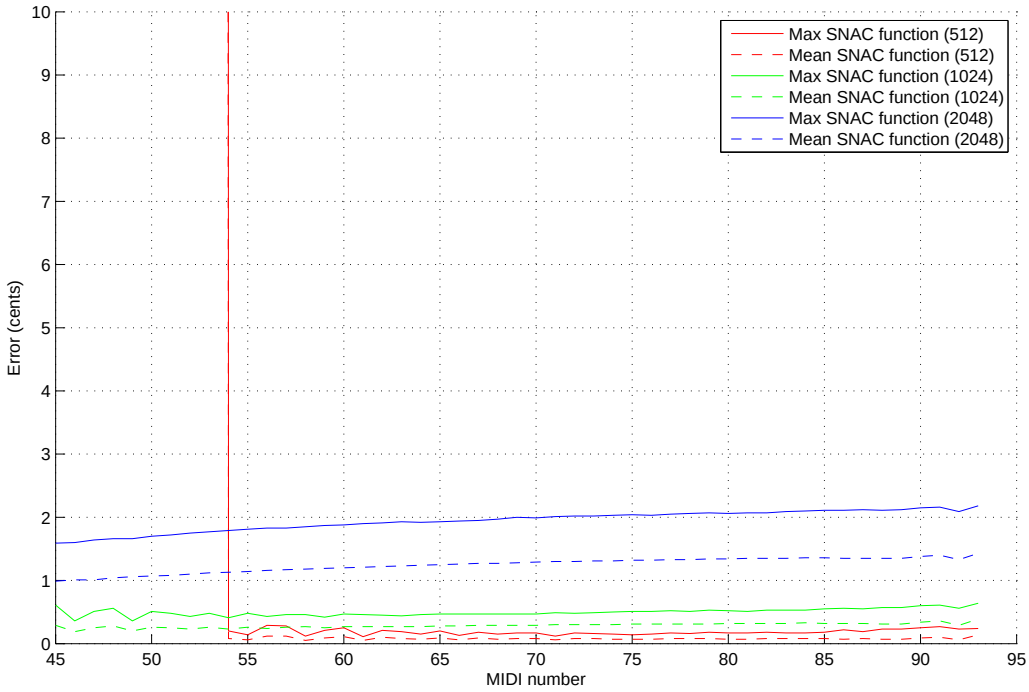
Experiment 5: Changing the window size.

This experiment is again similar to Experiment 3, except this time a range of different window sizes are used. To reduce clutter, each algorithm has its results plotted on a separate graph. The results are shown in Figure 5.12 (a) - (h); again each test is performed with the two sets of input: sine waves and complicated waveforms. Note that all the graphs are set to the same scale to allow for easy comparison.

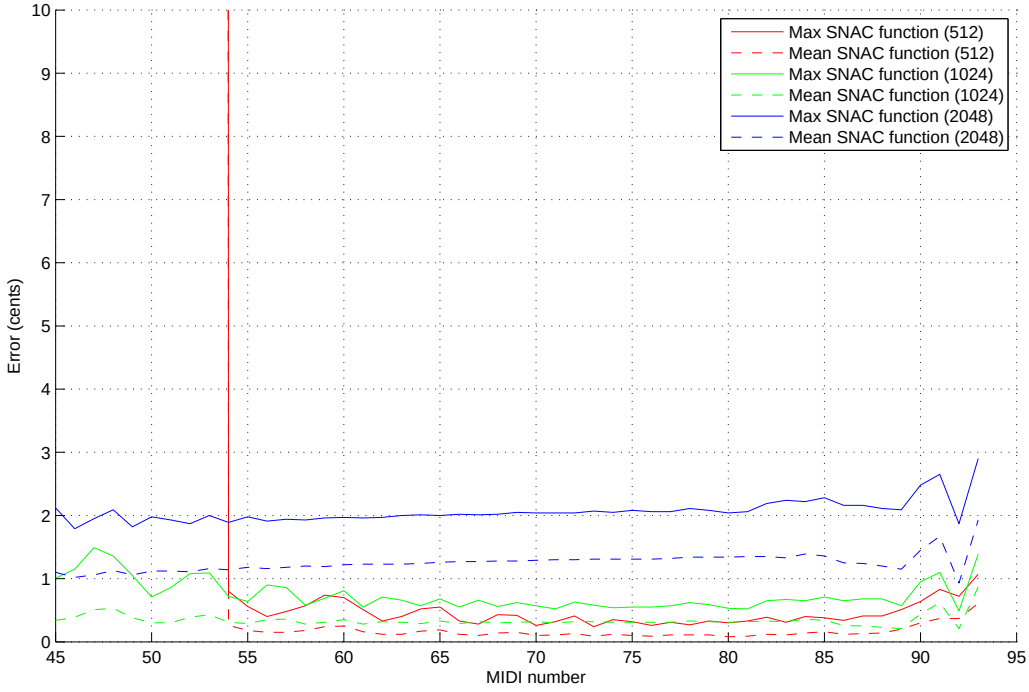
However, the window size can only be shortened reliably provided there will be at least 2 periods within the shortened size, otherwise the the algorithm can become unstable. This effectively will reduce the range of MIDI numbers we are able to find by 12 each time the window size is halved[3]. This can cause the fundamental's peak to move out of the lag domain analysis, and the wrong peak or no peak to be chosen. This is shown in the graphs by error values leaping off the top of the scale.

It can be seen for the SNAC and WSNAC functions that as the window size is decreased the accuracy increases, at frequencies for which results can be found. How-

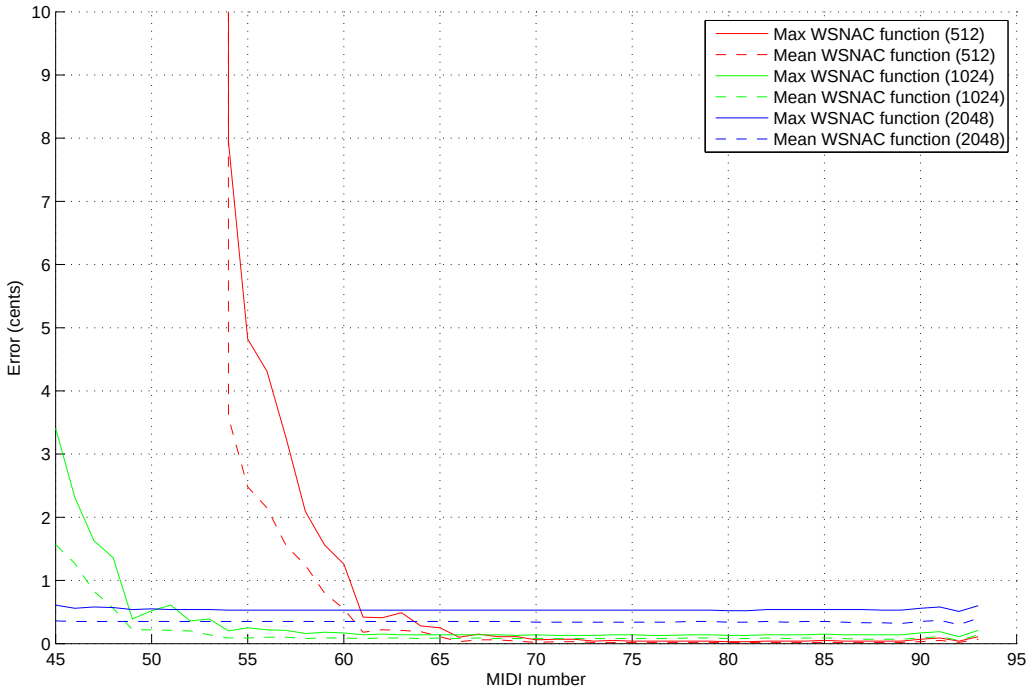
[3]There are 12 MIDI numbers one octave



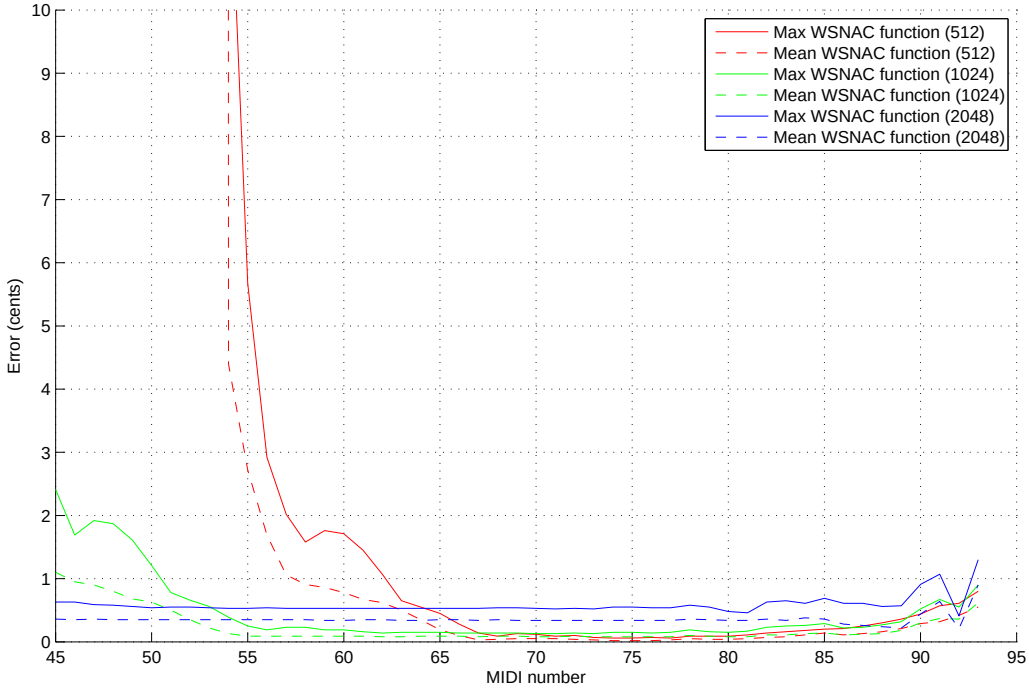
(a) SNAC function; sine wave input



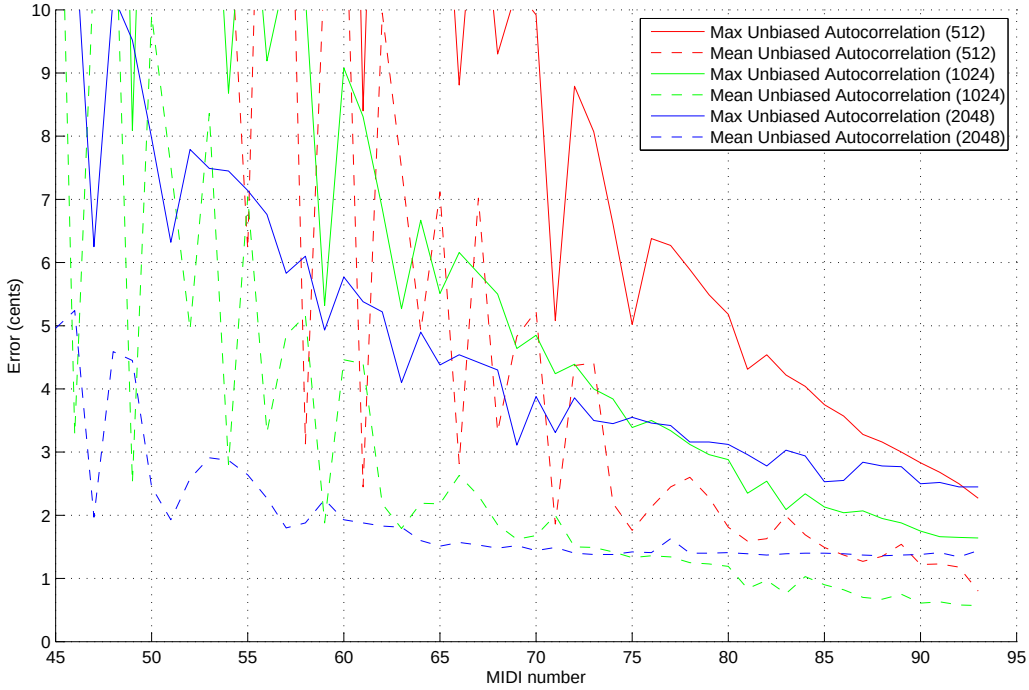
(b) SNAC function; complicated waveform input



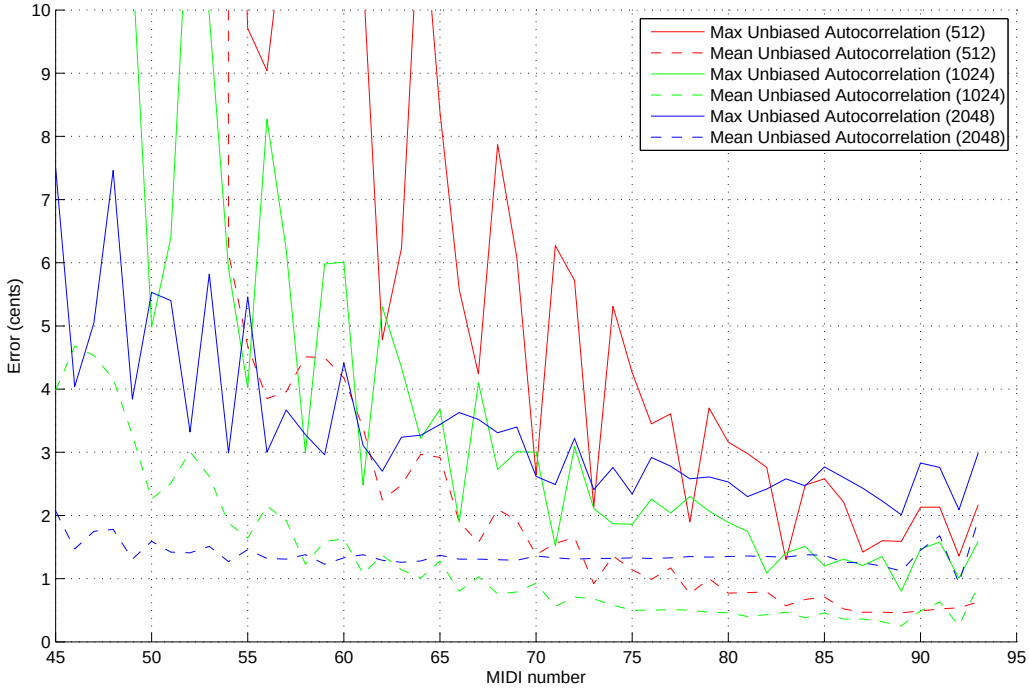
(c) WSNAC function; sine wave input



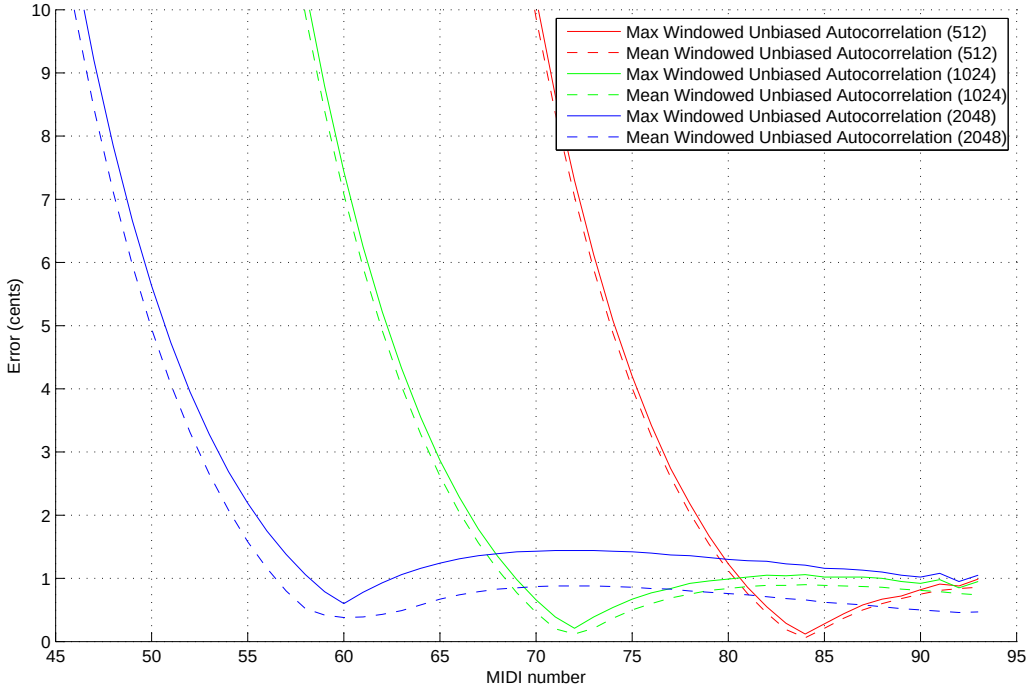
(d) WSNAC function; complicated waveform input



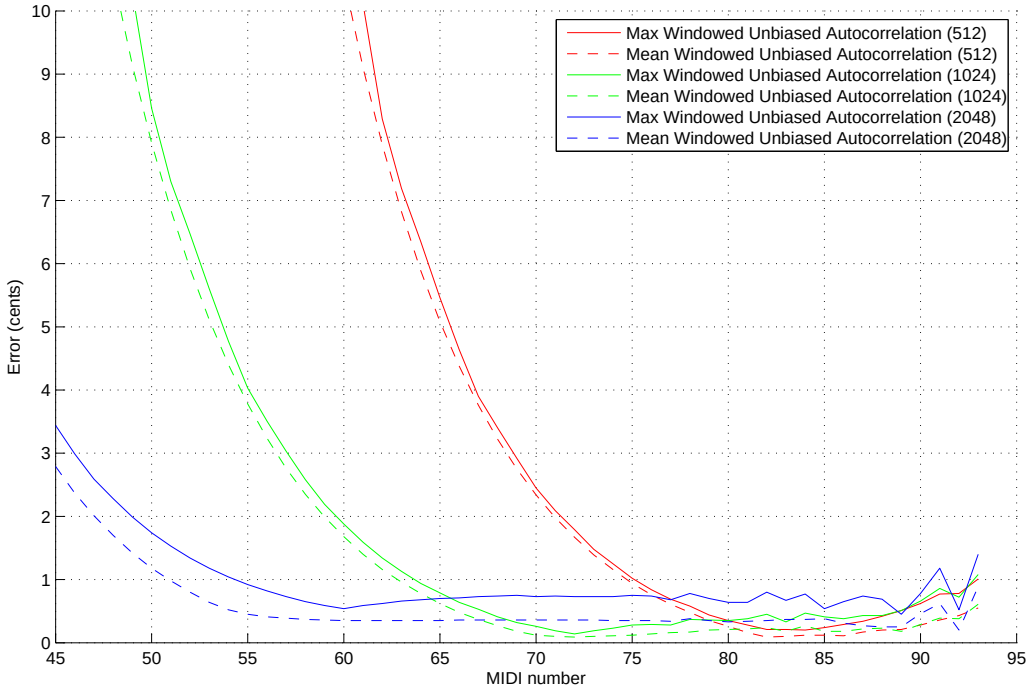
(e) Unbiased autocorrelation; sine wave input



(f) Unbiased autocorrelation; complicated waveform input



(g) Windowed unbiased autocorrelation; sine wave input



(h) Windowed unbiased autocorrelation; complicated waveform input

Figure 5.12: A comparison between autocorrelation-type methods at finding pitch using varying window sizes in the analysis. Here the input has a 5 Hz vibrato, and a vibrato width of 1 semitone was used on a 44100 kHz sample rate.

ever, the WSNAC takes about 12 MIDI numbers (twice the number of periods) from the first reading to achieve the same error as the window size twice as long, whereas the SNAC function achieves this straight away. This supports the idea that having a smaller window size improves the linear approximation of the pitch curve.

The results of the other algorithms do not necessarily support this idea, but this seems to be due to the reduction in the implicit errors of the algorithms as the window size is increased. Because the change in implicit errors is large, it seems to override the improvement in the pitch curve approximation.

In summary, the graphs show that the SNAC function can be used with a good time resolution and good frequency resolution at the same time. However, in order to choose the window size appropriate for minimising the error, one needs to have an idea of what the frequency is. Otherwise, if a window is chosen that is too small, then the possibility of finding a larger period is lost. Our method of dealing with this is to first use a bigger window to get an approximate fundamental frequency, and then perform the analysis again with a more optimal window size for improved accuracy. This is discussed in more detail in Section 8.1.

5.3 Amplitude Changes

The frequency of a waveform is not the only parameter that can be changing throughout the course of a window. The following experiments look at how amplitude changes in the signal affect the pitch detection accuracy of the autocorrelation-type algorithms. Again, linear ramps are investigated first, however this time it is the amplitude is that is increasing. It is worth noting the symmetry of the autocorrelation-type methods used here, meaning that reversing the input has no effect. Therefore any testing of an increasing linear ramp is the same as testing a decreasing linear ramp with the waveform reversed.

Trying to construct sensible experiments to test how amplitudes change in real music is a little tricky. For example, at the beginning of a note, the note onset, the amplitude of the sound can go from almost zero, up to a high value in only a few samples. This case is called the amplitude step, and is discussed in Section 5.3.2.

5.3.1 Linear Amplitude Ramp

Throughout the duration of a musical note, amplitude variations come about in typical usage, such as tremolo, crescendo and the decaying of a released note. These amplitude

changes can be approximated by a set of linear amplitude ramps over a series of short time intervals.

Experiment 6a: Sine wave with linear amplitude ramp.

This experiment takes a series of sine waves that vary linearly in amplitude across the window. The amplitude ramps are tested at different levels of steepness, and the effect on the autocorrelation-type algorithms' ability to correctly determine pitch is measured. Figure 5.13 shows an example of a sine wave with a linear amplitude ramp that is used as input to the experiments.

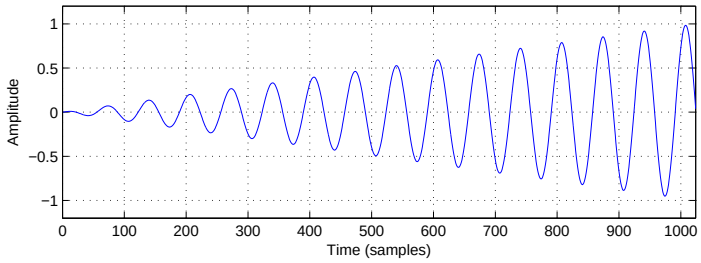
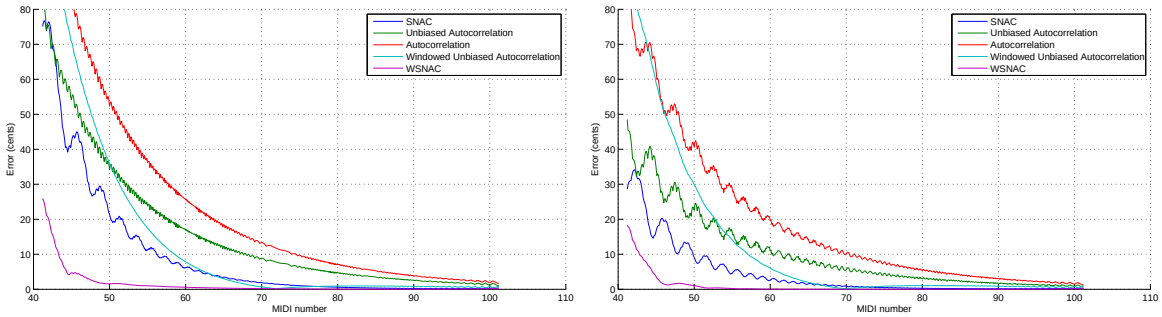


Figure 5.13: An example of a linear amplitude ramp function applied to a 660 Hz sine wave. This ramp ranges from 0 to 1 across the window. Note that the vertical axis represents the relative pressure of the sound wave.

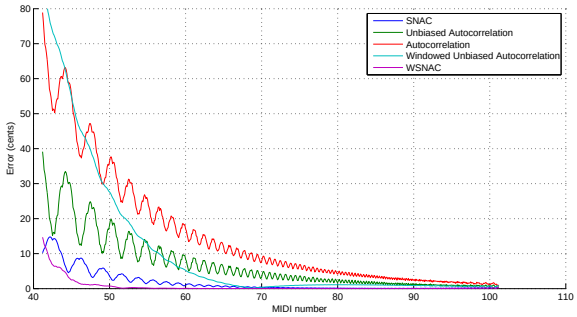
A series of different linear amplitude ramps were tested, each with a different steepness. The ramps are described by their beginning amplitude and ending amplitude. The ramps tested is this experiment were 0 to 1, 0.25 to 1, 0.5 to 1, 0.75 to 1 and 1 to 1. The results are shown in Figure 5.14. Note that the last test case is in fact the stationary signal case giving the same result as Figure 5.1. As in Experiment 1, for each frequency tested, the sine waves are generated at a series of different phases, and the pitch is calculated using the parabolic peak method, and only the maximum pitch error is shown. A window size of 1024 was used and a sample rate of 44100 Hz, hence the number of cycles of the waveform within the window's ramp increases from 2.05 at MIDI note 41.2 to 65.63 at MIDI note 101.2.

The results in Figure 5.14 show that as the steepness increased from flat, at 1 to 1, to the most steep, at 0 to 1, the errors increased slightly for all methods, with the unwindowed methods being affected the most. The SNAC function is affected the most, going from almost no error to an error of 21 cents at MIDI note 50; almost matching that of the windowed unbiased autocorrelation. The WSNAC function, although worsening, remains the best across the full range, showing that it has great ability to determine

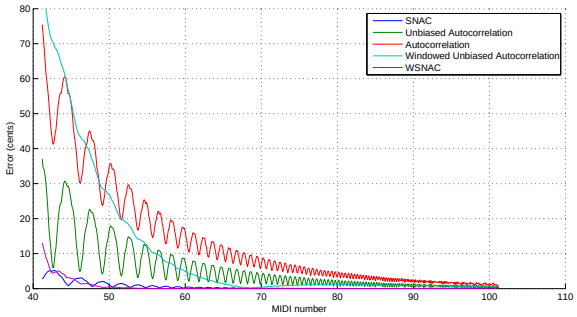


(a) Ramp from 0 to 1

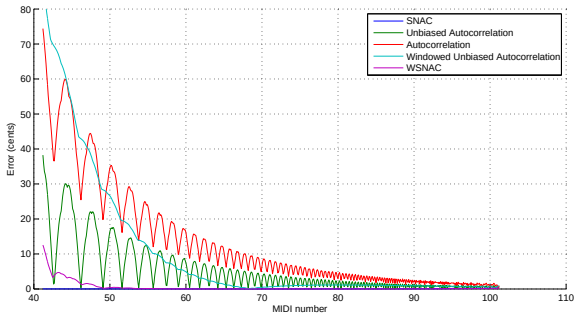
(b) Ramp from 0.25 to 1



(c) Ramp from 0.5 to 1



(d) Ramp from 0.75 to 1



(e) Ramp from 1 to 1

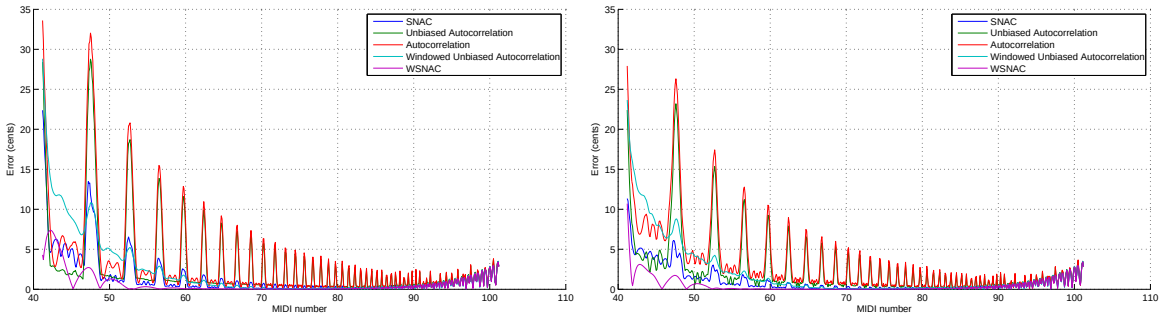
Figure 5.14: The pitch errors measured using different autocorrelation-type methods on sine waves with different linear amplitude ramps and varying input frequencies.

pitch even in the presence of a changing amplitude.

Experiment 6b: Complicated waveform with linear amplitude ramp.

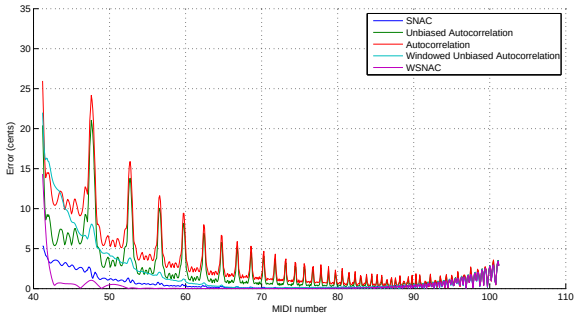
This experiment is the same as Experiment 6a, except the more complicated waveform is used. The results are shown in Figure 5.15. Note that the vertical scale used in these graphs is different from the previous figure.

Comparing Figures 5.14 and 5.15, the overall error remains lower for the complicated waveform than for the sine wave as discussed in Section 5.2.2, due to the higher

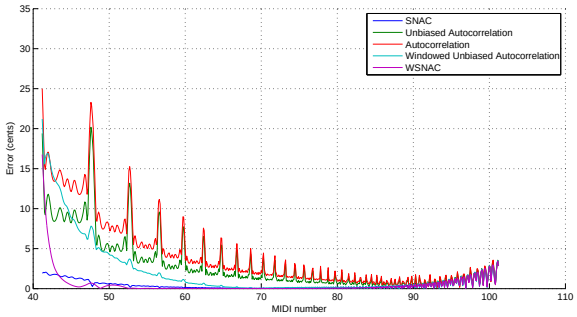


(a) Ramp from 0 to 1

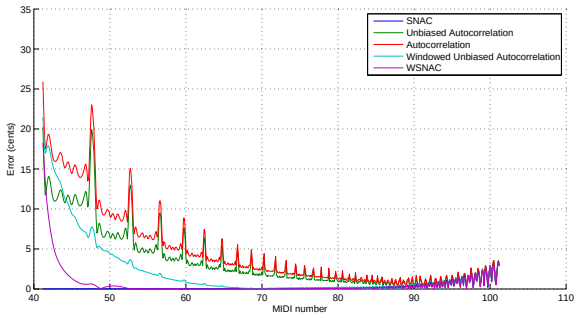
(b) Ramp from 0.25 to 1



(c) Ramp from 0.5 to 1



(d) Ramp from 0.75 to 1



(e) Ramp from 1 to 1

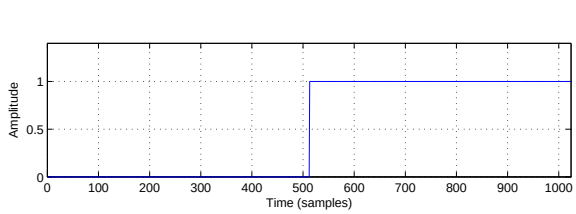
Figure 5.15: The pitch errors measured using different autocorrelation-type methods on complicated waveforms with different linear amplitude ramps and varying input frequencies.

frequency components creating narrower peaks in the lag domain. The graphs in Figure 5.15 show the usual peaks in the non-windowed methods caused by the non-integer number of periods within the input window. Moreover, the difference in error between the peak and troughs becomes quite significant. However, for the SNAC function, this only starts taking effect at a 0.25 to 1 ramp steepness. In contrast, the WSNAC function stays relatively smooth, and it has the most consistently low error rate for this complicated waveform.

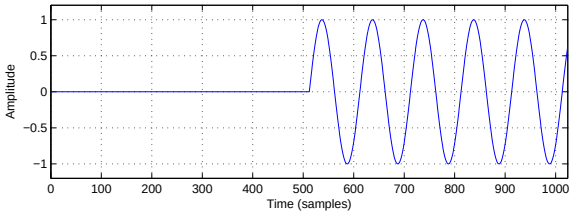
5.3.2 Amplitude Step

Experiment 7a: Sine wave with amplitude step.

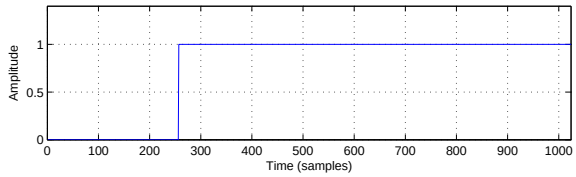
This experiment tests how well the different autocorrelation-type algorithms deal with an amplitude step. That is, a segment of sound with zero amplitude (no volume), that instantaneously jumps to sound of a constant amplitude level. This is to simulate a fast note onset. The test here is to see how long the algorithms take to respond to give an accurate reading of the new note pitch. Because all algorithms here are estimating the pitch at the centre of the window, the step at 50% of the way through the window, is the point at which we should first possibly expect to get a sensible pitch reading. Figure 5.16 shows an example of a step at 50% and a step at 25% and how it applies to a sine wave.



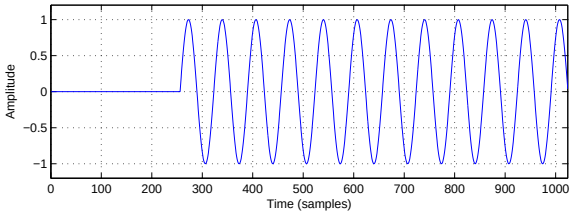
(a) A step at 50%, amplitude graph



(b) A 440 Hz sine wave with a step at 50%



(c) A step at 25%, amplitude graph

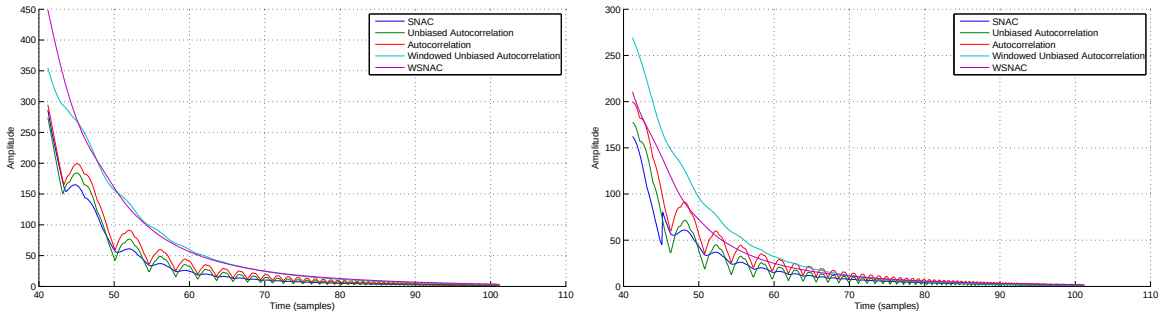


(d) A 660 Hz sine wave with a step at 25%

Figure 5.16: An example of an amplitude step function. The steps here are at 50% and 25% of the way through the window.

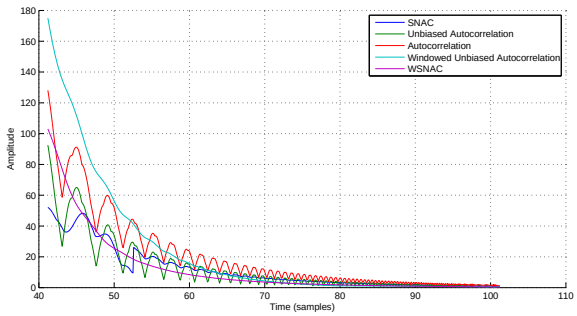
Figure 5.17 shows the results of the amplitude step experiments. Note the change in vertical scale between plots. The step positions tested are 50%, 37.5%, 25%, 12.5% and 0%. Note that having the step at 0% means that the signal takes up the whole window and has become the stationary signal case, so is the same as in Figure 5.1.

The first thing to notice is the errors get much larger for all methods as the step approaches 50%, with errors ranging up to 400 cents. Whereas, with a step at 0% the largest error is around 90 cents. Both windowed methods perform worst with the step at 50%, although the non-windowed methods do not perform much better. However,

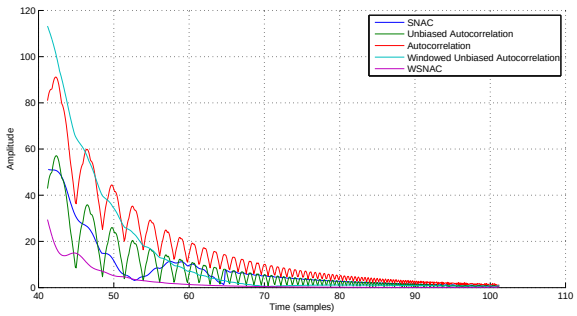


(a) Step at 50%

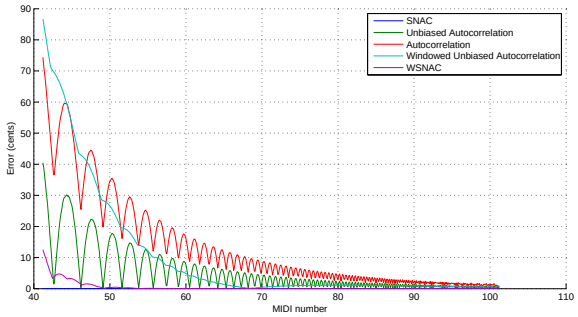
(b) Step at 37.5%



(c) Step at 25%



(d) Step at 12.5%



(e) Step at 0%

Figure 5.17: The pitch errors measured using different autocorrelation-type methods on sine waves with different step positions and varying input frequencies.

with a step at 12.5%, the WSNAC can still be classified the best method. As usual the the errors are smaller at higher MIDI numbers, as there are more periods of the waveform in the window.

Experiment 7b: Complicated waveform with amplitude step.

This experiment is the same as Experiment 7a, except the more complicated waveform shape is used for input. The results are shown in Figure 5.18. These plots are quite chaotic, and will vary greatly depending on the exact nature of the waveform

shape. In general low MIDI numbers with steps of 37.5% or greater produce large errors for all methods. The only sign of stability and low errors with the step at 12.5% comes from the WSNAC function; however with the step at 25% it too performs poorly. In comparison, the SNAC function's performance appears to be one of the worst with the step at 12.5%.

With the step at 50%, any MIDI numbers less than 52 contain less than two periods of actual signal, so it can be expected that the error can just fluctuate wildly, as there is no stability down there. It can be seen that all methods get an error around 400 cents at some point around MIDI number 44.

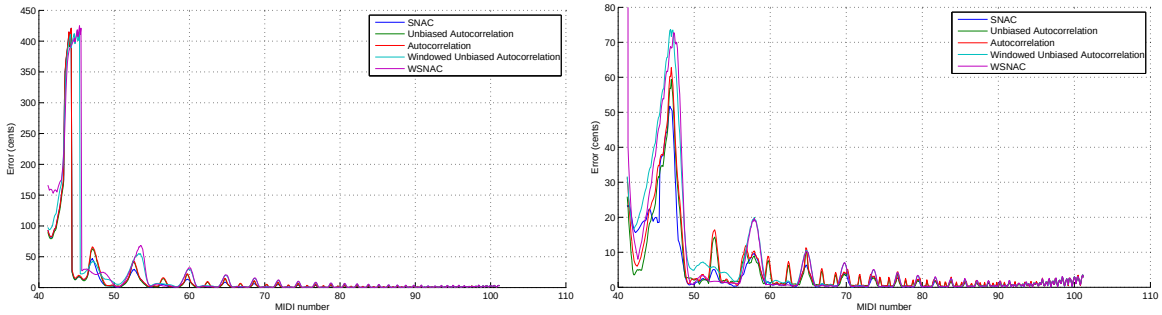
The conclusion to be drawn from the amplitude step experiments is that the window should be made up of around 85% waveform before the low MIDI numbers can consistently achieve low errors using any method. In that case, the WSNAC function performs the best. From another perspective, a delay of 35% of the window size is required before a reasonable pitch value can be determined at the start of a fast onset note. For example, on the 1024 sized window used here with a 44100 Hz sample rate, the first 0.008 seconds of the note could not be assigned an accurate pitch measurement.

Of course, a human cannot instantaneously detect the pitch of a note, either. Rossing [1990] states that a pitch can be determined in times as short as 0.003 seconds for tones that have a slow onset, but requires longer for an abrupt starting sound, like the amplitude step described here. The amplitude steps tend to give more of a clicking sound. In practice, most musical instruments take longer than 3 ms for their attack, so the amplitude steps described are a bit extreme, making it reasonable to accept the higher error rates found. This seems to show that the linear amplitude ramp makes for a better approximation of a note attack/onset.

5.4 Additive Noise

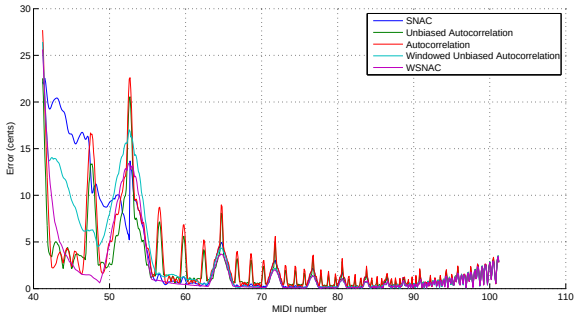
Experiment 8: Added noise.

This experiment is similar to Experiment 1, with the frequency and amplitude of the input remaining constant, except that this time 15% white noise is added to the input, *i.e.* random values ranging between -0.15 and 0.15. The results, shown in Figure 5.19, show a decrease in the accuracy of all methods relative to Figure 5.1. However, the SNAC and WSNAC functions remain more accurate than the others, especially at the lower MIDI numbers. Note that the peaks are still very prominent in the lag

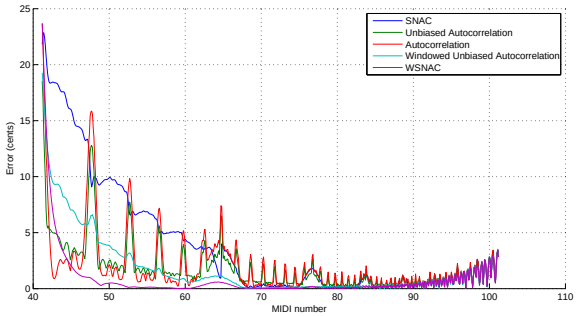


(a) Step at 50%

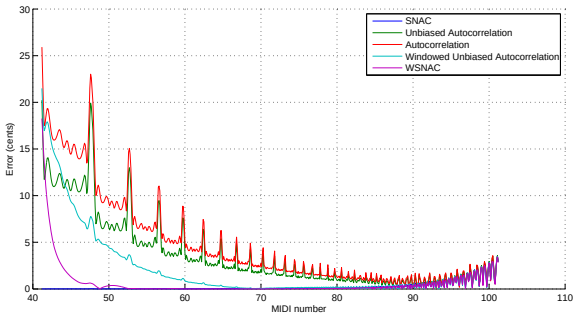
(b) Step at 37.5%



(c) Step at 25%



(d) Step at 12.5%



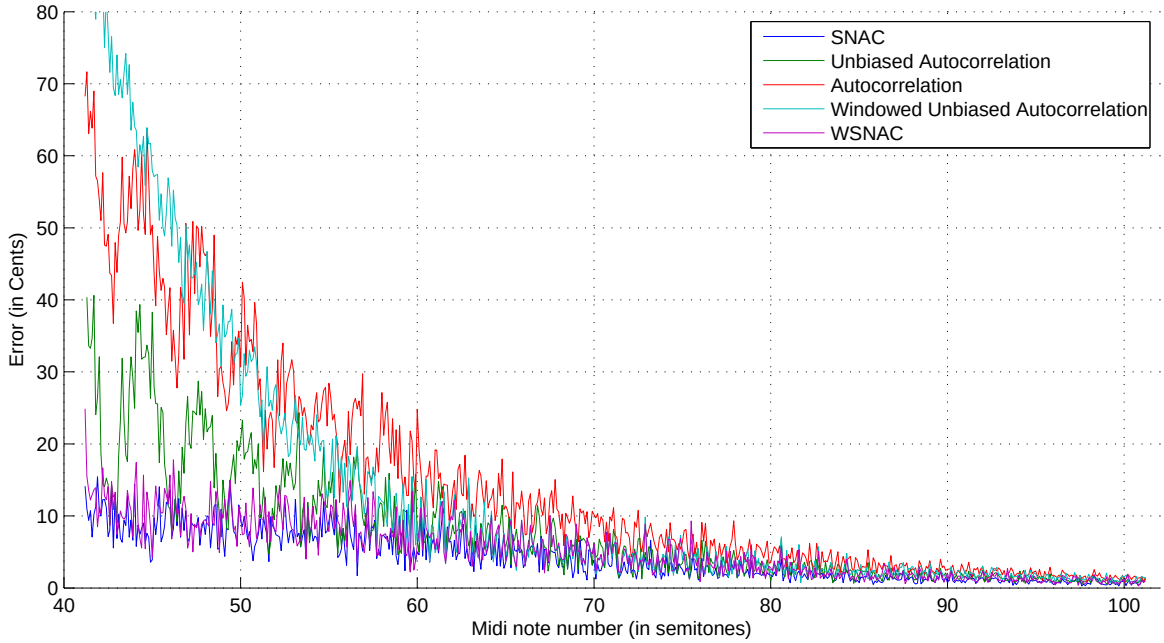
(e) Step at 0%

Figure 5.18: The pitch errors measured using different autocorrelation-type methods on complicated waveforms with different step positions and varying input frequencies.

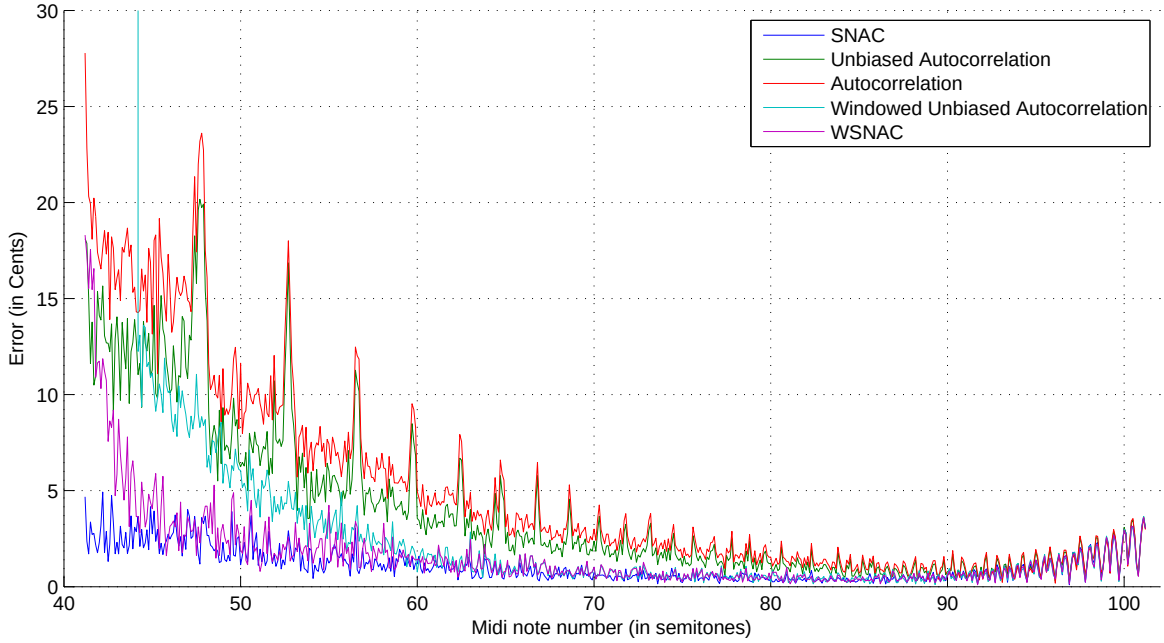
domain and can be found easily above the noise: the increase in error is due to the inability to find the exact top of the peak, or that the peak has actually moved.

5.5 Summary

The experiments described throughout this chapter have tried to simulate the type of variations that are typical in music, and that might be expected from a user. The inputs, however, have been constructed and controlled in such a manner that the



(a) Using a sine wave with 15% white noise added



(b) Using a more complicated waveform with 15% white noise added

Figure 5.19: The pitch errors, measured error using different autocorrelation-type methods on signals with added white noise.

desired outcome is known, and accurate error measurements can be made.

In general, when using a complicated waveform, *i.e.* with a number of harmonics, instead of a sine wave, the errors were actually smaller. This seems to be because the waveform has steeper slopes, and produces lag domain peaks that are narrower in width. This can in fact improve the parabola peak finding. However, when the higher MIDI numbers are used in conjunction with the complicated waveforms, errors start being introduced again. This is because the component frequencies become very high, making a peak's width in the lag-domain too narrow. With these very narrow peaks, the three point parabola peak fitting method does not make as good an approximation, and as a result errors start being introduced. This effect can be seen in Figures 5.2, 5.9, 5.15 and 5.18 as a small increase in the error rate from about MIDI numbers 90 and above. Nevertheless, in real music, the instruments that can play really high notes, such as the piccolo, do not tend to produce many harmonics, but sound quite pure and produce sinusoidal-like shapes. Therefore, the type of peak fitting problems just described do not usually occur in practice. However, in the extreme case that this effect does arise, one solution to this problem is to use a higher sampling rate, although this would result in more computation. Another approach could be to try using a more complicated peak approximation; however, this may not necessarily produce a better result.

In summary, across all the experiments carried out, including frequency changes, amplitude changes and additive noise, the WSNAC function has remained the most consistently well performing algorithm, with the SNAC function also performing well in a lot of cases. It was also shown that these two algorithms can work well at small window sizes, requiring little over two periods of the waveform in order to detect the pitch accurately to less than 5 cents in most cases. The exceptions were the amplitude step experiment, which might be described as too extreme a test for any sensible pitch to be found anyway, and the additive noise experiment, at which they could only achieve around 10 cents accuracy or less. However, they performed better than the other autocorrelation-type methods in this case, and getting more accurate than this may not even be possible, due to some of the signal's information being inherently lost in the localised noise.

In this chapter, because of the controlled generation of the signals, it was possible to choose the correct peak during the peak-picking process. However, in real musical signals, a waveform's shape can vary over the duration of a note due to the harmonics changing differently. This makes picking the correct peak difficult in practice. Never-

theless, if the correct peak is chosen, the experiments from this chapter remain valid. However, even using a suitably sized window such that the waveform's shape is approximately unchanged throughout, these results will only serve as a guideline to the accuracy that will be achieved on real musical signals. Chapter 6 describes testing using real musical signals, creating a more complete testing regime.

Chapter 6

Choosing the Octave

The previous chapter looked at a number of different functions and compared the errors in their estimates in the position of a peak's maximum in the lag domain. Knowledge of the expected position was used to guarantee that the correct peak was measured.

This chapter looks at the problem of how to choose which peak in the autocorrelation-type function corresponds to the pitch's octave as perceived by a listener. Section 6.1 discusses how to go about testing an algorithm and the datasets that were constructed to perform these tests. Section 6.2 discusses the difference between the fundamental frequency and the pitch frequency and how these are not always the same. This is followed in Section 6.2.1 by a brief look at how humans perceive sound, with aspects learnt from this applied in Section 6.2.2. The shape of the SNAC function is investigated in Section 6.3, with a simple method for choosing a peak presented. An alternative method for choosing the pitch's octave, based on the cepstrum algorithm, is discussed in Section 6.4 which can be used in conjunction with the SNAC function to achieve a good measurement of pitch.

6.1 Measuring Accuracy of Peak Picking

In order to measure the accuracy of an algorithm, a dataset of sounds with known pitch outcomes is required. No good standard databases for testing pitch could be found; however, a good database of sounds was found from Lawrence Fritts at the University of Iowa[1].

The Iowa database contains sounds from a range of musical instruments which include:

[1] http://theremin.music.uiowa.edu

- String - double bass, cello, viola, violin, piano.
- Woodwind - bassoon, oboe, bass clarinet, $E^\flat$ clarinet, $B^\flat$ clarinet, alto saxophone, alto flute, soprano saxophone, bass flute, flute.
- Brass - bass trombone, tuba, french horn, tenor trombone, $B^\flat$ trumpet.

This is an excellent test set which contains samples of the full range of notes from each instrument, with each note played distinctly with a gap between them. The sounds were recorded in an anechoic chamber and hence contain almost no reverberation, and very little background noise. Note that the cello, violin and viola recordings contain pizzicato, or plucked, notes as well as bowed versions. Also, the alto saxophone, soprano saxophone, trumpet and flute recordings contain both vibrato and no vibrato versions of the notes. The file-names labelled the content sufficiently clearly that the hand marking of pitches and beginning/end of notes using listening and inspection methods could be applied with confidence.

A second dataset of sounds was generated using MIDI, including a 3 octave chromatic scale, and the first half of 'Für Elise'. One noticeable difference between this dataset and the Iowa dataset is that in this dataset the notes are played with legato *i.e.* without gaps. The program TiMidity++ version 2.13.2 [Izumo and Toivonen, 2007] was used to convert MIDI to wave files. The command used to run TiMidity++ was "timidity -OwM –reverb=d,0 -o [outfile.mid] [infile.wav]". The second parameter is used to reduce the amount of reverberation. The great thing about using MIDI is that the sound of many different instruments can be created by simply changing the instrument number in the MIDI file, and leaving the hard work to the MIDI converter. The other reason is that the expected pitch output is relatively well known, although it appears that small variations in pitch are introduced by TiMidity++. However, the details of sound generation were intentionally left as a black box, so as not to influence the testing phase. Each of the sounds was generated using the following MIDI instruments:

- String - cello, violin, piano, nylon string guitar, steel string guitar, clean electric guitar and overdrive electric guitar.
- Woodwind - bassoon, clarinet, flute, oboe, recorder and soprano saxophone.
- Brass - tuba, trombone and trumpet.

The resulting generated sounds are, however, not as good as real sounds, so this dataset is considered useful, but secondary to the Iowa dataset.

Note that the sound files in both datasets have a sample rate of 44100 Hz.

Experiment 9a: Autocorrelation vs SNAC function at choosing the octave.

The ACF and SNAC function were tested using both datasets. For this experiment the pitch period is chosen as the global maximum peak, when the peak at $\tau = 0$ is excluded. In order to test the SNAC function, an artificial slope is applied so that if there are multiple peaks with a similar amplitude the left-most one will be chosen. That is, Equation 4.2 is modified to become

$$n'_{sloped}(\tau) = \Big(\frac{W}{W-\tau}\Big) \frac{2 \sum_{j=0}^{W-1-\tau} x_j x_{j+\tau}}{\sum_{j=0}^{W-1-\tau} (x_j^2 + x_{j+\tau}^2)} \tag{6.1}$$

In comparison the ACF, as stated in Equation 2.6, has a natural slope that tapers the function to zero at index W - so it is used unmodified.

The periodic peak chosen is considered to be a 'pitch octave error' if its maximum is more than 1.5 semitones from the expected pitch. The term 'pitch octave error' denotes that the incorrect peak has been selected, and may describe an error other than a note only in the incorrect octave; however, it is common when the incorrect peak is selected that the resulting pitch in the incorrect octave. This experiment allows for some variation in the exact position of the peak maximum, but is sufficiently small that if an incorrect peak is chosen an error will result. The analysis is performed frame by frame throughout all the sound files in the given database. The error rate shows the percentage of incorrect peaks from the frames which contain an expected pitch - *i.e.* frames that have no marked pitch are ignored.

Note that a window size of 2048 samples was used with a hop size of 1024 samples. Any notes below $F1$ were removed from the datasets, as $F1$ is the largest period which can be measured reliably using this window size.

The results in Table 6.1 show the percentage of frames in which the peak chosen was incorrect. The sloped SNAC function performs slightly worse than the ACF on both of the datasets. These figures show that on the main dataset, the Iowa dataset, one octave error occurs about every 25 frames on average. Moreover, at the frame rate of 43 frames per second, this is about 1.7 octave errors per second. This high rate of error would not only frustrate the user, but can upset any further analysis done on the

Percentage of pitch octave errors		
Dataset	Iowa	MIDI
ACF	3.94%	1.66%
SNAC (Sloped)	4.08%	1.70%

Table 6.1: A comparison between the ACF and SNAC function showing the percentage of frames with incorrect pitch octaves.

pitch signal. In order to make a satisfactory musical tool, the octave error rate needs to be reduced.

6.2 Can the Fundamental Frequency and the Pitch Frequency be Different?

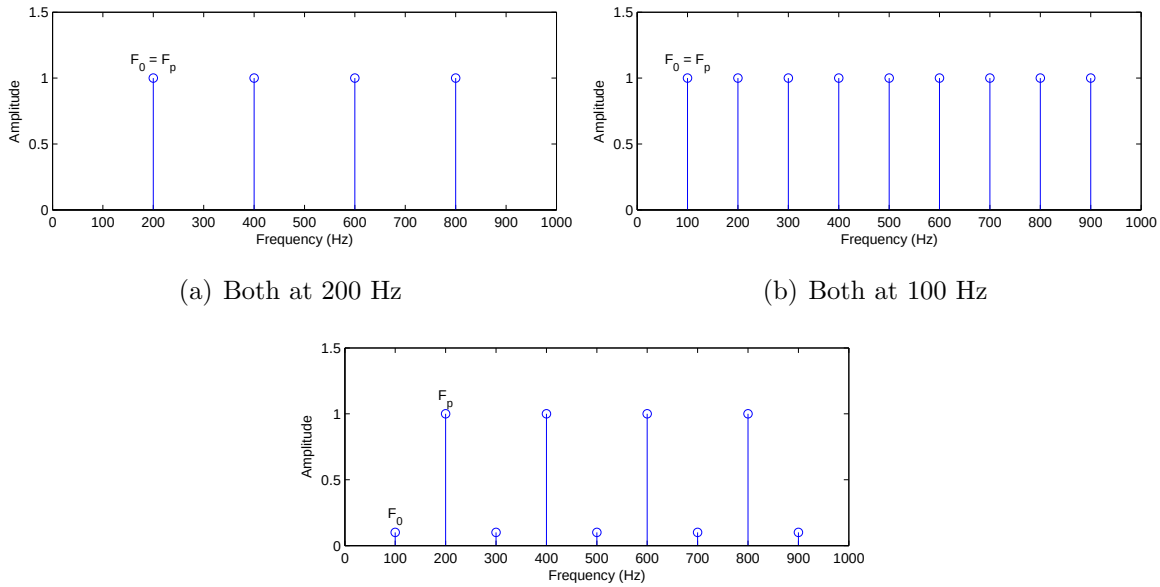
It may not be obvious as to how the fundamental frequency and the pitch freqency can be different. The following uses an example with 3 different cases to help explain why.

Case 1: If a sound is produced that contains frequencies which are all a multiple of 200 Hz, and are all the same amplitude, as shown in Figure 6.1a, then the fundamental frequency is 200 Hz, and the pitch frequency is also 200 Hz.

Case 2: If a sound is produced that contains frequencies which are all a multiple of 100 Hz, and are all of the same amplitude, as shown in Figure 6.1b, then the fundamental frequency is 100 Hz, and the pitch frequency is also 100 Hz.

Case 3: If a small amount of signal from case 2 is added to case 1, as shown in Figure 6.1c, then the perceived pitch frequency, F_p, is still associated with the dominent spetra at 200 Hz. However, the mathematical fundamental frequency, F_0, is 100 Hz, as it is the lowest common denominator frequency, and associates with the period of the signal.

If the amplitudes of the odd multiples of 100 Hz are increased, then at some point the pitch frequency will switch to be perceived at 100 Hz. The point at which this switch happens is a property of human pitch perception, and can vary with the frequency range, and harmonic structure of the signal.



(a) Both at 200 Hz

(b) Both at 100 Hz

(c) Different

Figure 6.1: Fundamental frequency vs pitch frequency example

6.2.1 Pitch Perception

This section examines the mechanism of human pitch perception, with some of these insights used to improve the pitch detection algorithms.

The human ear consists of three main parts: the outer, middle and inner ear. The outer ear consists of the pinna, the visible part on the outside and the ear canal, which helps focus energy on the ear drum. The middle ear consists of the ear drum and the ossicles, a series of three bones, which act as a lever to transfer the vibrations into the fluid and membranes in the inner ear. The inner ear contains the cochlea, which is where the actual sensing of the sound is done [Carlson and Buskist, 1997]. The cochlea contains the basilar membrane, which changes in width along its length, with different parts having different resonant frequencies. Inner hair cells along the membrane's length cause certain neurons to fire with localised movements of the membrane. These firings send electrical pulses up the auditory nerve [Moore, 2003].

Two major theories of pitch perception attempt to explain how the neuron firings are converted into pitch information. These are *place theory* and *temporal theory* [Moore, 2003]. In place theory the basic idea is that the pitch is found using information from the position or place of the hair cells that caused firings along the basilar membrane, whereas in temporal theory it is thought that the pitch is deduced from

information related to the timing between neuron firings. However, it is likely that a combination of these methods takes place with the different methods having greater influence at different frequencies. Therefore, the ear is probably performing both time analysis and frequency analysis of a sound wave before deciding on the pitch [Rossing, 1990].

An important consequence of the ear's design is that, as sound passes from the outer ear through the middle ear, not all frequencies pass evenly. The outer and middle ear have a filtering effect. The only information used by the inner ear for perceiving the pitch has had this filtering applied. Thus, in order to match the pitch perception of humans, it makes some sense to apply a similar filtering effect to the sound before performing any pitch detection.

6.2.2 Outer/Middle Ear Filtering

Robinson [2001] discussed a filter that was employed to approximate the sound that is actually received by the inner ear. We implemented this filter and applied it to the input as a pre-processing step, in an attempt to reduce pitch octave errors, making the overall algorithm more of a pitch detector, than a pure fundamental frequency estimator. This step can be considered more biologically based than other parts of the algorithm.

The SNAC and WSNAC functions, in conjunction with peak picking, can be used to find the fundamental frequency of any harmonic signal. The algorithms are not constrained to sound signals alone, but can be used on any type of signal, so long as the zero frequency component is first removed - which can be approximated by subtracting out the signal's mean. However, the filtering described here is only for use with sound signals when trying to determine musical pitch. To help understand where this filter design comes from, the concept of perceived loudness is first discussed.

Figure 6.2 shows a graph of perceived equal-loudness curves [Nave, 2007], generated from psycho-acoustic tests. The *phon* is a unit of loudness and is defined as the intensity, in decibels, of a 1000 Hz frequency which is perceived to have the same loudness.

Figure 6.3 takes an 80 phon curve and turns it upside down to give an attenuation curve. Here the smallest attenuation, at around 4 kHz, is set as the 0 dB attenuation, so the filter has no gain. This curve is used as an approximation to the attenuation that occurs in the outer/middle ear. Two digital filters from Robinson [2001] are constructed to approximate this target curve.

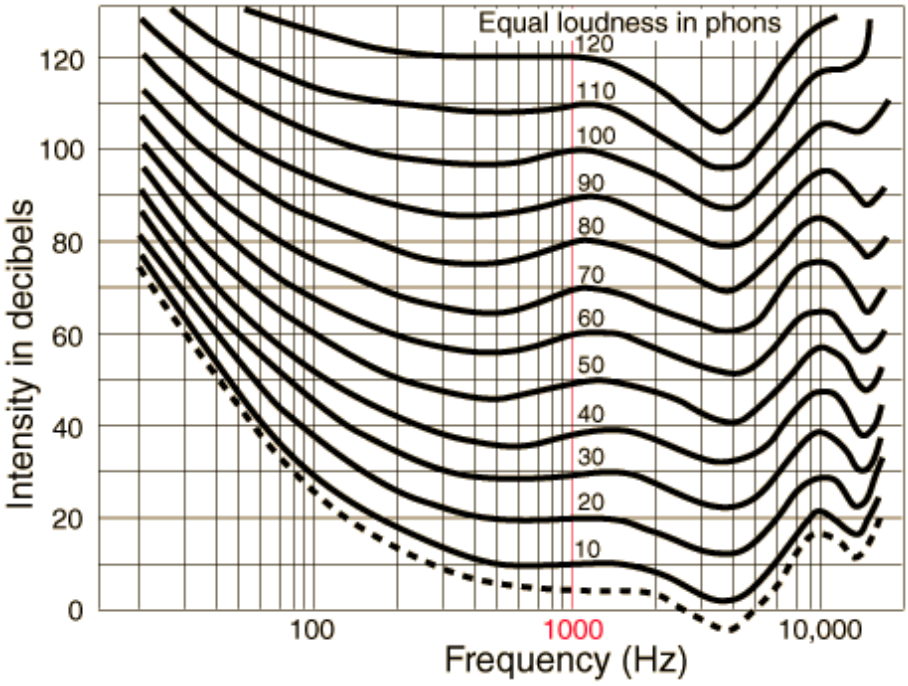


Figure 6.2: Equal-loudness curves [Nave, 2007]. This graph shows the intensity required at different frequencies to give the same perceived loudness.

The low frequency drop off of the outer/middle ear filtering is approximated using a second order Butterworth filter, giving a slow frequency roll-off. This filter is a high-pass *infinite impulse response* (IIR) filter with a 150 Hz cutoff frequency. That is, frequencies below 150 Hz are attenuated by 3 dB or more. Note that the frequencies below 150 Hz are not removed completely, but are reduced in strength. This filter is fast to run, and gives a significant improvement to the rest of the pitch algorithm when compared to using no filter at all. It reduces any low rumbling sounds that are below human hearing, as well as any DC offset component.

The next filter tries to approximate the more complicated shape on the right-hand side of the target curve. A 10^{th} order IIR filter is used which was designed using the Yulewalk method as implemented in Matlab's signal processing toolbox [MathWorks, 2006]. However, this filter is more computationally expensive and does not have as significant an effect as the first filter on the pitch recognition process. If computation time is a real problem, this filter can be skipped without causing any large differences in the result.

Both filters are combined to form a single outer/middle ear filter. Figure 6.3 shows the filter response compared with the target curve. This filter approximates the 80

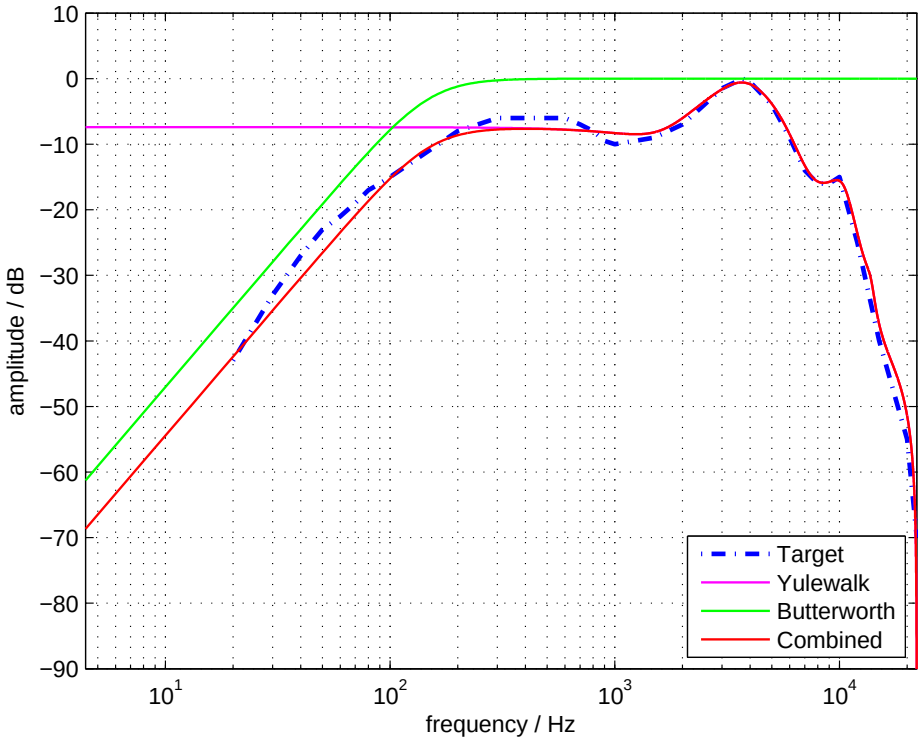


Figure 6.3: The target equal-loudness attenuation curve and the designed filters

phon attenuation curve very well. For further details of the filters see Appendix B.

Experiment 9b: Using the middle/outer ear filter.

This experiment is the same as Experiment 9a except that the middle/outer ear filter is applied to the sound before any pitch detection is performed.

Periodic Error		
Dataset	Iowa	MIDI
ACF	2.43%	1.68%
SNAC (Sloped)	2.55%	1.68%

Table 6.2: Comparison of periodic errors using middle/outer ear filtering

The results in Table 6.2 show the percentage of frames in which the peak chosen was incorrect. Notice that the error rate has been reduced for both algorithms on the Iowa dataset compared with Experiment 9a. However, on the generated MIDI dataset no significant change was made. Also note that the sloped SNAC function and ACF

have similar performance. The best error rate of 2.43% on the Iowa database is still quite high, considering at the speed of 43 frames per second it equates to about one pitch error per second. Thus further improvement in the error rate is necessary. This is addressed in the following sections with the development of a *peak picking* algorithm and some other variations of it.

6.3 Peak Picking Algorithm

This section investigates the properties that govern the shape of the SNAC function. Then, using this knowledge, a *peak picking* algorithm is described which attempts to choose the correct peak more often than the simple global maximum approach.

The SNAC function indicates how well the input function correlates with itself when delayed by varying amounts. The hope is for the SNAC function to reveal the period which corresponds to the musical pitch. However, it turns out that the truly mathematical fundamental period of a signal is not always the same as the musical fundamental period that is of interest. The differences between these is discussed here, as well as methods to find the musical period more consistently.

Firstly, the reader is familiarised the with the SNAC function's shape, before discussing the typical problems that arise when extracting the fundamental period. This is followed by a proposed peak picking algorithm, and experimental results.

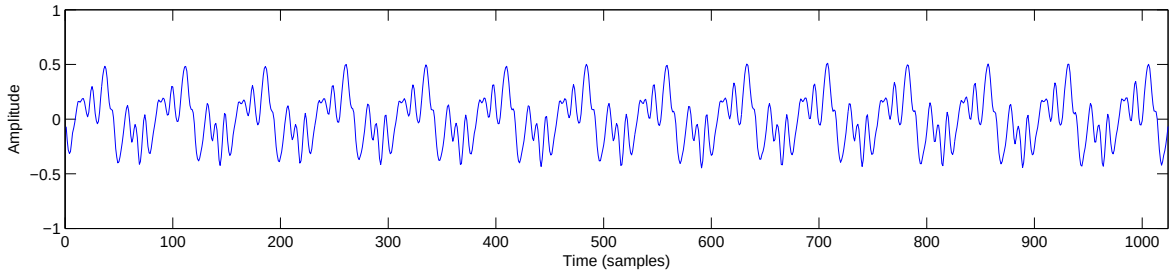
A form of peak picking is often done in the Fourier domain in order to find the frequency of harmonics or partials in a signal, as discussed in Section 2.4. However, harmonics can fluctuate greatly in amplitude and frequency, especially at the beginning/attack part of a note, or during vibrato. The lag domain was found to be a lot more stable in most cases than the Fourier domain, allowing peaks to be found more consistently.

6.3.1 Investigation

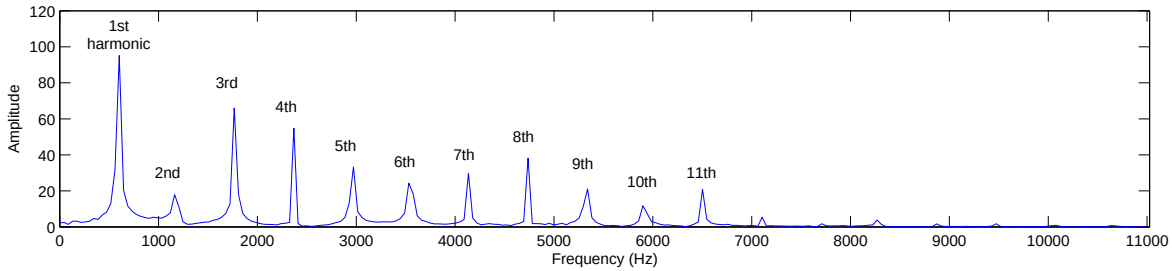
Here the pitch frequency, F_p, is referred to as the first harmonic, and a frequency n times F_p as the n^{th} harmonic. Analogous to this naming, we define the pitch period, P_p, as the first *'periodic'*, and a period n times P_p as the n^{th} periodic. The pitch period is defined as the inverse of the F_p and is approximately equal to the length of time for one cycle of the pseudo-periodic waveform.

When plotting a SNAC function, these periodics show up as primary-peaks. This means that delaying the original signal by any of the periodics will cause it to correlate

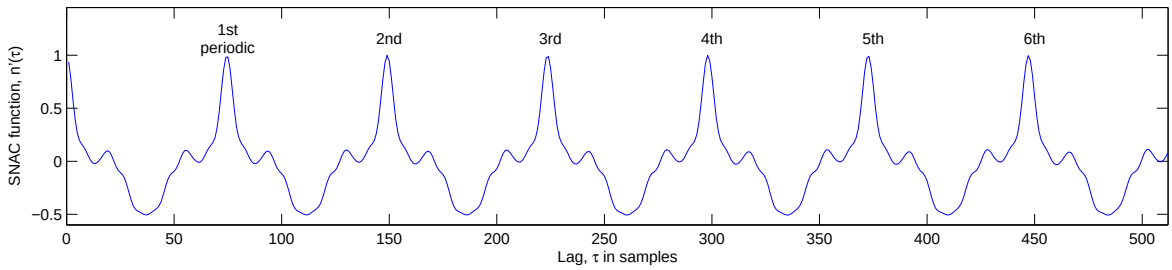
well with its original self. Figure 6.4 shows a segment of a violin recording, its Fourier transform, and the SNAC function of the violin recording. It can be seen in (c) that the first periodic appears at 75 samples, which is the length of time to repeat the waveform in (a). The second periodic in (c) shows that the waveform in (a) also repeats after 150 samples, and so on. Given that the sample rate is 44100 Hz, the fundamental periodic in (c) can be seen to be the reciprocal of the fundamental frequency in (b) of 588 Hz.



(a) A window, of size $W = 1024$, taken from a violin recording



(b) The frequency spectrum



(c) The SNAC function plot

Figure 6.4: (a) A piece of violin data, (b) the Fourier transform of the violin data, (c). The SNAC function of the violin data.

In simple cases, such as the violin example in Figure 6.4, the fundamental frequency dominates the signal. This typically has the property of producing clear outstanding periodic peaks in the SNAC function. In these simple cases, simply picking the first of the highest peaks as our pitch fundamental, or first periodic, is correct most of

the time. A periodic with a lag value greater than the first periodic, is referred to as a *super-periodic*. In real music, a super-periodic may contain a higher peak than the first periodic. This can come about because the signal is not stationary and may contain amplitude or frequency variations. Moreover, the signal may contain a certain amount of noise and other background sounds, and even some precision errors. These factors could, for example, result in the second periodic having a higher peak than the first periodic. Note that even though, mathematically, the second periodic may form a better F_0 for the frame, this is not the pitch period, F_p, that is of interest. If a super-periodic peak is mistaken for the first periodic, it results in a misclassification of pitch, typically an octave or more lower than expected.

In more complicated cases of a musical note, harmonics other than the first can dominate the signal. Figure 6.5 (a) shows an example of a signal with a strong 2^{nd} harmonic which can be seen in (b). This causes a strong *sub-periodic* to appear in the SNAC function as shown in (c). A sub-periodic is defined as any peak in the SNAC function that has a lag smaller that the first periodic. Sub-periodics can also be mistaken for the first periodic if they are sufficiently high. An example of a sub-periodic can be seen in Figure 6.5. Notice in (a) how the signal closely repeats after 88 samples, but repeats more accurately after 176 samples. On listening tests for this signal, a human will identify the pitch period, P_p, at 176 samples.

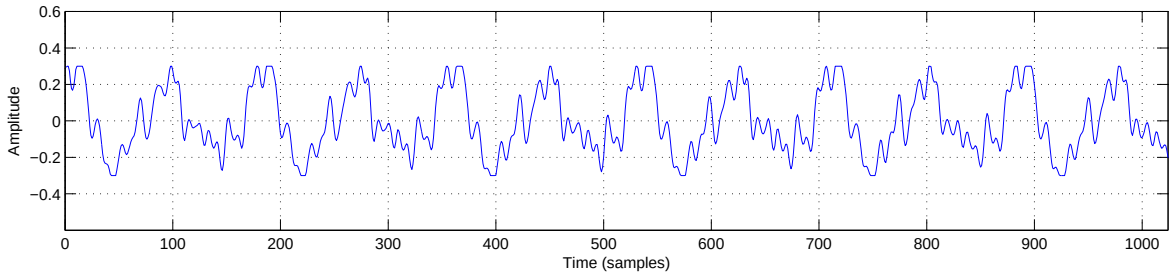
6.3.2 The Algorithm

In the general case, many sub-periodic peaks and super-periodic peaks can occur in the same SNAC function making it a difficult task to identify which peak to label as the first periodic. The peak picking algorithm described in the following was developed by experimentation.

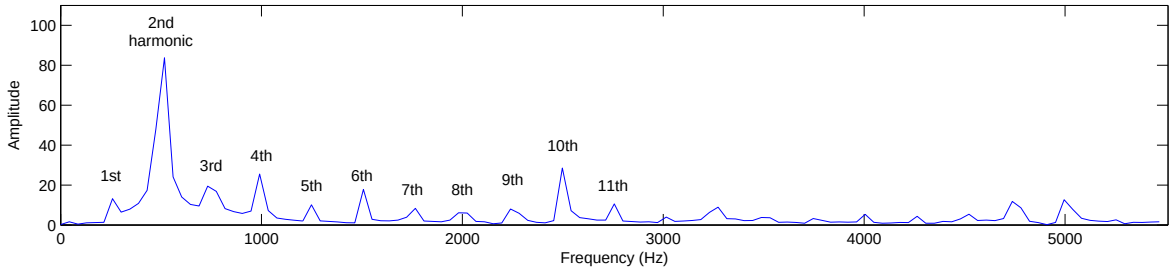
The first part of the algorithm involves categorising peaks into two types, primary-peaks, which are considered to be possible periodics, and secondary-peaks. For example, in Figure 6.6 the periodic is 325 samples, however, a lot of sub-periodics occur; caused by strong harmonics making a ripple across the graph. The two types of SNAC function peaks are described as follows:

- A primary-peak is defined as the maximum between a positively sloped zero-crossing and negatively sloped zero-crossing.

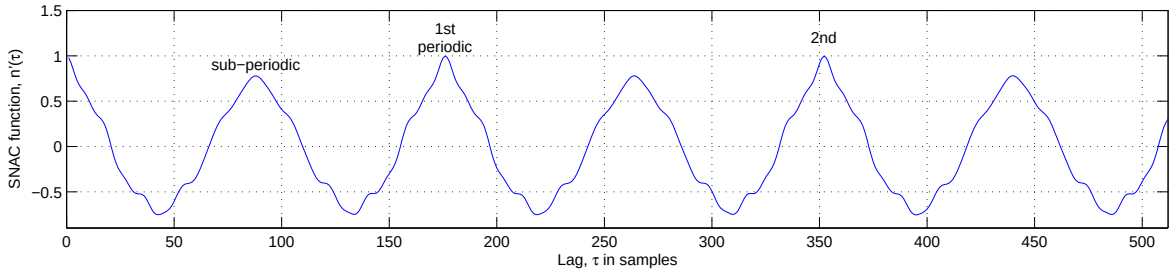
- All other maxima are labelled secondary-peaks, including any peaks before the first positively sloped zero-crossing. Secondary-peaks get discarded from the rest



(a) A sample taken from a violin recording with a strong second harmonic



(b) The frequency spectrum



(c) The SNAC function plot

Figure 6.5: (a) A piece of violin data with a strong second harmonic, (b) the Fourier transform of the violin data, (c). The SNAC function of the violin data.

of the algorithm.

- If there is a positively sloped zero-crossing toward the right-hand end of the graph without a succeeding negative zero-crossing, the highest peak since the zero-crossing is considered a primary-peak, if one exists.

The zero-crossing concept relies on the original signal having its DC or zero-frequency component removed. Note that in our implementation this happens during the outer/middle ear filtering discussed in Section 6.2.2. This ensures that a zero-crossing will occur at least once between any n^{th} periodic and the next, provided that there are no sub-periods that are higher than the first periodic.

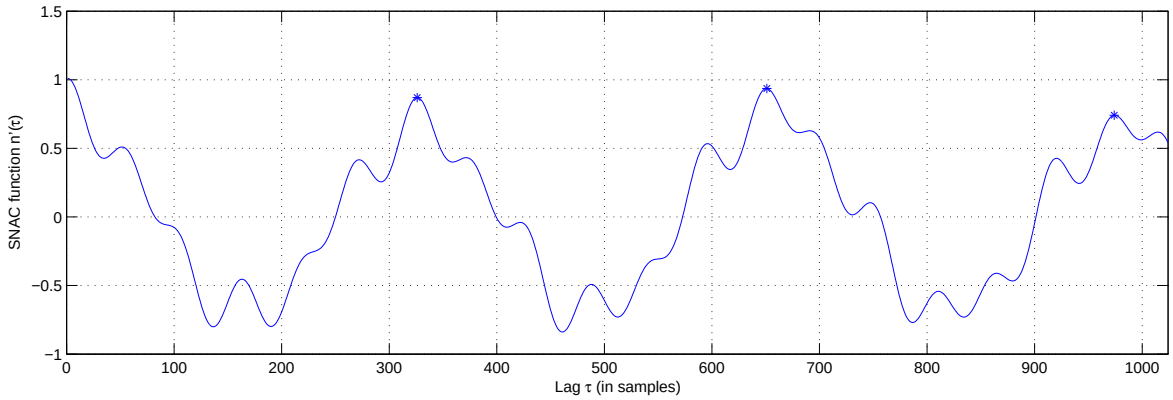


Figure 6.6: A SNAC function graph showing three primary-peaks marked with *'s, the highest being at $\tau = 650$, whereas the 1st periodic is at $\tau = 325$. All unmarked peaks are secondary-peaks.

In Figure 6.6 there are three primary-peaks which are marked with a $*$. However, if the value at $\tau = 720$ had gone below the zero-line, it would have created another primary-peak at $\tau = 750$. This new peak would be considered a spurious primary-peak and is hard to eliminate effectively and could therefore be included. However, these spurious peaks are normally a lot smaller than the other primary-peaks, making the likelihood of them being discarded in the later part of the algorithm very high. Nevertheless, the main concern in this part of the algorithm is to remove any ripple peaks near the top of a primary-peak which could be mistaken as a periodic in later stages. However, reducing the number of peaks also reduces the amount of calculation in the later part of the peak picking.

This algorithm can be used on any of the autocorrelation-type methods, however it cannot be used on the SDF because that has no concept of a centre or zero-crossing. This is one of the advantages of the SNAC function over the SDF.

The parabola technique described in Section 4.3 is used for each primary-peak to get a real-valued position. Let K_τ be the set of these primary-peak lag values. Let K_n be the set of corresponding $n'(\tau)$ values of peak heights. Note, $n'(\tau)$ is being treated as a continuous function here, of which Equation 4.2 only gives values at integer positions.

The next part of the peak-picking algorithm is to define a threshold, $v_{threshold}$. This threshold is equal to the $n'(\tau)$ value of the highest primary-peak, multiplied by a constant, c. That is to say,

$$v_{threshold} = c \, \texttt{max}(K_n). \tag{6.2}$$

Finally, the first periodic is then chosen to be at the first primary-peak which has

its $K_n \geq v_{threshold}$, with the corresponding lag value from K_τ giving the actual period value.

The constant, c, has to be large enough to avoid choosing sub-periodics caused by strong harmonics, such as those in Figure 6.7, but low enough as to not choose super-periodics. Recall, that choosing an incorrect first periodic will result in a pitch error, usually a wrong octave.

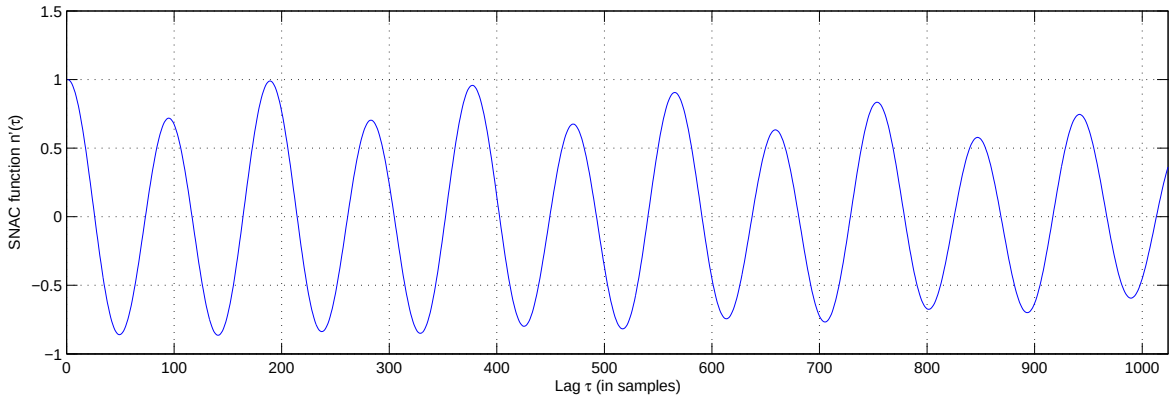


Figure 6.7: A graph showing the SNAC function of a signal with a strong second harmonic. The real pitch here has a period of 190 samples, but close matches are made at half this period.

6.3.3 Results

Experiment 10: The SNAC function with peak-picking, using varying constants.

The Peak Picking algorithm was tested in a similar way as the simple global maximum peak method described earlier, except non-sloping versions of the equations are used *i.e.* Equation 2.7 and 4.2. The experiments were run using a range of different values of the constant c, with the results shown in Table 6.3.

Periodic Error					
Constant c	0.8	0.85	0.90	0.95	1.0
Iowa database	2.34%	1.46%	0.96%	**0.93%**	26.1%
MIDI database	**0.66%**	1.03%	1.87%	4.00%	23.3%

Table 6.3: Results of the peak picking algorithm on the SNAC function

Notice how the error rate for $c = 1.0$ equals that of the simple global maxima approach when the non-sloped algorithms are used. The new algorithm shows some improvements, with an error rate of 0.93% on the Iowa database when $c = 0.95$, bettering the previous best error of 2.43%. The MIDI database improves from a previous best of 1.66% to 0.66% when $c = 0.8$. However, the problem is that these are not the same c value. One solution is for a program to provide the ability for the user to change the constant to suit their instrument, although this is by no means ideal.

Pitch is a subjective quantity and impossible to get correct all the time. In special cases, such as the 'tritone paradox' [Deutsch, 1986], the pitch of a given note may be judged differently by different listeners. We can endeavour to get the pitch agreed by the user/musician as often as possible. Note that the datasets used in this thesis use musical notes with a pitch that is fairly well defined. However, the value of c can be adjusted, usually in the within the range 0.8 to 1.0, to achieve different octave results.

The errors that do occur typically happen at the beginning/attack part of a note, or during fast pitch changes within a note, such as that in vibrato. Although, a small number of errors arise even in steady notes, due to the vast variation in harmonic structure between instruments. The following section takes another approach to try to improve the error rate. Moreover, these problems are discussed in more detail in Chapter 7, with some other proposed solutions.

6.4 Using the Cepstrum to choose the Periodic Peak

It seems that no matter how we try to select the first periodic in the SNAC function, only a small improvement in the error rate is made. Consider a more generalised value called the *'octave estimate'* to be an estimate of the period. The octave estimate is not expected to be precise in its period, but is expected to reside in the correct octave with a high degree of confidence. Furthermore, the closest primary-peak in the SNAC function to the octave estimate can be used to find the period within the given octave to a greater accuracy.

This section introduces the idea of using a completely different method to choose an octave estimate, and not restricting ourselves to the SNAC or autocorrelation domain values. The cepstrum algorithm, as discussed in Section 2.5.1, is investigated purely for its octave error robustness, before a modified version of the cepstrum algorithm is introduced in Section 6.4.1. The octave estimate is then used as guide to selecting the correct periodic peak in the SNAC function.

One of the limitations of using the cepstrum is that it struggles to choose the fundamental frequency when there are fewer than about three periods in the window. The problem here lies in the STFT, or windowed FFT, as the Hamming window function used produces frequency lobes with a noise equivalent bandwidth metric of 1.37, causing frequency bins to overlap in the Fourier domain. Hence, the Fourier transform of the spectrum, *i.e.* the cepstrum, has a reduced ability to find frequency components in the spectrum function.

Changing the windowing function can change the amount of data that is lost at the edge of the window. A rectangular window can use all the data in the window fully, but causes a lot of spectral leakage. A compromise might prove useful, for example a sine window. However, the only way to increase the frequency resolution whilst maintaining a low level of spectral leakage is to use a larger window, the downside being that some time resolution is lost. Although the concern here is only to generate an octave estimate, there is no need to follow the fast variations in pitch, such as those during vibrato - that is left up to the other algorithm. For now the important thing is to choose the octave correctly even if it requires using a larger window.

Experiment 11: The Cepstrum, using varying constants.

The cepstrum algorithm described in Section 2.5.1 which can be used for pitch detection is used here to obtain an octave estimate. A window size of 4096 is used with a hop size of 1024. Note that the split value was set to a *quefrency* (cepstrum axis) value of 5. This is a bit smaller than the shortest period size in the datasets. This experiment was conducted in the same manner as experiments 9 and 10, with an octave estimate that differs more than 1.5 semitones from the expected pitch counting as an error.

Octave Estimate Errors							
Constant c	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Iowa dataset	21.3%	15.8%	12.6%	**11.3**%	11.5%	13.4%	16.6%
MIDI dataset	21.1%	14.3%	10.9%	8.98%	8.16%	8.88%	11.0%

Table 6.4: Results of the octave estimates using the cepstrum with a Hamming window.

The octave error rates shown in Table 6.5 show the cepstrum has a best of 11.3% on the Iowa dataset with $c = 0.7$. This is a quite a high error rate in comparison to that of the SNAC function. Using the cepstrum does not seem beneficial at all.

6.4.1 The Modified Cepstrum

After investigating the shape of the cepstrum domain for a number of different inputs, it appeared that sometimes cepstrum peaks just formed in the wrong place for no particular reason, causing bad octave estimates.

After studying the shape of the log power spectrum which led to the unexpected cepstrum shapes, it appeared that fluctuations in the more negative values of the log power spectrum were having a large influence on the cepstrum shape. The large negative values observed seemed to come and go at random, with most of them toward the right-hand end of the log power spectrum. Figure 6.8 shows an example log spectrum with some downward spikes toward the right-hand end. The large negative values in the log power spectrum occur when the (linear) power spectrum value is close to zero. This is because the logarithm of a value less than one is negative, and as the value approaches zero, its logarithm approaches negative infinity.

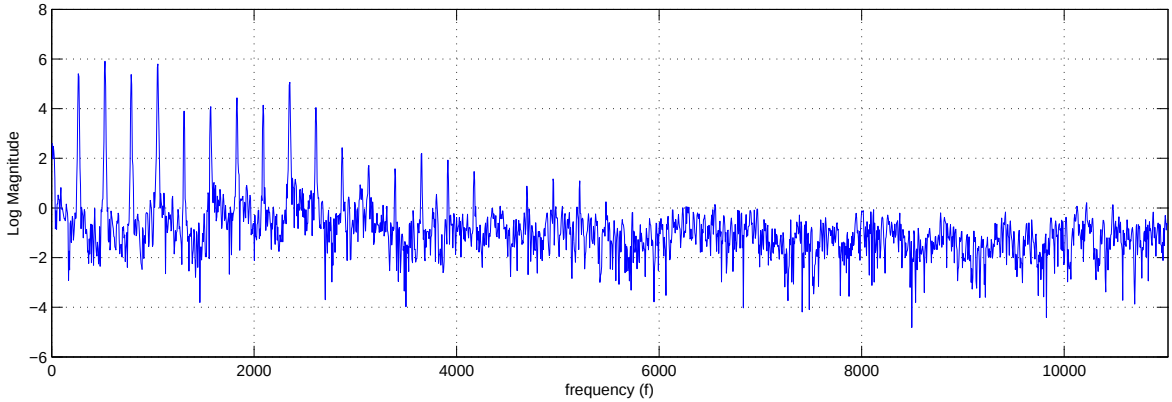


Figure 6.8: The log power spectrum of a segment of flute playing $B3$, using an STFT with a window size of 4096 and a Hann window function.

A reason for these large fluctuations at the right-hand end of the log power spectrum is that the data sets were either recorded in an anechoic chamber or were produced by computer so they have very little or no background noise. The left-hand end of the log power spectrum typically contains strong peaks from the harmonics in the instrument's sound. Spectral leakage occurs around the peaks, reducing the large number of negative values occurring between the peaks. However, the right-hand end of the log power spectrum contains little energy from the instrument itself, but consists mainly of background noise. Depending on the exact shape of the spectral leakage coming from the distant peaks, large negative values can occur. Even if there is some

background noise, groups of small negative values can have a significant effect on the cepstrum shape.

If a Hanning window is used, the spectral leakage curve drops off quite steeply, such that a given peak will influence its 20^{th} neighbouring bin by only -90 dB. In comparison the Hamming window still has a -60 dB influence on its 60^{th} neighbouring bin. As expected the Hanning window produces smaller power spectrum values to the right-hand end far from any strong peaks. This results in larger fluctuations in the log power spectrum compared to the Hamming window and a more chaotic cepstrum shape.

There appears to be no good reason why these groups of very quiet freqency bands should have such a large influence on our pitch detection. The reason for the use of the logarithm, is to achieve the property of addition between the source signal, $s(t)$, from the vocal chords, and the vocal tract filter, $h(t)$, for purposes of separation in the cepstrum domain. The property of having a peak at the fundamental period is merely a secondary effect. However, if our concern is only the fundamental period, there is no need for the logarithm rule to be used.

Experiment 12: The *modified cepstrum* using varying constants

A logical progression was to modify the cepstrum to use $\log(1 + |X(f)|)$ instead of $\log(|X(f)|)$ as originally described in Section 2.5.1. This maintains the curved shape of the logarithm, but makes any small values of $|X(f)|$ result in a zero instead of a large negative value when the logarithm is taken.

The shape of the modified cepstrum curve resembles the shape of the SNAC function somewhat, and it was discovered that the peak picking algorithm from Section 6.3 could be used on the modified cepstrum with great effect. However, a small modification to the algorithm was found to work better on the modified cepstrum. In this experiment Equation 6.2 was replaced with

$$v_{threshold} = m + (c - 1)m^2, \tag{6.3}$$

where $m = \texttt{max}(K_n)$.

The results in Table 6.5 show the modified cepstrum has a reduced error rate over the cepstrum on both datasets. An error rate of 0.617% was achieved on the Iowa dataset and 0.377% on the MIDI dataset. This is not only better than the peak picking result achieved using the SNAC function, but both results achieved well using the same constant of $c = 0.5$. Note that the inverse FFT of the $\log(1+|X(f)|)$ spectrum was actually used here. However, according to the Fourier inversion theorem this is

Octave Estimate Errors								
Constant c	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Iowa dataset	1.84%	0.975%	0.634%	**0.617**%	0.802%	2.16%	2.58%	7.55%
MIDI dataset	1.18%	0.459%	**0.377**%	0.511%	0.857%	1.6%	3.11%	9.05%

Table 6.5: Results of the octave estimates when changing the constant, c, using the modified cepstrum.

equivalent to the forward FFT for our real even function, *i.e.* $|X(f)| = |X(-f)|$.

Previously, changing the scaling of the power spectrum resulted in adding an offset to the cepstrum result. This can be shown by

$$\log(s|X(f)|) = \log(s) + \log(|X(f)|). \tag{6.4}$$

This constant offset does not affect the position of the maximum, and hence is irrelevant to our purpose. However, when using the $\log(1 + |X(f)|)$ function any scaling of the power spectrum becomes significant, as the simple offset relationship does not hold. *i.e.*

$$\log(1 + s|X(f)|) \neq \log(s) + \log(1 + |X(f)|) \tag{6.5}$$

Experiment 13: The modified cepstrum with varied scaling.

This experiment investigates how different values of the scalar s affect the number of periodic errors. Changing s effectively changes the curvature of the curve as shown by Figure 6.9. Note that $\log(1 + sx)$ tends toward a straight line, or linear function, as s approaches zero. Also note that a base 10 logarithm is used in this experiment.

This experiment is run under similar conditions to Experiment 12. The differences being that the peak-picking constant, c, was fixed to 0.5 and that the value of the scalar, s, is varied.

The results in Table 6.6 show little improvement over the previous experiment with a best error rate of 0.634% and 0.35% for the Iowa and MIDI dataset respectively. This small variation in error rate across the different scalar values indicates that the exact value of s is not of great significance. Hence, if the speed is of great concern the linear function performs almost as well, without the need to calculate a lot of logarithms. Note that in the linear case the algorithm becomes similar to the 'Fourier of Fourier' transform discussed in Marchand [2001], and is analogous to calculating a windowed autocorrelation using the FFT method from Section 2.4.5 without zero padding.

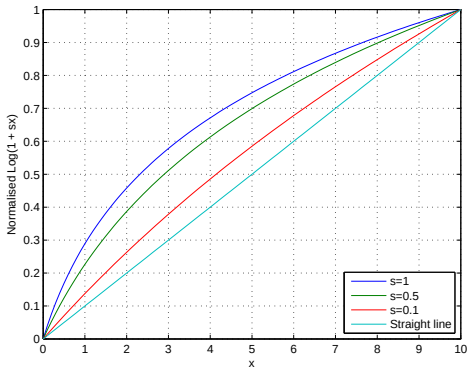


Figure 6.9: A comparson of of the $\log(1 + sx)$ function with different values of s. Note that the height has been normalised so it is 1 at $x = 10$. A straight line has been put in for reference.

Octave Estimate Errors						
Scalar value s	10.0	1.0	0.1	0.01	0.001	Linear
Iowa dataset	0.673%	0.634%	0.712%	0.720%	0.724%	0.724%
MIDI dataset	0.548%	0.377%	0.352%	0.35%	0.35%	0.35%

Table 6.6: Results of the octave estimates when changing the scalar, s, of the modified cepstrum.

In summary, this chapter introduced a filtering and peak picking method for the SNAC function which improved the error rate over the existing autocorrelation-type methods from 3.94% to 0.93% on the Iowa dataset, and 1.66% to 0.66% on the MIDI dataset. Further improvement to the error rate was made with the introduction of the modified cepstrum method to generate octave estimates. This method requires a larger window size to compensate for the Hamming window used. However, this method can be used in conjunction with a SNAC function of smaller size. Thus, it can maintain a high time resolution in order to follow rapid pitch variations, whilst maintaining a low octave error rate. The modified cepstrum was found by experiment to work best with the constant $c = 0.5$ and scalar $s = 1.0$, achieving an error rate of 0.634% on the Iowa dataset and 0.377% on the MIDI dataset.

Chapter 7

Putting the Pitch in Context

This thesis has so far discussed ways of accurately and quickly determining the pitch for a given window of a signal. That is to say, only local information about the sound is used in determining the correct periodic and hence the octave. However, in practice it can be beneficial and even necessary to use information from the surrounding context of the window to help determine the correct periodic.

This chapter looks first at *median smoothing*, a technique for removing gross errors using neighbouring values. Section 7.1 describes median smoothing and how it was applied in an attempt to reduce the pitch error rate. However, its success was limited.

A new method called the *aggregate lag domain* (ALD) is introduced in Section 7.2.1 which uses the idea of a consistent octave throughout all frames within a note and combines together their SNAC function or modified cepstrum space. An estimate of the period is then chosen from this combined function space. Furthermore, this is extended in Section 7.2.2 with a *warped aggregate lag domain* (WALD), in which any variations in pitch are scaled or warped out before combining them to maximise peak correlation. These methods are tested against the existing datasets to measure their pitch octave error rate.

7.1 Median Smoothing

Median smoothing, or median filtering, is a non-linear filtering method for removing gross errors or noise from a signal. A window consisting of an odd number of samples is sorted into numerical order and the median value selected as the output. Then the window is slid one to the right each time, gaining a new value and losing the oldest.

Any extreme values in the signal get pushed to the beginning or end of the sorted

list, so are discarded from the output. One benefit of median smoothing is that steep changes, or edges, in a signal are preserved quite well, in contrast to linear smoothing filters which tend to blur out the region around a steep change.

In the case of pitch, any value that takes on a different octave than its neighbours can be considered an extreme value to be removed. If linear filtering had been applied instead, the pitch contour would have taken on values between the octaves, producing an undesirable effect.

If the erroneous pitch values are few and far between, the median smoothing performs well. However, if there are a whole sequence of pitch errors grouped together, such that more than half of the window contains errors, median smoothing will get an incorrect result. It is possible for not just one error, but a group of incorrect outputs in this case.

One solution is to use a larger window size for the median filter, so that less than half the window has errors. However, the window size must remain small enough to identify short musical notes that contain only a small sequence of correct pitch values. Hence, the window size must be kept smaller than twice the length of the shortest possible note, which is quite short in practice.

Rabiner and Schafer [1978] suggest the use of median smoothing followed by linear smoothing, such as a Hanning filter, with a two pass scheme. However, this does not help solve the grouped error problem.

Experiment 14: Median Smoothing.

This experiment tests the use of a median smoothing filter and measures its effects on the octave estimate. Note that the input to the median filter is the set of octave estimate values, or pitch-input. The output is another set of octave estimate values. This experiment was conducted in the same manner as experiment 13, with values of $c = 0.5$ and $s = 1.0$ used for the modified cepstrum. The median filter was applied to the octave estimate values using varying window sizes.

Table 7.1 shows results from median filters of different sizes. It can be seen that some improvement was made on the MIDI dataset with a window size of 15 reducing the errors from 0.377% to 0.159%. However, there was only a small improvement for the Iowa dataset.

Investigation into the source of the errors found that most of them result from groups of pitch-input errors, or pitch-input errors at the beginning or ends of notes. This group error problem cannot be solved easily unless further information is known about the pitch-input signal.

Modified Cepstrum - Periodic Error		
Dataset	Iowa	MIDI
No median smoothing	0.634%	0.377%
Median smooth 5	0.645%	0.354%
Median smooth 7	0.604%	0.268%
Median smooth 9	0.587%	0.185%
Median smooth 15	0.598%	0.159%
Median smooth 51	1.03%	16.7%

Table 7.1: Comparison of periodic errors using the modified cepstrum and varying the size of the median smoothing.

7.2 Combining Lag Domain Peaks

In Chapter 6 the pitch or octave estimates were calculated for each frame using a SNAC function or modified cepstrum, both of which are considered lag domain functions here. The median smoothing uses only the pitch or octave estimate values from each frame. However, in this section the information from the lag domain is utilised to a greater degree by sharing values, other than the resultant pitch, between neighbouring frames.

Even using the new lag domain methods, such as the SNAC function or median smoothing, errors in the octave estimate can still occur. It is proposed that some of these errors are caused by fluctuating harmonics. That is to say, as a note is played, harmonics can change in amplitude with some amount of independence from each other. At certain frames within a note, it may appear that a different note is being played. Let us take an example near the end of a note, where all the harmonics have faded except the second. Using only the information within the single frame of one strong frequency, any method would have to associate the pitch frequency with the only remaining frequency, resulting in an octave error. However, a human listener would typically consider this sound as part of the ongoing note, and would not perceive it as an octave jump.

This section introduces a new idea in which information from all the frames of a note are combined. Then using this combined information, a single octave estimate is made for the entire note. The expectation is that this single combined estimate should be more robust, and the consistency in the octave across a note will be guaranteed. The method is based on the assumption that the pitch does not change wildly within a

single note. The pitch should be allowed to vary a reasonable amount, such as during vibrato or glissando, but it should not be allowed to jump drastically, for example by a whole octave. A jump this large should be considered a note change, and is discussed more in Section 7.3.

In order to implement this idea fully the positions of the beginnings and ends of notes need to be obtained. This in itself is a whole other field of study which this thesis will not go into in great detail. This section endeavours to test the concept of the combined octave estimate alone, and information of the beginning and ends of notes is used from the dataset mark up. Nevertheless, note onset/offset detection is discussed in Section 7.3.

Note that it may be possible to use information from the surrounding notes to help determine the correct pitch from the musical structure. However, because different music styles contain so much variation, without knowledge of the specific music being played it is possible for notes to vary just about anywhere. Since we do not wish to impose any restrictions on what the user can play, the methods discussed in this section combine together only information from within a single musical note.

7.2.1 Aggregate Lag Domain (ALD)

The basic idea is to use information from the whole of a single note to determine an overall octave estimate. This is in order to average out the effects of a changing harmonic structure, such as those during fast onsets, slow decays, tremolo or vibrato. This combined octave estimate is expected to be more stable than the estimates from a single frame, and will not follow the variations in pitch within the note. Note that the pitch is still found on a frame by frame basis, with the combined octave estimate used only as a guide.

The *aggregate lag domain* (ALD) is defined as the element-wise summation of the lag domains across all frames within the note. For normalised autocorrelation-type methods, the terms are weighted by the total energy, $r'_i(0)$, (or log of the energy) within the window, and then re-normalised. For example, the ALD using the SNAC function is as follows:

$$n'_{Ag}(\tau) = \frac{\sum_{i=s}^{e}(r'_i(0)n'_i(\tau))}{\sum_{i=s}^{e} r'_i(0)} \tag{7.1}$$

where s and e are the starting and ending frame indices of the note and n'_{Ag} denotes the ALD. The extra weighting and normalising is done in order to preserve the original amplitude of influence. For example, the louder parts of the note will take a higher proportion of the aggregate, as these sounds have a stronger influence on the listener. Moreover, this gives an effect similar to one large lag domain function applied across the whole length of the note, but at a much cheaper computation cost. For now, it is assumed that the indices of the starting and ending frames of the notes are known. However Section 7.3 discusses how to detect these automatically.

After the aggregate lag domain is calculated, the peak picking method described in Section 6.3 is applied to this in the same manner as before. The pitch period that results becomes the combined octave estimate, $\tilde{\lambda}_{Ag}$, and will govern the octave of all frames within the note. Note that the $\tilde{}$ indicates the value is an estimate. Further, frame by frame analysis of the note using the SNAC or WSNAC function will now choose the first periodic, P_p, to be the closest primary peak to the combined octave estimate, such that $|K_\tau - \tilde{\lambda}_{Ag}|$ is a minimum.

Experiment 15: Aggregate vs non-aggregate lag domain.

This experiment was created to compare the aggregate lag domain approach to the single frame peak picking approach. As before, if the octave estimate is within 1.5 semitones from the expected pitch it is considered correct, otherwise it is considered an octave error. Note the SNAC function is run using a threshold of 0.9, and the modified cepstrum was run with $c = 0.5$ and $s = 1.0$.

Octave Estimate Error		
Dataset	Iowa	MIDI
SNAC	0.99%	1.87%
SNAC - ALD	0.712%	3.07%
Modified Cepstrum	0.634%	0.377%
Modified Cepstrum - ALD	0.327%	0.0371%

Table 7.2: Comparison of octave estimate errors with and without the aggregate lag domain estimates.

Table 7.2 shows some variable results. The aggregate estimate used with the SNAC function does worse on the MIDI dataset than before, whereas the aggregate estimate used with the modified cepstrum makes a good improvement on both datasets.

Problems arise especially on sounds with lower fundamental frequencies but higher harmonics. These sounds contain periodics that have a narrow width in the lag domain. Any pitch variations within the note cause peaks to 'wobble' from side to side an amount proportional to their τ values, *i.e.* their τ values are scaled. This can cause the narrow peaks with larger τ values to move further than their width throughout the course of the note. This causes the peaks toward the right-hand end of the aggregate lag domain to become increasingly spread out, reducing their overall height. This increases the likelihood of peaks toward the left-hand side to be selected, because they are not affected as much. Thus, an octave error can result, especially with larger pitch variations. However, Section 7.2.2 discusses a method to reduce this effect.

7.2.2 Warped Aggregate Lag Domain (WALD)

The idea here is to warp the lag axis of the lag domain at each frame, in order to align the peaks before aggregating the frames together. This attempts to remove the effects of any pitch variations, such as vibrato, which can cause the misalignment of peaks between frames. A single octave estimate is then chosen from this more stable *warped aggregate lag domain* (WALD) and its value un-warped at each frame when choosing the first periodic.

A number of difficulties and instabilities can arise when trying to implement this idea. Firstly, it is possible for peaks to appear and disappear throughout the duration of a note. Secondly, peaks can move sufficiently far between frames that it is difficult to match up which peak went where. This problem is especially prominent at the higher lag values. Thirdly, in real music it is possible that some peaks appear which do not form a multiple of the first periodic, *i.e.* from non-harmonic components.

The following describes a method that attempts to solve most of these difficulties by scaling, or warping, the τ axis of the lag domain at each frame, in order to align the peaks.

Firstly, the starting frame of the note, n'_s, is calculated as normal, with its lag domain becoming the reference frame that other frames will align to, *i.e.* the starting frame has a warping value of $w_s = 1$.

For subsequent frames, the lag axis is warped, *i.e.* its τ values are scaled, by a warp factor w_i in order to align the peaks. To find the warp factor for the frame with index i, the position of the highest peak from the previous frame, $\tau_m(i-1)$, is used as a reference point. Then the closest peak to this in the current frame, $\tau_c(i)$, is found. The highest peak in the previous frame is used as a reference because it is the most likely

peak to follow on, and is likely to be stable, as being the loudest it is the least affected by noise. These things make it one of the easiest peaks to match up. Following this, the warp factor for the current frame, w_i, is found using

$$w_i = \frac{\tau_c(i)}{\tau_m(i-1)} w_{i-1}. \tag{7.2}$$

Even though this peak may not be the first periodic peak, it is assumed that the peak's position will scale in proportion to that of the first periodic. This should be approximately true if the peak is any of the super-periodics of the note, which is very likely since the highest peak was chosen. This warp factor tells us how much the frame i lag axis has to be scaled by in order to align it with the starting frame.

The lag domain function of each frame is warped - that is, new values are calculated by interpolating the old values. This warping can be described as

$$\breve{n}'_i(\tau) = n'_i(\frac{\tau}{w_i}) \qquad 0 \leq \tau < W, \tau \in \mathbb{Z}, \tag{7.3}$$

where the $\breve{}$ indicates the warped version. Note that the new function is of the original length, so if the warp factor is greater than 1, the extra values are just ignored. However, if the warp factor is less than 1, the remainder of the indices are filled with the last original value, $n'_i(W-1)$, although zeros could also be used. Currently linear interpolation is used to get the fractional indices, but other types of interpolation could also be used here.

Now that the functions have been aligned, the WALD is calculated similarly to Equation 7.1, except here the warped lag domains are used:

$$\breve{n}'_{Ag}(\tau) = \frac{\sum_{i=s}^{f} r'_i(0)\breve{n}'_i(\tau)}{\sum_{i=s}^{f} r'_i(0)} \tag{7.4}$$

Again the peak picking method from Section 6.3 is used to choose the octave estimate in the same way as Section 7.2.1. However, an extra de-warping step is required at each frame before the local periodic is chosen. This is done by multiplying the octave estimate by the warp factor for the given frame, giving

$$\tilde{\lambda}_i = \tilde{\lambda}_{Ag} w_i, \tag{7.5}$$

where $\tilde{\lambda}_i$ is the de-warped estimate. The first periodic, P_p, local to frame i, is now chosen as the closest primary peak to $\tilde{\lambda}_i$, making $|K_\tau - \tilde{\lambda}_i|$ a minimum. Note that K_τ is the set of primary peak lag values for a given frame as defined in Section 6.3.2.

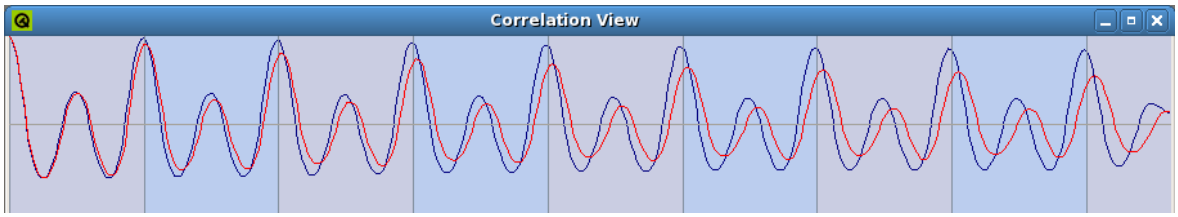


Figure 7.1: The ALD and WALD of a note with varying pitch. The blue line shows the warped peak aligned version, and the red line without warped peak alignment.

Figure 7.1 shows an example which compares the WALD to the ALD. At the left-hand side the peaks are of similar height, but by the right-hand side the red line has become lower in amplitude, with the peaks becoming slightly wider. In contrast, the blue line of the warped method maintains a higher amplitude, decreasing only slightly in height from the left-hand side, while maintaining a narrow width. Note that the alignment of the blue line is different from the red line in the figure. This is due to the first frame of the note having a small periodic, causing subsequent frames to be aligned to it, whereas with the non-warped method, the peaks drift further off to the right.

Experiment 16: Warped vs un-warped aggregate lag domain.

This experiment is the same as Experiment 15, except for the use of the warped context approach. Both the warped and non-warped aggregate lag domain methods are compared.

Octave Estimate Error		
Dataset	Iowa	MIDI
SNAC	0.99%	1.87%
SNAC - ALD	0.712%	3.07%
SNAC - WALD	0.757%	3.40%
Modified Cepstrum	0.634%	0.377%
Modified Cepstrum - ALD	0.327%	0.0371%
Modified Cepstrum - WALD	**0.118%**	**0.0371%**

Table 7.3: Comparison of octave estimate errors using un-warped, warped and no aggregate lag domain.

Table 7.3 shows improvement of warped methods over the un-warped methods on

the Iowa dataset, but a slightly worse error rate on the MIDI dataset when used with the SNAC function. Clearly, the modified cepstrum used with a warped aggregate lag domain performs best overall at achieving a stable octave estimate across both datasets.

The large errors in the aggregate SNAC methods shown for the MIDI dataset seem to be caused by the SNAC function's inability to deal with any reverberation effects, where harmonics from the previous note flow on into the next with decreasing strength. Any contribution to the SNAC function from neighbouring notes seems to have a great influence, especially if the neighbouring notes share some common harmonic frequencies. These common harmonic frequencies become accentuated and generally cause the *Tartini-tone* or combination-tone to become prominent. The window overlap at the edge of the notes also contributes to this effect. For example, if the window is centred at the beginning of the note then half of the window contains data from the previous joining note. Moreover, as the MIDI dataset contains legato notes, which flow directly from one note into the next without gaps, this effect is exaggerated. Note that the Iowa dataset contains long gaps between each note and these effects are not observed, in agreement with this theory.

In contrast, the modified cepstrum does not appear to be affected much by the harmonics of neighbouring notes.

Experiment 17: Reverberation

This theory was tested by letting the MIDI-to-wave converter add in reverberation to the MIDI dataset. The input files were generated with reverberation using the command "timidity -OwM -o [outfile.wav] [infile.mid]". Note that the default reverberation of the converter is used. This experiment is the same as Experiment 16, except that only the MIDI dataset is tested, both with and without reverberation.

Table 7.4 shows that all SNAC methods have an increase in error by at least 1% when reverberation is added, whereas the the modified cepstrum with WALD increased only to 0.109% error. This indicates that the reverberation does indeed affect the SNAC function's ability to choose the correct octave, more so than the modified cepstrum.

7.2.3 Real-time Use

Although the aggregate lag domain-type methods require knowledge of the starting and finishing times of the note in advance, this does not mean that they cannot be used in real-time. As soon as a note onset is detected, the aggregation can begin, and at any

Octave Estimate Error		
Dataset	MIDI - no reverb	MIDI - with reverb
SNAC	1.87%	2.71%
SNAC - ALD	3.07%	4.51%
SNAC - WALD	3.40%	4.62%
Modified Cepstrum	0.352%	0.671%
Modified Cepstrum - ALD	0.0371%	0.140%
Modified Cepstrum - WALD	**0.0371**%	**0.109**%

Table 7.4: A Comparison of the MIDI dataset with and without reverberation.

point in time throughout the note, the current frame can be treated as the end frame. This means that at every new frame in the note, the octave estimate is recalculated, and the periodic peaks from all the previous frames in the note are reselected to be closest to this new value.

The result is that as a musical note is played real-time into the system, the octave of the note is represented by the sound within the note so far. Moreover, this means that as more information comes later in the note, it is possible for the entire note to change octave. However, its new octave will be self consistent. By the time the note has finished being played the resulting octave will be the same as if the aggregated lag domain method had been performed only once on its completion.

7.2.4 Future Work

It might be possible, using smoothly interpolated warp values, to do one continuous warp of the time domain thoughout an entire note to remove the pitch variations - then the SNAC or modified cepstrum analysis performed on the result. This might achieve further improvements than the linear piece-wise warp.

7.3 Note Onset Detection

Both ALD and WALD methods need to know when a musical note starts and finishes. Note onset detection forms part of the mid-level analysis stage, and is needed in order to produce musical score or MIDI style output, or to perform certain higher-level analysis, such as the vibrato analysis discussed in Chapter 9.

A simple note onset detection algorithm uses two independent lowpass filters: one small, responsive, low-order filter and one longer, less responsive, higher-order filter. If we call our sound input stream S, then let L be the result of the low order filter applied to S, and H the result of the high order filter applied to S. It is necessary to add extra delay to L, such that the overall delay from S to L is the same as from S to H.

Then the basic idea is that at the start of a note there is a sudden increase in volume, called the attack. Here the values of H rise slowly due to a larger filter size, because it is being averaged over more data. However, the values of L will increase more quickly causing them to cross above the values of H. Therefore the point in time where the values of L cross from less-than H to greater-than H is classified as being a note onset.

This algorithm relies on a sudden volume increase at the beginning of a note in order to detect the note onset. On many instruments this is the case, such as piano, or guitar, or anything where a string is struck and is left to vibrate resulting in a natural exponential decay. However, in some instruments of our concern, such as the violin, the musician can control the volume throughout the duration of the note. For instance, the bow can apply a continuous force to the string, making any desired volume curve.

Marolt, Kavcic, and Privosnik [2002] use artificial neural networks for *note onset detection* based on a combination of a bank of auditory filters, a network of integrate-and-fire neurons and a multilayer perceptron.

Brossier, Bello, and Plumbley [2004] use a *high frequency content* (HFC) function, the value of which at a given frame is constructed by linearly weighting values of the STFT. Peaks in the smoothed HFC function then indicate the time of note onsets. Moreover, a dynamic thresholding is used to compensate for pronounced amplitude changes in the function profile, along with a silence gate to remove onsets below a certain loudness threshold - typically 80 dB.

An exhaustive discussion of note onset detection methods is not attempted here, as it is not of primary concern in this thesis. However, a good summary can be found in Bello [2003].

The following section discusses an investigation into detecting note changes using the pitch directly. A changing pitch may result in a change in the different frequency bands - giving a means to detect changes from one note to another using bandpass filter methods, such as in HFC. However, it would seem useful to use the pitch information more directly when determining if a note onset or note change had occurred.

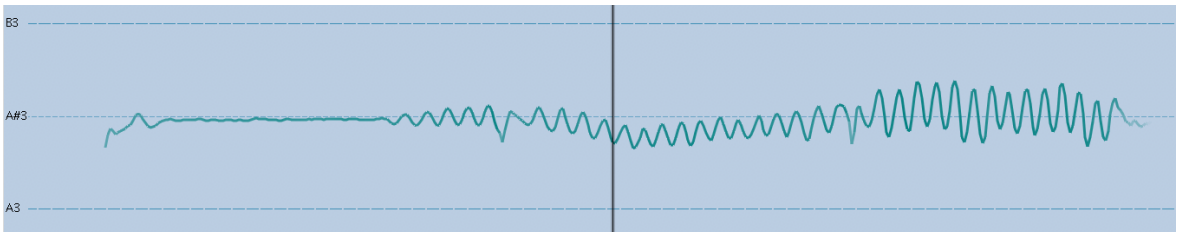


Figure 7.2: An example of some vibrato, that drifts around. The vertical axis indicates pitch and the horizontal axis indicates time (spanning about 11 seconds).

7.3.1 Detecting Note Changes using Pitch

Music can change smoothly between one note and another without a volume break, making the start of a note possible without a sudden onset. This is referred to here as a note change, and implies the finishing of one note and the start of the next, based primarily on a pitch change. Note changes are difficult to detect using a basic amplitude filtering method. However, detecting a note change using pitch is by no means a trivial task either. Variations during a note such as vibrato can cause complications, as these pitch variations are not considered note changes. Moreover, a vibrato may consist of a pitch variation greater than that of a single semitone, thus making a simple pitch change threshold approach unreliable.

Throughout the duration of a note, the pitch is rarely perfectly steady. Figure 7.2 shows a vibrato example that contains drift. Drift is defined here as a variation of the vibrato's centre-pitch, which can often happen on longer notes. Let us define μ_s as the short-term mean pitch: the average pitch over the last t_s seconds. t_s should be chosen to be as small as possible to detect fast-changing notes, but nevertheless should be sufficiently large enough to encompass half a cycle of slow vibrato. A value of around $1/8^{th}$ of a second is good. The idea is that the short-term mean pitch should smooth out any vibrato cycles while maintaining an accurate shape of the drift. This helps to ensure that the individual vibrato cycles are not considered as note changes.

In contrast, μ_l is defined as the long-term mean pitch. This is the average pitch over the last t_l seconds, where t_l is much greater than t_s, causing μ_l to have a more stable pitch, while still maintaining the ability to drift over a very long note.

The following discusses a basic algorithm for detecting note changes using pitch which is very similar to the two-filter amplitude method discussed in section 7.3, except that instead of using short-term and long-term filtered amplitudes, short-term and long-term mean pitches are used.

With the amplitude filter algorithm, when S crosses above L the note is considered as starting. However, with pitch there is no sense of a correct direction, as a note's pitch can change up or down to get to the next note. So instead, a deviation value d_l is used, which defines the amount μ_s has to deviate from μ_l before a note change is detected. This deviation can be on either side.

One of main difficulties with note-change detection using pitch is detecting the difference between a series of fast alternating notes, such as a trill, and a single note with vibrato. The speed of a trill can be as fast or faster than the slowest vibrato. This means that the pitch smoothing used to smooth out the vibrato also smooths out the trill.

Even without the vibrato smoothing, there is still a certain amount of implicit pitch smoothing that happens in the low-level part of the pitch algorithm, due to the finite window size. The square-wave shaped pitch contour of a trill can become significantly rounded at the corners, making it more sinusoidal in shape and difficult to distinguish from vibrato. However, in the case of trills it is likely that the notes will contain some onset amplitude change, so that a combination of pitch and amplitude detection methods can be used to distinguish these.

Once a note change has been detected, the delay introduced from the pitch smoothing needs to be taken into account, in which case time has already progressed into the next note. Once the algorithm has decided there was a note change, then it has to go back and find exactly where it happened. This process is called *back-tracking*, and is described in Section 7.3.2.

This results in the algorithm having to back-track to the place where the note change actually occurred. The averaging filter is then reset, causing it to start averaging from the start of the new note, otherwise the previous note will have a strong influence on the new note's short-term and long-term pitch.

7.3.2 Back-Tracking

Once a note change has been detected, the exact position of the note change needs to be determined. In the method described here, and referred to as *back-tracking*, it is assumed that the exact point of the note change is somewhere within the last t_s seconds, the size of the small smoothing filter, and is most likely to be near the centre of this, *i.e.* at the filter delay of $t_s/2$. The original pitch contour is consulted to find where the point of greatest gradient magnitude is within these bounds. The gradients can be weighted, with higher values at the centre and tending toward lower values at

the bounds. This serves to choose the gradient peak closest to the centre of the bounds in cases where there are bi-modal or tri-modal gradient peaks. The point in time of the greatest gradient magnitude peak is called t_{ggm}.

When back-tracking occurs, the WALD has to be rewound back to the point of the note change. This is done using a second array which stores the aggregate of the warped lag domains since t_{ggm}. If at any stage a higher gradient magnitude peak is found then t_{ggm} is updated and the second array is reset to zero. Note that when a note change is detected, the WALD is rewound back by simply subtracting the second array.

7.3.3 Forward-Tracking

Forward-tracking happens after the back-tracking has occurred, and involves the process of restarting the WALD from the point of note change to bring it up to the current frame. A number of actions need to be performed. Firstly, the WALD for the new note simply takes on the values from the second array. Secondly, the pitch values for the old and new notes need to be recalculated using their updated octave estimates. This forward-tracking method allows new pitch values of the note to be realised without the need to store the lag domain or perform any transformations from existing frames. It requires only three things: the positions of the primary-peaks to be stored at each frame, one array for the WALD, and a second array for the subset of WALD values since time t_{ggm}.

Chapter 8

Further Optimisations

This chapter looks at futher optimisations for the algorithms described so far. Firstly, the choice of window size is discussed in Section 8.1, with the introduction of a two part approach. This method chooses a window size appropriate for the specific data at that point in time. Secondly, Section 8.2 defines an incremental version of the SNAC function, which results in a pitch value for every sample of input, requiring only a small, fixed number of calculations to go from one pitch value to the next. This incremental algorithm is extended in Section 8.4 to work with the WSNAC function. However, Section 8.3 first introduces the *complex moving-average* (CMA) filter, which describes a fast method to perform a Hann weighted convolution which is then used in the *incremental WSNAC* function algorithm.

8.1 Choosing the Window Size

This section introduces the use of a two part window size method. Firstly, the window size is chosen to be sufficiently large to capture the lowest desired pitch. This could be chosen by the user directly, for example by means of the user specifying a note, or more indirectly, such as by the user's choice of instrument. The window size can then be made as small as twice the fundamental period of this lowest note.

Even though this window size is just large enough to determine the low notes accurately, it is still in fact relatively large compared to the period of higher pitched notes on the same instrument, as a typical instrument can range over 2 or 3 octaves. In this case, the fast variations in pitch that can come about at higher pitches are lost due to the sub-optimal time resolution. For example, at 3 octaves above the lowest note, the window would contain 16 periods, and no information on how the pitch could

have varied within this large window can be achieved.

Nevertheless, the large window does give a good estimate of the average fundamental period throughout the window. Using the period estimate, λ_e, of this large window, a second smaller window can be chosen in which a higher time resolution is achieved. Moreover, because the approximate fundamental period is known, and only the localised variations are to be found, there is no need to calculate all the lag domain coefficients. Only the coefficients around the period estimate are needed, with enough to cover for an allowed pitch variation. Using this property, the incremental algorithms described in Sections 8.2 and 8.4 are developed. These algorithms have a step size of one, providing a pitch value for every sample. This rate may seem excessive and a bit redundant, but remains efficient to calculate, because only a handful of coefficients need to be calculated for each step. In any case, samples can be ignored to give any desired rate.

At very small window sizes, there is a reduced number of samples being used in correlation calculations, causing less certainty with which the exact fundamental period is known. However, the exact time that the frequency occurred is more certain. This is the familiar time/frequency tradeoff that governs all analysis methods, and stems from the *uncertainty principle* [Serway, 1996]. However, if we assume that in musical signals the major pitch changes controlled by a human rarely go faster than 20 times a second, then based on this a window size of $1/40th$ of a second should be enough to capture these. However, the pitch changes are not perfectly sinusoidal, so a higher rate is needed to capture the shape. In practice capping the minimum window size to around 512 samples at 44100 Hz serves well, as a certain number of frames are needed for other parts of the analysis, such as the note change detection.

Like multi-resolution techniques, such as wavelets (Section 2.5.2), this two part window size method allows for a varying time/frequency tradeoff. However, rather than using an optimal constant Q[1] tradeoff *i.e.* a bandwidth proportional to the centre frequency, this method allows for customised tradeoffs that can be based on known prior physical properties. For example, if variations are known not to exceed a certain rate, then having an increased time resolution is not always useful. Instead, having a fixed frequency accuracy below a certain point, with constant Q elsewhere, can be more beneficial.

[1]The Q or quality factor describes the bandwidth of a system relative to its centre frequency

8.2 Incremental SNAC Function Calculation

The *incremental SNAC* function is a fast way of calculating a subset of the SNAC function coefficients over a set of continuous frames. Using the fundamental period estimate, λ_e, from the large window, only the local variations in pitch are left to be found. Because there is only a small amount of time that has passed between a group of frames, the pitch is assumed to be contained within the nearby neighbourhood of the estimate. For typical variations the position of all the local periodic peaks can be found using only the τ coefficients within a neighbourhood width of 9, *i.e.* the set of τ values used, T, contains the closest integer to the estimate and the four integers either side. For such a small set of values to evaluate, the SNAC function from Equation 4.2 can be calculated directly for the first frame, that is without the using the SDF via autocorrelation method from Section 3.3.4.

Once the SNAC coefficients for the first frame of the local window have been calculated, the window can be moved by one sample, *i.e.* a hop size of 1, by using an incremental method. Simply store the numerator and denominator parts of Equation 4.2 separately, as in the following:

$$n'_{top}(\tau) = 2\sum_{j=0}^{W-1-\tau} x_j x_{j+\tau}, \qquad \tau \in T \tag{8.1}$$

$$n'_{bottom}(\tau) = \sum_{j=0}^{W-1-\tau} (x_j^2 + x_{j+\tau}^2), \qquad \tau \in T. \tag{8.2}$$

Note, that in this section W refers to the size of the small window. Then to increment the window one sample forward, simply perform

$$n'_{top}(\tau) \mathrel{+}= 2(x_{W-\tau}x_W - x_0 x_\tau), \tag{8.3}$$

$$n'_{bottom}(\tau) \mathrel{+}= (x_{W-\tau}^2 + x_W^2) - (x_0^2 + x_\tau^2). \tag{8.4}$$

The new SNAC function coefficient can then be calculated using

$$n'(\tau) = \frac{n'_{top}(\tau)}{n'_{bottom}(\tau)}, \qquad \tau \in T \tag{8.5}$$

The meaning of what data lies in within the window has now changed, with the window one sample more to the right than before. This new window position is referred to as the next frame. Equations 8.3 to 8.5 can be calculated repeatedly for each value of τ in the set T, giving a window hop size of one sample each time.

The local periodic peak at each frame can be found in a way similar to Section 6.3. However, the maximum of the first periodic is assumed to be the highest value within

the neighbourhood. If the highest value is on the boundary of the neighbourhood, then the frame can be flagged as having too much variation, or the neighbourhood could be extended by adding the new τ value to the set T and calculating another whole run of SNAC coefficients.

The increments are repeated to give a number of frames equal to the hop size, t_{hop}, (in samples) of the large window, thus giving pitch values at a rate of one for every sample. If the largest value of T is greater than the maximum free sliding space, *i.e.*

$$\max(T) > \frac{W_{large} - t_{hop}}{2}, \tag{8.6}$$

then the small window will need to extend beyond the larger window edge, at the starting and finishing frames.

As musical notes get lower in frequency, the same pitch variation (in cents) causes a larger change in the lag value of the first periodic. This means that larger neighbourhoods can be required for low notes. However, at these lower frequencies the width of the periodic peaks are greater, and accurate peak positions can be found using only every second SNAC coefficient, *i.e.* the set T contains only every second integer. For example, a periodic estimate of $\lambda_e = 207.3$ would result from $T = \{199, 201, 203, 205, 207, 209, 211, 213, 215\}$.

8.3 Complex Moving-Average (CMA) Filter

Before describing the *incremental WSNAC* function in Section 8.4, the following algorithm must be introduced, as it forms a basis of the incremental WSNAC function algorithm. This new algorithm, we call the *complex moving-average* filter or CMA filter[2], was developed for sound processing in this work, however it is a general purpose algorithm in its own right, which could be used in many fields.

The CMA filter is an incremental algorithm, based on the moving-average filter. The CMA filter has the nice property of having a smooth windowing function or kernel, whilst being fast to compute a convolution. Let us first look at the moving-average filter and then progress our way into the new algorithm. The moving average filter takes the mean of the last n terms from the input.

$$y_t = \frac{1}{n} \sum_{j=0}^{n-1} x_{t-j} \tag{8.7}$$

[2]pronounced 'Karma filter'

where x is input, y is output, and t is the current index to calculate. This can be calculated incrementally as follows.

$$y_t = y_{t-1} + \frac{x_t - x_{t-n}}{n} \tag{8.8}$$

This can be considered as a *finite impulse response* (FIR) filter with each coefficient being $1/n$, giving it a rectangular window shape.

The CMA filter can be seen as a special case of *weighted moving-average* filter, where each value is multiplied by a corresponding weight coefficient, before being averaged, as shown in the following:

$$y_t = \frac{1}{s} \sum_{j=0}^{n-1} w(j) x_{t-j}, \tag{8.9}$$

where s is the sum of the weighting coefficients, *i.e.*

$$s = \sum_{j=0}^{n-1} w(j). \tag{8.10}$$

The weighting function used for the CMA filter is a symmetric Hanning window,

$$w(j) = 1 - \cos\Big(\frac{2\pi(j+1)}{n+1}\Big), \qquad 0 \leq j < n. \tag{8.11}$$

Note that the symmetric Hanning window is the same as a Hann window that is longer by two samples, but with the zero value at either end left off, thus saving a couple of wasted calculations. Also note, the equation is normally multiplied by 0.5, however this has been left out, as it gets cancelled out in Equation 8.9. Moreover, the weighting function can also be referred to as the filter kernel.

Now let us break up the combined Equation of 8.9 and 8.11 into separate summation parts, a_t and b_t, with the scaling part separated out, such that

$$y_t = \frac{1}{s}(a_t - b_t). \tag{8.12}$$

The first part is defined as the basic summation component,

$$a_t = \sum_{j=0}^{n-1} x_{t-j}. \tag{8.13}$$

which is the same as in the moving-average filter, and can be calculated incrementally as a running sum. Each time step a has a new x term added on, and an old x term subtracted off, *i.e.*

$$a_t = a_{t-1} + x_t - x_{t-n}. \tag{8.14}$$

The second part is defined as the cosine component,

$$b_t = \sum_{j=0}^{n-1} \cos\Big(\frac{2\pi(j+1)}{n+1}\Big) x_{t-j}. \tag{8.15}$$

It is not so obvious, but this too can be calculated incrementally, as a running complex sum. b_t may be considered as the real part of a complex number, $z_t = b_t + ic_t$, which represents the sum of terms, except that each term is rotated around a circle by a different angle at any instant. Note that c_t can initially be set to 0. If we define a rotation constant, r, as:

$$r = \cos(\theta) + i\sin(\theta) = e^{i\theta}, \qquad r \in \mathbb{C}, \tag{8.16}$$

where $\theta = 2\pi/(n+1)$, then Equation 8.15 can be rewritten as:

$$b_t = \sum_{j=0}^{n-1} \Re(e^{i\theta(j+1)}) x_{t-j} = \sum_{j=0}^{n-1} \Re(r^{j+1}) x_{t-j}. \tag{8.17}$$

Note that $\Re$ indicates 'the real part of'. Also note that because the x values are all real values this can be rewritten as:

$$b_t = \Re(\sum_{j=0}^{n-1} r^{j+1} x_{t-j}) = \Re(z_t). \tag{8.18}$$

Now multiplying z_t by r causes all the terms in the complex sum to be rotated, *i.e.*

$$rz_t = r\sum_{j=0}^{n-1} r^{j+1} x_{t-j} = \sum_{j=0}^{n-1} r^{j+2} x_{t-j}. \tag{8.19}$$

Rotating the complex sum by r is equivalent to rotating each of the terms by r and then summing the result - even though the terms already have varying amounts of rotation. This has the effect of moving the cosine component of the Hanning window across one step for all its elements, except that the terms at the ends need to be dealt with. The last term in the sum, at $j = n-1$, is now:

$$r^{n+1} x_{t-n+1} = e^{i2\pi} x_{t-n+1} = x_{t-n+1}. \tag{8.20}$$

This old element, x_{t-n+1}, can be removed by simply subtracting it out from the complex sum. However, because the position of the window has increased by one, the value of t has also increased by one. Thus, the subtracted element is now referred to as x_{t-n}. The new element to be added in is rx_t. However, if x_t is added to z_t before the rotation is performed then this gets done implicitly.

To summarise, the cosine component can be calculated incrementally using three simple operations. Firstly, add the new term x_t to z. Secondly, perform the rotation by multiplying z by r. Thirdly, subtract off the oldest term, x_{t-n} from z. This can be expressed in the form

$$z_t = r(z_{t-1} + x_t) - x_{t-n}. \tag{8.21}$$

Finally, because $b_t = \Re(z_t)$, this result, along with a_t, can be combined into Equation 8.12 to give our smoothed output. Note that only the current value of a and z need to be stored. That is, once y_t is calculated, the previous values a_{t-1} and z_{t-1} are never used again. Also note that s is constant, hence, only needs to be calculated once.

The following contains some C++ code to help clarify the process.

```
#include <cmath>
#include <complex>

/** @param x Input
    @param y Output
    @param n Size of Symmetric Hanning Window
    @param len The size of x and y
*/
void CMA_filter(const double *x, double *y, int n, int len) {
  double theta = 2.0 * M_PI / (n+1);
  complex<double> r = complex<double>(cos(theta), sin(theta));
  complex<double> z = complex<double>(0.0, 0.0);
  double a = 0.0, s = 0.0, old_x = 0.0;

  //precalculate the scaling factor
  for(int j=0; j<n; j++) s += 1.0 - cos((j+1)*theta);

  for(int t=0; t<len; t++) {
    if(t >= n) old_x = x[t-n];
    a += x[t] - old_x;
    z = (z + x[t]) * r - old_x; //Note: complex multiplication here
    y[t] = (a - z.real()) / s;
  }
}
```

Like the moving-average filter, the CMA filter has a delay of $(n-1)/2$ samples. Moreover, in this example code values of x before the start are considered as being zero. This causes the result to curve up from zero, which is not ideal in all situations. One solution to this is to assume $x_k = x_0$ for $k < 0$. Another solution is to scale y_t by the sum of coefficients used so far, instead of s, while $t < n$. This too can be created using the same incremental rotation scheme.

To understand why this works, it can be seen that a Hann window consists only of a cosine component and an offset component of which the real Fourier transform contains only two non-zero coefficients (for $f \geq 0$). The CMA filter can be seen as a convolution algorithm performed in Fourier space. However, because only two Fourier coefficients are needed, it is fast to calculate them incrementally. The net result is equivalent to a conversion to Fourier space and back, but without the use of the FFT. The CMA filter is really just a special case of an FIR filter that uses a Hann window.

A small modification to Equation 8.12 can allow for the Hamming window kernel, described in Equation 2.8, to be used in the CMA filter - instead of the Hann window. This modification is:

$$y_t = \frac{1}{s}(0.53836a_t - 0.46164b_t), \tag{8.22}$$

where s is the sum of coefficients from Equation 2.8.

This idea can be extended to any window shape which contains a small number of non-zero Fourier coefficients, whilst remaining efficient. It is also possible to use a window shape which is a sub-section of a window constructed using a small number of non-zero Fourier coefficients. This is done by simply adding and subtracting the new and old values of x from z at the appropriate angle of rotation. For example, it is possible to use only the first half of a sine function, *i.e.* from 0 to 180 degrees, to create a sine window. This window is generated from the equation

$$w(j) = \sin(\frac{\pi j}{n-1}), \qquad 0 \leq j < n, \tag{8.23}$$

Only one Fourier coefficient is being used with the sine window, and a smaller angle of rotation is needed for the incremental calculation. Here, Equation 8.16 is used with $\theta = \pi/(n-1)$.

As with the moving-average filter, the CMA filter may accumulate error after a while, which keeps propagating onward, also called drift. Exactly how much drift will occur depends on the precision being used. One method to reduce the error is to use a higher precision within the filter than the precision of the input and output data, for

example using variables of type double (64 bit) within the CMA filter even though the input/output is of type float (32 bit).

8.4 Incremental WSNAC Calculation

The following describes how to calculate the WSNAC function incrementally, in a similar fashion to the incremental SNAC function. Likewise, it is useful when a small set of τ values, T, need their coefficients calculated along a sliding window, *i.e.* a succession of frames all at a hop size of one apart. If the set T is small enough the 'Crosscorrelation via FFT' method from Section 4.2.1 becomes inefficient, as all of the τ values at a given time step have to be calculated, resulting in of the order of $O((W+p)\log(W+p))$ calcuations. In comparison, the incremental method has a $O(l_T)$ cost to move from one frame to the next, where l_T is the size of the set T. So when $l_T \ll W + p$ the incremental method becomes more efficient at calculating the values of T over a series of consecutive frames.

Because the computation is not bound by the 'Crosscorrelation by FFT' properties in which all values of $\widehat{n}'(\tau)$ need to be calculated using the same windowing function, there is no need for the windowing function to be implicitly difined as a combined two-part approach, as it was in Equation 4.22. Each τ coefficient can have its own windowing function tailored to the size it needs.

Equation 4.22 modified in this way becomes

$$\widehat{n}'(\tau) = \frac{2\sum_{j=0}^{W-\tau-1}(w_j(\tau)\cdot x_j x_{j+\tau})}{\sum_{j=0}^{W-\tau-1}\left[w_j(\tau)(x_j^2+x_{j+\tau}^2)\right]}, \tag{8.24}$$

where the windowing function coefficients are now a function of τ as follows:

$$w_j(\tau) = 1-\cos(\frac{2\pi(j+1)}{W_s-\tau+1}), \qquad 0 \le j < W_s-\tau. \tag{8.25}$$

Notice that the denominator inside the cosine term is $(W_s-\tau+1)$ and not $(W_s-\tau-1)$. This is because the Hanning window is being used and not a Hann window, similar to Equation 8.11. Note that the windowing function used must be compatible with the CMA filter method. That is, it must be a function, or sub-section of a function, with a small number of non-zero Fourier coefficients for it to be efficient.

The top and bottom parts of Equation 8.24 are calculated separately, and labelled $\widehat{n}'_{top}(\tau)$ and $\widehat{n}'_{bottom}(\tau)$ respectively. These individual parts can be calculated incrementally by treating each of them as a CMA filter, as described in Section 8.3. Let $p(\tau)$ be the complex sum that is used for the CMA filter of the top part, and $q(\tau)$ for the bottom part. Note that only the real parts of p and q represent the top and bottom of Equation 8.24.

To increment to the next frame, the window is moved one sample forward. To achieve this both p and q undergo rotations in the complex plane, while subtracting off the old terms and adding on the new terms as follows:

$$p(\tau) := r(p(\tau) + 2x_{W-\tau}x_W) - 2x_0x_\tau, \tag{8.26}$$

$$q(\tau) := r(q(\tau) + x^2_{W-\tau} + x^2_W) - (x^2_0 + x^2_\tau). \tag{8.27}$$

Note that the := operator means assignment, giving p and q the new values for the next frame. As with the incremental SNAC function, after these calculations the meaning of the data lying within the window has now changed. This means that the next time the calculations are performed, the values of x are different than before, *i.e.* effectively meaning x_k becomes x_{k+1} for all k simultaneously, although in practice this is not actually performed, as it would be far too expensive. Instead an offset is used based on the frame number.

To calculate the WSNAC coefficient, $\widehat{n}'(\tau)$, for the given frame, use

$$\widehat{n}'(\tau) = \frac{\widehat{n}'_{top}(\tau)}{\widehat{n}'_{bottom}(\tau)} = \frac{\Re(p)}{\Re(q)}. \tag{8.28}$$

Equations 8.26 to 8.28 can be calculated repeatedly for each value of τ required, giving a window hop size of one sample each time.

To find the pitch at a given frame, the same technique is used as for the incremental SNAC function from Section 8.2.

To sumarise, the incremental WSNAC function coefficients are calculated for all τ in T across all the required local frames. The number of local frames required is equal to the hop size of the large window. The alignment of the local frames is such that the centre of the middle frame's window is at the centre of the large window.

Chapter 9

Vibrato Analysis

With the ability to calculate pitch contours, one can start looking at high level analysis of the pitch. This chapter looks specifically at the analysis of vibrato, a common form of pitch variation used in music. Section 9.1 provides a background of vibrato from a musical perspective. Section 9.2 discusses some assumptions that are made about vibrato. Section 9.3 discusses how a variation of "Prony's spectral line estimation" method is applied to find vibrato parameters.

9.1 Background

Vibrato, also referred to as frequency vibrato, is a cyclic variation in the pitch of a note. It is often used to add expression, or beautify a note. Vibrato typically has a sinusoidal shape with frequencies in the range 5 - 12 Hz, with the average being around 7 Hz. The frequency of the pitch oscillation is referred to as the speed of the vibrato. Between 1 and 5 Hz a listener can recognise the periodicity of the pitch change [Rossing, 1990]. However, above 6 Hz the tone tends to be perceived as a single pitch with intensity fluctuations at the frequency of the vibrato. The interval between the the lowest and highest pitch values is called the width[1] of the vibrato. Singers tend to have a greater vibrato width than instrumentalists.

The historically accurate ("period") performances indicate that classical music from the renaissance, baroque, classical and romantic eras contained little use of vibrato in orchestra - thus keeping the sound pure. It was typically used only by soloists, both instrumentalists and singers, as a decoration. However, some musicians from the early nineteenth century such as Paganini and Sarasate started using vibrato a lot more -

[1] Also called height

to add more emotion, colour and flair to the story being portrayed. In the twentieth century the idea of using vibrato almost continuously throughout a piece became the norm, and even parts of the orchestra - mainly violins - would make use of vibrato.

Over the last 20 years or so, the tendency for large and excessive vibrato has subsided, although its use has become rather diverse. For example some 'Salvation Army' type brass bands still use a lot of vibrato. Some classical performances try to keep strictly to the way it was played when it was written, but it is common to hear music with medium use of vibrato at the performer's choice - especially music from the late romantic period such as Wagner and Mahler.

Vibrato is used in 'pop music', for example the singer Tom Jones makes good use of it, and it is common for slow ballad type songs to have vibrato on the longer notes.

Vibrato helps a soloist project his or her sound over the accompanying music, making it stand out and be heard, rather than being lost into the background.

The human voice tends to have some natural vibrato. Instrumentalists often add vibrato to try and bring their instruments to life - giving it these vocal like qualities.

Whether it be vocal or instrumental, it is the long drawn out notes where the musician has the most time to control the expressiveness of the note. The freedom is given to the performer to control how the vibrato takes form. Parameters such as speed, width and shape of the vibrato are under the control of the musician.

9.2 Parameters

Vibrato, defined in Section 9.1, has a number of parameters which could be useful for us to measure. In general, vibrato follows a sinusoidal-like shape, that is the pitch varies up and down repeatedly at a certain speed. Although, this speed may vary over the course of a note. Because of this, finding only a single value of vibrato speed for an entire note is not telling the whole story. An instantaneous vibrato speed at regular time intervals along the pitch curve would be more informative to the user.

Stricter constraints can be applied to a vibrato's shape than to a general periodic signal, such as those assumed for the pitch detection algorithms described in Chapter 2. In a general periodic signal any combination of harmonic structure is permissible, allowing a single period to contain many local maxima and minima. These more general pitch detection algorithms require at least 2 periods of signal to be reliable. However, the speed of vibrato can change relatively fast with respect to the time of a single period, for example a vibrato may change from 5 Hz to 8 Hz in only two periods. Pitch

detection algorithms would smear this speed change over the neighbouring periods, because of the window size required. It would be better if the speed could be found using a smaller window size, say one period or less.

Section 9.3 introduces the idea of using a least-squares regression algorithm to fit a sine wave to a small segment of the pitch contour. Because of the curved up-down nature of a vibrato's pitch contour, a single sine wave fit can describe a segment fairly well. This segment, or window, can then be slid along and the analysis repeated and so on a number of times along a musical note, for example 40 times a second.

Using the constraints of vibrato shape being approximately sinusoidal, a "least squares sine wave regression algorithm", based on Prony's method, can be used to find the vibrato speed - even using less than one period. The algorithm can be used to find a sine wave's frequency, amplitude, phase and y-offset that matches the vibrato's pitch curve within a small window, with the least squares error. It uses a few tricks from Prony's method to do this in a small number of steps.

9.3 Prony Spectral Line Estimation

This section describes how to find the frequency, amplitude, phase and y-offset of a sine wave which fits the data with the least squared error. This technique is a special variant of *Prony's method* as discribed in Hildebrand [1956] and Kay and Marple [1981]. First the general Prony method is introduced, before specialising to the one sine wave case of interest to us, the parameters of which can be found in a fixed number of steps.

In Prony's method the data, x, are approximated with the sum of p sine waves added together:

$$\hat{x}_n = \sum_{m=1}^{p} A_m \sin(2\pi f_m n \Delta t + \theta_m), \qquad n = 0, ..., N-1. \tag{9.1}$$

Here the frequency f_m, amplitude A_m, and phase θ_m, of each sine wave are parameters that are estimated. Δt is the time between each sample. The differences between the true and fitted data $\hat{x}_n$ are error terms e_n:

$$x_n = \hat{x}_n + e_n, \tag{9.2}$$

and the goal is to minimise the sum of squared errors

$$\sum_{n=0}^{N-1} e_n^2. \tag{9.3}$$

9.3.1 Single Sine Wave Case

The simplest and probably the most useful case is a single sine wave fit, where the data has approximately a sine wave shape, but may contain (uncorrelated) noise.

If you take a pure sine wave, shift it left in time by t' and add to it the original sine wave shifted right in time by t', the result is another sine wave of the same angular frequency, ω, and phase as the original. This can be expressed in the form

$$\sin(\omega(t - t')) + \sin(\omega(t + t')) = \alpha_1 \sin(\omega t) \tag{9.4}$$

where $\alpha_1 = 2\cos(\omega t')$. The amount that our original sine wave, $\sin(\omega t)$, has been scaled by, α_1, is a function of the time offset, t', and the angular frequency, ω. Hence, an unknown angular frequency can be found from the amplitude scaling factor α_1, and the time offset t'.

For our discrete data we set $t' = \Delta t$, the time between consecutive samples. For a data sequence $\{x_0, x_1 \ldots x_{N-1}\}$, the left-hand side of equation 9.4 becomes $\mathbf{Y}$, and the right-hand side $\mathbf{X}$, both single column matrices. These are

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-2} \end{bmatrix}, \mathbf{Y} = \begin{bmatrix} x_0 + x_2 \\ x_1 + x_3 \\ \vdots \\ x_{N-3} + x_{N-1} \end{bmatrix}. \tag{9.5}$$

Least squares regression is then applied on

$$\mathbf{X} \cdot \alpha = \mathbf{Y} \tag{9.6}$$

where the scaling factor $\alpha = [\alpha_1]$, giving

$$\alpha = \mathbf{X}^+ \cdot \mathbf{Y} \tag{9.7}$$

where $^+$ denotes the *Moore-Penrose generalised matrix inverse*[2] [Weisstein, 2006c]. This corresponds to

$$\mathbf{X}^+ = (\mathbf{X}^T \cdot \mathbf{X})^{-1} \cdot \mathbf{X}^T, \tag{9.8}$$

where ^{T} denotes matrix transpose.

The angular frequency ω can now be found from α_1 using

$$\omega = \frac{\cos^{-1}(\alpha_1/2)}{t'}. \tag{9.9}$$

[2]also known as the Moore-Penrose pseudo-inverse

Finding the amplitude and phase

Once the angular frequency ω has been found we can use it in finding the amplitude and phase of the sine wave. We construct a cosine wave and a sine wave of angular frequency ω and perform least squares regression again, using equation 9.7. This time we have

$$\mathbf{X} = \begin{bmatrix} 1 & 0 \\ \cos(\omega\Delta t) & \sin(\omega\Delta t) \\ \cos(2\omega\Delta t) & \sin(2\omega\Delta t) \\ \vdots & \vdots \\ \cos((N-1)\omega\Delta t) & \sin((N-1)\omega\Delta t) \end{bmatrix}, \alpha = \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix}, \mathbf{Y} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{bmatrix}. \quad (9.10)$$

Note that using

$$\alpha_0 \cos(wt) + \alpha_1 \sin(wt) = A_1 \sin(wt + \theta_1), \quad (9.11)$$

the amplitude, A_1, of the original sine wave can be found using

$$A_1 = \sqrt{\alpha_0^2 + \alpha_1^2}. \quad (9.12)$$

The phase, θ_1, of the original sine wave can be found using

$$\theta_1 = \frac{\pi}{2} - \text{atan2}(\alpha_1, \alpha_0), \quad (9.13)$$

where atan2 is the arctangent function which takes into account the signs of both angles to determine the quadrant, as used in the C language's standard library.

9.3.2 Estimation Errors

Estimation errors become more significant at low frequencies, or with a very densely sampled data set. As the difference between consecutive data values decreases, with respect to the overall amplitude, it leads to larger errors in the angular frequency estimation. This is because the $\alpha_1/2$ term in Equation 9.9 becomes close to one, and small changes in the input result in large changes in estimation.

There is no reason to restrict ourselves to $t' = \Delta t$ in Equation 9.4. Instead let us allow $t' = k\Delta t$, where k is an integer and $k >= 1$. Different values of k were experimented with. Figure 9.1 shows the squared error of a fitted sine wave to a sine wave with varying amounts of white noise at different offsets, t'. Provided there are enough data points we have found that having a t' around $\frac{1}{4}$ of a period is best, giving a 90° phase shift between the two sine waves added in Equation 9.5. On the other hand,

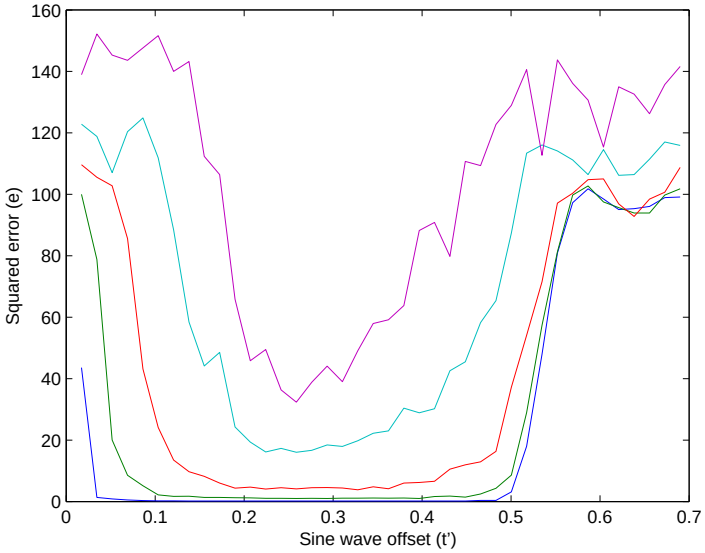


Figure 9.1: The squared error of a sine wave, fitted using different time offsets t', to a sampled sine wave buried in white noise. The lines represent a signal to noise ratio of 10:1, 4:1, 2:1, 1:1, 0.66:1 from bottom to top. This example contained 200 data points with 3.45 periods.

as k increases, the number of terms in Equation 9.5 decreases, leaving fewer terms for the regression. However, this is justified by the higher resilience to noise in the input, especially if there are sufficiently many data points. The drawback of this approach is that in order to choose the number of data points to offset, k, first an estimate of the input frequency is required. But as seen from Figure 9.1, at medium to low noise levels, this estimate does not need to be particularly accurate.

9.3.3 Allowing a Vertical Offset

The simplicity of Equation 9.4 depends on the sine wave having no DC offset, but Prony's method can be modified to allow for a y offset. This is useful in some cases, for example to find the speed of vibrato in pitch space. Here the sine wave to approximate is not centred on zero. One could apply a high pass filter to remove the DC component from the sine wave, but this would introduce extra delay into the pipeline. A vibrato can be of quite a short time span, maybe only a few cycles, making the response time loss of a filter problematic. Also, simply subtracting out the mean is not appropriate because there is no guarantee of an integer number of cycles. For example, averaging

the first 1.5 cycles of a sine wave would give a value above the sine wave's centre.

To solve this problem, a constant y-offset term is included into the angular frequency regression calculation by adding a column of ones on the left-hand side of $\mathbf{X}$ in Equation 9.5. An α_0 term is also added giving $\alpha = \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix}$, although its result is not used.

For calculation of the amplitude and phase term, a column of ones is also added on the left of $\mathbf{X}$ in Equation 9.10, with $\alpha = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{bmatrix}$. These equations work just as the ones before, but now a vertical offset is allowed and is found with the least squares error. Note that α_0 is the resulting y-offset.

9.4 Pitch Smoothing

The pitch values from the pitch detection algorithm, which are output every sample, can contain a small amount of noise variation. It can be useful to smooth out these small but fast variations using a smoothing filter before displaying them to the user. This could be done using a FIR filter, IIR filter or a moving-average filter. A low pass FIR filter is stable and can produce the necessary smoothing, however this can become fairly computationally expensive as large filter orders are needed for sufficient smoothing.

IIR filters can provide similar smoothing with a lower order, more computationally efficient algorithm. However, IIR filters are less stable than FIR filters, making them prone to numerical errors. IIR filters are useable, but can have problems when defining their delay - as they are not symmetrical.

A moving-average filter is fast to compute, but with its implied rectangular windowing function can be harsher in its smoothing. This harshness is seen when sudden changes move into the window, causing sudden jumps in the smoothed result. Again this is not ideal for smoothing the pitch values. However, we found the Hanning-weighted CMA filter described in Section 8.3 to work well, being both sufficiently stable and fast to compute.

Chapter 10

Implementation

This chapter discusses the program 'Tartini', which was initially written to develop and test the algorithms described in this thesis. However, the program grew into a full blown application which brings these tools to the end user. The application described in this chapter marks the realisation of the aim of this thesis. The functionality of the software is discussed in some detail, in which the progression from raw data to visual information becomes apparent. Having an end user application enables musicians to give direct feedback, which in turn helps steer the program design in the right direction.

Tartini is written in C++ and runs on a number of platforms including Windows, Mac OS X, and Linux. It uses a number of libraries including Qt [Trolltech, 2004], FFTW [Frigo and Johnson, 2005], RtAudio [Scavone, 2006], OpenGL and Qwt [Rathmann and Wilgen, 2005].

Tartini can perform real-time recording and analysis of sound, as well as play-back and analysis of pre-recorded sound. It can detect pitch in real-time and display this information the user in various ways. The user can choose between numerous musical scales and tuning methods to use as reference lines, as without these it is difficult to comprehend the appropriate meaning from the pitch contour. Section 10.2 provides a summary of the common musical scales and tuning methods that were implemented. Tartini can also show information about the harmonic structure of the note. This is described in Section 10.1.1. Tartini can also show further parameters about the pitch, such as vibrato speed, height and envelope shape. A further detailed pitch analysis can be performed using the incremental pitch algorithms to provide extra resolution when magnifying the pitch contour in areas of interest. However, this chapter first outlines the algorithm structure of Tartini in Section 10.1, with Section 10.3 discussing the user interface design.

10.1 Tartini's Algorithm Outline

This section discusses the implementation details of some of the non-visual aspects of Tartini. First a summary of the program structure is given, followed by certain details on the implemented tuning systems.

The following pseudo code shows the basic outline structure of the Tartini implementation.

For each frame {

- Read in *hop_size* new input samples and append them to x.
- Update y; the array of filtered values from x, using the middle/outer filter from Section 6.2.2.

- Calculate the Modified Cepstrum, from Section 6.4.1, using a window that consists of the latest W values from y.
- Warp the result and it add to the warped aggregate lag domain.
- Use the peak picking algorithm, from Section 6.3, on the aggregate lag domain to to find the pitch period estimate, P_p. Note that Equation 6.3 is used here for the peak thresholding.

- Calculate the SNAC or WSNAC function using the latest values from y. The window size can be W or less.
- Find and store all the primary-peaks using parabolic interpolation.
- Update the chosen primary-peak for every frame in current note using the new P_p - giving the pitch values at the effective centre of each frame in the note so far.

- if(doing harmonic analysis) {
  - Use the phase vocoder method, discussed in Section 10.1.1, to find the frequency and amplitude of harmonics.

  }

- if(doing detailed pitch analysis) {
  - Perform the Incremental SNAC or WSNAC using a small window based on the pitch period.
  - Smooth the pitch MIDI number values with a Hanning window CMA filter.

  }

- Calculate the vibrato parameters using the Prony method variant from Section 9.3.

- Calculate the short-term and long-term pitch mean-pitch as

```
    discussed in Section 7.3.1.
  • if(note is ending or transitioning) {
      - Do back-tracking (Section 7.3.2).
      - Update the chosen peaks in the finished note.
      - if(a new note has began) {
          - Perform forward-tracking (Section 7.3.3).
          - Update chosen peaks in the new note.
        }
    }

  • Update the display if required.
}
```

Note that the detailed pitch analysis produces a pitch value per sample. These values can be stored relative to the pitch at the effective centre of the frame. Hence, if the octave estimate for the note changes at a later stage only a single pitch period value needs to be updated for each frame in the note.

Also note the parameters used in Tartini include:

- A window size of 0.4 seconds is used for the vibrato Prony fit.
- The size of the CMA filter used to smooth the pitch values is $1/16^{th}$ of a second.

These values were deduced from the constraints involved and through direct experimentation.

10.1.1 Finding the Frequency and Amplitude of Harmonics

To find the frequency and amplitude of the harmonics in a given frame, the samples about the centre of the window are interpolated so that one pitch period now contains the number of samples n, equal to a power of two, where $n \geq P_p$. The phase vocoder method from Section 2.4.2 is used; except no windowing function is used. The coefficients 1-41 correspond to the first 40 harmonics. Note that two FFT's of size n are performed, one containing values to the left of the centre, and the other values to the right of the centre. The precise frequency of each harmonic is deduced from the difference between the measured phase advance and the expected phase advance of the given coefficient between the two FFT's. The amplitude of a harmonic is taken as the average magnitude of the given coefficients.

10.2 Scales and Tuning

The following summarises the common scales and tuning methods implemented in Tartini. This information is used to supplement the pitch information to provide a more relevant musical meaning of pitch to the user.

In music, a *scale* defines how a series of musical intervals are ordered within an octave. In Western music an octave is typically divided up into 12 semitones. For programming convenience we label these with indices 0-11, where 0 indicates the tonic note of a given key. The *chromatic* scale consists of all 12 semitone indices, whereas most other scales consist of a subset of 7 of these, such as the major and various minor scales.

The *major* scale can be described using the interval pattern T-T-S-T-T-T-S, where 'T' indicates a whole-tone, and 'S' a semitone. When a major scale is played in the key of C it uses only the white notes on the piano keyboard. In *movable do solfège*[1] the notes of the major scale are sung as the syllables "Do-Re-Mi-Fa-So-La-Ti-Do".

The *diatonic* scales are a group of scales with the same interval pattern as the major scale, but with a cyclic shift, *i.e.* starting in a different place within the pattern. The major scale forms the 1st, or Ionian, mode of the diatonic scales, and the *natural minor* scale forms the 6th, or Aeolian, mode of the diatonic scales - the two most common diatonic scales.

Other scales use different interval patterns, such as the *harmonic minor* and *ascending melodic minor* scales. A summary of the scales implemented in Tartini is given in Table 10.1. Note that the 'A' indicates an augmented second, or tone-and-a-half. In future work, we hope to implement other, less common, scales in Tartini as well.

Scale type	Interval pattern	Semitone indices
Chromatic	S-S-S-S-S-S-S-S-S-S-S-S	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, (12)
Major	T-T-S-T-T-T-S	0, 2, 4, 5, 7, 9, 11, (12)
Minor (natural)	T-S-T-T-S-T-T	0, 2, 3, 5, 7, 8, 10, (12)
Minor (harmonic)	T-S-T-T-S-A-S	0, 2, 3, 5, 7, 8, 11, (12)
Minor (ascending melodic)	T-S-T-T-T-T-S	0, 2, 3, 5, 7, 9, 11, (12)

Table 10.1: A summary of the common musical scales which are implemented in Tartini.

The scales define a systematic way of naming the notes within an octave; however,

[1] as used in most English speaking countries

it does not define the relative frequencies of these pitches. The relative frequencies of the pitches of the notes in an octave are defined by their *tuning*. A number of different tuning methods exist. The following contains a summary of the most common tuning methods, including Pythagorean tuning, just intonation, meantone temperament and even tempered.

There is no one type of tuning that is the correct one. Each tuning has its advantages and disadvantages. These main differences include the purity of the sound within intervals and chords, the power to modulate through different keys, and the practical convenience in building and playing the instruments. Barbour [1972] and Barton [1908] discuss in some depth different tuning methods.

Before the era of modern turning, a monochord was often used to create and use a tuning system. A monochord is a string stretched over a wooden base with lines indicating lengths for some tuning system. These lines were constructed geometrically to make the desired ratios for a scale.

An interval is the distance stepped in pitch from one note to another. The size of the interval is defined by the ratio of the frequencies of its component notes and if this is a ratio of small whole numbers the interval is usually held to be pleasing; for example, the octave having a ratio of 2:1, and the perfect fifth having a ratio of 3:2. In contrast, a ratio of say 540:481 is more dissonant, and tends have a harsher sound.

Pythagorean tuning

Pythagorean tuning is based upon the first two intervals of the harmonic series, the octave and the fifth. It involves successively tuning by fifths, for example *C*-*G*, *G*-*D*, *D*-*A* and dropping the note by an octave when necessary to bring it back into the range of the scale. The result is that all the fifths, starting from a given note, are perfect except one. This is because this process can never return exactly to the original note - hence one dissonant fifth is required to complete a scale. The dissonant fifth is called a wolf fifth [Barbour, 1972]. As a result this tuning method provides reasonable freedom to move between keys whilst maintaining pleasant sounding fifths. The values in Table 10.2 have been achieved from starting at C and going upwards by successive fifths, C, G, D, A, E, B, $F^{\sharp}$; and going downwards to arrive at F, $B^{\flat}$, $E^{\flat}$, $A^{\flat}$, $D^{\flat}$. Here the wolf fifth is the interval $F^{\sharp}$- $D^{\flat}$ which is left sounding badly out of tune. However, the wolf fifth can be made to appear at any part of the scale.

Just intonation

Just intonation, or the "just" scale is founded on the first five unique intervals of the harmonic series - the octave, fifth, fourth, major third and minor third [Barbour, 1972], and usually described with three overlapping major triads. There is no standard way of tuning all of the notes using this method; however, Table 10.2 contains some commonly used just intervals. Very pleasant sounding chords and intervals are achieved by this system. The process of changing key becomes less well defined, and usually requires adjustment of the pitch at certain positions - making it impractical for some instruments such as the piano. Just intonation is often used in unaccompanied singing, for example *a Capella* groups, and fretless string instruments such as the violin - where it can be used together in a small group. However, just intonation is not well suited for large-scale orchestras, or a singer with instrumental accompaniment - such as a piano.

Meantone temperament

Meantone temperament is usually defined as having a perfect major third; for example, C-E in ratio 4:5 for C-major. Then the 2^{nd} is defined as exactly half the number of cents as between the 1^{st} and the 3^{rd}, thus the ratio C-D is the same as D-E, *i.e.* the mean of the two tones on a logarithmic scale. The fifth, here C-G, is made slightly flattened from perfect, and the intervals G-D, D-A and A-E share the same ratio as this. Meantone temperament was used from the beginning upon keyboard instruments only. In the more general use of the word, a meantone temperament can have any choice of third - provided the other rules are followed.

Even tempered

The *even tempered* method, also called equal temperament, is the most common tuning method used today in western music. It consists of dividing the octave equally into twelve semitones. Hence, all semitones are described by the frequency ratio $1 : \sqrt[12]{2}$. Furthermore, a *cent* is defined as $1/100^{th}$ of an even tempered semitone (using the logarithmic scale). None of the intervals in even temperament form a perfect ratio. However, this tuning allows a musician to modulate to any key without having any intervals which are a long way from perfect. Today, people have become accustomed to hearing these slightly imperfect intervals, whereas in the 16^{th} century they were generally considered unpleasant.

A summary of tuning systems is shown in Table 10.2.

Semi-tone index	Note name	Interval (from tonic)	Pythagorean tuning ($D^\flat$to $F^\sharp$)	Just intonation	Meantone (cents)	Even temp-erament (cents)
0	C	1st (Tonic)	1:1	1:1	0	0
1	$C^\sharp$ ($D^\flat$)	Minor 2nd	256:243		76	100
2	D	Major 2nd	9:8	9:8	193	200
3	$D^\sharp$ ($E^\flat$)	Minor 3rd	32:27	(6:5)	310	300
4	E	Major 3rd	81:64	5:4	386	400
5	F	4th (Subdominant)	4:3	4:3	503	500
6	$F^\sharp$ ($G^\flat$)	Augmented 4th	729:512	(25:18)	579	600
7	G	5th (Domanent)	3:2	3:2	697	700
8	$G^\sharp$ ($A^\flat$)	Minor 6th	128:81	(8:5)	773	800
9	A	Major 6th	27:16	5:3	890	900
10	$A^\sharp$ ($B^\flat$)	Minor 7th	16:9		1007	1000
11	B	Major 7th	243:128	15:8	1083	1100
12	C	Octave	2:1	2:1	1200	1200

Table 10.2: A summary of the common tuning systems based on Helmholtz [1912], Jeans [1943] and Barbour [1972]. Here a C scale is used as an example for reference.

10.3 User Interface Design

Tartini's interface consists of a series of different widgets, or sub-windows, which each display output from different parts of the analysis. Each of these widgets will be discussed in turn.

10.3.1 File List Widget

A tool which allows a musician to play along with a pre-recorded song and see an immediate comparison of their pitches, as well as allowing a comparison between existing sound files, could prove useful. Tartini fulfils this role with the 'Play and Record' mode in which the user listens through headphones and plays into the microphone simultaneously for live comparisons. To achieve this, Tartini has support for multiple sound files to be open at the same time. The user can select the channels which are displayed through the File List widget. Figure 10.1 shows a screenshot of the File List widget containing a number of open files. For files with multiple channels, *e.g.* stereo sound,

each channel is opened and analysed separately. Check-boxes allow the user to show or hide any given channel as they desire. A single channel is marked with an 'A' in the last column. This represents the active channel. Most user actions throughout the program will apply only to the active channel. Widgets which can show information only about a single channel show the active channel.

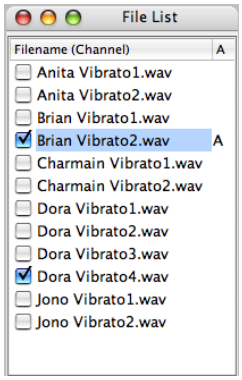


Figure 10.1: A screenshot of the File List widget running on Mac OS X.

10.3.2 Pitch Contour Widget

The Pitch Contour widget is one of the most important widgets in Tartini. It plots the pitch over time, as calculated from the lower/mid level pitch analysis discussed in Chapters 4 to 7. A screenshot of the Pitch Contour widget is shown in Figure 10.2. The following discusses features of the Pitch Contour widget which were employed to help facilitate the representation of the information to the user in a useful and convenient manner.

Horizontal lines are drawn in the background to indicate the pitch of notes on the selected musical scale. The names of the notes are indicated at the left-hand side of each line along with the octave number. A choice of reference scales is given to the user, as discussed in Section 10.2, which currently includes chromatic, major, natural minor, harmonic minor and ascending melodic minor. This allows a musician to display only the reference lines of a desired key, removing the excess clutter of lines. Moreover, a choice of temperament, as discussed in Section 10.2, is given to the user, which currently includes: even tempered, just intonation, Pythagorean tuning and meantone temperament. This allows a musician to set the exact size of the intervals between the notes in the scale to suit their needs. Note that the reference lines shown in the figure are chromatic and even tempered. Since Tartini version 1.2, the ability to offset the

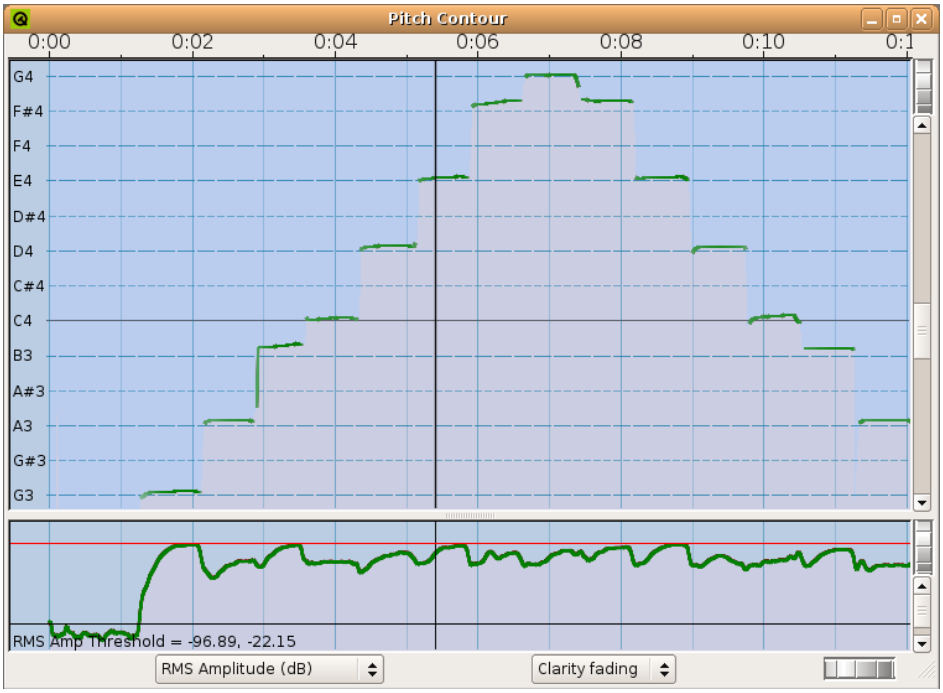


Figure 10.2: A screenshot of the Pitch Contour widget running on Linux showing a G major scale as played on violin. It can be seen that the 3^{rd} and 7^{th} (notes B and $F^{\sharp}$) are played quite sharp of the even tempered scale.

scales was introduced due to user requests. This is done by letting the user choose the frequency of $A4$, which is initially set to 440 Hz. Note that some scales do not contain the note A, so the offset is such that the tonic, or base note, of a key is taken from the chromatic even tempered scale which has an $A4$ set. However, there could be other ways of dealing with this; for instance, letting the user choose the note and frequency in order to define the scale offset.

The clarity measure discussed in Section 4.4 is put to use in the Pitch Contour widget as a parameter to control pitch-fading, or the level of opacity of the lines. Sound which has a loud and clearly defined pitch is shown completely opaque, in contrast to sound that is softer and more noise-like which becomes increasingly transparent. The primary use of this feature is to reduce the clutter of unwanted background sounds. For instance, in between two notes is the faint sound of a chair squeaking. Even though Tartini may be able to calculate the pitch for the squeak, it is likely the user is not so interested in this sound. It is therefore classified as being less significant and drawn almost invisibly. Hence, the two important notes will stand out to the user. However, a side effect of this is that a quiet instrument causes the pitch contour to become very

transparent. Tartini therefore allows pitch-fading to be disabled, so all notes are drawn with full opacity.

As well as the opacity of the pitch contour being an indication of volume, a volume curve, or more precisely the RMS value of the volume, is plotted on a logarithmic scale underneath the pitch contour view with its values time aligned with the pitch. Vertical reference lines were added in Tartini version 1.2 which are drawn in the background. At a quick glance the exact volume of a certain pitch can be realised by scanning one's eyes down the line. In comparison, programs such as Intonia [Agin, 2007] increase the thickness of the line with an increased volume; however, this tends to cause a loss in precision of the exact centre, whereas Melodyne [Celemony Software] uses a pitch line surrounded by a multi-coloured amplitude envelope. Melodyne's technique works well, although it could get very cluttered when multiple channels are overlaid.

Another feature of Tartini's Pitch Contour interface is pitch-underfill, in which the area underneath the pitch-contour is coloured slightly differently from the area above. This provides the user with an indication of the vertical direction to the pitch-contour when it is outside the current viewing area. The user immediately knows whether to scroll up or down to see the notes the interest, thus reducing the chance of getting lost when zoomed in on a large file. Moreover, pitch-underfill becomes useful to identify the active channel's pitch-contour amongst the other channels. Tartini allows the user to choose the colours of the areas, or disable the pitch-underfill completely if desired.

Auto Following is a feature which allows the musician to play their instrument with both hands, freeing them from the need to scroll up and down using the mouse/keyboard. The idea is that the view scrolls up and down automatically in a smooth fashion to keep the pitch contour close to the centre of the view. The pitch on which the view should be centred is based on a weighted average of the pitch around the current time. One limitation of this is if consecutively large intervals in pitch are played which contain notes that are further apart than the height of the view, then the view will be moved to look between them - making neither of them visible. One solution could be to automatically zoom out, however this requires zooming back in at some stage, and raises the issue of when, and how much, to zoom in. A number of users report that the Auto Following feature just makes them feel dizzy, and has since been dubbed the 'sea sickness' effect. A zoom feature is likely to disorient the users even more. Nevertheless, it is possible that other methods of automatically following the pitch could work better.

10.3.3 Chromatic Tuner Widget

The Chromatic Tuner widget was made to provide a familiar tool for musicians. However, this tuner utilises the added responsiveness of the almost instantaneous pitch detection of the SNAC-type algorithms. As a result, tuning an instrument can become quite efficient. Figure 10.3 shows a screenshot of the Chromatic Tuner widget. The slider allows the user to smooth out the pitch using weighted averaging to reduce the responsiveness if needed. This can be helpful in certain situations where there are small fluctuations inherent in the pitch.

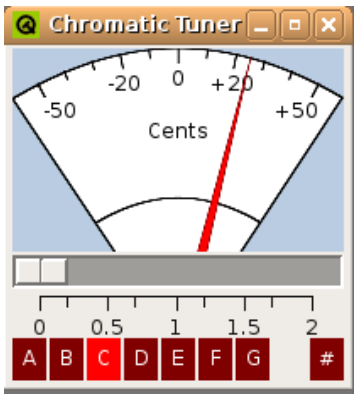


Figure 10.3: A screenshot of the Chromatic Tuner widget running on Linux, showing a pitch 25 cents above note C. The slider indicates no pitch smoothing (the default).

The highlighted letter indicates the nearest defined note with the dial indicating how far the pitch is from that note. The needle indicates the pitch with its angle, making it easily understood by the user at varying distances from the screen, as even though the user's perceived size of an object changes with distance, angles remain constant.

The tuner shows no sign of octave, making it only a reduced pitch problem. Even if the pitch measurement is off by an octave the tuner will still show the correct reading.

Currently the Chromatic Tuner only tunes to the even tempered scale; however, its usage with other temperaments is likely to be implemented in the near future.

10.3.4 Vibrato Widget

The Vibrato widget is a new interface aiming to display the information available from the vibrato analysis discussed in Chapter 9. The goal of this widget is to show detailed information about the vibrato of the current note, providing a tool for future musicians

and researchers to investigate the use of vibrato more thoroughly, including its musical and psychological effects.

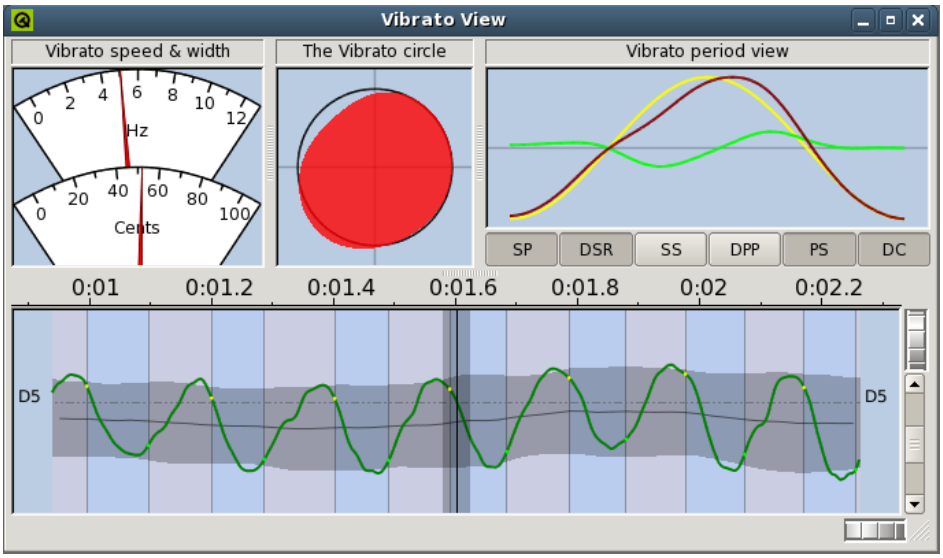


Figure 10.4: A screenshot of the Vibrato widget showing some violin vibrato around a $D5$. Notice how the dials indicate a vibrato speed of 5 Hz, and a vibrato width of 51 cents.

Figure 10.4 shows a screenshot of the Vibrato widget. The lower half of the widget shows the variation with time of the pitch of the current note at a higher magnification than the Pitch Contour widget. A grey line indicates the vibrato's moving centre pitch, as calculated from the Prony offset in Section 9.3.3.

The dials in the upper left show the instantaneous speed and width of the vibrato. This is useful for practising consistency or trying to match a target vibrato. A musician can see how their changes in playing affect these important parameters, watching them increase and decrease over time. Future work might investigate using a two dimensional plot for these parameters instead of dials, allowing the user to see the history of changes throughout the note all at once.

The top right of the widget shows the vibrato period view. This contains an extreme magnification of the vibrato pitch, drawn in red, with its trace always starting from the minimum part of the cycle, and its amplitude normalised. Initially the display method consisted of drawing the current pitch cycle as it was being played, making a tracing effect across the view. However, this was not found to be helpful as it changed too quickly, distracting the user. It was found that drawing only the completed cycles worked better. Other reference lines were added, including a grey centre line and a yellow curve indicating the phase of a sinusoid with the same frequency. The user can

compare how the phase of their vibrato cycle advances or recedes relative to a perfect sinusoidal shape. This vertical difference, called the sine-difference, is also indicated by the green curve.

The centre top of the widget shows the vibrato circle, a novel idea for displaying vibrato phase information. The basic concept is that over the course of one vibrato cycle, 360 degrees of a circle are carved out. The radius of the circle at any given angle is offset by the sine-difference described above. For instance, a perfectly sinusoidal vibrato would produce a perfect circle, whereas a slower curve up and faster curve down would produce a lop sided, egg-like shape. In fact, any pattern of sine-difference creates a unique solid shape. It was thought that a solid shape might be easier for a user to comprehend than the sine-difference curve, although so far this has not been tested. However, variations between successive vibrato cycles could be clearly seen. Finally, the idea of morphing the circle from one shape to the next was introduced to reduce the sudden jumping between cycles, creating a more pleasant user experience. The vibrato circle and the sine-difference concept are still at an early experimental stage. As future work, we hope to investigate ways of displaying the volume variations which are often associated with vibrato, in conjunction with the pitch variations.

10.3.5 Pitch Compass Widget

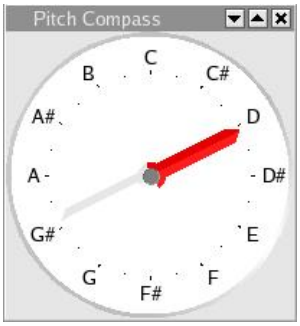


Figure 10.5: A screenshot of the Pitch Compass widget running on Linux.

The Pitch Compass widget was a novel idea that shows the instantaneous pitch as a direction needle on a compass. Its use is intended for people, especially children, who are learning music. The use of a circle helps indicate the repeating nature of notes in an octave. Figure 10.5 show a screenshot of the Pitch Compass widget. Initially, the needle remained fixed upward with the note names revolving around the outside. However, it was later found that rotating the needle whist keeping the note names fixed

seemed better at giving a sense of pitch bearing without disorienting the user. Notice how the Pitch Compass has no concept of the pitch's octave in a similar manner to the Chromatic Tuner widget in Section 10.3.3, which removes some complexity and emphasises the similarity of the same note in different octaves.

10.3.6 Harmonic Track Widget

The Harmonic Track widget is an attempt to bring the information about harmonic structure into a realm familiar to musicians. Figure 10.6 shows a 3D piano keyboard with the current note depressed and highlighted yellow. Protruding from the back of the keyboard are vertical walls, or tracks. Each track represents a harmonic frequency component of the note. Over time the tracks move further away from the keyboard, with the frontmost height of a track representing the strength of the harmonic at the current time. Odd harmonics are coloured blue and even harmonics green for ease of distinguishing tracks.

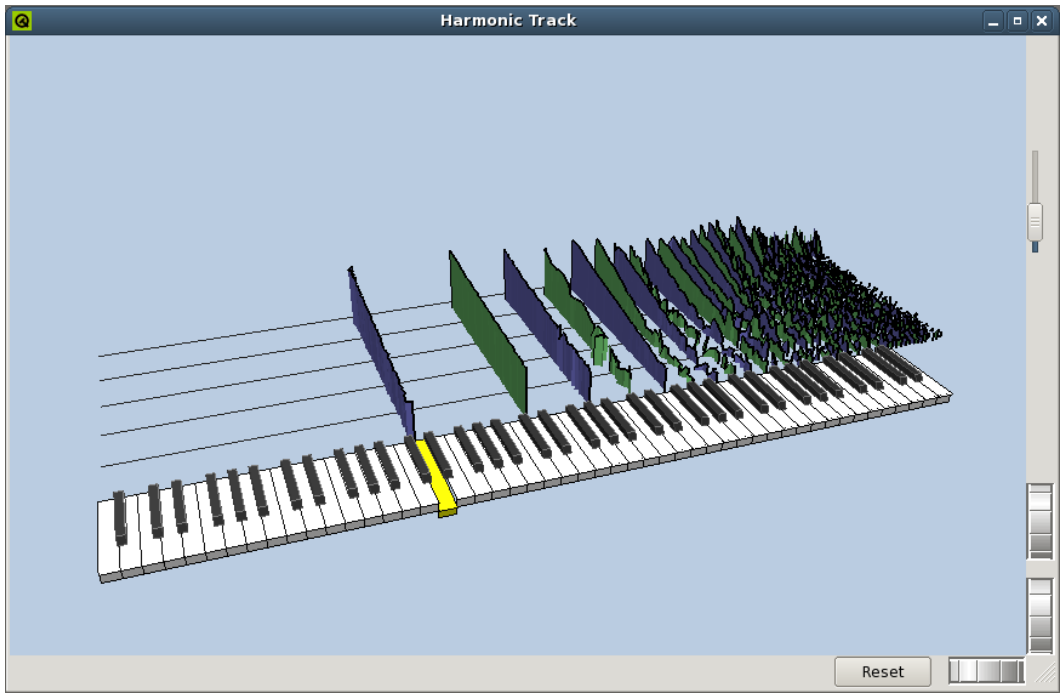


Figure 10.6: A screenshot of the Harmonic Track widget running on Linux.

Note that the second harmonic is one octave above the first, and the fourth harmonic is one octave above the second, and so on. The natural logarithmically scaled frequency axis of the keyboard makes the linear frequency spacing of the harmonics

appear increasingly close together. Figure 10.6 contains lower harmonics which are stronger and higher harmonics which fading into the noise level at the right-hand side. This is a typical pattern; however, different instruments produce different patterns.

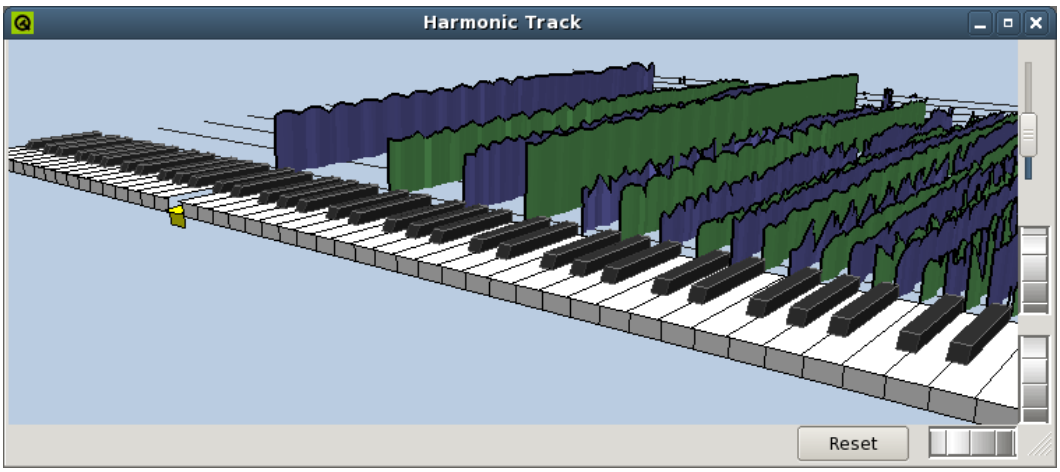


Figure 10.7: A screenshot of the Harmonic Track widget containing cello with vibrato.

Figure 10.7 shows the ability to change the camera's viewing angle. Note that the tracks look corrugated, depicting the vibrato, and some tracks take on height variations due to an induced tremolo affect on certain harmonics.

10.3.7 Musical Score Widget

The Musical Score widget provides a very basic representation of the notes played. There is an option to transpose the notes up or down an octave, and an option to hide any extreme notes. Also there is an option to draw notes with translucency based on their clarity measure, similar to pitch-fading in Section 10.3.2. A screenshot of the Musical Score widget is shown in Figure 10.8.

The widget was made to show the potential usage of the higher level information from the algorithms in this thesis, such as a note's beginning, length and average pitch. However, further work on combining the pitch techniques in this thesis with other note onset/offset methods are required for a more robust note detection scheme. Future work could include producing MIDI output that contains the expressive nature of pitch and volume variations within the notes.

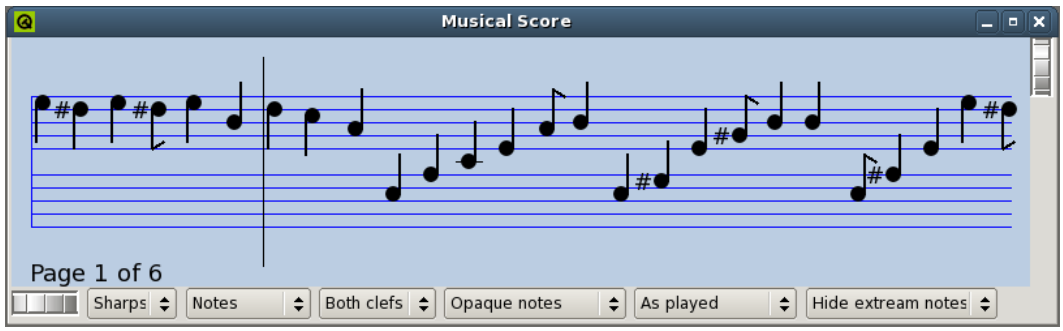


Figure 10.8: A screenshot of the Musical Score widget running on Linux.

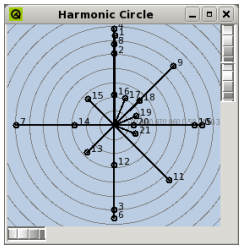
10.3.8 Other Widgets

Tartini contains a number of other widgets which display some of the more technical information contained within the analysis phases. These include:

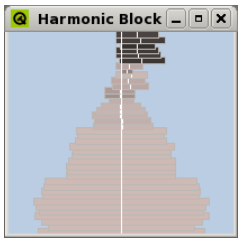
- An Oscilloscope widget which displays the waveform data of the current window as used for analysis.
- An FFT widget which shows the frequency spectrum of the current window.
- A Correlation widget which shows the result of the current autocorrelation-type function.
- A Cepstrum widget which shows the cepstrum, or modified cepstrum, of the current window.

Tartini also contains some more experimental widgets including:

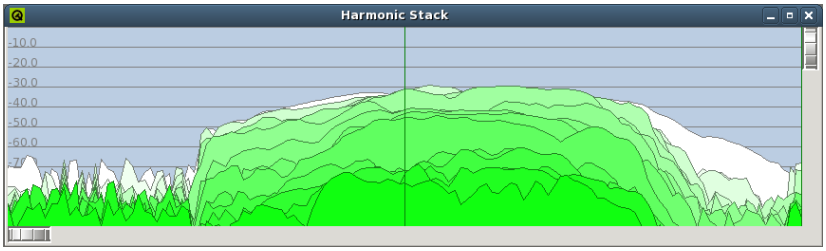
- A Harmonic Circle widget which shows radial spikes indicating the instantaneous normalised harmonic structure. The pitch frequency points up, with each 360 degrees clockwise representing an octave. The length of the lines indicate the strength of a harmonic in decibels. As a frequency component becomes flat or sharp of the expected harmonic frequency its angle changes with respect to the expected angle.



- A Harmonic Block widget which shows the instantaneous strength of each harmonic. The width of the lowest rectangle, or block, indicates the strength of the first harmonic; higher up blocks represent increasingly higher harmonics. Blocks offset from the centre line indicate that a harmonic is flatter or sharper than its expected frequency at an integer multiple of the fundamental frequency.



- A Harmonic Stack widget which shows how the strength of all the harmonics of a note change over time. The first harmonic is white, with higher harmonics gaining a darker green. Horizontal lines indicate the amplitude in decibels.



Note that around 5% of Tartini's code was produced by other authors, and thanks go to Rob Ebbers, Stuart Miller and Maarten van Sambeek for their contributions to the programming of graphical elements within the Vibrato widget, Pitch Compass widget, Harmonic Circle widget and Harmonic Stack widget.

Chapter 11

Conclusion

This thesis describes the first step in a larger project to provide an automatic analysis of many aspects of musical sound that would be of practical use to musicians. We set out to improve existing techniques for measuring pitch and its variations that can work well over a range of musical instruments, with the goal to develop a real-time tool which can aid musicians. We believe these goals have been achieved, although we have not yet conducted formal studies to prove the tool can aid musicians.

Our main contribution is in the development of algorithms to find the continuous musical pitch in a fast, accurate and robust manner. These algorithms have been shown to work well over a range of instruments through both systematic testing on sound databases such as the Iowa dataset, and through their practical use by instrumentalists. Also we have designed a method for measuring the parameters of vibrato, *i.e.* higher level information about repeating pitch variations. Our work is summarised in the following material.

We have developed a *special normalisation of the autocorrelation* (SNAC) function and *windowed SNAC* (WSNAC) function described in Chapter 4. These functions were shown in Chapter 5 to measure the short-time periodicity of signals with greater accuracy than existing autocorrelation methods. With a steady signal the pitch can be detected accurate to less than 0.1 cents with only 2 cycles. The algorithms were stress tested under fast variations in pitch, volume and different harmonic structure and maintained an accuracy within 5 cents. Chapter 4 also shows how these new functions can be calculated efficiently through use of the fast Fourier transform (FFT).

We developed a *modified cepstrum*, a *warped aggregate lag domain* (WALD) and a *peak picking* algorithm described in Chapters 6 and 7. When these are combined with a middle/outer ear filter, the algorithms achieved a 99.88% success rate of detecting

the correct octave associated with the perceived pitch on our test data. This combination, together with the SNAC or WSNAC function methods, can make accurate measurements of a varying musical pitch.

We describe in Section 7.3.1 a method for detecting note changes in a channel based on the pitch information. This method can detect note changes even when the volume is constant. It is designed to be used in conjunction with other note onset/offset detectors. Further experimentation is desirable to find to what extent this idea can improve existing methods. In the future we hope to implement a more robust note onset/offset detection system into Tartini.

We describe in Chapter 8 how the SNAC and WSNAC functions can be calculated incrementally - resulting in an efficient calculation of the pitch values at a sampling rate equal to the input sampling rate.

The algorithms described have proved effective at finding the pitch of single 'voiced' sounds across a range of instruments; however, in practice it can be difficult to restrict the user to a single voice on certain instruments, such as guitar - where plucking a series of single notes can leave several strings ringing at the same time. These types of polyphonic sounds can cause unexpected results, such as Tartini-tones or octave errors.

The *complex moving-average* (CMA) filter we developed is an algorithm for efficiently convolving certain large windows to smooth a signal. This method, described in Section 8.3, could have application in many fields of research.

In Chapter 9 we present a method for estimating the musical parameters of vibrato over a short-time, including the vibrato's speed, height, phase and centre offset. Using this method the shape of a vibrato's envelope can be found as it changes throughout the duration of a note.

We have created an application called 'Tartini' that implements these techniques into a tool for musicians and made it freely available. We have also shown numerous ways to display the pitch information to the user, providing musicians, researchers and teachers with new tools to help advance their fields.

Tartini can be found at `http://www.tartini.net` and has already had over 9000 unique visitors to the web site - as of April 2008, including some reputable musicians. Note that the most recent version of Tartini, which contains all the algorithms discussed in this thesis, will be released to the public on the conclusion of the marking process.

Some published parts of the Tartini algorithm have been ported into third party programs - this includes a SuperCollider [McCartney, 2006] plugin, thanks to Nick Collins [Collins].

In future research we would like to conduct further studies into how and what tools the Tartini users find most useful, and whether it helps them to learn faster. This also leads to the inclusion of teaching material, such as basic musical exercises and tutorials into Tartini.

With a solid foundation of pitch, we would like to see further investigation into the relationship between pitch variations and volume variations during vibrato, and how a musician can use these to control the emotion portrayed in a sound.

References

Antares 'Auto-Tune 5', 2007. URL `http://www.antarestech.com`.

Jerry Agin. Intonia, 2007. URL `http://intonia.com/`.

Rasmus Althoff, Florian Keiler, and Udo Zölzer. Extracting sinusoids from harmonic signals. In *Proceedings of the Digital Audio Effects (DAFx) Workshop*, pages 97–100, Trondheim, Norway, December 1999.
URL `http://citeseer.ist.psu.edu/althoff99extracting.html`.

James Murray Barbour. *Tuning and Temperament*. Da Capo Press, New York, 1972.

Edwin H. Barton. *A Text-book on Sound*. MacMillan and Co., Limited, London, 1908.

Juan Bello. *Towards the Automated Analysis of Simple Polyphonic Music: A Knowledge-based Approach*. PhD thesis, Department of Electronic Engineering, Queen Mary, University of London, 2003.

B. P. Bogert, M. J. R. Healy, and J. W. Tukey. The quefrency alanysis of time series for echoes: cepstrum, pseudo-autocovariance, cross-cepstrum, and saphe cracking. In *Proceedings of the Symposium on Time Series Analysis (M. Rosenblatt, Ed) Chapter 15*, pages 209–243. New York: Wiley, 1963.

Paul Brossier, Juan Pablo Bello, and Mark D. Plumbley. Real-time temporal segmentation of note objects in music signals. In *Proc. International Computer Music Conference*, Florida, USA, November 1-6 2004.

Jean Callaghan, William Thorpe, and Jan van Doorn. The science of singing and seeing. In *Proceedings of the Conference on Interdisciplinary Musicology (CIM04)*, Graz, Austria, 15-18 April 2004.

CantOvation Ltd. Sing & See, 2007. URL `http://www.singandsee.com`.

Neil R. Carlson and William Buskist. *Psychology: The Science of Behavior*. Allyn and Bacon, 5th edition, 1997.

Leonardo Cedolin and Bertrand Delgutte. Pitch of complex tones: Rate-place and interspike interval representations in the auditory nerve. *Journal of Neurophysiology*, 94:347–362, 2005.

Celemony Software. Melodyne, 2007. URL `http://celemony.com`.

Nick Collins. Tartini supercollider plugin. URL `http://www.informatics.sussex.ac.uk/users/nc81/code/tartini.tar.gz`.

James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301, 1965.

D Cooper and K C Ng. A monophonic pitch tracking algorithm. Technical Report 94.15, School of Computer Studies, The University of Leeds, UK, 1994. URL `http://citeseer.ist.psu.edu/313255.html`.

A. de Cheveign and H. Kawahara. Yin, a fundamental frequency estimator for speech and music. *The Journal of Acoustical Society America*, 111(4):1917–1930, April 2002.

D. Deutsch. An auditory paradox. *Journal of the Acoustical Society of America*, 80 (S1):S93, December 1986.

Marek Dziubiński and Bożena Kostek. High accuracy and octave error immune pitch detection algorithms. In *Archives of Acoustics*, volume 29, pages 1–21, 2004.

Epinoisis Software. Digital ear, 2007. URL `http://www.digital-ear.com`.

John Fitch and Wafaa Shabana. A wavelet-based pitch detector for musical signals. In *Proceedings 2nd COSTG6 Workshop on Digital Audio Effects DAFx99*, 1999. URL `http://citeseer.ist.psu.edu/395242.html`.

Matteo Frigo and Steven G. Johnson. Fastest Fourier transfom in the west, 2005. URL `http://www.fftw.org/`.

B. Gold and L. R. Rabiner. Parallel processing techniques for estimating pitch periods of speech in the time domain. *Journal of the Acoustical Society of America*, 46(2, Pt 2):442–448, August 1969.

Amalia De Götzen, Nicola Bernardini, and Daniel Arfib. Traditional (?) implementations of a phase-vocoder: The tricks of the trade. In *Proceedings of the COST G-6 Conf. on Digital Audio Effects (DAFX-00)*, Verona, Italy, December 7-9 2000.

S. Griebel. Multi-channel wavelet techniques for reverberant speech analysis and enhancement. Technical Report 5, HIMMEL, Harvard University, Cambridge, MA, Feb 1999. URL `http://citeseer.ist.psu.edu/griebel99multichannel.html`.

F. Harris. On the use of windows for harmonic analysis with the discrete Fourier transform. *Proc. of the IEEE*, Vol 66(No 1), Jan 1978.

Hermann von Helmholtz. *On the Sensations of Tone as a Physiological Basis for the Theory of Music*. London, New York, Longmans Green, 4th edition, 1912. Trans. A. J. Ellis.

Dik J Hermes. Measurement of pitch by subharmonic summation. *The Journal of the Acoustical Society of America*, 83(1):257–264, January 1988.

F. B. Hildebrand. *Introduction to Numerical Analysis*. New York: McGraw-Hill, 1956.

Masanao Izumo and Tuukka Toivonen. Timidity++, 2007.
URL `http://timidity.sourceforge.net`.

G Jacovitti and G Scarano. Discrete time techniques for time delay estimation. *IEEE Trans. on Signal Processing*, 41(2):525–533, February 1993.

Sir James Jeans. *Science and Music*. Cambridge University Press, 1943.

Joint Photographic Experts Group. Jpeg 2000 standard, 2000.
URL `http://www.jpeg.org/jpeg2000/`.

S. M. Kay and JR. S. L. Marple. Spectrum analysis - a modern perspective. In *Proceedings of the IEEE*, volume 69, pages 1380 – 1419. IEEE, November 1981.

Florian Keiler and Sylvain Marchand. Survey on extraction of sinusoids in stationary sounds. In *Proc. of the 5th Int. Conference on Digital Audio Effects (DAFx-02)*, Hamburg, Germany, September 26-28 2002.

Sylvain Marchand. An efficient pitch-tracking algorithm using a combination of Fourier transforms. In *Proceedings of DAFX-01*, Limerick, Ireland, 2001.
URL `http://citeseer.ist.psu.edu/marchand01efficient.html`.

Sylvain Marchand. Improving spectral analysis precision with enhanced phase vocoder using signal derivatives. In *Proc. DAFX98 Digital Audio Effects Workshop*, pages 114–118, Barcelona, November 1998.
URL `http://citeseer.ist.psu.edu/marchand98improving.html`.

M. Marolt, A. Kavcic, and M. Privosnik. Neural networks for note onset detection in piano music. In *Proc. Int. Computer Music Conference*, Gothenberg, Sweden, 2002.
URL `citeseer.ist.psu.edu/marolt02neural.html`.

MathWorks. Matlab, 2006. URL `http://www.mathworks.com/`.

James McCartney. Supercollider, 2006.
URL `http://supercollider.sourceforge.net/`.

Philip McLeod and Geoff Wyvill. Visualization of musical pitch. In *Proceedings of the Computer Graphics International*, pages 300–303, Tokyo, Japan, July 9-11 2003.

Philip McLeod and Geoff Wyvill. A smarter way to find pitch. In *Proceedings of the International Computer Music Conference*, pages 138–141, Barcelona, Spain, September 5-9 2005.

Adriano Mitre, Marcelo Queiroz, and Regis R. A. Faria. Accurate and efficient fundamental frequency determination from precise partial estimates. In *Proceedings of the 4th AES Brazil Conference*, pages 113–118, 2006.

Brian C. J. Moore. *An Introduction to the Psychology of Hearing*. Academic Press, forth edition, 2003.

R Nave. Equal-loudness curves, 2007.
URL `http://hyperphysics.phy-astr.gsu.edu/hbase/sound/eqloud.html`.

Truong Q. Nguyen. A tutorial on filter banks and wavelets. In *International Conference of Digital Signal Processing*, Cypress, Giugno, 1995.
URL `http://citeseer.ist.psu.edu/nguyen95tutorial.html`.

A. Michael Noll. Cepstrum pitch determination. *The Journal of the Acoustical Society of America*, Vol. 41:239–309, February 1967.

R. Plomp. Pitch of complex tones. *The Journal of Acoustical Society America*, 41: 1526–1533, 1967.

L. R. Rabiner. On the use of autocorrelation analysis for pitch detection. *IEEE Trans. Acoustics, Speech, and Signal Processing*, ASSP-25(1):24–33, February 1977.

L. R. Rabiner and R. W. Schafer. *Digital Processing of Speech Signals*. Prentice-Hall Signal Processing Series. Prentice Hall, 1978.

Uwe Rathmann and Josef Wilgen. Qt widgets for technical applications, 2005. URL `http://qwt.sourceforge.net/`.

David Robinson. Equal-loudness antenuation filter design, matlab code, 2001. URL `http://www.David.Robinson.org`.

Xavier Rodet. Musical sound signals analysis/synthesis: Sinusoidal+residual and elementary waveform models. *Applied Signal Processing*, 4:131–141, 1997.

Myron J Ross, Harry L Shaffer, Andrew Cohen, Richard Freudberg, and Harold J Manley. Average magnitude difference function pitch extractor. *IEEE Trans. Acoustics, Speech, and Signal Processing*, ASSP-22(5):353–362, October 1974.

Thomas D. Rossing. *The Science of Sound*. Addison-Wesley, second edition, 1990.

Gary P. Scavone. Rtaudio, 2006. URL `http://www.music.mcgill.ca/~gary/rtaudio/`.

Raymond A. Serway. *Physics for Scientists and Engineers with Modern Physics*. Saunders College Publishing, 4th edition, 1996.

M. M. Sondhi. New methods of pitch extraction. *IEEE Trans. Audio and Electroacoustics*, AU-16(2):262–266, June 1968.

Sony Computer Entertainment Europe. Singstar, 2007. URL `http://www.singstargame.com`.

Xuejing Sun. A pitch determination algorithm based on subharmonic-to-harmonic ratio. In *The 6th International Conference of Spoken Language Processing*, pages 676–679, Beijing, China, 2000. URL `http://citeseer.ist.psu.edu/sun00pitch.html`.

Xuejing Sun. Pitch determination and voice quality analysis using subharmonic-to-harmonic ratio. In *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing*, Orlando, Florida, May 2002. URL `http://citeseer.ist.psu.edu/sun02pitch.html`.

Trolltech. Qt, 2004. URL http://trolltech.com/products/qt.

Martin Vetterli and Cormac Herley. Wavelets and filter banks: Theory and design. *IEEE Trans. on Signal Processing*, 40(9):2207–2232, September 1992.

Eric W. Weisstein. Convolution theorem. From MathWorld - A Wolfram Web Resource, 2006a. URL http://mathworld.wolfram.com/ConvolutionTheorem.html.

Eric W. Weisstein. Wiener-khinchin theorem. From MathWorld - A Wolfram Web Resource, 2006b.
URL http://mathworld.wolfram.com/Wiener-KhinchinTheorem.html.

Eric W. Weisstein. Moore-penrose matrix inverse. From MathWorld - A Wolfram Web Resource, 2006c.
URL http://mathworld.wolfram.com/Moore-PenroseMatrixInverse.html.

Pat H Wilson, Kerrie Lee, Jean Callaghan, and C W Thorpe. Learning to sing in tune: Does real-time visual feedback help? In *CIM07: 3rd Conference on Interdisciplinary Musicology*, Tallinn, Estonia, 15-19 August 2007.

Alexander Wood. *The Physics of Music*. Methuen, London, 1944.

Appendix A

Pitch Conversion Table

Table A.1: Pitch conversion table

Pitch	MIDI note number	Frequency (Hz)	Period (samples at 44100 kHz)
$A0$	21	27.50	1603.64
$A^\sharp 0$	22	29.14	1513.63
$B0$	23	30.87	1428.68
$C1$	24	32.70	1348.49
$C^\sharp 1$	25	34.65	1272.81
$D1$	26	36.71	1201.37
$D^\sharp 1$	27	38.89	1133.94
$E1$	28	41.20	1070.30
$F1$	29	43.65	1010.23
$F^\sharp 1$	30	46.25	953.53
$G1$	31	49.00	900.01
$G^\sharp 1$	32	51.91	849.50
$A1$	33	55.00	801.82
$A^\sharp 1$	34	58.27	756.82
$B1$	35	61.74	714.34
$C2$	36	65.41	674.25
$C^\sharp 2$	37	69.30	636.40
$D2$	38	73.42	600.68
$D^\sharp 2$	39	77.78	566.97
$E2$	40	82.41	535.15

Continued on next page

Table A.1 – continued from previous page

Pitch	MIDI note number	Frequency (Hz)	Period (samples at 44100 kHz)
$F2$	41	87.31	505.11
$F\sharp 2$	42	92.50	476.76
$G2$	43	98.00	450.01
$G\sharp 2$	44	103.83	424.75
$A2$	45	110.00	400.91
$A\sharp 2$	46	116.54	378.41
$B2$	47	123.47	357.17
$C3$	48	130.81	337.12
$C\sharp 3$	49	138.59	318.20
$D3$	50	146.83	300.34
$D\sharp 3$	51	155.56	283.49
$E3$	52	164.81	267.57
$F3$	53	174.61	252.56
$F\sharp 3$	54	185.00	238.38
$G3$	55	196.00	225.00
$G\sharp 3$	56	207.65	212.37
$A3$	57	220.00	200.45
$A\sharp 3$	58	233.08	189.20
$B3$	59	246.94	178.58
$C4$	60	261.63	168.56
$C\sharp 4$	61	277.18	159.10
$D4$	62	293.66	150.17
$D\sharp 4$	63	311.13	141.74
$E4$	64	329.63	133.79
$F4$	65	349.23	126.28
$F\sharp 4$	66	369.99	119.19
$G4$	67	392.00	112.50
$G\sharp 4$	68	415.30	106.19
$A4$	69	440.00	100.23
$A\sharp 4$	70	466.16	94.60
$B4$	71	493.88	89.29
$C5$	72	523.25	84.28

Continued on next page

Table A.1 – continued from previous page

Pitch	MIDI note number	Frequency (Hz)	Period (samples at 44100 kHz)
$C^\sharp 5$	73	554.37	79.55
$D5$	74	587.33	75.09
$D^\sharp 5$	75	622.25	70.87
$E5$	76	659.26	66.89
$F5$	77	698.46	63.14
$F^\sharp 5$	78	739.99	59.60
$G5$	79	783.99	56.25
$G^\sharp 5$	80	830.61	53.09
$A5$	81	880.00	50.11
$A^\sharp 5$	82	932.33	47.30
$B5$	83	987.77	44.65
$C6$	84	1046.50	42.14
$C^\sharp 6$	85	1108.73	39.78
$D6$	86	1174.66	37.54
$D^\sharp 6$	87	1244.51	35.44
$E6$	88	1318.51	33.45
$F6$	89	1396.91	31.57
$F^\sharp 6$	90	1479.98	29.80
$G6$	91	1567.98	28.13
$G^\sharp 6$	92	1661.22	26.55
$A6$	93	1760.00	25.06
$A^\sharp 6$	94	1864.66	23.65
$B6$	95	1975.53	22.32
$C7$	96	2093.00	21.07
$C^\sharp 7$	97	2217.46	19.89
$D7$	98	2349.32	18.77
$D^\sharp 7$	99	2489.02	17.72
$E7$	100	2637.02	16.72
$F7$	101	2793.83	15.78
$F^\sharp 7$	102	2959.96	14.90
$G7$	103	3135.96	14.06
$G^\sharp 7$	104	3322.44	13.27

Continued on next page

Table A.1 – continued from previous page

Pitch	MIDI note number	Frequency (Hz)	Period (samples at 44100 kHz)
$A7$	105	3520.00	12.53
$A^\sharp 7$	106	3729.31	11.83
$B7$	107	3951.07	11.16
$C8$	108	4186.01	10.54

Appendix B

Equal-Loudness Filter Coefficients

The two filters from Robinson [2001] that are used in Tartini for outer/middle ear anttenuation. Firstly, a second-order high-pass filter was created in Matlab using

```
[B,A]=butter(2,(150/(sampleRate/2)),'high');
```

The resulting coefficients at 44100 Hz are

```
A =  1.0,
    -1.969778555826179999854730340302921831607818603515625O,
     0.970228475663497569314586144173517823219299316406250O

B =  0.985001757872419392292329121119109913706779479980468S,
    -1.970003515744838784584658242238219827413558959960937S,
     0.985001757872419392292329121119109913706779479980468S
```

Secondly, a 10th order IIR filter was constructed using the Yulewalk method. The resulting coefficients at 44100 Hz are

```
A =  1.0,
    -3.47845948550071,
     6.36317777566148,
    -8.54751527471874,
     9.47693607801280,
    -8.81498681370155,
     6.85401540936998,
    -4.39470996079559,
     2.19611684890774,
```

```
    -0.75104302451432,
     0.13149317958808

B =  0.05418656406430,
    -0.02911007808948,
    -0.00848709379851,
    -0.00851165645469,
    -0.00834990904936,
     0.02245293253339,
    -0.02596338512915,
     0.01624864962975,
    -0.00240879051584,
     0.00674613682247,
    -0.00187763777362
```

Appendix C

Detailed Results Tables

The following shows more detailed results from Experiments 9-17. However a summary of the key terms used in the tables is first given.

c - the threshold constant.
s - the scaler coefficient.
m - the median filter window size.
Correct - values which matched the expected pitch (within 1.5 semitones).
-1 - values 1 octave below the expected pitch (within 1.5 semitones).
-2 - values 1.5 octaves below the expected pitch (within 1.5 semitones).
-3 - values 2 octaves below the expected pitch (within 1.5 semitones).
+1 - values 1 octave above the expected pitch (within 1.5 semitones).
+2 - values 1.5 octaves above the expected pitch (within 1.5 semitones).
+3 - values 2 octaves above the expected pitch (within 1.5 semitones).
Other - values where a pitch was found that did not fall into the previous categories.
n/a - no pitch was found.
Error - The percentage of the of all non-correct values.
SNAC - the special normalisation of the autocorrelation algorithm.
SNAC-ALD - the SNAC used in combination with the aggregate lag domain.
SNAC-WALD - the SNAC used in combination with the warped aggregate lag domain.
MC - the modified cepstrum algorithm.
MC-ALD - the MC used in combination with the aggregate lag domain.
MC-WALD - the MC used in combination with the warped aggregate lag domain.

Iowa Dataset										
Algorithm	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
ACF	153477	2939	735	212	1120	138	5	1116	30	3.94%
SNAC(Sloped)	153259	2964	732	219	1230	140	7	1197	24	4.08%
MIDI Dataset										
Algorithm	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
ACF	47762	568	192	16	1	1	1	25	0	1.66%
SNAC(Sloped)	47739	581	202	13	2	1	1	27	0	1.7%

Table C.1: Experiment 9a - no outer/middle ear filtering

Iowa Dataset										
Algorithm	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
ACF	155883	1673	286	18	600	261	3	1030	18	2.43%
SNAC(Sloped)	155695	1705	291	22	628	281	6	1130	14	2.55%
MIDI Dataset										
Algorithm	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
ACF	47748	564	212	11	3	1	1	26	0	1.68%
SNAC(Sloped)	47748	567	210	8	4	1	1	27	0	1.68%

Table C.2: Experiment 9b - with outer/middle ear filtering

Iowa Dataset										
c	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
0.80	156038	108	60	8	2291	613	148	498	8	2.34%
0.85	157432	150	71	17	1495	280	2	317	8	1.46%
0.90	158245	233	108	20	814	183	2	159	8	0.96%
0.95	158294	879	205	35	146	44	0	161	8	0.93%
1.00	118059	20275	10645	4220	12	2	0	6551	8	26.1%
MIDI Dataset										
c	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
0.80	48244	134	81	16	50	0	1	40	0	0.66%
0.85	48065	221	163	43	8	0	1	65	0	1.03%
0.90	47656	335	330	100	0	0	1	144	0	1.87%
0.95	46625	631	703	239	0	0	1	367	0	4.00%
1.00	37260	3852	3499	1377	1	0	1	2576	0	23.3%

Table C.3: Experiment 10 - the SNAC function with peak picking

Iowa Dataset										
c	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
0.4	125681	1090	63	7	941	1231	1125	29633	1	21.3%
0.5	134497	1581	161	15	622	694	690	21511	1	15.8%
0.6	139715	2360	335	43	338	253	490	16237	1	12.6%
0.7	141712	3926	695	94	227	142	375	12600	1	11.3%
0.8	141395	6180	1355	206	73	77	293	10192	1	11.5%
0.9	138343	9576	2367	409	51	43	251	8731	1	13.4%
1.0	133219	14036	3751	777	32	49	182	7725	1	16.6%
MIDI Dataset										
c	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
0.4	38319	161	70	0	132	187	28	9669	0	21.1%
0.5	41618	407	242	1	59	116	15	6108	0	14.3%
0.6	43275	400	723	0	48	60	11	4049	0	10.9%
0.7	44207	658	909	5	23	40	7	2717	0	8.98%
0.8	44605	978	1015	7	12	22	0	1927	0	8.16%
0.9	44252	1668	1093	6	7	14	0	1526	0	8.88%
1.0	43243	2694	1271	56	2	9	0	1291	0	11%

Table C.4: Experiment 11 - the cepstrum

Iowa Dataset										
c	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
0.3	156838	243	43	16	1656	379	43	553	1	1.84%
0.4	158215	292	51	19	794	97	2	301	1	0.975%
0.5	158759	356	60	24	282	46	2	242	1	0.634%
0.6	158786	528	77	30	124	5	2	219	1	0.617%
0.7	158491	819	130	34	79	4	2	212	1	0.802%
0.8	157689	1486	283	52	39	0	2	220	1	1.3%
0.9	155652	2985	777	117	12	0	2	226	1	2.58%
1.0	147715	8113	2879	725	5	0	1	333	1	7.55%
MIDI Dataset										
c	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
0.3	47993	12	6	4	11	0	1	539	0	1.18%
0.4	48343	27	7	0	0	0	1	188	0	0.459%
0.5	48383	53	23	2	0	0	1	104	0	0.377%
0.6	48318	99	51	8	0	0	1	89	0	0.511%
0.7	48150	215	93	19	0	0	1	88	0	0.857%
0.8	47787	454	182	39	0	0	1	103	0	1.6%
0.9	47058	890	414	72	0	0	1	131	0	3.11%
1.0	44172	2382	1629	177	0	0	1	205	0	9.05%

Table C.5: Experiment 12 - the modified cepstrum

Iowa Dataset										
s	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
10	158696	570	68	31	164	6	2	234	1	0.673%
1	158759	356	60	24	282	46	2	242	1	0.634%
0.1	158635	344	59	24	388	76	2	243	1	0.712%
0.01	158622	343	59	24	402	76	2	243	1	0.72%
0.001	158616	343	59	24	408	76	2	243	1	0.724%
Linear	158616	343	59	24	408	76	2	243	1	0.724%
MIDI Dataset										
s	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
10	48300	78	42	4	0	0	1	141	0	0.548%
1	48383	53	23	2	0	0	1	104	0	0.377%
0.1	48395	46	21	2	0	0	1	101	0	0.352%
0.01	48396	46	20	2	0	0	1	101	0	0.35%
0.001	48396	46	20	2	0	0	1	101	0	0.35%
Linear	48396	46	20	2	0	0	1	101	0	0.35%

Table C.6: Experiment 13 - the modified cepstrum with varied scaling

Iowa Dataset										
m	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
1	158759	356	60	24	282	46	2	242	1	0.634%
5	158741	361	52	23	307	62	1	225	0	0.645%
7	158807	327	48	20	306	59	3	202	0	0.604%
9	158834	312	40	18	301	61	4	202	0	0.587%
15	158817	293	52	26	280	61	4	231	8	0.598%
51	158129	392	166	121	260	57	0	523	124	1.03%
MIDI Dataset										
m	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
1	48383	53	23	2	0	0	1	104	0	0.377%
5	48394	41	23	1	0	0	1	106	0	0.354%
7	48436	33	12	0	0	0	1	84	0	0.268%
9	48476	24	4	0	0	0	1	61	0	0.185%
15	48489	24	4	0	0	0	1	48	0	0.159%
51	40441	319	0	0	406	7	2	7391	0	16.7%

Table C.7: Experiment 14 - the modified cepstrum with median smoothing

Iowa Dataset										
Algorithm	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
SNAC	158191	228	90	22	888	182	4	159	8	0.99%
SNAC-ALD	158635	31	21	3	550	161	1	362	8	0.712%
SNAC-WALD	158581	42	18	3	550	161	1	408	8	0.745%
MC	158759	356	60	24	282	46	2	242	1	0.634%
MC-ALD	159250	41	11	5	67	0	3	394	1	0.327%
MC-WALD	159583	41	11	5	67	0	1	63	1	0.118%
MIDI Dataset										
Algorithm	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
SNAC	47656	335	330	100	0	0	1	144	0	1.87%
SNAC-ALD	47076	744	597	54	0	0	0	95	0	3.07%
SNAC-WALD	46916	850	661	68	0	0	0	71	0	3.4%
MC	48383	53	23	2	0	0	1	104	0	0.377%
MC-ALD	48548	1	0	0	0	0	1	16	0	0.0371%
MC-WALD	48548	1	0	0	0	0	1	16	0	0.0371%

Table C.8: Experiment 15 & 16 - a comparison of un-warped, warped and no aggregate lag domain

MIDI Dataset - No Reverb										
Algorithm	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
SNAC	47656	335	330	100	0	0	1	144	0	1.87%
SNAC-ALD	47076	744	597	54	0	0	0	95	0	3.07%
SNAC-WALD	46916	850	661	68	0	0	0	71	0	3.4%
MC	48383	53	23	2	0	0	1	104	0	0.377%
MC-ALD	48548	1	0	0	0	0	1	16	0	0.0371%
MC-WALD	48548	1	0	0	0	0	1	16	0	0.0371%
MIDI Dataset - With Reverb										
Algorithm	Correct	-1	-2	-3	+1	+2	+3	Other	n/a	Error
SNAC	47252	528	509	95	4	4	1	173	0	2.71%
SNAC-ALD	46374	1046	1050	42	0	0	0	54	0	4.51%
SNAC-WALD	46322	1073	1034	68	0	0	0	69	0	4.62%
MC	48240	74	33	0	0	2	1	216	0	0.671%
MC-ALD	48498	15	0	0	0	0	1	52	0	0.14%
MC-WALD	48513	15	0	0	0	0	1	37	0	0.109%

Table C.9: Experiment 17 - with and without reverberation

Appendix D

Glossary

ACF Autocorrelation function or just autocorrelation.

ALD *Aggregate lag domain*. Discussed in Section 7.2.1.

attack The beginning part of a note where the volume increases sharply.

autocorrelation A method in which a signal is convolved with itself. Described in section 2.3.2.

cepstrum The Fourier transform of the log power spectrum. See Section 2.5.1.

cent 1/100th of a semitone in the even tempered scale. So 1200 cents equals one octave.

channel A single stream of a sound recording. For example a mono sound recording contains one channel, stereo two.

CMA *Complex moving average* filter. Discussed in Section 8.3.

critical band The bandwidth of an auditory filter from which individual frequencies cannot be resolved.

DFT Discreate Fourier transform. Discussed in Section 2.4.

double stop Two strings played together, on a violin for example.

FFT Fast Fourier transform. An efficient method for calculating a DFT.

FIR filter *Finite impulse response* filter.

frame An index which describes a window's given position, although often refers to the data in the window at that position. Note that the window's position is $frame * hopsize$.

frequency bin A small range of continuous frequencies which fall largely into the same Fourier coefficient during a DFT.

fundamental or fundamental frequency, is the lowest frequency in a harmonic series. It is the inverse of the period.

harmonic A sinusoidal component of a sound that has a frequency which is an integer multiple of the fundamental frequency.

hop size The number of samples a window is slid to the right between frames, although often expressed in terms of window size.

IIR filter *Infinite impulse response* filter.

impuse A function that is zero everywhere except for a single non-zero value or spike.

impuse train A function that zero everywhere except for systematic spikes at a regular interval.

interval A musical *interval* describes the difference in pitch between two notes.

key A musical *key* is often used to describe the root, or tonic, note name along with its scale type. For example, in the key of G-major.

MIDI *Musical instrument digital interface* - a protocol for transmitting musical 'event messages'.

MIDI number A number in the range 0-127, of which is unique for each semitone on the even tempered scale. Appendix A lists the most common numbers. In this thesis real values are used to indicate pitches between the common notes, *e.g.* 69.12 indicates 12 cents sharper than a $A4$.

NSDF *Normalised square difference function.* This is equivalent to the SNAC function described in section 4.1.

octave error An error in pitch detection which is caused from choosing the incorrect peak in the frequency, lag or cepstrum domain. It does not have to be an octave from the expected result, it just happens to be common.

octave estimate An estimate of the pitch for a given frame. This estimate is expected to be robust in its choice of octave, but is not expected to be accurate in its exact pitch value.

offset A small region of time in which a note is considered to be ending.

onset A small region of time in which a note is considered to be starting, usually associated with a rise in amplitude from near zero to an initial peak.

partial Any sinusoidal component of a complex sound, even if it is not a harmonic.

peak A region encompassing a local maximum in a function.

period The smallest repeating unit of a periodic signal, or the length in time of this unit.

periodic A period which is an integer multiple of the fundamental period.

pitch fundamental The frequency of a sine wave that is perceived to be the same height as a complex harmonic tone.

pitch period An approximately repeating unit of a signal that is associated with the perceived pitch of a note. Also used to describe its length in time. This is the inverse of the pitch fundamental.

pizzicato The technique of plucking a stringed instrument instead of bowing.

quefrency The unit of the cepstrum axis. It is analogous to time.

sample One unit of data from a single channel within a sound.

sample rate The number of samples that occur in a second for a given channel within a sound. These samples usually occur at a constant rate *e.g.* 44100 Hz.

SDF *Square difference function.* Described in section 2.3.3.

SNAC function *Specially normalised autocorrelation* function, described in Section 4.1.

STFT *Short-time Fourier transform*, that is the Fourier transform of a signal which has been windowed to finite length.

sub-periodic A high peak that appears before the first periodic, which may be mistaken as the periodic.

super-periodic A periodic greater than the first periodic, which may be mistaken as the periodic.

Tartini The name of the software created during this work, as well as the name of the larger project that has grown out of it.

Tartini-tone An audible tone obtained from the beating of two other tones. Discussed in Section 2.2.

timbre The quality of a sound that distinguishes it from other sounds with the same pitch and volume. It is sometimes referred to as tone colour.

tonic The name of the root note in a key, *i.e.* the tonic of C-major is C.

transient A short segment of sound characterised by non-stationary spectral content, usually during note transitions or onset. The pitch can often be hard to define during a transient.

tremolo A cyclic change in amplitude, usually in the range of 5 to 14 Hz.

vibrato A cyclic change in pitch, usually in the range of 5 to 14 Hz.

WALD *Warped aggregate lag domain.* Discussed in Section 7.2.2.

window A small segment of a larger data set that is currently being viewed or processed on.

windowed A window which has been multiplied by a windowing function.

windowing The process of multiplying a window elementwise by a windowing function.

windowing function A function used to smoothly localise data about a point in time. Usually with a maximum in the centre, tending towards zero on either side. For example the Hamming window discussed in Section 2.3.2.

window size The number of data samples within a window.

WSNAC function *Windowed SNAC* function, described in Section 4.2.