IFSIC
Université de Rennes 1
Campus de Beaulieu
35042 RENNES Cédex

CNMAT
1750, Arch Street
BERKELEY, CA 94709
USA

Musical Signal Parameter Estimation

Tristan Jehan
DIIC − TST
1997

Under the direction of:
Adrian Freed (CNMAT)
Jean−jacques Fuchs (IRISA)

to my parents...

Preface

Personnal Thought

Staying in California for six months was an incredible and deeply enriching experience. Leaving my country gave me the opportunity to discover new cultures and landscapes which had been my wish. I really enjoyed living in Berkeley and especially sharing an apartment with the perfect roommate. The entire San Francisco Bay Area is very beautiful, definitely more sunny than Brittany and remarkably rich in culture. I am glad I had the chance to work on my thesis in such an interesting part of the United States.

Where and Why?

As a child I began playing the piano and experimenting electronics. My education drew me logically more and more towards the interdisciplinary field of "*Computer Music*". I finally had the opportunity to combine my two main interests through my research work, the last stage of the *Diplôme d'Ingénieur en Informatique et Communication*, or DIIC. The *Center for New Music and Audio Technologies*[1] (CNMAT[2]) at the University of California, Berkeley (UCB[3]) provided the ideal environment for this project. I was very inspired beyond learned a lot from the many people I met with an outstanding knowledge of computers issues.

The Work

The schedule was not defined in advance. My work started by reading recent papers about the topic and learning to use tools that exist at CNMAT and are used every day: Matlab, CAST (see section 1.5), Melvyl and Gladys (Universities of California's library online catalogs[4]), etc. After spending a lot of time at the UCB libraries and downloading papers from library databases through the internet, I finally oriented my research towards the two main methods developed in this paper: a statistical approach for segmentation and a wavelet based approach for pitch detection. The rest of my time has been spent developing programs, testing them with many sounds and improving results. I also have attended various seminars and the summer workshop of CNMAT. I used high quality recording materials and made a web page.

Although this thesis is submitted at the *Institut de Formation Supérieure en Informatique et Communication* (IFSIC) at the *Université de Rennes 1*, France, it is written in English since all the research was done at an American University. I'd like to mention that many parts of this paper come from different publications, books or web pages. They are signaled by quotes and are in italic. However, they might have been a bit modified or summarized.

[1] *http://www.cnmat.berkeley.edu/*
[2] *pronounced "senn'mat"*
[3] *also simply called "Cal"*
[4] *http://www.lib.berkeley.edu/Catalogs/*

Aknowledgements

A lot of people have supported this project in many different ways. At this point, I would like to thank all of them for their contribution.

Adrian Freed for offering me the opportunity to realize this research project in computer music and for supervising me. He has helped immensily in clarifying many points and suggesting interesting and fruitful directions to research. He has also helped a lot in creating me a confortable work environment.

Matthew Wright for being available every time I needed assistance with programming or UNIX. His effectiveness in fixing problems was amazing to me. He's also good at choosing restaurants.

I especially thank Régine André-Obrecht and Michèle Basseville (from France) for their advice, for sending me source codes and for communicating with me through e-mail messages. Without them, the statistical part of this thesis wouldn't have been possible.

Thanks to Raphael A.Irizarry who helped me a lot to understand the auto-regressive algorithms and to Mickael Goodwin who spent time with me to understand the wavelets based algorithm. I really appreciated this assistance.

Cyril Drame and Andreas Lücke have also been very helpful in my work, along with Pierre Korzilius, made my internship at CNMAT and my life in Berkeley more fun. Thank you all. Special thanks to Cyril for communicating to me his thoughts.

This work would also not be possible without personal support from others. My deepest thanks go to my wonderful roommate Amy Mayeno who made my internship in United States become six months of vacation. She always supported me and worried about my feelings. I really loved our trips around California that made me relax a lot.

My french friends who e-mailed me often and replied to my questions: Olivier Alloyer, Vincent Deshayes, Mehdi Hannouz and Jad Kfoury. Without them, I would have been very home sick for France,

CNMAT people for their warm hospitality and their availability. Special thanks to Vijay Iyer who lent me his Fender Rhodes for these few months,

All my friends in United States for their frienship, and

Finally, I would like to thank those who have been my best teachers, the ones who inspired me to learn and to continue learning: my parents.

Abstract

This thesis is result from research on two essential elements of Analysis/Synthesis of musical signals: segmentation and pitch detection. In music, segmention means extracting information of the onset and offset of events in a sound signal. Pitch detecting (or tracking) means following the trajectory of the pitch of sounds in a continuous way.

For segmentation, two main ideas have been exploited: the first one, in the frequency domain, is based on the Energy and the FFT; the second one, in the time domain, is a statistical approach based on the comparison between two auto-regressive models of the signal. The study of pitch detection is closely related to the extraction of the fundamental frequency detectable in the spectral domain. The studied and improved method relies on a wavelet decomposition and on zero-crossing detection by a derivative filtering function.

The programs have been written in Matlab, except for the divergence algorithm that I obtained in C. Good results have been obtained for a large variety of sounds. Therefore, this thesis serves real-time applications being developed at CNMAT.

Ce rapport de stage résulte d'une recherche sur deux éléments essentiels de l'Analyse/Synthèse de signaux musicaux: la segmentation et la détection des hauteurs de notes. En musique, segmenter signifie extraire l'information de debut et de fin d'évènements dans un signal sonore. Détecter (ou poursuivre) une hauteur de note signifie suivre la trajectoire de hauteur de note du son de manière continue.

Pour la segmentation, deux idées principales ont été exploitées: la première, dans le domaine fréquentiel, est basée sur l'énergie et la FFT, la deuxième, dans le domaine temporel, est une approche statistique basée sur la comparaison de deux modèles auto-régressifs du signal. L'étude de la hauteur des notes est étroitement corrélée à l'extraction de la fréquence fondamentale observable dans le domaine spectral. La méthode étudiée et améliorée s'appuie sur une décomposition en ondelettes et sur la détection des passages par zéro a l'aide d'une fonction dérivative filtrante.

Les progammes ont tous été écris en Matlab, sauf l'algorithme de divergence qui m'a été fourni en C. Des résultats significatifs ont été obtenus pour une grande variété de sons. Aussi, ce travail est dédié aux applications temps-réel en cours de développement au CNMAT.

Contents

List of Figures

Chapter 1

Introduction

1.1 Presentation of CNMAT

"The Center for New Music and Audio Technologies[1] *'s mission is to promote, produce and present the creative interaction between music and technology. A satellite of the UC Berkeley Department of Music, CNMAT is an interdisciplinary research center, drawing participants from many university departments including physics, mathematics, electrical engineering, psychology, computer science, cognitive science and music.*

1.1.1 Activities

The center presents concerts that cross cultural and musical barriers and could just as easily feature traditional ethnic music as state of the art computer music. It also provides a forum for diverse lectures and demonstrations for students and the community. The research focuses on new performance technology and performance-related research issues and problems such as real-time synthesis and control. Some of the current research activities are focused on the study of the perceptual and cognitive aspects of music, the creation of new computer architectures for synthesis and audio signal processing, and the development of software for composition and live performance. The main research project (the one I worked on) called CAST will be developed in section 1.5.

1.1.2 Facility

The CNMAT facility includes research labs, seminar rooms, performance spaces, offices, and digital audio recording, analysis and synthesis facilities. Computing resources available to faculty, researchers, students, and staff include Macintosh and Silicon Graphics computers (Indy and O2). A wide variety of music and audio peripherals is also available including various digital signal processing boards (single and multi-processor), MIDI[2] *synthesizers, and assorted MIDI and non-MIDI controllers and instruments.*

1.1.3 Software

Due in part to close affiliations with Stanford University's CCRMA[3] *and France's IRCAM*[4]*, CNMAT also maintains an unusually broad array of software tools for various musical and research uses. For example, users have access to the standard software synthesis programs (such as*

[1] *founder: Richard Felciano*
[2] *Musical Instrument Digital Interface*
[3] *Center for Computer Research in Music and Acoustics*
[4] *Institut de Recherche et Coordination Acoustique/Musique*

cmusic, csound, and cmix), sound analysis software, DSP software (filter design programs, signal analysis, and real-time instrumentation and control), various programming environments (C, C++, Common Lisp, Smalltalk, Max[5], Mathematica, Matlab, etc). Real-time control of music synthesis, both MIDI and at the DSP level, is possible through the use of the Max programming environment. CNMAT is pre-eminent in Max development, tools, and production."[6]

1.1.4 Set

Principals and staff are David Wessel (Director), Adrian Freed (Research Director), Matthew Wright (Musical Applications Programmer), Edmund Campion (Composer in Residence) and Richard Andrews (Administrator). And about a twenty people (Graduate Researchers, Graduate Composers, Musicians, Undergraduate Students, Visiting Scholars and Collaborators...) take part in the development of CNMAT.

1.2 History of Music Synthesis

1.2.1 Origins

"The history of electronic music synthesis dates back at least to the late 1800's when Thaddeus Cahill introduced the Telharmonium, a 200-ton keyboard-controlled assemblage of pipes and wheels designed to generate musical sounds and distribute them over telephone lines. In the 1920's, the Russian physicist Leon Theremin presented a device that incorporated advances in electromagnetic technology into a musical instrument consisting of two antennae; the output depended on the proximity of the performer's hands to the antennae, one of which controlled pitch and the other loudness. Despite the commercial failure of these inventions, the two fundamental concepts have endured into present-day music technology: from the Thelharmonium, the capability of synthesizing recognizable sounds as well as novel sounds not present in the repertoire of classical instruments; from the Theremin, the use of an alternate control mechanism without keys, frets, or other traditional interface utilities.

1.2.2 Digital Development

In the modern age, due to the maturation of high-speed digital signal processing devices and algorithms, electronic musical instruments are "readily" available. The combination of mathematical modeling with real-time computation enables simulation of unique instruments, for instance a hybrid clarinet-trombone, a parametric interpolation between the two instruments. It is essentially the power of the modern computer that creates opportunities for advances in music technology. The computer provides a tool for analysing and understanding musical signals, an interface for alternate controllers and gestural input devices, and a platform for simulating new and possibly adaptative instruments - instruments that could either aid in the instruction of their own use or even conceivably learn how to be played by a particular performer. The composer's desire is to use the analysis[7] results to drive a musically meaningful resynthesis process[8]." [37]

---

[5] *music oriented graphical programming environment from IRCAM/Opcode*

[6] *CNMAT's web page*

[7] *Signal analysis is the process by which a signal is transformed from one representation into another. Our input signal is a sampled time-domain waveform.*

[8] *Reconstruction of sounds using analysis parameters*

1.3 Digital Musical Sound Synthesis Issues

A sound synthesis technique maps time-varying musical control information into sound. Each different synthesis method can be evaluated not only in terms of the class of sounds it is able to produce, but also in terms of the *musical control*[9] it affords to the musician. However, certain fundamental ideas for sound synthesis are shared by multiple techniques.

1.3.1 Computer Approach

Computers can store and replay prerecorded, digitized sound, so we can use computers as a passive recording medium like a tape recorder or compact disc. Computers also allow us to synthesize sounds that we could never hear in nature and don't know how to produce any other way, either by processing and modifying natural sound or by generating sound entirely from mathematical formulae. Computers also provide us with new ways to control sound, to shape rhythm, loudness, timbre[10], and other features as sound is created. An essential feature of a sound synthesis technique is the type of degree of control it affords of the sound produced. Finally, computers allow and force us to make a distinction between the mechanism for control of sound and the musical parameter being controlled.

1.3.2 Sound Control

A note-oriented model[11], that supports the concept of starting and stopping notes with a fixed pitch and overall loudness determined when the note begins, is the fundamental concept of MIDI. But music is poorly represented with this kind of musical control modes. Many other gestural transducers can also be mapped to contol sound synthesis: joystick, wheels, knobs, sliders, pressure- and position-sensitive ribbons, breath or wind controllers, etc... CNMAT for instance has taken the pen-based digitizing tablets (which are used mainly by computers artists) and mapped their multidimentional output (three dimensions of pen position, pressure, orientation, tilt and rotation) to control musical processes.

Using these mechanisms for control, multiple parameters are then controlable: timing of events, pitch, loudness, brightness, timbre, morph position[12], etc... A sound synthesis technique takes in time-varying these musical control information and produces sound.

1.4 Additive Synthesis

At CNMAT, for the musical control reasons described in the previous section, the analysis-synthesis system does not use a physical modeling approach[13]. Instead, it uses an arguably more general approach that is loosely termed *signal modeling*; here the mathematical properties of the input signal are explored, possibly without regard for the originating mechanism.

1.4.1 Signal Model

"A viable model for a musical signal is that it consists of a deterministic component and a stochastic component. The deterministic part corresponds to the pitched part of the sound. The stochastic part accounts for such intrinsically random musical characteristics such as breath noise

[9] *expressivity*
[10] *or tone color*
[11] *electronic keyboards for example*
[12] *position in the interpolation between 2 sounds*
[13] *the analysis is concerned with the physical properties of the input mechanism*

and bow noise. The deterministic part, then, is modeled as a sum of sinusoids of time-varying amplitudes, frequencies, and phases, refered to as partials. *Mathematically, this signal model can be expressed as a sum of the deterministic component $d(t)$ and the stochastic process $s(t)$:*

$$x(t) = d(t) + s(t) \tag{1.1}$$

$$d(t) = \sum_{l=1}^{Q} A_q(t)\cos(w_q(t)t + \phi_q(t)) \tag{1.2}$$

where Q is the number of partials, and where $A_q(t)$, $w_q(t)$ and $\phi_q(t)$ are the amplitude, frequency and phase of the q–th partial at time t. In the discrete-time version of this model, it is assumed that these functions vary at a rate substantially slower than the sampling rate. This is called a sinusoidal signal model.

For this model to be useful, these parameters must be accurately estimable by some analysis method. One such analysis is depicted in figure 1.1.

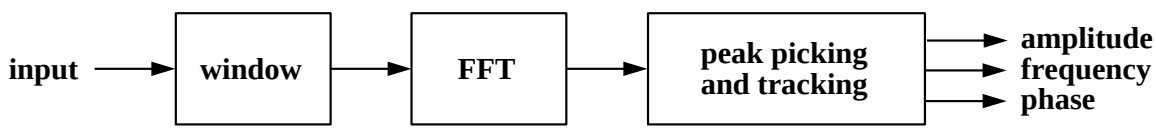

Figure 1.1: *Block diagram of the sinusoidal analysis system*

1.4.2 Reconstruction

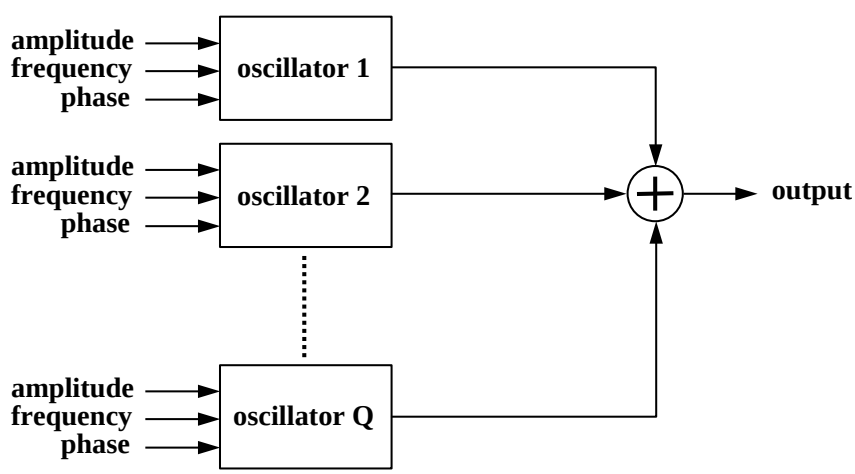

Figure 1.2: *Time-domain additive synthesis*

The sinusoidal analysis derives a set of analysis parameters that characterize the input sound. In the synthesis process, these parameters are used to reconstruct the original signal. In the event that modifications are made to the parameters, the synthesized signal is an altered version of the input. The reconstruction can be carried out either in the time-domain (see figure 1.2) or the

frequency-domain; in either case, however, the process is refered to as additive synthesis *because the partials are additively accumulated to create the deterministic component of the output signal.*" [37] Then, as shown in the signal model of equation (1.1), some noise is added to the synthesised sound: an appropriate amount and variety of noise is necessary for realistic music synthesis; without noise, the synthesized signal generally sounds unnatural and sometimes even unpleasant.

1.5 CAST Project

The *CNMAT Additive Synthesis Tools* project (CAST), has developed a srite of tools at CNMAT. It is polyphonic[14] and multitimbral[15]. Its architecture is based on the client/server principle. It is controlled by messages in the *OpenSoundControl* protocol[16] also under development at CNMAT. It is written in C and C++ for efficient, reliable, real-time and reactive control.

1.5.1 CAST Analyser

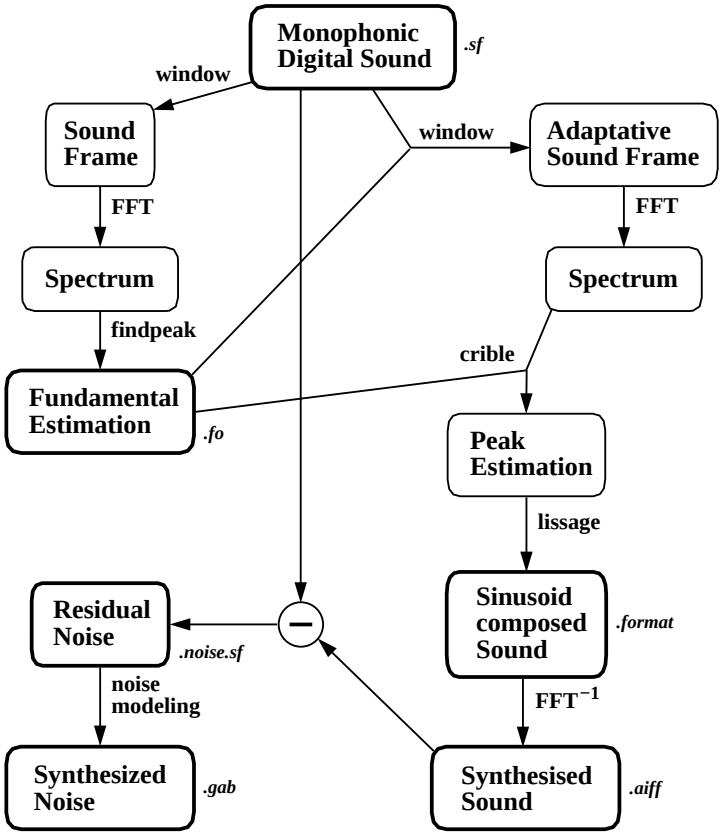

Figure 1.3: *Actual CAST harmonic-based analyser architecture*

[14] *it can play multi notes simultaneously*

[15] *it can have multi sounds (or multi voices) simultaneously*

[16] *OpenSoundControl (OSC) is an open, efficient, transport-independent, message-based protocol developed for communication among computers, sound synthetizers, and other multimedia devices. Also it is optimized for modern networking technology.*

- *findpeak* is a peak finder and estimator,
- *crible* is a harmonic matching sieve: picks largest peak within a frequency band around estimated harmonics,
- *lissage* is a harmonic track smoother,
- *.f0 .format .aiff .noise.sf .gab* are the file extensions.

Most of actual harmonic-based analysis software come originally from Xavier Rodet's and Philippe Depalle's analysis/synthesis group at IRCAM. My research work took place at this level of the CAST project: I looked for better ways for analysing music with real-time application objectives. The CAST analysis tool (analsf) is now outlined in Figure reffig:analsf. Another tool, developed at CCRMA by Xavier Serra, exists for non-harmonic sounds (*i.e.* drums) and is called SMS (Spectral Modeling Synthesis).

1.5.2 CAST Synthesizer

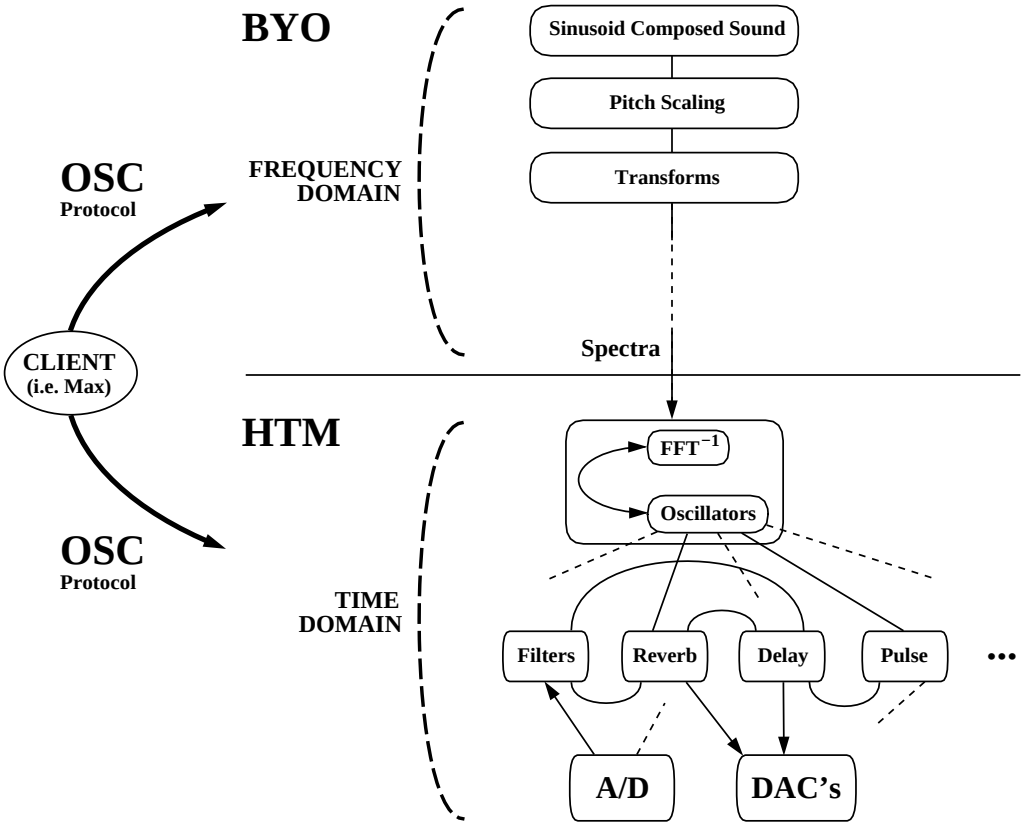

Figure 1.4: *Actual CAST synthesizer architecture*

The second component of CAST is the real-time synthesizer. It is a synthesis server that can be controlled in real-time, from the ethernet, by clients as Max (on Macintosh), Tcl/Tk, Matlab, Java, etc. After the synthesizer starts up, it listens for control messages from clients and interprets those messages to produce sound. It actually runs on SGI machines and can generate more than 1000 sinusoids partials at 44.1 KHz sampling rate on a O2. It offers real-time and expressive timbral control. A new feature currently in development, is a real-time control tool

of timbre by neural network; my project is related to this promising tool in a way that I am extracting parameters that are required as inputs of the neural network. The Figure 1.4 shows the CAST synthesiser architecture.

The synthesizer can be divided into three parts:

- The *client* controls the commands that are sent to the synthesizer through the OSC protocol,
- *BYO* (Bring Your Own) is a layer between high level programming tools (*i.e.* Max) and interpolated additive synthesis. It contains a set of functions that have been optimized for manipulations of spectral descriptions (partials and noise) of sound (transpose, vibrato, brightness, Filter/drop partials, inharmonicity, formant spectral envelopes, mixer, etc...)
- *HTM* (Hear The Music) is the low level of the synthesizer. It contains unit generators and converters. It schedules and rules messages to each generator.

Many elements of CAST have been ignored here. It doesn't seem very useful to detail it more since it is in a constant evolution. Also, many works actually under development have not been integrated to the CAST software yet: visualization tools or video features for example.

This thesis is loosely organized into two sections. The first part deals with "segmentation of events" while the second one is about "pitch detection". These two main areas of music analysis were studied independently but we'll see that the wavelet based pitch detection algorithm also offers a way for segmenting music.

Part I

SEGMENTATION OF EVENTS

Introduction

It has long been known that the onset of note, the *attack*, plays an important role in our perception of timbre. In traditional instruments, it is the phase during which resonances are building up, but before the steady state condition of standing waves has been established. Where the attack is short, there are many rapid changes, so that it can sound like a noise burst. For this reason, the attack transient is difficult to study. It is not surprising therefore, that attacks are not well understood and are not well represented within analysis-resynthesis models. However, this study goes farther than a single segmentation of notes: beyond the onset and the offset of musical notes, the developed algorithms here may detect many other kind of *event* in musical signals. An event is a part of the sound signal that extends over time and perhaps frequency. It has been defined in [47] as "an auditory phenomenon that exhibits constancy or at least continuity for its relatively short duration. It has an onset and an offset and represents the lowest time-extensive perceptual entity". It includes such things as single notes, change of timbre from a sound into another or even finer variations that a performer can control on his instrument. For instance, in singing voice, we may want to know when a change of vowel occurs, when the sound is voiced or unvoiced[17]; we want to detect unpitched sound events like percussive sounds; but the change can also simply be loudness or brightness. All these parameters belong to our signals and are musically useful.

At the beginning of the internship, I mainly worked in the frequency-domain, observing *Energy*, *HFC*, *Centroid*, etc. This is the topic of the next chapter. Then my approach of the segmentation issue has been to look for a time-domain method that could be fine enough to extract any event. I took inspiration from speech segmentation, already well advanced. After a few weeks of documentation research, I decided upon the statistical approaches described in a paperby Régine André Obrecht [12]. This is developed in chapter 3.

[17] *"vvvvv" and "zzzzz" are the voiced equivalent of unvoiced "fffff" and "sssss" sounds: the first ones have definite pitch while other ones don't.*

Chapter 2

A Frequency-domain Study

Most current analysis-resynthesis models represent sounds through a set of features, which are extracted from a time-frequency representation. So that each time-frame can present a good approximation to the instantaneous spectrum, it is necessary to analyse the waveform in short segments. This is achieved with a window function whose position is advanced by a fixed amount between frames. When the window encompasses a transient event, such as the percussive onset of a note, it contains information both before and after the event. These partially correlated spectra often become confused during analysis and cause audible "diffusion" upon resynthesis. The following algorithm [7], based on the energy observation, is a simple technique to avoid the problem. By synchronising the analysis window to transients events, the diffusion effect may be canceled.

2.1 The Detection Function

"Basically, the idea is that event locations are identified by observing short-term changes in the spectrum. The detection method was designed to recognise two signal properties associated with a sharp attack: the suddenness of the signal change and the increase in energy. A frequency domain method was chosen because of its ability to reveal both changes in overall energy and in energy concentration in the frequency-domain. The frequency location of energy is important because the sudden change to the signal will cause phase discontinuities; in the frequency spectrum this appears as high frequency energy. Naturally, the time resolution for detecting transient events must be smaller than that of the main analysis STFT, if any advantages is to be gained. This necessitates a reduction in frequency resolution, but fine frequency resolution is not an issue here; only the broad spectral envelope is required. A Hamming window function is used and the hop distance is set to half the window-length so that the maximum value that ensures each transient event will appear toward the center of the window in at least one frame.

2.1.1 An Energy and HFC based method

The energy function is calculated as the sum of the magnitude squared of each frequency bin[1] *(in the specified range):*

$$E = \sum_{k=2}^{\frac{N}{2}+1} \left(|X(k)|^2 \right) \tag{2.1}$$

[1] *result from the FFT*

$$where \begin{cases} E \text{ is the Energy function for the current frame,} \\ N \text{ is the FFT array length} \\ (\text{so } \frac{N}{2}+1 \text{ corresponds to the frequency } \frac{F_s}{2},\ F_s \text{ is the sample rate),} \\ X(k) \text{ is the kth bin of the FFT.} \end{cases}$$

The function to measure high frequency content was arbitrarily set to a weighted energy function, linearly increased toward the higher frequencies:

$$HFC = \sum_{k=2}^{\frac{N}{2}+1} \left(|X(k)|^2 * k \right) \tag{2.2}$$

where HFC is the High Frequency Content function for the current frame, other symbols as defined above.

In both cases, the lowest two bins are discarded, to avoid unwanted bias from DC or low frequency components.

The condition for detection combines the results from each pair of consecutive frames thus:

$$\frac{HFC_r}{HFC_{r-1}} * \frac{HFC_r}{E_r} > T_D \tag{2.3}$$

where subscript r denotes current frame (equals latter of two in detection function), subscript $r-1$ denotes the previous frame, T_D is the threshold, above which a hit is detected.

Note that HFC_{r-1} and E_r are constrained to have a minimum value of one, to avoid the potential "divide by zero" computation error.

The detection function is the product of the rise in high frequency energy between the two frames and the normalised high frequency content for the current frame. For attacks whose growth is slightly slower, but whose onset is nevertheless sudden, the detection function could be triggered on more than one frame. To avoid multiple detections, the algorithm is given with a parameter for the minimum closeness of two hits. In practice, setting this to two frames is adequate for the majority of sounds (i.e. only disallowing consecutive hits).

The synchronised analysis is not the interest of this study and will not be explained in this paper. The principle behind the method is that the spectra before and after an attack transient should be treated as different, the change happening instantaneously at the transient onset." [7]

2.1.2 Results

For the result outlined below I used a saxophone phrase from Steve Coleman. This two second phrase is composed of 10 notes more or less loud with more or less sharp attacks. The detection of note onsets seems closely tied to the sharpness of the attacks. In this example, a 400 sample long Hamming window with 50% overlapping has been used. Notice that the energy has been multiplied by 10 before plotting in order to have a better way of comparison. This method seems poorly performance.

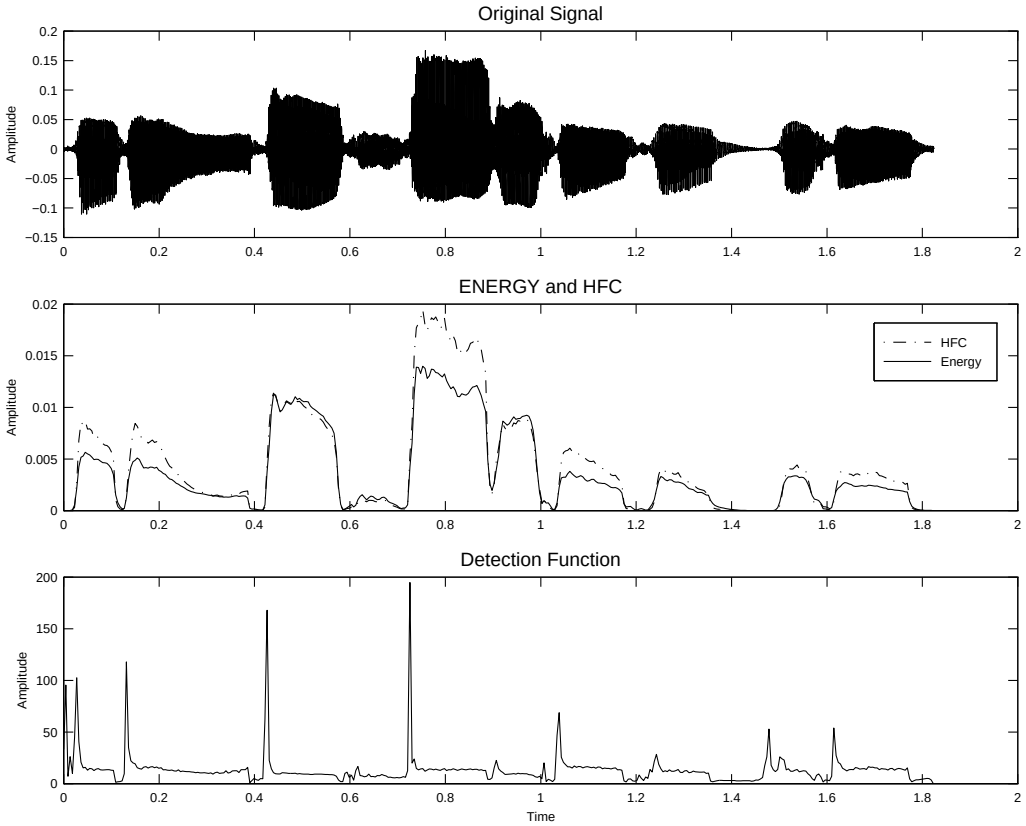

Figure 2.1: *Example of result with the detection function*

2.2 Filtering the HFC

As we learned in the previous section, the HFC is able to give us some information about the spectral envelope. The HFC result depends on the energy location in the spectrum. The HFC values become bigger if the energy is more important in the high part of the spectrum. It specially happens when sharp transients occur in the signal. We are able to detect these events in the HFC result if we can detect strong variations in the HFC time evolution. I drew my inspiration from edge detection in image processing. To detect strong variations in the grey scale between two pixels, it is comon to use a single highpass filter which the one dimensional pattern is simply $[-1\ 0\ 1]$. Applied on the HFC, it is able to detect onset (positive peaks) and offset (negative peaks) of notes. The example in the Figure 2.2 uses the same sound that the one used previously with the detection function (the original signal is actually showed in the Figure 2.1).

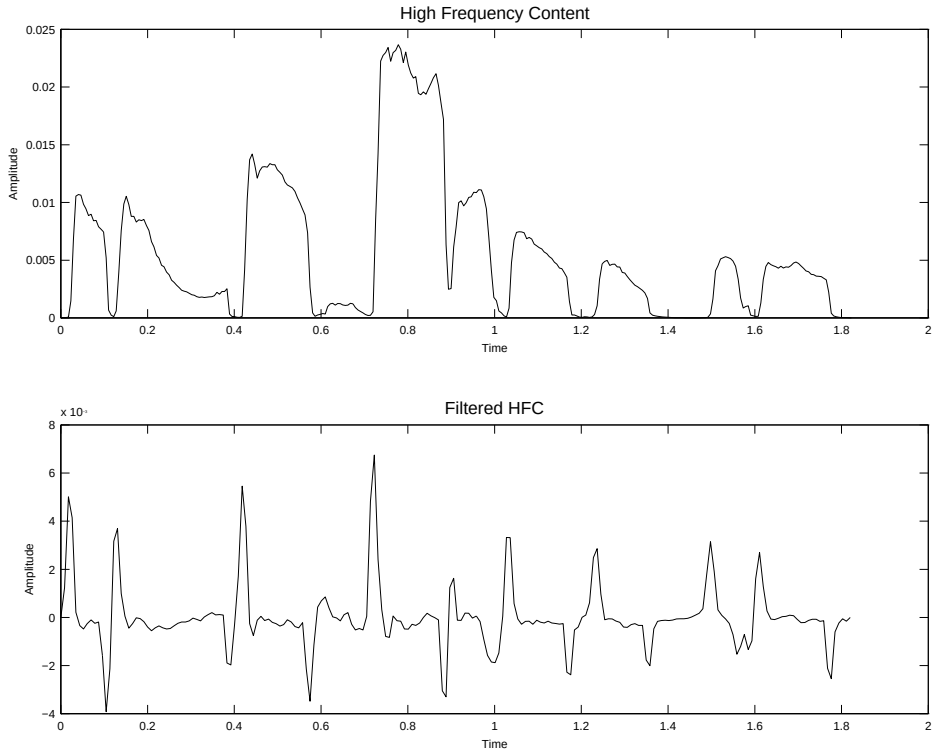

Figure 2.2: *Onset and offset of notes detection by filtered HFC*

2.3 Difference of spectra

Another way to detect variations in the spectrum is to compute a single difference between two succesive FFTs. As for the energy, the FFT results are linearly weighted in the high frequencies, in order to detect transients. The result is a distance measure. We use a sign detection of the energy gradient of slope to make the difference between onset (the gradient is positive) and offset (the gradient is negative).

$$Distance_j = sign(E_j - E_{j-1}) * \left(\sum_{k=2}^{\frac{N}{2}+1} [X_j(k) - X_{j-1}(k)]^2 * k \right)$$

where E_j is the energy and $X_j(k)$ the k-th bin of the FFT_j.

The example in the Figure 2.3 shows the single subtraction and the onset/offset distinction results. Nine over ten notes are clearly detectable here; the fourth one doesn't appear at all. I used a 800 sample long hamming window in order to have enough precision in the frequency domain. Using a wide window, we unfortunately lost accuracy in the time domain.

2.4 Brightness

Brightness is a psychoacoustic perceptual notion which is not mathematically precisely defined. It has been shown by David Wessel in [49] that it is very well correlated to the *centroid*. The

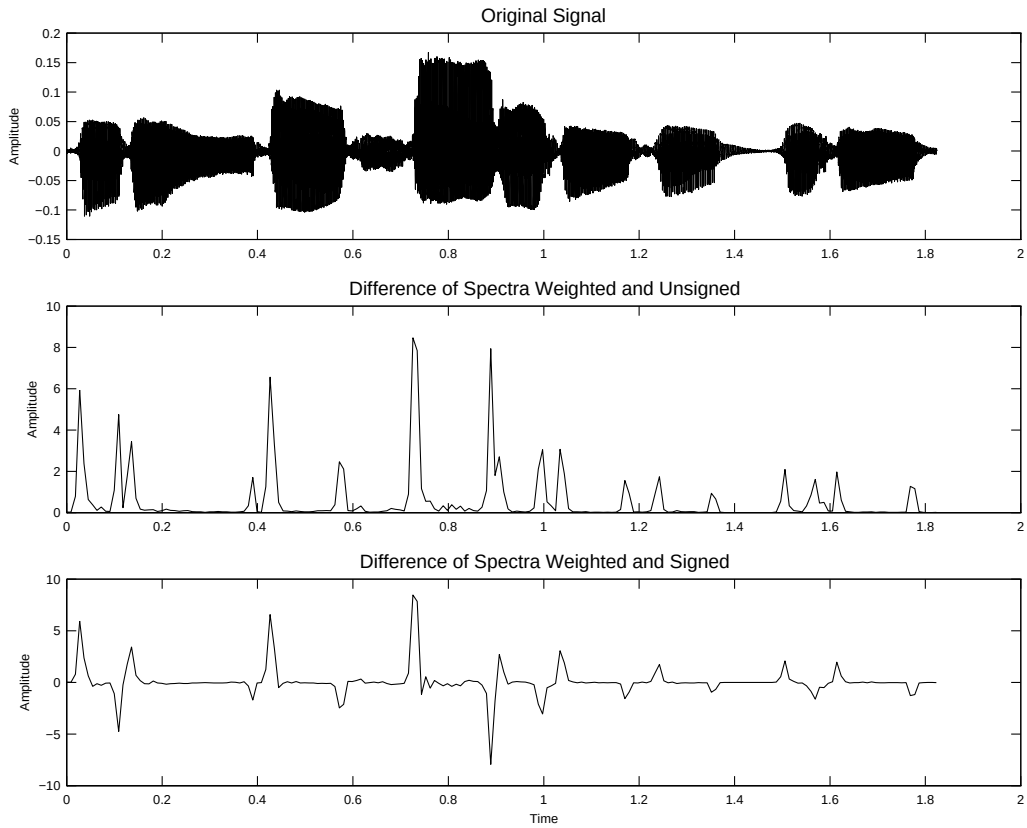

Figure 2.3: *Difference of two succesive FFTs*

centroid is equivalent to a mean of a transformed spectral energy distribution[2]. It is defined in [5] as follows:

$$centroid(t) = \frac{\sum_{b=1}^{m} f_b(t) * a_b(t)}{\sum_{b=1}^{m} a_b(t)} \tag{2.4}$$

where f is frequency (in Hz) and a is linear amplitude of frequency band b up to m bands which are computed by FFT.

An example of centroid calculation is given in Figure 2.4. A saxophone musical phrase has been resynthesized using neural network technique as described by Cyril Drame in [48]. The neural network allows the real-time control of additive synthesis in a very meaningful way. For instance, it allows the control of the timbral perceptual attribute: brightness. In this example, we changed it linearly from bright to dark[3].

The centroid gives us useful information for segmenting bright events in a signal. The second example, is a four-four kick drum (dark) pattern with syncopated high-hats[4] (bright). The centroid clearly enables extracting the high-hat events from the original signal.

[2] *a transformation on the acoustical spectrum that compensates properties of the auditory system*
[3] *term used for opposite of bright*
[4] *means four times "kick drum then high-hat"*

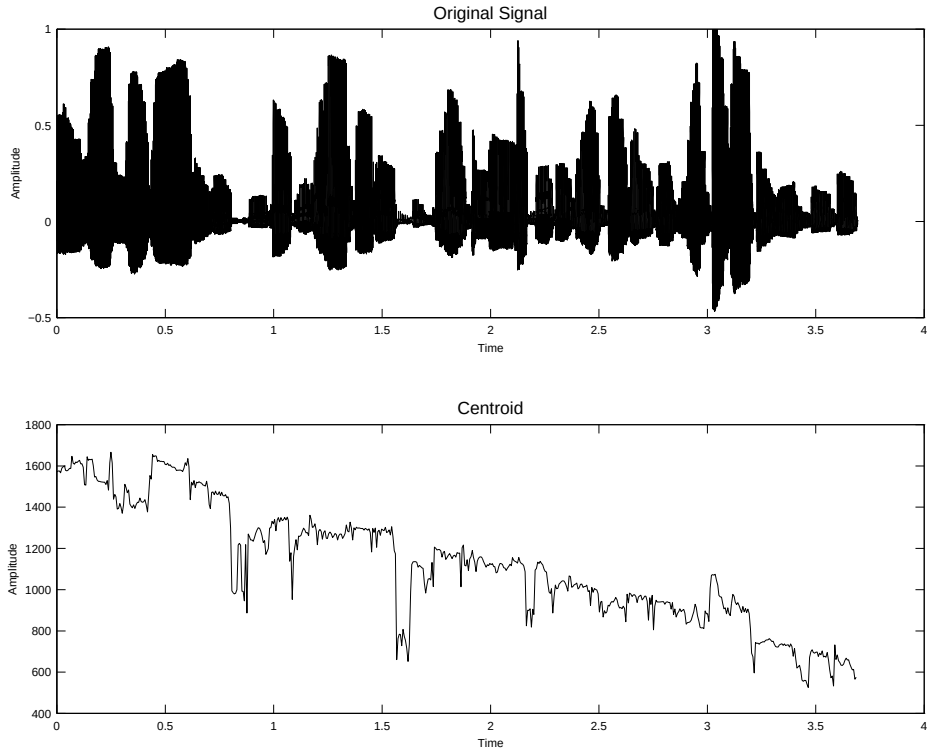

Figure 2.4: *Example of a brightness controlled sound*

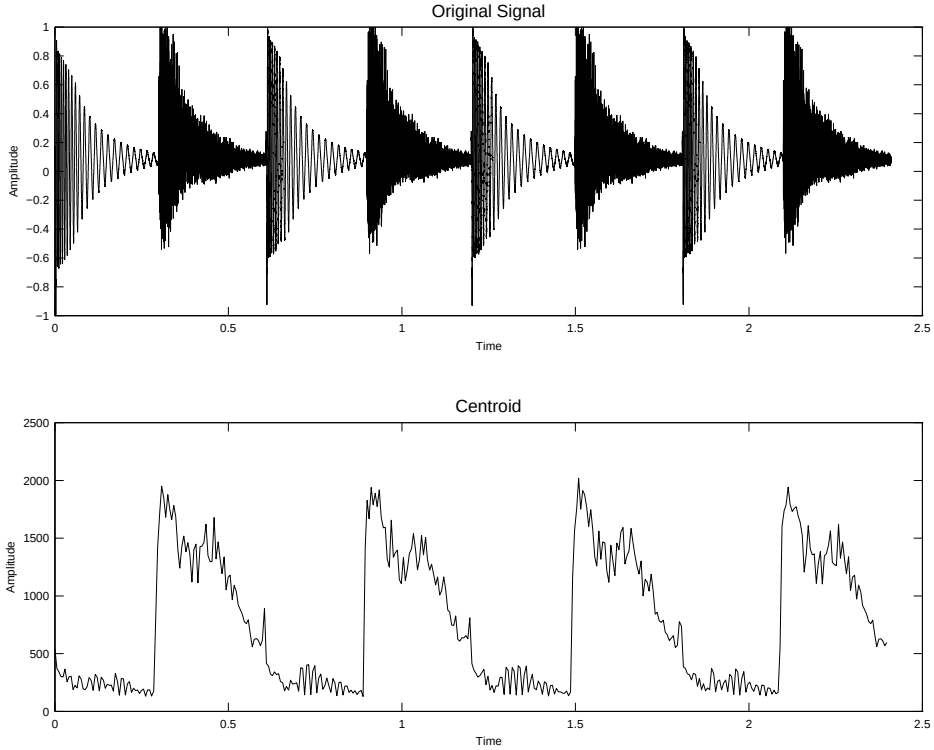

Figure 2.5: *High-hat extraction in drum sounds*

Chapter 3

A Statistical Time-domain Approach

"The main idea is to model the signal by an autoregressive (AR) statistical model and to use tests statistic to sequentially detect changes in the parameters of the model[1]*. The two segmentation algorithms presented here differ in the assumption of the excitation of the model and in the choice of the test statistic. Basically, a fixed and a growing window are used and then a suitable distance estimation compares the two spectra.*

3.1 The Brandt's Algorithm

The Generalized Likelihood Ratio (GLR) test of Brandt [1] is an efficient simplified realization of the GLR method of Willsky [2], which is the basic statistical approach for sequential signal segmentation [24].

3.1.1 Principles of the Approach

The signal is assumed to be described by a string of homogeneous units, each of which is characterized by a statistical model of the form:

$$y_n = \sum_{i=1}^{p} a_i y_{n-i} + e_n \tag{3.1}$$

where e_n is the excitation of the acoustic channel and (e_n) is an uncorrelated zero mean Gaussian sequence with $var(e_n) = {\sigma_n}^2$

The model is parametrized by the vector Θ defined by:

$$\begin{cases} \Theta^T = (\theta^T, \varphi^T) \\ \theta^T = (a_1, \cdots, a_p) \end{cases} \tag{3.2}$$

where φ is a parameter vector which determines the sequence σ_n.

The method consists in performing on line a detection of changes in the parameter Θ starting from the location of the previously detected.

[1] *Parametric AR methods are finer than Fourier*

$$\textit{So the algorithm} \begin{cases} 1. \textit{ Detect when a change occurs,} \\ 2. \textit{ Estimate the location of the change.} \end{cases}$$

3.1.2 The Algorithm description

$$\text{Let's take} \quad {\sigma_n}^2 = \sigma^2 \qquad (i.e.\ \varphi = \sigma) \tag{3.3}$$

which means that the noise variance is assumed to be a constant inside homogeneous units.

The test is then:
To monitor $(y_1, \cdots, y_n)$*, decide between the hypothesis:*

$$\begin{cases} \mathcal{H}_0 : \ \Theta = \Theta_0 & \textit{for } 1 \leq k \leq n \\ \mathcal{H}_1 : \ \exists\, r & \textit{such that} \quad \Theta = \Theta_1 \ \textit{for } 1 \leq k \leq r \\ & \textit{and} \quad \Theta = \Theta_2 \ \textit{for } r < k \leq n \end{cases}$$

We have to manage three windows as shown in Figure 3.1.

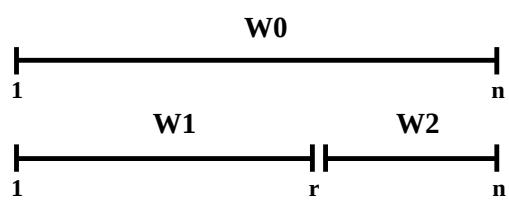

Figure 3.1: *Location of the three windows*

According to the GLR method, the decision is based on the likelihood ratio between these two hypothesis, where the time instant r *and the* Θ_i*'s are replaced by their maximum likelihood estimates, so that a change is detected if the distance:*

$$D_n = \max_r \max_{\Theta_1 \Theta_2} \max_{\Theta_0} \log \left(\frac{\mathsf{P}[y_1, \cdots, y_n \| \mathcal{H}_1]}{\mathsf{P}[y_1, \cdots, y_n \| \mathcal{H}_0]} \right) \geq \lambda \tag{3.4}$$

where λ *is a threshold.*

Then the estimate of the change instant $\hat{r}$ *is the argument of the maximum in the relation (3.4). The maximum likelihood estimates of the* Θ*'s are given by the formulae:*

$$\hat{\theta}(W_j) = \arg\min_{\theta} \sum_{k \in W_j} \left(Y_k - \sum_{i=1}^{p} a_i y_{k-i} \right)^2 \tag{3.5}$$

$$\hat{\sigma}^2(W_j) = \frac{1}{card(W_j)} \sum_{k \in W_j} \left(Y_k - \sum_{i=1}^{p} a_i y_{k-i} \right)^2 \tag{3.6}$$

where $j \in \{0, 1, 2\}$

W_j *denotes one of the tree windows depicted in Figure 3.1.*

This finally yields the following formula for D_n *:*

$$D_n = \max_r D_n(r) \tag{3.7}$$

$$D_n(r) = n \log \hat{\sigma}_0 - r \log \hat{\sigma}_1 - (n - r) \log \hat{\sigma}_2 \tag{3.8}$$

As you can see, the same identification of AR parameters is used for each of the three different windows. For each of them, we can write the signal as a vector $\mathcal{Y}^T = (y_1, \cdots, y_m)$.
The equation (3.1) gives us:

$$\begin{cases} y_t & = a_1 y_{t-1} + a_2 y_{t-2} + \cdots + a_p y_{t-p} + e_t \\ y_{t+1} & = a_1 y_t + a_2 y_{t-1} + \cdots + a_p y_{t-(p-1)} + e_{t+1} \\ & \vdots \\ y_{t+m} & = a_1 y_{t+(m-1)} + a_2 y_{t+(m-2)} + \cdots + a_p y_{t+(m-p)} + e_{t+m} \end{cases}$$

which can be written in the matrix form: $\mathcal{Y} = \mathcal{X}\theta + \mathcal{E}$
where $\mathcal{X}$ *is similar to Toeplitz matrix:*

$$\mathcal{X} = \begin{pmatrix} y_{t-1} & y_{t-2} & \cdots & y_{t-p} \\ y_t & y_{t-1} & \cdots & y_{t-(p-1)} \\ \vdots & \vdots & \ddots & \vdots \\ y_{t+(m-1)} & y_{t+(m-2)} & \cdots & y_{t+(m-p)} \end{pmatrix} \quad \textit{and} \quad \mathcal{E} = \begin{pmatrix} e_t \\ \vdots \\ e_{t+m} \end{pmatrix}$$

In the theorie, $\hat{\theta}$ *is simply given by the expression:*

$$\hat{\theta} = (\mathcal{X}^T \mathcal{X})^{-1} \mathcal{X}^T \mathcal{Y} \tag{3.9}$$

Unfortunately, this computation takes a long time because of the matrix inversion. The result also, may be inaccurate because of rounding errors by the machine. Thus, in the practical implementation, the different parameters Θ_0, Θ_1 *and* Θ_2 *are identified by a sample-by-sample growing memory covariance ladder method: the Durbin-Levinson algorithm, an exact least squares method.* [12]

3.1.3 The Durbin-Levinson's Algorithm

The linear prediction is a powerful analysis tool for the AR process that is extensively used in many areas. Efficient algorithms have been proposed to calculate the model parameters (i.e. the linear prediction coefficients of the AR process). However, in the real world, the AR process is often contaminated by noise. The estimation of model parameters by directly using Yule-Walker equations [22] on the noisy process may resulting errors. The Durbin-Levinson recursion presented allows us to cope with the problems of noise.

In the general case $(p = n)$*, the expression (3.1) can also be written:*

$$\hat{y}_{n+1} = a_{n1}y_n + \cdots + a_{nn}y_1, \qquad n \geq 1 \tag{3.10}$$

The mean squarre error of prediction will be denoted by v_n. Thus

$$v_n = E(y_{n+1} - \hat{y}_{n+1})^2, \qquad n \geq 1 \tag{3.11}$$

and clearly $v_0 = \gamma(0)$, where $\gamma()$ is the autocovariance function[2].

If $\{y_t\}$ is a zero mean stationary process with autocovariance function $\gamma()$ such that $\gamma(0) > 0$ and $\gamma(h) \to 0$ as $h \to \infty$, then the coefficient a_{nn} (where $n \in \{1, \cdots, m\}$) and mean squared errors v_n as defined by (3.11) satisfy $a_{11} = \gamma(1)/\gamma(0)$ and $v_0 = \gamma(0)$." [22]

$$a_{nn} = \left[\gamma(n) - \sum_{l=1}^{n-1} a_{n-1,l}\,\gamma(n-l)\right] {v_{n-1}}^{-1} \tag{3.12}$$

$$\begin{pmatrix} a_{n1} \\ \vdots \\ a_{n,n-1} \end{pmatrix} = \begin{pmatrix} a_{n-1,1} \\ \vdots \\ a_{n-1,n-1} \end{pmatrix} - a_{nn} \begin{pmatrix} a_{n-1,n-1} \\ \vdots \\ a_{n-1,1} \end{pmatrix} \tag{3.13}$$

$$v_n = v_{n-1}(1 - a_{nn}^2) \tag{3.14}$$

3.1.4 Results

The Matlab implementation of the brandt's method is not completely written yet. I still have to threshold properly the result in order to get the wanted segmentation information. In the current version of the program, I can adjust three parameters:

- The length of the sliding window n,
- the model order p,
- the number of computation per window s

The length of the sliding window must be chosen long enough to give a reliable parameter estimate, but not too long in order to detect small units. In the following example, the compromise led me to $n = 800$ corresponding to approximately 18 ms (which is too long for real-time applications). The order p can be incredibly small. Several trials were performed (until $p = 2$). For instance, I run my example with a model order $p = 6$. Finally, to make the computation time shorter, I chose $s = 12$, corresponding to 1200 points in the output vector with a 80000 sample long signal. It is almost 67 times faster and it gives me much enough resolution that I need for the graph.

We can observe a few peaks that correspond to onsets or offsets of notes. We could separate them looking at the slope of the energy at these moments (onset=positive slope and offset=negative slope). But no more work has been done with this method since I had a C version of an apparently better algorithm called the *divergence test*, now explained in the following section.

[2] *For a stationary process $\gamma(r,s) = \gamma(r-s,0)$ thus $\gamma(h) \equiv \gamma(h,0)$*

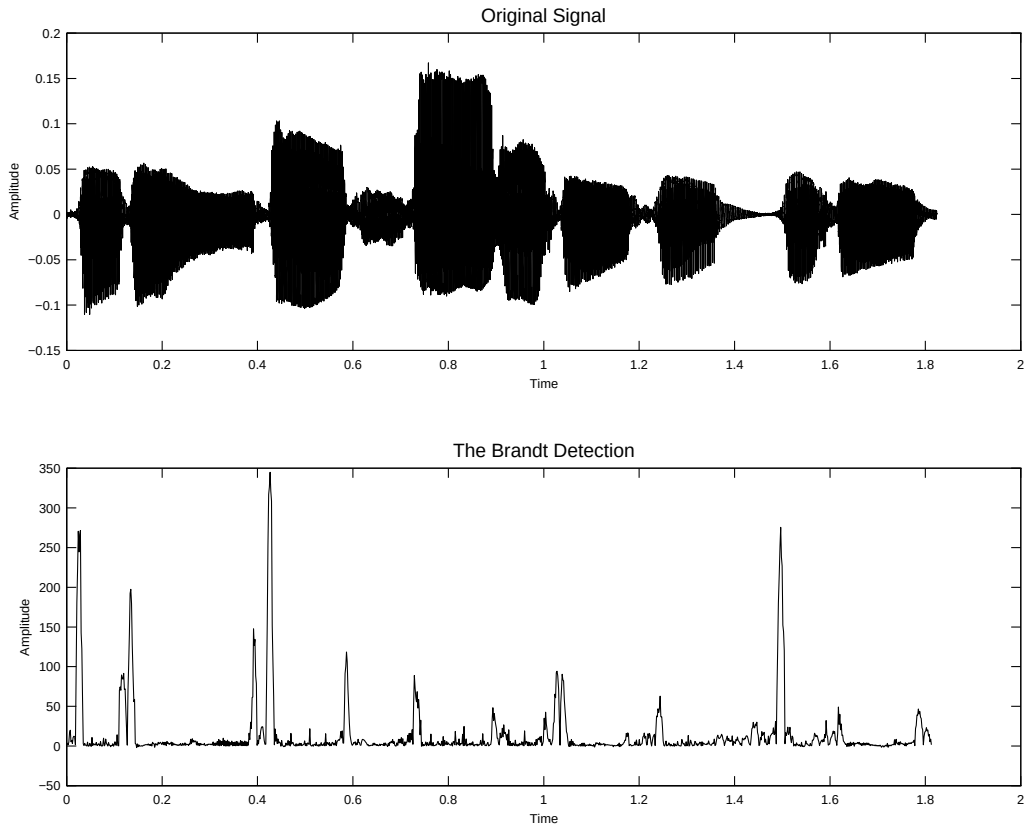

Figure 3.2: *Segmentation by the Brandt's method*

3.2 The Divergence Test

"*This procedure uses the J-divergence of conditional distributions as a measure of distance between models. The model set is the same as for Brandt's GLR method, so that the reader is again referred to the equations (3.1) to (3.3).*

3.2.1 The Test

The test is based on the monitoring of a suitable distance measure between the two models Θ_0 *and* Θ_1 *located as indicated in Figure 3.3. This distance measure is derived from the cross entropy between the conditional distributions of these two models.*

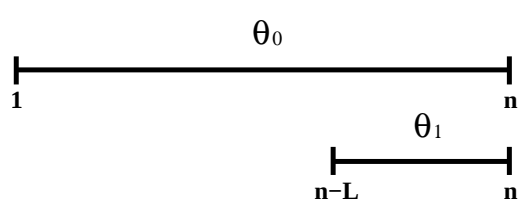

Figure 3.3: *Location of the two models*

Consider $\mathcal{Y}_m^T = (y_1, \cdots, y_m)$ *and denote by* $g_0(y_m \| \mathcal{Y}_{m-1})$ *and* $g_1(y_m \| \mathcal{Y}_{m-1})$ *the two conditional densities corresponding to the models of the Figure 3.3. Introduce the conditional cross entropy between the two models,* g_0 *and* g_1*:*

$$w_m = \int g_0(y\|\mathcal{Y}_{m-1}) \log \frac{g_1(y\|\mathcal{Y}_{m-1})}{g_0(y\|\mathcal{Y}_{m-1})}\, dy - \log \frac{g_1(y_m\|\mathcal{Y}_{m-1})}{g_0(y_m\|\mathcal{Y}_{m-1})} \tag{3.15}$$

which is in the Gaussian case [35] given by

$$w_m = \frac{1}{2}\left[2\frac{e_{0,m}e_{1,m}}{{\sigma_1}^2} - \left(1 + \frac{{\sigma_0}^2}{{\sigma_1}^2}\right)\frac{{e_{0,m}}^2}{{\sigma_0}^2} + \left(1 - \frac{{\sigma_0}^2}{{\sigma_1}^2}\right)\right] \tag{3.16}$$

$$\textit{where} \quad e_{q,m} = y_m - \sum_{i=1}^{p} a_{q,i} y_{q-i}, \qquad q = 0, 1$$

are the prediction errors corresponding to the 2 different models. Finally, we introduce the cumulative sum

$$W_n = \sum_{m=1}^{n} w_m \tag{3.17}$$

It can be shown that, under the hypothesis

$$\mathcal{H}_0 : \Theta = \Theta_0,$$

the statistics $(W_n)_{n\geq 1}$ have a zero conditional drift, while under the hypothesis

$$\mathcal{H}_1 : \Theta = \Theta_1,$$

the conditional drift is negative and equal to the opposite of the conditional Kullback's divergence

$$E_{H_1}(w_n\|\mathcal{Y}_{n-1}) = -\int g_0 \log \frac{g_0}{g_1}\, dy - \int g_1 \log \frac{g_1}{g_0}\, dy.$$

The behavior of the cumulative sum W_n is depicted in Figure 3.4(a).

3.2.2 The practical implementation

In practical implementations, the long term model parameter Θ_0 is identified using a sample-by-sample growing memory Burg algorithm, while the short term parameter Θ_1 is identified using the autocorrelation method. So we apply approached-least-squares parameter estimation methods instead of the exact one used for Brandt's algorithm. Michèle Basseville [11] and Régine André-Obrecht [34] showed that results are often the same and are much faster with approximated algorithms.

To obtain first estimates of the parameters Θ_0 and Θ_1, an initialization stage is performed for $n = 1, \cdots, L$ before the below-described detection procedure starts. A change detection occurs when the long term models disagree in the sense of the cumulative sum statistics (3.17). According to Figure 3.4(a), we have to detect the occurrence of a negative drift in W_n. This detection is performed as follows. Add to w_m a positive bias δ, so that the modified cumulative sum has a positive drift. The task is then to detect the occurrence of a negative drift as indicated

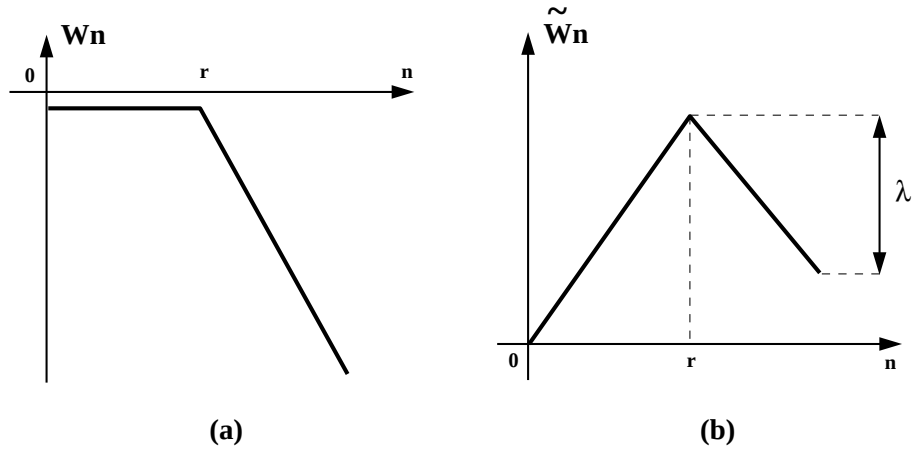

Figure 3.4: *Variations of the cumulative sum test: (a) The statistics (W_n), (b) The statistics coupled to Hinkley's stopping rule ($\hat{W}_n$)*

in Figure 3.4(b); this is done by comparing to a threshold the deviations of the current biased cumulative sum from its past maximum. This procedure, known as the Page-Hinkley rule is now given:

$$\tilde{W}_n = \sum_{m=1}^{n} (w_m + \delta) \tag{3.18}$$

$$\max_{1 \leq r \leq n} \tilde{W}_r - \tilde{W}_n > \lambda \tag{3.19}$$

where λ is a threshold, and δ is a bias chosen to let the cumulative sum behave as depicted in Figure 3.4(b). The estimated change time r is selected as the argument of the max in (3.19)." [12]

3.2.3 Parameters and Results

The C program offers various controlable parameters such as:

- The model order p,
- The voicing option control (0 for no special treatment, 1 for using voicing estimation for bias and threshold in the initialisation phase of each segment, 2 to also add the voicing estimation as a failure test),
- The window size (by default equivalent to 20 ms)

In a general way, it seems that the effect of these parameters depends on the structure of the sound and the note itself. In the tested examples, higher was the order, higher was the number of the voicing test, shorter was the sliding window and more there were segments. Best results were obtained using the voicing option #1. I had this remarkably good result with the saxophone lick (Figure 3.5) using a very short 56 sample long window and a second order model, which make the process really efficient in real-time. The segmentation shows the onset and offset of every notes but also the attack time and release of some of them. The difficulty now, would be to analyse the meaning of these segments in a musical and useful way.

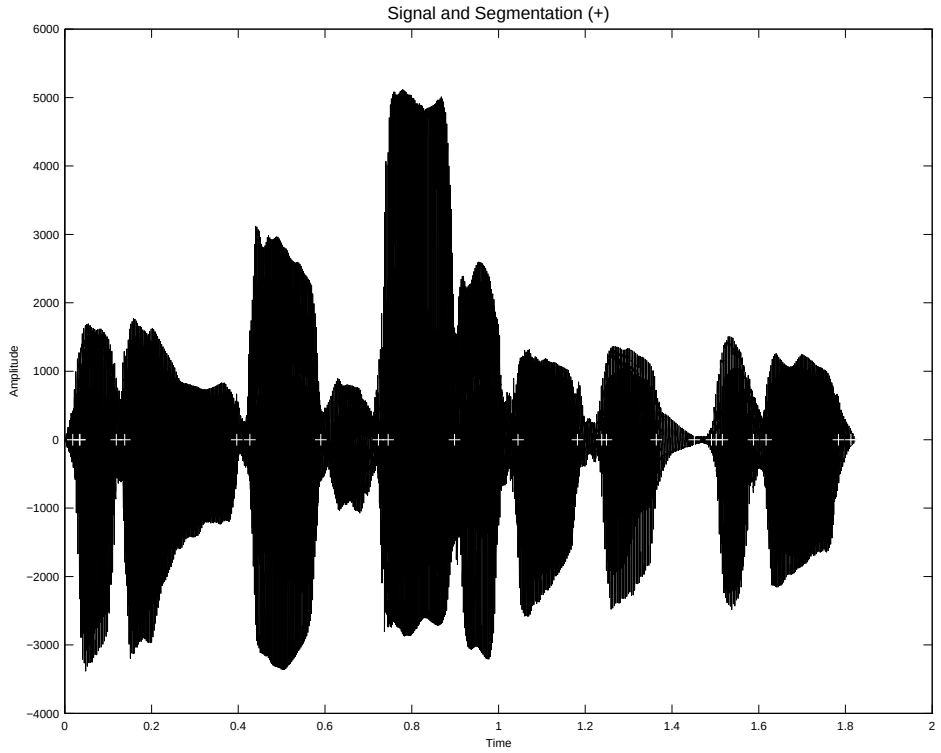

Figure 3.5: *Segmentation with the Divergence Test*

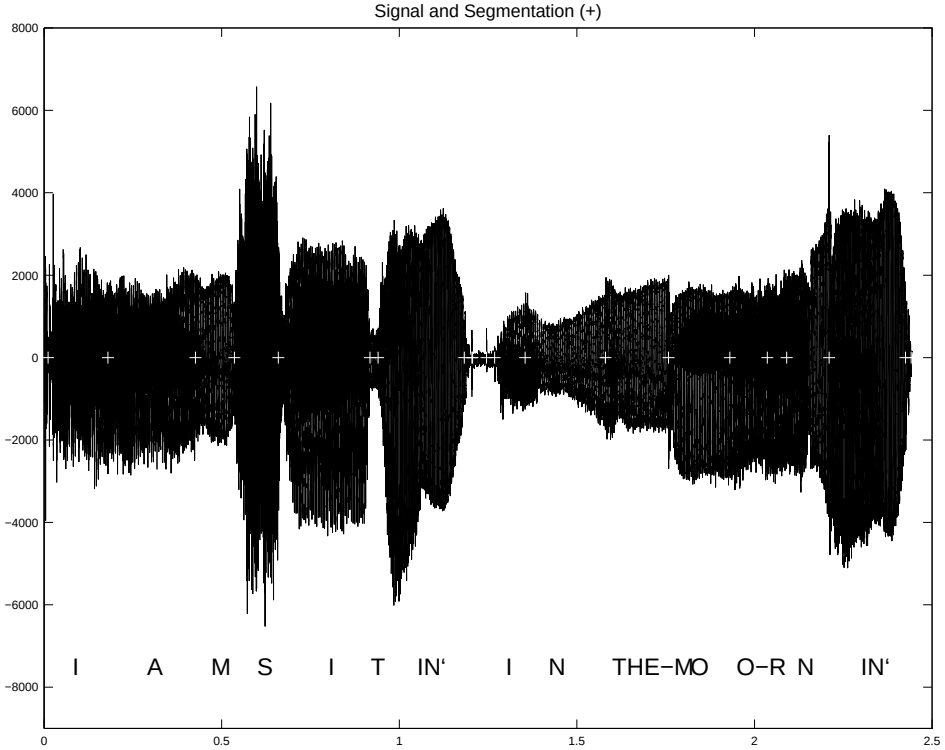

Figure 3.6: *Segmentation of the singing voice: "I am sitting in the morning"*

The second example (Figure 3.6) is the beginning of the Susan Vega's song "Tom's diner". She sings a capella the words "I am sitting, in the morning"... The segmentation, originally built for speech analysis, seems working well in the case of singing voice too. It gives us the information of onset of notes even if the energy doesn't change in the original signal (for example, "I" and "AM" are two seperate notes and the energy is constant). In this example, I used a 200 sample long sliding window (about 4.5 ms) and a eigth order model.

Part II

PITCH DETECTION

Introduction

The search for the perfect pitch tracker for musical applications has been underway for many years. This device can provide a close link between the musician and the computer. Whatever the application, the ideal pitch tracker described in [23] has no discernible delay, a stable and accurate output, a tolerance for noise and works for any monophonic instrument. Unfortunately, under these constraints, finding the pitch of musically interesting sounds is extremely difficult. Musical instruments, unlike speech, have an extremely large frequency range, so the pitch-tracking algorithm has to deal with a very large bandwidth. Musical sounds are harmonically rich and can contain inharmonic partials. For a minimum delay, the pitch has to be determined during the attack of a note, where the sound is noisiest and most harmonically complex. The pitch can also vary constantly without onset of notes. Many ambiguously pitched sounds exist, such as multiphonics, key clicks or unpitched sounds, which must be dealt with in a consistent manner. The wavelet based pitch tracker introduced in this thesis is able to estimate the pitch trajectory much more precisely and quickly than many classical techniques.

Chapter 4

Wavelets

Wavelets are a powerful mathematical tool for hierarchically decomposing functions. They allow a function to be described in terms of a coarse overall shape, plus details that range from broad to narrow: they offer an elegant technique for representing the levels of details present[1]. Signal characteristics can be efficiently located in the space and frequency domains. Thus, unlike the Short Time Fourier Transform (STFT), wavelets are adequate for the study of nonstationary and unpredictable signals with both low frequency components and sharp transitions. The wavelet transform is a multiresolutional, multi-scale analysis which has been shown to be very well suited for music processing because of its similarity to how the human ear processes sound. My approach has been to study algorithms for pitch detection of speech signals and one of them has been improved and applied to music signals.

4.1 Wavelet Transform

"*The Wavelet Transform (WT) basic cell is shown in Figure 4.1. This simple filter scheme performs a one-dimensional one-scale wavelet transform on any one-dimensional input signal. The $\mathcal{G}$ filter is a highpass wavelet filter and $\mathcal{H}$ is the complementary lowpass wavelet filter. The outputs are the lowpass residue for the $\mathcal{H}$ filter branch, represented by A_x called the scaling function and the highpass subband for the $\mathcal{G}$ branch, represented by D_x called the wavelet function.*

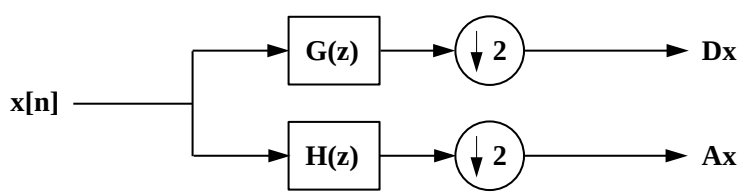

Figure 4.1: *Basic cell for the wavelet transform process*

The one-dimensional multi-scale scheme is achieved by iterating the basic cell on the lowpass residue of each of the previous cells, as shown in the Figure 4.2 for the 3-scale case.

The synthesis or reconstruction process is performed using the filter cell, shown in Figure 4.3, with the appropriate reconstruction filter set. The multi-scale reconstruction is performed with the same cell iterated so as to obtain the succesive lowpass residue for the following stage. [41]

[1] *No information is lost by this process*

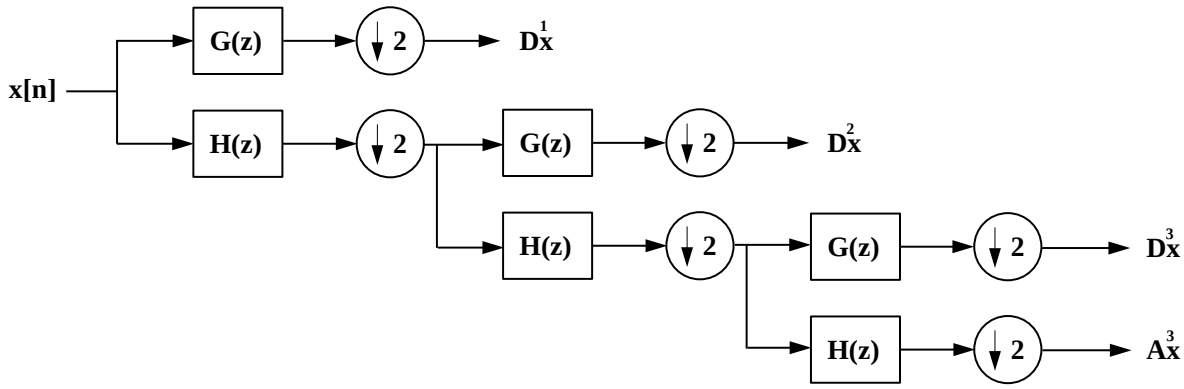

Figure 4.2: *Iterated filter scheme for the multi-scale transform*

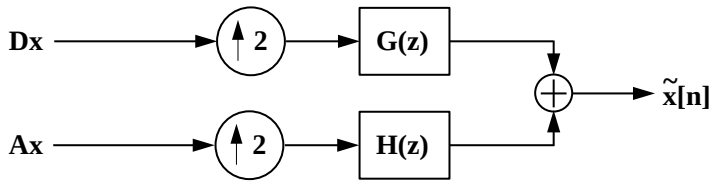

Figure 4.3: *Basic cell for one-dimensional inverse transform*

4.2 Multiresolution Analysis

In this section, A more general mathematical notation is used to widen the theory to any wavelet bases. We develop a mathematical framework known as multiresolution analysis for studying wavelets. We use the concept of a vector space from linear algebra. A one-sample sound is just a function that is constant over the entire interval [0, 1). We'll let V^0 be the vector space of all these functions. A two-sample sound has two constant pieces over the intervals [0, 1/2) and [1/2, 1). We'll call the space containing all these functions V^1. If we continue in this manner, the space V^j will include all piecewise-constant functions defined on the interval [0, 1) with constant pieces over each of 2^j equal subintervals. Thus, the starting point for multiresolution analysis is a nested set of vector spaces:

$$V^0 \subset V^1 \subset V^2 \subset \cdots$$

As j increases, the resolution of functions in V^j increases. The basis functions for the space V^j are the scaling functions. The next step in multiresolution analysis is to define wavelet spaces. For each j, we define W^j as the orthogonal complement of V^j in V^{j+1}. This means that W^j includes all functions in V^{j+1} that are orthogonal to all those in V^j undersome chosen inner product[2]. The functions we choose as a basis for W^j are the wavelets. [32] [33]

[2] $V^{j+1} = V^j \cup W^j$

4.3 A comparison between STFT and WT

The WT is of interest for the analysis of non-stationary signals, because it provides an alternative to the classical STFT or Gabor Transform (GT). The basic difference is as follows. In contrast to the STFT, which uses a single analysis window, the WT uses short windows at high frequencies and long windows at low frequencies. This is in the spirit of so-called "constant-Q" or constant relative bandwidth frequency analysis. While the STFT tiling is linear, the WT tiling is logarithmic[3] .

In order to get a better feeling about the difference between STFT and WT, let's look at a simple example. Suppose that we have a signal that contains two time pulses and two frequency pulses, such as

$$s(t) = \delta(t - t_1) + \delta(t - t_2) + e^{jw_1 t} + e^{jw_2 t} \tag{4.1}$$

Then its frequency presentation is

$$S(w) = e^{jw_1 t} + e^{jw_2 t} + 2\pi\delta(w - w_1) + 2\pi\delta(w - w_2)$$

Figure 4.4 compares STFT and WT for the signal (4.1). While time and frequency resolutions of STFT are uniform in the entire time-frequency domain, they vary in WT. At high frequencies, we have better time resolution and bad frequency resolution. At low frequencies, we have better frequency resolution and bad time resolution. However, the ratio of bandwidth and center frequency is constant. In terms of sound signals, it allows us to decompose the sound like the human hearing." [25]

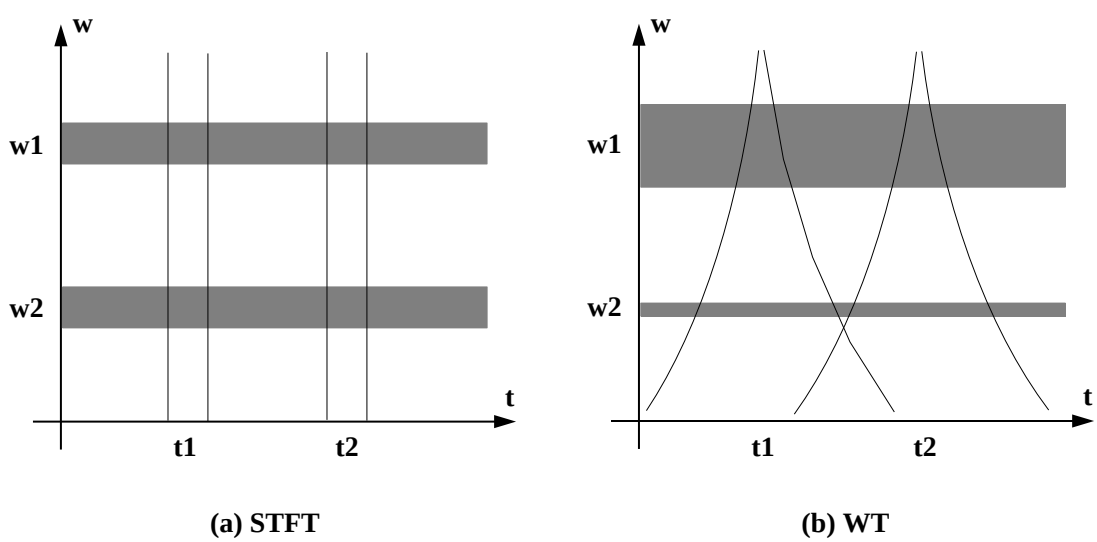

Figure 4.4: *Comparison of STFT and WT in the time-frequency domain*

[3] *like the human ear response*

4.4 A Pitch Determination Algorithm

As for music analysis, pitch determination is an important part of speech recognition and speech processing in general. Pitch detection algorithms can be classified in two separate categories, spectral-domain based and time-domain based period detection. Spectral pitch detectors such as Cepstrum, Maximum Likelihood, and Autocorrelation methods, estimate the pitch period of a signal directly using windowed segments of speech, applying a Fourier-type analysis to determine a pitch average. A time based pitch detector, however, estimates the pitch period by determining the Glottal Closure Instant[4] (GCI) and measuring the time period between each "event". Thus the speech signal is processed period-by-period. Many good results are obtained for speech with existing methods, but generally they are not very satifactory when applied to music. Often they are inadequate to music signals because of the large range of the fundamental frequency values or the variety of spectra encountered.

As I was interested in the non linearity of the WT (cf. 4.3), which seems to me a more natural way for windowing a large bandwidth signal, my research went to wavelet based speech oriented algorithms. We'll see however, that after improvements, good results have been obtained for many kind of sounds.

4.4.1 A simple principle

"In [30], a wavelet based pitch determination algorithm is proposed, based on Mallat's work on images [31]. Mallat showed that when analyzing images, the use of wavelet functions with derivative characteristics produces maxima in the wavelet transform across many coincident scales along sharp edges. In [30], Kadambe and Boudreaux-Bartels used the asumption that when a GCI occurs in a speech waveform, maxima also occur in the adjacent scales of the WT. The method proposed below improves reliability and further simplifies computation. In contrast with the method in [30], which chooses maxima if they occur in two adjacent wavelet coefficient scales, Wendt and Petropulu in [28] chose to utilize a single derivative filtering function defined to contain a specific bandwidth of voiced speech. This wavelet function when convolved with a speech signal will produce a filtered signal containing well defined local maxima where GCIs (or zero-crossing) occur in the speech (or music) signal. This method provides good results, a dramatic simplification in processing, utilizing only convolution and requiring only one set of coefficients to analyze and is robust to noise.

4.4.2 The proposed method

The choice of the mother wavelet, $\psi(t)$, is important because it defines the characteristics of the wavelet transform and coefficients. A voiced speech segment is often modeled as a filtered impulse train where the period between each impulse represents a pitch period. During each period of voiced speech, the glottis is excited and a GCI occurs. In the signal this phenomena corresponds with a zero-crossing in the waveform. In order to detect a zero-crossing or GCI, a derivative function can be used. If the signal is filtered by a derivative function (such as Daubechies filters), a maximum will occur at each zero-crossing. Thus the time period between each maximum represents the pitch period of the signal at that moment.

In order to construct a filtering function, the idea is to use a wavelet with the derivative properties, described by Mallat [31], that also combines the bandwidth properties of the wavelet

[4] *vocal string effect*

transform at different scales. Let $\psi(t)$ be the mother wavelet with derivative properties. The functions

$$\psi(t) = 2^{k/2}\psi(2^k t) \tag{4.2}$$
$$\varphi(t) = 2^{k/2}\varphi(2^k t) \tag{4.3}$$

represent both the wavelet and scaling functions respectively at each scale. $\varphi(t)$ is a lowpass function and is the conjugate mirror filter of $\psi(t)$, which is a highpass function. Since the range of voiced speech is between the 2 frequencies[5] f_1 and f_2 (Hz), the final filtering function constructed should have a similar bandwidth. Therefore, we determine the lowpass scaling function for the scale k_a, corresponding to approximately f_1, i.e.

$$2^{k_a} = \frac{F_s}{f_1} \tag{4.4}$$

where F_s is the sampling rate of the signal, and the highpass wavelet function for the scale k_b corresponding to approximately f_2, i.e.

$$2^{k_b} = \frac{F_s}{f_2} \tag{4.5}$$

The filtering function $\rho(t)$ is obtained as:

$$\rho(t) = \psi_{k_a}(t) * \varphi_{k_b}(t) \tag{4.6}$$

where $*$ is a linear convolution and k_a and k_b were given in 4.4 and 4.5." [28]

4.4.3 Maxima Detection

The filtered signal is a pseudo-sinusoid as shown in figure 4.5 and researched pitch at a given time t corresponds to the time period T separating both maxima in the function, before and after t. In order to detect these time locations properly, we approximate tops by parabolic functions: because of sampling restrictions, maxima can't be found by looking at simple highest samples (see figure 4.6). The parabola shall fit three points. Highest samples are detected by derivative filter[6] and signed-slope zero-crossing detection. Then with both samples arround it, we are able to compute the parabolic approximation $y = ax^2 + bx + c$ as follows:

$$\begin{cases} \text{A:} & y_1 = a{x_1}^2 + bx_1 + c \\ \text{B:} & y_2 = a{x_2}^2 + bx_2 + c \\ \text{C:} & y_3 = a{x_3}^2 + bx_3 + c \end{cases} \tag{4.7}$$

subtracting A with B and A with C:

[5] *30–500 Hz is commonly used in speech analyse*
[6] *three sample long* $[-1\ 0\ 1]$ *filter*

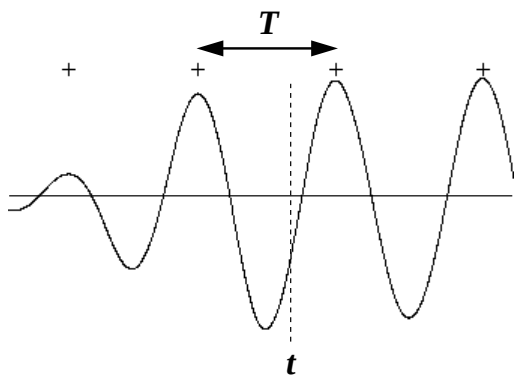

Figure 4.5: *Detail of the filtered signal with maxima detection*

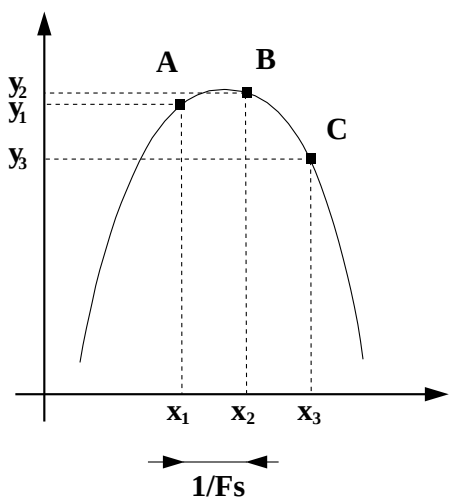

Figure 4.6: *Parabolic approximation*

$$\begin{cases} (y_1 - y_2) = a({x_1}^2 - {x_2}^2) + b(x_1 - x_2) \\ (y_1 - y_3) = a({x_1}^2 - {x_3}^2) + b(x_1 - x_3) \end{cases}$$

$$\begin{cases} \dfrac{y_1 - y_2}{x_1 - x_2} = a(x_1 + x_2) + b \\ \dfrac{y_1 - y_3}{x_1 - x_3} = a(x_1 + x_3) + b \end{cases}$$

subtracting these 2 relations:

$$\frac{y_1 - y_2}{x_1 - x_2} - \frac{y_1 - y_3}{x_1 - x_3} = a(x_2 - x_3)$$

and finally,

$$a = \frac{\dfrac{y_1 - y_2}{x_1 - x_2} - \dfrac{y_1 - y_3}{x_1 - x_3}}{x_2 - x_3}, \qquad b = \frac{y_1 - y_2}{x_1 - x_2} - a(x_1 + x_2) \qquad \text{and} \qquad c := 0 \tag{4.8}$$

then the maximum of this parabola is $max = \frac{-b}{2a}$

4.4.4 Error Detection

The filtered signal may not be perfectly smooth. For some reasons (noise, large bandwidth, etc), undesirable short ondulations or irregularities might appear in the signal as shown in Figure 4.7. Two different methods have been applied to eliminate unwanted maxima: a *median filter* and a case by case treatment. Both methods work on the time location of maxima previously detected. The program actually gives the number of corrections as information.

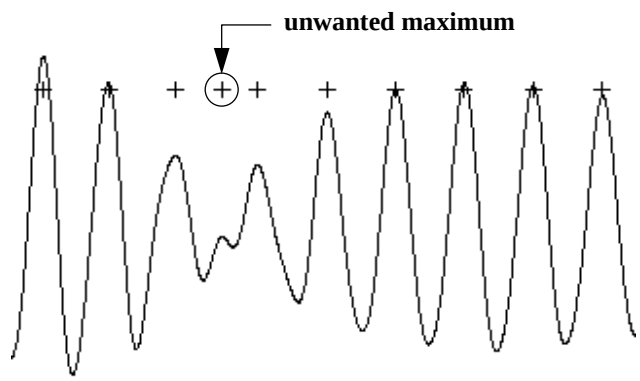

Figure 4.7: *Unwanted ondulation in the filtered signal*

Median Filter

This nonlinear filter is good for reducing "salt-and-pepper" noise[7]. Also, it tries to preserve transients while acting much like a lowpass filter. For each sample, it assigns the median value of samples within a prespecified window size to the sample. The main inconvenience of this method is the unnecessary modification of original samples. To minimize the distortion effects, I used a single three sample long window. Note that because of its interpolation effect, pitch result is never rejected but smoothed out by the filter.

Case by Case Treatment

Many different cases of irregularities have been noticed. Their elimination is based on the variation checking of two consecutive durations (each of them being the time between two successive maxima) and dynamic threshold. A maximum rate of variation of this duration is fixed in the program (for example 20%). Then, the processing eliminates samples that do not fit in the acceptable range, checking however that we do not have a real strong change of the pitch. The advantage of this method is that no modification of values occurs. The disadvantage is that the complete process requires knowledge of five successive maxima. Nevertheless, good results have been obtained with this method. The following figure is a good example of error correction. I selected a very short interval of time in order to show the filtering function. Note that in the second window, the two pitch trajectories have been separated by 10 Hertz before ploting (the scale is logarithmic).

[7] *speckly noise*

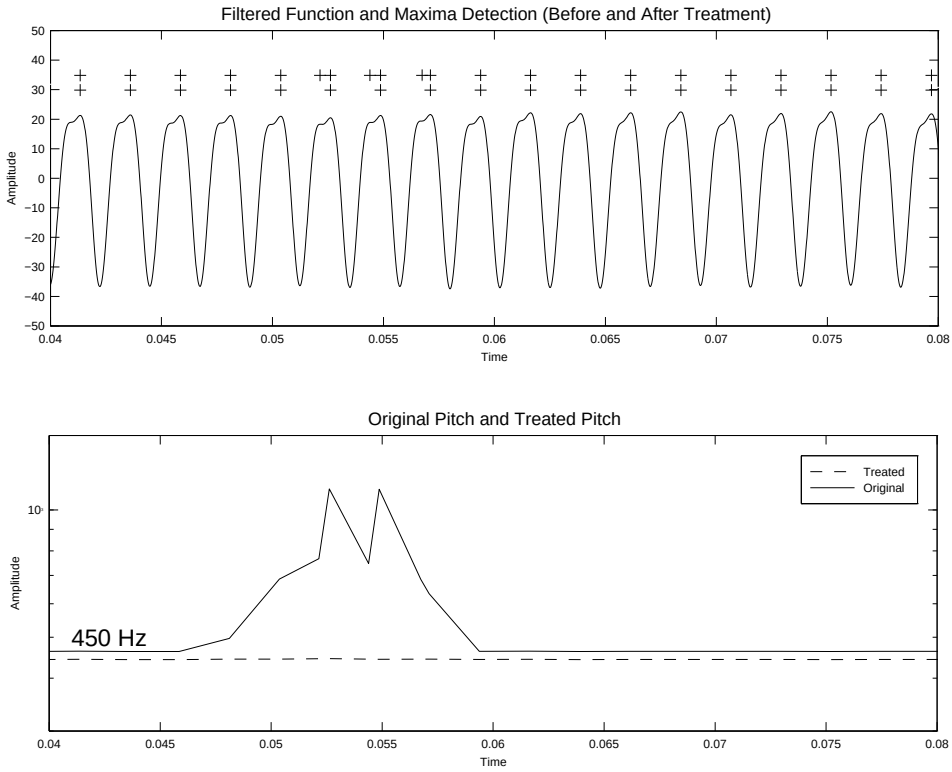

Figure 4.8: *Pitch error correction*

4.4.5 Unpitched and Error Rejection

This step tries to reject incorrect parts of the pitch result. There are three main reasons to reject the pitch:

- There is no Energy in the signal
- There is no Energy in the filtered signal
- The pitch estimation is wrong

Another dynamic threshold has been used to detect too strong variations in the pitch (we still work on the time maxima location but after error detection). A zero value is given to wrong pitch estimation. Also, the control of the energy by threshold on the signal and on the filtered signal[8] allowed us to reject incorrect pitch. An example is given in Figure 4.9. It is from a shakuachi phrase, an indonesian blowing instrument. Note that vibrato and fast changes of the pitch are clearly visible. A range from 600 to 4000 Hz has been used in this example. The threshold on the energy that determines the rejected pitch was 1% of the maximum of the energy.

[8] *there is no energy in the filtered signal if the sound is unpitched*

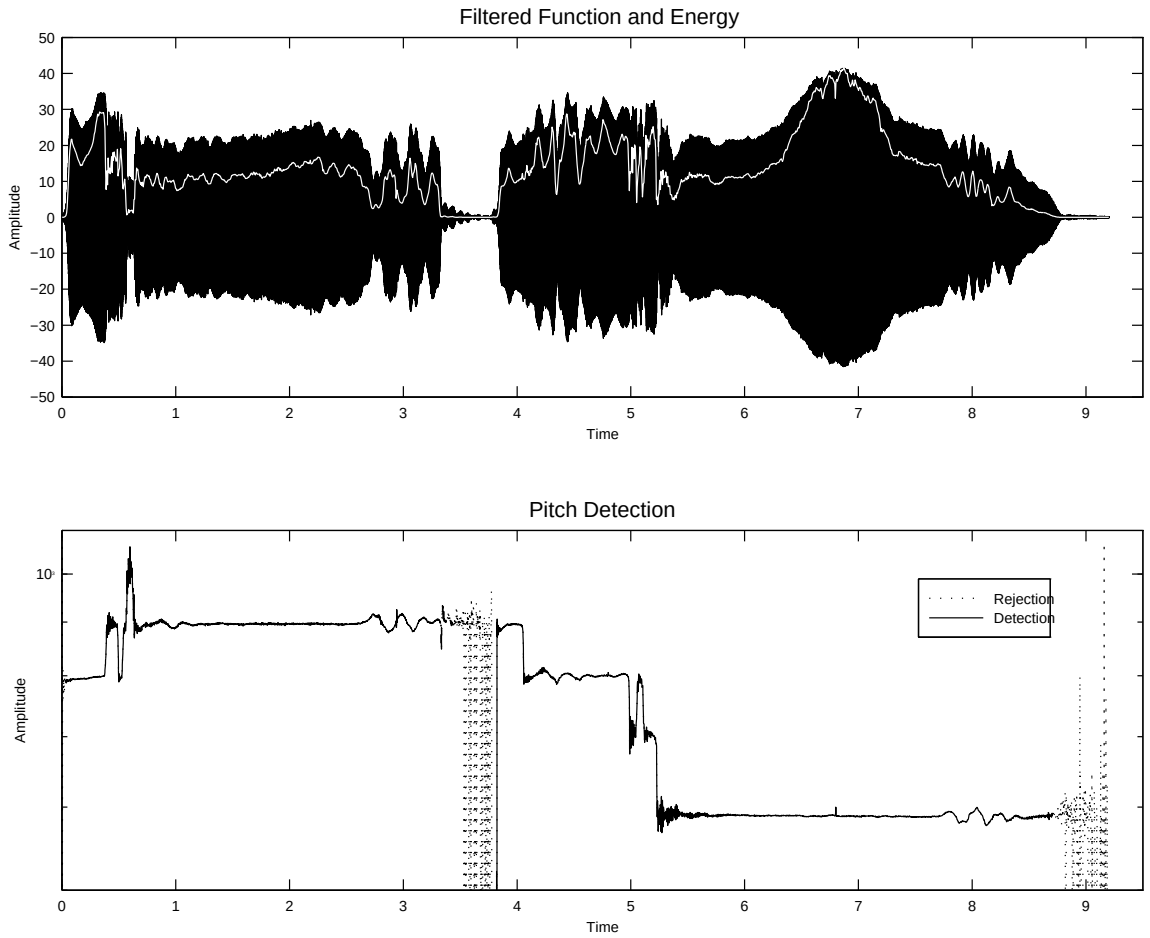

Figure 4.9: *Example of pitch detection after error detection and incorrect pitch rejection*

Conclusion

What I did

In this thesis I approached two important features of sound signal processing: the segmentation and the pitch tracking. For both of them, I drew my inspiration from speech analysis oriented algorithms. Also, real-time issues have been considered: all algorithms work with very local parts of the signal. The segmentation of sound events has been studied in the frequency-domain and the time-domain. Better results were obtained in the time-domain using a parametric autoregressive method: the divergence test. For instance, it has been shown that very short windows and order models can give good results. A wavelet approach were prefered for the pitch detection. Several improvements allowed the pitch tracking to work with many sounds and it is robust to noise. Since it is implemented in Matlab, no real-time application is predictable yet. These new methods in the music area may offer much interest in the CAST project.

What I learned

These results wouldn't have been possible whithout learned many features of computer music, including analysis, synthesis, real-time and control issues, psychoacoustics, etc. I also spent a lot of time understanding the statistical methods, the Fast Fourier Transform and the wavelet theory. I learned to work in a laboratory of research using many efficient tools and I learned to program in Matlab.

Part III

IMPLEMENTATION

Appendix A

Segmentation

All the programs are written in Matlab or C.

A.1 Frequency-Domain

The HFC and the Centroid functions are not given because they are very similar to the Energy. Here are the main lines to change:

```
HFC = sum(spec_abs(2:window_size).^2 .* weigh(2:window_size)) /window_size;
centroid = ( sum(freqs .* spec_abs1) / sum(spec_abs1) );
```

```
%%
%%                    -  ENERGY -
%%
%% This function computes the Energy of the input signal
%% and gives also the maximum value.
%%
%% Arguments input:
%% signal: original signal vector
%% Fs: the sample rate
%%
%% Arguments output:
%% time: time vector related to Energy
%% out: Energy vector
%% maxi: maximum in the Energy vector
%% stride: part of the window_size
%%
%% Written by Tristan JEHAN
%% Last modifications: 05/97
%%
%%


function [time,out,maxi,stride] = Energy(signal,Fs)

%% PARAMETERS
```

```
window_size = 512; % window length
jump        = 50; % per cent
stride     = window_size * jump/100; % part of window_size
window      = hamming(window_size); % hamming window vector

%% MEMORY ALLOCATION

spec_full = zeros(1,window_size);
spec_pos  = zeros(1,window_size);
spec_abs  = zeros(1,window_size);

%% INITIALISATION

frames  = floor(length(signal)/stride);
weigh   = [1:window_size];
scale_a = scale_analysis(window); % function from M.Goodwin
out     = [];

%% ANALYSIS LOOP

for i = 0:frames-2,

    in = signal(i*stride+1:i*stride+window_size);
    in = scale_a * window' .* in; % normalisation

    %% ENERGY CALCUL

    spec_full = fft(in,window_size); % X(k) vector (bins)
    spec_pos  = spec_full(1:window_size); % positive values vector
    spec_abs = abs(spec_pos); % magnitude vector

    Energy = sum(spec_abs(2:window_size).^2) /window_size;

    %% BUILDING THE VECTORS

    time = [0: stride*(1/Fs) :(frames-2)*stride*(1/Fs)];
    out  = [out Energy];

end

maxi = max(out);
```

The Detection Function section would be written like this:

```
%% BUILDING THE DETECTION FUNCTION

column      = size(vec_time,2);
vec_detect  = [vec_HFC(1)/vec_E(1)];
```

```
vec_squared = vec_HFC .* vec_HFC;
vec_multipl = vec_HFC(1:column-1) .* vec_E(2:column);
vec_detect  = [vec_detect vec_squared(2:column) ./ vec_multipl];
```

A.2 Time-Domain

A.2.1 Brandt Program

```
%%
%%          - AUTO REGRESSIVE SEGMENTATION USING BRANDT'S GLR METHOD -
%%
%% This function detects transients in musical phrases by Auto Regressive
%% process (Brandt's Generalized Likelihood Ratio method). This function calls
%% the sub-function lpc.m (Linear Predictive Coefficients) that computes the
%% estimation of coefficients 'A' for each window (3 per slide) by using the
%% Levinson-Durbin algorithm.
%%
%% Input Argument
%% in     : The complete original signal vector
%% p      : The Auto Regressive model order
%% W_size : Length of the sliding window
%% sample : Number of computations per window (<= W_size)
%%
%% Note that Fs must be defined as a global variable out of this function
%%
%% Output Argument
%% out : The 2 dimensional output signal
%%       1st dimension : time
%%       2nd dimension : amplitude
%%
%%
%% Written by Tristan JEHAN at CNMAT
%% Last modifications: 07/97
%%
%%


function out = brandt(in, p, W_size, sample)

global Fs % Sample rate

%% INITIALISATION
jump       = W_size/sample;
index_time = W_size; % Time control
V_size     = length(in); % Number of samples in the signal
Yplus      = zeros(1, W_size+p); % Memory allocation [t-p ... t+W_size-1]
time       = (1/Fs)*[0:V_size-1]; % Time vector construction
out        = [];
```

```
%% MAIN LOOP
while index_time <= V_size

   Yplus(p+1 : p+W_size) = in(index_time-W_size+1 : index_time);

   %% MAIN WINDOW COMPUTATION
   X      = toeplitz(Yplus(p : W_size+p-1), fliplr(Yplus(1:p)));
   A      = lpc(Yplus(p+1 : p+W_size), p);
   Yhat   = X * A(2:p+1)';
   sigma0 = sum((Yplus(p+1 : p+W_size) - Yhat').^2)/W_size;


   %% 2 OTHER WINDOWS LOOP
   for i = p+1: jump :p+W_size-1,

       r = i-p; % Index into the sliding window


       %% FIRST WINDOW COMPUTATION
       X1     = toeplitz(Yplus(p : i-1), fliplr(Yplus(1:p)));
       A1     = lpc(Yplus(p+1 : i), p);
       Yhat1  = X1 * A1(2:p+1)';
       sigma1 = sum((Yplus(p+1 : i) - Yhat1').^2)/r;

       %% SECOND WINDOW COMPUTATION
       X2     = toeplitz(Yplus(i : p+W_size-1), fliplr(Yplus(r+1 : i)));
       A2     = lpc(Yplus(i+1 : p+W_size), p);
       Yhat2  = X2 * A2(2:p+1)';
       sigma2 = sum((Yplus(i+1 : p+W_size) - Yhat2').^2)/(W_size-r);


       %% DETECTION
       Dr = W_size * log10(sigma0) - r * log10(sigma1) - ...
           (W_size-r) * log10(sigma2);

       %% OUTPUT MATRIX (2,x) BUILDING
       out = [out [time(index_time-W_size+r) Dr]'];


   end %% 2 other windows loop

   Yplus(1:p) = Yplus(W_size+1 : W_size+p); % Copy of the p last samples
   index_time = index_time + W_size; % Adjust time

end %% main loop


%% PLOT SECTION
```

```
set(0,'defaultaxesfontsize',6);

subplot(211), plot(time,in);
title('Original Signal','fontsize',9,'color','w');
xlabel('Time');
ylabel('Amplitude');

subplot(212), plot(out(1,:),out(2,:));
title('The Brandt Detection','fontsize',9,'color','w');
xlabel('Time');
ylabel('Amplitude');
```

A.2.2 Divergence Test Program

This program has been written in C by Régine André-Obrecht. It is the translation of a Fortran program. It is composed by `Divergence.c` that calls `ModelARIdent.c` and `ReadFile.c`. An file example of parameters `param.div` is also given.

```
/*
/*        SEGMENTATION USING DIVERGENCE TEST AND VOICING TEST (optional)
/*
/* This program needs ReadFile2.c, ModelARIdent2.c to build the segmentation
/* based on Auto-Regressive process (Divergence test).
/*
/* Input:
/*         * The binary/integer/16 bits signal file (maxi 600000 samples)
/*           which name is in param.div and which extension is ".sig"
/*         * The file param.div where are all the parameters.
/*
/* Output:
/*         * the file with the extension ".seg" which contains positions of samples
/*           (in ascii) where are the segmentation borders.
/*
/* Written by Regine Andre-Obrecht
/* Last modifications: June 2nd 1997
/*
/* Notice that this C program is translated from Fortran.
/*


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <memory.h>
#include <math.h>
#define TAILLEBLOC 128
#define BEGINNING 0
```

```
#define EXTSIGNAL ".sig"
#define EXTSEG ".seg"

/* GLOBAL VARIABLES */

        char NomPC[30];
        char NomSig[30]  ;

        float COR[22],VARF[22],RESF[22],RESB[22],VARB[22],RC[22];
        float SIGNAL[600000];
        float R[22],XA[500];


/* THE FUNCTION */

main (nb,argument)

   int  nb;
   char *argument[];
        {

        int M;
        int nopt,i,NbBloc;
        int NLIM;
        int NMAX,N1,N0,N2,NRUPT,NRUPT0,NN,NbEffectLu;
        int KPL,MQ,ECH,KMIN,KFECH,IECH,IIECH,KM1;
        int NRU,ANTV,AFTERV,PREDV,NTOT,NVOISC,NVOISP;
        int PREDVA,ANUM;
        int XK[500],XK1[500];
        int NUM[500],NAL;
        int I,J,K,IAU;
        int NBMAX,NTEST,nv;

        float B,BB,BV;
        float LAM,LAMB,LAMV;
        float ALPH;
        float NFECH;
        float U,QV,ZAU,Z1,Y,XX2;
        float VARA,RES,XAU,MAXI;

        char temp[30],temp1[30];

        FILE *pFile, *pcResul, *pParam;
        pParam=fopen("param.div","r");


/* OPENING AND READING THE SIGNAL  */

        fscanf(pParam,"%s\n",NomSig);
        fscanf(pParam,"%s\n",temp);
```

```
        strcat(NomSig,temp);
        strcat(NomSig,EXTSIGNAL);

        pFile=fopen(NomSig,"r+b");

        if (pFile)
        {
        printf(" File %s successfully opened...\n",NomSig);
        }
        else
        {
        printf(" File opening error in %s\n",NomSig);
        }

        fscanf(pParam,"%s\n",NomPC);
        strcat(NomPC,temp );
        fscanf(pParam,"%s\n",temp1);
        strcat( NomPC,temp1);
        pcResul=fopen(NomPC,"w");

        fscanf(pParam,"%d\n",&M);
        fscanf(pParam,"%d\n",&nopt);
        fscanf(pParam,"%d\n",&NbBloc);

        printf(" First sample: %d\n",0);
        printf(" Number of blocs to read: %d\n",NbBloc);

        NbEffectLu=EchLireFl( SIGNAL,0,NbBloc,pFile);
        printf(" Number of blocs completely read: %d\n",NbEffectLu);
        if (NbBloc != NbEffectLu)
           printf(" End of file...  \n");

        NMAX = NbEffectLu * TAILLEBLOC;
        printf(" Number of samples: %d\n",NMAX);
        fclose(pFile);

/* END OF SIGNAL DATA
   KMIN THEORICALLY DEPENDS ON THE SAMPLE RATE, BUT HERE IT IS 20ms
   USERS CAN CHANGE IT: "NFECH" IS THE SAMPLE RATE IN KHz.
*/

        fscanf(pParam,"%f\n",&NFECH);
        fscanf(pParam,"%d\n",&KMIN);

        KFECH = (int) (NFECH*20.);
        if(KMIN==0) KMIN=KFECH;
        printf(" Initialisation phase: %d\n",KMIN);

/* INITIALISATION OF CONSTANS (WINDOWS, THRESHOLDS, BIAS...)
   THEY CAN BE CHANGED BY USERS.
```

```
*/

        LAMB = 80.;
        LAMV = 40.;
        BB   = -0.8;
        BV   = -.2  ;

        printf(" Voicing: %d\n",nopt);
        if (nopt == 0)
           {
           ANTV= 2;
           AFTERV=2;
           PREDV=2;
           PREDVA=2;
           }

        ALPH=0.;
        MQ=M+1;
        KM1=KMIN-1;
        N1=0;
        N2=1+N1;
        NRUPT=N1;
        NRUPT0=N1;
        N0=N1;
        NAL=0;
        IECH=KMIN/2;
        IIECH=KMIN/4;


/* NEW INITIALISATIONS AFTER EACH FAILURE */

l3:     KPL=0;
        ZAU=0.;
        MAXI=0.;
        N1=N0;
        N2=N0+1;
        for (I=1; I <= 21; I=I+1)
            {
            VARF[I]=1.;
            VARB[I]=1.;
            RESF[I]=0.;
            RESB[I]=0.;
            COR[I]=0.;
            }
        VARF[MQ]=1.;
        VARB[MQ]=1.;

/* PARAMETERS FOR DETECTION OF VOICING AND CALCUL OF BIAS, THRESHOLDS... */

        if (nopt != 0)
```

```
            {
            ECH=0;
            NRU=0;
            NTOT=0;
            PREDVA=PREDV;
            NLIM=(int)(1.5*KFECH);
            NN=N2;
            XK1[1]=SIGNAL[NN];

            for (K =1 ; K <= (IECH-1); K=K+1)
                {
                NN=N2+K;
                Y=SIGNAL[NN];
                XK[K]=Y;
                XK1[K+1]=Y;
                }
            NN=N2+IECH;
            XK[IECH]=SIGNAL[NN];
            nv=VOIS(XK,XK1,&PREDV,&ANTV,&AFTERV,&NTOT,&NVOISP,&NVOISC,NFECH,&NRU);

            if (AFTERV == 2)
               {
               B=BV;
               LAM=LAMV;
               }
            else
               {
               LAM=LAMB;
               B=BB;
               }
            }
         else
            {
            LAM=LAMB;
            B=BB;
            }

/* ENERGY CONTROL DURING INITIALISATION IN CASE THERE IS A VOICING CONTROL (nopt=2) */

         if (nopt == 2)
            {
            N0=NN;
            XK1[1]=SIGNAL[NN];

            for ( K=1; K <= (IECH-1); K=K+1)
                {
                NN=N0+K;
                Y=SIGNAL[NN];
                XK[K]=Y;
                XK1[K+1]=Y;
```

```
               }
           NN=N0+IECH;
           XK[IECH]=SIGNAL[NN];
           nv=VOIS(XK,XK1,&PREDV,&ANTV,&AFTERV,&NTOT,&NVOISP,&NVOISC,NFECH,&NRU);
           }

/* END OF CONTROL */

        N0=N2+KMIN-1;
        goto l31;


/* TRAITING CURRENT SEGMENT */

l131:   ZAU=Z1;
l31:    N1=N1+1;
        if(N1 > NMAX) goto l998;
        KPL=KPL+1;
        Z1=0.;
        XX2=SIGNAL[N1];
        TREILV(KPL,XX2,M,&VARA,&RES);

        if(KPL < 0) goto l131;
        if(KPL > KMIN) goto l181;

        XA[KPL]=SIGNAL[N1];
        if(KPL < KMIN ) goto l131;
        IAU=1;
        XAU=0.;
        goto l19;

l181:   IAU=0;
        XAU=XA[1];
        I=0;

        for (I=1; I <= KM1; I=I+1)
            XA[I]=XA[I+1];
        XA[KMIN]=SIGNAL[N1];

l19:    NTEST=AUTOV(KMIN,IAU,M,&ALPH,&XAU);

/*  VOICING TEST EVERY 5ms (if nopt=2) */

        if (nopt == 2)
           {
           if ((ECH-IIECH) >= 0)
              {
              NN=N1-IECH;
              XK1[1]=SIGNAL[NN];
              for (K=1; K<=(IECH-1); K=K+1)
```

```
                 {
                 NN=NN+1;
                 Y=SIGNAL[NN];
                 XK[K]=Y;
                 XK1[K+1]=Y;
                 }
              NN=NN+1;
              XK[IECH]=SIGNAL[NN];
              nv=VOIS(XK,XK1,&PREDV,&ANTV,&AFTERV,&NTOT,&NVOISP,&NVOISC,NFECH,&NRU);
              ECH=0;
              if (NRU != 0)
                 {
                 if((ANTV != 2) & (AFTERV != 2)) goto l401;
                 N0=NN-NRU*IIECH;
                 goto l800;
                 }
              }
           else
              {
l401:         ECH=ECH+1;
              }
           }


/* DIVERGENCE-HINKLEY TEST */

        QV=ALPH/VARA;
        U=(2.*XAU*RES/VARA-(1.+QV)*RES*RES/VARA+QV-1.)/(2.*QV);
        Z1=ZAU+U-B;

        if (MAXI > Z1) goto l500;
        MAXI=Z1;
        N0=N1;
l500:   if((MAXI-Z1) < LAM) goto  l131;


l800:   NAL=NAL+1;
        NUM[NAL]=N0;
        if((N0+KMIN) <= NMAX) goto  l3;

l999:   N1=N0;
l998:   N1=N1+1;

        if(N1 > NMAX) goto l997;
        goto l998;


/* WRITING BORDERS AND VOICING RESULTS
   RESULTS ARE IN NUM
*/
```

```
1997:   printf(" The borders are:\n");
        for (J=1; J<=NAL; J=J+1)
            {
            fprintf(pcResul,"%d\n",NUM[J]);
            printf("  %d  %d\n",J,NUM[J]);
            }

        printf(" Number of alarms: %d\n",NAL);

        if (NUM[NAL] < NMAX)
           fprintf(pcResul,"%d\n",NMAX);

        printf(" The end...\n");
        }

/*
/*        SEGMENTATION USING DIVERGENCE TEST AND VOICING TEST (optional)
/*
/* This program needs ReadFile2.c, ModelARIdent2.c to build the segmentation
/* based on Auto-Regressive process (Divergence test).
/*
/* Input:
/*         * The binary/integer/16 bits signal file (maxi 600000 samples)
/*           which name is in param.div and which extension is ".sig"
/*         * The file param.div where are all the parameters.
/*
/* Output:
/*         * the file with the extension ".seg" which contains positions of samples
/*           (in ascii) where are the segmentation borders.
/*
/* Written by Regine Andre-Obrecht
/* Last modifications: June 2nd 1997
/*
/* Notice that this C program is translated from Fortran.
/*


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <memory.h>
#include <math.h>
#define TAILLEBLOC 128
#define BEGINNING 0
#define EXTSIGNAL ".sig"
#define EXTSEG ".seg"

/* GLOBAL VARIABLES */
```

```
        char NomPC[30];
        char NomSig[30]  ;

        float COR[22],VARF[22],RESF[22],RESB[22],VARB[22],RC[22];
        float SIGNAL[600000];
        float R[22],XA[500];


/* THE FUNCTION */

main (nb,argument)

   int  nb;
   char *argument[];
        {

        int M;
        int nopt,i,NbBloc;
        int NLIM;
        int NMAX,N1,N0,N2,NRUPT,NRUPT0,NN,NbEffectLu;
        int KPL,MQ,ECH,KMIN,KFECH,IECH,IIECH,KM1;
        int NRU,ANTV,AFTERV,PREDV,NTOT,NVOISC,NVOISP;
        int PREDVA,ANUM;
        int XK[500],XK1[500];
        int NUM[500],NAL;
        int I,J,K,IAU;
        int NBMAX,NTEST,nv;

        float B,BB,BV;
        float LAM,LAMB,LAMV;
        float ALPH;
        float NFECH;
        float U,QV,ZAU,Z1,Y,XX2;
        float VARA,RES,XAU,MAXI;

        char temp[30],temp1[30];

        FILE *pFile, *pcResul, *pParam;
        pParam=fopen("param.div","r");


/* OPENING AND READING THE SIGNAL  */

        fscanf(pParam,"%s\n",NomSig);
        fscanf(pParam,"%s\n",temp);
        strcat(NomSig,temp);
        strcat(NomSig,EXTSIGNAL);

        pFile=fopen(NomSig,"r+b");
```

```
        if (pFile)
        {
        printf(" File %s successfully opened...\n",NomSig);
        }
        else
        {
        printf(" File opening error in %s\n",NomSig);
        }

        fscanf(pParam,"%s\n",NomPC);
        strcat(NomPC,temp );
        fscanf(pParam,"%s\n",temp1);
        strcat( NomPC,temp1);
        pcResul=fopen(NomPC,"w");

        fscanf(pParam,"%d\n",&M);
        fscanf(pParam,"%d\n",&nopt);
        fscanf(pParam,"%d\n",&NbBloc);

        printf(" First sample: %d\n",0);
        printf(" Number of blocs to read: %d\n",NbBloc);

        NbEffectLu=EchLireFl( SIGNAL,0,NbBloc,pFile);
        printf(" Number of blocs completely read: %d\n",NbEffectLu);
        if (NbBloc != NbEffectLu)
           printf(" End of file...  \n");

        NMAX = NbEffectLu * TAILLEBLOC;
        printf(" Number of samples: %d\n",NMAX);
        fclose(pFile);

/* END OF SIGNAL DATA
   KMIN THEORICALLY DEPENDS ON THE SAMPLE RATE, BUT HERE IT IS 20ms
   USERS CAN CHANGE IT: "NFECH" IS THE SAMPLE RATE IN KHz.
*/

        fscanf(pParam,"%f\n",&NFECH);
        fscanf(pParam,"%d\n",&KMIN);

        KFECH = (int) (NFECH*20.);
        if(KMIN==0) KMIN=KFECH;
        printf(" Initialisation phase: %d\n",KMIN);

/* INITIALISATION OF CONSTANS (WINDOWS, THRESHOLDS, BIAS...)
   THEY CAN BE CHANGED BY USERS.
*/

        LAMB = 80.;
        LAMV = 40.;
```

```
        BB   = -0.8;
        BV   = -.2  ;

        printf(" Voicing: %d\n",nopt);
        if (nopt == 0)
           {
           ANTV= 2;
           AFTERV=2;
           PREDV=2;
           PREDVA=2;
           }

        ALPH=0.;
        MQ=M+1;
        KM1=KMIN-1;
        N1=0;
        N2=1+N1;
        NRUPT=N1;
        NRUPT0=N1;
        N0=N1;
        NAL=0;
        IECH=KMIN/2;
        IIECH=KMIN/4;


/* NEW INITIALISATIONS AFTER EACH FAILURE */

l3:     KPL=0;
        ZAU=0.;
        MAXI=0.;
        N1=N0;
        N2=N0+1;
        for (I=1; I <= 21; I=I+1)
            {
            VARF[I]=1.;
            VARB[I]=1.;
            RESF[I]=0.;
            RESB[I]=0.;
            COR[I]=0.;
            }
        VARF[MQ]=1.;
        VARB[MQ]=1.;

/* PARAMETERS FOR DETECTION OF VOICING AND CALCUL OF BIAS, THRESHOLDS... */

        if (nopt != 0)
           {
           ECH=0;
           NRU=0;
           NTOT=0;
```

```
            PREDVA=PREDV;
            NLIM=(int)(1.5*KFECH);
            NN=N2;
            XK1[1]=SIGNAL[NN];

            for (K =1 ; K <= (IECH-1); K=K+1)
                {
                NN=N2+K;
                Y=SIGNAL[NN];
                XK[K]=Y;
                XK1[K+1]=Y;
                }
            NN=N2+IECH;
            XK[IECH]=SIGNAL[NN];
            nv=VOIS(XK,XK1,&PREDV,&ANTV,&AFTERV,&NTOT,&NVOISP,&NVOISC,NFECH,&NRU);

            if (AFTERV == 2)
               {
               B=BV;
               LAM=LAMV;
               }
            else
               {
               LAM=LAMB;
               B=BB;
               }
            }
         else
            {
            LAM=LAMB;
            B=BB;
            }

/* ENERGY CONTROL DURING INITIALISATION IN CASE THERE IS A VOICING CONTROL (nopt=2) */

         if (nopt == 2)
            {
            N0=NN;
            XK1[1]=SIGNAL[NN];

            for ( K=1; K <= (IECH-1); K=K+1)
                {
                NN=N0+K;
                Y=SIGNAL[NN];
                XK[K]=Y;
                XK1[K+1]=Y;
                }
            NN=N0+IECH;
            XK[IECH]=SIGNAL[NN];
            nv=VOIS(XK,XK1,&PREDV,&ANTV,&AFTERV,&NTOT,&NVOISP,&NVOISC,NFECH,&NRU);
```

```
           }

/* END OF CONTROL */

        N0=N2+KMIN-1;
        goto l31;


/* TRAITING CURRENT SEGMENT */

l131:   ZAU=Z1;
l31:    N1=N1+1;
        if(N1 > NMAX) goto l998;
        KPL=KPL+1;
        Z1=0.;
        XX2=SIGNAL[N1];
        TREILV(KPL,XX2,M,&VARA,&RES);

        if(KPL < 0) goto l131;
        if(KPL > KMIN) goto l181;

        XA[KPL]=SIGNAL[N1];
        if(KPL < KMIN ) goto l131;
        IAU=1;
        XAU=0.;
        goto l19;

l181:   IAU=0;
        XAU=XA[1];
        I=0;

        for (I=1; I <= KM1; I=I+1)
            XA[I]=XA[I+1];
        XA[KMIN]=SIGNAL[N1];

l19:    NTEST=AUTOV(KMIN,IAU,M,&ALPH,&XAU);

/*  VOICING TEST EVERY 5ms (if nopt=2) */

        if (nopt == 2)
           {
           if ((ECH-IIECH) >= 0)
              {
              NN=N1-IECH;
              XK1[1]=SIGNAL[NN];
              for (K=1; K<=(IECH-1); K=K+1)
                  {
                  NN=NN+1;
                  Y=SIGNAL[NN];
                  XK[K]=Y;
```

```
                 XK1[K+1]=Y;
                 }
              NN=NN+1;
              XK[IECH]=SIGNAL[NN];
              nv=VOIS(XK,XK1,&PREDV,&ANTV,&AFTERV,&NTOT,&NVOISP,&NVOISC,NFECH,&NRU);
              ECH=0;
              if (NRU != 0)
                 {
                 if((ANTV != 2) & (AFTERV != 2)) goto l401;
                 N0=NN-NRU*IIECH;
                 goto l800;
                 }
              }
           else
              {
l401:         ECH=ECH+1;
              }
           }


/* DIVERGENCE-HINKLEY TEST */

        QV=ALPH/VARA;
        U=(2.*XAU*RES/VARA-(1.+QV)*RES*RES/VARA+QV-1.)/(2.*QV);
        Z1=ZAU+U-B;

        if (MAXI > Z1) goto l500;
        MAXI=Z1;
        N0=N1;
l500:   if((MAXI-Z1) < LAM) goto  l131;


l800:   NAL=NAL+1;
        NUM[NAL]=N0;
        if((N0+KMIN) <= NMAX) goto  l3;

l999:   N1=N0;
l998:   N1=N1+1;

        if(N1 > NMAX) goto l997;
        goto l998;


/* WRITING BORDERS AND VOICING RESULTS
   RESULTS ARE IN NUM
*/


l997:   printf(" The borders are:\n");
        for (J=1; J<=NAL; J=J+1)
```

```
            {
            fprintf(pcResul,"%d\n",NUM[J]);
            printf("  %d  %d\n",J,NUM[J]);
            }

        printf(" Number of alarms: %d\n",NAL);

        if (NUM[NAL] < NMAX)
           fprintf(pcResul,"%d\n",NMAX);

        printf(" The end...\n");
        }



/*
/*                    SUB PROGRAM FOR READING THE SIGNAL FILE
/*
/* This program is called by Divergence.c and reads a signal raw file
/* (integers/16 bits). His logic name is pFile.
/*
/* Written by Regine Andre-Obrecht
/* Last modifications: June 2nd 1997
/*

/* DECLARATIONS */

#include <stdio.h>
#include <memory.h>
#include <stdlib.h>
#define TAILLEBLOC 128
#define BEGINNING 1


int SigPositionEch (pFile, EchDep)

FILE *pFile;
int EchDep;

  {
  int status;
  unsigned long roffset;
  roffset = (unsigned long) (EchDep) * (unsigned long) (sizeof(short int));
  status = fseek(pFile, roffset, BEGINNING);

  if (!status)
     {
     printf(" Status: %d\n",status);
     return (status);
     }
```

```
   }

int EchLireFl ( Buffer, EchDep, NbBloc, pFile)
int EchDep, NbBloc;
float *Buffer;
FILE *pFile;

   {
   int status, i, j,Nmax,neffect;
   short int Signal[600000];
   status = SigPositionEch ( pFile, EchDep);

        i=0;
        neffect = NbBloc;
        while (i < NbBloc)
              {
              memset (Signal+(i*TAILLEBLOC),'\0',TAILLEBLOC*sizeof(short int));
              status=fread((char *)(Signal+(i*TAILLEBLOC)),sizeof( short int),
                     TAILLEBLOC,pFile);
              i++;
              if(status < TAILLEBLOC)
                {
                neffect = i-1;
                goto l222;
                }
              }

l222:   Nmax = i*TAILLEBLOC;
        for (j=1; j<=Nmax; j=j+1)
            Buffer[j]= (float) Signal[j-1];
        return (neffect);
   }



/*
/*  IDENTIFICATION OF AR MODEL BY AUTOCORRELATION METHOD (SHORT AND LONG TERM MODEL)
/*
/* This program is called by Divergence.c.
/*
/* Written by Regine Andre-Obrecht
/* Last modifications: June 2nd 1997
/*


/* IDENTIFICATION OF AR MODEL BY AUTOCORRELATION METHOD (SHORT TERM MODEL) */

/* GLOBAL VARIABLES */

extern float XA[500],R[22];
```

```
extern float VARF[22],COR[22],VARB[22],RC[22],RESF[22],RESB[22];
extern float SIGNAL[600000];

      int AUTOV(N,IAU,M,ALPH,XAU)
      int N,IAU,M;
      float *ALPH,*XAU;

      {
      int MQ,K,NK,NP,I;
      int MINC,IB,IP,MH;
      float AT,S;
      float AU[22],RCA[22],VALPH,VXAU;

      VALPH=0.;
      VXAU=*XAU;
      MQ=M+1;
      if(IAU == 0) goto l11;

      for (K=1; K<=MQ; K=K+1)
          {
          R[K]=0.;
          NK=N-K+1;
          for (NP=1; NP<=NK; NP=NP+1)
              R[K]=R[K]+XA[NP]*XA[NP+K-1];
          }
      goto l12;

l11:  for (K=2; K<=MQ; K=K+1)
      R[K]=R[K]-VXAU*XA[K-1]+XA[N-K+1]*XA[N];
      R[1]=R[1]-VXAU*VXAU+XA[N]*XA[N];

l12:  if (R[1] <= 0.) R[1]=1.;
      RCA[1]= -R[2]/R[1];
      AU[1]=1.;
      AU[2]=RCA[1];
      VALPH=R[1]+R[2]*RCA[1];
      if (M <= 1) goto l51;

      for (MINC=2; MINC<=M; MINC=MINC+1)
          {
          S=0.;
          for (IP=1; IP<=MINC; IP=IP+1)
          S=S+R[MINC-IP+2]*AU[IP];
          RCA[MINC]=-S/VALPH;
          MH=MINC/2+1;
          for (IP=2; IP<=MH; IP=IP+1)
              {
              IB=MINC-IP+2;
              AT=AU[IP]+RCA[MINC]*AU[IB];
              AU[IB]=AU[IB]+RCA[MINC]*AU[IP];
```

```
            AU[IP]=AT;
            }
        AU[MINC+1]=RCA[MINC];
        VALPH=VALPH+RCA[MINC]*S;
        if(VALPH <= 0.) return (-1);
        }

l51:  VALPH=VALPH/ (float)(N-1);
      *ALPH=VALPH;
      VXAU=0.;

      for (I=1; I<=MQ; I=I+1)
          VXAU=VXAU+ AU[I]*XA[N-I+1];
      *XAU=VXAU;
      return (1);
      }

/* IDENTIFICATION PROGRAM OF AR MODEL BY THE BURG METHOD (LONG TERM MODEL) */

      void TREILV (K,Y,M,VARA,RES)
      int K,M;
      float Y;
      float *VARA,*RES;

      {
      float RESBP[22], GAINV, GAINC;
      int IK,I,KK;

      if (Y == RESF[1]) Y=Y+1.;
      GAINV=1./(float)(K);
      VARF[1]=VARF[1]+GAINV*(Y*Y-VARF[1]);
      VARB[1]=VARF[1];
      RESF[1]=Y;
      IK=M;

      if ((K-1) < M) IK=K-1;
      if(K == 1) goto l2;

      for ( I=1; I <= IK; I=I+1)
          RESBP[I]=RESB[I];

      for (I=1; I<=IK; I=I+1)
          {
          KK=K-I;
          GAINC=1./(float)(KK);
          COR[I]=COR[I]+GAINC*(RESBP[I]*RESF[I]-COR[I]);
          RC[I]=2.*COR[I]/(VARF[I]+VARB[I]);
          RESF[I+1]=RESF[I]-RC[I]*RESBP[I];
          RESB[I+1]=RESBP[I]-RC[I]*RESF[I];
          VARF[I+1]=VARF[I+1]+GAINC*(RESF[I+1]*RESF[I+1]-VARF[I+1]);
```

```
          VARB[I+1]=VARB[I+1]+GAINC*(RESB[I+1]*RESB[I+1]-VARB[I+1]);
          }
l2 :  RESB[1]=Y;
      *VARA=VARF[IK+1];
      *RES=RESF[IK+1];
      }


      int  VOIS (XK,XK1,PREDV,ANTV,AFTERV,NTOT,NVOISP,NVOISC,NFECH,NRU)

/* SUB PROGRAM FOR VOICING CALCUL DURING INITIALISATION PHASE AND CHECKING OF
   THIS VALUE BY "NRU" (NRU, DIFFERENT OF 0, MEANS A VARIATION OF THE VOICING
   DURING THE SEGMENT
*/

      int *XK,*XK1,*PREDV,*ANTV,*AFTERV,*NTOT,*NVOISP,*NVOISC,*NRU;
      float NFECH;

      {
      int IECH,KMIN,NLONG,NANT,NAFT,I,NZERO,NVOIS;
      float  LSEUI1,LSEUI2,LSEUI3,LSEUI4,LSEUI5;
      float AU,EN,XKK,XKK1,Y,Z,H;

      KMIN= (int) (NFECH*20.);
      IECH=KMIN/2;

/* THRESHOLD ON ENERGY */

      LSEUI1=210000./128;
      LSEUI2=8500000./128;

/* THRESHOLD ON AUTOCORRELATION OF THE SIGNAL  */

      LSEUI3=190000./128;
      LSEUI4=5500000./128;
      LSEUI5=80./256.*IECH;
      NLONG=IECH;
      *NTOT= *NTOT+1;
      EN=0.000001;
      AU=0.000001;
      NZERO=0;
      NAFT=1;
      if(XK[1] < 0) NAFT=-1;

      for (I=1; I<=IECH; I=I+1)
          {
          XKK=XK[I];
          NANT=NAFT;
          NAFT=1;
          if(XKK < 0) NAFT= -1;
```

```
          if(XKK == 0) NAFT= NANT;
          if(NANT != NAFT) NZERO=NZERO + 1;
          XKK1=XK1[I];
          EN=EN+XKK*XKK/NLONG;
          AU=AU+XKK*XKK1/NLONG;
          }

      if (EN <= LSEUI1)
         {
         Z=0.;
         }
      else
         {
         if (EN <= LSEUI2)
            {
            Z=1.;
            }
         else
            {
            Z=2.;
            }
         }

      if (AU <= LSEUI3)
         {
         Y=0.;
         }
      else
         {
         if (AU <= LSEUI4)
            {
            Y=1.;
            }
         else
            {
            Y=2.;
            }
         }
      NVOIS=(int)((Z+Y)/2);

/* VOICING CORRECTION DEPENDING ON THE REFLEXION COEFFICIENT AND THE NUMBER
   OF ZERO-CROSSING
*/
      H=1.*AU/EN;
      if (NVOIS == 2)
         {
         if (AU < 0) NVOIS=1;
         if ((H < 0.85) && (EN < (10000000./128))) NVOIS=1;
         if (( Z == 2.) && (NZERO >= LSEUI5)) NVOIS=1;
         }
```

```
      if ((EN > (20000./128)) && (NVOIS == 0) && (H < 0.70)) NVOIS=1;

/* NTOT IS THE NUMBER OF BLOCS TRAITED DURING THE SEGMENT
   NRU IS THE NUMBER OF BLOCS SINCE THE LAST VOICING VARIATION IN THE SEGMENT
   WITHOUT FAILURE NRU=0
*/

      if (*NTOT == 1)
         {
         *ANTV=NVOIS;
         *PREDV=NVOIS;
         *AFTERV=NVOIS;
         *NVOISC=1;
         *NVOISP=0;
         }
      else
         {
         if (NVOIS == *AFTERV)
            {
            *NVOISC = *NVOISC + 1;
            if (*NVOISC == *NTOT) return(1);
            if (*NVOISC > 4)  *NRU=*NVOISC;
            }
         else
            {
            *NVOISP=*NVOISC;
            *PREDV=*ANTV;
            *ANTV=*AFTERV;
            *AFTERV=NVOIS;
            *NVOISC=1;
              if (*PREDV == *ANTV) return(1);
              if (*PREDV == *AFTERV)
                 {
                 *NVOISC=*NTOT;
                 *ANTV=*AFTERV;
                 }
              else
                 {
                 *NRU= *NVOISP+1;
                 }
            }
         }
      }



/* FILE EXAMPLE OF PARAMETERS

/usr/people/tristan/PROGRAMS/SEGMENTATION/
coleman
```

```
/usr/people/tristan/PROGRAMS/SEGMENTATION/
.seg
2
1
1500
44.1
56

/* WHAT'S THAT? */

1. Path for the signal file (char)
2. Signal file name without extension (char)
3. Path for result file (char)
4. extension for result file
5. Order (int) of the estimated autoregressive model
6. Voicing option: 0 = No special traitment
                   1 = Voicing used for bias and threshold for the
                       initialisation phase of each segment.
                   2 = like 1 + voicing used as a failure test
7. Number of blocs to read (int) (limited to 4780)
8. Sample rate (float in KHz)
9. New window size (int)
   if the size is equivalent to 20ms, put 0
   else put the size (number of samples)
```

Appendix B

Pitch Detection

This section is composed by the wavelet pitch tracking program in Matlab which is divided in several functions. Some of them are available in the UVI-Wave wavelet toolbox [41]. And the reader can also refer to Appendix A.1 for the function `Energy`.

```
%%
%%           THE PITCH DETECTION FUNCTION
%%
%% This function computes the pitch and the Energy of the signal and the filtered
%% signal and then build and plot a pitch segmented detection based on the pitch
%% estimation and the Energy amplitude.
%%
%% Arguments input:
%% signal: name of the signal vector
%% Fs: sample rate
%% fmin: The minimum freq
%% fmax: The maximum freq
%% threshold: on energy, for pitch segmentation
%%
%% Arguments output:
%% pitchseg: 2 columns vector containing time and associated pitch
%%
%% Written by Tristan JEHAN at CNMAT
%% Last modifications: 07/97
%%
%%

function pitchseg = SegPitch(signal,Fs,fmin,fmax,threshold)

%% COMPUTES THE SIGNAL ENERGY
[E_time_signal,E_signal,E_maxi_signal,E_stride_signal] = Energy(signal,Fs);

%% COMPUTES THE PITCH
[maxima,pitch,maxima3,pitch3,pitch4,filt] = Pitch(signal,Fs,fmin,fmax);

%% COMPUTES THE FILTERED SIGNAL ENERGY
[E_time_filt,E_filt,E_maxi_filt,E_stride_filt] = Energy(filt,Fs);
```

```
%% BUILD THE TIME VECTOR
len0 = length(signal);
time = [0:len0-1]*(1/Fs);

%% BUILD THE SEGMENTED PITCH
len1 = length(maxima);
len2 = length(E_filt);
len3 = length(E_signal);
pitch2 = pitch;

for i=1:len1,
    j=ceil( maxima(i)/(E_stride_signal*(1/Fs)) );
    if j<len2 & j<len3
       if E_filt(j) < E_maxi_filt*threshold/100 & E_signal(j) < ...
                                                E_maxi_signal*threshold/100
          pitch2(i)=0;
       end
    end
end

pitchseg = [maxima' pitch2'];

%% PLOT PARAMETERS

set(0,'defaultaxesfontsize',6);

E_maxsig_filt   = max(filt);
E_maxsig_signal = max(signal);
E_norm_filt     = E_maxsig_filt/E_maxi_filt;
E_norm_signal   = E_maxsig_signal/E_maxi_signal;
trace           = max(filt)/2;
maxpitch        = max(pitch2);

%% PLOT SECTION

subplot(311), plot(time,signal,'w',E_time_signal,E_signal*E_norm_signal,'k');
title('Original Signal and Energy','fontsize',9,'color','w');
ylabel('Amplitude');

subplot(312), plot(time,filt(1:len0),'y',maxima,ones(1,len1)*trace,'g+',maxima3, ...
  ones(1,length(maxima3))*(trace-1),'b+',E_time_filt,E_filt*E_norm_filt,'r');
title('Filtered Function, Energy and maxima detection (before and after ...
  processing)','fontsize',9,'color','w');
ylabel('Amplitude');

subplot(313), semilogy(maxima,pitch,'r',maxima,pitch2,'y',maxima3,pitch3+10,'g', ...
  maxima3,pitch4+20,'b');
axis([-inf time(len0) fmin-50 (maxpitch+50)]);
title('Original Pitch (b), median filter (g) and Traited and Segmented ...
```

```
  Pitch (y)','fontsize',9,'color',[0 0.5 1]);
ylabel('Amplitude');
xlabel('Time');

zoom on;



%%
%%           THE PITCH DETECTION FUNCTION
%%
%% This function detects the GCI (Glotal Closure Instant) which is usually used in
%% speech detection. It uses a wavelet based method. Then it detects maxima in the
%% filtered signal and computes the pitch signal.
%%
%% Arguments input:
%% in: The original signal
%% Fs: The sample rate
%% fmin: The minimum freq
%% fmax: The maximum freq
%%
%% Arguments output:
%% maxima1: maxima of the pseudo-function with the case-by-case method
%% pitch: complete pitch vector with the case-by-case method
%% maxima: maxima of the pseudo-function before treatment
%% pitch2: complete pitch vector with the median filter method
%% pitch3: complete pitch vector before treatment
%% out: the filtered function
%%
%%
%% Written by Tristan JEHAN at CNMAT
%% Last modifications: 07/97
%%
%%


function [maxima1,pitch,maxima,pitch2,pitch3,out] = Pitch(in,Fs,fmin,fmax)


%% WAVELET AND SCALE FUNCTIONS CONSTRUCT PARAMETERS

Ka = floor(log10(Fs/fmin)/log10(2));
Kb = ceil(log10(Fs/fmax)/log10(2));

%% WAVELET FILTERING FUNCTION

[h,g]   = daub(8); % generate Daubechies filters
[s,w] = wavelet(h,g,Kb); % Determination of scaling function S (High cut-off freq)
[s1,w1] = wavelet(h,g,Ka); % Determination of wavelet function W1 (Low cut-off freq)
filt = conv(s,w1); % Filtering function
```

```
out = conv(filt,in); % Filtered signal

%% TIME VECTOR

len   = length(in);
time  = [0:len-1] * (1/Fs);

%% MAXIMUM DETECTION

filt_size = 3;
filter    = [-1 0 1]; % Derivation filter
deriv     = zeros(1,len);
maxima    = [];
X         = (filt_size-1)/2;

%% FILTERING

for i = (filt_size+1)/2 : len-X,
    deriv(i) = sum(filter .* out(i-X : i+X)); % Derivation
    if i>1
       if ((deriv(i-1)>0) & (deriv(i)<0)) % Zero-crossing detection

              %% PARABOLIC APPROXIMATION

     a = ( ( (out(i-1)-out(i))   / (time(i-1)-time(i))   ) - ...
         ( (out(i-1)-out(i+1)) / (time(i-1)-time(i+1)) ) ) / (time(i)-time(i+1));
     b = ( (out(i-1)-out(i)) / (time(i-1)-time(i)) ) - a * (time(i-1)-time(i));

     maxima = [maxima (-b/(2*a))+time(i)];

        end
     end
end

%% PEAKS CLEANING 1

len2      = length(maxima);
newlen2   = len2;
precision = 15; % percentage of pitch error
precision2= 30;
errors    = 0;
maxima1   = maxima;

for i=1:len2-5,

    if i==newlen2-4
       break;
    end

    D1=maxima1(i+1)-maxima1(i); % Durations between 2 Zero-crossing
```

```
    D2=maxima1(i+2)-maxima1(i+1);
    D3=maxima1(i+3)-maxima1(i+2);
    D4=maxima1(i+4)-maxima1(i+3);
    D5=maxima1(i+5)-maxima1(i+3);

    if abs(1-(D2/D1))>precision/100 & abs(1-(D3/D2))>precision/100 & ...
      (abs(1-(D4/D1))<precision/100 | abs(1-(D5/D1))<precision/100)
       maxima1(i+2)=[];
       newlen2=newlen2-1;
       errors=errors+1;
    end

end

errors

%% PITCH ESTIMATION 1

pitch = zeros(1,newlen2); % Pitch vector

for i=2:newlen2-2,
    D1=maxima1(i)-maxima1(i-1);
    D2=maxima1(i+1)-maxima1(i);
    if abs(1-(D2/D1))<precision2/100
       pitch(i) = 1/((maxima1(i+2)-maxima1(i-1))/3);
    else
       pitch(i) = 1;
    end
end

maxima2 = maxima;

%% PEAKS CLEANING 2

for i=2:len2-2,

    D1=maxima(i)-maxima(i-1); % Durations between 2 Zero-crossing
    D2=maxima(i+1)-maxima(i);
    D3=maxima(i+2)-maxima(i+1);

    maxima2(i) = median([D1 D2 D3]);

end

%% PITCH ESTIMATION 2

pitch2 = zeros(1,len2); % Pitch vector

for i=2:len2-2,
    pitch2(i) = 1/((maxima2(i-1)+maxima2(i)+maxima2(i+1))/3);
```

```
end

%% PITCH ESTIMATION 3: THE NON FILTERED

pitch3 = zeros(1,len2); % Pitch vector

for i=2:len2-2,
    pitch3(i) = 1/((maxima(i+2)-maxima(i-1))/3);
end
```

Other many small programs have also been written for converting, loading or saving sound files with Matlab.

Bibliography

[1] A. Von Brandt,
Detecting and estimating parameters jumps using ladder algorithms and likelihood ratio test
Proc. ICASSP, Boston, MA, 1983, p. 1017–1020

[2] A. Willsky,
A survey of design methods for failure detection
Automatica, vol. 12, p. 601–611, 1976

[3] John R. Pierce and Scott A. Van Duyne,
A passive nonlinear digital filter design which facilitates physics-based sound synthesis of highly nonlinear musical instruments
Center for Computer Research in Music and Acoustics (CCRMA), Stanford University
Acoustical Society of America, February 1997

[4] Thomas P.Krauss, Lauren Shure and John N.Little,
Signal Processing TOOLBOX User's guide (for use with MATLAB)
http://www.mathworks.com/

[5] John Hajda,
A new model for segmenting the envelope of musical signal: the relative salience of steady state versus attack, revisited
University of California, presented at the 101st convention 1996 November 8–11, Los Angeles
AES (an Audio Engineering Society preprint)

[6] Ludger Solbach and Rolf Wohrmann,
Sound onset localization and partial tracking in gaussian white noise
ICMC Proceedings 1996, p. 324
{ludger, rolf}@tick.ti6.tu-harburg.de

[7] Paul Masri and Andrew Bateman,
Improved modelling of attack transients in music analysis-resynthesis
Digital Music Research Group, University of Bristol
ICMC Proceedings, 1996
http://www.fen.bris.ac.uk/elec/dmr/
Paul.Masri@bristol.ac.uk

[8] Andrew Choi,
Real-time fundamental frequency estimation by least-square fitting
IEEE Transactions on Speech and Audio Processing, vol 5 no 2, March 1997

[9] Duane Hanselman and Bruce Littlefield,
Mastering MATLAB (A comprehensive tutorial and reference)
Prentice-Hall, Inc. 1996

[10] Alan WESLEY PEEVERS,
A real-time 3D signal analysis/synthesis tool based on the overlap add Short Time Fourier Transform
Department of Electrical Engineering, University of California, Berkeley

[11] Michèle BASSEVILLE and Igor V.NIKIFOROV,
Detection of abrupt changes, Theory and Application
1993

[12] Régine ANDRÉ-OBRECHT,
A new statistical approach for the automatic segmentation of continuous speech signals
IEEE ASSP vol. 36 no 1, January 1988

[13] Jonathan FOOTE,
A similarity measure for automatic audio classification
Institute of systems science national University of Singapore
http://repulse.iss.nus.sg:8080/ jtf@iss.nus.sg

[14] C.S. BURRUS and T.W. PARKS,
DFT/FFT and convolution algorithms (Theory and Implementation)
Wiley-Interscience, 1985

[15] H.J. NUSSBAUMER,
Fast Fourier Transform and convolution algorithms (second corrected and updated edition)
Springer-verlag (Berlin), Heidelberg (New-York)

[16] Christopher E.REID and Thomas B.PASSIN,
Signal Processing in C
1992, John Willey & Sons, Inc.

[17] Wiliam H.PRESS, Saul A.TEUKOLSKY, Wiliam T.VETTERLING and Brian P.FLANNERY,
Numerical recipes in C (The art of scientific computing, Second edition)
Cambridge University Press, 1992

[18] Jae S.LIM and Alan V.OPPENHEIM,
Advanced topics in signal processing
Edited by Jae S.Lim and Alan V.Oppenheim

[19] Thomas F.COLEMAN and Charles VAN LOAN,
Handbook for matrix computations
SIAM, 1988

[20] Eric MÉTOIS,
Dossier long de troisième année: Plusieures approches de la détection et de l'estimation du pitch d'un signal
CNMAT, (Telecom PARIS), 1991

[21] John KITAMURA, Boris DOVAL and Xavier RODET,
A robust pitch tracker
IRCAM, 1990

[22] Peter J.BROCKWELL and Richard DAVIS,
Time Series: Theory and Methods, Second Edition
Springer-Verlag, 1991

[23] Boris DOVAL and Xavier RODET,
Fundamental frequency estimation and tracking using maximum likelihood harmonic matching and HMMs
IRCAM, April 1993

[24] Michèle BASSEVILLE and A. BENVENISTE,
Detection of abrupt changes in signals and dynamical systems
New York, Springer-Verlag, 1986

[25] Shie QIAN and Dapang CHEN,
Joint time-frequency analysis (methods and applications)
Prentice Hall PTR, 1996

[26] Sukandar KARTADINATA,
Thesis: Algorithms for combined sound analysis and synthesis and their implementation in a multi-processor environment
CNMAT, 1996

[27] Lee-Min LEE and Hsiao-Chuan WANG,
An extended Levinson-Durbin algorithm for the analysis of noisy Autoregressive process
IEEE Signal Processing Letters, vol. 3, no 1, January 1996

[28] Christopher WENDT and Athina P.PETROPULU,
Pitch determination and speech segmentation using the discrete wavelet transform
IEEE International Symposium on Circuits and Systems, 1996 p. 45–48 vol.2
Chris@cbis.ece.drexel.edu

[29] Glenn A.SHELBY, Christopher M.COOPER and Reza R.ADHAMI,
A wavelet-based speech pitch detector for tone languages
Proceedings of the IEEE-SP International Symposium on Time-Frequency and Time-Scale Analysis, 1994, p. 596–9

[30] Shubha KADAMBE and G.Faye BOUDREAUX-BARTELS,
Application of the wavelet transform for pitch detection of speech signals
IEEE Transactions on Information Theory, vol. 38, no 2, March 1992, p. 917–924

[31] S.G. MALLAT and S. ZHONG,
Characterization of signals from multiscale edges
IEEE Trans. of Patt. Analy. and Mach. Intell., vol. 14, p. 710–32, July 1992.

[32] Eric J.STOLLNITZ, Tony D.DEROSE and David H.SALESIN,
Wavelets for computer graphics: A primer, part 1
IEEE Computer Graphics and Applications, May 1995

[33] Eric J.STOLLNITZ, Tony D.DEROSE and David H.SALESIN,
Wavelets for computer graphics: A primer, part 2
IEEE Computer Graphics and Applications, July 1995

[34] Régine ANDRÉ-OBRECHT,
Segmentation automatique du signal de parole
Thèse de 3ième cycle, Université de Rennes 1, May 1985

[35] Michèle BASSEVILLE and A. BENVENISTE,
Sequential detection of abrupt changes in spectral characteristics of digital signals
IEEE Transactions on Information Theory, vol. IT–29, p. 709–723, Sept. 1983

[36] Olivier RIOUL and Martin VETTERLI,
Wavelets and Signal Processing
IEEE SP Magazine, October 1991

[37] Michael GOODWIN,
Frequency-domain analysis-synthesis of musical sounds
CNMAT and Department of Electrical Engineering and Computer Science, UCB
michaelg@cnmat.berkeley.edu

[38] Michael GOODWIN and Martin VETTERLI,
Time-frequency signal models for music analysis, transformation and synthesis
CNMAT and Department of Electrical Engineering and Computer Science, UCB

[39] Paolo PRANDONI, Michael GOODWIN and Martin VETTERLI,
Optimal time segmentation for signal modeling and compression
LCAV, Ecole Polytechnique Fédérale d Lausanne, Switzerland
EECS, University of California, Berkeley, USA
prandoni@de.epfl.ch, michaelg@eecs.berkeley.edu, vetterli@de.epfl.ch

[40] Michael GOODWIN and Xavier RODET,
Efficient Fourier synthesis of nonstationary sinusoids
CNMAT and IRCAM
michaelg@cnmat.berkeley.edu and rod@ircam.fr

[41] Nuria GONZALES PRELCIC and Sergio J. GARCIA GALAN,
UVI-Wave, Wavelet TOOLBOX for use with MATLAB (Version 1.0.2)
Grupo de Teoría de la Señal, Universidad de Vigo
ftp://ftp.cesga.es/pub/Uvi-Wave/matlab/

[42] Wing-kei YIP, Kwong-sak LEUNG and Kin-hong WONG,
Pitch detection of speech signals in noisy environment by wavelet
Chinese University of Hong Kong, Department of Computer Science, Shatin, Hong Kong
SPIE vol. 2491

[43] Matthew WRIGHT,
A pedagogical catalog of musical sound synthesis techniques and the control of sound they afford
CNMAT, 1997
matt@cnmat.berkeley.edu

[44] Ph. DEPALLE, G. GARCÍA and X. RODET
The recreation of a castrato voice, Farinelli's voice
IRCAM
{phd, garcia, rod}@ircam.fr

[45] Matthew WRIGHT and Adrian FREED
Open SoundControl: A new protocol for communicating with sound synthesizers
CNMAT, ICMC 1997
matt,adrian@cnmat.berkeley.edu
http://cnmat.CNMAT.Berkeley.EDU/People/

[46] Adrian FREED
Bring Your Own control to additive synthesis

CNMAT, ICMC 1995
adrian@cnmat.berkeley.edu
http://www.cnmat.berkeley.edu/ adrian/

[47] David K. Mellinger
Event formation and separation in musical sound
CCRMA, Department of music, December 1991

[48] Cyril Drame
Contrôle du timbre et de la synthèse additive par réseaux de neurones
IRCAM, INT, CNMAT Thesis, 1996

[49] David L. Wessel
Timbre space as musical control structure
Computer Music Journal 3(2), p. 45–52, 1979

[50] David R. Brillinger and Rafael A. Irizarry
An investigation of the second-and higher-order spectra of music
Statistics Department University of California Berkeley

[51] Mickael Lee and David Wessel
Connectionist models for real-time control of synthesis and compositional algorithms
CNMAT
wessel,lee@cnmat.berkeley.edu

[52] David Wessel
Instruments that learn, refined controllers, and source model loudspeakers
Computer Music Journal, vol. 15, no 4, winter 1991, MIT wessel@cnmat.berkeley.edu