Note Onset Detection using Sparse Over-Complete Representation of Musical Signals

Master's Thesis submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Master of Science in Informatics
Embedded Systems Design

presented by
Mina Shehata

under the supervision of
Prof. Malek Miroslaw
co-supervised by
Eng. Rami Baddour, Prof. Toon Van Waterschoot

September 2013

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Mina Shehata
Lugano, 2 September 2013

To my beloved Family

" Music expresses that which cannot be said and on which it is impossible to be silent."

Victor Hugo

Abstract

Music is the language of the universe, one simple well known statement that says a lot. That is exactly the reason why it is an asset to understand this language that is performed and used by many people. If music is a language, digital signal processing would be the technical dictionary or translator for this language.

Researchers have always shown interest in understanding one specific property of this language which is *Note Onsets*. *Onsets* are chosen instants in time to represent transients in a musical signal. In other words an onset represents somehow the time when a musical note is played - a piano key is pressed - in a melody. The problem we are trying to solve throughout this research is detecting those note onsets. It is a kind of events detection within a musical signal. This is extremely beneficial for many applications such as automatic music transcription, adaptive audio effects and audio information retrieval.

Among the different methods existing in literature for solving this detection problem, one recently proposed method is based on the *sparse over-complete representation of musical signals*. Using this method for the onsets detection problem is not well established in literature and an efficient implementation is not yet available. It depends on the musical signal ability to be projected into another dimension with a sparser representation. Our aim is to make this representation somehow expressing the note onset positions.

The model we use for the musical signal is derived from the three components model: Sines + Transients + Noise. It considers any musical signal as a linear summation of those three components. The solution implementation consists of projecting the musical signal onto an over-complete dictionary that consists of two kind of atoms: steady-state and transient. Our guess was that the transient coefficients should be sparse and giving good indication of onsets positions. Some Peak-Picking algorithms are also proposed to produce a definitive set of onsets out of the output coefficients.

A complete testing platform was developed to be able to analyse and get preliminary results to investigate the efficiency of the proposed solution. It is designed to model the musical signal synthetically, to mix different notes from different instruments to be used in the analysis and to apply the projection and detection algorithms on the test signal. All of those tasks are made generic to be able to compare many implementations of the solution.

The results coming from the analysis of simple cases are promising and are giving high detection probability with low false-alarm probaility. Many parameters are discovered throughout

the research and some initial guess for their optimal ranges are studied. Finally, we benefit from developing a generic testing platform and we suggest some ideas for future work.

Acknowledgements

I constantly congratulate myself for such providential luck I have experienced and continue to experience living a wonderful colorful life. Because I grew up in a family incarnating the concept that “We are created as a gift for others” I find my satisfaction in making others happy. The reason “why I learned to give” was that I received many gifts.

I would like to express my gratitude to Prof. Miroslaw Malek, scientific director at Advanced Learning and Research Institute (ALaRI) for his great help, understanding and support during the period i spent in ALaRI.

I also thank my ALaRI father Eng. Umberto Bondi, the program manager for his support to complete my thesis research.

I take this opportunity to express my profound gratitude and deep regards to my guide Prof. Toon Van Waterschoot for his exemplary guidance, monitoring and constant encouragement throughout the course of this thesis. I really enjoyed working under his supervision. He is the reason why one day i would like to be a professor.

Thanks to Eng. Rami Baddour who helped me to understand how to do a research. I want to thank him for his support, advises, encouragement and being always present to help.

I am ever so grateful to my parents for having endowed me with versatile talents and a pair of keen eyes guiding me to enjoy life. For every single word i wrote in this research, i should thank my parents and my sisters for teaching me how to live and love.

Thanks to my friends Bassem, Sherif, Rami and Luca for their support and help specially to survive the difficult moments.

I also want to thank Claire my dearest friend, she gave me the light and power to accomplish this work.

Thank you God for your love.

Contents

Contents **xi**

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Signals and Signal Processing . . . 2
1.3 Digital Signal Processing . . . 2
1.4 Music as a Signal . . . 3
1.4.1 Acoustic Music Sources (instruments) . . . 3
1.4.2 Musical Signal Dimensions . . . 3
1.5 Scope and Contribution . . . 4
1.6 Organisation . . . 5

2 Problem Statement **7**
2.1 Events Detection . . . 7
2.2 Music Signal and Onsets . . . 8
2.2.1 An envelope model for isolated musical sounds: ADSR . . . 8
2.2.2 Transients versus Onsets . . . 8
2.3 Problem: Note Onset Detection . . . 9
2.3.1 Pre-processing . . . 10
2.3.2 Reduction Function . . . 10
2.3.3 Peak-Picking . . . 10
2.4 Applications . . . 11
2.4.1 adaptive audio effects . . . 11
2.4.2 parametric audio coding . . . 12
2.4.3 audio information retrieval . . . 12

3 State Of The Art **13**
3.1 Reduction Methods Depending on Signal Features . . . 13
3.1.1 Temporal Features . . . 13
3.1.2 Spectral Features . . . 14
3.1.3 Spectral Features considering Phase . . . 15
3.2 Reduction Methods Depending on Probabilistic models . . . 15

3.2.1 Model-Based Change Point Detection methods . . . 15
3.2.2 Surprise Signal Based Detection Methods . . . 16
3.3 Reduction Methods Depending on Outputs . . . 16
3.3.1 Based on linear prediction . . . 16
3.3.2 Extraction of tonal content . . . 16
3.3.3 Three components model: *Sines + Transients + Noise* (STN) . . . 18
3.4 Approach: Sparse Overcomplete Representation of signals . . . 20
3.4.1 Matching pursuit . . . 21
3.4.2 Global Optimization . . . 21

4 Implementation **23**
4.1 Modeling Musical Signal Synthetically . . . 23
4.1.1 Modeling changes among successive notes . . . 23
4.1.2 A more realistic model: Exponentially Damped Sinusoids . . . 25
4.2 Loading And Mixing Notes . . . 25
4.3 Reduction Function Implementation . . . 28
4.3.1 Basic Concepts . . . 28
4.3.2 Components and Formulation . . . 37
4.3.3 Parameters . . . 41
4.4 Peak-Picking . . . 43

5 Analysis, Results and Future Work **45**
5.1 Musical Notes Analysis . . . 45
5.2 Synthetic signal Analysis . . . 53
5.2.1 Single sinusoid with no events . . . 53
5.2.2 Simple sinusoid with one event . . . 54
5.2.3 More realistic events . . . 59
5.3 Use Case: Piano signal analysis . . . 68
5.3.1 Selecting Tonal Atoms count . . . 68
5.3.2 Define a range for exactness Lambda (λ) . . . 68
5.3.3 Tuning sparsity lambdas (λ_t and λ_{ss}) . . . 71
5.3.4 Peak-picking . . . 76
5.4 Conclusion . . . 79
5.5 Future work . . . 80

6 Business Plan **81**
6.1 Executive Summary . . . 81

A Appendix **85**

Bibliography **107**

Figures

2.1 (*A*)ttack, (*D*)ecay, (*S*)ustain and (*R*)elease model (an isolated accordeon note). . 9

3.1 Parametric representation: Signal is a **parallel** sum of sinusoids + noise. 17
3.2 Subspace Projection: Signal is a **series** sum of damped sinusoids. 19

4.1 Modeling the input signal possible properties. 24
4.2 A more realistic model: Three successive exponentially damped sinusoids. 25
4.3 Different examples of musical notes loaded by the mixTones function. 26
4.4 Different examples of musical pieces generated by the mixTones function. 27
4.5 Some of the DFT atoms for N=100. 31
4.6 The DCT atoms for N=10. 33
4.7 The *Unit Ball* for each of the chosen L_p-norms. 35
4.8 Intersection between L_2-norm unit ball and the projection space vector *A*. 36
4.9 Intersection between L_∞-norm unit ball and the projection space vector *A*. . . . 36
4.10 Intersection between L_1-norm unit ball and the projection space vector *A*. 37
4.11 Alternating Direction Method. 40

5.1 Single note analysis piano versus accordeon. 47
5.2 Piano note frequency content (SS versus T). 48
5.3 Accordeon note frequency content (SS versus T). 49
5.4 Best Onset detection time (Piano note - frame size =500, overlap = 50%). . . . 50
5.5 Frame Size and Overlap for piano signal processing. 52
5.6 Projection of a single sinusoid with no events(LASSO-ADMM) using DCT atoms. 55
5.7 Projection of a single sinusoid with no events(YALL1) using DCT atoms. 56
5.8 Projection of a single sinusoid with no events(LASSO-ADMM) using DFT atoms. 57
5.9 Projection of a single sinusoid with no events(YALL1) using DFT atoms. 58
5.10 Synthetic input: sinusoid + one pulse (at t=1200). 59
5.11 [a,b] Projection of a single sinusoid with one pulse events(LASSO-ADMM). . . . 60
5.10 [c] Projection of a single sinusoid with one pulse events(LASSO-ADMM). 61
5.11 [a] Projection of a single sinusoid with one pulse events(YALL1). 61
5.10 [b,c] Projection of a single sinusoid with one pulse events(YALL1). 62
5.11 [a,b] LASSO-ADMM projection for different feature change (separate). 64
5.10 [c] LASSO-ADMM projection for different feature change (separate). 65
5.11 LASSO-ADMM projection for different feature change (mixed). 65
5.12 [a,b] YALL1 projection for different feature change (separate). 66
5.11 [c] YALL1 projection for different feature change (separate). 67

5.12 YALL1 projection for different feature change (mixed). 67
5.13 Max sparsity and no events coefficients ($\lambda = 0.05$). 69
5.14 Best sparsity ($\lambda = 0.15$). 69
5.15 Tonal coefficients less sparse ($\lambda = 0.95$). 70
5.16 More sparse events coefficients ($\lambda = 0.12$). 71
5.17 Less sparse events coefficients ($\lambda = 0.16$). 72
5.18 Choosing Lamda Tonal ($\lambda = 0.12, \lambda_{ss} = 1.1$). 72
5.19 Choosing Lamda Tonal ($\lambda = 0.12, \lambda_{ss} = 1.2$). 73
5.20 Choosing Lamda Tonal ($\lambda = 0.16, \lambda_{ss} = 1.1$). 73
5.21 Choosing Lamda Tonal ($\lambda = 0.16, \lambda_{ss} = 1.2$). 74
5.22 Choosing Lamda Tonal ($\lambda = 0.16, \lambda_{ss} = 1.3$). 74
5.23 Choosing Lamda Transients ($\lambda = 0.16, \lambda_{ss} = 1.3\ \lambda_t = 1.9$). 75
5.24 Choosing Lamda Transients ($\lambda = 0.16, \lambda_{ss} = 1.3\ \lambda_t = 4.4$). 75
5.25 Comparing Sum_U with a threshold. 76
5.26 Comparing Sum_U with a threshold using note spacing. 77
5.27 Comparing U with a threshold (First). 78
5.28 Comparing U with a threshold (Local Maxima). 78

Tables

4.1 Specifications of piano notes mix 28
4.2 Specifications of accordeon notes mix 28
4.3 Pairs of Frame Size and Overlap 41

5.1 Tonal Atoms count choice 46
5.2 Lambda (λ) range check. 70

Chapter 1

Introduction

Music introduces and accompanies several moments of our life. In this chapter let us first highlight the field of this research which is music applications and the reason behind exploring this area. We will also present some basic concepts to help the reader understanding subsequent chapters. The last two sections will describe the scope of the work, contribution and the thesis organisation.

1.1 Motivation

> *If I were not a physicist, I would probably be a musician. I often think in music. I live my daydreams in music. I see my life in terms of music.*- Albert Einstein

Music is not a phenomenon it is a part of human life. It is not a new fact but it is a friend of human beings since the begining of creation. Just to feel that it is really old, ancient egyptians credited the music invention to *Thoth* the god of entertainment [Wikipedia, 2013]. Humans first percieved it by listening to nature. Then they tried to regenerate it using their voice, body or some tools from nature like woods, bones and tree leaves.

With humanity evolution, music also evolved tremendously from middle ages music (476 to 1400) to Baroque (1600 to 1750) then Classicism (1750 to 1830) and finally a huge amount of styles of the twentieth and twenty-first century: jazz, rock, pop, blues, country, latin, oriental ....

The evolution did not only touch the music style and type but also music activities and what we can do with it. From listening to performing generating the need for composition and production. One emerging activity is music processing which came out in the last hundred years accompanying the technology revolution. First electronic music was created in mid 1940s. When the general purpose computing emerged, the number of music synthesisers - used to compose electronic music - growed. And lastly the revolution came out with **Music Signal Processing** which is a highly computationally demanding activity. That is why it came out later and it is growing very fast.

This fast growth in music processing is due to the revolution introduced by replacing the analog with digital signal representations. To understand the importance of music and which benefits we get by developing new music processing techniques we should ask ourselves four questions: *When* are we listening to music ?, *Where* do we want to listen to music ?, *How*

or the way we use to listen to it -listening to live music or recorded music, listening using loudspeakers or headphones, in a silent or noisy environment, ...? and For *Which* reason do we want to listen to it ?. All those questions have one answer: *Infinity*. There are an infinity of ways, places, conditions, ...which make music very present in our life. This answer captured the focus of many researchers towards music.

That is why it is very interesting to spend time on exploring and adding to this digital music world. The continuous increase and evolution of digital music means an equivalent rise in the demand for digital music processing. For that reason, innovative techniques are needed to efficiently perform, store, transmit and enhance digital music.

1.2 Signals and Signal Processing

In our daily life **signals** are one of the most encountered phenomena. Speaking is producing a signal that will be received by others through hearing it by means of their auditive receptive system. It could also be received by a microphone. Music could be thought in a similar way as a signal produced through speakers or musical instruments. Pictures and videos are other kinds of signals that are sent and perceived in a different way. A lot of mechanical systems and natural phenomenas like earthquakes produce vibrations which are also signals.

Mathematically speaking a *signal* is a function of one or more independent variables like time, pressure, position, and distance. Some signals are one-dimensional. For example music which will be the main topic of our research is representing variation of the pressure density as a function of time. Other signals have more dimensions like black and white images. A black and white image is a two-dimensional signal representing light intensity as a function of the spatial coordinates (x, y) independent variables.

Signal processing is the art of extracting the information contained in signals and in some cases modifying it using some operations. Applying signal processing requires first to define[Mitra, 2001]:

- **The signal representation:** in the domain of the original independent variable or in a different one. Domains could generally be time or frequency but they could be more precise. For example a frequency representation could be using the sines, cosines or complex basis.
- **The processing domain:** similar to the representation the algorithmic transformation applied to the signal could be in any of the domains.

Information extraction and modification depend on signal type and what kind of information to be extracted. Scaling, amplification, filtering, multiplexing, modulation and signal separation are examples of typical processing operations.

1.3 Digital Signal Processing

As mentioned in the previous section we need to define the data representation and the processing algorithm domain. DSP is a particular kind of processing using the *digital* domain for both parameters. Digital means discretization of both parameters.

- **Independent Variable** is *discrete*. The signal is sampled with respect to the independent continuous variable. Music signals are sampled at distinct points in time.

- **Signal value** is *discrete*. This is called the quantization process where one sample is allowed to assume one value from a set of distinct values. These values are to approximate the original signal values with a certain precision.

This leads to a binary representation for the signal.

1.4 Music as a Signal

To be able to analyse a musical signal we need first to characterize it from a signal processing perspective. First we'll try to discover its source and classify different acoustic instruments; then we'll present its dimensions.

1.4.1 Acoustic Music Sources (instruments)

To dsicover what a music signal is we need to know how it is generated. A good explanation was given by [Mitra, 2001]: "The natural sound generated by most musical instruments is generally produced by mechanical vibrations caused by activating some form of oscillator that then causes other parts of the instrument to vibrate. All these vibrations together in a single instrument generate the musical sound." Such explanation matches with the general definition of the *Sound* being a mechanical wave caused by vibrations (change in pressure) within some medium and being in the range of human hearing.

One classification for the musical signals could be in respect to their steady-state portion of the note. It is simply the portion of the note that contains some time-invariant frequency components. The different portions of a musical note will be explained in section 2.2.1 in more details. This quick classification has two groups[Mitra, 2001]:

- **Quasi-periodic**: This is the group of signals that have a larger steady-state portion in their notes. This would be signals generated from instruments like violin, piano and guitar.
- **Aperiodic**: This is the other group that almost does not have at all a steady-state portion. In that case the instruments are like drums and some percussions.

This classification is not that clear everytime we differentiate between instruments. In this research we will use another classification that is better suited for the *note onset detection* problem.

1.4.2 Musical Signal Dimensions

Here is a brief explanation of which so-called music dimensions as described in [Muller et al., 2011]:

I. **Musical Pitch**

This is a very important music characteristic that describes in a precise way instruments that generate notes with a well defined fundamental period. That means a harmonic series of sinusoids at the multiple of the fundamental frequency of the note. This is well noticed in *Wind* instruments like flutes, trumpets and clarinets or *String-based* instruments like pianos, violins and guitars. Alternatively we have also some *unpitched* instruments like drums and percussions. As it will be discussed later, this is one of the main reasons why music relates to events. It could be defined as a sequence of single pitched notes or events.

Some important notions based on the **pitch** definition that would help in defining the use cases are shown here:

- if the **fundamental frequency** of a note is f_0 the harmonics of this note are at $f_1 = 2f_0, f_2 = 3f_0, f_3 = 4f_0, \ldots$.
- **Octave equivalence** is used to state a fact about human auditive perception. From a human point of view two notes having their fundamental frequencies relation $2:1$ are very similar. This relation applies for an *Octave* difference hence the naming.
- a *Scale* is a culture or style specific sequence of a set of pitches that repeats every octave. For example the *Western Major* scale is defined by the well known sequence: $Do, Re, Mi, Fa, Sol, La, Si$.
- a **monophonic** music signal is formed by a sequence of pitches which is said to form a **melody**.
- **polyphony** instead is when multiple notes (having different fundamental frequencies) are played simultaneously. this forms the basis for musical harmony.

II. **Musical Harmony**

In few words musical harmony occurs when mixing more than one pitch at a time. It forms the basis for chords in music and somehow controls the music color. It requires a more sophisticated analytical study and is not within the scope of this research.

III. **Musical Timbre**

This is a property related to the auditory sensation. For simplicity let us use this example to briefly make it understandable: Listening to a violin and a flute having the same loudness and pitch, the human being is still able to differentiate between them. That is the timbre property. Timbre also relates to the (relative) amplitudes of the different harmonics that are present in a monophonic music signal (also referred to as the spectral envelope).

IV. **Musical Tempo and Beat**

Those properties are the ones responsible for making the music move forward. As stated in [Muller et al., 2011] they define the temporal framework of the musical piece. Beats are somehow the pulses of the music and tempo is defined as the rate of these pulses. **Onsets** which are the main purpose of our research are somehow related to musical beats in their definition. Both mark the start of played notes in a melody. But they are different when speaking about their timing properties and hence their detection techniques. In a constant tempo, beats are occurring periodically, and hence it is not so difficult to detect them. On the other hand, onsets might occur in arbitrary time instants, and are hence much more difficult to detect. Onsets only coincide with beats for those instruments that are intended to generate the musical rhythm, such as drums.

1.5 Scope and Contribution

It is very interesting to understand and manipulate music signals. Focusing on the *Quasi-periodic* group of music instruments, we want to investigate the ability to capture some information related to their note onsets.

In this research we are focusing on one of the emerging music processing problems which is the *Note Onset Detection Problem*. Finding an efficient solution for this problem would help a lot in many music applications like automatic music transcription, audio information retrieval and applying adaptive audio effects. We will investigate a new approach based on *Sparse Signal Representation* to solve the presented problem. Some papers suggested already this method for different music processing applications. Unfortunately there are no much information about the efficiency of applying it to resolve the *Note Onset Detection Problem*.

We will implement this approach to have some preliminary results about its efficiency for solving our specific problem. Our objective is to build a complete testing platform to be able to model the input signal, analyse the algorithm efficiency and compare different solutions varying the input parameters. We will focus our attention on two types of musical notes: the pitched percussive like piano notes and the pitched non-percussive like accordeon.

1.6 Organisation

Having introduced the research area in chapter 1, chapter 2 will discuss the problem in details and the benefits we get by solving it. In chapter 3 the solution process together with a classification of the existing solutions in literature are presented. This chapter will also present the approach proposed by this thesis and the chapter 4 presents step by step the full implementation of this solution. Chapter 5 presents the analysis results together with the conclusion and the possible future work. A business plan is presented in chapter 6.1. Finally in Appendix A parts of the source code of the testing platform are presented.

Chapter 2

Problem Statement

Music is an event-based phenomenon for both performer and listener[Bello et al., 2005]. For a performer to play a piece of music he needs to play a sequence of notes or change note's timbre or color. And the same from a listener point of view, he is usually noting the transitions between subsequent notes, timbers and colors occupying the successive periods of time. Therefore music is most likely meaningful when having transitions, when having **events**.

This chapter starts by explaining events detection in general. Sections 2.2 and 2.3 describe in details the kind of events we are interested in when dealing with music and a possible process for their detection. Finally section 2.4 gives some examples of the applications that will benefit from this algorithm.

2.1 Events Detection

"Events are things that happen" This is a short and simple philosophical definition as described by [Casati and Varzi, 2010]. Those things could be smiles, sounds, explosions, decisions. They can be of many kinds and follow many segmentations. For example they can be thoughts, actions or things that are perceived. Speaking about *events detection* we will be mostly interested in the kind of things that are understood through human perception.

A lot of applications and researches nowadays try to imitate human perception. Making the machine more intelligent and being able to perceive events, understand them and sometimes take actions in response is a very interesting evolution and humanity profits more and more from this. Another word that describes event detection in the domain of computer science is **abstraction**. Extracting semantics from a large set of data aims at reducing the data set and maintaining only a relevant subset.

One could consider the human being as a very sophisticated system full of sensors and actuators. That's why we should be intelligent in imitating this complex system in order to achieve a faster increase in the efficiency. When abstracting the events that are sensed we need to take into consideration some factors:

- What is the application-specific information that needs to be extracted?
- What is the perception precision of the receiver –in our case is the human being? That means we should consider the events that have higher perceptual effect.

2.2 Music Signal and Onsets

In reality audio signals are much more complicated to deal with. They are noisy polyphonic signals which means that the processing needs to consider more than one source and that some of them - noise sources - are irrelevant and need to be filtered out. One way to describe them is that they are additive and oscillatory. An audio signal results from the sum of many sinusoidal signals, each group of them is coming from a different source. This description suggests that the processing would not consist in looking to the signal in the time domain only but instead the signal needs to be represented in a different domain. The outcome of this transformation is called the detection function [Bello et al., 2005]

Most of the research in sound analysis and synthesis has been focused on the tonal part of the music signals (locally steady-state) which holds information about the pitch. Recently researchers have shown interest in analyzing the transients in music signals because it appears that these have more weight from the human auditive perception point of view. It is very difficult to define what *transient* is and where exactly it exists [Daudet, 2006]. Let us first try to understand how the different parts of an isolated music note are represented and how this could help to define transients.

2.2.1 An envelope model for isolated musical sounds: ADSR

ADSR is a model for the isolated musical sounds (notes) generated by acoustic musical instruments. It is a well known modeling scheme used in synthesizers and musical instruments. It is modeling the evolution in the amplitude envelope of the played note by a musical instrument [Brossier et al., 2004]. In other words the ADSR is expressing the change of short-term signal power over time. The name comes from the abbreviation of the model components Attack, Decay, Sustain and Release. Figure 2.1 shows this model applied on an accordeon note. The transient is containing the attack an decay parts. The rest is assumed to be the steady-state portion of the signal.

Attack, decay and release are all time parameters while the sustain is a level parameter. Those time and level parameters are instrument-dependent [Gu, 2013].

- **Attack**: It represents how much time is needed by the signal amplitude (a played note) to reach its maximum value. As we are studying acoustic instruments, the attack time is very short for this kind of input signals. That is what makes the detection for transients in acoustic signals very challenging.
- **Decay**: This is the time to go from the highest reached level to the sustain level.
- **Sustain**: This is the constant or steady-state level at which the signal stays for some time until the note is released.
- **Release**: Finally this is the time during which the signal fades out from the sustain level until theoretically reaching the zero level.

2.2.2 Transients versus Onsets

After having an idea about the evolution of an isolated note, let us now consider the terminology that we need for our specific problem. As it will be discussed later, the detection algorithm - we are developing throughout this research - goal is to detect transients.

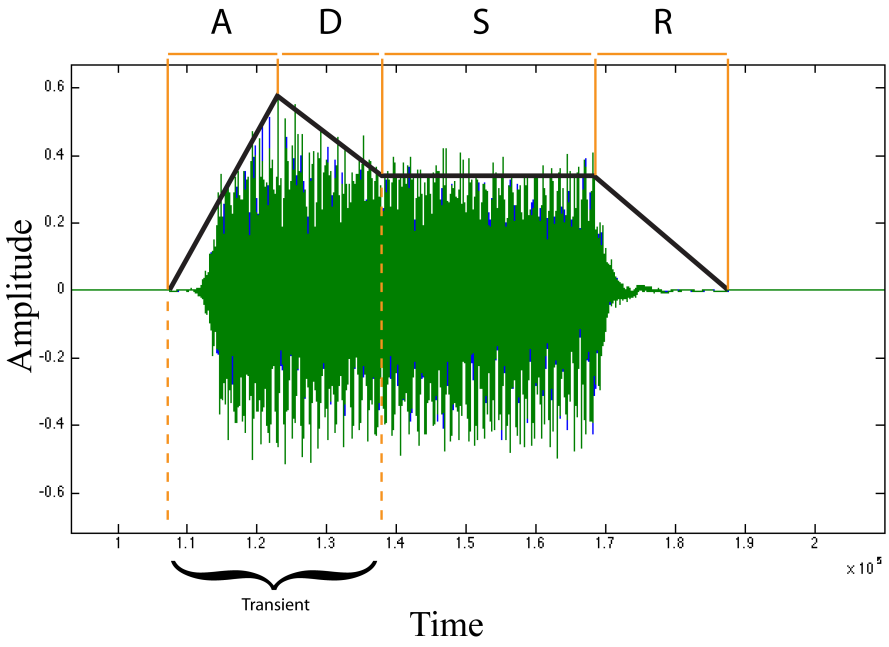


Figure 2.1. (*A*)ttack, (*D*)ecay, (*S*)ustain and (*R*)elease model (an isolated accordeon note).

- **transient**: short interval of time in which the signal energy or statistical properties change rapidly. It is the part of the signal representing the excitation and not the resonance. To relate to the previous model, transients are the portion of the note that contains the *attack* and *decay* parts. That is intuitive because it is the portion of the note where the energy content increases then decreases as shown in figure 2.1. Speaking about auditory perception, the human ear could be assumed to distinguish between transients that are more than 10 ms apart [Moore, 2003].

- **onset**: is the chosen instant to represent a transient. It is chosen to be as close as possible to the start of the transient[Bello et al., 2005]. As it is representing transients or events, it is representing discontinuities. This is the main idea behind most of the approaches used to search for onsets: searching for discontinuities in amplitude, frequency or phase.

2.3 Problem: Note Onset Detection

From the previous section we conclude that *onsets* are somehow the physical representation of the start of notes or music events. Therefore the *note onset detection* problem could be defined as "looking for the transient part of a signal (musical signal), a short period of time in which the signal energy or statistical properties change rapidly ". In this section it will be briefly described the three steps needed for the overall detection process [Bello et al., 2005]. Starting by explaining the pre-processing phase then the reduction algorithm and finishing with the peak-picking phase. More attention will be given to phase I and III as they will not be investigated in details later, their only mission is to ameliorate the algorithm efficiency. On the other hand phase II is the main focus of this research. At this point after understanding what are events,

detection and onsets, phase *II* will describe the problem we are trying to solve as a function of those basics.

2.3.1 Pre-processing

It is simply preparing the signal before undergoing the detection function by emphasizing some aspects which make it easier to detect the onsets.

I. **Multiple Bands technique**

   Instead of dealing with the signal as a single entity, this technique divides it into sub-bands and applies the reduction (detection) function to each sub-band separately. The final output is then calculated by combining the outcomes of each of the sub-bands individual processing. The difference in the efficiency could be noticed as in the method proposed by [Duxbury et al., 2002]. They applied different reduction criteria for different sub-bands. They used energy changes with higher frequencies and spectral changes for the lower ones.

II. **Transient/Steady-State Separation**

   This technique will not be discussed in details here as it is the starting point for the solution that is proposed through this thesis. In few words, it is based on modeling the tonal component of the signal and retrieving the transient component by getting the residual. The residual is calculated by subtracting the modeled tonal part from the original signal. This technique uses many methods for modeling like: Spectral Modeling Synthesis (SMS), Transient Modeling Synthesis (TMS) and lapped orthogonal transforms.

2.3.2 Reduction Function

This is the other name for the **Detection Function** because it is in a way a downsampling of the original signal. This is the main point for this research because a significant part of event detection is finding a way to reduce the amount of data to represent a signal - i.e. representing the signal with some parameters. Actually event detection could be defined as intelligent reduction of the input signal. It is smart because we need that the detected events could be able to fully or partially represent the original signal.

For completeness the detection function is also called **Novelty Function** when speaking about onsets. It is like searching for something new. We call the **reduction function** output **novelty curve**, and usually the peaks of this curve represent the onsets.

So the problem is to find an effective way to deduce music events from a piece of music. Those music events are represented physically by onsets. The solution to this problem is easier in the case of percussive instruments as the transients are very often detected through peaks in the amplitude envelope. In case of non-perussive instruments, this problem become much more challenging. In the next chapter 3 we state some of the state-of-the-art reduction algorithms that give possible solutions to the problem.

2.3.3 Peak-Picking

The complexity of this step in the onset detection algorithm depends on how well did the reduction function performs. In a very simplified way of thinking it is searching for local maxima. In practice it could be divided into three steps:

I. **Post-Processing**

This is an optional step to prepare the novelty curve to be processed. It is essentially used to enhance the consistency of the detected features representing events. It is used for two main reasons: Decreasing the noise effects by *smoothing* and making the thresholding useful to a wide range of signals/applications through *normalization* and *DC removal*.

II. **Thresholding**

This step is to differentiate between peaks that are related to events and irrelevant peaks. It could be *Fixed Thresholding*: comparing the novelty curve with a constant threshold chosen before running the algorithm. Alternatively *Adaptive Thresholding* is used. This is more suitable for signals that have higher dynamics. It adapts the threshold to the overall variation of the signal (for example quiet and loud passages in music).

III. **Peak-Picking**

Finally in this step the algorithm searches for the local maxima.

2.4 Applications

Abstracting music signals to a set of events is very useful for many music applications to cope with the technology evolution nowadays. The search engines need faster and smarter ways to search for a music file among the huge number of music files existing in the web servers. Being able to represent music more intelligently by a sequence of events could tremendously decrease the time used by the retrieval algorithms. Another important application would be the compression, which becomes more efficient knowing more information about the signal subject to the compression. Content delivery will benefit a lot from these intelligent compression techniques. Not only music customers can benefit from this, but the music industry itself can make use of this kind of music representation. It can make the music editing more signal-adaptive [Bello et al., 2005]. Here are some of the proposed applications.

2.4.1 adaptive audio effects

An audio effect is simply modifying or modulating an audio signal. This transformation becomes smarter when being adaptive. An adaptive effect is a time-varying control depending on some of the input signal features. That is why it offers a meaningful and coherent type of effects Verfaille et al. [2006].

For example enhancing or suppressing the transients for an input signal [Goodwin and Avendano, 2006]. Applying the filtering separately to the transient and the tonal - steady-state - parts gives much better quality for the effects and decreases the introduction of spectral artifacts from one component to the other. One good example is the time stretching effect using phase-vocoder-like methods. The time scaling needs to be applied differently to tonal and transient parts. While the tonal part needs to be stretched normally, transients need to be shifted to match the new positions but remain localized in time and not stretched to maintain their properties [Duxbury et al., 2001].

2.4.2 parametric audio coding

Parametric coding means measuring some of the vital statistics of the audio signals and storing them as descriptive information rather than coded audio signals.

The fact that the tonal and transient parts are encoded separately is not only facilitating the signal analysis but it is offering more robustness against noise. By this newly introduced coding scheme, the residual is not only representing noise but a totally new model is used consisting of sines+ transients + noise [Verma et al., 1997].

2.4.3 audio information retrieval

In the digital world almost everything is abstracted to zeros and ones in order to maintain the information they provide in a physical form. The same applies to audio signals, it is encoded with different schemes.

Now it is obvious that by encoding the signal more intelligently it makes life much easier for the search engines. Instead of searching with a large amount of samples, it is sufficient to search with a small amount of data. Therefore onsets count and position could be used as search keys [Daudet, 2006].

Chapter 3

State Of The Art

Searching for an efficient and innovative solution for the *Note Onset Detection* problem, we found different ideas for the **novelty algorithm** - the algorithm searching for transients. This chapter presents the state-of-the-art reduction algorithms classified in three different ways: first with respect to the signal features then to some probabilistic models [Bello et al., 2005] and finally a classification that concentrate on the algorithm output [Daudet, 2006].

3.1 Reduction Methods Depending on Signal Features

3.1.1 Temporal Features

The main and most simple way to detect onsets is the change in the signal's amplitude envelope. It is very significant for a large number of applications whose purpose is to detect events. For example in detecting anomalies in vibration patterns, most probably events coincide with a rise in the amplitude (or energy) envelope of the signal. The same happens in music played by percussive instruments in which almost every note starts with an increase in the amplitude envelope. Techniques based on temporal features include

- **Envelope Follower**: This is done by rectifying the input signal then smoothing it.

  Let n be the discrete time index and the input music signal is represented with the vector $x(n)$. The input is divided into frames having N samples then the reduction algorithm processes frame by frame. The envelope follower function noted by $E(n)$ is considered as the novelty function in our solution. It is mathematically defined as:

$$E(n) = \frac{1}{N} \sum_{m=-\frac{N}{2}}^{\frac{N}{2}-1} |x(n+m)| w(m). \tag{3.1}$$

  As equation 3.1 shows the *rectification* is first applied as $|x(n+m)|$ then *smoothing* is applied by simply averaging over a window $w(m)$ having N points which is the frame length.

- **Monitoring the Energy**: Instead of the rectifying step in the *envelope follower* it squares

the input before smoothing it. This gives a measure for the short-term signal energy.

$$E(n) = \frac{1}{N} \sum_{m=-\frac{N}{2}}^{\frac{N}{2}-1} [x(n+m)]^2 w(m). \tag{3.2}$$

- **Monitoring the change in the Energy**: This is achieved by taking the *derivative* $d(E)/dn$ of the signal energy which makes the peaks resulting from the reduction narrower therefore easier to abstract to onsets.
- **Monitoring the derivative of the *logarithm* of the energy**: Takes into consideration how human ear perceives loudness. This method helps to reduce the generation of spurious onsets.

Those techniques are simple and computationally efficient but they are very dependent on the processed signal. The signal should contain clearly identifiable energy increases identifying the transients. Even though it could perform efficiently with percussive signals, it suffers when facing the amplitude modulations like vibrato or the increase in the amplitude in case of the superposition of many sources. These shortcomings are not overcome by using multiple band preprocessing.

3.1.2 Spectral Features

Using this feature is somehow more computationally efficient because it can give good results without the need of pre-processing. Two main approaches are discussed:

I. **Monitoring the magnitude of the spectrum**

Analyzing the signal spectrum, the energy is concentrated at low frequencies for the major part of the signal corresponding to the steady-state portion of the note. Energy appears at higher frequencies when detecting transients. *Short Time Fourrier Transform* (STFT) could be used together with a weighting function that gives more weight to the higher frequencies. In that case the reduction algorithm will hopefully generate peaks at transient (onset) positions. One implementation of this approach is the **High Frequency Content** (HFC) feature proposed by [Masri, 1996] which uses $W_k = |k|$ as weighting function.

$$\tilde{E}(n) = \frac{1}{N} \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} W_k |X_k(n)|^2. \tag{3.3}$$

Equation 3.3 shows the generic function producing the weighted spectrum energy where $X_k(n)$ is the frequency spectrum - calculated using STFT - of one frame. $\tilde{E}(n)$ is different from $E(n)$ in that it is calculated from the energy of the input spectrum and not from the time-doamin input. As we assume that transients increase the higher energy coefficients, using $W_k = |k|$ will result in higher $\tilde{E}(n)$ when having transients.

II. **Monitoring the change in the spectrum magnitude**

Here instead, not the energy itself but the difference in the energy content between successive bins is monitored. In this way the evolution of the signal over time is taken into

consideration. As an example of the difference function the one proposed by [Duxbury et al., 2002] is given here,

$$\bar{E}(n) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} \{H(|X_k(n)| - |X_k(n-1)|)\}^2, \tag{3.4}$$

Where

$$H(x) = \frac{(x+|x|)}{2}. \tag{3.5}$$

This function is using the L_2-*norm* distance and is assuming transients exist when there is an increase in the energy content from one frame to the next one. Using the function $H(x)$ is to take into consideration the increase in the spectrum energy and neglect the decrease.

Those methods are good for percussive signals but less efficient with the low-pitched and non-percussive ones. That happens because transients in those signals are not necessarily reflected at higher frequencies. So it will be difficult to distinguish transient from steady-state components.

3.1.3 Spectral Features considering Phase

These features provides an enhancement to the previous features by also taking into account the phase variable. The difference between the phases of subsequent bins is expected to be almost constant in the steady-state portion of the note. This could be mathematically defined by the second difference of the phase as in [Bello et al., 2005]

$$\Delta\varphi_k(n) = \varphi_k(n) - 2\varphi_k(n-1) + \varphi_k(n-2) \approx 0. \tag{3.6}$$

where $\varphi_k(n)$ is the 2π-*unwrapped* phase of the STFT coefficient $X_k(n)$.

For transients the $\Delta\varphi$ is expected to have significant values. Using this feature is very sensitive to the preprocessing. If the phase information is somehow affected by the preprocessing like tonal removal the detection performance will be affected.

The phase-based methods are good in being less dependent on the pitch of the events. In theory it could detect events reflected in either higher or lower frequency bands. Unfortunately it suffers from the noisy low-energy components and phase distortions due to preprocessing.

3.2 Reduction Methods Depending on Probabilistic models

This approach could only be efficient depending on the ability to construct some probabilistic models for the signals under processing. Then the detection algorithm role is to capture deviations from these models. There are two different techniques in literature [Bello et al., 2005] :

3.2.1 Model-Based Change Point Detection methods

This method expects that the signal is following one out of two models at a specific time - period. By monitoring the switch from one model to the other the onsets are detected. One example is to use the log-likelihood ratio of the probability density functions of the two models.

Let P_{SS} and P_T denote the probability density functions for the steady-state (SS) and transient (T) model respectively. The log-likelihood function would be:

$$S = \log \frac{P_T}{P_{SS}} \tag{3.7}$$

The sign of the function S is considered to be the *novelty function*.

3.2.2 Surprise Signal Based Detection Methods

Alternatively this method assumes the whole signal as a single probabilistic model and interpretes any deviation from this model as an onset.

3.3 Reduction Methods Depending on Outputs

3.3.1 Based on linear prediction

This method is efficient for signals having their steady-state component predictable. It considers the signal as a result of filtering an excitation signal by an all-pole filter. This all-pole filter should be representative of the instrument generating the signal. Using the Yule Walker equations we can estimate the parameters for this filter. By constructing the inverse of this filter (stable because it will be an all-zero filter) and applying it to the signal, it produces a sequence of transients. The method based on linear prediction works well with instruments that generate sounds as an excitation plus its resonance through the instrument body.

Three main issues for this method:

- How to estimate the filter order.
- Which training set to use for estimating the filter coefficients.
- The estimation of the filter coefficients should be done on an isolated note (since each note should be represented by different filter coefficients), and so this method actually requires the notes to be already separated from each other.

An enhancement would be estimating a *pole-zero* filter instead of the *all-zero* one. This gives better results but makes the estimation process much more complex.

3.3.2 Extraction of tonal content

One way of extracting what is required - *transients*, is by extracting what is not required - *tonals* - and subtracting it from the original input. The residual would practically contain the transients with some noise. This kind of methods are assuming a model for *tonals* and not for *transients*. Here we present different methods applying this concept:

I. **Segmentation to Transient (T) or Steady State(SS) after applying STFT**

The signal is analyzed by Short Time Fourier Transform. Every Time-Frequency bin is marked as T or SS depending on certain criteria related to the instantaneous phase evolution. Actually for the steady-state portion of the signal the phase increases or decreases

slowly. The phase is predicted for the subsequent bin, then compared with the measured value.

Let $\phi_{n,k}^{\text{pred}}$ and $\phi_{n,k}$ represent respectively the predicted phase of the next time-frequency bin and the measured one for the same bin. Every Time-Frequency bin is then marked to be steady-state (SS)

$$\text{if} \quad |\phi_{n,k}^{\text{pred}} - \phi_{\text{n,k}}| < \varepsilon$$

where ε is a threshold for the maximum change in phase for steady-state portions of the signal. Otherwise the bin is marked transient (T).

Then the SS and T components could be easily reconstructed from the corresponding bins only. This technique was initially used for transient/steady-state separation. To use it for the onset detection problem, it is convenient to have a more complex measure. For example we could consider the amplitude change between the time-frequncy bins in addition to the phase change.

II. **Parametric representation of the signal**

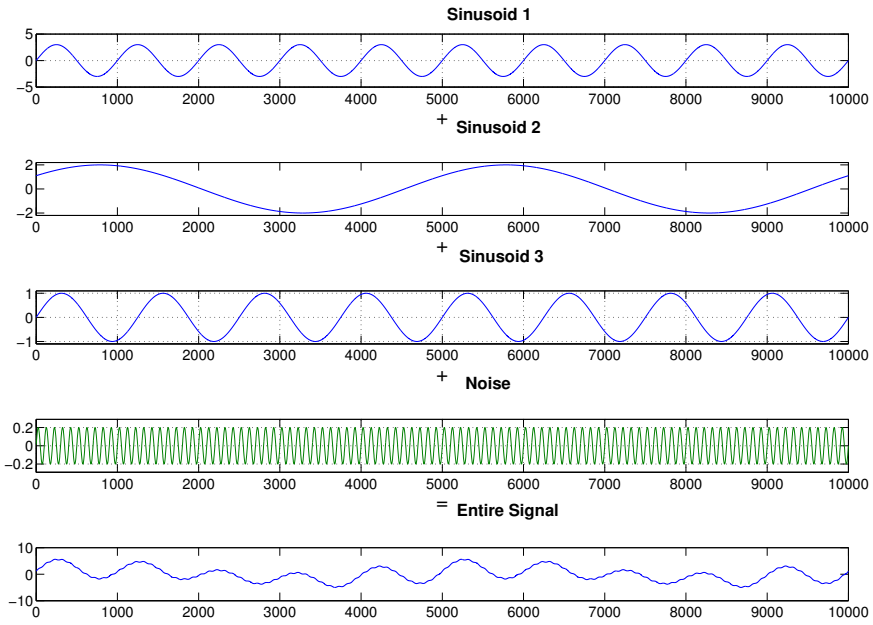


Figure 3.1. Parametric representation: Signal is a parallel sum of sinusoids + noise.

The basic assumption is to consider the steady-state portion of the signal as the sum of sinusoids with different parameters. This is somehow a parallel summation of harmonics. McAulay-Quatieri proposed a model for speech that could be refined to handle music signals [McAulay and Quatieri, 1986].

$$x(t) \approx \sum_{i=1}^{I} A_i(t)\, sin(\varphi_i(t))$$

$$\varphi_i(t) = \int_0^t w_i(\tau) d\tau + \varphi_i(0)$$

Where I is the number of sinusoids in the model, A_i is the amplitude of the i-th sinusoid, w_i its frequency and φ_i its instantaneous phase. The idea is simply explained in figure 3.1 which shows the addition of three sinusoids with different parameters -frequency, amplitude and phase - together with a noise signal. Those parameters again are supposed to evolve slowly for each of the sinusoids forming the signal. They are estimated for each frame and their increase/decrease is checked. At the end the parts of the signal containing transients will not be well estimated. These residuals will represent the transients.

III. **Subspace projection**

Finally, this is another way to model the musical signal can be represented as a series of notes. It assumes the signal as the sum of successive exponentially damped sinusoids. It is a reasonable guess because it uses the fact that the free response of oscillatory systems decreases exponentially. The model is expressed mathematically

$$x(t) = \sum_{j=1}^{J} A_j z_j^t + n(t)$$

A_j is the amplitude of exponentially damped sinusoid j and $z_j = e^{\delta_j + i2\pi f_j}$ where f_j is the sinusoid's frequency and δ_j is its damping factor. An additional term - *noise* - $n(t)$ which represents the residual of the model which is not shown in the explanation figure 3.2. This technique typically gives much better resolution than the Fourier Transform. On the other hand it faces a limitation which is the estimation of the number of components of the signal. The subspace projection is best done with isolated notes otherwise it is not sure that all harmonics decay with the same ratio.

This specific approach is used during this research to construct a synthetic musical signal to undergo the preliminary tests. That will be described in details in section 4.1.2.

Actually examples described in II and III can be interpreted either as a parametric representation or as a subspace projection. In other words, the parametric representation is not limited to the sinusoidal model and the subspace projection should is not limited to the damped sinusoidal model.

3.3.3 Three components model: Sines + Transients + Noise (STN)

The extraction of the components could be performed sequentially by extracting the sines (tonal) then extracting the transients from the residual. Alternatively it could be done simultaneously which is more efficient but this is more computationally expensive.

I. **Sequential extraction**

One approach is the transient modeling synthesis (TMS) proposed in [Verma et al., 1997]:

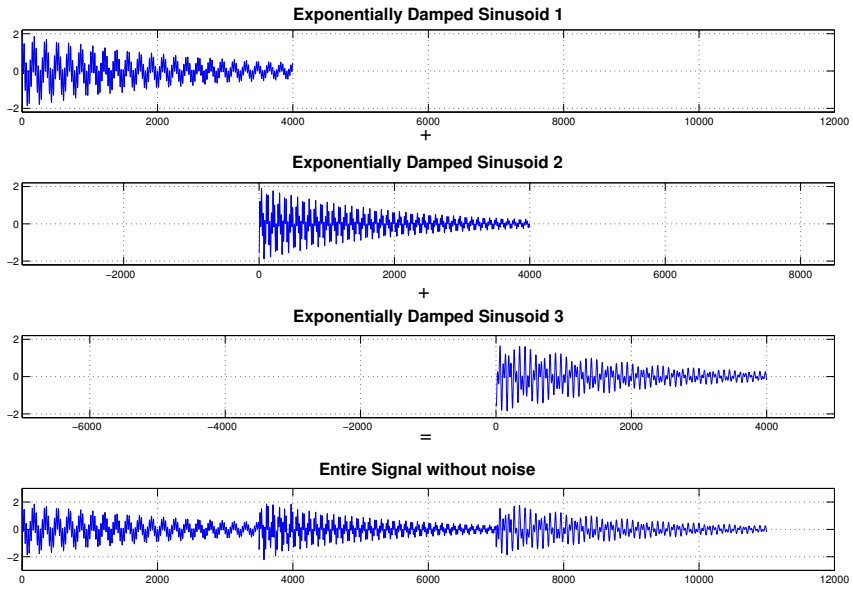


Figure 3.2. Subspace Projection: Signal is a series sum of damped sinusoids.

```
1 Apply modified discrete cosine transform (MDCT) to the input ...
      signal;
2 Extract the tonal portion of the signal by subtracting the ...
      components corresponding to large MDCT coefficients;
3 Apply Fourier Transform to the remainder (in the frequency ...
      domain) which transforms the signal to a pseudo-time domain;
4 The large pseudo-time domain coefficients will represent ...
      transients;The residual represents the noise.
```

A different approach implementing sequential extraction is starting with the first two steps of the previous approach then:

```
1 Apply discrete wavelet transform ( DWT ) to the time domain ...
      signal of the remainder after the tonal extraction;
2 The large DWT coefficients represent transients and the ...
      residual is the noise.
```

Even if those methods are simple they lack a proper definition of what is a large coefficient. They also have the problem that the estimation of one type of components (for example SS) is biasing the estimation of the other one (T).

II. **Simultaneous extraction: Adaptive Time-Frequency resolution**

Briefly the idea is to adapt at each step the transform resolution with respect to a certain sparsity criterion. It was applied to music for transient and steady-state separation by [JAILLET and TORRESANI, 2007] "Time-frequency jigsaw puzzle " using a smaller adaptation step called super-tile. As in the sequential extraction it suffers from the threshold estimation.

III. **Simultaneous extraction: Sparse Overcomplete Methods**

This is a new idea that is starting to attract researchers in many application domains as well as musical signal processing. Some papers have discussed the use of these methods to represent the musical signals and how this could be benificial[Plumbley et al., 2010]. In this research we will consider this concept of sparse overcomplete representation of music signals. We will try to benefit from this new upcoming idea to find a solution for the presented problem: "*note onset detection*". The next section is reserved to explain the idea in more details.

3.4 Approach: Sparse Overcomplete Representation of signals

In this research thesis we try to investigate the idea of modeling the input signal using overcomplete methods. This concept is discussed in many researches and papers but the application to note onset detection (or other music signal processing applications) is not well established and currently searching for a good implementation with well suited parameters to be comparable with the old methods.

The basic idea is to model the signal as the linear summation of a number of waves. They are the atoms that build up a dictionary that we use to represent the signal. Those waves could be of any type depending on the application area, the kind of the signal and the features to be detected. For example they could be the sinusoids forming the transform matrices for **discrete Fourier Transform** (DFT), **discrete Cosine Transform** (DCT), **discrete Wavelet Transform** (DWT) or a mixture of them.

Mathematically speaking, a signal $x \in \mathbb{C}^N$ is represented

$$x = \sum_k \alpha_k \varphi_k, \quad k = 1, 2, \ldots, M; \tag{3.8}$$

as the linear summation of the atoms φ_k where $\varphi_k \in \mathbb{C}^N$ are length-N complex valued vectors. Each atom's contribution in constructing the signal is different which is dictated in the equation by the coefficients α_k. As we are dealing with music signal we are considering the case of real-valued vector $x \in \mathbb{R}^N$.

When M=N, this representation could be seen as any normal *transform*. For example applying Formula 3.8 to the Discrete Cosine Transform, φ_k will be the cosine waves that form the DCT transform matrix. α_k in that case will be the DCT coefficients resulting from the transformation of the input signal to the DCT domain.

One could ask *where is the innovation here and why is it called over-complete methods?* The explanation comes from the fact that the used dictionaries are allowed to contain more atoms than the dimension N of the space $\mathbb{R}^N$ of the signal to be transformed. That is why those dictionaries are called over-complete dictionaries.

And here comes the real benefit from this method where we enforce sparsity to the coefficients. What we aim to is to get a solution that has only few non-zero coefficients α_k so that

$K << N$ to approximate the signal.

$$x \approx \sum_{j=1}^{K} \alpha_{k_j} \varphi_{k_j}, \tag{3.9}$$

For our specific case the note onset detection we could construct the dictionary in such a way to contain two kind of atoms. One kind of atoms to represent each of the components: the *steady-state*(SS) and the *transients*(T) components. It is easily deduced that the residual of the approximation will be considered as the third component which is the *noise*. For the **note onset detection** the onset vector could be constructed by using the transient atoms' coefficients only.

One of the main issues will be the approximation algorithm selection. We will call it **Projection Algorithm** as it will be projecting the input signal onto the dictionary atoms. The sparse projection problem is an NP-hard problem and therefore a number of approximate algorithms exist to solve it. Here we will briefly discuss the general idea of two of them including the one that we will use for the thesis implementation:

3.4.1 Matching pursuit

This algorithm is analogous to the sequential extraction in the sense that it searches and extracts in each iteration the best approximating atom. The next few lines shows the algorithm:

For each iteration

- select the atom that best approximates the input to this iteration, defined as the atom that has the highest correlation to the input.

$$k_{chosen} = \arg\max_{k} |\langle x, \varphi_k \rangle|$$

- subtract this atom multiplied by the appropriate coefficient from the input, and remove the atom from the dictionary.
- iterate until a stopping criterion depending on the residual is reached.

This method is simple but suboptimal because it does not guarantee that it will result in the sparsest solution.

When applied for note onset detection and to overcome the sub-optimality an extension of this algorithm is used. In the extension called "Molecular Matching Pursuit" in each iteration, instead of extracting a single atom, a group of neighboring atoms is selected. This group of atoms is called a molecule. A molecule contains only one kind of atoms i.e it cannot contain DCT and DWT atoms simultaneously. This enhancement prevents portions of the signal to be perceived as both SS and T if analyzed by single atoms. It forces them to be classified as only one type depending on the local context. It suffers a little more from the computational complexity.

3.4.2 Global Optimization

Alternatively, this technique is more analogous to the simultaneous extraction. Here the projection over all the atoms is done simultaneously and the search for the best approximation is done globally. The approximation problem is formulated mathematically containing a relaxed sparsity term.

$$u = \arg\min_{u} \|x - \Phi u\|_2^2 + \lambda \|u\|_p^p \tag{3.10}$$

One possible formulation as shown in equation 3.10 where x is the input signal, Φ is the matrix representing the overcomplete dictionary built out of the projection atoms and u is the solution - the sparse vector of coefficients. The first term **Approximation Term** is using L_2-*norm* to minimize the error between the input signal and the weighted atoms. The second term **Sparsity Term** which is responsible for the coefficients' sparsity. The parameter λ is the sparsity weight and is used to tune the ratio of sparsity to correctness of the solution.

This is actually will be the method that will be used throughout this research and will be discussed in section 4.3 while developing the reduction algorithm.

Chapter 4

Implementation

The aim of this research is to develop a testing platform for exploring solutions for the note onset detection problem; specifically through the projection of the musical signal onto an over-complete dictionary. This chapter shows the methodology we used and explains step by step how this testing platform was developed. Section 4.1 presents how the musical signal is modeled then section 4.2 shows how real music notes are mixed to generate melodies for testing. The novelty function implementation in section 4.3 is divided into three parts. First some basic concepts needed to build up the algorithm, then how it is constructed from the two main components: *Over-complete Dictionary* and *Optimization Equation* and finally a description for the problem parameters. The last section 4.4 presents the peak-picking phase of the solution.

4.1 Modeling Musical Signal Synthetically

The first step of this research was to understand the musical signal and try to model it. For the sake of the problem to be studied *Note Onset Detection* we will consider a note as the atom of any music signal. We first consider the generic changes that happen between successive notes then try to model notes with a more realistic model.

The output is a sequence of synthetic notes that will be used in the next step to get preliminary results of the reduction algorithm.

4.1.1 Modeling changes among successive notes

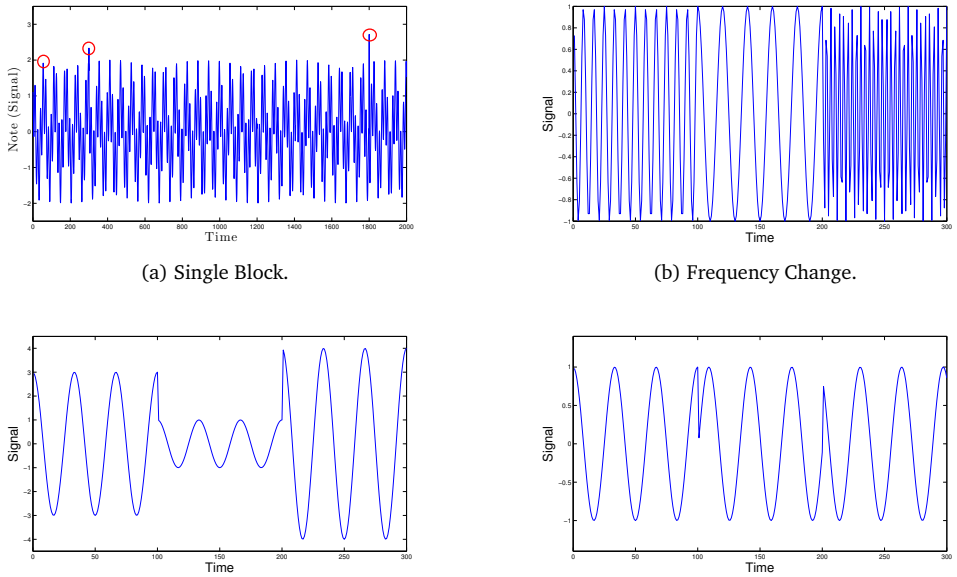
As discussed in section 3.3.2 a musical note is a mixture of many sinusoids. Therefore the changes that could happen from one note to the next one could be in frequency, amplitude or phase. A function ***ModelInputSignal*** is used to model those changes. It is a user interface that gets model parameters and generates an output vector that models the notes matching those parameters. Within a single block the user could mix two kinds of signals: sinusoids representing steady-state portion of a signal and impulses representing onsets. The parameters are:

- *nBlocks:* the number of successive changes (number of successive notes).

- *nOfCosines*: The number of cosines (Harmonics) to be mixed for this block. Those are used to model tonals.
- *nOfPulses*: The number of pulses (δ's) to be mixed for this block. Those were used for modeling the onsets. Then we did not use it anymore as the onsets should be detected from changes in frequency, amplitude and phase not from explicit onsets.
- *nSamples*: The number of time samples to be generated in the current block.
- *frequency*, *A*, *phi*: Those are the parameters for each cosine in the block. They represent frequency, amplitude and phase respectively. We used only cosines and not sines to construct the model because changing *phi* could generate sines.
- *timeOfPulse*, *pulse*: Those are the parameters for each pulse in the block.

It is a very simple function that iterates for the number *nBlocks* and mixes the described tonal and pulses part for each. It generates a sequence of blocks modelling the possible mutations between successive notes.

Figure 4.1 shows some examples of modeled signals. The first one 4.1a shows a single block consisting of two sinusoids having different parameters: *amplitude*, *frequency* and *phase*. It also contains three pulses (*synthetic onsets*) that are marked with red circles. The rest of the figures 4.1b, 4.1c and 4.1d are showing how the change in the three parameters could be produced. For each parameter a figure is showing the change hapening between three successive notes or blocks. The first figure 4.1a shows a single block having mixture of sinusoids and impulses. The rest of the figures 4.1b, 4.1c and 4.1d shows the three successive blocks changing in frequency, amplitude and phase respectively.



(a) Single Block.

(b) Frequency Change.

(c) Amplitude Change.

(d) Phase Change.

Figure 4.1. Modeling the input signal possible properties.

4.1.2 A more realistic model: Exponentially Damped Sinusoids

One of the state-of-the-art reduction methods described in section 3.3.2 considered the music signal as the summation of exponentially damped sinusoids. We used this idea to make the synthetic model more realistic by slightly modifying the function ***ModelInputSignal***. The user is asked to enter a damping factor *decayingFactor* which is used to control the amplitude decrease of the block. And finally using the modeling function the blocks could interleave imitating the real notes. This is done using a variable which saves the number of overlapping samples *blockOffset*.

Here is a figure 4.2 showing the concatenation of three notes that overlap over few samples. The red circles indicate the places in the signal that should imitate note onsets. The green circle is showing an offset - where a note terminates.

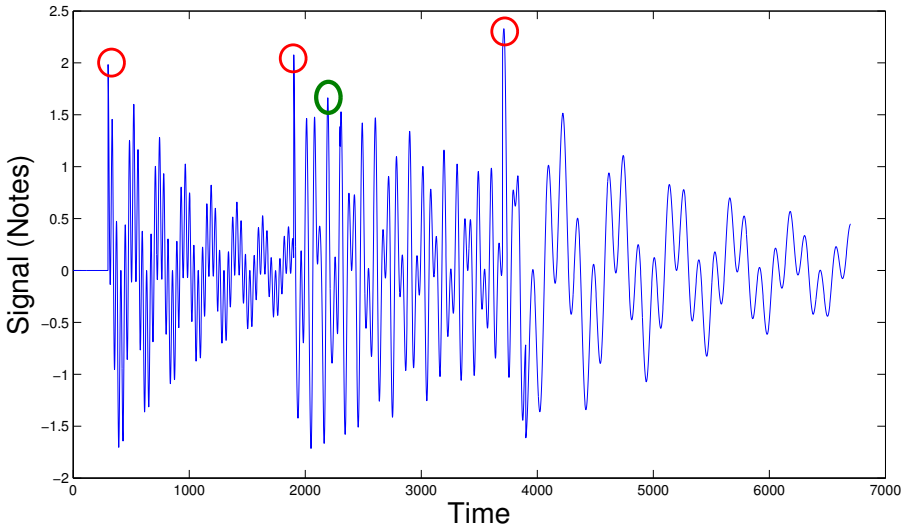


Figure 4.2. A more realistic model: Three successive exponentially damped sinusoids.

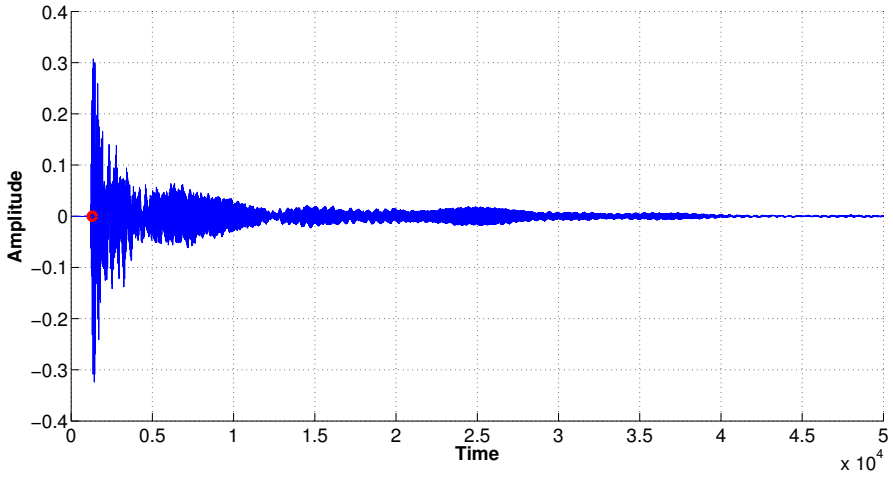
4.2 Loading And Mixing Notes

For testing purpose we used a library containing a huge amount of different musical single notes coming from different instruments. At least to cover two different types of music signal we used the *piano* notes for percussive pitched sounds and *accordeon* ones for the non-percussive pitched notes. For this reason we developed a function **mixTones** the role of which is to load and mix single notes from the different instruments in order to form some well-defined musical sequences.

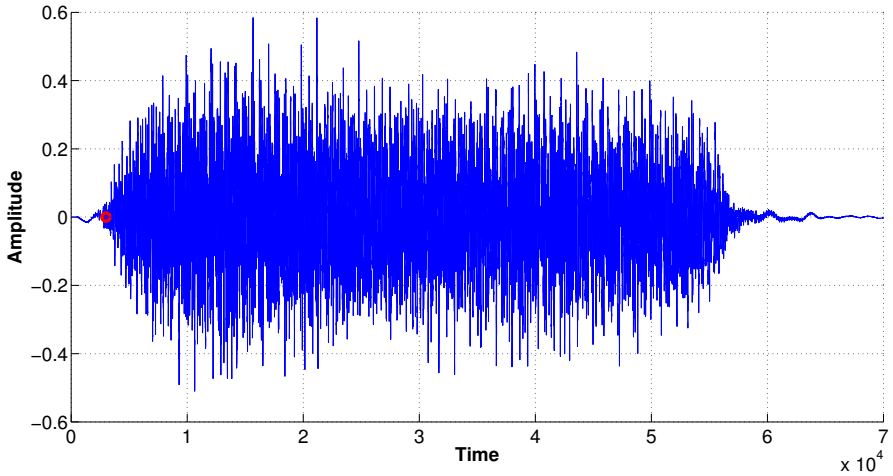
This function has a simple user interface that helps the user to characterise the needed signal:

- **Instrument**: to define the instrument from which the notes are to be chosen. In all our tests we used melodies constructed from single instruments.

- **Note Position**: This is used to define where to insert the current note in the musical piece.
- **Note Onset**: This is a parameter that is very useful for evaluation purposes, it is used to save the reader guess about where the onset should be. This info is used later by the detection evaluation algorithm to check the detection algorithm efficiency. In that case the detection evaluation depends on how accurate those onsets provided by the user are.



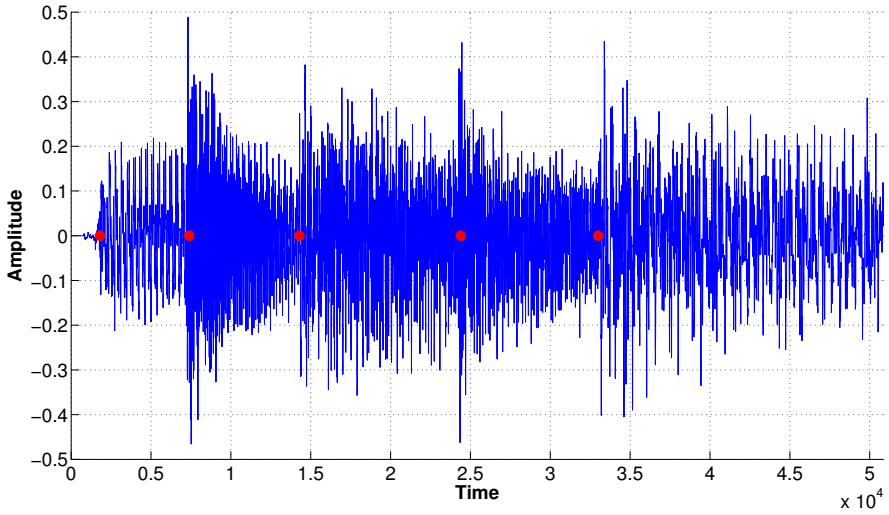
(a) Single Piano Note.



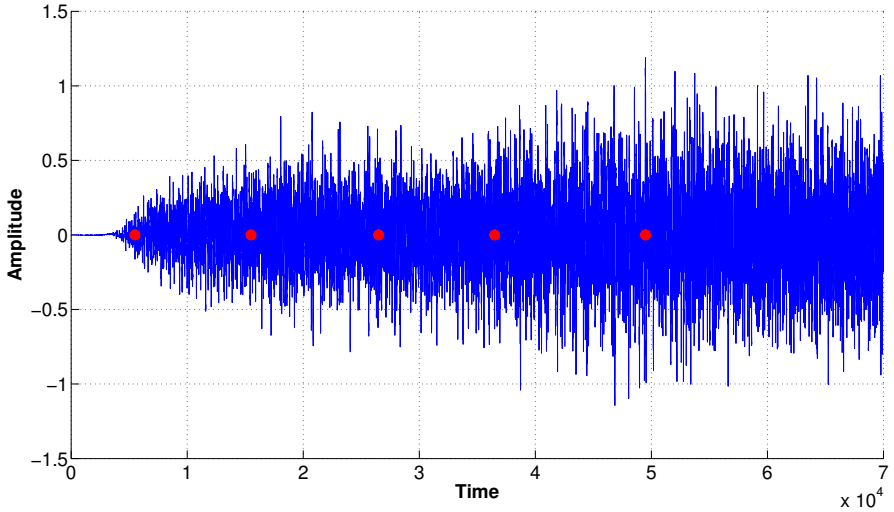
(b) Single Acordeon Note.

Figure 4.3. Different examples of musical notes loaded by the mixTones function.

Figures4.3 and 4.4 show different examples of the function output. Figure 4.3a shows a single piano note from which the reader can clearly notice it is higly percussive. Alternatively the accordeon note in figure 4.3b is not percussive at all: it has large attack and sustain times. Every note onset is marked with a red dot. Finally figures 4.4a and 4.4b show a sequence of



(a) Five consecutive Piano (percussive) Notes.



(b) Five consecutive Acordeon (non-percussive) Notes.

Figure 4.4. Different examples of musical pieces generated by the mixTones function.

five different notes for a piano and an accordeon respectively. Those generated examples will be the ones used for our testing and algorithm charachterization. Later the function will be used to generate longer sequences for the algorithms evaluation. Tables 4.1 and 4.2 state the specification we used to produce the note sequence examples in figure 4.4.

Table 4.1. Specifications of piano notes mix

Note #	**Note Start**	**Onset**	**Distance from Previous One**
1	500	1800	–
2	6500	7400	5300
3	13500	14200	6900
4	23500	24400	10100
5	31500	32900	8600

Table 4.2. Specifications of accordeon notes mix

Note #	**Note Start**	**Onset**	**Distance from Previous One**
1	500	5500	–
2	8500	15500	10000
3	22500	26500	11000
4	31500	36500	10000
5	44500	49500	13000

4.3 Reduction Function Implementation

4.3.1 Basic Concepts

I. **Projection**

> *I like the night. Without the dark, we'd never see the stars* - Stephenie Meyer, Twilight.

Sometimes we need to go to another place to see better. The same for signals, they sometimes need to be moved to other domains to emphasize their characteristics. Moving from one domain to another or in other words changing the basis of the representation and processing of signals is mathematically applied through projection.

Recalling how the projection is described mathematically, let's consider the simple case of projecting one vector x over another one y. Let $P_y(x)$ denote the projection of x over y. This is calculated through the equation [Smith, 2007][1]:

$$P_y(x) = \frac{\langle x, y \rangle}{\|x\|^2} x.$$

[1] http://ccrma.stanford.edu/~jos/mdft/Complex_Basics.html

Here $\langle x, y \rangle$ is the dot product between the two vectors x and y and $\|x\|^2$ is the normalization factor. It could also be rewritten as

$$P_y(x) = \frac{\langle x, y \rangle}{\|x\|} \frac{x}{\|x\|}.$$

which consists of two fractions. The first one $\frac{\langle x,y \rangle}{\|x\|}$ is the coefficient and the second $\frac{x}{\|x\|}$ is the unit vector in the direction of vector x.

Generally speaking a vector $x \in \mathbb{C}^N$ could be moved to another domain represented by a basis that constitutes of N different vectors. This will result in N different coefficients and that what we alternatively call base changing. To be able to reconstruct the original vector x from the coefficients and the N vectors those vectors must satisfy two conditions:

- The N vectors should be *linearly independent*. That means that one vector cannot be represented as a function of the others.
- They are *orthogonal* to each other. Their dot product is equal to zero.

II. **Tonal Atoms: DFT and DCT**

In this section we will introduce two different projections: the *Discrete Fourier Transform* and the *Discrete Cosine Transform*. They are the ones we chose to start with when investigating the note onset detection problem. We will briefly define them mathematically [Smith, 2007] but then we will define them with respect to our specific problem. They will be used to analyse a subset of musical notes so we can deduce some information about the frequency content of musical signals and their coefficient sparsity in both domains. These deductions will help us in the choice of some parameters like number of tonal atoms, frame size and overlap which will be explained in section 4.3.3.

Discrete Fourier Transform

The most famous transform is the *Fourier Transform* (continuous),

$$X(w) = \int_{-\infty}^{\infty} x(t) \exp^{-jwt} \, dt \, , \; w \in (-\infty, \infty); \tag{4.1}$$

which projects a signal $x(t)$ from time domain into the continuous frequency domain. So we get the frequency content $X(w)$ of this signal at all the possible frequencies.

When moving to the digital signal processing, there exists a version of this transform called *discrete fourier transform* where both the input signal $x(t_n)$ and the output frequencies w_k are sampled therefore digital. This is stated mathematically:

$$X(w_k) = \sum_{n=0}^{N-1} x(t_n) \; \exp^{-jw_k t_n} \, , \; k = 0, 1, \ldots, N-1; t_n = nT; w_k = \frac{2\pi f_s k}{N}; \tag{4.2}$$

And the inverse of this transform would be:

$$x(t_n) = \frac{1}{N} \sum_{k=0}^{N-1} X(w_k) \; \exp^{jw_k t_n} \, , \; n = 0, 1, \ldots, N-1 t_n = nT; w_k = \frac{2\pi f_s k}{N}; . \tag{4.3}$$

Where T is the period between two samples, f_s is the sampling frequency ($f_s = 1/T$) and N is the samples count.

Let us introduce another way to explain this transform which relates more to the projection scheme and helps us to develop a DFT projection matrix. If we define a projection atom s_k as

$$s_k = \exp^{jw_k t_n} = \exp^{j\frac{2\pi f_s k}{N} nT} = \exp^{\frac{2\pi nk}{N}}$$

then if we want to calculate the frequency content of the input $x(nT)$ at frequency w_k which we could simply call $X(k)$:

$$X(k) \;=\; \langle x, s_k \rangle \;=\; \sum_{n=0}^{N-1} x(n)\overline{s_k(n)}; \tag{4.4}$$

which is a projection of vector $x(n)$ over the conjugate of vector $s_k(n)$. Now it is easy to form the projection matrix out of the different s_k atoms. To fully describe a signal $x(n) \in \mathbb{C}^N$ we need N atoms with N samples each. Those atoms are linearly independent and orthonormal when f_k of each is equal to $\frac{kfs}{N}$ which is satisfied in the s_k's. In a matrix form the equation would be:

$$\begin{pmatrix} X(0) \\ X(1) \\ \vdots \\ X(N-1) \end{pmatrix} = \begin{pmatrix} \overline{s_0(0)} & \overline{s_1(0)} & \cdots & \overline{s_{N-1}(0)} \\ \overline{s_0(1)} & \overline{s_1(1)} & \cdots & \overline{s_{N-1}(1)} \\ \vdots & \vdots & \ddots & \vdots \\ \overline{s_0(N-1)} & \overline{s_1(N-1)} & \cdots & \overline{s_{N-1}(N-1)} \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \\ \vdots \\ x(N-1) \end{pmatrix}; \tag{4.5}$$

where the $X(k)$ are the coefficients of the atoms s_k when projecting $x(n)$ onto them. This is the most relevant representation to this thesis:

$$x(n) \;=\; \sum_{k=0}^{N-1} \widetilde{X}_k s_k(n),\; n = 0, 1, \ldots, N-1 \;,\; \widetilde{X}_k = \frac{X(k)}{N}; \tag{4.6}$$

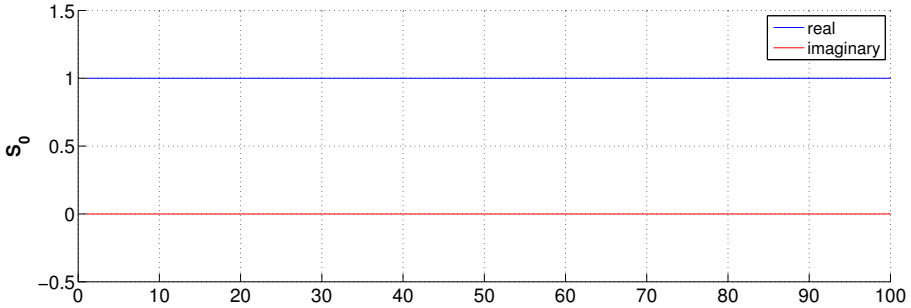
OR

$$\begin{pmatrix} x(0) \\ x(1) \\ \vdots \\ x(N-1) \end{pmatrix} = \begin{pmatrix} s_0(0) & s_1(0) & \cdots & s_{N-1}(0) \\ s_0(1) & s_1(1) & \cdots & s_{N-1}(1) \\ \vdots & \vdots & \ddots & \vdots \\ s_0(N-1) & s_1(N-1) & \cdots & s_{N-1}(N-1) \end{pmatrix} \begin{pmatrix} X(0) \\ X(1) \\ \vdots \\ X(N-1) \end{pmatrix}; \tag{4.7}$$

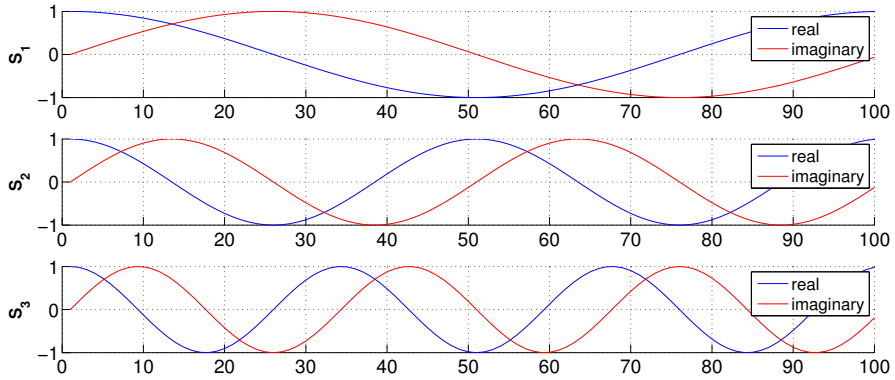
This is representing a signal as a summation of weighted atoms, atoms are the s_k's and weights are the $X(k)$'s which we call coefficients. Note that the s_k atoms here are the atoms of the inverse discrete fourier transform IDFT. This is the basis for the optimization equation developed in section 4.3.2.

A final and important notion before moving to the next transform is to look at the atoms themselves. Their samples are complex-valued and if we plot the $\mathbb{R}$eal and $\mathbb{I}$maginary parts of the subsequent atoms $s_0, s_1, s_2, \ldots, s_{N-3}, s_{N-2}, s_{N-1}$ (For example consider $N = 100$):

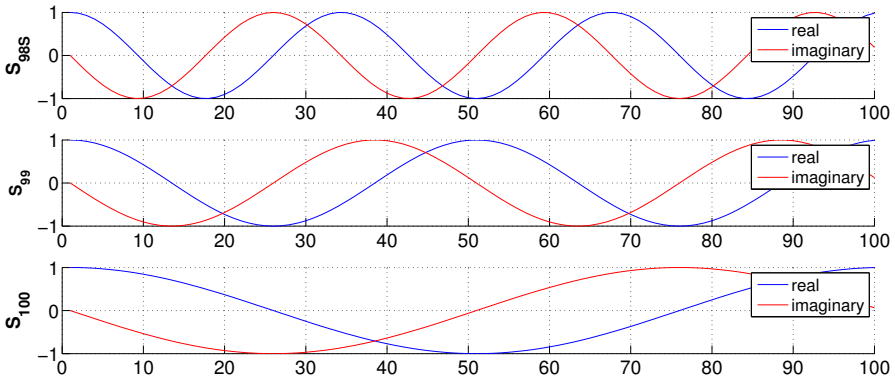
We can deduce the following:



(a) First DFT atom.



(b) First DFT atoms.



(c) Last DFT atoms.

Figure 4.5. Some of the DFT atoms for N=100.

- s_0 atom is the one that extracts the DC component from the signal.
- Each pair of atoms starting from the matrix border and moving toward the center are conjugate. For example in figure 4.5 the pairs $(s_1, s_{100}), (s_2, s_{99}), (s_3, s_{98})$ are conjugate.
- Although when moving from the matrix border to the center, the frequency of the sine and cosine is increasing. This leads to an important notion that the low frequency content is shown at borders and higher frequency content is present around the center. This plays an important role in the dictionary atoms' choice.

Discrete Cosine Transform

From its name in this transform the atoms of the transform matrix are real cosine signals that are different in frequency and phase in such a way to fully represent a signal $x(t_n)$ in frequency domain $X(w_k)$. The transform equation has the form:

$$X(w_k) \;=\; \alpha(k) \sum_{n=1}^{N} x(t_n) \cos\left(\frac{\pi(2n-1)(k-1)}{2N}\right) \quad k = 1, 2, \ldots, N \tag{4.8}$$

Where

$$\alpha(k) = \begin{cases} \frac{1}{N}, & k = 1 \\ \sqrt{\frac{2}{N}}, & 2 \leq k \leq N \end{cases}$$

The same discussion about the projection and projection matrix apply here as we have done for the DFT. Again the input signal $x(t)$ could be represented as the summation of DCT atoms s_k multiplied by the weights which are the DCT coefficients $X(w_k)$. The first difference as we mentioned before is that the atoms are real. The second property is that the DCT concentrates the frequency content in fewer coefficients when input signals are real. Therefore allows for a sparser representation. Note that here the k iterator for the frequencies starts from 1 and ends in N (this is specific to this formulation). Figure 4.6 shows how the atoms progress along the inverse discrete cosine transform IDCT matrix with $N = 100$. It is obvious that the atoms' frequency increases as long as we move to higher k. Which means that the low frequency content will be concentrated at the first DCT coefficients and the higher ones will be near the end of the matrix.

III. **Norms And Approximations**

To make sure that the reader understands how the reduction algorithm is developed and specifically how its input optimization function was constructed, in this section we introduce the mathematical operation called **Norm**. We will although go in depth with some notions that are relevant to the reduction equation characterization. Let us introduce the basic definitions and concepts as described by [El Ghaoui, 2012].

It is easy to answer the question about size or length when speaking about scalar variables. It will be the absolute value (positive magnitude) of the variable which is defined mathematically $|x|$. When dealing with vectors or matrices we need another definition. Here comes the notion *norm* which is generally defining the size (length) of a vector or a matrix. Formally described as a function $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ that assigns a positive value representing the length for each vector v.

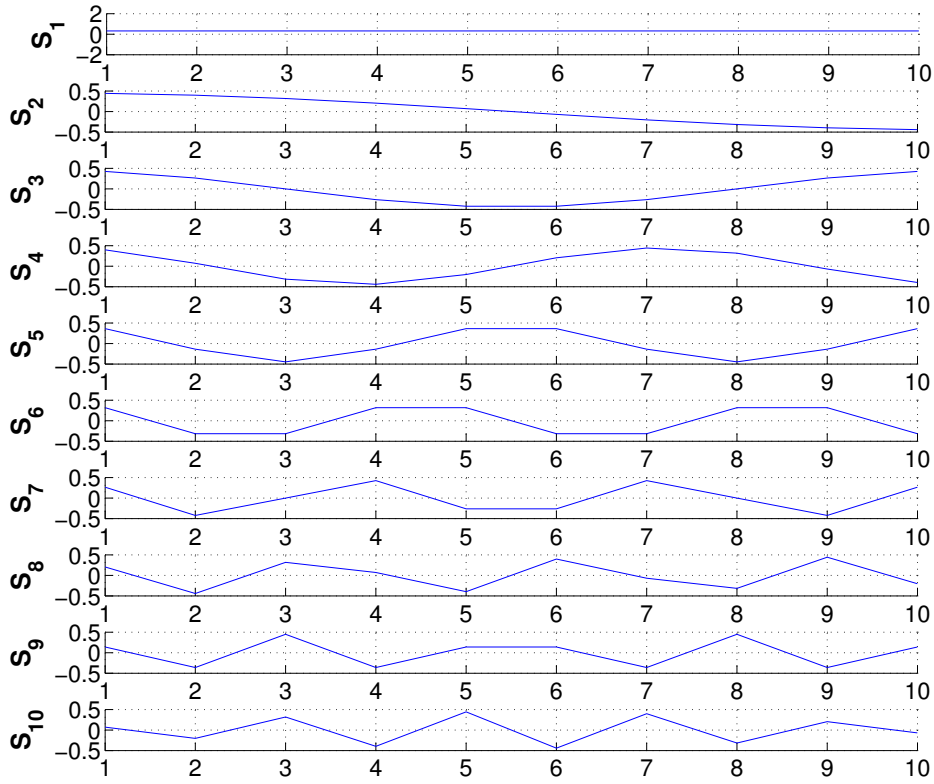


Figure 4.6. The DCT atoms for N=10.

To be a norm this function has to respect three properties (let us assume a vector v):

- For $v \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}^+ \longrightarrow f(\alpha.x) = \alpha.f(x)$ **Positive Homogeneity**.
- $f(v_1 + v_2) \leq f\ f(v_1) + f(v_2)$ **Triangle Inequality**.
- $f(v) = 0 \iff v = 0$ **Definiteness**.

It is a convex function resulting in non-negative values. One popular norm that will be used in the reduction algorithm is the L_p-*norm*.

$$\|v\|_p := \left(\sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}}, \tag{4.9}$$

where n is the vector v dimension. This *norm* function is convex only for $1 \leq p < \infty$.

Now we will focus on three possible values for p which are the mostly used. They although will help us to develop a better understanding of the meaning and usefulness of the norms. First let us introduce the three values together with their formal definition and meaning. Assuming a vector v defined in $\mathbb{R}^n$:

I. **Euclidean Norm l2-norm (p=2)**: This is the well known definition for the distance in two or three dimensions. It could be generalized for n dimensions.

$$\|v\|_2 := \sqrt{\sum_{i=1}^{n} v_i^2} = \sqrt{v^T v}, \tag{4.10}$$

II. **Manhattan Norm l1-norm (p=1)**: Is also representing the distance but travelled on a rectangular grid.

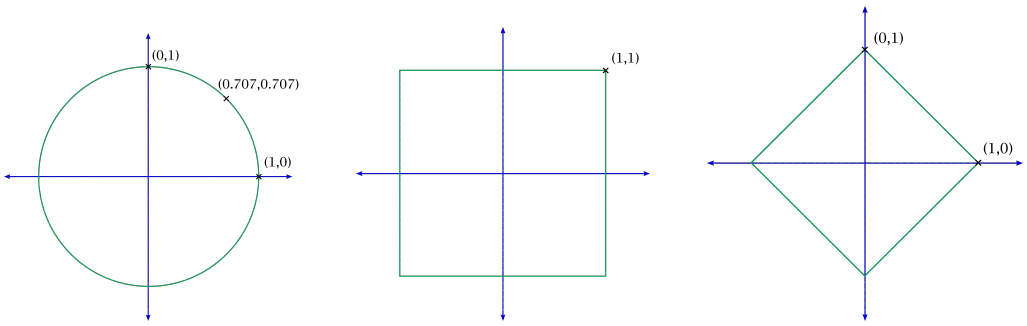
$$\|v\|_1 := \sum_{i=1}^{n} |v_i|, \tag{4.11}$$

III. **Maximum Norm linf-norm (p=inf)**: From its name it is used for peak picking. It gets the maximum value in a vector.

$$\|v\|_\infty := \max_{1 \leq i \leq n} |v_i|, \tag{4.12}$$

And now let us look how we can benefit from these norms while applying them for **Approximations** in the optimization functions as described in [Davenport]. Most commonly the euclidean distance is used to estimate the approximation error. This is the simplest and very intuitive way to interpret error. It is representing the energy in the error. In some applications we need a different definition for error which will be explained in the next few lines. First let us understand the notion of **unit ball**. It is the set of all vectors having a norm of one. It could be simply understood by thinking of the two dimensional space example.

As shown in the figure 4.7 above the *unit ball* is different for every L_p-*norm*. For the Euclidean ($p = 2$) all the points having $norm = 1$ (for example $(1,0); (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}); \ldots$) draw a circle. Instead, for the L_∞-*norm* they draw a square. This comes from the fact that this



(a) L_2-norm. (b) L_∞-norm. (c) L_1-norm.

Figure 4.7. The Unit Ball for each of the chosen L_p-norms.

norm calculates the maximum of the vector which is always equal to one in all the points of the square of figure 4.7b. Finally for the Manhattan norm as it is the sum of the absolute values of the vector elements, it is easy to deduce that the points will draw this diamond shape (rotated square) passing through the points $(0,1),(1,0),(0,-1)$ and $(-1,0)$.

Let us understand the meaning of the *approximation error* in different L_p*-norms* through an example. We want to approximate a point $x \in \mathbb{R}^2$ (a vector with two elements) in a one dimensional space. let $x = [1,2]$ and the one dimensional space represented by $A = span([2,-1])$ and we need to find the point $\hat{x} \in A$ that approximates x with a minimum error.

The error defined in lp-norm by

$$\varepsilon = \|x - \hat{x}\|_p \ ,$$

and $\hat{x} \in A$ could be written

$$\hat{x} = \alpha.A = [2.\alpha, -\alpha] \ ,$$

And the error will be

$$\varepsilon = [2-2\alpha, 1+\alpha].$$

And now to get $\hat{x}$ we use the **unit ball** concept. $\hat{x}$ is the intersection of the vector representing A with the growing ball centered at x. For understanding the approximation meaning for each of the L_p*-norms* we will calculate the error.

- In figure 4.8 showing L_2*-norm* the error is orthogonal to the vector defining A. Therefore $\langle \varepsilon, [2,-1]^T \rangle$ which leads to $2(2-2\alpha)-1(1+\alpha)=0$. Which simplifies to $\alpha = \frac{3}{5}$ and an error vector $\varepsilon = [\frac{4}{5}, \frac{8}{5}]$.
- In figure 4.9 showing L_∞*-norm*, it is clear that the error is equal in both direction. Therefore $2-2\alpha = 1+\alpha$. Which simplifies to $\alpha = \frac{1}{3}$ and an error vector $\varepsilon = [\frac{4}{3}, \frac{4}{3}]$.
- And in figure 4.10 showing L_1*-norm*, here the error equals to zero in the horizontal dimension. Therefore $2-2\alpha = 0$. Which simplifies to $\alpha = 1$ and an error vector $\varepsilon = [0,2]$.

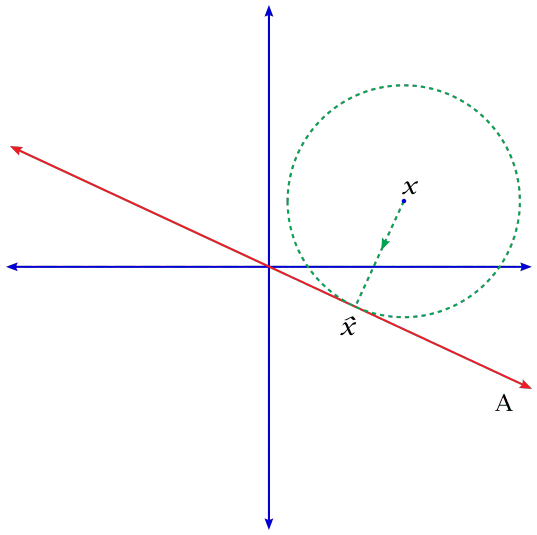


Figure 4.8. Intersection between L_2-norm unit ball and the projection space vector A.

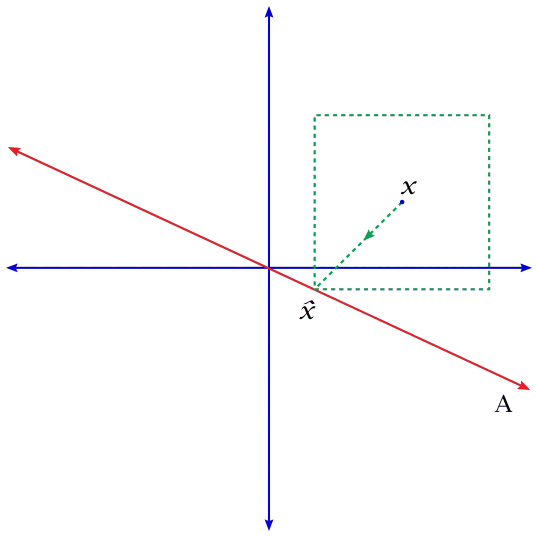


Figure 4.9. Intersection between L_∞-norm unit ball and the projection space vector A.

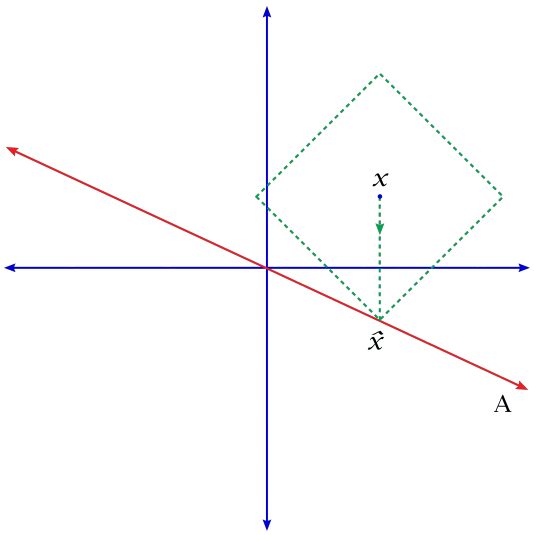


Figure 4.10. Intersection between L_1-norm unit ball and the projection space vector A.

This example makes it clear that for large values of p, the L_p-*norms* distributes the error evenly among all dimensions trying to minimize the maximum error. When p is small the error is concentrated in few dimensions. In this thesis to build up the reduction equation we will be using two different L_p-*norms*. The L_2-*norm* will be used to decrease the approximation error (distance between the original signal and the projection). The L_1-*norm* will be used for the sparsity part of the equation as it is concentrating the error. This will be discussed in more details in the next section while explaining the reduction algorithm.

4.3.2 Components and Formulation

We suppose the reader now has basic knowledge about the main building blocks we will use to build up our reduction algorithm. Let us now describe the steps together with the usage of each of these blocks. For the rest of this discussion we assume that the unit of processing is a frame $f \in \mathbb{R}^n$ - each frame has n samples. The reduction process is pre-processing then projecting frame by frame. Then the peak-picking is applied on the set of transient coefficients of the whole signal. Here we focus on the reduction phase and how we can formulate the problem mathematically.

I. **Build The Over-Complete Dictionary**

Let us name this over-complete dictionary A. It is a dictionary made up of two kinds of atoms: one representing tonals and the other one for transients. In our implementation we used pulses with amplitude equal to one to represent transients. This is a very basic assumption considering that the onsets - as described in 2.2.2 - repesent discontinuities in the input frame. We suppose a discontinuity will appear as a single pulse or a sequence of pulses that do not follow the steady-state signal evolution. So one transient atom is a vector with all elements equal to zero except one single element equal to one: $v = [0, 0, \ldots, 1, \ldots, 0, 0]$. An onset could happen at any point in a frame, so we need an atom representing each point (in a different implementation we could select only a subset). The

part of the dictionary that represents transients will have the form of an Identity matrix I:

$$A_{Transients} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

For the other type of atoms to represent tonals we used Discrete Cosine Transform (DCT) vectors for one implementation and Discrete Fourier Transform (DFT) vectors for another one. In case of using DCT atoms, we expected all the resulting coefficients - tonals and transients - will be real. That is due to the fact that DCT atoms are real. Alternatively in the DFT atoms case we will get real coefficients for the transients and complex coefficients for the tonals. In both cases we are only interested in the transients coefficients as they are the ones that should represent *onsets*. Both the DCT and the DFT atoms are sinusoids with different parameters, in general one different sinusoid occupying each of the dictionary columns. For the normal transforms those atoms were chosen to fully describe a frame $f \in \mathbb{R}^n$ with n orthogonal sinusoids, each sinusoid has n samples. In the case of over-complete dictionaries that we are using, there are different constraints:

- *Number of steady-state atoms*: It is not restricted to be n. The only restriction is on the full dictionary. To be over-complete, the sum of transient and steady-state atoms should be greater than n. So let us denote the number of steady-state atoms m.
- *Number of samples for each atom*: This should be equal to the frame f length n as before.

Suppose $a_{i,j}$ is sample i of sinusoid j, the matrix representing the steady-state part of the dictionary will have the form:

$$A_{SteadyState} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix}$$

Then we have the full dictionary as the concatenation of both parts:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} & 1 & 0 & \cdots & 0 \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} & 0 & 0 & \cdots & 1 \end{pmatrix}.$$

This is easily implemented in *MATLAB* by first constructing the $n \times n$ DCT or DFT matrix. Then only choosing the needed atoms from them - matrix columns - to represent the tonal components of the frame f. At this step we have the matrix $A_{SteadyState}$ which has m atoms n samples each. Finally concatenate the selected atoms to the *identity* matrix $A_{Transients}$.

II. **Equation Construction**

The basic idea is that we want to project the input frame into another space. Let us denote the frame with the vector $b \in \mathbb{R}^n$ as it is a portion of the input musical signal which is real in the time domain. The projection space is defined by the over-complete dictionary $A_{n\times(n+m)}$ we described before. When projected into this newly defined space having a number of elements equal to $m+n$ where m is the number of steady sate atoms and n is the transient atoms count, the frame b will be represented by another vector of coefficients $x \in \mathbb{C}^{m+n}$ with the following formula:

$$b = Ax. \tag{4.13}$$

As mentioned while introducing the research approach, the idea is in finding a sparse vector x that has few non-zero elements k such that $k << (n+m)$. The number of non-zero elements in a vector is represented by the L_0-*norm* $\|x\|_0$. And now we could describe our optimization problem including the sparsity condition through:

$$\min_{x\in\mathbb{C}^{n+m}} \{\|x\|_0 \ : \ Ax = b\}; \tag{4.14}$$

But the minimization of the L_0-*norm* is an NP-hard problem and not easy to represent mathematically. That's why it is approximated with the L_1-*norm* generating the so-called *Basis Pursuit* problem. We have seen before in section 4.3.1 how the L_1-*norm* is concentrating the approximation error therefore producing a sparse output. This *Basis Pursuit* problem is the same as before but changing the norm:

$$\min_{x\in\mathbb{C}^{n+m}} \{\|x\|_1 \ : \ Ax = b\}. \tag{4.15}$$

This approximation to the L_0-*norm* works well only if b is noisy or x' sparsity is not exact. Those conditions match with most of nowadays applications and of course with musical signals.

The previous formulation for the problem is using a hard constraint for the projection $Ax = b$. Relaxing this constraint would give more importance to the sparsity than exactness. In that case the *Constrained Basis Pursuit* is formulated:

$$\min_{x\in\mathbb{C}^{n+m}} \{\|x\|_1 \ : \ Ax - b \leq \delta\}. \tag{4.16}$$

Finally we reach to the form we are using to solve the *note onset detection* problem by transforming equation 4.16 from constrained to *Unconstrained Basis Pursuit* which is written as:

$$\min_{x\in\mathbb{C}^{n+m}} \|x\|_1 + \frac{1}{2\mu}\|Ax - b\|_2^2. \tag{4.17}$$

δ and μ are two real positive values. As they approach to zero, the sparse solutions for 4.16 and 4.17 approximate more and more the solution of equation 4.14. Equation 4.17 is the one describing the optimization equation formulation used in our implementation. It is called L_1/L_2 formulation as the sparsity is implemented by an L_1-*norm* and minimizing the projection error through an L_2-*norm*. For understanding why those specific *norms* were chosen refer to section 4.3.1 where norms and approximations are explained in details. To solve this optimization problem we used a package called YALL1 [Zhang et al., 2011] implementing the Alternating Direction Method (ADM).

III. **Convex Optimization Algorithm: Alternating Direction Method (ADM)** For solving the kind of problem we have, we need dedicated algorithms and not just normal convex optimization algorithms. Those algorithms are constructed to serve the aim of having a *sparse* solution. Several algorithms were proposed such as Gradient Method, Iterative Shrinkage/Thresholding, Fixed Point Continuation, SPaRSA and Coordinate Gradient descent. To apply the L_1/L_2 equation for optimization, we used a package *YALL1* developed by [Zhang et al., 2011] implementing a first-order primal-dual algorithm based on ADM. The primal and dual variables are updated each iteration. As in figure 4.11, the algorithm is alternating the search direction going toward the optimal solution which in our case is the sparsest.

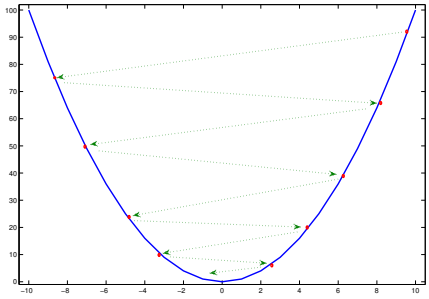


Figure 4.11. Alternating Direction Method.

The *YALL1* package offers different solutions for the L_1-norm problem that are covering the different formulations described before. The one we chose is the one implementing L_1/L_2 as mentioned before. Equation 4.18 shows how it is charachterized in the package. This will be the notation we will use throughout the rest of the research.

$$\min_{x\in\mathbb{C}^{n+m}} \|Wx\|_1 + \frac{1}{2\rho}\|Ax-b\|_2^2 \,; \tag{4.18}$$

Where W is the weights vector for the sparse vector x and ρ replaces μ of equation 4.17. They are used to tune the parameter λ that will be explained while defining the algorithm parameters in section 4.3.3.

An important issue for the reader to consider is the variety of packages we used to solve the convex optimization problem. Even if we show most of the results generated by the YALL1 package, in the begining of the research we used a matlab-based convex modeling system called **CVX** [Grant and Boyd, 2012] [Grant and Boyd, 2008]. It was used for the first trials but no results shown here are from CVX as we decided not to use it anymore because of its high computational comlexity. It gives good results but it is impossible to build a real-time system using this package. Another implementation we used for the optimization problem was the **Alternating Direction Method of Multipliers (ADMM) algorithm for solving l1-regularized linear regression (LASSO) problem** [van Waterschoot, 2013]. Throughout the thesis we continue using it and comparing its results with the YALL1 results. The important difference in dealing with both algorithms is in how they interpret the λ parameter. The one we use in the YALL1 implementation is the definition in the parameters

explanation, which is simply the measure of the exactness of the solution (higher λ means less approximation error). Instead when mentioning λ in the LASSO-ADMM implementation it refers to the sparsity measure. It replaces both λ_{ss} and λ_t with only one value for both. There is no λ for the exactness in this implementation.

4.3.3 Parameters

We started this research thinking about some parameters influencing the algorithm behavior, but during the research itself we came out with much more. So we chose some parameters to consider and the others are left for future work. In this section we describe only the parameters we considered in the implementation together with some value ranges for each. We got those ranges through a lot of *run and tune* loops. Most of the variables depend on each other, that's why even if each one has its own definition they have to be tuned together. We will divide the chosen parameters into two groups. The first type is the group of parameters that are signal-dependant which are *frame size*, *overlap*, *number of tonal atoms* and their *type* (DCT or DFT). The second group are parameters that are strictly related to the detection algorithm: *lambda*, *threshold*, *note spacing* and *optimization iterations*. Here we define each variable and try to present the heuristics we used to guess their best ranges.

I. **Frame Size and Overlap**

One of the most important parameters for the implementation is deciding on the processing step. This is implemented using two variables the frame size n and the overlap between frames. The choice of both should ensure that we don't miss an onset and the reader should note that an onset is not perceived in the same way by the algorithm if it happens at the beginning, the middle or at the end of the frame. As mentioned in [Moore, 2003] the human ear can distinguish two transients that are 10 ms apart. In a musical signal sampled at 44 100 Hz those 10 ms translate to $441 \approx 500$ samples apart. In that case we can theoretically use pairs of frame size and overlap like:

Table 4.3. Pairs of Frame Size and Overlap

Frame Size	**Overlap**
500	$\geq 0\%$
1000	$\geq 50\%$
2000	$\geq 75\%$

Note that the first choice when the overlap is zero % leads to a very bad perception when transients happens to be at the frame borders. We have to take care that this table is only a guess for not loosing onsets but the real values have to come from experiments.

Another constraint on the frame size is the computational complexity. It is better from the computational point of view not to have large frame sizes because this will require bigger dictionary atoms and larger projection evaluation times.

The last issue would be to chose the frame size large enough to differentiate between the normal frequency content that is found in a single note and the coefficients that represent transients.

II. **Number of tonal Atoms**

As previously defined while constructing the *projection matrix*, this is the number of dictionary atoms that are supposed to represent the tonal (steady-state) portion of the signal. They are a chosen subset from the DCT or DFT atoms. They are m atoms with n samples each where n is the frame size.

The motivation behind the choice of m depend on the maximum number of coefficients that are actually used to represent one frame in a musical note. As we were studying the piano and the accordeon signals, we ran some experiments on different single notes from each to select a reasonable amount of *SS* atoms m. This value depends on the frame size choice: the larger the frame, the signal will have more frequencies and therefore more tonal atoms to represent it. So using a smaller frame leads to a more concentrated frequencies which means smaller tonal atoms count.

We need also to look at the tonal atoms count from the computational point of view. It is very important to remove from the dictionary all irrelevant atoms - tonal atoms that will have a coefficient of almost zero value after the projection. Reducing this value m to only the useful atoms will decrease tremendously the computation time.

III. **Tonal Atoms Choice (DFT or DCT)**

For our implementation of the projection algorithm we had to choose which kind of atoms to represent the tonal portion of the signal either DFT or DCT. This choice depends on how well the signal is represented in both domains. For example how much sparse is the coefficient vector and how efficiently can we differentiate between transients and steady-state components. In a previous section 4.3.1 the two kind of atoms were described in details.

IV. **Lambda**

This is the parameter responsible for the trade-off between sparsity and exactness of the optimization equation solution. Remember when we constructed the projection equation 4.17, it contained two parts: the first part $\|x\|_1$ responsible for coefficients sparsity and the other part $\|Ax - b\|_2^2$ to enhance approximation (increase exactness). For that reason we need a parameter to tune the ratio of sparsity to exactness. We started with one parameter λ eventually we needed three different λ's. Here is the definition of each and how they are implemented.

- **Lambda approximation** "λ": this is the one responsible for the approximation weight. It is implemented with YALL1 package by making $\rho = \frac{1}{2\lambda}$ in equation 4.18. Increasing this lambda means tuning the optimization solution to be more exact.
- **Lambda steady-state** "λ_{ss}": this one is responsible for the tonal coefficients sparsity. Increasing its value generates less tonal coefficients and therefore perhaps more events coefficients. This is useful when detecting transients as they contain more tonal coefficients and when forced to be sparse they will move to the events side.
- **Lambda transient** "λ_t": finally this is exactly as λ_{ss} but for the transients coefficients. It is responsible for their sparsity. But here it is included for another reason, namely to retain only the coefficients that represent transients and not all the coefficients migrated from the tonal side.

λ_{ss} and λ_t are tuned together. They are also implemented together by setting the W variable in the YALL1 equation 4.18.

V. **Detection Threshold**

This parameter implements the thresholding algorithm in the peak-picking step of the detection process. In our implementation we applied the fixed thresholding approach as we are in the investigation phase of the research. We believe that continuing the algorithm development in future work will be much more efficient if dynamic thresholding is applied.

Thresholding is very dependent on the output of the reduction algorithm. It could be simply comparing coefficients with a fixed value. But a more complex thresholding procedure depending on a group of coefficients and not just one could also be employed.

VI. **Note Spacing**

Finally we introduce a parameter that is important when evaluating the algorithm. It is defined as the minimum distance between two consecutive notes that the algorithm could handle with a small detection error probability. The smaller this value is, the better the algorithm (both reduction function and peak-picking algorithms). This parameter is set when applying the peak-picking algorithm and in our implementation it is the expected number of *frames* between two onsets. To translate this value into seconds we use the following equation:

$$\textit{Time Between Notes} = \frac{\textit{Note Spacing} \times \textit{Frame Size}}{\textit{Sampling Frequency}}; \tag{4.19}$$

VII. **Optimization Iterations**

This represents the maximum number of iterations the algorithm will use in the optimization. As mentioned before while speaking about the ADM projection algorithm, it iterates until finding a good approximation with the maximum sparsity. Throughout the analysis the effect of tuning this parameter is checked making some trials. It is obvious that increasing this value is increasing the computation time.

4.4 Peak-Picking

Before introducing the basic ideas we used for the peak-picking phase of the detection process, we want to mention that every output from the projection is normalized. The benefit from this pre-processing step is to make the detection more generic as described in 2.3.3. When the projection is performed onto the over-complete dictionary, the output is a vector containing the transients coefficients concatenated to the steady-state ones. The normalization is done on this vector before dividing them into two groups - tonals and transients - for analysis. This is done to divide the frequency content among both types of atoms independently of the original frame energy.

We used two kinds of outputs to analyse the detection algorithm:

- **Events' Coefficients** (U)**:** Those are the portion of the output vector which represent the events coefficients. They are the weights related to the projection on the impulses atoms. They are always real and their count is equal to the number of transient atoms which is equal to the frame length.
- **Sum of Events' Coefficients** (Sum_U)**:** This is the sum of the events' coefficients for each processed frame.

The four thresholding and maxima picking algorithms we developed to be applied on the projection output are:

i. **Comparing Sum_U with a threshold:**

   Here we simply compare the Sum_U with a threshold and declaring what is above this value as onset.

ii. **Comparing Sum_U with a threshold using note spacing:**

   An enhancement to the previous method by respecting the *note spacing*. In this method after detecting a value above the threshold therefore an onset, we neglect the subsequent Sum_U for a *note spacing* distance. The inconvenience of this method is when detecting a false onset, the subsequent onsets are masked. It works well only with small *note spacing* values.

iii. **Comparing U with a threshold (First):**

   This is exactly the previous idea but applied directly to the coefficients U. Only the first value above the threshold is declared onset and the subsequent frames are discarded from analysis.

iv. **Comparing U with a threshold (Local Maxima):**

   A final enhancement to the previous algorithm is to search for the local maxima within a *note spacing* distance and choosing it as the onset. This method gives a better precision for onset detection.

Note that onsets are not well defined as mentioned before and that they are only an approximation for the transients position. That is why detection algorithms simplicity doesn't necessarily mean less detection efficiency. In the analysis section we show few results when applying those simple algorithms.

Chapter 5

Analysis, Results and Future Work

In this chapter we describe the different analysis steps and the methodology we used to check the efficiency of the proposed solution. We started by analysing the musical signal in section 5.1 then we analysed a synthetic model for music events in section 5.2 to obtain some preliminary results. Section 5.3 presents a case study for a piano melody. Finally the conclusions of this work are exposed in 5.4 and the future work is indicated in 5.5.

5.1 Musical Notes Analysis

Before starting with the detection algorithm, we needed to have a deeper look at the musical signal in order to give initial values or ranges for the parameters affecting the process. In order to do that we run some experiments on both types of signals: percussive (piano) and non-percussive (accordeon). Each of them was projected onto the two different kind of atoms DCT and DFT using different values for frame size and overlap. The tonal atoms count in this analysis step was always equal to the frame size. Here we show the results from these experiments in the form of figures all having the same format. Each has five different plots presenting (Top-down):

- The original musical signal (1-D vector) subject to the analysis.
- The sequence of frames to be processed taking into account the frame overlap. Prior to running the projection algorithm we construct a 1-D (vector) representation by concatenating successive signal frames to be prrcessed. The obtained vector is longer than the original signal vector, where the increase in length is proportional to the amount of frame overlap.
- Real DFT coefficients.
- Imaginary DFT coefficients.
- DCT coefficients.

Note that in the first two sub-plots the x-axis represent *time* and for the remaining sub-plots they represent *frequency*. DFT and DCT coefficients are the results of the projection of frames that are plotted in the second sub-plot (The frames to be processed). So to be able to analyse the

plots, the reader has to consider frame by frame in the second sub-plot and its corresponding transformations in sub-plots three, four and five.

And now we want to discuss the most important deductions from those experiments together with the chosen initial ranges for the parameters' charachterization. Note that for the experiments, the projection output was normalised for each frame separately. Here we neglect the effect of the signal amplitude envelope increase and decrease but we focus on the amount of frequency content and how it varies due to many factors. The output normalization is used to divide the frequency energy content among tonal and steady-state coefficients independent of frame position in the signal.

It is much easier to differentiate the amount of frequency content evolution from transient to steady-state in case of percussive instruments than in non-percussive ones. At this point it is good to notice that increasing the sparsity requirement on the tonal coefficients, would generate more events coefficients in case of transients - as described in the explanation of lambda parameter in section 4.3.3. That is shown in figure 5.1 where in the piano example 5.1a there are more frequency coefficients in frames where the transient occurs than in tonal frames. In the case of accordeon 5.1b we cannot differentiate. Another deduction from this experiment is that the DCT transform is a bit more clear than the DFT one, that comes from the fact that the coefficients are distibuted in two parts at the transform borders (as described before).

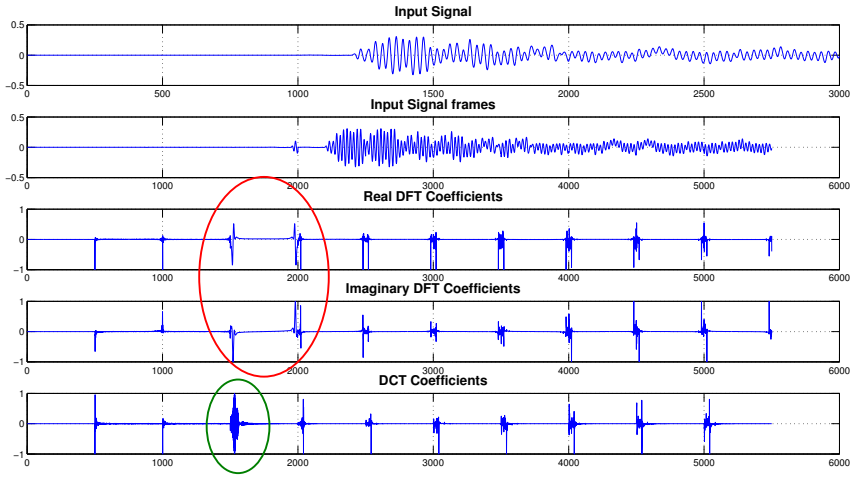
To settle on a range for the tonal atoms count, a closer view to the frequency content of each signal is required. We will look at the two different signals piano and accordeon, checking the amount of coefficients for transient and steady-state portions. By this experiment we need to get a range for the maximum amount of DCT and DFT coefficients needed to represent one processed frame. One experiment run for each of the signals - piano and accordeon - is shown in figures 5.2 and 5.3 respectively. This is only to show how the ranges are chosen, but actually the same experiment was run on different notes from both instruments. And here in the table 5.1 we state the approximate values that we got and that are also highlighted in figures 5.2a, 5.2b, 5.3a and 5.3b using the arrows. Those values are very useful when constructing the dictionary, they are used to select the amount of atoms to differentiate between transients and steady state portions.

Instrument	**Transform**	**SS**	**T**
Piano	DFT	2×80	2×120
	DCT	120	250
Accordeon	DFT	2×120	2×120
	DCT	240	260

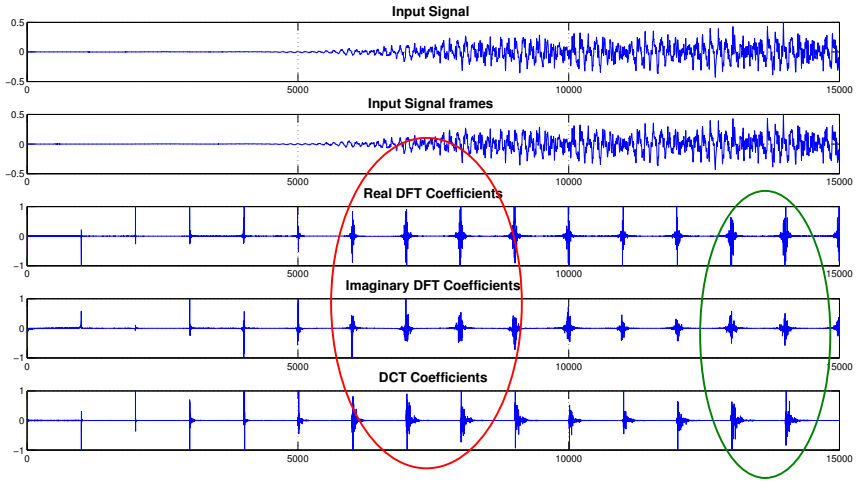
Table 5.1. Tonal Atoms count choice

Note that for DCT the coefficients are on one side of the transform, alternatively for the DFT coefficients exist on both sides. That's why in the implementation cutting the DCT and DFT transform matrices is done differently. For the DCT the first m matix columns are chosen as the tonal atoms. Alternatively for the DFT, the first and the last $m/2$ columns are chosen to be the atoms.

When first introducing transients, it was mentioned that it is very hard to position it and that is why we use onsets to mark the transients positions. Figure 5.4 shows that the transient is best detected when only a small part of the signal start is within the processing frame. That is the best place to mark an onset.

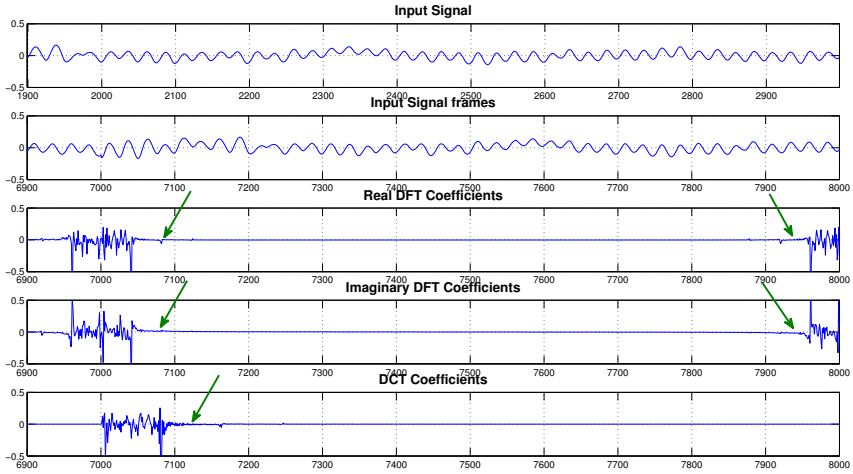


(a) Piano single note analysis (frame=500, overlap =50%).

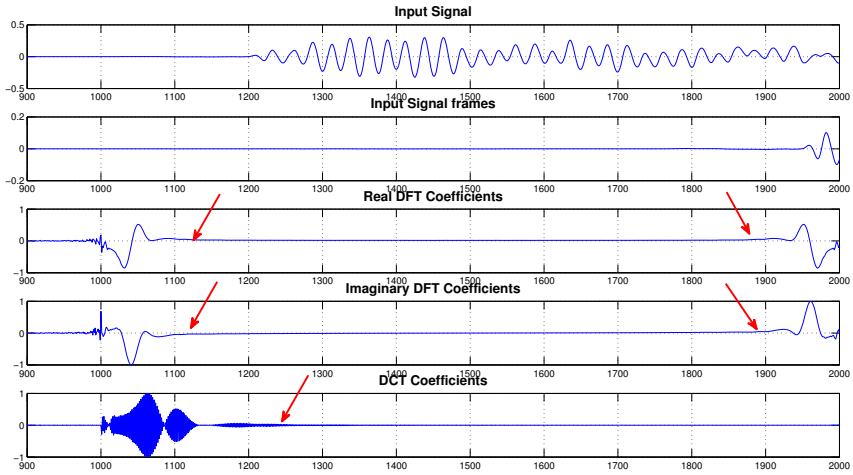


(b) Accordeon single note analysis (frame=1000, overlap =0%).

Figure 5.1. Single note analysis piano versus accordeon.

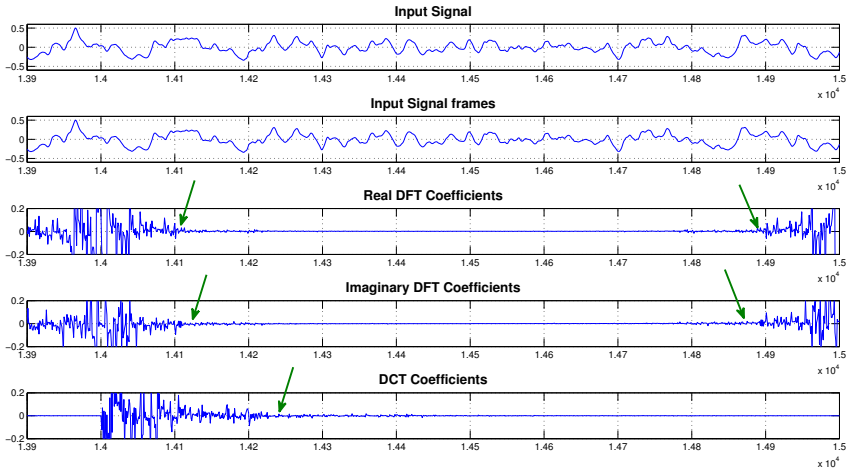


(a) Piano note frequency content (Steaty State).

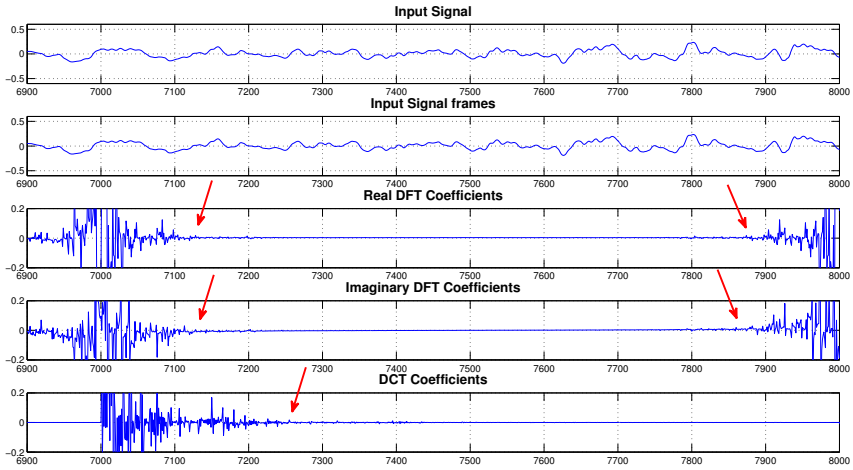


(b) Piano note frequency content (Transient).

Figure 5.2. Piano note frequency content (SS versus T).



(a) Accordeon note frequency content (Steaty State).



(b) Accordeon note frequency content (Transient).

Figure 5.3. Accordeon note frequency content (SS versus T).

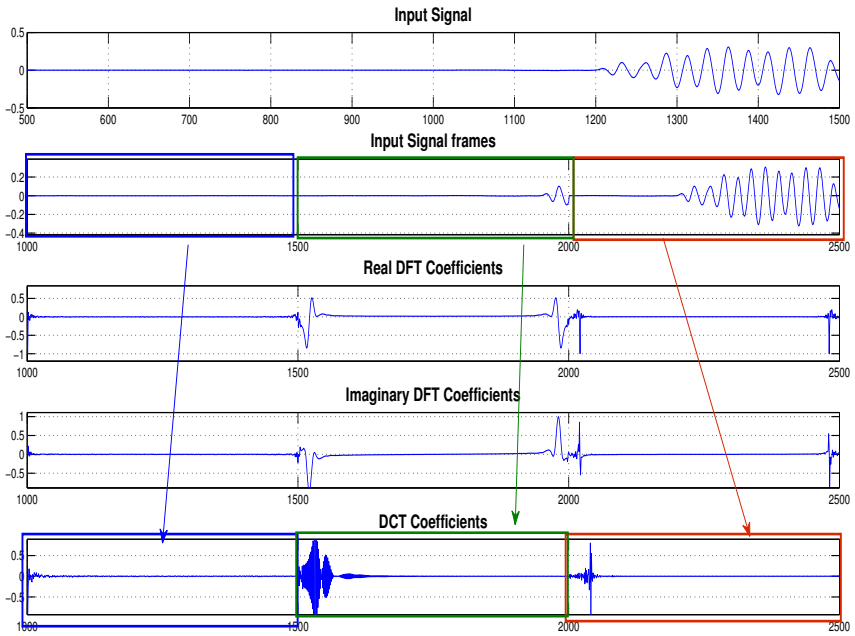
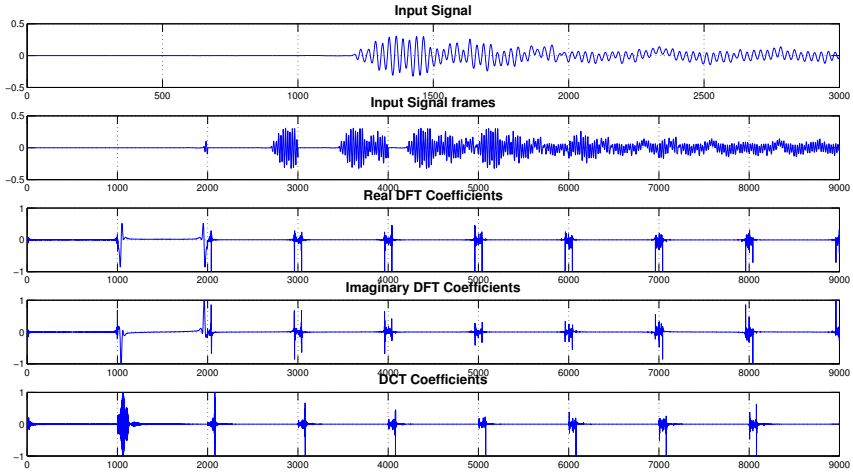


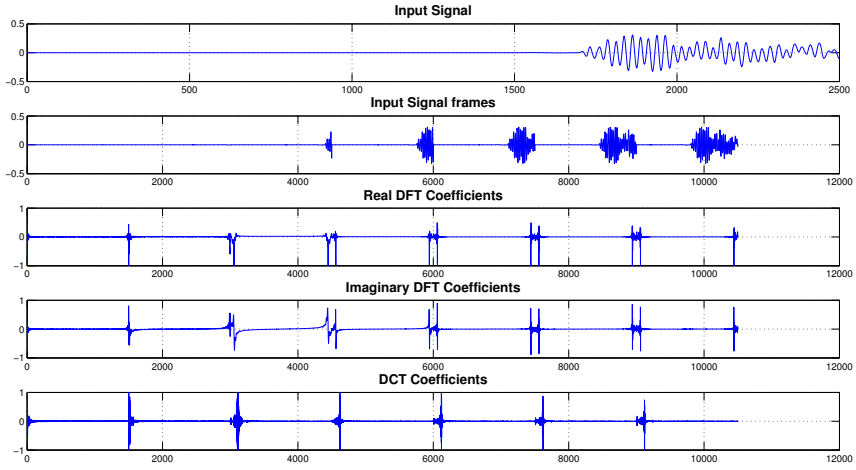
Figure 5.4. Best Onset detection time (Piano note - frame size =500, overlap = 50%).

The last point we want to discuss here is the frame size and overlap. We tried three different values for the frame size that are chosen to simplify the analysis. As we mentioned before it is a must to have higher overlap values together with larger frames in order not to miss transients that are close to each other.

Considering the plots in 5.5 and comparing them with the one in 5.1a we can deduce that increasing the frame size as in 5.5b moves the coefficients concentration to higher frequency bin indices. That is intuitive as the frame includes more samples but this is not favorable for our case: it makes it harder to differentiate between higher frequency content due to larger frames or due to signal discontinuity (Onsets). On the other hand no big difference between $n = 500$ or $n = 1000$ in the results, but with the latter together with a higher overlap ratio we have faster computation.



(a) Piano note (frame=1000, overlap =75%).



(b) Piano note (frame=1500, overlap =90%).

Figure 5.5. Frame Size and Overlap for piano signal processing.

5.2 Synthetic signal Analysis

Having developed the projection equation and built up the projection and detection process, let us analyse the results when applying this projection onto the synthetic signal model. We need to check the effect of signal discontinuity (amplitude, phase, frequency and their mixture) on coefficients' distribution among tonal and transients atoms. All the projection results are plotted in the same format which consists of five different plots representing (top - down):

- **Tonal Coefficients**: this is a vector formed by concatenating the tonal coefficients for succesisve frames, and then plotted along a single axis, which represents the steady-state coefficient index within each frame, and represents time along the frames .
- **Events Coefficients**: This is the other vector representing the events coefficients and it is constructed in the same way as tonal coefficients. Actually the projection output for each frame contains both steady-state and transient coefficients. They are then separated and fed to the two concatenation vectors: Tonal and Transient coefficients.
- **Input Signal**: this is the entire signal to be processed by the algorithm. In our case it is either the synthetic or the musical signal.
- **Input Frames to the algorithm**: the input signal is divided into overlapping frames which are then normalised and processed separately. This plot is used to understand exactly what was the input to the algorithm to generate the corresponding coefficients. It helps to know when the onsets are detected while moving with the frame window(i.e. if the onset is detected being at the beginning, middle or the end of the input frame).
- **Analysis Plot**: this is a plot to check possible characteristics of the events coefficients in order to choose a detection method. For the above examples we used the *sum of events coefficients per frame*.

To better understand the plot when analysing the events, all the five sub-plots were constructed to expand the whole figure width. In that way an event existing in a position i in time, would theoretically match a detected event around the same position in the other plots. Depending on the overlap value an input signal event could be perceived more than once at the output events coefficients.

First we will use an over-complete dictionary where the tonal atoms represent a full DCT (DFT) matrix $n \times n$ - the number of tonal atoms m is equal to the frame size n. Let us call this kind of dictionary *Full Over-Complete Dictionary*. Because the full DCT (DFT) transform matrix $n \times n$ is able to fully represent any frame of size n in the frequency domain, by setting the parameter λ to enforce exactness we are expecting a coefficient vector where no transients (events) coefficients exist. The only coefficients that will show up will be the ones chosen to model the synthetic signal - they could be verified as they are selected for each experiment.

5.2.1 Single sinusoid with no events

In the first set of figures 5.6 and 5.7 we are comparing the results from analysing a synthetic signal having these specifications:

- a single sinusoid (cosine) having 3000 samples.
- frequency= 41; amplitude= 1.
- divided into 1000 sample-frames with a processing overlap equal to 90%.

Theoretically no events should appear: first because it is a simple sinusoid having zero discontinuities and second as mentioned before it is a projection over a Full Over-Complete Dictionary. In practice the optimization algorithm produces events coefficients that are slightly different than zero. Figures 5.6a and 5.6b shows the extreme cases for the events coefficients after running the experiment varying λ from 0.1 to 1.5 with a step of 0.1. The events coefficients absolute value ranges between 0.02 and 2×10^{-5} depending on the sparsity value imposed by λ. We can see that increasing lambda to increase sparsity at some point - here at $\lambda \geq 0.8$ - is not giving a sparse solution anymore. It is instead decreasing the coefficients values with no sparsity at all.

In the case of YALL1, 5.7a and 5.7b, the range is between 0.2 and 2×10^{-18}. We can deduce that the YALL1 package gives much better results (being less far from the theoretical deduction) when a higher sparsity lambda is used. Events coefficients here are too small in the order of 10^{-18} which is negligible when it will be compared to coefficients generated due to real transients.

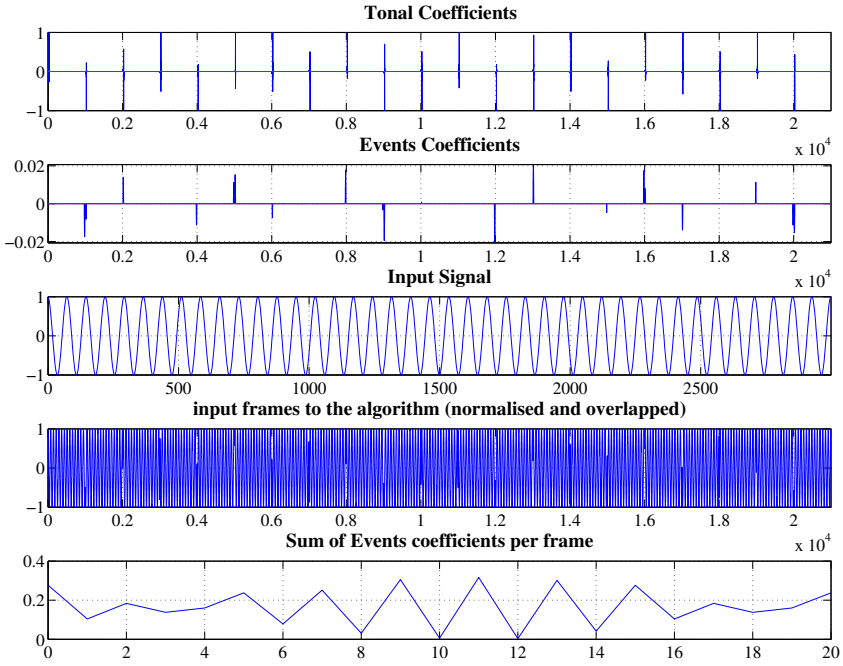
We tried the same experiment using the DFT atoms instead and the results are shown in figures 5.8 and 5.9. It is obvious that the events coefficients ranges are the same. The main disadvantage in using the DFT atoms is that they need more processing time specially in case of the LASSO-ADMM implementation. That is due to the fact that the algorithm needs to calculate complex coefficients for the tonal part of the dictionary. In the tonal coefficient plots the reader can see that the red points have values greater than zero representing the imaginary part of the coefficients. In the case of DCT atoms, the DCT coefficients had the imaginary component equal to zero.

Our testing platform is supporting the different implementations (CVX, LASSO-ADMM and YALL1) and the different kind of atoms (DCT and DFT). For simplicity in the major part of the analysis we prefered to use DCT atoms together with the LASSO-ADMM and YALL1 optimizations and comparing their results.

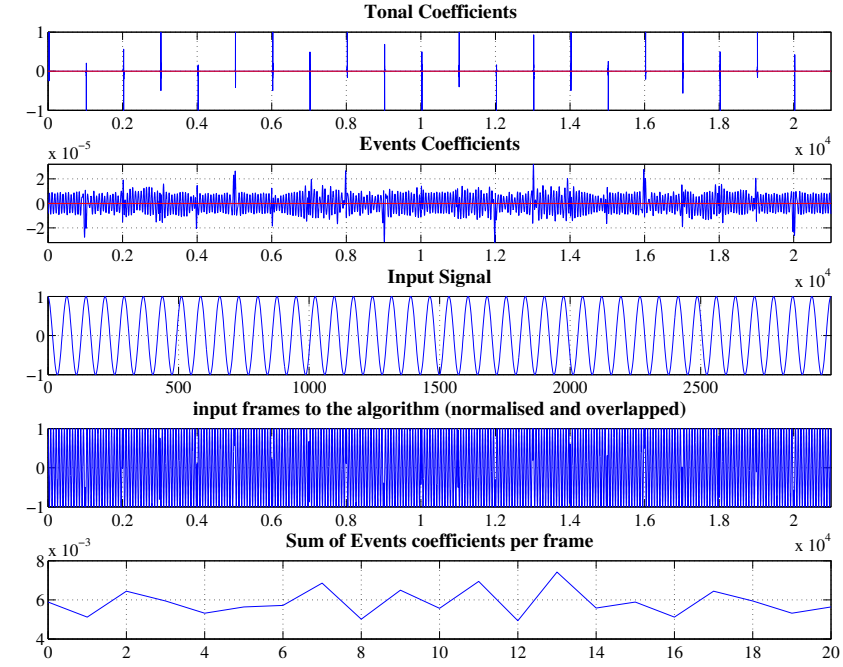
5.2.2 Simple sinusoid with one event

In a second step we want to analyse the coefficients in case of a synthetic signal similar to the previous one with an additional pulse. We want to see if the algorithm is able to identify this pulse as an event by first looking at the coefficients and analysis data changes that will happen due to it. Figure 5.10 shows the synthetic signal input to the algorithm highlighting the pulse with a green circle. The two sets of figures 5.10 and 5.10 show the projection results and analysis of this signal with LASSO-ADMM and YALL1 respectively. Again the experiment was run varying λ within the range from 0.1 to 1.5 and here we present only the consecutive plots where the output changes significantly.

We can see that for a $\lambda = 0.6$ - which in the case of LASSO-ADMM represents a less sparse solution - there is a rise in the events coefficients values around the time position 1200 in time. The two green squares in figure 5.11b point to this rise region in the projected frames (events coefficients) and the analysis parameter plots. Even it is not very clear but it is comparable to

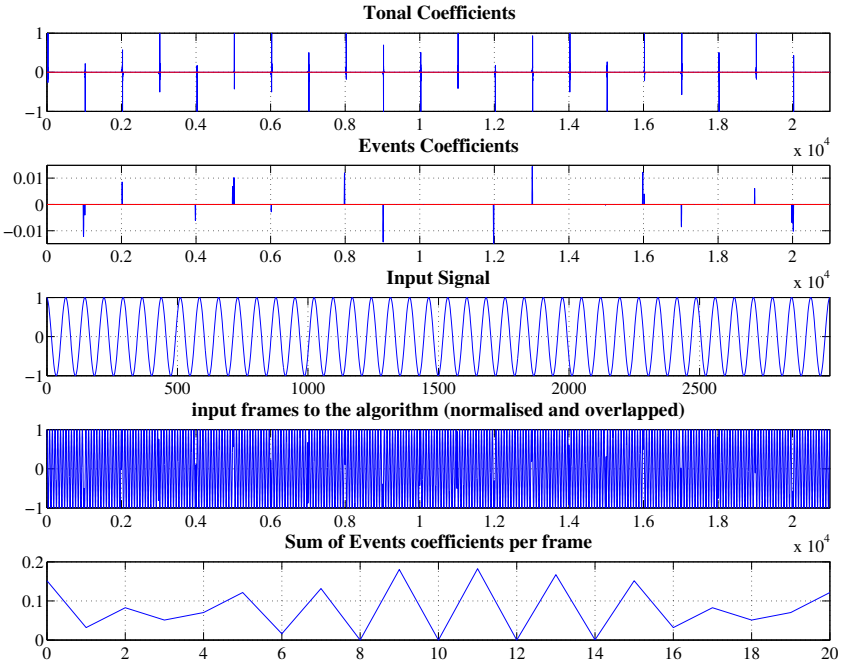


(a) Lower Sparsity $\lambda = 0.3$.

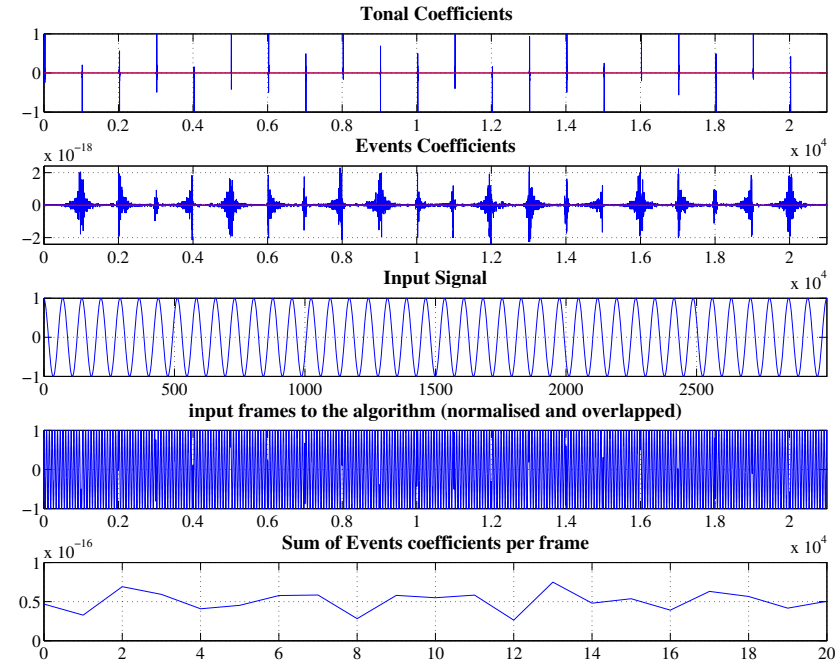


(b) Higher Sparsity $\lambda = 0.8$.

Figure 5.6. Projection of a single sinusoid with no events(LASSO-ADMM) using DCT atoms.

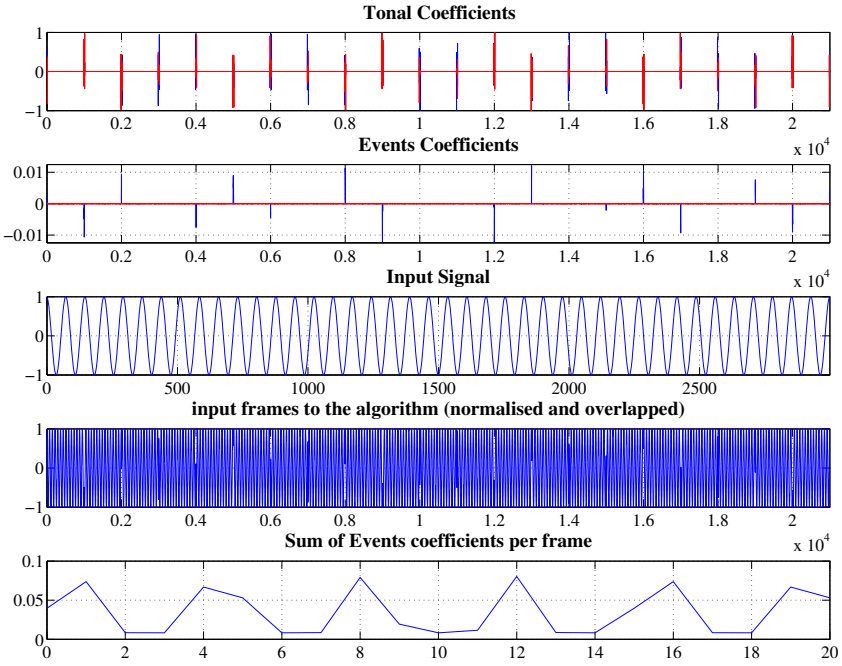


(a) Higher Exactness (Lower Sparsity) $\lambda = 1.1$.

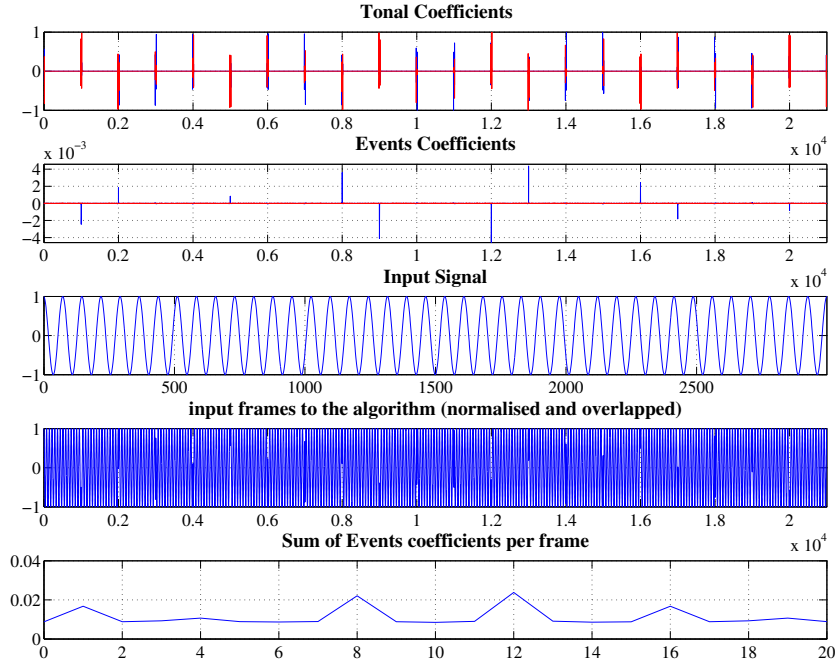


(b) Lower Exactness (higher Sparsity) $\lambda = 0.6$.

Figure 5.7. Projection of a single sinusoid with no events(YALL1) using DCT atoms.

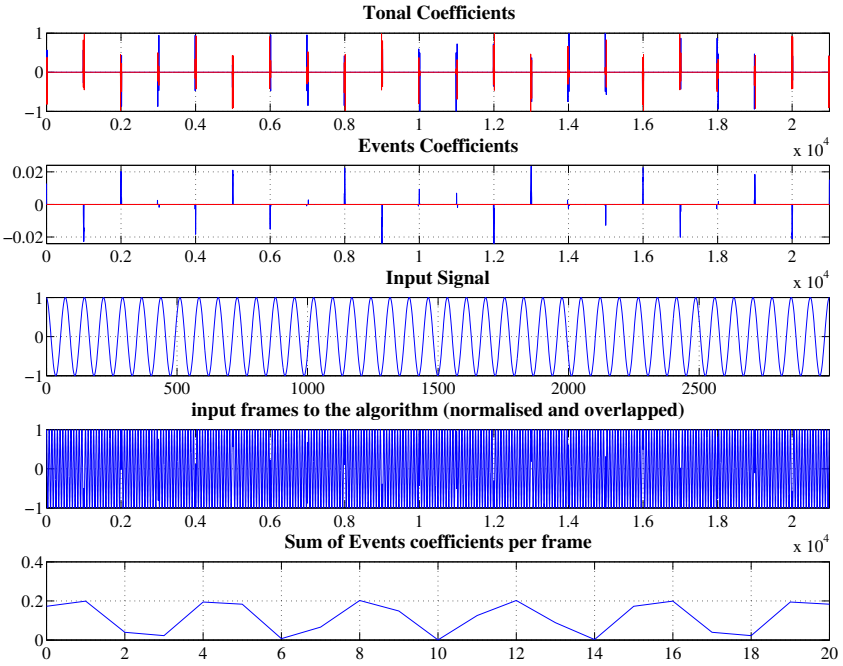


(a) Lower Sparsity $\lambda = 0.7$.

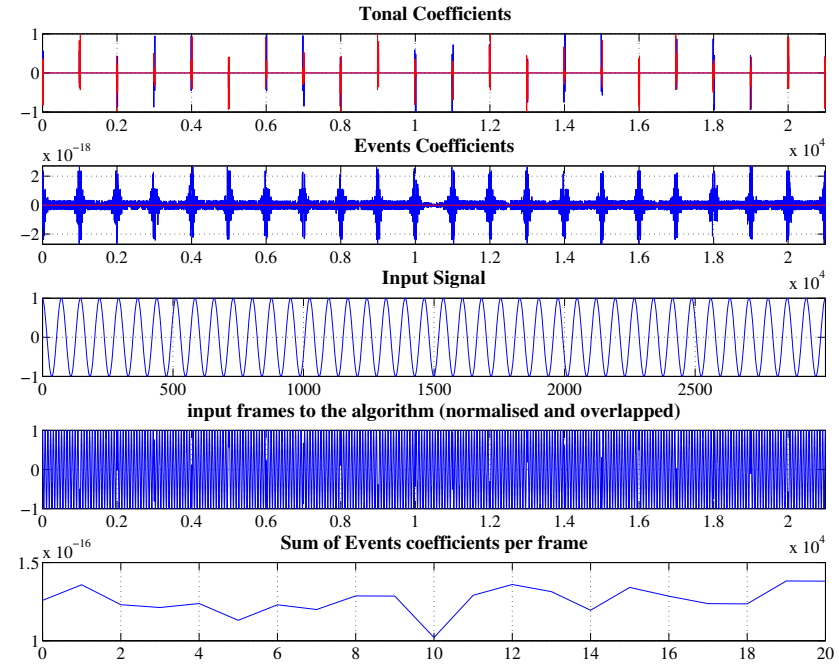


(b) Higher Sparsity $\lambda = 0.8$.

Figure 5.8. Projection of a single sinusoid with no events(LASSO-ADMM) using DFT atoms.



(a) Higher Exactness (Lower Sparsity) $\lambda = 0.9$.



(b) Lower Exactness (higher Sparsity) $\lambda = 0.5$.

Figure 5.9. Projection of a single sinusoid with no events(YALL1) using DFT atoms.

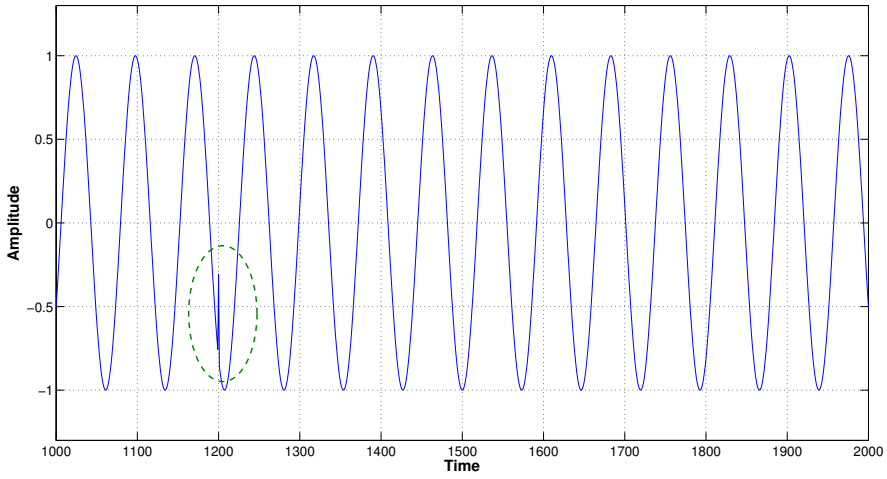


Figure 5.10. Synthetic input: sinusoid + one pulse (at t=1200).

the previous analysis example which was only a tonal without event. This result is the same for all sub-figures in 5.10 and 5.10.

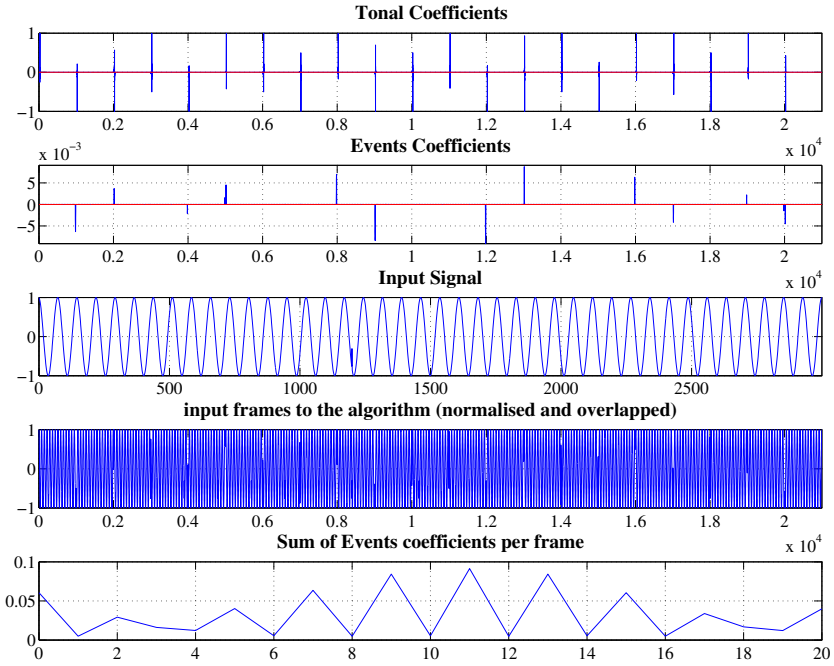
A second result is that moving to a medium sparsity $\lambda = 0.7$ in both implementations 5.11b and 5.11b leads to the most clear result. Finally when increasing sparsity in 5.11c we can realize that the information about the event can still be seen but difficult to detect mathematically as they are mixed with the events coefficients not representing events - we mean the ones that are generated by the algorithm without the existence of events. On the other hand the event info is totally lost in the YALL1 implementation shown in figure 5.11a.

An important deduction from the second step in the analysis is that the value of lambda is very crucial to the algorithm effectiveness. It is very dependent on the kind of events we want to detect. For fixing the ambiguity in the YALL1 results when a higher sparsity is enforced we will tune also λ_t and λ_{ss} in the following experiments.

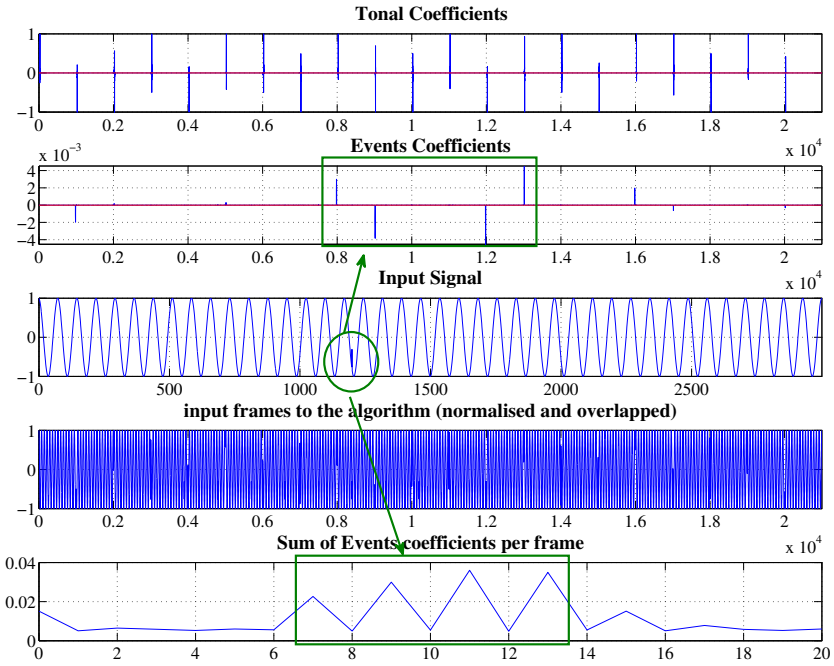
5.2.3 More realistic events

Moving towards a more realistic model for the musical signal we will analyse some of the signal features that are related to transients. Features will be analysed separately and then mixed together. For each feature the signal will be analysed by the two algorithms mentioned before LASSO-ADMM and YALL1 and then results will be compared and discussed. The analysis is done in two steps for the LASSO-ADMM and three for the YALL1:

- First the signal is analysed by a full over-complete dictionary to check the amount of tonal coefficients generated.
- In a second step we cut down the number of tonal atoms to a smaller one to increase the algorithm speed and effectiveness in differentiating between tonals and events. Then the

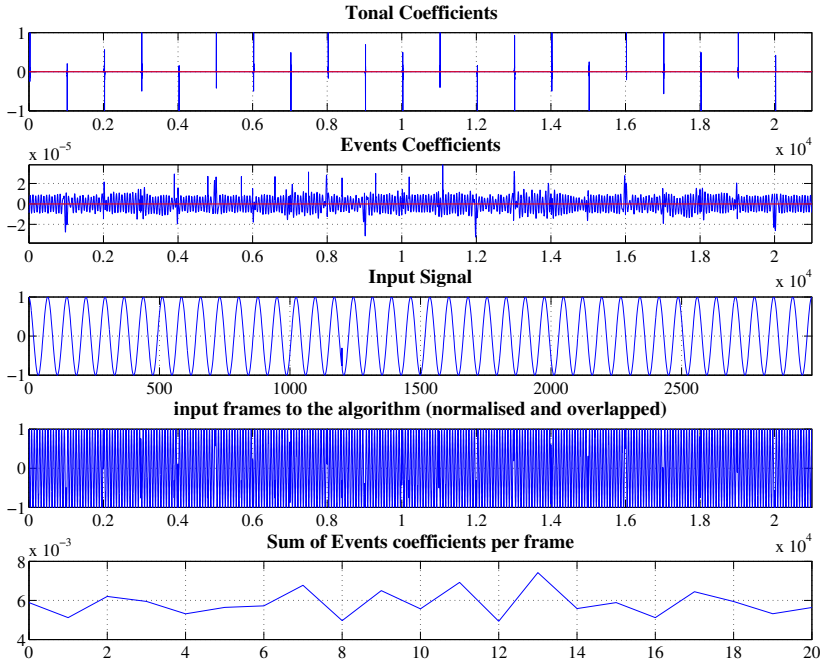


(a) Lower Sparsity $\lambda = 0.6$.



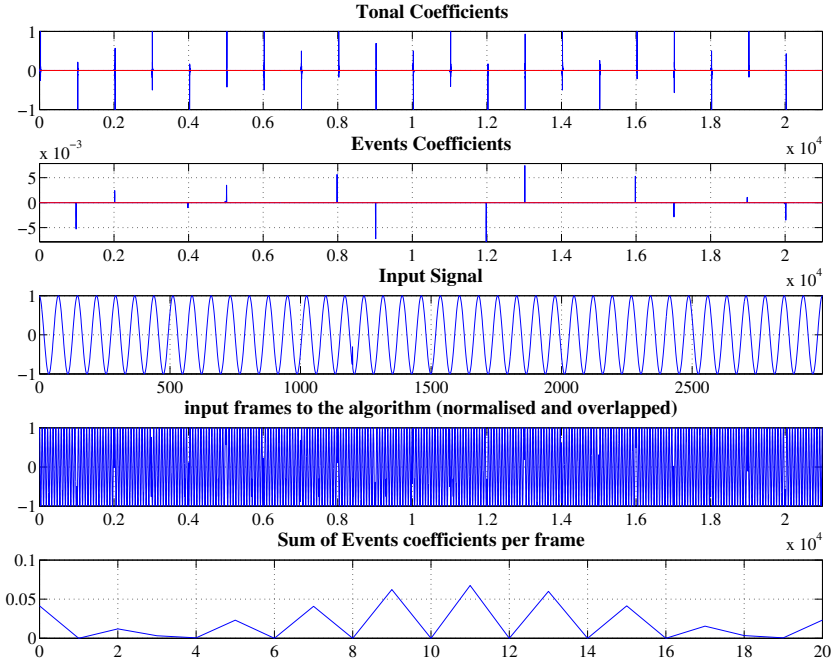
(b) Medium Sparsity $\lambda = 0.7$.

Figure 5.11. [a,b] Projection of a single sinusoid with one pulse events(LASSO-ADMM).



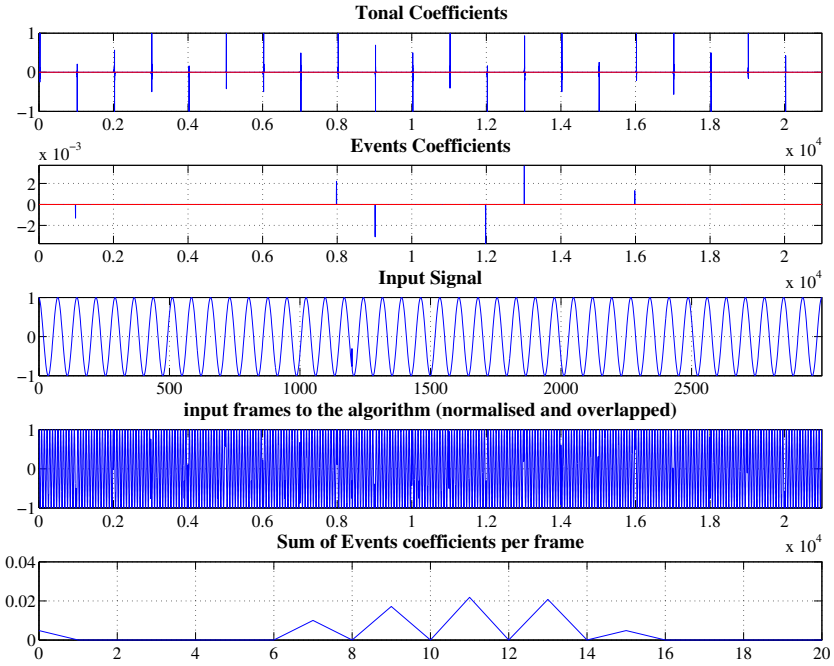
(c) Higher Sparsity $\lambda = 0.8$.

Figure 5.10. [c] Projection of a single sinusoid with one pulse events(LASSO-ADMM).

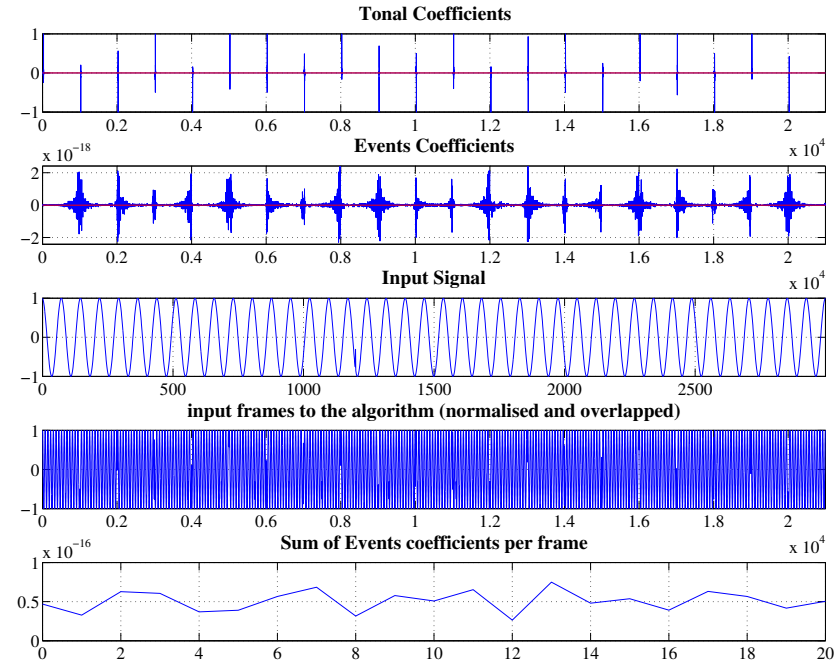


(a) Lower Sparsity $\lambda = 0.8$.

Figure 5.11. [a] Projection of a single sinusoid with one pulse events(YALL1).



(b) Medium Sparsity $\lambda = 0.7$.



(c) Higher Sparsity $\lambda = 0.6$.

Figure 5.10. [b,c] Projection of a single sinusoid with one pulse events(YALL1).

algorithms are applied again varying the parameter λ in a certain range. Moving forward this range is shrinked and a smaller step is used in order to obtain a good representation for onsets at the output coefficients.

- The third step applied only for the YALL1 implementation consists of tuning the λ_t and λ_{ss} to get a better events representation.

For simplicity every input signal - in the separate feature analysis - is constructed in such a way to have the feature discontinuities happening every 2000 samples. Instead when doing the experiment of mixed feature changing, the input signal is a sequence of four exponentially damped sinusoids starting at times (0,2000,5000 and 7000). We expect that the projections produce as much as possible higher events coefficients at times where those artificial events are created.

Here we show the best final results for those synthetic signals representing the various features. First the results for the LASSO-ADMM are presented in 5.10 and 5.11, then the results for YALL1 are shown in 5.11 and 5.12.

To understand the results of the separate features analysis, in each figure we will look at three particular sub-plots: the *events coefficients*, the *normalized and overlapped frames* and the *sum of events coefficients* respectively the second, fourth and fifth sub-plots. In each experiment we generated four discontinuities in the selected feature. Therefore the analysis consists of looking at the positions of those three changes (2000,4000,6000 and 8000) and check sub-plots 2 and 5 for the corresponding positions. Sub-plot 4 is showing more precisely when the feature discontinuity starts to be processed taking into consideration the moving overlap window. Sub-plots 2 and 5 present the possible information we can get from the projection algorithm in order to detect onsets.

LASSO-ADMM Results Analysis

We can see in figure 5.11a that the amplitude discontinuities are very well reflected in both the events coefficients U and the sum of events coefficients Sum_U. Instead for the frequency feature in 5.11b one change masked the others. Finally the phase change in 5.11c is non sparse but still relevant when looking at the Sum_U sub-plot.

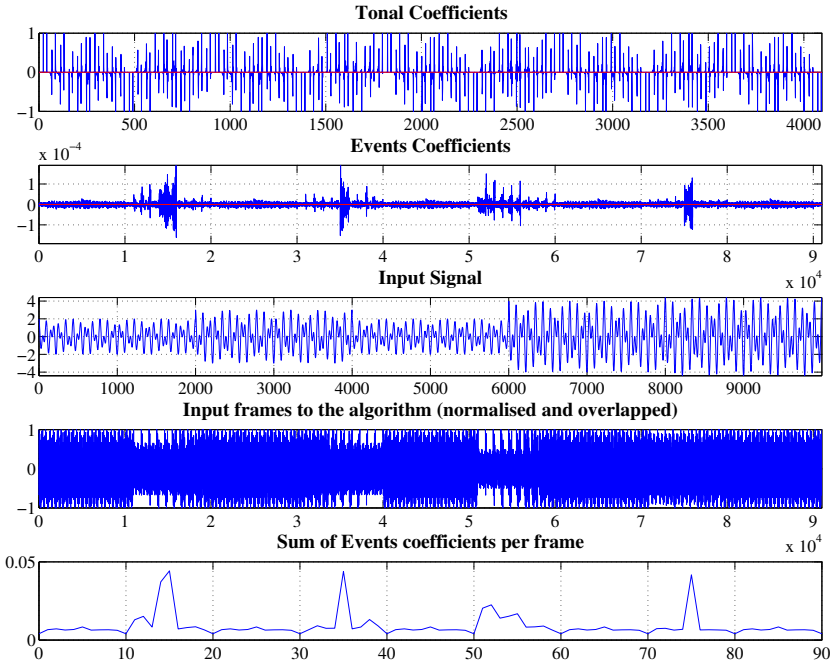
For the three features the peaks are more likely to be when the frame contains equal portions from both values of the feature. In other words, when the signal has the feature equal to a then its value became b the input frame that has the first half of its samples respecting value a and the remaining samples b, this frame generates the highest projection coefficients.

Analysing the exponentially damped sinusoids in 5.11 we see that the coefficients are following the synthetic notes shape while not giving a very sparse solution.

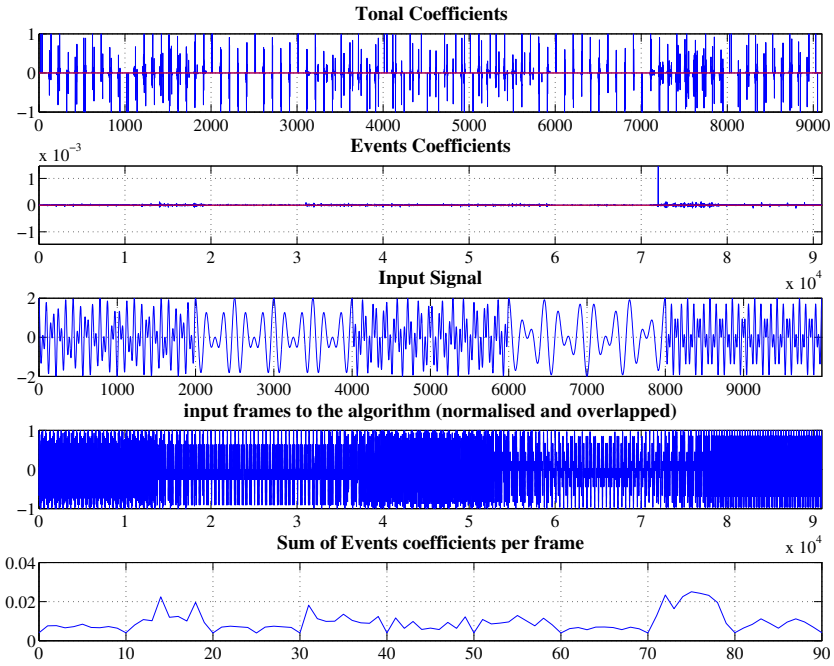
YALL1 Results Analysis

From a first look to 5.12a it seems giving weak info about the amplitude change. But looking at the Sum_U, the peaks could be distinguished except for the last one as it is a very small rise in amplitude. And here comes a very nice result for the frequency change shown in figure 5.12b, the result is sparse and representative for 3 out of 4 discontinuities. For the phase feature 5.12c, the result is disappointing unless for one event. This may be due to the example choice.

Finally when looking at the damped sinusoids projection 5.12 the results are sparse and somehow representing the events even if there exist much noise. This noise could be reduced by forcing note distance.

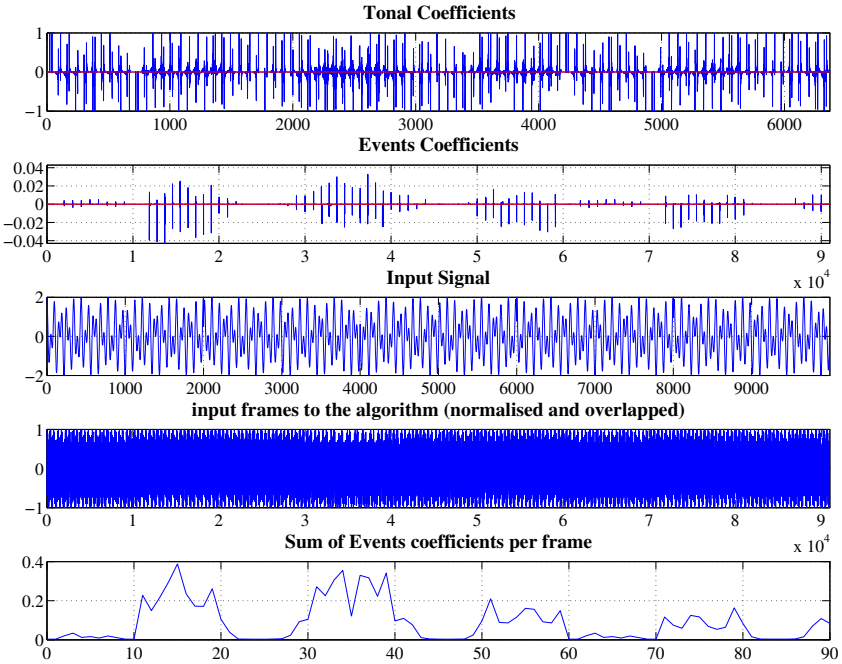


(a) Amplitude change $\lambda = 0.7$.



(b) Frequency change $\lambda = 0.7$.

Figure 5.11. [a,b] LASSO-ADMM projection for different feature change (separate).



(c) Phase change $\lambda = 0.2$.

Figure 5.10. [c] LASSO-ADMM projection for different feature change (separate).

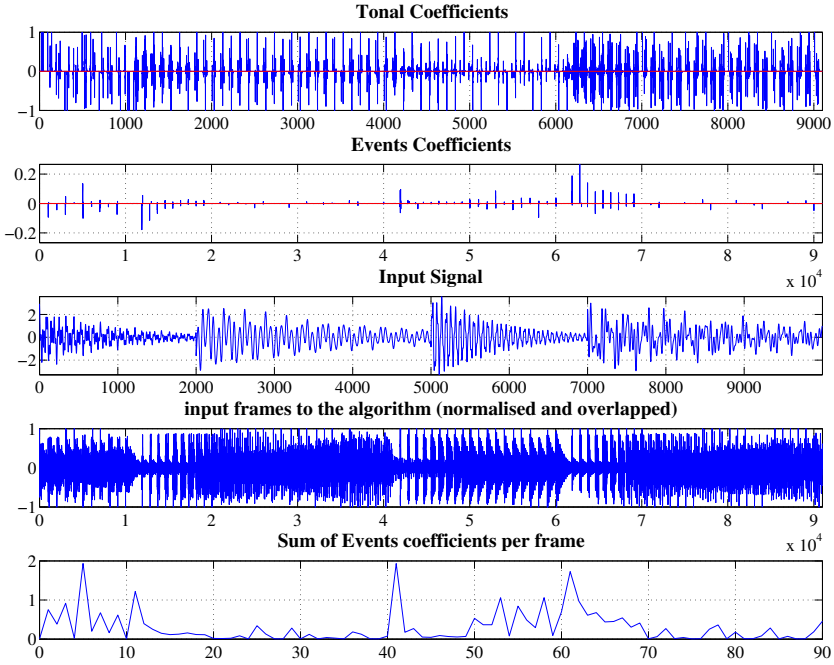
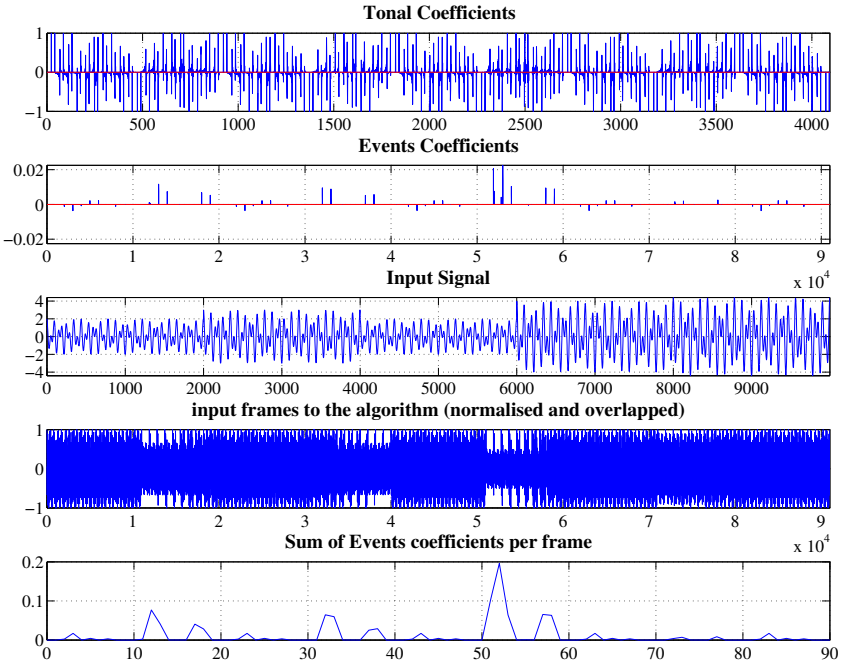
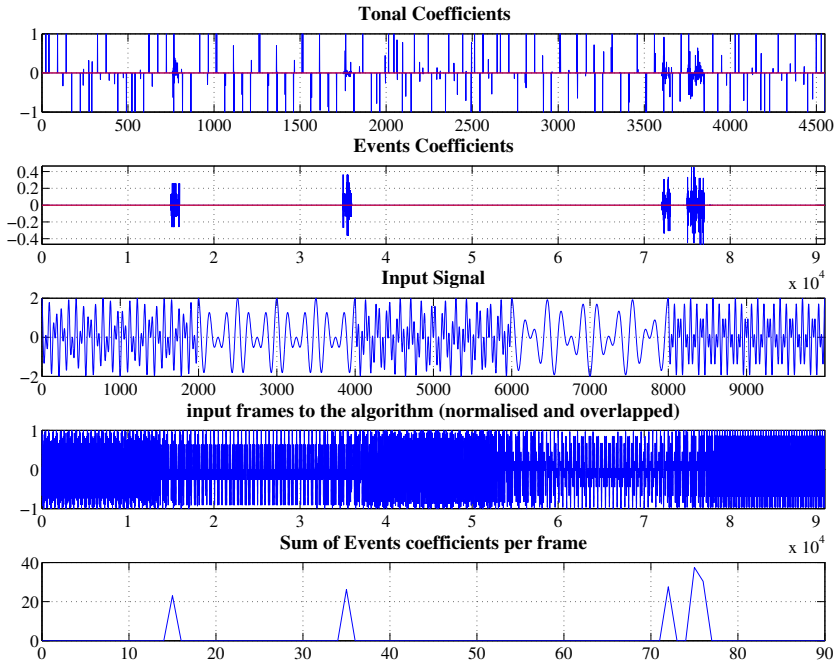


Figure 5.11. LASSO-ADMM projection for different feature change (mixed).

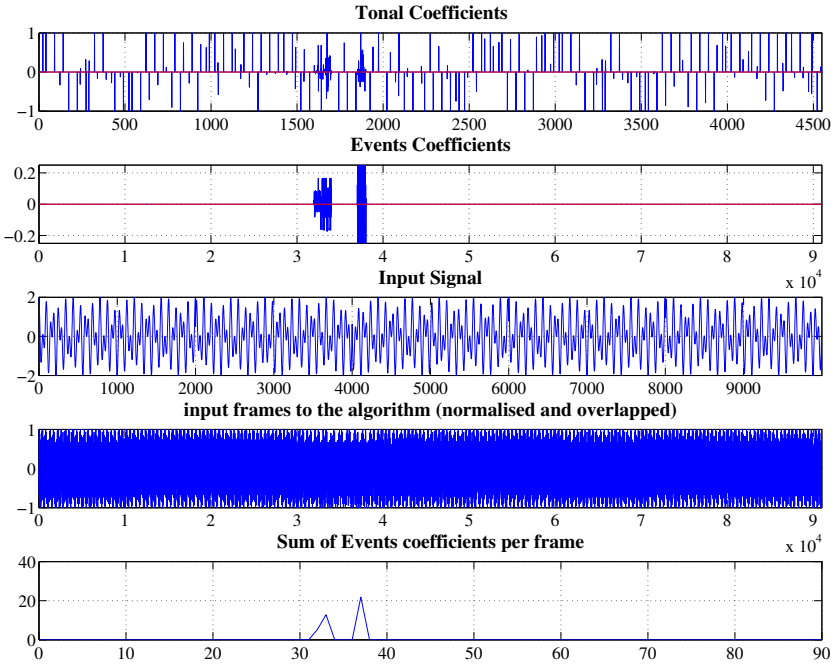


(a) Amplitude change.



(b) Frequency change.

Figure 5.12. [a,b] YALL1 projection for different feature change (separate).



(c) Phase change.

Figure 5.11. [c] YALL1 projection for different feature change (separate).

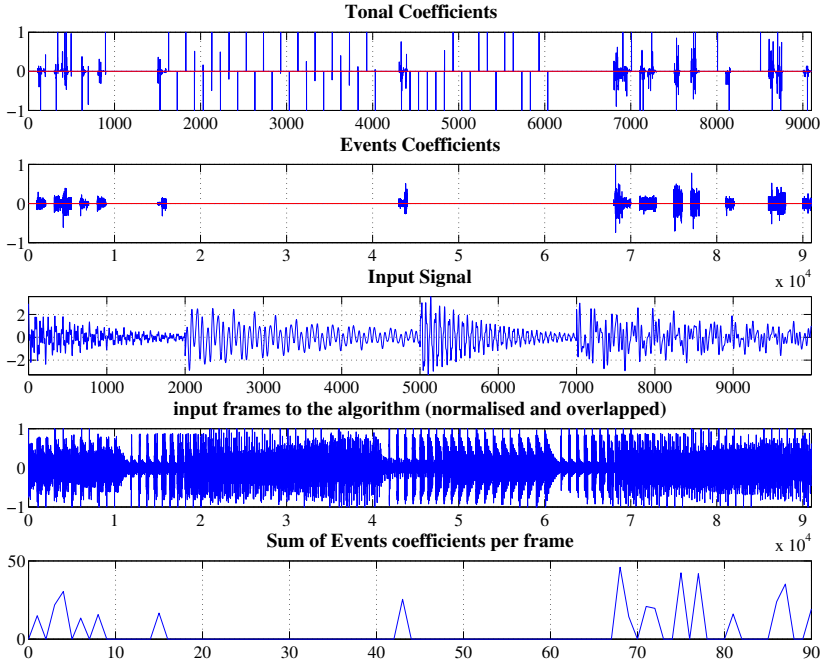


Figure 5.12. YALL1 projection for different feature change (mixed).

5.3 Use Case: Piano signal analysis

Finally we will discuss step by step the analysis for a sequence of four piano notes. The aim of this analysis is to check wether or not it is a good idea to apply sparse representation for solving note onset detection problem. It will also help us to tune the mentioned parameters to efficiently detect the note onsets in a piano melody. Later those parameters' ranges could be used to run larger experiments depending on the obtained results. Note that for this experiment we don't take into consideration the first played note as it comes out from silence, that makes it higly detectable and need to be manipulated separately. The signal will be projected using only the YALL1 implementation for the convex optimization problem.

5.3.1 Selecting Tonal Atoms count

This step is useful to increase the processing speed. When an appropriate value is selected - depending on the instrument - the algorithm can be more efficient in differentiating steady-state from transients. From the previous experiments it was noticed that the frequency content in the projection is not only dependent on the input signal itself but also on the value of λ. For that reason we ran the experiment on a range of λ having exactly those specifications:

- Tonal atoms type = DCT.
- Tonal atoms = 200 (this number is larger than the normal frequency content in a piano note projected with full DCT matrix).
- Frame size = 1000 samples.
- Overlap = 90%.
- Every input (output) frame to (from) the projection algorithm is normalised. The output normalisation is normalising transient and steady-state coefficients together so that the energy is distibuted in the same way for all frames.
- Lambda range is $0.1 \leq \lambda \leq 1.5$ with a step size equal to 0.1.
- Lambdas λ_t and λ_{ss} are set to one.

Figure 5.13 shows the limit when we keep decreasing lambda. Below a certain lambda, here lambda = 0.05 for a piano signal, all coefficients disappear and that is normal because decreasing lambda is enforcing more sparsity and less exactness. This defines a lower limit for our choice of lambda λ.

Looking at the tonal coefficients plots in both figures 5.14 and 5.15 we can realize the previously mentioned deduction, that the frequency tonal coefficients are less sparse when increasing lambda. Note that the tonal coefficients plots are zoomed in to check the average maximum tonal coefficients that represent any frame. We base our choice for the tonal atoms count on the results in figure 5.15 which reaches 100 at most.

5.3.2 Define a range for exactness Lambda (λ)

Now we refine our experiment by setting the tonal atoms count to 100 which makes the projection process much faster without information loss. From the previous experiment also we found the results shown in table 5.2.

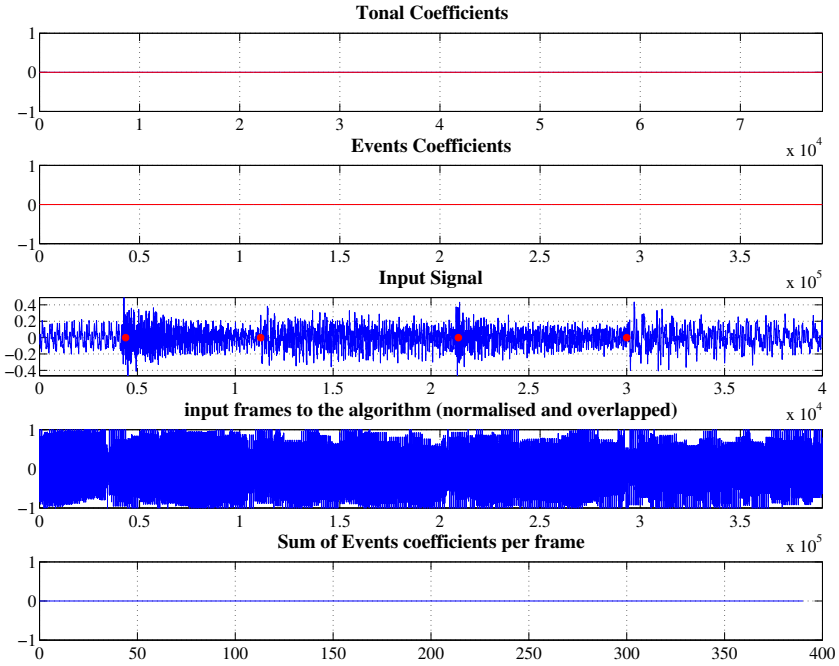


Figure 5.13. Max sparsity and no events coefficients ($\lambda = 0.05$).

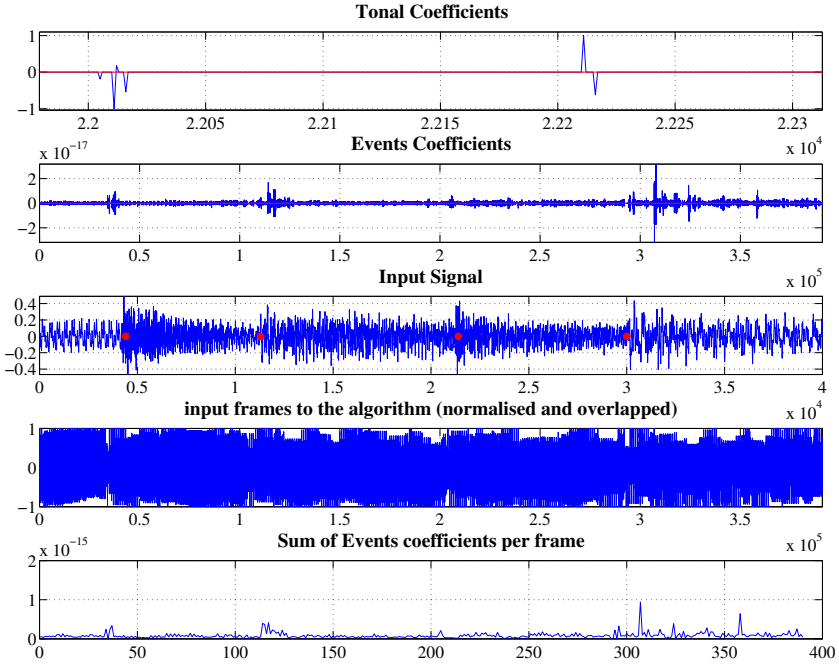


Figure 5.14. Best sparsity ($\lambda = 0.15$).

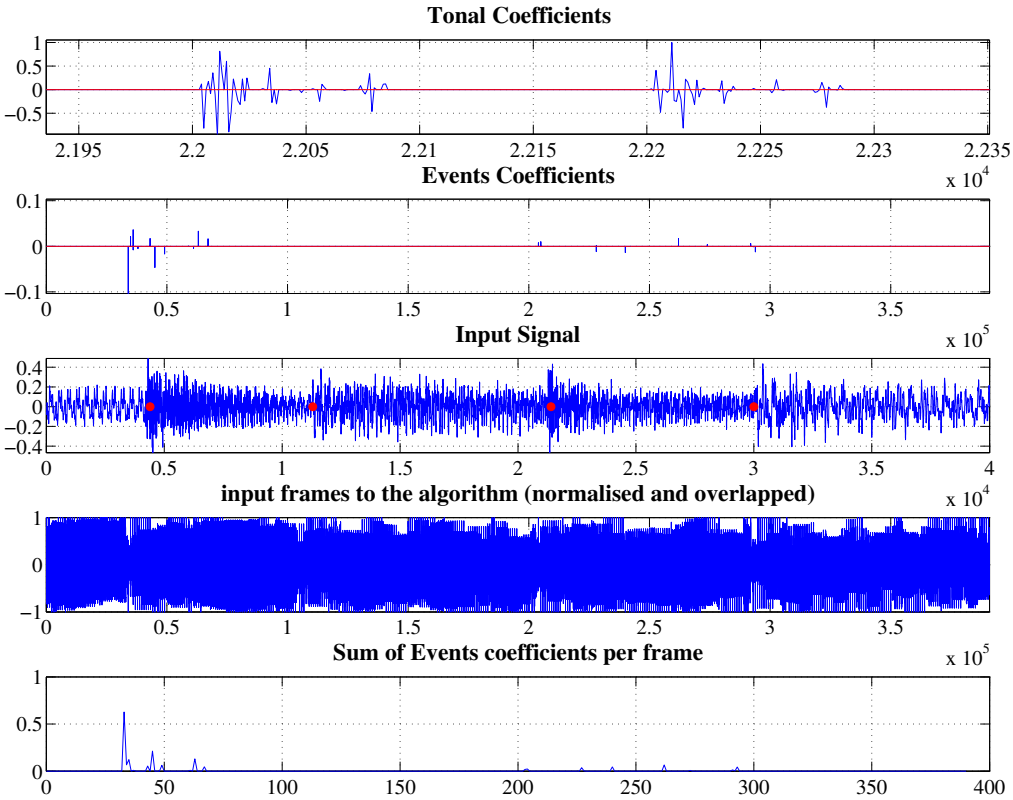


Figure 5.15. Tonal coefficients less sparse ($\lambda = 0.95$).

Lambda λ	**Sparsity**	**Order of Events Coefficients value**	**Relevant to Onsets**
≤ 0.05	max	zeroes	not at all
0.1	sparse	10^{-15}	somehow relevant
$\geq$ 0.1and $\leq$ 0.2	sparse	10^{-17}	more relevant
≥ 0.2	not sparse	10^{-17} to 0.5	difficult to extract info

Table 5.2. Lambda (λ) range check.

Having these results we decreased the experiment range for lambda: $0.05 \leq \lambda \leq 0.2$ with a tep size of 0.01. The best results obtained were for $\lambda = 0.12, 0.13, 0.14, 0.15$ and 0.16. Figures 5.16 and 5.17 are showing the results for the range borders. We can notice that the events coefficients are present near the onsets positions.

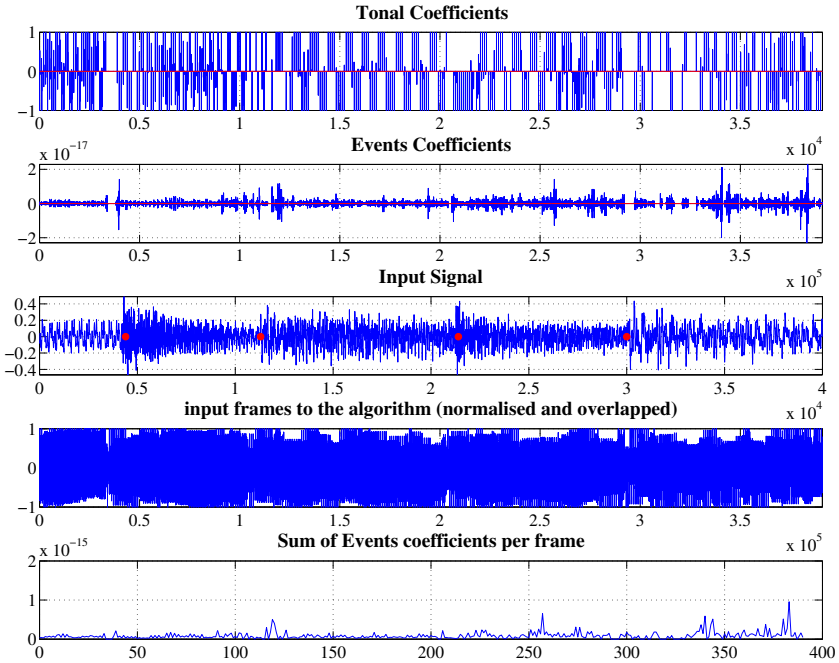


Figure 5.16. More sparse events coefficients ($\lambda = 0.12$).

5.3.3 Tuning sparsity lambdas (λ_t and λ_{ss})

First we start by tuning the lambda λ_{ss} responsible for the tonal coefficients sparsity. To decrease the experiment time we will fix lambda λ_t to one and lambda λ to only the borders of the range defined in the previous experiment (i.e. $\lambda = 0.12$ and 0.16). The experiment will run for a range $1 \leq \lambda_{ss} \leq 2$ with a step equal to 0.1.

We can see from figures 5.18 and 5.19 that when $\lambda = 0.12$ the events coefficients are sparse and representing events if $\lambda_{ss} = 1.1$ but then increasing λ_{ss} to 1.2 the result start to be less sparse. Instead when moving to $\lambda = 0.16$ the sparsity and efficiency of onset representation is preserved until $\lambda_{ss} = 1.3$ as shown through the figures 5.20, 5.21 and 5.22. That is the reason why we defined a range for λ_{ss} between 1.1 and 1.3.

Continuing the experiments to tune λ_t we fixed λ and λ_{ss} to the values that gives more or less the best results for the previous experiments ($\lambda = 0.16$ and $\lambda_{ss} = 1.3$). We ran experiments with a range from 1 to 5 for λ_t using a step of 0.11. We found that changing its value is not affecting that much the results. This can be seen in the plots 5.23 and 5.24 where there is no difference between the events coefficients. It is still possible that this parameter could affect the result when trying on larger input signals or in other application areas.

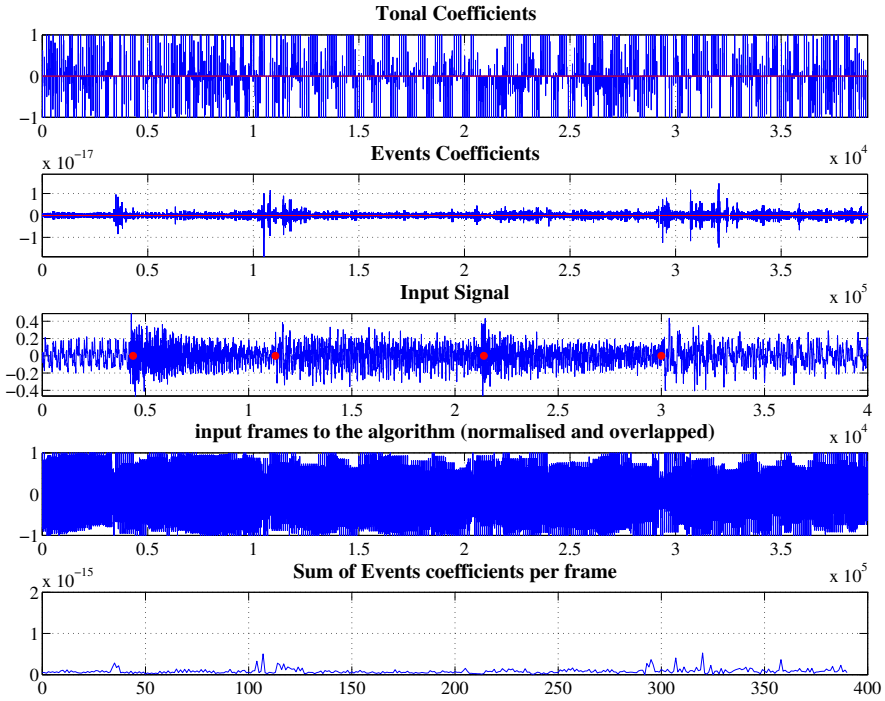


Figure 5.17. Less sparse events coefficients ($\lambda = 0.16$).

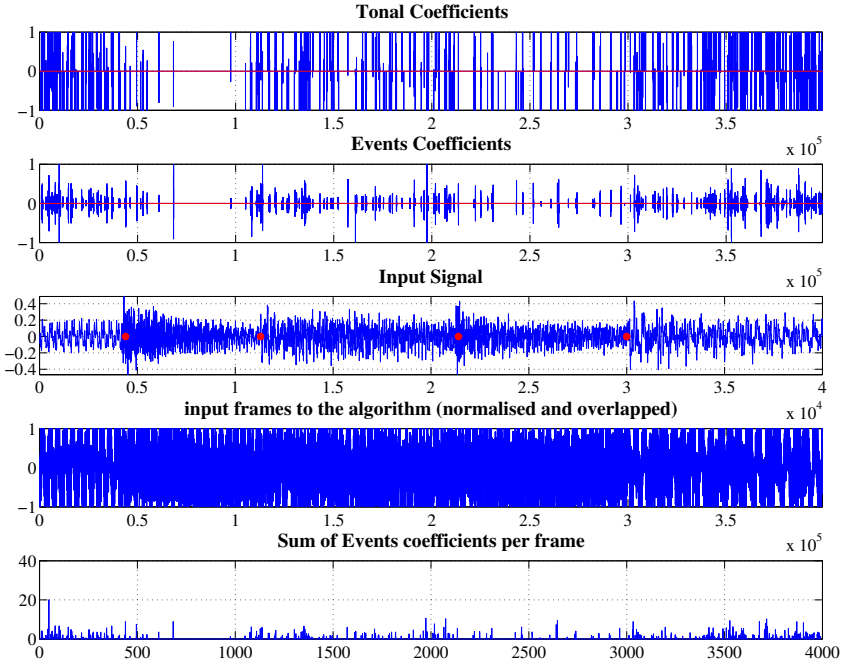


Figure 5.18. Choosing Lamda Tonal ($\lambda = 0.12, \lambda_{ss} = 1.1$).

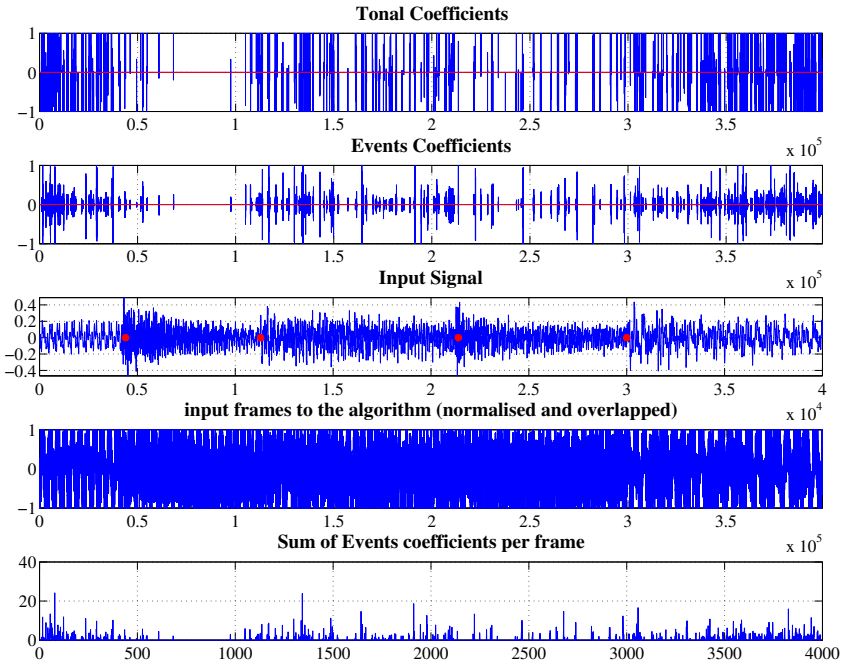


Figure 5.19. Choosing Lamda Tonal ($\lambda = 0.12, \lambda_{ss} = 1.2$).

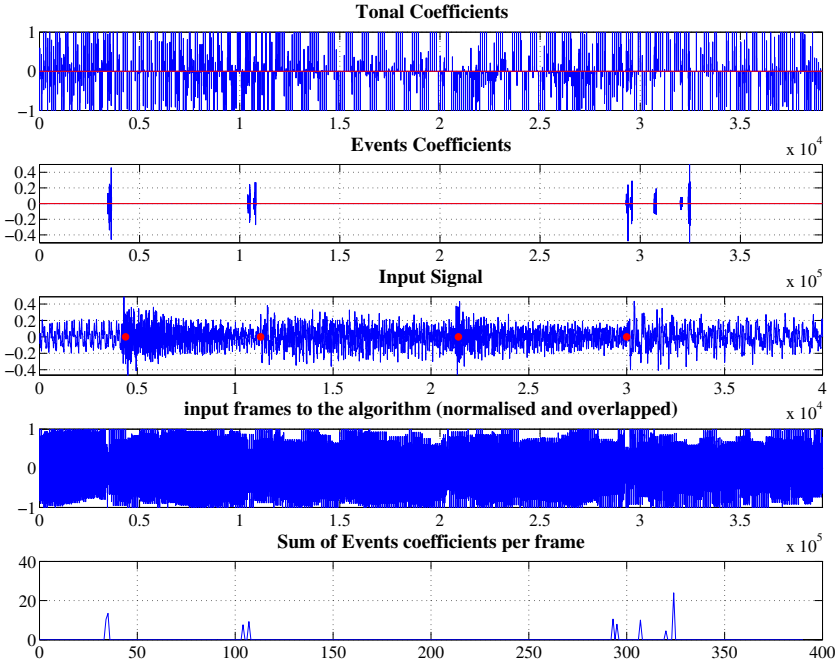


Figure 5.20. Choosing Lamda Tonal ($\lambda = 0.16, \lambda_{ss} = 1.1$).

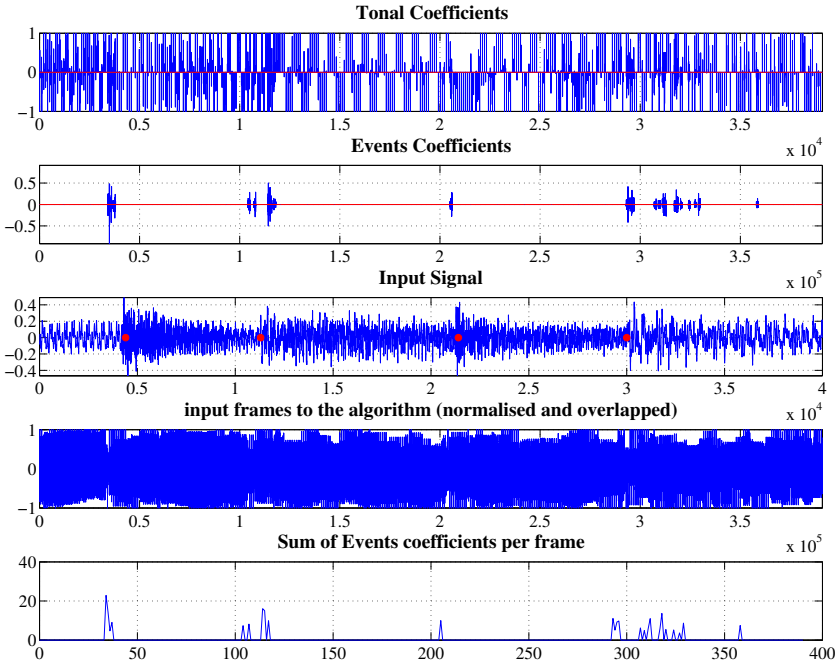


Figure 5.21. Choosing Lamda Tonal ($\lambda = 0.16, \lambda_{ss} = 1.2$).

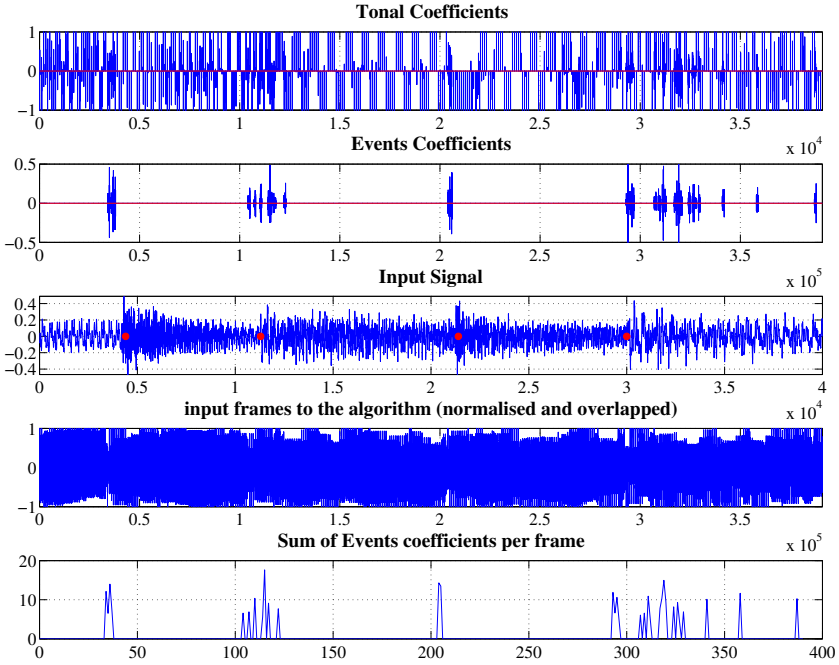


Figure 5.22. Choosing Lamda Tonal ($\lambda = 0.16, \lambda_{ss} = 1.3$).

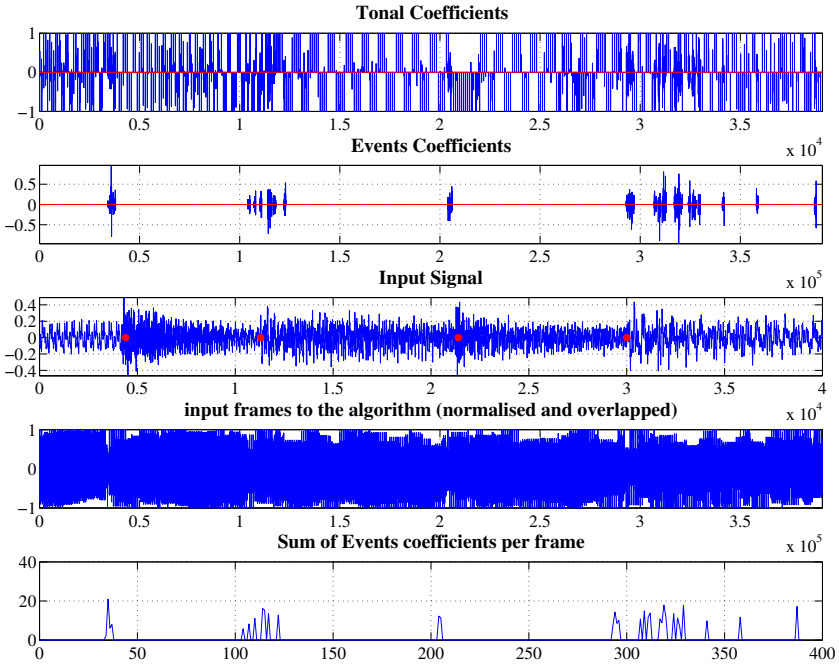


Figure 5.23. Choosing Lamda Transients ($\lambda = 0.16, \lambda_{ss} = 1.3\ \lambda_t = 1.9$).

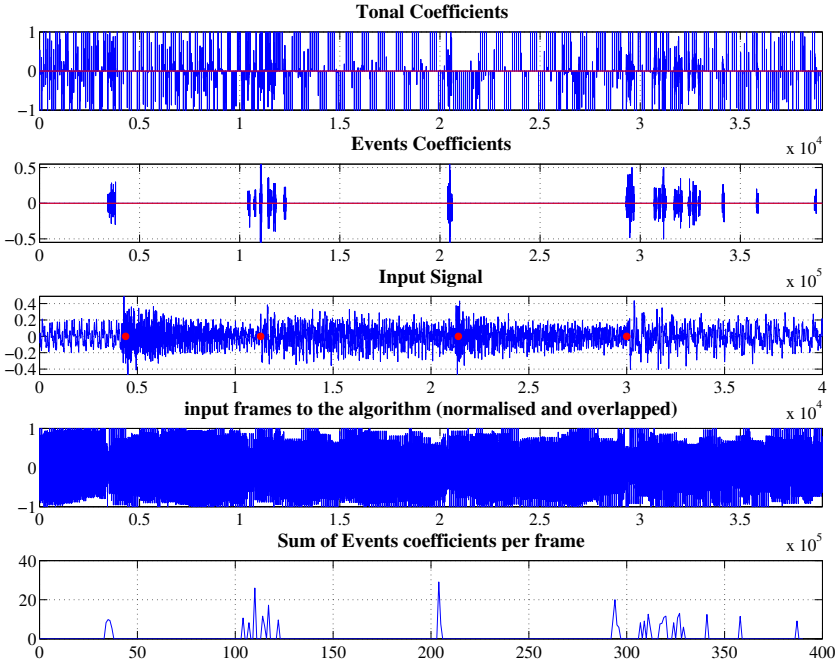


Figure 5.24. Choosing Lamda Transients ($\lambda = 0.16, \lambda_{ss} = 1.3\ \lambda_t = 4.4$).

5.3.4 Peak-picking

The final point in the analysis use case is to apply the detetion peak-picking algorithm on the projection output (both the events coefficients and the sum of events coefficients). That is to show how the output for a full onset detection process would be. Given an input melody formed by a sequence of notes, our problem was to find few points that represent those notes (their onsets). Here we show some of the results when the proposed peak-picking algorithms are applyed to the piano use case after adjusting the other parameters. Those experiments were applied on the YALL1 projection output having the parameters: $\lambda = 0.16$, $\lambda_{ss} = 1.3$, $\lambda_t = 1.9$, tonal atoms count = 100, frame size = 1000, overlap=90% and the atoms kind is DCT. For the last three experiments we used a note distance equal to 6 as we know that the minimum distance between any two note onsets is approximately 7 frames. For the threshold we chose the best results to plot here.

Note that in the first two figures 5.25 and 5.26, the detected onsets are shown on a time scale from 1 to 400. They represent the 400 frames resulting from processing 40000 samples with 90% overlap. In a subsequent step each of the detected onsets should be mapped to the corresponding one from the original 40000/1000 = 40 frames. For simplicity in this case we can just divide by 10.

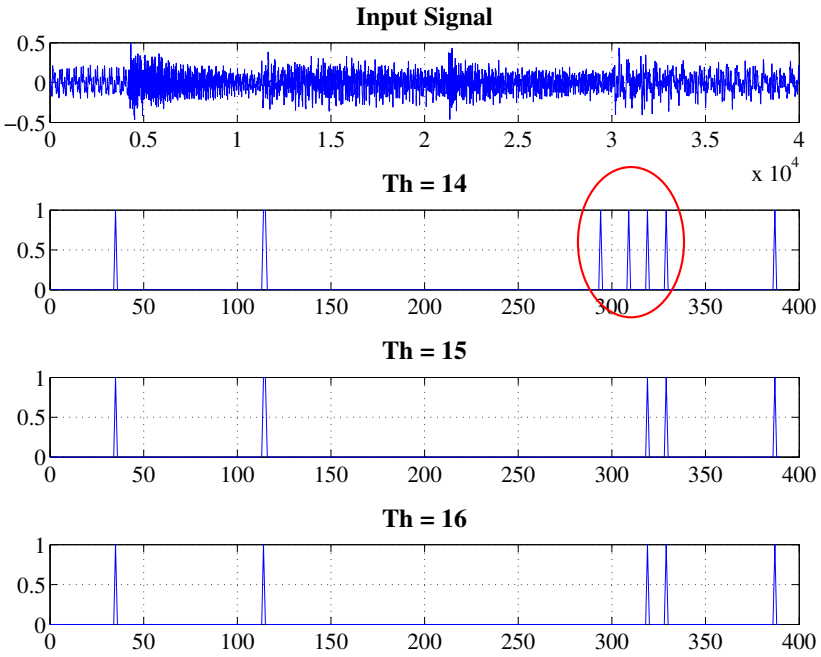


Figure 5.25. Comparing Sum_U with a threshold.

In 5.25 we can easily notice the detection of many onsets that are near. This is because the first algorithm is not respecting the note distance while processing the Sum_U's. This is fixed when applying the second algorithm as shown in 5.26.

To better understand the results, the output is marked with different markers:

- *Green circle:* This is used to denote that an onset is detected when it is supposed to happen.

- *Red circle:* This is used to denote that an onset is detected when there are no onsets to be detected at this point. This is called a **False Alarm**.
- *Red arrow:* This is used to denote that an onset was not detected when it is supposed to happen.

It is easy to deduce that increasing the thresholds reduces the number of *False Alarms* but could decrease the number of detected real onsets. In other words it decreases the *false alarm probability* and *detection probability* simultaneously.

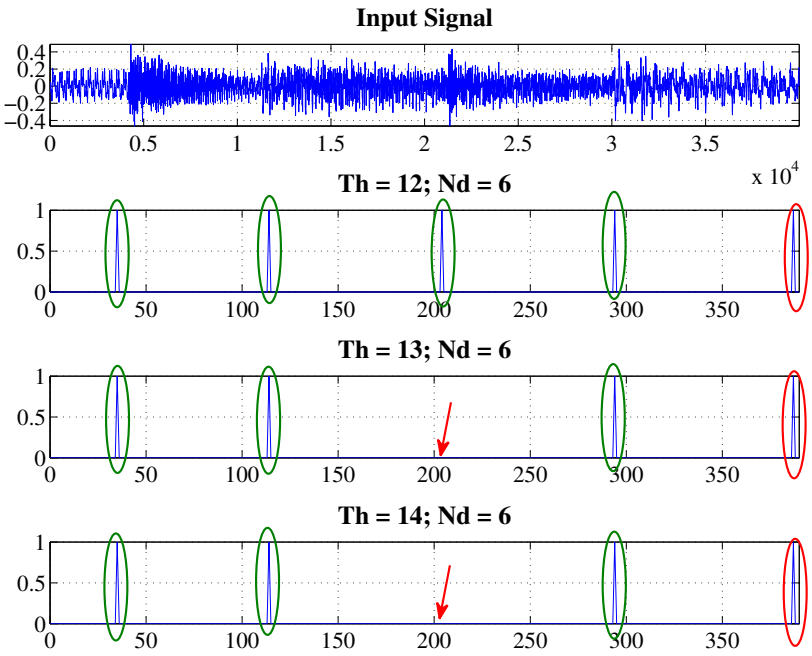


Figure 5.26. Comparing Sum_U with a threshold using note spacing.

For the last two algorithms where the first one selects the first maximum peak and the other one selects the locally maximum peak in the events coefficients.Comparing their results by looking at figures 5.27 and 5.28, we can see that some detected onsets moved slightly their positions - this is marked with the dotted circles. This is due to the difference between the two algorithms in applying the note distance constraint as described in 4.4.

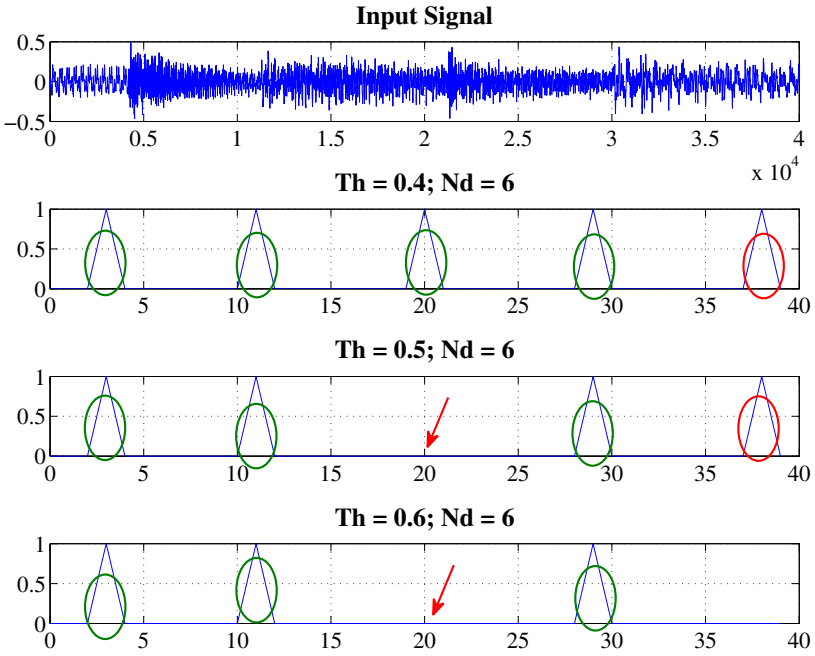


Figure 5.27. Comparing U with a threshold (First).

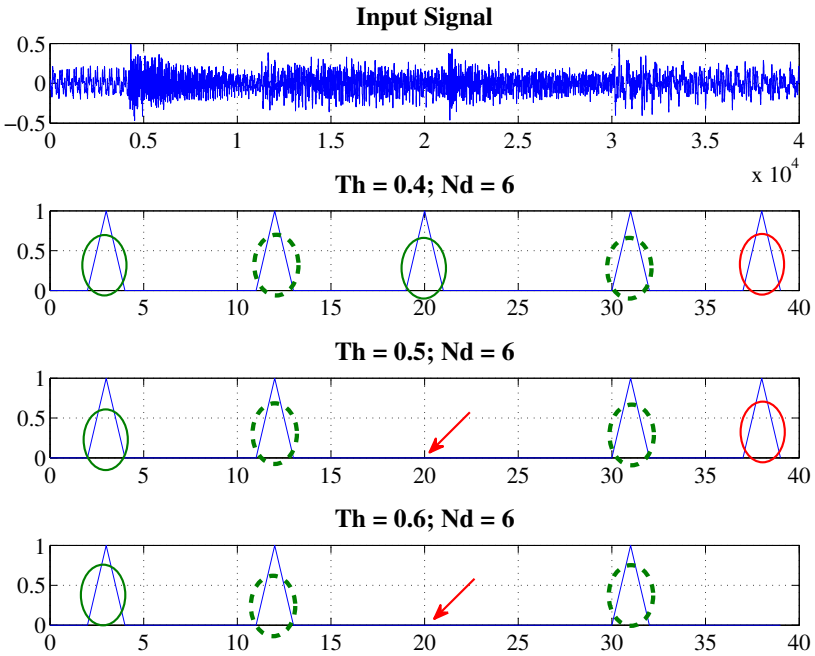


Figure 5.28. Comparing U with a threshold (Local Maxima).

5.4 Conclusion

Different methods were proposed for note onset detection. In the presented work we decided to implement one of the newly emerging ideas that is based on sparse signal representation techniques. The main purpose of this research was to investigate the efficiency of the proposed solution, discover the different parameters that play a role in getting better detection probability and get some preliminary results from different models and use cases.

The real benefit is that we came up with a good testing platform consisting of various components implementing different analysis steps. By getting different parameters from the user it is able to model a generic input signal synthetically. In our case it modeled a sequence of musical notes. Another component is responsible for generating musical pieces by mixing different pre-recorded notes from different instruments. Then a component for processing the input signal and providing two kinds of visual results. The first consists of the projection output which is used to tune the algorithm parameters for a specific case study. The second provides the detection output which represents the detected onsets which could be used later to evaluate the detection algorithm for different parameters settings. This testing platform was made as generic as possible to check and compare the efficiency of different optimization algorithms with different settings.

Analysing the isolated notes helped us to understand where in time an onset is detected hence we could define more precisely the position of transients for musical signals. With respect to our analysis results an onset is detected when only a very small portion of transient is within the processing frame. This notion enhances the resolution of the detection algorithm output. The detection algorrithm signals some frames as containing onsets. More precisely those onsets' position on the time axis is near the end (border) of those frames.

This frame is having the highest projection coefficients. So an onsets are at the end of frames having largest coefficients.

From the various analysis steps presented earlier in this chapter, we found some promising but not yet complete results when applying the sparse representation for musical signals in order to detect onsets. By looking at the simple case study for the piano musical sequence, we see that the detection algorithm is giving a good detection probability - it is detecting three out of four onsets. In the same time it is having a low false-alarm probability as it makes only one false-alarm detection. For more generic results, exhaustive experiments on long note sequences should be run as will be discussed in section 5.5.

We also had some important conclusions about the tuning parameters. It is much better from a computational point of view to use DCT atoms rather than DFT ones. They almost give the same projection results but the DFT atoms make the projection much slower because of their complex nature. Three parameters λ, λ_{ss} and λ_t were used to tune the optimization solution exactness, sparsity of steady-state coefficients and transient coefficients respectively. Out of the three lambdas we only found that λ and λ_{ss} are the ones that affect the solution. For the piano use case the best ranges were from 0.12 to 0.16 for λ and from 1.1 to 1.3 for λ_{ss}.

Finally for the peak-picking step, the algorithms applying *local maxima search* are giving slightly better resolution than the basic ones. The note distance parameter is an important

measure for the solution efficiency. For the tested piano sequence, this parameter was set to a value of six when a frame size of 1000 samples was used. This means that the algorithm was detecting efficiently for a minimum distance of 0.136 sec between two consecutive notes.

5.5 Future work

There are a number of open issues to move towards a more complete analysis for the musical signal and detecting the onsets position. While developing our testing platform we came out with many ideas that may help a lot in the problem solution and making the platform more generic. Here we present some of those ideas:

- Trying different types of tonal atoms to represent the steady-state portion of the signal. For example the *Modified Discrete Cosine Transform* (MDCT) which is well suited when the input signal is processed on frame by frame basis and those frames are overlapping. This lapped transform may be more efficient in producing a sparse signal representation.
- Also it is useful to try more sophisticated atoms construction for the transients like the *Discrete Wavelet Transform* which transforms the signal showing its frequency and temporal properties.
- Continuing our research by running exhaustive searches for many instruments and trying to find the best values for the algorithm tuning parameters. In future we want to cover special note sequence cases that are not covered throughout this research such as:
  1. one note played after silence, for example the first note.
  2. one note happening two or more times in a sequence.
  3. two notes sharing a number of harmonic frequencies played one after another (same note but different octaves).
- One important issue is to be able to differentiate between *Onsets* and *Offsets*. An Offset is within the release phase of the played note. Through our research we assumed that Onsets have higher impact than Offsets which is not always the case specially when having multiple overlapping notes.
- Another direction for enhancement would be applying some pre-processing for the input signal before undergoing the reduction algorithm. A good idea is to downsample the input frames in a way to remove the signal information irrelevant to human perception for transients.
- Adding a consensus component to our testing platform that gathers detected events resulting from different solvers and decide which of them are correct depending on a decision policy. The different solvers could be solvers with different algorithm implementations but this is very computationally expensive. The solvers we propose are different in the way of detection (peak picking).
- We want to run the detection algorithm on recordings of single and multiple instruments. This is to really check the application in the real world with a fully random experiment and not just the notes we are mixing and knowing their properties.

Chapter 6

Business Plan

6.1 Executive Summary

Dealing with digital technology instances has conventionally proceeded in accordance with hand-manipulated controls such as keyboard, mouse, joystick ...etc. At that time voice control was viewed as an unacceptable substitute for such hand-operated devices because of the complexity involved implementing this new input method and because of the time constraints associated with issuing and recognizing the spoken commands [Reimer, 1987].

Today the reality is totally different; voice control is becoming one of the most important digital interaction solutions. Vlad Sejnoha, chief technology officer of Nuance Communications said "We are at a transition point where voice and natural-language understanding are suddenly at the forefront ". she thinks that speech recognition is really going to upend the current [computer] interface [Knight, 2012]. Confirming that it is a revolution Aaron Shapiro, CEO at Huge said "businesses will have to adapt themselves for a new form of digital interaction that most are completely unprepared for. It's one that will potentially be more challenging than the social and mobile revolutions that have shaken up so many business models to date"[Shapiro, 2013].

VOCONINO, Inc. is a start-up company in switzerland created to provide efficient voice control solutions and services. The company aims to provide different customers (individuals and businesses) with solutions customized to their needs competing with the few existing ones in terms of speed, simplicity and efficiency.

Voice control is beneficial in many domains that could be divided in software application control and hardware control. For the software applications, the technology was introduced twenty years ago and its market is controlled by one big company *Nuance*. On the other hand, hardware voice control is used in cars, home automation, monitoring people in hospitals, retirement homes and even monitoring animals for early detection of diseases. Most of the voice control in the hardware application domain are designed and implemented by those businesses themselves (home made simple solutions). As the technology is evolving everyday and those companies are not specialized in voice control, their solutions are less efficient.

Therefore there is a potential market for serving those businesses to coop with this new interaction method. Starting the business, **VOCONINO** will focus on two of the mentioned markets: car production and home automation. In 2012 the number of cars produced globally reached 84 millions per year. For example Volex Wagon produces 8 million per year, BMW produces 2.5 millions and Mercedes produces 2 millions. They are either producing their own voice control modules or buying it from each other. For the home automation VOCONINO will initially focus on the Swiss market. Voice control of appliances is usually considered as a luxury. That is true and would be a good reason why it will be successful entering the Swiss market: Switzerland has 45.2% of its population having more than 50 years old. Old people and especially in Switzerland are the ones who have money and ready to pay for luxury.

It may seem that voice control is only a luxury tool and that it is better to use a remote control for home automation. In other words speaking a command would take more time to be processed and understood than just pressing a button because in most of the current voice control modules or software, the speaker has to adapt his voice to the voice control module and not the other way around. This is not the case for persons with physical impairments where voice control is a viable solution for them. Voice control could improve significantly the quality of life of a handicapped person.

As a result from a research conducted the last few months in **ALaRI** (Advanced Learning and Research Institute) we came out with a new algorithm detecting events in audio data. **VOCONINO** will use this algorithm in implementing its first product **VOCOIC** a voice control chip that will surprise the market with its efficiency. It is supposed to have an efficiency increase of 25% with respect to the available solutions. This chip will be designed in such a way to be generic, fast, small and with high robustness.

Simultaneously **VOCONINO** will introduce another product **VOCOSAYIT** which is a basic home automation module controlled with voice. This module is based on the **VOCOIC** chip in addition to a small control module for appliances that could be controlled wirelessly. Introducing the **VOCOSAYIT** with a medium price and high efficiency aims principally to brand the start-up company name. The customers would be able to check the technology efficiency in their homes before requesting it in different appliances. On the other hand, **VOCONINO** considers introducing this product as an indispensible part of its role towards the society. The algorithm behind this technology was designed to adapt itself with the speaker's voice and not the other way around.

The first year will be dedicated for designing and implementing the chip and the module. The marketing will start after 6 months by launching a website that is vocally controlled. The main objective of this website is to push customers to use voice control and feel how comfortable it is especially with our efficient algorithm. The website access is free and the users can also download from it a trial app on their phones.

VOCONINO is founded by Mina Shehata who has discovered the new idea for extracting events from voice within his research. He will also take hands-on management role in the company. He will be managing 3 teams: HW Engineering Team, SW Engineering Team and Marketing and Sales Team.

The company will finance its operation through an entrepreneur's capital of 5 million CHF and a bank loan of 2 million CHF. The company is very promising as it starts to achieve a revenue equal to 25% of the initial capital starting from the second year and the bank loan will be totally reimbursed after four years.

To check the Business Plan in details go to ALaRI webpage:
`http://www.alari.ch`

Appendix A

Appendix

Matlab Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                   Mix Tones
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Action:
%   A function to mix some registered tones from a specific instrument
%   to form a song. The generated song is used to test the projection
%   algorithm. It saves the melody together with the expected onset
%   positions.
%
%%  Input:
%
%%  Output:
%   fileName: the name of the file to which the melody is saved
%
%   outSong: the constructed melody out of single notes.
%
%   onsetPosition: a vector containing the expected onset positions that
%   are given by the user to be used for checking the detected onsets.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% [onsetPosition]=mixTones()
function [fileName,outSong,onsetPosition]=mixTones()

%% select the input instrument
instrument = 'acordeon';
instrument = input('enter instrument name:  ');

% input number of tones to load
nTones = input('enter number of tones to load:  ');

%*** modify to load tones randomly
% load n tones
tonesLength = zeros(nTones,1);
tonesStarting = ones(nTones+1,1);
tonesLibrary=[];
```

```
for iTones=1:1:nTones
    toneNumber = input('enter tone number(name):  ');
    tempTone=[];
    [tempTone,fs] = ...
     wavread(['../musicIO/' instrument '/' num2str(toneNumber) '.wav']);
    tonesLibrary = [tonesLibrary;tempTone];
    tonesLength(iTones) = length(tempTone);
    tonesStarting(iTones+1) = length(tonesLibrary)+1;
end

figure;
plot(tonesLibrary);

% ploting the read notes to help in mixing
figure;

for iTones=1:1:nTones
    subplot(nTones,1,iTones);
    hold on
    plot(tonesLibrary(tonesStarting(iTones):...
        tonesStarting(iTones)+tonesLength(iTones)-1,1),'b');
  %  plot(tonesLibrary(tonesStarting(iTones):...
  %      tonesStarting(iTones)+tonesLength(iTones)-1,2),'r');
    hold off
    grid;
end

disp('press any key to continue');
pause;


%% mixing and creating the song
zerosAtTheBegining=1000;
nNotes=input('enter number of notes to mix(>=1):');

iNote=1;
outSong=[];
onsetPosition=zeros(nNotes,1);
while(iNote<=nNotes)
    toneSeqInLibrary=...
        input(['enter note sequence(1-' num2str(nTones) '): ']);
    onsetPosition(iNote) = input('Enter onset position: ')...
        +zerosAtTheBegining;

    if(iNote~=1)
        notePosition = ...
        input('enter the position to insert the note: ')...
        +zerosAtTheBegining;

        zerosAtTheEnd=tonesLength(toneSeqInLibrary)-...
            (length(outSong)-notePosition+1);
        if(zerosAtTheEnd>=0)
            old=[outSong;zeros(zerosAtTheEnd,2)];

            new=[zeros(notePosition-1,2);...
                tonesLibrary(tonesStarting(toneSeqInLibrary):...
                tonesStarting(toneSeqInLibrary)+...
                tonesLength(toneSeqInLibrary)-1,1:2)];
        else
```

```
            old=outSong;

            new=[zeros(notePosition-1,2);...
                tonesLibrary(tonesStarting(toneSeqInLibrary):...
                tonesStarting(toneSeqInLibrary)+...
                tonesLength(toneSeqInLibrary)-1,1:2);...
                zeros(length(outSong)-notePosition+...
                1-tonesLength(toneSeqInLibrary),2)];
        end
        outSong=old+new;

    else
       outSong = [zeros(zerosAtTheBegining,2);...
           tonesLibrary(tonesStarting(toneSeqInLibrary):...
        tonesStarting(toneSeqInLibrary)+...
        tonesLength(toneSeqInLibrary)-1,1:2)];
    end
    iNote=iNote+1;
end

tic
soundsc(outSong,fs)
toc
fileName=input('Enter a name for the file to save(use ''''): ');
save(['../musicIO/' fileName '.mat'],'outSong','onsetPosition');

% plotting mix with selected onsets
figure;
hold on
plot(outSong(:,1));
plot(onsetPosition,zeros(length(onsetPosition)),'or')
hold off
%% spectrogram analysis
% figure
% plot(outSong(1:100000,1));
% figure
% spectrogram(outSong(1:80000,1),2205,1000,fs,'yaxis');
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                            Model Input Signal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Action:
%   generates a virtual(synthetic) input to be analysed by the system,
%   normally the generated signal is used to model a specific signal
%   kind mixing harmonics or successive decaying exponential tones
%   (blocks)
%
%%  Input:
%
%%  Output:
%   inputSignalSynthetic: is the generated model for the input signal
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%  [inputSignalSynthetic] = ModelInputSignal()
function [inputSignalSynthetic] = ModelInputSignal()

```

```
20
21 %% get how many successive blocks to be created
22 nBlocks=input('Enter number of successive blocks to be created:  ');
23 while(nBlocks<0)
24     nBlocks = input('Please enter valid number (≥0): ');
25 end
26
27 for i=1:1:nBlocks
28
29     disp(['Block #' num2str(i)]);
30
31     % get the number of cosines (harmonics) in one block
32     nOfCosines=input('Enter number of cosines:  ');
33     while(nOfCosines<0)
34         nOfCosines = input('Enter a valid number of cosines(≥0): ');
35     end
36
37     % get the number of pulses in one block
38     nOfPulses=input('Enter number of pulses:  ');
39     while(nOfPulses<0)
40         nOfPulses = input('Please enter valid number of pulses(≥0): ');
41     end
42
43     % get the number of samples to be in one block
44     nSamples=input('Enter number of samples in this block:  ');
45     while(nSamples<0)
46         nSamples = input('Enter valid number of nSamples (≥0): ');
47     end
48     t=1:nSamples;
49     block=zeros(1,nSamples);
50
51     % mix harmonics and pulses for one block
52     for j = 1:nOfPulses
53         disp(['pulse #' num2str(j)]);
54         pulse=zeros(1,nSamples);
55         timeOfPulse=input('Enter time of pulse (1<t<framesize):  ');
56         pulse(1,timeOfPulse)=input('Enter amplitude of pulse: ');
57         block=block+ pulse;
58     end
59
60     for k = 1:nOfCosines
61         disp(['cos #' num2str(k)]);
62         frequency=input('Enter frequency of cosine:  ');
63         A=input('Enter Amplitude of cosine:  ');
64         phi=input('Enter phi of cosine:  ');
65         block=block+ A*cos(pi*2*t*frequency/nSamples + phi);
66     end
67
68     % get and multiply by a decaying factor e^(-x)
69     decayingFactor=input('Enter number of decayingFactor:  ');
70     while(decayingFactor<0)
71         decayingFactor = input('Please enter (10>decayingFactor≥0): ');
72     end
73
74     block = exp(- decayingFactor/nSamples*t).*block;
75
76 %%ploting
77 %     subplot(nBlocks+1,1,i);
78 %     plot(block);
```

```
79
80      % add current block to the previous ones
81      % get where to insert the current block
82      if(i≠1)
83          blockOffset = ...
84          input('enter the number of overlapping samples: ');
85      inputSignalSynthetic=...
86          [inputSignalSynthetic zeros(1,length(block)-blockOffset)]+...
87          [zeros(1,length(inputSignalSynthetic)-blockOffset) block];
88      else
89         inputSignalSynthetic = block;
90      end
91  end
92
93  % transform signal to vector
94  inputSignalSynthetic=inputSignalSynthetic';
95
96  %%plotting
97  % subplot(nBlocks+1,1,i+1);
98  % plot(inputSignalSynthetic);
```

```
1   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2   %                          Testing Environment
3   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4   %%  Action:
5   %   this code is used to help the user to run different kind of tests
6   %   varying the parameters, the input type and selecting which output
7   %   is needed to be shown.
8   %   The list of different tests are:
9   %       1) Generate an artificial input to model the signal.
10  %       2) Mix notes from different instruments to generate melodies for
11  %          test.
12  %       3) testing the algorithm varying the parameters.
13  %       4) Plotting the detected events
14  %       5)Evaluate the detection algorithm.
15  %
16  %%  Input:
17  %
18  %%  Output:
19  %
20  %   coefficients: the projection coefficients (Tonal and Events).
21  %
22  %   analysisData: Some exploited charachteristics from coefficients.
23  %
24  %   inputSignal: this is the entire input signal.
25  %
26  %   detectedVector: the detected onsets
27  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
28
29  %%  [coefficients,analysisData,inputSignal,detectedVector]
30  %       = TestEnvironment()
31  function [coefficients,analysisData,inputSignal]=...
32      TestEnvironment()
33
34  %% Get User Input
35  [testName,inputSignal,onsetPosition,starting,projectionParam,...
36      algoParam] = UserInterface();
```

```
37
38 %% Plot input signal (signal under test)
39 figure
40 plot(inputSignal);
41 disp('if this is the intended signal to test press any key'...
42     'to continue');
43 pause
44
45 %% preparing the overcomplete dictionary
46 % if matrix size exeeds the 1000, it is better to be saved and ...
       loaded to
47 % use and not generated each time. Which kind of dictionary atoms?
48 % selecting the atoms is different for each
49 switch projectionParam.atomsKind
50     case 0
51         DCT_matrix=dctmtx(projectionParam.frameSize);
52         IDCT_matrix=DCT_matrix';
53         projectionMTX= ...
54             [IDCT_matrix(:,1:projectionParam.tonalAtomsCount)...
55             projectionParam.pulseValue*eye(projectionParam.frameSize)];
56     case 1
57         proTonalAtomsDFT=projectionParam.tonalAtomsCount/2;
58         DFT_matrix=dftmtx(projectionParam.frameSize);
59         % normalised idft matrix
60         IDFT_matrix=conj(DFT_matrix)/sqrt(projectionParam.frameSize);
61         projectionMTX= [IDFT_matrix(:,1:proTonalAtomsDFT)...
62             IDFT_matrix(:,projectionParam.frameSize +1- ...
63             proTonalAtomsDFT:projectionParam.frameSize) ...
64             projectionParam.pulseValue*eye(projectionParam.frameSize)];
65     otherwise
66
67 end
68
69 % Check other transient representation (more pulses)
70 % f=projectionParam.frameSize;
71 % IDCT_matrix= [IDCT_matrix(1:projectionParam.frameSize,:) ...
72 %     eye(f)+[zeros(1,f); eye(f-1,f)]];
73
74 % to generate a 3D plot of the coefficients evolution each iteration
75 % coeffEvol=[];
76
77 % To test different combination of lambda's
78 % loop on lamda general range
79 for lamdaGeneral = ...
80         algoParam.lamda.general.lower:...
81         algoParam.lamda.general.precision:...
82         algoParam.lamda.general.upper
83 % loop on lamda events range
84 for lamdaEvent = ...
85         algoParam.lamda.eventAtoms.lower:...
86         algoParam.lamda.eventAtoms.precision:...
87         algoParam.lamda.eventAtoms.upper
88 % loop on lamda tonal range
89 for lamdaTonal = ...
90         algoParam.lamda.tonalAtoms.lower:...
91         algoParam.lamda.tonalAtoms.precision:...
92         algoParam.lamda.tonalAtoms.upper
93 %% Run the algorithm
94 [coefficients,analysisData] = ...
```

```
95      RunAlgorithm(projectionParam,algoParam,lamdaGeneral,lamdaEvent,...
96      lamdaTonal,projectionMTX,inputSignal);
97
98  % x=abs(real(coefficients.event))';
99  % x(x<0.025)=0;
100 % coeffEvol=[coeffEvol ;x];
101
102 %% Plot Analysis Results
103 PlotProjectionResults(testName,algoParam,lamdaGeneral,lamdaEvent,...
104     lamdaTonal,projectionParam,inputSignal,... %(:,1) to fix smthn
105     onsetPosition(:,1)-starting,coefficients,analysisData);
106
107
108 %% Detect Onsets
109 algorithm =input('enter peak-picking algorithm choice: ');
110 threshold =input('enter threshold 1st: ');
111 noteDistance =input('enter noteDistance: ');
112 detectedVector= ...
113     DetectOnsets(algorithm,projectionParam.overlapPercent,...
114     projectionParam.frameSize,length(inputSignal),coefficients.event,...
115     analysisData.sumEventsCoeff,threshold,noteDistance);
116 figure;plot(detectedVector);
117
118
119
120
121
122 end
123 end
124 end
125
126 % figure;
127 % surf(coeffEvol);
```

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %                              User Interface
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %%  Action:
5  %   this function holds all interaction with the user. It helps him to
6  %   select the input (and generate it if needed). Select which detection
7  %   algorithm to run and asks for the needed parameters.
8  %   It asks the user also for the desired output.
9  %
10 %%  Input:
11 %
12 %%  Output:
13 %   testName: to store the test name
14 %
15 %   inputSignal: is the signal under test (artificial,music,vibration)
16 %
17 %   onsetPosition: this is a vector containing the guess for the onset
18 %   positions entered by the user
19 %
20 %   startMusic: the user choose a point in the input signal from which
21 %   the processing starts.
22 %
23 %   projectionParam: contains all the needed parameters for the
```

```
%   projection
%       #tonalAtomsCount: the number of tonal (sines/cosines) atoms used
%                         in the projection procedure
%       #frameSize: the basis for the projection algorithm
%       #overlapPercent: to determine the algorithm step
%
%   algoParam: a structure containing the lamdas,
%       #K_max: max iterations for the lasso algorithm
%       #optType: optimization algorithm Lasso,cvx or Yall1
%       #lamdas intervals for lamda,lamdaT,lamdaE
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% [testName,inputSignal,projectionParam,algoParam] = UserInterFace()
function [testName,inputSignal,onsetPosition,startMusic,...
    projectionParam,algoParam] =UserInterface()

%% Set the default configurations
inputSignal=0;
onsetPosition=[0 0 0];
startMusic=0;

projectionParam.tonalAtomsCount=1000;
projectionParam.atomsKind=0;
projectionParam.pulseValue=1;
projectionParam.frameSize=1000;
projectionParam.overlapPercent=0;
projectionParam.normalizeIPframe=0;
projectionParam.normalizeOPframe=0;

algoParam.optType=1;
algoParam.k_max=50;

algoParam.lamda.general.lower=.5;
algoParam.lamda.general.upper=.5;
algoParam.lamda.general.precision=1;

algoParam.lamda.tonalAtoms.lower=0;
algoParam.lamda.tonalAtoms.upper=0;
algoParam.lamda.tonalAtoms.precision=1;

algoParam.lamda.eventAtoms.lower=0;
algoParam.lamda.eventAtoms.upper=0;
algoParam.lamda.eventAtoms.precision=1;

testName = 'yall amplitude';
load(['../tests/artificialSignals/' testName ...
    '/' testName '.mat'],'inputSignal');

%% User interactive testing

UserConfig = ...
    input('Enter "0" for running with default configurations: ');

if (UserConfig ≠ 0)
    %get the projection parameters
    projectionParam.tonalAtomsCount = ...
        input('Enter number of tonal atoms: ');
```

```
83      while(projectionParam.tonalAtomsCount≤0)
84          projectionParam.tonalAtomsCount =...
85              input('Please enter valid tonal atoms(>0): ');
86      end
87      projectionParam.atomsKind=input('Enter atoms kind:0)DCT 1)DFT: ');
88
89      projectionParam.frameSize=input('Enter frame size:  ');
90      while(projectionParam.frameSize≤0)
91          projectionParam.frameSize =...
92              input('Please enter valid frameSize greater than zero :');
93      end
94
95      projectionParam.pulseValue = input('Enter pulse value (>0): ');
96      projectionParam.overlapPercent = input('enter overlap ratio:  ');
97      projectionParam.normalizeIPframe =...
98          input('Normalize Input Frame(0/1):');
99      projectionParam.normalizeOPframe = ...
100         input('Normalize Output Frame(0/1):');
101
102     %select the input signal
103     selectInput=input...
104         (['Select i/p 1)create Model 2)saved Model 3)create Music'...
105         ' 4)load Music 5)Vibration : ']);
106
107     switch selectInput
108     case 1
109         testName = input('enter test name: ');
110         inputSignal = ModelInputSignal();
111         mkdir('../tests/artificialSignals/',testName);
112         save(['../tests/artificialSignals/' ...
113             testName '/' testName '.mat'],'inputSignal');
114
115     case 2
116         testName = input('enter test name: ');
117         load(['../tests/artificialSignals/' testName ...
118             '/' testName '.mat'],'inputSignal');
119
120     case 3
121         [inputFile,outSong,onsetPosition]=mixTones();
122         startMusic=input('enter the start of input:  ');
123         endMusic=input('enter the end of input:  ');
124
125         inputSignal=outSong(startMusic:endMusic,1);
126         testName=inputFile;
127         mkdir('../tests/artificialSignals/',testName);
128
129     case {4,5}
130         inputFile=input('Enter the name of the i/p file (use''''):');
131         load(['../musicIO/' inputFile '.mat'],'outSong',...
132             'onsetPosition');
133
134         startMusic=input('enter the start of input:  ');
135         endMusic=input('enter the end of input:  ');
136
137         inputSignal=outSong(startMusic:endMusic,1);
138         testName=inputFile;
139         mkdir('../tests/artificialSignals/',testName);
140     otherwise
141         disp('Unknown input.')
```

```
142     end
143
144     %get the algorithm parameters
145     algoParam.optType=input(['enter optimization type: 1)lasso_admm'...
146         '     2-3-4-5-6)CVX flavors   7)YALL1  :']);
147
148
149     algoParam.lamda.general.lower = input('enter lower bound lamda: ');
150     algoParam.lamda.general.upper = input('enter upper bound lamda: ');
151     algoParam.lamda.general.precision= input('enter lamda precision: ');
152
153     algoParam.k_max = ...
154             input('enter k_max (max number of loops for the algo): ');
155
156     if(algoParam.optType~=1)
157         algoParam.lamda.tonalAtoms.lower =...
158             input('enter lower bound lamdaSS:  ');
159         algoParam.lamda.tonalAtoms.upper =...
160             input('enter upper bound lamdaSS:  ');
161         algoParam.lamda.tonalAtoms.precision =...
162             input('enter lamdaSS precision: ');
163
164         algoParam.lamda.eventAtoms.lower =...
165             input('enter lower bound lamdaT:  ');
166         algoParam.lamda.eventAtoms.upper =...
167             input('enter upper bound lamdaT:  ');
168         algoParam.lamda.eventAtoms.precision=...
169             input('enter lamdaT precision: ');
170     end;
171 end
```

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %                          Run Algorithm on Input
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %%  Action:
5  %   divides the input into frames (respecting the overlap ratio) and
6  %   runs the chosen algorithm on each separately then combines the
7  %   output coefficients and return them.
8  %
9  %%  Input:
10 %   projectionParam: structure having projection parameters
11 %   described in user interface
12 %
13 %   algoParam: a structure containing the lamdas,
14 %       #K_max: max iterations for the lasso algorithm
15 %       #optType: optimization algorithm Lasso,cvx or Yall1
16 %
17 %   lamdaGeneral: lamda exactness for YALL and Sparsity for others
18 %   lamdaEvent: lamda for events sparsity in YALL
19 %   lamdaTonal: lamda for tonal sparsity in YALL
20 %
21 %   projectionMTX: this is the projection matrix (tonal + identity)
22 %
23 %   inputSignal: the input signal
24 %
25 %%  Output:
26 %   coefficients: the projection coefficients (Tonal and Events).
```

```
%
%   analysisData: Some exploited charachteristics from coefficients.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% function [coefficients,analysisData] = RunAlgorithm(projectionParam,
%    algoParam,lamdaGeneral,lamdaEvent,lamdaTonal,projectionMTX,
%    inputSignal)
function [coefficients,analysisData] = RunAlgorithm(projectionParam,...
    algoParam,lamdaGeneral,lamdaEvent,lamdaTonal,projectionMTX,...
    inputSignal)
% move global to local variables
tonalAtomsCount=projectionParam.tonalAtomsCount;
frameSize=projectionParam.frameSize;
overlapPercent=projectionParam.overlapPercent;

% do the calculation and divide in two vectors Tonal and Event
% coefficients
coefficients.tonal=[];
coefficients.event=[];

% here are the analysis data
analysisData.sumEventsCoeff=[];
analysisData.nomalisedInput=[];


% calculate the overlap in samples
overlap =round((1-overlapPercent/100)*frameSize);

% calculate the signal processing time
tic
% this is the lasso algorithm iteration counter
lasso_counter=0;
start=1;
while (start+frameSize-1)<=length(inputSignal)

    % normalising the input frame
    frame=inputSignal(start:start+frameSize -1);
    normFrameIP = frame;
    if(projectionParam.normalizeIPframe)

        max_abs_frame=max(abs(frame));
        if(max_abs_frame ~= 0)
            normaliser=1/max_abs_frame;
            normFrameIP = frame*normaliser;
        end
    end

    % calling the projection algorithm
    if(algoParam.optType==1)
        [u,p,d,lasso_counter] = lasso_admm(normFrameIP,projectionMTX,...
            tonalAtomsCount,lamdaGeneral,lasso_counter,algoParam.k_max);
    else
        u = ...
            CalculateCoeff(algoParam.optType,normFrameIP,projectionMTX,...
        tonalAtomsCount,lamdaGeneral, ...
            lamdaEvent,lamdaTonal,algoParam.k_max);
    end

```

```

    % normalising the output frame
     normFrameOP = u;
    if(projectionParam.normalizeIPframe)
        max_abs_frame=max(abs(u));
        if(max_abs_frame ~= 0)
            normaliser=1/max_abs_frame;
            normFrameOP = u*normaliser;
        end
    end


    % deriving analysis data
    coefficients.tonal=...
        [coefficients.tonal ;normFrameOP(1:tonalAtomsCount)];

    coefficients.event=[coefficients.event...
        ;normFrameOP(tonalAtomsCount+1:tonalAtomsCount+frameSize) ];

    analysisData.sumEventsCoeff=[analysisData.sumEventsCoeff;...
        sum(abs(normFrameOP(tonalAtomsCount+1:...
        tonalAtomsCount+frameSize)))];

    analysisData.nomalisedInput=...
        [analysisData.nomalisedInput;normFrameIP];

    % incrementing the iterator with the overlap value
    start = start + overlap;
end
%lasso_counter
%printing the total time of processing
toc
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%          Use optimization to calculate coefficients ...                      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Action:
%   This function is projecting one frame with the selected optimization
%   implementation choice : LASSO-ADMM, 4 CVX impl, YALL1.
%%  Input:
%
%   optType: this is the selected optimization implementation number.
%
%   Input: this is one input frame.
%
%   projectionMTX: this is the projection overcomplete dictionary.
%
%   atomsCount:this is the number of tonal atoms.
%
%   lamda: lamda exactness for YALL and Sparsity for others
%   lamdaI: lamda for events sparsity in YALL
%   lamdaDCT: lamda for tonal sparsity in YALL
%
%   iteration: this is the max number of algorithm iterations to find
%   the solution. (used with LASSO-ADMM and YALL1)
```

```
%
%%  Output:
%   coefficients: the projection coefficients (Tonal and Events) for one
%   frame.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% function [coefficients] = CalculateCoeff(optType,Input,projectionMTX,
%    atomsCount,lamda,lamdaI,lamdaDCT,iteration)
function [coefficients] = CalculateCoeff(optType,Input,projectionMTX,...
    atomsCount,lamda,lamdaI,lamdaDCT,iteration)

 frameSize=length(Input);


%%
if(optType==2)
    %%cvx version (the objectif function is defined using the IDCT )
    %%not neccesarily the exact solution .. best effort
    cvx_begin %quiet
        variable u(atomsCount+frameSize)
        minimize(lamda* norm(u,1)+...
            square_pos(norm(Input-projectionMTX*u,2)));

    cvx_end
end



    %% cvx version (the objectif function is defined using the IDCT )
    %%not neccesarily the exact solution .. best effort
    %%using different lamda for DCT and Impulses
if(optType==3)

            uDCTmtx=[eye(atomsCount) zeros(atomsCount,frameSize)];
            uImtx=[zeros(frameSize,atomsCount) eye(frameSize)];
            cvx_begin %quiet
                variable u(atomsCount+frameSize)

minimize(lamdaDCT * norm(uDCTmtx * u,1) + lamdaI * norm(uImtx * u,1)...
    + square_pos(norm(Input-projectionMTX*u,2)));

            cvx_end
end

%%
if(optType==4)
        %%cvx version (the objectif function is defined using the IDCT
        %%+subject to) therefore it is the exact solution

        cvx_begin %quiet
            variable u(atomsCount+frameSize)
            minimize(lamda* norm(u,1));

             subject to
            Input==projectionMTX*u;

        cvx_end
end
```

```
82
83  %%
84  if(optType==5)
85          %%cvx version (the objectif function is defined using the IDCT
86          %%+subject to) therefore it is the exact solution
87
88          uDCTmtx=[eye(atomsCount) zeros(atomsCount,frameSize)];
89          uImtx=[zeros(frameSize,atomsCount) eye(frameSize)];
90
91          cvx_begin %quiet
92              variable u(atomsCount+frameSize)
93              minimize(lamdaDCT * norm(uDCTmtx * u,1) + ...
94                  lamdaI * norm(uImtx * u,1));
95
96               subject to
97              Input==projectionMTX*u;
98
99          cvx_end
100
101 end
102
103 %%
104 if(optType==6)
105     %%cvx version (the objectif function is defined using the IDCT )
106     %%not neccesarily the exact solution .. best effort
107     cvx_begin %quiet
108         variable u(atomsCount+frameSize) complex
109         minimize(lamda* norm(u,1)+...
110             square_pos(norm(Input-projectionMTX*u,2)));
111
112     cvx_end
113 end
114
115 %% YALL1 implementation of the optimization algoriyhm
116 % the parameters are set then the yall algo is called
117 if(optType==7)
118     opts.weights=[lamdaDCT*ones(atomsCount,1);lamdaI*ones(frameSize,1)];
119     opts.tol = 5e-3;
120     opts.rho = 1/(2*lamda);
121     opts.print = 0;
122     opts.maxit = [iteration];
123     %opts.maxit = [100];
124     t0 = tic; [u,Out] = yall1(projectionMTX, Input, opts);
125     Iter = Out.iter
126     T = toc(t0)
127 end
128
129 % return the coefficients of one frame
130 coefficients=u;
```

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %                          Peak-Picking algorithm
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %%  Action:
5  %   This function contains 4 different detection algorithms implementing
6  %   The peak-picking step of the onset detection process.
7  %%  Input:
```

```
8  %   algorithm: choice of algorithm:
9  %    #1# compare the sumUI to a threshold
10 %    #2# compare the sumUI to a threshold considering notedistance
11 %    #3# compare the uI to a threshold and choosing the first in each
12 %        detected set.
13 %    #4# compare the uI to a threshold and choosing the max in each
14 %        detected set.
15 %
16 %   overlap: this defines the overlap percentage between successive
17 %   processed frames.
18 %
19 %   frameSize: number of samples in a frame.
20 %
21 %   InputSize: input length (samples count).
22 %
23 %   uI: events coefficients.
24 %
25 %   sumUI: the sum of events coefficients in each frame.
26 %
27 %   threshold: the comparaison threshold for the detection algorithms.
28 %
29 %   noteDistance: the minimum enforced distance between two onsets.
30 %
31 %%  Output:
32 %   detectedVector: vector containing the detected onsets.
33 %
34 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
35
36 %% function [detectedVector]= ...
       DetectOnsets(algorithm,overlap,frameSize,...
37 %    InputSize,uI,sumUI,threshold,noteDistance)
38 function [detectedVector]= DetectOnsets(algorithm,overlap,frameSize,...
39     InputSize,uI,sumUI,threshold,noteDistance)
40
41 % initializations
42 detectedVector=zeros(floor(InputSize/frameSize),1);
43 detectedVectorValues=zeros(floor(InputSize/frameSize),1);
44 %% #test
45 %length(detectedVector)
46
47 %% compare the sumUI to a threshold
48 if(algorithm==1)
49     detectedVector = (sumUI ≥ threshold);
50 end
51
52 %% compare the sumUI to a threshold considering note distance
53 % and make sure of the distance between 2 onsets
54 % The first detected onset is considered the real one and the others are
55 % neglected.
56
57 if(algorithm==2)
58     % update the note distance to include the overlap
59     noteDistance= 100*noteDistance/(100-overlap)-1;
60     detectedVector = (sumUI ≥ threshold);
61     for i=1:1:(length(detectedVector)-noteDistance)
62         if(detectedVector(i)==1)
63             detectedVector(i+1:i+noteDistance)=0;
64             i=i+noteDistance+1;
65         end
```

```
    end
end

%% compare the uI to a threshold and choosing the first in each
% noteDistance
if(algorithm==3)
    indices = find(abs(uI)≥threshold);
    indices=floor(indices*(100-overlap)/(frameSize*100))+1;
    detectedVector(indices)=1;
    for i=1:1:(length(detectedVector)-noteDistance)
        if(detectedVector(i)==1)
            detectedVector(i+1:i+noteDistance)=0;
            i=i+noteDistance+1;
        end
    end

end

%% compare the uI to a threshold and choosing the max in each
% noteDistance
if(algorithm==4)
    indices = find(abs(uI)≥threshold);

    uiGTthreshold=uI(indices);

    frameIndices=floor(indices*(100-overlap)/(frameSize*100))+1;

    indexAndValue= [frameIndices uiGTthreshold ];

    detectedVector(frameIndices)=1;

    for j=1:1:length(frameIndices)
        detectedVectorValues(frameIndices(j))=max(indexAndValue...
            ((find(indexAndValue(:,1)==frameIndices(j))),2));
    end

    i=1;
    maxIDXFlag=0;
    while (i≤(length(detectedVector)-noteDistance))
        if(detectedVector(i)==1)
            [val,maxIDX]=max(detectedVectorValues(i:i+noteDistance));

            if(maxIDXFlag ≠ 1)
                detectedVector(i:i+noteDistance)=0;
                detectedVector(maxIDX+i-1)=1;
                i=maxIDX+i-1;
                maxIDXFlag=1;
            else
                maxIDXFlag=0;
                i=i+1;
            end

        else
            i=i+1;
        end
    end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                            Plot Analysis Results
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Action:
%   used to plot the projection results. The user can compare the
%   coefficients of the tonal and event parts with the input signal
%   itself. The normalised overlapping frames are also plotted.
%   Some other analysis parameters are also shown (sum of coeff. , ...)
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function []=...
    PlotProjectionResults(testName,algoParam,lamdaGeneral,...
    lamdaEvent,lamdaTonal,projectionParam,inputSignal,onsetPosition,...
    coefficients,analysisData)

    % Change default axes fonts.
    set(0,'DefaultAxesFontName', 'Times New Roman')
    set(0,'DefaultAxesFontSize', 16)


    %plot parameters
    pad =0;
    margin =0.05;
    space =0.08;

    filename=[testName ...
        ' l' num2str(lamdaGeneral) ...
        ' lE' num2str( lamdaEvent) ...
        ' lT' num2str( lamdaTonal) ...
        ' k' num2str(algoParam.k_max) ...
        ' a' num2str(projectionParam.tonalAtomsCount) ...
        ' fs' num2str(projectionParam.frameSize) ...
        ' o' num2str(projectionParam.overlapPercent) '%'...
        ' opt' num2str(algoParam.optType) ...
        ' pV' num2str(projectionParam.pulseValue)...
        ' aK' num2str(projectionParam.atomsKind)];

    analysisFig = figure('name',filename);

    subaxis(5,1,1, 'Spacing', space, 'Padding', pad, 'Margin', margin);
    hold on
    plot(0:length(coefficients.tonal)-1,real(coefficients.tonal),'b');
    plot(0:length(coefficients.tonal)-1,imag(coefficients.tonal),'r');
    xLimL=0;
    xLimH=length(coefficients.tonal)-1;
    Ylimit=max(abs(coefficients.tonal));
    if(~Ylimit)
       yLimL=-1;
       yLimH=1;
    else
        yLimL=-Ylimit;
        yLimH=Ylimit;
    end
    axis([xLimL xLimH  yLimL yLimH]);
    grid;
    title('Tonal Coefficients','FontSize',18,'FontWeight','demi');
    hold off
```

```
59
60      %     subplot(5,1,2);
61  %     stem(0:length(coefficients.event)-1,abs(coefficients.event));
62  %     axis([0 length(coefficients.event)-1 min(coefficients.event) ...
63  %       max(coefficients.event)]);
64  %     grid;
65  %     title('events coefficients (absolute value)');
66
67      subaxis(5,1,2, 'Spacing',space, 'Padding', pad, 'Margin', margin);
68      hold on
69      plot(0:length(coefficients.event)-1,real(coefficients.event),'b');
70      plot(0:length(coefficients.event)-1,imag(coefficients.event),'r');
71      xLimL=0;
72      xLimH=length(coefficients.event)-1;
73      Ylimit=max(abs(coefficients.event));
74      if(~Ylimit)
75         yLimL=-1;
76         yLimH=1;
77      else
78          yLimL=-Ylimit;
79          yLimH=Ylimit;
80      end
81      axis([xLimL xLimH  yLimL yLimH]);
82      grid;
83      title('Events Coefficients','FontSize',18,'FontWeight','demi');
84      hold off
85
86      subaxis(5,1,3, 'Spacing', space, 'Padding', pad, 'Margin', margin);
87      hold on
88      plot(0:length(inputSignal)-1,inputSignal);
89
90      % To plot the onsets
91      %     plot(onsetPosition,zeros(length(onsetPosition),1),'o',...
92      %         'MarkerFaceColor', 'r', 'MarkerEdgeColor','r', ...
93      %         'MarkerSize', 6);
94
95      axis([0 length(inputSignal)-1 min(inputSignal) max(inputSignal)]);
96      grid;
97      title('Input Signal','FontSize',18,'FontWeight','demi');
98      hold off
99
100     subaxis(5,1,4, 'Spacing',space, 'Padding', pad, 'Margin', margin);
101     plot(0:length(analysisData.nomalisedInput)-1,...
102         analysisData.nomalisedInput);
103     axis([0 length(analysisData.nomalisedInput)-1 ...
104         min(analysisData.nomalisedInput) ...
105         max(analysisData.nomalisedInput)]);
106     grid;
107     title('Input frames to the algorithm (normalised and overlapped)'...
108         ,'FontSize',18,'FontWeight','demi');
109
110
111     subaxis(5,1,5, 'Spacing',space, 'Padding', pad, 'Margin', margin);
112     plot(0:length(analysisData.sumEventsCoeff)-1,...
113         abs(analysisData.sumEventsCoeff));
114     grid;
115     title('Sum of Events coefficients per frame','FontSize',18,...
116         'FontWeight','demi');
117
```

```
118
119
120 %     saveas(analysisFig,['../tests/artificialSignals/' testName '/' ...
121 %         filename '.fig']);
122
123
124     figureSize=get(gcf,'Position');
125     figureSize(3)=800;
126     figureSize(4)=620;
127     set(gcf,'Position',figureSize);
128
129
130 %     saveas(analysisFig,['../tests/figuresLatex/' testName '/' ...
131 %         filename '.eps']);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                          Testing Environment ...                            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Action:
%   this code is used to help the user to run different kind of tests
%   varying the parameters, the input type and selecting which output
%   is needed to be shown.
%   The list of different tests are:
%       1) Generate an artificial input to model the signal.
%       2) Mix notes from different instruments to generate melodies for
%          test.
%       3) testing the algorithm varying the parameters.
%       4) Plotting the detected events
%       5)Evaluate the detection algorithm.
%
%%  Input:
%
%%  Output:
%   coefficients: the projection coefficients (Tonal and Events).
%
%   analysisData: Some exploited charachteristics from coefficients.
%
%   inputSignal: this is the entire input signal.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%  [] = TestEnvironment()
function [coefficients,analysisData,inputSignal]=...
    TestEnvironment()

%% Get User Input
[testName,inputSignal,projectionParam,algoParam] = UserInterface();

%% Plot input signal (signal under test)
figure
plot(inputSignal);
disp('if this is the intended signal to test press any key to ...
    continue');
pause

%% preparing the overcomplete dictionary
% if matrix size exeeds the 1000, it is better to be saved and ...
    loaded to
% use and not generated each time.
% Which kind of dictionary atoms? selecting the atoms is different ...
    for each
switch projectionParam.atomsKind
    case 0
        DCT_matrix=dctmtx(projectionParam.frameSize);
        IDCT_matrix=DCT_matrix';
        projectionMTX= ...
            [IDCT_matrix(:,1:projectionParam.tonalAtomsCount)...
            projectionParam.pulseValue*eye(projectionParam.frameSize)];
    case 1
        proTonalAtomsDFT=projectionParam.tonalAtomsCount/2;
        DFT_matrix=dftmtx(projectionParam.frameSize);
        % normalised idft matrix
```

```
54          IDFT_matrix=conj(DFT_matrix)/sqrt(projectionParam.frameSize);
55          projectionMTX= [IDFT_matrix(:,1:proTonalAtomsDFT)...
56              IDFT_matrix(:,projectionParam.frameSize +1- ...
57              proTonalAtomsDFT:projectionParam.frameSize) ...
58              projectionParam.pulseValue*eye(projectionParam.frameSize)];
59      otherwise
60
61  end
62
63  % Check other transient representation (more pulses)
64  % f=projectionParam.frameSize;
65  % IDCT_matrix= [IDCT_matrix(1:projectionParam.frameSize,:) ...
66  %     eye(f)+[zeros(1,f); eye(f-1,f)]];
67
68  % to generate a 3D plot of the coefficients evolution each iteration
69  % coeffEvol=[];
70
71  % To test different combination of lambda's
72  % loop on lamda general range
73  for lamdaGeneral = ...
74          algoParam.lamda.general.lower:...
75          algoParam.lamda.general.precision:...
76          algoParam.lamda.general.upper
77  % loop on lamda events range
78  for lamdaEvent = ...
79          algoParam.lamda.eventAtoms.lower:...
80          algoParam.lamda.eventAtoms.precision:...
81          algoParam.lamda.eventAtoms.upper
82  % loop on lamda tonal range
83  for lamdaTonal = ...
84          algoParam.lamda.tonalAtoms.lower:...
85          algoParam.lamda.tonalAtoms.precision:...
86          algoParam.lamda.tonalAtoms.upper
87  %% Run the algorithm
88  [coefficients,analysisData] = ...
89      RunAlgorithm(projectionParam,algoParam,lamdaGeneral,lamdaEvent,...
90      lamdaTonal,projectionMTX,inputSignal);
91
92  % x=abs(real(coefficients.event))';
93  % x(x<0.025)=0;
94  % coeffEvol=[coeffEvol ;x];
95
96  %% Plot Analysis Results
97  PlotProjectionResults(testName,algoParam,lamdaGeneral,lamdaEvent,...
98      lamdaTonal,projectionParam,inputSignal,coefficients,analysisData);
99  end
100 end
101 end
102
103 % figure;
104 % surf(coeffEvol);
```

Bibliography

J.P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and Mark B. Sandler. A tutorial on onset detection in music signals. *Speech and Audio Processing, IEEE Transactions on*, 13(5): 1035–1047, 2005. ISSN 1063-6676. doi: 10.1109/TSA.2005.851998.

Paul Brossier, Juan Pablo Bello, and Mark D. Plumbley. Realtime temporal segmentation of note objects in music signals. In *In Proceedings of the International Computer Music Conference, ICMC 04*, 2004.

Roberto Casati and Achille Varzi. Events. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2010 edition, 2010.

Laurent Daudet. A review on techniques for the extraction of transients in musical signals. In *Computer Music Modeling and Retrieval*, volume 3902, pages 48 – 51, 2006. doi: 10.1007/11751069_20.

Mark A. Davenport. Approximation in lp norms.

C. Duxbury, M. E. Davies, and M. B. Sandler. Separation on transient information in musical audio using multiresolution analysis techniques. In *Proceedings COST Conference on Digital Audio Effects*, December 2001.

C. Duxbury, M. Sandler, and M. Davis. A Hybrid Approach to Musical Note Onset Detection. In *In Proc. Digital Audio Effects Workshop (DAFx*, 2002.

L. El Ghaoui. *Optimization Models and Applications*. EECS Department, UC Berkeley, 2012. "online book".

Michael M. Goodwin and Carlos Avendano. Frequency-domain algorithms for audio signal enhancement based on transient modification. *J. Audio Eng. Soc*, 54(9):827–840, 2006. URL `http://www.aes.org/e-lib/browse.cfm?elib=13904`.

Michael Grant and Stephen Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008. `http://stanford.edu/~boyd/graph_dcp.html`.

Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.0 beta. `http://cvxr.com/cvx`, sep 2012.

Paul Gu. Adsr envelope, 2013. URL `http://en.wikiaudio.org/ADSR_envelope`.

FLORENT JAILLET and BRUNO TORRESANI. Time-frequency jigsaw puzzle: Adaptive multiwindow and multilayered gabor expansions. *International Journal of Wavelets, Multiresolution and Information Processing*, 05(02):293–315, 2007. doi: 10.1142/S0219691307001768. URL http://www.worldscientific.com/doi/abs/10.1142/S0219691307001768.

Will Knight. Mit technology review about speech recognition, 2012. URL http://www.technologyreview.com/news/427793/where-speech-recognition-is-going/.

Paul Masri. *COMPUTER MODELLING OF SOUND FOR TRANSFORMATION AND SYNTHESIS OF MUSICAL SIGNALS*. PhD thesis, University of Bristol, Bristol, U.K., december 1996.

R. McAulay and T.F. Quatieri. Speech analysis/synthesis based on a sinusoidal representation. *Acoustics, Speech and Signal Processing, IEEE Transactions*, 34(4):744 – 754, august 1986. ISSN 0096-3518. doi: 10.1109/TASSP.1986.1164910.

Sanjit K. Mitra. *Digital Signal Processing, A Computer Based Approach*. McGraw-Hill, Santa Barbara, CA, USA, 2001. ISBN 0-07-232105-9.

Brian Moore. *An Introduction to the Psychology of Hearing*. Emerald Group Publishing Ltd, 2003. ISBN 978-0125056281.

M. Muller, D.P.W. Ellis, A. Klapuri, and G. Richard. Signal processing for music analysis. *IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING*, 5(6):1088–1110, 2011. ISSN 1932-4553. doi: 10.1109/JSTSP.2011.2112333.

M.D. Plumbley, T. Blumensath, L. Daudet, R. Gribonval, and M.E. Davies". Sparse representations in audio and music: from coding to source separation. *Proceedings of the ieee*, 98(6): 995–1005, 2010.

Jay B. Reimer. Method and apparatus for voice control of a computer, 1987. URL http://www.google.ch/patents?hl=en&lr=lang_en|lang_fr|lang_it&vid=USPAT4704696&id=1PItAAAAEBAJ&oi=fnd&dq=voice+control&printsec=abstract#v=onepage&q&f=false.

Aaron Shapiro. Are you and your company ready for the voice-recognition revolution?, 2013. URL http://www.fastcompany.com/3017302/work-smart/is-your-company-ready-for-the-voice-recognition-revolution.

Julius O. Smith. *Mathematics of the Discrete Fourier Transform (DFT)*. http://-ccrma.stanford.edu/~jos/mdft/, 2007. online book.

Toon van Waterschoot. Alternating direction method of multipliers (admm) algorithm for solving l1-regularized linear regression (lasso) problem, version 1.0, january 2013.

V. Verfaille, U. Zolzer, and D. Arfib. Adaptive digital audio effects (a-dafx): a new class of sound transformations. *Trans. Audio, Speech and Lang. Proc.*, 14(5):1817–1831, September 2006. ISSN 1558-7916. doi: 10.1109/TSA.2005.858531. URL http://dx.doi.org/10.1109/TSA.2005.858531.

T. Verma, S. Levine, and T. Meng. Transient modeling synthesis: a flexible analysis/synthesis tool for transient signals. In *International Computer Music Conference, Greece*, volume STAN-M-101, pages 48 – 51, 1997.

Wikipedia. Music — wikipedia, the free encyclopedia, 2013. URL http://en.wikipedia.org/w/index.php?title=Music&oldid=566438742. [Online; accessed 1-August-2013].

Y. Zhang, J. Yang, and W. Yin. Yall1 basic solver code, version 1.4, july 2011.