

Investigation of Techniques for Automatic Polyphonic Music Transcription Using Wavelets



by

John C. McGuiness

Submitted in fulfilment of the requirements for the degree of

Master of Science

in the School of Computer Science

University of KwaZulu-Natal

Pietermaritzburg

November 2009

Declaration of Originality

This thesis has not previously been submitted for a degree in this or any other university, and, to the best of my knowledge, contains no material previously published or written by another person or persons, except where due reference has been made.

Candidate: John C. McGuiness (206519178)

Declaration of Suitability

As the candidate's supervisor, I have approved this thesis for submission for examination.

Supervisor: Hugh C. Murrell

Acknowledgements

My sincerest thanks goes to:

- Professor Hugh Murrell, my supervisor, for his continued guidance and support.
- The Natal Society Foundation, for their generous bursary.
- Marcus Henning and the Hexagon Theatre, for the use of their recording studio and equipment.
- Shona Wallis, Ivan Frommurze and Marc Mervin: thank you for the music.
- Curtis, for proof-reading this thesis and for being a great friend.
- Freda, for friendship, kind generosity and support.
- Father Hugh Ross, the original inspirator for this research while I was still in high school, without whom I might not have “bashed on regardless” in the first place.
- Dr. JPG “Paddy” Ewer, my internal examiner, for contributing a meticulously comprehensive and invaluable set of corrections.
- Professor Geoff Wyvill, my external examiner, for his likewise very insightful and useful comments.

Abstract

It has been said (although sadly I have no source) that music is one of the most useful yet useless phenomena known to mankind. Useless in that it has, apparently, no tangible or immediately practical function in our lives, but extremely useful in that it is a truly universal language between human beings, which transcends boundaries and allows us to express ourselves and experience emotions in rather profound ways.

For the majority of us, music exists to be listened to, appreciated, admired (sometimes reviled) but generally as some sort of stimulus for our auditory senses. Some of us feel the need to produce music, perhaps simply for our own creative enjoyment, or maybe because we crave the power it lends us to be able to inspire feelings in others. For those of us who love to know “the reason why” or “how things work” and wish to discover the secrets of music, arguably the greatest of all the arts, there can surely be no doubt that a fascinating world of mathematics, harmony and beauty awaits us. Perhaps the reason why music is able to convey such strong emotions in us is because we are (for whatever strange evolutionary reason or purpose) designed to be innately pattern pursuing, sequence searching and harmony hungry creatures. Music, as we shall discover in this research, is chock-a-block full of the most incredible patterns, which are just waiting to be deciphered.

Table of Contents

1 Introduction	1
1.1 Music Recognition in General.....	1
1.2 Research Problem Description	1
1.3 Aims, Objectives and Scope of Research.....	2
1.4 Importance and Usefulness of Research	3
1.5 Thesis Structure.....	4
2 Audio Signal Analysis – Basic Concepts	5
2.1 Waves.....	5
2.2 Digital Representation of a Signal	10
2.3 Transforms	13
2.4 The Fast Fourier Transform Algorithm.....	17
3 An ABC of Music Theory	26
3.1 Musical Pitch.....	26
3.2 Musical Modes	30
3.3 Music Notation.....	35
3.4 Diatonic Intervals and Chords.....	41
3.5 Chord Progressions	49
4 The Easy Problem – Single Pitch Extraction.....	54
4.1 Windowing in the Time Domain.....	54
4.2 The Phase Vocoder.....	58
4.3 The McLeod Pitch Method (MPM)	61
4.4 Drawing a Pitch/Time Graph	66
4.5 Comparison of Output.....	69
5 The Hard Problem – Multiple Pitch Extraction.....	78
5.1 Previous Attempts – Exploration of Existing Software	78
5.2 Introducing Wavelets	85
5.3 The Discrete Wavelet Transform	91
5.4 The Fast Haar and Daubechies Transforms	93
5.5 The Fast Redundant Haar Transform	101
5.6 Windowing in the Frequency Domain	103
5.7 Output.....	104
6 Drawing a Spectrogram Using the Continuous Wavelet Transform.....	111
6.1 The Continuous Wavelet Transform	111
6.2 Four Wavelets	113
6.3 Implementing the Fast CWT	121
6.4 Comparison of Output.....	122
7 Interpreting the CWT Spectrogram.....	127
7.1 The Scale of the CWT Spectrogram	127
7.2 Beats.....	129
7.3 Difference Tone Analysis.....	130
7.4 An Improved Pitch Detection Method	137

8 Experimental Pitch Detection on Live Audio Recordings	144
8.1 Preparation of Audio Data	144
8.2 Methods and Results	145
9 From Pitch Graphs to Musical Scores	163
9.1 Temporal Note Detection.....	163
9.2 Note Classification via Image Processing Techniques	165
10 Conclusions	169
10.1 Evaluation of Results	169
10.2 Ideas for Further Research.....	171
Appendix A – Mathematical Notation	173
Appendix B – Algebraic Workings.....	176
Appendix C – Code Listings.....	182
Appendix D – Musical Scores.....	186
Appendix E – Wave Processor 3.0.....	189
References and Sources	192

List of Figures

Figure 2.1 – An example of a wave	5
Figure 2.2 – Three sine wave graphs	6
Figure 2.3 – Sine and cosine graphs, both with frequency 5Hz.....	7
Figure 2.4 – Polar graph showing two angles θ_1 and θ_2 with phase difference ϕ	8
Figure 2.5 – Diagram showing phase difference of a wave at each ear.....	8
Figure 2.6 – Composite sinusoidal wave	9
Figure 2.7 – Sampled versions of Figure 2.6	11-12
Figure 2.8 – Frequency domain representation of signal in Figure 2.6	14
Figure 2.9 – Bit Reversal	18
Figure 2.10 – Decimation 1.....	18
Figure 2.11 – Decimation 2.....	19
Figure 2.12 – Polar complex number diagrams of combinations	23-24
Figure 2.13 – Polar graph representation of a complex Fourier coefficient, v	25
Figure 3.1 – A freely vibrating string.....	27
Figure 3.2 – A vibrating string stopped half-way	27
Figure 3.3 – Dividing the string into thirds.....	27
Figure 3.4 – Four divisions of the string: third harmonic	28
Figure 3.5 – Octaves resonance of two cosine waves.....	29
Figure 3.6 – An example of a musical staff	36
Figure 3.7 – Notes drawn on leger lines	36
Figure 3.8 – Clef symbols	36
Figure 3.9 – Relative vertical positions of staff systems	37
Figure 3.10 – The Tenor clef.....	37
Figure 3.11 – Ties and dots example	38
Figure 3.12 – Previous example staff with bar lines added	39
Figure 3.13 – Previous example staff with bar lines and time signature	41
Figure 3.14 – Tonic triads of C Major and A Minor.....	44
Figure 3.15 – Diatonic triads of C Major.....	44
Figure 3.16 – Diatonic triads of A Harmonic Minor	44
Figure 3.17 – Inversions of C Major tonic triad.....	45
Figure 3.18 – Triads comprising fourths and seconds	46
Figure 3.19 – Suspensions and resolutions	46
Figure 3.20 – Examples of cluster chords.....	47
Figure 3.21 – Quartads.....	47
Figure 3.22 – A wide quartad.....	47
Figure 3.23 – C Major dominant seventh in different inversions	49
Figure 3.24 – Perfect cadences in C Major and A Minor	49
Figure 3.25 – Plagal cadences in C Major and A Minor.....	50
Figure 3.26 – Imperfect cadences in C Major and A Minor	50
Figure 3.27 – Interrupted cadences in C Major and A Minor.....	51
Figure 3.28 – Three types of relative voice motion	51
Figure 3.29 – Doubling of notes in four-part harmony	52
Figure 4.1 – DFT histogram of $f(t) = \frac{1}{3} \sin(10\pi t) + \frac{1}{3} \sin(20\pi t) + \frac{1}{3} \sin(40\pi t)$	55
Figure 4.2 – $f(t)$ split into four windows	56
Figure 4.3 – DFT histogram of $f(t)$ for each window.....	56
Figure 4.4 – Gaussian function and its product with the first window of $f(t)$	57

Figure 4.5	– Fourier transform of windowed signal	58
Figure 4.6	– $f(t)$ split into seven overlapping windows	59
Figure 4.7	– Windowed signal from 0.25 to 0.75 and its Fourier transform	60
Figure 4.8	– An example NSDF output.....	65
Figure 4.9	– First two bars of <i>Nkosi Sikeleli Africa</i> melody.....	68
Figure 4.10	– MPM pitch graph of first two bars of <i>Nkosi Sikeleli Africa</i> melody.....	68
Figure 4.11	– Fourier transform of stationary signal comprising three frequencies	70
Figure 4.12	– Linear chirp signal (time domain).....	70
Figure 4.13	– Fourier transform of linear chirp signal	71
Figure 4.14	– Short Time Fourier Transform of linear chirp signal.....	71
Figure 4.15	– STFT of linear chirp signal with Gaussian window function	72
Figure 4.16	– STFT of linear chirp signal with larger window width.....	72
Figure 4.17	– Pitch graph of chirp calculated by Phase Vocoder.....	73
Figure 4.18	– Pitch graph of chirp calculated by McLeod Pitch Method.....	73
Figure 4.19	– Scale of D Minor, one octave, ascending and descending	74
Figure 4.20	– STFT of D Minor scale	74
Figure 4.21	– Phase Vocoder pitch graph of D Minor scale	75
Figure 4.22	– MPM pitch graph of D Minor scale	75
Figure 4.23	– MPM pitch graph of <i>Tartini</i> example scale	76
Figure 4.24	– Phase Vocoder pitch graph of first 2 bars of <i>Nkosi Sikeleli Africa</i> melody	77
Figure 5.1	– Musical score of marimbaphone melody from DNA example	78
Figure 5.2	– Spectrogram of marimbaphone melody	79
Figure 5.3	– Probability distribution spectrogram for frequencies belonging to 12 th note.....	81
Figure 5.4	– <i>AudioScore</i> pitch graph of marimbaphone melody.....	83
Figure 5.5	– Marimbaphone melody imported into <i>Sibelius</i> from <i>AudioScore</i>	83
Figure 5.6	– Manual transcription of <i>Nkosi Sikeleli Africa</i> quartet example in <i>Sibelius</i>	84
Figure 5.7	– <i>AudioScore</i> transcription of NSA quartet imported into <i>Sibelius</i>	84
Figure 5.8	– <i>AudioScore</i> pitch graph of NSA quartet example	84
Figure 5.9	– Haar component vectors drawn on the Cartesian plane	87
Figure 5.10	– The Haar mother wavelet as a square function	88
Figure 5.11	– Haar wavelets at first scale ($s = 1$)	89
Figure 5.12	– Haar wavelets at second scale ($s = 2$).....	90
Figure 5.13	– Heisenberg boxes for the STFT	92
Figure 5.14	– Heisenberg boxes for the DWT.....	92
Figure 5.15	– Lifting Scheme diagram	96
Figure 5.16	– Daubechies 4 scaling and mother wavelet functions	98
Figure 5.17	– Haar wavelet transform of chirp signal.....	105
Figure 5.18	– Daubechies 4 wavelet transform of chirp signal	105
Figure 5.19	– Dyadic graph of D4 wavelet transform of chirp signal.....	106
Figure 5.20	– Haar wavelet transform of <i>lohi . wav</i>	107
Figure 5.21	– D4 wavelet transform of <i>lohi . wav</i>	107
Figure 5.22	– Redundant Haar wavelet transform of <i>lohi . wav</i>	108
Figure 5.23	– Two-part arrangement of first two bars of <i>Nkosi Sikeleli Africa</i>	108
Figure 5.24	– Three-part arrangement of first two bars of <i>Nkosi Sikeleli Africa</i>	109
Figure 5.25	– Automatic transcription of two-part arrangement using DWT/MPM.....	109
Figure 5.26	– Automatic transcription of three-part arrangement using DWT/MPM.....	110
Figure 6.1	– Comparison of DWT and CWT spectrograms of a chirp signal	111
Figure 6.2	– The Morlet wavelet – Graph showing real and imaginary components	114
Figure 6.3	– Fourier transform of Morlet wavelet.....	115
Figure 6.4	– Second order real Derivative of Gaussian (Mexican Hat) wavelet.....	117

List of Tables

Table 3.1 – Harmonic Series of 262Hz.....	28
Table 3.2 – Harmonic Series of 196.5Hz.....	29
Table 3.3 – Harmonic Series of 87.333Hz.....	30
Table 3.4 – Closely related pitches in ascending order	30
Table 3.5 – Pitch Intervals	31
Table 3.6 – Modes	33
Table 3.7 – Dodecachordon layout	34
Table 3.8 – Note types	38
Table 3.9 – Keys and key signatures	40
Table 3.10 – Names of Intervals	42
Table 3.11 – Degrees of the scale	45
Table 3.12 – Seventh Chords	48
Table 4.1 – Comparison of Fourier coefficients from first and second time windows	60
Table 5.1 – Frequencies and starting times of notes in marimbaphone melody.....	79
Table 5.2 – Frequency filtering for first note in first window	80
Table 7.1 – Difference tones for quarter-tone intervals above C_4	134
Table 7.2 – Frequencies and nearest pitches corresponding to peaks in Figure 7.14	138
Table 7.3 – Frequencies and nearest pitches corresponding to peaks in Figure 7.14	139
Table 8.1 – Results of automatic transcriptions for two-part string recordings	148
Table 8.2 – Results of automatic transcriptions for two-part piano recording	148
Table 8.3 – Results of automatic transcriptions for two-part synthesized waves.....	148
Table 8.4 – Results for DWT/MPM	149
Table 8.5 – Results for Phase vocoder.....	149
Table 8.6 – Results for CPM with Morlet transform.....	150
Table 8.7 – Results for CPM with Paul transform.....	151
Table 8.8 – Results for CPM with DoG transform	151
Table 8.9 – Results of automatic transcriptions for three-part string recordings	152
Table 8.10 – Results of automatic transcriptions for three-part piano recordings.....	152
Table 8.11 – Results of automatic transcriptions for three-part synthesized waves.....	152
Table 8.12 – Results for Phase Vocoder	152
Table 8.13 – Results for DWT/MPM	154
Table 8.14 – Results for CPM with Morlet transform.....	154
Table 8.15 – Results for CPM with Paul transform.....	156
Table 8.16 – Results for CPM with DoG transform	156
Table 8.17 – Results of automatic transcriptions for four-part string recordings.....	158
Table 8.18 – Results of automatic transcriptions for four-part piano recordings	158
Table 8.19 – Results of automatic transcriptions for four-part synthesized waves	158
Table 8.20 – Results for Phase Vocoder	158
Table 8.21 – Results for CPM with Morlet transform.....	160
Table 8.22 – Results for CPM with Paul transform.....	160
Table 8.23 – Results for CPM with DoG transform	161-162
Table 9.1 – Quantization of component widths	167

Table A.1 – Mathematical notation conventions.....	173-174
Table A.2 – Window functions for the STFT (images from www.wikipedia.org).....	175
Table A.3 – Continuous wavelet functions and their Fourier transforms	175
Table E.1 – System requirements for running <i>Wave Processor</i>	189
Table E.2 – Transform options in <i>Wave Processor</i>	190
Table E.3 – Pitch detection options in <i>Wave Processor</i>	191

1 Introduction

“Admittedly, however, it is difficult or impossible to picture what goes on in the air when a chord is struck. The mind is staggered at the thought of the thousands of superposed vibrations (or ‘waves’) in the air space in a concert-room when an orchestra is playing.”

– Percy A. Scholes, *The Oxford Companion to Music*, 1965, on *Acoustics*

1.1 Music Recognition in General

The problem of dissecting musical sounds and attempting to identify their make-up has interested musicians and scientists alike for centuries. Perhaps one of the most well-known attempts to transcribe music by ear is the story of the 14-year-old Wolfgang Amadeus Mozart, on tour of Italy with his father in 1770 [Scholes65L]. Mozart was so entranced by the beauty of Gregorio Allegri’s nine-voice setting of Psalm 51, *Misere Mei, Deus* [Tallis01M], when he heard it in the Sistine Chapel, that he felt compelled to record it, and wrote out the score after hearing the piece just twice – once from memory and a second time just to make some minor corrections. He did this despite the fact that it was forbidden, “on pain of excommunication”, to make a copy of any part of the piece in any form. Copyright was just as important in the 18th century as it is today, but, of course, for different reasons – all sacred music, in the end, belonged to the Pope. Since Allegri presumably never wrote it down himself, or else his original was never made available, we have only Mozart (and others who subsequently may have transcribed it wrongly) to blame if the piece does not now sound quite the way it used to! Nevertheless, we would probably not be able to hear it in any form today if he had not been able to record it in this way.

What the incredible mind of Mozart was able to do about 240 years ago still remains quite elusive for signal processing researchers, who have tried to accomplish exactly the same thing via automatic means.

1.2 Research Problem Description

In layman’s terms, the main idea behind this research is to work towards creating a computer system which is, to some extent, able to “hear” music and subsequently generate a musical score, i.e. a visual representation of the music as typically read and interpreted by a musical performer. This is akin to the problem of optical character recognition, except in this case, the inputs are audio signals and the output is printed musical notes. Ideally, our state-of-the-art music recognition system should have the ability to perform the following tasks:

- i) Detect and separate all pitches in any given polyphonic musical recording.
- ii) Determine where these pitches occur in time.
- iii) Find any beat, pulse or rhythmic patterns in the music and determine the meter (see the following chapter for an explanation of this musical term).
- iv) Detect changes in tempo (speed).
- v) Give a measure of clarity of pitches, i.e. distinguish between what is a clear musical note and what is “noise”.
- vi) Recognize the instruments and/or voices which make up the recording.
- vii) Represent the music correctly in a musical score.

The above tasks are no mean feat, even for the trained musician. Music dictation is widely regarded as a skill which requires years of training and practice to master. The reality (and this is perhaps testimony to how remarkable the human brain is) is that these are extremely difficult problems to solve synthetically, given current technology and understanding. Research in this field has only quite recently gained some momentum, and there is still much work to be done.

Although it may seem like an “artificial intelligence” type project, much of the difficult work in attempting to solve this problem comes at the pre-processing stage, and so emphasis has been placed on research into pertinent methods in this area. Most likely, the solution to all of the above problems will combine purely empirical methods with artificial intelligence, but it remains to be seen when and where the latter should be utilized in the process.

1.3 Aims, Objectives and Scope of Research

This study is, first and foremost, an exploration of techniques for solving the first two of the seven problems identified in the previous section, i.e. developing a good pitch/time detection method which is able to deal adequately with polyphony. At the heart of this method is a relatively new mathematical tool, namely *wavelets*, which is the main focus of the research. While temporal detection (the attempt to correctly position the detected pitches in time) has been looked at very briefly in Chapter 9, problems **iii**) and **iv**) are outside of the scope of this research. Note that without tempo or meter information (in musical terms) we are unable to draw bar lines or provide a time signature – both vital components of any musical score (unless one is a thirteenth century monk using neumatic notation). Therefore, all final graphical results of pitch/time analysis are presented as unbarred “piano roll”-type graphs, examples of which are first presented in Chapter 4. These may be compared with the modern musical score notation used in the examples in Chapter 2.

The quotation marks around the word “noise” in problem **v**) are to differentiate between what is considered musical “noise” – that which is intended to be there – and background or accidental noise. An example of the latter would be hisses, clicks or pops due to a poor quality recording, or else coughs and the rustle of sweet papers in a live audience. Musical “noise”, on the other hand, would be content produced by non-pitched instruments, such as drums and other percussion. In order to reduce the complexity of the problem, this study assumes that all music and noise in the recording is intended, i.e. the “Garbage In, Garbage Out” principle applies. This also means that any artistic interpretation of the music (good or otherwise) when performed by a particular musician or group of musicians will, necessarily, be transcribed literally. Therefore, given that different musical performances of the same written work invariably differ greatly from one another, it should not be expected that automatic transcriptions of different recordings of the same piece should all render identical scores, or that the scores should be one hundred percent similar to the original from which the music was performed.

In [Cont07L] an important point is made: the technique of digitally mixing synthetic musical instruments or post-mixing two or more real instruments which have been recorded separately is likely to yield different “spectral fusions” to those common in ensemble recordings, where all instruments have been recorded at the same time, and their sound has blended in the air before reaching the recording equipment. That is to say, the harmonic mess described by Scholes at the beginning of this chapter is likely to be different depending on how a particular recording has been created in the studio. It must necessarily be assumed that these particular effects will not alter the fundamental notes of the instruments too drastically, although when

attempting to identify the instruments themselves, this would be a major concern. While multiple instrument detection (problem **vi**) is not within the scope of this research, this particular issue and other similar difficulties should be kept in mind.

Finally, in connection with the preceding problem (since rooms, being resonating chambers, can also be thought of as musical instruments) it should be assumed that the acoustic conditions of the recording are reasonably dry – too much reverberation or echo interferes with any system’s ability to determine note lengths or correct harmonic structures of chords.

1.4 Importance and Usefulness of Research

The reaction in the music community, whenever a breakthrough is made in the field of music recognition, is the first clear indicator that there is an important problem to be solved, especially in music production, for amateurs and professionals alike. An example is the discovery, by Canadian mathematics professor Jason Brown, of the precise arrangement and notation for the infamous opening chord of the Beatles song, *A Hard Day’s Night*, using “the mathematics and physics of sound” [Brown04L]. This story was reported on many internet fora, such as Slashdot, and caused considerable excitement, since it pretty much settled a forty-year long debate amongst Beatles enthusiasts and music transcribers regarding the chord’s construction.

Similar excitement surrounds newly released software capable of pitch recognition, namely Neuratron’s *AudioScore Ultimate 6* (for *Sibelius*) [Neuratron09W] and Celemony’s *Melodyne*, which boasts “groundbreaking technology” in the description of its main pitch manipulation feature, *Direct Note Access* [Celemony09W]. Quoting from the celemony.com web site:

“Apr. 4. At this year’s m.i.p.a. (Musikmesse International Press Awards), *Direct Note Access* was voted most innovative product of the year by more than 100 music magazines from all over the world.”

AudioScore has also enjoyed high praise from its reviewers, and its pitch recognition technology seems fairly robust (see performance tests done on this software in Chapter 5), although it would seem that it has a long way to go before it could be said to be better than a basic solution to the first two problems listed in section **1.2**.

The reason for the hype surrounding the above software releases is due to the fact that the problem of automatic transcription is still largely unsolved, and yet if a solution did exist, the jobs of music producers, transcribers, composers and archivers would be made considerably easier. Firstly and most importantly, such software makes the job of writing down music, for whatever purpose, far less time-consuming. Anybody in the music engraving industry would agree that the task of notating music, and doing so accurately and correctly, is both an art and a science, and it takes a very long time to master. Professional looking musical scores are not easy to produce, so any software capable of automating part or all of the procedure would be, for musicians, akin to suddenly being able to fly having previously had to crawl everywhere. In fact music recognition has been described by one excited forum user as being “the holy grail of music technology”. While this statement is perhaps a little excessive, there is still no denying the usefulness of such a piece of software to the music production industry.

In music archiving, an automatic transcription system would enable musicians who do not know how to write the music they perform, or have a limited understanding of music notation,

to publish music they would otherwise have no way of sharing with others. There have been many such talented musicians throughout history whose music has been forgotten because it has never been written down.

Automatic music transcription systems would also be very useful in developing music education software. Although still mostly vapourware at the time of writing this thesis, new applications are currently being developed which will allow users to listen to individual tracks within popular songs, read the musical scores for each part, follow animated instruments and observe guitar and keyboard fingerings used by professional musicians [MusicIcon09W]. The job of creating scores, animations and fingering metadata for the hundreds of songs a user may demand would be overwhelming without the aid of an automated system, and it is likely that such software will soon require a major increase in effort devoted to music recognition research.

1.5 Thesis Structure

The rest of this thesis is divided into three main parts over the ten chapters, though the boundaries between these parts cannot, due to the inter-relatedness of the material, be rigidly defined.

The bulk of Chapters 2 to 5 is literature review, presenting various fundamental concepts and theory which is vital to understanding and implementing existing techniques, as well as evolving new solutions. A whole chapter has been dedicated to music theory and the development of Western tonal systems. It is extremely important to the task at hand to be aware of how music came into being, why it exists and what its building blocks are. This makes the decoding job a little easier. For music transcribers, there really is no substitute for experience and practice. The best we can do when designing machines (which cannot learn and do not have human experiences) to do human tasks is to know as much as possible about the system which we are modeling and to have some knowledge of elements which occur frequently and of those which do not.

Chapter 5 is also the start of the development section of the thesis. A new (albeit somewhat naïve) method of multiple pitch extraction for two to three part polyphony is presented, following introductory wavelet theory. Chapters 6 and 7, however, contain the core of the main work in this research. Here, the author's own development of an important algorithm, based on the mathematical theory preceding it, is discussed. Chapter 8 is a presentation of experiments and results on studio recorded audio data, and the last chapter before the conclusion is a very brief look at ideas for solving those tasks in the main research problem description which begin to fall outside the scope of this study. This chapter has been included in order to show that it is possible to base a more complete solution on the proposed techniques. The final chapter evaluates the results from experiments and draws some conclusions, before ending with some ideas for further research in this field.

In all chapters where mathematical methods are discussed, algorithms have also been described, which have been implemented in software. The main sound processing application which contains most of these algorithms is called *Wave Processor*, which may be installed from the project CD accompanying this thesis (see **Appendix E**). Experimental audio has also been included on the CD, and so all experiments carried out using *Wave Processor* may be reproduced.

2 Audio Signal Analysis – Basic Concepts

Before launching into the problem of pitch identification, it is necessary for some vital fundamental concepts to be explained, in order to achieve a greater understanding of what is being attempted in the first place. The main purpose of this chapter is to introduce the subject of signal analysis and, specifically, to describe the important mathematical tools needed if one is to attempt to solve any problem in this field, and especially the main problems in this particular research.

2.1 Waves

In order first to understand what exactly it is we are analyzing, this section covers the basic building blocks of audio signals: waves.

2.1.1 What is a Wave?

At one very basic conceptual level, a wave is the pattern created by a particle changing its position with respect to time. This pattern may be drawn by plotting the position of the particle on the y -axis of a graph against time on the x -axis, as illustrated in **Figure 2.1** below:

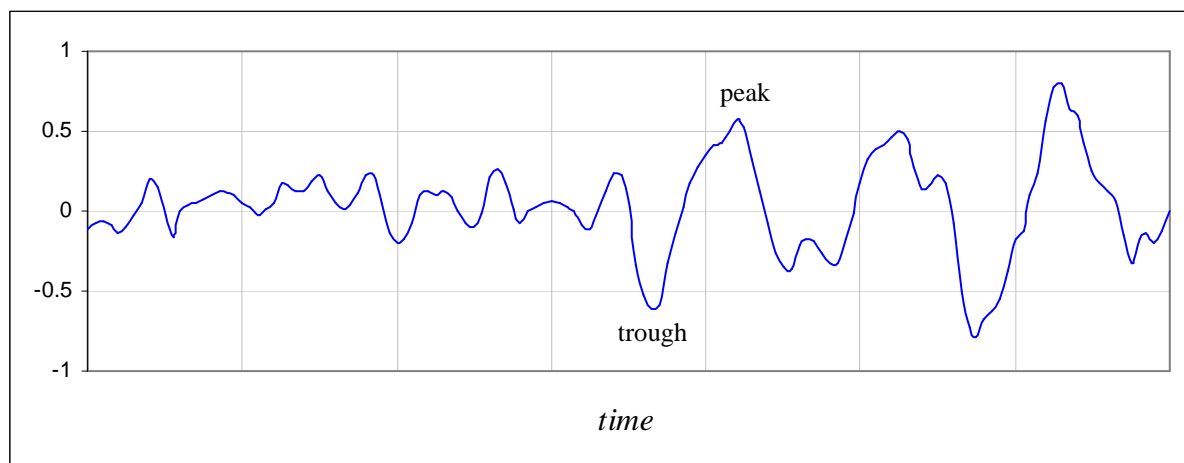


Figure 2.1 – An example of a wave

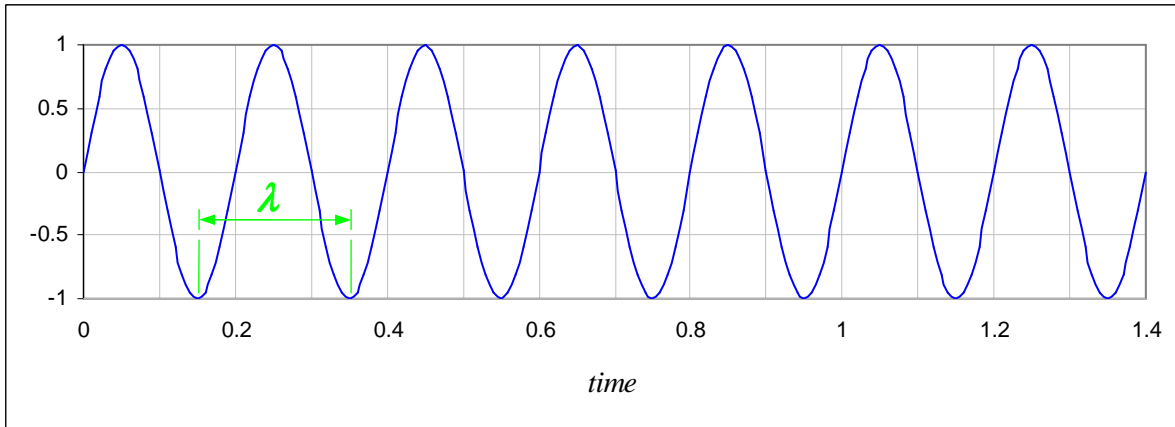
Note that the vertical axis of a graph such as this could actually represent any mathematical or physical variable (not just position) which changes over time. For example it may represent the voltage of an AC power supply or even, thinking on a larger time scale, the number of newspapers sold per day by a press. The value or magnitude of this variable quantity at a certain point in time is known as the *amplitude* of the signal. The word *signal* can be used interchangeably with the word *wave*.

2.1.2 Frequency and Wavelength

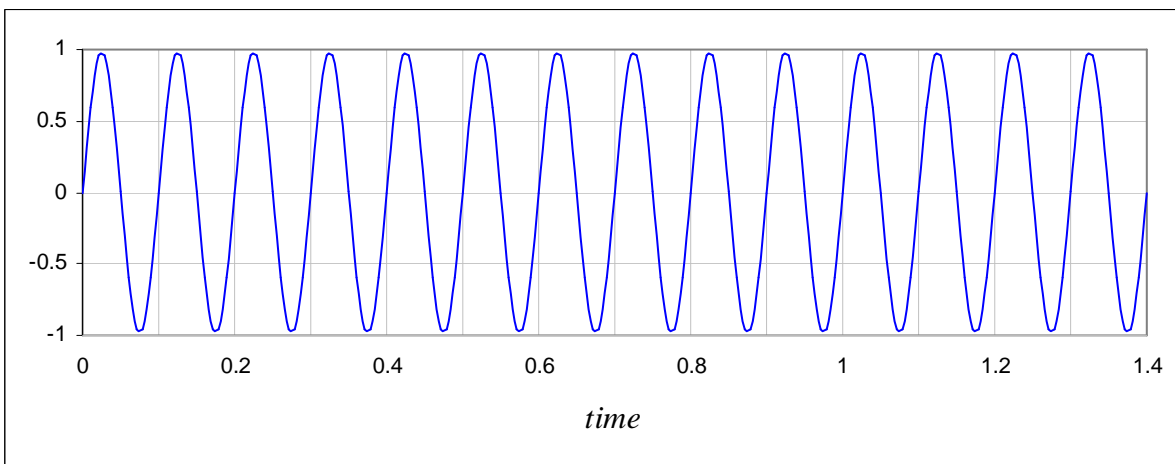
Perhaps the most important term to define when talking about waves is *frequency*. Frequency is a property of signals that repeat some particular pattern. The frequency is said to be *high* if the repetition is rapid, whereas if it is more gradual, it is *low*. As indicated in **Figure 2.1**, the wave pattern forms a series of *peaks* and *troughs*. A *peak* is the highest value reached as a

wave's amplitude increases before decreasing again (or a change from a positive gradient on the graph to a negative one). A *trough* is the lowest amplitude reached, as values decrease, before they increase again (or a change from a negative to a positive gradient).

$$f(t) = \sin(10\pi t) \quad \lambda = 0.2, \nu = 5\text{Hz}$$



$$f(t) = \sin(20\pi t) \quad \lambda = 0.1, \nu = 10\text{Hz}$$



$$f(t) = \sin(40\pi t) \quad \lambda = 0.05, \nu = 20\text{Hz}$$

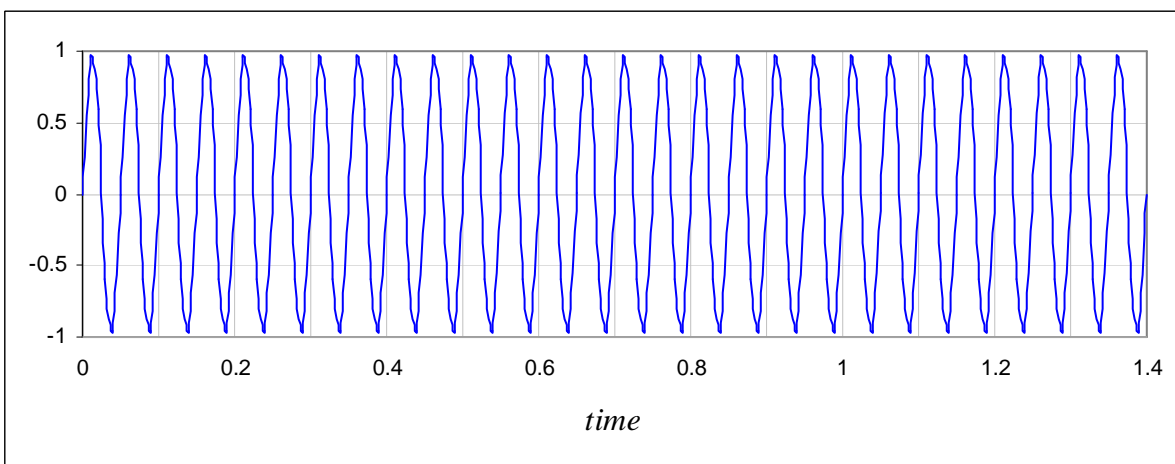


Figure 2.2 – Three sine wave graphs

Referring to **Figure 2.2**, which shows three simple sine wave graphs, the complete traversal of a wave from one similar peak to the next (or between two successive troughs) is known as a *cycle*. The horizontal distance travelled in one cycle is known as the *wavelength* of the signal (indicated by the green λ on the first graph). Wavelength is, however, a spatial measure. When discussing distance in the time dimension, one should rather refer to the *period* of the frequency. Frequency, ν , is measured in terms of the number of cycles per second, and has the unit *Hertz* or *Hz* for short. Incidentally, Google’s inbuilt calculator [Google09W] reports that the frequency of “once in a blue moon” is $1.16699016 \times 10^{-8}$ Hz – about once every two and half years! This is a good example to demonstrate the relationship between frequency and time: they are inversely proportional to each other, and so, the shorter the period or wavelength, the higher the frequency of the wave, and vice versa.

2.1.3 Phase

A necessary concept to be grasped fairly early on (the relevance of which will become more apparent later) is that of *phase*. Perhaps the easiest way to explain what is meant by this term is to compare sine and cosine functions with the same frequencies. We know from the basic properties of sine and cosine that the amplitude of the former begins at zero (at time $t = 0$) and the latter at its maximum, 1.

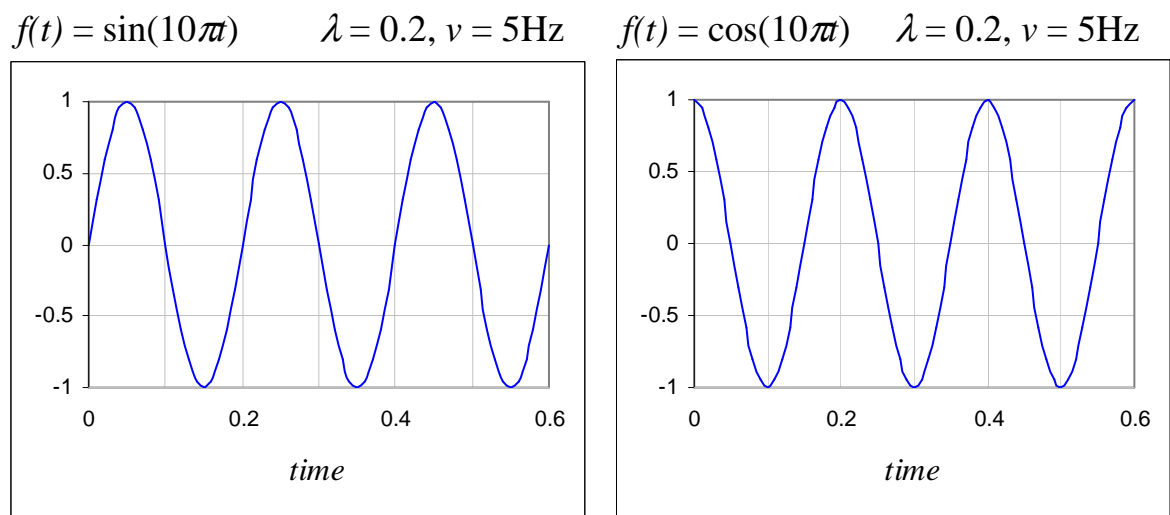


Figure 2.3 – Sine and cosine graphs, both with frequency 5Hz

It can be seen that the graphs in **Figure 2.3** are waves which have exactly the same period length, and yet the cosine graph is displaced in time from the sine graph by a quarter of a cycle (or 0.05 seconds). This displacement in time is what is known as the *phase difference* between the two signals, and it is measured in terms of the difference in the position of their periods at any given instant. In other words, the phase difference is the change in *angle* between two signals, which we shall refer to as ϕ , measured in radians. For sines and cosines, $\phi = \pi/2$, since

$$\cos \theta = \sin(\theta + \pi/2).$$

Figure 2.4 further illustrates phase difference, ϕ , between two angles, θ_1 and θ_2 , which are different locations in the cycle of a wave with unspecified frequency, ν .

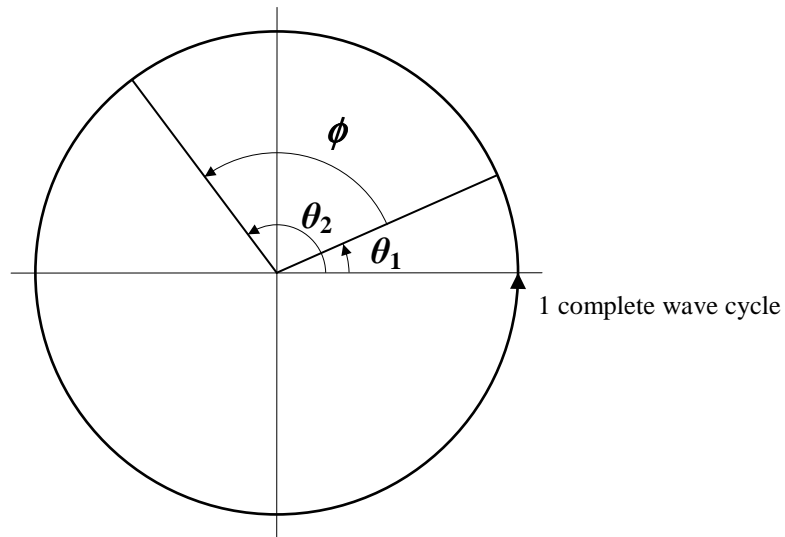


Figure 2.4 – Polar graph showing two angles θ_1 and θ_2 with phase difference ϕ

Note that if one were to remove the time axis in **Figure 2.3** and imagine the waves carrying on to infinity in both directions, there would be no way to tell the difference between the two. Phase therefore becomes meaningless without a point of reference. Thus, for practical purposes, waves which could be perceived as either a sine or a cosine are referred to generally as *sinusoids*.

It is also interesting to note that the human auditory system is able to detect extremely subtle phase differences between signals arriving at the left and right ear. In fact this ability is partly what enables us to locate sounds spatially, in particular, laterally. This idea was first proposed by English physicist John W. Strutt, Baron Rayleigh, in 1907 [Stevens65L]. Rayleigh performed a series of experiments to test his theories on *binaural location*, that is our ability to locate sounds by using both of our ears. Amongst other things, Rayleigh rationalized the following:

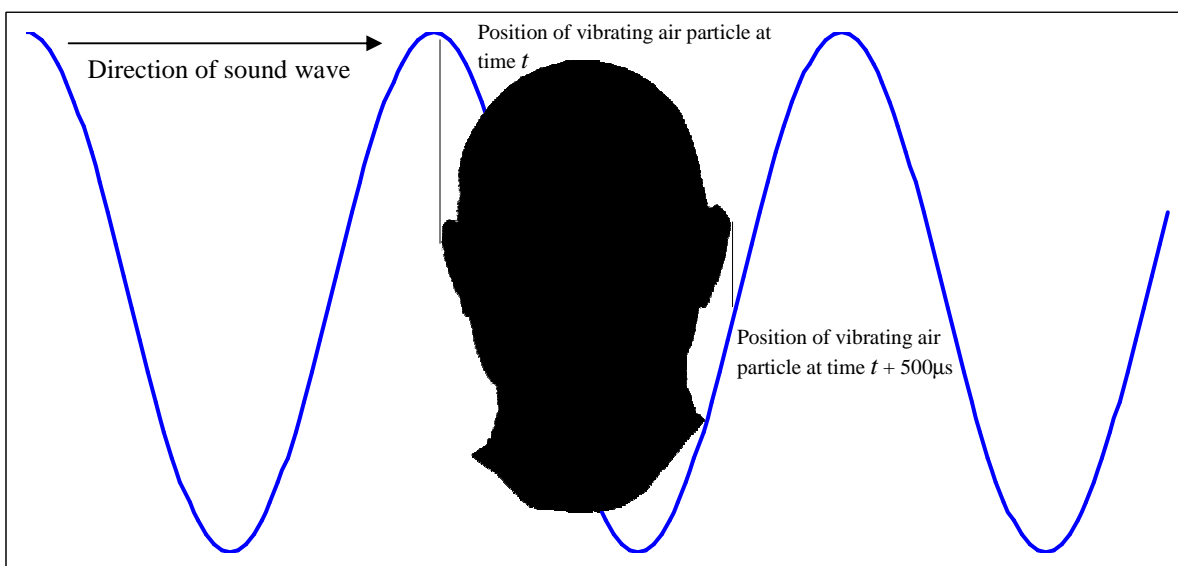


Figure 2.5 – Diagram showing phase difference of a wave at each ear

Regarding **Figure 2.5**, if one imagines a wave approaching the head from left to right, peaks and troughs in the signal arrive at the left ear a small fraction of time before the right – to be more precise, about half a millisecond, assuming that the distance between an average person’s ears is about 17 to 20 centimetres and the speed of sound is 343 metres per second. Considering that the wave period of audible sounds is, for the average human, between 50µs and 50ms (i.e. frequencies between 20kHz and 20Hz) there is a very definite phase difference between the signal at each ear, especially for middle to low frequencies. The brain must somehow be measuring this phase difference and determining location from this information. This assumes that ears analyze sound at the same time, rather than one after the other.

Rayleigh proved this theory by using tuning forks which produced waves with slightly different frequencies in order to create the effect of a sound with a constantly changing phase. As expected, to the observer it appeared as if the sound source was moving from left to right and back again. More than a hundred years on, we now have much more sophisticated technology to exploit binaural location by phase difference and are able to synthesize this and other effects using computer programs. A rather enjoyable demonstration* may be found at QSound.com – *The Virtual Barber Shop* [QSound96W].

2.1.4 Stationary and Non-stationary Waves

It is of vital importance at this point to make it clear that “cycle” implies repetition. One should not assume that wavelengths, and thus frequencies, can simply be measured from one arbitrary peak to the next, because the wave may contain different frequencies *at the same time* – so the next peak along the wave may belong to a different frequency. The graphs in **Figure 2.2** all represent waves with fixed frequencies. A wave of this type, for which the frequency does not change over time, is known as a *stationary* signal. On the other hand, if, as is the case in **Figure 2.1**, the wave’s frequency is variable, it is called a *non-stationary* signal. Waves which contain more than one simultaneous frequency (a musical example of this would be a *chord*) can also be stationary, as long as all of those frequencies remain constant throughout the signal. Another way of looking at it is that any waveform which comprises a continuously repeating pattern is thus stationary and vice versa.

$$f(t) = \frac{1}{3} \sin(10\pi t) + \frac{1}{3} \sin(20\pi t) + \frac{1}{3} \sin(40\pi t).$$

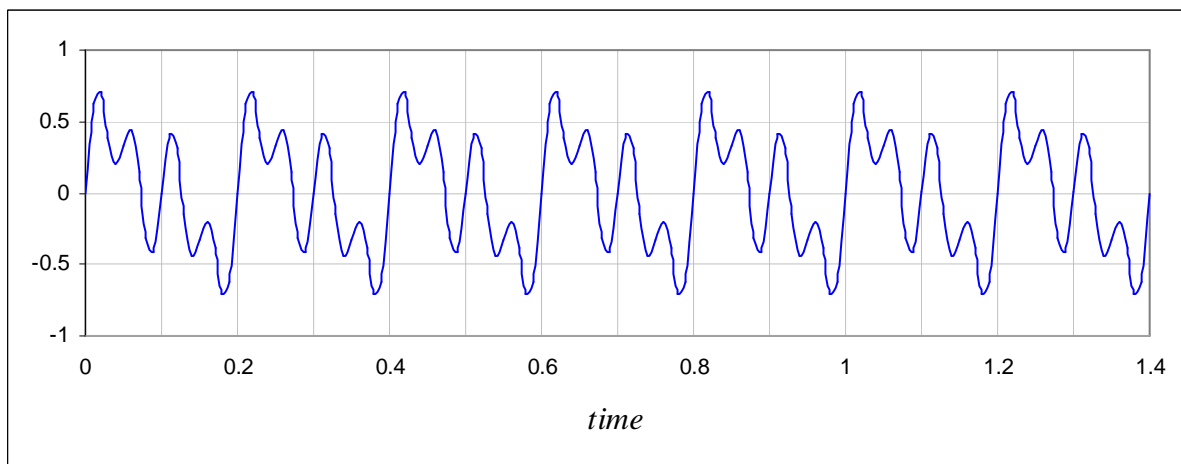


Figure 2.6 – Composite sinusoidal wave

* For a further more simple demonstration of binaural location by phase difference detection, please run *Phaser.exe* on the project CD, located in Software\Phaser. This will create a demo wave file.

This is demonstrated in **Figure 2.6**, which shows a wave containing three frequencies. These are, in fact, the three sine waves from **Figure 2.2** added together and scaled.

The final important point to make about waves is that non-stationary signals could possibly be made stationary simply by repeating the entire signal; however this would, in general, require a slight alteration of the waveform at its beginning and end in order to create a seamless join. Nevertheless, it is something to bear in mind while thinking about the significance of stationarity vs. non-stationarity in the discussion about transforms in section **2.3**.

2.2 Digital Representation of a Signal

Before we can begin to do any sound signal analysis with the aid of a computer, we need a way of representing sound waves, which are continuous / analogue, as discrete / digital entities. This can be done by extracting values or *samples* of the wave's amplitude at regular intervals – this process, naturally, is called *sampling*. The main question is how often, or how close together should these samples be to ensure that all frequencies in the wave are properly represented? Also, how “often” is “often enough” so that no samples are simply redundant information?

2.2.1 The Nyquist-Shannon Sampling Theorem

This theorem, one of the most important in the field of telecommunication, states the following [Shannon49L]:

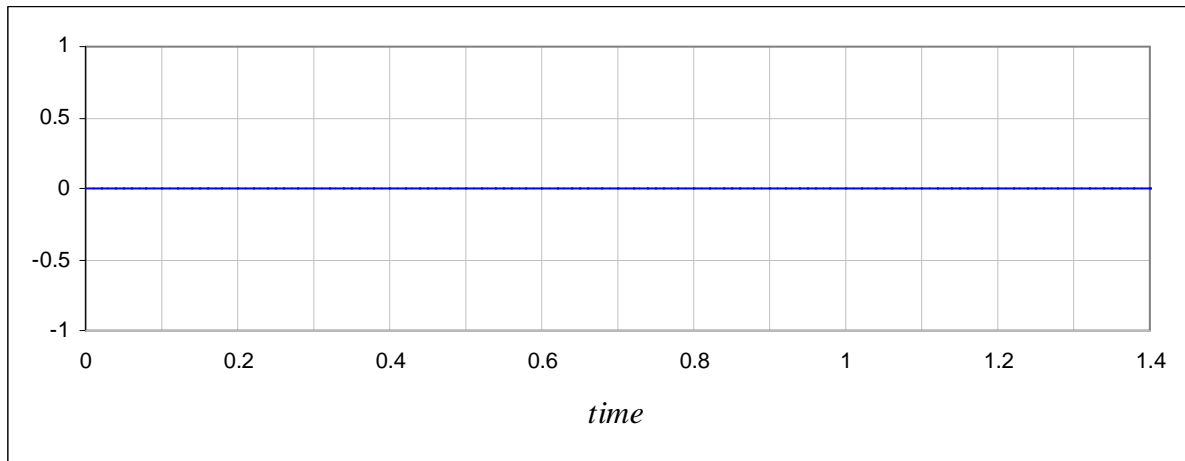
“If a function $f(t)$ contains no frequencies higher than B cps*, it is completely determined by giving its ordinates at a series of points spaced $\frac{1}{2B}$ seconds apart.”

Although Claude Shannon proved and formalized this theory in 1949, Harry Nyquist, a physicist and research engineer at Bell Laboratories, should also be credited with its discovery. The $\frac{1}{2B}$ second rule is alluded to in Nyquist's paper, *Certain Topics in Telegraph Transmission Theory* [Nyquist28L] written several years earlier in 1928 and cited by Shannon in his own paper. Shannon therefore calls this time spacing the *Nyquist interval*. W , the highest frequency limit of a signal, is also known as the *bandwidth* of the signal.

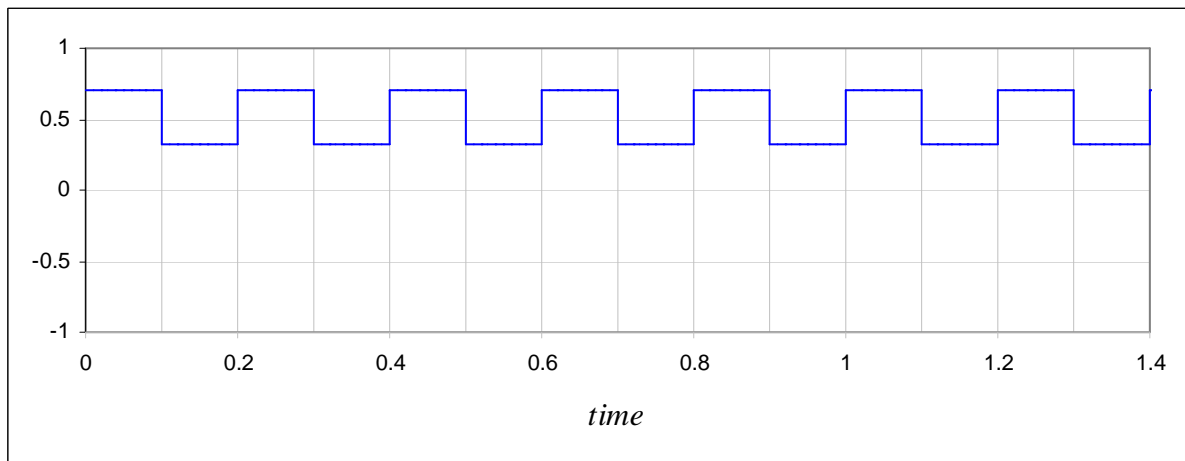
For a proof of the sampling theorem, see [Shannon49L]. However, it is perhaps easier to understand why it should be true by looking at **Figure 2.7**, which is the signal in **Figure 2.6** sampled at different rates. The first of these is useless as it does not represent the signal at all. Since the wave passes through zero every 0.1 seconds (and this is due to the lowest frequency component of 5Hz) if we start sampling at $t = 0$, then every sample will be zero, hence the straight line along the x -axis. This is an extreme example of an effect in signal processing called *aliasing*, which occurs when the sample rate is too low to be able to represent the signal properly. Thus it could represent a number of other continuous functions as well, which can be thought of as aliases of the same digital signal. At a sample rate of 10 per second, this straight line graph could in fact represent *any* continuous function which contains frequencies above 5Hz. At first, this case may look like a counter-example to the sampling theorem, because the signal has been sampled every $\frac{1}{2 \times 5}$ seconds, and so, surely at least the lowest frequency of 5Hz should be supported. However, one must note that the theorem does not state where sampling should begin – only the sampling *interval* is given.

* cps = cycles per second, which is the same as Hertz.

a) 10 samples per second



b) 10 samples per second



c) 20 samples per second

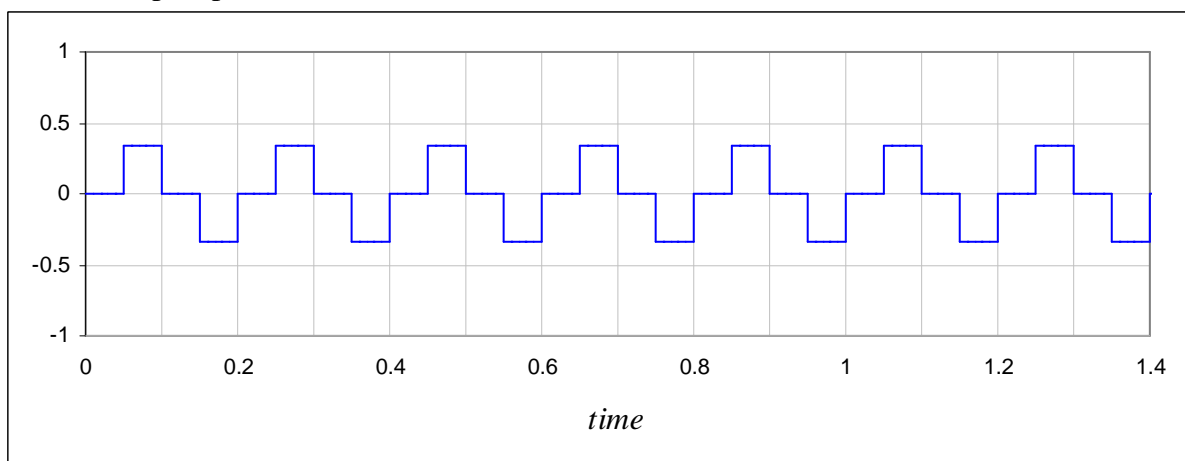
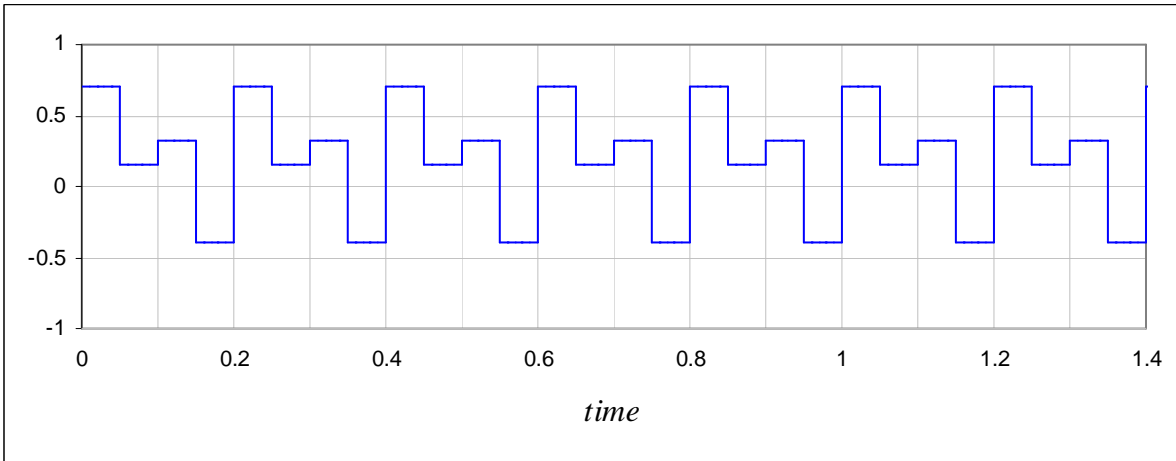
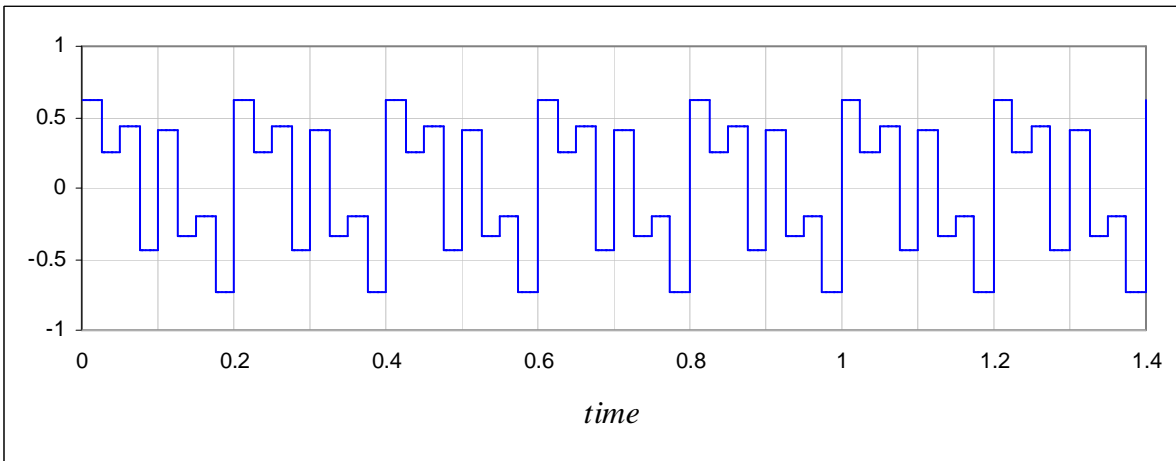


Figure 2.7 – Sampled versions of Figure 2.6

d) 20 samples per second



e) 40 samples per second



f) 50 samples per second

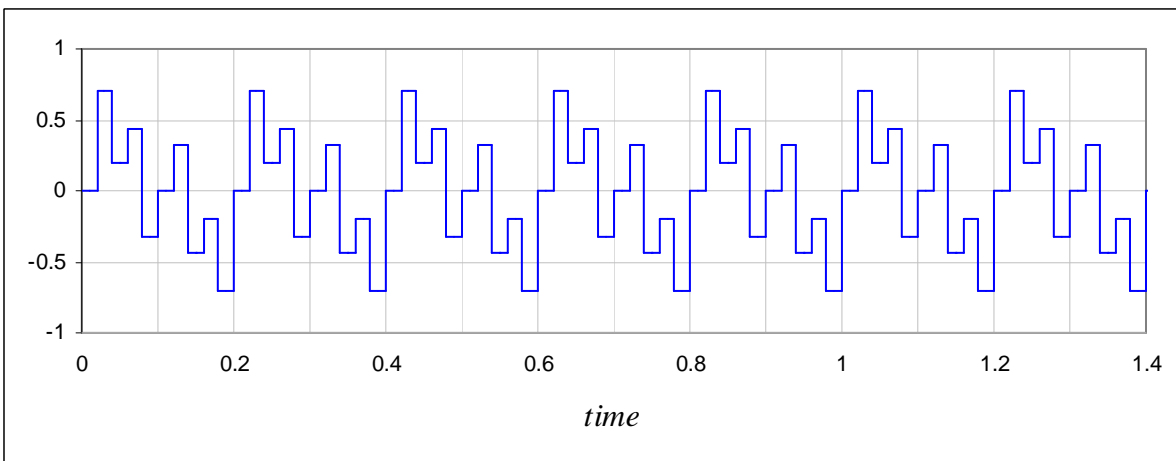


Figure 2.7 (contd.) – Sampled versions of Figure 2.6

The second graph has the same sample rate but the sampling is started at $t = 0.02$. Now this lowest frequency appears as a square wave, showing that it is indeed possible to represent 5Hz at this rate. Graphs **c)** and **d)** are double the sample rate of the previous two. Again, graph **c)** does not yield the middle frequency since sampling began at $t = 0$, however it does appear in **d)**, where sampling once again begins at $t = 0.02$. Graphs **e)** and **f)** both fully represent the signal, although **f)** is *over-sampled*. In **e)** the sampling begins at $t = 0.01$ while in **f)** it is back to $t = 0$. It can be seen that although the waveform is beginning to look less square, and more like **Figure 2.6**, the extra samples in **f)** do not give any further information about the signal than **e)** already did, and they are thus redundant. According to Shannon's theorem, graph **e)** is the optimal way to sample this signal so as to ensure all frequencies are represented, since the Nyquist interval, N , should be

$$N = 1 / (2 \times 20) \text{ seconds.}$$

In other words, 20Hz being the highest frequency in the wave, it should be sampled 40 times per second. Since the sample rate is also a value "per second", it is also, in practice, usually given in Hertz.

As mentioned previously, the normal human hearing range is between about 20Hz and 20kHz. This is the reason why standard quality digitally recorded music, for example on a compact disc, is sampled at over 40kHz (actually 44.1kHz for standard audio CDs) since this sample rate ensures that the highest perceivable frequencies are preserved in the recording process. Please see the excellent video lecture from Academic Earth [Osgood09W] presented by Stanford professor of electrical engineering, Brad Osgood, for a demonstration of aliasing effects caused by undersampling music.

2.3 Transforms

The graphs in **Figures 2.1, 2.2 and 2.3** are all drawn in the *time domain*, meaning that the wave's amplitude is plotted as a function of time. A mathematical *transform* is a conversion from one domain to another in order to obtain a different representation of the original function.

2.3.1 The Most Important Signal Analysis Tool

Arguably, the most well-known and incredibly useful mathematical transform (at least amongst engineers) is the *Fourier transform*. It is named after Jean-Baptiste Joseph Fourier, a French mathematician and physicist who investigated Fourier series, of which the transform is a generalization. Fourier discovered that any continuous square-integrable function could be expressed as an infinite sum (or integral) of sine and cosine functions at certain amplitudes. Fourier first published his findings in 1822 in his work, *Théorie Analytique de la Chaleur* (*The Analytic Theory of Heat*) [Fourier22L], applying his method to finding a solution to the heat equation – a fundamental equation of thermodynamics.

Note that the equation given for $f(t)$ in **Figure 2.6** is in fact already expressed precisely in this way – the amplitude of each of its component sines in this case is $1/3$. Also, each sinusoid is one specific frequency, so putting frequencies on the horizontal axis and amplitudes on the vertical axis, we could draw the bar graph in **Figure 2.8** below as a representation of the signal instead. Thus, when applied to a signal, which is in the time domain, the Fourier transform effectively transmigrates it to the frequency domain.

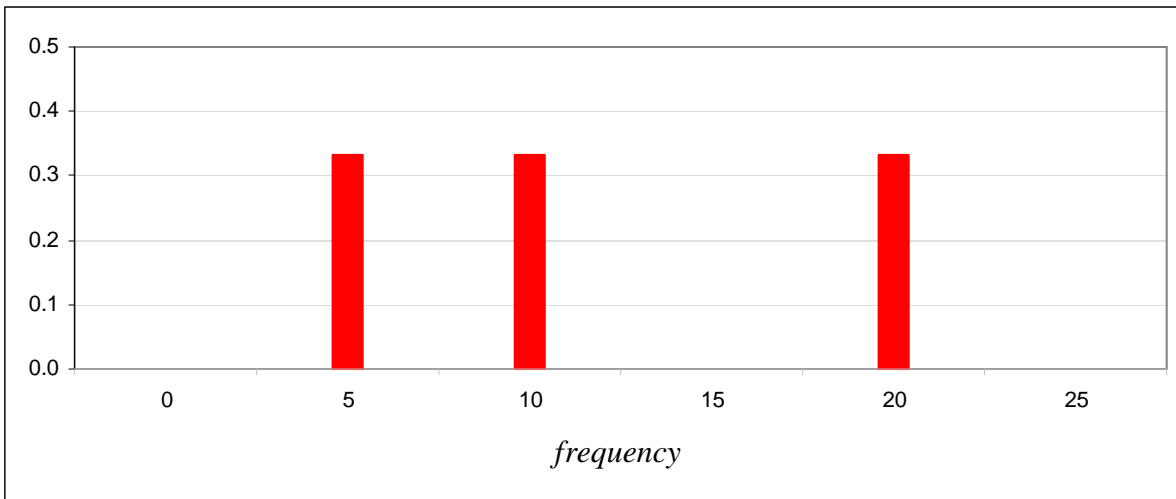


Figure 2.8 – Frequency domain representation of signal in Figure 2.6

The Fourier transformed function, $F(\nu)$ of a signal, $f(t)$ is given by [Wolfram09W]:

$$F(\nu) = \int_{-\infty}^{+\infty} f(t)e^{-i2\pi\nu t} dt. \quad [2.1]$$

Here, Euler's formula ($e^{i\theta} = \cos \theta + i \sin \theta$, where $\theta = 2\pi\nu t$) is being used to express the component sines and cosines in a more simplified form. As is evident from the limits of the integral, the domain of the transform is infinite – amplitudes are calculated for every frequency from negative infinity to positive infinity. Since negative frequencies are, practically speaking, indistinguishable from positive frequencies, the resulting graph of a Fourier transform will always be symmetrical (unless the original function is complex – see Chapter 6). For practical purposes, negative frequencies are almost always ignored, but theoretically they are very important to bear in mind, as will be seen in Chapter 7.

The Fourier transform may be inverted to return a function of frequency to the time domain. The inverse Fourier transform is given by [Wolfram09W]:

$$f(t) = \int_{-\infty}^{+\infty} F(\nu)e^{i2\pi\nu t} d\nu. \quad [2.2]$$

It can be seen that the only real difference between the two equations is the sign of the exponent.

2.3.2 The Discrete Fourier Transform

Shannon's sampling theorem gives us a way of expressing waves as discrete signals, but we need to do the same with the continuous Fourier transform so that we have a way of processing those waves using a computer algorithm. In other words, we need a discrete version of the transform which operates over a fixed or finite domain.

The discrete Fourier transform (DFT) can be derived from the continuous case by considering the transformed signal simply as another discrete set of samples in the frequency domain with some frequency spacing, $\Delta\nu$.

Firstly, from the sampling theorem, we know that a frequency, B , will be the largest supported if a signal is sampled with time spacing, Δt , of $\frac{1}{2B}$ seconds. This is really just saying that the upper bound of the DFT graph will be B . Remembering the symmetry of the Fourier transform, the largest supported negative frequency is therefore $-B$, and so the entire bandwidth will be $2B$ (since the range of the DFT is from $-B$ to $+B$). In the time domain, the discrete signal comprises an array of N samples, f_n from $n = 0$ to $N - 1$. Therefore, if L is the length of the signal in time,

$$N \times \Delta t = L. \quad [2.3]$$

Hence

$$2B = N/L. \quad [2.4]$$

Similarly, in the frequency domain, assuming the transformed function will have the same number of samples, N , as the original signal, we have, for the frequency sample spacing, $\Delta \nu$,

$$N \times \Delta \nu = 2B = N/L, \quad \text{from [2.4]}$$

so

$$\Delta \nu = 1/L. \quad [2.5]$$

We now have both the range and sampling interval needed for the discrete Fourier transform and so we can begin sampling the continuous function, which we recall from [2.1] is defined as:

$$F(\nu) = \int_{-\infty}^{+\infty} f(t) e^{-i2\pi\nu t} dt.$$

The first step towards discretization is to approximate this integral by a finite sum and to use index variables, n for samples of t and m for samples of ν [Ewer10L]:

$$F(\nu_m) \approx \sum_{n=0}^{N-1} f(t_n) \cdot e^{-i2\pi\nu_m t_n} \Delta t.$$

Now, from the prescribed sampling intervals above, in the time domain the value of t at index n will be:

$$t_n = n/(2B),$$

and in the frequency domain, the m^{th} value of ν is:

$$\nu_m = m/L.$$

So

$$\begin{aligned} v_m t_n &= mn/2BL \\ &= mn/N. \end{aligned} \quad \text{from [2.4]}$$

Also

$$\Delta t = \frac{L}{N}. \quad \text{from [2.3]}$$

And so, for each of the sampled values of $F(v)$ we have [Ewer10L]:

$$F(v_m) \approx \frac{L}{N} \sum_{n=0}^{N-1} f(t_n) \cdot e^{-i2\pi mn/N}.$$

The reverse discrete transform may be derived similarly, thus [Ewer10L]:

$$f(t) = \int_{-\infty}^{+\infty} F(v) \cdot e^{i2\pi vt} dv. \quad \text{from [2.2]}$$

Then

$$\begin{aligned} f(t_n) &\approx \sum_{m=0}^{N-1} F(v_m) \cdot e^{i2\pi v_m t_n} \Delta v \\ &= \frac{1}{L} \sum_{m=0}^{N-1} F(v_m) \cdot e^{i2\pi mn/N}. \end{aligned} \quad \text{from [2.4] and [2.5]}$$

Note the normalization factors, L/N in the forward transform and $1/L$ in the reverse transform. Since multiplying by L in the forward transform and dividing by L in the reverse transform cancels them out, the L 's may be omitted [Ewer10L]. Also, to get these functions in their generalized discrete forms, since the index variables, n and m , are now arbitrary, we may re-express the set of signal samples, $f(t_n)$, as a discrete array, f_n . Similarly, the frequency samples, $F(v_m)$, together become F_m . So, finally, we define the forward transform to be

$$F_m = \frac{1}{N} \sum_{n=0}^{N-1} f_n \cdot e^{-i2\pi mn/N},$$

whose inverse will be

$$f_n = \sum_{m=0}^{N-1} F_m \cdot e^{i2\pi mn/N}.$$

Furthermore, since in much of what follows our primary interest is relative strengths of frequencies, it is not important to include the factor of $1/N$ [Ewer10L]. It must not be forgotten, however, if the intention is to resynthesise a transformed signal from the frequency domain. Therefore, again, the only practical difference between the forward and reverse DFTs is the sign of the exponent.

The order of the complex number calculations for the DFT (and its inverse) is N^2 . This is rather slow and quite impractical, especially when dealing with sound signals, which usually have many thousands of samples. In 1965, American mathematicians James Cooley and John Tukey developed a divide-and-conquer type algorithm capable of doing the same calculation in $M\log N$ operations [Cooley65L]. Their work was based on that of Gordon Danielson and Cornelius Lanczos [Danielson42L] who, in 1942, discovered a method for re-expressing the DFT as a combination of two DFTs over two halves of the original signal, now called the Danielson-Lanczos Lemma. This algorithm later became known as the Fast Fourier Transform.

A final note before proceeding with the next section: the sample intervals, Δt and $\Delta \nu$, affect the granularity of the time and frequency domains which they respectively break down. Thus they are usually referred to as the *resolution* of those domains.

2.4 The Fast Fourier Transform Algorithm

The Cooley-Tukey Radix-2 Decimation In Time (DIT) algorithm is still the most popular method for calculating DFTs. Although it is not the fastest algorithm available (higher radices are slightly faster), as will be seen, it is fairly easy to understand and implement, and it is more than adequate for the purposes of this research. A fairly detailed explanation of the implementation of the algorithm is given here, since this method is at the very core of the entire process of music recognition. Thus it is important to be clear about how it works exactly.

The following sources were used to develop an implementation of the algorithm as it appears in the software written for this research:

- *fftw-3.0.1* (`libbench2/mp.c`) [FFTW03S] and [Frigo03L]
- Code by N. M. Brenner from *Numerical Recipes in C: The Art of Scientific Computing* (Chapter 12) [Brenner92L]
- *An in-place complex-complex FFT* [Bourke93W]
- *Fast Fourier Transform* – Don Cross [Ackers00W]
- *woD_FFT* from *A Simplified Approach to Image Processing* [Crane97W]

2.4.1 Bit Reversal

The transform begins with a bit reversal step. The “bit reversal” applies to the indices of the samples in the input data array, rather than the samples themselves. It is a reordering of the array, since the decimations in time split and reorder the samples into even and odd sets at each step. Prior shuffling ensures that the output data is in the correct sequence at the end of the calculation.

The method of reordering is illustrated below in an example, using a set of 8 samples. As will become apparent, the number of samples in the digital signal must always be a power of two in order for this algorithm to work.

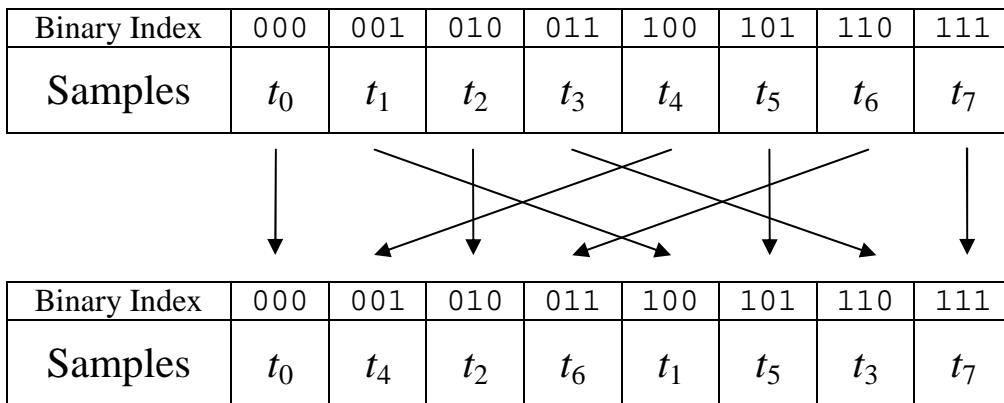


Figure 2.9 – Bit Reversal

In **Figure 2.9**, each sample, t_n is swapped with another, the binary index of which has a reversed bit order. E.g. sample t_1 (001) is exchanged with sample t_4 (100). Note that samples with palindromic binary indexes, such as t_5 (101), do not get switched. The method of shuffling by decimations in time may be clarified by examining what happens to the order of the data with each decimation. As shown in **Figures 2.10** and **2.11**, a decimation sorts the previous set of samples into two sets, placing even indexed samples on the left and odd samples on the right.

As can be seen, the result of the two decimations is that the data becomes ordered as shown previously in **Figure 2.9**. As for the explanation of why the data is re-ordered in this fashion, some more mathematics is necessary. The next sub-section shows the derivation of the previously mentioned Danielson-Lanczos Lemma, which is fundamental to Cooley & Tukey’s algorithm.

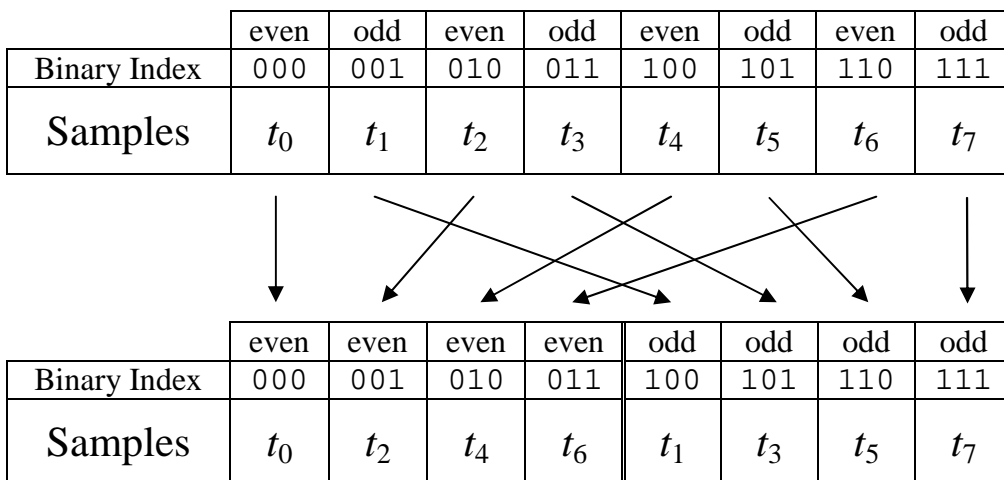


Figure 2.10 – Decimation 1

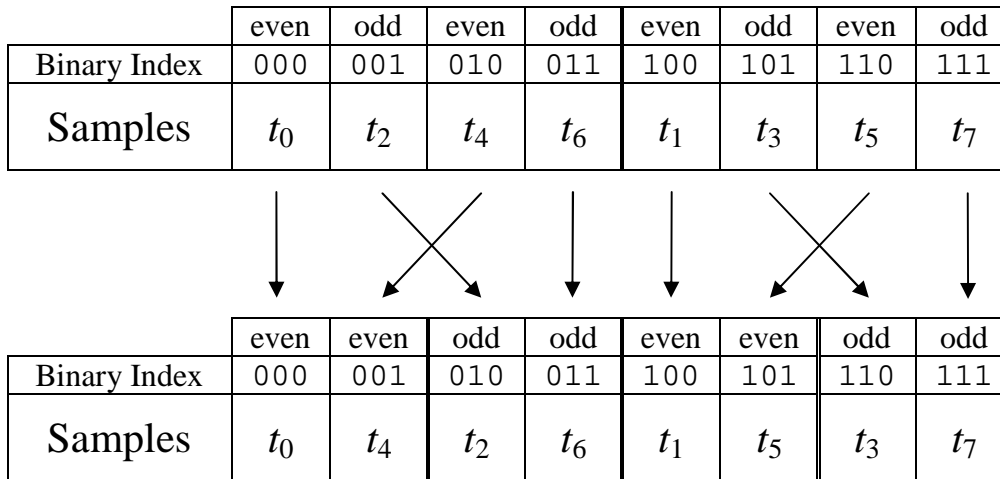


Figure 2.11 – Decimation 2

2.4.2 The Danielson-Lanczos Lemma

Using the same notation as in section 2.4, if $f(t)$ is a wave function from which N samples are taken (where $N = 2^m$) to yield a discrete signal, f_n , then the discrete Fourier transform, F_m , of this signal is [Wolfram09W]:

$$F_m = \sum_{n=0}^{N-1} f_n e^{-i2\pi mn / N}.$$

For simplicity, let

$$W_N = e^{-i2\pi / N}. \quad [2.6]$$

So, ignoring the normalizing for further simplicity, we have:

$$F_m = \sum_{n=0}^{N-1} f_n W_N^{mn}.$$

Note that the above calculation, as it stands, involves on the order of N^2 operations. As mentioned above, the radix-2 algorithm splits the data into 2 halves of even and odd indexed samples. This is due to the following important first step:

$$\begin{aligned} F_m &= \sum_{\text{even}(n)} f_n W_N^{mn} + \sum_{\text{odd}(n)} f_n W_N^{mn} \\ &= \sum_{n=0}^{\frac{N}{2}-1} f_{2n} W_N^{2mn} + \sum_{n=0}^{\frac{N}{2}-1} f_{2n+1} W_N^{m(2n+1)}. \quad [2.7] \end{aligned}$$

Next, a little algebra – Recall from [2.6] that $W_N = e^{-i2\pi/N}$, so

$$\begin{aligned} W_N^{2mn} &= e^{-i2\pi 2mn/N} \\ &= e^{-i2\pi mn/\frac{N}{2}} \\ &= W_{N/2}^{mn}. \end{aligned} \quad [2.8]$$

Also,

$$\begin{aligned} W_N^{m(2n+1)} &= e^{-i2\pi m(2n+1)/N} \\ &= e^{-i2\pi 2mn/N - i2\pi m/N} \\ &= e^{-i2\pi 2mn/N} \cdot e^{-i2\pi m/N} \\ &= W_{N/2}^{mn} \cdot W_N^m. \end{aligned} \quad [2.9]$$

Let g_n and h_n be two discrete signal functions, such that

$$g_n = f_{2n} \quad \text{and} \quad h_n = f_{2n+1}.$$

Then, substituting into [2.7],

$$F_m = \sum_{n=0}^{\frac{N}{2}-1} g_n W_N^{2mn} + \sum_{n=0}^{\frac{N}{2}-1} h_n W_N^{m(2n+1)}.$$

From [2.8] and [2.9], this becomes:

$$F_m = \sum_{n=0}^{\frac{N}{2}-1} g_n W_{N/2}^{mn} + W_N^m \sum_{n=0}^{\frac{N}{2}-1} h_n W_{N/2}^{mn}.$$

So, if G_m and H_m are defined as the $N/2$ -point Fourier transforms of g_n and h_n respectively, then this becomes:

$$F_m = G_m + W_N^m H_m, \quad 0 \leq m < N/2.$$

Now $G_m = G_{m+N/2}$, and $H_m = H_{m+N/2}$, since G_m and H_m are periodic (the pattern repeats for the next half N samples). Also, $W_N^{m+N/2} = -W_N^m$, since

$$\left(W_{N/2}^n\right)^{N/2} = e^{-i2\pi n} = 1, \quad \text{and} \quad W_N^{N/2} = e^{-i\pi} = -1.$$

So, for the complete transform:

$$F_m = G_m + W_N^m H_m, 0 \leq m < N/2, \quad [2.10]$$

and

$$F_{m+N/2} = G_m - W_N^m H_m, 0 \leq m < N/2. \quad [2.11]$$

The factor, W_N , is known as the N^{th} root of unity. Collectively, these are sometimes called the *twiddle factors*.

2.4.3 Recombining of Transforms

Now that F_m can be expressed as a combination of two DFTs on the even and odd halves of the input data, each of those DFTs may be broken down (decimated) recursively, all the way down to single point transforms. At this stage, all that needs to be done is to copy the input data directly to output, which is precisely what happens at the bit reversal stage. Note that calculation time is now much improved and is on the order of $M \log(N)$ operations.

What remains to be done is to re-combine, from the inside out, pairs of samples into 2-point transforms, then those pairs into 4-point transforms and so on, until finally the two halves of the entire data set are combined to render the complete transform. Since the “combining” is all complex number multiplication and addition, this part of the algorithm is a little more fiddly. However, here is a rough explanation of how it works:

2.4.3.1 Loop Control

There are three loops in the combination part of the code. The outside loop keeps track of the number of points – `npoints` – per transform, at each combination. This starts at 1 and is doubled at each step until it is half the length of the input data. Nested within, the next loop counts the current point, `i`, from 0 to `npoints`. This provides a starting point for the third, innermost loop, which is nested within the second. This loop keeps track of the current position in the data array with 2 variables, `j` and `k`. `j` marks the position of data for the part of each transform given by equation [2.10] above, and `k` marks the location of data for equation [2.11]. The variable `jstep` moves the markers to the position of the next pair of transforms, to be combined into one at each iteration of the loop. The shell code looks like this:

```
for(npoints = 1; npoints < n; npoints = jstep) {
    jstep = npoints << 1;
    ...
    for(i = 0; i < npoints; i++) {
        for(j = i; j < n; j += jstep) {
            k = j + npoints;
            ...
        }
        ...
    }
    ...
}
```

Below is an example of how this description of flow works on a set of 8 samples. It is taken from the output of a debugging version of the sample FFT program on the project CD:


```

Loop 1: npoints = 1, jstep = 2
  Loop 2: i = 0
    Loop 3: j = 0, k = 1
    Loop 3: j = 2, k = 3
    Loop 3: j = 4, k = 5
    Loop 3: j = 6, k = 7

```

Combination 1:
4 pairs of DFTs of length 1 into 4 DFTs

j ₀₀	k ₀₀	j ₀₁	k ₀₁	j ₀₂	k ₀₂	j ₀₃	k ₀₃
0	1	2	3	4	5	6	7



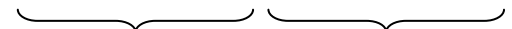
```

Loop 1: npoints = 2, jstep = 4
  Loop 2: i = 0
    Loop 3: j = 0, k = 2
    Loop 3: j = 4, k = 6
  Loop 2: i = 1
    Loop 3: j = 1, k = 3
    Loop 3: j = 5, k = 7

```

Combination 2:
2 pairs of DFTs of length 2 into 2 DFTs

j ₀₀	j ₁₀	k ₀₀	k ₁₀	j ₀₁	j ₁₁	k ₀₁	k ₁₁
0	1	2	3	4	5	6	7




```

Loop 1: npoints = 4, jstep = 8
  Loop 2: i = 0
    Loop 3: j = 0, k = 4
  Loop 2: i = 1
    Loop 3: j = 1, k = 5
  Loop 2: i = 2
    Loop 3: j = 2, k = 6
  Loop 2: i = 3
    Loop 3: j = 3, k = 7

```

Combination 3:
1 pair of DFTs of length 4 into 1 final DFT

j ₀₀	j ₁₀	j ₂₀	j ₃₀	k ₀₀	k ₁₀	k ₂₀	k ₃₀
0	1	2	3	4	5	6	7



2.4.3.2 Twiddle Factors

The rest of the code – the meat inside the loops – calculates w_r and w_i , the real and imaginary parts of the twiddle factors at each point in the transforms. Recall that

$$W_N^m = e^{-i2\pi m/N}.$$

If instead this is expressed using trigonometry, then it becomes:

$$W_N^m = \cos(2\pi m/N) - i \sin(2\pi m/N).$$

Now the complex components of the twiddle factors, w_r and w_i may be calculated by increasing the angle, $\theta = 2\pi m/N$, between 0 and π by a factor of θ_k . θ_k begins at π and is halved for each transform combination at the end of the outer loop, corresponding to the halving of the number of iterations of the third loop. The variables w_{kr} and w_{ki} are the real and imaginary parts of the complex number derived from θ_k , the life cycles of which are shown below:

Initially:

```

wkr = -1.0; // theta = pi
wki = 0.0;

```

Then:

```

for(npoints = 1; npoints < N; npoints = jstep) {
    ...
    wki = -sqrt((1.0 - wkr) / 2.0);
    wkr = sqrt((1.0 + wkr) / 2.0);
}

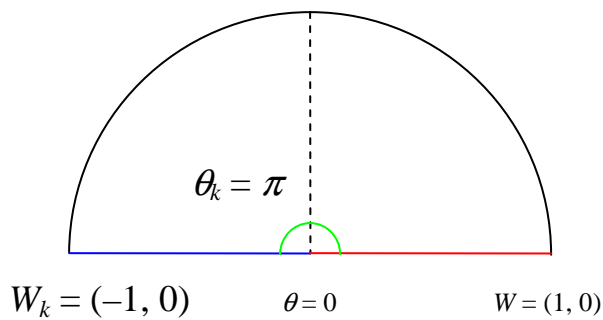
```

As implemented here, it can be seen that the values may be calculated quite neatly using square roots, rather than with sine and cosine functions, by using the double angle formula for cosine.

For the full code of the FFT, please see the listing in **Appendix C.1**, or else view the backend source code of *Wave Processor**.

Finally, the cycle of values for $W = (w_r, w_i)$ may best be illustrated by the following diagrams in **Figure 2.12**, which pertain, once again, to a set of 8 samples. For simplicity, positive imaginary components have been used, which would thus pertain to the inverse transform:

Combination 1 – 4 × length 1 transforms



Combination 2 – 2 × length 2 transforms

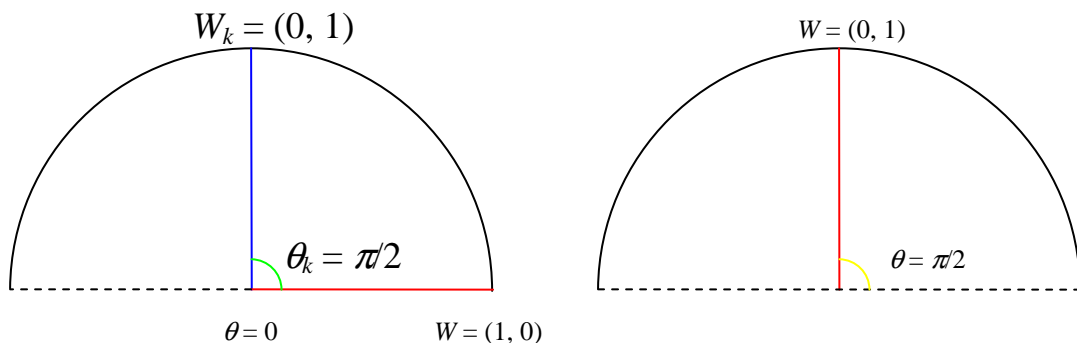


Figure 2.12 – Polar complex number diagrams of combinations

* This may be found on the project CD in the folder Software\Wave Processor\Source

Combination 3 – 1 × length 4 transforms

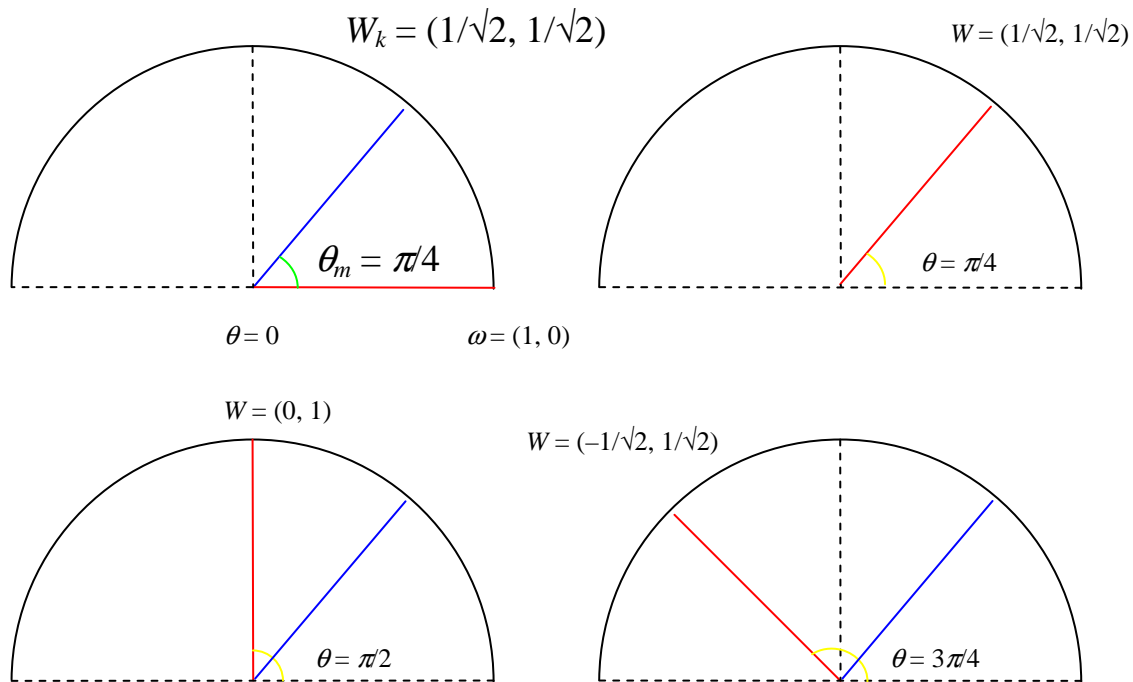


Figure 2.12 (contd.) – Polar complex number diagrams of combinations

From the above, it becomes clear that the only change to the code which needs to be made in order to render the inverse transform is to make `wmi` positive instead of negative at each update, thus the relevant piece of code becomes:

```
wki = sqrt((1.0 - wkr) / 2.0);
```

In the implementation in the software for this research, as is common practice, the forward and inverse transforms have been combined in the same function. The variable `dir` controls which is to be performed: for the forward transform it is set to 1 and for the inverse, `-1`. The last thing to remember is to do the normalizing (by $1/N$) for the forward transform only.

2.4.4 Output

The output of the DFT is very often misunderstood and should be clarified at this point. It is important to realise that the discreteness means that values are quantized in the frequency domain, i.e. there are no “in-between” values. The DFT is possibly better thought of as a probability distribution histogram with frequency *bins*. In other words, amplitudes on a DFT graph actually represent probabilities of a frequency being a certain value, rather than exact measures. This draws a direct parallel with the uncertainty principle in quantum theory which states that certain pairs of properties (in this case time and frequency) cannot be measured precisely simultaneously. The importance of this realisation was pointed out by Dennis Gabor, who formalized the uncertainty relation between frequency and time resolution in the Fourier transform [Gabor46L] introduced in subsection 2.3.2.

Referring back to that discussion, the frequency resolution is inversely proportional to the length of the signal in time. So, the longer the signal, the smaller $\Delta\nu$ becomes. However, $\Delta\nu$ is also directly proportional to the bandwidth, which in turn is affected by the sample rate and

thus Δt . The following summarizes this three-way *reciprocity relationship* between Δt , Δv and N , the number of sample points [Osgood09W].

Time Limitedness: $L = N\Delta t$

Bandwidth: $2B = N\Delta v$

$$\begin{aligned}\Delta t \Delta v &= \frac{L}{N} \cdot \frac{2B}{N} \\ &= \frac{N}{N^2} \\ &= \frac{1}{N}.\end{aligned}$$

While the input of the DFT is, for most practical purposes, a function of time, strictly over a discrete set of real values, the output of the transform is complex and so it comprises two arrays. Going back to the original description of the Fourier transform as the re-expression of a function in terms of component sines and cosines, from complex number theory, the real values are thus the cosine amplitudes, and the imaginary array is the set of sines. **Figure 2.13** illustrates this. In the graph, x is the real component of the complex coefficient, y is the imaginary part. The distance of v from the origin, $r = |v|$, is then the *magnitude* of v , and it is calculated by the Pythagorean theorem:

$$r = \sqrt{x^2 + y^2}.$$

When drawing a DFT graph, this is the value usually plotted. Finally, θ is a measure of the *instantaneous phase* of the frequency, which can be calculated with the argument function, $\arg(v)$, which is $\arctan(y/x)$ when $x \neq 0$, and π or $-\pi$ when $x = 0$ and $y > 0$ or $y < 0$ respectively. The importance and usefulness of phase information in the DFT will be discussed further in Chapter 4 in the section on the Phase Vocoder.

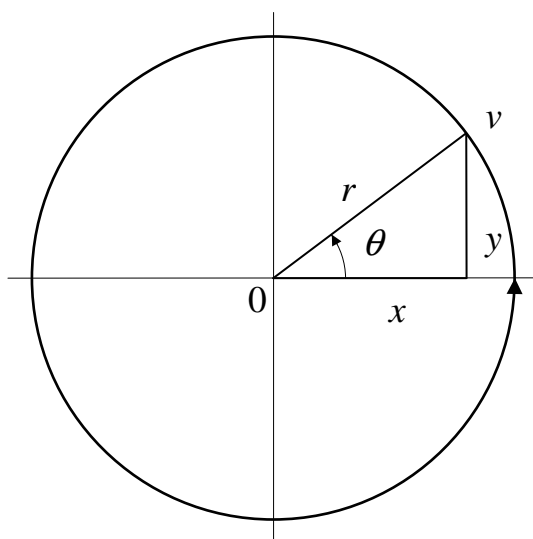


Figure 2.13 – Polar graph representation of a complex Fourier coefficient, v

3 An ABC of Music Theory

Having covered the basics of signal analysis, it is also necessary to provide the reader with an overview of music theory from a more technical point of view, so that the musical terminology used throughout the rest of this thesis may be understood and used freely. The discussions here also begin to link up some of the ideas from the previous chapter in the context of music. A discourse on the origins of music and the natural tonal system has been included, mainly for interest's sake, but also in order to further familiarize the reader with the language. The primary sources for this chapter were [Scholes65L] and [Grove00L], although much of this information may be found in any suitably comprehensive music encyclopedia.

3.1 Musical Pitch

From the Oxford English Dictionary, pitch, in the context of music, is defined as “*The quality of a sound, esp. one produced by a musical instrument or voice, which is governed by the frequency of the vibrations producing it, and which determines its highness or lowness of tone (a rapid vibration corresponding to a high tone)*” [OED05W].

Here, the notion of “high” and “low” tones is, of course, not a physical one, but, as the phrase in parentheses suggests, more of a convention decided upon. Perhaps human beings first associated “high” notes with bird calls – these particular sounds came from above and were thus, perhaps subconsciously, associated with concepts of “up” and “high”. The idea of the height of a pitch also corresponds to frequency measurements – “Lower” pitches have smaller numeric frequency values and vice versa.

3.1.1 The Relationship between Pitch and Frequency

The exact relationship between frequency and pitch is a *dyadic* one, meaning that factors which are powers of 2 are involved when converting between the two. This is a well known relationship, discovered (at least, first recorded) by Pythagoras [Scholes65L], who performed experiments with different lengths of cord, observing that if he plucked a string and noted its pitch, then stopped the string halfway and plucked it again, he got a note which sounded very similar to the first, but which was higher. Dividing it in half again (i.e. into quarters) produced a note, once again with the same sound quality but which was even higher. What Pythagoras was actually generating were pitches an *octave* apart from each other. In music, this is exactly how the term *octave* is defined: the interval between two pitches which have the same quality of sound, one of which is double the frequency of the other. The reason for the name, *octave*, which surely implies something to do with *eight*, will be explained in due course. Pythagoras continued with further experiments where he divided the string into thirds and so on, and went on to generate what was later called the *harmonic series*.

3.1.2 The Harmonic Series

Whenever you play a single note on any musical instrument, you are not actually hearing just one frequency, but a composite spectrum of frequencies, just as white light is made up of multiple colours. This spectrum of frequencies is known as the *harmonic series* of the note, and each individual frequency within is called a *harmonic*. The most clearly audible pitch is usually the lowest frequency in the series, and is called the *fundamental note* – often referred to by acoustic engineers as f_0 . The series follows a pattern which may be explained by

examining all the ways in which a string, for example on a guitar, may vibrate along its length. Firstly, the fundamental note is a result of the string vibrating along its entire length as illustrated in the figure below:

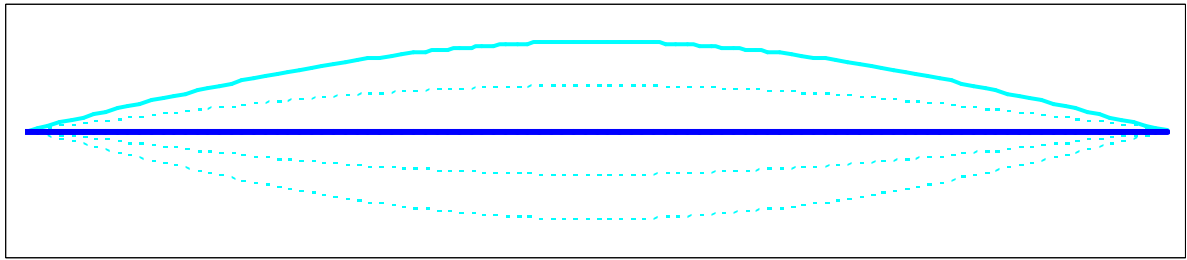


Figure 3.1 – A freely vibrating string

If the string is stopped halfway along its length, so that the midway point is stationary, it may also vibrate freely about that point and its two endpoints, which are also stationary. This is depicted in **Figure 3.2**. Now the wavelength of the resulting signal has been halved, and therefore the frequency is doubled. Subsequently we have a pitch which sounds an octave above the fundamental. This pitch is the *first harmonic* in the series.

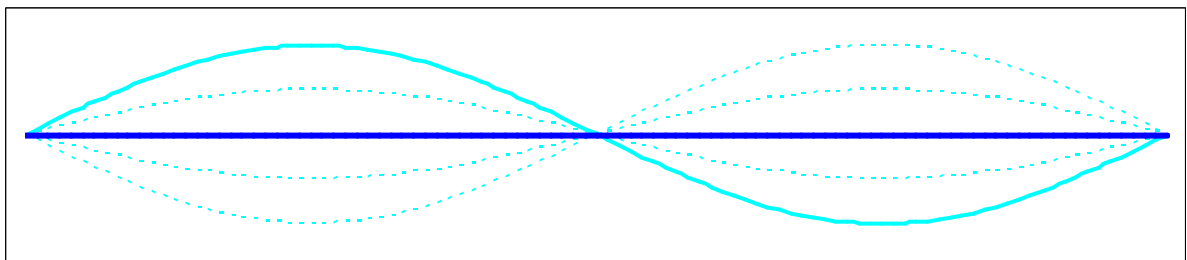


Figure 3.2 – A vibrating string stopped half-way

The next harmonic is found by dividing the string into thirds, so that there are two stationary points in between the two ends. The resulting frequency – the *second harmonic* – is then three times that of the fundamental note.

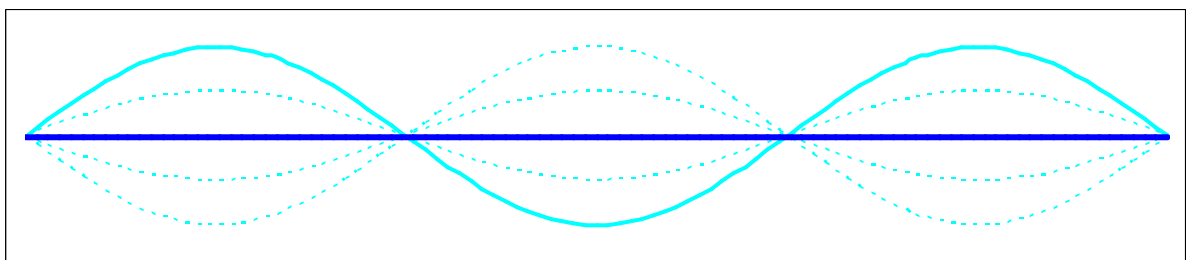


Figure 3.3 – Dividing the string into thirds

We may continue generating the series by dividing the string up into increasing numbers of equal parts, the frequency of each harmonic increasing by that same factor in proportion to the fundamental frequency at each stage. As the frequencies get higher, the intervals in pitch between the harmonics become smaller and smaller and tend towards zero (as the number of subdivisions of the string tends towards infinity). In reality, only the first few harmonics in the series are ever perceived by our ears, to varying degrees. This is to do with the fact that

the string is able to vibrate most freely and easily about fewer stationary points than more, so lower order harmonics (especially the fundamental) will normally have higher amplitudes.

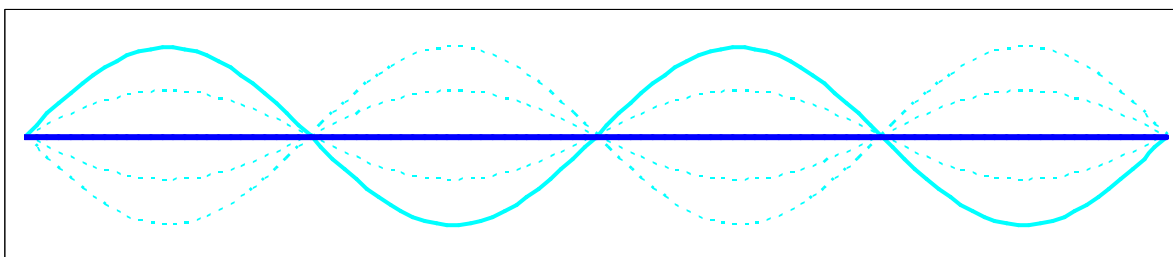


Figure 3.4 – Four divisions of the string: third harmonic

3.1.3 Natural Pitch Intervals

Armed with the harmonic series, we now have a basis by which pitch may be more formally defined, and (hypothetically) a natural explanation for the existence of the entire Western tonal system [Scholes65L]. In order to do this, we need to look more closely at the *intervals* between different pitches within this tonal system, that is the change in pitch (or frequency) from one note to the next, and consider how they may have come into being. The thought process presented here is probably quite similar to what Pythagoras and other Greek philosophers in his time (the first to ponder the mathematical side to music) would have reasoned in about 550BC [Scholes65L].

As mentioned previously, although the harmonic series in theory has an infinite domain, in practice our ears are able to hear only a few harmonics consciously. Therefore let us begin by examining more closely the first few notes in the harmonic series for an arbitrary frequency, say 262Hz (the choice of which will become clear in the following subsections). In order to calculate the frequencies of each note in a harmonic series, all that is required is to divide by its relative wavelength, i.e. multiply by 2, 3, 4 etc., since the frequency of a (stationary) signal is the inverse of its wavelength (see Chapter 3).

Note in Series	Wavelength	Frequency (Hz)
F ₀ (Fundamental Note)	λ (= 1/262)	262
H ₁ (First Harmonic)	$\lambda/2$	524
H ₂ (Second Harmonic)	$\lambda/3$	786
H ₃ (Third Harmonic)	$\lambda/4$	1048
H ₄ (Fourth Harmonic)	$\lambda/5$	1310

Table 3.1 – Harmonic Series of 262Hz

Note that the intervals between the fundamental note and the first and third harmonics are both octaves, since the frequencies of these tones double from one to the next. As stated before, it may be perceived, audibly (certainly at least by Pythagoras) that the quality of the sound of two pitches which are an octave apart is very similar. This makes sense in a way, since one would expect the very first harmonic of a note to be its closest relative.

For the sake of formulating a hypothesis, let us first assume that the very closest *natural* relationships are those between the first pair of notes in any given harmonic series, using the above argument that they are the sounds which appear to be most similar to our brains. The reason they appear similar to us may be explained by the fact that when two such “related”

itches are played simultaneously, every few wavelengths, the peaks of the two signals coincide, boosting each others' amplitudes, i.e. they *resonate* with one another. The smaller the lowest common multiple of their wavelengths, the more they will resonate in this way. The graph below demonstrates this principle clearly for the interval of an octave: the two thin waves are $y = \cos(x)$ and $y = \cos(2x)$, and the thick blue signal is their sum. Note how the peaks of the lower frequency wave are now effectively amplified to twice their original value.

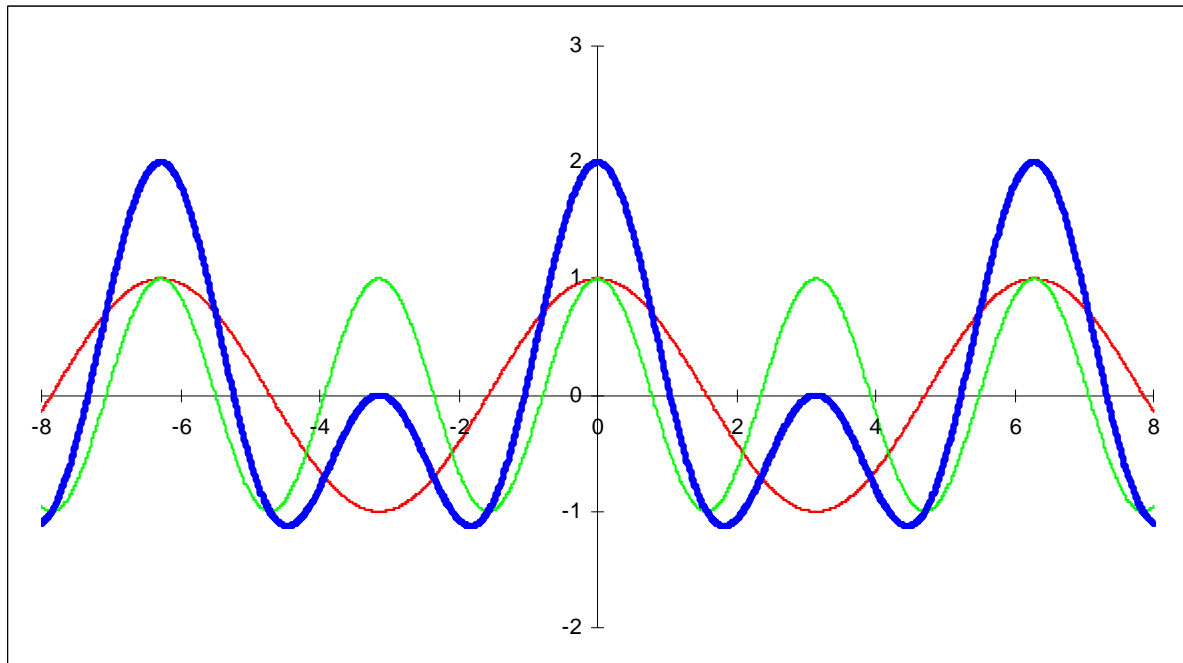


Figure 3.5 – Octaves resonance of two cosine waves

Note also that the average listener may not naturally discern the above signal as two separate pitches*, but rather as the lower frequency signal with a different sound quality or *timbre* to the plain cosine wave.

Secondly, let us define pitches whose frequencies are related by whole powers of 2 (i.e. that are an octave or octaves apart, as in **Figure 3.5**) as being closely related enough as to be *equivalent*, thus F_0 , H_1 and H_3 in the **Table 3.1** are all technically the same note. Since H_2 is the first closest relative which is “different” to F_0 , let us now calculate the first few values of *its* harmonic series, but, for clarity’s sake, starting at an equivalent note which has a frequency closer to F_0 . N.B. By our definition, dividing any frequency by a power of two will yield an equivalent pitch: $786\text{Hz} / 4 = 196.5\text{Hz}$ – the closest possible to 262Hz :

Note in Series	Wavelength	Frequency (Hz)
F_0 (Fundamental Note)	λ	196.500
H_1 (First Harmonic)	$\lambda/2$	393.000
H_2	$\lambda/3$	589.500
H_3	$\lambda/4$	786.000
H_4	$\lambda/5$	982.500

Table 3.2 – Harmonic Series of 196.5Hz

* Listen to a demonstration of this effect in the Sound folder on the project CD. A3 .wav and A4 .wav are two sinusoidal signals an octave apart, and A3A4 .wav is their sum.

Lastly, let us look at the harmonic series of the note, for which 262Hz is the second harmonic, i.e. its other closest relative, which will of course have a frequency of one third of this value: $262\text{Hz} / 3 = 87.3\text{Hz}$

Note in Series	Wavelength	Frequency (Hz)
F ₀ (Fundamental Note)	$\lambda (= 1/87.3)$	87.333
H ₁ (First Harmonic)	$\lambda/2$	174.667
H ₂	$\lambda/3$	262.000
H ₃	$\lambda/4$	349.333
H ₄	$\lambda/5$	436.667

Table 3.3 – Harmonic Series of 87.333Hz

As a final step, let us assemble the above fifteen frequencies from all three tables in ascending order, but normalizing all values so that they appear in the same frequency range. In each case, $F_0 \equiv H_1 \equiv H_3$ – these and other duplicates may be eliminated. We can also, at the same time, label all of the frequencies with letters of the alphabet, starting with ‘A’. As is seemingly true to history, the chosen starting frequency in the note-naming is arbitrary. (In fact the frequency defined as ‘A’ centuries ago is very different to what it is today):

Frequency (Hz)	Equivalent To	Pitch
218.333	H ₄ of 87.3Hz	A
245.625	H ₄ of 196.5Hz	B
262.000	F ₀ of 262.0Hz	C
294.750	H ₂ of 196.5Hz	D
327.500	H ₄ of 262.0Hz	E
349.333	F ₀ of 87.3Hz	F
393.000	F ₀ of 196.5Hz	G
436.667	H ₄ of 87.3Hz	A

Table 3.4 – Closely related pitches in ascending order

The final pitch at the end of the table, which has been included to complete the cycle of eight notes (hence *octave*) is also ‘A’, since it has a frequency which is double that of the starting note and is therefore technically equivalent.

It is perhaps a small wonder that this series of closely related pitches, called a *scale* in musical terms, quite naturally became the basis for the very earliest forms of music and still defines and influences the current twelve-tone *equal-tempered* system used extensively throughout the Western world [Scholes65L]. The word “scale” also comes from the Ancient Greeks: Instead of adding the extra ‘A’ to complete the octave as in the above table, they added a lower G or Γ – Gamma. This gave origin to the Middle English for “scale”, *gamut*, which in turn came from “gamma” + “ut” – the Medieval Latin word for the first note. Any series of ascending or descending pitches is still called a scale today.

3.2 Musical Modes

In order to understand how the transformation from the above naturally occurring scale to the modern system was completed, we need to look more closely at the intervals between each of the notes in **Table 3.4**, which follow a certain pattern known in music as a *mode*.

3.2.1 Just Intonation

As stated previously, the relationship between pitch and frequency is dyadic, and therefore when examining pitch differences or intervals between the notes in **Table 3.4**, we need to look at frequency *ratios* rather than differences. These ratios are as follows:

Pitch	Frequency (Hz)	Interval (Freq. Ratio)
A	218.333	
		1.125 (= 9:8)
B	245.625	
		1.067 (= 16:15)
C	262.000	
		1.125 (= 9:8)
D	294.750	
		1.111 (= 10:9)
E	327.500	
		1.067 (= 16:15)
F	349.333	
		1.125 (= 9:8)
G	393.000	
		1.111 (= 10:9)
A	436.667	

Table 3.5 – Pitch Intervals

It may be observed that smallest differences in frequency are between the notes B & C and E & F, while the other intervals are roughly the same, D – E and G – A being slightly closer than the remaining three. The intervals between these eight related notes, as described by ratios of whole numbers, are known in music as *just intervals*, and the natural tuning system, discovered by the Greeks, is called *just intonation*.

Practically, the ratios 9:8 and 10:9 are close enough as to be indistinguishable to the average ear. At this point, in order to avoid getting too deep into the subject of tuning systems, we shall jump forward several centuries in time to the point where the larger two steps of what is now called a whole *tone*, were regarded as the same interval, and the ratio 16:15 became the defining interval for the smallest possible step between two notes: the *semitone*.

3.2.2 Doh-Ray-Me

Guido d’Arezzo, a Benedictine Monk who was born around the turn of the first millennium, is generally regarded as being the first music theorist to formalize the scale. He considered the naturally occurring pitches as being sacred and pure, giving them holy names from an old Latin hymn written by Paul the Deacon for St. John the Baptist’s day [Scholes65L]:

*“Ut queant laxis
Resonare fibris
Mira gestorum
Famuli tuorum,
Solve polluti,
Labbii reatum,
Sancte Iohannes.”*

Ut was changed to *Do* (the first syllable of *Dominus* meaning “Lord”) fairly early on, since it was more singable. The other names were also changed and anglicized in the 19th century by an English music teacher, named Sarah Glover, to *Doh, Ray, Me, Fah, Soh, Lah* and *Te*. She altered the seventh note from *Si* so that each name could begin with a different letter and thus be shortened to just its initial letter for easy writing purposes: d, r, m, f, s, l and t.

The first six pitches Guido called the *hexachord*, from the Greek words for “six” and “note”. The reason these were particularly special to him (and also his choice of nomenclature) was because it just so happened that each line of the hymn began one note higher than the previous one, and so they formed a scale up to *La*. He thus had a useful educational tool for pitching notes, and is quoted as saying [Scholes65L]:

“If an experienced singer shall so know the opening of each of these sections that he can, without hesitation, begin forthwith any one of them that he pleases, he will easily be able to utter, without absolute correctness, each of these six notes, wherever he may see them.”

The seventh note, *Si*, was a special case for which Guido found he had to have two versions. He wanted to create three hexachords which overlapped with one another, starting on the notes G (the Greek starting note), C and F. Beginning on G, the intervals between each note in the hexachord are:

tone, tone, semitone, tone, tone

This pattern is also the case if C is the starting note. However, if we start on F, we get the following pattern:

tone, tone, tone, semitone, tone

Guido and other music theorists of his time saw this particular arrangement of three tones at the start of the scale as ugly and unholy. They called it “*Diabolus in Musica*” – the “Devil in Music” – and forbade the interval formed by three whole tones, the *tritone*, from being used, especially in sacred music. In order to exorcise this demon from music which centred itself around the F hexachord, Guido lowered the pitch of the B so that it was a semitone above the A and a whole tone above the C, calling it *B molle* – a “soft B” [Grove00L, Scholes65L]. This simple change removed the chance of a tritone occurring and restored the sacred pattern. Since the hexachord beginning on G contained the “hard B”, Guido called it the *Hard Hexachord* or *Hexachordum Durum*. The hexachord on F was then *Hexachordum Molle*, and, because it contained no B at all, the C hexachord he described as *Hexachordum Natrum* – *Natural Hexachord*. The symbols which Guido used to denote a soft or a natural B were rounded and square-shaped letter ‘b’s respectively, called *B rotundum* and *B quadratum* in Latin [Grove00L]. These turned into the modern-day symbols: the *flat*: \flat and the *natural*: \natural .

The flat sign is nowadays used to indicate the lowering of any pitch by one semitone (not just a B) while the latter restores any altered note to its natural pitch.

3.2.3 Gregorian Modes

The simple change made by Guido to the B also incidentally revolutionized the way musicians began to think about the arrangement of tones and semitones. Without realising it, Guido had developed a system known today as a *Moveable Doh*, where a starting note may be chosen arbitrarily from one of the seven natural pitches, but adjustments need to be made to the pitches in order to preserve the natural pattern.

Long before Guido d'Arezzo, however, proponents of early Christian Church music, notably St. Ambrose, the Bishop of Milan (4th C) and Pope Gregory the Great (6th C) had formalized the set of musical modes based on the 7 natural pitches in the Greek scale. The patterns of tones and semitones comprising each mode were defined simply by changing the starting point of the scale. The names of the modes were as follows:

Starting note	Mode
A	<i>Aeolian</i>
B	<i>Locrian</i>
C	<i>Ionian</i>
D	<i>Dorian</i>
E	<i>Phrygian</i>
F	<i>Lydian</i>
G	<i>Mixolydian</i>

Table 3.6 – Modes

Practically all medieval music was written based on these modes, and they are still the foundation upon which the vast majority of Jazz music is built. The Ionian and Mixolydian modes would have been the most interesting to Guido, since they are a combination of his hexachords beginning on C and G. The pattern of the Ionian Mode is as follows:

tone, tone, semitone, tone, tone, tone, semitone

The only difference with the Mixolydian mode is in the last interval:

tone, tone, semitone, tone, tone, semitone, tone

To clarify, for the Ionian Mode, the interval between the last two notes, the (hard) B and the C is a semitone, whereas in the Mixolydian Mode, the last interval is between the F and the G – a tone. The use of the Lydian mode would have been inappropriate to Guido since it begins with three tones:

tone, tone, tone, semitone, tone, tone, semitone

however if the pattern were altered slightly by using a soft or flattened B, this effectively turns the Lydian mode into the Ionian mode, but starting on F instead of C. Similarly, if we raise the seventh note in the pattern starting on G, this would also alter the Mixolydian mode to become Ionian. Raising the pitch of a note by one semitone became known as *sharpening*, and the *sharp* symbol denoting this effect is \sharp . This sign has similar roots to the flat and

natural symbols and is also in the form of a square ‘b’, but one which has been crossed out or cancelled. Its original Latin name was *B cancellatum* – “the cancelled B” [Grove00L].

3.2.4 The Dodecachordon

It was now, in theory, possible to play in any of the seven modes starting on any of the seven notes by sharpening or flattening notes as necessary. Almost a millennium later than Pope Gregory, a new idea was put forward in 1547 (although composers were already using it by then in practice) by a Swiss monk named Henry Glareanus [Scholes65L]. If we count all the semitones in any given mode of a scale, we find that there are twelve. Therefore, in theory, there could be twelve starting points rather than just seven. Glareanus called the set of twelve pitches making up a scale the *dodecachordon*, continuing with the Greek theme (*dodeca* meaning twelve). If we consider all the possible names of the notes, for every semitone, we find that each of the in-between notes (those in the top row in the layout below) have two possible names:

	C#/D \flat	D#/E \flat		F#/G \flat	G#/A \flat	A#/B \flat	
C	D	E	F	G	A	B	C

Table 3.7 – Dodecachordon layout

This arrangement will be recognizable to pianists and other keyboard instrumentalists, since keyboards are still laid out in this way. Technically, the correct naming of the so-called *accidental notes* (possibly from the Latin *nota adventitia* – additional note – as described by Joachim Burmeister [Grove00L]) will always depend on the mode and the starting point. For example, if the starting note is F and the mode is Ionian, the adjusted fourth note will be B \flat . Whereas, if the same mode is used beginning on the note B, then the seventh note should rather be called A \sharp .

3.2.5 Keys and Equal Temperament

Eventually, the modal system of writing broke down, giving way to the idea of *keys*. Since the Ionian and Aeolian modes lent themselves best to harmonization of melodies, these were retained, becoming our current *major* and *minor* modes respectively. The *key* of almost all pieces of Western music is described by assigning Doh to a starting note and then following the patterns prescribed by one of these two modes. C Major and A Minor are simply the old Ionian and Aeolian modes, respectively. These keys contain no accidental notes, however in the case of A Minor, the seventh note is often sharpened (becoming G \sharp) as a kind of strengthening effect.

Back to tuning systems briefly: given that the natural tuning system discovered by the Greeks comprised two slightly different sounding whole tones, it was soon realised that attempting to play in modes or keys which were too far removed from their original intended starting point sounded very strange and out of tune. A new system of tuning was needed which made each semitone in the scale the same interval in order to allow total freedom to composers. The solution, which was most likely also first proposed by the Ancient Greeks and Chinese, is the answer to the question, “If the frequency of a pitch increases by a factor of two at every octave, then by what factor should the frequency of a note be multiplied to move up a semitone?” This may be rephrased more simply as, “What number, when multiplied by itself twelve times, is equal to two?” The answer, of course, is the twelfth root of two.

The system of using this factor to tune every semitone in the scale so that the pitches are all equidistant is called *equal temperament*. Despite the logic behind this method, equal-tempered tuning of instruments only became more popular around the 18th century, after Johann Sebastian Bach wrote his famous set of twenty-four preludes and fugues – a pair of pieces (for piano or organ) for each of the twelve major and twelve minor keys, which he called *The Well-tempered Clavier*. This above all demonstrated the usefulness and flexibility of the tuning system which Bach chose for his own keyboards. It allowed composers to write freely in any key and to modulate freely (that is change key during the piece of music) without having to worry about ugly out-of-tune sounds [Scholes65L].

Bach's *Well-tempered Clavier* was a masterpiece of its time, and the preludes and fugues are still popular today amongst keyboardists. In fact the work lead some to believe that Bach himself had invented the tuning system it demonstrated, and he must have enjoyed this association with equal temperament as he went on to write a second volume!

3.3 Music Notation

A huge amount of information has been written on this subject, which, like any other written language, has developed and grown over many centuries, from its most rudimentary medieval forms.

Although this section of the chapter could fill an entire thesis on its own, it is not the main point of this research, and so only the fundamental rules of music notation have been presented. It is important to realise that the conventions as described here are subject to change – particularly in more modern times, composers have deliberately tended to break away from normal practices, in their wish to leave the interpretation of their compositions much more up to the performer. Since the point here is to extract and present as much musical information as possible from an audio source, such vague representations of music notation must be avoided.

As an additional source, *The Rudiments and Theory of Music*, published by the Associated Board of the Royal Schools of Music [ABRSM38L] has also been referred to. Sir George Grove was the founding director of the Royal College of Music and became somewhat of an authority on music theory. Much of the theory presented in his dictionary [Grove00L] has not changed and has been adopted by many as a standard.

3.3.1 A Note on Music Engraving

Music engraving is the special art of musical typography, in which a copyist follows a special set of rules (which vary in subtle ways from publisher to publisher) which govern the design and layout of printed music [Sibelius09L]. Since it is a skill which is gained primarily in the master-apprentice tradition, not a great deal of literature exists on the subject. Although the main principle of music engraving is to notate music as clearly as possible, it is not essential to the problem at hand. An analogy may be drawn with contents of this thesis: the rules dictating its format, layout, font etc. are to music engraving as its language, tables and diagrams are to music notation. For this reason, the software written for this project does not produce fully rendered printable scores, but rather pitch/time graphs. Besides graphs (see the following chapter) MusicXML [Recordare09W] is another good choice of output, since this increasingly popular and easily written format may then be imported into music publishing packages such as *Sibelius*. The incredibly difficult job of score rendering is best left to the engraving masters.

3.3.2 Representation of Pitch

Musical pitches are indicated by the vertical location of glyphs drawn on a set of five horizontal lines. These lines, known collectively as a *staff*, may be thought of as grid lines on a pitch/time graph. An example of a musical staff with four different pitches drawn on it is shown below in **Figure 3.6**.

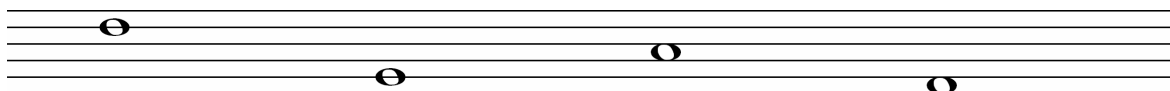


Figure 3.6 – An example of a musical staff

It can be seen that notes are either drawn *on a line* or in between – *on a space*. The third note, for example, is therefore three pitches above the second note and four above the last. Note also that a pitch may be drawn below the bottom line (or above the top line) of the staff. If even lower or higher pitches need to be written, additional short lines, called *leger lines*, may be drawn to accommodate them and provide a measure of their position relative to the staff. **Figure 3.7** shows an example of leger lines drawn below and above a staff.

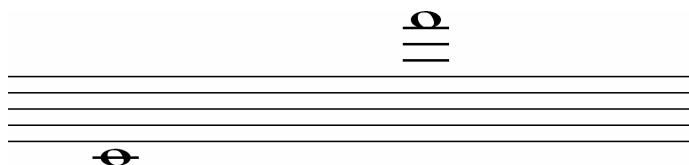


Figure 3.7 – Notes drawn on leger lines

The above staves lack one very important symbol, which effectively acts as a label, indicating the range of the “pitch axis”. This symbol, known as a *clef* (from the French word for *key*) is in fact one of three highly decorative alphabetic characters, namely G, C or F, depicted below:

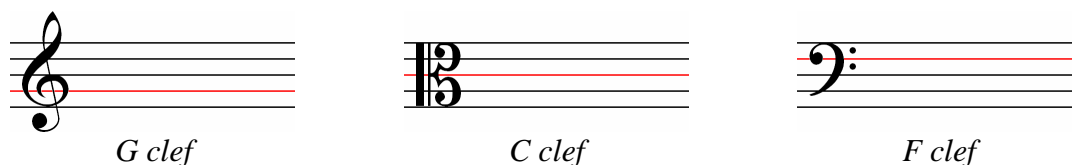


Figure 3.8 – Clef symbols

The lines drawn in red in **Figure 3.8** are the particular pitch levels to which these three different labels refer. They act as base pitch identifiers, from which all other pitches may be measured, based on their relative vertical positions to these lines. Thus, if a G clef were to be drawn on **Figure 3.6**, the first note would be four steps above G, i.e. D. The other pitches would be E, A and another D (an octave below the first). The choice of letters presumably came from Guido d’Arezzo’s three original hexachords which began on G, C and F.

The position of the clefs relative to each other, shown in **Figure 3.9**, reveals that lower pitches should be written using the F clef, while higher pitches fall into the G staff range. The C clef

is used for writing music for middle-range instruments. Thus the G, C and F clefs are also known as the *Treble*, *Alto* and *Bass* clefs, respectively. The note right in the middle of the three staff systems is called *Middle C* (presumably for this very reason), which is the pitch C_4 (262Hz). The small 4 tells us which octave, on some grand scale (from C_j to C_{j+1} by convention) the pitch lies. The note C_0 is then the master base pitch, to which is normally assigned the frequency 16.375Hz. This is already close to being inaudible, although some music theorists prefer to use C_{-1} as the lowest conceivable pitch. This is not very logical, however, since at a frequency of around 8Hz this note cannot be heard by humans.

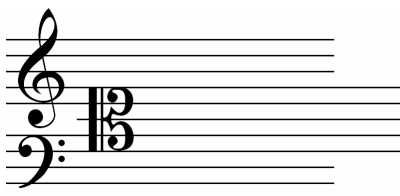


Figure 3.9 – Relative vertical positions of staff systems

The full range of pitches which may be represented, from the lowest line on the bass clef to the space just above the treble staff, is three octaves. Most of the time, musicians use only one of these clefs at a time, but may change between different staff systems (by specifying a different clef symbol) in the same line of music, sometimes several times in one piece, to save from writing too many leger lines.

The clef symbols may also be moved to different lines on the staff, depending on the particular range of the instrument for which the music is being written. For example, the tenor trombone – a middle to low range instrument – reads from a staff written using the Tenor clef, shown in **Figure 3.10**, which is the C clef shifted to the fourth line. Other positions of the C clef are less common, and the G and F clef are very rarely shifted, however in theory they may be placed on any of the five lines in the staff. The particular pitches which the F, C and G clefs always define, regardless of their position, are F_3 , C_4 and G_4 respectively.

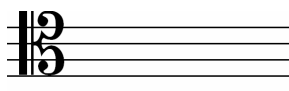


Figure 3.10 – The Tenor clef

3.3.3 Representation of Time

Notes are read from a staff and played in order from left to right. However, the horizontal spacing between notes gives only a very rough indication of their durations. Instead, precise note lengths are represented by different note symbols, the longest in current use being the semibreve or whole note, which is the type of all the notes drawn in **Figures 3.6 & 3.7**. This idea of having note durations determined by their appearance rather than by their position or context was first formalized by French music theorist, Franco of Cologne in the 13th century [Scholes65L]. *Franconian notation*, described in Franco's treatise, *Ars Cantus Mensurabilis* (The Art of Measurable Music), is still the standard used today for indicating note lengths.

The various note symbols and their equivalent *rests* (silences lasting the same duration as their counterpart notes) are shown in **Table 3.8**. As can be deduced from their American names, each note value in the table decreases in duration from its predecessor by a factor of two. So, for example, if the constant speed of a piece of music is set so that a whole note lasts for 4 seconds, then all quarter notes will last for 1 second, eighth notes will be half a second, sixteenth notes will have a duration of 0.25 seconds and so on. Theoretically, note values may be halved further by adding more hooks, although anything beyond the demisemi-quaver is rare. A note with four hooks is a *hemidemisemi-quaver* or sixty-fourth note, and with five it is a *semihemidemisemi-quaver* or *quasihemidemisemi-quaver** (hundred twenty-eighth note).













Note symbol	Rest symbol	English name	American name
		<i>Semibreve</i>	<i>Whole note</i>
		<i>Minim</i>	<i>Half note</i>
		<i>Crotchet</i>	<i>Quarter note</i>
		<i>Quaver</i>	<i>Eighth note</i>
		<i>Semiquaver</i>	<i>Sixteenth note</i>
		<i>Demisemi-quaver</i>	<i>Thirty-second note</i>

Table 3.8 – Note types

Other indications of time are the *dot* and the *tie*. The latter symbol is a curved line which effectively joins two notes together. The tied notes then last for the duration of their sum. More than two notes may be tied together in this way as long as they are the same pitch.

A dot after a note means that the note is to be sustained an additional length of time equal to half of its value. For example, a dotted crotchet will be held for a crotchet plus a quaver; a dotted whole note will be held for a whole note plus a half note, and so on. Examples of dots and ties are shown in **Figure 3.11** below. A flat, sharp and natural have also been drawn against the notes to demonstrate their positioning. A tied note assumes the same accidental as the first in the group, therefore the first three notes are all B \flat . Because the third G (sixth note) is not tied to the previous two, in order to cancel the effect of the sharp, a natural is drawn.



Figure 3.11 – Ties and dots example

* An example of these notes may be found in the 1st movement of Beethoven's *Pathétique* Piano Sonata (Op. 13)







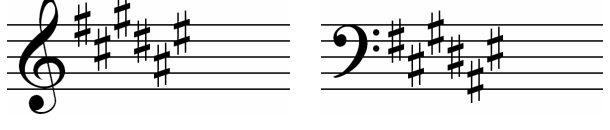






Major	Minor	Number of # or ♭	Key Signature
C	A	None	
G	E	1 sharp: F#	
D	B	2 sharps: F#, C#	
A	F#	3 sharps: F#, C#, G#	
E	C#	4 sharps: F#, C#, G#, D#	
B	G#	5 sharps: F#, C#, G#, D#, A#	
F# or Gb	D# or Eb	6 sharps: F#, C#, G#, D#, A#, E# 6 flats: Bb, Eb, Ab, Db, Gb, Cb	 
Db	Bb	5 flats: Bb, Eb, Ab, Db, Gb	
Ab	Fb	4 flats: Bb, Eb, Ab, Db	
Eb	C	3 flats: Bb, Eb, Ab	
Bb	G	2 flats: Bb, Eb	
F	D	1 flat: Bb	

Table 3.9 – Keys and key signatures

As a quick check, in the first bar we have a dotted half note plus a quarter note, which equals four quarter notes. In the second bar, we have a sixteenth note plus a sixteenth rest, followed by an eighth tied to a dotted sixteenth, then a thirty-second, and lastly an eighth tied to a half note. Adding up all these fractions, we get four quarters again.

The time signature itself looks a little bit like a vulgar fraction, but without the dividing line. The top number tells us how many beats there are in every bar, and the bottom number is the beat denominator, i.e. it describes what type of note the beat is. Thus, using the same example, **Figure 3.13** shows the complete picture, including the time signature. Unlike a key signature, the time signature is only written once at the beginning of a section of music and not on every line.



Figure 3.13 – Previous example staff with bar lines and time signature

The final point to make about time signatures is that the beat type could also be a dotted note. Such time signatures are called *compound*, since the numerator (top number) must be subdivided by three to get the actual number of beats. The time signature 6 over 8, for example, means that there are six quavers in every bar, but that these should be subdivided into 2 dotted crotchet beats – each beat containing three quavers. The rule of thumb is if the top number of a time signature is divisible by three then it is *compound*, otherwise it is *simple*.

3.4 Diatonic Intervals and Chords

With some basic music theory under our belt, we can now begin to write some elementary harmony constructs, the most important of which is the *chord*. This is simply two or more notes played simultaneously. On a staff, chords are depicted by drawing pitches above one another.

3.4.1 Dyads

Not unrelated to the term “dyadic”, a *dyad* is the most basic type of musical chord. It comprises just two notes and is usually described by the interval between them. The interval of the perfect fifth was mentioned in the previous section. Each of the twelve intervals between the twelve possible pairings of notes within an octave has a similar name. The other two “perfect” intervals are the fourth and the octave. Note that the latter is the interval between any fundamental note and its first harmonic, while the *perfect fourth* is the interval defined by the second and third harmonic. Since the word “octave” is a unique description of that interval, the qualifier, “perfect”, is usually omitted.

Whereas the interval C – F is a perfect fourth, the interval F – B is not, since it comprises six semitones and not five. F – B is thus called an *augmented fourth* because of this extra semitone. Similarly, the interval B – F contains one less semitone than C – G, and so it is called a *diminished fifth* rather than a perfect fifth. Note that the diminished fifth and the augmented fourth in fact have the same number of semitones, however the interval must always be described in terms of the gap between the letter names of the notes, and so technically they are different. The interval from a note to itself (i.e. no interval) is called a

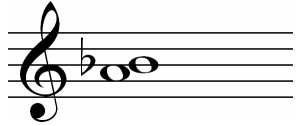
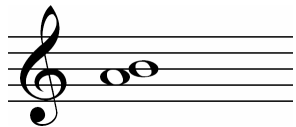
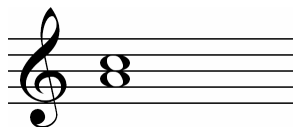
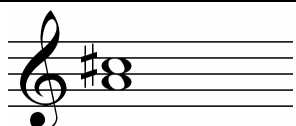
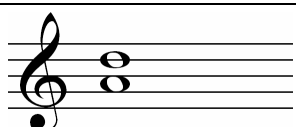
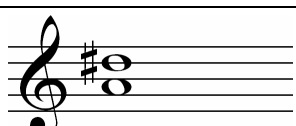
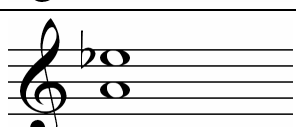
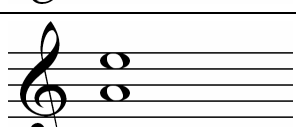
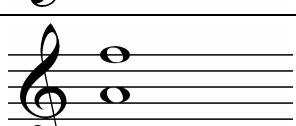
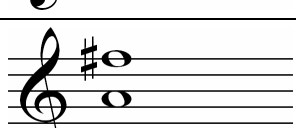
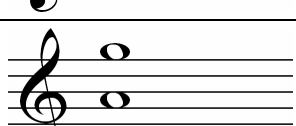
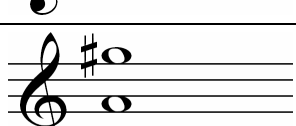

Interval	No. of semitones	Name	Staff notation
A ₄ – B ₄ ^b	1	<i>Minor Second</i>	
A ₄ – B ₄	2	<i>Major Second</i>	
A ₄ – C ₅	3	<i>Minor Third</i>	
A ₄ – C ₅ [#]	4	<i>Major Third</i>	
A ₄ – D ₅	5	<i>Perfect Fourth</i>	
A ₄ – D ₅ [#]	6	<i>Augmented Fourth</i>	
A ₄ – E ₅ ^b	6	<i>Diminished Fifth</i>	
A ₄ – E ₅	7	<i>Perfect Fifth</i>	
A ₄ – F ₅	8	<i>Minor Sixth</i>	
A ₄ – F ₅ [#]	9	<i>Major Sixth</i>	
A ₄ – G ₅	10	<i>Minor Seventh</i>	
A ₄ – G ₅ [#]	11	<i>Major Seventh</i>	
A ₄ – A ₅	12	<i>(Perfect) Octave</i>	

Table 3.10 – Names of Intervals

unison (rather than a “first”). Two instruments playing *in unison* are therefore playing exactly the same notes at the same time.

All other intervals are described as being either major or minor. The former name is used for wider intervals, while the latter is for closer intervals. For example, the interval from A to C is a third, as is C – E. However C – E is four semitones while A – C is only three. Therefore A – C is a *minor third* and C – E is a *major third*. **Table 3.10** on the previous page summarizes all possible intervals between the note A and any other note within an octave. A note which forms a basis for measuring intervals, such as the A used below, is known in music as the *tonic*. It is also therefore the base or *root* of a scale built on top of it: *Ut* or *Doh*.

It is important to stress the strict usage of, for example, C# instead of D♭ and G# instead of A♭. While these pitches are to all intents and purposes the same, the intervals theoretically are not. A – A♭, for example, would have to be called a diminished octave rather than a major seventh, since it is the interval from one type of A to another. Since in music theory there is no such thing as a diminished octave, this interval would not technically be correctly spelt.

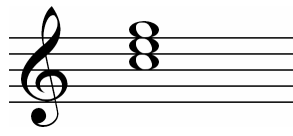
The word *diatonic* is used to describe pitches and the intervals they form with a tonic note that belong to a particular mode or scale. All other pitches which create intervals with the tonic that cannot be described by the above twelve are called *chromatic* intervals, which comes from the Greek, *chromos*, meaning “colour”. For example, G♭ is a chromatic note in this context, since it should be called F# in order to form a major sixth with the tonic note, A. However, if E♭ were the tonic instead, G♭ would be diatonic, since the interval would be a minor third. F# would form an augmented second, which is not a diatonic interval.

Some music theorists argue that the diminished fifth and augmented fourth should not be included in the diatonic intervals, saying that the word implies strictly major or minor intervals related to the tonic. The troublesome tritone is, nevertheless, one of the naturally occurring intervals and is (at least since Guido’s time) a very important construct which frequently crops up in music, and so it definitely has its place in the above table.

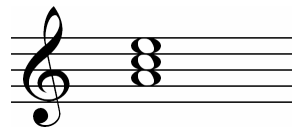
3.4.2 Triads

As its name suggests, a triad is formed from three notes. It may also be thought of as two dyads sharing one note in common. Triads are also usually formed by superposing diatonic intervals, although chromaticism became much more widely used in Romantic music (from the early 19th century) and by the 20th century, this conformity was generally ignored. It is useful to be aware of the most common triad formations which occur in music, since this knowledge allows us to predict more accurately which solution is more likely whenever ambiguities arise in pitch detection. It also means that we can spell the chords and, if we have enough information, determine the key of the music being analysed.

The most frequently occurring triads in music are the two which define the major or minor quality of the key. These are the *major tonic triad* and the *minor tonic triad*, which are notated in **Figure 3.14**. The keys of C Major and A Minor have been chosen for clarity’s sake, since they contain no sharps or flats to cloud the issue. The chords are called *tonic triads* as they are built on top of the tonic note of these keys. Examining the dyads used to build up the two chords, C Major comprises a minor third stacked on top of a major third, while the A Minor triad has the minor third at the bottom and the major third at the top. This is precisely why the terms major and minor are applied to the names of keys – they describe the size and hence the quality of the third above the tonic.



C Major tonic triad



A Minor tonic triad

Figure 3.14 – Tonic triads of C Major and A Minor

Aurally, the two triads are, for most people, easy to distinguish. For some psychological and/or physiological reason, most find that the major triad sounds happy while the minor triad sounds sad. This may have something to do with the harmonic series: the notes in the C Major triad are actually the third, fourth and fifth harmonic of the C two octaves below. To our pattern seeking brains, perhaps these sounds blend better (and make us happy?) They may also be causing us to imagine the absent lower C – this is sometimes called the *ghost fundamental*. For the minor triad, a C is to be found much higher up the harmonic series from the A, and even then it is slightly out of tune. In just intonation, the frequency ratio between a tonic note and its minor third is 5:6. The major third ratio is 4:5, and so for the major triad, the three ratios combined are 4:5:6. For the minor triad, however, one has to find higher common multiples – 10:12:15. It may be argued that this arrangement is more complex and therefore not quite so pleasing to our brains (and makes us sad?!)

Other common triads may be built on top of each of the notes in C Major and A Minor, as in **Figures 3.15** and **3.16** respectively. In the A Minor scale, the sharpened seventh note (G#) has been used, since this is more common in music harmony writing. This form of the mode is known as the *harmonic minor*.

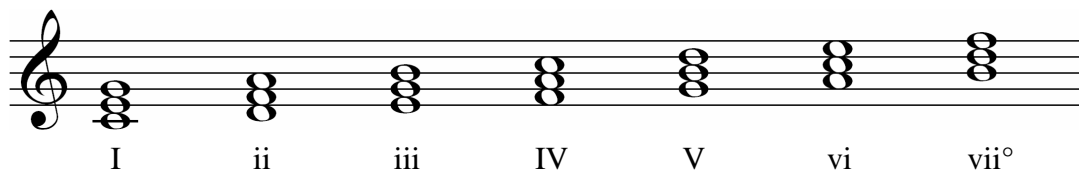


Figure 3.15 – Diatonic triads of C Major

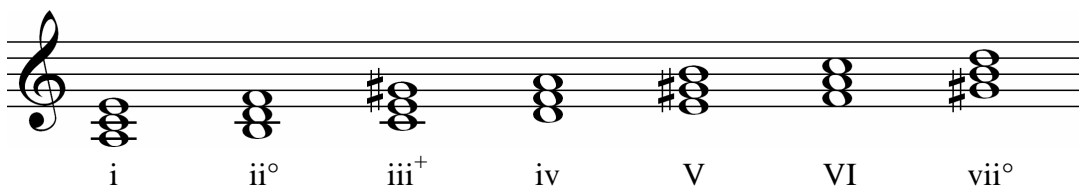


Figure 3.16 – Diatonic triads of A Harmonic Minor

Most of these triads have either a major or minor quality. This has been indicated by using different case Roman numerals underneath the chords. The numeral itself is just the degree of the scale, but upper case means the chord is major and lower case denotes minor. The other two possible combinations of two stacked thirds are denoted by a ° for two superposed minor thirds and a + for a pair of major thirds. The circle assigns the quality *diminished* to the chord, and the plus means the chord is *augmented*. Note that the interval from a C to a G# is actually an augmented fifth and in fact the raised seventh creates more intervals than those in **Table**

3.10. Since these intervals rarely need to be described, most music theorists do not normally include them amongst the regular diatonic intervals, even though technically they are diatonic to the harmonic minor mode.

In addition to the Roman numerals used to index the notes in the diatonic scale, they are also given the following names:

Degree	Name
1	Tonic
2	Supertonic
3	Mediant
4	Subdominant
5	Dominant
6	Submediant
7	Leading Note

Table 3.11 – Degrees of the scale

Note that we have only been looking at triads which are contained within an octave. Of course chords may be formed using wider intervals, however employing our original assumption that notes an octave apart are identical (see subsection **3.1.3**) this means that we can always respell a triad using close intervals.

Similarly, it is not necessary to examine chords built with a combination of fourths and thirds. The reason for this becomes evident when the triad is stacked differently, with a note other than the tonic on the bottom. Since there are three notes in the the triad, there are three ways of arranging them. The different configurations, shown in **Figure 3.17** for the tonic triad of C Major, are called *inversions*. The arrangement with the tonic on the bottom is known as *root position*.

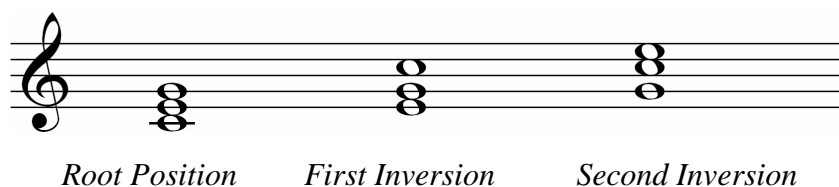


Figure 3.17 – Inversions of C Major tonic triad

As can be seen, the first inversion is a fourth on top of a third, and the second inversion is a third on top of a fourth. Depending on the mode and the degree of the scale on which the chord is built, the fourths may be perfect, diminished or augmented. Again, technically we seldom talk about diminished fourths but we still need to know about them in order to spell chords correctly after their component pitches have been identified. Taken out of context, diminished fourths sound identical to major thirds, just as augmented fifths sound the same as minor sixths.

In order to indicate the inversion of a chord, figures are added to the Roman numerals used in **Figure 3.15** and **3.16** which specify the interval between the bottom note and the other two. The chords in **Figure 3.17** would be described as follows:

- Root position: I_3^5
- First inversion: I_3^6
- Second inversion: I_4^6

In practice, the figures for root position are always left out – a Roman numeral with no figures is therefore always assumed to be in root position. Similarly, the subscript 3 for the first inversion is omitted and so this chord is fully described by just I^6 . Both numbers are always written for the second inversion to distinguish it from the first.

Triads created from seconds and fourths are not as common as the above set, but they do occur fairly frequently and are worth looking at. Again, inverting a chord comprising only fourths yields a second and a fourth with no further possible configurations:

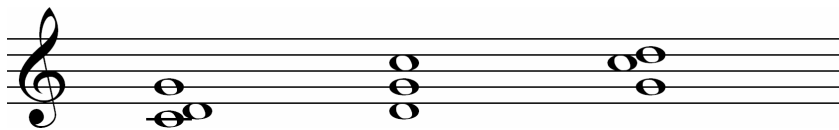


Figure 3.18 – Triads comprising fourths and seconds

The usual function of these types of chords in music is to create a suspense or tension for which our minds demand some kind of resolution. If the composer wishes to grant relief to the tension, the chord will *resolve* onto one of the more “comfortable” diatonic triads. Creating moments of tension and resolution in music is one of the most important skills that a composer can master. The close interval of the second is the reason for the tension – the difference in frequency between these notes is small so their cycles move in and out of phase less rapidly, i.e. they do not resonate as well as, say, frequencies in fifths and octaves.

Although they are common in Classical music, there is no symbol for this type of chord in the theoretical literature. However, in Jazz and on guitar music, they are labelled “sus”, short for *suspension*. **Figure 3.19** shows how the same suspension may resolve in two different ways.



Suspension resolving up to C Maj., 1st Inv. Suspension resolving down to G Maj., 2nd Inv.

Figure 3.19 – Suspensions and resolutions

Least common in music, but still entirely possible are *cluster chords* which comprise close intervals, no bigger than a major second or whole tone. Three is the minimum number of notes in a cluster and usually there are more. Obviously inverting them turns them into other chromatic chords which can no longer be called clusters. These types of chords (and their inversions) occur in music of the late 19th century / early 20th century onwards, when musicians began to break away from the rules of harmony and experiment with new and different ideas.

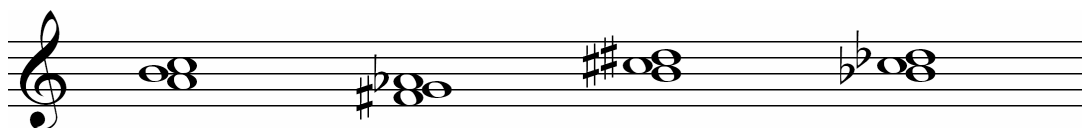


Figure 3.20 – Examples of cluster chords

3.4.3 Quartads

Adding one more note into the mix, we get the four-note chord or *quartad*. More often than not, for diatonic chords, quartads are simply triads with one of the notes repeated or *doubled* at the octave. **Figure 3.21** shows some of the chords from **Figure 3.15** and **3.16** with different doubled notes.

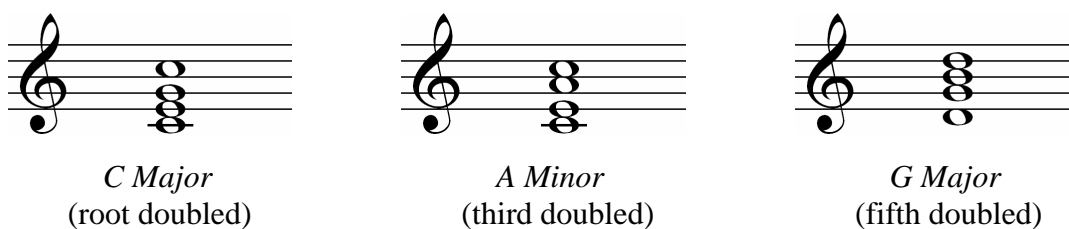


Figure 3.21 – Quartads

Quartads may also be inverted, as can be seen in **Figure 3.21**. The inversion name of a quartad is the same as the inversion of the lower triad. For example, the C Major chord is in root position, while the G Major chord is in its second inversion.

As with triads, quartads may be spread out so as to cover a wider range, however, when describing and labelling them, they are always imagined in close position above the lowest note. Inversions of quartads are labelled using the same Roman numerals and figures as for triads, i.e. the intervals between the lowest note and the two notes above it (in close position) are specified. **Figure 3.22** shows another version of the C Major quartad from **Figure 3.21** with wider intervals between the notes. It is still made up of two Cs, an E and a G, but it is spread out over three octaves and drawn over two staves. We know that this chord is still in root position because the bottom note (the bass) is still the tonic, C:

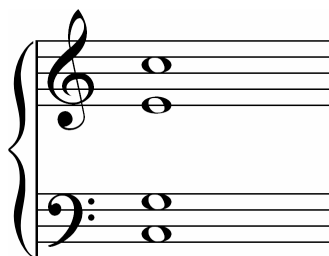


Figure 3.22 – A wide quartad

There are many other different types of quartads which comprise four different pitches, i.e. with no doubled notes. Of these, the most common are the *sevenths* which describes the interval between the two outside notes. Without going into their musical function – an enormous topic on its own – the most common sevenths are constructed by stacking three

thirds in different configurations. Each type of seventh quartad with its name and its component thirds is shown in **Table 3.12**.

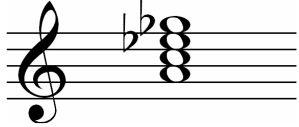
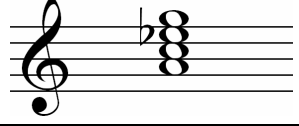
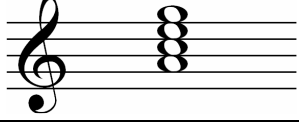
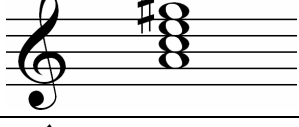
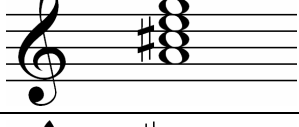
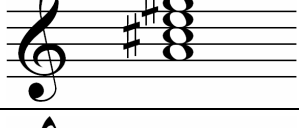
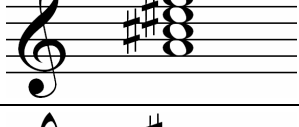
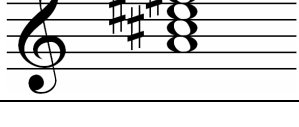
Combination of thirds	Name	Notation
minor, minor, minor	<i>Diminished seventh</i>	
minor, minor, major	<i>Half-diminished seventh</i>	
minor, major, minor	<i>Minor seventh</i>	
minor, major, major	<i>Minor-major seventh</i>	
major, minor, minor	<i>Dominant seventh</i>	
major, minor, major	<i>Major seventh</i>	
major, major, diminished	<i>Augmented minor seventh</i>	
major, major, minor	<i>Augmented major seventh</i>	

Table 3.12 – Seventh Chords

Although the penultimate chord seems to be the odd one out, with its diminished third on the top, the interval between the bottom and top note is still a type of seventh. Note that the combination of three major thirds is not possible since the interval formed by the two outer notes would sound like an octave but technically would have to be called an augmented seventh, which does not exist in music theory as a valid interval description. Note also that the word “augmented” in the descriptions “augmented minor seventh” and “augmented major seventh” refers to the augmented *fifth* within and not to the seventh, which is always diminished, minor or major.

A seventh chord in its four possible inversions requires different figures for its Roman numeral description, which also describe the intervals between the bottom note and two of the other notes (except for root position, where just the superscript ‘7’ is written). The dominant seventh of C Major, which has G as its root, has been labelled in all its inversions in **Figure 3.23**. The dominant seventh may be thought of as the dominant chord, V, in any key, with an added seventh above the root (hence its name).

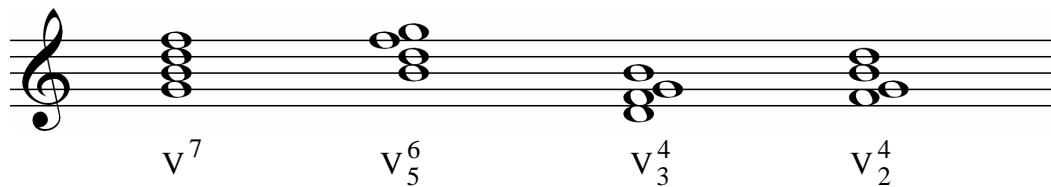


Figure 3.23 – C Major, dominant seventh in different inversions

3.5 Chord Progressions

The final section of this chapter takes a brief look at commonly occurring sequences of chords found in almost all types of music. Special attention must be paid to these sequences for the reasons mentioned in the introductory chapter – the more clues and *a priori* knowledge we have about the music we are analysing, the easier it is to predict the likelihood of a certain solution being correct over others.

3.5.1 Cadences

A *cadence* is a sequence of two chords which commonly occurs at the ends of phrases in a piece of music. Phrases can be thought of as musical sentences and so cadences are analogous to punctuation. There are four main types of cadence to be found. These are:

- Perfect
- Plagal
- Imperfect
- Interrupted

A *perfect cadence* is the strongest type, which gives a phrase of music a very definite sense of coming to an end. Continuing the grammar metaphor, perfect cadences are closest to being full stops and commonly mark the ends of musical paragraphs. The two chords comprising a perfect cadence are the dominant followed by the tonic in root position, i.e. chord V going to chord I (or i in a minor key). Occasionally chord V will be in its first inversion (but hardly ever in its second inversion) which has the effect of weakening the cadence slightly.

Examples of the perfect cadence are shown in **Figure 3.24** in the keys of C Major and A Minor. For minor keys, as in the example, the leading note is always sharpened, i.e. the harmonic minor is always used. Also in the minor example, a dominant seventh has been written instead of the regular chord V. This usage occurs frequently in music as it has the effect of further strengthening the cadence. The *double barline* is drawn to indicate the end of a section in a piece of music, and so it has been used here after each cadence.

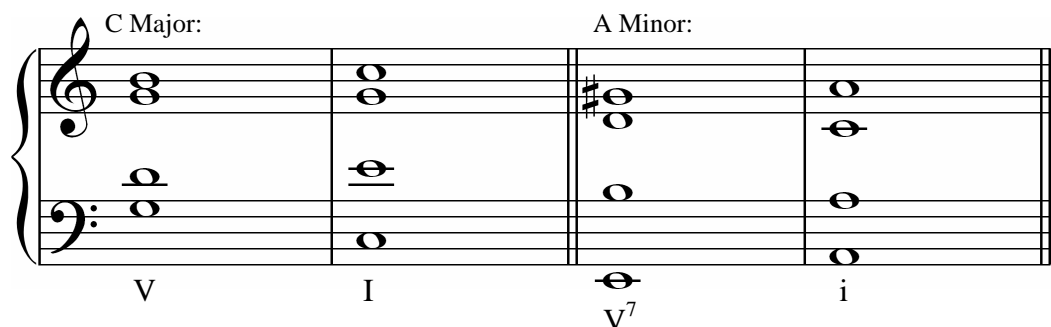


Figure 3.24 – Perfect cadences in C Major and A Minor

Plagal cadences may also occur at the ends of musical paragraphs and final endings, however they are considered weaker than perfect cadences. The most recognizable use of the plagal cadence in Western music is the “Amen” sometimes added to the ends of hymns. A plagal cadence is formed by chord IV followed by chord I (or iv followed by i in a minor key).

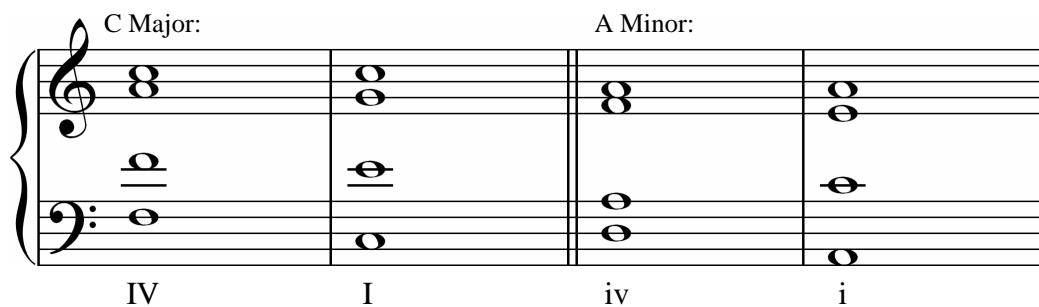


Figure 3.25 – Plagal cadences in C Major and A Minor

Sometimes in a plagal cadence, the minor iv is used in a major key. This technique is known as *borrowing* and is similar to the idea of using the major chord V in the harmonic minor mode in order to strengthen perfect cadences. In this case, however, it has the effect of further weakening the cadence. In the C Major example above, the alto A becomes A^b, which is a closer step to the next note, G, making the transition between the two chords more subtle.

The *imperfect cadence* is like a musical semicolon; it always appears in between phrases and never at endings, unless the intention is to have the music sound unfinished. It is the reverse of the perfect cadence: chord I is followed by chord V (and i is followed by V in minor keys):

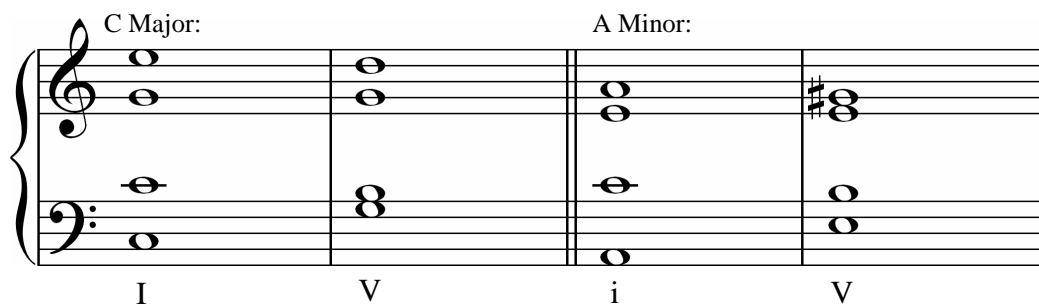


Figure 3.26 – Imperfect cadences in C Major and A Minor

Note that if instead it were assumed the second chord is the tonic, the imperfect cadence becomes the plagal cadence, since the two are identical aurally. This sort of context-dependency for cadences is a further useful clue for harmonic analysis when trying to determine the key of a piece. The second inversion of chord V is very common at Imperfect cadences. When it is used in this instance, it is known as a *cadential six-four*. The numbers naturally refer to the fingering of the chord in its description, which is V_4^6 .

Finally, the *interrupted cadence* is so-called because it sounds like the interruption of what could be a perfect cadence. It also begins with chord V or V^7 but instead moves to chord vi (or chord VI in a minor key) again, giving a feeling of incompleteness, since the music has not arrived at the comfortable tonic. Note the notation of a unison in the final bar.

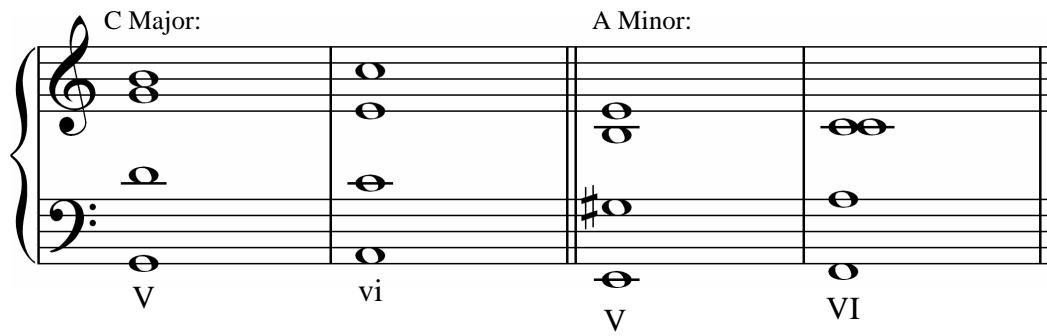


Figure 3.27 – Interrupted cadences in C Major and A Minor

3.5.2 Some Basic Voice Leading Rules

Voice leading is the motion of a note within a melody or chord to its the next note. There are three possibilities: the pitch may rise, fall or stay the same. Although the following rules about voice leading are often broken, they are more often the case than not. The word “voice” is used to describe one particular line of music within a polyphonic structure. Think of a human voice within a choir singing a particular harmonic line. More often than not, there are four voice parts in a choir. These parts are named, from lowest to highest, *bass*, *tenor*, *alto* and *soprano*.

3.5.2.1 Parallel, Contrary and Oblique Motion

The three ways in which one voice may move give rise to three ways in which a pair of voices may move in relation to each other. These three types of relative motion are illustrated in **Figure 3.28**:

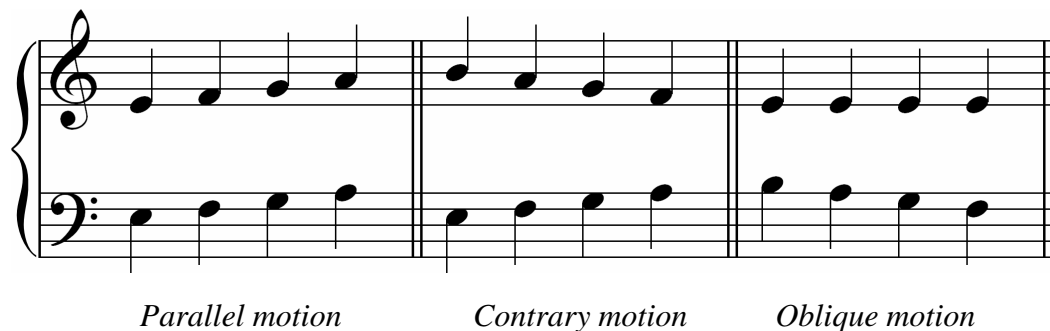


Figure 3.28 – Three types of relative voice motion

There are some rules regarding the usage of parallel motion (the first type in the figure), which tend to be obeyed more often than not in polyphonic music, unless a specific effect is desired. In this type of motion, voices move in the same direction by the same step. Since the interval of an octave and a perfect fifth are the intervals between the first three pitches of the harmonic series of a note, it is not desirable to have two voices moving in parallel at these intervals. Referring back to **Figure 3.5**, two pitches an octave apart resonate well with each other, as do perfect fifths, and aurally the effect is that one voice gets absorbed by the other. Instead of two different pitches, sometimes only one is heard with a different sound quality or timbre. Therefore moving in parallel octaves or fifths effectively loses a harmonic line.

The study of J. S. Bach's three and four part harmony writing in particular reveals that he and other composers were very aware of the undesirable effects of writing parallel fifths and octaves, and they tended to avoid doing so as much as possible in their music. Only rarely will one find a counter-example of this rule, especially in writing where all four voices have equal importance.

An important case where writing parallel octaves is used as a deliberate technique is for bass instruments in an orchestra or band. Very often the bass line will double the roots of chords an octave below, which has the effect of boosting the entire harmonic series of these pitches, resonating with other instruments and yielding a much richer sound. In an orchestra, the lowest string instrument is called the *double* bass for this reason. Removing double basses and bass guitars from orchestras and bands has a very noticeable effect – the sound texture immediately becomes much thinner and less full-bodied.

The most common use of parallel voice-leading is motion in thirds or sixths, and occasionally fourths. Motion in parallel seconds and sevenths has an interesting effect and occurs more frequently in 20th century music and Jazz. The other two types of motion, contrary, where voices move in opposite directions, and oblique, where one voice stays on the same pitch and the other moves against it, are less restrictive in their usage. Both occur frequently in all types of music.

3.5.2.2 Doubling of Notes in Quartads

As with parallel motion, the rules regarding the doubling of certain notes within a chord rather than others are not set in stone, but certain patterns do occur much more frequently in general, and so it is useful to be aware of them.

More often than not, in a major chord, the root note (i.e. the bottom note of a chord in its root position) is doubled. Less frequently, the fifth is doubled, although usually only to avoid writing parallel octaves and/or fifths, and more when a chord is in its first inversion. The third is seldom doubled, unless the chord is minor, in which case it is quite a likely candidate. **Figure 3.29** shows a chord progression where different doubling has been used for each chord. Note that since each of the four notes of the final dominant seventh is unique, there is no doubling in this chord. Note also that the motion between the soprano (top voice) and tenor (second from bottom) would be considered bad writing by Bach, since parallel octaves are formed between the second and third chord, as indicated by the parallel lines. By convention, when two parts are written per staff, such as in the example below, the stems of the upper voice notes should always point upwards and lower voice stems should point downwards, regardless of whether or not the pitches are above or below the middle line.

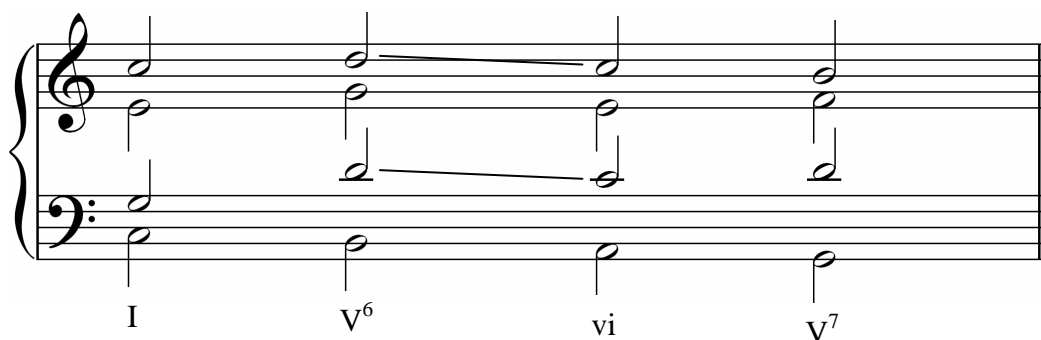


Figure 3.29 – Doubling of notes in four-part harmony

The doubled notes in the above chords are as follows:

- I – Root doubled – bass & soprano C
- V⁶ – Fifth doubled – tenor & soprano D
- vi – Third doubled – tenor & soprano C
- V⁷ – Four unique notes (no doubling)

3.5.2.3 More miscellaneous rules

Finally, the following should be observed when writing for voices within chords forming cadences especially and elsewhere in general. Again, these rules are not always the case, but are much more likely to occur in music than not:

- Voices should not cross over one another or move too far into the range of a neighbouring voice part. That is to say, if one were to draw contour lines through all the notes belonging to two neighbouring voices, these lines should not cross or exhibit too much deviation.
- Intervals between bass and tenor voices may be over an octave, but should be less than an octave between other pairs of neighbouring voices.
- In chord progressions, notes which are shared by two different chords should stay in the same voice. For example in a plagal cadence in C Major, both chord IV and chord I have a C in them. Accordingly, in **Figure 3.25**, the C in the first cadence stays in the soprano part.
- In a perfect cadence, the leading note – the seventh degree of the scale (e.g. a B in C Major) should rise to the tonic (e.g. C in C Major). It may also fall to the fifth degree of the scale, i.e. the dominant (G in C Major).
- If the first chord of a perfect cadence is a dominant seventh, the seventh of chord V⁷ should fall to the third of chord I (e.g. F should fall to E in C Major).

While there are several other rules of harmony, those outlined in the above sections are more than enough to start with and they cover the basics for an already extremely wide range of musical styles. As with good English grammar, the rules do not exist because somebody has decided that they should be law, but rather because when music is written in adherence to them, it simply sounds better, and so therefore they have survived the test of time. While an attempt has been made to understand exactly why this should be so, according some scientific, physical or physiological reason (such as resonance and the harmonic series) it can perhaps never be known for sure why our minds tend to favour one set of patterns over another. There is in fact a whole branch of scientific study dedicated to exploring human perception of sounds and music, and how sound affects us psychologically. This subject is known as *psychoacoustics*. Although it is perhaps extremely important in this kind of research, further discussion of this field cannot be accommodated by the scope of this already bulky chapter, and so hopefully this brief mention of it will suffice.

4 The Easy Problem – Single Pitch Extraction

Perhaps the word “easy” isn’t quite right here, since the problem of determining f_0 for even single musical pitches and hence transcribing melodies is no mean feat, and a fast and accurate method has eluded many great thinkers for many years. Nevertheless, the complexity of this problem pales in comparison to that of multiple pitch and multiple instrument recognition, which will be dealt with in the following chapters. The current favourite method of single pitch extraction (at least in the author’s opinion) will be discussed in the third subsection of this chapter, and some examples of this highly successful algorithm working on live recorded music will be shown. Credit to this algorithm goes to Philip McLeod who has implemented it in his real-time music analysis tool, *Tartini* [Tartini07S].

4.1 Windowing in the Time Domain

The Fourier transform is able to provide accurate information about the frequency content of signal. However, there is no way of telling *when* those frequencies occur, since the underlying assumption of the transform is that all frequencies making up the signal are stationary and extend infinitely over time [Gabor46L], or at least in the discrete case, for the entire duration of the signal. Therefore on its own it is not much use for melody extraction, unless the melody consists of just one continuous note!

4.1.1 The Short Time Fourier Transform

The first attempt to get around this particular problem seems a logical one: split the audio signal into small segments (or windows) and identify the frequency content with separate Fourier transforms in each. This idea was first alluded to by Dennis Gabor in 1946 in his paper *Theory of Communication* [Gabor46L], although the link between the *Gabor transform* and the more modern name for this technique, the *Short Time Fourier Transform* (STFT) was only discovered much later in 1980 by Dutch engineer, Martin Bastiaans [Bastiaans80L].

The discrete STFT, $F_{k,m}$, is a two dimensional sampled function indexed by m over frequency, and k over a time shift. kW is then the k^{th} position of a sliding window of width W along the signal. So, we have:

$$F_{k,m} = \sum_{n=0}^{N-1} f_n \cdot e^{-i2\pi mn/W} \cdot w_{n-kW}.$$

As can be seen, $F_{k,m}$ is the DFT multiplied by another discrete function, w , known as the window function, whose starting time is shifted from the signal’s starting time by the duration of k window widths. Before discussing window functions, however, we need to know their necessity due to the following problems which arise from cutting up a signal in this way.

4.1.2 Time Slicing Issues

As discussed previously, the Fourier transform of a signal becomes less meaningful for non-stationary waves, that is waves which do not comprise repeating patterns and therefore do not have any discernable frequencies.

Referring to **Figure 2.6** in chapter two once again, the discrete Fourier transform of this stationary wave, sampled over 2 seconds with high resolution time spacings of $1/8192$ seconds, is shown in **Figure 4.1**, still represented as a bar graph. As expected, we get three spikes at 5Hz, 10Hz and 20Hz (the symmetrical spikes in the negative domain of the graph have been omitted).

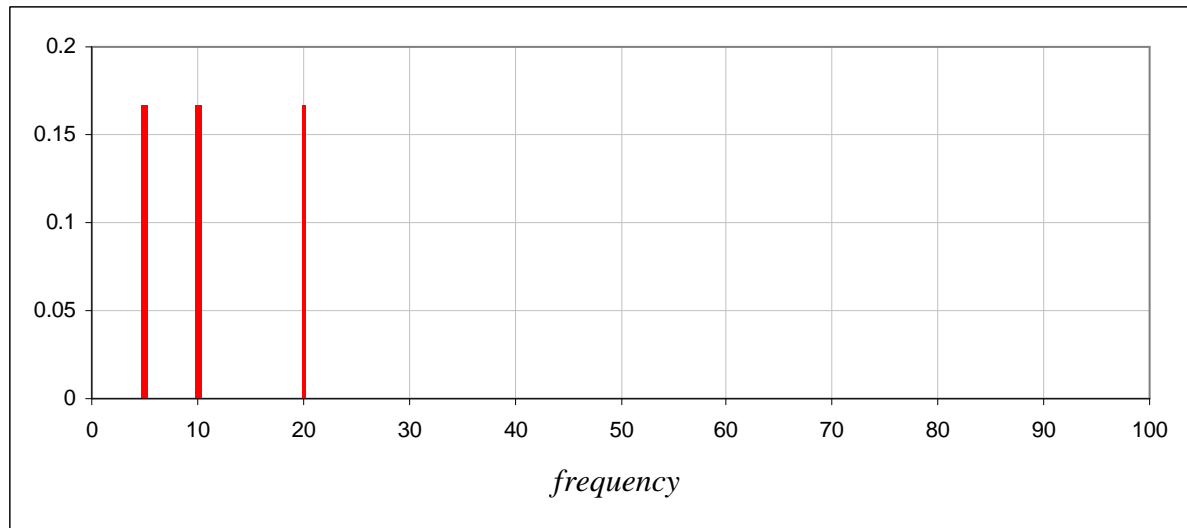


Figure 4.1 – DFT histogram of $f(t) = \frac{1}{3} \sin(10\pi t) + \frac{1}{3} \sin(20\pi t) + \frac{1}{3} \sin(40\pi t)$

In this case, the reciprocity relationship discussed in chapter two tells us that the sampling rate and the length of the signal are more than adequate to yield an accurate discrete transform into the frequency domain. Chosen values are:

$$L = 2,$$

$$\Delta t = \frac{1}{8192}.$$

So:

$$N = \frac{L}{\Delta t}$$

$$= 16384.$$

$$2B = \frac{N}{L}$$

$$= 8192.$$

$$\Delta \nu = \frac{2B}{N}$$

$$= \frac{1}{2}.$$

The highest frequency which needs to be supported is 20Hz, which falls well within the bandwidth (-4096Hz to $+4096\text{Hz}$). At the other end of the scale, the lowest frequency of 5Hz is also supported, since the frequency resolution is 0.5Hz.

Now, if we chop this signal into four equal portions, as in **Figure 4.2** (in preparation for a STFT) we have altered the phase and effectively destroyed the stationarity of the wave. Furthermore, the value of L is now only 0.5 seconds. Although the bandwidth, $2B$, stays the same, the values for N and $\Delta\nu$ have changed:

$$N = \frac{16384}{4}$$

$$= 4096.$$

$$\Delta\nu = \frac{8192}{4096}$$

$$= 2.$$

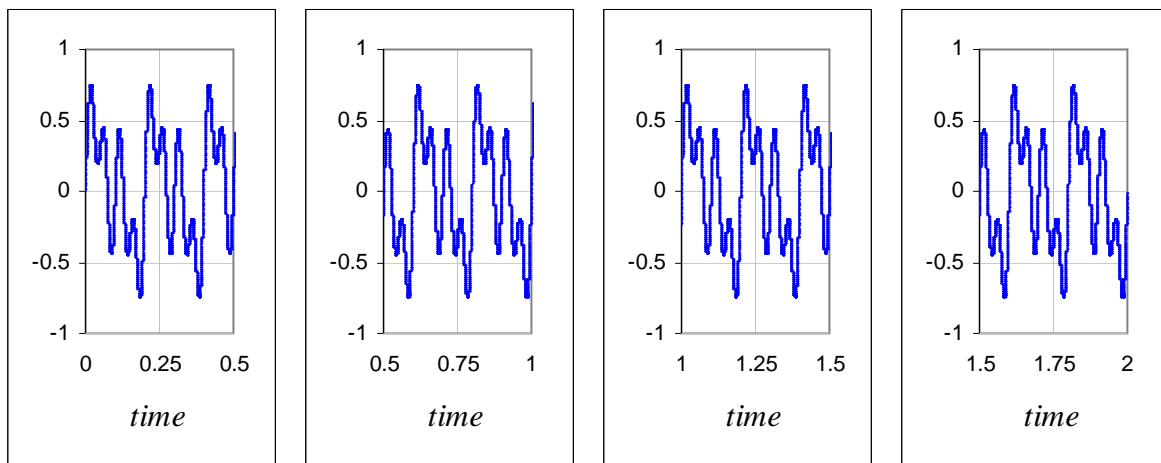


Figure 4.2 – $f(t)$ split into four windows

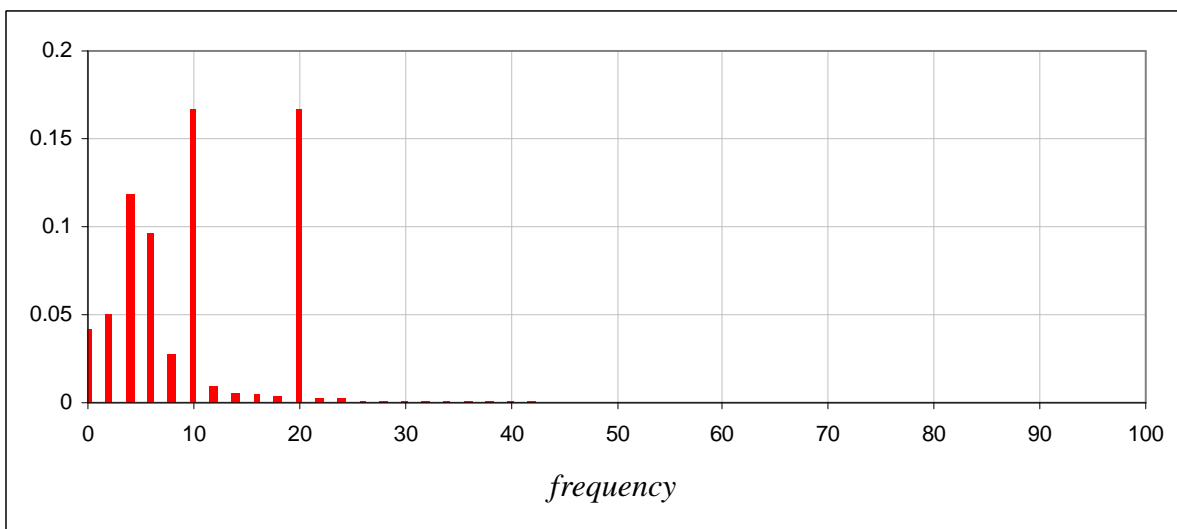


Figure 4.3 – DFT histogram of $f(t)$ for each window

The adverse effects expected from the phase alteration and the poorer frequency resolution may be seen in **Figure 4.3**, which shows the DFT of each of the four windows of the signal: the 5Hz peak (which no longer has its own bin) appears to have leaked into neighbouring frequency bins, hence the name for this phenomenon – *spectral leakage*. Another way of looking at the above leakage at 5Hz is to think of the wavelength of this signal component, 0.2 seconds, in comparison with the width of the window, 0.5 seconds. The window width is not wholly divisible by this wavelength – two and a half oscillations of the wave component fit into each window. The other two frequency components, 10Hz and 20Hz, are unaffected because five and ten complete cycles, respectively, fit neatly in 0.5 seconds, and so their phase is always zero at the start of each window.

The above example signal is specifically designed for the concepts which it demonstrates. Naturally, real world audio signals rarely contain such conveniently divisible frequency values, and so these issues almost always arise when slicing up waves. To try and combat the phase problem, a window function (other than the current Rectangle Function, $w(t) = 1$) which tapers the left and right edges of the signal slice should be used. This effectively makes each window one complete cycle of a wave, regardless of the original signal, and still preserves the frequency content, at least in the middle of the window.

4.1.3 Window Functions

There are many window functions, which have varying degrees of effectiveness. Amongst the more common ones in use, are the Triangle (or Bartlett), Hann, Hamming and Cosine functions*, but (largely thanks to Gabor and Bastiaans) we know that the best improvement, i.e. the best localization of frequencies in Fourier space, is achieved by using a Gaussian Function [McLeod02L], where $\sigma \leq 0.5$:

$$w(t) = e^{-\frac{1}{2}\left(\frac{t-W/2}{\sigma W/2}\right)^2}.$$

Figure 4.4 shows a Gaussian curve ($\sigma = 0.5$) and one of the windows of the same $f(t)$ again after the function has been applied to it. The FT of this window is shown in **Figure 4.5**.

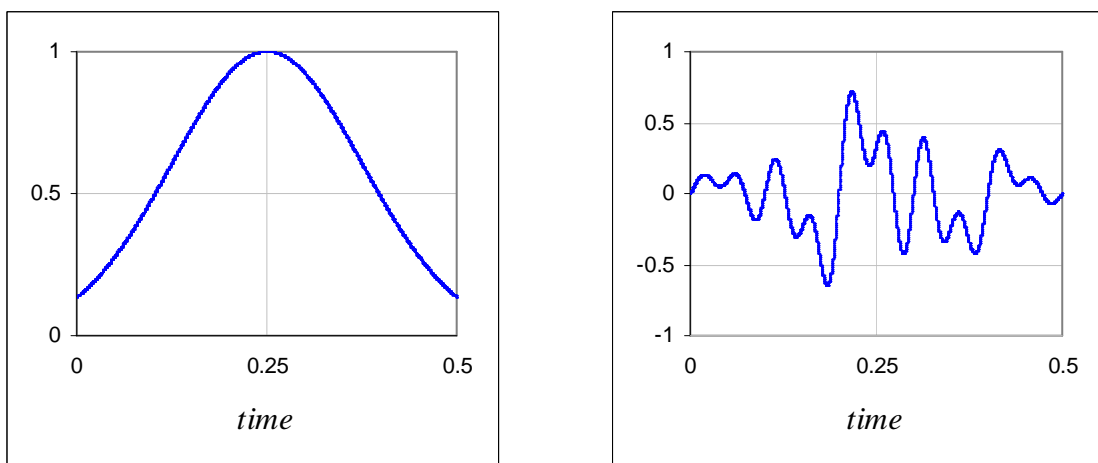


Figure 4.4 – Gaussian function and its product with the first window of $f(t)$

* Please see **Appendix A.2** for a list of window functions supported by *Wave Processor*.

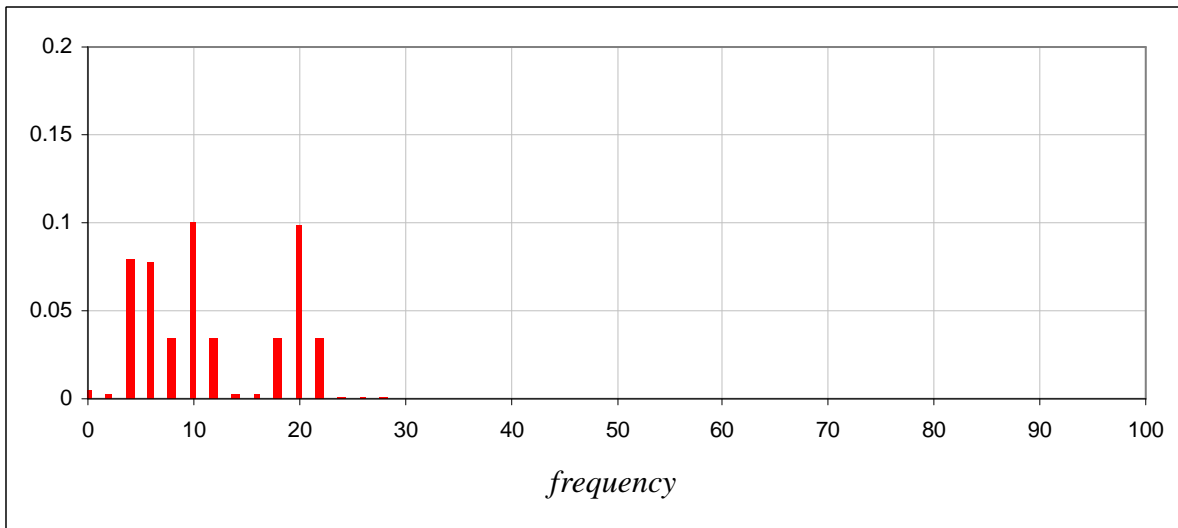


Figure 4.5 – Fourier transform of windowed signal

Although the amplitudes in the graph are diminished, due to the damping effect of the window function, and there is still some leakage, the localization is now much better for the 5Hz component.

Note that since the quantization of the frequency domain does not provide for a 5Hz bin, the 4Hz and 6Hz bins claim roughly the same probability of realising the 5Hz component – it should be clear by now that these two peaks do not necessarily indicate the presence of two weaker 4Hz and 6Hz components instead. The next section describes a very popular technique for combating this type of ambiguity, which achieves even more accurate frequency estimation with the STFT by exploiting the phase information in the Fourier transform, hitherto ignored in favour of the much more obvious magnitude information.

4.2 The Phase Vocoder

The term “phase vocoder” is not to be confused with the vocoder, short for voice encoder, which is a telecommunications device designed in 1928 by Homer Dudley, a Bell Laboratories engineer [Raphael06L]. The vocoder was originally built to encode speech before transmission, so that secure messages could be sent over radio. Dudley also created another device capable of outputting synthesized speech – the voder, which stood for Voice Operation Demonstrator. This machine was operated by a technician who would control a set of keys and a pedal which manipulated different aspects of a carrier signal, such as spectral content, frequency and type of sound emitted*.

4.2.1 A Brief Look at the Vocoder

The vocoder works by analysing speech and measuring changes in frequency spectra over time. As will be seen in the next subsection, this is why the name was borrowed for the analysis technique. It then splits the speech signal into several frequency bands (often ten) and uses the measured spectrum changes to determine a level of the signal’s energy in each band at any given time, thus creating a set of filters. This may be likened to performing a

* For a full demonstration of this machine, please listen to `voder.wav` in the Sound folder on the project CD.

Short Time Fourier Transform with a very coarse frequency resolution, but instead only recording the *differences* in frequency per bin between each time window. In order to recreate the speech, in the decoding process a noise signal with a large bandwidth, called the *carrier signal*, is passed through the filters, yielding the original speech, but sounding somewhat dehumanized.

Vocoders became very popular in music and the entertainment industry in general. Instead of noise, the carrier signal for this type of musical vocoder is synthesized musical sounds. A good example of the use of a vocoder in music is the song “Hide and Seek” by Imogen Heap, which may be found on the project CD [Heap05M]. Another amusing and clever example of the use of a vocoder is in the remix of Carl Sagan’s *Cosmos*, which appeared on YouTube to great acclaim in September 2009 [Boswell09M].

4.2.2 From Frequency to Phase Difference

As mentioned previously, the vocoder records differences in the spectral content between time frames. The Phase Vocoder, which is a computer algorithm rather than a physical electronic device, also determines differences in frequency information, but with special regard to phase. The method was introduced in the 60s by James Flanagan and Roger Golden, two American researchers at Bell Laboratories [Flanagan66L].

The first step in the Phase Vocoder algorithm is to generate more windows of the signal which overlap with windows in the regular STFT. Usually a sufficient overlapping is from half-way through one window into the next, as indicated by **Figure 4.6**, which shows the same signal from the previous section, $f(t)$, now split into seven windows (the original four plus three in between). Each of these sections has been multiplied by the Gaussian window function.

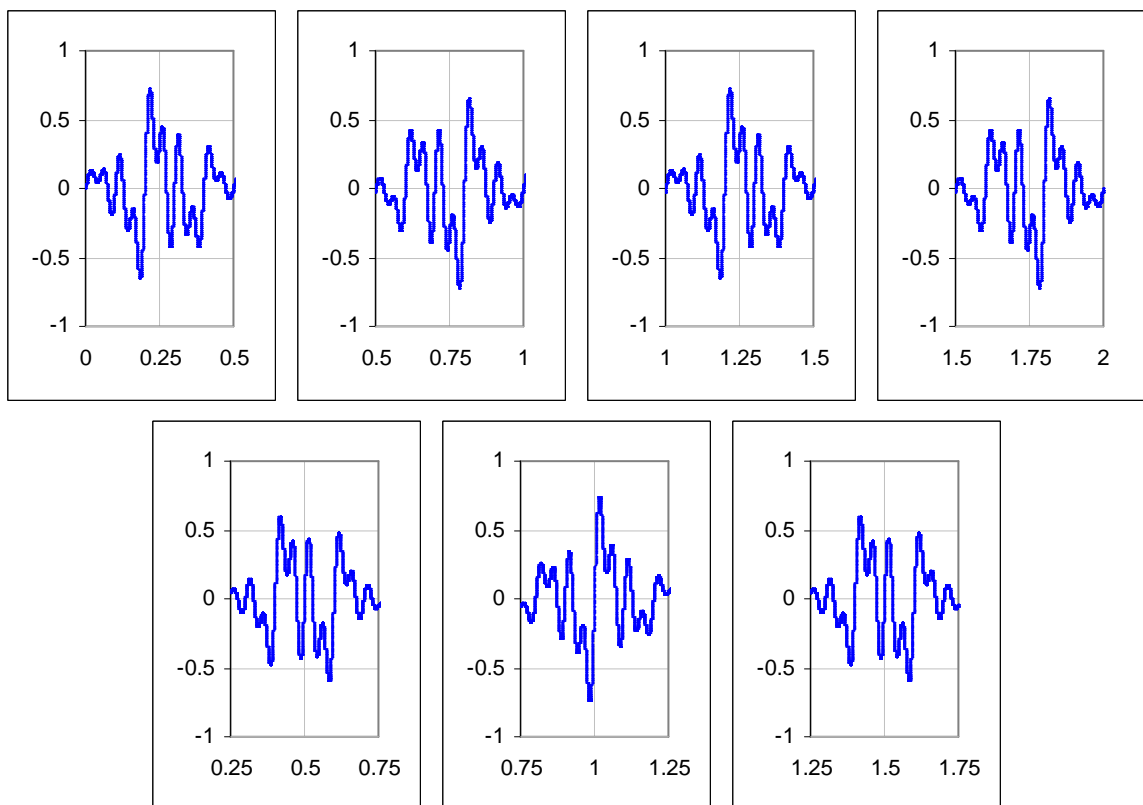


Figure 4.6 – $f(t)$ split into seven overlapping windows

The reason for doing this is to create a slight phase difference for the same frequency component between two windows, and then to examine this difference in the frequency domain to obtain a much more accurate measure of its value. Referring back to **Figure 2.4** from the previous chapter, if θ_1 represents the phase of the frequency in the first window, then θ_2 is its phase in the second window and ϕ is the phase difference caused by the small time shift between the two windows, $t_2 - t_1 = \Delta t$. Depending on the angular velocity of the frequency, the phase position at θ_2 may be the very next position in the cycle of the wave in time Δt , or it may be the position plus an unknown number, n , of complete cycles, i.e.

$$\theta_2 - \theta_1 = \phi + n2\pi.$$

The value of ϕ restricted to the range $-\pi$ and π is known as its *principle argument*, which may be written as the function $\text{princarg}(\phi)$.

Figure 4.7 shows the first of the new overlapped windows (from 0.25 to 0.75 seconds) and its Fourier transform histogram. Comparing with **Figure 4.5** for the first window, the frequency magnitudes are much the same. However for the peaks at 4Hz and 6Hz, the real and imaginary parts of the Fourier coefficients are quite different when one examines the raw output. **Table 4.1** shows these values and the resulting magnitudes and phases.

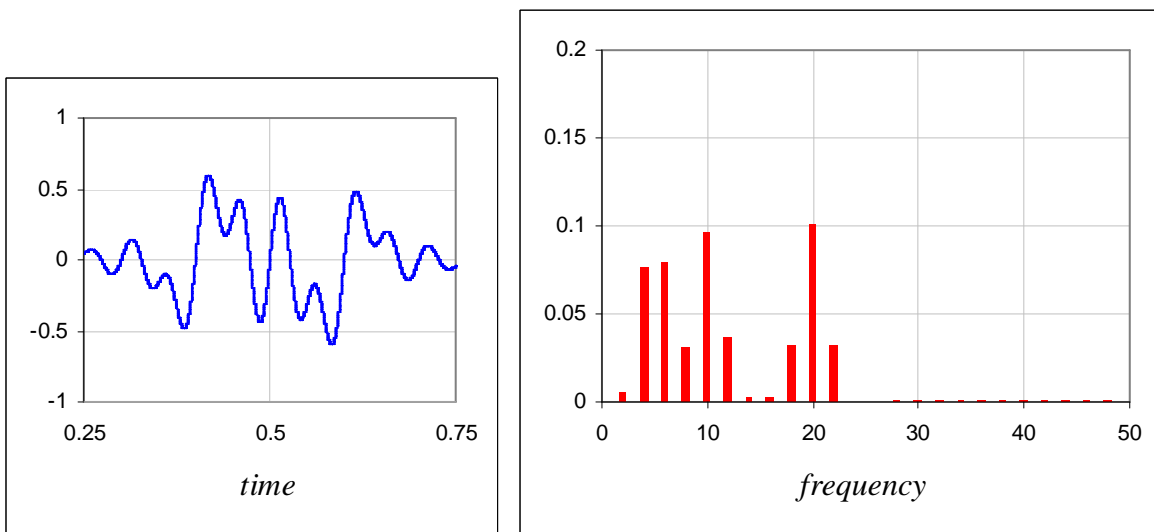


Figure 4.7 – Windowed signal from 0.25 to 0.75 and its Fourier transform

	Window	Real Part	Imaginary Part	Magnitude	Phase
4Hz Bin	1	0.079526	0.000843	0.079531	0.010600
	2	0.000011	0.075847	0.075847	1.570651
					ϕ 1.560051
6Hz Bin	1	0.076778	0.001078	0.076785	3.127553
	2	0.000011	-0.080098	0.080098	-1.570659
					ϕ -4.698212

Table 4.1 – Comparison of Fourier coefficients from first and second time windows

Now, given that the *angular* frequency, $\omega = 2\pi\nu$, is the time taken to complete one cycle of the wave, it follows that the change in angle, ϕ plus n complete cycles, must be equal to ω multiplied by the change in time, Δt , or:

$$\phi + n2\pi = 2\pi\nu \cdot \Delta t.$$

$$\therefore \nu = \frac{\phi + n2\pi}{2\pi \cdot \Delta t}.$$

Substituting the values into the equation for the 4Hz frequency bin, we get:

n	ν
0	0.99316
1	4.99316
2	8.99316
3	12.99316

Since 4.99316 is the closest value to 4, the correct value of n in this case is 1, and thus actual frequency represented by this spike is 4.99Hz – a huge improvement for the measure of the 5Hz component.

Similarly, for the 6Hz frequency bin, the values when substituted into the equation for n and ν are:

n	ν
0	-2.99097
1	1.009025
2	5.009025
3	9.009025

In this case, since the value of ν for $n = 2$ is closest to 6Hz, this is the correct value for n . So the peak in the 6Hz bin also in fact represents a measurement which is much closer to 5Hz.

The refined accuracy possible with the Phase Vocoder technique makes it an extremely popular frequency estimation method, and so it has been included for experiment in *Wave Processor*. However the following section describes a possibly even better technique which works extremely well with melodies in particular, as will be seen.

4.3 The McLeod Pitch Method (MPM)

A fairly detailed description of MPM is given in the paper aptly titled *A Smarter Way to Find Pitch* [McLeod05L]. It was developed as an improved solution to an earlier effort – *Visualization of Musical Pitch* [McLeod02L] – which made use of a Short Time Fourier Transform with a Gaussian window.

Since this paper was the only source (apart from the *Tartini* source code), the implementation of the algorithm in the *Wave Processor* application has been based entirely on this one paper. This section is, for the most part, to show how the code was derived, and so there is some repetition of McLeod’s work. However, along the way, one or two gaps in the argument

needed to be filled – some of the more basic steps which it appears were originally left to the reader to decipher.

4.3.1 General Algorithm Description

The “smarter way” is also a time-window method. McLeod has shown a useful relationship between two functions, namely the *Autocorrelation Function* and the *Square Difference Function*, and also how they may be used to calculate the *Normalized Square Difference Function*. This is all described in the following two subsections. From the normalized square differences, a peak picking algorithm is used to extract the strongest frequency (usually the fundamental) hence the pitch at each time frame. This process is explained in subsection 4.3.4.

4.3.2 The Autocorrelation and Square Difference Functions

The autocorrelation function, $R(\tau)$, is, as its name suggests, a measure of how much a set of data is similar to itself at different time intervals, τ . It is useful in signal analysis for finding repeating patterns, or, as in this case, identifying the fundamental note of a sound signal comprising a certain frequency plus its harmonics. There are many autocorrelation functions to choose from, depending on the application at hand. For signal processing, the function that is most often used is given by:

$$R(\tau) = \int_{-\infty}^{+\infty} f(t) f^*(t - \tau) dt,$$

where $f(t)$ is the signal function, τ is the time interval or lag and $*$ denotes the complex conjugate.

For a discrete set of samples, f_n , autocorrelation may be computed over windows of width W [McLeod05L], starting at a time index, n , according to:

$$R_{k,n} = \sum_{j=n}^{n+W-1-k} f_j f_{j+k}. \quad [4.1]$$

Notice that as k increases, the number of terms in the summation decreases. This is because the windowed signal is, out of necessity, padded with zeros beforehand, and so for larger k , fewer non-zero terms are being used in the calculation.

The square difference function, $D_{k,n}$, which is the one we eventually want a normalized version of, is given by:

$$D_{k,n} = \sum_{j=n}^{n+W-1-k} (f_j - f_{j+k})^2. \quad [4.2]$$

This function is useful since it yields minima when the lag, k , is a multiple of the number of samples in a pitch period in the signal. Revisiting the example of a vibrating string briefly (see the section on harmonics in chapter two), these minima relate to the stationary points at

regular intervals along the string, which will always have an amplitude of zero. Expanding the brackets of equation [4.2], we get:

$$D_{k,n} = \sum_{j=n}^{n+W-1-k} (f_j^2 - 2f_j f_{j+k} + f_{j+k}^2) \quad [4.3]$$

From [4.1], this then becomes:

$$D_{k,n} = \sum_{j=n}^{n+W-1-k} (f_j^2 + f_{j+k}^2) - 2R_{k,n}, \quad [4.4]$$

and we see that autocorrelation is actually contained within the squared differences. The remainder of the squared differences we shall call the ‘‘Sum of Squares Function’’ or $S_{k,n}$, defined by:

$$S_{k,n} = \sum_{j=n}^{n+W-1-k} (f_j^2 + f_{j+k}^2) \quad [4.5]$$

So, finally, we can rewrite [4.4] as:

$$D_{k,n} = S_{k,n} - 2R_{k,n}. \quad [4.6]$$

4.3.3 The Normalized Square Difference Function

To normalize, McLeod divides $D_{k,n}$ through by $S_{k,n}$ and subtracts the result from one, so that minima become maxima and the range of the resulting graph is now between -1 and $+1$.

Thus, we have the Normalized Square Difference function, $N_{k,n}$, defined by:

$$N_{k,n} = 1 - \frac{D_{k,n}}{S_{k,n}}. \quad [4.7]$$

Then, from [4.6]:

$$N_{k,n} = 1 - \frac{S_{k,n} - 2R_{k,n}}{S_{k,n}} \quad [4.8]$$

$$= \frac{2R_{k,n}}{S_{k,n}}. \quad [4.9]$$

Thus to compute the normalized square differences, McLeod must compute both the autocorrelation and the sum of squares. He uses Fourier theory to calculate these efficiently as follows:

In order to compute $R_{k,n}$, McLeod uses an algorithm which incorporates the FFT (derived from the Wiener-Khinchin Theorem [Wolfram09W]), so the computation time is reduced from $O(Ww)$, when calculated by summation, to $O((W+w)\log(W+w))$. The steps, as given in part 6 of the paper [McLeod05L] are worth outlining again here:

1. Zero pad the window by the number of normalized values required, W .
2. Take a Fast Fourier Transform of this real signal.
3. Multiply each complex coefficient by its conjugate (yielding power spectral density).
4. Take the inverse Fast Fourier Transform.

When implemented in software, this part of the algorithm is actually only a few lines of code, as shown in **Appendix C.2** (see the ACF code listing).

The sum of squares part of equation [4.9], $S_{k,n}$, may also be calculated quickly by using the result from $S_{k-1,n}$ and subtracting the appropriate f_j^2 , thus:

$$S_{k+1,n} = S_{k,n} - f_{k+n}^2 - f_{n+W-1-k}^2.$$

In the case of $k = 0$, from [4.5] we have:

$$\begin{aligned} S_{0,n} &= \sum_{j=n}^{n+W-1} 2f_j^2 \\ &= 2R_{0,n}, \end{aligned} \tag{4.10}$$

which has already been calculated. Given the ACF implementation, this particular computation for $k = 0$ is achieved in one line of code:

```
double ss = ACF(in, out, N) * 2;
```

Thus, the full computation of the normalised square differences is easily implemented in **Appendix C.3** (see the NSDF code listing).

Note that $N_{0,n}$ will always be equal to 1:

$$\begin{aligned} N_{0,n} &= \frac{2R_{0,n}}{S_{0,n}} && \text{from [4.9]} \\ &= \frac{2R_{0,n}}{2R_{0,n}} && \text{from [4.10]} \\ &= 1, \end{aligned}$$

and as mentioned above, $N_{k,n}$ will never exceed +1 or fall below -1, since the greatest possible magnitude of $2R_{k,n}$ is $S_{k,n}$, i.e.:

$$|2R_{k,n}| \leq S_{k,n}.$$

An example of NSDF output is shown in **Figure 4.8** (drawn in Microsoft Excel from debug output of the testing software). It can be seen that the graph tapers somewhat, due to more zero terms being used in the calculation (since the window is zero-padded) as k , the sample index, increases. The main peaks of the graph tell us the pitch period of the fundamental note as well as those of any harmonics. Usually the fundamental frequency is the strongest, and in the case of the example graph, it seems that the main peaks occur approximately every 225 samples within this particular window (which is 1024 samples wide). To obtain to the actual frequency, however, we must divide the number of samples per second in the digital signal (in this case 44.1kHz) by this value:

$$\frac{44100}{225} = 196.$$

196Hz is equivalent to the musical pitch G₃.

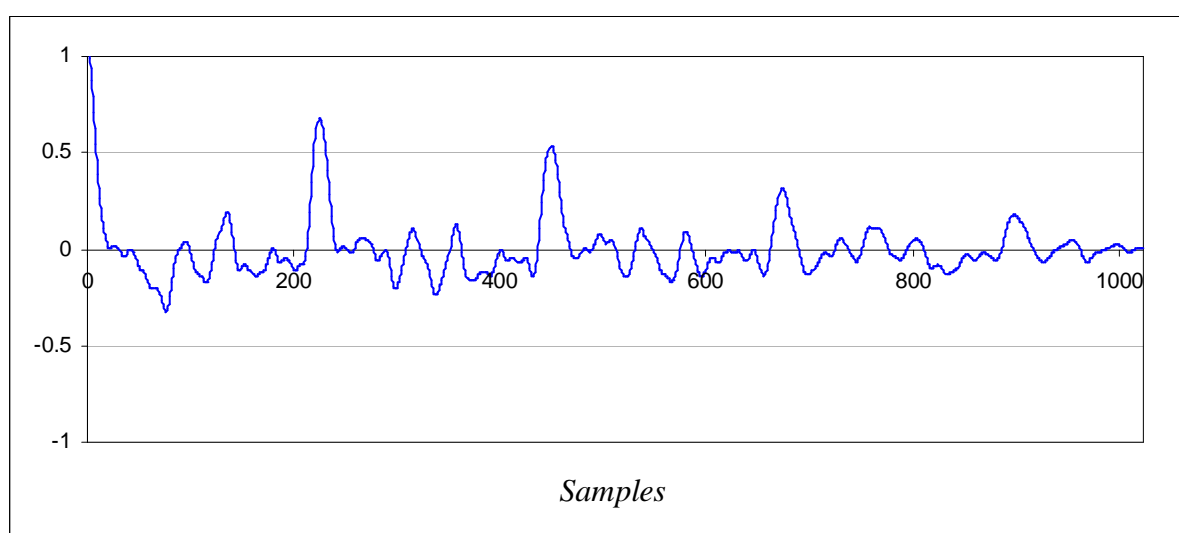


Figure 4.8 – An example of NSDF output

4.3.4 Peak Picking Algorithm

An efficient method is needed to choose peaks, as was done manually in the example above. The source code for an adapted version of McLeod’s peak-picking routine, based on his description in section 5 of [McLeod05L], is listed in **Appendix C.4**. This function takes the NSDF output and fills an array of integers with the values of τ at the chosen maxima. The number of peaks found is then returned.

As a kind of trade-off, McLeod’s parabolic interpolation step has been deliberately left out, since the same kind of accuracy that he was after is not really required here. His method yields a slightly more refined calculation of the pitch period by fitting a curve through the maximum and the two points either side of it and then finding the turning point of the parabola, rather than just picking the maximum. Without using parabolic interpolation it is still possible to calculate pitches to well within the nearest semitone.

The following pseudo-code should suffice as an explanation of how the peak-picking function works, without simply repeating McLeod’s description:

```

PICKPEAKS(NSDF, PEAKS[], n) {
  Find first negative zero crossing
  LET N_PEAKE_FOUND = 0
  WHILE not at end of NSDF {
    Find next positive zero crossing
    LET MAX_PEAKE = 0
    WHILE graph is positive and not at end of NSDF {
      Find next LOCAL_PEAKE
      IF LOCAL_PEAKE > MAX_PEAKE {
        LET MAX_PEAKE = LOCAL_PEAKE
      }
    }
    Record MAX_PEAKE in PEAKS[] array
    Increment N_PEAKE_FOUND
  }
  RETURN N_PEAKE_FOUND
}

```

Once the “key maxima” have been found, a threshold is defined, which is equal to the product of the **highest maximum** and a constant, κ , which has an effect on how well the algorithm is able to discern between the fundamental note and a strong harmonic. The lower the value, the more likely the chance that a pitch will be identified, but the less likely the correct fundamental note (rather than a harmonic) will be chosen, and vice versa. As stipulated in [McLeod05L], κ should be between 80% and 100%. The dominant frequency for the window is chosen by taking the lag, k , for the first key maximum above the threshold and performing the calculation as shown in the example in the section above. When a clearly defined pitch is not found, the frequency is set to zero.

A measure of the clarity of a pitch is also obtained directly from the amplitude of the chosen key maximum, which then indicates how coherent the detected pitch is. As is done in *Tartini*, clarity may be represented by changing the intensity of the colour when plotting each pitch in the output – the clearer the pitch, the brighter the colour. The clarity value is also set to 0 if a pitch has not been found.

4.4 Drawing a Pitch/Time Graph

In the previous chapter, a musical score was described as a graph of pitch against time. Of course, a score is much more than just a graph and contains many other symbols and written instructions which indicate how the music it represents should be played. Since the main purpose of this study is pitch identification and not full score re-synthesis and rendering, it is necessary that the output be much more simple and in the form of a pitch/time graph.

4.4.1 Calculation of Relative Pitches from Frequencies

As explained in chapter three, pitches and frequencies are not the same, although they have a dyadic relationship. The McLeod Pitch Method renders frequencies rather than pitches, therefore in order to draw something which more closely resembles a musical score, a conversion is needed. For yielding a note on the Western musical even-tempered scale, the following pitch formula (adapted from the one given in [McLeod02L]) should be used:

$$p = \frac{\log(v/c_0)}{\log(\sqrt[12]{2})}$$

In program code, this translates as:

```
pitch = log(freq / C0) / log(TWELFTH_ROOT_2);
```

C_0 is the base frequency, i.e. the note whose pitch we shall assign a value of 0. For musicians, this is equivalent to the C four octaves below Middle C on a piano (in fact a few notes lower than the lowest note on most pianos – usually A_0), which has a frequency of 16.375Hz. All other pitches may then be calculated in terms of how many semitones they are above the base note, hence the twelfth root of two factor (see chapter two). For example, the note A_4 has a frequency, $v = 440$ Hz. Substituting the values into the pitch formula, we get:

$$\begin{aligned} p_{A_4} &= \frac{\log(440/16.375)}{\log(\sqrt[12]{2})} \\ &= \frac{1.429}{0.025} \\ &\approx 57. \end{aligned}$$

Thinking in terms of pitches, this result is as expected, since A_4 is 4 octaves + 9 semitones above C_0 :

$$4 \times 12 + 9 = 57.$$

4.4.2 Graphical Representation

Using the pitch formula on the fundamental frequency in each window, we can produce a set of linear pitches by shifting the window (in our case, $\frac{1}{4}$ window length at a time). We need to plot these pitches on some sort of graph that finds a happy compromise between something a mathematician would understand and that which a musician is used to reading, preferably leaning towards the latter – the closer the output is to an actual score, the better.

At this point we are unable to divide notes into measures, draw barlines or even use Franconian notation (see chapter three) because further analysis of the output is required in order to determine the duration of a beat and decide on correct quantization. Therefore, for now, time should be represented by length on the x -axis.

As for pitch, further analysis of content based on simple rules of Harmony, as outlined in chapter two, is also required. This is necessary in order to write correct key signatures according to the detected mode, so that music may be written on the much more compact five line staff without having to use too many accidentals. Since this kind of analysis is a post-processing problem, for the time being the key is always assumed to be C Major, which does not have a key signature. Notes are thus spelt according to diatonic pitches within this key, e.g. the note between F and G is $F\sharp$ rather than $G\flat$. See **Table 3.10** in chapter three for all correctly spelt diatonic pitches in the scale beginning on C. In the case of $D\sharp / E\flat$, $D\sharp$ has been chosen since the key in which it appears, E Major, is more closely related to C Major /

Note the following with regards to the pitch graph:

- Accidentals are indicated by the use of different colours: blue means the note is a natural Guidonian pitch, red is for sharps and green (not seen in this example) denotes flats. According to the key signature in the score, the melody is in the key of G Major, and so Fs should be sharpened, hence the colour of the second pitch in the graph.
- Clarity of pitch is shown by the depth of colour. Less clear pitches, e.g. the artefacts on the bass staff, are fainter.
- Listening carefully to the recording, where the algorithm has apparently miscalculated some pitches, these are all moments when the violinist makes a certain movement such as a finger shift, bow direction / bow pressure change, or slight vibrato, causing a short noise to occur. This sensitivity to noise may be controlled by adjusting the threshold, κ , discussed in subsection 4.3.4 and also a clarity threshold, which can be set at a lower level to reject pitches which are not very clear.
- The lack of temporal quantizing and the use of pixel width to show durations means that the melody must be drawn over more staff systems, in this case three instead of one. It is unfortunately necessary for now to make do without the useful compactness of Franconian notation.
- The notes are not neatly measured and spaced to fit exactly into one staff line as with the *Sibelius* score. This is a music engraving issue which will not be tackled here. Good-looking note distribution over multiple staves is considered a difficult problem, even with proper notation.
- Because of the lack of quantizing, some notes at the ends of staves are cut part-way through and wrap round onto the next line.
- There is no way of telling where some notes begin and end if they have the same pitch. For example, the Bs and As in the middle of the melody could represent minims instead of pairs of crotchets.

4.5 Comparison of Output

A performance comparison of the various signal analysis and pitch identification techniques was conducted for various different signals and the results of these experiments are presented as a conclusion to this chapter.

4.5.1 A Stationary Signal

Firstly, a simple stationary signal was constructed using *Wave Processor*'s wave creation feature. This signal is similar to the example used in chapter two – see **Figure 2.6** – in that it is also a sinusoid comprising three frequencies. The chosen frequencies were 440Hz, 555Hz and 660Hz, which, when translated into pitches, form the tonic triad of A Major. A simple Fourier transform was done, and the three frequencies appear very clearly in the graph shown in **Figure 4.11**, which is in the form of a *spectrogram*. This is a type three dimensional graph which makes use of different colours or shades to indicate amplitudes for a spectrum of frequencies. Unlike the histograms encountered previously, frequency is now on the vertical axis. Spectrograms are particularly useful representations for the purposes of this study, since, keeping time on the horizontal axis, we have a time-frequency representation of a signal – one step closer to the desired time-pitch graph. Since this particular signal is stationary, the three frequencies do not change over time.

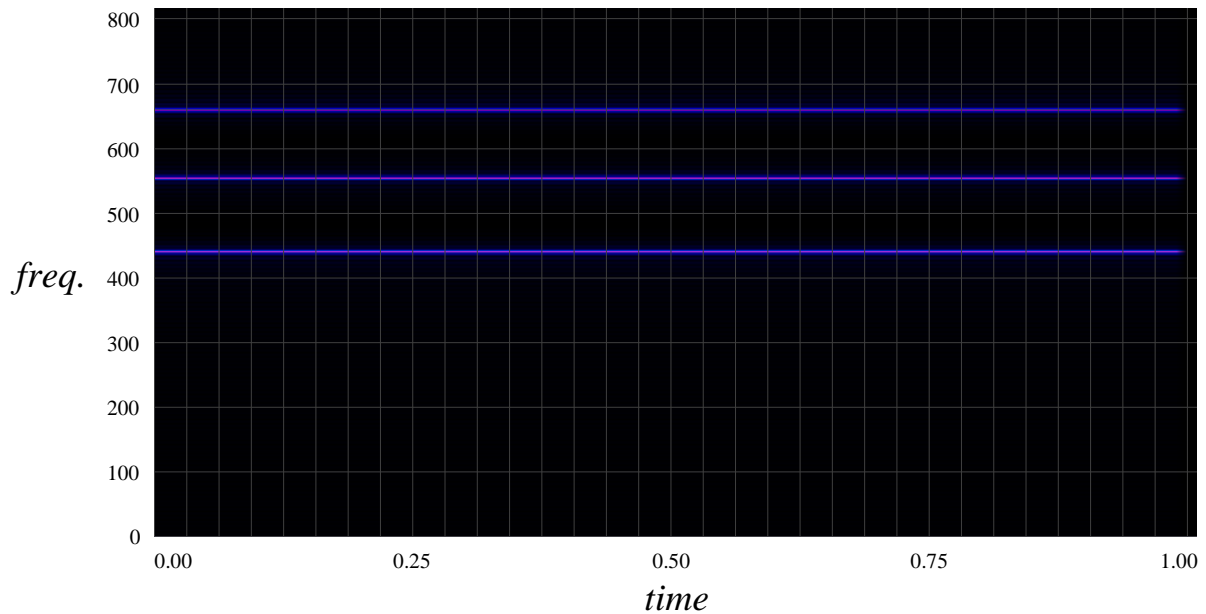


Figure 4.11 – Fourier transform of stationary signal comprising three frequencies

4.5.2 A Chirp Signal

The best example of a non-stationary wave is one which does not contain any fixed frequencies at all, and this is precisely the case with the *chirp signal*. The one constructed here is a sinusoid which oscillates over two seconds and whose frequency constantly increases throughout, starting at 32Hz and ending at 8192Hz. In chirp signals, the frequency may increase linearly or logarithmically. In this case the choice was linear, so, aurally, the sound seems to rise rapidly in pitch at first, and then the increase becomes more gradual as the frequency gets higher. This is due to the dyadic relationship between frequency and pitch, discussed in the previous chapter. Thus a linear change in frequency results in a logarithmic change in pitch and vice versa.

In the time domain, the first part of the signal is shown in **Figure 4.12**. As can be seen, the peaks of each oscillation get closer and closer as the wavelength constantly decreases and the frequency increases.

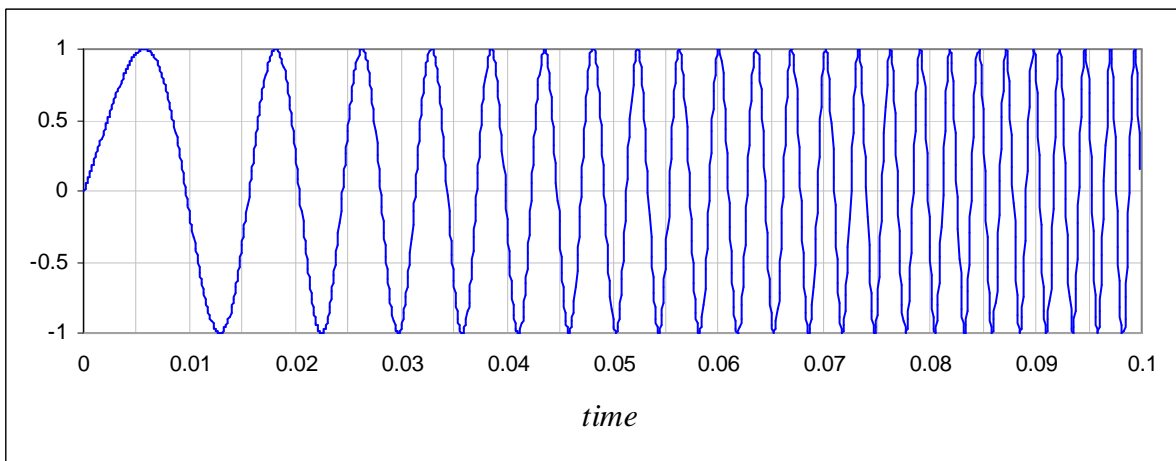


Figure 4.12 – Linear chirp signal (time domain)

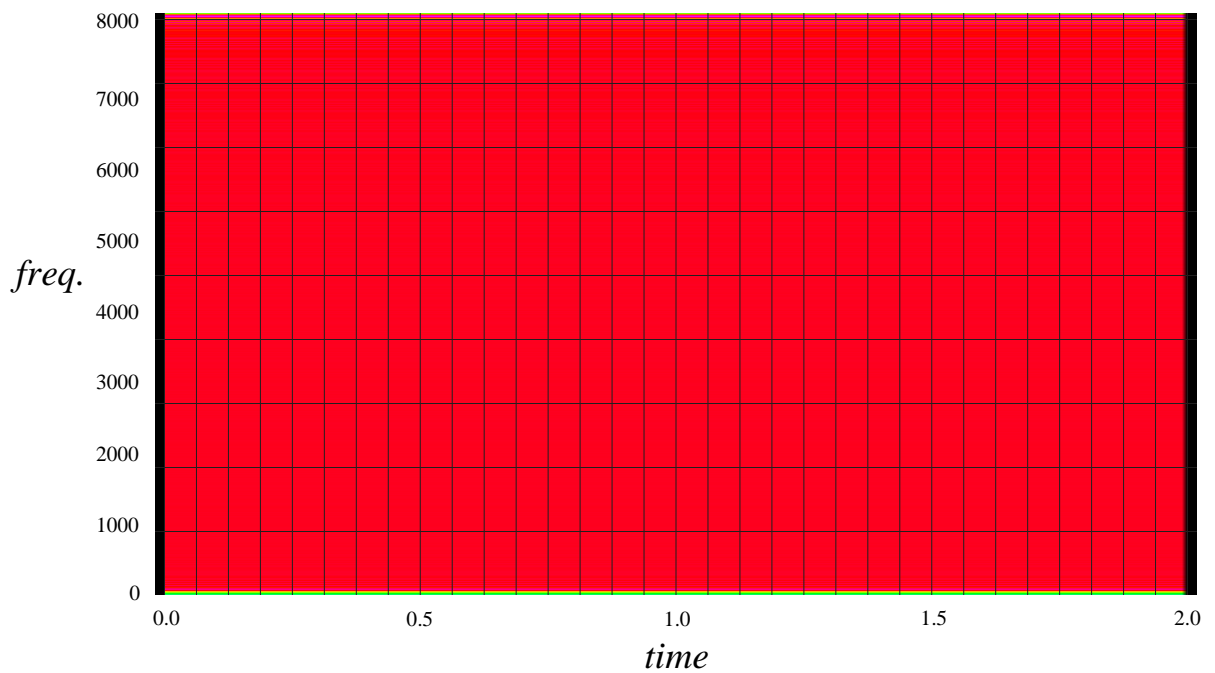


Figure 4.13 – Fourier transform of linear chirp signal

The figure above shows an ordinary Fourier transform of this signal. To explain why this spectrogram is simply a solid block of colour, one needs to consider that the Fourier transform does not operate over the time domain, hence *all* frequencies in the chirp appear with constant amplitude over *all* time. Instead of spectrograms for non-windowed Fourier transforms, *Wave Processor* is also capable of drawing histograms, similar to those encountered here and in chapter two. Bar graphs are better representations of the transform in any case where the signal is assumed to be stationary. **Figure 4.14** shows a Short Time Fourier Transform spectrogram of the chirp. Now the time axis becomes much more meaningful and the linearity of the increase in frequency is clear. In this case, the window width used was 512 samples, and no tapering function was applied.

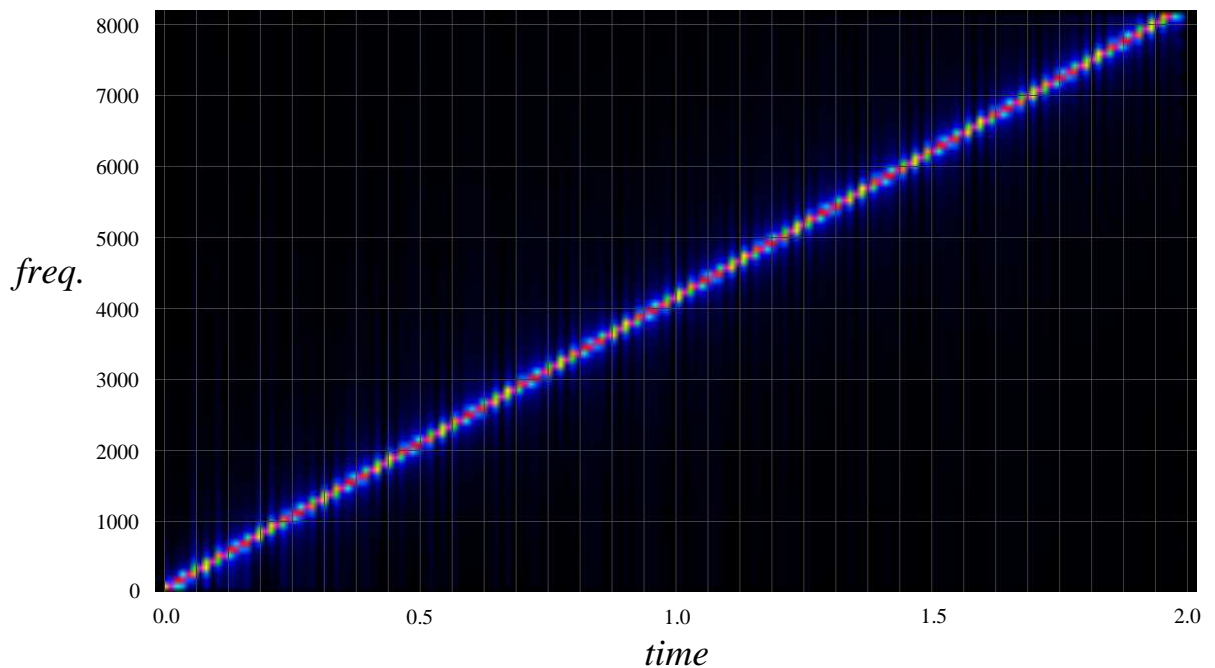


Figure 4.14 – Short Time Fourier Transform of linear chirp signal

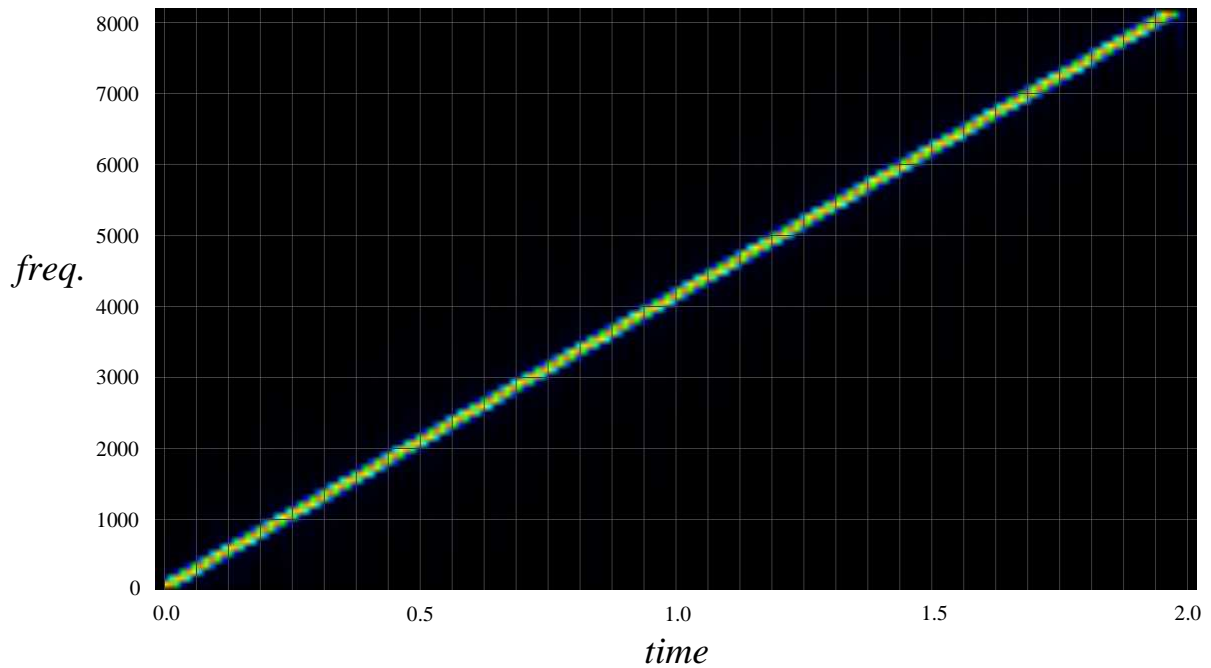


Figure 4.15 – STFT of linear chirp signal with Gaussian window function

With a Gaussian window function, **Figure 4.15** reveals how much of the blurring caused by spectral leakage disappears, despite the fact that the same window width has been used for this transform.

Finally, **Figure 4.16** shows another STFT of the chirp, also with a Gaussian window function, but with a larger window size of 4096 samples. It can be seen that frequencies are now less well-localized in time but the frequency detail is better in each window.

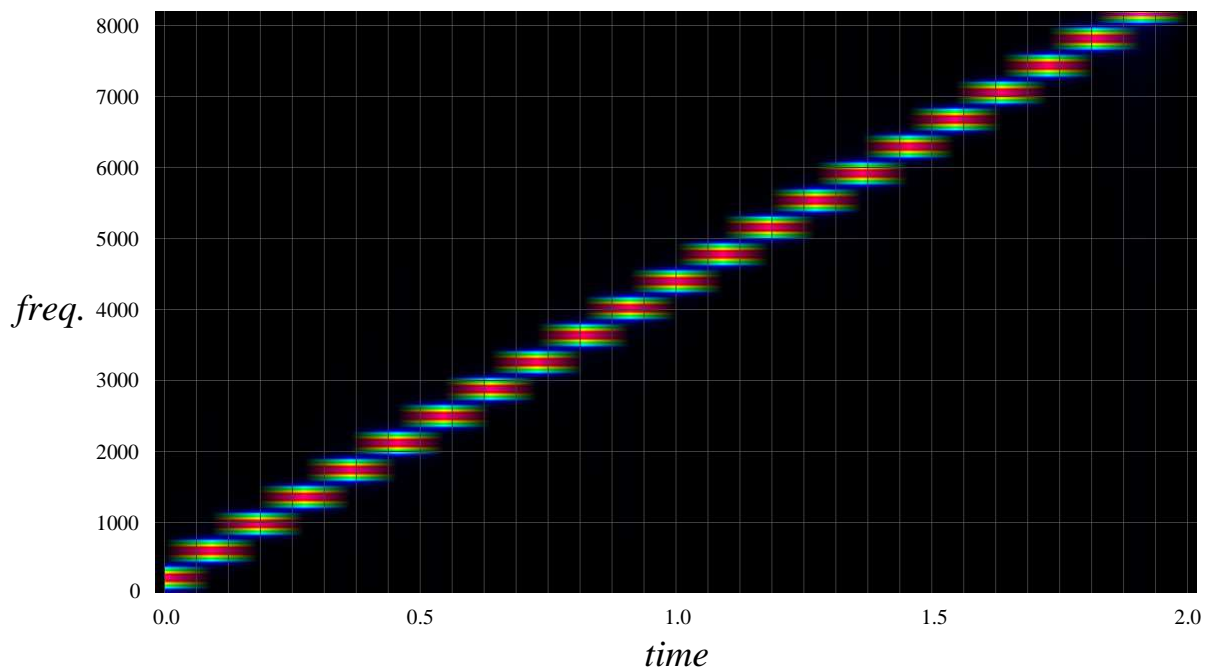


Figure 4.16 – STFT of linear chirp signal with larger window width

The Phase Vocoder and McLeod Pitch Method were then used to determine f_0 and hence the pitch of the chirp. The resulting pitch-time graphs are shown in **Figures 4.17** and **4.18** respectively. Note the clearly logarithmic change in pitch through time. The lines labelled 8^{va} , 16^{va} and 24^{va} are used in music to indicate that a range of pitches sounds one, two or three octaves higher respectively.

As can be seen, MPM is able to achieve a finer time resolution and a more precise measures of mid-range pitches, due to the smaller window shifts. This is also why its output is stretched out over two staves. However, for higher frequencies it does not seem to perform as well as the Phase Vocoder, which detected all pitches up to C_9 , which is 8384Hz – the closest estimate of the actual ending frequency of 8192Hz. Neither method performed well for the lowest frequencies, and both began an octave higher at C_1 – roughly 65Hz as opposed to the correct value of 32Hz.

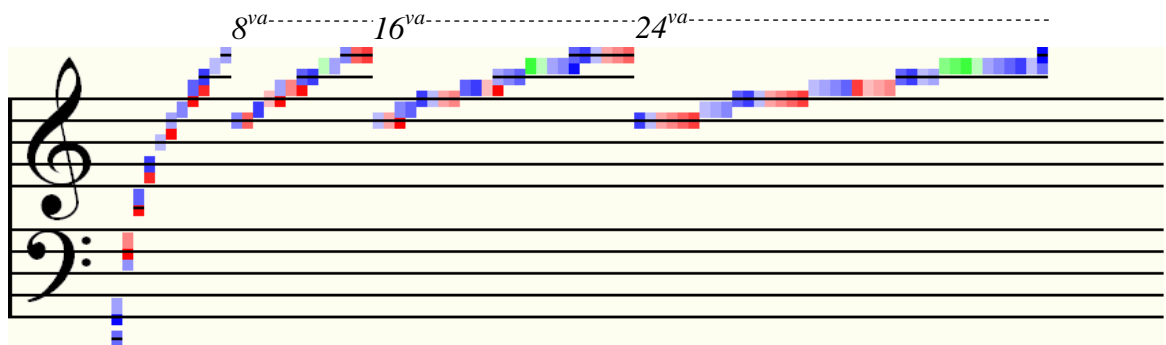


Figure 4.17 – Pitch graph of chirp calculated by Phase Vocoder

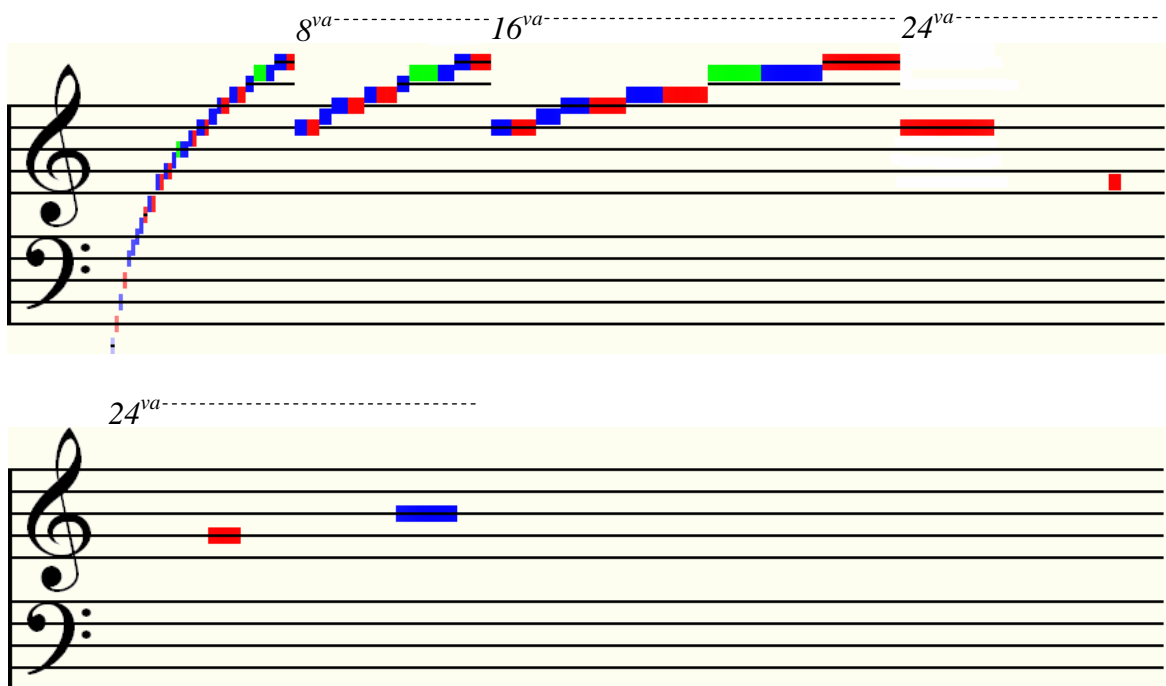


Figure 4.18 – Pitch graph of chirp calculated by McLeod Pitch Method

4.5.3 A Scale Played by a Synthetic Instrument

A scale in D Minor, as shown in the score in **Figure 4.19**, was written using Sibelius and played back using a flute sound. The output was captured and saved as a wave file*.



Figure 4.19 – Scale of D Minor, one octave, ascending and descending

The STFT, Phase Vocoder and MPM outputs are shown in **Figures 4.20, 4.21 and 4.22** respectively. For the STFT / Phase Vocoder, the window size was set to 1024, and for MPM it was 4096. Note that each B is flattened because of the key signature, and therefore should indeed appear in green, as is the case on both pitch graphs. Also, the harmonic minor form of the scale has been used, hence the red C#s.

Although not by a great amount in this instance, MPM somewhat outperforms the Phase Vocoder in general. Pitches are clearly defined and at no point is a harmonic chosen over the fundamental note. The reason for the transitional pitches, showing as different colours at the beginnings and endings of some notes in the MPM pitch graph, may be due to the timbre of the synthetic instrument. Listening to the recording it may be heard that a slight breath sound occurs at the attack of each note, presumably in order to make the samples sound more authentically like a flautist blowing. The Phase Vocoder does not detect a change in pitch at these places, however, and is as clear as MPM, although there are one or two other artefacts.

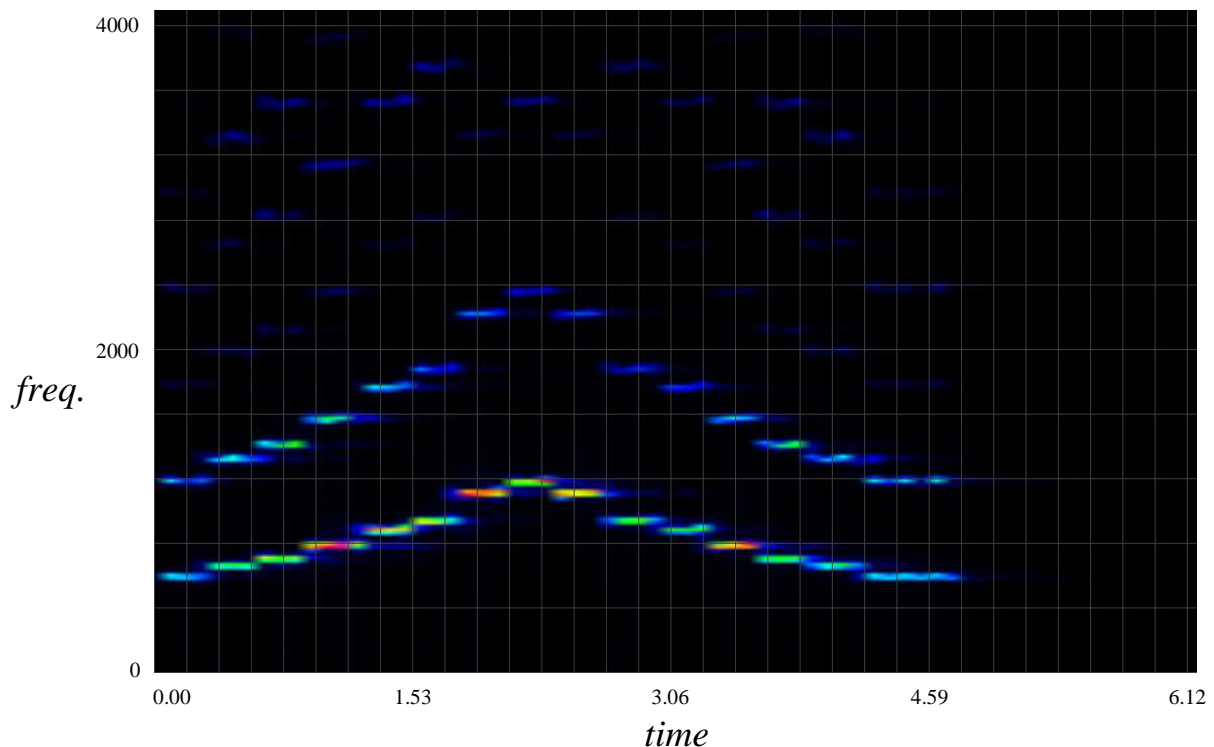


Figure 4.20 – STFT of D Minor scale

* Flute - D Minor Scale.wav on the project CD in Sound

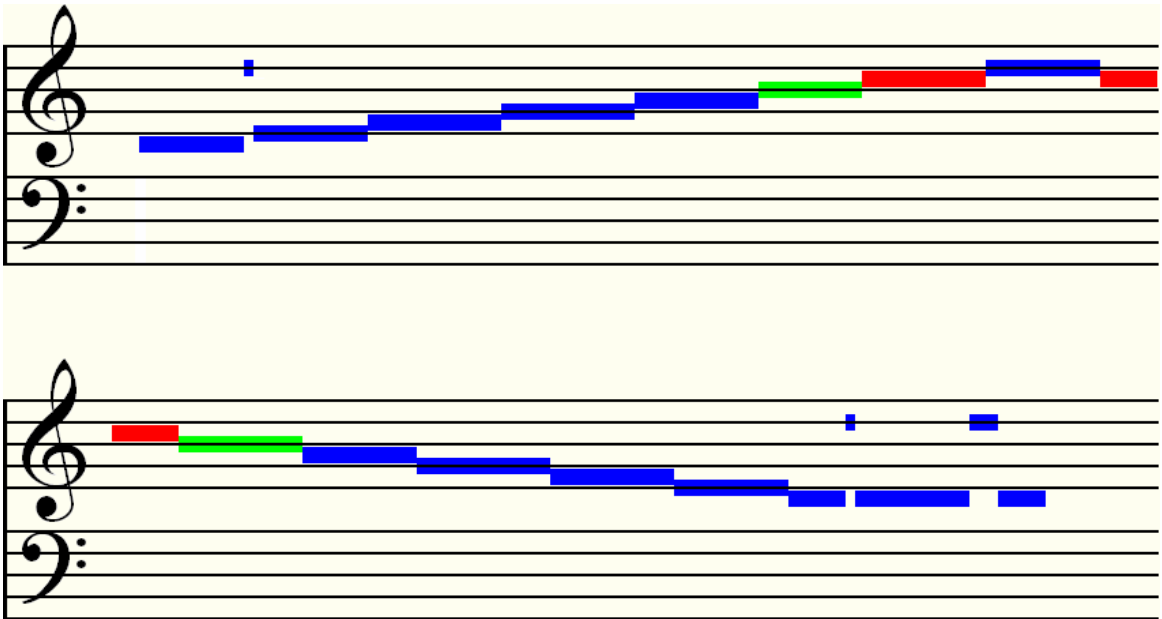


Figure 4.21 – Phase Vocoder pitch graph of D Minor scale

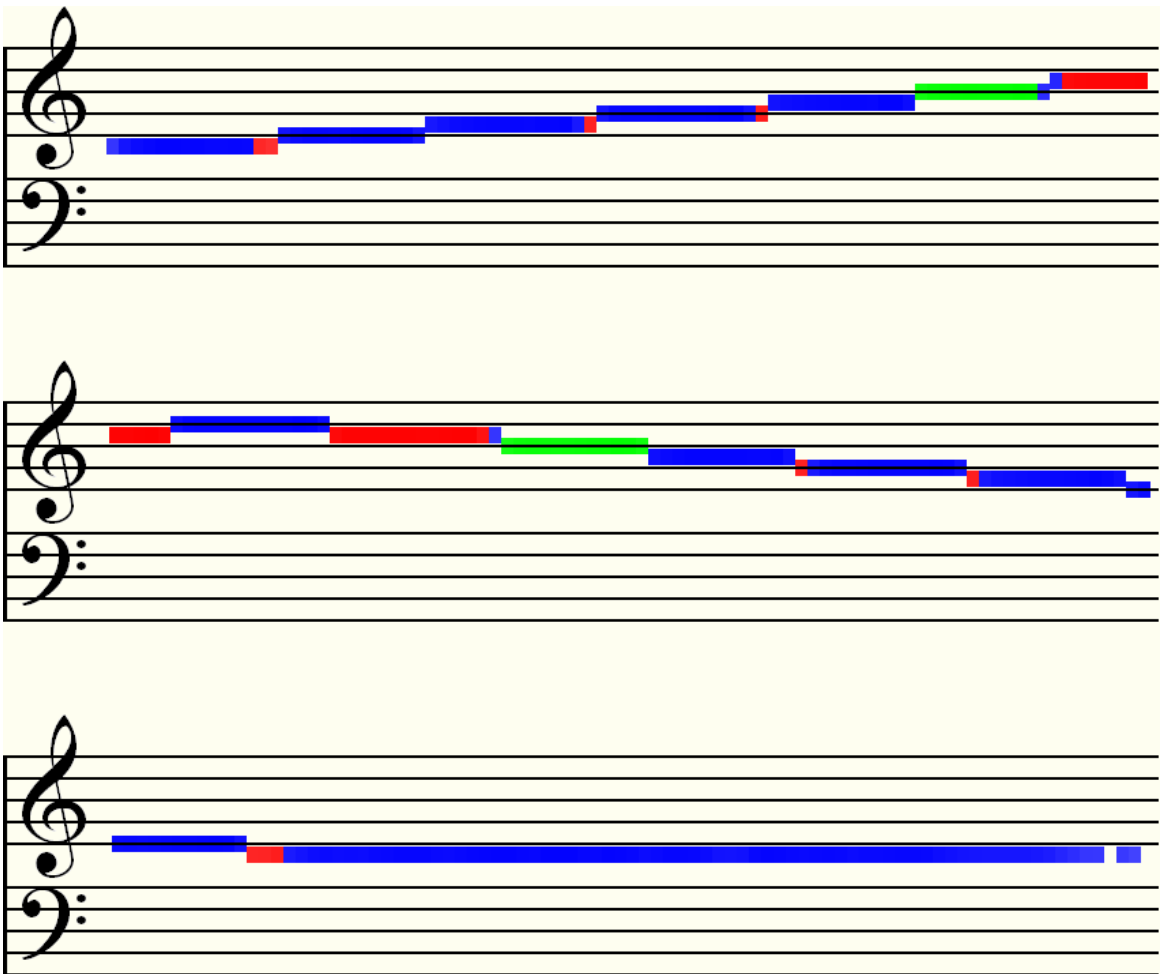


Figure 4.22 – MPM pitch graph of D Minor scale

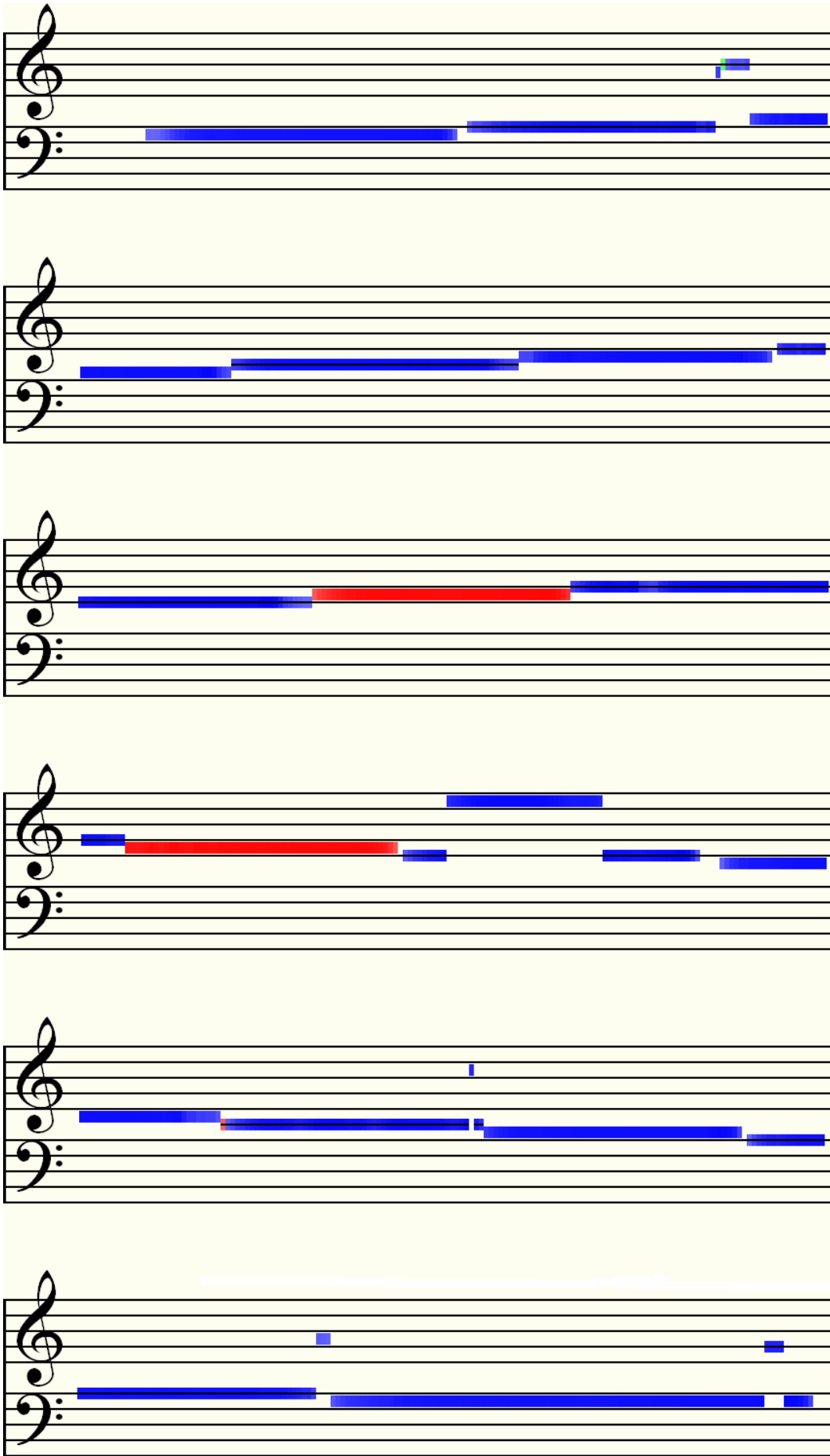


Figure 4.23 – MPM pitch graph of *Tartini* example scale

4.5.4 A Scale Played by a Real Instrument

An example wave file is included with the original *Tartini* program. This sample is also an ascending and descending scale, but is recorded from a real violin playing the notes. As can be seen in the pitch graph in **Figure 4.23**, the key of the scale may be deduced to be G Major, since the range is from G₃ to G₄, and each F is red and thus sharpened. A window width of 2048 was used in this case. The sudden jump up the octave on the note E in the fourth staff system reflects a pressure change in the violinist's bow which caused the note to squeak slightly. This may also be heard clearly in the recording.

4.5.5 A Melody

Lastly, the Phase Vocoder was also used to detect pitches in the violin melody of **Figure 4.9** – *Nkosi Sikeleli Africa* for comparison with the MPM output of **Figure 4.10**. The results were not as good, however. Only by lowering the frequency upper bound in the STFT was it possible to force the algorithm not to select some strong high harmonics over fundamental notes and get it as accurate as seen in **Figure 4.24**. It is important to note that MPM does not require any such restrictions. Based on these few tests, on the whole MPM seems to be the preferred melody detection algorithm, but it could be worth re-exploring the peak-picking part of the Phase Vocoder to see if any improvements may be made.

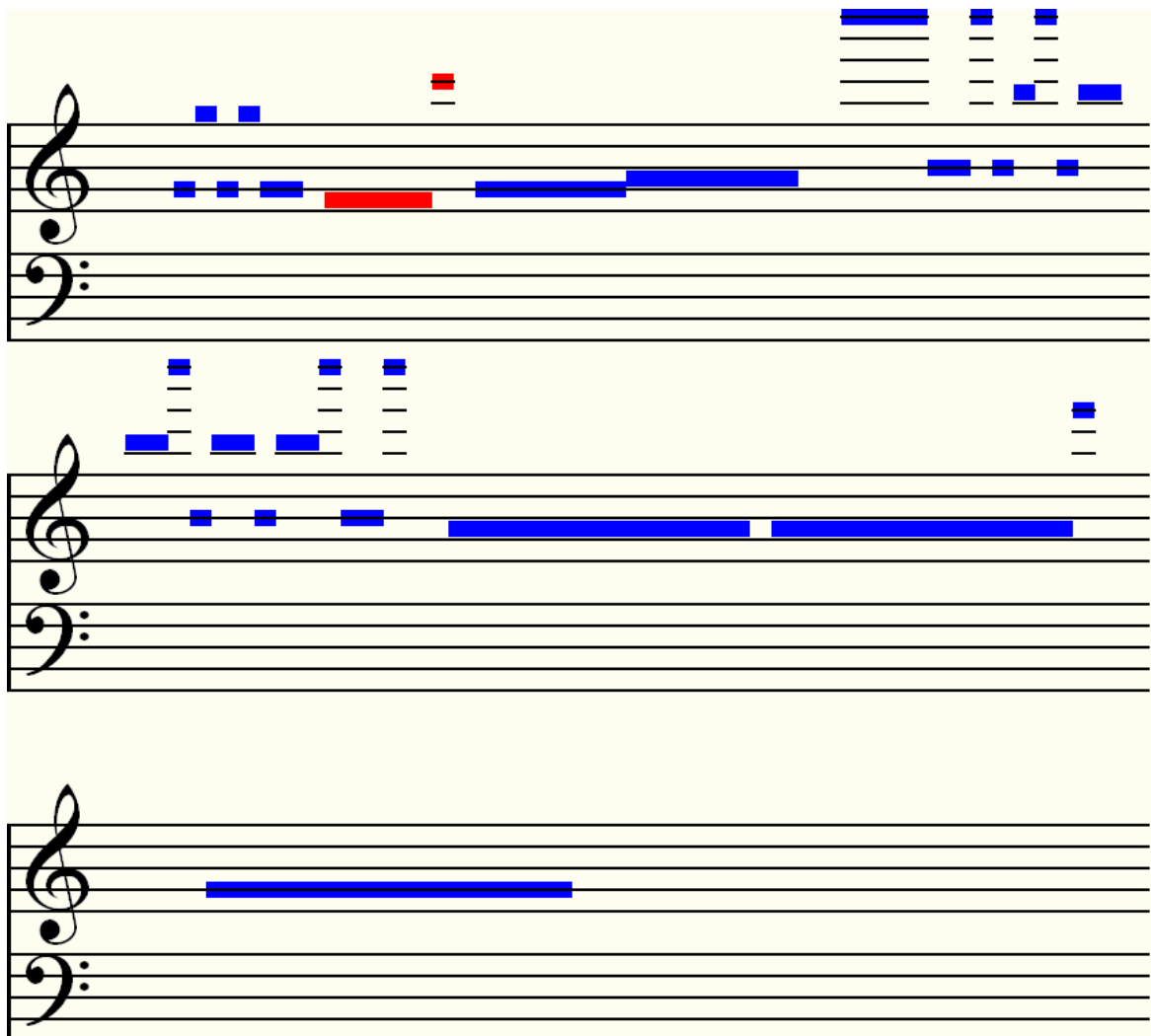


Figure 4.24 – Phase Vocoder pitch graph of first 2 bars of *Nkosi Sikeleli Africa* melody

successfully analyses, filters and shifts frequency spectra, although somewhat crudely, and for this particular example only. The following is a description of how the algorithm works:

Firstly, the manual/aural inspection of both the waveform and an STFT spectrogram (shown in **Figure 5.2**) was done in order to estimate the starting times of the notes and their frequencies. As suggested, these values were stored as constants or “magic numbers” in the program, to be used in dictating precisely when and where to look for fundamental notes and their harmonics. The STFT here used a window width of 2048 samples and no window function. With these settings, the resulting image is closest to the same output that Neubäcker demonstrates in the video, except that the latter seems to have been converted into pitch rather than frequency information, i.e. his graph is dyadic.

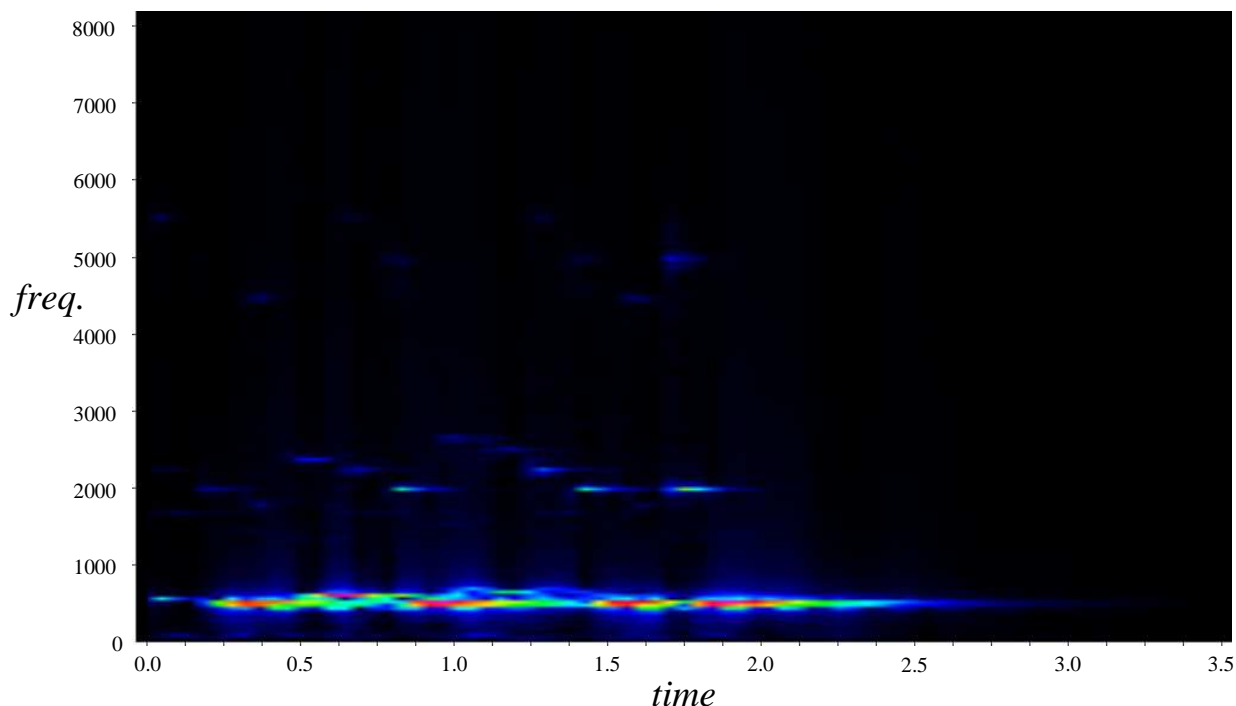


Figure 5.2 – Spectrogram of marimbaphone melody

Note	Pitch	Frequency (Hz)	Start Time (s)	Starting STFT Window
0	C#	277.58	0.04	0
1	B	247.30	0.21	4
2	A	220.31	0.37	7
3	D	294.09	0.51	10
4	C#	277.58	0.68	14
5	B	247.30	0.84	18
6	E	330.10	1.00	21
7	D#	311.57	1.15	24
8	C#	277.58	1.29	27
9	B	247.30	1.44	31
10	A	220.31	1.62	34
11	B	247.30	1.75	37

Table 5.1 – Frequencies and starting times of notes in marimbaphone melody

The manually estimated values of the starting points of each pitch and its frequency are shown in **Table 5.1** above. Each note was assumed to resonate for as long as the final note, which appeared to sustain for at least 1.67 seconds – about 36 windows. In fact, some of the notes do not ring for this long, because they are either dampened by other interfering frequencies or else they are repeated, as, for example, with the first note, C#, which is hit 3 times. However, if the notes are to be pitch-shifted, one would not want a note ending prematurely when it could, in theory, resonate longer. As an experiment, we attempt to repeat what Neubäcker might have done to detect the notes and then re-synthesize the result for aural comparison with the original piece.

Given that this spectrogram is yielded by a STFT rather than some other method, it is almost certain that Neubäcker used a Phase Vocoder to refine his detected frequencies so that he could determine which harmonic series they belonged to. Even if he did not, this technique certainly yields a more accurate extraction, since there is no guesswork involved. Thus a separate array of refined frequency information was created for each window of the transform.

The next part of the algorithm is the most difficult, and certainly the current method used here could do with some improvements. The general idea is that each of the frequencies identified from the transform in each window needs to be assigned to one of the twelve notes, or else rejected as background noise. The approach here attempts to assign a probability to each frequency for belonging to a particular note, based firstly on whether or not the frequency

Frequency in Window (Hz)	Closest Matching Harmonic (Hz)	Interval – Frequency Ratio (Hz)	Probability Freq. belongs to Note
292.01	$f_0 = 277.58$	1.052	0.91
2749.06	$f_{10} = 2775.79$	1.010	0.95
255.07	-	-	0.00
44.27	$f_{-5} = 46.26$	1.045	0.92
286.10	$f_0 = 277.58$	1.031	0.93
1102.17	$f_4 = 1110.32$	1.007	0.95
2752.05	$f_{10} = 2775.79$	1.009	0.95
217.48	-	-	0.00
220.10	-	-	0.00
324.22	-	-	0.00
42.04	-	-	0.00
824.85	$f_3 = 832.74$	1.010	0.95
214.52	-	-	0.00
1110.38	$f_4 = 1110.32$	1.000	0.96
2789.80	$f_{10} = 2775.79$	1.005	0.96
325.91	-	-	0.00
87.09	-	-	0.00
370.52	-	-	0.00
832.58	$f_3 = 832.74$	1.000	0.96
43.45	-	-	0.00
131.76	-	-	0.00
861.04	-	-	0.00
127.10	-	-	0.00

Table 5.2 – Frequency filtering for first note in first window

falls within the correct time range for the note and secondly how close is it to a frequency within in the harmonic series of the fundamental note, f_0 .

For example, **Table 5.2** shows all the frequencies identified in the second window of the transform, in order of their magnitudes, for which the first note is the only candidate for ownership (since the other notes have not yet started sounding at this point). These were all compared with the assumed frequency of the first note, 277.58Hz, and each of the frequencies in its harmonic series. A probability value was calculated, based on the interval between the closest matching harmonic and the frequency being analysed, as well as how much the note may have decayed by this window. If the interval between the frequency and the closest harmonic was larger than a semitone, the frequency was rejected as unrelated background noise and the probability value was set to zero. Note that the harmonic series examined also includes lower related harmonics, hence, for example, f_{-5} .

Again, by listening carefully to the first note alone, one can concur with the detection of f_{10} as a strong harmonic due to the particular timbre of the instrument, although the other harmonics are harder to hear. When a frequency such as this was re-detected, the note with the higher probability value was selected. **Figure 5.3** shows what could be described as a probability distribution spectrogram for frequencies which belong to the final note. As can be seen, the lower frequencies are more difficult to separate, and most of these have equal probability of belonging to some of the other notes too.

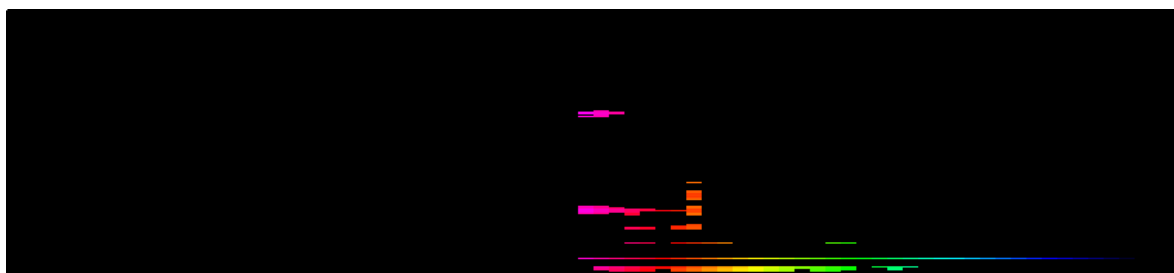


Figure 5.3 – Probability distribution spectrogram for frequencies belonging to 12th note

With the complete set of probabilities per frequency bin, per window, per note, separation of the notes may now be attempted and the pitch of each individual note may thus be adjusted. This was tried with the fourth note, which was shifted up three semitones*. In order to do the pitch shift, each of the detected frequencies belonging to the note was multiplied by two to the power of three twelfths (see Chapter 3 for an explanation for this particular factor) and a new bin was assigned. The phase, θ , in every k^{th} window of each new frequency bin, j , was calculated by the following:

$$\theta_k^j = \theta_{k-1}^j + 2\pi\nu^j \Delta t.$$

where ν^j is the frequency in the j^{th} bin and Δt is the difference in time from one window to the next. Real and imaginary parts of the affected Fourier coefficients were recalculated thus:

$$\nu_{\text{Re}}^j = |\nu| \cos \theta, \quad \nu_{\text{Im}}^j = |\nu| \sin \theta.$$

* The result of this may be heard in Sound\marimbaphone_shifted.wav on the project CD

To re-synthesize the output wave, an inverse Fourier transform of each window was taken, i.e. an inverse STFT.

The slight scratchiness of the resulting signal is most likely due to the lack of proper calculation of the new phases for the shifted peaks, and also the magnitudes of the bins from which the notes are shifted should probably not be set to zero. The phases of surrounding bins should also be re-adjusted. Also, it would be better not to include note decay as a factor in calculating probabilities, although knowing the attack-decay-sustain-release envelope for a particular instrument is very helpful. While fixing these issues and other possibly naïve methods within the algorithm constructed here would certainly improve the quality of the output, the result clearly demonstrates that it is indeed possible to adjust overlapping spectra and preserve the timbre of the instrument as well as other frequencies using this kind of approach.

5.1.2 AudioScore for Sibelius

Although the algorithms which drive *AudioScore* are, of course, also a trade secret, there are a few clues as to how its pre-processing might work in the specifications listed on the *Sibelius* web site [Sibelius09W]. For most, *Sibelius* is the music publishing application of choice to plug *AudioScore* into, and so this chapter also includes some screenshots of output exported to *Sibelius* and transformed into scores, as well as some pitch graphs from *AudioScore* itself.

The software specification details of interest are the following:

- Opens polyphonic MP3s and CD tracks and locates up to 16 notes/instruments playing at a time
- Pitch recognition range F_0 to C_8 (22Hz to 4186Hz)
- Timing accuracy down to $1/86^{\text{th}}$ of a second
- Pitch accuracy 0.3Hz (about $1/100^{\text{th}}$ of a semitone at A_4)

Firstly, the clue that the program reads CD tracks and the given time resolution mean that the algorithm expects a standard sample rate of 44.1kHz as input, and that the frequencies are measured in time windows of duration $\Lambda \approx \frac{1}{86}$ seconds. Thus, we can calculate the number of samples in each window by the formulae discussed in Chapter 2:

$$\begin{aligned} N &= 2B\Lambda \\ &\approx 44100/86 \\ &\approx 512.79. \end{aligned}$$

The fact that this result is close to being a power of two is surely not a coincidence. It suggests that the pre-processing algorithm of choice is, again, a Short Time Fourier Transform with a window width of 512 samples. The claimed pitch accuracy of 0.3Hz also indicates that a Phase Vocoder algorithm has been used, since with an ordinary STFT, the value for Δv per window would be:

$$\begin{aligned} \Delta v &= 2B/W \\ &= 44100/512 \\ &= 86.13. \end{aligned}$$

This would be an unacceptable resolution for low to middle range musical pitch identification, since at A₄, which is a mid-range pitch, a semitone is a change of about 26Hz. Instead of attempting to work out further how pitches are chosen, an investigation of the abilities and limitations of this program is perhaps more useful.

Firstly, the DNA marimbaphone example was opened and analysed. The resulting pitch graph, shown in **Figure 5.4**, while not entirely accurate, at least demonstrates that the software is capable of determining most of the correct fundamental notes, even though their location in time is somewhat confused.



Figure 5.4 – AudioScore pitch graph of marimbaphone melody

When imported into *Sibelius*, the result is not very satisfactory, and would require a lot of patch-editing on the part of the user to get it looking like **Figure 5.1**. The *Sibelius* score derived from *AudioScore*'s exported data and unaltered is shown in **Figure 5.5** below. The time signature was chosen manually in *AudioScore* and not detected.

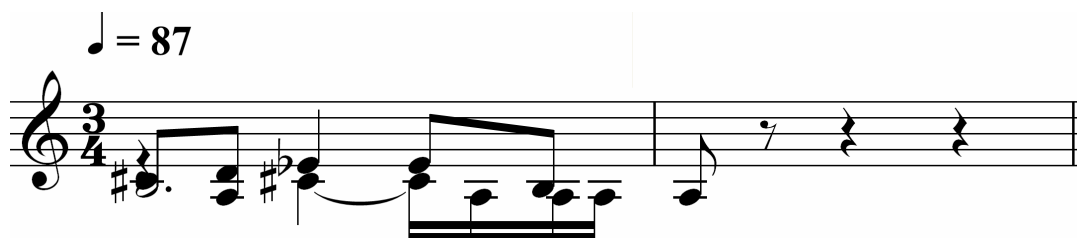


Figure 5.5 – Marimbaphone melody imported into Sibelius from AudioScore

The indication at the top of the score, ♩ = 87, is called a *metronome mark*. The metronome was originally a clockwork device, invented by Johann Maelzel at the beginning of the 19th century. It had a sliding weight which could be set at the level of the number indicated. When its pendulum was set in motion, it would tick precisely that number of times per minute. Thus the metronome mark is an indication of speed or *tempo*. In this case, the beats specified by the time signature and the metronome mark are crotchets, or quarter notes. If there are 87 of them in one minute, this means that each crotchet beat will last 0.69 seconds. Given that there are four sixteenth notes in every crotchet beat, then the length of each note has been estimated to be 0.17 seconds. The actual length of the notes, given **Table 5.1**, is on average very close to that, and so this is a good tempo estimate.

The software performed adequately with a string quartet arrangement (four voices) of the first two bars of the main demonstration tune, *Nkosi Sikeleli Africa**. The actual score, obtained by doing an aural / manual transcription (with 100% confidence of its accuracy) is shown in **Figure 5.6**, while the transcription from *AudioScore* into *Sibelius* is in **Figure 5.7**. The latter

* Listen to Sound\NSA\Quartet - 2 bars.wav on the project CD

From the pitch graph in **Figure 5.8** it appears that some of the errors happen after the frequency analysis stage, for example the final bass note, G2, which is present in the graph, is absent in the final transcription. Going by its lighter shade of green, this is possibly because it was not as clearly defined and detected as the other pitches and so it was rejected as being noise.

On the whole *AudioScore* seems to be an adequately sturdy pitch detector, but could do with some improvements with regards to correct note time location and other post-processing issues. These improvements are vital if it is to become a useful automatic transcriber, since the amount of work needed to correct its output when exporting to *Sibelius* is the same as, if not more than, doing a manual transcription. Furthermore, unless the user is a skilled music transcriber, mistakes made by the program, such as incorrect timings and choosing harmonics over fundamentals as pitches, are likely to be overlooked.

For further theoretical reading about other algorithms and ideas not presented here, see [Cont07L] which presents a real-time multiple pitch recognition method, as well as the excellent three papers by Yipeng Li and DeLiang Wang from Ohio State University [Li07L], [Li08L] and [Li09L] on pitch detection and separation.

5.2 Introducing Wavelets

Much of the basic theory and mathematics in this and the following sections is drawn from a few very good introductory papers on the subject of wavelets, [Graps95L], [Strang89L] and [Strang94L], which are highly recommended for entry level reading. Ingrid Daubechies's book, *Ten Lectures on Wavelets* [Daubechies92L], parts of which were available via a Google books preview, was also used as a primary source, as well as [Olver05W], which is perhaps the clearest explanation of Fourier and wavelet analysis yet. Lastly, Robi Polikar's web-based tutorial, [Polikar03W], provides an excellent broad overview of wavelets and signal processing in general, despite its lack of official publication.

5.2.1 What is a Wavelet?

In 1940, a seismologist by the name of Norman Ricker coined the word *wavelet* to describe a short, travelling wave caused by a sharp seismic disturbance [Ricker40L]. Ricker noted that these wavelets changed shape and broadened as they moved away from their source through different media in the earth's crust, and wanted to find a mathematical explanation for this phenomenon.

In the current context of signal processing, wavelets were first given their formal definition in the early 1980s by engineers, Jean Morlet and Alex Grossman. Morlet also needed to describe a set of short, finite, oscillatory functions which could be broadened or dilated, but which, unlike Ricker's original wavelets, did not change their basic form. Grossman and Morlet first called them "*ondelettes de forme constante*" – "wavelets of constant shape" in their groundbreaking paper [Grossman84L] in 1984, which revolutionized their field by providing a new way of describing a signal using these wavelets as basis functions. For the first time, an alternative to the Fourier transform was available. Firstly though, to understand what is meant by *basis function*, we need to go back a little further in History to the work of Hungarian mathematician, Alfréd Haar.

5.2.2 The Haar Function

In his paper, written almost a century ago [Haar10L], Haar explored what are known as *orthogonal function systems*. Without going too much into the mathematics, these are infinite families of functions, $\psi_n(t)$, which exhibit *orthogonality* on a Hilbert space, in other words the inner product, or Lebesgue integral of the product, of any two distinct functions is zero:

$$\langle \psi_p(t), \psi_q(t) \rangle = 0, \quad (p \neq q, p, q = 0, 1, 2, \dots)$$

or

$$\int_a^b \psi_p(t) \psi_q(t) dt = 0.$$

Haar also includes in his definition the property that the integral of the square of each function should be equal to one – this means that the functions are *orthonormal*:

$$\int_a^b (\psi_p(t))^2 dt = 1.$$

A *complete orthogonal function system* is then a set of orthogonal and orthonormal functions which satisfy a *completeness relation*. That is to say, on an interval (a, b) , for any square-integrable function, f , a series may be written in terms of the functions:

$$\int_a^b (f(t))^2 dt = \left[\int_a^b f(t) \psi_0(t) dt \right]^2 + \left[\int_a^b f(t) \psi_1(t) dt \right]^2 + \dots$$

The most well-known orthogonal function system is the set of sine and cosine functions on the interval $(-\pi, \pi)$. These are the components of Fourier series, which may be used to expand any periodic square-integrable function, f [Fourier22L, Wolfram09W]:

$$f(t) = a_0 + \sum_{n=1}^{\infty} [a_n \cos(nt) + b_n \sin(nt)],$$

where

$$a_0 = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) dt,$$

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cos(nt) dt,$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \sin(nt) dt.$$

The sines and cosines are known as the *basis functions* for the Fourier transform, which is derived from this series. Haar's study led him to construct a new set of orthogonal basis functions, which were later to become recognized as the first wavelets [Daubechies92L].

Haar wavelets, the name by which Haar's new basis functions are now known, may best be understood initially in terms of vectors and matrices on the simplest of Hilbert spaces – the Euclidean plane. Consider the vectors $(1, 1)$ and $(1, -1)$. If we draw these on the plane, as in **Figure 5.9**, we can easily see that they are orthogonal, since they are at right angles to each other. To confirm the orthogonality of these two vectors, we should check that their dot product (the inner product in Euclidean space) is zero, and this is indeed the case:

$$\begin{aligned} &(1, 1) \cdot (1, -1) \\ &= 1 \times 1 + 1 \times -1 = 0. \end{aligned}$$

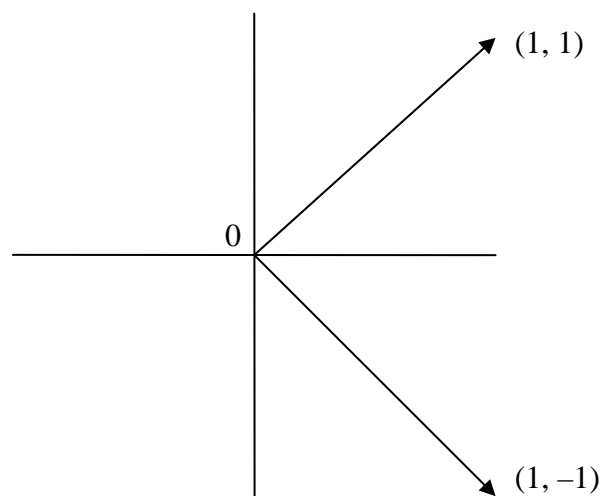


Figure 5.9 – Haar component vectors drawn on the Cartesian plane

In the above graph, the horizontal and vertical axes are the generated vectors $(1, 0)$ and $(0, 1)$ respectively. These vectors are likewise orthogonal, and they define the Cartesian coordinate system. Just as we can represent all points on the plane in terms of these two vectors, we can also represent the same points in terms of the new diagonal vectors, $(1, 1)$ and $(1, -1)$. Such vectors are called *basis vectors* [Strang94L] and are the simplest examples of basis functions, being in only two dimensions. We can convert between the Cartesian coordinate system and the 2D Haar space by constructing a transformation matrix, H_2 , from the two vectors:

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Consider a very simple discrete signal comprising only two samples, with amplitudes of say 4 and 2. This signal can be represented by a point in Euclidean space: in terms of the axes vectors, it is $4 \times (1, 0) + 2 \times (0, 1)$. Using the two-dimensional Haar basis vectors instead, the signal is, equivalently, $3 \times (1, 1) + 1 \times (1, -1)$. In terms of matrix multiplication, the transform is:

$$\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}.$$

The first basis column vector, $(1, 1)$, may be thought of as a steady signal with a constant amplitude of 1, whereas the second basis column vector, $(1, -1)$, is a square wave – half the signal is up and the other half is down, as shown in **Figure 5.10**. The two functions which define these signals over the interval $(0, 1]$ are:

$$\phi(t) = \begin{cases} 1, & 0 < t \leq 1 \\ 0, & \text{otherwise,} \end{cases} \quad \text{and} \quad \psi(t) = \begin{cases} 1, & 0 < t \leq \frac{1}{2} \\ -1, & \frac{1}{2} < t \leq 1 \\ 0, & \text{otherwise.} \end{cases}$$

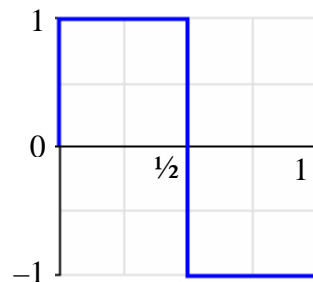


Figure 5.10 – The Haar mother wavelet, $\psi(t)$, as a square function

$\phi(t)$ is called the *father wavelet* and $\psi(t)$ is known as the *mother wavelet*, sometimes written $\psi_0(t)$ to properly differentiate it from other wavelets. The reasons for this nomenclature will become apparent as we see how all other wavelets in successive generations of the Haar family are derived from these two.

For longer signals, we need more dimensions. Consider, for example, one with four samples $(4, 2, 5, 5)$. Extending the standard basis vectors in two-dimensional space to four, we can express the signal as $4 \times (1, 0, 0, 0) + 2 \times (0, 1, 0, 0) + 5 \times (0, 0, 1, 0) + 5 \times (0, 0, 0, 1)$. In order to get the Haar basis functions in four dimensions, we first expand the existing 2-point vectors to 4-points, retaining their respective constant and square wave shapes. Thus $(1, 1)$ becomes $(1, 1, 1, 1)$ and $(1, -1)$ becomes $(1, 1, -1, -1)$. Then, the other two vectors are obtained by shifting the $(1, -1)$ vector to the first half of an otherwise null vector: $(1, -1, 0, 0)$, and then to the second half: $(0, 0, 1, -1)$. Thus, the four-dimensional Haar matrix is:

$$H_4 = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & 0 & -1 \end{bmatrix}.$$

The corresponding square wave graphs of the last two bases are shown in **Figure 5.11**. Note that the interval is still over $(0, 1]$ and so the wavelets have effectively been scaled.

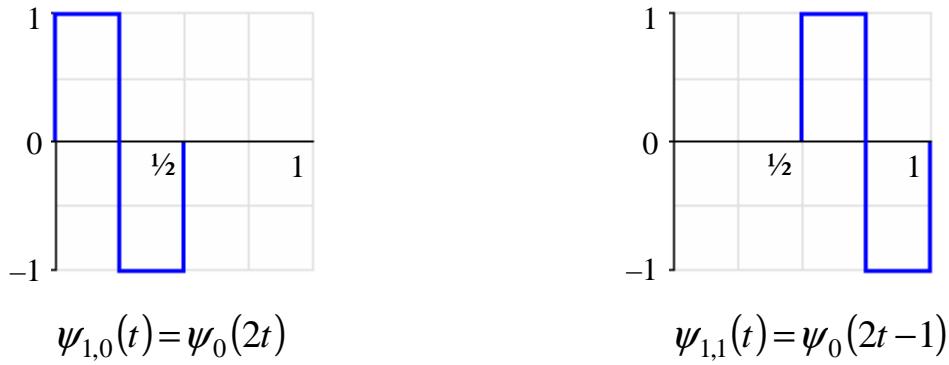


Figure 5.11 – Haar wavelets at first scale ($s = 1$)

The function which squashes and shifts the mother wavelet, $\psi_0(t)$, to generate the next set at each scale is defined by:

$$\psi_{s,\tau}(t) = \sqrt{2^s} \psi_0(2^s t - \tau), \quad [5.1]$$

where s is the scale and τ is the shift, which, for audio signals, may be thought of as a time delay before the onset of the scaled wavelet. The mother wavelet, ψ , may also be expressed in terms of the father wavelet, ϕ , similarly by compression and translation:

$$\psi(t) = \phi(2t) - \phi(2t - 1). \quad [5.2]$$

For this reason, the father wavelet is also known as the *scaling function*. We can confirm [5.2] holds, for example, for $t = 1/2$, $t = 1/4$ and $t = 3/4$:

$$\begin{aligned} \psi\left(\frac{1}{2}\right) &= \phi(1) - \phi(0) \\ &= 1 - 0 \\ &= 1. \end{aligned}$$

$$\begin{aligned} \psi\left(\frac{1}{4}\right) &= \phi\left(\frac{1}{2}\right) - \phi\left(-\frac{1}{2}\right) \\ &= 1 - 0 \\ &= 1. \end{aligned}$$

$$\begin{aligned} \psi\left(\frac{3}{4}\right) &= \phi\left(\frac{3}{2}\right) - \phi\left(\frac{1}{2}\right) \\ &= 0 - 1 \\ &= -1. \end{aligned}$$

The normalizing factor, $\sqrt{2^s}$, in [5.1] is chosen so that over $(0, 1]$ the orthonormal property of the wavelet still holds at each scale. With the Haar wavelets, the constant is easy to derive since the Lebesgue Integral – the area under the graph – of the mother wavelet function is simply the sum of the two rectangles. This is $1/2 + 1/2 = 1$ for the mother wavelet, and

subsequently the area halves at each scale. At a scale of 3, for example, the area is $1/2^3$. The square root of this multiplied by the square root of its inverse is 1, as it should be. The example signal, (4, 2, 5, 5), transforms into (4, -1, 1, 0) in terms of the new basis vectors. The next sections of the chapter will reveal how these coefficients are determined, i.e., how the Haar transform is computed, so that the transformed signal, when multiplied by the transformation matrix yields the original signal:

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & -1 & 0 \\ 1 & -1 & 0 & 1 \\ 1 & -1 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ -1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \\ 5 \\ 5 \end{bmatrix}. \quad [5.3]$$

We can continue scaling and shifting the basis vectors in order to generate matrices and thus basis functions for higher dimensions indefinitely. For eight dimensions, the Haar matrix is:

$$H_8 = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & -1 & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & -1 & 0 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & -1 & 0 & 1 & 0 & 0 & -1 & 0 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & -1 \end{bmatrix}.$$

Now there are four positions of the wavelet at the new scale, $s = 2$, and thus four values of τ . **Figure 5.12** shows these four new wavelets as square waves.

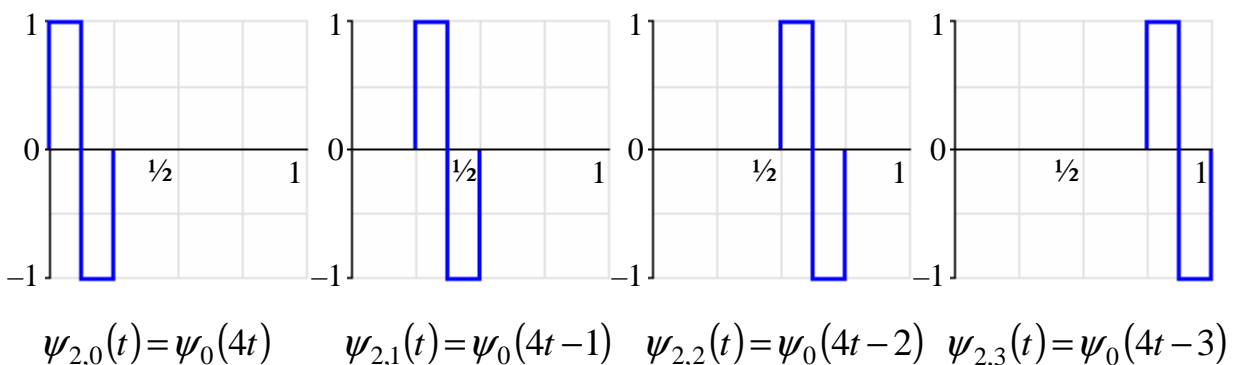


Figure 5.12 – Haar wavelets at second scale ($s = 2$)

5.3 The Discrete Wavelet Transform

Assuming, as Haar tells us, that we can carry on applying the scaling function to yield an infinite orthogonal function system, we can thus express a function, f , by the expansion:

$$f(t) = a\phi(t) + b\psi(t) + \sum_{s=1}^{\infty} \sum_{\tau=0}^{2^s-1} c_{s,\tau} \psi_{s,\tau}(t), \quad [5.4]$$

where a , b , $c_{1,0}$, $c_{1,1}$... $c_{s,\tau}$ are the wavelet coefficients and $\psi_{s,\tau}$ is the scaled and shifted wavelet function and s and τ are now indexes rather than values. For discrete signals, we need to limit s so that 2^s is the number of samples in the signal, thus, as with the Fourier transform, the number of samples (per scale) will always be a power of two. The list of wavelet coefficients for scales 0 to $S - 1$ form the discrete wavelet transform (DWT) of f_n . As seen in the first column of the Haar matrices, the scaling function, $\phi(t)$, is a constant value of one at all times in the interval (0, 1], and so sometimes $\phi(t)$ is left out of the equation when talking about the Haar transform in particular. Also, the mother wavelet, $\psi(t)$, is sometimes included in the summation, which then goes from $s = 0$. In the Haar case, the signal may be decomposed by:

$$f_n = c_0 + \sum_{s=0}^{S-1} \sum_{\tau=0}^{2^s-1} c_{s,\tau} \psi_{s,\tau}^{Haar}.$$

Under a different light, the wavelet transform is a measure of correlation between a signal and another signal, the wavelet, at different time shifts. At each scale, the wavelet is compressed to half of its previous wavelength, and therefore its frequency (if we were to continue the signal for more than just one oscillation) doubles. So in actual fact, the coefficients of the transform are telling us how much of a certain frequency exists in the signal *and when* it exists. The output of the transform is therefore in the time-frequency (or rather time-scale) domain and thus, like the STFT, it is usually presented in the form of a spectrogram or a 3D graph.

The wavelet transform is what is known as a *multi-resolution analysis* technique since it allows analysis of a signal at different frequencies with different time resolutions [Polikar03W]. At lower scales (or frequencies) we can only shift the wavelet a few times, but can obtain a better frequency resolution, whereas at higher scales (frequencies) the time resolution is much better and the frequencies are less well localized. On this point, the transform may be compared to the STFT by drawing *Heisenberg boxes* [Graps95L]. It was mentioned previously, in Chapter 2 that attempting to determine both the precise time and the exact instantaneous frequency of a signal at that time is limited by the Heisenberg Uncertainty Principle, i.e. one cannot measure both with the same arbitrary precision. Let us first look at the range of frequencies which may be measured by a Short Time Fourier Transform per window, given a certain window width. **Figure 5.13** illustrates how the time and frequency resolution, defined by a particular, fixed window width, is always the same for every row of the transform in which it is used, regardless of the window's location:

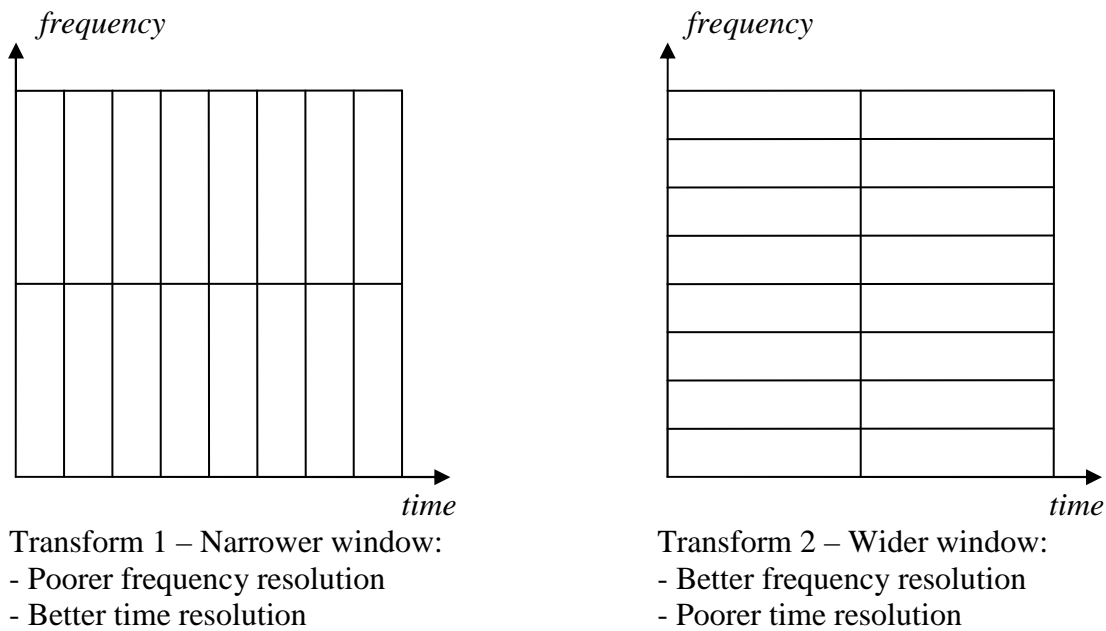


Figure 5.13 – Heisenberg boxes for the STFT

For the discrete wavelet transform, however, the frequency spacing increases per row – in fact it doubles at every scale – and also the time spacing or window width halves. This is illustrated in **Figure 5.14**. As can be seen, the area of the Heisenberg boxes, $\Delta t \cdot \Delta \nu$, remains the same. This represents the Heisenberg limit, which is $\frac{1}{4\pi}$ [Polikar03W].

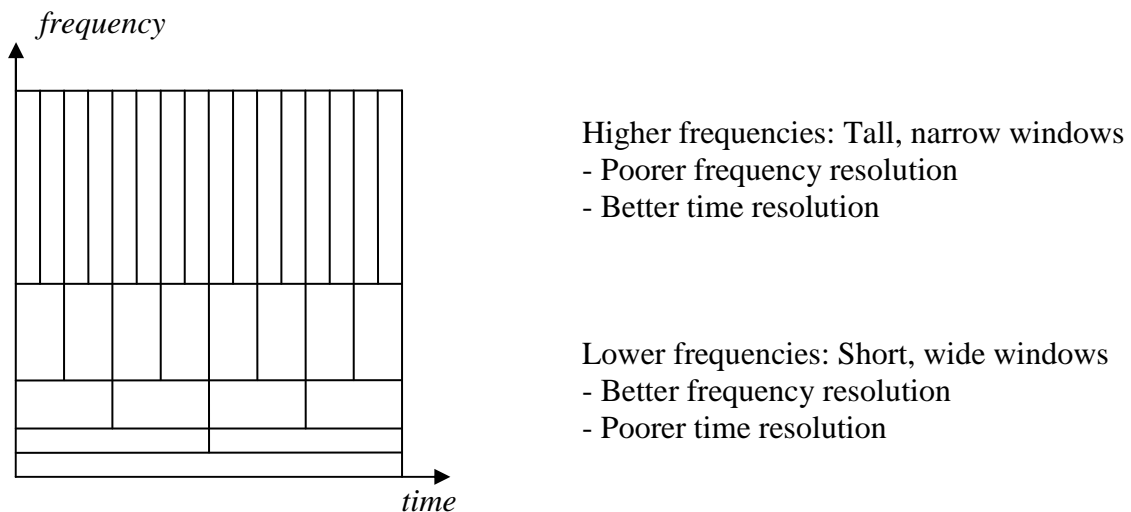


Figure 5.14 – Heisenberg boxes for the DWT

The DWT is therefore more flexible than the STFT, since it can analyse higher frequencies with good time resolution but poor frequency resolution and vice versa. This makes it suitable for analysing musical sound signals in particular, since we are most interested in the mid-range frequencies and not so much the extremities. Before looking at some output of the Haar wavelet transform, we should first examine the fast algorithm used to compute it, and also look at one other discrete wavelet which uses a different set of basis vectors.

5.4 The Fast Haar and Daubechies Transforms

Revisiting the example signal from section 5.2, which was (4, 2, 5, 5), the method by which the Haar wavelet coefficients, (4, -1, 1, 0), are calculated is quite simple, as will be seen. The inverse of H_4 is obtained by scaling its column vectors by a factor of 2^{s-2} and transposing them. Multiplying both sides of [5.3] by $(H_4)^{-1}$ yields [Strang89L]:

$$\begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\ \frac{1}{2} & -\frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & -\frac{1}{2} \end{bmatrix} \cdot \begin{bmatrix} 4 \\ 2 \\ 5 \\ 5 \end{bmatrix} = \begin{bmatrix} 4 \\ -1 \\ 1 \\ 0 \end{bmatrix}.$$

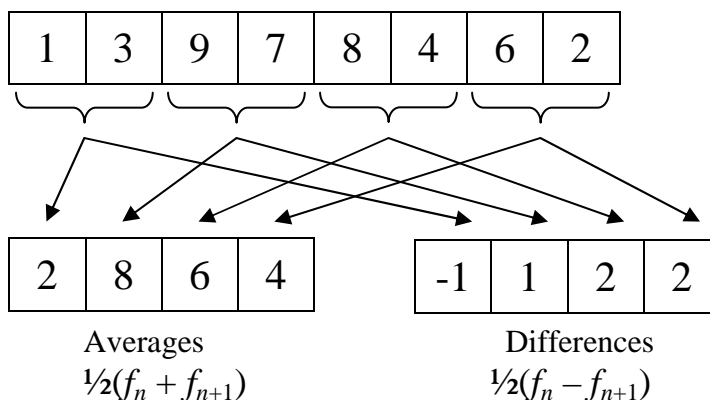
Normally, this multiplication requires N^2 steps, where N is the number of samples. However if we examine more precisely what is going on at each stage, a recursive pattern emerges which allows the same calculation to be done in only $2N$ steps. The following example outlines how the fast Haar wavelet transform works to achieve this. This algorithm has been constructed from the description given in [Strang94L].

5.4.1 The Fast Haar Transform

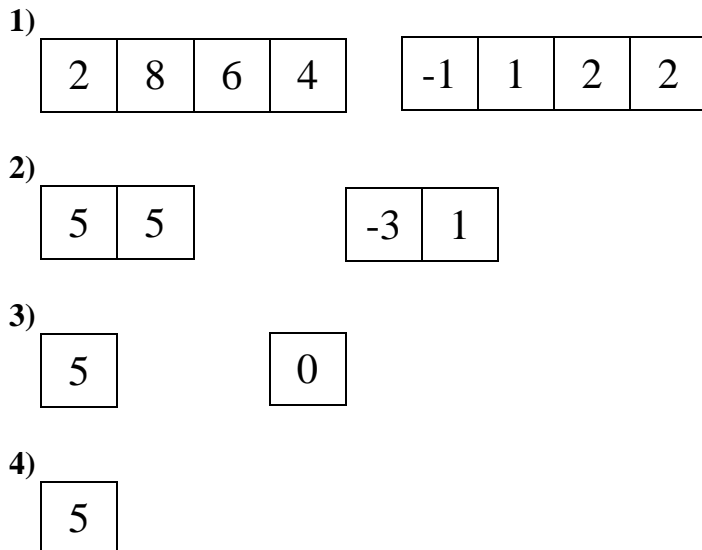
Consider the following array of data, an 8-sample signal:

1	3	9	7	8	4	6	2
---	---	---	---	---	---	---	---

We begin by calculating the set of averages and the set of half the differences for each pair of samples. The averages go on the left and the differences go on the right, as follows:



The process is repeated on the set of averages recursively, until there is only one average left – this sample is then the average of the entire data set, or the data at its lowest “resolution”. Thus, continuing with the same example, we have:



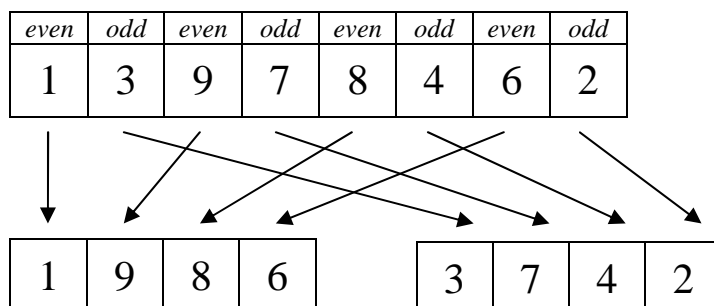
The set of wavelet coefficients is then the last average (a in equation [5.4]), and all the sets of differences (b and $c_{s,\tau}$ in equation [5.4]), collected by scanning from left to right back up through the differences. Note that the number of scales or resolutions is one less than the number of steps required to complete the transform, the final result of which is shown below:

5	0	-3	1	-1	1	2	2
---	---	----	---	----	---	---	---

5.4.2 The Haar Wavelet Lifting Scheme Algorithm

Looking at the above method from a programmer's point of view, it would be desirable to transform the data without having to create a temporary array of the same size, which for a sound signal, could be very large. This does seem necessary at first glance, since some calculations would overwrite values that have not yet been read. However, the algorithm described here shows how it is possible to use the same array for both input and output. Although it was eventually decided that the algorithm would not be implemented in Wave Processor, it is worth examining it anyway.

The lifting scheme begins, again, by splitting the data in half. Again, the number of data elements must be a power of 2. This time the split is not down the middle, but into odd and even indexed samples, moving the odd samples to the end of the array and leaving the even ones at the beginning. Using the same data set, this is demonstrated below:



This involves quite a lot of array element switching. After experimenting with some code originally written by Ian Kaplan [Kaplan03W] it was realised that the time taken to perform the re-shuffling is very costly with large data sets, and in fact the duration of the entire operation grows exponentially with an increasing array size. This unfortunately defeats the whole purpose of creating a fast algorithm.

The next step is to calculate the set of differences. This is done, accordingly, by subtracting the odd samples from the even samples (and dividing by 2) and storing the results in the “odds” array:

1	9	8	6
---	---	---	---

-1	1	2	2
----	---	---	---

Finally, the averages are calculated. Since the odd samples have already been overwritten, some algebra is needed in order to recover the original values. Recall from the fast Haar transform algorithm that the averages are calculated from each pair by:

$$x = \frac{1}{2}(x_1 + x_2) \text{ or } \textit{even}' = \frac{1}{2}(\textit{even} + \textit{odd}). \quad [5.5]$$

Now each new *odd* is given by:

$$\textit{odd}' = \frac{1}{2}(\textit{even} - \textit{odd}). \quad [5.6]$$

Putting the equation in terms of *odd* (the original) we get:

$$\textit{odd} = \textit{even} - 2\textit{odd}'. \quad [5.7]$$

Substituting this back into equation [5.5], this becomes:

$$\begin{aligned} \textit{even}' &= \frac{1}{2}(\textit{even} + [\textit{even} - 2\textit{odd}']) \\ &= \frac{1}{2}(2\textit{even} - 2\textit{odd}') \\ &= \textit{even} - \textit{odd}'. \end{aligned} \quad [5.8]$$

So, in other words, all that needs to be done is to subtract the new odd samples from the current even ones to get the new even samples. It can be seen that the resulting pair of arrays is now identical to those in step 1) of the fast Haar transform in the preceding section:

2	8	6	4
---	---	---	---

-1	1	2	2
----	---	---	---

As with the fast Haar transform, the lifting scheme is repeated recursively on the new even samples until there is only one left (the gross average). The reason why this algorithm is called the “lifting” scheme may be illustrated in the diagram in **Figure 5.15**. As the data is split recursively into evens and odds, the even averages get “lifted” and processed at the next scale, while the odd differences (coefficients) are output at the bottom.

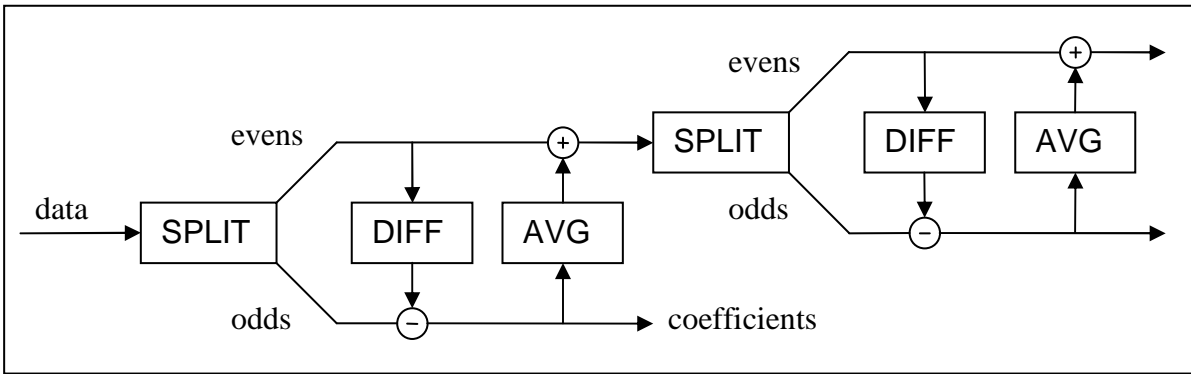


Figure 5.15 – Lifting Scheme diagram

5.4.3 Daubechies Wavelets

The Daubechies family of wavelets, discovered by the Belgian mathematician, Ingrid Daubechies in 1988 [Strang94L], are, like the Haar wavelet, orthogonal basis functions with increasing complexity. They were the first family of wavelets to have *compact support*, which means that the interval on which they are defined is bounded, but an infinite number of points in the interval describes their functions. This is also basically an engineer’s way of saying that they are easy to implement, practically, as a digital filter – almost as easy as the Haar wavelet. More simply put, they are easy to use because the formula for each of the Daubechies father wavelets does not need to be known, only a set of “magic numbers” [Strang94L] – the scaling function coefficients.

The Haar wavelet is actually the first in the Daubechies series, code-named D2, and the next is D4. The ‘2’ and ‘4’ simply refer to the dimension of the father wavelet and hence the number of scaling function coefficients – Haar (D2) has two, (1, 1), D4 has four and D20 has twenty, and so on. In each case, the mother wavelet, and subsequently the child wavelets, are derived from the scaling function by compressing and shifting it, as in equation [5.1]. Given that there may be more than just two scaling function coefficients, the general form of the equation which expresses the mother wavelet in terms of the father is:

$$\begin{aligned}\psi(t) &= h_n\phi(2t) - h_{n-1}\phi(2t-1) + h_{n-2}\phi(2t-2) - \dots \pm h_0\phi(2t-n) \\ &= \sum_{k=0}^n (-1)^k h_{n-k}\phi(2t-n),\end{aligned}\quad [5.9]$$

where h_n are the function coefficients of a $(n + 1)$ -dimensional father wavelet. The equation to which the father wavelet must be a solution is called a *dilation equation*, and has a similar form:

$$\begin{aligned}\phi(t) &= h_0\phi(2t) + h_1\phi(2t-1) + \dots + h_n\phi(2t-n) \\ &= \sum_{k=0}^n h_k\phi(2t-k).\end{aligned}\quad [5.10]$$

It is fairly trivial to see that the Haar scaling function, $\phi(t) = 1$, is a 1-dimensional solution to the above:

$$\phi(t) = \phi(2t) + \phi(2t - 1).$$

so [5.10] is satisfied if $h_0 = h_1 = 1$ and $h_k = 0$ otherwise [Ewer10L].

The brilliant discovery made by Daubechies was that one can indeed find scaling functions which solve [5.10] with *any* arbitrary *even* number of coefficients. Odd numbers of coefficients do not yield orthogonal functions, and so cannot be used. Skipping over the mathematical proofs here, Daubechies found that the scaling function coefficients for D4 had to satisfy the following conditions so that they rendered orthogonal wavelet bases:

$$\begin{aligned} h_0 + h_1 + h_2 + h_3 &= 2, \\ h_0^2 + h_1^2 + h_2^2 + h_3^2 &= 2, \\ h_0h_2 + h_1h_3 &= 0. \end{aligned}$$

Furthermore, the following conditions must also apply [Ewer10L], [Strang94L]:

$$\begin{aligned} h_0 - h_1 + h_2 - h_3 &= 0, \\ h_3 - 2h_2 + 3h_1 - 4h_0 &= 0. \end{aligned}$$

These equations yield the following values for the coefficients:

$$\begin{aligned} h_0 &= \frac{1}{4}(1 + \sqrt{3}), \\ h_1 &= \frac{1}{4}(3 + \sqrt{3}), \\ h_2 &= \frac{1}{4}(3 - \sqrt{3}), \\ h_3 &= \frac{1}{4}(1 - \sqrt{3}). \end{aligned}$$

Note that another solution set exists where the signs in front of the square roots are reversed. Usually this second is discarded because otherwise $h_0 < 0$, which is inconvenient [Ewer10L].

From equation [5.10], the D4 scaling function, which is bounded on the interval (0, 3) is then given by [Strang94L]:

$$\phi(t) = \frac{(1+\sqrt{3})}{4} \phi(2t) + \frac{(3+\sqrt{3})}{4} \phi(2t-1) + \frac{(3-\sqrt{3})}{4} \phi(2t-2) + \frac{(1-\sqrt{3})}{4} \phi(2t-3).$$

while the D4 mother wavelet is:

$$\psi(t) = \frac{(1-\sqrt{3})}{4} \phi(2t) - \frac{(3-\sqrt{3})}{4} \phi(2t-1) + \frac{(3+\sqrt{3})}{4} \phi(2t-2) - \frac{(1+\sqrt{3})}{4} \phi(2t-3).$$

The appearance of the scaling function on the right hand side of these equations makes it somewhat difficult to draw the graph of these two functions. However, this may be done

recursively by substituting values of t at increasing levels of detail, after finding the points at the integer values, $t = 1$ and $t = 2$. Tackling the scaling function first, we have:

$$\begin{aligned}\phi(1) &= h_1\phi(1) + h_0\phi(2). \\ \phi(2) &= h_3\phi(1) + h_2\phi(2).\end{aligned}$$

Note that the other points are zero since the function does not operate outside the interval $(0, 3)$, i.e. $0 < t < 3$. To solve for ϕ , we must find the eigenvalues, λ_k , of a matrix comprising the four coefficients, such that:

$$\begin{bmatrix} h_1 & h_0 \\ h_3 & h_2 \end{bmatrix} \cdot \begin{bmatrix} \phi(1) \\ \phi(2) \end{bmatrix} = \lambda_k \begin{bmatrix} \phi(1) \\ \phi(2) \end{bmatrix}.$$

This yields the values $\lambda_1 = 1$ and $\lambda_2 = 1/2^*$. The components of the normalized eigenvector associated with λ_1 gives us the values of $1/2(1+\sqrt{3})$ and $1/2(1+\sqrt{3})$ for $\phi(1)$ and $\phi(2)$ respectively*. To get these components, we need to make the magnitude of the eigenvector $\sqrt{2}$ – the prescribed normalization factor at each scale. All other points in the function may be found by subdividing the intervals recursively, i.e., next we find $\phi(\frac{1}{2})$, $\phi(\frac{3}{2})$, and $\phi(\frac{5}{2})$, then the quarter integers and so on.

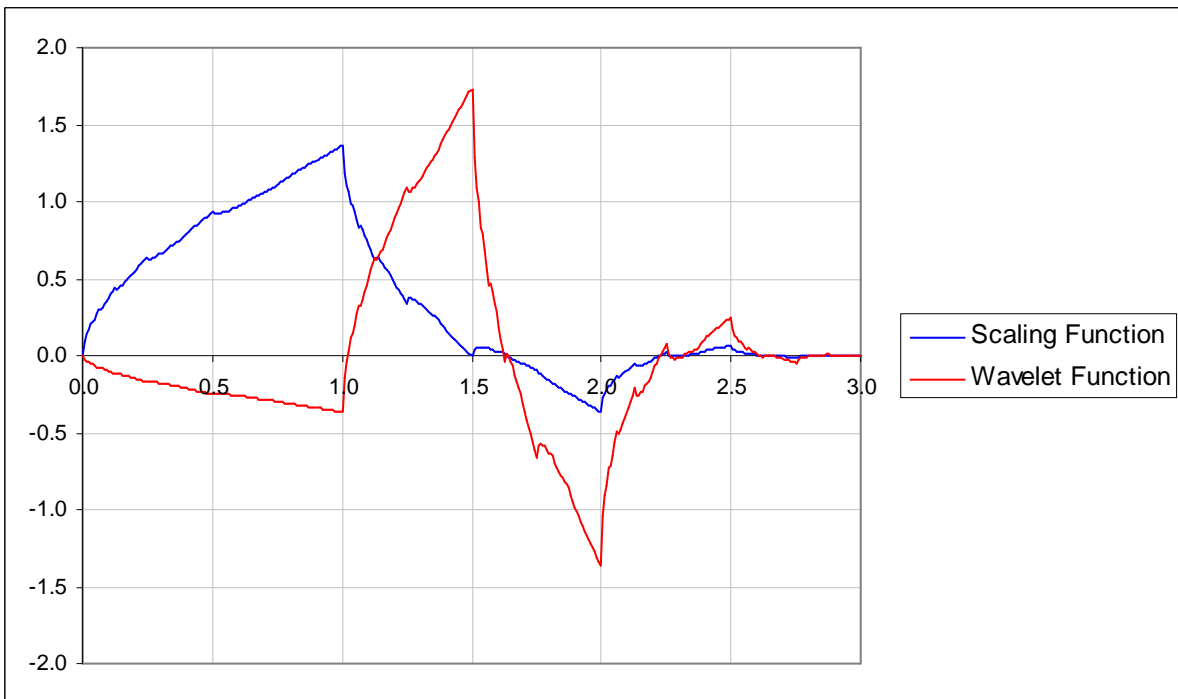


Figure 5.16 – Daubechies 4 scaling and mother wavelet functions**

* Full workings are given in **Appendix B.1**.

** These graphs were drawn in Microsoft Excel using values outputted by the program `d4.exe` which, with its source code, may be found in `Software\Daubechies4\` on the project CD.

Shown in **Figure 5.16** is the graph of this function together with the D4 mother wavelet, which is easily derived from the father wavelet by equation [5.9]. As can be seen, the graphs are not very smooth, and have some interesting fractal-type properties, due to the recursion relationships.

5.4.4 The Fast D4 Transform

As with the Haar transform, there exists a fast method of computing the Daubechies wavelet transform. In order to construct a filter matrix (as with Haar) we start by renaming the coefficients of the mother wavelet function to match those of the scaling function, which are the same but in reverse order and with alternating signs:

$$\begin{aligned} g_0 &= h_3 = \frac{1}{4}(1 - \sqrt{3}), \\ g_1 &= -h_2 = \frac{1}{4}(-3 + \sqrt{3}), \\ g_2 &= h_1 = \frac{1}{4}(3 + \sqrt{3}), \\ g_3 &= -h_0 = \frac{1}{4}(-1 - \sqrt{3}). \end{aligned}$$

The next step in constructing the matrix is to normalize by dividing everything by $\sqrt{2}$. In a computer program, one would want to include this factor in the definition of the wavelet coefficients so that this particular step does not need to be recalculated each time they are used. To write the matrix, we multiply out and shift the coefficients along each row by two elements at each iteration. Instead of doing the smoothing and differencing separately, we can combine the two filters into one and, furthermore, represent the whole operation (at each scale) in one big 2D matrix, which will have N columns and N rows, where N is the number of samples in the signal, or portion thereof. The transform begins by multiplying the filter matrix by the signal vector, as shown below for an 8-sample signal:

$$\begin{bmatrix} h_0 & h_1 & h_2 & h_3 & 0 & 0 & 0 & 0 \\ g_0 & g_1 & g_2 & g_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & h_0 & h_1 & h_2 & h_3 & 0 & 0 \\ 0 & 0 & g_0 & g_1 & g_2 & g_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & h_0 & h_1 & h_2 & h_3 \\ 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & g_3 \\ 0 & 0 & 0 & 0 & 0 & 0 & h_0 & h_1 \\ 0 & 0 & 0 & 0 & 0 & 0 & g_0 & g_1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}$$

It can be seen the last two rows in the transform matrix are problematic, in that there is no space left for h_2 & h_3 , and g_2 & g_3 . There are several ways in which the edge problem is treated, all of which produce very slightly different results:

- 1) Simply ignore it, as has been done in the above matrix.
- 2) Wrap the signal data around, so that we use x_0 and x_1 again in the calculation, and expand the above matrix, for example, to 10×10 to accommodate the extra coefficients.
- 3) Mirror the signal data at the end, i.e. use x_7 , then x_6 again in the calculation.

There are other techniques which may be used. However, since the overall effect is minor, it is not a big problem to use the first option. In *Wave Processor*, a method has been implemented whereby the Haar averages and differences – only requiring two coefficients – were used at the boundary. This method was not seen in any of the sources examined, but is certainly just as good as (if not better than) the three given above, since what we are trying to do is find some sort of smooth value, which Haar provides.

The result of the matrix multiplication is an array which is then split in half – on the left side of the array we have the newly smoothed values (even elements) and on the right are the differences (odd elements), which become the final output. As with the Haar algorithm, the D4 function is called recursively on the first half of the array – the smooth values – until there are only four left, at which point the calculation can go no further.

The pseudo-code for the algorithm is fairly clear and easy to understand. As can be seen, the splitting is done by creating pointers to the first and second half of the output array and simply writing the smoothed values and differences to these locations.

```

FD4T(N) {
    SMOOTH_PTR = TRANSFORM_PTR
    IF N < 4 { // Copy last four values and return
        FOR I = 0 TO 4
            SMOOTH[I] = SIGNAL[I]
        RETURN
    }
    HALF = N / 2 // Differences half
    DIFFS_PTR = TRANSFORM_PTR + HALF

    // Calculate smoothed values and differences
    FOR I = 0 TO HALF {
        J = I * 2
        SMOOTH[I] = H0 * SAMPLES[J] + H1 * SAMPLES[J + 1] +
                    H2 * SAMPLES[J + 2] + H3 * SAMPLES[J + 3]
        DIFFS[I] = G0 * SAMPLES[J] + G1 * SAMPLES[J + 1] +
                    G2 * SAMPLES[J + 2] + G3 * SAMPLES[J + 3]
    }

    // Use Haar for final pair of coefficients
    I = N - 1; J = I * 2
    SMOOTH[I] = (SIGNAL[J] + SIGNAL[J + 1]) / 2
    DIFFS[I] = (SIGNAL[J] - SIGNAL[J + 1]) / 2

    // Copy the smoothed values and recurse
    FOR I = 0 TO HALF - 1
        SIGNAL[I] = SMOOTH[I]
    FD4T(HALF)
}

```


5.5 The Fast Redundant Haar Transform

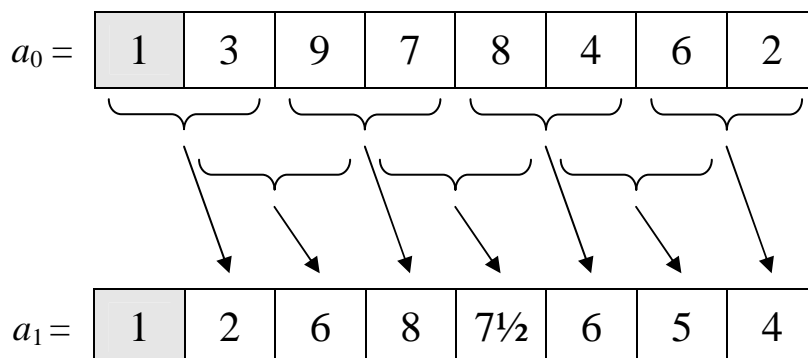
The last type of discrete wavelet transform, the *à trous* or *redundant* transform, provides an improvement on the ordinary Haar transform in that it yields extra detail, which makes it a better frequency filter.

The source used for the following algorithm was a paper entitled *Wavelet-Based Combined Signal Filtering and Prediction* [Murtagh05L]. The technique presented here is relatively new, having been developed in 2005. The reader is referred to section IIC of the paper specifically, which deals with the transform in question.

The redundant Haar transform is very similar to the original transform, except that the decimation step is not done. This means that at every scale, extra (redundant) information is kept, which is the set of differences between the averaged data at the current scale and the previous set of averages. The output is a two dimensional array which thus requires a lot more storage space than the original data, since it comprises S arrays, each of size N , where N is the length of the original wave data and S is the number of scales used in the transform. This time the function is not recursive, since the same number of points is transformed for every scale, and therefore it is calculated by iterating a loop. From here, rather than repeating the already clear description given in the source paper, the transform will be presented by working through an example.

5.5.1 Redundant Haar Transform Example

We start, as usual, by calculating the set of averages and differences, this time starting with *every* pair of elements, not just for every odd and even pair as with ordinary Haar. Sticking with the same array of data as previously, which we'll call a_0 , then for a_1 , the new set of averages, we have:



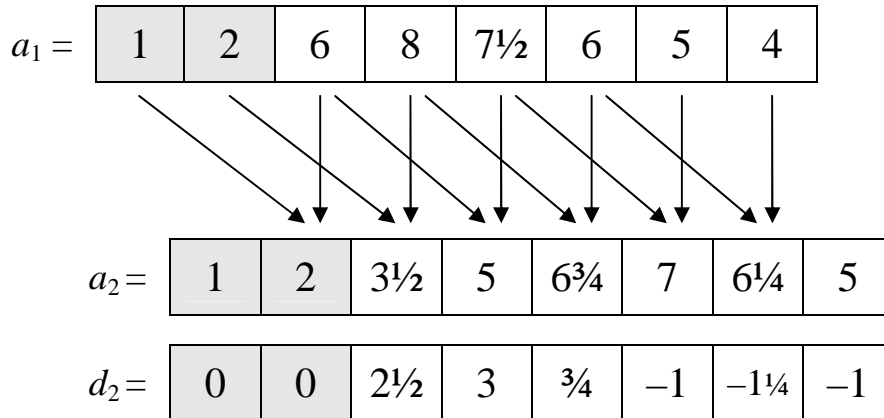
Note that the first element of a_0 , is simply copied to the first element of a_1 (shown in grey). As the transform progresses, we shall see why this is done. The differences we shall name d_s , where s is the current scale. These differences may be calculated either by subtracting the current set of averages from the previous, or by taking the difference between array elements (as opposed the sum) and dividing by 2, which comes to the same thing:

$$d_1 =$$

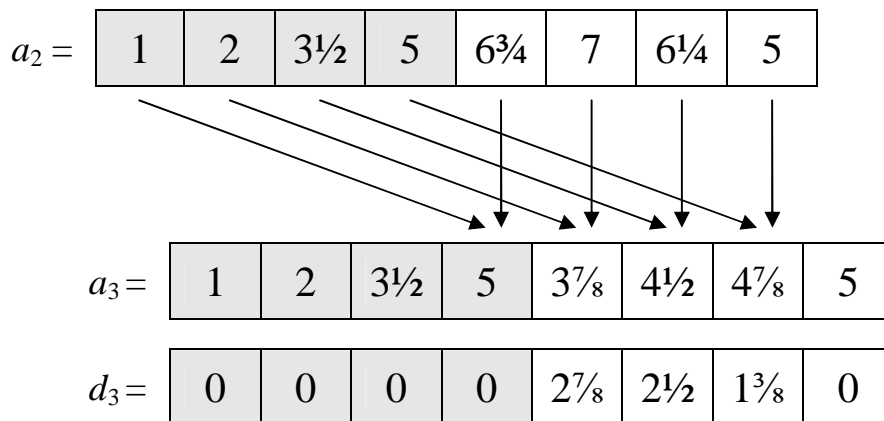
	0	1	3	-1	½	-2	1	-2
--	---	---	---	----	---	----	---	----

The set of averages, a_1 , then becomes the input for the calculation at the next scale, and the set of differences, d_1 , is the set of wavelet coefficients at the first scale. Note that the original dataset, a_0 must also have a length which is a power of 2. This is because as s increases, τ , the interval between array elements used in the calculation, doubles, i.e. $\tau = 2^s$ (where s is a positive integer).

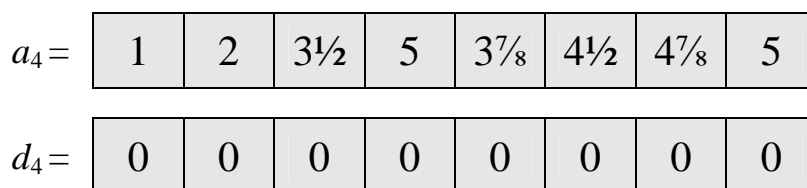
As observed in the first step, those elements on the left hand side, which do not have a “partner” (i.e. their position in the array is less than τ at some scale, s) are copied to the beginning of the next averages array, thus for $s = 1$ ($\tau = 2$):



The pattern continues with $s = 2$ ($\tau = 4$): ...



... until no more calculations can be performed, i.e. $\tau = N$, or in this case, 8:



In order to reconstruct the original signal, we take the final set of smoothed averages, a_4 , and add it to the sum of all the d_k arrays. Thus our final array of coefficients – the complete transform for this example is:

1	2	$3\frac{1}{2}$	4	$3\frac{7}{8}$	$4\frac{1}{2}$	$4\frac{3}{4}$	$4\frac{1}{2}$
0	1	3	-1	$\frac{1}{2}$	-2	1	-2
0	0	$2\frac{1}{2}$	3	$\frac{3}{4}$	-1	$-1\frac{1}{4}$	-1
0	0	0	0	$2\frac{7}{8}$	$2\frac{1}{2}$	$1\frac{3}{8}$	0

Note that the sum of all the columns is equivalent to the original dataset.

5.5.2 Source Code for the Algorithm

With no source on which to base code for the algorithm in *Wave Processor*, the method may be slightly naïve in terms of memory usage, however it is still very efficient as far as speed goes and is thus an acceptable approach. It may be seen from the source code that a temporary array is used to store the previous set of averages at each scale – a waste of memory if there is another way of recovering this, as in the lifting scheme algorithm*.

Smooth values are also not copied to the final row of the output array, since these really only need to be retained so that a reverse transform back to the original signal is possible, and this was not needed. This could be done, however, by including the two lines which have been currently commented out (at the beginning of the main loop) in the source, although this means that all the zero values in the d_s rows will contain, instead, corresponding elements from the a_s rows. This doesn't actually make all that much of a difference to the final result, since the very lowest frequency bands (where this has the greatest significance) do not contain much information that is useful anyway and could even be ignored completely.

Note that in this and all other DWT algorithms implemented in *Wave Processor*, the transformed signal samples have been squared so as to get the energy of the function at their points, or instantaneous power. This is a better and more correct representation of magnitude, which is the desired output for the spectrogram.

5.6 Windowing in the Frequency Domain

The heading of this sub-section is one way of looking at a new method proposed here which may be used to achieve multiple pitch extraction. Just as the STFT first divides the signal up into time slices – windows in the time domain, we now have a facility, the redundant Haar transform, which allows us to divide the signal up into separate frequency bands – windows in the frequency domain. We can then analyse each of these bands separately and, with MPM, pick out only those clear frequencies which should lie in a particular band.

* Close to the time of publishing of this thesis, this algorithm has now been improved, as suggested.

5.6.1 Combining Redundant Haar with MPM

While all the above wavelet theory may be quite complicated and takes some digesting, in practice this algorithm is quite simple. Given that the McLeod Pitch Method, described in the previous chapter is encapsulated in a function `MPM()`, the pseudo-code for the proposed `PITCH_EXTRACTION()` function, for single or multiple pitches, looks like this:

```
PITCH_EXTRACTION(WAVE, MULTIPLE) {
    Get length of wave
    Make length a multiple of specified window width
    IF(MULTIPLE)
        REDUNDANT_HAAR_TRANSFORM(WAVE)
    ELSE
        DATA = WAVE.DATA
    FOR N = 0 TO NUM_FREQ_BANDS - 1 {
        IF(MULTIPLE)
            DATA = WAVE.TRANSFORM_DATA[N]
        MPM(DATA)
    }
}
```

If single pitch recognition is being done, the FOR loop will only be iterated once, since the “frequency band” in this case is the whole signal. Note that for the Redundant Haar, the power spectrum should not be used in the analysis, since the signal bias is, undesirably, shifted above zero due to the squaring of all the sample values.

5.6.2 Calculation Time

Since all functions within this multiple pitch extraction method have been optimized in terms of efficiency, the total calculation time is actually very short, given the combined complexity of the various transforms. Furthermore, it depends only on the length of the wave file being analyzed, and not its content. Using a fast Intel Core 2 Duo processor, the entire operation, for a five-second wave file containing 3-part harmonies, takes only about eight seconds. This could in fact be made even faster in the peak-picking stage of MPM, where the search for a particular pitch period could start and end only within the limits that are applicable to the particular frequency band being analysed, i.e. it would be more efficient to look for only those frequencies that are expected to lie in a certain band.

5.7 Output

Having covered a large amount of theory in this chapter, it is now time to put it into practice. The following spectrograms and pitch graphs are all screenshots from *Wave Processor*, and the experiments may be repeated by loading the given wave files into the application and selecting the correct parameters.

5.7.1 Spectrograms from Discrete Wavelet Transforms

The first signal of interest is, again, the chirp from Chapter 2. **Figures 5.17** and **5.18** show the Haar and D4 transforms of the wave, respectively. Note how the Daubechies transform is a much better filter than the Haar, which has rather poor frequency localization in comparison [Daubechies92L].

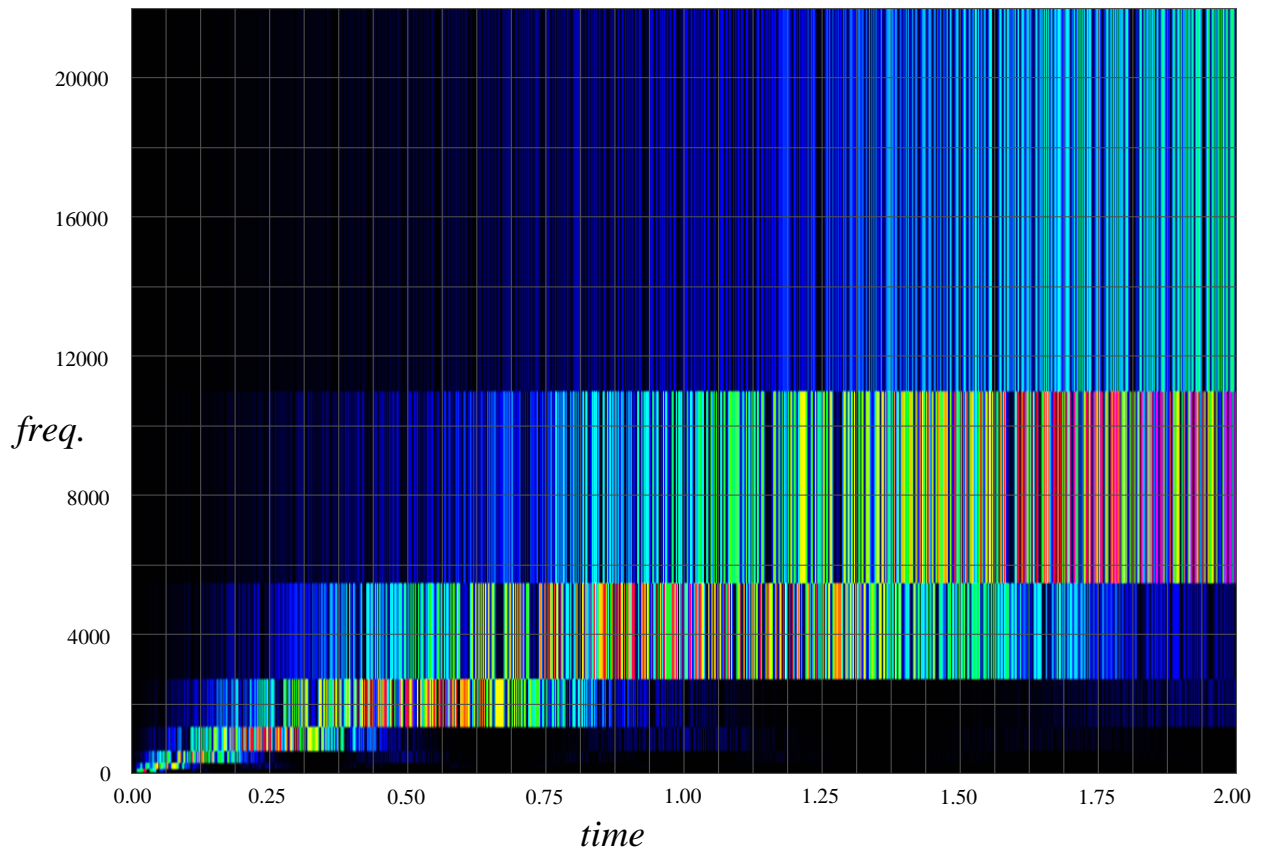


Figure 5.17 – Haar wavelet transform of chirp signal

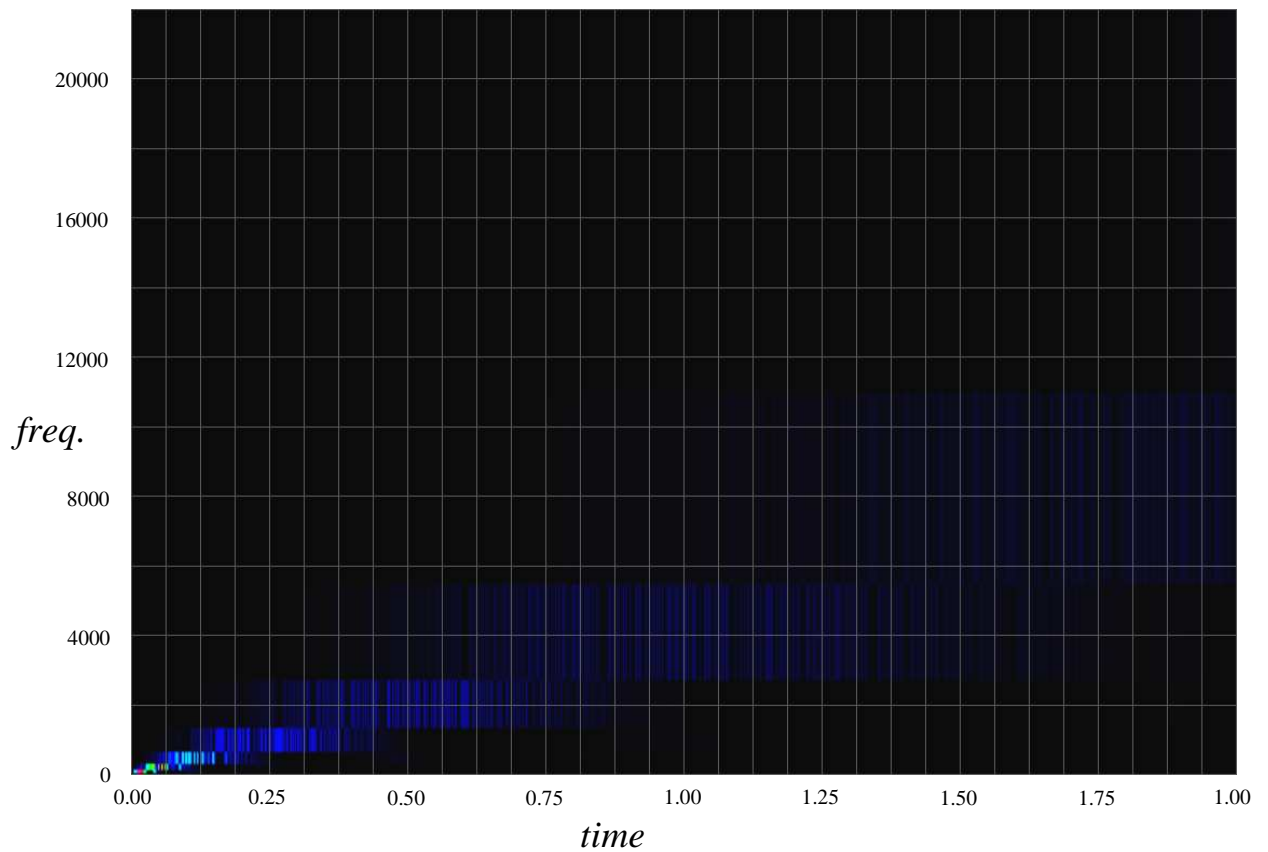


Figure 5.18 – Daubechies 4 wavelet transform of chirp signal

The entire bandwidth of the frequency axis in these spectrograms is half the sample rate of the signal, which in this case is 44,100 bits per second. Since the duration of the signal is 2 seconds, there are 88,200 samples altogether. This means that there are 131,072 ($= 2^{17}$) points in the transform – the nearest power of two – and thus 17 scales. At each row of the transform, scale halves and frequency doubles. Another way of displaying the graph is to have rows of equal height in the transform, but have a dyadic frequency scale. This is an option which may be set for each transform in *Wave Processor*, the result of which is shown in **Figure 5.19**, which is, again, the D4 transform of the same signal.

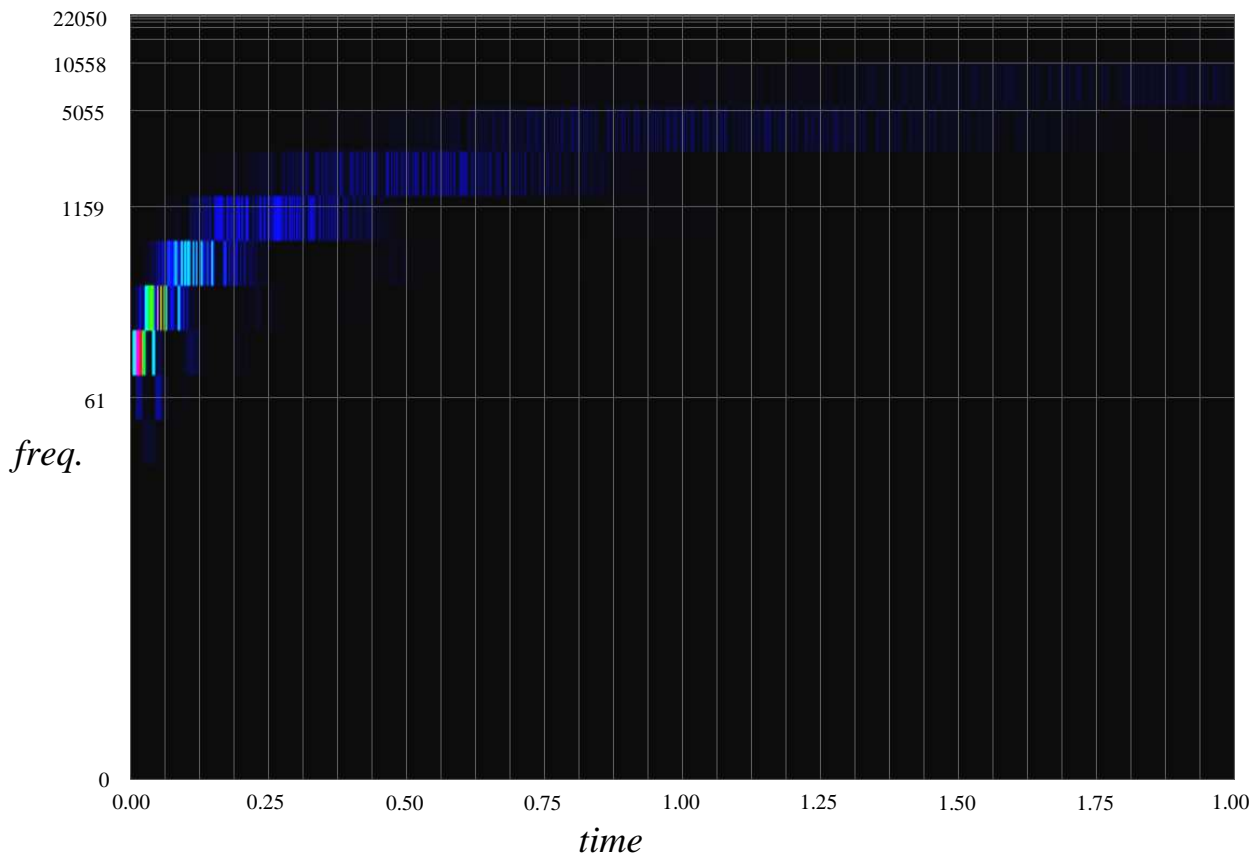


Figure 5.19 – Dyadic graph of D4 wavelet transform of chirp signal

Note that in this graph, the bands of colour – the rows of the transform at each scale – are all the same height. Again, the dyadic relationship between frequency and pitch is illustrated here, since each row could also be thought of as representing the musical interval of one octave. The frequency at the top of each band in this graph is calculated as follows:

$$f = B \times 2^{b-S},$$

where S is the number of scales, b is the band and B is the whole bandwidth of the spectrogram, i.e. the maximum of the frequency axis.

The next test wave was a synthesized piano sound playing high and low Cs at different octaves*. **Figures 5.20** and **5.21** show the dyadic spectrograms of the Haar and D4 transforms of this signal, respectively.

* This is saved as Sound\lohi.wav on the project CD

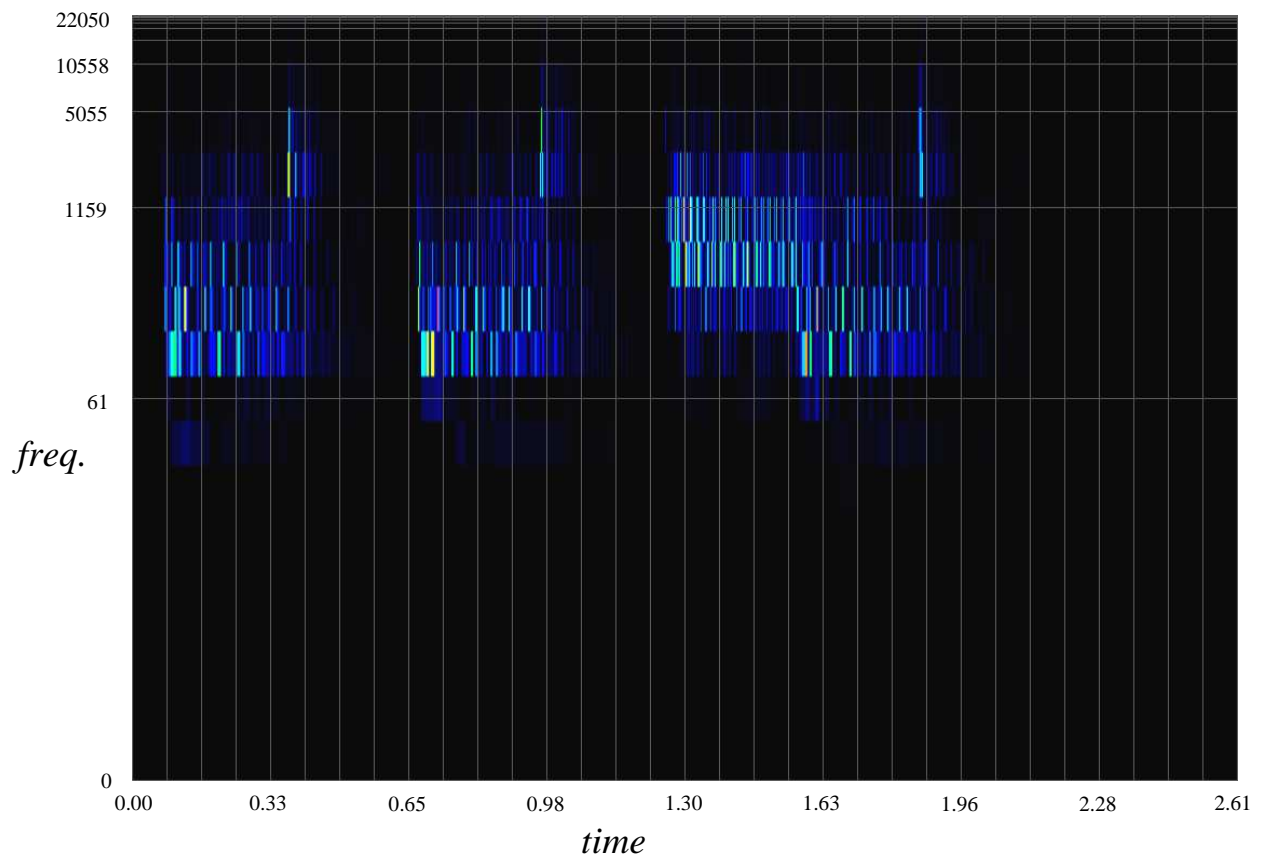


Figure 5.20 – Haar wavelet transform of lohi .wav

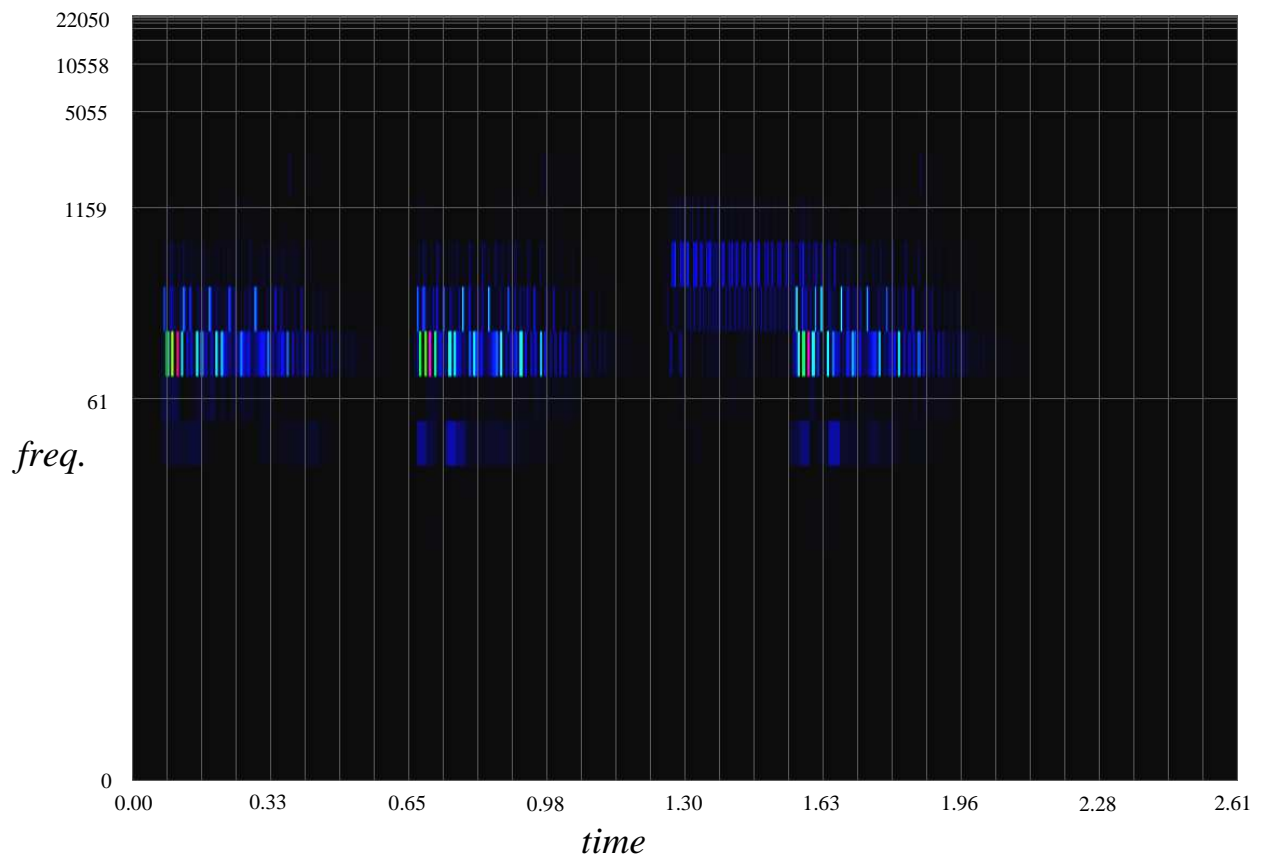


Figure 5.21 – D4 wavelet transform of lohi .wav

As can be seen, the D4 transform does not operate as well as the Haar transform in the detection of higher frequencies – the high Cs, which have a frequency of 2096Hz, are barely visible on the D4 spectrogram.

The spectrogram of the redundant Haar transform, shown in **Figure 5.22**, is similar to the normal Haar, except that it has equal smoothness (or lack thereof) at lower scales. The reason for Renaud, Starck and Murtagh’s French name, *à trous*, meaning “holey” or “full of holes” [Murtagh05L] becomes apparent upon close inspection of the spectrogram.

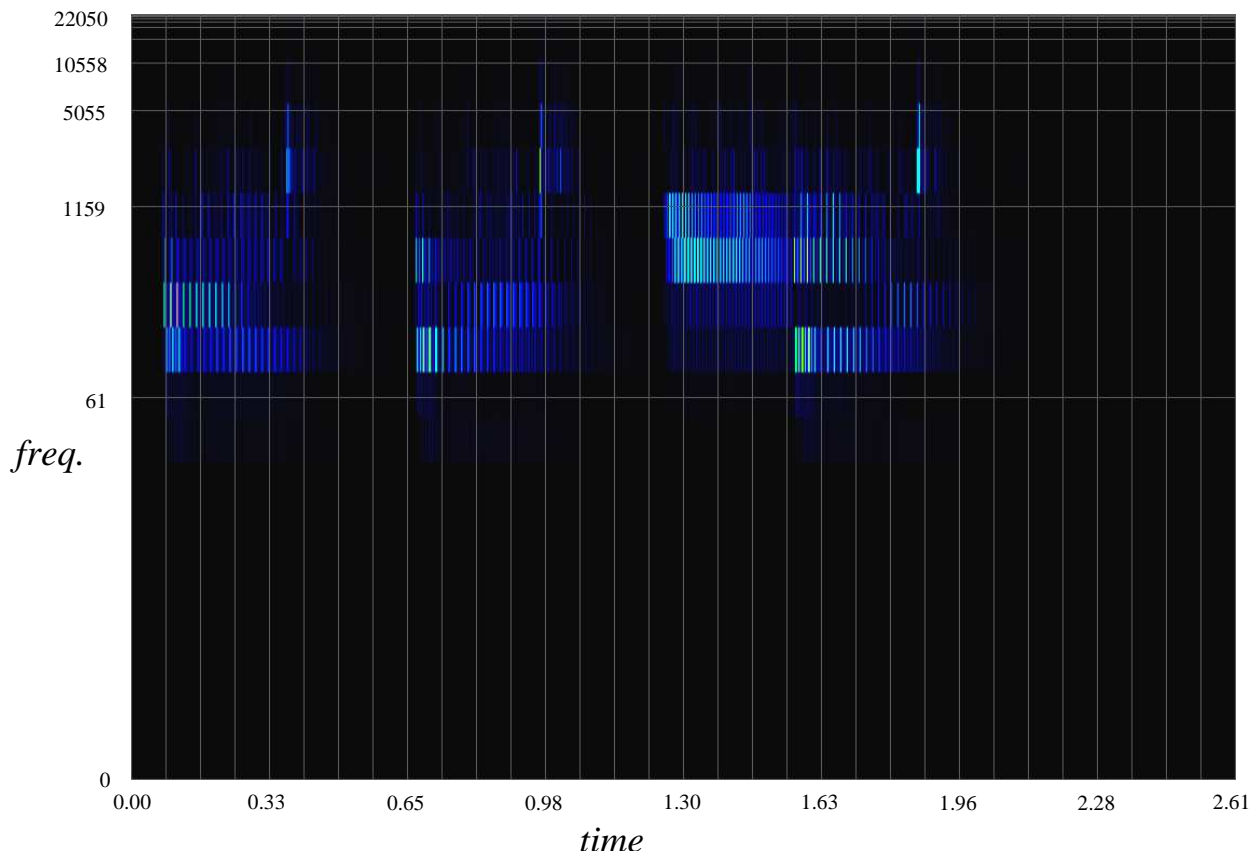


Figure 5.22 – Redundant Haar wavelet transform of lohi .wav

In order to demonstrate the performance of the algorithm described in section 5.6, two and three part arrangements of the opening two bars of *Nkosi Sikeleli Africa* were created by recording synthesized sounds from *Sibelius*. The scores used to create these arrangements are shown in **Figures 5.23** and 5.24.

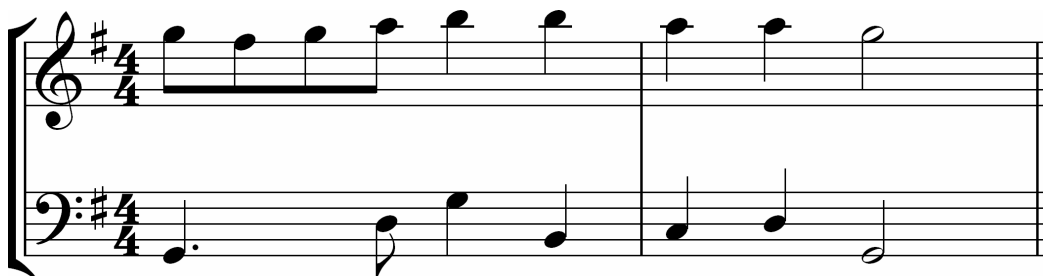


Figure 5.23 – Two-part arrangement of first two bars of *Nkosi Sikeleli Africa*



Figure 5.24 – Three-part arrangement of first two bars of *Nkosi Sikeleli Africa*

The middle and upper voices were assigned a flute patch in *Sibelius*, which has a relatively simple timbre, while the bass part was given a string bass sound. The results of the adapted McLeod Pitch Method for multiple pitch recognition on these two arrangements are shown in **Figures 5.25** and **5.26**. Apart from one or two anomalies, the two-part transcription is extremely accurate – the melody line is in fact 98.4% accurate if one counts the slight visible pitch bends at the ends of four of the notes as errors.

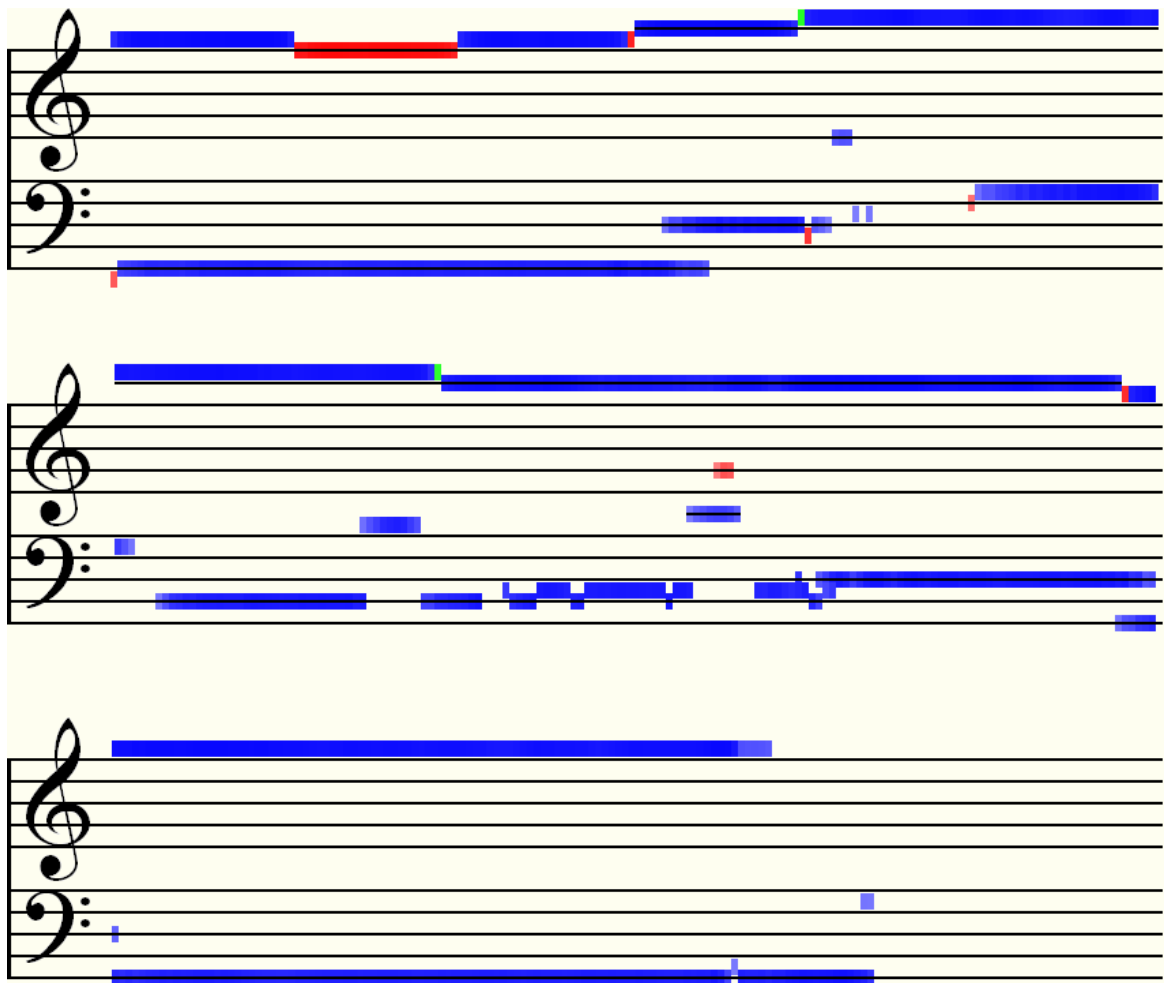


Figure 5.25 – Automatic transcription of two-part arrangement using DWT/MPM

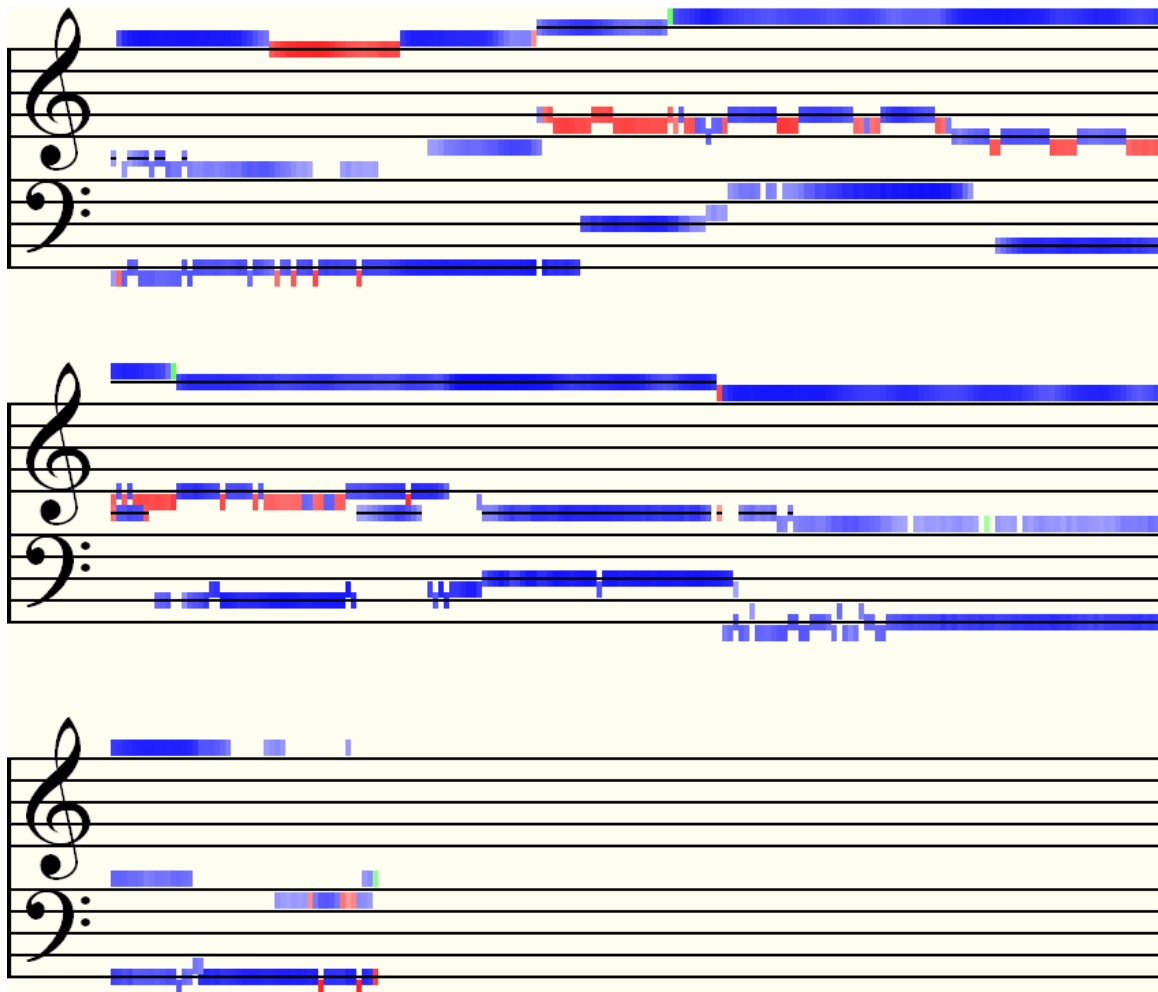


Figure 5.26 – Automatic transcription of three-part arrangement using DWT/MPM

The three-part arrangement is also a good rendering of the score with really only one poor estimation of the pitch of the fifth note in the middle voice line, which should be a D₄.

While these results are encouraging, the method is not quite so discerning if the parts are not balanced properly – a slight change in volume to the middle voice, for example, rendered it almost invisible to the algorithm. Also, when changing this voice’s instrument to an oboe or clarinet patch, most of the notes in the output for this line disappeared. Nevertheless, given that very little tweaking of parameters was required to produce the results shown here, this method is certainly worth investing some time in improving, and should definitely be considered in the case of simple two and three-part transcriptions, as it proves to be fast and generally robust.

The settings for MPM which produced the best results in each case were as follows:

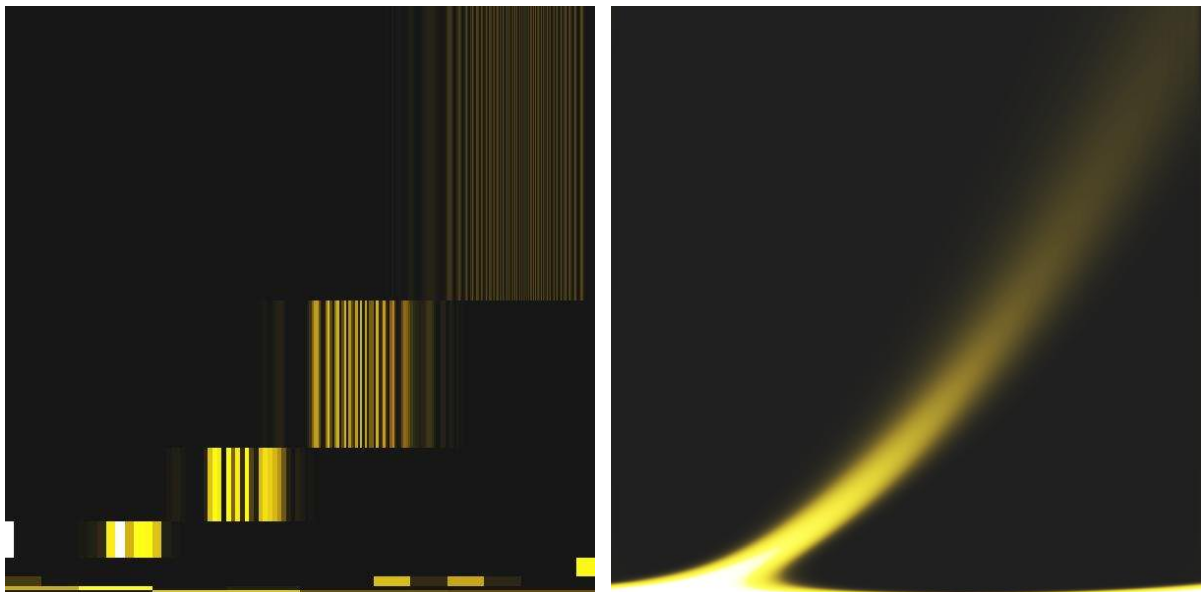
- Two-part arrangement – window width: 2048 samples, pitch detection threshold: 90%, poor clarity threshold: 50%
- Three-part arrangement – window width: 2048 samples, pitch detection threshold: 90%, poor clarity threshold: 35%

6 Drawing a Spectrogram Using the Continuous Wavelet Transform

Up to this point we have looked at algorithms for transforming signals using discrete methods. Unfortunately, with the discrete wavelet transform in the previous chapter, regardless of which wavelet is used, the frequency resolution is not good enough for fine musical pitch detection. We can only be confident about the particular *octave* in which a detected pitch lies, since frequencies double at every scale of the transform. This is the reason why the transcription method in the previous chapter worked well for the two-part arrangement and acceptably for three parts – the intervals between each voice are all wide, and so they are well separated by the redundant Haar transform. However, for closer intervals, the method must pick whichever frequency is strongest in each octave and reject all the others, which is quite restrictive and unrefined – we need semitone accuracy at the very least. What is required then is a multi-resolution analysis method like the DWT, but one which allows us to choose any arbitrary set of scales and not be restricted to just powers of two.

6.1 The Continuous Wavelet Transform

Fortunately another method exists for discretizing the wavelet transform which allows the approximation of a much smoother function. Compare the images in **Figure 6.1** below, which show a DWT and the proposed *continuous wavelet transform* (CWT) of a chirp signal (pictures courtesy of Petr Klapetek [Klapetek02W]):



a) Discrete wavelet transform

b) Continuous wavelet transform

Figure 6.1 – Comparison of DWT and CWT spectrograms of a chirp signal

As we can see immediately, the plot on the right renders a much more detailed picture of what is happening to the frequency of the signal. As with the discrete transform on the left, the higher frequency information is more blurred; however, for music analysis, the practical frequency range falls well within the boundaries of these graphs and so this is not too much of

a problem. Due to the large amount of information obtained from the CWT, it is well worth examining as a likely candidate for use within an accurate pitch detection algorithm. This hypothesis has been corroborated fairly recently in a comparative study by Michael Cowling as part of his PhD thesis [Cowling04L], in which he recommends using the CWT for a number of different applications. The only major drawback about this transform is that it is very slow to compute. However, for the purposes of this study, accuracy and high resolutions have been considered much more important than speed, hence the dedication of most of the rest of this thesis to methods that use the CWT at their core.

6.1.1 The CWT Function

Suppose ψ_0 is an unscaled, unshifted wavelet function (in other words a mother wavelet) the continuous wavelet transform is defined by the following [Zhan06L]:

$$\Psi^f(s, \tau) = \int_{-\infty}^{+\infty} f(t) \frac{1}{\sqrt{s}} \psi_0^* \left(\frac{t - \tau}{s} \right) dt, \quad [6.1]$$

where * denotes the complex conjugate. As with the Fourier transform, the integration is performed over all time. Although some elements of this equation could be recognizable from the previous discussion of the DWT, it is worth explaining in full again:

$\Psi^f(s, \tau)$ is the wavelet transform of the signal $f(t)$. It is a two-dimensional function over s , scale and τ , translation. The s is inversely proportional to the frequency of the wavelet – as the frequency increases, scale decreases. The s is therefore technically equivalent to the wavelength of the wavelet (in proportion to the duration of the signal being analyzed). While on the subject of scale, the concept is analogous to that of maps: a scale of 1:1 is life size, while at 1:10,000 a map shows a zoomed out view. Likewise, with wavelets, a scale of 1 means that the wavelet is stretched across the entire signal, whereas when $s = 1/10,000$, the wavelet is squashed so that 10,000 cycles may be fitted*. The τ is the shift in time, i.e. the location of the wavelet within the signal, subtracted from t because it is, in effect, a time delay before the convolution of ψ with $f(t)$.

The scaling function**, included in the above equation, is sometimes written separately for clarity [Zhan05L]:

$$\psi_{s,\tau}(t) = \frac{1}{\sqrt{s}} \psi_0 \left(\frac{t - \tau}{s} \right),$$

Accordingly, a plot of $\Psi^f(s, \tau)$ will reveal information about the signal, $f(t)$, in the time-frequency domain. It is interesting to note here that this function bears some similarity to autocorrelation (see section 4.3.2 in Chapter 4) except that the wavelet transform is a measure of similarity between the signal and another function – the wavelet – rather than itself. Autocorrelation does not operate over different scales, so it is a function of time only.

* Please read the article [Mackenzie01L] for a good overview about how viewing things at different scales – multi-resolution analysis – is one of the ways in which we ourselves sense information.

** In order to demonstrate the wavelet scaling function, the CWT Options Dialog box in *Wave Processor* includes an animated image of each wavelet, which shows the effects of changing the scale (linearly) between 1:1 and 1:2.

6.1.2 The Discretized CWT

Equation [6.1] indicates that the CWT can be regarded as an image, the rows of which are a set of convolutions of the signal with a scaled wavelet function. Thus, each row of the discretized transform may be calculated at any arbitrary scale, s , for N samples, f_n , of a discrete signal, f , by the following:

$$\Psi_{s,\tau}^f = f \otimes \psi_{s,0} = \sum_{n=0}^{N-1} f_n \cdot \frac{1}{\sqrt{s}} \psi_0^* \left(\frac{n-\tau}{s} \right).$$

In the equation, $\Psi_{s,\tau}^f$ represents rows of coefficients of the transform at chosen scales, s . The τ is again the position of the wavelet in the signal. Since the transform has the same time resolution as the signal, τ will also run from 0 to $N-1$. \otimes denotes a convolution with $\psi_{s,\tau}$, the sampled wavelet, which is scaled and shifted.

6.1.3 A Useful Mathematical Trick

Unfortunately, an algorithm based on the above summation is far too slow ($O(N^2)$ per scale) to be of any practical use. A much faster algorithm may be constructed using the Convolution Theorem which states that a convolution is simply a point-wise product in Fourier space [Wolfram09W]. Thus, if f and g are two functions and \mathcal{F} denotes a Fourier transform operator, with \mathcal{F}^{-1} as its inverse, then:

$$f \otimes g = \mathcal{F}^{-1}[\mathcal{F}(f) \cdot \mathcal{F}(g)].$$

Now, we can calculate the discrete Fourier transform of a discrete signal using the Fast Fourier Transform, and, as will be shown in the next section, the Fourier transform of the scaled wavelet function is also known empirically. Thus the Discretized CWT may be expressed very simply by the following, where the hat symbol (^) denotes a DFT:

$$\hat{\Psi}_{s,m}^f = \hat{f}_m \cdot \hat{\psi}_{s,m}.$$

Note that this is a function of scale and frequency, as opposed to time shift. Both signal and transform row have the same number of points, so this may be indexed by the same variable, m , which is different from the scale variable, s . All that remains to be done to complete the transform is the inverse DFT to return from Fourier space. This calculation is now order $M \log(N)$ per scale, which, although not fast enough to be implemented in real-time, is still a great improvement.

6.2 Four Wavelets

There are many different wavelet functions that may be used as the kernel for the CWT. The choice of wavelet depends largely on the intended application. Four popular wavelets have been chosen for experiment and are implemented in *Wave Processor*. Of these, the most frequently used in practice (and the original wavelet) is the first.

6.2.1 The Morlet Wavelet

The original work done by French geophysicist Jean Morlet in the 1970s was really the precursor to the wavelet transform. Morlet called his new method the *cycle-octave transform*, [Goupillaud 84L] extending the ideas of Gabor's transform [Gabor46L] but using time windows of varying width. As mentioned in the introduction to wavelets in Chapter 5, Morlet realised that the whole key to the transform was finding a basis function which retained its shape when scaled, thus the Morlet wavelet was born. Although Morlet's new transform was originally published in 1982 [Morlet82L], the technique was more properly formulated with the help of his friend and colleague, Pierre Goupillaud, together with physicist Alex Grossman in [Goupillaud 84L] and in [Grossman84L].

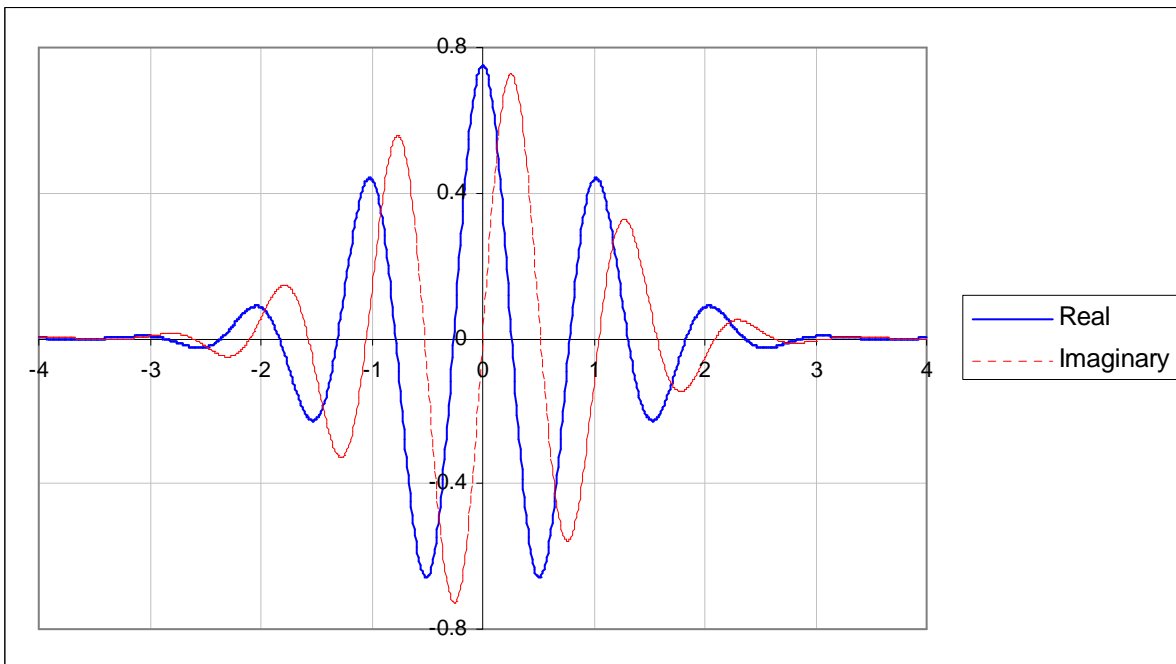


Figure 6.2 – The Morlet wavelet – Graph showing real and imaginary components

The Morlet mother wavelet is defined as [Grossman84L, Zhan05L]:

$$\psi_0^M(t) = \pi^{-\frac{1}{4}} \left(e^{i\omega_0 t} - e^{-\omega_0^2} \right) e^{-\frac{t^2}{2}}.$$

This is basically a damped complex sinusoidal function. The constant, ω_0 , is the central angular frequency of the wavelet and is called its *wavenumber*. For the wavelet to meet certain suitability criteria, ω_0 should be greater than 5. This means that the second term inside the bracket, known as the correction term, becomes negligibly small and may be left out completely, yielding the simple Morlet wavelet [Kronland87L]:

$$\psi_0^M(t) = \pi^{-\frac{1}{4}} e^{i\omega_0 t} e^{-\frac{t^2}{2}}.$$

The Fourier transform of the scaled Morlet wavelet is [Torrence98L, Zhan05L]

$$\hat{\psi}_s^M(\omega) = \pi^{-\frac{1}{4}} e^{-\frac{1}{2}(s\omega - \omega_0)^2}.$$

where $\omega = 2\pi\nu$. This is simply a Gaussian curve, the peak of which is at $1/s$ when the wavelet is scaled. **Figures 6.2** and **6.3** show the Morlet mother wavelet with a wavenumber of 6, and its Fourier transform, respectively. Since the FT is very nearly zero for negative values when ω_0 is suitably large, and because we must assume that $\psi^M(t)$ contains only positive frequencies [Kronland87L], it makes for easy and more rapid computations if we use:

$$\hat{\psi}_s^M(\omega) = \pi^{-\frac{1}{4}} H(\omega) e^{-\frac{1}{2}(s\omega - \omega_0)^2},$$

where $H(\omega)$ is the Heaviside (or Step) function, defined as [Wolfram09W]:

$$H(\omega) = \begin{cases} 0, & \omega \leq 0; \\ 1, & \omega > 0. \end{cases}$$

The fact that the Fourier transform of this and other complex wavelets is strictly positive means that unlike the FT, the CWT does not necessarily detect negative frequencies, though the integral is from negative infinity.

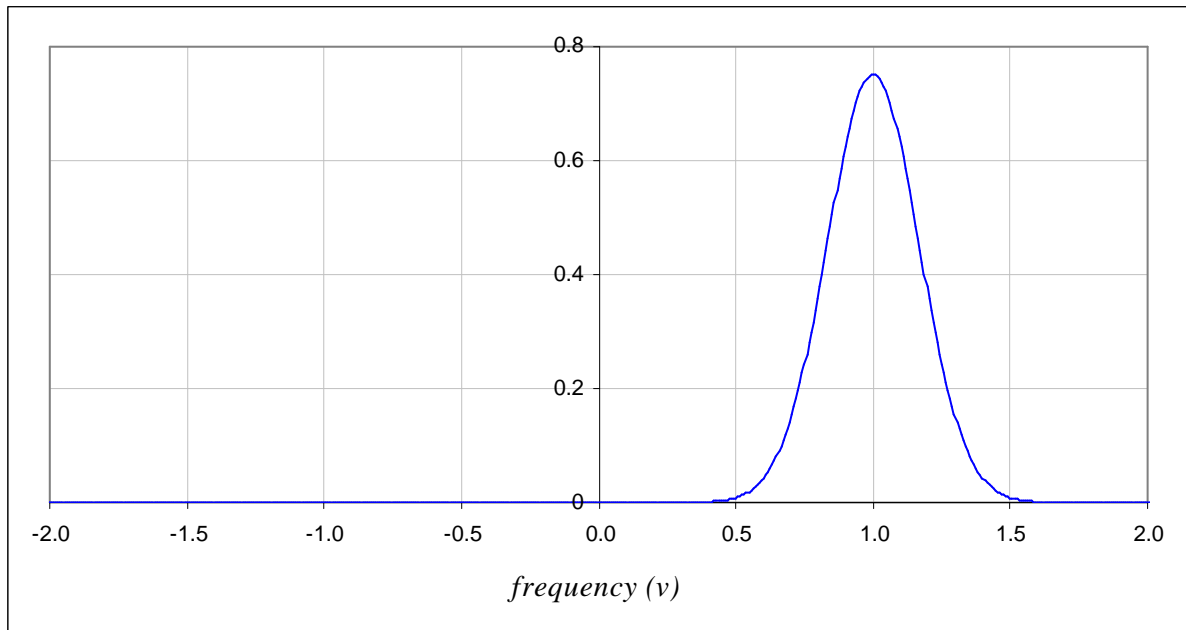


Figure 6.3 – Fourier transform of Morlet wavelet

6.2.2 The Derivative of Gaussian Wavelet

The real component of the Derivative of Gaussian mother wavelet is given by [Torrence98L]

$$\psi_0^{D_n}(t) = \frac{(-1)^{n+1}}{\sqrt{\Gamma(n + \frac{1}{2})}} \frac{d^n}{dt^n} e^{-\frac{t^2}{2}},$$

where n is the order of the derivative and $\Gamma\left(n + \frac{1}{2}\right)$, the Gamma Function, is calculated by

$$\Gamma\left(n + \frac{1}{2}\right) = \frac{(2n-1)!!}{2^n} \sqrt{\pi}.$$

Note that the double factorial notation (!!) does not mean the factorial of a factorial, but is defined as [Wolfram09W]:

$$n!! = \begin{cases} 1, & n = 0, n = 1; \\ n[(n-2)!!] & n > 1. \end{cases}$$

For example, $7!! = 1 \times 3 \times 5 \times 7 = 105$.

Putting the normalizing factor aside for a moment, the first three derivatives of the Gaussian are as follows:

$$\frac{d}{dt} e^{-\frac{t^2}{2}} = -te^{-\frac{t^2}{2}},$$

$$\frac{d^2}{dt^2} e^{-\frac{t^2}{2}} = (t^2 - 1)e^{-\frac{t^2}{2}},$$

$$\frac{d^3}{dt^3} e^{-\frac{t^2}{2}} = -(t^3 - 3t)e^{-\frac{t^2}{2}}.$$

It should be noted that the n^{th} derivative of the Gaussian is the Gaussian itself multiplied by alternately positive and negative n^{th} Hermite probabilistic polynomials, $He_n(t)$ [Abramowitz65L, Arfken85L]:

$$\frac{d^n}{dt^n} e^{-\frac{t^2}{2}} = (-1)^{n+1} He_n(t) e^{-\frac{t^2}{2}}.$$

Thus, a recursive algorithm may be created in order to generate the wavelet function for any order, using the formula [Arfken85L]:

$$He_{n+1}(t) = tHe_n(t) - nHe_{n-1}(t),$$

where $He_0(t) = 1$ and $He_1(t) = t$. With an order of 2, the Derivative of Gaussian (DoG) wavelet is also known as the *Mexican Hat* wavelet. After simplifying the normalization factor, this instance of the function is then:

$$\psi_0^{D_2}(t) = \frac{2}{\sqrt{3}} \pi^{-\frac{1}{4}} (1-t^2) e^{-\frac{t^2}{2}}.$$

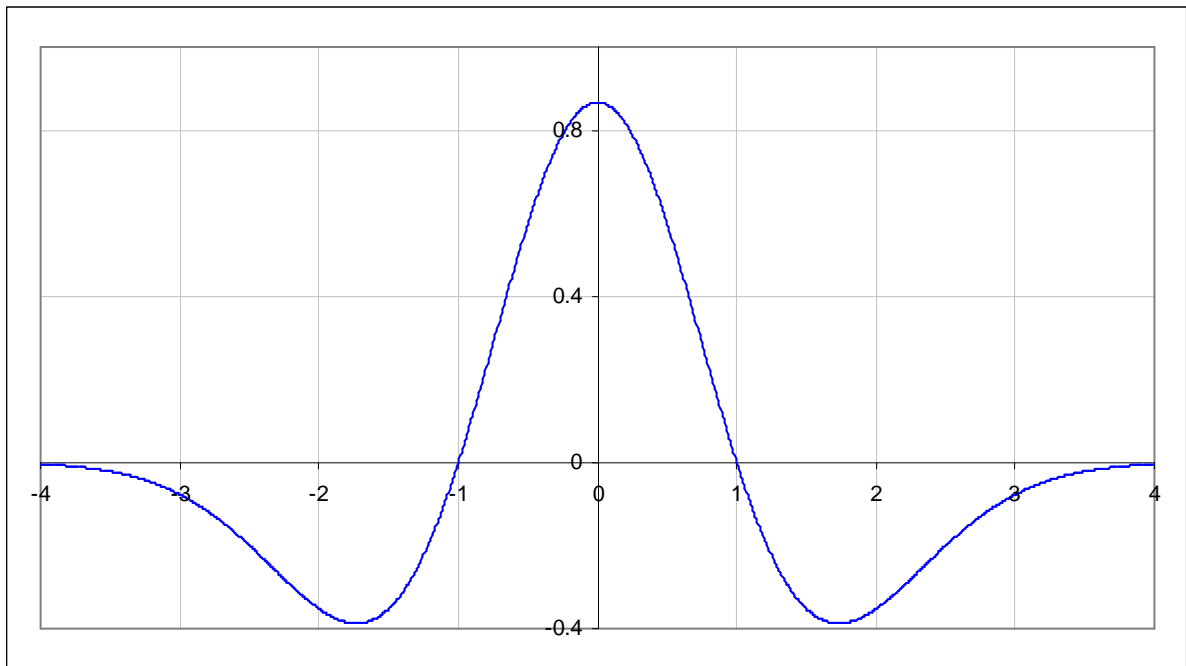


Figure 6.4 – Second order real Derivative of Gaussian (Mexican Hat) wavelet

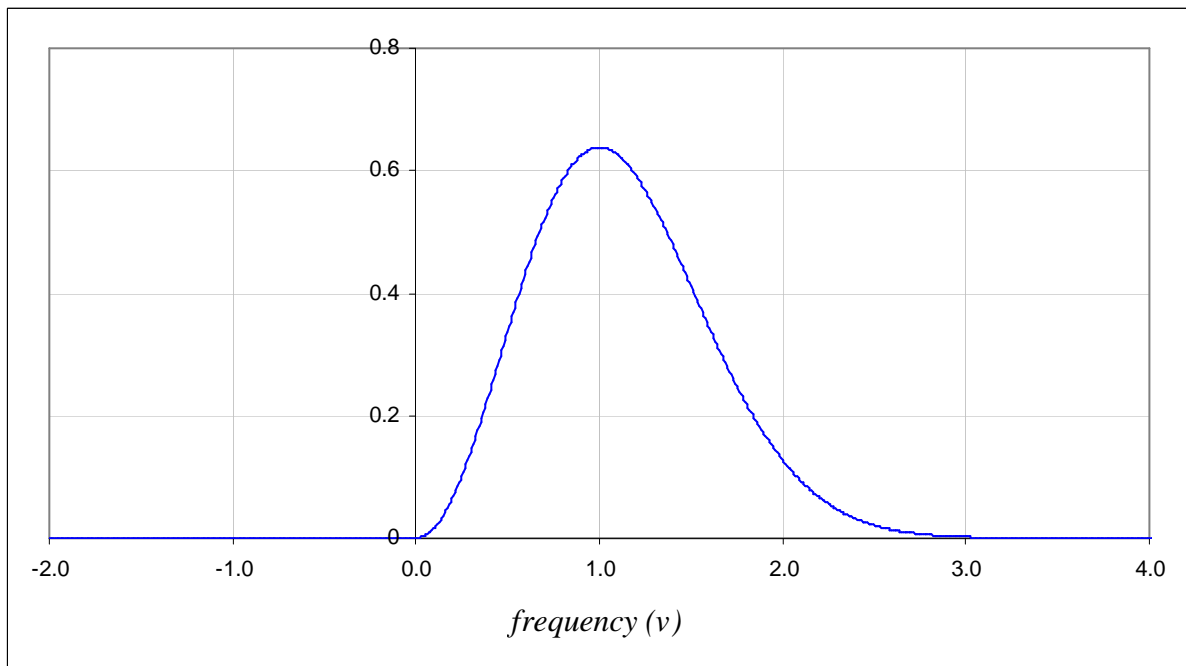


Figure 6.5 – Fourier transform of complex Mexican Hat wavelet

Unlike the Morlet wavelet, the DoG wavelet does not have an imaginary component. However, a complex wavelet may be created by including the Heaviside function in the Fourier transform, which (for the real wavelet) is given by:

$$\hat{\psi}_0^{D_n}(\omega) = \frac{-i^n}{\sqrt{\Gamma(n + \frac{1}{2})}} \omega^n e^{-\frac{\omega^2}{2}}.$$

For the scaled complex Mexican Hat wavelet, the Fourier transform is thus:

$$\hat{\psi}_s^{D_2}(\omega) = \frac{2}{\sqrt{3}} \pi^{-\frac{1}{4}} H(\omega) (s\omega)^2 e^{-\frac{(s\omega)^2}{2}}.$$

6.2.3 The Paul Wavelet

Thierry Paul [Paul09W] is the French mathematician for whom this wavelet was named and the mother wavelet of order n is given by [Torrence98L]:

$$\psi_0^{P_n}(t) = \frac{2^n i^n n!}{\sqrt{\pi(2n)!}} (1 - it)^{-(n+1)}.$$

Figure 6.6 shows the fast-decaying components of the order 4 Paul wavelet.

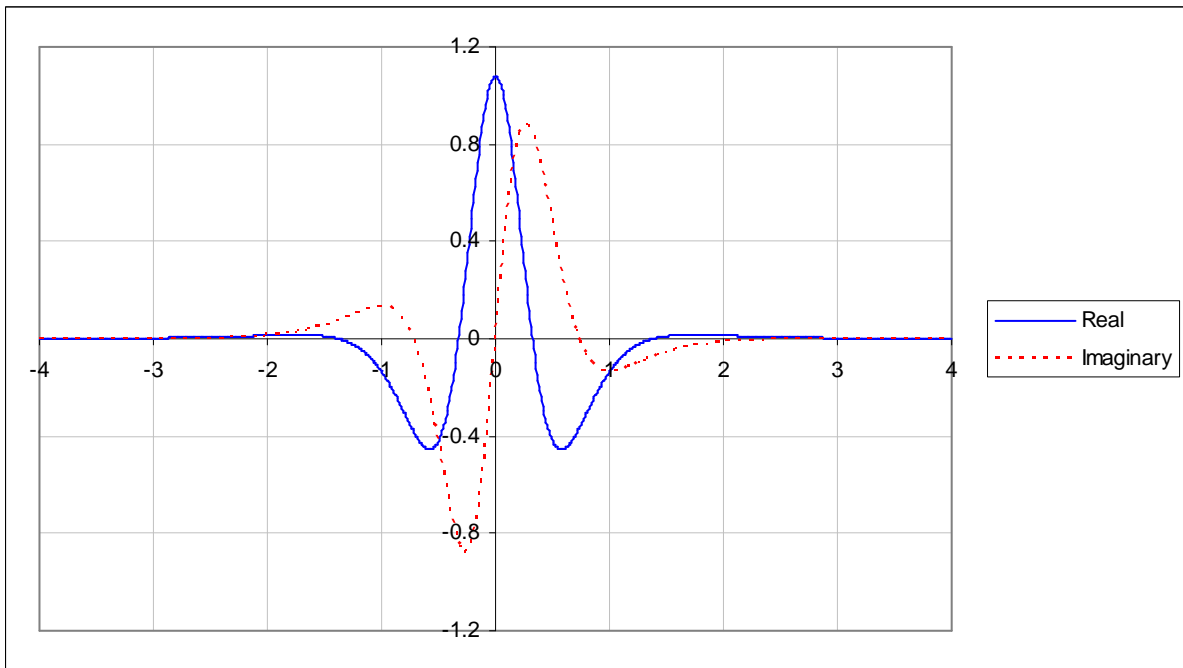


Figure 6.6 – Paul wavelet, order 4

For this wavelet to meet the admissibility requirements (see [Grossman84L] and/or [Kronland87L]) the order of the wavelet, n , should be at least 4. As with the preceding wavelets, the Paul wavelet may be separated into its real and imaginary components. However, the rather ugly algebra required in order to do so has been omitted from all available sources referring to this wavelet, and so a proof for the following polynomial generator has been included in **Appendix B.2** without reference to any source:

$$\psi_0^{P_n}(t) = \frac{2^n n!}{\sqrt{\pi(2n)!}} \left[\Phi_n^{\text{Re}}(t) + i\Phi_n^{\text{Im}}(t) \right] \cdot (1 + t^2)^{-(n+1)},$$

where $\Phi_n^{\text{Re}}(t)$ and $\Phi_n^{\text{Im}}(t)$ are defined recursively as:

$$\begin{aligned} \Phi_0^{\text{Re}}(t) &= 1, & \text{and} & & \Phi_0^{\text{Im}}(t) &= t, \\ \Phi_{n+1}^{\text{Re}}(t) &= -\Phi_n^{\text{Im}} - t\Phi_n^{\text{Re}}; & & & \Phi_{n+1}^{\text{Im}}(t) &= \Phi_n^{\text{Re}} - t\Phi_n^{\text{Im}}. \end{aligned}$$

The Fourier transform of the scaled Paul wavelet is given by the following equation:

$$\hat{\psi}_s^{P_n}(\omega) = \frac{2^n}{\sqrt{n(2n-1)!}} H(\omega)(s\omega)^n e^{-s\omega}.$$

Figure 6.7 shows a graph of this function with a scale of 1:4.

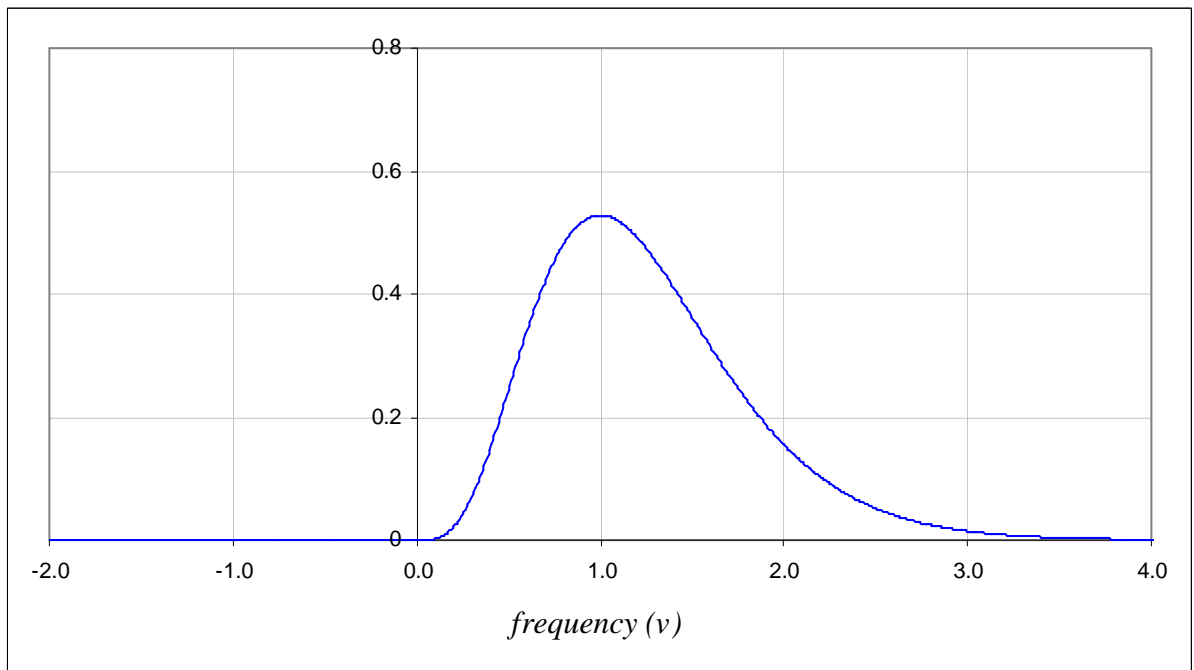


Figure 6.7 – Fourier transform of Paul wavelet, order 4, scale = 1:4

6.2.4 The Shannon Wavelet

The last wavelet of interest, shown in **Figure 6.8**, is named after the father of signal sampling, and is given by:

$$\psi_0^S(t) = \text{sinc}\left(\frac{\pi t}{2}\right) \cos\left(\frac{3\pi t}{2}\right),$$

where $\text{sinc}(t) = \frac{\sin t}{t}$. When t is 0, $\text{sinc}(t)$ is defined as 1 [Wolfram09W].

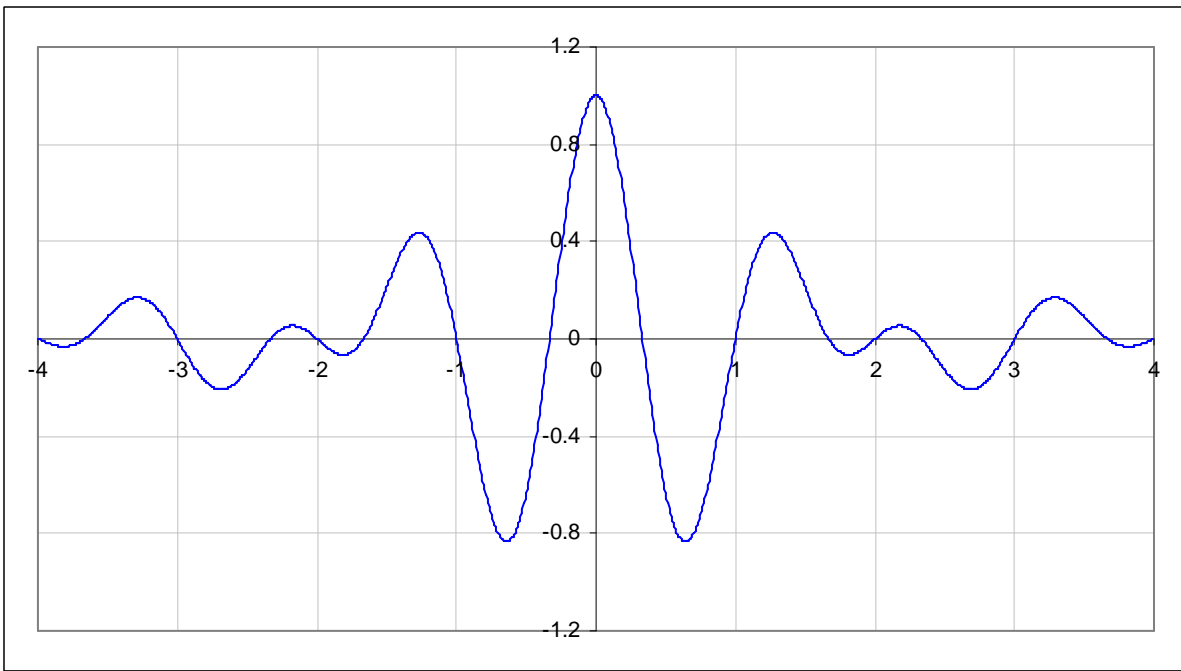


Figure 6.8 – The Shannon wavelet

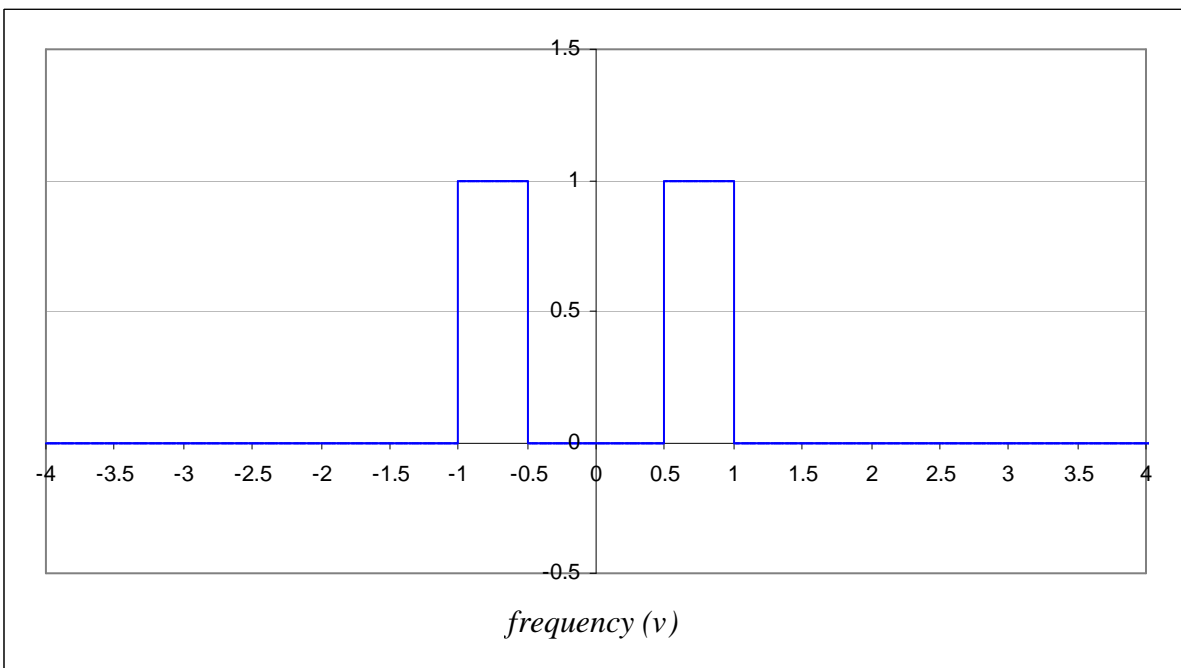


Figure 6.9 – Fourier transform of real Shannon wavelet

Via some trigonometric identities, this wavelet function may also be expressed as:

$$\psi_0^S(t) = 2 \operatorname{sinc}(2\pi t) - \operatorname{sinc}(\pi t).$$

The FT of the wavelet is a simple box function as can be seen in the graph in **Figure 6.9**:

$$\hat{\psi}_s^S(\omega) = \prod\left(\frac{s\omega - 3\pi/2}{\pi}\right) + \prod\left(\frac{s\omega + 3\pi/2}{\pi}\right),$$

where

$$\prod(x) = \begin{cases} 1, & |x| \leq \frac{1}{2}, \\ 0, & |x| > \frac{1}{2}. \end{cases}$$

Note that since this is a real wavelet, there is no Heaviside function included in the FT.

6.3 Implementing the Fast CWT

Translating the above mathematical theory into practical program code requires some careful consideration, especially with regards to the following important issues.

6.3.1 Calculation Time

Since colours in spectrograms are interpolated between the maximum and minimum values in any given transform, it is not necessary to calculate normalizing factors in the FT for any type of wavelet. This saves a bit of calculation time, especially for the Paul wavelet, which has quite a complicated normalizing factor.

For complex wavelets, only half of the points in each row need to be calculated. The rest may all simply be set to zero due to the Heaviside function component which excludes negative values in Fourier space.

Using the Convolution Theorem does save a lot of calculation time, but one still has to be careful to keep the innermost loop as succinct as possible. As seen in the pseudo-code at the end of this section, which is the shell of the CWT algorithm for the Morlet wavelet transform, the loop in question is the one which multiplies each point of the DFT of the signal by the DFT of the scaled wavelet.

With regard to this point, a small time-saver is to do any calculations involving scale before the innermost loop. This means that instead of the calculation being in terms of ν , which is the instantaneous frequency at \hat{f}_m , it is in terms of the frequency sample index, m .

To get each ν from m , we divide by the number of points in the transformed signal and multiply by the bandwidth, i.e. the sample rate. As demonstrated in the pseudo-code, if the division is done once and the multiplication of “`sr_np`” is moved to the calculation of the scale factor, we can avoid having to do $S \times N$ extra multiplications (number of scales \times number of points). Since it is easier to think in terms of analyzing frequencies than scales, and furthermore, both linear and dyadic forms of the transform should be supported, the frequency corresponding to each row is calculated, from which the scale is then derived.

Pseudo-code for the fast continuous Morlet wavelet transform:

```
FCWT(samples[], n_scales, n_points, min_freq, max_freq) {
    LET f = min_freq
    LET df = (max_freq / min_freq) ^ (1 / n_scales)
    LET sr_np = sample_rate / n_points
    FFT(FWD, samples[], n_points)
    FOR row = 0 TO n_scales - 1 {
        LET scale = sr_np / f
        FOR(col = 0 TO n_points / 2) {
            LET x = scale * col - sigma
            LET y = e ^ (-0.5 * x * x) // for Morlet wavelet
            LET cwt[row][col] = samples[col] * y
        }
        FFT(REV, cwt[row], n_points)
        LET f = f * df
    }
    RETURN cwt[][]
}
```

6.3.2 Memory Usage

The CWT contains as many points per scale as there are samples in the signal. Typically, one would want to analyse music at 12 scales per octave as a minimum for semitone accuracy, and preferably double that. Thus, say for three octaves – the range of a grand staff, excluding notes on leger lines – one would want about 72 scales. The maximum number of scales supported in *Wave Processor* for any frequency range is 512. In this case, at a signal sample rate of 44.1kHz, even a 1 second wave requires $44100 \times 512 \times 8$ bytes (the size of a double floating point number) = approximately 172MB, just to store the final values. In fact much more memory than this is needed in the calculation, since for the Fourier transform the number of samples must be a power of two. Further memory is required for temporary input and output arrays for both sets of real and imaginary components of the initial FT of the signal.

Wave Processor handles the memory usage problem by swapping out to a temporary file and doing a transform in sections if more than 256MB is required. In drawing a spectrogram however, the image is much more compressed, and so only the initial plot is slow.

6.4 Comparison of Output

For each of the spectrograms in **Figure 6.10**, the grid lines and time / frequency labels have been removed for clarity, as by now the chirp signal from previous transform examples should be immediately recognizable and the limits and scales of the graphs have not changed. As is clear from the width of the bands of colour in each graph, the Morlet wavelet seems to provide the best localization of frequencies of the four (corroborating other sources such as [DeMoortel06L] which report the same), while the Paul wavelet performs slightly less well than the DoG wavelet, although at even higher orders, the Fourier transform of the DoG is as sharp a peak as that of the Morlet wavelet with the wavenumber used to generate the image shown here.

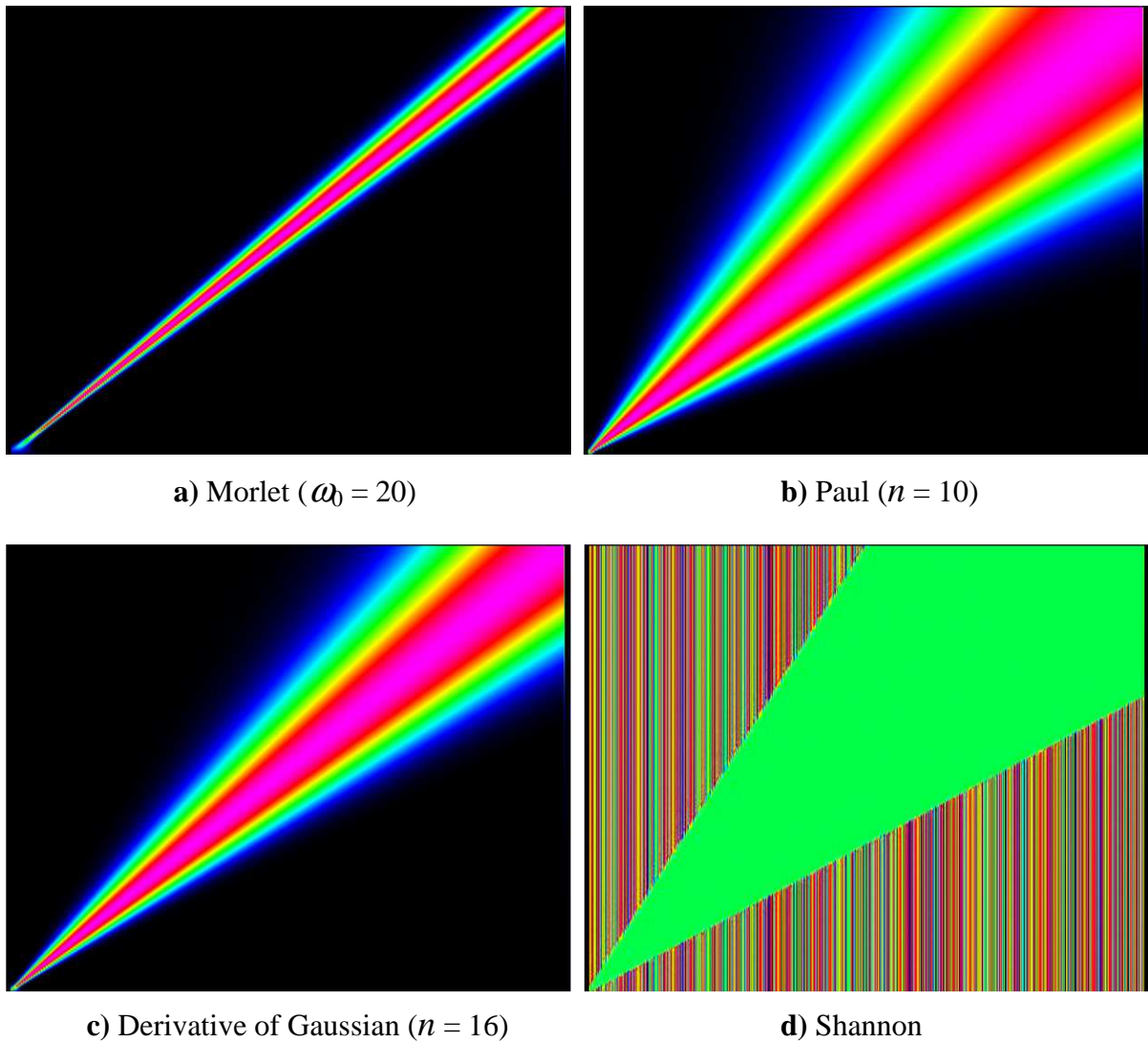


Figure 6.10 – CWT spectrograms of chirp signal using four different wavelets

The appearance of the sharp edge of the solid green band in the Shannon CWT spectrogram is due to the square shape of the wavelet's Fourier transform, and the noise in background is as a result of the slow damping of the wavelet function (see **Figure 6.8**) and hence its lack of compact support. Given the image above, this wavelet may be a good choice for creating a *band pass filter*, which is a device or algorithm that allows certain frequencies in a certain range to pass through while rejecting others. However, it does not seem suitable for music analysis, given the noisy background, which would be difficult to separate from relevant frequency content.

In order to further test the frequency localizing ability of each wavelet, tests were done on the three-part arrangement of *Nkosi Sikeleli Africa* from the previous chapter (**Figure 5.24**). A simple pitch extraction algorithm was constructed, which bears some similarity to McLeod's peak-picking method, but simply involves choosing the highest power levels of the CWT in windowed sections. To calculate the general power of each window at each scale, the average was taken of the magnitudes. For each window, the highest peak was found and then subsequent peaks were chosen which were within a certain threshold of this maximum. **Figure 6.12** shows an example graph drawn by capturing the values across one column of the transformed signal, whose spectrogram is in **Figure 6.11**.

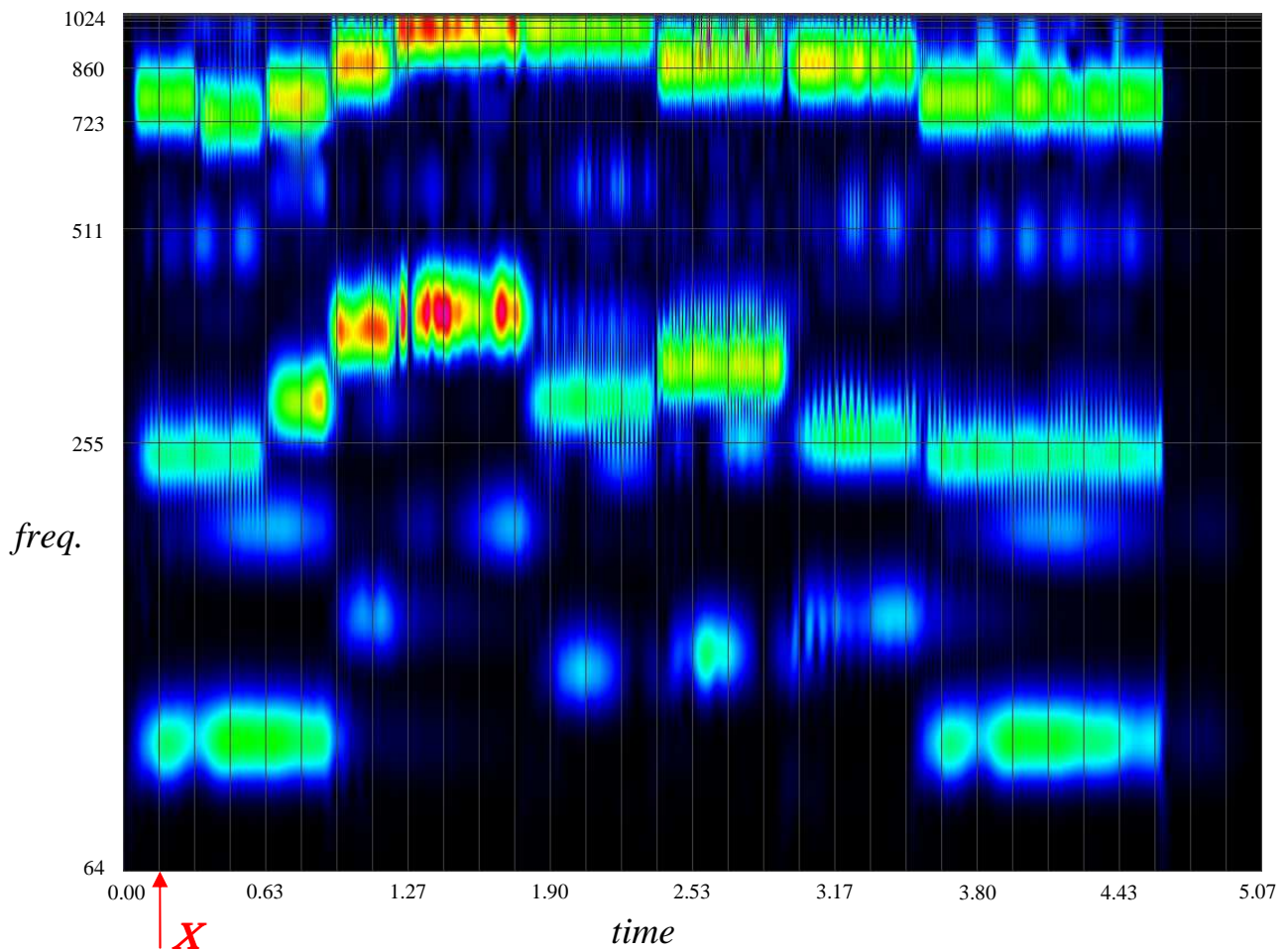


Figure 6.11 – CWT spectrogram of 3-part NSA example using Morlet wavelet ($\omega_0 = 10$)

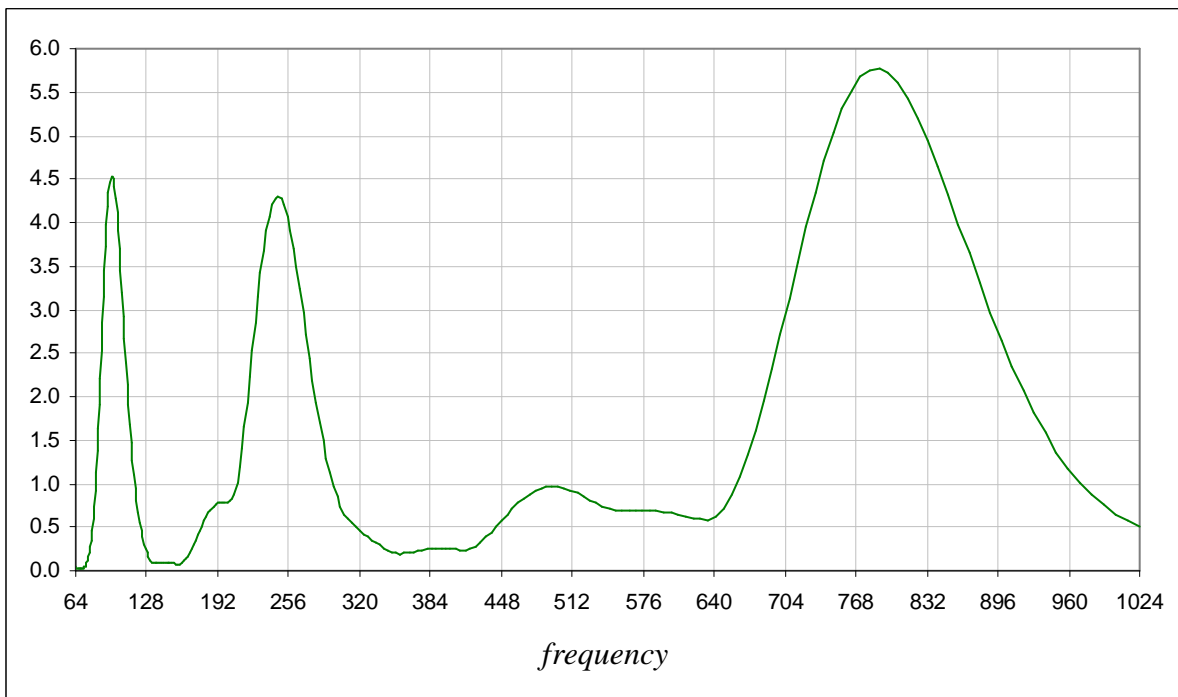


Figure 6.12 – Average powers at each frequency for time window X of Figure 6.11

The three major peaks in the graph occur at corresponding frequencies of 96.7Hz, 246.4Hz and 788.8Hz, which are, equivalently, the musical pitches G_2 , B_3 and G_5 respectively, to the nearest semitone. These are precisely the notes comprising the first chord of the example. In this case, the wavelet which rendered the best pitch extraction, shown in **Figure 6.13**, was the Derivative of Gaussian (order 6), which tended to ignore the slightly weaker harmonics picked up by the Morlet transform. In general it was observed that wavelets with more oscillations, such as the Morlet and higher order DoG and Paul wavelets, are more sensitive to harmonics and high frequencies but less sensitive to bass frequencies. A window width of 2048 was used with a clarity threshold of 53% for the peak-picking stage of the algorithm.

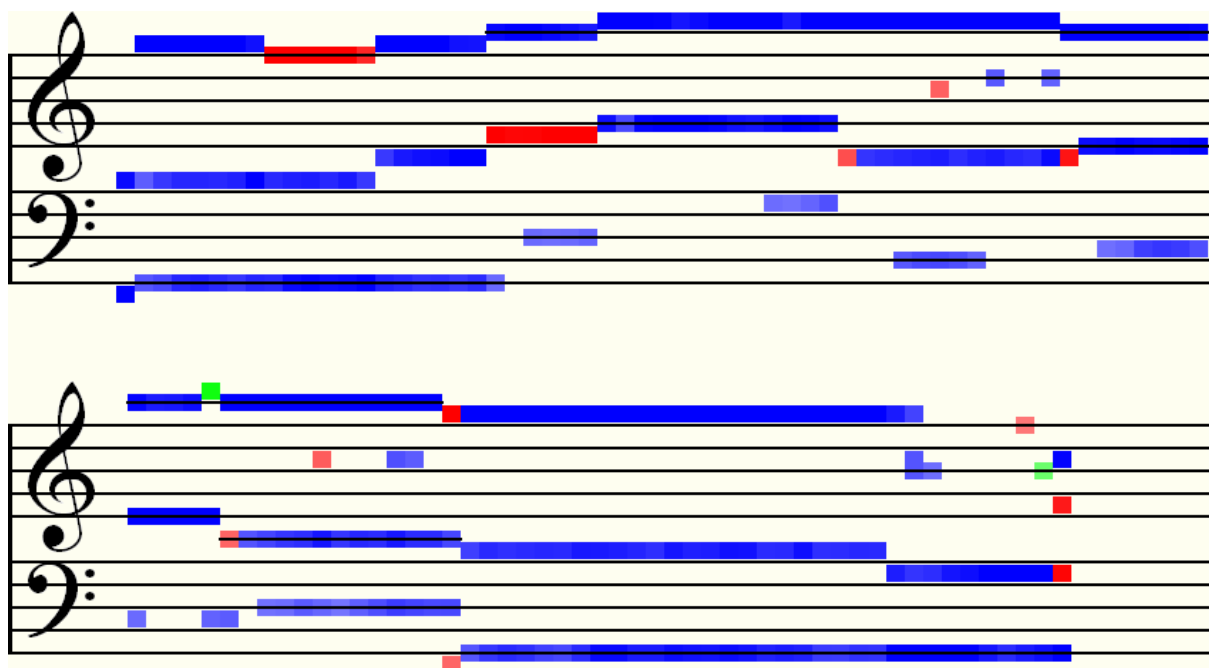


Figure 6.13 – Pitch Extraction of three-part NSA example using DoG wavelet transform

This result is most encouraging, given that the algorithm was allowed to choose up to sixteen peaks, which means it is already a very discerning and precise method, particularly for detecting middle to high range frequencies.

The method does not perform quite so well, however, if frequencies are not well-separated from each other, i.e. wide intervals between notes yield better results. **Figure 6.14** shows the result of the CWT pitch detector on the string quartet example previously experimented upon in Chapter 5 (see **Figure 5.6**). For the CWT, the Morlet wavelet with a wavenumber of 16 was used.

In actual fact, the result below is only this good because the upper analysis frequency bound was lowered, allowing the algorithm to reject many strong harmonics outside of the narrow band from 64Hz to 512Hz (approximately C_2 to C_5). This, however, did not stop the algorithm from picking up the first and last B_3 in the viola part as a strong harmonic at B_4 . It can also be seen, from the jumble of artefacts at the start and end, how sensitive the transform is to noise, which does not bode well for poorer quality recordings, or music such as this which is rich in harmonics. Despite these issues, the algorithm constructed so far managed to pick up every pitch that is present in the original score. This is a very good starting point on which to base a robust pitch detection method.

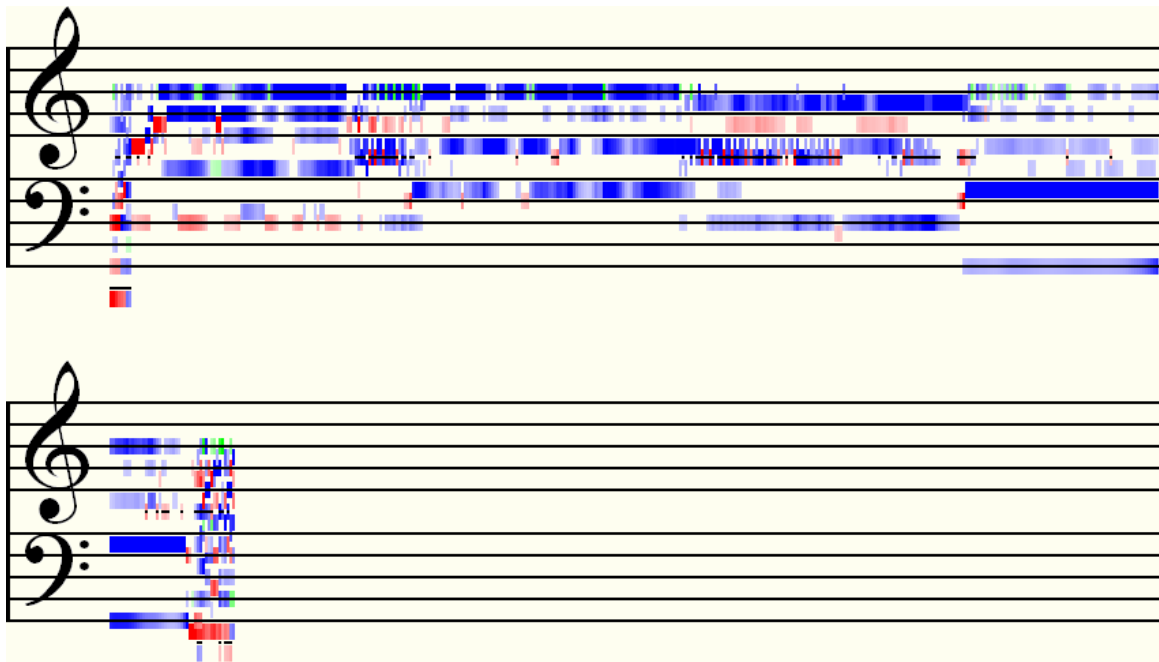


Figure 6.14 – Pitch extraction of quartet example with band-limited Morlet transform

The following chapter takes a closer look at the CWT spectrogram and offers some further insights for refining the above results, which could lead to a much improved multiple pitch detector than previously demonstrated in this study.

7 Interpreting the CWT Spectrogram

As seen in the previous chapter, the CWT is capable of extremely accurate pitch detection, but again, its application is only really suitable for music containing wider intervals and instruments of simple timbre. If the CWT is not interpreted correctly, more naïve methods of pitch extraction may not perform satisfactorily when notes are closer together in pitch.

7.1 The Scale of the CWT Spectrogram

It is important to note that the spectrogram has identical resolution in time to the signal of which it is a transform. **Figure 7.1** shows the dyadic spectrogram of the *Nkosi Sikeleli Africa* quartet example analyzed previously (see **Figure 5.6** and subsequent transcriptions) zoomed in to the sixth chord, which comprises the notes G_3 , D_4 , G_4 and B_4 . The image was generated using the Morlet wavelet with a wavenumber of 10.

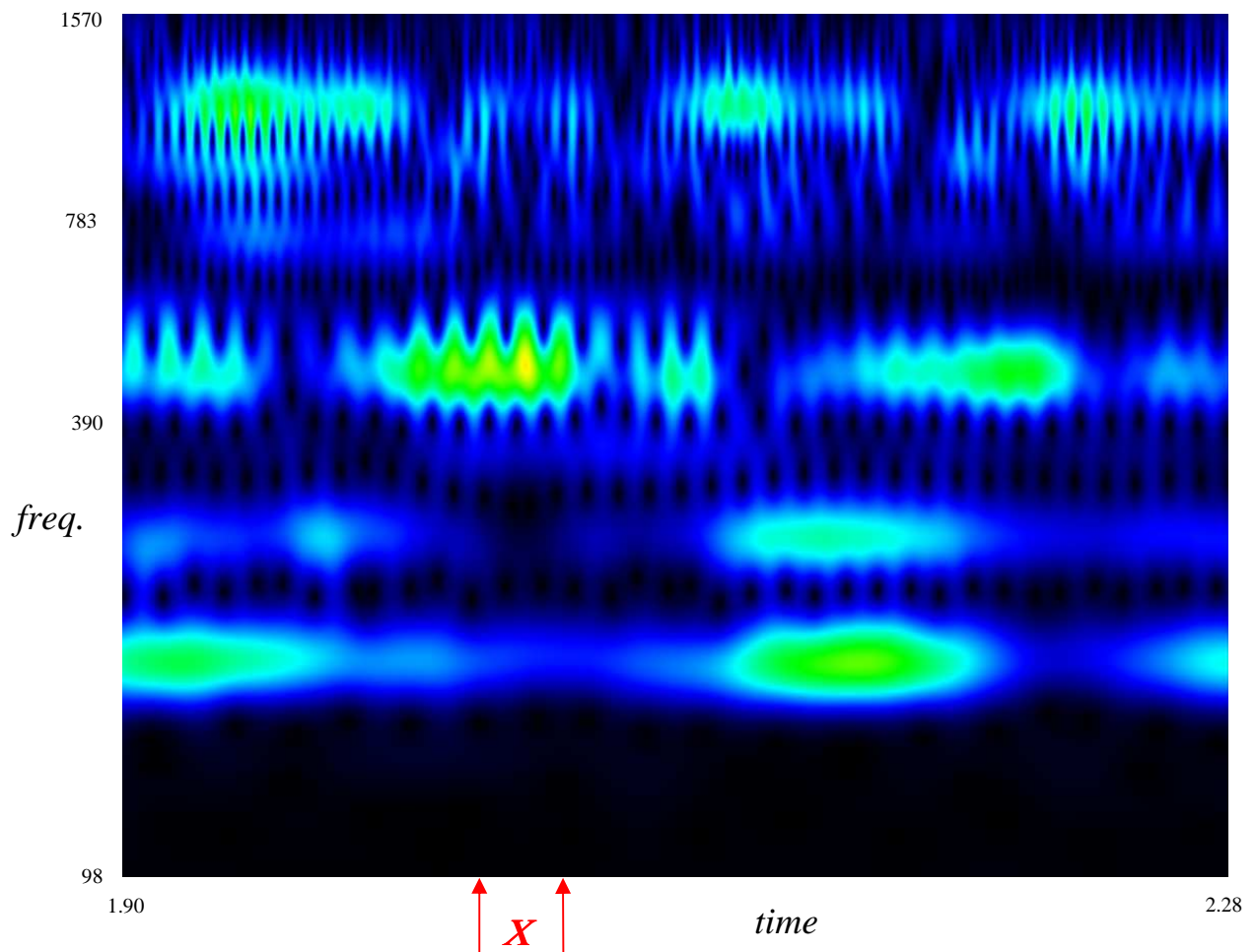


Figure 7.1 – CWT spectrogram of fourth chord of NSA quartet example

Much detail is to be observed in this image, especially in the upper frequency range, which is mostly harmonics in this case. The top two notes in the chord, G_4 and B_4 , which have frequencies of 392.6Hz and 494.6Hz respectively, are seen to be overlapping. The exact location of their peaks in the frequency domain is therefore unclear and presents a problem.

We can zoom in even further on the time axis: **Figure 7.2** shows the spectrogram in a time window, X , demarcated in the previous figure by the red arrows. The width of this window is just 1024 samples and this image is now at the highest level of detail.

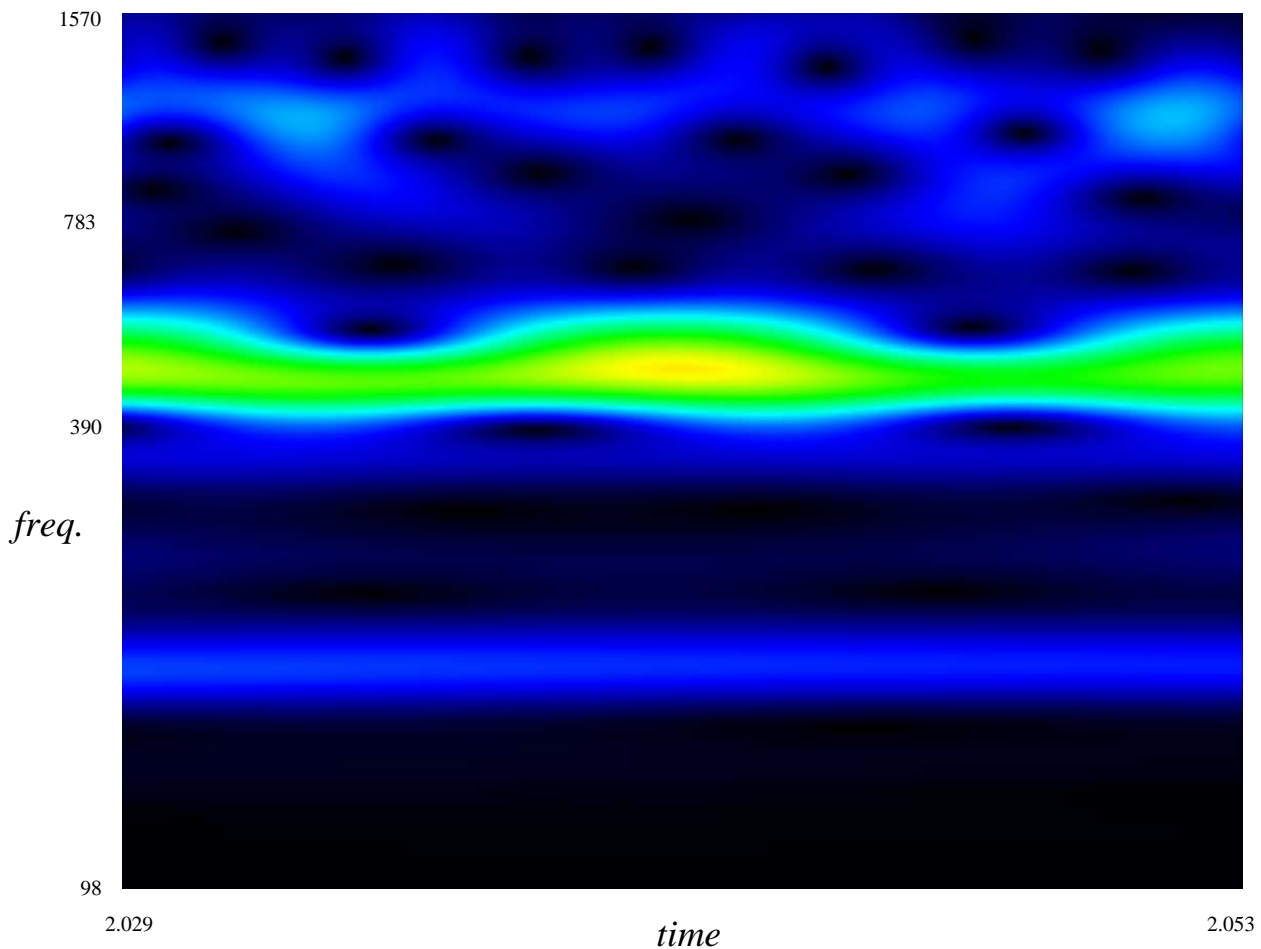


Figure 7.2 – CWT spectrogram of detail X

It is very clear from the detail in these time stretched spectrograms that the frequency magnitudes in each row of the transform are not constant, but vary at certain frequencies of their own. In the short time frame in **Figure 7.2** the bass and tenor notes appear to be at a point of low power, while the overlapping treble and soprano voices exhibit peaks in the middle and at the edges of the window. The power variation of the latter pair of voices appears to have a shorter cycle, and in general it may be observed that the higher the frequencies the faster the modulation of their signal strength.

If, however, we examine a pure stationary sine wave at the same level of detail, the result is a constant signal. This is demonstrated in **Figure 7.3**, which shows 1024 samples of a CWT spectrogram of the wave function $f(t) = \sin(2\pi vt)$, where $v = 440\text{Hz}$ (the pitch A_4).

If more frequencies are added to the audio signal, similar patterns to those seen in **Figure 7.2** begin to emerge. **Figure 7.4** is a similar CWT spectrogram of a stationary wave function with constituent frequencies of 440Hz, 524Hz and 623Hz.

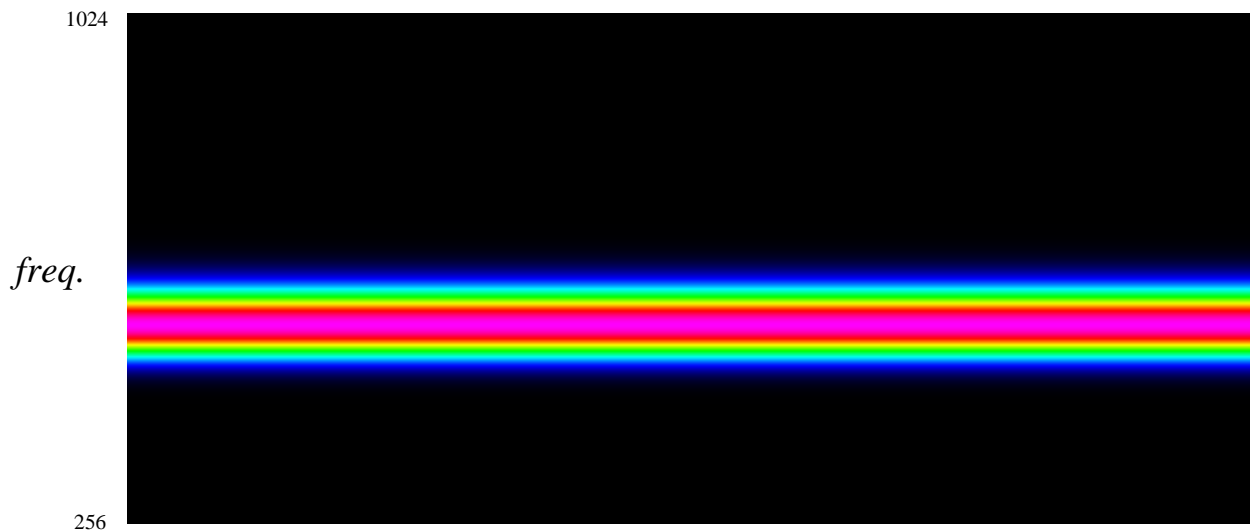


Figure 7.3 – CWT spectrogram 440Hz signal (width = 1024 samples)

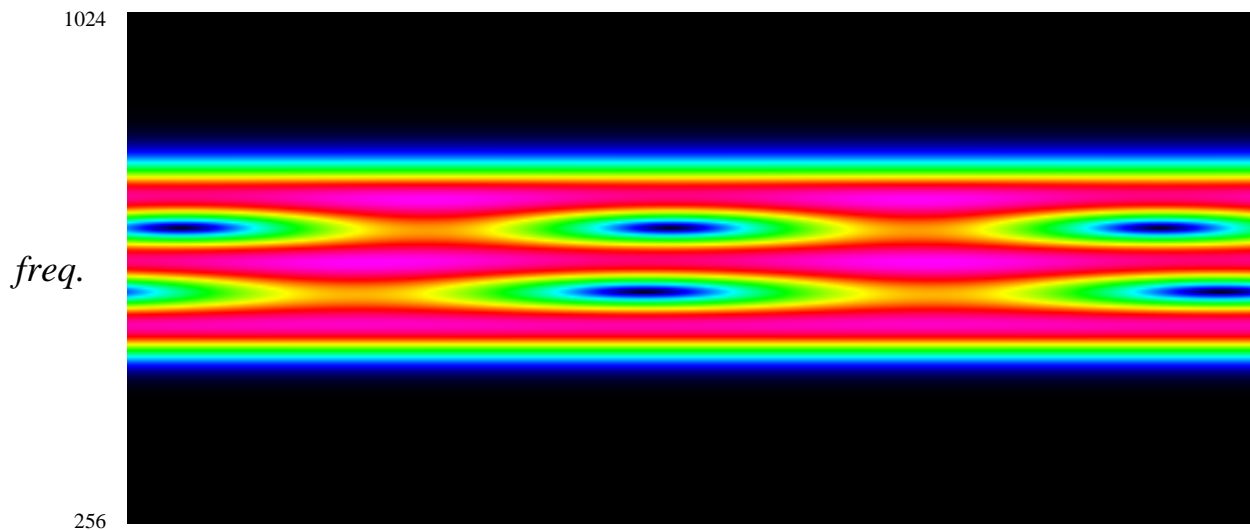


Figure 7.4 – CWT spectrogram of composite stationary signal (width = 1024 samples)

These images are evidence that the varying magnitudes across each row of the CWT are caused by interference of different frequencies with each other. The interference patterns may be explained by a phenomenon in the field of acoustics known as *beats*.

7.2 Beats

Usually when talking about beats, we are referring to the pulse which indicates tempo, rhythm and meter of a piece of music. In acoustics, beats are the faintly audible knocking sound caused by the interference of two waves oscillating at slightly different frequencies [Scholes65L]. The reason for the varying signal strength is due to the changing phase between added frequencies. Consider **Figure 7.5**, which shows superimposed sinusoids of 6Hz (blue) and 4Hz (red). At $t = 0, \frac{1}{2}$ and 1, the phase of the two waves coincide, and so around these points, the sum of their amplitudes results in a strong signal. However, around the $\frac{1}{3}$ and $\frac{2}{3}$ points, the two signals cancel each other out, since their phase difference moves through the maximum of 180 degrees.

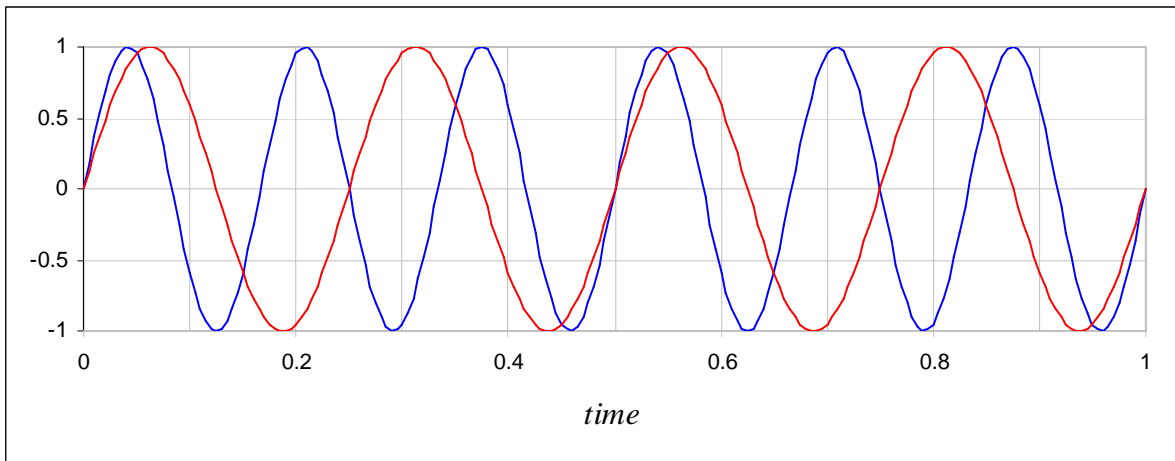


Figure 7.5 – Comparison of 6Hz and 4Hz sinusoids

The occurrence of acoustic beats is well known and has been used by musicians for centuries in order to fine tune pianos and organs [Scholes65L]. The composer and violinist Giuseppe Tartini (1692 – 1770) was the first to describe beats as “the third sound” (*il terzo suono*). The reason he called it this is because when a beat frequency is fast enough, it becomes audible as an extra tone underneath the two pitches causing it. Tartini discovered that by listening carefully for this note, he could ensure that his double-stopping* was perfectly in tune [McLeod05L, Scholes65L]. He knew (but could not explain why) that the third tone could be used as a precise measure of the interval between any two notes played simultaneously.

7.3 Difference Tone Analysis

Using the simple case of sine waves, the following shows, mathematically, why beats are also known as *difference tones* in acoustics:

Given two signals with different frequencies, $f(t) = \sin(2\pi\nu_1t)$ and $g(t) = \sin(2\pi\nu_2t)$, $\nu_1 \neq \nu_2$, then

$$f(t) + g(t) = \sin(2\pi\nu_1t) + \sin(2\pi\nu_2t).$$

By the sum-to-product trigonometric identity for $\sin(\theta_1) + \sin(\theta_2)$,

$$f(t) + g(t) = 2 \cos\left(\frac{2\pi(\nu_1 - \nu_2)t}{2}\right) \cdot \sin\left(\frac{2\pi(\nu_1 + \nu_2)t}{2}\right). \quad [7.1]$$

Dissecting equation [7.1], we can see that this is a sinusoidal signal with a general frequency that is the average of the two integral frequencies, ν_1 and ν_2 . Its amplitude is controlled by the cosine component, the frequency of which is the half the *difference* between the two frequencies. This gives the shape of the signal’s envelope and causes the periodic variations in its magnitude, hence the beat.

* This is a technique on the violin whereby two or more strings are stopped on the fingerboard and played simultaneously, producing a chord.

In **Figure 7.6**, the dashed wave is the sum of the 6Hz and 4Hz sinusoids from the previous example. The red wave is the sine component and the blue wave is the cosine component, assuming that $\nu_1 = 6\text{Hz}$ and $\nu_2 = 4\text{Hz}$.

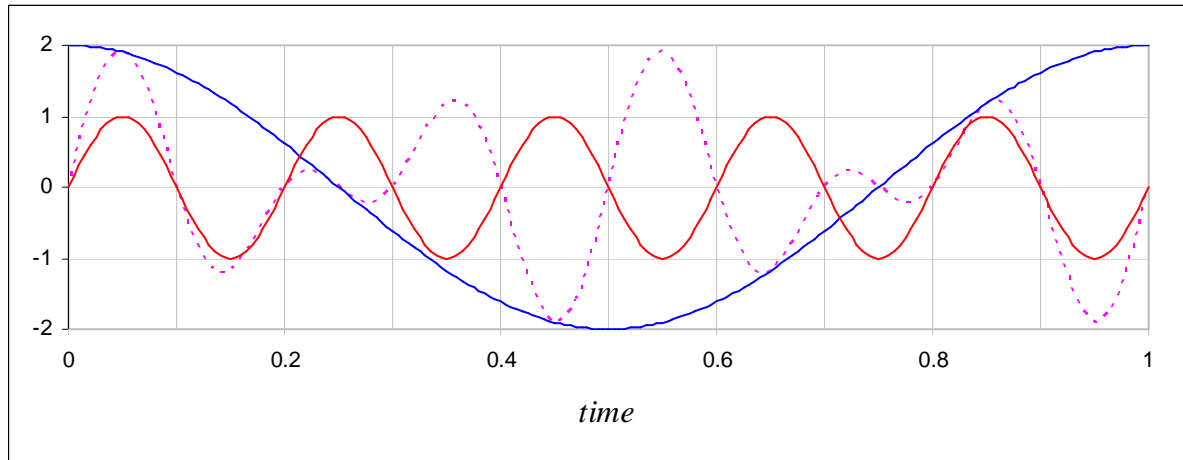


Figure 7.6 – Sine and cosine components, $\nu_1 = 6\text{Hz}$, $\nu_2 = 4\text{Hz}$, and their product

Acoustics tells us that the actual beat frequency, however, is the difference between ν_1 and ν_2 [Howard06L] and not half the difference, as equation [7.1] would seem to suggest. Recall from Chapter 2, section 2.3 the discussion about the symmetry of a Fourier transform: Frequencies may, in theory, be positive or negative. It may be argued that since we can also let $\nu_1 = 4\text{Hz}$ and $\nu_2 = 6\text{Hz}$, the sign of the beat frequency in this case should be negative, since $4 - 6 = -2$. N.B. We are not arguing that $\cos(\alpha)$ is positive and $\cos(-\alpha)$ is negative because clearly this is false. Instead we are saying that if there exists a beat frequency, $\nu = \cos(\alpha)$, then Fourier theory predicts there will also be a complementary signal, $\nu' = -\cos(\alpha)$.

This is clarified if one considers the half-angle formula for cosine, extending [7.1] as follows:

$$f(t) + g(t) = 2 \left(\pm \sqrt{\frac{1 + \cos(2\pi(\nu_1 - \nu_2)t)}{2}} \right) \cdot \sin\left(\frac{2\pi(\nu_1 + \nu_2)t}{2}\right). \quad [7.2]$$

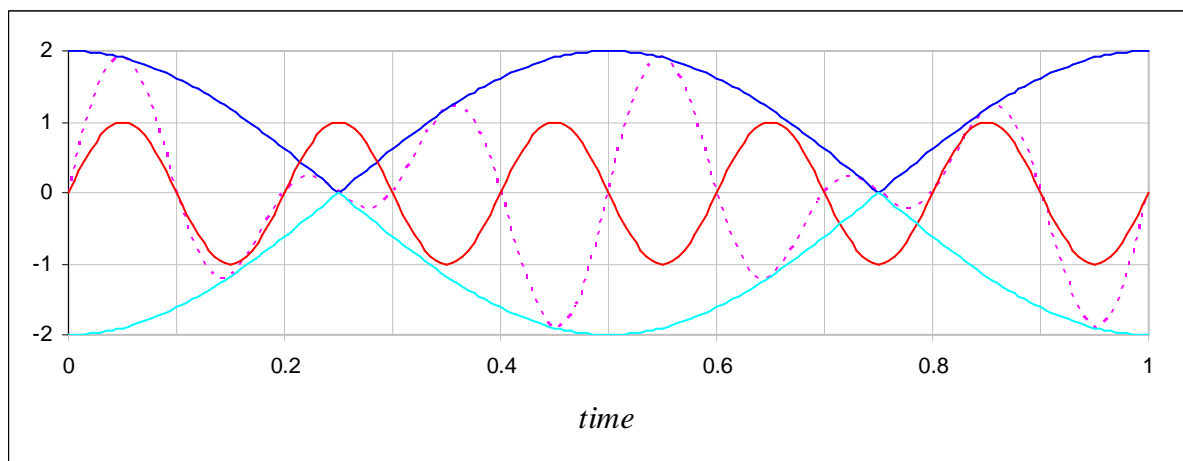


Figure 7.7 – Graph including both roots of the cosine component

If we plot both of these roots on the same graph, as in **Figure 7.7**, we see the resulting double envelope which has a frequency of not 1 but 2Hz. In order to complete the picture, the product of the sine wave with both roots of the cosine component is shown in **Figure 7.8**.

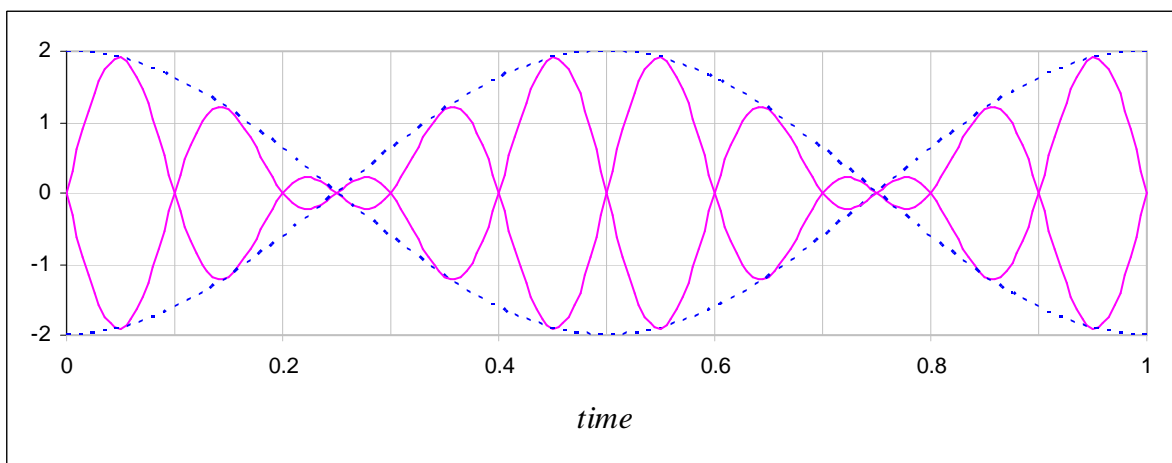


Figure 7.8 – Difference tone showing double cosine envelope

In telecommunications theory, this type of signal is fairly common. It is used in television broadcast, for example, to vary pixel colour levels in transmitted images. The technique of varying the strength of a carrier signal in this way is known as *amplitude modulation*. In the case of beats, the carrier is the sine part of equation [7.2] which we shall denote by χ . As an example, taking two of the frequencies in the signal from **Figure 7.4**, 440Hz, 524Hz, if the beat frequency, β , is the difference between the two, then:

$$\beta = 524 - 440 = 84, \quad \text{and} \quad \chi = (524 + 440) \div 2 = 482.$$

Here, β is a low (yet audible) frequency, since β the interval between the two notes is small.

In **Figures 7.5** and **7.8** the beat frequency can clearly be seen from the coincidental phase points and the resulting envelope as being 2Hz – half second knocks. It is also apparent in **Figure 7.4** that almost two cycles of the beat signal between the lower pair of bands fit in 1024 samples. Two cycles of 84Hz last about 0.024 seconds, which, at a sample rate of 44.1kHz, is 1050 samples; this does seem to match the visible amplitude modulation at a glance. In order to be absolutely certain that this observation is correct, however, we should apply a Fourier transform to rows of the spectrogram where the modulation is greatest to see if the predicted beat frequencies are indeed present in the transform. The Fourier transform of a spectrum such as this as known as the power *cepstrum* [Bogert63L].

As a thorough test, 24 sinusoid waves were generated, each containing a base pitch of C₄ (262Hz) and a second note above this, in increasing quarter-tone intervals. The final interval is an octave: C₄ to C₅. Spectrograms were generated* using the Morlet wavelet with a suitably high wavenumber (20) so as to render thin bands – well-localized frequencies. For each transform, a power graph was created, similar to that of **Figure 6.2**, except the average power across the whole signal was used, instead of in a window. **Figure 7.9** shows a graph for the interval of C₄ plus 8 quarter tones (E₄), which is equivalent to a major third.

* An animation of the 24 spectrograms, representing an ascending and descending scale, may be seen on the project CD at Sound\Quarter Tone Intervals\animation.wmv

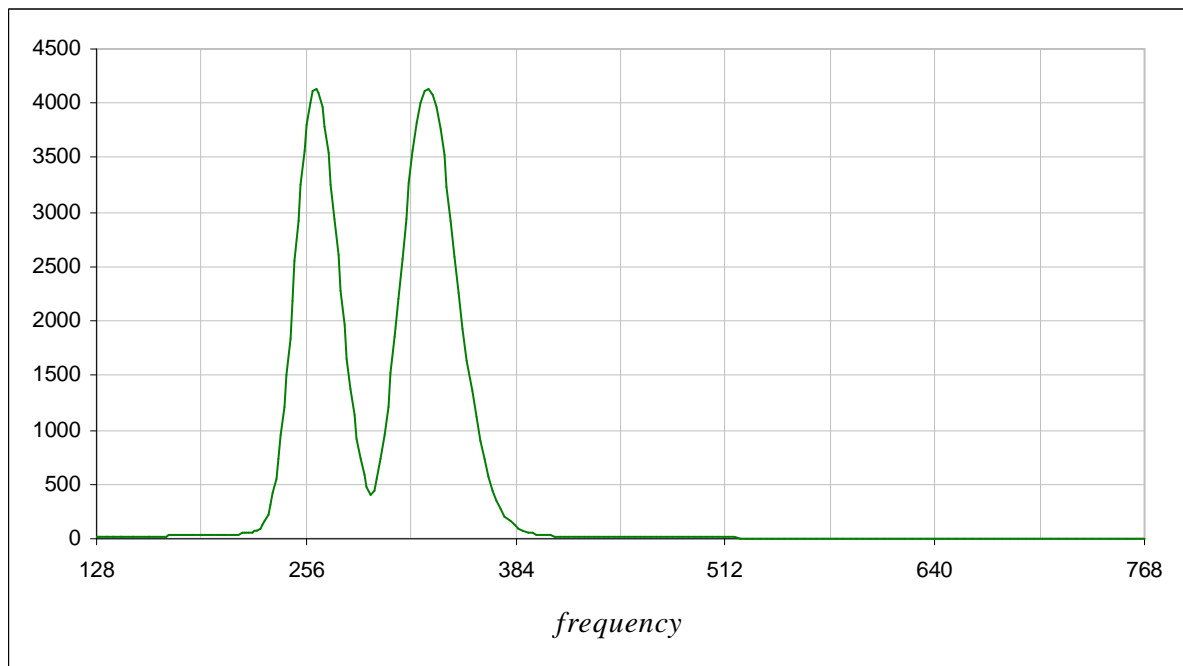


Figure 7.9 – Average power distribution per frequency for the interval C₄ – E₄

The two peaks in the graph correspond to frequencies of 262.1Hz and 330.5Hz, which are, as expected, good estimates for the constituent frequencies in this example. The trough in between corresponds to a frequency of 295.357Hz. If we take the average of the peak frequencies, we find that it is equal to this value. It is precisely this halfway point that equation [7.1] also predicts the beat frequency will be defined, i.e. it is the location of χ . Therefore this is the row of the transform which should be analyzed. **Figure 7.10** shows a Fourier transform of the relevant row in the current example interval.

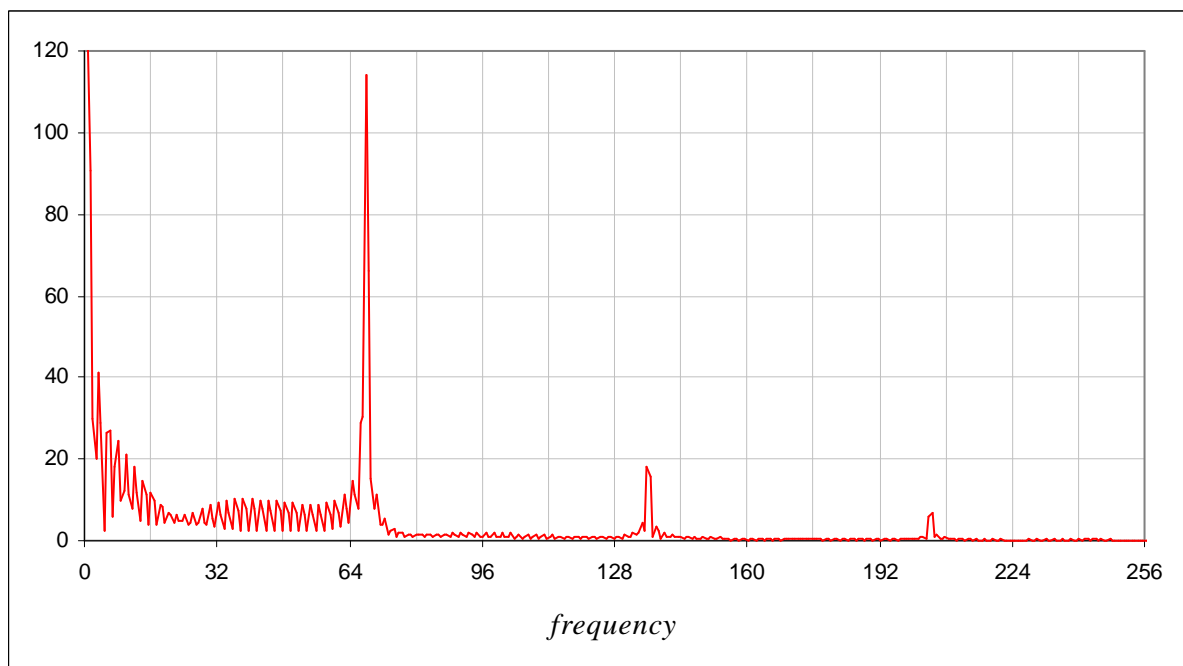


Figure 7.10 – FT showing a peak at the beat frequency for the interval C₄ – E₄

The peak at 0Hz in the graph is explained by the fact that this is a Fourier transform of strictly positive values. This peak may be eliminated by shifting the bias of the signal back to zero, i.e. making zero the average amplitude. The falloff from 1Hz is also logically explained by the fact that the signal begins at 0 seconds and ends at 1 second, and so also has a one second cycle according to the Fourier transform. This problem may also be solved by applying a Gaussian function to the signal prior to doing the FT (see Chapter 2).

Taking similar FTs of the row of each spectrum halfway between the two rows of highest energy, the set of strong low frequencies in **Table 7.1** was obtained for each interval from 262Hz.

Upper Frequency (Hz)	¼ tones from C ₄	Actual frequency difference (Hz)	Measured frequency of difference tone (Hz)	Error (Hz)
269.677	1	7.677	-	-
277.579	2	15.579	15.477	-0.102
285.713	3	23.713	23.552	-0.161
294.085	4	32.085	32.300	0.215
302.702	5	40.702	40.375	-0.327
311.572	6	49.572	49.796	0.224
320.702	7	58.702	58.543	-0.159
330.099	8	68.099	67.964	-0.135
339.772	9	77.772	78.058	0.286
349.728	10	87.728	87.479	-0.249
359.976	11	97.976	(98.245)	0.269
370.524	12	108.524	(108.339)	-0.185
381.381	13	119.381	(119.106)	-0.275
392.557	14	130.556	(130.545)	-0.011
404.059	15	142.059	-	-
415.899	16	153.899	-	-
428.086	17	166.086	-	-
440.630	18	178.630	-	-
453.541	19	191.541	-	-
466.831	20	204.831	-	-
480.510	21	218.510	-	-
494.590	22	232.590	-	-
509.083	23	247.083	-	-
524.000	24	262.000	-	-

Table 7.1 – Difference tones for quarter-tone intervals above C₄

Note that after the interval from C₄ to G₄ (14 quarter-tones), the strength of the difference tone diminishes considerably and only appears as a small bump in the Fourier transform, which cannot reasonably be counted as a clear peak. In fact the bracketed values in the table were also rather low peaks in comparison to those of lower frequencies.

We can be content, however, with only being able to detect difference tones for sufficiently close intervals, since this is really the only time when it becomes necessary to do so. Beyond the interval of about a perfect fourth, two frequencies are separated enough as to be independently determined by their position in the frequency domain alone.

The missing difference tone for the first quarter tone interval was due to the fact that the peak-finding algorithm did not detect two clear peaks between which to locate a trough, since the peaks merged into one, being very close together. For now, this is not too much of a problem, however, and eventually it will be seen that the final algorithm does not depend on this issue.

We could also use the McLeod Pitch Method in order to measure the beat frequency more accurately, except that the algorithm does not work for strictly positive signals such as this. **Figure 7.11** shows a window of the difference tone signal for which **Figure 7.10** was the Fourier transform. In a way, this is similar to taking **Figure 7.8** and removing all points below the time axis, since the signal comprises only positive magnitudes. It would have to be re-biased so that it is centered around zero before being passed to the MPM function.

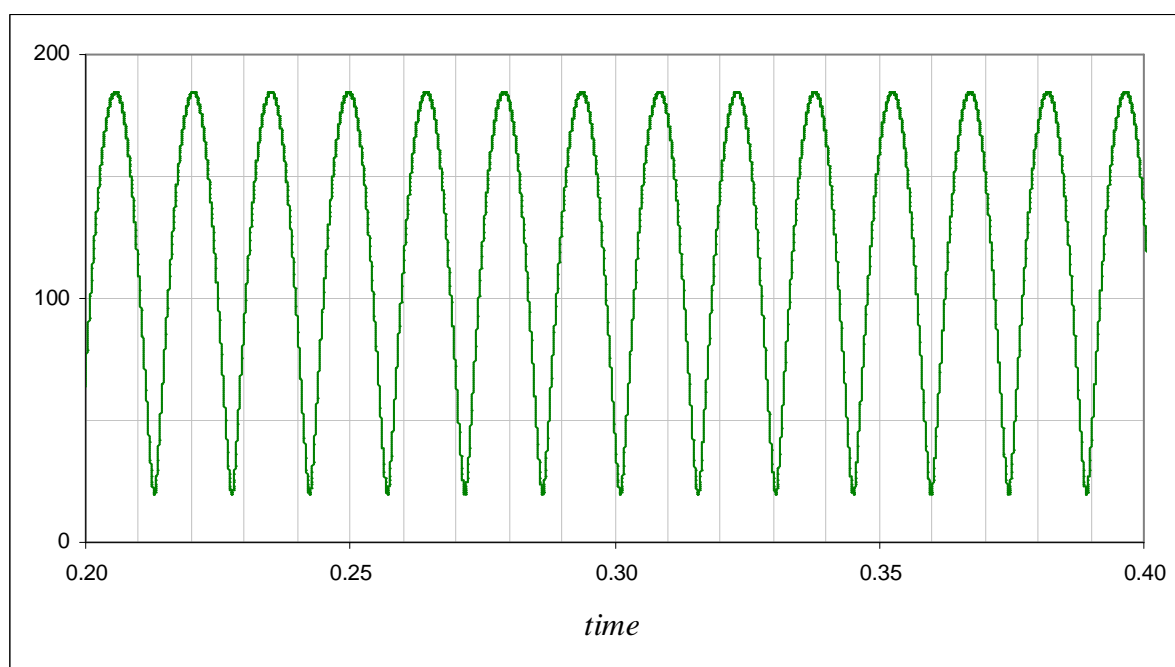


Figure 7.11 – Difference tone signal for the interval $C_4 - E_4$

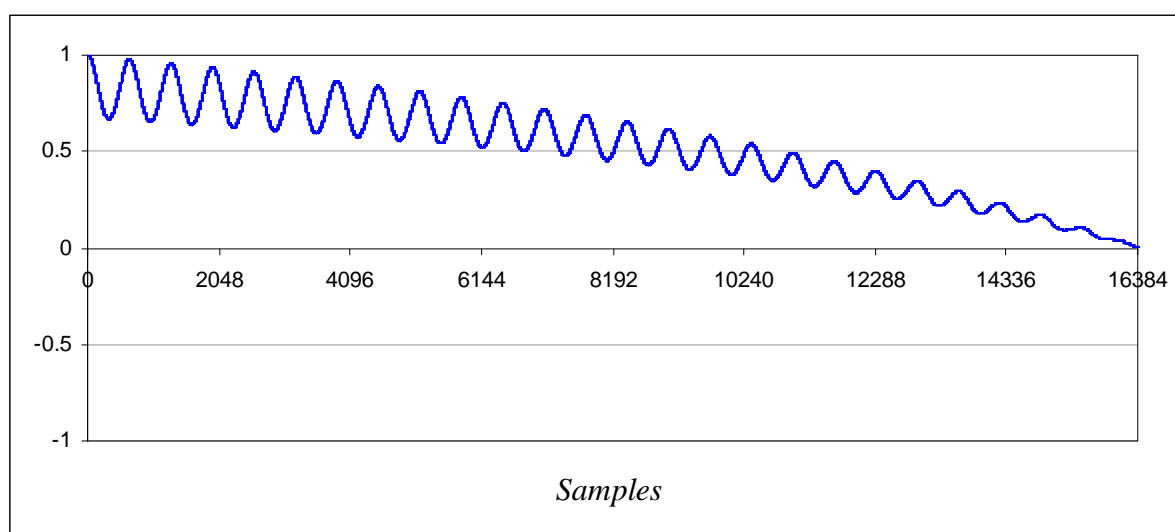


Figure 7.12 – NSDF output for difference tone signal for the interval $C_4 - E_4$

The graph of the normalized square differences for this window, shown in **Figure 7.12**, reveals why MPM is unsuitable for determining the period of the beat signal. A signal with no negative values will yield only strictly positive normalized square differences, since autocorrelation of positive values is also always positive. MPM is designed to look for the first zero crossing point and then take the first peak after that [McLeod05L]. Since in this case the graph never crosses over zero, the algorithm will fail to find this first peak.

We can also get a finer estimation of the beat frequency by applying the Phase Vocoder algorithm to a windowed FT, but in actual fact, these advanced methods are not necessary. Close examination of another example difference tone signal, shown in **Figure 7.13**, reveals that all we have to do is measure the average distance between successive peaks in the signal, i.e. get the shortest wavelength present directly from the time series.

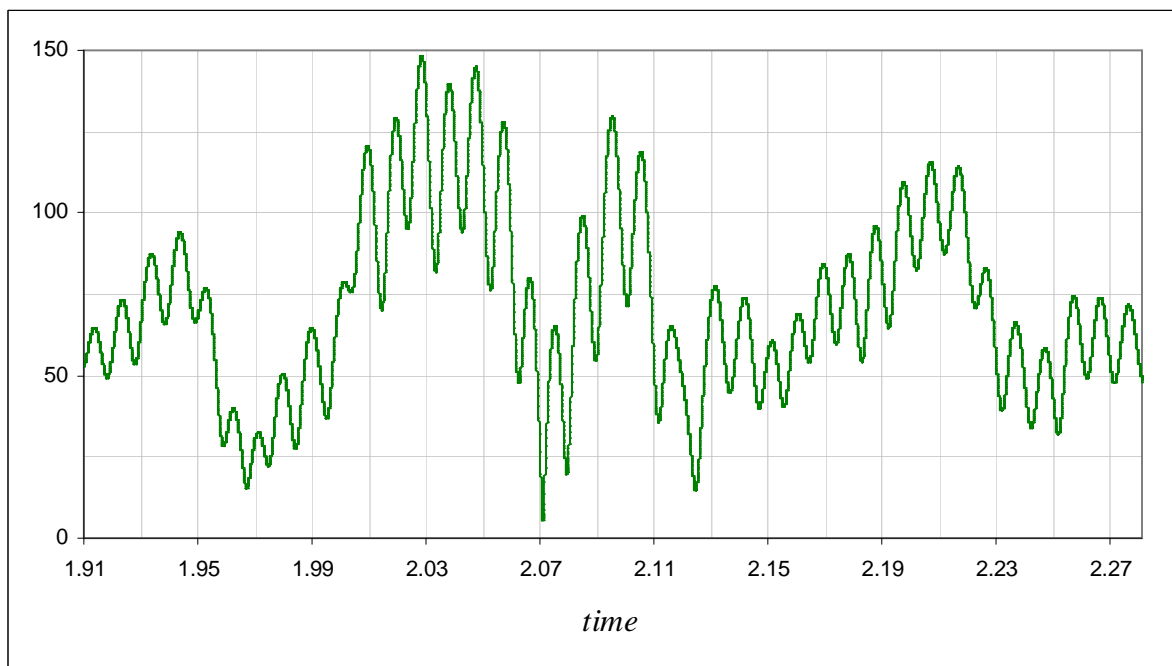


Figure 7.13 – Difference tone signal across row 139 of Figure 7.1 spectrogram

This example is the cross section of row 139 of the window shown in **Figure 7.1**. The frequency to which this row corresponds is 444.520Hz. If this theory is correct, then we should get a measure close to the difference tone for the interval $G_4 - B_4$, since this row's frequency is close to being their average and therefore an approximation of the carrier, χ . The frequency difference, β , between the two notes is $494.590 - 392.557 = 102.033$. We don't even have to measure the peaks differences in this particular case, although doing so gives better precision and is the preferable method. Counting the peaks, there are 38 of them. 16384 samples at 44100 samples per second means that the window's duration is 0.372 seconds. So, in one second, the number of peaks will be $38 \div 0.372 = 102.151$.

This is an extremely accurate measure, with an error comparable to those in **Table 7.1**. Furthermore, this is a much faster method, since we never have to leave the time domain. Using a large window such as the one here is always a good idea, since from **Table 7.1**, beat frequencies which are detectable for mid-range pitch intervals are mostly low-frequency: the more wavelengths available to measure, the more precise the estimation of β .

7.4 An Improved Pitch Detection Algorithm

Applying the above technique of detecting difference tones at any scale in the CWT spectrum now gives us a method of unmixing overlapping frequencies, by creating useful data from the apparently confusing interference patterns in the spectrogram. In the example in the previous section, we knew where to look for certain difference tones because we knew the frequencies of the pitches causing them *a priori*. If we need to reverse the procedure, strong difference tones could be searched for first, and then the frequencies which caused them may be found by the following:

If two frequencies ν_1 and ν_2 interfere with each other and create a difference tone with central frequency χ and amplitude modulation frequency β , then from equation [7.1]:

$$\nu_1 + \nu_2 = 2\chi, \quad [7.3]$$

and

$$\nu_1 - \nu_2 = \beta. \quad [7.4]$$

Since we can measure both χ and β , ν_1 and ν_2 may be determined from these simultaneous equations. We may solve for ν_1 and ν_2 in general:

$$\nu_1 = \chi + \frac{\beta}{2}, \quad [7.5]$$

and

$$\nu_2 = \chi - \frac{\beta}{2}. \quad [7.6]$$

Using the $C_4 - E_4$ interval example again, as detected, $\chi = 295.357$ and $\beta = 67.964$. Substituting these values into equations [7.5] and [7.6].

$$\nu_1 = 329.339,$$

and

$$\nu_2 = 261.375.$$

This is a good approximation for the frequencies of these simple sine waves, but let us now apply this technique to the same large window of the four-voice chord, as extracted previously in **Figure 7.1**. First, another CWT was taken of the signal using the Morlet wavelet with a high wavenumber of 20, so that power maxima for rows of the transform could be found easily.

As seen in **Figure 7.14**, there are several peaks for this chord.

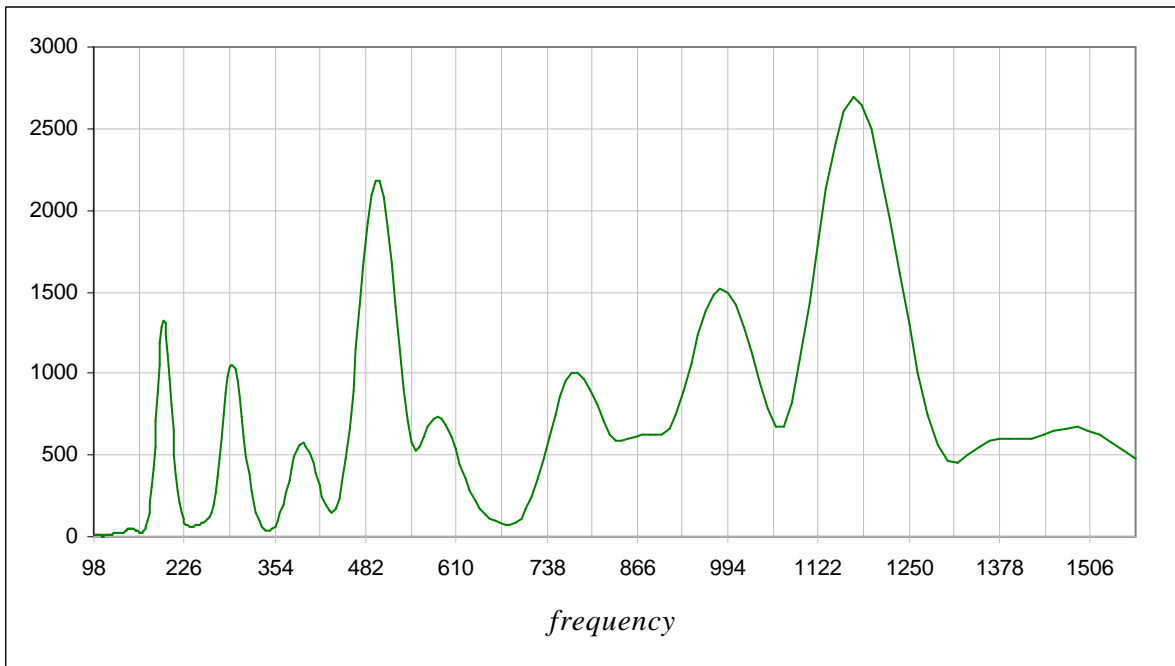


Figure 7.14 – Average power distribution per frequency for NSA quartet, chord six

The peaks in the graph were found to correspond to the following frequencies and pitches:

Peak	Frequency	Nearest Pitch
1	196.5964	G ₃
2	294.0162	D ₄
3	394.3892	G ₄
4	495.6024	B ₄
5	583.4404	D ₅
6	782.6187	G ₅
7	983.4643	B ₅
8	1170.431	D ₆

Table 7.2 – Frequencies and nearest pitches corresponding to peaks in Figure 7.14

This is already a good detection of the pitches in the chord, despite the inclusion of four strong harmonics as well. We can corroborate these results by looking for difference tones in the locations which they predict.

A Morlet transform was taken again, but with a lower wavenumber (10) for better time localization. The difference tones in **Table 7.3** were detected by measuring the average wavelength between peaks along rows of the CWT for chord six. Rather than just obtaining one value, the 16384 points across each row were divided up and frequencies were found in four windows of 2048 samples each. These were weighted according to their corresponding clarities. Clarity was measured by comparing the height of peaks in a window with the maximum power for that window. The average clarity for all windows across each row is shown in the table. The chosen rows correspond closest to the averages of adjacent pairs of frequencies in **Table 7.2**. Values for ν_1 and ν_2 were determined via equations [7.2] and [7.3], and subsequently their nearest pitches, which are also included in the table.

CWT Row	Carrier Frequency (α)	Modulation Frequency (β)	Clarity	ν_1	p_1	ν_2	p_2
85	247.050	96.180	13%	295.140	D ₄	198.960	G ₃
115	342.381	99.819	18%	392.290	G ₄	292.472	D ₄
139	444.520	101.719	72%	495.379	B ₄	393.661	G ₄
157	540.664	88.960	55%	585.144	D ₅	496.184	B ₄
178	679.416	201.790	17%	780.311	G ₅	578.521	D ₅
202	882.097	173.268	36%	968.731	B ₅	795.463	G ₅
220	1072.883	191.227	51%	1168.496	D ₆	977.270	B ₅

Table 7.3 – Frequencies and nearest pitches corresponding to peaks in Figure 7.14

These results are quite accurate, despite the low power of some of the amplitude modulations resulting in poor clarity. The shaded row in the table is where there was the most error, but when rounding the frequencies to their nearest equal-tempered values, all of the difference tones were quantized to their correct pitches.

The best way to handle the extra data provided by the difference tone analysis is debatable and would require further experimentation with different post-processing methods. One simple idea, which has been implemented in *Wave Processor*, is to increase the clarity of pitches already found if their detection is supported by the presence of accurately measured difference tones. The algorithm is summarized by the following pseudo-code, which basically checks each detected frequency against each difference tone and boosts its clarity if the difference tone is caused by frequencies matching it. This allows for an error of one quarter tone.

```

FOREACH detected FREQUENCY with clarity CLAR {
  FOREACH WINDOW {
    FOREACH ROW {
      Get detected BEAT.FREQ for this WINDOW in this ROW
      Calculate CARRIER.FREQ for this ROW
      V1 = CARRIER.FREQ + BEAT.FREQ / 2
      V2 = CARRIER.FREQ - BEAT.FREQ / 2
      IF FREQUENCY within quarter tone of V1 or V2 {
        CLAR = CLAR ^ (1 - BEAT.CLAR)
      }
    }
  }
}

```

Figures 7.15 and **7.16** show the full transcription of the NSA quartet example before and after applying difference tone detection. The Morlet wavelet with a wavenumber of 15 was used for both transforms. Unfortunately, if the window size is too small, the method does not perform so well, since lower frequency beats cannot be measured accurately enough, as illustrated by **Figure 7.4**. Thus, the window size used in this example is 2048 samples. While not greatly different, **Figure 7.16** is a definite improvement nevertheless.

Note that this method still does not solve the problem of confusing harmonics with fundamental notes; if anything it compounds it. However this is a post-processing issue involving harmonic analysis and instrument timbre recognition, which is not within the scope of this study.

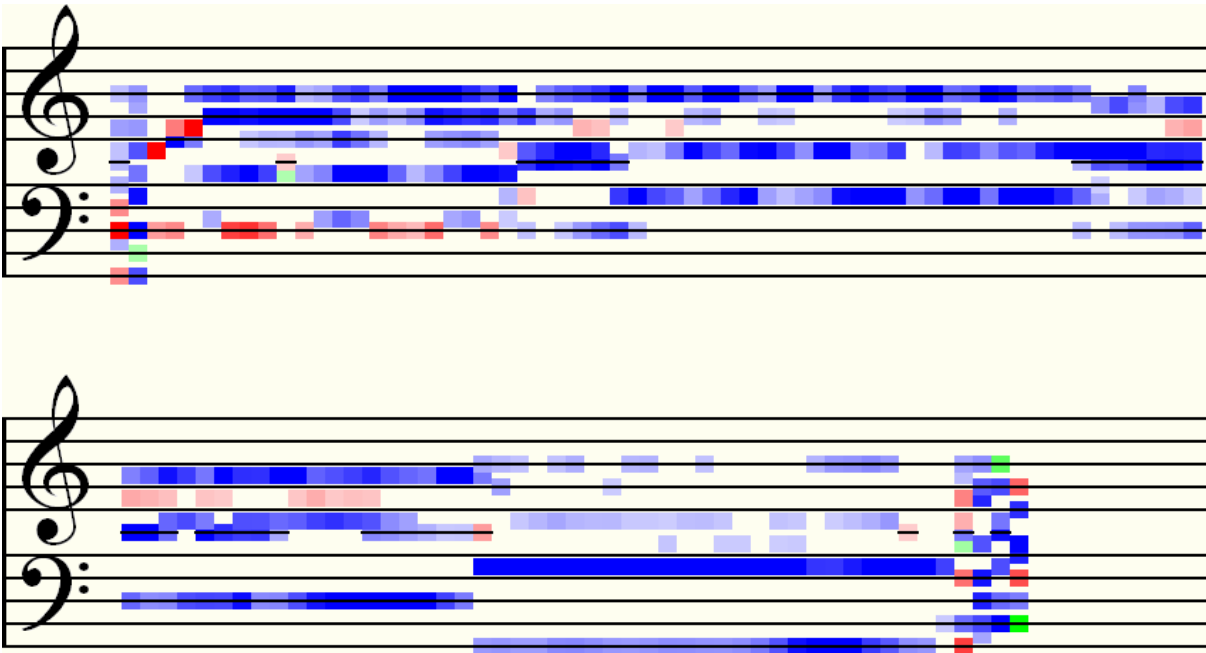


Figure 7.15 – Transcription of NSA quartet example without difference tone analysis

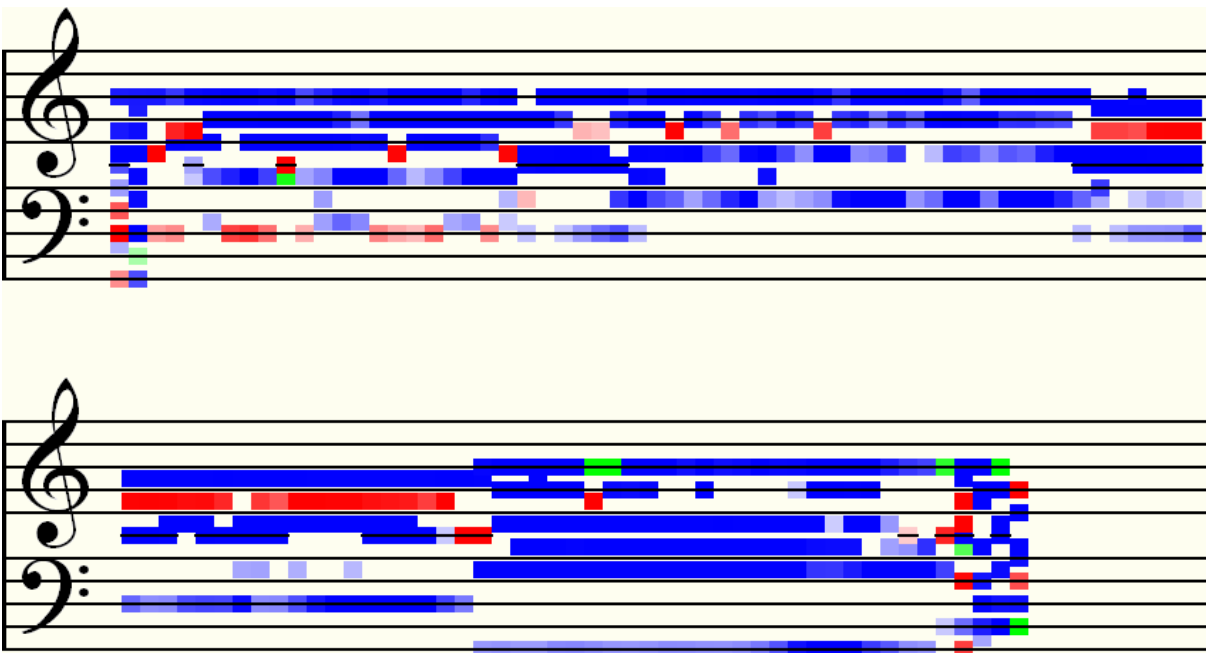


Figure 7.16 – Transcription of NSA quartet example with difference tone analysis

For another demonstration of this algorithm, the pitch extraction of the three-part *Nkosi Sikeleli Africa* excerpt from the previous chapter was repeated with the same settings (see **Figure 6.13**), except this time with difference tone analysis switched on. The improvement is most noticeable in the bass part, where most of the faintly detected notes have now been filled in. Scrutinizing the transcription, even counting the artefacts at the end, which are due to noise in the recording, the result shown in **Figure 7.17** is now 90% accurate. Percentage error was measured in each time frame by how far (in semitones) the transcription strays from the actual notes in each voice.

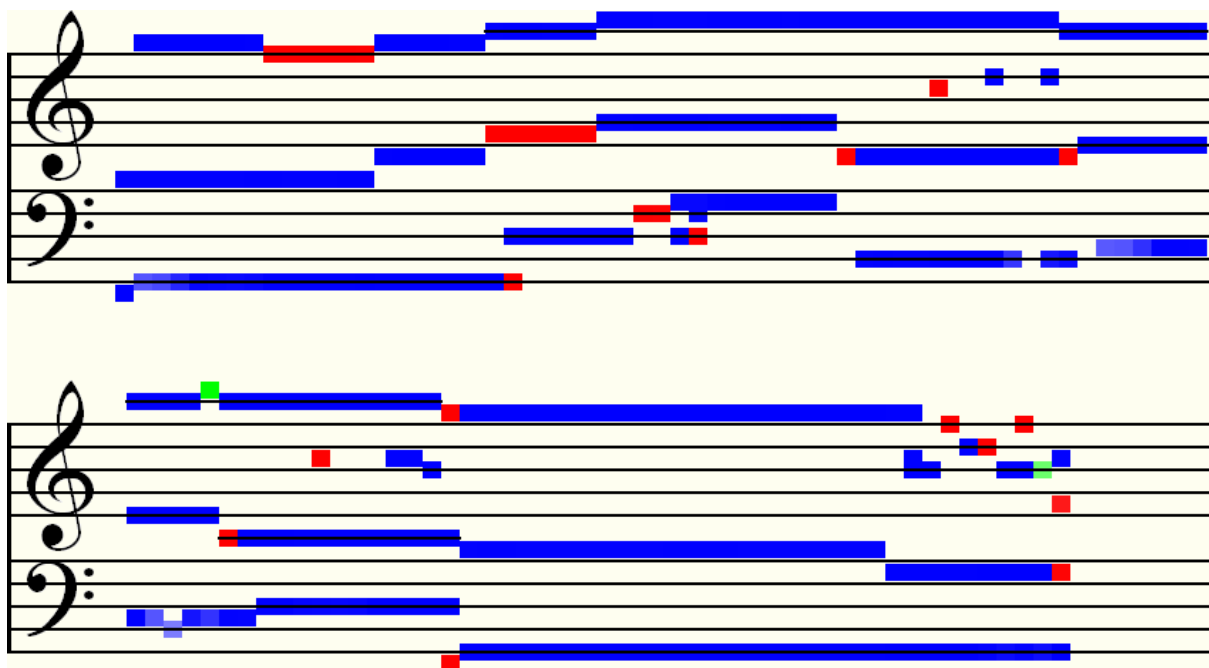


Figure 7.17 – Pitch Extraction of three-part NSA example using DoG wavelet transform

For a final example to end off this chapter, **Figure 7.18** shows an attempt at transcribing a five-part choral excerpt with *Wave Processor*. This is the opening five bars of Gregorio Allegri's *Miserere Mei, Deus*, which was mentioned in the introductory chapter*. The chosen frequency range for the CWT was 80Hz – 640Hz over 512 scales, and a Morlet wavelet with a wavenumber of 32 was used. While the difference tone analysis has been included, it had little effect on this particular output, since the pitch detection window width was very narrow (256 samples). The calculation utilized 272MB of memory and 3.75GB of hard disk swap space. A Pentium Core2 Duo Processor took 12 minutes and 13 seconds to complete the task.

Note the following:

- It was discovered that this particular recording is slightly flat, and so the base pitch, C_0 , had to be adjusted from 16.375Hz to 16Hz so that frequencies would convert to their correctly tuned pitches. It is intended that a fine tuning feature be added to *Wave Processor* so that the user may tweak the output in this way, if and when necessary.
- For this example, the program was manually altered to write D^\sharp as E^\flat , since this is the correct diatonic spelling of this pitch in the context of B^\flat Major – the key of the piece (see section 3.4 in Chapter 3 and section 4.4 in Chapter 4).
- Bar lines have also been added manually so that this result may be more easily compared with the actual musical score in **Figure 7.19**.
- There is much reverberation and acoustic delay in the recording, which is why many notes in each voice, especially in the top soprano line, appear to continue sounding after the next note has begun.
- Checking the score pitch for pitch confirms that *Wave Processor* has managed to detect *all* notes in all parts, though some of them less clearly than others. The least well-detected pitch is the bass B^\flat in the fourth beat of the third bar, but it does still appear nevertheless.

* Please find this excerpt at Sound\Allegri - Miserere - 5 bars.wav on the project CD.

- The vertical overlapping of the B \flat and the C in the two uppermost voices at the beginning of bar 4, creating the interval of a major forms a typical suspension (see Chapter 3, section 3.4.2). This has been accurately detected, though not drawn very well. In general, *Wave Processor* could benefit from some improvements to the graphical representation of pitches, especially in cases such as this. Compare this rendering of two adjacent notes on a staff with that of the score notation.
- Apart from the noise at the beginning, very few of the pitches detected in each time frame are harmonics or invalid fundamental notes, in comparison to the number which have been identified correctly.
- The note type in the penultimate bar of the score in **Figure 7.19**, $\text{||}\circ\text{||}$, is a *breve*. It lasts twice the length of a semibreve or whole note, \circ .

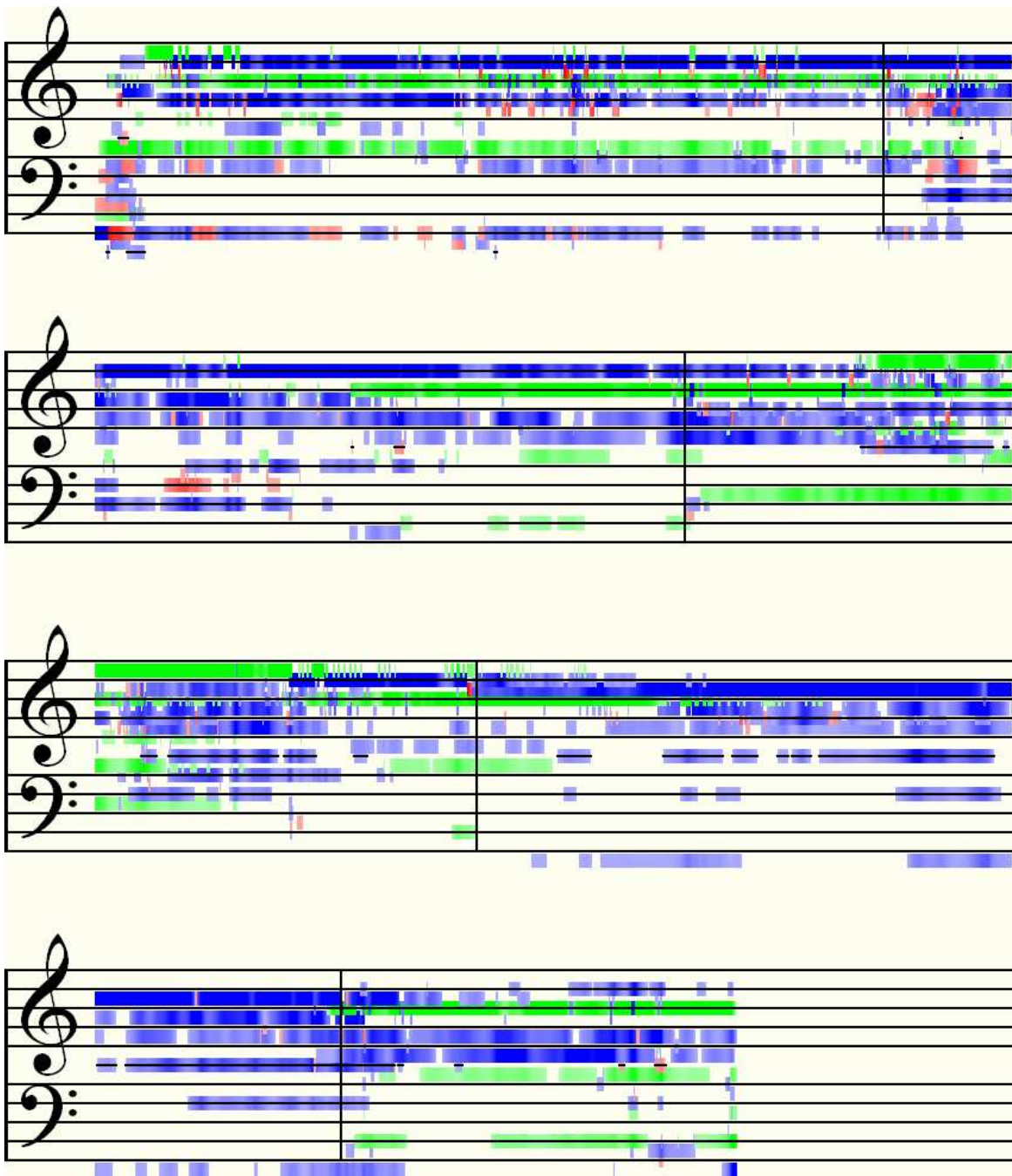


Figure 7.18 – Automatic transcription of first five bars of Allegri's *Miserere Mei, Deus*

Mi - se - re - re me - i,
 Mi - se - re - re me - i, De -
 Mi - se - re - re me - i, De -
 Mi - se - re - re me - i,
 De - - - - - us.
 - - - - - us.
 - - - - - us.
 De - - - - - us.

Figure 7.19 – Opening five bars of *Miserere* (manual transcription from actual score)

8 Experimental Pitch Detection on Live Audio Recordings

This chapter is mainly a record of experiments carried out on the pitch detection theories discussed in this thesis. As mentioned in Chapter 1, it is important that an automatic transcription system be tested on live audio recordings and not just synthetically produced sounds, and so all experimental data is from live recordings. The intention here is to demonstrate the performance of the algorithms implemented in *Wave Processor* and also to examine the effects on the output of varying certain transform parameters. In this way, a better understanding may be gained of how best to configure settings, given different musical situations or methods of recording.

As a space-saver, only spectrograms and pitch graphs which show the most crucial results have been included. There are, however, discussions about all settings attempted, and for each set of sounds, one has been chosen to show details of the experimentation, with respect to adjusting transform parameters.

8.1 Preparation of Audio Data

A number of studio recordings were made of some real instruments playing various simple harmonic constructs. The musical scores for these have all been included in **Appendix D** and references to these are given. It was considered important that these sound samples should be *acoustically* mixed, i.e. the instruments were recorded playing simultaneously and not just on separate tracks to be mixed later on [Cont07L].

Listed below are brief descriptions of the various sets of recordings made. String instruments were chosen specifically because they have a very rich timbre, since it is vital the detection algorithm be configurable so as not to be too sensitive that it always detects harmonics as fundamental notes. Piano recordings of the same scores were also made for the same reason, and they provide some comparison for analysis. For control sound samples, simple sinusoidal signals were created synthetically and mixed, following the same pitch sequences as in the recordings. These, of course, have no timbre. The software used to create these clips was a command line program* which reads and parses an input text file containing pitch and time information, and outputs a PCM wave file.

Set 1 – Two Similar Instruments (see **Appendix D.1**)

- i) Middle C pedal (held) note and C Major scale above, one octave, ascending
- ii) C₅ pedal note and C Major scale below, one octave, ascending
- iii) G₄ pedal note and G Minor scale above, one octave, ascending
- iv) A₃ pedal note and chromatic (semitones) scale above, 1 octave, ascending

Set 2 – Three Similar Instruments (see **Appendix D.2**)

- i) Close interval diatonic triads built on each note of a scale (root position only)
Keys: F Major and D Minor
- ii) Wide interval diatonic triads on notes of a scale.
Keys: C Major (root positions) and A Minor (2nd inversions)

* Please find `synth.exe` in `Software\Synthesizer` on the project CD

Set 3 – Four Similar Instruments (see **Appendix D.3**)

- i) Perfect cadence in C Major
- ii) Two chord progressions in G Major and A Minor

The studio recordings were produced as follows: Four different microphones were set up, namely:

- RØDE – NT1-A condenser microphone [RØDE08W]
- RØDE – NT3 condenser microphone [RØDE09W]
- SHURE – Beta 57A instrument microphone [Shure09W]
- AKG – D 112 large-diaphragm microphone [AKG09W]

For more information about these microphones and their specifications, including envelopes and frequency responses, please visit the web sites given in each reference. Note that the last of these, the D 112, is designed specifically for use with bass instruments and its most common use is for kick drums.

These four microphones were then used to record four separate tracks at the same time, of exactly the same music. It was supposed that microphones with different frequency responses would yield slightly different signals and thus different transformed signals, which in turn could affect pitch detection. The extent of the differences and their effects may be measured by comparing spectrograms of four samples of the same piece of recorded sound. Purely for interest's sake, the output of the four different microphones was also mixed together (post recording) to see what the effect would be, as an additional experiment.

The tracks were all recorded digitally and saved as continuous mono 16 bit PCM WAV files, at a sample rate of 44,100kb/s (CD quality.) They were each then cut into smaller files – one for each of the various melodic or harmonic constructs – using *Cool Edit Pro 2.0* [Syntrillium02S]. Since all five master tracks were recordings of exactly the same thing, start and end times were noted for each cut made in one of the tracks, and then automatically applied in the same places to the other four tracks, resulting in identically timed clips.

No further processing was done or effects applied to the sound bites, other than to normalize them, in order to make them all the same volume. A normalizing threshold of 90% was used in order to avoid clipping or distortion of the waveforms.

The synthesized control clips (employing sine waves) were all created from short programs written in C++, three examples of which have been listed in **Appendix B**. Again, the clips were normalized to 90%, so as to conform with the other experimental clips, although this step was incorporated into the program code, instead of using *Cool Edit Pro 2.0*. Note: to save space on the project CD, the raw recorded tracks have not been provided, but the cut tracks used in the experiments are available.

8.2 Methods and Results

In these experiments, the terms *sensitivity* or *sensitive transform*, and the initialism *DTA* have been utilized. The former refers to a transform for which values have been calculated by taking the square root of the magnitude. This yields a more “sensitive” result because the difference between maxima and minima is reduced and the curve of values between them becomes flatter. *DTA* stands for Difference Tone Analysis, which may be turned on or off.

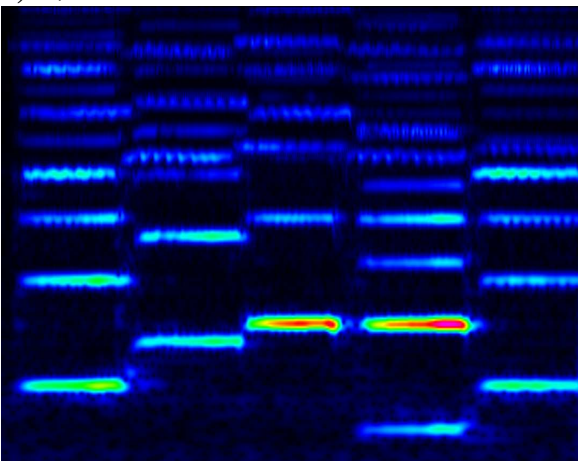
8.2.1 Microphone Testing

The first experiment was to see if using different microphones in the recording had a definite effect on each algorithm's ability to detect frequencies. This experiment made some necessary assumptions about how best to set parameters in the CWT (the subject of the second set of experiments) but since these parameters were fixed and not variable for each microphone test, the results were independent of them. The parameters chosen for the transform were:

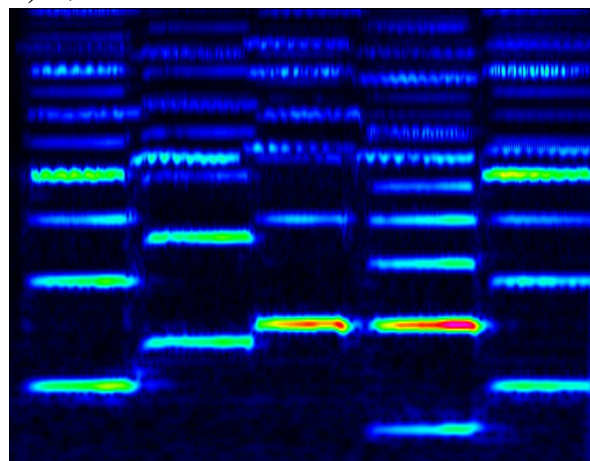
- Wavelet: Morlet
- Wavenumber: 40
- Upper frequency bound: 1320Hz (E_6)
- Lower frequency bound: 65Hz (C_2)
- Number of scales: 256
- Sensitivity: On

Since frequency response was the main issue, it was also not necessary to test more than one of the tracks, as long as its frequency range was suitably large. The chosen clip was taken from Set 3 ii) – the four-part chord progressions. The A Minor progression had the widest range and so was chosen for this reason. The resulting spectrograms are shown below:

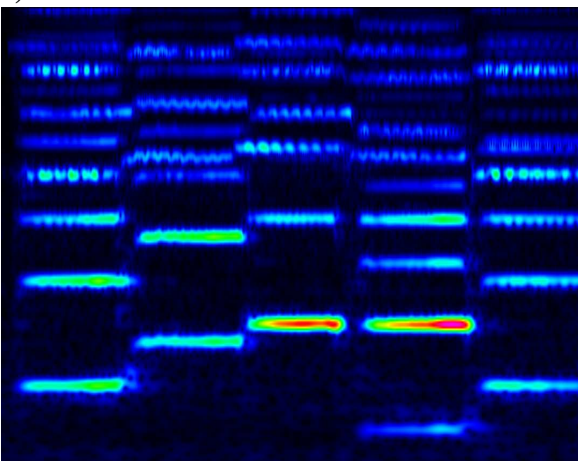
a) RØDE NT1-A



b) RØDE NT3



c) SHURE Beta 57A



d) AKG D 112

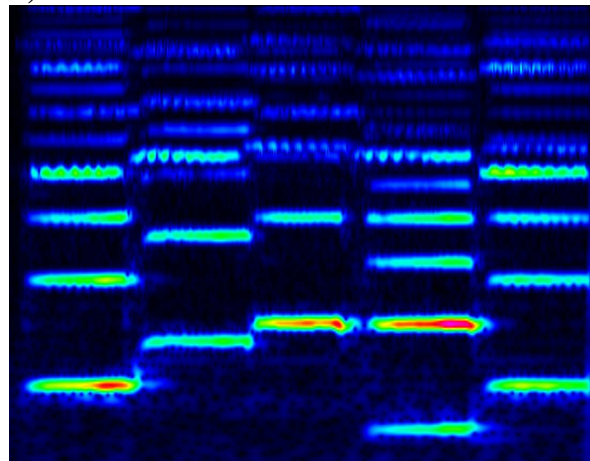


Figure 8.1 – Spectrograms showing different frequency responses of four microphones

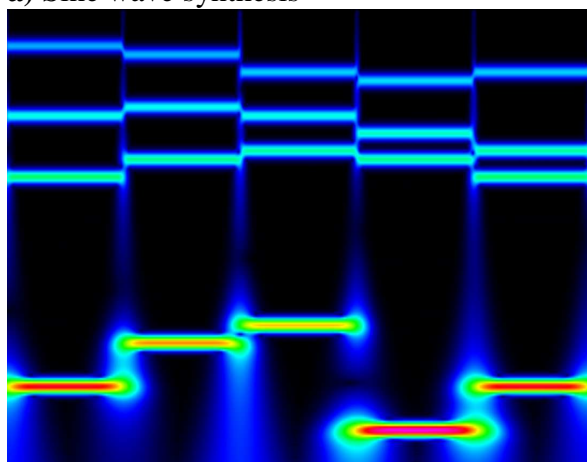
It can be seen that while there are differences in each image, they are not very profound. Looking at the first of the bass notes in each case, the AKG D 112 microphone performed the best in this range, picking this frequency up strongly. This is as one would expect, since it is designed as a high-performance bass microphone. In general, the AKG D 112 performed better than the other microphones, possibly with the exception of the RØDE NT3, which also produced good all-round results. The Shure microphone seems to be slightly better at picking up higher frequencies than the others, but the difference in performance is not large. In corroboration to the claim on the RØDE web site that the NT1-A is the “world’s quietest studio condenser microphone”, it can be seen that the spectrogram is indeed somewhat more clear of noise in comparison to the others. However, the detection of higher frequencies has suffered slightly as a result. The AKG D 112 in particular has picked up much noise especially in the lower frequency range.

Based on these results, it was decided that the microphone of choice for the rest of the experiments would be the AKG D 112, since it was the most sensitive microphone and so yielded the most frequency information in its output.

Only the spectrograms were examined for differences because the experiment concerns effects at the pre-processing stage. The pitch extraction from the frequency information (where difference tone analysis is done, for example) is a post-processing step.

To give an idea of harmonics vs. fundamental tones picked up by each microphone, **Figure 8.2** shows the same transform on the synthesized wave. Also in the same figure, the result from mixing all four microphones is presented.

a) Sine wave synthesis



b) All microphones

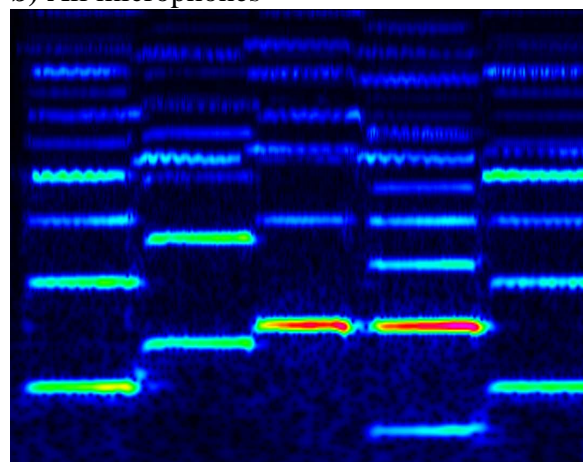


Figure 8.2 – Spectrograms showing a) location of fundamental notes in chord progression and b) mixed frequency response of all four tested microphones

Spectrogram **b)** in the above figure shows a large amount of noise, which is the combined product of all the microphones. Amongst other things, it demonstrates the loss of quality as a result of using multiple close-up microphones in a studio. Microphone placing and angle is an important consideration for a studio sound engineer, but instead of going into the specifics of this subject here, this instruction booklet from Shure [Shure09L] is instead recommended as an excellent guide.

8.2.1 Set 1 – Two Similar Instruments

For these experiments, the lower and upper bounds of the frequency scale were set from 128Hz to 1024Hz, which covers the range of all scales. Also, for these tests and all those following, the number of scales in each CWT remained fixed at 128. This provides ample frequency resolution for these particular audio samples. Whereas in the previous experiments spectrograms were examined, here the full pitch extraction step was added, and pitch graphs were compared with the known scores for similarity. The settings for which the output was deemed most similar to the original, in terms of pitch and time location, were recorded.

8.2.1.1 Best Parameter Settings

It was found that for many of the parameter configurations for wavelet transforms and also the Phase Vocoder, the first harmonic of the pedal note, in both the strings and in the piano recordings, was detected instead of its fundamental. The tables below show the best parameter settings found by varying each in turn while keeping the others fixed. The first table is for the recordings of string instruments and the second contains results for the piano samples. For the control set in the third table, it was possible to get the transcriptions near to perfect and in a couple of cases 100% accurate with the CWT Pitch Method (CPM for short).

Strings		
Scale	Configuration yielding best pitch detection	
	Method	Parameters
i)	CPM	Morlet wavelet, wavenumber: 40, sensitive, DTA disabled
ii)	CPM	Morlet wavelet, wavenumber: 40, sensitive, DTA disabled
iii)	CPM	Morlet wavelet, wavenumber: 40, sensitive, DTA disabled
iv)	CPM	Morlet wavelet, wavenumber: 40, sensitive, DTA disabled

Table 8.1 – Results of automatic transcriptions for two-part string recordings

Piano		
Scale	Configuration yielding best pitch detection	
	Method	Parameters
iii)	CPM	Morlet wavelet, wavenumber: 40, sensitive, DTA disabled
i, ii, iv)	Could not be tested due to lack of similar quality data and content	

Table 8.2 – Results of automatic transcriptions for two-part piano recording

Synthetic		
Scale	Configuration yielding near to perfect pitch detection	
	Method	Parameters
i)	CPM	Morlet wavelet, wavenumber: 16, not sensitive, DTA enabled
ii)	CPM	Morlet wavelet, wavenumber: 40, sensitive, DTA enabled or disabled
iii)	CPM	Morlet wavelet, wavenumber: 40, sensitive, DTA enabled or disabled
iv)	CPM	Morlet wavelet, wavenumber: 40, sensitive, DTA enabled or disabled

Table 8.3 – Results of automatic transcriptions for two-part synthesized waves

8.2.1.2 Detailed Analysis

The next set of tables shows the experimentation process, with observations made on results for a chosen sample from this set of sounds. In this case, the piano scale in G Minor with the lower held note (scale **iii**) is presented. Although CPM with the Morlet transform was observed to work a little better than other techniques, the DWT / MPM result should also get a mention, since it eliminated more unwanted harmonics than the others. However, it did not detect the pedal note very well either. Note that the best setting of the parameters was the same as for each of the strings scales.

Redundant Haar / MPM	
Window Width	Comments
512	Fundamentals in scale generally correct, but no pedal note was detected. Some first harmonics detected and other minor artefacts.
2048	Clearer detection and fewer artefacts. Harmonics also reduced.

Table 8.4 – Results for DWT / MPM

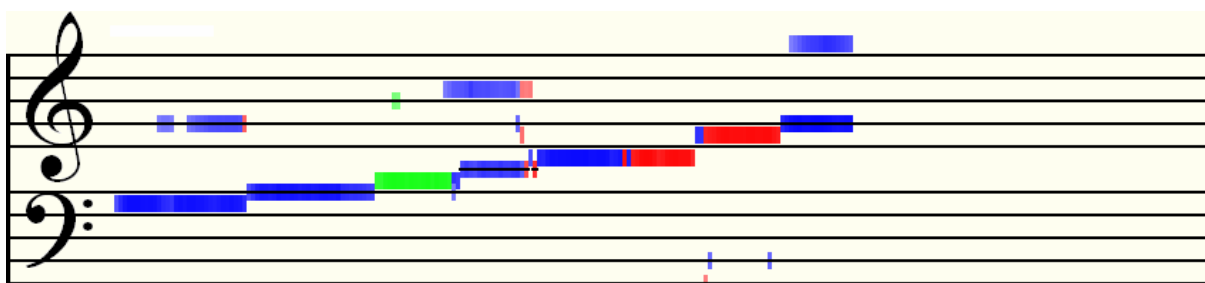


Figure 8.3 – DWT / MPM transcription (window width = 2048)

Phase Vocoder	
Window Function	Comments
Gaussian	Fundamentals and first harmonics of scale detected equally strongly. Pedal note detected as being held over nearly four and a half notes, but at the wrong octave. Several artefacts.
Blackman	Worst result: too many harmonics detected, confusing with correctly detected pitches.
None	Average result, though with several artefacts eliminated when using the Gaussian window.

Table 8.5 – Results for Phase vocoder

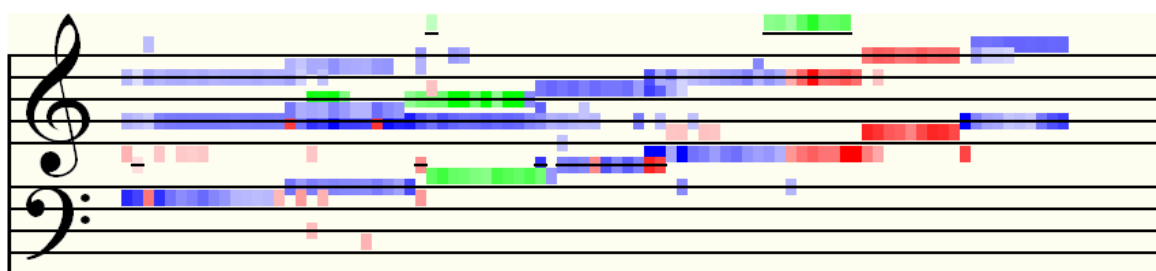


Figure 8.4 – Phase vocoder (with Gaussian window) transcription

Morlet Wavelet		
Wavenumber: 5	Comments	
Ordinary transform	Correct fundamentals roughly detected. Strong first harmonics detected	
With sensitivity	Poorer detection of fundamentals. First harmonics detected slightly more strongly instead.	(Both) Better clarity of both fundamentals and extra harmonics detected with sensitivity.
With DTA	Better clarity for both fundamentals and harmonics. Some more harmonics detected.	
Wavenumber: 20	Comments	
Ordinary transform	More correct fundamentals, but harmonics also detected clearly Low note was detected an octave high, i.e. at the first harmonic	
With sensitivity	Better detection of both fundamentals and harmonics. Lower note is clearer but still at the wrong octave.	(Both) Many more harmonics detected.
With DTA	More harmonics and upper fundamentals detected.	
Wavenumber: 40	Comments	
Ordinary transform	BEST RESULT – fewer harmonics accepted, but first harmonic of lower note was still detected as being stronger than the actual fundamental.	(Both) No difference to ordinary detection with sensitivity and without DTA.
With sensitivity	Better detection of both fundamentals and harmonics. Lower note is clearer but still at the wrong octave.	
With DTA	No difference.	

Table 8.6 – Results for CPM with Morlet transform

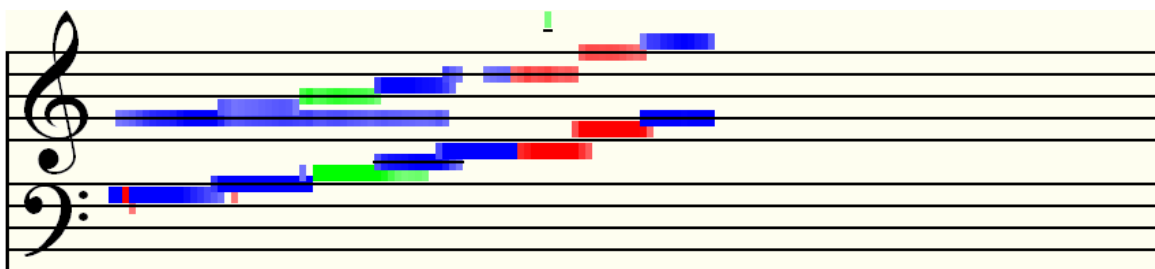


Figure 8.5 – CPM (Morlet 40) transcription

Paul Wavelet		
Order: 4		
Ordinary transform	Very poor result. Hardly any fundamentals detected, mainly a few harmonics.	
With sensitivity	Slightly better detection, but still largely inaccurate.	(Both) No difference to ordinary detection with sensitivity and without DTA.
With DTA	No difference.	
Order: 10		
Ordinary transform	Much improved detection of fundamentals, but first harmonics are also stronger.	
With sensitivity	Greater clarity for both fundamentals and harmonics.	(Both) Even better clarity for both fundamentals and harmonics.
With DTA	Greater clarity for both fundamentals and harmonics.	

Table 8.7 – Results for CPM with Paul transform

Derivative of Gaussian Wavelet		
Order: 2 (Mexican Hat)		
Ordinary transform	Poor result. Hardly any fundamentals detected, mainly a few harmonics. However, lower note G is present as a first harmonic.	
With sensitivity	Slightly different result, but quality is still as poor as without this option.	(Both) No difference to ordinary detection with sensitivity and without DTA.
With DTA	No difference.	
Order: 16		
Ordinary transform	Much improved detection of fundamentals, but first harmonics are also stronger.	
With sensitivity	Better detection of both fundamentals and harmonics. More harmonics detected. Lower note is much clearer but still at the wrong octave.	(Both) Even better clarity for both fundamentals and harmonics, but extra harmonics also begin to appear.
With DTA	Better detection of both fundamentals and harmonics. Not as many harmonics detected as with sensitivity.	

Table 8.8 – Results for CPM with DoG transform

8.2.2 Set 2 – Three Similar Instruments

The next set of sounds were the triads in different keys. As with the previous example, it should be remembered that the software is not yet capable of key detection, and so, although the notes may not be diatonically correct to the scores, notes detected as D#, for example, are still equivalent to E \flat if the key is say G Minor. It is already becoming clear in these examples that the inclusion of DTA, while useful, is not always a good thing, since, more often than not, it has the effect of clarifying weaker harmonics which should be filtered out of the transcription.

8.2.2.1 Best Parameter Settings

Strings		
Triads	Configuration yielding best pitch detection	
	Method	Parameters
i) F Major	CPM	Morlet wavelet, wavenumber: 40, sensitive, DTA disabled
i) D Minor	CPM	Morlet wavelet, wavenumber: 32, sensitive, DTA disabled
ii) C Major	CPM	DoG wavelet, order: 16, sensitive, DTA enabled
ii) A Minor	CPM	DoG wavelet, order: 16, sensitive, DTA enabled

Table 8.9 – Results of automatic transcriptions for three-part string recordings

Piano		
Triads	Configuration yielding best pitch detection	
	Method	Parameters
i) F Major	CPM	Morlet wavelet, wavenumber: 40, not sensitive, DTA disabled
i) D Minor	CPM	Morlet wavelet, wavenumber: 36, sensitive, DTA disabled
ii) C Major	CPM	Morlet wavelet, wavenumber: 40, sensitive, DTA disabled
ii) A Minor	CPM	Morlet wavelet, wavenumber: 40, sensitive, DTA disabled

Table 8.10 – Results of automatic transcriptions for three-part piano recordings

Synthetic		
Triads	Configuration yielding near to perfect pitch detection	
	Method	Parameters
i) F Major	CPM	Morlet wavelet, wavenumber: 34, sensitive, DTA enabled
i) D Minor	CPM	Morlet wavelet, wavenumber: 33, sensitive, DTA enabled
ii) C Major	CPM	Morlet wavelet, wavenumber: 37, DTA enabled or disabled
ii) A Minor	CPM	Morlet wavelet, wavenumber: 32, DTA enabled

Table 8.11 – Results of automatic transcriptions for three-part synthesized waves

8.2.2.2 Detailed Analysis

The chosen clip in this case was the set of wide interval triads in C Major, played by the string instruments. Note that although CPM with the Morlet wavelet performed best, no result was all that satisfactory in this case.

Phase Vocoder	
Window Function	Comments
Gaussian	Many high frequency harmonics detected. Only the final note of the lowest voice appeared in the output.
Blackman	Best out of all windows for detection of fundamentals, but more harmonics and other artefacts appeared.
None	Slightly clearer detection than with Gaussian window, but still poor. Some harmonics filtered out.

Table 8.12 – Results for Phase Vocoder

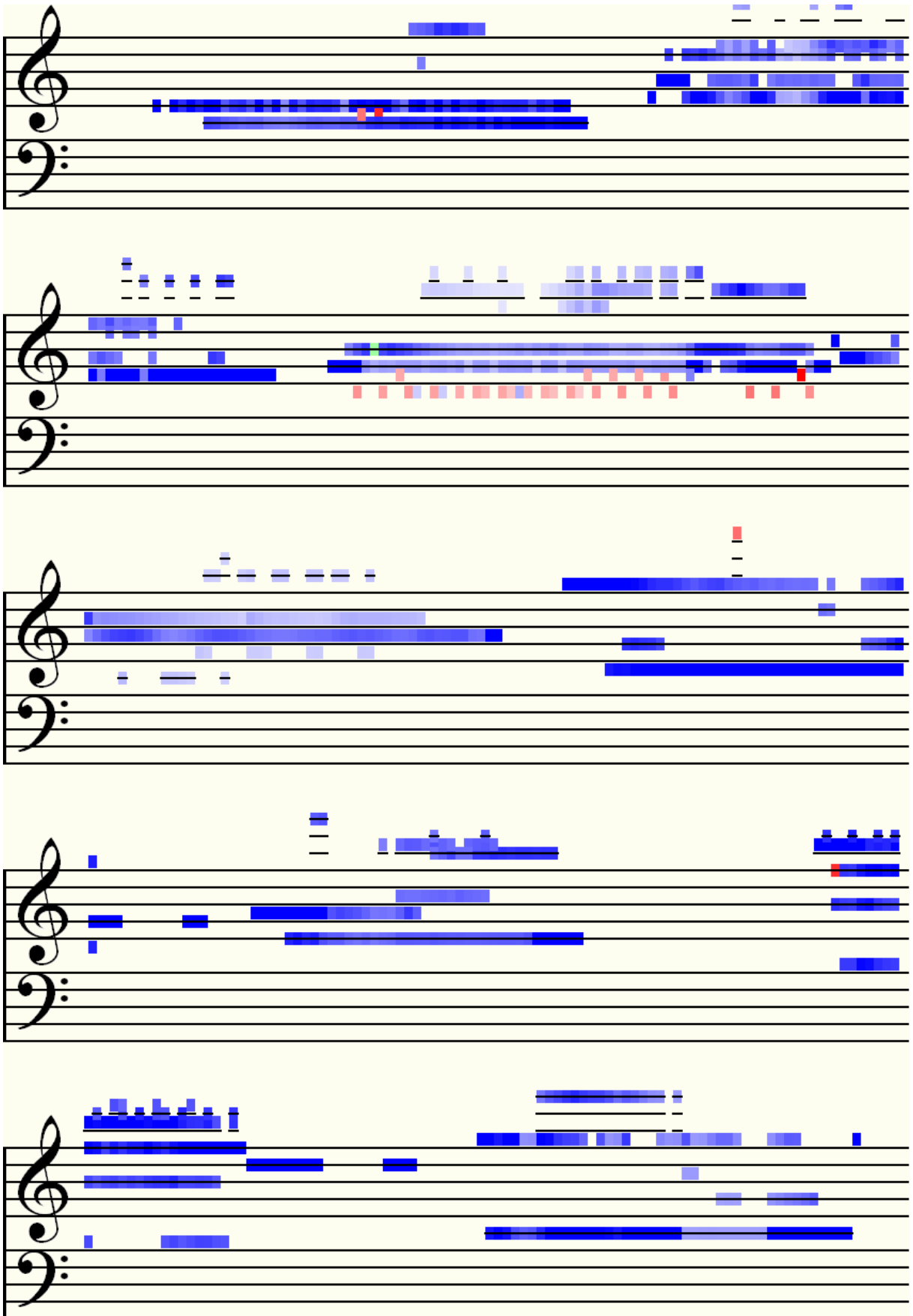


Figure 8.6 – Phase Vocoder (with Blackman window) transcription

Redundant Haar / MPM	
Window Width	Comments
512	Many harmonics chosen over fundamentals.
2048	Same harmonics chosen over fundamentals, but clearer. Screenshots were not taken for this result, which was even less satisfactory than the Phase Vocoder.

Table 8.13 – Results for DWT / MPM

Morlet Wavelet		
Wavenumber: 5	Comments	
Ordinary transform	Generally poor, with many artefacts and harmonics detected. Most fundamentals were incorrectly identified.	
With sensitivity	More notes detected, but tuning was calculated to be slightly sharper than it should be.	Greater clarity of both fundamentals and harmonics, but tuning is still wrong.
With DTA	More harmonics detected and general clarification of all pitches.	
Wavenumber: 20	Comments	
Ordinary transform	Result is an improvement from the previous, and notes are correctly tuned, however many fundamentals are still missing, especially in the lowest line. Generally hardly anything is detected below Middle C (C ₄).	
With sensitivity	Detection of lower frequencies is better, but more harmonics also appear.	Harmonics detected with sensitivity enabled now very strongly detected. Last four notes of viola part appear, and middle voice is also slightly stronger.
With DTA	Many more notes detected, mostly harmonics. Viola remains very weak and only appears with the last two chords.	
Wavenumber: 40	Comments	
Ordinary transform	Last two notes of viola part now strongly detected, but the rest are not present.	
With sensitivity	Further improvement in detection of lower notes. All but the first two viola notes are now present, albeit weakly. Result is not very different from when wavenumber is set at 20.	Slight improvement from detection with sensitivity turned on only.
With DTA	Little difference. One or two additional harmonics are detected.	

Table 8.14 – Results for CPM with Morlet transform

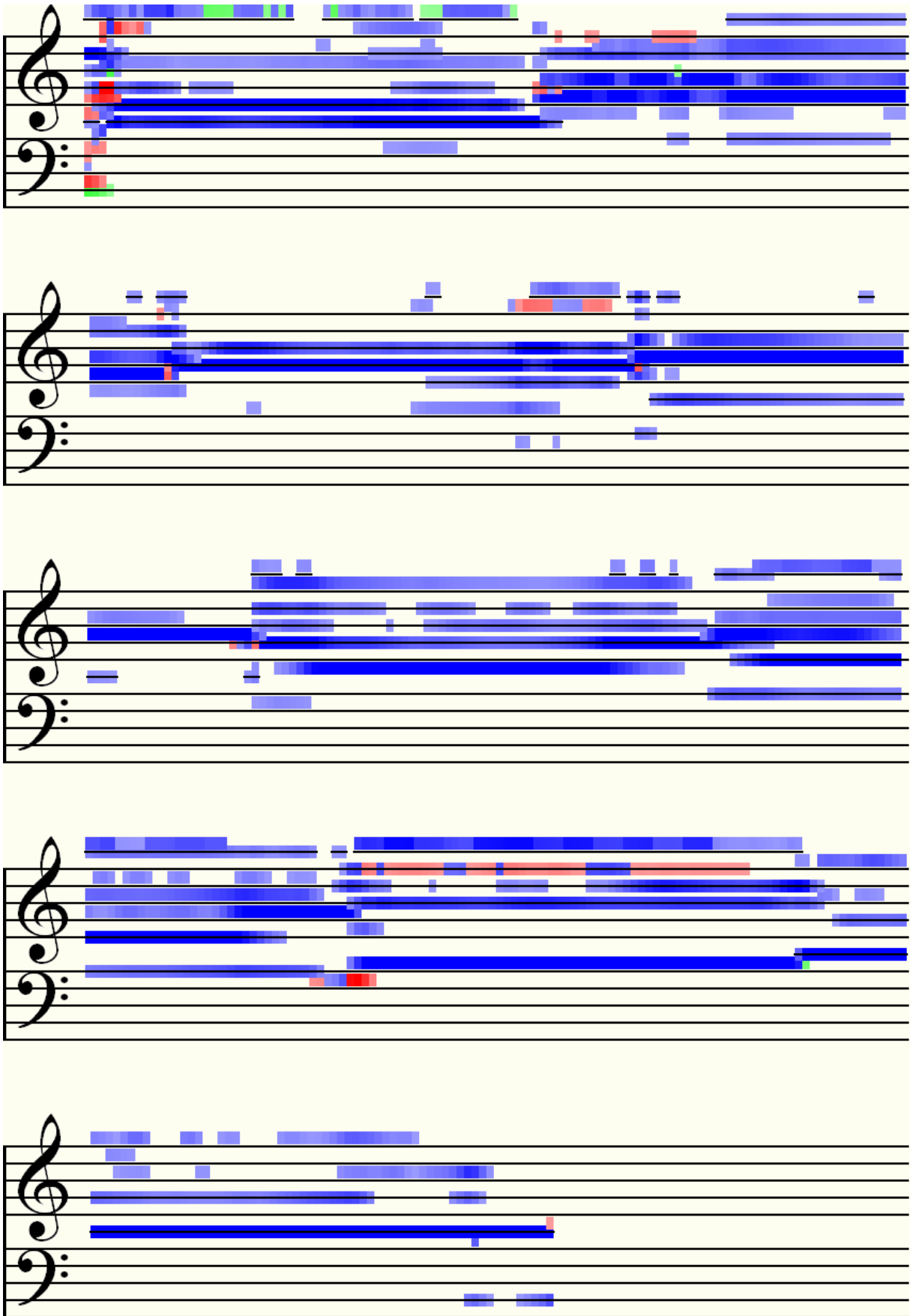


Figure 8.7 – CPM (Morlet 40) transcription

Paul Wavelet		
Order: 4		
Ordinary transform	Result was generally very poor. Only one or two fundamentals correctly identified. Many artefacts.	
With sensitivity	Hardly any improvement than without.	No difference to ordinary detection with sensitivity and without DTA.
With DTA	No difference to ordinary detection.	
Order: 10		
Ordinary transform	Result is still very poor despite increasing the order of the wavelet.	
With sensitivity	Hardly any improvement than without.	Slight increase in clarity of all detected components.
With DTA	No difference to ordinary detection.	

Table 8.15 – Results for CPM with Paul transform

Derivative of Gaussian Wavelet		
Order: 2 (Mexican Hat)		
Ordinary transform	Result was generally very poor. Only one or two fundamentals correctly identified. Many artefacts.	
With sensitivity	Hardly any improvement than without.	No difference to ordinary detection with sensitivity and without DTA.
With DTA	No difference to ordinary detection.	
Order: 16		
Ordinary transform	Fewer harmonics than with Morlet 20, but also fewer fundamentals detected. Also none detected below Middle C.	
With sensitivity	More harmonics detected. Last three notes of viola part are now present.	BEST RESULT – While only two chords, the third and fourth in the progression, are now quite clear, the result is only marginally better than the Morlet 40 detection. Both screenshots have been included for further comparison.
With DTA	More harmonics are clarified.	

Table 8.16 – Results for CPM with DoG transform

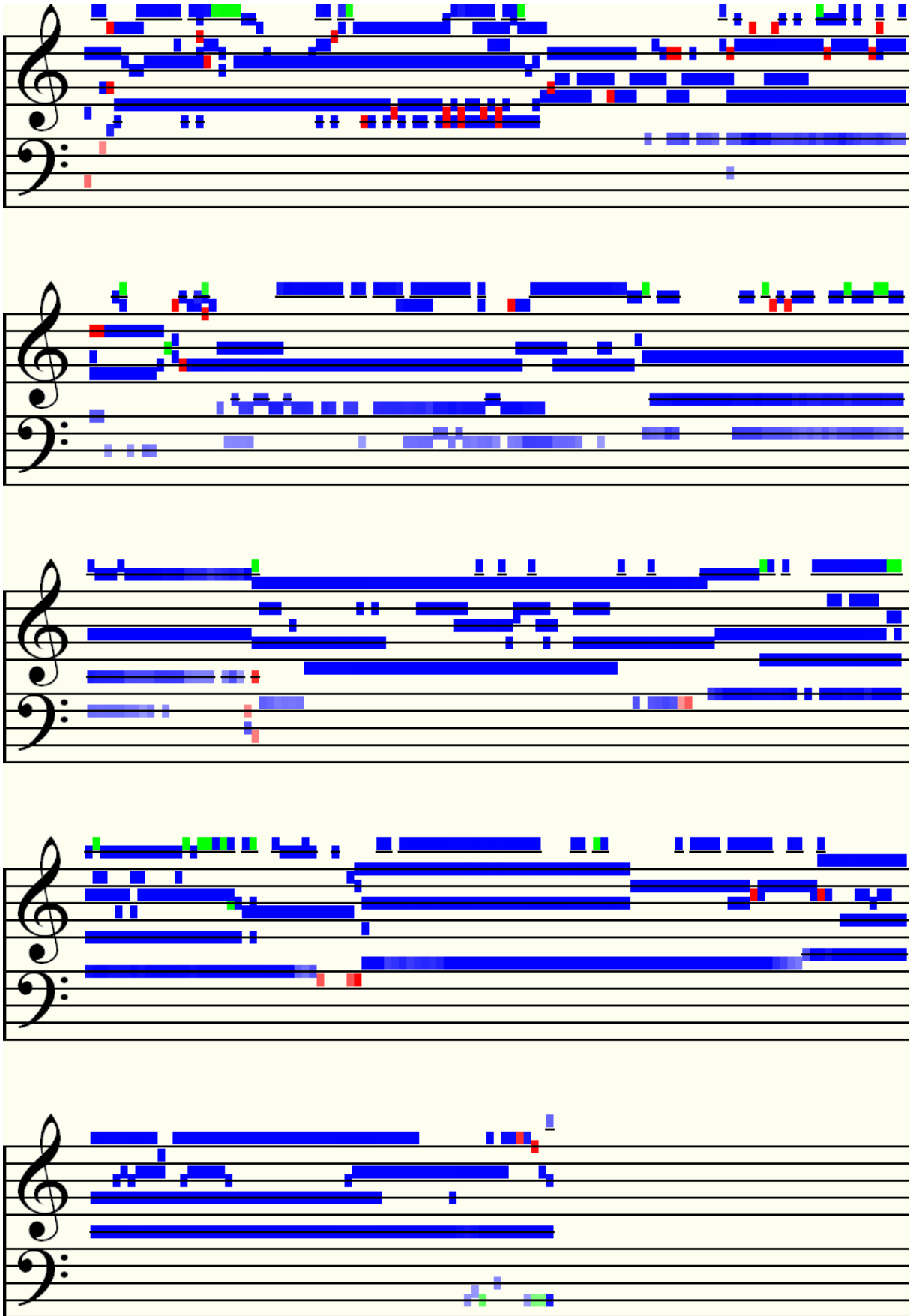


Figure 8.8 – CPM (DoG 16) transcription

8.2.3 Set 3 – Four Similar Instruments

The final set of experiments were those done on the four-part harmonies and the clip chosen for the more detailed analysis was the progression in A Minor, as played on the strings.

8.2.3.1 Best Parameter Settings

Strings		
Quartads	Configuration yielding best pitch detection	
	Method	Parameters
i) Cadence	CPM	Morlet wavelet, wavenumber: 40, sensitive, no DTA
ii) G Major	CPM	Morlet wavelet, wavenumber: 40, sensitive, no DTA
ii) A Minor	Phase Vocoder	Gaussian window, window width: 1024

Table 8.17 – Results of automatic transcriptions for four-part string recordings

Piano		
Quartads	Configuration yielding best pitch detection	
	Method	Parameters
i) Cadence	CPM	Morlet wavelet, wavenumber: 40, sensitive, DTA disabled
ii) G Major	CPM	Morlet wavelet, wavenumber: 40, sensitive, DTA disabled
ii) A Minor	CPM	Morlet wavelet, wavenumber: 40, sensitive, DTA disabled

Table 8.18 – Results of automatic transcriptions for four-part piano recordings

Synthetic		
Quartads	Configuration yielding near to perfect pitch detection	
	Method	Parameters
i) Cadence	CPM	Morlet wavelet, wavenumber: 16, not sensitive, DTA enabled
ii) GMajor	CPM	Morlet wavelet, wavenumber: 12, not sensitive, DTA enabled
ii) A Minor	CPM	Morlet wavelet, wavenumber: 10, not sensitive, DTA enabled

Table 8.19 – Results of automatic transcriptions for four-part synthesized waves

8.2.3.2 Detailed Analysis

Here the Phase Vocoder only marginally outperformed CPM. Both screenshots are included.

Phase Vocoder	
Window Function	Comments
Gaussian	BEST RESULT – Most of the fundamental pitches were detected with some harmonics. Some base notes were faint but were nevertheless present.
Blackman	Not a good result – out-of-tune detection of some fundamentals
None	A clearer result than with the Gaussian function, but more harmonics and other artefacts appeared.

Table 8.20 – Results for Phase vocoder

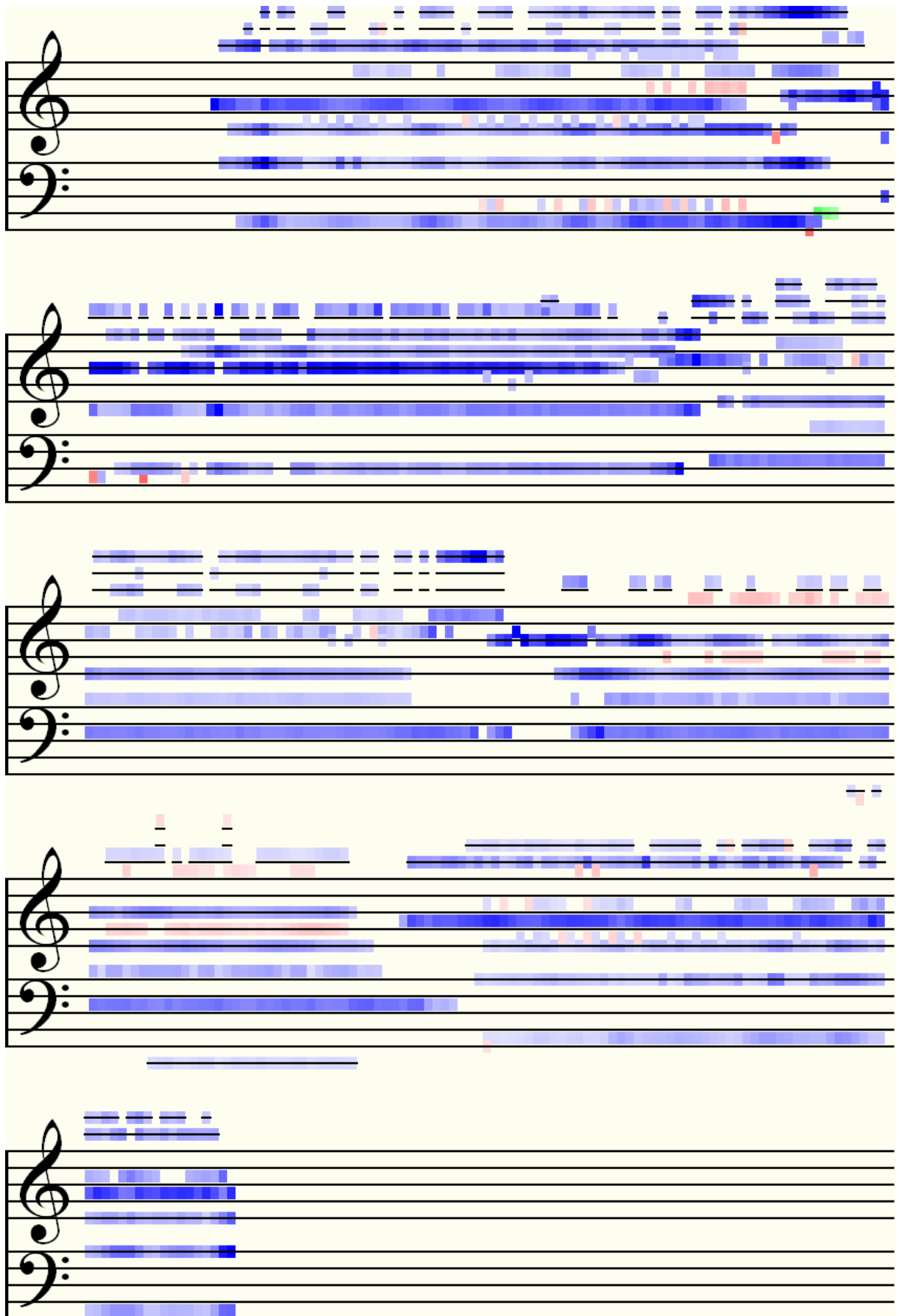


Figure 8.9 – Phase vocoder (with Gaussian window) transcription

Morlet Wavelet		
Wavenumber: 5		
Comments		
Ordinary transform	A sparse result, but not bad in that it did not contain too many artefacts. One or two bass notes detected as their first harmonics.	
With sensitivity	More notes detected, but tuning was calculated to be slightly sharper than it should be.	All notes, including out-of-tune pitches were clearer still.
With DTA	Clarity is much improved, but some notes are detected out of tune.	
Wavenumber: 20		
Comments		
Ordinary transform	An improved detection in terms of tuning, but there are still a number of fundamental notes missing.	
With sensitivity	Better result, but the penultimate bass note is still mostly being detected as its first harmonic.	Generally a good result, but more harmonics were detected which made the pitch graph look rather cluttered. However, with this setting, the lower bass notes are clear.
With DTA	Result is improved, with more of the notes in the higher frequency range now appearing.	
Wavenumber: 40		
Comments		
Ordinary transform	Only marginally better result than with wavenumber set to 20.	
With sensitivity	Slightly better result with more of the notes in the higher frequency range appearing.	Only a few more harmonics detected.
With DTA	Only a few more harmonics detected, otherwise hardly any difference.	

Table 8.21 – Results for CPM with Morlet transform

Paul Wavelet		
Order: 4		
Ordinary transform	Poor result with only a few notes correctly identified and many artefacts.	
With sensitivity	An improvement – one or two of the bass notes are clear, but much detail is still missing.	A slight improvement in clarity.
With DTA	A few more artefacts and harmonics appear.	
Order: 10		
Ordinary transform	A somewhat improved result with more notes correctly identified.	
With sensitivity	Further improvement, especially in the upper frequency range.	Greater clarity achieved, yielding a fair result, although many notes appear broken up and there are still artefacts.
With DTA	The result is improved and the bass line is now almost fully rendered.	

Table 8.22 – Results for CPM with Paul transform

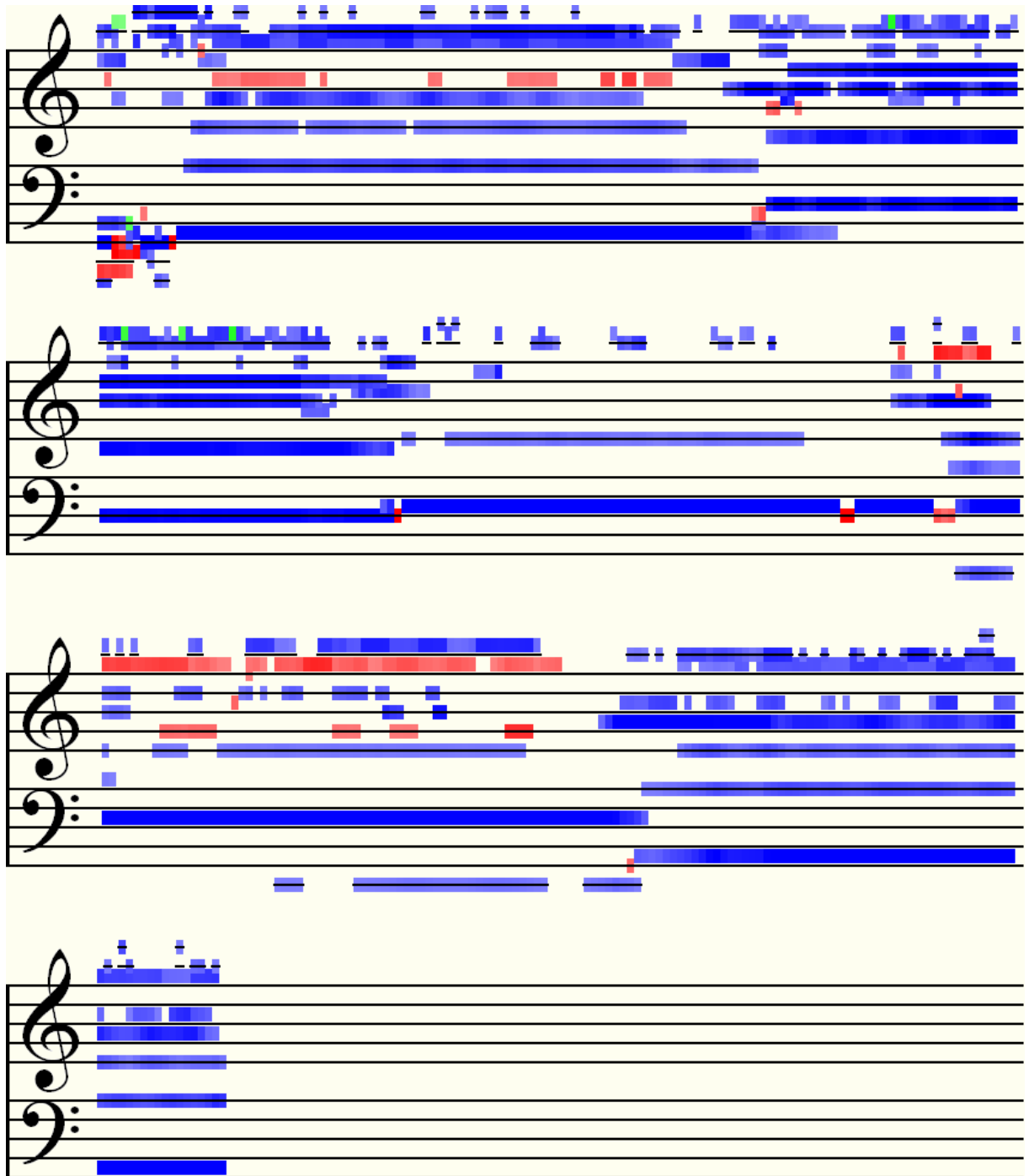


Figure 8.10 – CPM (Morlet 20) transcription

Derivative of Gaussian Wavelet		
Order: 2 (Mexican Hat)		
Ordinary transform	A very sparse result, similar to that of the Paul transform / extraction.	
With sensitivity	A slight improvement.	No difference to ordinary detection with sensitivity and without DTA.
With DTA	A few more frequencies detected, but still very bare.	

Table 8.23 – Results for CPM with DoG transform

Derivative of Gaussian Wavelet		
Order: 16		
Ordinary transform	Result is much improved, and although pitches are rather weakly detected, most are present and correct.	
With sensitivity	A very good result, but unfortunately the penultimate bass note is detected as its first harmonic.	Very good clarity was achieved with this setting, but was only rejected as being the best result because of the important bass note which the other two methods managed to detect.
With DTA	Clarity is very much improved for all detected pitches. Penultimate bass note still detected as its first harmonic.	

Table 8.23 (contd.) – Results for CPM with DoG transform

For comparison, as a matter of interest, **Figure 8.11** shows the result of pitch detection on the control clip, using the parameters given in **Table 8.19** which produced the best output. This is almost 100% accurate.

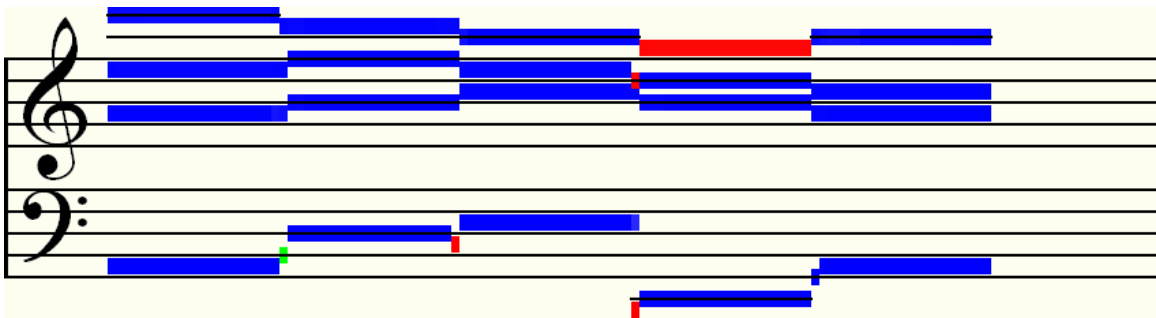


Figure 8.11 – CPM (Morlet 10) transcription of four-part synthesized control clip

9 From Pitch Graphs to Musical Scores

Up to this point, pre-processing techniques for pitch detection have been discussed in depth, but this is really only half the problem of music recognition. In order to complete a solution, further processing is required of the pitch and clarity information – the output of the pitch detection algorithms. Without this vital step, we cannot draw anything better than the current pitch graphs seen in Chapters 4 to 8, which are unacceptable representations of music in the eyes of a musician.

The issues which need to be dealt with in order to render a score from the pitch data comprise tasks **ii)** to **iv)** and also **vii)** from the overall research problem description (see Chapter 1). These may be combined into two main objectives:

- 1) Properly segment the time domain by locating note edges and determining durations from this, then quantize each note and fit them into bars of equal length, determined by a time signature.
- 2) Further scrutinize the detected pitches and eliminate artefacts, via melodic and harmonic analysis, then from this determine keys (if any) and spell pitches diatonically.

In many ways these steps are dependent on each other, but more information is to be acquired for the second task by tackling temporal note detection first. While methods have been offered for achieving **1)**, research into **2)** would fill a new thesis on its own, and so only the most tentative suggestions have been made as to how to approach this problem heuristically. The second step of task **1)** may require a non-trivial method if the time signature was not known *a priori*, however ideas for solving this have also been offered.

In general for this chapter, since the post-processing techniques presented here begin to fall outside the scope of this study, their descriptions have been kept intentionally short, and complete solutions have not been provided. Furthermore, the methods discussed have not been fully implemented in *Wave Processor* and so could not be tested thoroughly.

9.1 Temporal Note Detection

A useful by-product of the McLeod Pitch Method described in Chapter 4 is that it gives a very precise indication of the beginnings and endings of different notes within a melody, unless two notes in succession share the same pitch. Therefore the problem of (monophonic) temporal note detection is already nearly solved, quite incidentally. This is also the case with the CWT Pitch Method in Chapter 6, but for polyphonic music this becomes much more difficult, especially when dealing with non-percussive instruments which blend in well with each other. It was shown in Chapter 7 that difference tones yield different interference patterns in the spectrogram for different intervals, and therefore the onset of a new note (or change in harmony) would also be the point at which a particular pattern changes most dramatically. It is still not altogether clear, however, whether these changes are necessarily due to the evolution of a note or chord in terms of an instrument's particular timbre or note articulation, or whether they are as a result of a definite change in pitch. It is therefore useful, perhaps, to consider treating the problem as a separate issue and then to use the results to aid in the task of temporal segmentation. In [Bello04L] and [Brossier04L], some note onset detection functions have been offered, which will now be examined and reiterated here.

9.1.1 Note Onset Detection Functions

The first function described by Brossier is the *High Frequency Content function* (originally defined in [Masri96L]). This function has the effect of highlighting changes which occur in the higher frequency range of the spectrum, which means that it is suitable for locating percussive sounds, since these contain many high frequencies at their attack points. The HFC function takes windows of an STFT as its input, and is given by:

$$D_k^H = \sum_{m=0}^N m |F_{k,m}|,$$

where the STFT, $F_{k,m}$, has been defined in Chapter 4, section 4.1. As can be seen, the value of each point in the function is the sum of linearly weighted frequency magnitudes in each window. Since the values used are the bin magnitudes, we can also use windows of the CWT if so desired. Such an approach, however, suffers from the same problems as already exist from just examining MPM or CPM output – smoother transitions between chords and note attacks of non-percussive instruments will not be detected successfully by this function.

The next pair of functions compare bin frequencies in neighbouring windows of the STFT by their magnitude and their phase, resulting in a measure of *spectral difference* and *phase deviation*. The *Spectral Difference function* is defined by the following sum of differences function, which also operates on magnitudes:

$$D_k^S = \sum_{m=0}^N (|F_{k,m}| - |F_{k-1,m}|).$$

According to the Phase Vocoder algorithm (see Chapter 4, section 4.2.2), the difference in phase between similar frequency bins in neighbouring windows of the STFT will be ϕ plus a certain number of cycles of the frequency. If the frequency of a bin is stationary, then ϕ will change at a constant rate between successive windows. However, when a frequency changes, due to either a smooth transition to a new pitch or else a note onset/offset, a deviation in ϕ will occur. We define phase deviation as the second derivative of the instantaneous phase $\theta_{k,m}$ of bin m , window k as follows:

$$\tilde{\phi}_{k,m} = \frac{\partial^2 \theta_{k,m}}{\partial k^2}.$$

This may be estimated using finite differences as:

$$\tilde{\phi}_{k,m} = \theta_{k,m} - 2\theta_{k-1,m} + \theta_{k-2,m}.$$

The *Phase Deviation function* is then the average of all the phase deviations in each bin for each window. Note that the principle argument should be used for $\tilde{\phi}$, i.e. $-\pi < \tilde{\phi} \leq \pi$:

$$D_k^\phi = \frac{1}{N} \sum_{m=0}^N |\tilde{\phi}_{k,m}|.$$

Finally, by converting phase and magnitude back to the complex domain, we can use the phase deviation measure also to define overall frequency deviation for bins of the STFT as follows:

$$\tilde{F}_{k,m} = |F_{k,m}| e^{i\tilde{\phi}_{k,m}}.$$

9.1.2 The Combined Onset Detection Function

If we combine D_k^S and D_k^ϕ we can measure the difference between deviated and actual values, including phase information as well as magnitude, thereby obtaining a function which is much better at emphasizing new note or chord onsets by subtle changes in frequency. The difference is measured by calculating the Euclidean distance between \tilde{F} and F :

$$D_k^C = \frac{1}{N} \sum_{m=0}^N \|\tilde{F}_{k,m} - F_{k,m}\|^2.$$

According to Brossier, this effectively solves the problem of detecting both percussive onsets and smooth note transitions.

9.2 Note Classification via Image Processing Techniques

Since in the CWT spectrogram we have a detailed pictorial representation of sound, it is well worth looking at a few feature extraction techniques borrowed from image processing theory which could be highly beneficial. Revisiting the three-part arrangement of *Nkosi Sikeleli Africa* one last time, the image in **Figure 9.1** below shows the result of an experimental feature extraction, using a combination of techniques, as implemented in [McGuinness06S].

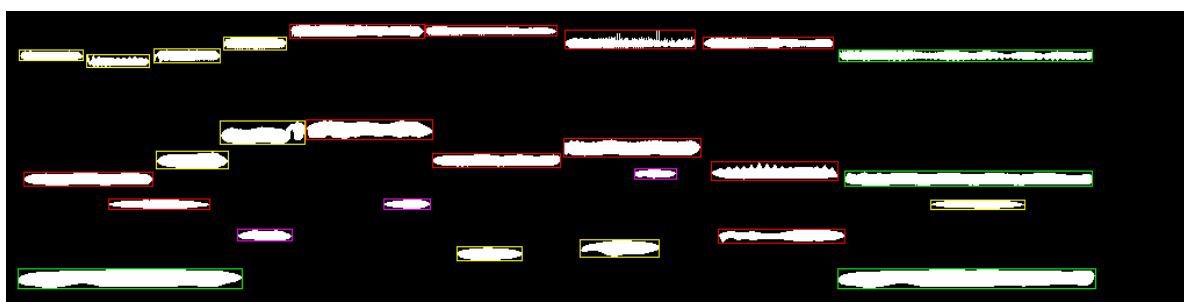
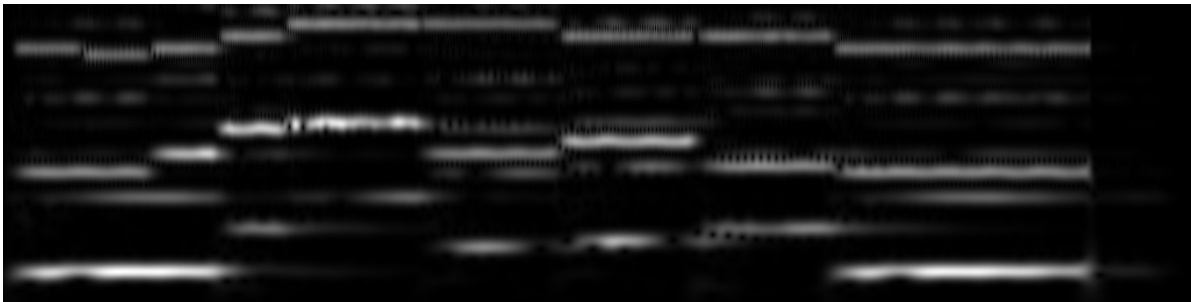


Figure 9.1 – Note identification of 3-part NSA example using Image Processing methods

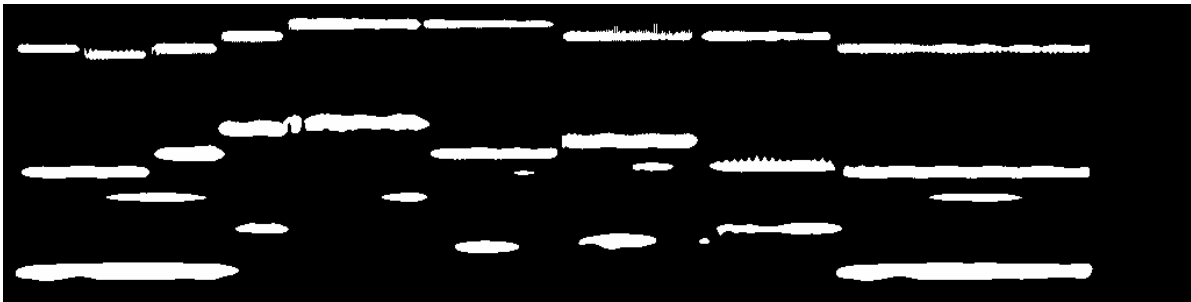
As can be seen, the notes have been extracted and identified as being one of four different types (which are explained below). Apart from some of the bass notes, the categorization is accurate and furthermore only three harmonics remain unfiltered.

There is not room in the scope of this study to allow for a very detailed discussion of all the theory, but the following is a summary of the process which yielded the above result. **Figure 9.2** shows the image at the first three stages of the procedure.

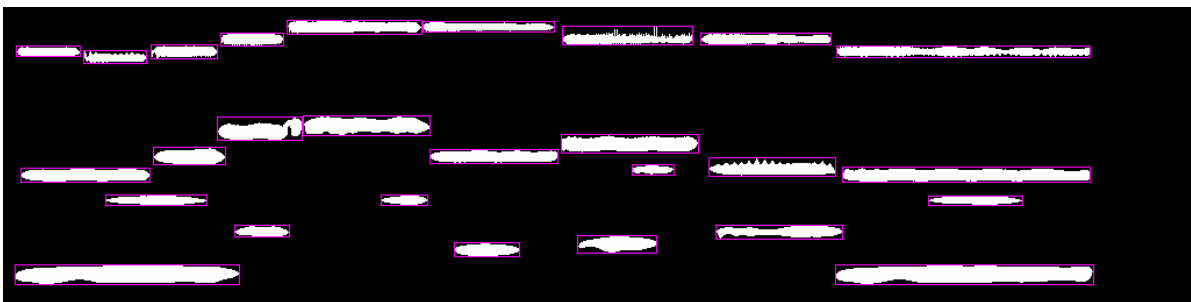
- a) Firstly a greyscale CWT spectrogram was generated, using the Morlet wavelet with a moderately high wavenumber of 20, so as to get good frequency localization but without sacrificing too much time resolution. Instead of stretching the spectrogram vertically, as for normal viewing purposes, the image was saved at its original height of 1 pixel per scale, which in this case was 256.
- b) The saved image was then binarized using a threshold of 72. This means that greys with a level of 72 – 255 (maximum) were selected and the rest filtered out. This simple process effectively removed a lot of the noise and weaker signal components.
- c) Next, connected components in the black and white image were identified and separated. The algorithm used for this is a scanner which examines the image, row by row, and determines whether or not a path exists from the current pixel to each of its neighbours. If a separated pixel is found, a new component is created, otherwise the pixel is added to that of the connected neighbour. The algorithm was set to reject components with a total pixel area of less than 200, thereby removing more noise.
- d) The final stage of the analysis was separation of the image components into four classes – one for each type of note. This was achieved by calculating various attributes of each shape, namely perimeter, compactness and elongation, and then classifying them by their similarity using *k*-means clustering [MacQueen67L].



a) CWT spectrogram saved as a greyscale image



b) Image converted to black and white by thresholding



c) Connected component analysis identifying notes

Figure 9.2 – Stages of note identification method leading to the result in Figure 9.1

In the classification stage, component elongation is an obvious choice by which to categorize, since it best represents duration. The reason for the inclusion of the other criteria is due the effects on the components' shapes from the way their extraction was done. Components caused by bass sounds tend to be slightly shorter and fatter, being more spread out in the frequency domain, whereas higher frequencies end up longer and thinner. This is evident in the images above. Note that if the spectrogram had not been drawn dyadically, however, the effect is reversed, but it is much more exaggerated and therefore cannot be exploited in this way. Compactness is simply a measure of how well a component fits inside its bounding rectangle. It was found via experimentation with different settings that including this attribute made the algorithm perform better on the whole, and so it was retained.

There is one more step before the data is ready to be combined with information from pitch methods, and this will clarify the purpose of doing the component classification. Most music is constructed using regular length beats, and, even if the tempo changes, the durations of almost all other nearby notes will be multiples of the shortest ones (see Chapter 3, section 3.3.3). Note types may therefore be more formally identified from the image by taking the average length of a few of the shortest type of component found nearby, and then measuring approximately how many of these fit into each of the other types of component. At the beginning of the current example, it may be seen for the first bass component, three of the shortest type (of which there are three in the top line and one in the middle) fit its length, therefore it may be identified as some type of dotted note. Notes which appear to stop and start between beats and out of synch with the general trend may be ignored, since they are likely to be harmonics, which vary in strength according to different reasons compared to sounds which have deliberate placement in measured time. However, notes which appear within the time boundaries of other larger notes should not be discarded, since they may be weaker fundamentals. These should be snapped to whichever beat position they are closest to. It may not always be the case, but in this example, the first rule eliminates one of the troublesome harmonics at the start, just below the middle line.

The lengths in pixels of each class of component in **Figure 9.1** (excluding the one eliminated harmonic) were measured and are shown in **Table 9.1**. This has been drawn so that the table columns are in proportion to the average width of the smallest component, in order to demonstrate how the quantization may be carried out.

56	55	58	55	117	114	113	115	219
112	63	74	110	111	119	110	110	214
						37	82	
194		48	41	57	69	110		203

Table 9.1 – Quantization of component widths

From here, it becomes a relatively straightforward task to combine the above appropriately time-quantized output with pitch information from the methods discussed in Chapters 6 and 7. This data may then be parsed, and a MusicXML file, for example, generated. Finally, the MusicXML could be imported into *Sibelius* or another music publishing package which supports this format. **Figure 9.3** shows a *Sibelius* score, obtained by using the pitch and note type information from combined algorithms. Compare this with **Figure 5.24** in Chapter 5, which shows the original. This result may be improved even further with the likes of **Figure 7.17** in Chapter 7, which completely eliminates the problem of the fading bass notes and would also get rid of all but the last harmonic.



Figure 9.3 – Full automatic transcription of 3-part NSA example, imported into *Sibelius*

The key signature of this score was determined by first taking all identified pitches in the passage and arranging them in a scale, within one octave:

A B C D E F# G

By examining the intervals between the notes in this scale, it may be seen that the Ionian mode (major) may be formed by starting on G, and the Aeolian mode (minor) would begin on E. These two options account for 90% of all Western music (which is a rough yet educated estimate). By virtue of the fact that the musical phrase begins and ends on a G Major triad, and that there is a perfect cadence in G formed by the last two chords, the most likely key is thus G Major. Three and four-part chords which form common patterns such as this may easily be stored and searched for in a lookup table, and thus cadences may be recognized; however, we don't even need to go nearly this far with the analysis. Simply detecting that any F in the passage of music is always sharpened is enough to write a correct key signature, even if we are wrong and the key is technically E Minor and not G Major.

The only remaining assumption which had to be made was concerning the number and type of beats per bar, but nine times out of ten in music, the beat type will be a quarter note, as defined here. As for the number of beats, one clue is that the total number of the smallest denomination note (the quaver) which fits into the whole phrase is 16. This number is a multiple of 2 and of 4, as opposed to 3, or 5, or 6. Therefore, assuming a whole number of bars, the best subdivision of the passage is into either two sets of eight quavers or four sets of four quavers. The latter configuration is less common, but does exist in Music, and would have a time signature of either $\frac{2}{4}$ or $\frac{4}{8}$. The choice of $\frac{4}{4}$ (the former division) is not just because it is more popular or commonly occurring, but also because the best way in Music Theory to identify the beat is to determine which note type is most frequent. This is logical, since repetitiveness tends to give something a defining quality. The most frequent class of note which appeared in the component classification was that of the red bounding rectangles in **Figure 9.1**. This note type is twice the smallest in length, and so the smallest notes are therefore subdivisions rather than main beats. Given the assumption about the beat type, eight crotchets per bar is an extremely rare time signature, and so the best solution is two bars of four crotchets each.

Although done manually in this demonstration, it has now been shown that the score in **Figure 9.3** may also be computed by a fully automatic method, given the audio signal alone.

10 Conclusions

Thorough attempts have been made to solve research problems **i)** and **v)**, and to some extent **ii)** and **iii)**, as defined in Chapter 1, section 1.2. This chapter summarizes and evaluates the current solutions, and suggests ways in which they may be improved. Some ideas for further development of methods, towards a more complete solution to automatic transcription, have also been offered.

10.1 Evaluation of Results

10.1.1 Monophonic Recognition

Several years before the time of writing of this thesis, single pitch recognition was already well understood and successfully implemented in a number of pitch detection applications. The point of including a thorough discourse on melody transcription methods again was to find ways of improving and evolving them, or at least use them as part of a broader solution. With the McLeod Pitch Method, this was indeed seen to be the case, since fair to decent results were achieved by combining it with the Redundant Haar wavelet transform, as proposed in Chapter 5.

Certainly with the Phase Vocoder this is also the case, since the implementation shows that by searching for the next highest peak in windows of the STFT, multiple pitch extraction may be achieved. Although this algorithm was tested less thoroughly, not being the main focus of this research, it is very important to consider, since it is the current algorithm of choice for most automatic transcription applications in the field. For this reason, the Phase Vocoder may be used as a standard against which to measure the level of success of other methods.

The note onset detection method discussed briefly in the previous chapter works well for single melody lines, but could not be tested on polyphonic material. This is because the algorithm only provides information about when a change has occurred, throughout the entire spectrum of frequencies, and not to which note the variance measure applies. In a way, it is similar to the case of the Fourier transform yielding frequency but not time information. This would seem to suggest another frequency domain windowing approach be used. However the connected components analysis demonstrated in the same chapter may be a better method. This would, however, require further testing in order to construct a more conclusive argument.

10.1.2 Polyphonic Recognition

Given the high level of accuracy in the results of four and five-part harmony transcriptions, the new method of multiple pitch detection using the continuous wavelet transform has proved, at the very least, to be comparable with current techniques implemented in the field. While the pre-processing part of this algorithm has been well-examined and implemented, the actual pitch extraction is in need of further development and exploration of different techniques, of which there are many, as implied in Chapter 9.

It would appear from the general clarity achieved in the relatively simple examples in Chapter 8 that the CWT pitch method almost always outperforms the Phase Vocoder for multiple pitch recognition. However, there are many parameters which must be set, as shown in the experiments, in order for the new method to yield these good results every time. It is also

apparent that the content affects the correct choice of method and parameter settings, sometimes dramatically, which presents a problem. While one configuration may be perfect for some types of music, it may not work as well for others, even given similar instruments and recording techniques but different harmonic material. For example, for higher frequency content, a lower Morlet wavenumber may be used, since the frequency spread it provides in this range gives a slightly clearer measure.

One solution is to make better use of the Phase Vocoder. Since this is a much faster algorithm, it could do very well as a *pre*-pre-processing tool, feeding the CWT algorithm data concerning the overall complexity of the music about to be analyzed. If relatively few frequencies are detected by the Phase Vocoder, then the transcription system may choose a simpler method in preference to the CWT, such as the Redundant Haar / MPM combination. If, however, the Phase Vocoder found the audio to be much richer in harmonic content, the more scrutinous CWT pitch method would be assigned to the pre-processing task. The type and order of the wavelet to be used could also be determined from this data. For example, it is apparent that thicker musical texture requires higher order wavelets, but higher frequency content does not, and so on. Thus, inclusion of this step means that development of a robust automatic algorithm-selection method is entirely possible.

From one or two of the results in the previous chapter, where the Paul wavelet does not seem very suitable for this particular application, the Derivative of Gaussian is certainly worth further attention. In some cases, quieter lower to middle range frequencies were best detected by this transform. Also, unwanted harmonics tended to be filtered out better than in the Morlet transform. The only reason higher orders had not been implemented in *Wave Processor* was because of the difficulty of calculating the large factors in the normalizing constants due to the Gamma function (see **Appendix A.3**) but this was only for display purposes. However, higher order wavelets could prove more accurate, just as the higher wavenumber Morlet wavelets have shown to be.

10.1.3 Usefulness of Music Transcription Software

One of the primary and most important measures of the usefulness of any given piece of software is how much time is saved by using it to perform a certain task. The most popular applications are either those which provide entertainment or those which make otherwise tedious work relatively quick and easy to complete.

First and foremost, a program claiming to assist in the job of music transcription *must* make the problem faster and easier to solve than doing it by ear, even if it is only a partial solution which the user must then complete. It must also prove to be more accurate and more discerning than the average experienced music transcriber – a quick, yet poor solution is still of little use if too many corrections need to be made.

Figure 10.1 shows another **unedited** automatic transcription of the same three-part arrangement of *Nkosi Sikeleli Africa* as demonstrated in Chapter 9. This transcription was computed by one of the current most highly rated commercial music transcription systems available, Neuratron's *AudioScore Ultimate 6* [Neuratron09W].

While the program does indeed get all of the pitches correct, with only four or five extraneous harmonics detected, there are many major errors in the assignment of note values and their placement in time, which appear to be due to rather perfunctory post-processing techniques. These errors make the correctional editing job for the user much more difficult, demonstrating that precise temporal location is just as important as correct pitch detection when transcribing



Figure 10.1 – *AudioScore* automatic transcription of NSA 3-part arrangement

music. For this reason, as it stands, *AudioScore* cannot truly be deemed useful. It is clear, however, that this is not in fact the “ultimate” version of the software, since the tempo and time signature options dialog box currently shows many greyed out settings, which could help improve the output. Presumably these features are yet to be implemented by the developers. The other somewhat more obvious requirement of an automatic music transcription system is that its output should be easily imported into a music publishing application, such as *Sibelius*, or else provide its own comprehensive set of engraving and publishing tools. For this reason, although **Figure 9.3** is a better result than **Figure 10.1**, this link has not yet been provided in *Wave Processor*, and therefore it cannot claim to be a complete solution, as *AudioScore* can.

10.2 Ideas for Further Research

10.2.1 Improving the Speed of Algorithms

Attempts have been made to optimize most methods implemented in *Wave Processor*, but there is still much room for further improvement. Firstly, there exist more advanced algorithms than those presented here which are able to calculate transforms faster. For example, as mentioned in Chapter 4, the FFT may be computed using higher radices. There also exist hardware devices, such as certain field-programmable gate arrays, configured to dedicated FFT calculation. Some graphics processing units, such as [Wang07W] are now able to calculate DWTs, since Daubechies wavelets have become part of the JPEG2000 standard, and so wavelet transforms have become necessary stock requirements for codecs.

Studies have also been conducted with regards to finding a more efficient CWT algorithm, including one by Michael Vrhel [Vrhel97L] claiming to achieve $O(N)$ per scale. This means that the FFT does not have to be used in the calculation at all.

10.2.2 Artificially Intelligent Pattern Recognition Techniques

This study mainly concerns itself with pre-processing methods of music transcription. Of course this is really only half the battle. Purely empirical sound analysis methods are analogous with what goes on in our inner ear. Sounds are filtered and converted into signals which are more suitable for our brain to digest, but further analysis must be done before we are able to recognize sounds as musical pitches and glean their context.

Chapter nine touches lightly on this subject, but does not suggest methods beyond the empirical. There are many other “artificial intelligence” techniques of which the intention is to explore thoroughly, given the findings of research such as [Abdallah02L], [Andreão07L], [Cont07L] and others. The subjects techniques of interest are:

- Machine Learning
- Neural Networks
- Hidden Markov Models

The last in the list looks particularly promising as a method for harmonic analysis, since it provides a method of modelling based on the evaluation of the probability of a sequence of observations occurring. Given the rules, or rather general trends, of music harmony discussed in Chapter 3, an analysis/prediction algorithm could be constructed, whose output is information about the likelihood of a pattern of detected pitches belonging to, say, melodic lines or counterpoint, or else being part of the structure of a cadence. It may also possibly be used to predict the instrument on which a line of music is being played, although this is only hypothetical. Transcription of notes based on their harmonic function is a very powerful technique, and one that is recommended by musicologists for manual transcription [Scholes65L].

10.2.3 Importing pitch/time data into music engraving software

The previous section discussed the importance of making output of an automatic transcription immediately accessible and editable to the user, but the favoured method of doing so has only been tentatively suggested in Chapter 3. While further theory will not be discussed, since this is the concluding chapter, it is intended that the MusicXML standard [Recordare09W] be explored thoroughly and incorporated into *Wave Processor*, since this is arguably the most widely supported format for representing music. Research, such as [Cunningham04L], has also shown that it is one of the best, in that:

- It allows representation of music from very simple to extremely complex scores.
- It enjoys full support and continued development by the company which created it, Recordare LLC.
- The data structuring and organization of MusicXML is based on an already robust and widely used mark-up language, namely XML.
- The use of plain text to store data means that it is easily edited manually, but also that it is compact and takes up little space when stored. It may also be compressed further if desired by standard lossless techniques.
- Implementation of MusicXML is included in all major applications which deal with either music engraving or music transcription.
- It comes with a free licence agreement, which means it may be used freely by software developers in the music transcription / engraving industry, provided that the terms of the licence are observed.

Appendix A – Mathematical Notation

A.1 Notation Conventions

The following table summarizes the notational standard for most of the mathematical formulae found in this thesis:

Symbol	Designation	Notes
t	time	value in seconds
τ	shift in time	value in seconds
$f(t)$	continuous function of time	
$f[t]$	sampled version of $f(t)$	
f_n	discrete function (of time) indexed by n	
λ	wavelength	λ_k is also used to denote eigenvalues (chapter five)
ν	frequency	value in Hertz
ω	angular frequency	where $\omega = 2\pi\nu$
p	pitch	measured in semitones from the base pitch, defined as $C_0 = 0$
$F(\nu)$	continuous Fourier transform	this is specifically of $f(t)$
F_m	discrete Fourier transform, indexed by m	also written \hat{f}_m
$\mathcal{F}, \mathcal{F}^{-1}$	Fourier transform operator and its inverse	
k, n, m	index variables	n is normally used for time series and m for frequency arrays. t_n & ν_m are values of t & ν at indexes n & m respectively. n is also used for Paul and DoG wavelet orders.
B	bandwidth of a signal	measured in Hertz
N	number of samples in a discrete signal	
L	duration or time limit of a non-stationary signal	measured in seconds
Δt	change in time between two adjacent points in a discrete time series	measured in seconds
$\Delta \nu$	change in frequency between two adjacent points in a discrete frequency series	measured in Hertz
$F_{k,m}$	discrete Short Time Fourier Transform, indexed by k (windows), m (frequencies)	
W	window width in windowed transforms, i.e. the number of samples in the window	also W_N for N^{th} root of unity (<i>twiddle factors</i>) in Cooley-Tukey FFT algorithm
Λ	duration or time limit of a window	measured in seconds

Table A.1 – Mathematical notation conventions

w_{n-kW}	discrete window function (shifted)	The window is at position n shifted by k window widths
θ	instantaneous phase angle	measured in radians
$\phi(t)$	father wavelet or scaling function	ϕ is also used to denote phase difference (chapters four and nine)
$\psi(t)$	mother wavelet or wavelet function	also $\psi_0(t)$
$\psi_{s,\tau}(t)$	wavelet function at scale, s and shift, τ	
s	scale	
S	number of scales in a discrete transform	
$\Psi^f(s,\tau)$	continuous wavelet transform	of $f(t)$
$\Psi_{s,\tau}^f$	discretized CWT	of discrete signal f_n N. B. s and τ are now indexes as opposed to values
*	complex conjugate	
\otimes	convolution	
!!	double factorial	$n!! = \begin{cases} 1, & n = 0, n = 1 \\ n(n-2)!!, & n > 1 \end{cases}$
$H(\omega)$	Heaviside or Step function	$H(\omega) = \begin{cases} 0, & \omega \leq 0 \\ 1, & \omega > 0 \end{cases}$
$\Gamma(z)$	Gamma function	$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$
$\Pi(t)$	Box function	$\Pi(t) = \begin{cases} 1, & t \leq \frac{1}{2} \\ 0, & t > \frac{1}{2} \end{cases}$
$He_n(t)$	n^{th} Hermite probabilistic polynomial in t	$He_0(t) = 1$ $He_1(t) = t$ $He_{n+1}(t) = tHe_n(t) - nHe_{n-1}(t)$
$\Phi_n^{\text{Re}}(t)$ and $\Phi_n^{\text{Im}}(t)$	Paul wavelet generator polynomials in t	See Appendix B.2

Table A.1 (contd.) – Mathematical notation conventions

A.2 Window Functions

Name	Formula	Parameter	Diagram
Rectangular	$w(t) = 1$		
Gaussian	$w(t) = e^{-\frac{1}{2}\left(\frac{t-W/2}{\sigma w/2}\right)^2}$	$\sigma \leq 0.5$	
Triangular	$w(t) = 1 - \left \frac{2t}{W} - 1\right $		
Cosine	$w(t) = \sin\left(\frac{\pi t}{W}\right)$		
Hann	$w(t) = 0.5\left(1 - \cos\left(\frac{2\pi t}{W}\right)\right)$		
Hamming	$w(t) = 0.54 - 0.46\cos\left(\frac{2\pi t}{W}\right)$		
Blackman	$w(t) = \frac{1-\alpha}{2} - \frac{1}{2}\cos\left(\frac{2\pi t}{W}\right) + \frac{\alpha}{2}\cos\left(\frac{4\pi t}{W}\right)$	$\sigma = 0.16$	

Table A.2 – Window functions for the STFT (images from www.wikipedia.org)

A.3 Continuous Wavelet Functions

Name	$\psi_0(t)$	$\hat{\psi}_0(\omega)$
Morlet ($\omega_0 \geq 5$)	$\psi_0^M(t) = \pi^{-\frac{1}{4}} e^{i\omega_0 t} e^{-\frac{t^2}{2}}$	$\hat{\psi}_s^M(\omega) = \pi^{-\frac{1}{4}} H(\omega) e^{-\frac{1}{2}(s\omega - \omega_0)^2}$
Derivative of Gaussian ($n \geq 2$)	$\psi_0^{D_n}(t) = \frac{(-1)^{n+1}}{\sqrt{\Gamma(n + \frac{1}{2})}} \frac{d^n}{dt^n} e^{-\frac{t^2}{2}}$	$\hat{\psi}_0^{D_n}(\omega) = \frac{-i^n}{\sqrt{\Gamma(n + \frac{1}{2})}} \omega^n e^{-\frac{\omega^2}{2}}$
Paul ($n \geq 4$)	$\psi_0^{P_n}(t) = \frac{2^n i^n n!}{\sqrt{\pi(2n)!}} (1 - it)^{-(n+1)}$	$\hat{\psi}_s^{P_n}(\omega) = \frac{2^n}{\sqrt{n(2n-1)!}} H(\omega) (s\omega)^n e^{-s\omega}$
Shannon	$\psi_0^S(t) = \text{sinc}\left(\frac{\pi t}{2}\right) \cos\left(\frac{3\pi t}{2}\right)$	$\hat{\psi}_s^S(\omega) = \prod\left(\frac{s\omega - 3\pi/2}{\pi}\right) + \prod\left(\frac{s\omega + 3\pi/2}{\pi}\right)$

Table A.3 – Continuous wavelet functions and their Fourier transforms

Appendix B – Algebraic Workings

B.1 Finding the Points of the Daubechies 4 Scaling & Wavelet Functions

The D4 function coefficients are given as:

$$h_0 = \frac{1}{4}(1 + \sqrt{3}),$$

$$h_1 = \frac{1}{4}(3 + \sqrt{3}),$$

$$h_2 = \frac{1}{4}(3 - \sqrt{3}),$$

$$h_3 = \frac{1}{4}(1 - \sqrt{3}).$$

B.1.1 The Scaling Function

$\phi(t)$ is given by the following equation, where $0 \leq t < 3$:

$$\phi(t) = h_0\phi(2t) + h_1\phi(2t - 1) + h_2\phi(2t - 2) + h_3\phi(2t - 3). \quad [\text{B.1}]$$

The integer points, $\phi(1)$ and $\phi(2)$, are then:

$$\phi(1) = h_1\phi(1) + h_0\phi(2).$$

$$\phi(2) = h_3\phi(1) + h_2\phi(2).$$

To find eigenvalues, λ_1 and λ_2 , we must find the determinant of the matrix, M_2 , such that $M_2\mu = \lambda_k\mu$, where:

$$M_2 = \begin{bmatrix} h_1 & h_0 \\ h_3 & h_2 \end{bmatrix}, \quad \text{and} \quad \mu = \begin{bmatrix} \phi(1) \\ \phi(2) \end{bmatrix}.$$

Thus:

$$(h_1 - \lambda)(h_2 - \lambda) - h_0h_3 = 0.$$

$$\therefore \lambda^2 - (h_1 + h_2)\lambda + h_1h_2 - h_0h_3 = 0.$$

$$\therefore 16\lambda^2 - 4(3 + \sqrt{3} + 3 - \sqrt{3})\lambda + (3 + \sqrt{3})(3 - \sqrt{3}) - (1 + \sqrt{3})(1 - \sqrt{3}) = 0.$$

$$\therefore 16\lambda^2 - 3\lambda + 1 = 0.$$

$$\therefore (2\lambda - 1)(\lambda - 1) = 0.$$

$$\therefore \lambda_1 = 1, \lambda_2 = \frac{1}{2}.$$

The eigenvector for $\lambda_1 = 1$ is:

$$\begin{bmatrix} h_1 & h_0 \\ h_3 & h_2 \end{bmatrix} \begin{bmatrix} \phi(1) \\ \phi(2) \end{bmatrix} = \begin{bmatrix} \phi(1) \\ \phi(2) \end{bmatrix}.$$

Hence

$$(h_1 - 1)\phi(1) + h_0\phi(2) = 0.$$

$$h_3\phi(1) + (h_2 - 1)\phi(2) = 0.$$

Observe that

$$h_0 = 1 - h_2, \quad \text{and} \quad h_3 = 1 - h_1.$$

Therefore

$$\phi(1) = \frac{h_0}{h_3}\phi(2), \quad \text{and} \quad \phi(2) = \frac{h_3}{h_0}\phi(1). \quad \text{[B.2]}$$

$$\therefore \mu = \begin{bmatrix} \phi(1) \\ \phi(2) \end{bmatrix} = \phi(1) \begin{bmatrix} 1 \\ \frac{h_3}{h_0} \end{bmatrix} = \phi(2) \begin{bmatrix} \frac{h_0}{h_3} \\ 1 \end{bmatrix}.$$

Let $|\mu| = \sqrt{2}$, then

$$\phi(1)^2 + \left(\phi(1)\frac{h_3}{h_0}\right)^2 = 2.$$

$$\therefore \phi(1)^2 = \frac{2}{\left(1 + \left(\frac{h_3}{h_0}\right)^2\right)}.$$

$$\therefore \phi(1) = \frac{1 + \sqrt{3}}{2}.$$

Finally, from [B.2]:

$$\begin{aligned} \phi(2) &= \frac{h_3}{h_0}\phi(1) \\ &= \frac{1 - \sqrt{3}}{2}. \end{aligned}$$

The half integer points, $\phi(\frac{1}{2})$, $\phi(\frac{3}{2})$, and $\phi(\frac{5}{2})$, and subsequently all dyadic fractional points, are then determined from equation [B.1] as follows:

$$\begin{aligned}\phi\left(\frac{1}{2}\right) &= h_0\phi(1) \\ &= \frac{(1+\sqrt{3})}{4} \cdot \frac{(1+\sqrt{3})}{2} \\ &= \frac{1+2\sqrt{3}+3}{8} \\ &= \frac{2+\sqrt{3}}{4}.\end{aligned}$$

$$\begin{aligned}\phi\left(\frac{3}{2}\right) &= h_1\phi(2) + h_2\phi(1) \\ &= \frac{(3+\sqrt{3})}{4} \cdot \frac{(1-\sqrt{3})}{2} + \frac{(3-\sqrt{3})}{4} \cdot \frac{(1+\sqrt{3})}{2} \\ &= 0.\end{aligned}$$

$$\begin{aligned}\phi\left(\frac{5}{2}\right) &= h_3\phi(2) \\ &= \frac{(1-\sqrt{3})}{4} \cdot \frac{(1-\sqrt{3})}{2} \\ &= \frac{1-2\sqrt{3}+3}{8} \\ &= \frac{2-\sqrt{3}}{4}.\end{aligned}$$

From here, the quarter, eighth, sixteenth integer points, and so on, may be found recursively by subdividing intervals ad infinitum.

B.2.2 The Wavelet Function

The derivation of the points for the D4 mother wavelet function, $\psi(t)$, is trivial once the scaling function has been calculated. This is achieved simply by substituting values of the points found for $\phi(t)$ in the wavelet equation:

$$\psi(t) = h_3\phi(2t) + h_2\phi(2t-1) + h_1\phi(2t-2) + h_0\phi(2t-3).$$

B.2 Extracting the Real and Imaginary Components of the Paul Wavelet

The Paul wavelet (order n) is defined as:

$$\psi_0^{P_n}(t) = \alpha i^n (1 - it)^{-(n+1)},$$

where

$$\alpha = \frac{2^n n!}{\sqrt{\pi(2n)!}}.$$

In order to plot the real and imaginary parts of the function, the expression $i^n(1 - it)^{-(n+1)}$ needs be written as the sum of its real and imaginary components, which are unknown polynomials in t .

$$i^n(1 - it)^{-(n+1)} = P_n^{\text{Re}}(t) + iP_n^{\text{Im}}(t). \quad [\text{B.3}]$$

B.2.1 Examining the First Four Orders

The first four orders of the complex function expressed in terms of separate real and imaginary components are as follows:

$n = 0$:

$$\begin{aligned} i^0(1 - it)^{-1} &= \frac{1}{1 - it} \\ &= \frac{1 + it}{(1 - it)(1 + it)} \\ &= \frac{1}{1 + t^2} + i \frac{t}{1 + t^2}. \end{aligned}$$

$n = 1$:

$$\begin{aligned} i^1(1 - it)^{-2} &= \frac{i}{1 - 2it + i^2 t^2} \\ &= \frac{1}{-2t + i(t^2 - 1)} \\ &= \frac{-2t - i(t^2 - 1)}{4t^2 + (t^2 - 1)^2} \end{aligned}$$

$$\begin{aligned}
&= \frac{-2t - i(t^2 - 1)}{4t^2 + t^4 - 2t^2 + 1} \\
&= \frac{-2t - i(t^2 - 1)}{t^4 + 2t^2 + 1} \\
&= \frac{-2t}{(1+t^2)^2} + i \frac{1-t^2}{(1+t^2)^2}.
\end{aligned}$$

$n = 2$:

$$\begin{aligned}
i^2(1-it)^{-3} &= \frac{-1}{1-3it-3t^2+it^3} \\
&= \frac{-(1-3t^2)+i(-3t+t^3)}{(1-3t)^2+(-3t+t^3)^2} \\
&= \frac{-(1-3t^2)-i(3t-t^3)}{1+3t^2+3t^4+t^6} \\
&= \frac{3t^2-1}{(1+t^2)^3} + i \frac{-3t+t^3}{(1+t^2)^3}.
\end{aligned}$$

$n = 3$:

$$\begin{aligned}
i^3(1-it)^{-4} &= \frac{-i}{1-4it-6t^2+4it^3+t^4} \\
&= \frac{1}{4t-4t^3+i(1-6t^2+t^4)} \\
&= \frac{4t-4t^3-i(1-6t^2+t^4)}{(4t-4t^3)^2+(1-6t^2+t^4)^2} \\
&= \frac{4t-4t^3-i(1-6t^2+t^4)}{1+4t^2+6t^4+4t^6+t^8} \\
&= \frac{4t-4t^3}{(1+t^2)^4} + i \frac{-1+6t^2-t^4}{(1+t^2)^4}.
\end{aligned}$$

We see that a pattern begins to emerge:

$$i^n (1 - it)^{-(n+1)} = \frac{\Phi_n^{\text{Re}}(t) + i\Phi_n^{\text{Im}}(t)}{(1 + t^2)^{n+1}},$$

where $\Phi_n^{\text{Re}}(t)$ and $\Phi_n^{\text{Im}}(t)$ are the unknown polynomials in equation [B.3] multiplied by $(1 + t^2)^{n+1}$. From the first few cases, we hypothesize the following recurrence relations:

$$\Phi_{n+1}^{\text{Re}}(t) = -\Phi_n^{\text{Im}}(t) - t\Phi_n^{\text{Re}}(t);$$

$$\Phi_{n+1}^{\text{Im}}(t) = \Phi_n^{\text{Re}}(t) - t\Phi_n^{\text{Im}}(t).$$

B.2.2 Proof

Given that

$$i^n (1 - it)^{-(n+1)} (1 + t^2)^{n+1} = \Phi_n^{\text{Re}}(t) + i\Phi_n^{\text{Im}}(t),$$

then

$$\begin{aligned} & i^{n+1} (1 - it)^{-(n+1+1)} (1 + t^2)^{(n+1+1)} \\ &= i^n (1 - it)^{-(n+1)} (1 + t^2)^{n+1} \left[i \frac{(1 + t^2)}{(1 - it)} \right] \\ &= [\Phi_n^{\text{Re}}(t) + i\Phi_n^{\text{Im}}(t)] \cdot \left[\frac{i(1 + t^2)(1 + it)}{(1 - it)(1 + it)} \right] \\ &= [\Phi_n^{\text{Re}}(t) + i\Phi_n^{\text{Im}}(t)] \cdot \left[\frac{i(1 + t^2)(1 + it)}{(1 + t^2)} \right] \\ &= [\Phi_n^{\text{Re}}(t) + i\Phi_n^{\text{Im}}(t)] \cdot (i - t) \\ &= -\Phi_n^{\text{Im}}(t) - t\Phi_n^{\text{Re}}(t) + i[\Phi_n^{\text{Re}}(t) - t\Phi_n^{\text{Im}}(t)] \\ &= \Phi_{n+1}^{\text{Re}}(t) + i\Phi_{n+1}^{\text{Im}}(t). \end{aligned}$$

Appendix C – Code Listings

C.1 Fast Fourier Transform (FFT)

```
void FFT(int dir, DWORD n, double *rdata, double *idata = NULL,
        BOOL c2r = FALSE) {
    /* Real to real, real to complex, complex to real or complex to complex
       n-point Fast Fourier Transform

       idata == NULL: real to real
       c2r == TRUE: complex to real

       Forward - dir == 1
       Inverse - dir == -1
    */

    // Check for valid input
    if(!(dir == 1 || dir == -1))
        return;
    if(n <= 1)
        return;
    if(!rdata) return;

    BOOL r2r = FALSE;
    // If real to real transform, create temp imaginary parts array and set
    to 0
    if(!idata) {
        r2r = TRUE;
        c2r = FALSE;
        idata = new double[n];
        memset(idata, 0, n * sizeof(double));
    }

    // Bit reversal of real parts
    DWORD i, j = 0, k;
    double tmp;
    for(i = 0; i < n - 1; i++) {
        if(i < j) {
            tmp = rdata[i];
            rdata[i] = rdata[j];
            rdata[j] = tmp;
            if(!r2r) {
                tmp = idata[i];
                idata[i] = idata[j];
                idata[j] = tmp;
            }
        }
    }

    k = n;
    do {
        k >>= 1;
        j ^= k;
    } while(!(j & k));
}
```

```

// Combine Transforms
DWORD npoints, jstep;
double wr, wrt, wi, wmr, wmi, tempr, tempi;

wmr = -1.0; // w = 2pi
wmi = 0.0;

for(npoints = 1; npoints < n; npoints = jstep) {
    jstep = npoints << 1;
    wr = 1.0;
    wi = 0.0;
    for(i = 0; i < npoints; i++) {
        for(j = i; j < n; j += jstep) {
            k = j + npoints;
            // wG2(x)
            tempr = wr * rdata[k] - wi * idata[k];
            tempi = wr * idata[k] + wi * rdata[k];
            // F(x + n/2) = G1(x) - wG2(x)
            rdata[k] = rdata[j] - tempr;
            idata[k] = idata[j] - tempi;
            // F(x) = G1(x) + wG2(x)
            rdata[j] += tempr;
            idata[j] += tempi;
        }
        wr = (wrt = wr) * wmr - wi * wmi;
        wi = wrt * wmi + wi * wmr;
    }
    wmi = -dir * sqrt((1.0 - wmr) / 2.0);
    wmr = sqrt((1.0 + wmr) / 2.0);
}

// If real to real transform, calculate magnitude of complex numbers
// If forward transform, scale by 1/n
if(r2r || c2r || dir == 1) {
    for(i = 0; i < n ;i++) {
        if(r2r || c2r)
            rdata[i] = sqrt(rdata[i] * rdata[i] + idata[i] * idata[i]);
        if(dir == 1) {
            rdata[i] /= n;
            idata[i] /= n;
        }
    }
}

// Finished with imaginary parts
if(r2r) delete idata;
}

```

C.2 Autocorrelation Function (ACF)

```
double ACF(double *in, double *out, DWORD N) {
    /* Fast ACF algorithm using Fourier Transform */

    // in - Input samples
    // out - Output data
    // N - Width of padded window

    // Copy data to output array
    memcpy(out, in, N * sizeof(double));
    // Create empty imaginary array
    double *idata = new double[N];
    memset(idata, 0, N * sizeof(double));

    // Do forward Fourier Transform
    FFT(1, N, out, idata);

    // Multiply by complex conjugates
    for(DWORD i = 0; i < N; i++) {
        out[i] = out[i] * out[i] + idata[i] * idata[i];
        idata[i] = 0.0;
    }

    // Do inverse Fourier Transform
    FFT(-1, N, out, idata);
    delete idata;
    return out[0];
}
```

C.3 Normalized Square Difference Function (NSDF)

```
double NSDF(double *in, double *out, DWORD N, DWORD n) {
    /* Normalized Square Difference Function */

    // Find ACF
    double ss = ACF(in, out, N) * 2; // SSF[0]

    // Subtract appropriate normalized terms to get subsequent terms of SSF
    for(DWORD i = 0; i < n; i++) {
        out[i] = 2.0 * out[i] / ss;
        ss -= in[i] * in[i] / N;
        ss -= in[N - i - 1] * in[N - i - 1] / N;
    }

    return out[0];
}
```

C.4 Adapted Peak-Picking Algorithm

```
UINT PickPeaks(double *nsdf, UINT *Peaks, DWORD n) {
    /* Algorithm adapted from MPM (Phil McCleod) */

    // nsdf - Normalized Square Difference Function as input
    // n - Number of terms in NSDF
    // Peaks - Array of maxima as output
    // k - Peak picking threshold constant
    // Output - Number of maxima found

    memset(Peaks, 0, MAX_PEAKS * sizeof(UINT));
    UINT i, np, pmax;

    // Find first negative zero crossing
    for(i = 0; i < n - 1 && nsdf[i] >= 0.0; i++);

    // Start looking for peaks
    for(np = 0; np < MAX_PEAKS && i < n - 1; np++) {
        // Find next positive zero crossing
        for(; i < n - 1 && nsdf[i] <= 0.0; i++);
        if(i == n - 1) return np;

        // Crossed over zero line - Look for a peak
        pmax = i;
        for(; nsdf[i] > 0.0; i++) {
            for(; i < n - 1 && nsdf[i] > 0.0 &&
                (nsdf[i] <= nsdf[i - 1] || nsdf[i] < nsdf[i + 1]); i++);
            if(i == n - 1) return np;
            if(nsdf[i] <= 0.0) break;

            // Found a local peak
            if(nsdf[i] > nsdf[pmax])
                pmax = i; // local is greater than current maximum
        }

        // Crossed back over zero line - record peak's tau and continue
        // TODO - Parabolic interpolation - more accurate time location of peak
        Peaks[np] = pmax;
    }
    return np;
}
```

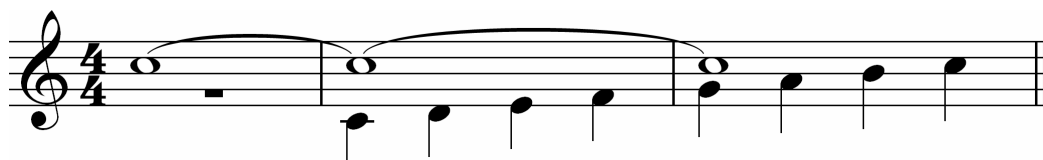
Appendix D – Musical Scores

D.1 Two-Part Harmonies

D.1.1 Middle C pedal note with C Major scale above, one octave, ascending



D.1.2 C₅ pedal note and C Major scale below, one octave, ascending



D.1.3 G₄ pedal note and G Harmonic Minor scale above, one octave, ascending



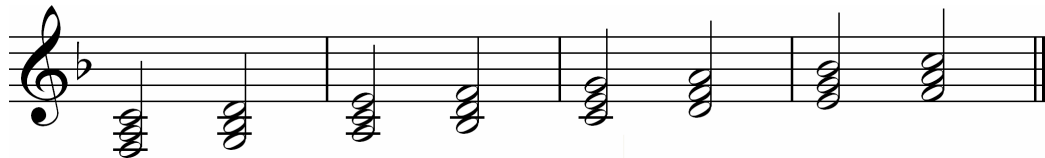
D.1.4 A₃ pedal note and chromatic (semitones) scale above, one octave, ascending



D.2 Three-Part Harmonies

D.2.1 Close Position Triads

F Major:

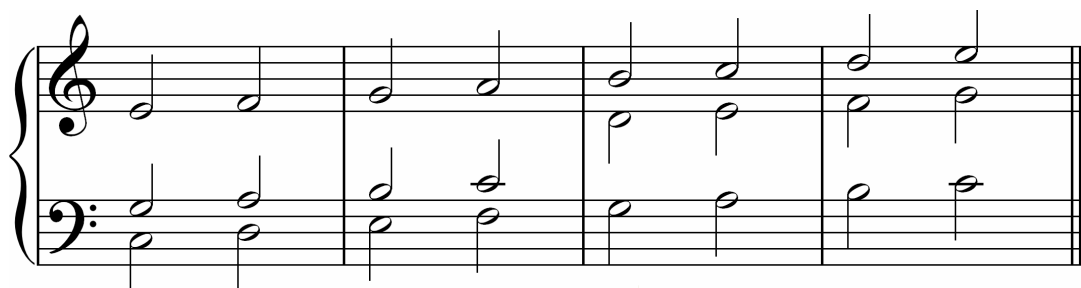


D Minor:

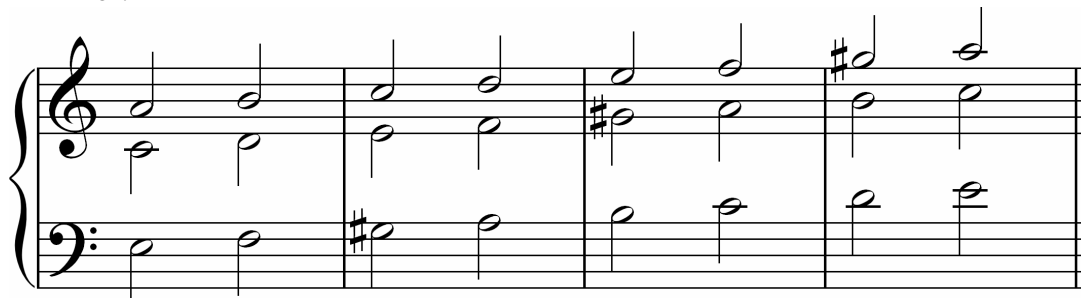


D.2.2 Wide Position Triads

C Major:



A Minor:



D.3 Four-Part Harmonies

D.3.1 Perfect Cadence

C Major:

Musical notation for a perfect cadence in C Major. The piece is in 4/4 time. The first measure shows a C major triad in the treble clef (C4, E4, G4) and a single C2 in the bass clef. The second measure shows a C major triad in the treble clef (C4, E4, G4) and a single C3 in the bass clef. The piece ends with a double bar line.

D.3.2 Two Chord Progressions

G Major:

Musical notation for a two-chord progression in G Major. The piece is in 4/4 time. The first measure shows a G major triad in the treble clef (G4, B4, D5) and a single G2 in the bass clef. The second measure shows a G major triad in the treble clef (G4, B4, D5) and a single G3 in the bass clef. The third measure shows a G major triad in the treble clef (G4, B4, D5) and a single G4 in the bass clef. The fourth measure shows a G major triad in the treble clef (G4, B4, D5) and a single G5 in the bass clef. The fifth measure shows a G major triad in the treble clef (G4, B4, D5) and a single G6 in the bass clef. The piece ends with a double bar line.

Musical notation for a two-chord progression in G Major. The piece is in 4/4 time. The first measure shows a G major triad in the treble clef (G4, B4, D5) and a single G2 in the bass clef. The second measure shows a G major triad in the treble clef (G4, B4, D5) and a single G3 in the bass clef. The third measure shows a G major triad in the treble clef (G4, B4, D5) and a single G4 in the bass clef. The fourth measure shows a G major triad in the treble clef (G4, B4, D5) and a single G5 in the bass clef. The piece ends with a double bar line.

A Minor:

Musical notation for a two-chord progression in A Minor. The piece is in 4/4 time. The first measure shows an A minor triad in the treble clef (A4, C5, E5) and a single A2 in the bass clef. The second measure shows an A minor triad in the treble clef (A4, C5, E5) and a single A3 in the bass clef. The third measure shows an A minor triad in the treble clef (A4, C5, E5) and a single A4 in the bass clef. The fourth measure shows an A minor triad in the treble clef (A4, C5, E5) and a single A5 in the bass clef. The piece ends with a double bar line.

Appendix E – Wave Processor 3.0

Although it has a large number of completed features, *Wave Processor* is still very much a work in progress*. It was written in Microsoft Visual C++ (Platform SDK) under Microsoft Visual Studio 2008 and has the working title *Mozart* – named after one of the most notorious music transcribers.

E.1 Installation

E.1.1 System Requirements

The following requirements are the minimum recommended for the computer system on which you want to install *Wave Processor*:

Operating System	Microsoft Windows XP. Does not yet work in Vista. An upgrade is planned for Windows 7.
Processor	Suitably fast, preferably Intel Core2 Duo or better.
RAM	At least 2GB, preferably more for finer analysis methods.
Hard disk free space	1GB – 10GB swap space, depending on the duration of music needing to be analysed and the depth of the analysis. 2MB for program files and additional space for storage of wave files, which are approx. 5MB per minute in size using the default wave creation settings.

Table E.1 – System requirements for running *Wave Processor*

E.1.2 Setup

Wave Processor may be installed from the project CD as follows:

- Insert the project CD and locate the installation folder:
Software\Wave Processor\Install\
- Launch `setup.exe`
- Click **Next** to accept the copyright agreement
- Choose a destination folder for the program and data files by clicking **Browse...** or click **Next** to accept the default location
- Click **Next** to start the installation and wait while the software installs
- Click **Close** to exit the setup program

* The latest version of the software may be downloaded from <http://mars.cs.ukzn.ac.za/~johnmcg/>

E.1.3 Manual Installation

Should the above setup program fail for any reason, *Wave Processor* may also be installed manually as follows:

- Create a destination folder on your computer in a location of your choice
- Insert the project CD and locate the folder:
Software\Wave Processor\Executable\
- Copy all files (*Mozart.exe* and *.dat) from this folder to your installation folder
- If desired, create a shortcut to *Mozart.exe*

E.2 Functionality

The following features have been implemented in the application:

- Open and view the waveform of a Windows 16-bit PCM wave file. Note that opening and manipulating stereo tracks is not complete and so it is disabled in this version
- Play back the opened wave (recording is disabled in this version)
- Create an audio signal with up to eight frequencies at different amplitudes (sinusoidal only)
- Perform a variety of mathematical transforms on a wave (see **Table E.2**)
- Perform a variety of pitch detection methods on a wave (see **Table E.3**)
- View the histogram, spectrogram and pitch graphs following a transform or pitch extraction
- Export a spectrogram image to a bitmap
- Options and preferences settings (system options are not adjustable in this version)

Transforms	Options
Discrete Fourier	Choice of spectrogram or histogram view
	Setting of upper frequency bound
	Windowed (STFT) or ordinary transform
	Phase Vocoder correction
	Choice of seven window functions
	Window width adjustment for STFT and Phase Vocoder
Discrete Wavelet	Choice between Haar, Daubechies 4 or Redundant Haar
	Detection threshold adjustment
	Choice of dyadic or linear spectrogram
Continuous Wavelet	Choice between Morlet, Paul, Derivative of Gaussian (Mexican Hat) or Shannon wavelets
	Adjustment of wavelet parameters (wavenumber or order)
	Adjustment of lower and upper frequency bounds of output spectrogram
	Adjustment of number of scales to use in the transform
	Choice of dyadic or linear spectrogram
	Sensitivity option
	Greyscale spectrogram option
Animated view of each wavelet	

Table E.2 – Transform options in *Wave Processor*

Method	Options
Melody / single pitch extraction (MPM)	Adjustment of window width, peek-picking threshold and clarity threshold
Melody / single pitch extraction (Phase Vocoder)	See Fourier Transform options – Phase Vocoder is assumed
Two / three part extraction (DWT / MPM)	Same as MPM options
Full / multiple pitch extraction (Phase Vocoder)	See Fourier Transform options – Phase Vocoder is assumed
Full / multiple pitch extraction (CWT)	Adjustment of window width (for post-processing) and pitch detection threshold
	Option to enable or disable Difference Tone Analysis
	See Continuous Wavelet Transform options for more settings

Table E.3 – Pitch detection options in *Wave Processor*

E.3 Known Issues

Since this version of *Wave Processor* is still in Beta phase, there are a number of issues which remain unresolved. These are listed below:

- Some features (but not those listed above) are not fully implemented, but may not have been greyed out on menus and dialog boxes. Unknown behaviour may result when attempting to use these features.
- The program remains untested on very long musical samples. The most file sections it is known to be able to handle for a large CWT is 16. Unknown behaviour may result if the hard disk on which swap files are made runs out of space, or if memory availability is low on the system on which it is running.
- The y-axes of dyadic graphs are being mislabelled due to an incorrect calculation, however the spectrograms themselves are drawn with the correct range and scale.
- The pitch meter is not yet complete, but in this feature a marker will be implemented which indicates in real-time the value of a pitch sung or whistled into a microphone, using MPM to determine the value.
- Other visible features not functioning are the record and wave navigation buttons (fast-forward and rewind). It is intended that the retractable control panel have features such as volume controls and VU meters, however for now it is just a blank extendable panel (which is fun to play with if you are very bored!)
- If all tools are pulled off the docking toolbar, and the pitch meter is re-docked, unfortunately it resizes to fill the width of the toolbar, and so the other tools will not fit back on again. Simply restart the application if this happens.
- It is possible to open some options dialog boxes twice. Since only one handle is used for each dialog box, the other copies spawned become orphaned and so cannot be closed. Again, the application must be restarted to get rid of them.
- Due to the idiosyncrasies of the Windows *waveOut* interface, the application may occasionally crash during wave playback, although this is a rare occurrence.

References and Sources

Where possible, for each of the references listed below, the location on the project CD of a soft copy of the source has been provided. For web sources, due to the dynamic nature of the World Wide Web, copies of some of the web pages (only at the time of writing) have been stored, as well as providing references to the original sites.

Literature

- [Abdallah02L] S. A. Abdallah, *Ph.D Thesis: Towards music perception by redundancy reduction and unsupervised learning in probabilistic models*, 2002, Department of Electronic Engineering, King's College London
Literature\Abdallah - PhD.pdf
- [Abramowitz65L] M. Abramowitz, I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables* (Chapter 22), 1st Ed., 1965, Dover
Literature\
Abramowitz & Stegun - Chapter 22*.png
- [ABRSM38L] ABRSM, *Rudiments and Theory of Music*, 1938, The Associated Board of the Royal Schools of Music, London
(soft copy unavailable)
- [Andreão07L] R. V. Andreao, J. Boudy, *Combining Wavelet Transform and Hidden Markov Models for ECG Segmentation*, 2007, EURASIP Journal on Advances in Signal Processing, vol. 2007, Article ID 56215, 8 pages
Literature\Andreão - Combining CWT and HMM.pdf
- [Arfken85L] G. B. Arfken, H. J. Weber, *Mathematical Methods for Physicists* (Hermite Functions), 3rd Ed., 1985, Academic Press
(soft copy unavailable)
- [Bastiaans80L] M. J. Bastiaans, *Gabor's Expansion of a Signal into Gaussian Elementary Signals*, April 1980, Proceedings of IEEE, vol. 68, pp. 538-539
Literature\Bastiaans - Gabor's Expansion.pdf
- [Bello03L] J. P. Bello, *PhD Thesis: Towards the Automated Analysis of Simple Polyphonic Music: A Knowledge-based Approach*, 2003, Department of Electronic Engineering, Queen Mary, University of London
Literature\Bello - PhD.pdf
- [Bello04L] P. Brossier, J. P. Bello, M. D. Plumbley, *Fast Labelling of Notes in Music Signals*, October 2004, Proceedings of 5th International Conference on Music Information Retrieval (ISMIR 2004), Barcelona, pp. 331-336
Literature\Brossier - Fast Labelling of Notes in Music Signals.pdf

- [Bogert63L] B. P. Bogert, M. J. R. Healy, J. W. Tukey, *The Quefrequency Alanysis of Time Series for Echoes: Cepstrum, Pseudo Autocovariance, Cross Cepstrum and Saphe Cracking*, 1963, Proceedings of Symposium on Time Series Analysis (M. Rosenblatt, ed.), Chapter 15, pp. 209-243, John Wiley, New York, NY
(partially available via Google Books preview)
- [Brenner92L] N. M. Brenner, Numerical Recipes Software, excerpt from *Numerical Recipes in C: The Art of Scientific Computing*, 1988 – 1992, Cambridge University Press
Literature\Numerical Recipes in C - 12.2.pdf
- [Brossier04L] P. Brossier, J. P. Bello, M. D. Plumbley, *Real-time Temporal Segmentation of Note Objects in Music Signals*, November 2004, Proceedings of International Computer Music Conference (ICMC 2004), Miami, FL, pp. 458-461
Literature\Brossier - Real-time Temporal Segmentation of Note Objects.pdf
- [Brown04L] J. I. Brown, *Mathematics, Physics and A Hard Day's Night*, 2004, Dalhousie University (public domain)
Literature\Brown - A Hard Day's Night.pdf
- [Cont07L] A. Cont, S. Dubnov, D. Wessel, *Real-time Multiple-pitch and Multiple-instrument Recognition for Music Signals Using Sparse Non Negative Constraints*, September 2007, Proceedings of the 10th International Conference on Digital Audio Effects (DAFx-07), Bordeaux
Literature\Cont - Real-time Multiple-pitch and Multiple-instrument Recognition.pdf
- [Cooley65L] J. W. Cooley, J. W. Tukey, *An Algorithm for the Machine Calculation of Complex Fourier Series*, 1965, Mathematics of Computation, vol. 19, pp. 297-301
Literature\Cooley-Tukey - FFT.pdf
- [Cowling04L] M. Cowling, *PhD Thesis: Non-Speech Environmental Sound Classification System for Autonomous Surveillance*, 2004, Griffith University, Gold Coast Campus
Literature\Cowling - PhD.pdf
- [Cunningham04L] S. Cunningham, *Suitability of MusicXML as a Format for Computer Music Notation and Interchange*, 2004, Proceedings of the IADIS International Conference Applied Computing, Lisbon
Literature\Cunningham - Suitability of MusicXML.pdf
- [Danielson42L] G. C. Danielson, C. Lanczos, *Some Improvements in Practical Fourier Analysis and their Application to X-ray Scattering from Liquids*, 1942, J. Franklin Inst. 233, pp. 365-380 and 435-452
(soft copy unavailable)

- [Daubechies92L] I. Daubechies, *Ten Lectures on Wavelets*, January 1992, SIAM (partially available via Google Books preview)
- [DeMoortel06L] I. De Moortel, *Propagating Magnetohydrodynamics Waves in Coronal Loops*, February 2006, Philosophical Transactions of the Royal Society, A15, vol. 364, no. 1839, pp. 461-472
 \Literature\DeMoortel - Propagating Magnetohydrodynamics Waves.pdf
- [Ewer10L] J. P. G. Ewer, private communication, 2010
- [Flanagan66L] J. L. Flanagan, R. M. Golden, *Phase Vocoder*, November 1966, Bell System Technical Journal, vol. 45, pp. 1493-1509 (soft copy unavailable)
- [Fourier22L] J. B. J. Fourier, *Théorie Analytique de la Chaleur*, 1822, Firmin Didot – Père et Fils (now public domain, scanned by Google)
 \Literature\Fourier - Théorie Analytique de la Chaleur.pdf
- [Frigo03L] M. Frigo, S. Johnson, *FFTW Manual*, 2003, Massachusetts Institute of Technology, Cambridge, MA
 \Literature\Frigo & Johnson - fftw3.pdf
- [Gabor46L] D. Gabor, *Theory of Communication*, November 1946, Journal of IEEE, London, vol. 93, Part III, pp. 429-457
 Literature\Gabor - Theory of Communication.pdf
- [Goupillaud 84L] P. Goupillaud, A. Grossmann, J. Morlet, *Cycle-Octave and Related Transforms in Seismic Signal Analysis*, 1984/85, Geoprospection, vol. 23, pp. 85-102 (soft copy unavailable)
- [Graps95L] A. Graps, *An Introduction to Wavelets*, Summer 1995, IEEE Computational Science and Engineering, vol. 2, no. 2, pp. 50-61
 Literature\Graps - Introduction to Wavelets.pdf
- [Grossman84L] A. Grossman, J. Morlet, *Decomposition of Hardy Functions into Square Integrable Wavelets of Constant Shape*, July 1984, SIAM Journal on Mathematical Analysis, vol. 15, no. 4, pp. 723-736
 Literature\Grossman & Morlet - Wavelets.pdf
- [Grove00L] G. Grove, *A Dictionary of Music and Musicians*, 2nd Ed., 1900, Macmillan, London
 Literature\Grove - Dictionary (107).jpg (page 107 only)

- [Haar10L] A. Haar, *Zur Theorie der Orthogonalen Funktionensysteme*, 1910, *Mathematische Annalen*, vol. 69, pp331-371, translated by G. Zimmermann, pub. in C. Heil and D. F. Walnut, *Fundamental Papers in Wavelet Theory*, pp. 155-188, Princeton 2006, Princeton University Press
Literature\Haar - Orthogonal Function Systems.pdf
- [Howard06L] D. M. Howard, J. A. S. Angus, *Acoustics and Psychoacoustics*, 1st Ed., 2006, Focal Press, Elsevier, Oxford
Literature\Howard - Acoustics (242).jpg
(page 242 only)
- [Kronland87L] R. Kronland-Martinet, J. Morlet, A. Grossman, *Analysis of Sound Patterns Through Wavelet Transforms*, 1987, *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 1, no. 2, pp. 97-126
Literature\Kronland - Analysis of Sound Patterns.pdf
- [Li07L] Y. Li, D. Wang, *Pitch Detection in Polyphonic Music using Instrument Tone Models*, April 2007, *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, Honolulu, HI, pp. II. 481-484
Literature\Li-Wang - Pitch Detection.pdf
- [Li08L] Y. Li, D. Wang, *Musical Sound Separation using Pitch-based Labeling and Binary Time-frequency Masking*, March 2008, *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, Las Vegas, NV, pp. 173-176
Literature\
Li-Wang - MusicalSoundSeparation08.pdf
- [Li09L] Y. Li, D. Wang, *Musical Sound Separation based on Binary Time-frequency Masking*, 2009, *EURASIP Journal on Audio, Speech and Music Processing*, vol. 2009, Article ID 130567, 10 pages
Literature\
Li-Wang - MusicalSoundSeparation09.pdf
- [Mackenzie01L] D. Mackenzie, ast. I. Daubechies, D. Kleppner, S. Mallat, Y. Meyer, M. B. Ruskai, G. Weiss, *Wavelets: Seeing the Forest and the Trees*, 2001, National Academy of Sciences, Washington DC
Literature\Mackenzie - Wavelets.pdf
- [MacQueen67L] J. B. MacQueen, *Some Methods for Classification and Analysis of Multivariate Observations*, 1967, *Proceedings of 5th Berkeley Symposium On Mathematical Statistics and Probability*, Berkeley, University of California Press, vol. 1, pp. 281-297
Literature\MacQueen - k-means.pdf

- [Masri96L] P. Masri, *PhD Thesis: Computer Modeling of Sound for Transformation and Synthesis of Musical Signals*, 1996, University of Bristol, UK
(soft copy unavailable)
- [McLeod02L] P. McLeod, G. Wyvill, *Visualization of Musical Pitch*, July 2003, Proceedings of Computer Graphics International, Tokyo, Japan, pp. 300-303
Literature\Visualization of Musical Pitch.pdf
- [McLeod05L] P. McLeod, G. Wyvill, *A Smarter Way to Find Pitch*, September 2005, Proceedings of International Computer Music Conference, Barcelona, Spain, pp. 138-141
Literature\A Smarter Way to Find Pitch.pdf
- [Morlet82L] J. Morlet, G. Arens, E. Fourgeau, *Wave Propagation and Sampling Theory Part I: Complex Signal and Scattering in Multilayer Media, Part II: Sampling Theory and Complex Waves*, 1982, Geophysics, vol. 47, no. 2, pp. 203-236
(soft copy unavailable)
- [Murtagh05L] O. Renaud, J-L. Starck, F. Murtagh, *Wavelet-Based Combined Signal Filtering and Prediction*, December 2005, IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics, vol. 35, no. 6, pp. 1241-1251
Literature\Murtagh - Signal Filtering and Prediction.pdf
- [Nyquist28L] H. Nyquist, *Certain Topics in Telegraph Transmission Theory*, February 1928, Transactions of the AIEE, New York, pp. 617-644
Literature\Nyquist - Certain Topics in Telegraph Transmission Theory.pdf
- [Raphael06L] L. J. Raphael, G. J. Borden, K. S. Harris, *Speech Science Primer: Physiology, Acoustics, and Perception of Speech*, 5th Ed., 2006, Lippincott Williams & Wilkins
(soft copy unavailable)
- [Ricker40L] N. H. Ricker, *The Form and Nature of Seismic Waves and the Structure of Seismograms*, October 1940, Geophysics, vol. 5, no. 4, pp. 348-366
(soft copy unavailable)
- [Scholes65L] P. A. Scholes, *The Oxford Companion to Music*, 9th Ed., 1965, Oxford University Press
(soft copy unavailable)
- [Shannon49L] C. E. Shannon, *Communication in the Presence of Noise*, January 1949, Proceedings of the Institute of Radio Engineers, vol. 37, no. 1, pp. 10-21
Literature\Shannon - Communication in the Presence of Noise.pdf

- [Shure09L] Shure, *Microphone Techniques for Recording*, 2009, A Shure Educational Publication, Shure Incorporated
Literature\Shure - Microphone Techniques.pdf
- [Sibelius09L] D. Spreadbury, B. Finn, J. Finn, *Sibelius 6 Reference*, 6th Ed., 2009, Avid Technology, Inc.
Literature\Sibelius 6 Reference.pdf
- [Stevens65L] S. S. Stevens, F. Warshofsky, *Life Science Library – Sound and Hearing*, 1965, Time-Life Books, New York, pp. 102-103
(soft copy unavailable)
- [Strang89L] G. Strang, *Wavelets and Dilation Equations: A Brief Introduction*, December 1989, SIAM Review, vol. 31, no. 4, pp. 614-627
Literature\Strang - Wavelets and Dilation Equations.pdf
- [Strang94L] G. Strang, *Wavelets*, April 1994, Appendix 1, American Scientist 82, 250-255
Literature\Strang - Wavelets.pdf
- [Torrence98L] C. Torrence, G. P. Compo, *A Practical Guide to Wavelet Analysis*, 1998, Bulletin of the American Meteorological Society, vol. 79, no. 1, pp. 61-78
Literature\Torrence & Compo - A Practical Guide to Wavelet Analysis.pdf
- [Vrhel97L] M. J. Vrhel, C. Lee, M. Unser, *Rapid Computation of the Continuous Wavelet Transform by Oblique Projections*, April 1997, IEEE Transactions on Signal Processing, vol. 45, no. 4, pp. 891-900
Literature\Vrhel - Rapid Computation of the CWT.pdf
- [Zhan06L] Y. Zhan, D. Halliday, P. Jiang, X. Liu, J. Feng, *Detecting Time-Dependent Coherence Between Non-Stationary Electrophysiological Signals*, 2006, Elsevier Journal of Neuroscience Methods, no. 156, pp. 322-332
Literature\Zhan - Detecting Time-Dependent Coherence.pdf

World Wide Web

- [Ackers00W] F. Ackers, presentation of original Fast Fourier Transform code by Don Cross, 2000, (no longer available online)
Web\Introduction FFTs.doc
- [AKG09W] AKG D 112 microphone specifications, AKG,
http://www.ake.com/site/products/powerslave.id,261,pid,261,nodeid2_language,EN.html
Web\D 112.htm

- [Bourke93W] P. Bourke, *DFT (Discrete Fourier Transform) FFT (Fast Fourier Transform)*, 1993,
<http://local.wasp.uwa.edu.au/~pbourke/other/dft/>
 Web\Discrete Fourier Transform.htm
- [Celemony09W] Celemony, *Melodyne (Direct Note Access)*, 2009,
<http://www.celemony.com/>
 Web\DNA - Demo.wmv (demonstration video)
 Web\DNA - Interview.flv (interview video)
- [Crane97W] R. Crane, excerpt from *A Simplified Approach to Image Processing*, 1997, Prentice Hall PTR,
http://zone.ni.com/devzone/conceptd.nsf/webmain/A61876074AE0B9918625684600522CF4?opendocument&node=1301_US
 Web\FFT Tutorial.htm
- [Google09W] Google, *Google Calculator*, 2009,
<http://www.google.com/search?q=once+in+a+blue+moon>
 Web\Once in a blue moon.htm
- [Graps04W] A. Graps, *An Introduction to Wavelets*, 1995 – 2004, Institute of Electrical and Electronics Engineers, Inc.,
<http://www.amara.com/IEEEwave/IEEEwavelet.html>
 Web\Introduction to Wavelets\
- [Klapetek02W] P. Klapetek, *Wavelet Transform, Discrete Transform and Continuous Wavelet Transform*, 2002,
<http://klapetek.cz/index.html>
 Web\Petr Klapetek\
- [Misiti96W] M. Misiti, Y. Misiti, G. Oppenheim, J. M. Poggi, *Wavelet Toolbox*, 1996, The Maths Works Inc.,
<http://www.mathworks.com/access/helpdesk/help/toolbox/wavelet>
 Web\Wavelet Toolbox\
- [MusicIcon09W] Music Icon, *Music Icon*, 2009, Music Icon, Inc.,
<http://www.musiciconinc.com/>
 Web\MusicIcon.htm
- [Recordare09W] Recordare, *MusicXML*, 2009, Recordare LLC,
<http://www.recordare.com/xml.html>
 Web\MusicXML Definition.htm
- [Neuratron09W] Neuratron, *AudioScore Ultimate 6*, 2009,
<http://www.neuratron.com/>
- [OED05W] *Oxford English Dictionary*, Oxford University Press, 2005,
<http://www.oed.com>
 Web\OED pitch.htm

- [Olver05W] P. J. Olver, *Applied Mathematics Lecture Notes, Chapter 13 – Fourier Analysis*, 2005, School of Mathematics, University of Minnesota, MN, http://www.math.umn.edu/~olver/am_fa.pdf
Web\Olver - Fourier Analysis.pdf
- [Osgood09W] B. G. Osgood, *Aliasing Demonstration With Music*, 2009, The Fourier Transform and its Applications – Video Lecture 19 of 30, Stanford Engineering Everywhere, <http://academicearth.org/lectures/aliasing-demonstration-with-music>
(video not included on project CD due to its size)
- [Paul09W] T. Paul, home page at Département de Mathématiques et Applications (DMA), <http://www.dma.ens.fr/~paul/>
Web\Thierry Paul.htm
- [Polikar03W] R. Polikar, The Wavelet Tutorial, 1993, Rowan University, <http://users.rowan.edu/~polikar/WAVELETS/WTtutorial.html>
Web\Robi Polikar - Wavelets Tutorial\
- [QSound96W] QSoundLabs, *Virtual Barber Shop – Binaural Audio Demo*, 1996, http://www.qsound.com/demos/virtualbarbershop_long.htm
- [RØDE08W] RØDE NT1-A condenser microphone specifications, RØDE Microphones, <http://www.rodemic.com/microphone.php?product=NT1-A>
Web\RØDE Microphones - NT1-A.htm
- [RØDE09W] RØDE NT3 condenser microphone specifications, RØDE Microphones, <http://www.rodemic.com/microphone.php?product=NT3>
Web\RØDE Microphones - NT3.htm
- [SHURE09W] SHURE Beta 57A instrument microphone specifications, SHURE Pro Audio, http://shure.com/ProAudio/Products/WiredMicrophones/us_pro_Beta57A_content
Web\Shure - Microphones - Beta 57A Instrument Microphone.htm
- [Sibelius09W] Sibelius, *Sibelius 6*, 2009, Avid Technology, Inc., <http://www.sibelius.com>
- [Wang07W] J. Wang, T-T. Wong, P-A. Heng, C-S. Leung, *Discrete Wavelet Transform on GPU*, 2007, Department of Computer Science and Engineering, The Chinese University of Hong Kong, <http://www.cse.cuhk.edu.hk/~ttwong/demo/dwtgpu/dwtgpu.html>
Web\Discrete Wavelet Transform on GPU.htm
- [Wolfram09W] E. W. Weisstein, *Wolfram MathWorld*, <http://mathworld.wolfram.com>

Music

- [Boswell09M] *A Glorious Dawn – Cosmos Remixed*, music by John Boswell, lyrics by Carl Sagan & Stephen Hawking, 2009, Colorpulse Music, USA
Sound\Carl Sagan - A Glorious Dawn.mpg
- [Heap05M] *Hide and Seek, Speak For Yourself*, music and lyrics by Imogen Heap, 2005, White Rabbit / SonyBMG, UK
Sound\Imogen Heap - Hide And Seek.mp3
- [Sontonga97M] *Nkosi Sikeleli Africa*, music and Xhosa lyrics by Enoch Sontonga, 1897, South Africa
Sounds\NSA\ (various arrangements)
- [Tallis01M] *Misere Mei, Deus*, music by Gregorio Allegri, performed by The Tallis Scholars, directed by Peter Phillips, April 2001, Gimell Records, UK
Sound\Allegri - Misere.mp3

Software

- [FFTW03S] M. Frigo, S. Johnson, *fftw-3.0.1*, 2001 – 2003, Massachusetts Institute of Technology, Cambridge, MA
- [McGuinness06S] J. C. McGuinness, *Image Processor 3.0*, 2006, Written as part of requirement for Image Processing module of BSc. Hons. degree, School of Computer Science, University of KwaZulu Natal, KZN
- [Syntrillium02S] Syntrillium Software, *Cool Edit Pro version 2.0*, 1992 – 2002, Syntrillium Software Corporation, Phoenix AZ
- [Tartini07S] P. McLeod, *Tartini 1.2*, 2002 – 2007, Written as part of PhD work, Department of Computer Science, University of Otago, Otago