AUTOMATIC MUSIC TRANSCRIPTION

Dylan Quenneville

Adviser: Professor Daniel Scharstein

A Thesis

Presented to the Faculty of the Computer Science Department

of Middlebury College

in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Arts

May 2018

ABSTRACT

Music transcription is the process of creating a written score of music from an audio recording. Musicians and musicologists use transcription to better understand music that may not have a written form, from improvised jazz solos to traditional folk music. Automatic music transcription introduces signal-processing algorithms to extract pitch and rhythm information from recordings. This speeds up and automates the process of music transcription, which requires musical training and is very time consuming even for experts. This thesis explores the still unsolved problem of automatic music transcription through an in-depth analysis of the problem itself and an overview of different techniques to solve the hardest subtask of music transcription, multiple pitch estimation. It concludes with a close study of a typical multiple pitch estimation algorithm and highlights the challenges that remain unsolved.

ACKNOWLEDGEMENTS

I would like to thank my thesis adviser, Daniel Scharstein, for showing me the world of academic research and helping me through this thesis, my major adviser Pete for always being a good voice of reason and helping me define my academic path, and the rest of the Middlebury Computer Science Department for teaching me so many diverse aspects of computer science. This thesis also would not have happened without the love and support of my family. I also want to thank Dick Forman for teaching me so much of what I know about music, especially how important music transcription is. Finally, thank you to all my friends who have done so much to keep me (mostly) sane throughout this process.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Music's universality and cultural significance is not easily exaggerated. Every known civilization has some form of music. Whether it be a well-structured art form like Western classical music, traditional folk songs orally passed down for generations, or contemporary pop music, the sharing of music seems to provide endless fascination and joy for humans.

Music transcription is, broadly defined, the task of converting music from sound into a written, abstract notation. This can be thought of as the inverse operation of music performance, which often involves a performer reading music notation and producing soundwaves with an instrument or their voice. While playing an instrument well is no easy feat, music transcription can prove more challenging and far more time consuming, even for the most skilled musicians. The goal of this thesis is to introduce the field of *automatic music transcription*, the use of computers to automatically convert digital recordings of music into practical music notation. This field of computer science research has been developing over the past two decades and has numerous unsolved problems. Still, every year shows new research with improved algorithms for the various subtasks of music transcription.

By way of introduction and with hope of building a concrete foundation of terminology, this thesis begins in Chapter 2 with a description of music, both in terms of its fundamental theoretical components and in terms of how humans actually hear and process music. Next, in Chapter 3, the problem of music transcription is formally defined, both in the more common case of transcription by hand, and in the case discussed in this thesis, automatic music transcription. After this foundation is laid, Chapter 4 describes the derivation and underlying concepts of the Fourier transform, which allows us to interpret audio recordings not in terms of their amplitude over time, but as a sum of

different frequencies, much in the way human ears work on a biological level. This provides an important tool for Chapters 5 and 6, which get to the heart of automatic music transcription: using algorithms to extract musical pitch from digital audio recordings. Specifically, Chapter 5 covers the history of pitch estimation techniques, both applied to speech analysis and music analysis, while Chapter 6 describes multi-pitch estimation, a more difficult problem.

For Chapters 5 and 6, I implemented several pitch-estimation algorithms to better understand and illustrate the steps and results of different strategies. I implemented these algorithms in Python using NumPy [36] and SciPy [43] for data manipulation, SciPy's `wavfile` module for reading audio files, MIDIUtil [53] for transcription output, and Matplotlib's `pyplot` module [20] for creating figures. All spectrograms and frequency plots in this thesis were generated with Python programs I implemented, except where cited otherwise. Relevant original code is included in Appendix A.

At the writing of this thesis, multi-pitch estimation of music recordings is an unsolved problem. The best algorithms only achieve about 70% accuracy [13]. With this in mind, this thesis seeks to provide a context for and introduction to the problem, but by no means a full picture of this still-active area of research.

CHAPTER 2

HOW MUSIC WORKS

Music transcription is most often performed by someone with musical training simply by repeatedly listening to a piece of music, writing down what they hear. Before treating music transcription as a signal processing problem, it is important to understand the basic structures and concepts in Western music theory and how transcription is usually performed.

Section 2.1 lays the foundations for the basic components of music, namely pitch and rhythm, generally in the language of Western music. It also introduces loudness and the relationship of instrumentation and timbre. While Section 2.1 is primarily devoted to explaining music creation from concept to sound, Section 2.2 deals with how humans process music as sound, from the ear to the processing that happens within the brain.

2.1 How Music is Made

At its most basic level, music is a series of pitched tones played in a particular rhythmic sequence. This structured sequence of pitched sound is generally played by something that generates vibrations of certain frequencies, whether it be a musical instrument that generates physical vibrations, human vocal folds, or an electronic instrument that uses either electrical components or digital electronics to create waveforms. Russel Barton defines music as "sound, structure and artistic intent," and sound as "pitch, duration, loudness, timbre, texture and spatial location" [7]. From this definition, four components are particularly relevant for automatic music transcription. These are pitch, duration, loudness, and timbre. For the sake of this thesis, we will expand duration slightly to include structure and speak of it more broadly as *rhythm*. These four components were chosen because ultimately transcription of a piece need only accurately record pitch,

rhythm, loudness, and timbre, though certainly artistic vision and large-scale structure of the music can be determined or interpreted.

Modern music is most commonly played by a performer reading sheet music and translating the music written on the page into positions on their instrument or vocal pitch, then producing the sound. At its most basic, sheet music conveys the above-mentioned four components, that is the actual sequence of pitches along with timing and loudness of each to be played by the performer. Meta-information about the composition is also likely included, such as tempo, the speed at which notes are played; time signature, which indicates the underlying rhythmic structure of the piece; and key signature, which suggests the harmonic basis for the piece, among other features. Missing, then, is timbre, which is prescribed only indirectly by specifying the instrument. Timbre is also unlike the other three—pitch, rhythm, and loudness—in that an automatic transcription system does not need to specify timbral qualities of the notes, though it is relevant. The timbre of notes in a recording is very important information for automatic music transcription, and instrument identification—made possibly only through timbral analysis—is often considered a subset of automatic music transcription.

2.1.1 Pitch

The pitch of a note corresponds to its fundamental frequency. All music starts with periodic vibrations, that is a physical or electronic object oscillating at regular intervals to create vibrations in the air, which must ultimately reach human ears. For string and percussion instruments, these vibrations are generated by a physical object's resonant frequency, determined by its tension and size. For wind instruments, length of tubing determines the resonant frequency that is activated by blowing air or vibrating one's lips into the instrument. Human vocal folds act in a similar way to reeds and strings, vibrating back and forth at a frequency determined by tension and size. For electronic

music, these vibrations are either generated as alternating voltage in a circuit or as digital waves, but for these waves to become audible, they must be connected to some sort of loudspeaker that will produce the frequencies generated by the electronic instrument.

What makes music more than just a collection of overlapping periodic vibrations, then, is the highly structured set of frequencies used in music. Pitches are quantized according to *half-steps*, the smallest increment of pitch in Western music. Each musical note, specified by a letter from A to G and an accidental, either 'sharp' or 'flat', is defined in how many half-steps from A it is. This pattern then repeats, with A coming after G. The largest repeating structure of pitch is called an octave, defined as 12 half-steps. Notes are often specified by their letter note and octave. An octave is comprised of 7 natural notes and 5 accidental notes, which correspond to the white and black keys on a piano, respectively.

There is a more interesting mathematical basis for the octave, however. A key feature of musical pitch is that, while directly related to frequency, it is not linearly related. Instead, pitch follows a logarithmic scale. By definition, a note one octave above another note will have double the frequency. The modern musical pitches are defined according to A4 = 440 Hz; that is, playing an A4 on any instrument should cause vibrations that repeat 440 times per second. A5, then, is 880 Hz, while A3 is 220 Hz. Figure 2.1 shows two and a half octaves of notes, the corresponding piano keys, and their frequencies plotted on a linear scale.

This logarithmic relationship of pitches is deeply connected to harmony. Harmony refers to the relationships of simultaneously sounded notes. This is accomplished either by a polyphonic instrument—such as a piano, which has dedicated strings for each pitch—or by multiple performers playing together. Because of the relationships of different pitches' frequencies, different combinations of pitches have different qualities. This placement of frequencies along a logarithmic curve is important for understanding

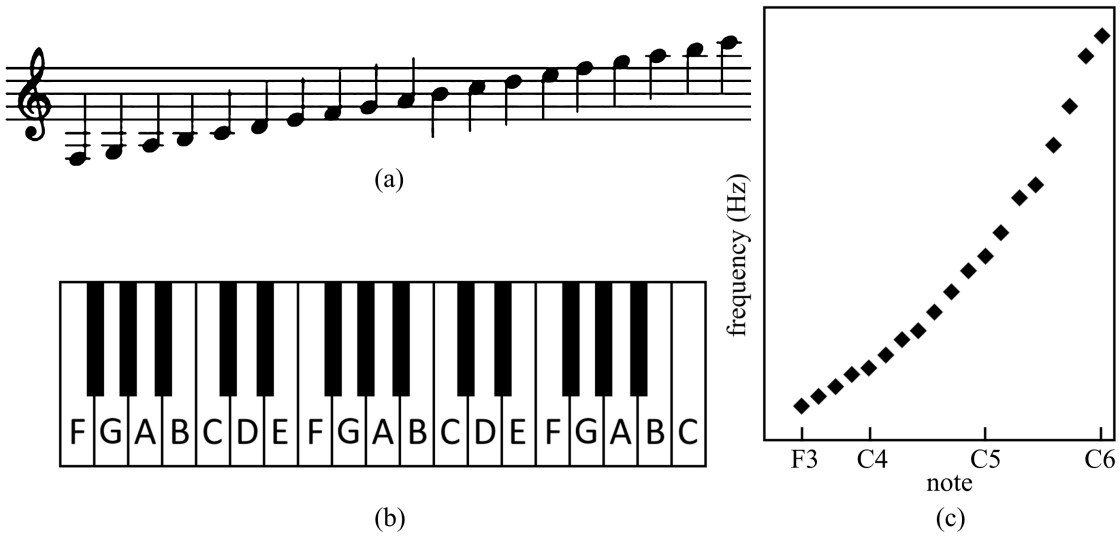


Figure 2.1: Two and a half octaves, F3-C6, represented as (a) notes, (b) piano keys, and (c) frequencies.

frequency hypotheses generated and utilized for automatic music transcription.

Accounting for variations in pitch is also important for automatic music transcription. Most instruments require some amount of minor adjustment to play perfectly in tune with A4 = 440 Hz, but there is no guarantee that every pitch produced by a performer and their instrument will exactly match the expected pitches of each note. Furthermore, there are also cases of intentional variation of pitch. Vibrato, for example, refers to a rhythmic variation of pitch up and down to create a certain effect. Some instruments can also achieve pitches between the 12 pitches of the octave by applying extra tension to strings or with careful use of air. Intermediate pitches and gradual changes in pitch can pose challenges for algorithmic understanding of music that assume a fairly rigid and discrete definition of musical pitches.

2.1.2 Rhythm

Rhythm is, in short, the underlying temporal structure of music. In general, music is comprised of notes played in certain intervals of time. The main unit of rhythm is the

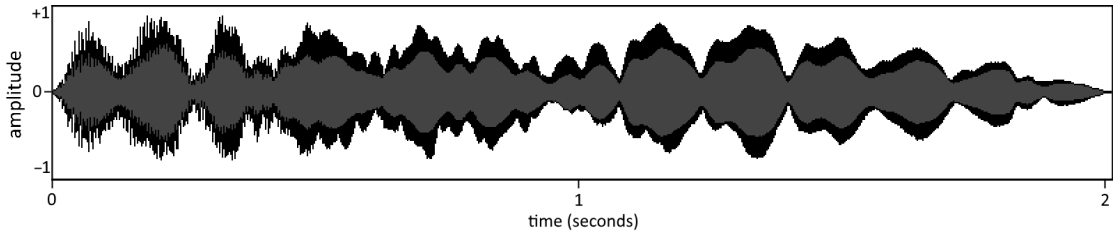


Figure 2.2: A single, sustained note played on a flute shows periodic variations in loudness known as tremolo. This change in loudness can easily be mistaken for different note activations by naive algorithms.

beat, and the speed at which music is performed is usually counted in terms of beats per minute (bpm). Beats are usually grouped into measures, which can be anywhere from two to twelve beats long, though three or four beats per measure is most common. Notes, however, can be played at any subdivision of a beat as well. Beats are split in half, then into quarters, then into eighths, and so on, giving room for rhythmic precision and rapid sequences of notes.

2.1.3 Loudness

In music, the term *dynamics* refers to the relative loudness of notes. Dynamics are generally specified in terms of two volume levels: *piano* and *forte*, which mean quiet and loud, respectively. Between piano and forte are mezzo-piano and mezzo-forte, meaning moderately soft and moderately loud. There are also extremes, pianissimo and fortissimo. Besides these six loudness categories, there can also be gradual changes in volume called crescendos and decrescendos. Particularly troublesome for algorithmic understanding of music is *tremolo*, rhythmic change in loudness, common when playing flute, vibraphone, among other instruments (see Figure 2.2). If there is a significant depth to the tremolo, automatic transcription systems may mistake peaks of the tremolo wave as new note activations, falsely increasing complexity of the transcription.

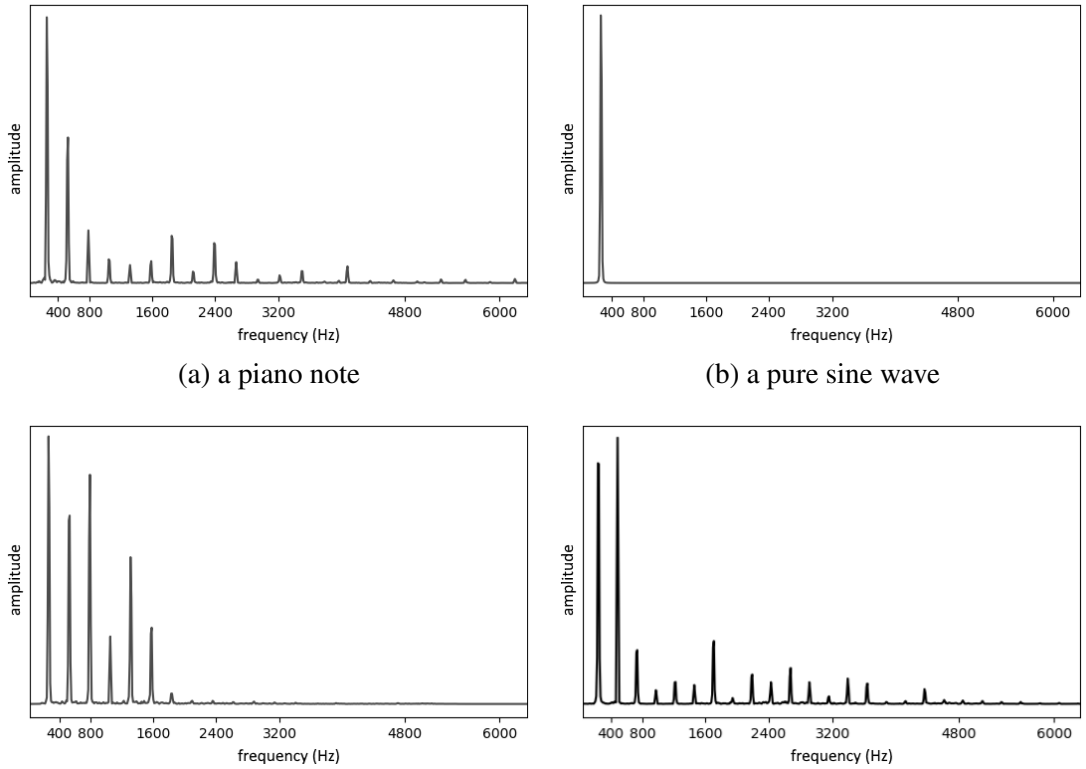
When dealing with recordings of music, loudness becomes a factor of how the music

was recorded and processed as well. Poor microphone placement may mean certain instruments have higher apparent volume than others. Microphone and qualities of the room may also affect frequency response of the recording, meaning lower or higher pitches are not recorded at the same volume. After recording, processing can either create further disparities in loudness through equalization and stylistic effects, or can even out loudness through dynamic compression. In contemporary recorded music, it is common to compress the dynamics of commercially released music to the point of relatively constant volume throughout the recording. These variations in volume, of course, are distinct from dynamics in that they would not be written down in sheet music, and thus should not be captured by transcription.

2.1.4 Timbre

Timbre refers to the harmonic content of a single note and can also be referred to as the 'color' of a note. Timbre accounts for the difference between a given pitch played on a trumpet and that same pitch played on a flute, violin, or synthesizer. Unlike pitch, rhythm, and loudness, timbre is not directly indicated on written music. Instead, timbre is controlled by specifying the instrument for which a piece is written. Timbre, though most profoundly affected by instrument choice, can also be controlled by the performer's use of air, mutes, plucking style, or the leftmost pedal on a piano, which softens the tone. Timbre is usually discussed in terms of 'softness' and 'richness'; in a softer timbre, higher harmonics will be less present, while a richer tone will have a heavier layering of harmonics.

Anyone who has heard different musical instruments will have an intuitive sense of how their tonal qualities (i.e., timbres) differ, but what then is the concrete basis for timbre? Timbre is closely related to pitch, though distinct. In most cases, timbre results from the same principles of physics that create pitched vibrations. When a string is



(a) a piano note

(b) a pure sine wave

(c) a trumpet note

(d) a violin note

Figure 2.3: The harmonic content of different instruments playing the note C4 = 261.62 Hz ('middle C').

plucked or a wind instrument is sounded, the instrument resonates at a certain frequency determined chiefly by the length of the instrument. The oscillation of air pressure or a tensioned string will tend to vibrate at a frequency with a certain wavelength or period—the time interval between repetition of the wave. But if an instrument is resonant at one wavelength, it will also be resonant at half that wavelength, one-third that wavelength, one-quarter, and so on. This is the *harmonic sequence* of a pitch. As wavelength divides, frequency multiplies, and pitch increases. If the fundamental frequency is A4 = 440 Hz, then most instruments will produce lower volumes of 880 Hz (A5), 1320 Hz (E6), 1760 Hz (A6), ... Because of this series of increasing pitches that all resonate with the fundamental frequency's wavelength, instruments produce many harmonic frequencies at once [44]. Figure 2.3 shows the harmonic makeup of several different instruments.

The layering of harmonics is integral to harmony between multiple notes. If two notes are played together, the overlapping of their various harmonics is largely responsible for apparent harmonic consonance. This also holds implications for algorithmic pitch estimation, particularly in the presence of multiple simultaneous notes. This is one of the most challenging tasks of automatic music transcription: to interpret a harmonically rich signal and determine the single fundamental frequency that was the source of all of the harmonic frequencies.

Another interesting feature of timbre is how it changes over time. With a bowed instrument like a violin or a wind instrument, the pitch, loudness, and timbre can all be kept relatively constant over a long period of time, but if one plays a note on a piano, it gradually quiets, and also becomes softer in tone. Higher harmonics will fade out more quickly than lower harmonics, meaning the relative volumes of harmonics change, not just total volume of the note. Often, wind players will intentionally soften their timbre as they hold a note as well. This change of timbre over time will also present challenges for pitch detection.

2.2 How Humans Process Music — Ear and Brain

The previous section describes how instruments create music, but how do humans actually process music as they hear it? First, music must travel as sound waves through air. When an acoustic instrument is played, it generates pressure waves in the air that radiate outward at the speed of sound. For electronic instruments or recordings of music, there is an extra step where alternating voltage is sent to a loudspeaker that vibrates according to the voltage applied, generating similar waves of air pressure. When these waves arrive at the human ear, they push a small bone called the stapes back and forth, which passes the pressure variations into the inner ear, or cochlea.

The cochlea is filled with a fluid that varies in pressure with outside sound waves. In-

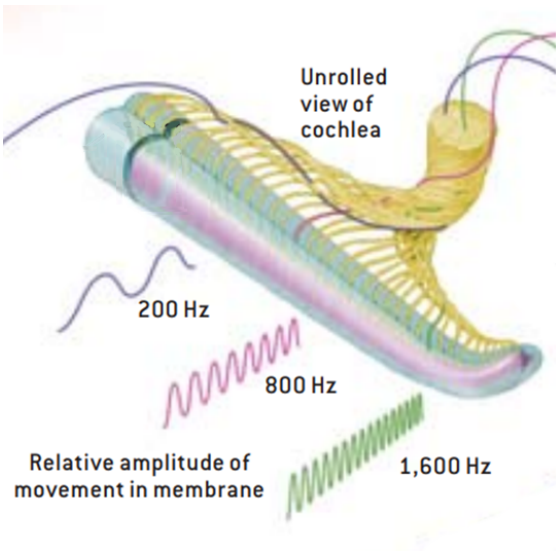


Figure 2.4: The cochlea (unrolled) responds only to higher frequencies as sound waves travel deeper [52].

side the cochlea are microscopic hair cells that resonate with various frequencies. Each hair cell has multiple hairs of varying length so as to resonate with a wide range of frequencies. The activation of these hair cells is transmitted to the brain as electrical signals via nerve endings [52]. The cone-shaped cochlea treats layered frequencies in a remarkable way. As vibrations move further down the cochlea, lower frequencies get filtered out and only very high frequencies travel to the narrowest point of the cochlea. As a result, hair cells toward the outside of the cochlea transmit low-frequency soundwaves to the brain, while cells further in correspond directly to higher-frequency soundwaves.

Once this frequency information reaches the brain, specifically the auditory cortex, the brain must convert a series of component frequencies to a unified pitch, as described in Sections 2.1.1 and 2.1.4. Different brain cells are specialized to respond to certain learned pitches. For example, over time one brain cell will respond most strongly to the pitch D, and another to the pitch A. The brain also develops its own tuning over time; with enough exposure to pitches in a certain frequency range, the brain will devote more cells to frequencies in that range and be able to perceive those frequencies more

accurately [52].

The signal processing chain from stapes to auditory cortex is not only interesting, but also largely parallel to the signal processing chain used in automatic pitch detection. The sequence of first splitting a continuous sound wave into its individual component frequencies, then interpreting those frequencies and their combinations as pitch(es) is exactly the approach most sophisticated pitch detection algorithms take.

CHAPTER 3

MUSIC TRANSCRIPTION

A transcribed piece of music is a powerful resource. Unfortunately, transcribing can be time-consuming and error-prone, and takes a certain level of musical expertise. This chapter provides an overview of transcription. First, Section 3.1 discusses the methods by which music transcription is generally accomplished. Section 3.2 describes automatic music transcription, with emphasis on the two largest subtasks, pitch estimation and note tracking. Finally, Section 3.3 discusses 'semi-automatic' music transcription, which can refer to anything from automatic transcription that requires certain contextual information, to very basic transcription tools that are simply the first step of manual transcription.

3.1 Traditional Music Transcription

When a someone is playing music, it is often the case that they are playing based on written-out music that they are reading. As they read the music, they convert the notes they see to a fingered position on their instrument or a pitch of their voice and produce the sound according to the rhythm and dynamics as written on the page. Transcription is essentially the inverse of this process. The goal is to produce a written representation of a musical piece given the audio, most likely a recording. The purpose of this is generally to learn how to play a piece of music that does not have an available written form or to understand it musicologically.

Unfortunately, music transcription can be difficult and time-consuming, especially when the recording has many overlapping pitches [16]. The difficulty of this task can be understood in comparison to the ease with which humans can read passages of text and the relative difficulty of writing down what someone is saying. To add to the complexity of the problem, humans often process pitch relatively, rather than absolutely. Some

humans have what is known as *perfect pitch*, the ability to recognize pitches in isolation. For those without perfect pitch, a guess-and-check method is the best option, but this can be very time-consuming, as the transcriber may have to listen to a passage dozens of times to get both the pitches and the rhythm completely accurate.

Despite the difficulty and tedium of music transcription, it is useful as an educational tool and the resulting transcribed piece of music can be very valuable for musicians and musicologists alike. This section offers an overview of the use cases for music transcription, both the process and the product, followed by a description of the common practices of hand-transcription of music.

3.1.1 Use Cases

The goal of music transcription is two-fold. The first goal relates to the process of transcription, while the second relates to the product.

When a musician or composer in training sits down to transcribe a piece of music, it is often with the goal of better understanding musical vocabulary and technique through the process of transcription. On a basic level, transcribing music forces the transcriber to listen to each note of a recording and acknowledge its pitch, timing, and relation to other notes around it. For a composer, this helps build a vocabulary of patterns of intervals and rhythmic choices that do or do not sound appealing or catch the listener's attention. This level of focused listening can also help the musician memorize a piece of music, an excerpt, or simply a small piece of a melody. There may also be some neurological benefit to focused and processing of melodies; Weinberger [52] notes that brain cells 'remap' themselves according to perceived 'important' pitches. Thus, music transcription by ear and hand also has benefits for general music processing power.

The above benefits may be thought of more as side-effects, however. The stated goal of music transcription is rather to produce a written-out form of an unwritten piece

of music. Analysis of music general starts with written music. Though music is an auditory art, it is structured according to patterns of frequencies and rhythms, all based on a rigid quantization of pitch and rhythm in terms of letter and measure subdivisions. As a result, in seeking evaluation or understanding of a composition, the most direct way to discuss the composition is in terms of the stripped-down version of the recording as represented on paper. As a side note, the performer of a piece of music often contributes to the emotive power through playing technique and interpretation of the composition. This level of expression would not be captured by sheet music. Even so, in seeking to understand musical trends or melodic and rhythmic choices, written music is most practical.

The most obvious use case for music transcription is music that has no written form. This could be improvisation, such as a jazz solo or an improvised section of a classical piece, or it could be orally-transmitted musical traditions such as folk music. In either case, the goal is to be able to notice the musical concepts at work in the pieces by writing them down in the form of sheet music. In the case of improvisation, no written form of the music exists because it is spontaneously created at the time of performance. In most cases, however, the improvisation is based on a melodic theme or a harmonic progression. Particularly in jazz, even when playing a written-out part there is often liberal interpretation of rhythm and added embellishments, which can also be a target for transcription. For folk music, there may be improvised components, but in general there is no written form of the music because the music has been passed down from generation to generation through performance and learning by practice.

Transcriptions are often sold in volumes for mass consumption. For a musician studying jazz, for example, there may be a book of music that contains various jazz compositions as written, each followed by a transcription of a famous jazz player's interpretation of the melody, and finally a transcription of an improvised solo for that

piece.

Transcriptions are also useful for musicology. In the subdiscipline of ethnomusicology, the close study of folk music is a central part of understanding the ways songs and melodies are morphed in oral tradition in different social contexts [51]. Transcription of folk music based purely on oral tradition in different areas is an important first step in such musicological analysis.

3.1.2 The Process

At its most basic, *traditional music transcription*, also called *manual transcription*, usually involves a guess-and-check approach for writing down the correct pitches and rhythms [27]. But before actually attempting to write down the rhythms and notes the transcriber hears, they will usually begin with gathering metainformation about the recording to be transcribed. Upon first listen of a recording, people with a musical background are usually quick to pick up on tempo, time signature, which can be helpful in dividing the transcription process into measure-by-measure subtasks. Knowing the key signature can be helpful in making educated guesses about pitch values; if a composition is known to be in the key of D major, then the notes in a D major scale are far more likely than those outside a D major scale. The key signature can be determined by attempting to accompany the piece with different low-pitched notes and finding which fit best. Music theory also suggests that compositions often end with notes of the chord that makes up the key signature. In the case of jazz improvisation transcription, the solo often follows the same chord progression as the initial melody, usually already available in some written form. If so, these chords, as well as melodic ideas from the original tune, are useful in making better note and interval hypotheses.

It is also important to take note of instrumentation and what specifically the transcriber is hoping to produce. In the case of a jazz solo, there is likely a group of ac-

companying musicians playing piano, bass, drums, or other instruments, but the focus of the transcription is only on the lead instrument. If the transcription is being done with musicological goals in mind, or if it is a transcription of a full arrangement of a piece, then it may be necessary to transcribe each individual instrument.

Once the above preparation is accomplished, the transcription process usually starts by listening to a brief segment of the recording and attempting to play it back on an instrument. Often this means sitting next to a piano and repeatedly playing different approximations of the recording on the piano and deciding what sounds right. Many musicians can identify intervals more easily than isolated pitches, so often the transcriber is able to listen to the sequence of intervals, find the starting pitch, and make highly informed guesses about the sequence of notes based on the key signature and the intervals they have heard. Still, this strategy can be very time-consuming. Exact transcription of rhythms also requires a lot of repeated listening.

Today, hand-transcription can also be done on a computer using music notation software, rather than on paper with a pen or pencil. This software often has the option to playback the music that has been entered. Listening to this can be invaluable for getting precise rhythms and pitches. This form of computer-aided transcription does likely not qualify as automatic or semi-automatic transcription, however, as the transcription part is still entirely done by a trained musician and focused listening, simply utilizing computer music generation to check their work.

3.2 Automatic Music Transcription

Automatic music transcription, the topic of this thesis, seeks to automate algorithmically the process of converting recorded audio music to written music. It has been an area of research for over 40 years and is a central part of music signal processing [34, 2]. This section covers the general process, subtasks, and evaluation of automatic music

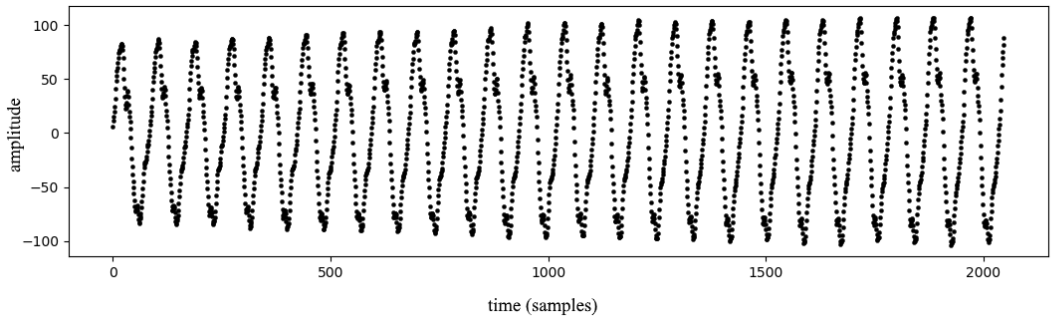


Figure 3.1: 2048 samples (46ms) of a violin note recorded at 44.1 kHz.

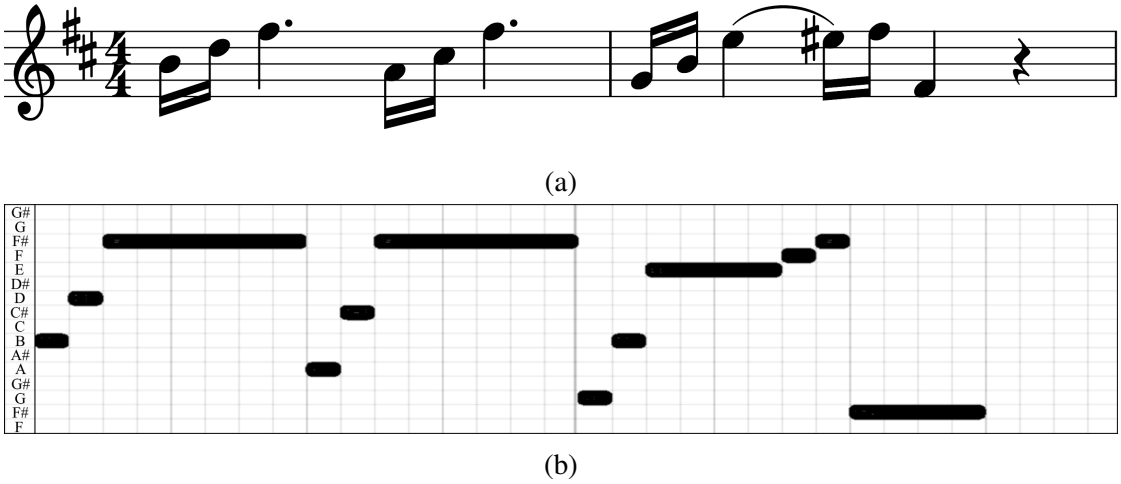
transcription, while Chapters 5 and 6 cover the subtask of pitch estimation in detail.

3.2.1 Input and Output

Automatic music transcription generally begins with so-called CD-quality audio [16, 15]. This consists of music recorded with a sampling rate of 44.1 kHz and 16-bit samples. Figure 3.1 shows 46ms of a violin note sampled at 44.1 kHz. The recordings may be stereophonic (stereo, two-channel) or monaural[1](mono, single-channel) depending on the source, but because most algorithms assume mono input, stereo sources are often converted to mono before processing [47, 54]. Though most contemporary sample databases use a sample rate of 44.1 kHz, some older pitch detection systems used significantly lower sample rates or artificially down-sample input recordings to improve runtime [34, 49], but with modern computers, this down-sampling is unnecessary.

The output of automatic music transcription is usually a digital representation of notes with specified pitch, rhythm, and sometimes loudness. This may be a stripped-down notation such as a piano-roll, or it may be a score (see Figure 3.2). A common output format is a MIDI file [40, 39, 2]. The MIDI (Musical Instrument Digital Inter-

[1]The term *monophonic* may also be used to refer to single-channel recordings, but monophonic is also used in contrast to polyphonic, referring to the number of notes played simultaneously in a recording and having no relation to the number of audio channels recorded. As such, the terms *monaural* and *mono* will be used exclusively for describing audio recording choices while the term *monophonic* will refer exclusively to the musical context of a composition or performance.



(a)

(b)

Figure 3.2: The first two measures of Telemann's Fantasia No. 3 in B minor represented as (a) a music score and (b) a piano-roll.

face) standard has the benefit of easy conversion into piano-roll notation, sheet music, or interfacing with digital instruments.

3.2.2 Subtasks

Automatic music transcription can be divided into various related subtasks. The most important subtasks are pitch estimation and note tracking [2]. Additional goals of automatic music transcription may include instrument detection and harmony identification. Auditory scene analysis and music scene description are related tasks of automated music analysis but generally considered distinct from transcription [22, 16].

Pitch Estimation

There are two versions of the pitch estimation problem—single pitch estimation and multi-pitch estimation. Single pitch estimation has its roots in speech-oriented pitch detection algorithms, going back over 50 years [14, 48]. This is generally considered a solved problem, and various techniques are described in Chapter 5. The real challenge

of automatic music detection lies in recordings with multiple simultaneously sounded pitches. Emmanouil Benetos writes that "The core problem in automatic transcription is the estimation of concurrent pitches in a time frame, also called multiple-F0 or multi-pitch detection" [2, p. 408]. Different strategies for multi-pitch detection are covered in Chapter 6. Unlike single-pitch estimation, multi-pitch estimation is still unsolved [1]. As of 2017, the best algorithms were able to achieve around 70% accuracy [13, 37].

Note Tracking

In general, pitch detection will produce a continuous stream of pitch hypotheses. The resolution of this stream of pitch hypotheses will depend on the window size used by the algorithm (see Sections 4.4.1 and 5.2). The task of *note tracking* is to move from an indexed stream of pitch hypotheses to a note-based representation of these pitches quantized by time—specifying duration and activation time—and by frequency—labeled by letter note instead of frequency value. Note tracking is also related to the problem of *beat tracking*, which seeks to find precise rhythmic structure for a recording in terms of tempo and rhythmic patterns [18].

Simple approaches to note tracking use thresholds based on note duration and loudness [10]. Duration-based thresholding is referred to as minimum duration pruning, by which note hypotheses that are present only for a very short duration are discarded as unlikely candidates. This helps to smooth out potentially noisy streams of pitch hypotheses. Loudness-based thresholding accounts for moments in the recording with no notes sounded. After thresholding, neighboring samples that indicate the same note value are combined into a single note. More complex methods use hidden Markov models, of which an example is shown in Figure 3.3 [40].

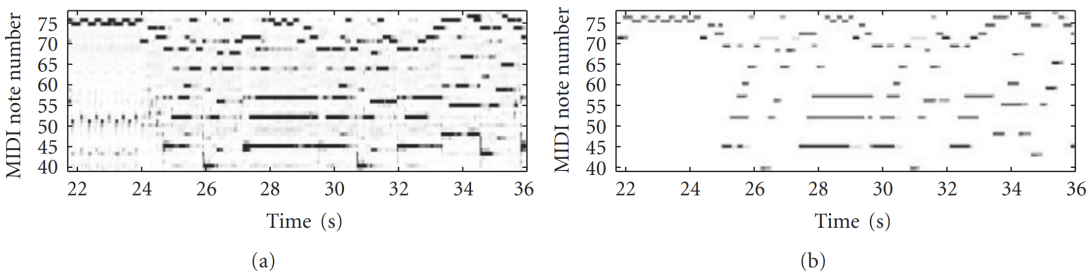


Figure 3.3: (a) Pitch hypotheses from an audio recording of Beethoven's Für Elise. (b) Smoothed note estimations using a hidden Markov model approach to note tracking [40].

Instrument Identification

Another common task related to music transcription is musical instrument identification. Given an audio recording, the goal is to identify the musical instrument(s) playing each note. The problem of instrument identification faces many of the same challenges as pitch estimation. Like pitch estimation, instrument detection has a monophonic and polyphonic version of the problem, with the monophonic version more easily solved and the polyphonic version still unsolved [30]. The problem can be difficult even for humans very accustomed to listening to music. Distinguishing between trombone and French horn (both low-pitched brass instruments), for example, can be a challenge, even for a trained human listener. The problem is often simplified to classifying a recording in terms of *instrument family*, not between specific instruments.

In all forms of the instrument identification problem—monophonic or polyphonic, specific instrument or instrument family—the most common strategy is an evaluation of various features present in a note as its harmonic shape develops over time. These features are generally based on timbral qualities (Section 2.1.4) and could be temporal—the sharpness of attack and speed of loudness decay—or could be based on the harmonic content of the note's frequency spectrum. Livshin and Rodet separate this latter category into three categories, namely energy features, spectral features, and harmonic features [30]. These three categories use different recording processing methods to cap-

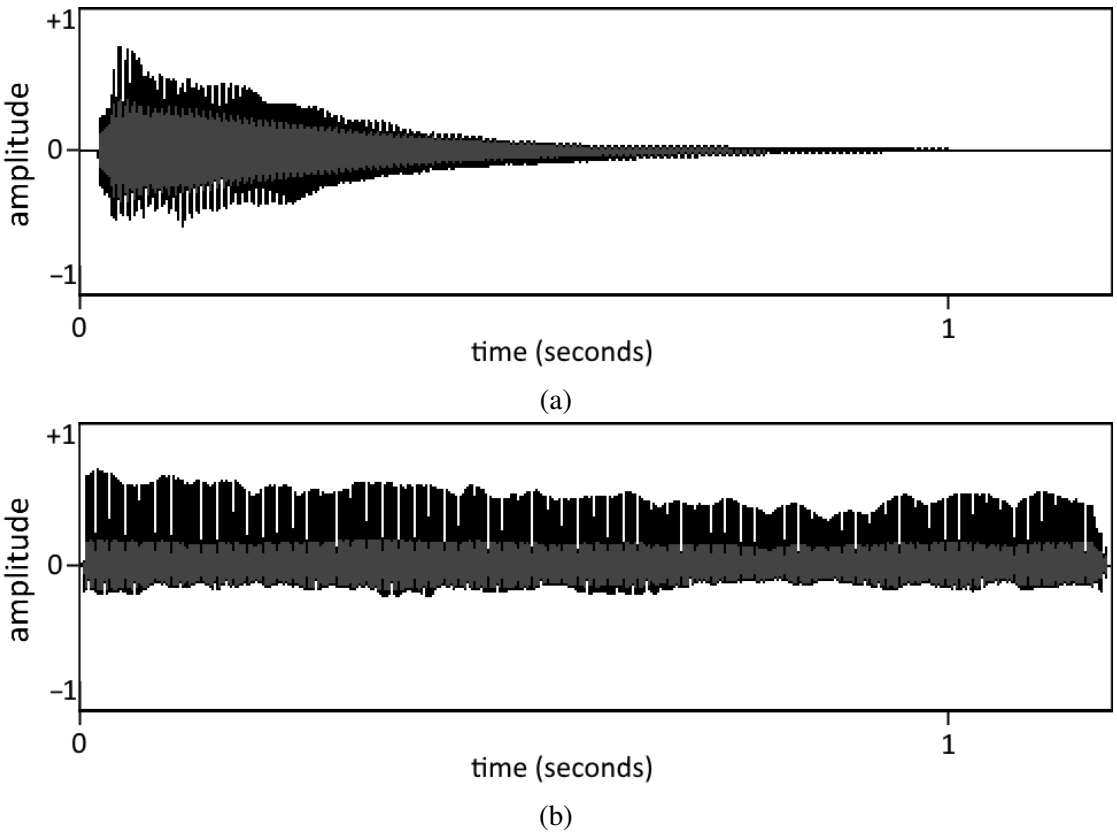


Figure 3.4: The dynamic envelope of a note played on (a) piano and (b) trumpet. The piano has a softer attack and its loudness decays over time. The trumpet has a fast attack and sustained loudness. These temporal features can be used to distinguished between different families of instruments.

ture similar traits of the recording's harmonic content and inharmonicity. Figure 3.4 shows visual representations of temporal features between trumpet and piano. Once recordings are analyzed based on these features, classifications can be learned using a k-nearest-neighbors approach, decision trees, or artificial neural networks, all of which Herrera-Boyer et al. explain in detail [19]. Martin and Kim were able to achieve 93% accuracy in identifying musical instrument family in monophonic recordings using a pattern-recognition approach [32]. Meanwhile, Herrera-Boyer et al. describe algorithms that identify instruments in duet recordings and so-called 'complex mixtures.' Essid et al. [12] use a hierarchical classification method specifically designed to identify instruments present in jazz recordings of 1-4 musical instruments, achieving an average

accuracy of 65%, but as high as 91% classification accuracy in certain instrument combinations.

3.2.3 Evaluation

A key resource in developing and evaluating automatic music transcription systems is a large base of audio with accompanying ground-truth results. Audio is generally distributed at 44.1 kHz uncompressed audio on CD [38, 17] or mp3 compressed audio available online for download [33]. MIREX (Music Information Retrieval Evaluation eXchange) also performs annual evaluation of new automatic music transcription systems [13]. Ground truth is generally released as MIDI files. The generation of ground truth for audio recordings is inherently difficult and error-prone, requiring extensive traditional transcription. The RWC (Real World Computing) Music Database is one of the few datasets entirely developed for the sake of developing automatic music transcription systems and related music information retrieval tasks [17]. For the RWC Music Database, 315 recordings and accompanying ground truth MIDI files were newly created over two years by the RWC Music Database Sub-Working Group [15]. Most other datasets are comprised of computer-generated audio. The process entails generating MIDI files and then rendering these MIDI files as audio using existing software [55, 33, 13]. This drastically increases the available evaluation data, but computer-generated audio risks ignoring challenges that real recordings present, such as reverberations, noise, variations in frequency response, etc.

3.3 Semi-Automatic Music Transcription

Somewhere between traditional music transcription and automatic music transcription is *semi-automatic music transcription*, also called *user-assisted transcription*. Kirchhoff

et al. define semi-automatic music transcription as "systems in which the user provides a certain amount of information about the recording under analysis which can then be used to guide the transcription process" [23]. For certain use-cases, semi-automatic music transcription is more practical than fully automatic and is often faster and more accurate than manual transcription. For other applications, however, semi-automatic music transcription is insufficient. One promise of automatic music transcription is the development of transcription databases too large to be done by hand, with big-data-style ethnomusicology in mind. With such a project, semi-automatic transcription could require too much user input.

In general, semi-automatic systems employ user-given information as priors for better pitch hypothesis generation and evaluation [1]. This could be tempo, time signature, or other metainformation. In the case of Kirchhoff et al. 2012 [23], the user first gives a transcription of a few notes which the algorithm then uses to develop frequency profiles for the source audio. This allows for more accurate timbre modelling of the instruments, a process that is generally based on basic assumptions of harmonicity [26].

CHAPTER 4

THE FOURIER TRANSFORM

This chapter introduces the Fourier transform, a mathematical tool used for many kinds of signal processing, including audio and music. The Fourier transform does mathematically what the cochlea and inner ear's hair cells do for humans and animals. Starting with a continuous, quasiperiodic waveform, the Fourier transform of that waveform is a function of frequency that gives their respective amplitudes and phases. The frequency domain is often more practical and intuitive for signal processing applications than direct analysis of the raw time-domain recording.

The Fourier transform (FT) has a 200-year history [4]. In nature it occurs in prisms—which break light rays into their composite frequencies, heat transfer—the phenomenon that inspired Jean Baptiste-Joseph Fourier himself to describe the concepts of the Fourier transform—and, of course, ears in animals. In 1965, the transform was brought into the modern world as the Fast Fourier Transform by J. W. Cooley and J. W. Tukey with their publication "An algorithm for the machine calculation of complex Fourier series" [9]. The Fast Fourier Transform (FFT) reduces the complexity of the transform significantly, making it practical for various signal processing techniques.

This chapter first explains how the algorithm works, both in the general case (FT) in Section 4.1 and the discrete case (DFT) in Section 4.2. Both sections contain informal derivations of the procedures. Next is an in-depth description of Cooley and Tukey's Fast Fourier Transform in Section 4.3. Section 4.4 describes the specific use case of the Fourier transform applied to audio and music recordings, and finally Section 4.5 briefly discusses alternative methods to the Fourier transform for analyzing audio.

4.1 Informal Derivation of the Fourier Transform

The Fourier transform is a remarkably diverse tool for understanding natural phenomena in physics, biology, signal processing, and many other disciplines. This section provides an informal derivation of the Fourier transform, following the explanation given in [8].

The Fourier transform begins with the fundamental principle that any periodic (i.e., repeating) curve or function can be described by an infinite sum of sinusoids. This is known as the Fourier expansion of periodic function $h(t)$, and is described mathematically as

$$h(t) = a_0 + \sum_{m=1}^{\infty} a_m \cos \frac{2\pi mt}{T} + \sum_{n=1}^{\infty} b_n \sin \frac{2\pi nt}{T} \tag{4.1}$$

Here, $T/2\pi$ is the period of $h(t)$, the coefficients a_0, a_1, a_2, ... and b_1, b_2, ... are the amplitudes of a component sine or cosine at frequency $2\pi m/T$. This was Jean Baptiste-Joseph Fourier's remarkable claim, that any periodic function $h(t)$ is equal to an infinite sum of cosines and sines at varying frequencies and amplitudes. When combined together, all the frequencies at the amplitudes and phases given by the Fourier transform recreate the original waveform.

The general task of the Fourier transform, then, is to find these coefficients a_n and b_n, which correspond to the amplitude of the frequency n. A much more detailed discussion of the transform can be found in [21]. The general strategy is based on unique properties of sines and cosines. Though Fourier expansions must account for all periodic functions, thus an infinite sum of both sines and cosines, it is easiest to explain the derivation in terms of one then the other. Assuming the given function $h(t)$ is an even function, that is $h_e(t) = h_e(-t)$, this sum can be simplified to

$$h_e(t) = \sum_{n=0}^{\infty} a_n \cos(n\omega_0 t), \tag{4.2}$$

where ω_0 refers to the frequency of $h(t)$, $\omega_0 = 2\pi/T$.

We seek to find a_n for all harmonic frequencies $n\omega_0$. This is accomplished with the

use of certain trigonometric identities and a special feature of the integral of sinusoids. Equation (4.2) is first multiplied by $\cos(m\omega_0 t)$ for some m on both sides and integrated over one period, giving

$$\int_T h_e(t)\cos(m\omega_0 t)dt = \int_T \sum_{n=0}^{\infty} a_n \cos(n\omega_0 t)\cos(m\omega_0 t)dt. \tag{4.3}$$

Expanding using the identity $\cos(a)\cos(b) = 1/2(\cos(a+b)+\cos(a-b))$, and simplifying some, we have

$$\begin{aligned}\int_T h_e(t)\cos(m\omega_0 t)dt &= \sum_{n=0}^{\infty} a_n \int_T \cos(n\omega_0 t)\cos(m\omega_0 t)dt \\ &= \frac{1}{2}\sum_{n=0}^{\infty} a_n \int_T \cos((m+n)\omega_0 t)\cos((m-n)\omega_0 t)dt.\end{aligned} \tag{4.4}$$

Here, the special feature of integrals of sinusoids is invoked. For any integral over a multiple of a sinusoid's period, the integral will always sum to zero because it will have some number of complete oscillation, each integrating to zero. Splitting Equation (4.4), we find that the left integral reduces to zero for all integer values of m, n

$$\begin{aligned}\int_T h_e(t)\cos(m\omega_0 t)dt &= \frac{1}{2}\sum_{n=0}^{\infty} a_n \int_T \cos((m+n)\omega_0 t)\cos((m-n)\omega_0 t)dt \\ &= \frac{1}{2}\sum_{n=0}^{\infty} a_n \left(\int_T \cos((m+n)\omega_0 t)dt + \int_T \cos((m-n)\omega_0 t)dt\right) \\ &= \frac{1}{2}\sum_{n=0}^{\infty} a_n \int_T \cos((m-n)\omega_0 t)dt.\end{aligned} \tag{4.5}$$

Further, the right integral is zero for all values except when $m = n$. If $m = n$, then we have $\cos(0) = 1$. This means the integral will just be the period T, so the right side of Equation (4.4) reduces to:

$$\int_T h_e(t)\cos(m\omega_0 t)dt = \frac{1}{2}a_m T. \tag{4.6}$$

This gives a concise equation for finding any coefficient a_m:

$$a_m = \frac{2}{T} \int_T h_e(t) \cos(m\omega_0 t) dt. \tag{4.7}$$

A very similar strategy can be employed for any odd function $h_o(t)$ using the sine instead, giving

$$b_n = \frac{2}{T} \int_T h_o(t) \sin(n\omega_0 t) dt. \tag{4.8}$$

Individually, Equations (4.7) and (4.8) give a complete description of any periodic function, provided it is even or odd. However, to handle period functions in general, Fourier analysis uses the complex form of sine and cosine. Moving into the general case requires the use of the following trigonometric identity:

$$a_n \cos(n\omega_0 t) + b_n \sin(n\omega_0 t) = c_{-n} e^{-j2\omega_0 t} + c_n e^{+j2\omega_0 t}. \tag{4.9}$$

Representing sine and cosine as a complex exponents according to this identity means we can describe any given sinusoid not as a sum of a sine and cosine at the same frequency but different amplitudes, but as a single sinusoid specified by frequency and *phase*. Phase is the distance the sinusoid is shifted in the t (horizontal) direction from the origin. Thus, there are half as many coefficients to contend with and it is generalizable to all periodic functions. The complex Fourier expansion is given as

$$h(t) = \sum_{-\infty}^{+\infty} c_n e^{jn\omega_0 t}, \tag{4.10}$$

where c_n is a complex coefficient, whose real component captures the magnitude of the sine and whose imaginary component corresponds to the magnitude of the cosine. Its magnitude, then, is the power of that frequency, and its complex angle is that frequency's phase.

Following a similar derivation as above, we have a more concise means of obtaining the coefficients:

$$c_n = \frac{1}{T} \int_T h(t) e^{-jn\omega_0 t} dt. \tag{4.11}$$

With Equation (4.11) we have a ready way of finding the coefficients for all sinusoids with frequencies $n\pi/T$ from a function of period T. The Fourier transform takes this method for finding a single coefficient and defines a singular equation that describes these coefficients as a function of frequency:

$$H(f) = \frac{1}{2\pi} \int_{-\infty}^{\infty} h(t) e^{-j2\pi ft} dt. \tag{4.12}$$

$H(f)$ is called the Fourier transform of $h(t)$.

4.2 Discrete Fourier Transform

This general form of the Fourier transform assumes $h(t)$ is a continuous function that extends from $-\infty$ to ∞. This, then, will yield a Fourier transform that also extends from $-\infty$ to ∞. However, in most practical applications, $h(t)$ is not used as a continuous function, but only as a set of discrete, evenly sampled values of a curve [41]. Furthermore, we are not interested in the Fourier transform of an entire recording of music or a single periodic waveform, but rather the transform of small, sequential windows of the recording. The discrete, finite nature of our sampled data allows for simplifications of the general, continuous Fourier transform.

The discrete Fourier transform (DFT) of $h(t)$ assumes as input N sampled points at a sampling rate of Δ, where $h_k = h(k\Delta)$. A consequence of having only N data points is that it will only be possible to detect N different frequencies. More specifically, if $h(t)$ is sampled with a spacing of Δ, then the maximum detectable frequency, known as the Nyquist critical frequency, will be $1/2\Delta$. The discrete Fourier transform gives amplitudes and phases for N frequencies from $-1/2\Delta$ to $1/2\Delta$. It is given by approximating the integral form of the Fourier transform as

$$H(f) = \int_{-\infty}^{\infty} h(t) e^{j2\pi ft} dt \approx \sum_{k=0}^{N-1} h_k e^{j2\pi f t_k} \Delta = \Delta \sum_{k=0}^{N-1} h_k e^{j2\pi kn/N}, \tag{4.13}$$

and will produce N complex numbers corresponding to the amplitude and phase of the sinusoids from $-1/2\Delta$ to $1/2\Delta$, at an interval of $1/N\Delta$, the frequency resolution determined by the sampling rate and number of samples [41].

4.3 Fast Fourier Transform

In theory, the discrete Fourier transform described in Section 4.2 is sufficient for spectrum analysis of audio signals. But an age-old computational problem presents itself: efficiency. This is where the Fast Fourier Transform (FFT) comes in. In its pure form, the discrete Fourier transform H_n must sum values from h_0 to h_{N-1} for every frequency n, meaning an $O(N^2)$ time complexity for N data points. With the FFT, this can be reduced to $O(N \log_2 N)$ using a divide-and-conquer approach.

The fundamental discovery of the FFT is that the discrete Fourier transform of N points can be described as the sum of two DFTs, each of $N/2$ points. Specifically, it is the sum of the Fourier transform of the even points E_k and a complex exponent W^k times the Fourier transform of the odd points O_k, where W is defined as $W \equiv e^j 2\pi/N$ for ease of writing, because this exponent is so frequently used. The following proof appears in [41], Equation 12.2.3:

$$\begin{aligned} H_k &= \sum_{n=0}^{N-1} h_n e^{j2\pi nk/N} \\ &= \sum_{k=0}^{N/2-1} h_{2n} e^{j2\pi k(2n)/N} + \sum_{k=0}^{N/2-1} h_{2n+1} e^{j2\pi k(2n+1)/N} \\ &= \sum_{k=0}^{N/2-1} h_{2n} e^{j2\pi kn/(N/2)} + W^k \sum_{k=0}^{N/2-1} h_{2n+1} e^{j2\pi kn/(N/2)} \\ &= H_k^e + W^k H_k^o \end{aligned} \tag{4.14}$$

All that then needs to be done is to use this feature recursively. There is one important restriction on the recursive use of this function, however. Equation (4.14) assumes an

equal number of even and odd points, hence N must be an even number. In order to be able to repeatedly and evenly split a data set into its even and odd parts, all divisions must remain even in length, hence N must be a power of two. In practice this presents very few problems, as padding or shifting the data can achieve this goal.

The discrete Fourier transform of $h(t)$ can then be described recursively as

$$H_k = \begin{cases} h_k & N = 1 \\ E_k + W^k O_k & 0 \leq k < N/2 \\ E_{k-N/2} + W^k O_{k-N/2} & N/2 \leq k \leq N \end{cases} \tag{4.15}$$

The base case is quite simple. When the data size $N = 1$, the function being transformed is simply a horizontal line at $h(k)$. Thus, the Fourier transform will yield $H(k) = a_0 = h(k)$. This can be implemented quite straightforwardly using two recursive calls and combining the results according to Equation (4.15). There are many different implementations of the Fast Fourier Transform, all based on this divide-and-conquer strategy. The best implementations all achieve a $O(N \log_2 N)$ time complexity and require no additional memory space. A detailed explanation and implementation of the Fast Fourier Transform is given in Chapter 12 of Press et al.'s book *Numerical Recipes in C++: The Art of Scientific Computing* [41].

4.4 Fourier Transform for Audio and Music

Chapters 2 and 3 have explained the theoretical foundations for music and transcription. Given the nature of how musical sounds are created and processed by humans, the need for the Fourier transform should be evident. The Fourier transform is the purest way to break down a segment of music into its composite frequencies, which in turn reveals pitch, harmony, and timbre. What owes more explanation, however, is how the Fourier transform—specifically, the short-time FFT—is used in practice for music and audio

processing.

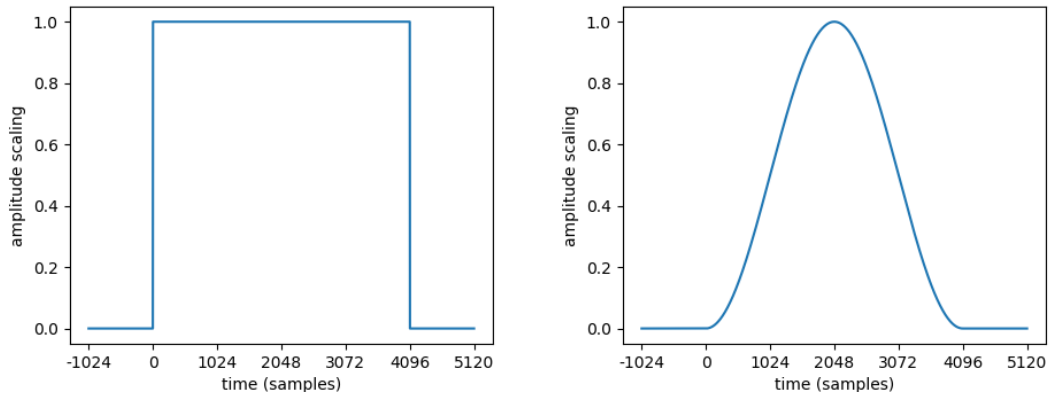
In this discussion we focus on the spectrum preprocessing methods proposed by Kunieda et al. [28] and Klapuri [26].

4.4.1 Short-time Fourier Transform

The short-time Fourier transform, as the name suggests, is used for very small windows of a sampled curve. Whereas Fourier analysis of, for example, an earthquake will perform analysis on the entire seismic curve, giving a singular transform vector, audio analysis needs to produce not a single spectrum of all frequencies present, but rather the changing spectrum over time, or *spectrogram*. The most common way to utilize the Fast Fourier Transform for music is to break the recording into windows or frames. This also serves to make the operations of the Fourier transform more tractable by dealing with equal-length, fairly small windows.

All audio recordings used in the experiments were 16-bit WAV files, sampled at 44.1 kHz. This means in one second of audio there are 44,100 data points. Various window sizes have been used for pitch detection algorithms over the past several decades, from 512 samples [28] to as many as 4096 or 8192 samples [26]. The window size has varied over time as computers have gotten more efficient and larger vectors became more manageable for analysis. A more important factor in choosing a window size, however, is the use case [26]. For speech analysis, smaller frames are necessary to get more rapid changes in pitch. Because most pitch detection algorithms from the 1960s to the 1990s was narrowly focused on voices, small frames were the norm.

Spectrum-based tempo and beat detection algorithms also use very small windows because it is focused on the time domain more than the frequency domain [29]. In such systems, it is common to use a much tighter window spacing than window size. Goto and Muraoka [18] use audio downsampled to either 22.05 kHz or 11.025 kHz and 1024-



(a) (b)

Figure 4.1: A comparison of windowing functions for a 4096-sample frame of audio. The rectangle window (a) fulfills the bare requirements of a window function but leaves artifacts after processing. Using the Hann window (b) produces a more accurate Fourier spectrum.

sample windows, but the window shift is only 256 or 128 samples, meaning there is time-precision of 11.61ms.

For music pitch detection, however, frequency precision is often more important than time precision. Consider a common tempo, 120 beats per minute (bpm). This means a quarter note is half a second or 500ms in length, an 8th note is 250ms, and a 16th note 125ms. Klapuri [26] uses either a 93ms (4096-sample) or 186ms (8192-sample) window, meaning 16th-note resolution can be fully achieved with the smaller window. A common technique to improve time precision is to use overlapping windows, spaced more narrowly than their width. This, however, must be done only within reason to avoid high computational cost.

Windowing Function

A special window function is also usually used to avoid negative effects at the edges of the frame. In general, a window function is defined as a function that produces values within a range (the window) and values of zero everywhere outside the window. In audio processing this is usually the Hann window or the Hamming window [26, 28].

This window serves as a scaling function for input amplitudes and is easily calculated for any frame length. Figure 4.1 shows a comparison of the amplitude scaling factors of a simple rectangle window and of a Hann window on a 4096-sample frame of audio.

Zero Padding

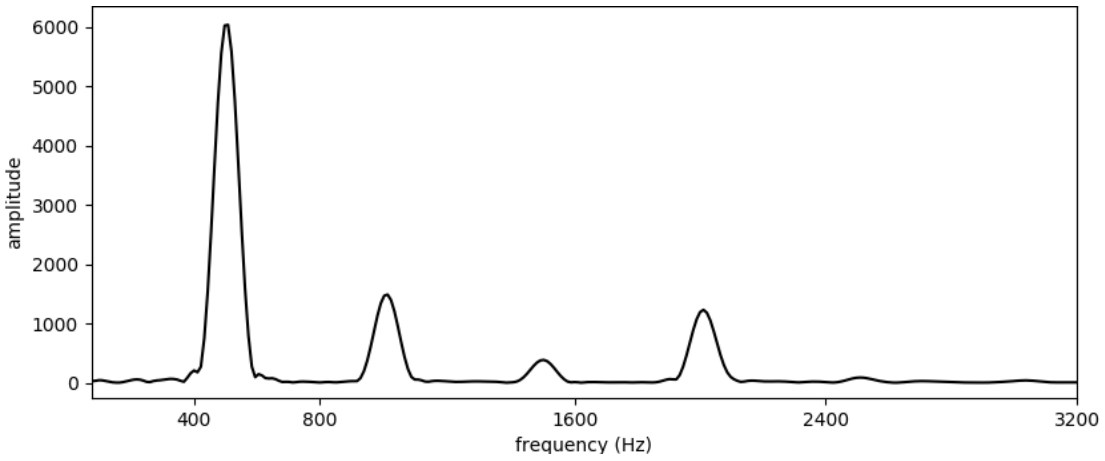
After applying a windowing function, a common preprocessing step before performing a Fourier transform is *zero padding* [24, 39]. Zero padding does not increase resolution of the discrete Fourier transform of a window, but it does improve precision of spectral peaks. This is helpful in converting from index in the discrete Fourier transform to true frequency (Hz) and musical note. Figure 4.2 shows the difference between a 1024-sample audio frame processed using a Fast Fourier Transform with and without zero padding of 3072 samples, creating a frame width of four times the original.

4.4.2 Processing the Spectrum

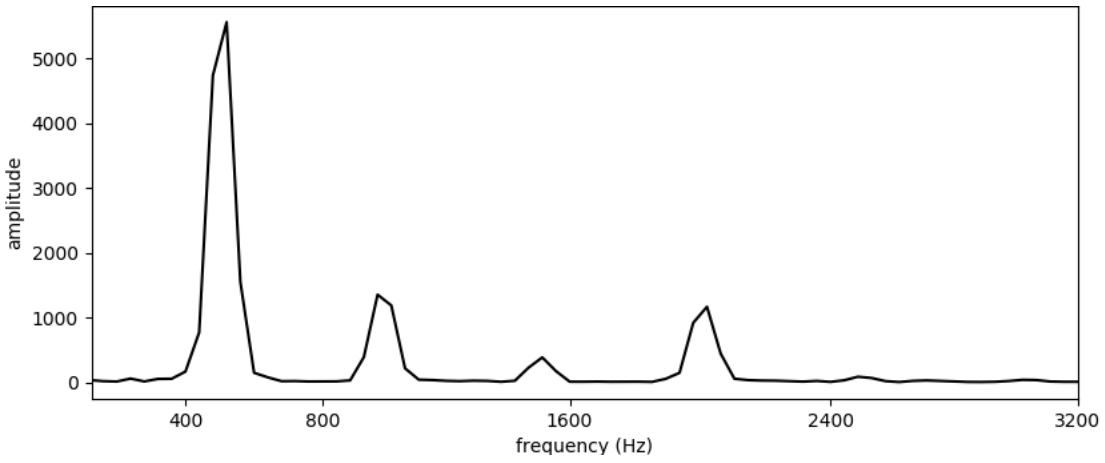
The Fourier transform does the bulk of the work for frequency detection, but many pitch detection algorithms use some form of scaling and processing on the spectrum (a notable exception is Pertusa and Iñesta [39] which processes the Fourier power spectrum directly). The most common and basic of these is converting to the log power spectrum [49]. This involves applying a Gaussian norm to of each frequency amplitude, so as to remove the imaginary component, then performing a natural log

$$L_k = \ln ||H_k|| \tag{4.16}$$

The natural log's gradual roll-off means particularly high-weight frequencies will get scaled down more significantly than lower-weight frequencies, thus giving a more even distribution of frequency magnitudes while still maintaining the same ordering of relative weights. In some algorithms the log power spectrum is also defined as $\ln ||H_k^2||$ or $(\ln ||H_k||)^2$ to keep slightly more clarity of spectral peaks.

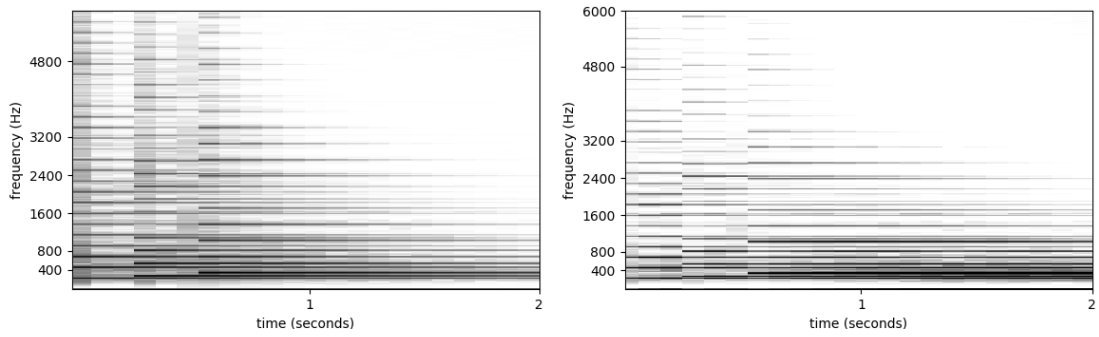


(a)



(b)

Figure 4.2: A 1024-sample frame of audio processed using the Fast Fourier transform, (a) with and (b) without zero padding. The zero-padded spectrum has a smoother curve and more precisely located peaks, despite not actually increasing spectrum resolution.



(a) (b)

Figure 4.3: The spectrogram of guitar notes in (a) the log power spectrum and (b) the spectrum preprocessed according to Equation (4.17). Note that in (b) there is less noise between harmonics and the higher harmonics have less dynamic decay over time.

In his paper on multi-pitch detection, Klapuri develops a more involved preprocessing chain [26]. Starting from the power spectrum, instead of performing a raw $\log$ operation, Klapuri's algorithm performs a log of the power scaled by the total power of the frame over the relevant frequencies $k_0 ... k_1$, described by

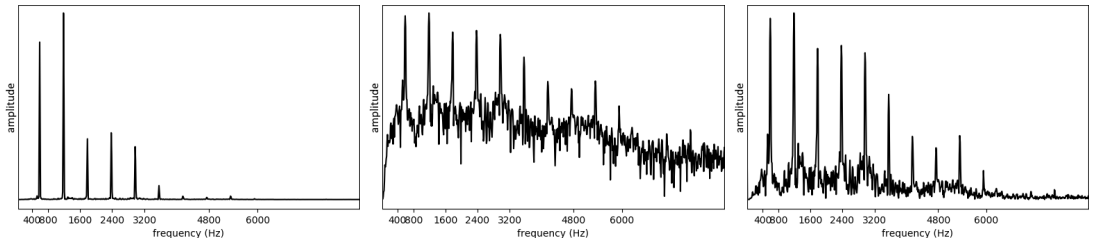
$$Y(k) = \ln\left(1 + \frac{1}{g}H(k)\right) \tag{4.17}$$

where

$$g = \left(\frac{1}{k_1 - k_0 + 1}\sum_{l=k_0}^{k_1} H(l)^{\frac{1}{3}}\right)^3. \tag{4.18}$$

This accomplishes both the evening out of spectral peaks and normalization of volume. Figure 4.3 shows how this preprocessing technique evens out the dynamic envelope in instruments with quickly decaying sounds such as string and percussion instruments, but is very helpful for recordings with large dynamic range. Figure 4.4 shows a comparison of the power spectrum, log power spectrum, and spectrum preprocessed according to Equation (4.17).

Noise suppression is also commonly used but is somewhat outside the scope of this thesis. The most basic means of avoid noise is squaring the spectrum, which brings out spectral peaks more clearly than low-level noise. But noise suppression can get much



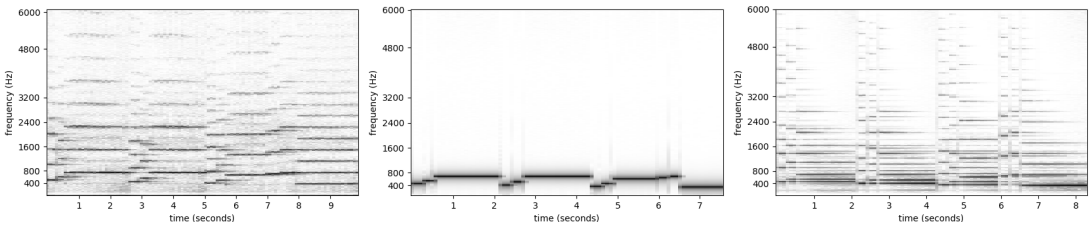
(a) (b) (c)

Figure 4.4: The Fourier transform of a trumpet note. (a) Unprocessed Fourier transform. (b) Log power spectrum, processed using Equation (4.16). (c) Spectrum processed according to Equation (4.17).

more involved. One approach is to subtract the noise from the spectrum using a moving average. This is described in detail in [24].

4.4.3 Interpreting Power Spectra

Visualizing a sequence of Fourier transforms on windowed audio yields a spectrogram (spectra plotted over time). Figure 4.5 shows the preprocessed power spectrogram of the first four measures of Georg Philipp Telemann's Fantasia No. 3 in B minor, written for flute, performed on three different instruments: flute, pure sine wave, and synthesized guitar. In the case of the flute recording, a few things stick out. Though just one note is played at a time, the harmonics of the fundamental frequencies are clearly visible, occurring at all detected multiples of the fundamental frequency. This can be contrasted with the pure sine wave rendition of Telemann's piece, which contains only the fundamental frequency. Meanwhile, rendered on computer-synthesized guitar, different harmonics appear. Also clearly visible is the quick decay of the guitar's upper harmonics, whereas the flute and sine wave maintain relatively constant volume.



(a) (b) (c)

Figure 4.5: The Fourier spectrogram of Telemann's Fantasia No. 3 in B minor, performed on (a) flute, (b) sine wave, and (c) synthesized guitar. All spectrograms were preprocessed according to Equation (4.17) for clarity.

4.5 Alternatives to the Fourier Transform

The Fourier transform is the cleanest way to break an audio recording into its composite frequencies and their weights, but there are other techniques used by pitch detection systems. One alternative is to do analysis directly on the waveform (i.e., do analysis in time domain rather than frequency domain) and is the focus of Section 5.2.

Another alternative is the *constant Q transform* [5], which is very similar to the Fourier transform but uses a different set of frequency indices better suited to music's log-frequency scale. The Fast Fourier Transform generates a vector of weights indexed by frequency, with the input frequencies evenly spaced in the range $-1/2\Delta...1/2\Delta$. But musical pitches operate on a log-frequency scale. This means the frequency resolution required to distinguish between low frequencies, where pitches can be as little as 5Hz apart from each other is much higher than the frequency resolution for higher frequencies where one half-step can be a difference of 200 Hz. The idea of the constant Q transform is to give a transform that is indexed by frequencies with a spacing defined in terms of half-steps rather than Hertz. Q refers to the ratio of pitches along the log spectrum. In [5], the frequencies are indexed by 1/24-octave intervals, or two bands per half-step. Remarkably, the constant Q transform can be calculated directly from the Fast Fourier Transform, and actually yields some improvement in efficiency [6].

CHAPTER 5

FUNDAMENTAL FREQUENCY ESTIMATION

The core task of automatic music transcription is pitch estimation [2]. This is often referred to more broadly as estimation of the *fundamental frequency* of a waveform, notated F0. This chapter introduces the foundational techniques used for single-pitch F0 estimation. It begins with a historical context for F0 estimation in Section 5.1, but the bulk of the chapter is divided into two categories of pitch estimation: periodicity-based in Section 5.2 and spectrum-based in Section 5.3. In most cases, of course, music transcription will have input with more than one fundamental pitch present. Chapter 6 describes the process of multi-pitch estimation as applied to automatic music transcription.

5.1 History and Applications

The development of pitch estimation algorithms goes back to the 1960s with Bernard Gold's 1962 paper "Computer program for pitch extraction" [14], A. Michael Noll's "Cepstrum pitch determination" [35], and Mohan Sondhi's paper "New methods of pitch extraction" [48]. However, most research before the 1990s was heavily oriented toward vocal pitch detection with speech analysis in mind. The main motivation for Gold [14] and others [46] was more accurate vocoder technology, in the aim of low-bandwidth telecommunication. This came in response to a paper written in 1940 on the use of a vocoder for a more compact representation and transmission of speech [11]. Prior to research in the 1960s, pitch estimation was largely done using filter banks, where the fundamental frequency was guessed based on the filter with the strongest output [14]. Much research on speech analysis and encoding in the mid-twentieth century was carried out at Bell Telephone Laboratories [35, 11] and Lincoln Laboratories at MIT [14]. The 1990s saw the publishing of "A robust algorithm for pitch tracking (RAPT)" [49], which

discussed and synthesized various developments in pitch tracking over the previous 30 years, culminating in a robust and heavily-cited algorithm for pitch estimation. The applications of pitch estimation for music were being explored as early as 1977 [34], but the joining of musical analysis and signal processing did not become an active area of research until the early 2000s.

Speech Analysis

Pitch estimation for music and pitch estimation for speech are deeply connected, and indeed the former evolved out of the latter. Still, there are important differences in the two that owe some attention. The first distinction is that, with few exceptions, pitch detection for speech analysis safely assumes there is only one significant or relevant pitch to be detected at any given moment. For telecommunication and speech detection purposes, this is not a problem (unless two speakers are talking over each other). For music, however, in almost every case there are overlapping pitch sources, either from a polyphonic instrument such as a piano, or from multiple monophonic instruments playing together. Section 5.2 discusses the implications for the applicability of period-based approaches—which assume a single periodic waveform—for multiple-F0 detection and automatic music transcription. As Chapter 6 shows, the presence of multiple relevant pitch sources is the greatest unsolved problem of automatic music transcription.

There are also certain problems in speech pitch detection that are less relevant for musical pitch detection. The clearest of these is unvoiced speech sounds. RAPT [49] adds an additional preliminary pitch detection step to isolate regions of the audio recording where there is no strong pitch hypothesis, which indicates the presence of an unvoiced consonant. Speech analysis also requires potentially higher time accuracy than music transcription, because speech events can be very short-lived compared to music notes. As a result, music transcription systems generally have flexibility to use much

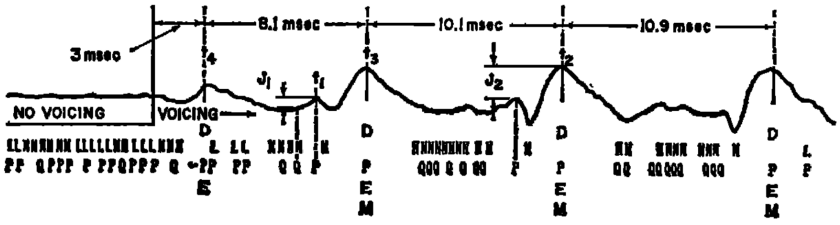


Figure 5.1: A demonstration of the pitch detection algorithm described in Bernard Gold's 1962 paper "Computer program for pitch extraction" [14]. The algorithm matches distances between individual peaks and labels regions of the waveform of different rate of change to match occurrences of irregular waveforms.

longer frames for frequency analysis [26, p. 806].

5.2 Period-Based Pitch Estimation

Many early pitch estimation algorithms performed analysis directly on the input waveform. The work by Gold in 1962 [14] was one of the first attempts to develop a reliable algorithmic pitch detection system. It was based on finding patterns directly present in a sampled waveform, namely the spacing of peaks in the wave and regularity of wave shape, as shown in Figure 5.1. While direct calculations and comparisons of waveform peaks are not used in any direct way in automatic music transcription systems, the goal of finding periodicity (i.e., regularity) in the waveform inspired a category of *correlation*-based pitch-detection algorithms. The following section details the features and implementations of two influential period-based pitch estimation techniques. Both scan a waveform and produce a series of period hypothesis weights, from which the highest-weight hypothesis is chosen as the most likely fundamental frequency. These are the *autocorrelation function* (ACF), as described by Lawrence Rabiner [42] and the related *normalized cross-correlation function* (NCCF) as utilized in RAPT [49].

The autocorrelation function and its relative the cross-correlation function are based on the principle of regularity in waveforms. Both Gold's algorithm and others assume a

quasiperiodic audio waveform, that is, a time-amplitude curve that has significant repetition, but not one that is necessarily fully *periodic*. A fully periodic waveform would have an identical shape every repetition, whereas quasiperiodic curves gradually change over time. Given a quasiperiodic function of amplitude over time, the corresponding *correlation function* is a function of period length (time) that gives a measure of how well that periodicity estimate lines up with the regularity of the waveform. Correlation functions are known for being relatively noise-resistant and reliable. Two main drawbacks are its tendency to support hypotheses that are multiples of the true period and the need for a long window of relatively constant frequency [49]. The first drawback is a simple fact of periodicity; if a function is periodic at some time interval T, then it is also period at $2T, 3T, ...$. The second problem, meanwhile, is relatively speech-specific. For speech recordings, very rapid changes in frequency and voice quality are normal, while music recordings tend to have relatively sustained pitch and waveform shape over time.

The greatest problem with correlation function-based pitch detection for our application, however, is their expectation of a *single* quasiperiodic waveform. This effectively disqualifies the autocorrelation function and cross-correlation function for the purpose of multi-pitch estimation. If a waveform is comprised of the harmony of two musical notes, periodic at time intervals T_1 and T_2, then the resulting waveform will not be periodic at either of their periods, unless one is a multiple of the other. For the component frequencies that make up a harmonic sound, this presents no problem because the fundamental frequency F0 will always be the greatest common factor of all its harmonics, but in the case of two different notes sounded each with their respective harmonics present, correlation functions will not produce helpful hypotheses. Still, the algorithms and problems they face have been instrumental in the development of pitch estimation algorithms. This is the topic of Section 6.1.

5.2.1 The Autocorrelation Function

This section follows the description of the autocorrelation function for pitch estimation as given by Lawrence Rabiner in 1977 [42] and David Talkin in 1995 [49]. Rabiner defines the general form of the autocorrelation function of a periodic function $h(t)$ as

$$R(k) = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} h(n)h(n+k), \tag{5.1}$$

where k is a period or *lag* hypothesis, n represents a moment in time, and N is the span of the periodic function used for calculation. Rabiner's definition assumes an infinitely long stream of audio, but we are interested in the *short-time* autocorrelation. Talkin defines the short-time autocorrelation of the i^{th} frame as

$$R_{i,k} = \sum_{j=m}^{m+N-k-1} h_j h_{j+k}, \;\; m = iz, \tag{5.2}$$

where z is the length in samples of the spacing between audio frames, and N is the length of the frame in samples. The window length is usually chosen to cover at least twice the longest expected period of the quasiperiodic function being analyzed. The frame of the recording is also usually calculated according to a window function such as the Hann window (Section 4.4.1). In order to use the autocorrelation function for pitch detection, an algorithm can compute the short-time autocorrelation function of a windowed audio recording, then select the strongest peak. The period hypothesis can also be weighted by period, helping to avoid choosing higher multiples of the true F0's period.

Figure 5.2a shows the autocorrelation function spectrogram calculated with a 2048-sample (46 ms) window calculated over a 10 second recording of a flute. Different moments in time have different contrast of peaks in the autocorrelation function. This variation in strength of period hypothesis peaks over time shows the autocorrelation function's relative sensitivity to changes in source loudness. In this case, the variations arise from the flautist's use of tremolo. Figure 5.2b shows the calculation of the strongest

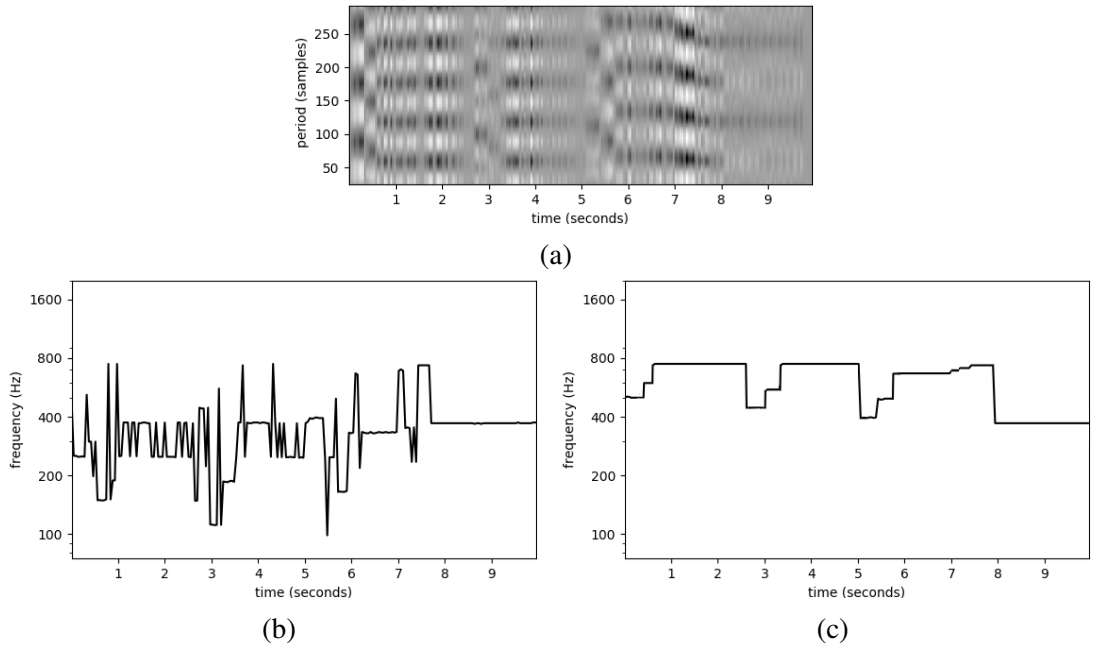


Figure 5.2: Results of the autocorrelation function on Telemann's Fantasia No. 3 in B minor performed on flute, using a window size and spacing of 2048 samples. (a) the autocorrelation spectrogram. (b) The strongest F0 hypotheses extracted from the autocorrelation function. (c) Ground-truth pitches.

F0 hypotheses over time according to the autocorrelation function in Figure 5.2a. Frequent jumps to half or one-third the actual frequency are a result of the periodicity at twice or three times the true F0's period.

5.2.2 Normalized Cross-Correlation and RAPT

RAPT [49] seeks to address the problems evident in Figure 5.2. It is based on the *normalized cross-correlation function*, which makes two major modifications to the autocorrelation function. One major weakness is its changing sensitivity based on the lag hypothesis used. To accurately correlate functions with long lags (very low frequencies), the frame must be sufficiently wide, but if a constant frame length is used, the autocorrelation of higher frequencies and shorter lags ends up being unnecessarily long, meaning greater sensitivity to pitch variation. Meanwhile, long lags use fewer occurrences of the

waveform and thus are more sensitive to noise. Cross-correlation solves this by defining its summation independent of the lag hypothesis:

$$R_{i,k} = \sum_{j=m}^{m+N-1} h_j h_{j+k}, \;\; m = iz. \tag{5.3}$$

Again N is the length of the frame in samples. As a result, there is flexibility for the summation to wander outside the bounds of the frame in the name of keeping the summation more frequency agnostic.

The second major change is normalization. Talkin [49] defines the normalized cross-correlation as

$$R_{i,k} = \frac{\sum_{j=m}^{m+N-1} h_j h_{j+k}}{\sqrt{\sum_{j=m}^{m+N-1} h_j^2 \cdot \sum_{j=m}^{m+N-1} h_{j+k}^2}}, \;\; m = iz, \tag{5.4}$$

again where N is the frame width [49]. Thus, for every frame of the recording that is processed by the cross-correlation function, it is normalized according to the total power across that frame and the total power across the frame offset by the lag hypothesis. Figure 5.3 shows the contrast and consistency of the peaks in the normalized cross-correlation function as compared to the autocorrelation function in Figure 5.2. Unlike results from the autocorrelation function, results from normalized cross-correlation show strong contrasts in period hypothesis strength even when the recording loudness decreases. As a result, somewhat more consistent and reliable frequency hypotheses are generated, but it is still not totally successful (Figure 5.3c).

Finally, RAPT provides a system for an end-to-end strategy for voice pitch estimation that takes into account robustness and efficiency. One goal of RAPT that does not apply directly to automatic music transcription is the detection of unvoiced sounds. To this end, it uses multiple passes at varying sampling rates, where the first is designed only to demarcate regions without strong pitch hypothesis, so the higher-resolution pass does not waste time. The other strategies that the algorithm employs, however, can be

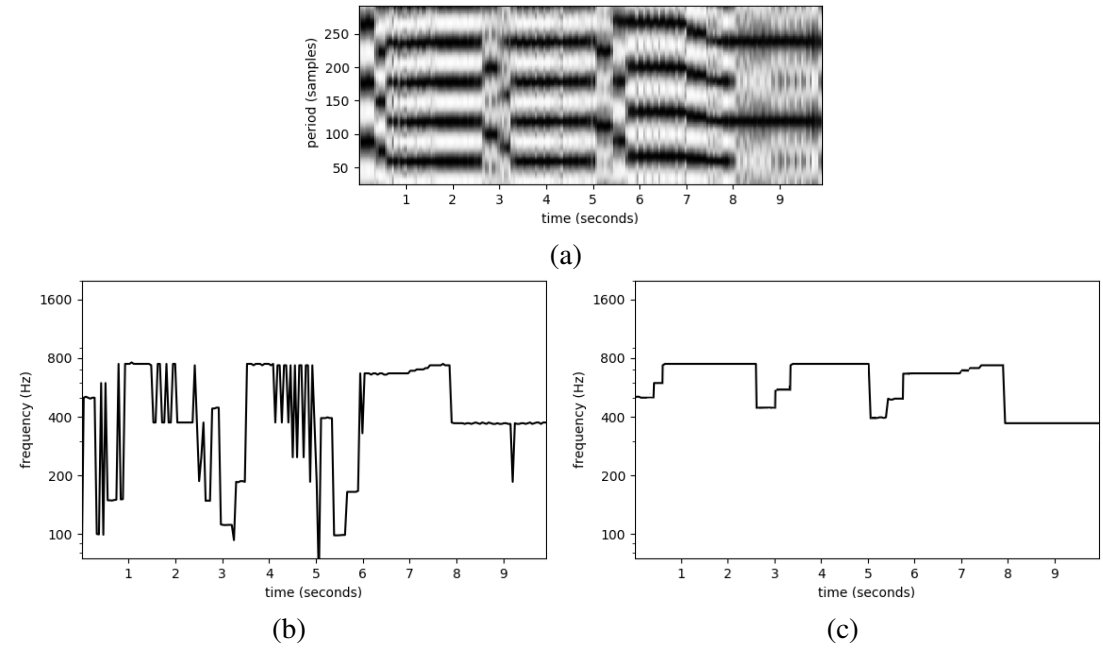


Figure 5.3: Results of the normalized cross-correlation function on Telemann's Fantasia No. 3 in B minor performed on flute, using a window size and spacing of 2048 samples. (a) The normalized cross-correlation spectrogram. (b) The strongest F0 hypotheses extracted from the normalized cross-correlation function. (c) Ground-truth pitches.

applied to any pitch estimation algorithm. These are (1) F0 selection using dynamic programming, and (2) amplitude-based detection of onset and offset time. The first is just one example of a strategy for choosing between competing F0 hypotheses, since correlation functions inherently create ambiguities at multiples of the true F0's period. The second is a valuable technique used for note tracking in automatic music transcription.

5.3 Spectrum-Based Pitch Estimation

Aside from period-based pitch estimation techniques, there is another class of pitch estimation techniques that begin with the power spectrum of a recording. The spectrum is extracted using a short-time Fourier transform of a small window of the recording, as described in Section 4.4.1. Instead of attempting to find frequency based on the shape and periodicity of the curve in the time domain, we turn to the frequency domain.

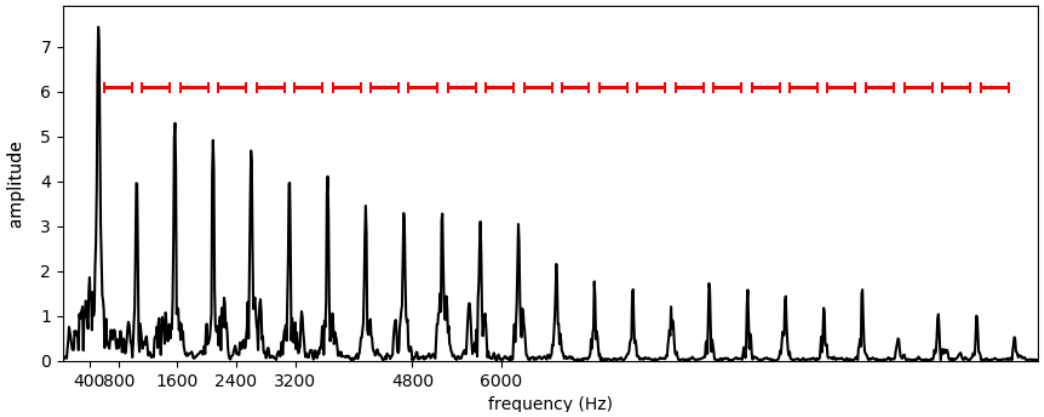


Figure 5.4: The equally-spaced harmonic frequencies generated by a violin note.

Within the frequency domain, the challenge of pitch estimation is to make sense of a pattern of harmonics of the base frequency, generated as described in Section 2.1.4. The harmonicity principle, which states that a frequency's harmonics are all multiples of the fundamental frequency, hence evenly spaced along the frequency spectrum, causes an interesting phenomenon—the spectrum of a single note is itself periodic, where the fundamental frequency is the period. This is demonstrated in Figure 5.4. All spectrum-based approaches take advantage of this harmonic pattern in some way.

One question that arises immediately is, once the frequency-domain spectrum is generated, why not just choose the spectral peaks with the greatest amplitude or the lowest significant spectral peak? Figure 5.5 shows the first significant spectral peaks of a recording's frequency domain, with a box drawn around the fundamental frequencies. These could potentially be selected with a combination of thresholding and weighting spectral peaks by frequency. Thresholding is necessary to rule out local maxima that may be a result of noise (either from the source or from the recording). Meanwhile, weighting by frequency helps prevent harmonics being chosen over their fundamental frequency. However, because of variations in timbral quality between different voices and between different instruments, the fundamental frequency is not reliably the

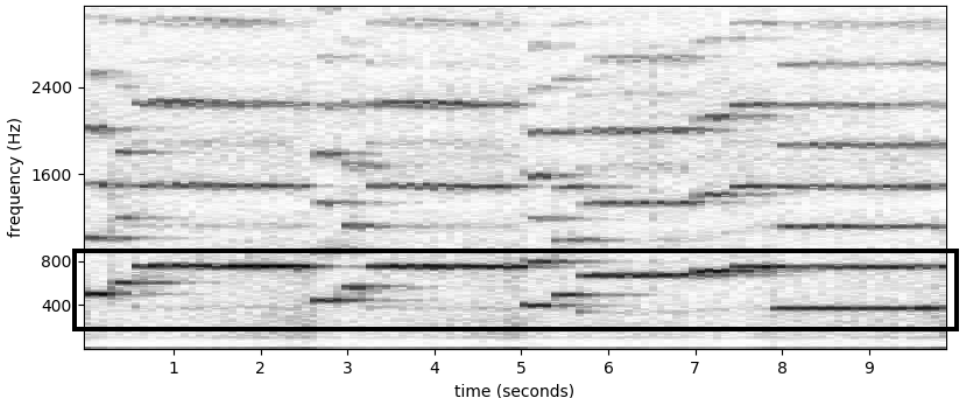


Figure 5.5: The peaks in a musical recording's spectrogram show the true F0s.

strongest spectral peak, and in fact may be of a much lower amplitude than the first or second harmonic. Figure 5.6 shows erroneous pitch hypotheses from harmonics. Periodicity of the spectrum is simply a much more reliable pattern for determining the fundamental frequency that generated all of the apparent harmonics. In multi-pitch estimation, there are too many coinciding peaks, so that the use of periodicity in the spectrum is necessary to distinguish between multiple harmonics of one sounded note and multiple sounded notes' fundamental frequencies. This is the subject of Section 6.1.

5.3.1 Cepstrum

The so-called *cepstrum* (a play on words, reversing the first four letters of the word *spectrum*[1]) was first defined by Bruce Bogert in 1963 as the spectrum of the log spectrum [3]. It was first used for pitch estimation in A. Michael Noll's 1967 paper "Cepstrum pitch detection" [35]. This section follows Talkin's explanation of the cepstrum [49]. We can recall that the short-time log-power spectrum of a sampled audio function h_t is

[1]Bogert actually went much further and coined an entire vocabulary of similar words, such as *quefrency* (the domain of the spectrum, frequency of frequencies) and *saphe* (the phase of the cepstrum). Only the term cepstrum has stood the test of time.

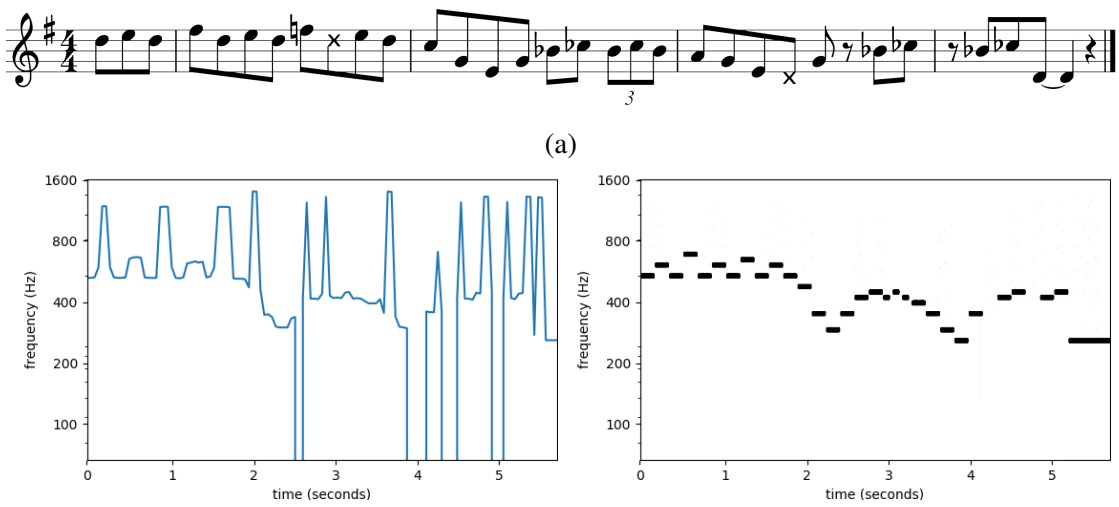


Figure 5.6: (a) A hand-transcription of an improvised solo by Chet Baker on trumpet. Below, a comparison of F0 hypotheses (b) generated by choosing the highest-amplitude frequency in the Fourier power spectrum and the true F0s (c). The naive use of the power spectrum shows frequent incorrect jumps to double, triple, or quadruple the true F0 due to high-amplitude harmonics.

defined as $\ln ||H_k||$, or

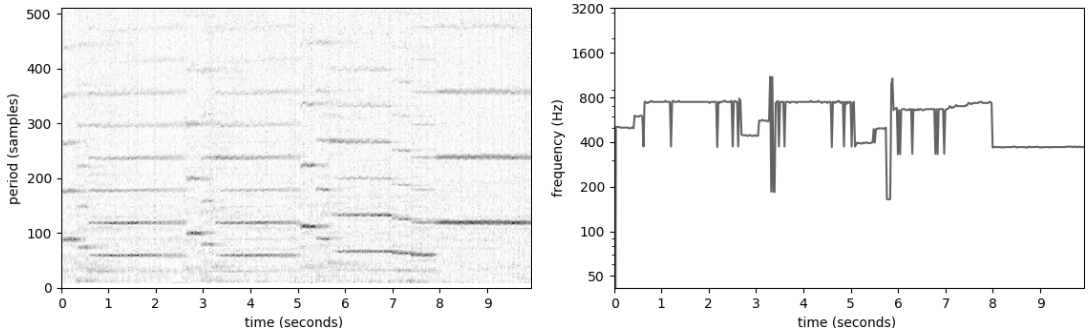
$$L_k = \ln || \sum_{r=0}^{N-1} h_r e^{j2\pi rk/N} ||, \tag{5.5}$$

where N is the number of samples in the frame (see Sections 4.2 and 4.4.2). The cepstrum, in turn, is defined as

$$c_k = \frac{1}{N} || \sum_{r=0}^{N-1} L_r e^{j2\pi rk/N} ||. \tag{5.6}$$

The use of the $\log$ function, as described in Section 4.4.2, yields more even amplitudes of spectral peaks, which in the case of the cepstrum means more clearly defined periodicity and cepstral peaks. The domain of the cepstrum is the frequency of the spectral peaks in the spectrum, or cycles per Hz (i.e., seconds), but the goal is the period of these spectral peaks, which, by the harmonicity principle, is equal to F0. To find this, we just choose the best candidate of cepstrum and compute its inverse, because for a wave of period T, the wave's frequency f is given by $f = 1/T$.

Figure 5.7 illustrates the use of the cepstrum to find pitch hypotheses. Like the



(a) (b)

Figure 5.7: The cepstrum of Telemann's Fantasia No. 3 in B minor. A window size of 2048 samples and spacing of 1024 samples was used. (a) The cepstrum spectrogram. (b) The strongest F0 hypotheses extracted from the cepstrum.

autocorrelation function, the cepstrum is prone to finding pitch hypotheses at multiples of the true period, hence fractions of the true fundamental frequency. This arises because of the sharpness of peaks in the spectrum and the consequent presence of harmonics in the cepstrum. This is clearly shown in Figure 5.7b. The cepstrum is also known for being prone to errors in noisy recordings, compared with the autocorrelation function [28]. Still, the notion of the cepstrum and its basic principles have been widely influential in contemporary pitch estimation algorithms. One of the cepstrum's merits is its elegant rendition of the harmonic patterns of sounds as the spectrum of a spectrum.

5.3.2 Autocorrelation of Log Spectrum

In 1996, Kunieda et al. took the concept of the cepstrum, but instead processed the log power spectrum with the autocorrelation function in what they called "ACLOS – AutoCorrelation of LOg Spectrum" [28]. It is defined as

$$R_k = \frac{1}{N} \sum_{j=0}^{N-1} L_j L_{j+k}, \tag{5.7}$$

where L_k is defined according to Equation (4.16), and N is the highest relevant frequency in the spectrum. Like the cepstrum, it seeks the regularity of spectral peaks in

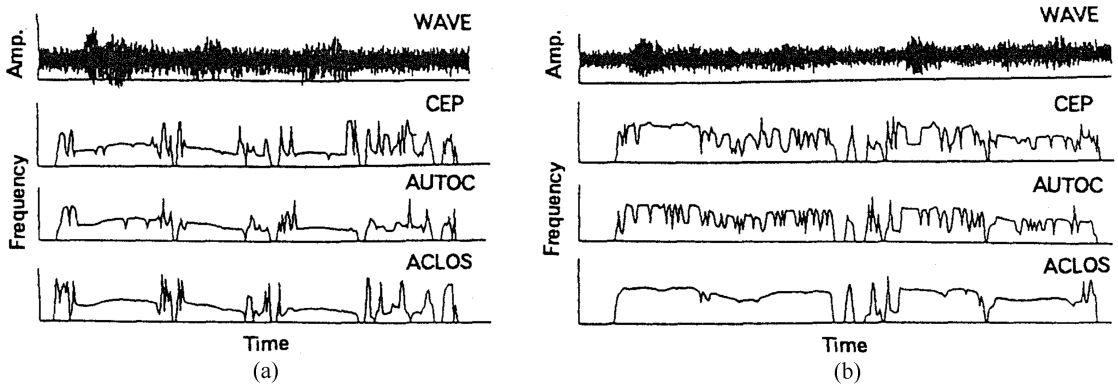


Figure 5.8: A comparison of ACLOS against cepstrum (CEP) and autocorrelation function (AUTOC) on noisy speech recordings for a male (a) and a female (b) speaker. The cepstrum and autocorrelation pitch curves on (b) show frequent jumps to half to true F0, a common problem with these pitch estimation algorithms [28].

the power spectrum, but by using the autocorrelation function it actually finds directly the period of the harmonic peaks, hence the fundamental frequency.

The relative reliability of ACLOS in cases of extreme noise when compared with the autocorrelation function and cepstrum is shown in Figure 5.8. Of note is its resistance to generating hypotheses at one-half the true F0 that are particularly evident in the results shown in Figure 5.8b. This is due to its strong, regular peaks from which a reliable F0 hypothesis can be readily extracted.

5.3.3 (In)harmonicity-Based Pitch Estimation

This final category of pitch estimation algorithms is effective for both single and multi-pitch estimation. The strategy, employed by both Klapuri [26] and Pertusa and Iñesta [39], is based on the same idea as cepstrum and ACLOS, the harmonicity principle, but rather than finding sinusoidal correlation in the entire harmonic curve, these *harmonicity-based* algorithms isolate local peaks in the Fourier spectrum and match peaks that could be harmonics of different fundamental frequency hypotheses. For a given fundamental

frequency f_0, its h^{th} harmonic frequency is given by

$$f_h = hf_0 \pm f_r, \tag{5.8}$$

where f_r captures the possible deviation of frequency due to *inharmonicity* [39]. Klapuri specifies possible harmonic deviation according to

$$f_h = hf_0\sqrt{1 + (h^2 - 1)\beta}, \tag{5.9}$$

where β is the *inharmonicity factor* [26]. For experiments in this thesis, Pertusa and Iñesta's simpler treatment of inharmonicity was used with a constant f_r. If there are multiple peaks in the range of $\pm f_r$, the highest-power peak is used.

Pertusa and Iñesta use the power spectrum, given by

$$P(k) = ||H(k)||, \tag{5.10}$$

where $H(k)$ is the complex Fourier transform. From the power spectrum, all peaks above a certain power threshold μ are extracted, as shown in Figure 5.9. By reducing the curve in Figure 5.9a to the sparse peaks in Figure 5.9b, computations can be significantly limited, while inharmonicity can be more easily dealt with. From these peaks, all frequencies within a specified range $[f_{min}, f_{max}]$ are considered as potential fundamental frequencies. For a given fundamental frequency candidate c with fundamental frequency f_c, its harmonic pattern is specified:

$$p_c = \{P(k_{c,1}), P(k_{c,2}), P(k_{c,3}), ..., P(k_{c,H})\}, \tag{5.11}$$

where $k_{c,h}$ is the index in the discrete Fourier transform of hf_c specified in Equation (5.8), and H is the highest harmonic index tested. Then, the frequency's total loudness is defined as

$$l(c) = \sum_{h=1}^{H} P(k_{c,h}) = \sum_{h=1}^{H} P(hk_c \pm k_r). \tag{5.12}$$

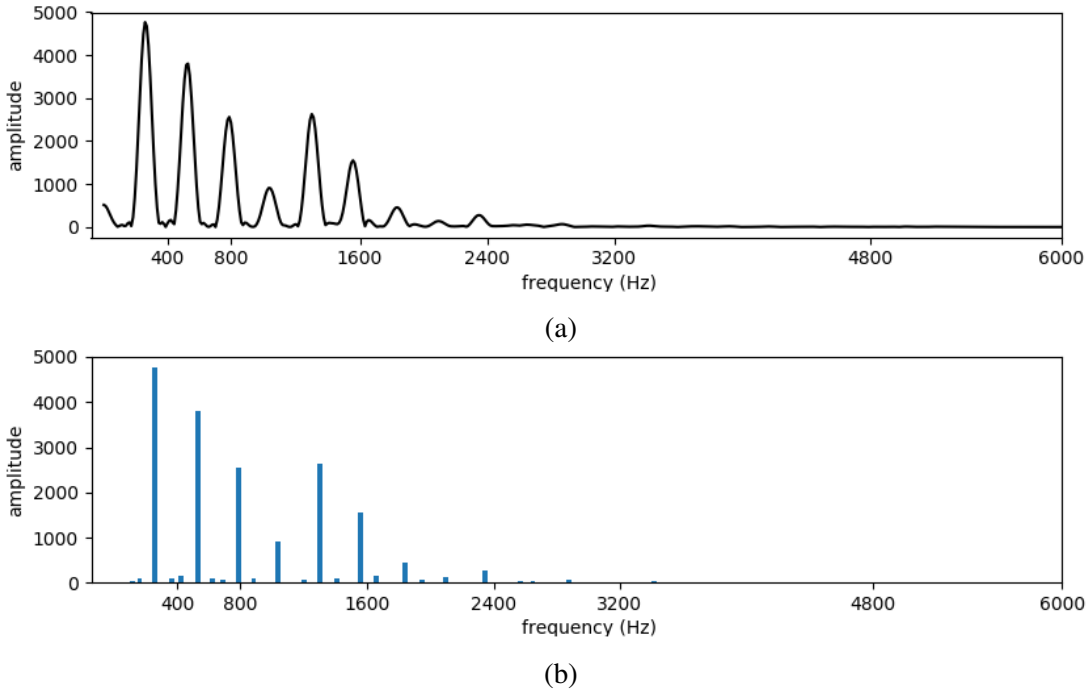


Figure 5.9: (a) The power spectrum of a trumpet note and (b) its extracted spectral peaks, using a minimum amplitude threshold of $\mu = 10$.

Using this algorithm in a single-pitch application, frequency hypotheses can be weighted based on their total loudness given by Equation (5.12) and choosing the highest-weight frequency hypothesis.

Figure 5.10 shows the frequency hypotheses using this harmonicity-based loudness model compared with ground truth frequencies. For this experiment, I used $\mu = 10$ and $f_r = 25$ Hz and set the range of potential fundamental frequency hypotheses to $[f_{min}, f_{max}] = [50 \text{ Hz}, 2100 \text{ Hz}]$, which captures the notes from G1 to C7. It can be seen that the algorithm accurately extracts all fundamental frequencies in both recordings. Further, the results on the second recording (Figure 5.10c) correctly generate no hypothesis when there is not note being played, at around four seconds. In the longer notes of Figure 5.10a, some slight variation in pitch is visible, which is true to the recording and still within the frequency range of the target notes.

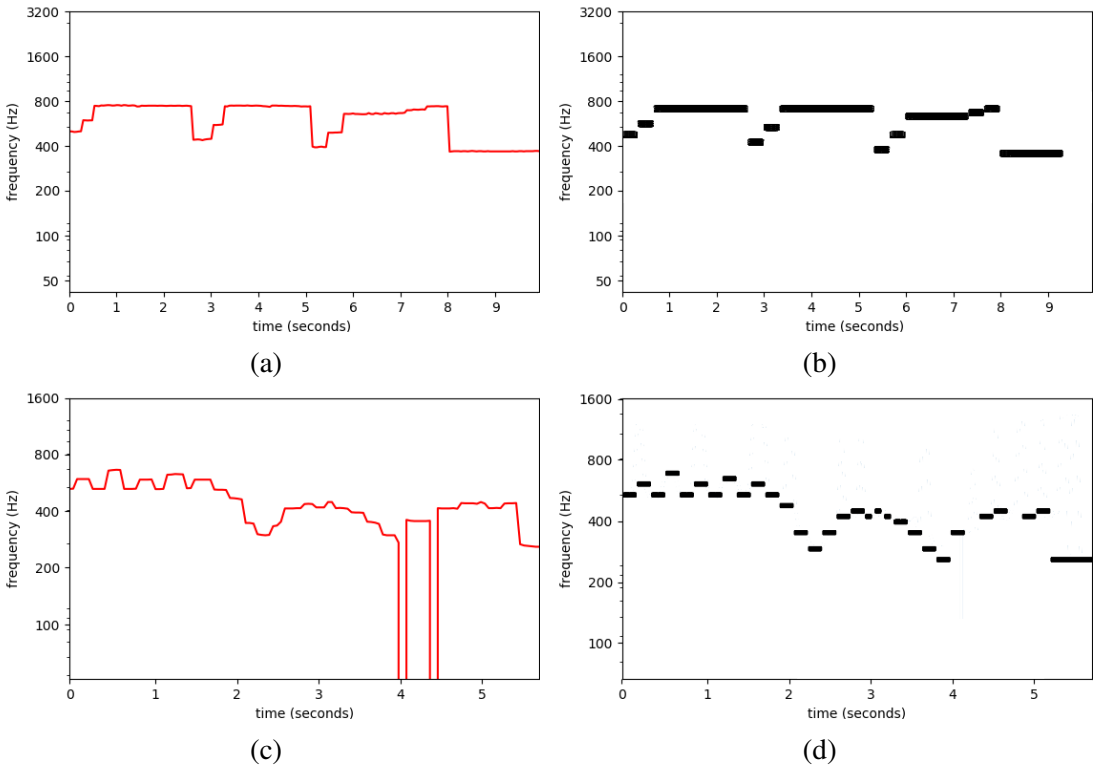


Figure 5.10: Pitch hypotheses generated from harmonicity-based loudness weighting on (a) Telemann's Fantasia No. 3 in B minor and (c) an improvised trumpet solo. (b) and (d) are the respective ground-truth pitches.

CHAPTER 6

MULTI-PITCH ESTIMATION

This chapter covers the tasks and challenges of pitch estimation with multiple fundamental frequencies, primarily through the example of Antonio Pertusa and José Iñesta's 2008 paper "Multiple fundamental frequency estimation using Gaussian smoothness" [39]. Section 6.1 explains what makes this problem more difficult than the single-pitch version described in Chapter 5. Next, Section 6.2 describes different methods for spectral pattern modeling, the most important task unique to multi-pitch estimation. Section 6.3 details the process of hypothesis generation through the example of Pertusa and Iñesta's algorithm. Finally, Section 6.4 discusses results of experiments on different piano recordings using both iterative and joint versions of Pertusa and Iñesta's algorithm.

6.1 From Monophonic to Polyphonic

The algorithms in the previous chapter are based on the assumption that there is only one pitch sounded at any given moment, the *monophonic* case, but most music is *polyphonic*, meaning many notes are played at once. This could be a single performer playing multiple notes at once on a polyphonic instrument such as a piano or guitar, or it could be an ensemble performance with multiple musicians playing together on different instruments. A variation of instruments present in a recording can make the task of distinguishing overlapping notes even more challenging than polyphonic recordings on a single type of instrument.

In the single-pitch case, the problem is characterized as trying to find the most likely fundamental frequency hypotheses from a frame of sampled audio, using either correlation- or spectrum-based pitch estimation algorithms. To find a set of several fundamental frequencies that have generated a frame of sampled audio turns out to be a much more difficult task. For one, correlation-based pitch estimation algorithms are al-

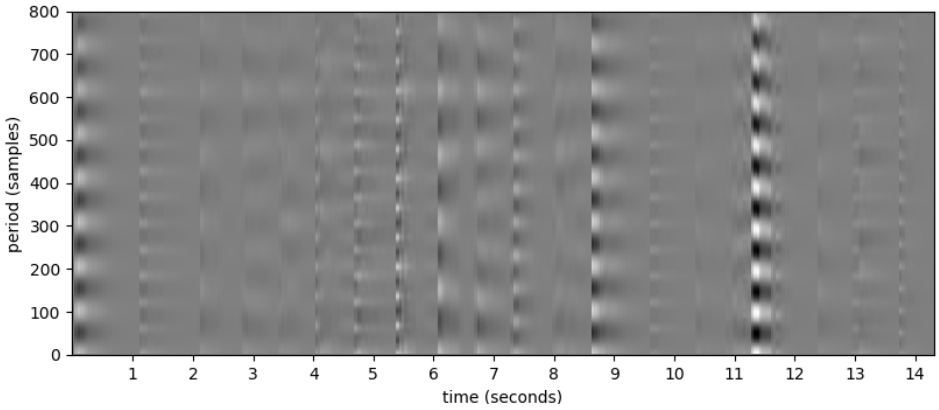
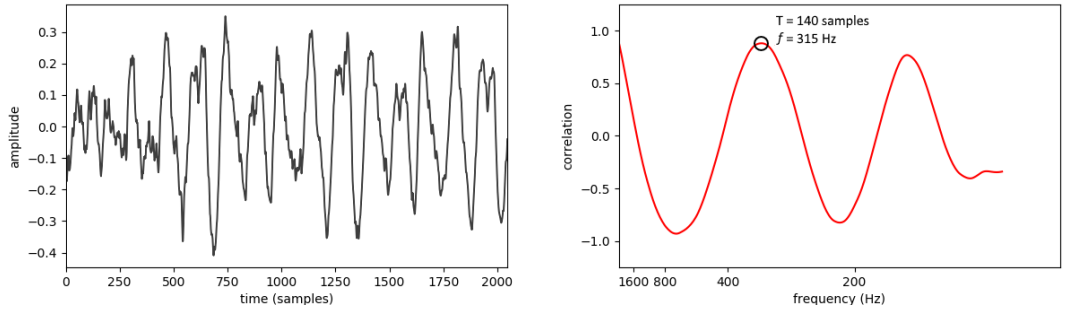


Figure 6.1: The autocorrelation function applied to a recording of Frédéric Chopin's Nocturne in E-flat major, Op. 9, No. 2. Moments with relatively visible autocorrelation ripples only occur when one note is played at a time.



(a) (b)

Figure 6.2: (a) The waveform of a C major chord played on a piano. (b) The autocorrelation function of the chord. The first peak is at $T = 140$ samples, which would suggest a fundamental frequency of $f_0 = 315$ Hz, a frequency that does not appear in any of the notes of the chord.

most entirely fruitless on polyphonic audio, as shown in Figure 6.1. Only brief moments in the recording where one note is played at once give particularly strong correlation hypotheses. Meanwhile, Figure 6.2 shows what is happening in the apparently gray regions. Attempting to use the autocorrelation function of a C major chord (C4, E4, and G4) gives a meaningless hypothesis of period $T = 140$ samples, i.e., $f_0 = 315$ Hz (derived from $f = \Delta/T = 44100/140 = 315$ Hz), just below E4 = 329 Hz (Figure 6.2b). Because autocorrelation highlights repetition and periodicity, if the notes were perfectly aligned, the best hypothesis would depend on the smallest time interval where all three

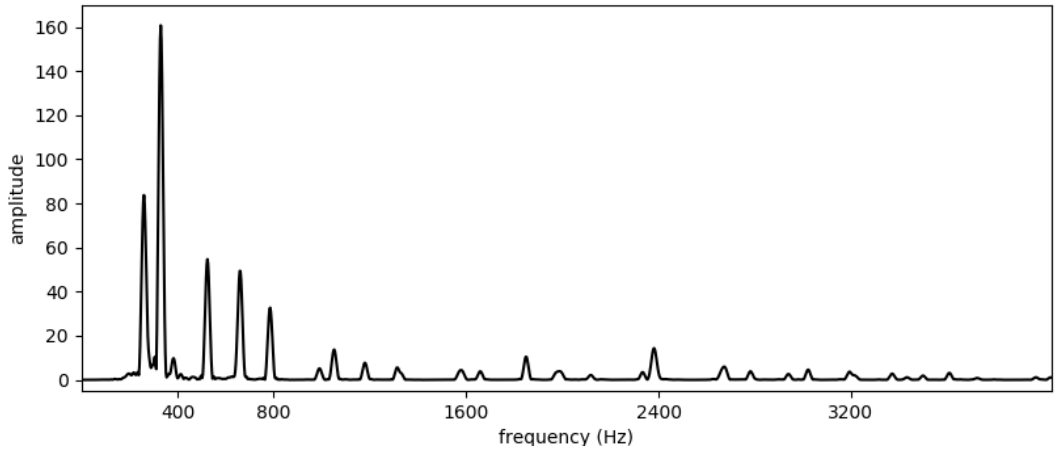


Figure 6.3: Fourier spectrum of a C major chord. Unlike the spectrum of a single-note recording, there is no immediately obvious pattern to the spectral peaks.

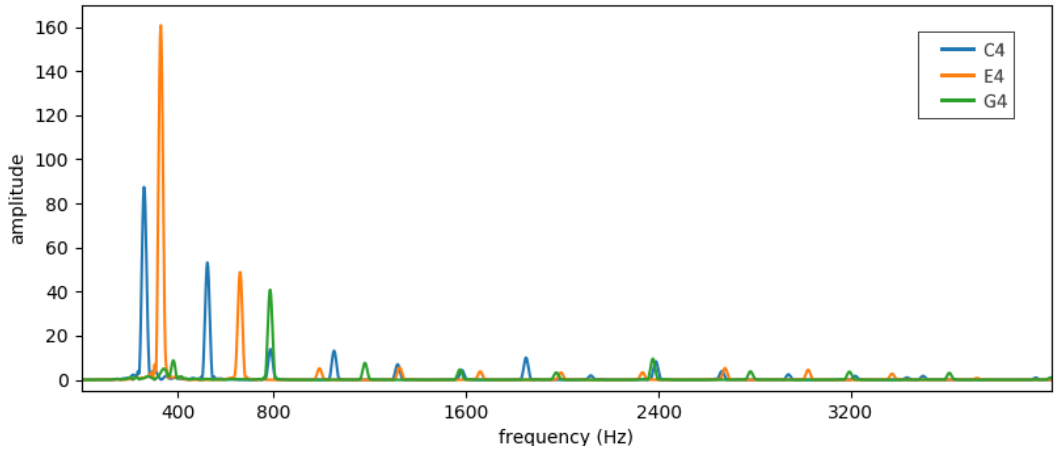


Figure 6.4: The spectra of a C major chord split into its three component notes.

notes are periodic, but ultimately the autocorrelation function will be dominated by the fundamental frequency with the highest relative loudness, which in the case of Figure 6.2 is E4.

There is more potential in spectrum-based pitch estimation algorithms. Figure 6.3 shows the spectrum of the same C major chord. Unlike a single note's spectrum, there is no immediately apparent even spacing of the spectral peaks. This rules out the cepstrum and ACLOS approaches. However, the spectral peaks are actually just the sum of the spectral peaks in the contributing notes of the chord—C4, E4, and G4. Figure 6.4 shows the same chord, but with the spectral peaks colored by the note that produced them.

This reveals the underlying periodicity of each note's peaks that sums to generate the spectrum we see in Figure 6.3. The goal of all multi-pitch estimation systems is to extract from a spectrum like Figure 6.3 its contributing notes based combinations of note candidates and assumptions of those notes' spectral patterns. Based on these peaks, a harmonicity-based approach like that described in Section 5.3.3 can be employed.

6.2 Spectral Modeling

Given a Fourier spectrum of a windowed frame of audio, the primary goal of multi-pitch estimation systems is to label spectral peaks according to their source note, as in Figure 6.4. In general, if we assume a signal model like that proposed by Klapuri [26], our input spectrum can be described as a function of frequency $X(k)$,

$$X(k) = H(k)S(k) + N(k), \tag{6.1}$$

where k is the frequency index in the Fourier transform, $S(k)$ is the frequency output of the musical instrument sources, and $H(k)$ is the *frequency response* of the recording, or how well different frequencies are picked up from room acoustics, microphones, and other factors [26], and $N(k)$ is the additive noise in the recording. While Klapuri [26] uses spectral preprocessing to extract an approximation of $S(k)$, many algorithms simply use spectral peaks above a threshold to avoid noise and assume the effects of $H(k)$ are minor enough to ignore. For multi-pitch estimation, we can then break up $S(k)$ into its component frequencies, as

$$S(k) = \sum_{c=1}^{F} S_c(k), \tag{6.2}$$

where c refers to an individual source note, and F is the total number of overlapping notes. Each of these hypothetical source note shapes $S_c(k)$ will have its own set of frequencies and spectral peaks. The goal of spectral modeling is, given $X(k)$, to estimate each of the component source spectral shapes and ultimately provide fundamental

frequency hypotheses for each candidate c. This section covers different strategies for estimating the component spectral shapes for a given F0 hypothesis, namely modeling through the *spectral smoothness principle* in Section 6.2.1 and estimation through *learned models* in Section 6.2.2.

In both cases, there is an overarching challenge that demands accurate modeling of the shapes of the individual-note spectra $S_c(k)$: shared harmonics. Based on the harmonicity principle, if we hypothesize one contributing note has a fundamental frequency f, we can infer that the peaks at each of its harmonics hf, for $h = 1, 2, ..., H$, will also be part of that note's spectral shape. If we hypothesize another note with fundamental frequency f', and it should happen that $f' = nf$, we would like to recognize that, though f' will have spectral peaks at many of its harmonics, it is more likely that those peaks are all in fact generated by our previous hypothesis, f. To avoid such erroneous hypotheses, we subtract f's harmonics from the original source spectrum $S(k)$, either in a greedy, iterative fashion (Section 6.3.1) or with a joint energy-minimization approach (Section 6.3.2). But what if the source spectrum $S(k)$ actually was generated by two such frequencies f and f'? Or, in a more likely scenario, what if $S(k)$ was generated by two frequencies f and f'', where $hf = h'f''$ for some harmonics h and h'? This scenario is shown in Figure 6.5. The purpose of spectral modeling is, then, more precisely, to subtract harmonics of a winning F0 hypothesis from the source spectrum $S(k)$ in an informed way that preserves shared harmonics but avoids false positives from harmonics that actually belong only to one F0 hypothesis.

6.2.1 Spectral Smoothness

The simpler of the two categories of spectral modeling covered in this section is smoothness-based modeling. Klapuri defines spectral smoothness as "the expectation that the spectral envelopes of real sound sources tend to be continuous" [25]. In other words,

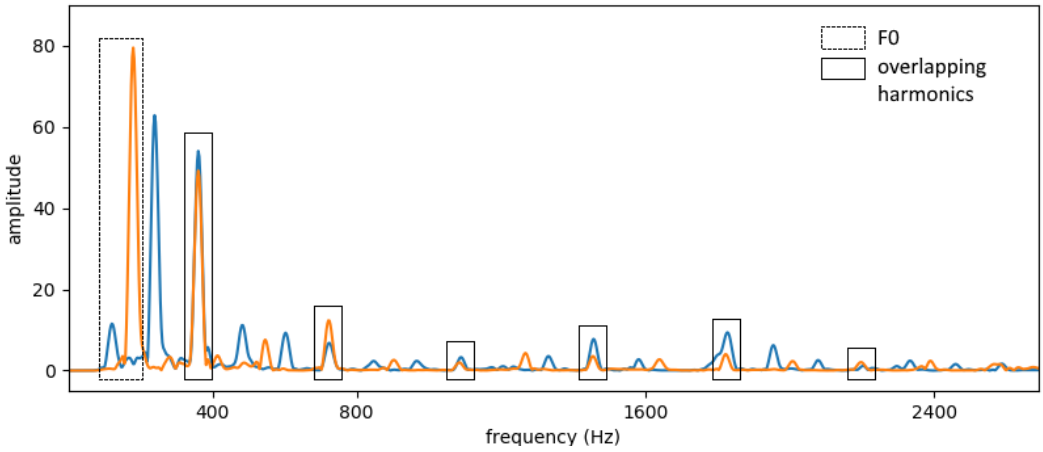


Figure 6.5: The spectra of two notes, C3 and G3, overlaid to show the overlapping harmonic partials, boxed. Because C and G are a perfect fifth apart, every third harmonic of C will match every second harmonic of G.

it is rare to find large jumps in amplitude from one harmonic to the next. There is also a related assumption that as a note's harmonics increase in frequency, they decrease in amplitude. Based on this smoothness principle, harmonic amplitudes that appear to be unsmooth can be presumed as actually resulting from the combined harmonics of multiple notes. Smoothness can also be used as a factor in an energy-minimization approach, where hypotheses are punished for having unsmooth jumps in amplitude between neighboring harmonics.

In Pertusa and Iñesta's algorithm, two different smoothed spectra are used [39]. First, a linearly interpolated spectrum is computed for a given note hypothesis. For each harmonic in the candidate's spectral shape, if it is higher in amplitude than the amplitude given by linearly interpolating its harmonics on either side, the remainder is assumed to be a harmonic generated from another note. The interpolated value then gets used in that candidate's updated spectral shape:

$$p_{c,h} = \min(p_{c,h}, \frac{p_{c,h-1} + p_{c,h+1}}{2}), \tag{6.3}$$

where $p_{c,h}$ is the amplitude of candidate c's h^{th} harmonic. This is demonstrated in Figure 6.6. This is the extent to which a spectral smoothness model is used for separation

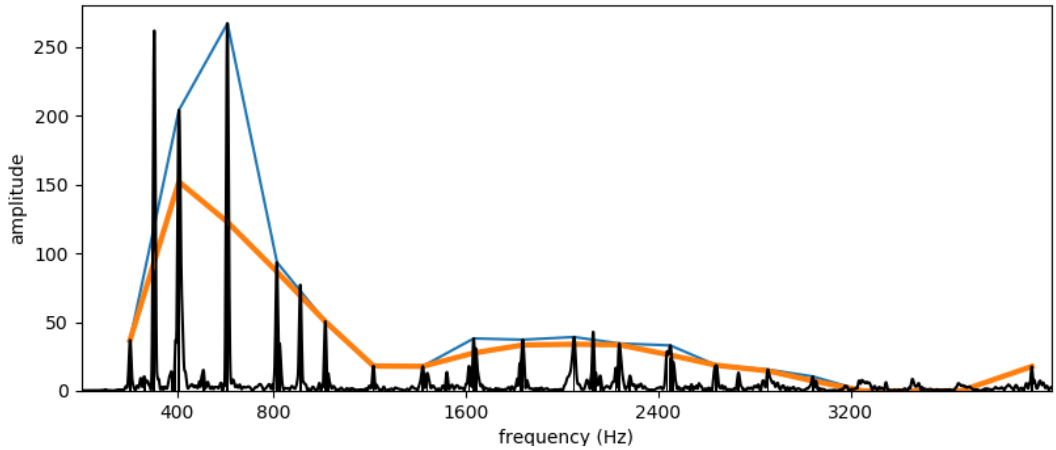


Figure 6.6: The spectrum of two notes, G3 = 196 Hz and D4 = 293 Hz, played simultaneously on a violin. The thin blue curve shows G3's original spectral shape, while the thick orange curve shows G3's spectral shape smoothed using interpolated amplitudes according to Equation (6.3).

and labeling of spectral peaks in Pertusa and Iñesta's algorithm, but a more involved smoothness model is also employed in evaluating different candidate's saliences. For a given candidate c and its spectral shape p_c, first a smoothed shape $\tilde{p}_c$ is generated by

$$\tilde{p}_c = \mathcal{N} * p_c, \tag{6.4}$$

where $\mathcal{N}$ is a Gaussian distribution window and $*$ is the convolution operator. Because the spectral shape is relatively small—no more than 20 harmonics—a Gaussian window of $\mathcal{N} = [0.21, 0.58, 0.21]$ is used. The smoothness $\sigma(c)$ of the original pattern p_c is then calculated as $1 - \bar{s}(c)$, where $\bar{s}(c)$ is the normalized sharpness of p_c, the difference between p_c and the smoothed $\bar{p}_c$:

$$\sigma(c) = 1 - \frac{1}{H} \sum_{h=1}^{H} |\tilde{p}_{c,h} - p_{c,h}|, \tag{6.5}$$

where H is the number of harmonics in the shape p_c.

Klapuri has developed similar smoothness models for candidate separation [25]. The first is reminiscent of Pertusa and Iñesta's simple interpolated spectrum technique, but instead of replacing a harmonic's amplitude with the interpolated amplitude between its

two neighbors, the amplitude is replaced with the minimum of its amplitude and its next neighbor's amplitude:

$$p_{c,h} = \min(p_{c,h}, p_{c,h+1}). \tag{6.6}$$

An alternative smoothing method proposed by Klapuri is more akin to Pertusa and Iñesta's second smoothing method, but instead of using a Gaussian window, Klapuri uses a Hamming window. This smoothed curve is generated by

$$\tilde{p}_c = \mathcal{H} * p_c, \tag{6.7}$$

where $\mathcal{H}$ is an octave-wide Hamming window. Note that because $\mathcal{H}$ is defined in terms of the octave, it will include more neighboring harmonics as h increases. Then, like the above method, the amplitude of a harmonic is replaced with the minimum of its original amplitude and its smoothed amplitude:

$$p_{c,h} = \min(p_{c,h}, \tilde{p}_{c,h}). \tag{6.8}$$

6.2.2 Learned Note Models

Smoothness-based models can be very effective, but they are only ever loose approximations. If, however, more precise spectral models based on the actual spectral patterns of specific instruments are used, not only is accuracy improved, but also an entire new joint estimation approach based on non-negative matrix factorization can be used. This section follows the modeling techniques described in Yin et al. [55] and Smaragdis and Brown [47].

The instrument model technique provided by Yin et al. [55] is relatively simple and relatively compatible with Pertusa and Iñesta's algorithm. In Yin et al., before any processing is done on the target audio recording, a sample note is analyzed to create an *instrument feature vector*, I. This vector I captures the harmonic content of an individual note on a given instrument. Then, for any note candidate, this vector can be scaled

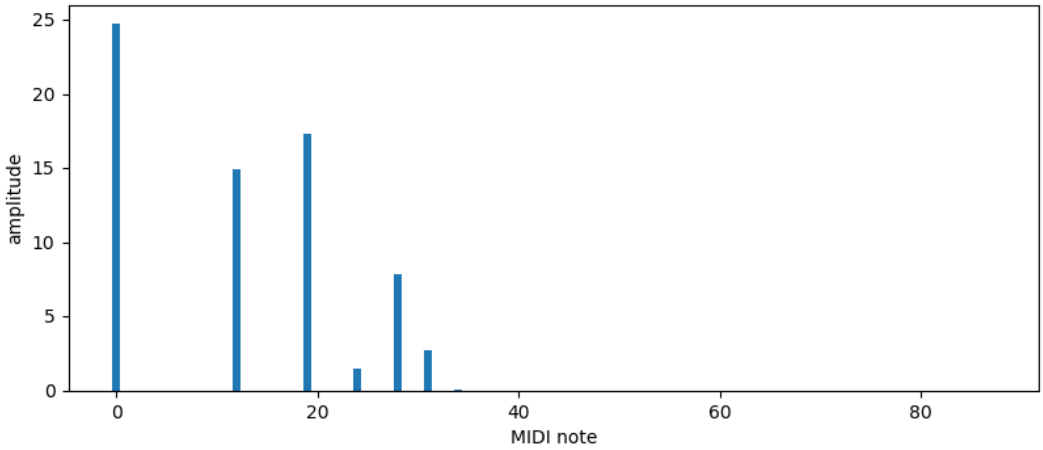


Figure 6.7: The instrument feature vector for a trumpet, extracted from a recording of a trumpet C4, based on Yin et al.'s approach [55].

and shifted in the frequency domain—to match the target fundamental frequency—and scaled in the amplitude domain to match the note's loudness. Yin et al. use a spectrum Z indexed by semitone, with $i \in [\text{A0}, \text{C8}]$, not frequency, like the constant Q spectrum described in Section 4.5. This makes the job of scaling I for different notes much easier, because shifting all spectral amplitudes by a constant value will retain the logarithmic spacing of the harmonics. For a candidate c which has pitch n semitones from A0, the shifted note's spectrum Z_c will simply be

$$Z_{c,i} = aI_{i-n}, \tag{6.9}$$

for all $i > n$ where a is the amplitude scalar [55]. Figure 6.7 shows an instrument vector generated from a trumpet C4.

There are two major drawbacks to this approach. The first is that an instrument's spectral shape changes both over time and also for different fundamental frequencies. This variation is explained in depth in Section 2.1.4, but the important takeaway is that one instrument feature vector I cannot fully capture every note and every moment in its envelope. A partial solution to this problem is to develop a set of instrument feature vectors for different fundamental frequencies. This is the spirit of the approach that Smaragdis and Brown use, building a matrix of basis vectors for each different note

[47]. But, in this approach, the second problem becomes even more significant: there must be reliably-labeled training data for finding these feature vector(s). In Yin et al.'s case, these could be general approximations of common instruments' spectral patterns that are reused for any recording, as long as they are close enough—it will still be more reliable than the more simplistic smoothness principle. But in Smaragdis and Brown, the expectation is several seconds of labeled audio for each recording to extract this matrix of basis vectors. This enters the territory of semi-automatic music transcription and scales poorly for large datasets of untranscribed audio.

6.3 Hypothesis Generation and Scoring

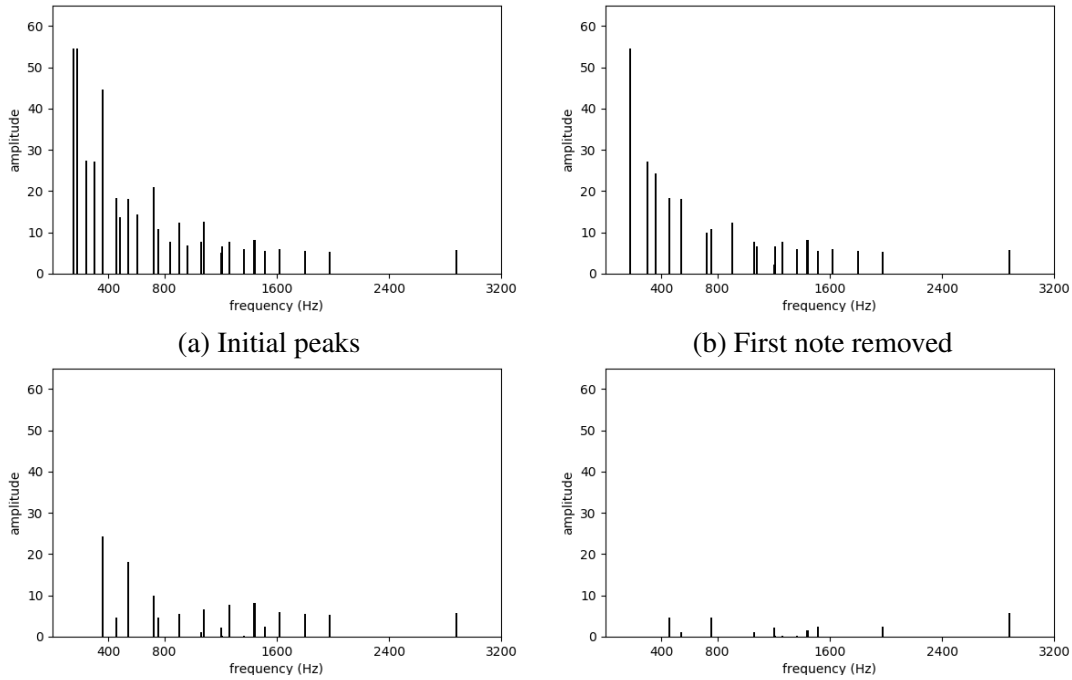
This section describes the two larger categories of multi-pitch estimation systems, *iterative* and *joint*, through the example of Pertusa and Iñesta's 2008 multi-pitch estimation algorithm [39]. The iterative-joint distinction was articulated by Yeh [54], but Benetos has since rejected the dichotomy, preferring to distinguish algorithms based on their "core techniques" [2]. For the sake of simplicity, this thesis will follow Yeh's categorization in the case of what Benetos calls *feature-based* algorithms, of which Pertusa and Iñesta is an example.

A key advantage of the iterative-joint distinction is that it highlights the predominant tradeoff in automatic music transcription algorithms: simplicity and efficiency versus complexity and precision. The iterative approach can be more intuitive and is generally much more efficient in terms of computation, but joint approaches often perform better, using energy minimization, statistical models, or matrix factorization, despite tradeoffs with time complexity [2]. Pertusa and Iñesta's joint approach is fairly intuitive, but has a very high time complexity. For this section, I implemented both the original joint version of Pertusa and Iñesta's algorithm and an iterative version and then tested them on different recordings.

6.3.1 Iterative Multi-Pitch Estimation

Iterative multi-pitch estimation is a greedy approach that iterates through different fundamental frequency hypotheses, either in order of loudness [26] or in order of pitch [55]. In each iteration, a hypothesis is generated, and its harmonic content is removed from the frame's Fourier spectrum, then the next iteration uses the remaining spectrum. This is either repeated for a set number of possibly note hypotheses, or until no strong hypotheses can be extracted from the spectrum. In the simplest case, once an initial strongest fundamental frequency hypothesis is found, all peaks that line up with the hypothesis's harmonic frequencies can be removed from the spectrum completely. Better systems rely heavily on the spectral modeling described in the previous section to remove harmonics of the winning hypothesis only partially [26]. Yin's algorithm is an exception; because it uses a predefined instrument model, rather than continually generating a strongest hypothesis from the available spectrum, hypotheses are evaluated in order of frequency, matched to the instrument model, and thrown out if they do not exceed a loudness threshold.

A simple but effective automatic music transcription system can be created using the harmonicity-based approach described in Section 5.3.3. For each frame of sampled audio, first spectral peaks are isolated, then a vector of total loudnesses for all viable fundamental frequency hypotheses is generated per Equation (5.12). Then, the highest-loudness hypothesis is chosen. If this hypothesis's loudness is above a certain threshold γ, the frequency is added to the running list of frequencies present in that audio frame, otherwise the algorithm proceeds to the next frame because no strong hypotheses remain. Then, a smooth version of the hypothesis's harmonic shape is generated according to Equation (6.3) and subtracted from the vector of spectral peaks. Next, the process is repeated on the remaining and diminished spectral peaks, starting with the candidate loudness calculation stage. Figure 6.8 illustrates this process.



(a) Initial peaks

(b) First note removed

(c) Second note removed

(d) Third note removed

Figure 6.8: The iterative subtraction of spectral peaks from a C major chord based on harmonics of winning fundamental frequency candidates.

6.3.2 Joint Multi-Pitch Estimation

Joint multi-pitch estimation encompasses a broader range of algorithms. In contrast to iterative algorithms as described in the previous section, joint approaches enumerate many different possible note combinations and attempt to minimize some energy function based on loudness, spectral smoothness, or other features [54, 39]. This is the type of joint estimation discussed in this section, but there are also other entirely different approaches that fall into the category of 'joint', because they do not perform any iterative estimation and manipulation of the spectrum, but simply produce one all-encompassing several-note hypothesis that takes the entire spectrum into account. Aside from harmonicity and smoothness-based algorithms, these also include statistical model-based approaches, which attempt to fit the observed harmonic peaks to the set of all possible pitch combinations and infer the most likely combination of pitches that generated the

spectrum [2]. A third category of joint approaches is factorization-based approaches, which attempt to build a set of spectral basis vectors for each note in a recording and, for each observed spectrum, break it down into its most likely basis vectors and their respective weights [47, 10]. These approaches are incredibly interesting and can perform quite well, but for the sake of brevity this thesis will only cover feature-based joint approaches.

Pertusa and Iñesta's algorithm [39] is one feature-based approach. It follows the harmonicity-based strategy from Section 5.3.3, but one of their key contributions is the use of smoothness for both spectrum separation and hypothesis evaluation. Once spectral peaks are detected, and a harmonic loudness vector is generated for all possible fundamental frequency candidates, a set of all possible combinations of candidates is generated. This is the main source of large time complexity. In Pertusa and Iñesta's paper, the number of candidates is limited to $F = 10$, but this could generate as many as $2^{10} = 1024$ possible combinations, based on whether each pitch is present or not.

For each combination, a *salience score*, S is calculated based on two factors, loudness and smoothness. For each combination hypothesis, an iterative approach is used to calculate these two variables for each fundamental frequency candidate c. The candidates are tested in order of frequency. First, a candidate's loudness is calculated as the sum of its harmonic pattern powers, per Equation (5.12). Then, as in the iterative case, from the spectral pattern p_c an interpolation-smoothed spectral shape p'_c is generated using interpolation and subtracted from the vector of spectral peaks.

Next, unlike the iterative case, a *smoothness* factor is calculated using a Gaussian smoothness model. First, a smoothed pattern $\tilde{p}_c$ is generated according to Equation (6.4). Then, for each harmonic, a *sharpness* factor s is calculated as

$$s(c) = \sum_{h=1}^{H} |\tilde{p}_{c,h} - p'_{c,h}|, \tag{6.10}$$

where H is the highest-calculated harmonic. This sharpness is normalized by the num-

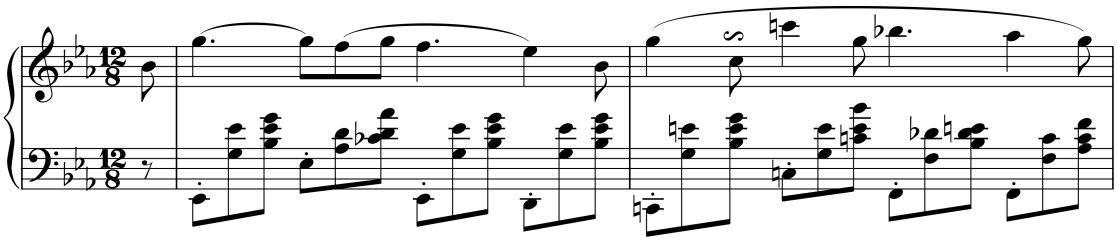


Figure 6.9: The first two measures of Frédéric Chopin's Nocturne in E-flat major, Op. 9, No. 2 [31].

ber of harmonics H and the smoothness σ is calculated as

$$\sigma(c) = 1 - s(c). \tag{6.11}$$

Finally, the salience of the combination is defined as

$$S = \sum_{c=1}^{C} (l(c) \cdot \sigma(c))^2, \tag{6.12}$$

where C is the total number of fundamental frequency candidates in the combination hypothesis. The highest-salience combination is chosen and the candidates from the combination are labeled as active in that audio frame. As one pruning technique, Pertusa and Iñesta also disqualify any combination that have candidates with $l(c) < \gamma L$, where L is the maximum loudness of any candidate in that combination. This helps to avoid false positives.

6.4 Experiments

Experiments were carried out using Python implementations of both the iterative and joint versions of Pertusa and Iñesta's algorithm (Appendix A.5 and A.6). Two well-known compositions for piano by Frédéric Chopin served as test data (Figures 6.9 and 6.10). I chose $[f_{min}, f_{max}] = [50\text{ Hz}, 2100\text{ Hz}]$, which captures all notes between G1 and C7. I set the minimum power threshold as $\mu = 10.0$ and the maximum number of simultaneous notes as $F = 7$, which was sufficient for all test audio sources. In the joint

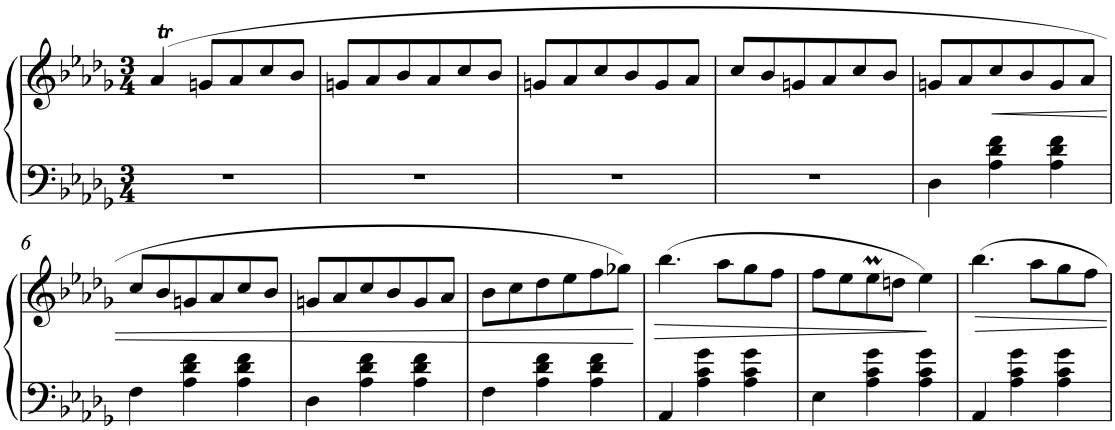
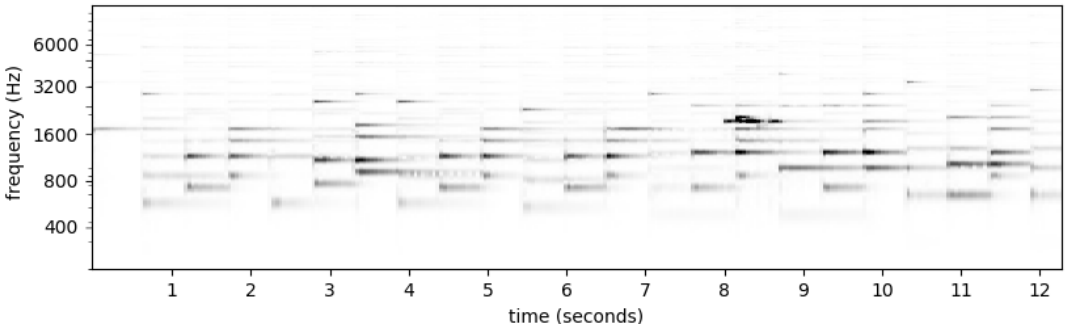


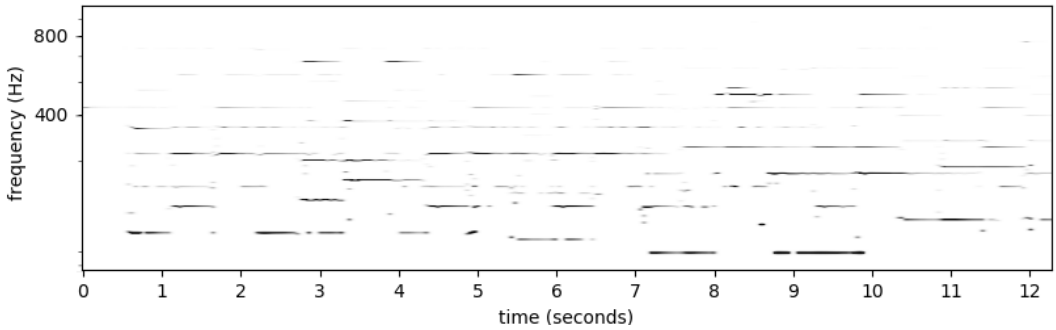
Figure 6.10: The first eleven measures of Frédéric Chopin's Waltz in D-flat major, Op. 64, No. 1, also known as the "Minute Waltz" [31].

case, I chose the minimum loudness ratio as $\gamma = 0.1$. For post-processing, all notes were required to be present in at least 3 frames in a row to avoid brief false positives.

Figure 6.11 shows the first steps of the algorithm on Chopin's Nocturne in E-flat major, Op. 9, No. 2. Figure 6.11a is the power spectrogram of the recording, while Figure 6.11b shows the initial loudness calculations for different F0 hypotheses over time. Figure 6.12 shows the results of the iterative and joint algorithms on this recording. Both approaches perform quite well, with two notable failure cases. First, the highest notes' durations are noticeably shorter than the ground-truth durations. This can be attributed to the quick decay of notes in this range on the piano. In fact, even though the ground truth specifies their duration as lasting almost two seconds in some cases, the notes are fully inaudible well before this, so no automatic music transcription system would likely come up with the ground-truth durations. The second error, however, is the lowest notes, which seem to be completely missing in the estimated pitches. This is because the first harmonic lower pitches are barely audible in the recording, as shown in Figure 6.11a. Instead, the algorithms detect the notes as being played one octave up, the second harmonic. This is a limitation of the spectral smoothness model; if a better-defined instrument model was employed, there would be more likelihood of accounting



(a) Fourier spectrogram.

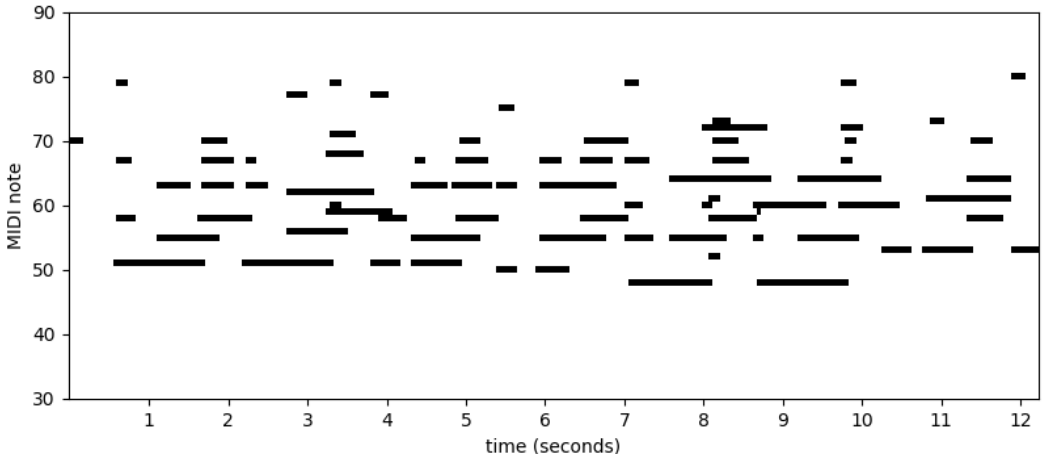


(b) Fundamental frequency loudness weights by Equation (5.12).

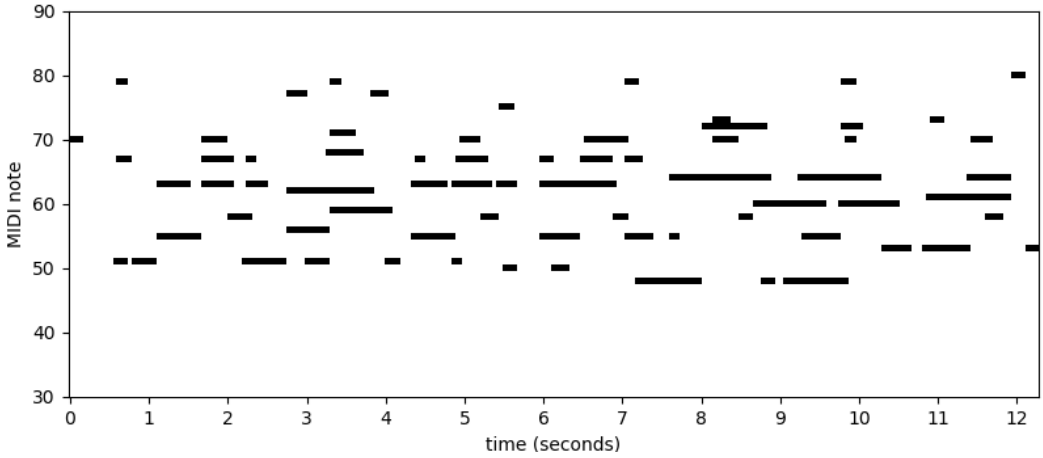
Figure 6.11: The Fourier spectrogram and resulting fundamental frequency weights for Chopin's Nocturne in E-flat major, Op. 9, No. 2.

for the uneven spectral shape of piano notes as low as C2 $= 65$ Hz.

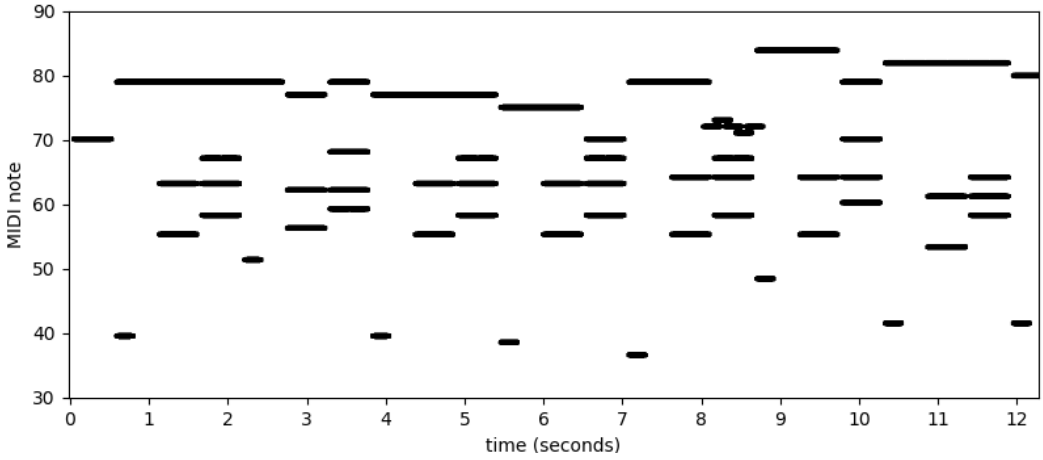
Turning to another Chopin composition, his Waltz in D-flat major, Op. 64, No. 1, different challenges arise (Figure 6.13). As in the nocturne, some of the lower-register notes are detected as one octave higher than the ground truth due to the frequency response of the recording, though less severely. Unlike the nocturne, however, this composition features many very brief notes, which both the iterative and joint algorithms occasionally fail to detect, most notably in the opening trill and the ascending line in measure nine, roughly two-thirds into the recording. The most obvious error, however, is likely the large amount of low-pitch, short-duration false-positives in the iterative approach (Figure 6.13a). This shows the great shortcoming of a greedy iterative approach that, while significantly faster, fails to find the best combination of notes that accounts for the most reliable layer of harmonics based on loudness and smoothness.



(a) Pitches extracted by the iterative approach.

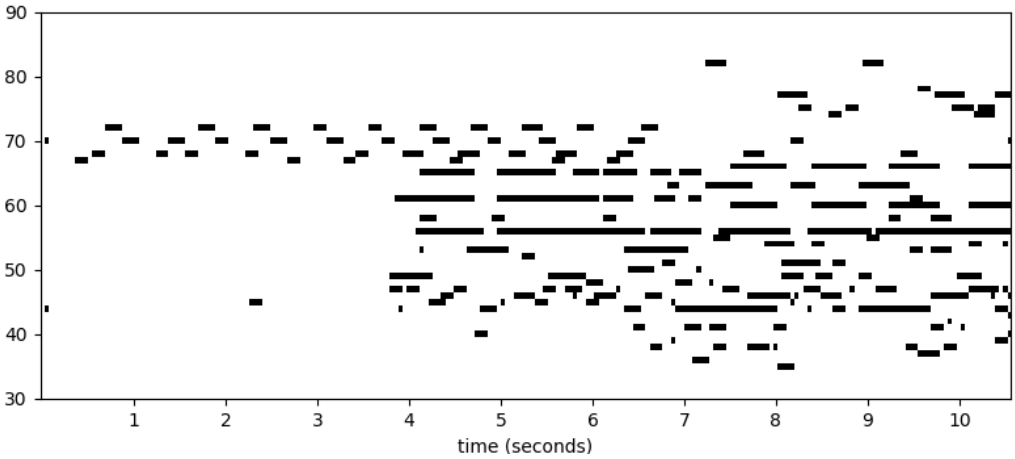


(b) Pitches extracted by the joint approach.

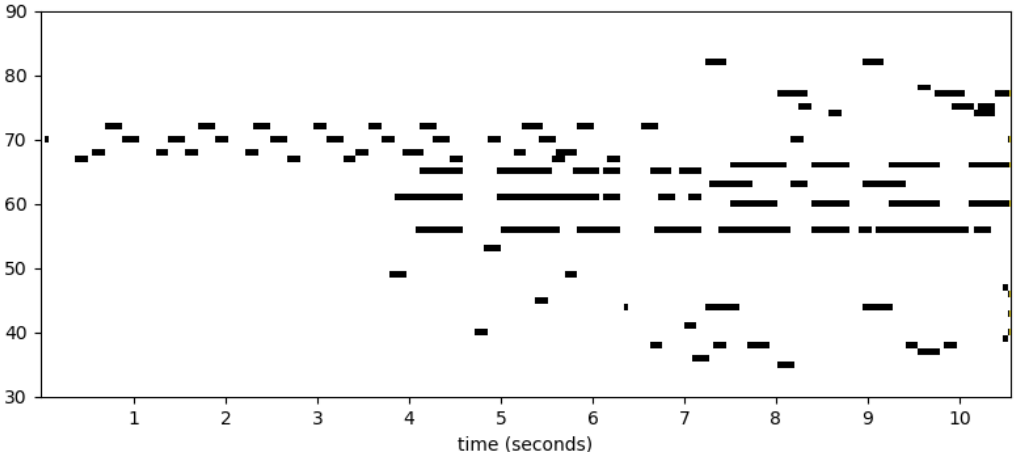


(c) Ground truth pitches.

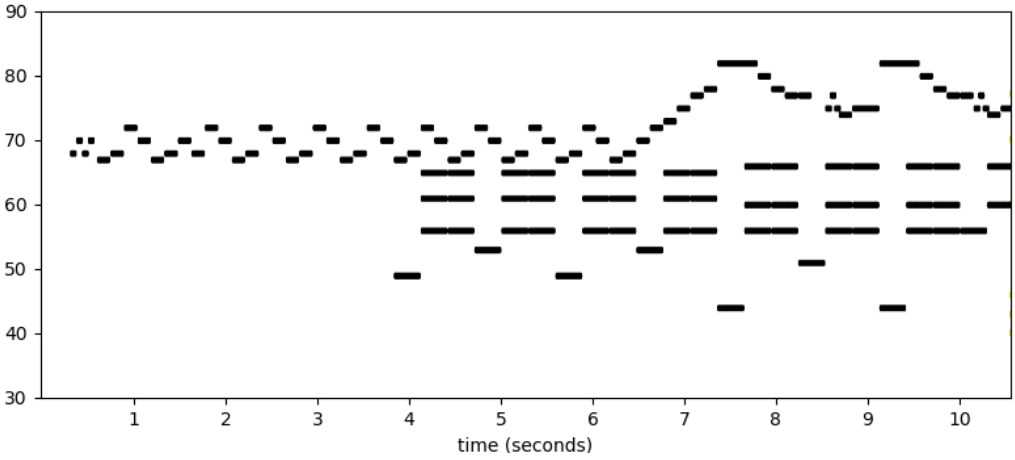
Figure 6.12: The results of iterative and joint estimation on the first four measures of Chopin's Nocturne in E-flat major, Op. 9, No. 2.



(a) Pitches extracted by the iterative approach.



(b) Pitches extracted by the joint approach.



(c) Ground truth pitches.

Figure 6.13: The results of iterative and joint estimation on the first twelve measures of Chopin's Waltz in D-flat major, Op. 64, No. 1, or "Minute Waltz".

CHAPTER 7

CONCLUSION

This thesis has introduced the problem of *automatic music transcription* and described the makeup of algorithms that attempt to perform this challenging task. Chapter 2 gave an in-depth description of music and its components, leading up to Chapter 3, which detailed the history and common approaches to music transcription, as well as a formalization of the problem of automatic music transcription. Chapter 4 described and informally derived the Fourier transform, a signal processing tool that allows us to examine audio recordings in the frequency domain, not just the time domain of raw audio. Chapter 5 contained descriptions and examples of different single-pitch estimation algorithms developed over the past half-century, using the Fourier transform as a preprocessing step. Finally, in Chapter 6, solutions the central problem of automatic music transcription, *multi-pitch estimation*, were outlined. In particular, focus was given to harmonicity-based approaches, specifically those of Pertusa and Iñesta [39] and Klapuri [26]. Examples closely followed harmonicity-based pitch estimation, which strikes a balance of intuitive sense and high accuracy [2].

Of course, some very exciting approaches were not covered in this paper. This includes matrix-factorization based approaches and recent publications taking advantage of the advances in deep learning with artificial neural networks. Music transcription, like seemingly every other field of signal processing, shows a great deal of potential in deep-learning assisted systems [45, 50]. Even more exciting music transcription systems are yet to come.

APPENDIX A

RELEVANT CODE

This appendix contains selected source code files for showing how different pitch estimation algorithms are used in practice. All code was written in Python 3.6.4. All files are available on GitHub at `https://dquenne.github.io/cs702/`.

A.1 Autocorrelation Pitch Detection

```
#!/usr/bin/env python

# compute and display a frame-wise pitch estimation of an
# audio recording using the autocorrelation function

# Dylan Quenneville 2018-05-15

# usage:
#    ./ac.py in.wav

import matplotlib.pyplot as plt
from tqdm import tqdm
import numpy as np
import sys
import util

if len(sys.argv) == 2:
    input_file = sys.argv[1]
else:
    print("usage: ./ac.py input.wav")
    sys.exit()

# constants
f_min = 50
f_max = 2000
frame_width = 2048
spacing = 2048
bitrate = 44100

# compute autocorrelation of data
def autocorrelation(data, min_lag, max_lag):
    n = len(data)
    result = list(np.zeros(min_lag))
    for lag in range(min_lag, max_lag):
        sumarray = np.zeros(n+lag)
        sumarray[:n] = data
        sumarray[:n-lag] *= data[lag:]
        sum = np.sum(sumarray[:n-lag])
        result.append(float(sum/(n-lag)))
    return result

data = util.load_wav(input_file)
ac_correlogram = []
best_frequencies = []
hann = np.hanning(frame_width)
for i in tqdm(range(0, int((len(data)-frame_width)/spacing))):
```

```
    frame = data[i*spacing:i*spacing+frame_width]
    c = frame*hann
    ach = autocorrelation(c, bitrate // f_max, bitrate // f_min)
    ac_correlogram.append(ach)
    best_lag = np.argmax(ach)
    if best_lag <= 1:
        best_frequencies.append(0)
    else:
        best_frequencies.append(bitrate/best_lag)

plt.title("autocorrelation pitches")
util.plot_pitches(best_frequencies, spacing, bitrate)

plt.title("autocorrelation correlogram")
util.plot_correlogram(ac_correlogram, frame_width, spacing, bitrate)
```

A.2 Normalized Cross-Correlation Pitch Detection

```
#!/usr/bin/env python

# compute and display a frame-wise pitch estimation of an
# audio recording using a normalized cross-correlation function

# Dylan Quenneville 2018-05-15

# usage:
#    ./nccf.py in.wav

import matplotlib.pyplot as plt
from tqdm import tqdm
import numpy as np
import sys
import util

if len(sys.argv) == 2:
    input_file = sys.argv[1]
else:
    print("usage: ./nccf.py input.wav")
    sys.exit()

# constants
f_min = 50
f_max = 2000
frame_width = 2048
spacing = 2048
bitrate = bitrate = 44100

# compute cross-correlation and normalized cross-correlation
def nccf(data, frame_index, e, min_lag, max_lag):
    n = frame_width
    ccf = list(np.zeros(min_lag))
    nccf = list(np.zeros(min_lag))
    for lag in range(min_lag, max_lag):
        lag_sum = 0
        sumarray = np.zeros(n+lag)
        sumarray[:n] = data[frame_index:frame_index+n]
        sumarray[:n] *= data[frame_index+lag:frame_index+n+lag]
```

```
        lag_sum = np.sum(sumarray[:n])
        ccf.append(lag_sum)
        nccf.append(lag_sum/np.sqrt(e[frame_index]*e[frame_index+lag]))
    return ccf, nccf

# storage
data = util.load_wav(input_file)
cc_correlogram = []
nccf_correlogram = []
best_frequencies = []
hann = np.hanning(frame_width)

# pre-calculate normalization factor data
squared_data = []
for i in range(0, len(data)):
    squared_data.append(data[i]**2)

e = [0.0]
for i in range(0, frame_width-1):
    e[0] += squared_data[i]

for i in range(0, len(data)-frame_width):
    e.append(e[i-1]-squared_data[i-1]+squared_data[i+frame_width])

for i in tqdm(range(0, int((len(data)-frame_width*2)/spacing))):
    cc,ncc = nccf(data, i*spacing, e, bitrate//f_max, bitrate//f_min)
    cc_correlogram.append(cc)
    nccf_correlogram.append(ncc)
    best_lag = np.argmax(ncc)
    if best_lag == 0:
        best_frequencies.append(0)
    else:
        best_frequencies.append(bitrate/best_lag)

plt.title("normalized cross-correlation pitches")
util.plot_pitches(best_frequencies, spacing, bitrate)

plt.title("cross-correlation correlogram")
util.plot_correlogram(cc_correlogram, frame_width, spacing, bitrate)

plt.title("normalized cross-correlation correlogram")
util.plot_correlogram(nccf_correlogram, frame_width, spacing,
    bitrate)
```

A.3 Cepstrum Pitch Detection

```
#!/usr/bin/env python

# compute and display a frame-wise pitch estimation of an
# audio recording using the cepstrum

# Dylan Quenneville 2018-05-15

# usage:
#  ./cepstrum.py in.wav

import matplotlib.pyplot as plt
```

```
from scipy.fftpack import fft, ifft
from tqdm import tqdm
import numpy as np
import sys
import util

# get input filename
if len(sys.argv) == 2:
    input_file = sys.argv[1]
else:
    print("usage: ./cepstrum.py input.wav")
    sys.exit()

# basic i/o constants
frame_width = 2048
hamming = np.hamming(frame_width)
spacing = 1024
bitrate = 44100

# storage
data = util.load_wav(input_file)
spectrogram = []
cepstrum = []
best_frequencies = []

for i in tqdm(range(0, int((len(data)-frame_width)/spacing))):
    frame = data[i*spacing:i*spacing+frame_width]
    frame = frame*hamming
    complex_fourier = fft(frame)
    fft_len = int(np.floor(len(complex_fourier)/2))
    power_sp = np.log(abs(complex_fourier))

    spectrogram.append(power_sp[:(fft_len-1)])
    cepst = abs(ifft(power_sp))[:fft_len//2]/frame_width
    cepstrum.append(cepst)
    cepst[:8] = np.zeros(8) # these give strong false positives
    maxperiod = np.argmax(cepst[30:]) + 30
    best_frequencies.append(bitrate/maxperiod)

plt.title("cepstrum pitches")
util.plot_pitches(best_frequencies, spacing, bitrate)

plt.title("fourier power spectrogram")
util.plot_spectrogram(spectrogram, frame_width, spacing, bitrate)

plt.title("cepstrum spectrogram")
util.plot_correlogram(cepstrum, frame_width, spacing, bitrate)
```

A.4 Harmonicity-Based Pitch Detection

```
#!/usr/bin/env python

# compute and display a frame-wise pitch estimation of an
# audio recording based on a harmonicity model

# Dylan Quenneville 2018-05-15

# usage:
#  ./harmonicity.py in.wav

import matplotlib.pyplot as plt
from scipy.fftpack import fft
from tqdm import tqdm
import numpy as np
import sys
import util # commonly used functions for pitch detection

if len(sys.argv) == 2:
    input_file = sys.argv[1]
else:
    print("usage: ./harmonicity.py input.wav")
    sys.exit()

# basic i/o constants
frame_width = 8192
hann = np.hanning(frame_width)
spacing = 1024
bitrate = 44100

f_0 = 50
f_1 = 2000
f_r = 10
power_thresh = 10

data = util.load_wav(input_file)
all_weights = []
fft_len = frame_width*4
zeropad = np.zeros(frame_width*3)
best_frequencies = []

k0 = int(np.floor(util.hz_to_fourier(f_0, frame_width*4, bitrate)))
k1 = int(np.ceil(util.hz_to_fourier(f_1, frame_width*4, bitrate)))
# iterate through frames
for i in tqdm(range(0, int((len(data)-frame_width)/spacing))):

    # spectrum generation and preprocessing

    frame = data[i*spacing:i*spacing+frame_width]
    window = frame * hann
    raw_fft = fft(np.concatenate((window, zeropad)))
    spectrum_len = int(np.floor(len(raw_fft)/2))
    power_sp = abs(raw_fft)[:spectrum_len]

    # actual F0 estimation part

    hypotheses = []
    peaks = {}

    prev = power_sp[0]
    higher = power_sp[1]
```

```
    for k in range(1, len(power_sp)-1):
        power = higher
        higher = power_sp[k+1]
        if (power > power_thresh):
            if power > prev and power > higher:
                peaks[k] = power
                if k > k0 and k < k1:
                    hypotheses.append(k)
        prev = power

    total_powers = np.zeros(k1)
    for f0 in hypotheses:
        total_powers[f0] = power_sp[f0]
        for harmonic in range(2, 20):
            best_power = 0
            best_freq = 0
            if harmonic*f0 + f_r > len(power_sp):
                break
            for inharmonicity in range(-f_r, f_r):
                h_f = harmonic*f0 + inharmonicity
                if h_f in peaks:
                    h_power = power_sp[h_f]
                    if h_power > best_power:
                        best_power = h_power
                        best_freq = h_f

            total_powers[f0] += best_power

    all_weights.append(total_powers)
    best_frequencies.append(np.argmax(total_powers))

plt.title("harmonicity-based pitches")
util.plot_pitches(best_frequencies, spacing, bitrate)

# display spectra:

plt.title("harmonicity-based loudness weights")
util.plot_spectrogram(all_weights, fft_len, spacing, bitrate)
```

A.5 Iterative Multi-Pitch Detection

```
#!/usr/bin/env python

# compute multi-pitch estimation of an audio recording and
# output as MIDI file

# based on an iterative version of Antonio Pertusa and Jose Inesta's
# 2008 paper ``Multiple fundamental ferquency estimation using
# Gaussian smoothness.''

# Dylan Quenneville 2018-05-15

# usage:
#    ./pertusa-iterative.py in.wav out.mid

import matplotlib.pyplot as plt
from scipy.fftpack import fft, ifft
```

```
from tqdm import tqdm
import numpy as np
import itertools
import sys
import util # commonly used functions for pitch detection

# get i/o filenames
if len(sys.argv) == 3:
    input_file = sys.argv[1]
    output_file = sys.argv[2]
else:
    print("usage: ./pertusa-inesta.py input.wav out.mid")
    sys.exit()

# basic i/o constants
frame_width = fw = 8192
hann = np.hanning(frame_width)
spacing = frame_width//4
bitrate = br = 44100

# algorithm constants
f_0 = 50
f_1 = 2000
f_r = 10
power_thresh = 5
big_f = 10
gaussian = [0.21, 0.58, 0.21]
gamma = 0.1

use_zeropad = True

# storage
data = util.load_wav(input_file)
f0_weights = []
midi_result = []
all_notes = []
if use_zeropad:
    fft_len = fw*4
    zeropad = np.zeros(fw*3)
else:
    fft_len = fw
    zeropad = np.array([])

# convert relevant frequency range to FFT indices
k0 = int(np.floor(util.hz_to_fourier(f_0, fw*4, br)))
k1 = int(np.ceil(util.hz_to_fourier(f_1, fw*4, br)))

def get_spectral_peaks_and_hypotheses(spectrum):
    peaks = {}
    hypotheses = []
    prev = spectrum[0]
    higher = spectrum[1]
    for k in range(1, len(spectrum)-1):
        power = higher
        higher = spectrum[k+1]
        if (power > power_thresh):
            if power > prev and power > higher:
                peaks[k] = power
                if k > k0 and k < k1: # F0 frequency range
                    hypotheses.append(k)
        prev = power
    return peaks, hypotheses

```

```
# iterate through frames
for i in tqdm(range(0, int((len(data)-frame_width)/spacing))):

    # spectrum generation and preprocessing

    frame = data[i*spacing:i*spacing+frame_width]
    window = frame * hann
    raw_fft = fft(np.concatenate((window, zeropad)))
    spectrum_len = int(np.floor(len(raw_fft)/2))
    power_sp = abs(raw_fft)[:spectrum_len]

    peaks, hypotheses = get_spectral_peaks_and_hypotheses(power_sp)

    # compute initial loudnesses for F0 candidates

    total_powers = np.zeros(k1)
    patterns = {}
    harmonic_owners = {} # hashmap of harmonic ownership
    loudnesses = {}
    for f0 in hypotheses:
        patterns[f0] = [(f0, power_sp[f0])]
        total_powers[f0] = power_sp[f0]
        if f0 in harmonic_owners:
            harmonic_owners[f0].append(f0)
        else:
            harmonic_owners[f0] = [f0]

        for harmonic in range(2, 20):
            best = 0
            best_freq = 0
            if harmonic*f0 > len(power_sp):
                break
            for inharmonicity in range(-f_r, min(f_r, spectrum_len)):
                h_f = harmonic*f0 + inharmonicity
                if h_f in peaks:
                    h_power = power_sp[h_f]
                    if h_power > best:
                        best = h_power
                        best_freq = h_f
            if best_freq == 0:
                best_freq = harmonic*f0
                peaks[harmonic*f0] = 0.0
            if best_freq in harmonic_owners:
                harmonic_owners[best_freq].append(f0)
            else:
                harmonic_owners[best_freq] = [f0]

            total_powers[f0] += best
            patterns[f0].append((best_freq, best))
        loudnesses[f0] = total_powers[f0]

    kept_notes = []
    for dummy in range(0, big_f):
        best = np.argmax(total_powers)
        if total_powers[best] <= 90.0:
            break
        total_powers[best] = 0.0
        big_l = 0
        smallest_l = 100000
        p = patterns[best]
        peaks[p[0][0]] = 0.0
        for h in range(1, len(p)-1):
            h_f = p[h][0]
```

```
143             if len(harmonic_owners[h_f]) > 1: # shared
144                 h_fl = p[h-1][0]
145                 h_fr = p[h+1][0]
146                 interpolated = (p[h-1][1] + peaks[h_fr])/2
147                 if interpolated < peaks[h_f]:
148                     p[h] = (h_f, interpolated)
149                     peaks[h_f] = peaks[h_f] - interpolated
150                     for owner in harmonic_owners[h_f]:
151                         total_powers[owner] -= interpolated
152                 else:
153                     p[h] = (h_f, peaks[h_f])
154                     peaks[h_f] = 0.0
155                     for owner in harmonic_owners[h_f]:
156                         total_powers[owner] -= peaks[h_f]
157             else:
158                 if h_f < len(peaks):
159                     p[h] = (h_f, peaks[h_f])
160         powers = np.array(p).T[1]
161         loudness = np.sum(powers)
162         if loudness > big_l:
163             big_l = loudness
164         if loudness < smallest_l:
165             smallest_l = loudness
166         if smallest_l > gamma*big_l:
167             kept_notes.append(best)
168 
169     result = np.zeros(k1)
170     midi_notes = []
171     for freq in kept_notes:
172         hzfreq = util.fourier_to_hz(freq, fft_len, br)
173         note = util.hz_to_midi(hzfreq)
174         midi_notes.append(note)
175         result[freq] = loudnesses[freq]
176     f0_weights.append(result)
177 
178     all_notes.append(midi_notes)
179 
180 # post processing (note pruning based on duration)
181 
182 for i in range(0, len(all_notes)):
183     midi_array = np.zeros(140)
184     for note in all_notes[i]:
185         if (i >= 2 and i+2 < len(all_notes)):
186             if note in all_notes[i-1] and note in all_notes[i-2] or \
187                 note in all_notes[i-1] and note in all_notes[i+1] or \
188                 note in all_notes[i+1] and note in all_notes[i+2]:
189                 midi_array[note] = 1.0
190             elif note+1 in all_notes[i-1] and note+1 in
                    all_notes[i+1]:
191                 midi_array[note+1] = 1.0
192             elif note-1 in all_notes[i-1] and note-1 in
                    all_notes[i+1]:
193                 midi_array[note-1] = 1.0
194         elif (i < 2 or i+2 >= len(all_notes)-1):
195             midi_array[note] = 1.0
196 
197     midi_result.append(midi_array)
198 
199 print('writing midi file')
200 util.write_midi(midi_result, output_file, spacing/br, 4)
```

A.6 Joint Multi-Pitch Detection

```
#!/usr/bin/env python

# compute multi-pitch estimation of an audio recording and output as
# a MIDI file

# based on the original joint version of Antonio Pertusa and Jose
# Inesta's 2008 paper ``Multiple fundamental ferquency estimation
# using Gaussian smoothness.''

# Dylan Quenneville 2018-05-15

# usage:
#  ./pertusa-inesta.py in.wav out.mid

import matplotlib.pyplot as plt
from scipy.fftpack import fft, ifft
from scipy.io import wavfile
from tqdm import tqdm
import numpy as np
import itertools
import sys
import util # commonly used functions for pitch detection

# get i/o filenames
if len(sys.argv) == 3:
    input_file = sys.argv[1]
    output_file = sys.argv[2]
else:
    print("usage: ./pertusa-inesta.py input.wav out.mid")
    sys.exit()

# basic i/o constants
frame_width = 8192
hann = np.hanning(frame_width)
spacing = 1024
bitrate = 44100

# algorithm constants
f_0 = 50
f_1 = 2000
f_r = 10
power_thresh = 5
big_f = 7
gaussian = [0.21, 0.58, 0.21]
gamma = 0.1

use_zeropad = True

# storage
data = util.load_wav(input_file)
f0_weights = []
midi_result = []
all_notes = []
if use_zeropad:
    fft_len = frame_width*4
    zeropad = np.zeros(frame_width*3)
else:
    fft_len = frame_width
    zeropad = np.array([])

```

```
# convert relevant frequency range to FFT indices
k0 = int(np.floor(util.hz_to_fourier(f_0, frame_width*4, bitrate)))
k1 = int(np.ceil(util.hz_to_fourier(f_1, frame_width*4, bitrate)))

def get_spectral_peaks_and_hypotheses(spectrum):
    peaks = {}
    hypotheses = []
    prev = spectrum[0]
    higher = spectrum[1]
    for k in range(1, len(spectrum)-1):
        power = higher
        higher = spectrum[k+1]
        if (power > power_thresh):
            if power > prev and power > higher:
                peaks[k] = power
                if k > k0 and k < k1: # F0 frequency range
                    hypotheses.append(k)
        prev = power

    return peaks, hypotheses

# iterate through audio frames
for i in tqdm(range(0, int((len(data)-frame_width)/spacing))):

    # spectrum generation and preprocessing

    frame = data[i*spacing:i*spacing+frame_width]
    window = frame * hann
    raw_fft = fft(np.concatenate((window, zeropad)))
    spectrum_len = int(np.floor(len(raw_fft)/2))
    power_sp = abs(raw_fft)[:spectrum_len]

    # get spectral peaks & F0 hypotheses

    peaks, hypotheses = get_spectral_peaks_and_hypotheses(power_sp)

    total_powers = np.zeros(k1)
    patterns = {}
    harmonic_owners = {} # hashmap of harmonic ownership
    for f0 in hypotheses:
        patterns[f0] = [(f0, power_sp[f0])]
        total_powers[f0] = power_sp[f0]
        if f0 in harmonic_owners:
            harmonic_owners[f0].append(f0)
        else:
            harmonic_owners[f0] = [f0]

        for harmonic in range(2, 20):
            best_power = 0
            best_freq = 0
            if harmonic*f0 > spectrum_len:
                break
            for inharmonicity in range(-f_r, min(f_r, spectrum_len)):
                h_f = harmonic*f0 + inharmonicity
                if h_f in peaks:
                    h_power = power_sp[h_f]
                    if h_power > best_power:
                        best_power = h_power
                        best_freq = h_f
            if best_freq == 0: # no harmonic peak
                best_freq = harmonic*f0
                peaks[harmonic*f0] = 0.0
            if best_freq in harmonic_owners:
```

```
                harmonic_owners[best_freq].append(f0)
            else:
                harmonic_owners[best_freq] = [f0]

            total_powers[f0] += best_power
            patterns[f0].append((best_freq, best_power))

    # get 10 best hypotheses (ordered by total loudness)
    # and sort by frequency

    top10 = np.argpartition(total_powers, -10)[-10:]
    top10stripped = []
    for candidate in top10:
        if total_powers[candidate] > 0:
            top10stripped.append(candidate)
    top10 = np.sort(np.array(top10stripped))

    # if there are no strong hypotheses, go to next audio frame

    if len(top10) == 0:
        all_notes.append([])
        continue

    all_combinations = []
    for num_pitches in range(1, min(big_f, len(top10))+1):
        for combo in itertools.combinations(top10, num_pitches):
            all_combinations.append(combo)

    saliances = []
    for combination in all_combinations:
        hpeaks = peaks.copy() # undo spectral subtractions
        saliance = 0
        big_l = 0
        smallest_l = 100000
        for c in combination:
            if total_powers[c] <= 0.0:
                continue
            p = patterns[c]
            for h in range(1, len(p)-1):
                if len(harmonic_owners[p[h][0]]) > 1: # shared
                    h_f = p[h][0]
                    interpolated = (p[h-1][1] + peaks[p[h+1][0]])/2
                    p[h] = (h_f, min(interpolated, hpeaks[h_f]))
                    hpeaks[h_f] = hpeaks[h_f] - p[h][1]
                else:
                    if h_f < len(hpeaks):
                        p[h] = (h_f, hpeaks[h_f])
            powers = np.array(p).T[1]
            loudness = np.sum(powers)
            if loudness > big_l:
                big_l = loudness
            if loudness < smallest_l:
                smallest_l = loudness
            if len(powers) > 2:
                smooth = np.convolve(powers, gaussian, 'same')
                sharpness = np.sum(abs(smooth - powers))
                sharpness /= len(powers)
            else:
                sharpness = 0
            saliance += (loudness * (1 - sharpness))**2
        if smallest_l < gamma*big_l:
            saliance = 0.0 # disqualify this hypothesis
        saliances.append(saliance)
```

```
187
188     best_combination = all_combinations[np.argmax(saliances)]
189
190     result = np.zeros(k1)
191     midi_notes = []
192     for freq in best_combination:
193         hzfreq = util.fourier_to_hz(freq, fft_len, bitrate)
194         note = util.hz_to_midi(hzfreq)
195         if total_powers[freq] > 90: # additional loudness pruning
196             midi_notes.append(note)
197         result[freq] = total_powers[freq]
198     f0_weights.append(result)
199
200     all_notes.append(midi_notes)
201
202 # post processing (note pruning based on duration)
203
204 for i in range(0, len(all_notes)):
205     midi_array = np.zeros(140)
206     for note in all_notes[i]:
207         if (i >= 2 and i+2 < len(all_notes)):
208             if note in all_notes[i-1] and note in all_notes[i-2] or \
209                 note in all_notes[i-1] and note in all_notes[i+1] or \
210                 note in all_notes[i+1] and note in all_notes[i+2]:
211                 midi_array[note] = 1.0
212             elif note+1 in all_notes[i-1] and note+1 in
                    all_notes[i+1]:
213                 midi_array[note+1] = 1.0
214             elif note-1 in all_notes[i-1] and note-1 in
                    all_notes[i+1]:
215                 midi_array[note-1] = 1.0
216         elif (i < 2 or i+2 >= len(all_notes)-1):
217             midi_array[note] = 1.0
218
219     midi_result.append(midi_array)
220
221 print('writing midi file to', output_file)
222 util.write_midi(midi_result, output_file, spacing/bitrate, 4)
```

BIBLIOGRAPHY

[1] Emmanouil Benetos, Simon Dixon, Dimitrios Giannoulis, Holger Kirchhoff, and Anssi Klapuri. Automatic music transcription: Breaking the glass ceiling. In *13th International Society for Music Information Retrieval Conference (ISMIR 2012)*. FEUP Edições, 2012.

[2] Emmanouil Benetos, Simon Dixon, Dimitrios Giannoulis, Holger Kirchhoff, and Anssi Klapuri. Automatic music transcription: challenges and future directions. *Journal of Intelligent Information Systems*, 41(3):407–434, 2013.

[3] Bruce P. Bogert. The quefrency analysis of time series for echoes: cepstrum, pseudo-autocovariance, cross-cepstrum and saphe cracking. In *Proceedings of the Symposium on Time Series Analysis*. John Wiley & Sons, 1963.

[4] Ronald N. Bracewell. The Fourier transform. *Scientific American*, 260(6):86–95, 1989.

[5] Judith C. Brown. Calculation of a constant Q spectral transform. *Journal of the Acoustical Society of America*, 89(1):425–434, 1991.

[6] Judith C. Brown and Miller S. Puckette. An efficient algorithm for the calculation of a constant Q transform. *The Journal of the Acoustical Society of America*, 92(5):2698–2701, 1992.

[7] Russell Burton. The elements of music: What are they, and who cares? In *Music: Educating for life. ASME XXth National Conference Proceedings*, page 22. Australian Society for Music Education, 2015.

[8] Erik Cheever. Derivation of Fourier Series. `http://lpsa.swarthmore.edu/Fourier/Series/DerFS.html`, 2015. Accessed: 2018-04-11.

[9] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[10] Arnaud Dessein, Arshia Cont, and Guillaume Lemaitre. Real-time polyphonic music transcription with non-negative matrix factorization and beta-divergence. In *11th International Society for Music Information Retrieval Conference (ISMIR 2010)*, pages 489–494, 2010.

[11] Homer Dudley. The carrier nature of speech. *Bell Labs Technical Journal*, 19(4):495–515, 1940.

[12] Slim Essid, Gaël Richard, and Bertrand David. Instrument recognition in polyphonic music. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2005)*, volume 3, 2005.

[13] Music Information Retrieval Evaluation eXchange. 2017:Multiple Fundamental Frequency Estimation & Tracking Results. `http://www.music-ir.org/mirex/wiki/2017:Multiple_Fundamental_Frequency_Estimation_\%26_Tracking_Results_-_MIREX_Dataset`, 2017. Accessed: 2018-04-23.

[14] Bernard Gold. Computer program for pitch extraction. *The Journal of the Acoustical Society of America*, 34(7):916–921, 1962.

[15] Masataka Goto. Development of the RWC music database. In *Proceedings of the 18th International Congress on Acoustics (ICA 2004)*, volume 1, pages 553–556, 2004.

[16] Masataka Goto. Music scene description. In *Signal Processing Methods for Music Transcription*, pages 327–359. Springer, 2006.

[17] Masataka Goto, Hiroki Hashiguchi, Takuichi Nishimura, and Ryuichi Oka. RWC music database: Popular, classical and jazz music databases. In *3rd International Society for Music Information Retrieval Conference (ISMIR 2002)*, volume 2, pages 287–288, 2002.

[18] Masataka Goto and Yoichi Muraoka. Real-time beat tracking for drumless audio signals: Chord change detection for musical decisions. *Speech Communication*, 27(3-4):311–335, 1999.

[19] Perfecto Herrera-Boyer, Anssi Klapuri, and Manuel Davy. Automatic classification of pitched musical instrument sounds. In *Signal Processing Methods for Music Transcription*, pages 163–200. Springer, 2006.

[20] John D. Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 2007.

[21] John Francis James, Raymond N. Enzweiler, Susan McKay, and Wolfgang Christian. A student's guide to Fourier transforms with applications in physics and engineering. *Computers in Physics*, 10(1), 1996.

[22] Kunio Kashino. Auditory scene analysis in music signals. In *Signal Processing Methods for Music Transcription*, pages 299–325. Springer, 2006.

[23] Holger Kirchhoff, Simon Dixon, and Anssi Klapuri. Multi-template shift-variant non-negative matrix deconvolution for semi-automatic music transcription. In *13th International Society for Music Information Retrieval Conference (ISMIR 2012)*, pages 415–420, 2012.

[24] Anssi Klapuri. Qualitative and quantitative aspects in the design of periodicity estimation algorithms. In *10th European Signal Processing Conference, 2000*, pages 1–4, 2000.

[25] Anssi Klapuri. Multipitch estimation and sound separation by the spectral smoothness principle. In *IEEE International Conference on Acoustics, Speech, and Signal processing (ICASSP 2001)*, volume 5, pages 3381–3384, 2001.

[26] Anssi Klapuri. Multiple fundamental frequency estimation based on harmonicity and spectral smoothness. *IEEE Transactions on Speech and Audio Processing*, 11(6):804–816, 2003.

[27] Anssi Klapuri. Introduction to music transcription. In *Signal Processing Methods for Music Transcription*, pages 3–20. Springer, 2006.

[28] Nobuyuki Kunieda, Tetsuya Shimamura, and Jouji Suzuki. Robust method of measurement of fundamental frequency by ACLOS: autocorrelation of log spectrum. In *IEEE International Conference on Acoustics, Speech, and Signal Processing, 1996 (ICASSP-96)*, volume 1, pages 232–235, 1996.

[29] Jean Laroche. Efficient tempo and beat tracking in audio recordings. *Journal of the Audio Engineering Society*, 51(4):226–233, 2003.

[30] Arie Livshin and Xavier Rodet. Musical instrument identification in continuous recordings. In *Digital Audio Effects 2004*, pages 222–227, 2004.

[31] Music Files Ltd. Free Sheet Music: Classical, Traditional, Original. `https://www.mfiles.co.uk/sheet-music.htm`, 2018. Accessed: 2018-05-11.

[32] Keith D. Martin and Youngmoo E. Kim. Musical instrument identification: A pattern-recognition approach. *The Journal of the Acoustical Society of America*, 104(3):1768–1768, 1998.

[33] Cory McKay, Daniel McEnnis, and Ichiro Fujinaga. A large publicly accassible prototype audio database for music research. In *7th International Society for Music Information Retrieval Conference (ISMIR 2006)*, pages 160–163, 2006.

[34] James A. Moorer. On the transcription of musical sound by computer. *Computer Music Journal*, pages 32–38, 1977.

[35] A. Michael Noll. Cepstrum pitch determination. *The journal of the acoustical society of America*, 41(2):293–309, 1967.

[36] NumPy developers. NumPy. `http://www.numpy.org/`, 2018.

[37] Cian O'Brien and Mark D. Plumbley. Automatic music transcription using low rank non-negative matrix decomposition. In *25th European Signal Processing Conference (EUSIPCO)*, pages 1848–1852, 2017.

[38] Frank Opolko and Joel Wapnick. McGill University master samples. Technical report, McGill University, Montreal, QC, Canada, 1987.

[39] Antonio Pertusa and José M. Iñesta. Multiple fundamental frequency estimation using Gaussian smoothness. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2008)*, pages 105–108, 2008.

[40] Graham E. Poliner and Daniel P.W. Ellis. A discriminative model for polyphonic piano transcription. *EURASIP Journal on Advances in Signal Processing*, 2007(1), 2006.

[41] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, 2nd edition, 1992.

[42] Lawrence Rabiner. On the use of autocorrelation analysis for pitch detection. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25(1):24–33, 1977.

[43] SciPy developers. SciPy. `https://www.scipy.org/`, 2018.

[44] William A. Sethares. *Tuning, Timbre, Spectrum, Scale*. Springer Science & Business Media, 2005.

[45] Siddharth Sigtia, Emmanouil Benetos, Srikanth Cherla, Tillman Weyde, Artur S d'Avila Garcez, and Simon Dixon. RNN-based music language models for improving automatic music transcription. In *15th International Society for Music Information Retrieval Conference (ISMIR 2014)*, 2014.

[46] R. Sluyter, H. Kotmans, and A. von Leeuwaarden. A novel method for pitch extraction from speech and a hardware model applicable to vocoder systems. In *IEEE In-*

ternational Conference on Acoustics, Speech, and Signal Processing (ICASSP'80), volume 5, pages 45–48, 1980.

[47] Paris Smaragdis and Judith C. Brown. Non-negative matrix factorization for polyphonic music transcription. In *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 177–180, 2003.

[48] Mohan Sondhi. New methods of pitch extraction. *IEEE Transactions on Audio and Electroacoustics*, 16(2):262–266, 1968.

[49] David Talkin. A robust algorithm for pitch tracking (RAPT). *Speech Coding and Synthesis*, 495:518, 1995.

[50] John Thickstun, Zaid Harchaoui, Dean Foster, and Sean M. Kakade. Invariances and data augmentation for supervised music transcription, 2017. arXiv:1711.04845.

[51] Anja Volk, Frans Wiering, and Peter van Kranenburg. Unfolding the potential of computational musicology. In *Proceedings of the 13th International Conference on Informatics and Semiotics in Organisations*, 2011.

[52] Norman M. Weinberger. Music and the brain. *Scientific American*, 291(5):88–95, 2004.

[53] Mark C. Wirt. MIDIUtil: A pure python library for creating multi-track MIDI files. `https://pypi.org/project/MIDIUtil/`, 2018.

[54] Chunghsin Yeh. *Multiple fundamental frequency estimation of polyphonic recordings*. PhD thesis, Paris VI - Pierre and Marie Curie University, 2008.

[55] Jun Yin, Terence Sim, Ye Wang, and Arun Shenoy. Music transcription using an instrument model. In *IEEE International Conference on Acoustics, Speech, and Signal Processing. (ICASSP 2005)*, volume 3, 2005.