N° d'ordre : 3373

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Martin Raspaud**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Modèles spectraux hiérarchiques pour les sons et applications

---

Soutenue le : 24 Mai 2007

Après avis des rapporteurs :

Daniel Arfib	Directeur de Recherche
Philippe Depalle	Professeur

Devant la commission d'examen composée de :

Christophe Schlick	Professeur	Président et Rapporteur
Daniel Arfib	Directeur de Recherche	Examinateur
Philippe Depalle	Professeur	Examinateur
Myriam Desainte–Catherine	Professeur	Examinateur
Laurent Girin	Maître de Conférences .	Examinateur
Sylvain Marchand	Maître de Conférences .	Examinateur

– 2007 –

Contents

Remerciements

FAIRE ici la liste exhaustive de toutes les personnes qui ont participées de près ou de loin, de manière directe ou indirecte à l'élaboration de ma thèse est une tâche ardue. C'est pourquoi je prie les personnes que j'ai oubliées de bien vouloir me pardonner, en sachant qu'un bout d'elles se trouve de toute façon dans les pages qui suivent.

Tout d'abord je tiens ici à remercier mes rapporteurs Daniel Arfib et Philippe Depalle pour avoir accepté de critiquer ce document, et je veux ici exprimer toute ma gratitude à Philippe pour avoir eu l'honnêteté de faire repousser la fin de la thèse, puisqu'en fin de compte cela m'a permis de soutenir dans de bien meilleures conditions. Merci aussi à Laurent Girin pour avoir accepté d'assister à la soutenance et à Christophe Schlick d'avoir présidé le jury, mais aussi de m'avoir chacun éclairé lors de nos séances de travail.

Ensuite, un grand merci à Sylvain Marchand qui, en bon encadrant et conseiller, m'a guidé pendant ces quatre dernières années, avec une rigueur toujours impeccable, et à Myriam Desainte-Catherine, qui a su intervenir et apporter son soutien aux bons moments.

Merci aussi à Mathieu Lagrange, collègue et ami, avec qui j'ai eu le plaisir de travailler et de discuter longuement, que ce soit devant un écran d'ordinateur ou à deux mille mètres d'altitude. Merci à Vincent Verfaille pour nos (trop rares) discussions qui m'ont souvent inspirées. Merci à Robert Strandh pour m'avoir montré sa vision souvent différente des choses, vision pour le moins motivante.

Je tiens à remercier tout particulièrement Arnaud Casteigts qui a su être collègue, confident et ami, voisin de bureau et compagnon d'idéaux politiques et libristes, toujours disponible, même dans les moments difficiles. Merci aussi à Lionel Barrère pour son entrain permanent et sa tchatche sans pareil, à Julien Bernet de m'avoir toujours titillé musicalement, à Ève Atallah d'apporter une touche de féminité dans le bureau, à Achraf Karray pour ses très bons petits gâteaux, à Rémy Synave et Fabien Baldacci pour leur bonne humeur, à Matthias Robine pour sa disponibilité, à Alexandre Pinlou pour nos discussions sur l'éducation nationale, à Fabrice Bazzaro pour avoir partagé sa passion des jeux. Il y a encore beaucoup de monde au LaBRI et en dehors que je veux remercier les uns pour des discussions intéressantes, les autres pour leur disponibilité, ou même pour une toute autre raison : je pense à Mickaël Montassier, Florian Levet, Guillaume Gardey, Jérémie Chalopin, Jocelyn Fréchot, Maxime Amblard, Louis Esperet, Joan Mouba, Guillaume Meurisse, Pierre Hanna, Iban Hatchondo, Matthieu Villeneuve, Pascal Grange, François De Vieilleville, Carole Blanc, Aude Decaune, Fabienne Clairand, Ida Nerestan, Cathy Roubineau, Philippe Biais, Irène Durand, Géraud Sénizergues, Marie Aimard, Pascal Desbarrats, Abdou Guermouche, Arnaud Pêcher, Anne Dicky et tous les autres...

Ensuite, je veux remercier mon père pour m'avoir transmis le goût de la recherche, ma mère pour sa dévotion, mon frère qui m'a poussé à être meilleur.

Merci aussi à mes amis Graziella, Marie-Laure, Christian, Fab et Vincent qui voudront bien me pardonner de n'avoir pas été toujours très disponible ces trois dernières années.

Enfin, je tiens à remercier du fond du cœur ma petite femme qui, malgré la distance, m'a accompagnée et soutenue tout au long de cette épreuve et plus encore.

English Introduction

Since the ancient times of civilisation, music has attracted people. From hitting a stone on another to the use of most complex synthesisers, the production of sound by men has become an art over the centuries. For now about two centuries, the music is not only performed but also recorded and reproduced, first through mechanical means, then through digital equipments. As soon as music (and sound in general) can be captured and engraved on a support, it is also possible to modify this sound, for example by changing the speed it is being played at.

On computers, it is also possible to record and play music. With the computing power of these machines, we can create more sophisticated sound effects on the recordings. These sophisticated effects are based on a thorough analysis of the sound in order to be able to process specific components of the sound only.

Using more and more complex analysis methods allows for isolating the different components of the sound. Once these components are extracted, we can reproduce them individually. Hence, we could decide that the different parts of a piece of music would be played in a different manner than recorded originally, for example by setting louder sound volumes for a given instrument or adding a couple of verses to a song.

Those sophisticated effects are the goal of our research. The work presented here aims at proposing refinements to the existing sound models that would make such futuristic sound and music transformations possible. More specifically, the transformations such as source separation (obtaining the individual signals of the instruments of a sound mix), music transcription (finding the score from a recorded piece of music), respatialisation (moving the different sources of the sound in space), and sound and music scaling (changing the durations of sounds and music) among others are of particular interest.

The latter has been the principal motivation for our work over the last three years. Indeed, sound and music scaling is still at an early stage of its development and we think we could contribute to the improvement of this effect. When we talk about sound scaling, we mean changing the duration of a sound while keeping the sound as natural as possible, preserving the modulations of the sound if needed. Concerning music scaling, the objective is approximately the same, that is adding new notes while keeping the music coherent and preserving the rhythm.

These effects would of course have many applications, the main applications being vocal synthesisers, rapid audio-message listening, language learning, audiovisual quick search, and music creation. For example, most popular or commercial music is created using musical samples, often transformed ones. The scaling of these samples would allow the composer to be freer about the samples he could use in his music, since he could modify them to suit his needs while they remain natural (*i.e.* no artifact is added by the transformation).

Currently, it is however not possible to perform these effects without audible artifacts. The evolved algorithms until now can preserve the pitch and timbre of the sound. The modulations just begin to be taken into account and such time-scaling methods usually do not preserve the modulations intact. As for music scaling, the main research is still focused on pattern searching. In both domains, many well-working methods exist, but all are limited. What we propose as a framework is a gradual time-scaling (that is different types of scaling for different scaling ratios) and the possibility to infinitely extend a sound and/or a melody.

In our thesis, we worked on sound scaling mainly. The sounds that we are focusing on for now are mainly natural harmonic sounds (so that the frequency components are easily identifiable) with low-noise levels and slow evolutions (since the methods we use to analyse the sound rely on these assumptions). This allows the sinusoidal model we base our work on to be well adapted to the sound. As for music scaling, we focus for now on very repetitive MIDI signals, since we try to analyse them with tools designed for periodicity detection, such as the Fourier transform.

This work surely enhanced the way to perform sound scaling. We propose a method to scale the sounds while preserving sound modulations, such as vibrato and tremolo. We also defined a model for sound that can be used at different levels, first at the temporal sound level, then at the level of the parameters of the sound. This model rests on polynomials and sums of sinusoids, and outperforms both polynomial model and sinusoidal model under certain conditions. We also propose a way to correctly resample the sound parameters. We introduced the use of linear prediction in partial-tracking algorithms.

However, the results on sound-scaling have some limitations. Indeed, the estimation of sound modulation is far from perfect: the model we proposed is mainly valid for stationary parts of the sound. Thus, attacks and releases of sound are smoothed by the model we chose, because these parts of the sound are usually not stationary at all. However, we suggested a way to enhance the estimation by jointly estimating the different parameters of the sound that will be explored in the future.

Some exploratory work has been done on music scaling, searching a transform to find periodicities of music.

Our overall plan for time-scaling can thus be described as follows:

- For factors very close to 1, temporal domain resampling is sufficient.

- For small time-scaling factors, we can imagine that the sound can be scaled using order-1 sinusoidal model, since the variation in modulation rate would not be noticeable.

- For larger time-scaling factors, the order-2 sinusoidal model would take over and scale the sound by keeping both the timbre and the rate of the modulations.

- At a third level, if the time-scaling factor starts to be important and that by using order-2 the rhythm of the tune would start to decrease or increase too much for one's taste, it should be possible to keep the rhythm and be able to replicate small portions of the music directly.

- Eventually if the time-scaling factor is becoming either really small or big, it should be possible either to suppress any number chorus and verse, or add them, according to the situation.

The layout of this thesis follows that plan.

Overview

Chapter 1: Temporal modelling

The first chapter deals with the basics of sound processing. More precisely, we first introduce the temporal model of sound, which is the first model. The concepts of sampling and Nyquist frequency are explained. We also present the general idea of the Overlap-Add techniques that allow time-scaling of sound in the temporal model.

Chapter 2: Spectral modelling

In the second chapter, we show with the Fourier series that the sound, or more generally any signal, can be decomposed as a (potentially infinite) sum of sine functions. This decomposition of the sound in sinusoids can be represented on a Fourier spectrum. The Fourier spectrum then moves our topic to the spectral domain.

In the spectral domain, the sound is not represented as the amplitude of the movement of a membrane as a function of time, but it is represented as the amplitude as a function of frequency (the frequency of a given sinusoid composing the sound). This representation comes in handy as soon as sound transformation are needed. However, this representation lacks the reference to time. In order to add the time dimension, we use the short-time Fourier transform.

In this chapter, we also introduce the vocoder, which allows time-scaling of sound without alteration of the timbre and pitch of the sound. However this method lacks phase coherence in certain situations. This issue can be solved by using sinusoidal modelling.

Chapter 3: Sinusoidal modelling

McAulay & Quatieri and Serra & Smith proposed the sinusoidal model. The sinusoidal analysis of the sound extracts the main sinusoidal components of the sound (the spectral peaks of the Fourier transform) and attempts to gather them in structures called partials. These partials contain the frequency, amplitude, and phase information of the main sinusoids present in the sound, as well as their evolution over time.

We present in this chapter some enhancements to the partial-tracking algorithm (the analysis procedure to obtain the parameters of the sinusoidal model), some of these enhancements being our own.

We indeed developed a way to enhance the prediction of partials, thus allowing the partial tracking algorithm to perform better with non-stationnary frequency and amplitude tracks.

This sinusoidal model allows even more sound transformations than the spectral domain. We show how an enhanced version of the resampling algorithm we designed can be used to time-scale sounds without alteration of pitch and timbre.

Chapter 4: Order-2 spectral modelling

The next chapter introduces higher level of control over sound. Indeed, we have come to consider at this point that partials are in fact control signals of the sound. As signals, they can be modelled in the same fashion as any signal, using either polynomial or sinusoids.

After studying both possibilities, each having advantages and drawbacks, we decided to design a model based on both sinusoids and polynomial in order to obtain the best of each method.

Hence, we can analyse those signals, obtaining what we call order-2 spectral modelling. We now have a time-frequency representation of the partials. This gives interesting information that can be applied in a variety of manners.

The first application is an enhancement to the partial tracking algorithm. Considering the idea that partial trajectories have to be "smooth", it is possible to discriminate partials peaks that would undermine this hypothesis.

A second application is a similarity measure of partials based on the Fourier transform of the frequencies and amplitudes of the partials. Hence, two partials that have the same source will probably have very similar modulations, and the Fourier transforms of those two partials would be strongly correlated.

This order-2 spectral modelling suffers however some limitations similar to spectral modelling, and a sinusoidal model at order-2 could comes in handy, thus allowing time-scaling of sounds preserving pitch, timbre, and modulations of the partials (such as vibrato and tremolo).

Chapter 5: Order-2 sinusoidal modelling

In the same way we built partials from short-time Fourier transform, it is possible to build order-2 partials from the short-time Fourier transform of the parameters of the (order-1) partials.

Thus, in the same way we performed time scaling on order-1 (classic) sinusoidal parameters, we perform it on order-2 sinusoidal parameters. First we scale the frequency and amplitude parameters of the sound. Then, we base the scaling on the phase and amplitude parameters, using the model we designed, which is based on polynomials and sums of sines. This approach however suffers the same limitation as sinusoidal modelling, even with higher intensity at order 2: the attack of the signal is smoothed and the sound is supposed to be quasi-stationary. Hence we achieve a first part of the artifact-free time-scaling effect since the sound is scaled with conservation of the timbre and vibrato and tremolo.

Chapter 6: Higher order modelling

In this chapter, we attempt to reach an even higher level: the note level of songs. However the modulations of the melodies do not seem to be formed in the same way as low-level partials, and thus we would need a new analysis method in order to find periodicities. Thus, we open the way to the creation of a new basis in order to transform the sound and gain knowledge about the structure of a melody that would allow us to perform time-scaling of sounds at a higher level. This has not been successful since we did not find a suitable basis for musical structures.

Chapter 7: The Clicks software

Eventually, we present the software program developed to process sounds and apply the research we have been carrying.

This piece of software is thus capable of performing sinusoidal analysis at orders 1 and 2, using the different enhancements presented in this thesis.

Moreover it is aimed at becoming an audio processing platform written in Common Lisp, thus allowing direct interaction through a prompt and high-level programming. It also has a small graphical user-interface for visualising data.

Introduction en Français

Depuis les temps immémoriaux des débuts de la civilisation, la musique a intrigué les hommes. À travers les siècles, du simple choc entre deux pierres aux synthétiseurs les plus complexes, la production de son par l'homme est devenu un art. Depuis maintenant environ deux cent ans, la musique est même passée du stade de la performance seule à l'enregistrement et le reproduction, d'abord de manière mécanique, puis plus récemment de manière numérique. Cet enregistrement a ouvert de nouvelles perspectives, puisqu'il est dès lors possible de modifier ce son, par exemple en changeant sa vitesse de lecture.

Sur les ordinateurs, il est aussi possible d'enregistrer et de jouer de la musique. Avec leur puissance de calcul, il est possible de créer des effets sophistiqués sur ces enregistrements, basés sur une analyse minutieuse du son pour pouvoir n'en manipuler que quelques composantes seulement.

L'utilisation de méthodes d'analyse de plus en plus complexes permet d'isoler les différentes composantes du son, pour par exemple les restituer individuellement. Nous pourrions aussi décider que les différentes parties d'un morceau de musique soit jouées différemment de l'original, par exemple en changeant le volume d'un instrument donné, ou en ajoutant un ou deux couplets à une chanson.

Ces effets sophistiqués sont le but de notre recherche. Le travail présenté dans ce document a pour objectif d'apporter des précisions aux modèles sonores existants de manière à rendre possible ces transformations sonores et musicales futuristes. Plus précisément, les transformations telles que la séparation de source (obtenir les signaux individuels de chaque instrument d'une pièce de musique), la transcription musicale (retrouver la partition d'un morceau de musique enregistré), la respacialisation (déplacer les différentes sources sonores dans l'espace) et la l'étirement du son et de la musique (changer la longueur du son et de la musique) entre autres sont particulièrement intéressantes.

Ce dernier effet à été notre principale motivation pendant ces trois dernières années. En effet, l'étirement du son et de la musique n'en sont encore qu'à leurs prémices et nous pensons que nous pouvons contribuer à l'amélioration de cet effet. Nous entendons par étirement du son le changement de durée du sons tout en conservant au maximum l'aspect naturel du son, c'est-à-dire en préservant les modulations quand nécessaire. Pour l'étirement de la musique, l'objectif est à peu près le même, c'est-à-dire qu'on ajoute de nouvelles notes tout en préservant la cohérence de le musique et le rythme.

Ces effet ont bien sûr moultes applications, les principales étant les synthétiseurs vocaux, la lecture rapide de messages audio, l'apprentissage des langues, la recherche rapide dans les contenus audiovisuels et la création musicale. Par exemple, la majorité des musiques populaire ou commerciale actuelle est crée en utilisant des échantillons musicaux, souvent transformés. L'étirement de ces échantillons donnerait au compositeur une liberté accrue dans leur utilisation puisqu'il pourrait les modifier à son goût tout en restant naturels (c'est-à-dire qu'il n'introduirait pas d'artefacts dûs à la transformation).

Pourtant actuellement, il n'est pas possible de produire ces effets sans artefacts audibles. Les algorithmes évolués d'aujourd'hui préservent la hauteur et le timbre du son, mais les modulations, qui commencent à être prises en compte, ne sont pas encore préservées intactes. En ce qui concerne l'étirement de la musique, la recherche se base sur la détection de motif. Dans les deux domaines, beaucoup de méthodes existent, mais elles ont toutes des limitations. Ce que nous proposons comme cadre dans ce travail est une approche échelonnée de l'étirement temporel (c'est-à-dire différents types d'étirements pour différents facteurs d'étirement) et la possibilité d'étirer un son ou une mélodie à l'infini.

Dans cette thèse, nous acons travaillé principalement sur l'étirement du son, en particulier sur les son naturel harmoniques (de telle sorte que les différentes composantes fréquentielles soient facilement identifiables) peu bruités (puisque les méthodes que nous utilisons requièrent de telles conditions). De cette manière, le modèle sinusoïdal, sur lequel repose notre travail, sera bien adapté aux sons. Pour ce qui est de l'étirement de la musique, nous nous concentrerons sur des signaux MIDI très répétitifs, puisque nous les analyserons avec des outils de détection de périodicité, telle que la transformée de Fourier.

Ce travail améliore en effet l'étirement temporel. Nous proposons en effet une méthode d'étirement du son avec préservation des modulations telles que le vibrato et le trémolo. Nous définissons aussi un modèle du son qui peut être utilisé à différents niveaux : d'abord au niveau temporel du son, puis au niveau des paramètres du son. Ce modèle repose sur des polynômes et des sommes de sinusoïdes, et donne de meilleurs résultats que les modèles polynomial et sinusoïdal dans certaines conditions. Nous proposons aussi une méthode de rééchantillonnage correct des paramètres du son, ainsi que l'utilisation de le prédiction linéaire dans l'algorithme de suivi de partiels.

Toutefois, les résultats de l'étirement du son ont quelques limitations. En effet, l'estimation des modulations du son n'est pas parfaite : le modèle proposé est valide pour les parties stationnaires du son, et par conséquent les attaques et les relâchements sont lissés par le modèle choisi, puisque ces parties du son ne sont pas stationnaires. Cependant, nous indiquons une méthode d'estimation conjointe des paramètres du son que nous l'intention d'investiguer prochainement.

Un travail exploratoire sur l'étirement de la musique est aussi présenté, basé sur la recherche d'une transformée détectant les périodicités de la musique.

Notre vision graduelle de l'étirement temporel peut donc être décomposé comme suit :

- Pour les facteurs d'étirement proches de 1, le rééchantillonnage dans le domaine temporel est suffisant
- Pour des petits facteurs d'étirement, nous utilisons le modèle sinusoïdal d'ordre 1, puisque les variations des modulations du sont ne sont pas remarquables.
- Pour des facteurs d'étirement plus grands, le modèle sinusoïdal d'ordre 2 prend le relais et effectue l'étirement en préservant le timbre et la vitesse des modulations.
- À un troisième niveau, si le facteur d'étirement devient trop important, c'est-à-dire qu'en utilisant un étirement à l'ordre 2 le rythme diffère trop de l'original, nous

devons être en mesure de préserver le rythme en répliquant directement de portions de musique.

- Finalement, si le facteur d'étirement deviens démesuré, il doit être possible de supprimer ou d'ajouter un nombre arbitraire de refrain ou de couplets suivant la situation.

Le document de thèse est structuré comme suit.

Plan du mémoire

Chapitre premier : Modélisation temporelle

Ce premier chapitre traite des bases du traitement du son. Plus précisément, nous introduisons d'abord le modèle temporel du son, ce qui constitue le premier modèle. Les concepts de l'échantillonnage et de la Fréquence de Nyquist sont présentés, ainsi que les techniques d'Overapp-Add qui permettent l'étirement temporel du son dans le modèle temporel.

Chapitre second : Modélisation spectrale

Dans le second chapitre, nous utilisons les séries de Fourier pour montrer que le son, ou plus généralement tout signal, peut être décomposée en une somme potentiellement infinie de fonctions sinusoïdales. Cette décomposition du son peut être représentée par un spectre de Fourier. C'est ainsi que nous entrons dans le domaine spectral.

Dans ce domaine, le son n'est pas représenté par l'amplitude du mouvement de la membrane en fonction du temps, mais par l'amplitude en fonction de la fréquence (la fréquence de d'une sinusoïde donnée composant le son). Cette représentation facilite généralement les transformations du son. Cependant, cette représentation ne fait pas référence au temps, ce que l'on compense en utilisant la transformée de Fourier à court terme.

Dans ce chapitre, nous introduisons aussi le vocoder qui permet l'étirement temporel du son sans altération du timbre et de la hauteur. Malheureusement, cette méthode manque de cohérence de phase dans certaines situations. C'est pourquoi nous présentons ensuite le modèle sinusoïdal.

Chapitre troisième : Modélisation sinusoïdale

McAulay & Quatieri et Serra & Smith ont proposé le modèle sinusoïdal. L'analyse sinusoïdale du son extrait les composantes sinusoïdales principales du son (les pics spectraux de la transformée de Fourier) et les assemble dans la mesure du possible en structures appelées partiels. Ces partiels contiennent les informations de fréquence, amplitude et phase des principales sinusoïdes présentes dans le son, ainsi que leur évolution au cours du temps.

Nous présentons dans ce chapitre quelques améliorations apportées à l'algorithme de suivi de partiels (qui est la procédure permettant d'obtenir les paramètres du modèle sinusoïdal), quelques-unes étant de notre cru.

En effet, nous avons développé un moyen d'améliorer la prédiction des partiels, permettant donc un meilleur suivi des partiels non-stationnaires en fréquence et en amplitude.

Ce modèle sinusoïdal permet encore plus de transformations sonores que le modèle spectral. Nous montrons ici comment une version améliorée de l'algorithme de rééchantillonnage que nous avons conçus peut être utilisé pour étirer des sons sans modification de la hauteur ni du timbre.

Chapitre quatrième : Modélisation spectrale d'ordre 2

Le chapitre suivant introduit des niveaux supérieurs de contrôle sur le son. En effet, nous somme amenés à ce point à considérer que les partiels sont en fait des signaux de contrôle du son. En tant que signaux, ils peuvent êtres modélisés de la même manière que n'importe quel autre signal, en utilisant soit des polynômes, soit des sinusoïdes.

Après l'étude de ces deux possibilités, chacune ayant ses avantages et ses inconvénients, nous décidons de créer un modèle basé sur les polynômes et les sinusoïdes en même temps pour obtenir le meilleur des deux méthodes.

Nous pouvons donc analyser ces signaux, obtenant ce que nous appelons le modèle spectral d'ordre 2. Nous avons donc maintenant une représentation temps-fréquence des partiels. Cela produit des informations intéressantes qui peuvent être exploitées de plusieurs façons.

La première application est une amélioration de l'algorithme de suivi de partiels. Prenant compte du fait que les trajectoires de partiels doivent être "douces", il est possible de discriminer les pics (des partiels) qui saboteraient cette hypothèse.

Une seconde application est une métrique de similarité de partiels basée sur la transformée de Fourier des fréquences et des amplitudes des partiels. Deux partiels ayant la même source auront donc probablement des modulations très similaires, et les transformées de Fourier de ces deux partiels seront fortement corrélées.

Cette modélisation spectrale d'ordre 2 souffre cependant de quelques limitations de la modélisation spectrale, et c'est pourquoi un modèle sinusoïdal d'ordre 2 peut sembler intéressant, permettant donc des étirements de sons conservant la hauteur, le timbre et les modulations des partiels (telles que le vibrato et le trémolo).

Chapitre cinquième : Modélisation sinusoïdale d'ordre 2

De la même manière que nous avons construits des partiels à partir de la transformée de Fourier à court terme, il est possible de construire des partiels d'ordre 2 à partir de la transformée de Fourier à court terme des paramètres des partiels (d'ordre 1).

Par conséquent, de la même manière que nous avons étiré le son à l'ordre 1, nous étirons les paramètres sinusoïdaux à l'ordre 2. Tout d'abord nous étirons les paramètres de fréquence et d'amplitudes du son, puis nous basons l'étirement sur les paramètres de phase et d'amplitude, en utilisant le modèle que nous avons créé, lui-même basé sur les polynômes et les sommes de sinus. Cependant, cette approche souffre des mêmes limitations que le modèle sinusoïdal d'ordre, avec encore plus d'intensité à l'ordre 2 : l'attaque du signal est lissée puisque le son est supposé quasi stationnaire. Pourtant nous concrétisons ici la première partie de l'effet d'étirement temporel sans artefact puisque le son est étiré en conservant le timbre, la hauteur, le trémolo et le vibrato.

Chapitre sixième : Modélisation d'ordre supérieur

Dans ce chapitre, nous tentons d'atteindre un niveau supérieur : le niveau des notes d'une chanson. Pourtant les modulations des mélodies ne semblent pas être formées de la même manière que les partiels de bas niveau, et donc il nous faudrait une nouvelle méthode d'analyse pour trouver des périodicités. Par conséquent, nous ouvrons la voie de la création d'une nouvelle base de façon à transformer le son et trouver la structure d'une mélodie qui nous permettrait de faire de l'étirement temporel à un niveau supérieur. Cela n'a pas été concluant puisque nous n'avons pas trouvé de base appropriée pour les structures musicales.

Chapitre septième : Le logiciel Clicks

Enfin, nous présentons le logiciel développé pour traiter les sons et appliquer la recherche menée durant ces trois années.

Ce logiciel est donc capable de faire des analyses sinusoïdales d'ordres 1 et 2, en utilisant les différentes améliorations proposées.

De plus, l'objectif est d'obtenir un plate-forme de traitement audio écrite en Common Lisp, donc dédiée à l'interaction directe par une ligne de commande et de la programmation de haut niveau. Il propose aussi une interface graphique simple pour visualiser les données.

Chapter 1

Temporal Modelling

The physical phenomenon that human beings perceive as sound is made of pressure variations of the air. As an old dream of many early scientists, the recording and reproduction of a sound was indeed the science of capturing and reproducing the pressure variations of the air. The capture is done by recording the positions of the membrane of a microphone over time, and the reproduction displaces the membrane of a loud-speaker using the positions of the membrane of the microphone previously recorded. Hence, wanting something more elaborate than an elementary amplifying set, the scientists were faced with the problem of storing the positions of the membrane over time.

The first solution was to carve the displacement of the centre of the membrane into wax, which is equivalent to recording the position of the membrane over time, or to store the variations of air pressure around the microphone over time. This storage system was (and still is) a quite accurate way to record sounds, since it keeps track of the continuity of the air pressure variations.

However the wax solution – and their vinyl reproductions – have a fast deterioration rate that is not acceptable to modern standards. Moreover, the digital recording is now the standard when it comes to high quality recording.

This chapter will deal with the digital counterpart of the wax carving solution called the temporal model.

1.1 Temporal model

Recording the fluctuations of air pressure through the displacement of the membrane of a microphone over time can be translated into mathematical form as a function of time. This function will be called a *time signal* as it is a mathematical function that carries information of the *amplitude* of the movement of the membrane over time. This representation of sound as a time signal is called the *temporal representation*.

We will consider in this document that the sound signal is a continuous function (and thus defined on a dense set). Indeed, our assumption is that the instantaneous displacement of the membrane of a microphone is not possible in nature, just as it is not possible to have instantaneous changes in air pressure. Even when very sudden, a change of position can only have a transition time different than zero. Thus the natural sounds we base our study on will be considered as continuous.

Capturing the continuous form of the sound in a continuous signal would ensure that the reproduction of the sound is perfect. Indeed, all the variations of the amplitude of the sound signal are recorded, and at any time the amplitude value can be retrieved.

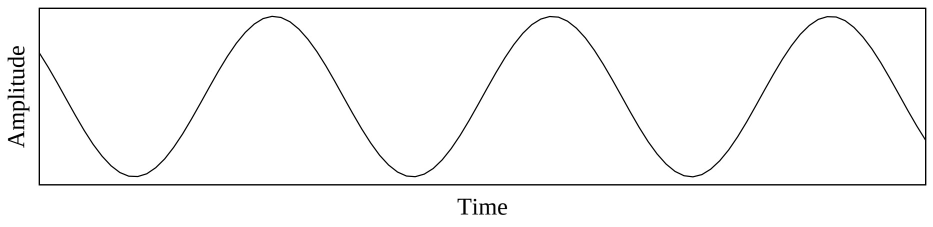


Figure 1.1: *A continuous-time signal.*

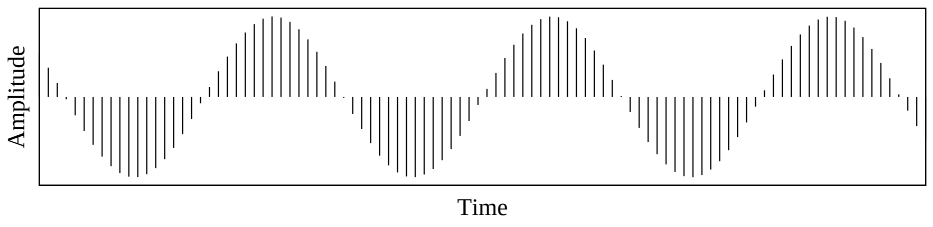


Figure 1.2: *A discrete-time signal.*

This continuous form, as approximated by the recording on vinyl and wax, is not recordable on digital equipment. Indeed storing a finite continuous signal as is – *i.e.* without any knowledge of its model – is not possible, because it would require an infinite space. Indeed, in the case where the model of the signal we want to record is unknown, the only solution to have a perfect recording of a continuous signal is to record all the values of the signal. However, since the time is continuous (meaning the (continuous-time) signal is defined on a dense set), the quantity of amplitude values to record would be infinite. Thus a recording of a continuous-time signal on digital equipment would require an infinite storage space on the digital support.

Hence, on digital equipment, only a finite number of values are recorded. This process is called *sampling*. This means that values of the signal are taken at given times in order to have a *discrete-time* signal instead of the continuous-time signal.

We will consider in this document that the sound signals we work on are finite. Thus, the sampling of a finite continuous-time signal gives a finite discrete-time signal, represented by a finite-length vector of values.

When it comes to digital sound, the sampling is often uniform, meaning that the values are taken at evenly spaced time intervals. The sampling is then defined by the *sampling period*, herein noted T_s. We can also define the *sampling frequency* as the inverse of the sampling period: $F_s = T_s^{-1}$

The sampling of a sound might however induce a quality loss of the digitally recorded sound. Indeed, even if it is possible to rebuild a continuous signal from the sampled sound, the rebuilt signal may not contain all the elements from the original signal. As a matter of fact, the sampling frequency of the signal defines the maximum frequency of the components present in the sound. This is defined by the Nyquist condition, that states that the maximal frequency present in a discrete signal is twice lower than the sampling frequency:

$$F_N = \frac{F_s}{2} \tag{1.1}$$

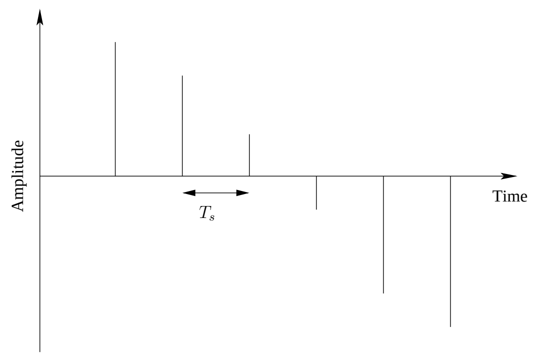


Figure 1.3: *The sampling period of the sinusoidal signal shown on Figure 1.2. A zoom has been performed for clarity sake.*

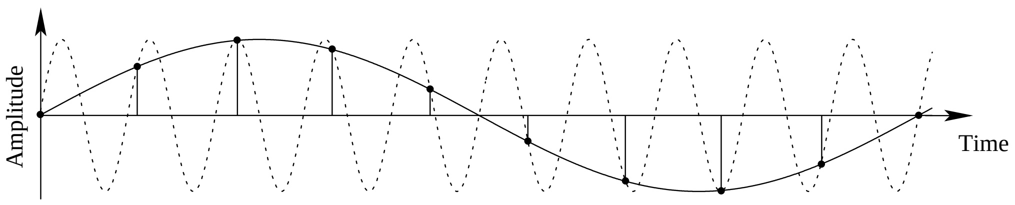


Figure 1.4: *Sampling a signal above the Nyquist frequency. The dashed line shows the original signal. The impulses show the sampled form of this signal. The solid line shows the reconstructed form of the signal, considering that the sampled signal had a frequency lower than the Nyquist frequency, which thus gives the wrong signal.*

F_N is the maximum frequency of the signal and is called the *Nyquist frequency*. In the case of a signal containing components at a frequency higher than F_N being discretised (meaning that a continuous-time signal is converted to a discrete-time signal) without prior treatment, the sampling will introduce ambiguities. Figure 1.4 shows how the sampling of a high-frequency sinusoidal component (dashed-line) will be rebuilt as a lower frequency sinusoidal (solid line) in a discrete-time signal. Hence in order to avoid such low frequency artifacts to be introduced in the discrete-time signal, the signals to be sampled should first be band-limited by F_N.

This band limitation means that the high-frequency components (higher than F_N) will not be present in the sound. In order to have a digital sound that is not perceptually different from the original sound, we have to take into account the perceptive capabilities of human beings. It has been measured that the adult human is capable of hearing frequencies up to 16000 Hz. Thus, the band-limitation should be done at least at this frequency, and this frequency will be the Nyquist frequency of our discrete-time signal. From the Nyquist frequency, it is then easy to compute the minimum sampling frequency of our signal.

Indeed, since the human adult can hear frequencies up to 16000 Hz, the corresponding sampling frequency necessary for the signal to describe the frequency content up to 16000 Hz is at least 32000 Hz (twice the highest frequency we want to be able to represent). Thus, in this document, we will usually consider a sampling frequency of 44100 Hz. This value is the sampling frequency that is used on Compact Discs (CDs). This value is high enough to fool the human auditory system, since it is higher than 32000 Hz. The fact that the CDs are sampled at 44100 Hz and not at 32000 Hz may come from the fact that the signal recorded on the CD has to be low-pass filtered before it is sampled and reconstructed, and since the frequency response of the low-pass filter is not perfectly square in shape, the filter used is almost perfect until 16 kHz, then it filters the higher frequencies until effectively cutting the signal frequencies at 22050 Hz. Thus, the sampling frequency of CDs is twice 22050 Hz, that is 44100 Hz.

The sampling on discrete-time signal is done in order to overcome the fact that time-dependent signals are defined on a dense set (thus a set composed of an infinite number of values). However the amplitude values of the sound are also defined on a dense set, which might be a problem when digitally recording a sound. Indeed, the values from a real signal are not representable by a computer, since a computer would require finite length precision for real numbers. Hence, the amplitude values of the signal have to be rounded, or quantised. A common practice, also used in CDs, is to quantise the amplitude values on 16-bit numbers, which represents 65536 signed values. This means that the quantisation error is -96 decibels (dB). Even if this error is still audible by the human auditory system (a value blow -120 dB would have been necessary), it is small enough for the humans to consider recordings quantised at 16 bits per sample to be good enough in comparison to the original.

The amplitude values will be represented as floating numbers throughout the remainder of this work, and we will consider only discrete-time signals (where only the time values are sampled, as opposed to discrete signals which are both sampled and quantised). This assumption is (according to us) acceptable since the quantisation of the amplitudes will not have any consequence on the way we perform analysis and transformations in this document, as opposed to the importance of sampling, which conducts from the start the way to handle sounds in our experiments. Moreover, on computers we work with 64-bits floating-number arithmetic (inaudible quantisation error), the 16-bit quantisation being done when exporting the sound files.

1.2 Time-scaling in the temporal model

The time-scaling of sound has been studied in the temporal model for a long time. The first attempt of time-scaling is of course based on the variation of the speed of recorded material. Many other methods exist, but we will present here only a few of them, since they are all related and based on the same principles.

1.2.1 Varying the playing speed

The first method for time-scaling of sound must have been discovered as soon as the first recordings and playback devices have been build. Indeed, in order to change the duration of a sound, the simplest solution is to change the playing speed accordingly. Of course, since frequency and time of sounds are strongly related ($F = 1/T$), the main drawback of this is the change in frequency and then in timbre of the sound. However, DJ's still use this technique in a musical fashion on vinyl records called "scratch".

1.2.2 Overlap-Add (OLA)

In order to avoid the change in frequency and timbre of sound, a method based on the temporal model has been designed and explained by Fairbanks *et al.* as soon as in 1954 [FEJ54]. The principles of this method is to either duplicate or remove some parts of the sound in order to change the length of the sound. The machine designed at the time by Fairbanks *et al.* is based on magnetic tape records and a revolving drum of four playback heads. The way the scaling is performed is illustrated in Figures 1.5 to 1.6 on the next page.

When the drum does not move and the tape moves at a normal speed, the recorded sound is played as-is. In the case where the drum rotates "following" the tape, with a relative speed being equal to the normal reading speed, the sound is compressed since some parts of the tape are not played. In the other case, the sound is expanded since some parts of the sound are played twice. When the relative speed of the tape and the revolving drum is either greater or lesser than the normal reading speed, the pitch of the sound is (respectively) either upshifted or down-shifted. The fact that the playback heads are not instantaneously switched on or off but rather slowly getting closer or farer from the tape creates fades-in and fades-out on the sound extracts being played, thus adding the overlapping sounds at the segment boundaries. Hence this method is called the *Overlap-Add* or *OLA* method.

This method has the advantage of preserving the pitch of the sound. However this method can lead to the repetition or the deletion of transients. Morever, the mixing of out-of-phase signals can create artifacts during playback (Illustrated in Figure 1.7 on page 19).

Hence, joining the segments with more care would enhance significantly this time-scaling method. This is the objective of the SOLA and TD-PSOLA methods.

1.2.3 SOLA and TD-PSOLA

The objective of the SOLA [RW85] and TD-PSOLA [ML95] methods is to avoid the phase problems at frame boundaries. In order to do this, the SOLA (Synchronous OLA) method computes the best location for fade-in and fade-out using a correlation function of the signal (See Figure 1.8 on page 19). The resulting sound is better, but the phase problem is not entirely solved for non-harmonic and non-monophonic sounds.

The TD-PSOLA (Time-Domain Pitch Synchronous OLA) is an enhancement to the SOLA since it detects precisely the fundamental frequency of the voiced part of a speech signal. Thus, it cuts and splices the temporal signal very precisely, keeping the pieces coherent in phase. This gives even better results than SOLA, but only for harmonic and monophonic sounds. The transients of the scaled signals may however be duplicated or deleted.

1.3 Conclusion

In this chapter, we have presented the basic representation of sound, that is the amplitude of the pressure variation as a function of time. In its digital form this function is sampled,

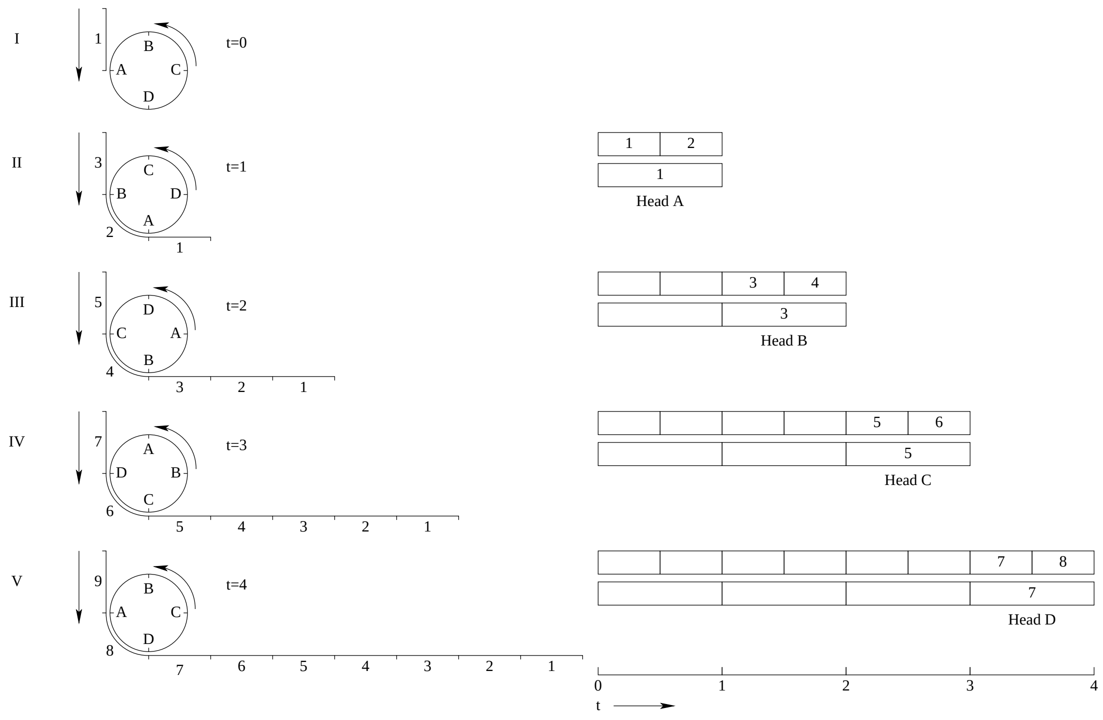


Figure 1.5: *Sound compression on tape recordings. Here, the compression rate is 0.5. Every second segment is skipped in order to shorten the length of the sound.*

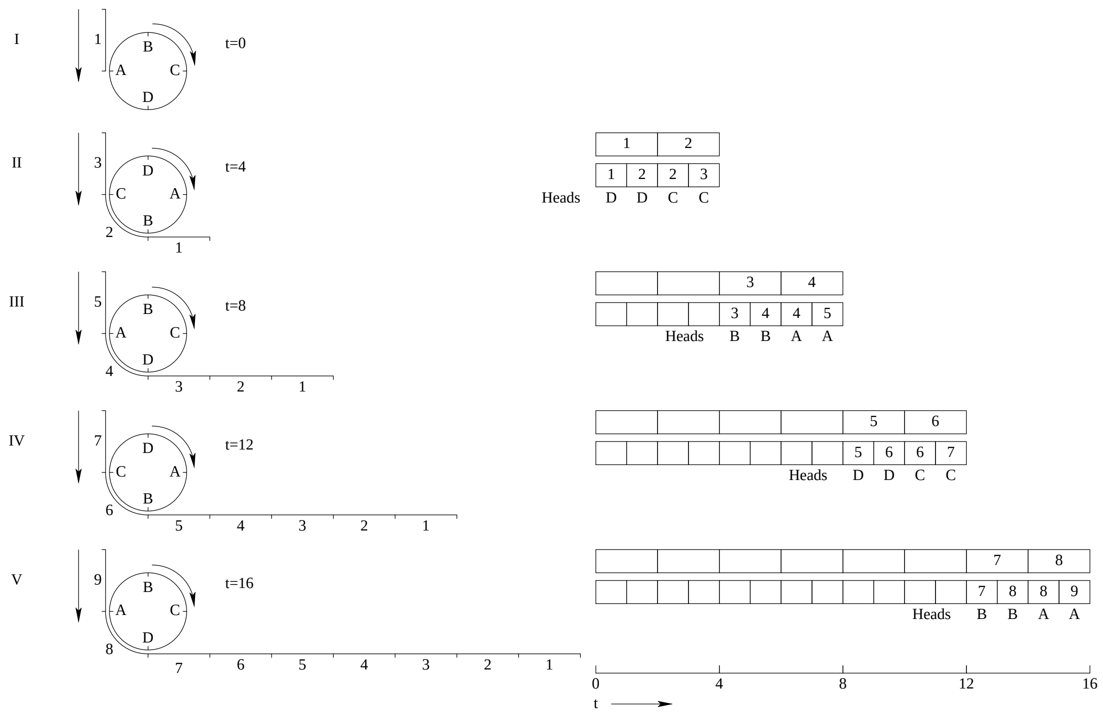


Figure 1.6: *Sound stretching on tape recordings. The stretching rate is 2. Every sound segment (except the first) is duplicated in order to lengthen the sound.*

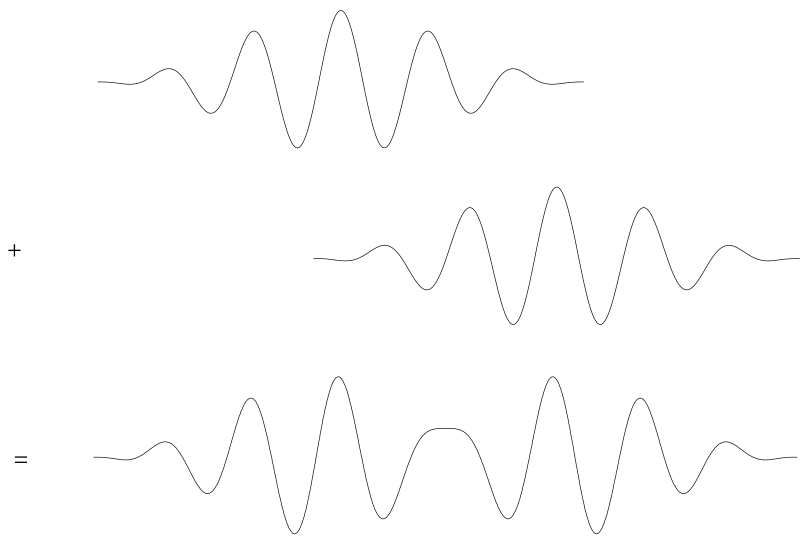


Figure 1.7: *Performing OLA in the standard way. If the added signals are out of phase, the resulting signal has not much to do with the original. Hence, artifacts might appear.*

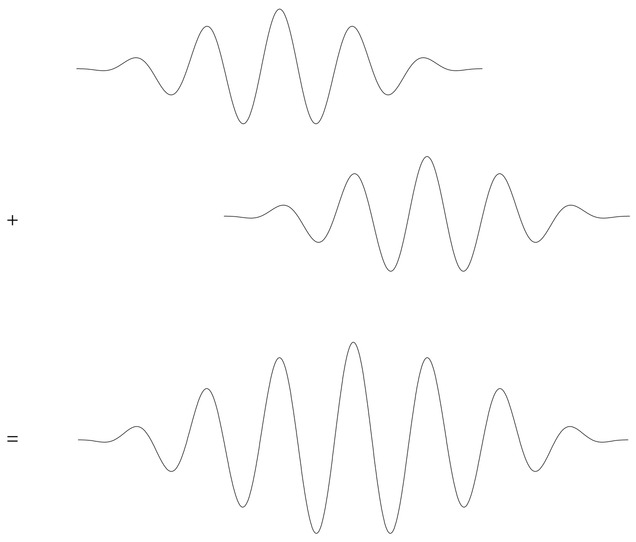


Figure 1.8: *Using SOLA or TD-PSOLA, the signals are put in phase before being added. The resulting signal is much more consistent with the original signal.*

and the highest frequency that can be contained in the sound signal is dependent on the sampling rate.

We have also presented methods to perform time-scaling – based on the OLA technique – that preserve the pitch of the sound. These technique are of great interest, but at high scaling factors, the transient suppression or deletion problem arises. Moreover, they are designed for harmonic and monophonic sounds, while we want a method for music, comprising inharmonic and polyphonic sounds. This is thus not acceptable for an artefact-free time-scaling method.

Chapter 2

Spectral Modelling

The temporal representation has been used since the beginning of sound recordings. This representation has limitations, among which the fact large amounts of data is needed for the storage of sounds containing complex frequency components. Moreover, transformations in the temporal model are quite limited, particularly for artifact-free sound scaling.

Artifact-free sound scaling of polyphonic music is a major objective of the work presented here.

Thus, the first step is to find a model that makes this transformation easy. The time representation not being the best way to perform these kinds of transformations, we will now have a look at the frequency representation of signals. Spectral models, and subsequently sinusoidal models, use this frequency representation and might thus be appropriate for such transformations.

2.1 The Fourier transform

The Fourier transform was named after the French mathematician and physicist Jean-Baptiste Joseph Fourier (Auxerre, 1768 - Paris, 1830, see Figure 2.9 on the following page). He claimed that any function could be expressed as a (potentially infinite) sum of sinusoids. However, this theory – called the *Fourier series* – is not generally true. In its original form, it was designed to work only for *periodic functions*. However, it has been adapted to non-periodic functions. This adaptation to non-periodic functions is called the *Fourier transform*.

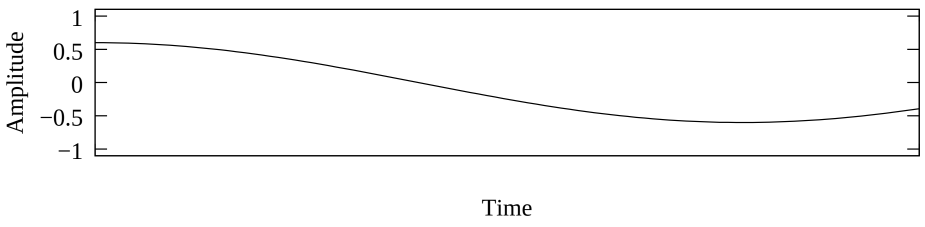
Hence, the Fourier transform of a continuous function expresses this function in terms of sinusoidal basis functions, that is to say as a sum (integral) of sine (complex exponential) functions multiplied by some coefficients called *amplitudes*. These amplitude coefficients, along with the *frequency* and the *phase*[1] parameters entirely characterise a sinusoid.

The intuition of this transform is to correlate the signal with a set of complex exponentials at various frequencies, and thus to obtain the amplitude response and phase for each exponential. Figure 2.11 on page 23 shows the Fourier decomposition of the signal pictured in Figure 2.10 on the following page.

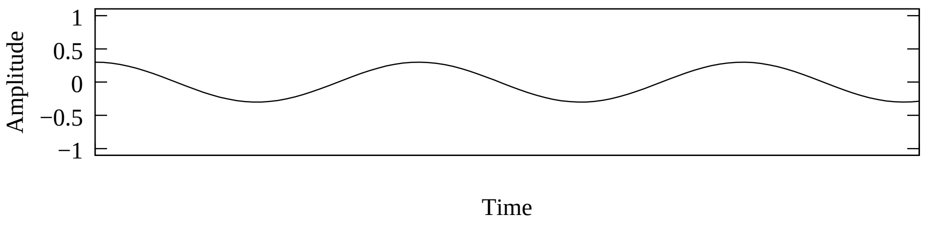
In the spectral model, a signal is indeed represented as a sum of sinusoids. Each sinusoid is defined at a given time by its frequency, amplitude, and phase parameters. This model is based on the Fourier theorem which states that a periodic signal can be

[1]The phase is related to the temporal shift of a sinusoidal signal.

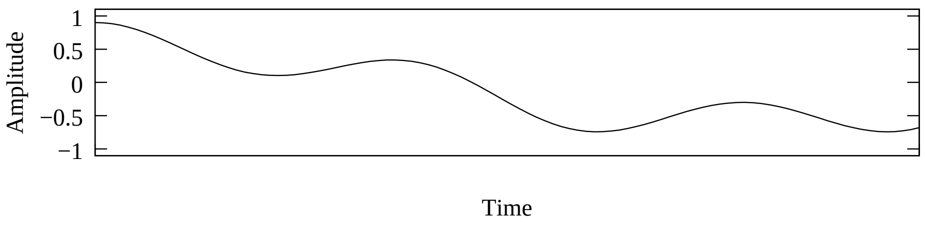
Figure 2.9: *Jean-Baptiste Joseph Fourier [Soc40].*



(a) First sinusoid.



(b) Second sinusoid.



(c) Sum of the two preceding sinusoids.

Figure 2.10: *A signal made of two harmonically related sinusoids.*

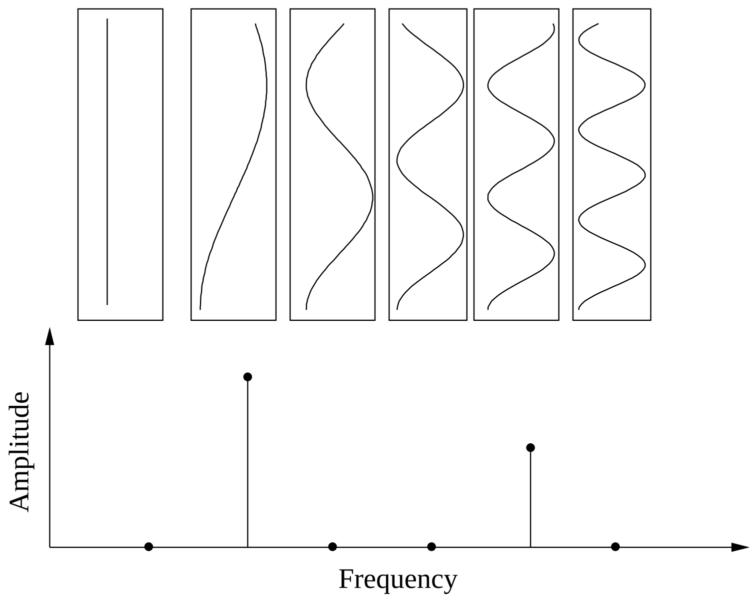


Figure 2.11: *Intuition for the computation of a Fourier spectrum. The signal shown in Figure 2.10(c) is in turn multiplied (using the dot product) by each function of the set of sinusoids (here represented in boxes). In this case, the sinusoids of the analysed signal correspond exactly to the frequency of two sinusoids from the set of sinusoids, and thus, since two sinusoids compose the original signal, two bins are filled in the Fourier spectrum, one for each sinusoid of the signal. For clarity sake, what we have shown here is actually a Fourier series.*

modelled as a sum of sinusoids of given amplitudes and harmonically related frequencies[2].

The Fourier transform of a continuous-time signal s is then expressed as follows:

$$S(\omega) = \int_{-\infty}^{+\infty} s(t)e^{-i\omega t}dt \tag{2.2}$$

S is called the Fourier spectrum of the signal. i is the imaginary unit: $i^2 = -1$. ω is the frequency and is given in radian per second (rad.s^{-1}). The relation between ω and f (the frequency in Hz) is:

$$\omega = 2\pi f$$

We can also notice that:

$$e^{-i\omega t} = \cos(\omega t) - i\sin(\omega t)$$

Hence these correlations of the analysed signal with $e^{-i\omega t}$ give complex values (out of cosinus for the real part and sinus for the imaginary part). A Fourier spectrum is shown on Figure 2.12 on the next page. On this Figure, we can see the real and imaginary parts of the complex spectrum. The Fourier spectrum is complex due to the fact that the Fourier transform uses complex exponentials to compute the spectrum and in our case the signals we analyse are real ones. We can also see some properties of the Fourier transform on this figure: For real signals, the real part of the Fourier transform is even and the imaginary part is odd.

The complex values of the Fourier transform can then be expressed in polar form as:

$$z = ae^{i\phi}$$

with a being the modulus of the complex number, herein called *amplitude*, and ϕ being the angle of the complex number, herein called *phase*.

Thus for each given frequency, we have a couple of values: amplitude and phase, which are the parameters of the complex exponentials present in the signal. This representation of the signal is a frequency-domain representation since the values of amplitude and phase are frequency dependent.

2.2 Discrete case

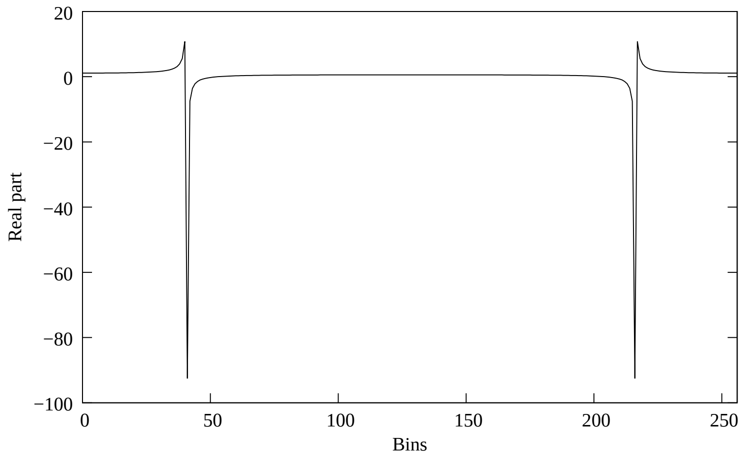
The previously defined Fourier transforms are valid for continuous-time signal. In the case of discrete-time signals, the Fourier transform becomes:

$$S(\omega) = \sum_{m=-\infty}^{+\infty} s[m]e^{-i\omega m} \tag{2.3}$$

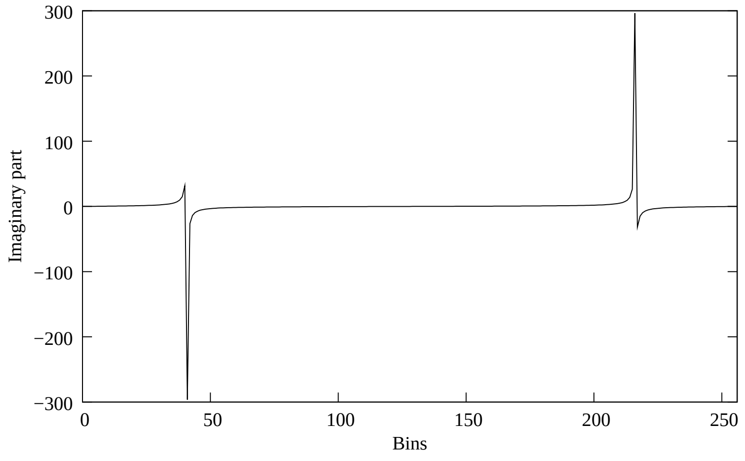
where ω is still the continuous frequency (given in rad.s^{-1}) and $s[n]$ is a sampled signal such as

$$s[] : \mathbb{Z} \to \mathbb{R}$$

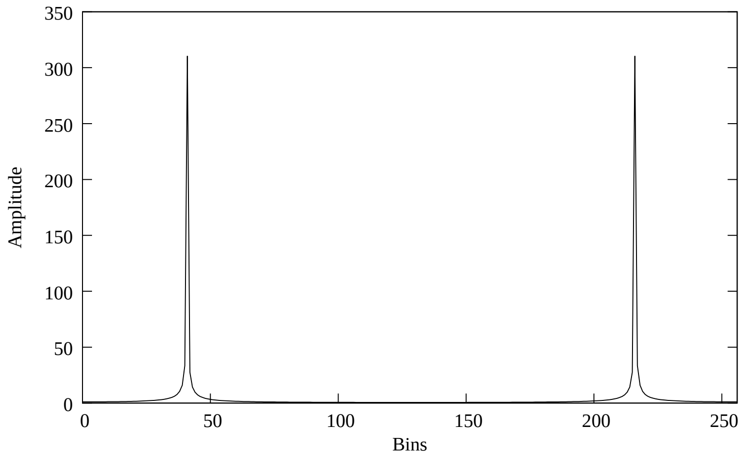
[2]Harmonic frequencies are frequencies that are linked by an integer proportionality relationship with a base frequency called *fundamental frequency*. For example, if f_0 is the fundamental frequency, then $f_1 = 2f_0$, $f_2 = 3f_0$ are harmonic frequencies of f_0, and f_0,f_1, and f_2 are harmonically related.



(a) Real part of the Fourier spectrum.



(b) Imaginary part of the Fourier spectrum.



(c) Amplitude Fourier spectrum.

Figure 2.12: *The discrete Fourier spectrum of a single sinusoid. We can see that the bin is replicated on both sides of the spectrum (even if inverted for the imaginary part of the spectrum). This is due to the fact that a single sinusoid is decomposed in a sum of two complex exponentials.*

$$s() : \mathbb{R} \to \mathbb{R}$$

and

$$s[n] = s\left(\frac{n}{F_s}\right)$$

F_s is the sampling frequency (given in Hz). Since the signal is discrete-time, the relation between ω and f is now:

$$\omega = \frac{2\pi f}{F_s}$$

and ω is defined on $[-\pi, \pi]$.

In our work, we deal with discrete-time signals that are finite, and thus the Fourier transform has to be adapted. Let us now consider $s[n]$ as being a finite sampled signal, $s[n]$ can then be written as: $s : E \subset \mathbb{Z} \to \mathbb{R}$, where E is a subset of $\mathbb{Z}$. Let us now consider a part of $s[n]$ described by the values $s[0], s[1], \ldots, s[N-1]$, with N being the number of values of this sequence, much smaller than the size of E.

From these assumptions, the Fourier transform of this part is [OS99]:

$$S(\omega) = \sum_{m=0}^{N-1} s[m] e^{-i\omega m} \tag{2.4}$$

On a computer, the explicit computation of $S(\omega)$ can only be done on a finite set of values of ω if we want the computation to terminate (since the set on which ω is defined is dense). Hence, a computable discrete Fourier transform must have discrete frequency. Such a discrete Fourier transform (DFT) can be written as:

$$S[k] = \sum_{m=0}^{N-1} s[m] e^{-i2\pi \frac{km}{N}} \tag{2.5}$$

where k is the discrete frequency. The frequency samples will henceforth be called *bins*.

2.3 Short-time Fourier transform

The Fourier transform thus shifts a signal from the temporal domain to the frequency domain. Indeed, the signal is no longer dependent to the time dimension. In some cases, however, it might be convenient to keep track of the time, hence trying to find the evolution of the Fourier spectrum over time. An alternative is then to analyse the signal by small pieces. This is called the *Short-Time Fourier Transform* (STFT) [All77]. Its aim is to determine the sinusoidal frequency and phase content of the signal over local sections of the signal. The result of this transform is thus a two-dimension function of frequency and time. The continuous form of the STFT is written as:

$$S(\omega, t) = \int_{-\infty}^{+\infty} w(\tau) s(t+\tau) e^{-i\omega\tau} d\tau \tag{2.6}$$

In this formula, $w(t)$ is a window function. Interesting results are obtained for window functions that are non-zero over a short-period of time only, so that the locality of the STFT is effective. In our work, we apply a *Hann window* on the signal: the piece of signal

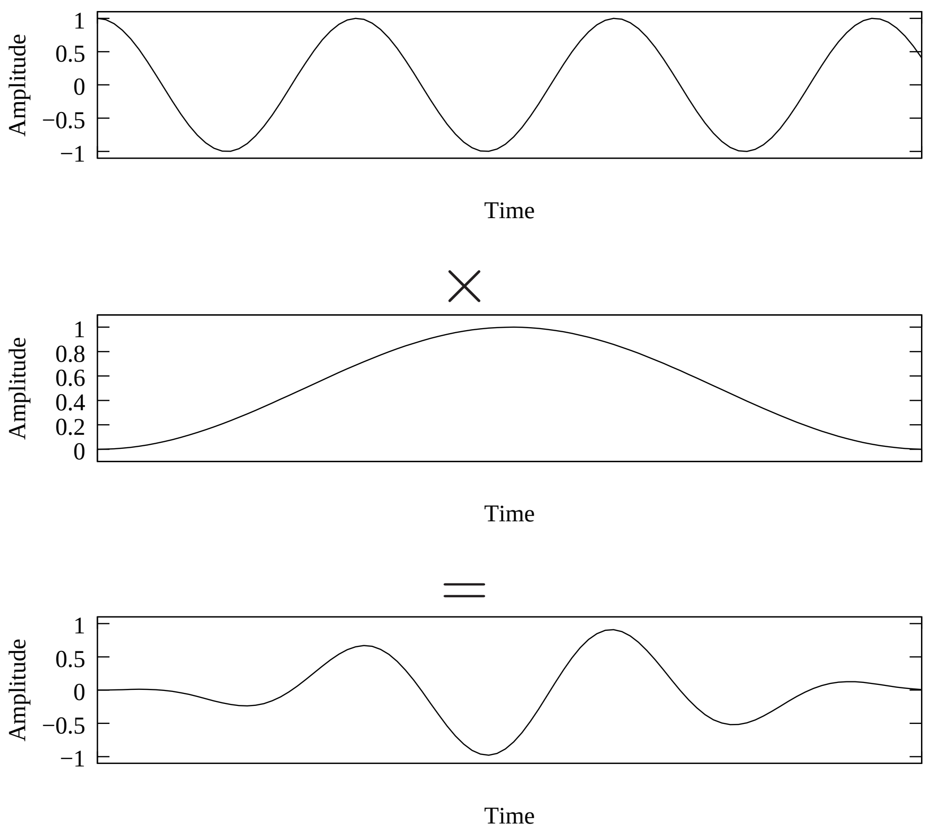


Figure 2.13: *Applying a Hann window on a sinusoidal signal.*

and the analysis window are multiplied (see Figure 2.13). The Hann window is in fact a period of the cosine function shifted between 0 and 1:

$$w_{\mathrm{Hann}}(t) = \frac{1}{2}\left(1 - \cos(2\pi t)\right) \tag{2.7}$$

for $t \in [0, 1[$, and

$$w_{\mathrm{Hann}}(t) = 0 \tag{2.8}$$

otherwise. Different analysis windows will be shown later in this section.

In the same fashion as in the continuous-time case, it can be interesting to perform short-time Fourier transforms over discrete-time signals, particularly in the case of large non-stationary discrete-time signals, so that we can analyse the frequency evolutions of the signal over time. Hence, based on the continuous equation of the STFT (Equation 2.6 on the preceding page), a discrete-time form is written as $S(\omega, n]$ since ω is the continuous frequency and n is the discrete time.

$$S(\omega, n] = \sum_{m=0}^{N-1} w\left(\frac{m}{N}\right) s[n+m] e^{-i\omega m} \tag{2.9}$$

where w is the continuous-time window function used in the continuous-time STFT.

Knowing that ω can only have a finite number of values on a computer, the corresponding discrete STFT is then:

$$S[k,n] = \sum_{m=0}^{N-1} w\left(\frac{m}{N}\right) s[n+m] e^{-i2\pi\frac{km}{N}} \tag{2.10}$$

This is the discrete STFT we will use for the practical computation of short-term Fourier spectra.

A way to plot the STFT over time is to use a *spectrogram* (Figure 3.17 on page 36, spectrogram of a note played on an alto saxophone). A spectrogram is a representation of the amplitudes (colour intensity) of the frequencies (vertically) as a function of time (horizontally).

2.3.1 Convolution

We just saw that in the STFT, an analysis window is applied in order to preserve the locality of the short-term analysis. In order to study in more depth the influence of an analysis window, we need to define the convolution operation.

The convolution operation is defined as:

$$(s * u)(t) = \int_{-\infty}^{+\infty} s(\tau)u(t-\tau)d\tau \tag{2.11}$$

or in discrete form as:

$$(s * u)[n] = \sum_{m=-\infty}^{+\infty} s[m]u[n-m] \tag{2.12}$$

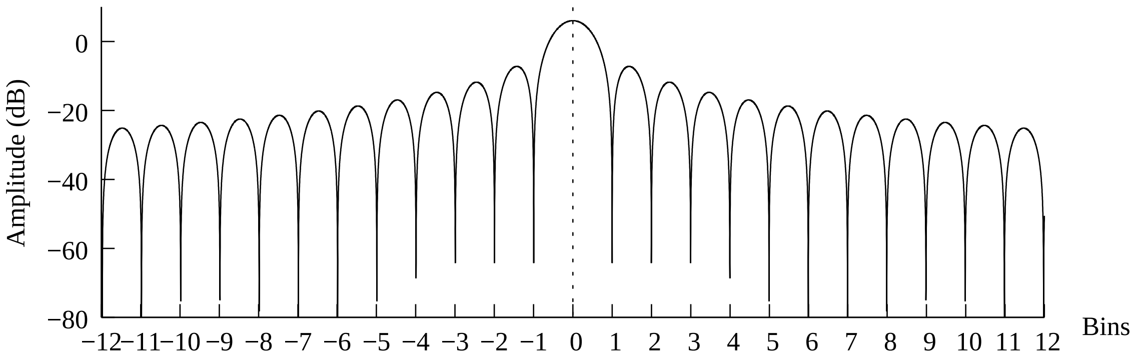
Here, we define that for a given signal $s(t)$, the notation $S(\omega)$ is the Fourier spectrum of $s(t)$. The convolution theorem (see [OS99]) states that if $v(t) = (s * u)(t)$ then $V(\omega) = S(\omega)U(\omega)$, but also if $v(t) = s(t)u(t)$ then $V(\omega) = (S * U)(\omega)$. This means that the convolution operation is the spectral counterpart of the multiplication and *vice-versa*.

2.3.2 Analysis windows

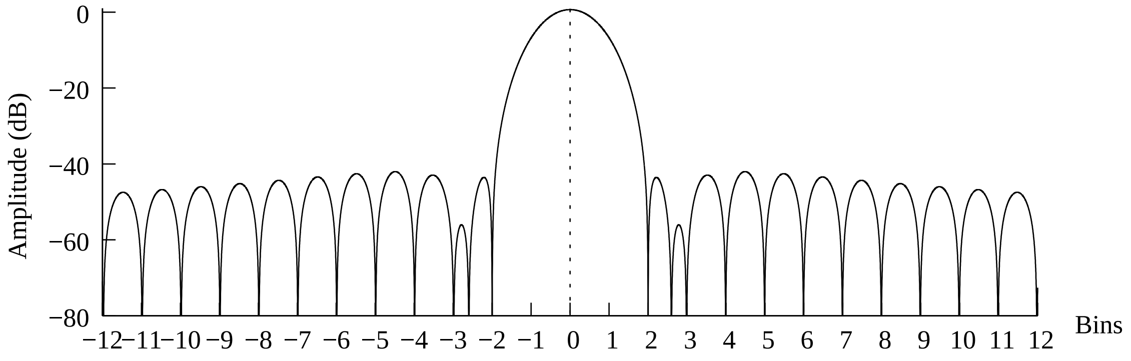
According to the convolution theorem, applying a window on a signal is thus equivalent to convoluting the spectrum of the signal with the spectrum of the window. Applying a window for the STFT is mandatory, and thus the choosing of this window is of some importance. Applying a rectangular window is not advisable, since it adds some unpleasant effects. By multiplying the signal by the Hann window, some lesser artifacts are added to the spectrum.

Several other types of windows exist, each having different effects on the resulting spectrum (since they all have different spectra). The spectra of four different analysis windows are shown on Figure 2.14 on the next page.

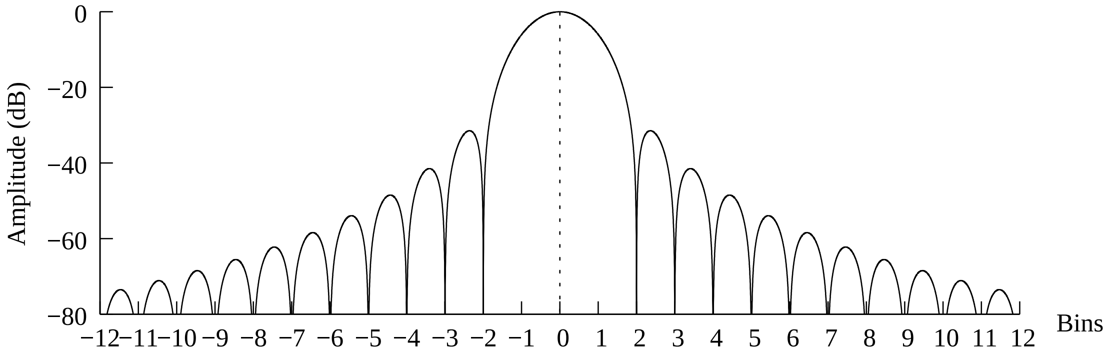
The corresponding continuous-time equations of these windows are:



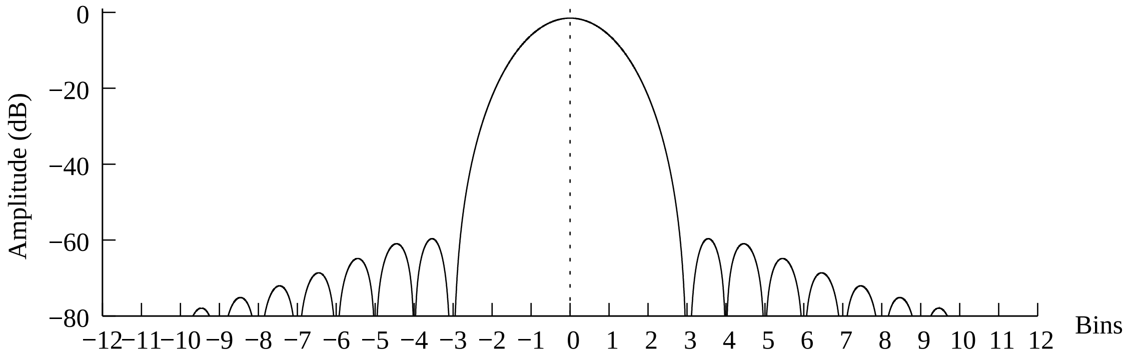
(a) Rectangular



(b) Hamming



(c) Hann



(d) Blackman

Figure 2.14: *Magnitude spectra of some analysis windows.*

$$\begin{aligned}
w_{\text{Rectangular}}(t) &= 1 && (2.13)\\
w_{\text{Hamming}}(t) &= 0.54 - 0.46\cos(2\pi t) && (2.14)\\
w_{\text{Hann}}(t) &= \frac{1}{2}\left(1-\cos(2\pi t)\right) && (2.15)\\
w_{\text{Blackman}}(t) &= 0.42 - 0.5\cos(2\pi t) + 0.08\cos(4\pi t) && (2.16)
\end{aligned}$$

where $t \in [0, 1[$, and $w(t) = 0$ otherwise.

Even more windows exist. However, their study is beyond the scope of this work, hence the interested reader is advised to refer to the work of Harris [Har78] and Oppenheim & Schafer [OS99] for more information.

Thus, since the spectrum of this window has an effect on the resulting spectrum of the signal, it might be interesting to have the analytical form of the spectrum of the Hann window. The discrete-time rectangular and Hann windows can be defined as (details in Keiler and Marchand [KM02]):

$$\begin{aligned}
w_{\text{Rectangular},N}[n] &= w_{\text{Rectangular}}\left(\frac{n}{N}\right) && (2.17)\\
w_{\text{Hann},N}[n] &= w_{\text{Hann}}\left(\frac{n}{N}\right) && (2.18)
\end{aligned}$$

for $n \in \mathbb{N}, 0 \leq n < N$.

The discrete-time Fourier transform of the discrete-time Hann window of size N of can then be computed as:

$$W_{\text{Hann},N}(\omega) = \frac{1}{2}W_R(\omega) - \frac{1}{4}W_R\left(\omega - \frac{2\pi}{N}\right) - \frac{1}{4}W_R\left(\omega + \frac{2\pi}{N}\right) \qquad (2.19)$$

where

$$\begin{aligned}
W_R(\omega) = W_{\text{Rectangular},N+1}(\omega) &= \sum_{m=0}^{N} e^{-i\omega m} && (2.20)\\
&= e^{-i\omega\frac{N}{2}}.\frac{\sin\left(\omega\frac{N+1}{2}\right)}{\sin\left(\frac{\omega}{2}\right)} && (2.21)
\end{aligned}$$

The convolution of this spectrum with the spectrum of the signal has several effects. In particular, the effects of the Hann window will be studied, since this Hann window will be used throughout the remainder of this document. Indeed, the properties of the Hann window are interesting for sound analysis.

The first interesting property is that the analytic form of the spectrum of the window is known. This will be very helpfull to compute a good approximation of the frequencies of the sinusoids present in the signal (this is described later on).

Another interesting property is the relative simpleness of the equation of the Hann window, since it is a period of the cosine function, thus enhancing computation time. It is also interesting to notice that the Hann window is differentiable and continuous.

Moreover, the Hann window provides a resolution of 2 bins. The resolution is the frequency difference that is needed between two sinusoids in a signal to be distinguishable

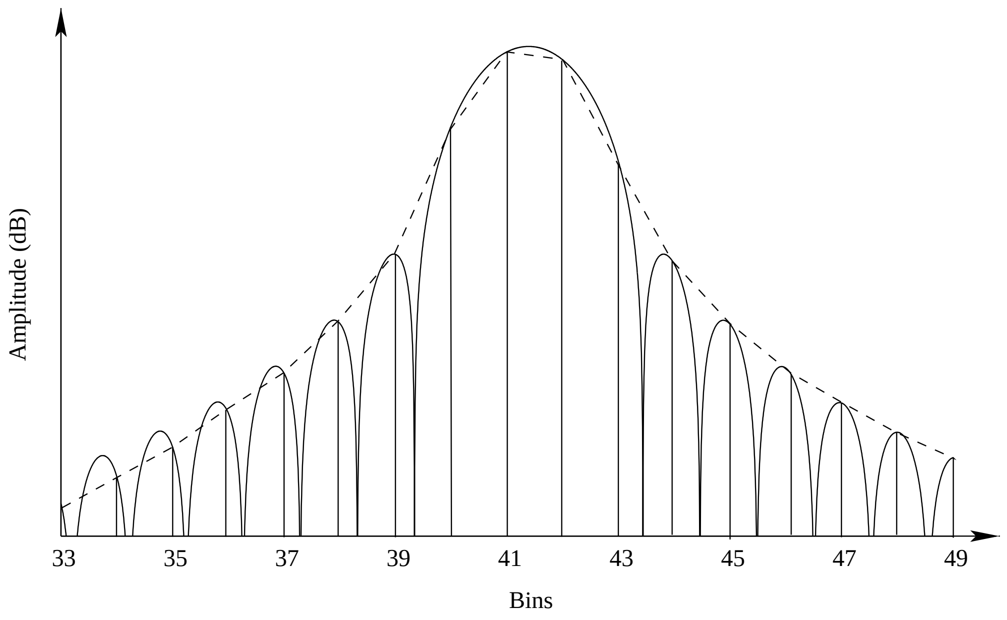


Figure 2.15: *Discrete Fourier transform of the Hann window. The sequence of amplitude values at the sampled frequency locations (shown as a dashed line) form a single peak. Hence, on discrete Fourier transforms, the Hann window has the interesting property of forming a single peak for each sinusoid (which is the case with the Hamming window for example).*

in the spectrum. The Hann window is not the best window for that matter[3], but it is sufficient in most cases.

There is also a good ratio between the amplitude of the main lobe in its spectrum and the secondary lobes. We call the "bumps" seen on Figure 2.14 on page 29 lobes, the main lobe being the widest one that is centred on zero. Indeed, when this window is applied, the peaks corresponding to sinusoids in the signal are more prominent on the spectrum than with the rectangular window for example.

Finally, when no zero padding[4] is applied, the secondary lobes are not distinguishable. Indeed, the discrete-frequency sequence formed by the values of the Hann window spectrum contains only one local maximum. Figure 2.15 illustrates this. In other words, if only one sinusoid is present in the signal, the maximal value of the corresponding Fourier spectrum will be in the bin corresponding to the frequency of the sinusoids, and there will be no other local maximum. This is described in further details in the work of Desainte-Catherine and Marchand [DCM00].

All these properties make the Hann window the best choice for our purpose.

[3]The window that would allow the best resolution (from the four windows we mentioned) would be the rectangular window, since the width of the main lobe is only 2 bins, while the Hann window has a 4-bins wide main lobe. However, the rectangular window have very high side lobes, which in some way degrades that good resolution.

[4]Zero padding: Adding zeroes to a signal. This is done on discrete-time signals to obtain a higher precision on the discrete Fourier spectrum. The resolution of the spectrum, however, stays the same.

2.4 Time-scaling using the spectral model

The spectral model, as representing the evolution of the spectrum of a signal over time, allows to perform time-scaling with better properties than its temporal counterpart.

In this section, we will present the phase-vocoder technique and the way to perform time scaling with it.

2.4.1 The phase vocoder

The vocoder is the name given to the invention of Homer Dudley, which he presented in 1939 as the channel coder, or voice coder [Dud39]. The vocoder operates on the principle of deriving voice codes to re-create the speech which it analysed. During analysis, the fundamental frequency is first estimated, then the spectral information is obtained using a filter bank of a dozen filters, thus obtaining a dozen amplitude values. For synthesis, voiced and unvoiced speech is discriminated using the energy of the fundamental frequency. The voiced sounds are then generated using a buzz, and the unvoiced sounds are generated using random noise for the hiss. The signal is then filtered using the spectral information from the analysis. This device was created in order to reduce the bandwidth for speech transmission.

The phase vocoder was first introduced by Flanagan and Golden [FG66] and is an extension of Dudley's vocoder. The phase vocoder still uses the same principle of a filter bank, but in addition of computing the amplitudes of the spectra of the signal, it also computes the phases of the spectra of the signal.

The phase vocoder has also been presented in Dolson's phase-vocoder tutorial [Dol86]. In this article, Dolson presents two equivalent approaches for the phase vocoder: one based on the traditional filter bank, and another based on the Fourier transform. Hence, the STFT can be used for the phase vocoder.

The phase vocoder seems to give good results for stationary sounds. Considering that the spectral peaks stay in the same bin over the duration of the whole sound, it is possible to compute the phases of the sinusoids whithout the 2π modulo factor (using the frequency as increment). This computation is called *phase unwrapping*.

However, the hypothesis that the spectral peaks stay in the same bin over the duration of the whole sound might not be fulfilled. In this case, the phase looses coherence, and a phenomenon called *phasiness* can be heard in the synthesised signal.

In order to avoid the phasiness, Puckette developed the idea of the phase-locked vocoder [Puc95]. The phases of the different bins are indeed supposed to be constrained if the signal is constant. Hence, forcing this phase constraint over signal reduces the phasiness effect. However it can still be heard in some cases.

In order to go even further, Laroche and Dolson [LD97] designed a way to lock the phases of neighbouring bins of the spectral peaks, so that the bin shift is no longer a problem. The phasiness problem is not yet solved using this method, but the results are better. However, they concluded saying that the ideal solution would be to follow explicitly the spectral peaks over time. This is done in the sinusoidal model we will present in the next chapter.

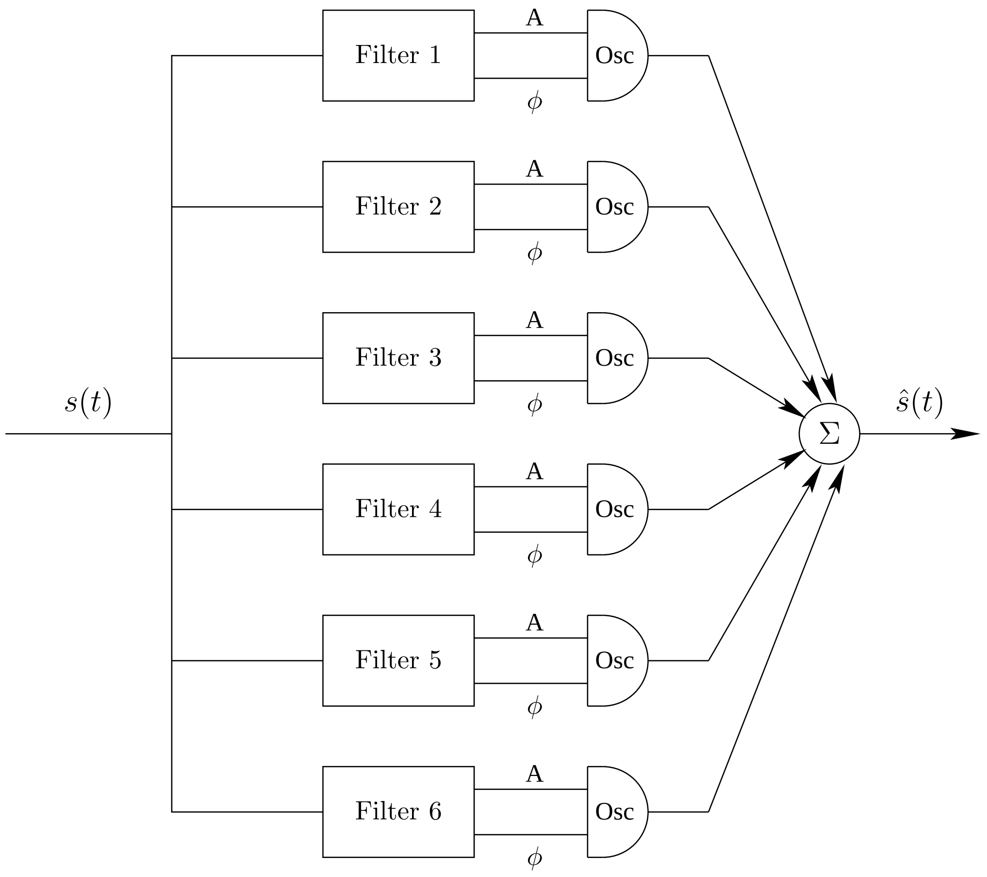


Figure 2.16: *The phase vocoder with six filters. An input signal $s(t)$ is decomposed in six pairs of amplitude-phase values (A and ϕ), which are then fed to oscillators producing the output signal $\hat{s}(t)$.*

2.4.2 Time-scaling using the phase vocoder

The time-scaling of a signal using the phase vocoder can be done very simply using a different hop-size than the original. Indeed, if h is the original hop-size, and r is the desired scaling factor, the synthesis step has to be $r \times h$. This time scaling is however based on the hypothesis that there is at most one sinusoidal component per frequency band.

In order to perform the synthesis at each frame, the values to use are the original amplitudes and modified phases. Indeed, the phase of a bin being dependent over the frequency of the bin and time, it has to be recomputed for each bin. This is done using a phase propagation formula that is:

$$\phi_k(t_{n+1}) = \phi_k(t_n) + 2\pi f_k(t_n) r\tau$$

where $\phi_k(t)$ is the phase of the k^{th} sinusoid at time t, $f_k(t)$ is its frequency at time t, and $\tau = \frac{h}{F_s}$.

This formula ensures that the phase is coherent within each bin. However, as we said earlier, the phase coherence is not preserved among bins. Using the phase-locking mechanisms, it is possible to improve this formula. However, it might be preferable to track the spectral peaks in order to avoid phase problems. This idea will be developed in the next chapter.

2.5 Conclusion

In this chapter, we have presented the Fourier transform, which is the basis of the spectral representation of sound. We have also introduced the time-frequency representation of sound, which has been developed using short-time Fourier transforms of the sound. This time-frequency representation allows to monitor the frequency content of the sound over time, hence allowing time-scaling without pitch modification of the sound.

Therefore, we have presented a time-scaling method based on the vocoder technique: The frequency content of the sound is scaled to match desired scaling factor of the sound. However, the vocoder technique suffers phase problems since the frequency content is not structured over time.

The sinusoidal model provides this structure, and thus solves this problem, as we will see in the next chapter.

Chapter 3

Sinusoidal Modelling

The observation of a spectrogram (on Figure 3.17 on the following page) provides interesting perspectives. Indeed, we can see that there are quasi-linear frequency structures in the sound over a background of noise. One idea would be to extract these structures in order to have a different description of the sound and probably new manipulation possibilities.

In the case of slow varying sounds, the frequency components representing the sinusoids can be "followed", thus forming coherent frequency structures. In order to perform the grouping of each of these frequency structures, the sinusoidal model was designed.

In our work, we will use short-time Fourier transforms, since the sound signals may not be periodic. The Fourier transform implies that the transformed sound signal is stationary. However, this assumption is not realistic for most natural sounds, since the parameters of these sounds are very unlikely to be constant over a long time.

One solution for this stationarity problem is to use the STFT we defined earlier. In this case, the length of the pieces of signal we transform should be sufficiently small for the parameters to stay quasi-constant over the span of each part of the signal. Of course, the words "small" and "quasi-constant" are not very precise, but we will consider that a signal complies if it can be considered stationary over about 50 ms.

Each piece of the signal thus complies with the fact that the signal is composed of sinusoids. In that case, we can write that a small piece of signal $s_n(t)$ from the n^{th} frame is:

$$s_n(t) = \sum_{p=0}^{P-1} s_{n,p}(t)$$

with

$$s_{n,p}(t) = a_{n,p} \cos\left(2\pi f_{n,p}.(t - n\tau) + \phi_{n,p}\right)$$

for $n\tau \leq t < n\tau + \Delta t$. Here, $a_{n,p}$, $f_{n,p}$, $\phi_{n,p}$ represent respectively the amplitude, frequency, and phase ($a_{n,p}$ and $f_{n,p}$ are considered constant over $[n\tau, n\tau + \Delta t[$). τ is called the *hop size* (in seconds) and is the time difference between two locations where the STFT is performed. Δt is called the *frame length* (in seconds) and is the size of the piece of signal that is transformed (the piece of signal itself is the *frame*).

This *short-term modelling* has been widely used: for stationary sounds [Por76, NLVS99], for noisy sounds [MC97, HDC01]. Global short-term models where designed, trying to handle all kinds of sounds at once [GS97, VVHK99]. Non-stationary models were also proposed for sinusoids with linearly varying frequency [MA86, Mas96, ML03]. These methods are

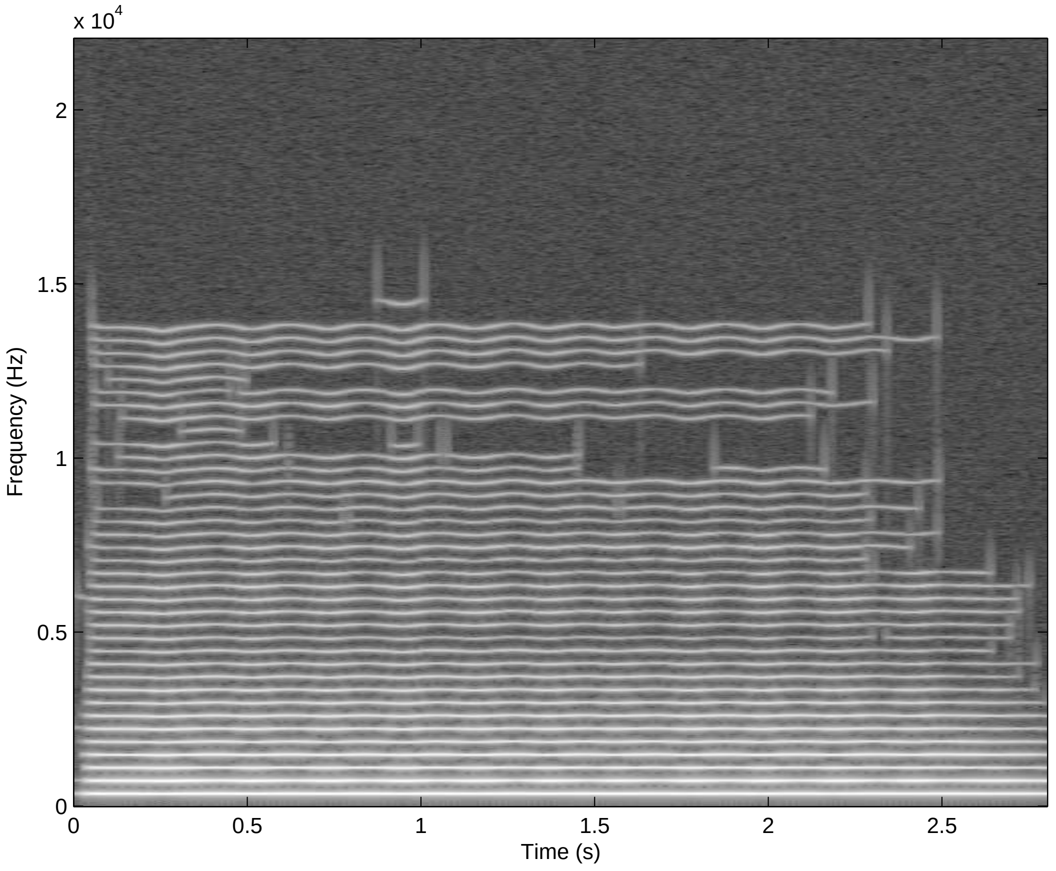


Figure 3.17: *Spectrogram of a note played on an alto saxophone: amplitudes (colour intensity) of the frequencies (vertically) as a function of time (horizontally).*

very interesting, but we aim at doing a better job by modelling the evolutions of each sinusoid.

On the other hand, we also have *long-term modelling* that is based on the assumption that sounds are quasi-periodic. The sound being quasi-stationary, the sinusoidal components of successive frames evolve slowly with time. Hence, instead of defining the amplitude, frequency, and phase parameters for each frame independently, we consider the successive parameters as discrete-time functions. This can be summarised by:

$$s(t) = \sum_{p=1}^{P} a_p(t)\ \cos(\phi_p(t)) \tag{3.22}$$

$$\phi_p(t) = \phi_p(0) + 2\pi \int_0^t f_p(u)\ du \tag{3.23}$$

Hence, the temporal signal is described as a sum of P sinusoids of varying amplitude $a_p(t)$, phase $\phi_p(t)$, and frequency $f_p(t)$ for a given sinusoid p. $a_p(t)$, $\phi_p(t)$, and $f_p(t)$ are the discrete-time functions that describe the evolution of the parameters of a sinusoid p over time.

In the remainder of the document we will use sounds adapted to the use of long-term modelling. This narrows the range of sounds we can use, that is sounds that have a low noise level. However, we choose to work precisely under these conditions, so that the sounds we experiment will fit the long-term sinusoidal model. This model has however some advantages, since the description of the sound can become rather concise and that the manipulation of sinusoidal parameters is easy.

This long-term sinusoidal model was used by McAulay and Quatieri in 1986 [MQ86] for speech signals and by Serra and Smith [Ser89] for musical signals.

In the remainder of this document, when "sinusoidal model" is used, it will refer to the long-term sinusoidal model.

3.1 Description of the sinusoidal analysis process

The analysis process for the sinusoidal model was described by McAulay and Quatieri in 1986 [MQ86]. The algorithm uses a STFT. The first step on each frame is to compute the spectrum and to detect the spectral peaks. Once this has been done, the objective is to collect the spectral peaks (representing sinusoids) in so-called *partials*. In order to accomplish this, three steps are performed:

- The new peaks are matched with the already existing partials.
- If some of the new peaks are not matched to any partial, new partials are created.
- If some partials did not receive any new peak, they are stored aside and not used for further matching anymore.

When the entire signal has been analysed, the remaining partials are stored with the other ones.

The resulting partials of such an analysis can be seen on Figure 3.18 on the next page.

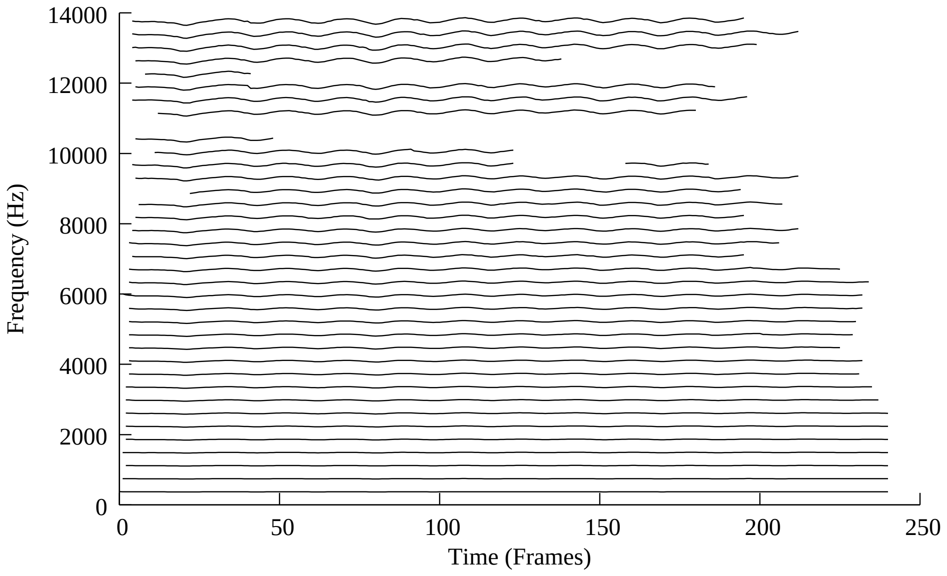


Figure 3.18: *Representation of the frequencies of the partials of an alto saxophone.*

Several questions can arise from this algorithm. The first question is how to match the new peaks with the existing partials. This is answered by McAulay and Quatieri in the following way: Each new peak is associated to the partial that has the closest latest peak, frequency-wise, in a given frequency range. This implies that the frequency of the sinusoids are supposed to be quasi-constant.

The second question is about the phase of the partials. Indeed, the phases that are computed by the Fourier transform have values within the interval $[-\pi, \pi]$, and thus a peak at time $n+1$ in a given partial can have a smaller phase than the peak of this partial at time n. This is not realistic since the phase $\phi(t)$ is the integral of the frequency $f(t)$, which is positive. So the phase must be a monotonous growing function of time. We then have to correct the phases by adding the multiple of 2π $(2M\pi)$ to its value that best fits with the preceding peaks. This process is called *phase unwrapping*.

The McAulay and Quatieri sinusoidal model defines the phase as:

$$\phi(t) = \phi_n + \omega_n t + \alpha t^2 + \beta t^3 \tag{3.24}$$

where $\phi(t)$ is the phase function and ϕ_n and ω_n are the measured phase and frequency at a given frame n.

There are then four constraints to fulfil at the frame boudaries:

$$\phi(N) = \phi_n + \omega_n N + \alpha N^2 + \beta N^3 = \phi_{n+1} + 2\pi M \tag{3.25}$$

$$\omega(N) = \omega_n + 2\alpha N + 3\beta N^2 = \omega_{n+1} \tag{3.26}$$

Solving these constraints gives:

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \begin{bmatrix} \frac{3}{N^2} & -\frac{1}{N} \\ -\frac{2}{N^3} & \frac{1}{N^2} \end{bmatrix} \cdot \begin{bmatrix} \phi_{n+1} - \phi_n - \omega_n N + 2\pi M \\ \omega_{n+1} - \omega_n \end{bmatrix} \tag{3.27}$$

where N is the size of the frame and M is the *phase unwrapping factor* given by:

$$M = i\left[\frac{1}{2\pi}\left((\phi_n - \phi_{n+1}) + (\omega_n + \omega_{n+1})\frac{N}{2}\right)\right] \tag{3.28}$$

where $i[x]$ is the nearest integer from x. M is the phase unwrapping factor that ensure the "smoothest" frequency track.

This sinusoidal analysis algorithm works quite well in most of the natural monophonic and pseudo-periodic sounds. However some enhancement were made to this algorithm, among others by Depalle *et al.*[DGR93], Girin *et al.*[GMdM$^+$03], Desainte-Catherine *et al.*[DCM00], Marchand *et al.*[ML06], and Lagrange [Lag04].

3.2 Enhancing the precision of the discrete Fourier transform in peak estimation

As said in the previous section, prior to performing the gathering of spectral peaks into partials, it is essential to have those peaks. The better the estimation of the peaks, the more precise the partials.

Hence, the first enhancement that can be applied is the enhancement to the precision of the Fourier transform. The discrete form of the Fourier transform may not be very precise when dealing with the position of the spectral peaks. Indeed, the peaks are constrained to be determined by the bins of the Fourier transform. Figure 3.19 on the following page shows how imprecise a discrete Fourier transform can be.

This imprecision might cause some problems, in particular for sinusoidal analysis (exposed in Section 3.1 on page 37). However, it is possible to enhance the precision of the discrete Fourier transform for example using signal derivatives. This technique was proposed in 2000 by Desainte-Catherine & Marchand [DCM00]. Other methods include the reassignment method [AF95] and the difference method (used by the phase vocoder) [AKZ02, chapter 9, pp. 299–372]. Marchand and Lagrange have shown recently that these three methods are equivalent [ML06].

Using an order-one signal derivative, it is possible to have a very good estimation of both frequency and amplitude of given peaks. We place ourselves in the context of sinusoidal modelling. We suppose that each peak in the spectrum represents only one stationary sinusoid (*i.e.* that the resolution of the Fourier transform is good enough).

We suppose also that the amplitude and the frequency of the sinusoidal components are constant over the analysis window.

The expressions of the signal $s(t)$ and its derivative $s'(t)$ in the sinusoidal model are:

$$s(t) = \sum_{p=1}^{P} a_p(t)\cos(\phi_p(t)) \tag{3.29}$$

$$s'(t) = \frac{ds(t)}{dt} = \sum_{p=1}^{P} 2\pi f_p(t) a_p(t)\cos\left(\phi_p(t) - \frac{\pi}{2}\right) \tag{3.30}$$

In the discrete-time form, the approximate first derivative of the signal (noted $s'[n]$) can be:

$$s'[n] = F_s(s[n] - s[n-1]) \tag{3.31}$$

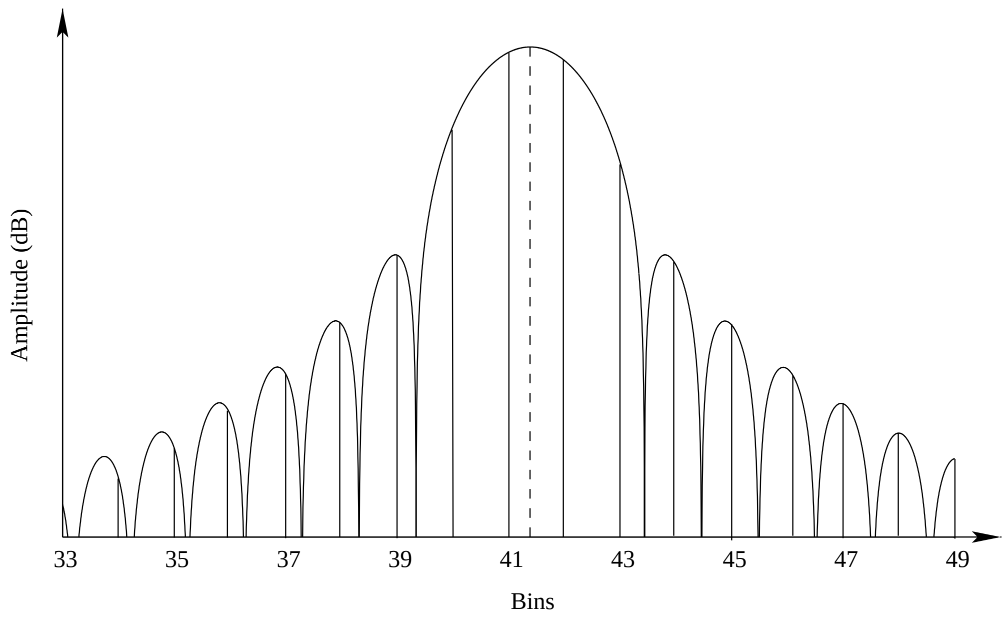


Figure 3.19: *Discrete Fourier transform. The dashed line represents the exact spectral peak of a sinusoid in the continuous Fourier spectrum. Since a Hann window has been applied on the signal prior to the transformation, the peak is convoluted with the spectrum of the window. The spectrum of the window is shown as the solid curve on the figure. Moreover, the actual transformation that was performed is a DFT. Hence, the spectrum is sampled, and the result is shown as the solid lines. Therefore bin 41 has the maximum amplitude value, but it is not situated on the exact frequency of the peak and thus, the amplitude of bin 41 is not the exact amplitude of the considered peak. Hence some precision enhancement has to be performed, both for frequency and amplitude estimations.*

The objective is to find the precise frequency f_p and amplitude a_p of a given sinusoidal component p from the discrete-time signal and its discrete Fourier spectrum. The sinusoidal component p is represented on the discrete spectrum $S[m]$ of $s[n]$ at bin m_p and has a measured amplitude of $\widetilde{a_p}$. On the discrete spectrum $S'[m]$ of $s'[n]$, the measured amplitude of the peak at location m_p is $\widetilde{a'_p}$.

$$\widetilde{a_p} = |S[m_p]|$$

$$\widetilde{a'_p} = |S'[m_p]|$$

We can show that [DCM00]:

$$a'_p = 2a_pF_s \sin\left(\frac{\omega_p}{2}\right) \tag{3.32}$$

where ω_p is the frequency in $rad.s^{-1}$, and $\omega_p = \frac{2\pi f_p}{F_s}$, and finally we can show that

$$\hat{f_p} = \frac{F_s}{\pi} \arcsin\left(\frac{\widetilde{a'_p}}{2\widetilde{a_p}F_s}\right) \tag{3.33}$$

is a good estimator of f_p, and that

$$\hat{a}_p = \frac{\widetilde{a_p}}{\left|W\left(2\pi\left|\frac{\hat{f_p}}{F_s} - \frac{m_p}{N}\right|\right)\right|} \tag{3.34}$$

is a good estimator of a_p, with $|W(\omega)|$ being the amplitude spectrum of the analysis window (as seen in Equation 2.19 on page 30) .

This better estimation of the frequency and amplitude parameters can be performed in the context of the sinusoidal model. This enhancement is especially useful for strong and low frequency partials, since their corruption is the most likely to be heard.

3.3 Enhancing the partial tracking

Other enhancements to the sinusoidal model deal with partial tracking. Indeed, the weak point in the McAulay and Quatieri algorithm is the partial-peak matching. No consideration is made about the global shape of the partial, amplitudes are not taken into account and punctually missing peaks can be lethal to a partial.

A first enhancement (presented in [DGR93], and in [Lag04]) is to take the amplitudes of the peaks into account. Indeed, it is very likely in natural sounds that the amplitude of a sinusoidal sound remains slow-varying in time (respectively to the window-size of the analysis process). Hence, it is possible to enhance the global quality of the sinusoidal analysis by giving priority to the louder partial in the peak matching step. Indeed, we consider in this work that the partial having the highest amplitude on its last peak tries to find the best matching peak in a given frequency range. This takes into account the fact that louder partials are less likely to have missing data, and also that an error in louder partials is probably more annoying for human perception than an error in quieter partials.

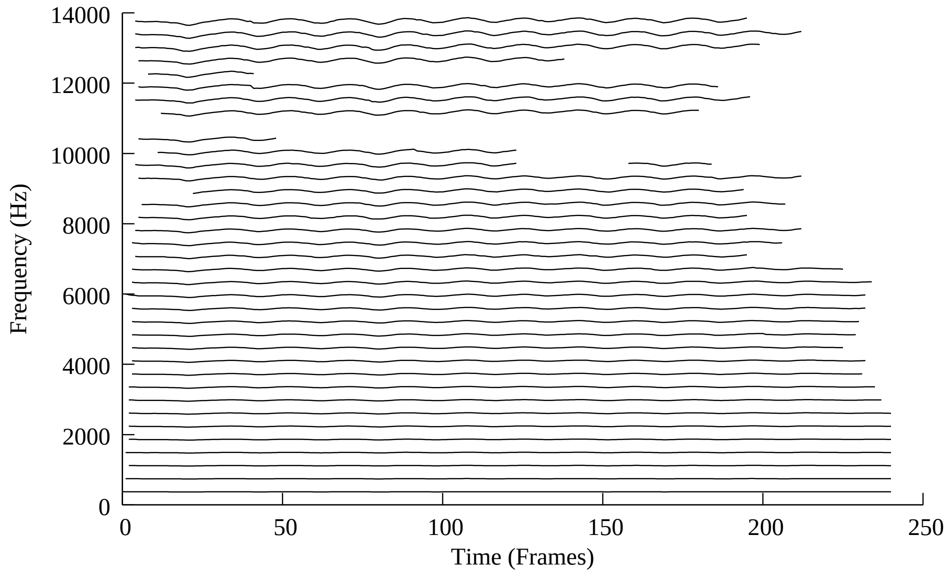


Figure 3.20: *Representation of the frequency of the partials of an alto saxophone in the sinusoidal model. The frame rate is about 86 Hz. The partial seem to be pseudo-periodic and slow-varying.*

A last enhancement concerns missing data. It can happen that in an analysis frame one or more peaks are lost due to bin contamination[1] and that this loss is temporary, meaning that in the next frame, the corresponding peaks are present again. In this situation, the classical algorithm puts the partials that do not have a matching peak in the current analysis frame aside and they are said to be "dead". Hence, what would be a single partial could be split in several small partials. In the case that small partials are discarded, this phenomenon could be quite problematic.

A solution (presented by Marchand [Mar00]) for this would be to add to the partials a fixed-size ghost peak buffer (that is a buffer containing peaks that are not really there). Indeed, in the case where a partial does not have a matching peak in the current analysis frame, the estimated peak would be placed in this ghost peak buffer. Then, during the next analysis frame, considering the ghost peak as the last peak of the partial, it might be possible to find a matching peak. The ghost peak would then be part of the partial and taken out of the ghost peak buffer. However, if after a certain amount of peaks (typically equivalent to 10 ms of real time), if no further matching peaks have been found, the ghost peaks are discarded and the partial is put aside in the state it had before the ghost peaks were added. This solves the partial splitting problems present in the McAulay & Quatieri algorithm.

3.3.1 Partial trajectories as control signals

The base hypothesis of this part of our work is that the partial parameters can be considered as control signals of the sound. Indeed, it is clear that the partial parameters are

[1]Bin contamination is the phenomenon occuring when two or more spectral peaks are really close together, so that it becomes impossible to distinguish them.

functions of time. These functions convey information about the sinusoids present in the sound.

Figure 3.20 on the preceding page represents the evolutions of the frequency of the partials of a sound of an alto saxophone with vibrato (and tremolo). This representation of the sound in function of time gives an indication that each partial can be considered as a control signal. More precisely, each partial contains information about the amplitude, the frequency, and the phase of the sound signal in function of time. Thus, a partial is a set of three control signals of the sound signal.

This supposition leads then to the use of the partials as signals. By observing the global shape of the partials (see for example Figure 3.20 on the facing page), the partial trajectories appear to be pseudo-periodic and slow-varying. Hence we can suppose that the trajectories of the partials are deterministic. From this second hypothesis, it is possible to enhance the partial tracking in the sinusoidal model by trying to predict the trajectories of the partials. Hence, if we can predict the trajectories of the partials, we can better track the peaks constituting the partials during the sinusoidal analysis. This prediction is performed using signal processing tools, since we consider the partial trajectories to be control signals.

While dealing with sound control, it is important to note that software exist that use this principle of sound control. For example, CSound [cso] is a music programming language. CSound was written by Barry Vercoe, and is based on an earlier language called Music360, created by Max Mathews. It is a piece of software that synthesises music. The system is fed with two kinds of input: a score file and an orchestra file. The score file, just as in music, describes the symbolic patterns of a piece, at a very low rate (this very low rate representation of symbolic patterns is what we aim for in chapter 6 on page 121). On the other hand, the orchestra file describes instruments that the composer can use. These instruments are defined as a combination of generating functions (oscillators, buzz) that output sound at a given sample-rate. However, the parameters of these functions can be played with at a lower sampling-rate called the control rate (`kr` in the software). The ways of controlling these sound parameters include envelopes, the inner parameters of the generating functions (frequency for the oscillator), and digital effects. These parameters that control the sound are thus simpler than the sound in itself and at a lower sample-rate.

This is hence very close to what we can achieve with partials. Indeed, the partials are simpler than the sound and do not need an as high sampling-rate as the sound itself. Thus, we can consider the partials as control parameters of the sound.

3.3.2 Predictable control signals

Figure 3.20 on the facing page represents the evolutions of the partials of a sound of an alto saxophone with vibrato (and tremolo). The partials represented here have an oscillating behaviour around a centre frequency, representing the vibrato in the sound. This oscillating behaviour of the frequency trajectory of the partial seems to be deterministic. One direct consequence of this signal being deterministic is that the future values of the signal can be calculated from past values. Hence the signal can be said to be predictable.

This assumption that the partials are deterministic is of course quite strong and might not suit every kind of sound. However, we place ourselves in a context where this assumption is valid, that is slow-varying quasi-periodic partials parameters.

This includes stationary signals (e.g. the frequency signal of a non varying sound), oscillating signals (e.g. the frequency signal of sound with vibrato), or a monotonously increasing or decreasing function (e.g. the frequency signal of sound with glissando).

About the global evolution of the partial, a simple enhancement to the matching process can be made. Instead of taking the last peak frequency as reference, it is possible to take into account the frequencies of the two last peaks of the partial and keep the first derivative continuity when estimating the matching peak. Let f_k and f_{k-1} be the frequencies of the two last peaks of the partial we consider, and f_e be the estimate of the frequency for the potentially matching peak. We can then compute f_e as:

$$f_e = 2f_k - f_{k-1} \tag{3.35}$$

This simple enhancement makes it a lot easier to follow partial having a straight line as frequency evolution.

Another way to estimate the future values of the signal is to use linear prediction. Linear prediction is based on a stochastic Auto-Regressive model (AR model), and works for deterministic signals. Before studying the use of linear prediction on sinusoidal models, let us first have a look at the use of these methods in the temporal domain.

The objective of linear prediction is to predict future samples of a signal.

Principles of linear prediction

Linear prediction is the prediction of the evolution of a supposedly autoregressive (AR) system using a given number of observations. A given sample $s(n)$ (s being a signal) is approximated by a linear combination of these observations (the passed-produced samples). The predicted sample $\hat{s}(n)$ is computed by finite impulse response (FIR) filtering of the last M observations:

$$\hat{s}(n) = \sum_{m=1}^{M} b(m)s(n-m) \tag{3.36}$$

M is called the *order* of the model.

Supposing N successive observations, we want to compute the $b(m)$ coefficients that give the best approximation of signal $s(n)$. The approximation quadratic error is given by:

$$E = \sum_{n=0}^{N-1} (s(n) - \hat{s}(n))^2 \tag{3.37}$$

Several methods exist to minimise this error, among which the autocorrelation method, the covariance method (both method are described in [Mak75]) and the Burg method [Bur67], that we will use and explain here (see also [Lag04]).

The Burg method Let $e_k^f(n)$ and $e_k^b(n)$ be the forward and backward prediction errors for a given order k.

$$e_k^f(n) = s(n) - \sum_{m=1}^{k} b_k(m)s(n-m) \tag{3.38}$$

$$e_k^b(n) = s(n-k) - \sum_{m=1}^{k} b_k(m)s(n-k+m) \tag{3.39}$$

The objective of the Burg method [Bur67] is to recursively minimise the forward and backward variances (mean quadratic errors) on a finite support. To obtain b_k, we minimise:

$$\epsilon_k = \frac{\epsilon_k^f + \epsilon_k^b}{2} \tag{3.40}$$

with

$$\epsilon_k^f = \frac{1}{N-k} \sum_{n=k}^{N-1} |e_k^f(n)|^2 \tag{3.41}$$

$$\epsilon_k^b = \frac{1}{N-k} \sum_{n=0}^{N-k-1} |e_k^b(n)|^2 \tag{3.42}$$

and

$$b_k(m) = \begin{cases} b_{k-1}(m) + r_k b_{k-1}(k-m) & m = 1, \ldots, k-2, k-1 \\ r_k & m = k \end{cases} \tag{3.43}$$

where r_k are called the reflection coefficients. Then we use Equation 3.43 in Equations 3.41 and 3.42, and we find a recursive expression for the forward and backward errors:

$$e_k^f(n) = e_{k-1}^f(n) + r_k e_{k-1}^b(n-1) \tag{3.44}$$

$$e_k^b(n) = e_{k-1}^b(n-1) + r_k e_{k-1}^f(n) \tag{3.45}$$

where we define

$$e_0^f(n) = e_0^b(n) = s(n)$$

Using these equations, it then is possible to compute the linear prediction coefficients for $s(n)$.

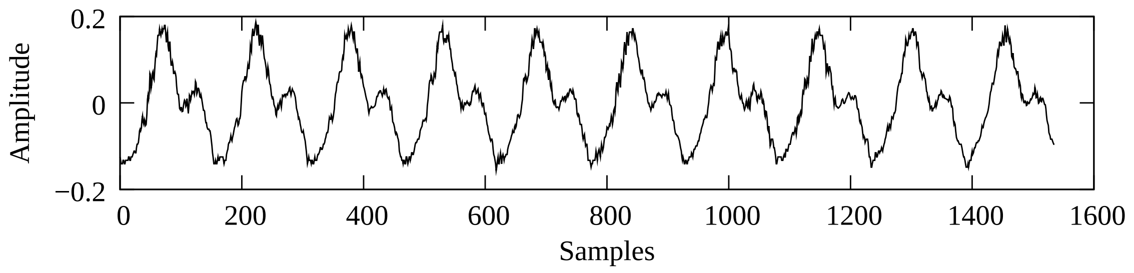
Thus, the two parameters necessary for the Burg method are the order of the model and number of samples of the signal to feed the algorithm. These parameters have indeed an impact on the quality of the signal. One has to know that the higher the order, the better the estimation (simulating more complex exponentials) and the higher the number of samples the better (more data to train the model).

Linear prediction for temporal signals

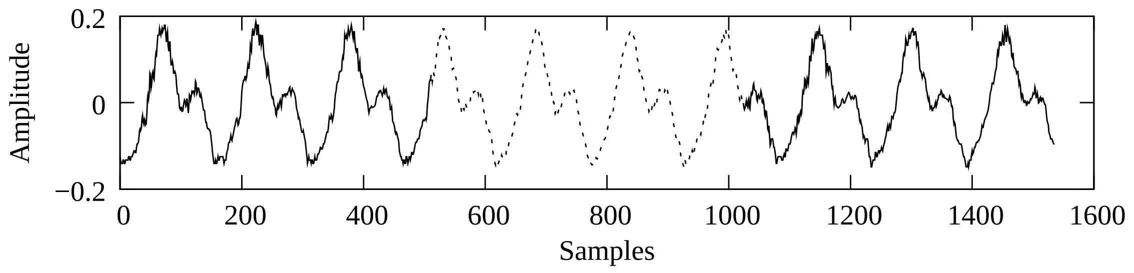
Even though the AR models were designed to be applied on stochastic signals, they can be applied to deterministic signals. It is possible to apply linear prediction coefficients of the sound.

Kauppinen *et al.* use the Burg method [KKS01, KR02] in order to extrapolate temporal sound signals. Indeed, they assumed that temporal signals are deterministic and thus they can be predicted.

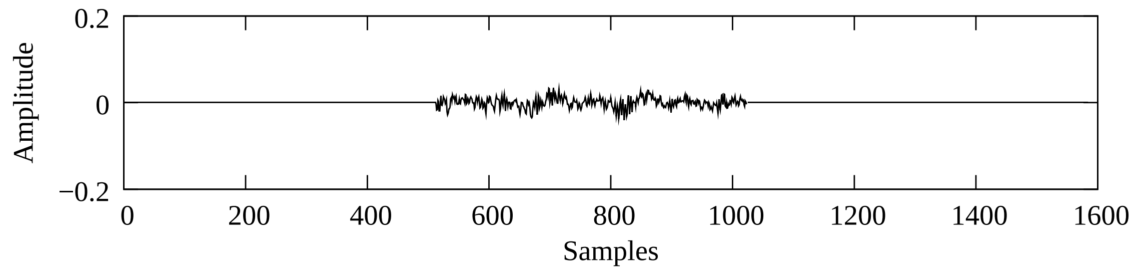
The objective of this was to extrapolate audio signal in order to recover lost data in the signal. As said earlier, if the signal contains n sinusoids, the order of the linear prediction has to be at least $2n$. When this constraint is satisfied, the signal can be extrapolated very well, provided that the parameters of the sinusoids are constant.



(a) Original sound represented in the temporal model.



(b) Same sound rebuilt with the Burg method after making a hole in it. The samples before the hole were used for forward extrapolation and the samples after the hole were used for backward extrapolation. The two extrapolation results were mixed using a linear interpolation.



(c) Error of the reconstruction of the sound.

Figure 3.21: *Reconstruction of a temporal sound signal using linear prediction (Burg method).*

However, the autoregressive model is also capable of modelling exponential variations of amplitudes in the sinusoids. This allows the Burg method to be quite efficient even for real cases (having a constant frequency).

In the case of signal reconstruction using extrapolation, it is however possible that the frequency is not perfectly constant for the duration of the gap. Thus, having situated the gap in the signal, using only preceding samples to compute the filter and reconstruct the signal might not be sufficient. Indeed, if the frequencies of the sinusoids changed, the reconstruction at the end of the gap might not match at all the samples following the gap. To overcome this problem, and provided that the samples following the gap are known, it is also possible to extrapolate the signal backwards. This makes sense, since the signal is considered to be deterministic, and thus the past of the signal is supposed to be computable from present and future samples of the signal.

Once the forward and backward extrapolations are computed (respectively from the samples preceding and following the gap), a simple linear interpolation between the two extrapolations is performed to obtain a more realistic reconstruction of the gap. Indeed, the reconstruction is not only dependent on the past but also on the future samples, and thus the transition between the past and the future is continuous.

This method works well in many cases, and even if the lost signal would never be recovered exactly, the result on sinusoidal examples fools the human perception easily.

Figure 3.21 on the facing page shows an example of signal reconstruction using this method. We can see that even if there is a slight error in the reconstruction (without which the AR model would not be applicable by the way), the error is really small and probably negligible. The SNR measures conducted by the Kauppinen *et al.*[KKS01] give indeed really good results.

It is also interesting to notice that the residual error of the extrapolation of the signal seems to be the noisy (purely random) part of the signal. The authors used this property to propose a noise reduction method [KR05] since the reconstructed signal tends to be noiseless.

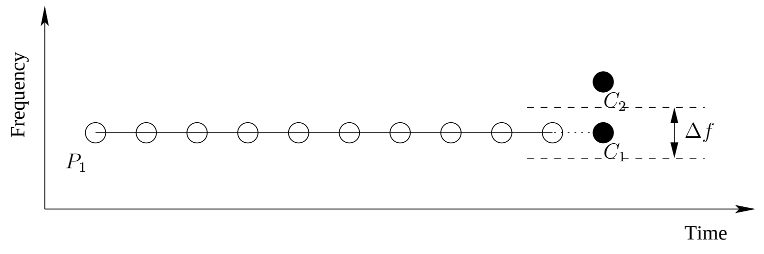
Some factors that improve the extrapolation of the signals include taking many samples from the signal (the more the samples, the better the reconstruction), increasing the order of the linear prediction filter (more exponentials in the model, thus a potentially better estimation of the predicted samples) and making the gap as small as possible.

Linear prediction in the sinusoidal model

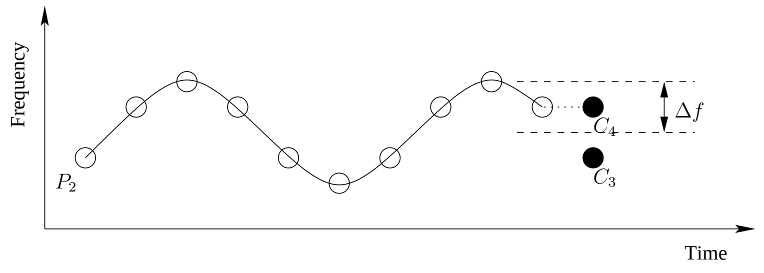
Figure 3.20 on page 42 shows the frequency trajectories of the partials of an alto saxophone with vibrato. We can see that these frequency trajectories seem to be more or less deterministic. Hence, it should be possible to use linear prediction with partial trajectories, and in particular partial extrapolation.

Why would partial extrapolation be of interest?

As we saw in section 3.1 on page 37, the partial tracking algorithm chooses peak candidates in order to lengthen a given partial. The classic method for this choice is to take the closest peak – in terms of frequency – to the last peak of the given partial. Figure 3.22 on the next page shows the partial tracking for two sets of peaks. On the first partial (P_1, shown on Figure 3.22(a) on the following page), the algorithm seems to work well. Indeed, the spectral peak that will be chosen by the method is C_1 because it falls



(a) Partial tracking on a constant trajectory



(b) Partial tracking on a sinusoidal trajectory

Figure 3.22: *Performing partial tracking with the McAulay & Quatieri method. Performs well for linear trajectories but not for sinusoidal trajectories. Indeed, on the second 3.22(b) the peak C_4 is chosen, while C_3 would have suited the partial better.*

perfectly in the Δf interval and is the closest to the last peak of P_1. Thus C_2 will be discarded (for this partial) as it should.

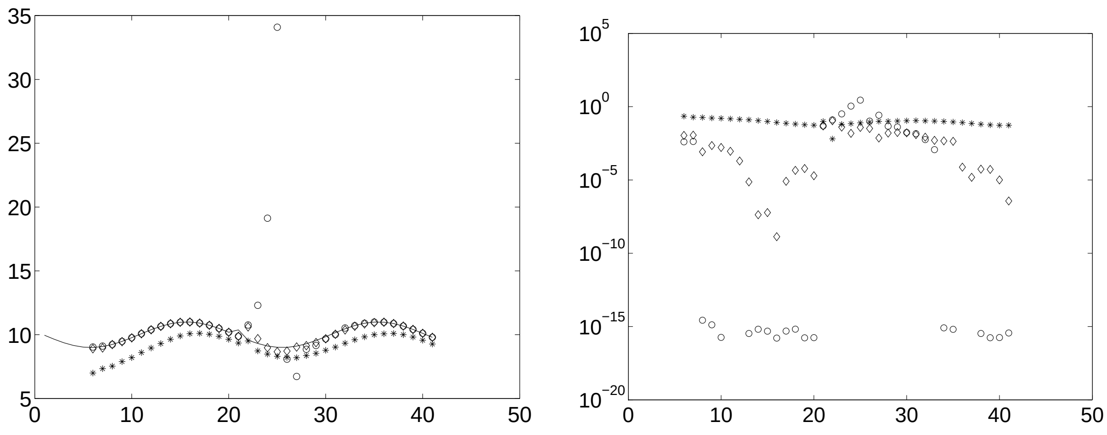
However, let us now consider the second partial (P_2, shown on Figure 3.22(b)). According to the first nine peaks of the partial, the trajectory of the partial seems to be pseudo-sinusoidal. Thus the ideal candidate for this partial at the current step would be C_3, but the McAulay & Quatieri method would pick C_4. This is because C_4 falls within a $\frac{\Delta f}{2}$ distance of the frequency of the last peak of P_2. Hence, in this particular case, the McAulay & Quatieri method is not performing correct partial tracking.

In order to overcome this problem, and more generally the case of partials having a locally periodic shape, we showed (in [LMRR03]) that we can use linear prediction in order to obtain a more accurate partial tracking algorithm.

Indeed, linear prediction methods are well suited for this partial peak estimation problem. The fact that we want to estimate a peak from the past peaks parameters falls exactly in the domain of extrapolation, and linear prediction methods can allow us to find good filter coefficients for this purpose.

In the article we presented in 2003 [LMRR03], we applied three different linear prediction methods and compared them with respect to the partial trajectory estimation. The three methods are the covariance method, the autocorrelation method, and the Burg method. Before computing the linear prediction coefficients for the autocorrelation method, the signal was zero-centred by subtracting its mean.

The results of this comparison are shown on Figure 3.23 on the next page. The autocorrelation method performs badly. This is due to the small amount of data used for the prediction, and since the parameters of the partials are sampled at a very low frequency this poor performance is unacceptable. The covariance method seems to work



(a) Prediction using LP methods (b) Errors

Figure 3.23: *Comparing three linear prediction methods on a slightly modified sinusoid (simulating one error). The covariance method is represented as circles, the Burg method is represented as diamonds and the autocorrelation method is represented as stars. In average, the Burg method performs best.*

better, but it can be unstable in the presence of bursting noise (frame 21 on Figure 3.23). This instability can lead to erroneous estimations in the presence of noise. The best result is obtained with the Burg method, which is reactive but yet resistant to noise.

The model order and the number of samples used to estimate the linear prediction coefficients are of great importance for the quality of the prediction. We choose parameter range values by both theoretical and experimental considerations.

For frequency evolutions, since we want to model exponentially increasing or decreasing evolutions (portamento) and sinusoidal evolutions (vibrato), the order of the LP model should not be below 2. Experimental testing showed that a model order in the $[2, 8]$ range is convenient. Moreover, the number of samples we use has to be twice the order of the model plus one.

Our experimental tests were processed on the known evolutions of frequencies of already-tracked partials of different kinds of pseudo-stationary monophonic signals such as a saxophone, a guitar, and various singing voices. We considered the mean prediction error and the maximal error.

The number of samples used has to be large enough to be able to extract the signal periodicity, and short enough not to be too constrained by the past evolution. The short-term analysis module uses a short-time Fourier transform with a hop size of 512 samples on sound signals sampled at CD quality (44.1 kHz). This means that the frequency and amplitude trajectories are sampled at ≈ 86 Hz. Since we want to handle natural vibrato with a frequency about 4 Hz, we need at least 20 samples to get the period of the vibrato. Experimental testing showed that for most cases a number of samples in the $[4, 32]$ range is convenient.

In Tables 3.1 on the following page and 3.2 on page 51, we present experimental test results of the Burg prediction method with several orders and numbers of samples on already-tracked partials of natural sounds: a saxophone and a singing voice. Additionally,

k:	Constant	Linear	Mirror
1	0.35 (0.9)	0.16 (0.8)	0.16 (0.8)
2	0.69 (1.8)	0.42 (1.4)	0.51 (1.6)
3	1.01 (2.5)	0.76 (2.4)	1.00 (3.1)
4	1.31 (3.1)	1.18 (3.7)	1.63 (4.6)

k:	order : 2	4	6	8
	0.20 (0.8)	- (-)	- (-)	- (-)
1	0.17 (0.6)	0.17 (0.6)	0.18 (0.7)	- (-)
	0.16 (0.6)	0.14 (0.6)	0.14 (0.6)	0.14 (0.6)
	0.16 (0.7)	0.13 (0.6)	0.13 (0.7)	0.12 (0.6)
	0.48 (1.4)	- (-)	- (-)	- (-)
2	0.43 (1.3)	0.42 (1.3)	0.45 (1.6)	- (-)
	0.41 (1.3)	0.35 (1.3)	0.34 (1.4)	0.34 (1.3)
	0.41 (1.3)	0.32 (1.2)	0.29 (1.1)	0.28 (1.1)
	0.85 (2.3)	- (-)	- (-)	- (-)
3	0.75 (2.3)	0.77 (2.3)	0.80 (2.7)	- (-)
	0.72 (2.2)	0.62 (2.0)	0.57 (2.2)	0.57 (2.1)
	0.71 (2.1)	0.52 (1.7)	0.46 (1.7)	0.43 (1.4)
	1.29 (3.5)	- (-)	- (-)	- (-)
4	1.13 (3.5)	1.18 (3.6)	1.20 (4.1)	- (-)
	1.09 (3.5)	0.92 (3.3)	0.83 (3.1)	0.81 (3.0)
	1.06 (3.3)	0.77 (2.5)	0.65 (2.3)	0.58 (1.9)

Table 3.1: *The mean (and maximal) prediction errors for different prediction methods on the frequency evolution of partials of a note of alto saxophone with vibrato. Prediction errors are computed for several simple prediction methods (top chart) and the LP prediction method (bottom chart) for different values of k (the distance in frame indexes between the last observation and the predicted value). Concerning the part dedicated to the LP prediction method, the model order grows from left to right, and for each values of k the number of samples considered is in [4,8,16,32]. The prediction errors of the LP prediction method are lower than those of the best simple prediction method. The improvement is getting more and more significant when k is increasing. For each experiment, the mean error value is presented and long with the maximum error value (in parentheses). The Burg method performs generally better, especially at high orders. These results were presented in [LMRR03].*

k:	Constant	Linear	Mirror
1	2.69 (26.3)	1.16 (10.6)	1.16 (10.6)
2	5.32 (44.9)	3.19 (30.6)	3.99 (36.8)
3	7.81 (61.2)	6.01 (56.8)	8.06 (56.8)
4	10.17 (74.9)	9.45 (84.5)	12.76 (72.7)

k:	order : 2	4	6	8
	1.39 (10.1)	- (-)	- (-)	- (-)
1	1.22 (10.0)	0.97 (12.2)	1.09 (11.5)	- (-)
	1.19 (9.8)	0.86 (12.4)	0.89 (11.4)	0.92 (11.3)
	1.17 (9.6)	0.83 (12.4)	0.84 (11.1)	0.86 (11.2)
	3.66 (29.8)	- (-)	- (-)	- (-)
2	3.24 (28.7)	2.67 (35.1)	2.98 (33.1)	- (-)
	3.17 (27.7)	2.35 (35.2)	2.35 (33.3)	2.43 (33.9)
	3.12 (26.7)	2.24 (34.4)	2.24 (32.5)	2.26 (33.0)
	6.61 (55.2)	- (-)	- (-)	- (-)
3	5.88 (53.0)	4.96 (62.2)	5.39 (61.3)	- (-)
	5.76 (51.0)	4.33 (61.8)	4.21 (60.8)	4.32 (61.6)
	5.68 (49.1)	4.10 (59.2)	4.00 (57.9)	3.99 (58.9)
	10.02 (81.6)	- (-)	- (-)	- (-)
4	8.91 (78.0)	7.60 (88.6)	8.18 (86.1)	- (-)
	8.77 (74.9)	6.64 (86.8)	6.40 (85.1)	6.51 (85.5)
	8.67 (71.9)	6.29 (81.5)	6.07 (79.6)	6.00 (80.3)

Table 3.2: *The mean (and maximal) prediction errors for different prediction methods on the frequency evolution of partials of a note of singing voice. Prediction errors are computed for several simple prediction methods (top chart) and the LP prediction method (bottom chart) for different values of k (the distance in frame indexes between the last observation and the predicted value). Concerning the part dedicated to the LP prediction method, the model order grows from left to right, and for each values of k the number of samples considered is in [4,8,16,32]. The prediction errors of the LP prediction method are lower than those of the best simple prediction method. The improvement is getting more and more significant when k is increasing. For each experiment, the mean error value is presented and long with the maximum error value (in parentheses). The Burg method performs generally better, especially at high orders.These results were presented in [LMRR03].*

we compare it with three straightforward prediction methods, the constant method, used in the extended McAulay-Quatieri algorithm, the linear method and the mirror method, respectively defined as:

- *Constant method*: $\hat{s}(n+k) = s(n)$,
- *Linear method*: $\hat{s}(n+k) = s(n) + (s(n) - s(n-1)) \cdot k$,
- *Mirror method*: $\hat{s}(n+k) = 2s(n) - s(n-k)$

where k is the distance in frames between the last observation and the predicted value. The mean and the maximum (in parentheses) on Tables 3.1 and 3.2 on the preceding page of the prediction error are considered. Prediction errors are computed for several simple prediction methods presented above (top part) and the LP prediction method (bottom part) for different values of k. Concerning the part dedicated to the LP prediction method, the model order grows from left to right, and for each values of k the number of samples considered is $[4, 8, 16, 32]$.

*
* *

The Burg prediction method is efficient for constant, sinusoidal, and exponentially increasing or decreasing signals. Concerning constant evolutions (frequency evolution of partials of a piano tone), the constant prediction method is sufficient. Concerning sinusoidal evolutions of the partials (vibrato and tremolo), provided that the number of samples is sufficient to get the vibrato period, the Burg prediction method shows great results both in mean and maximal errors especially when k is increasing. For exponentially increasing or decreasing evolutions of the partials (portamento and fade-in or fade-out) – the mean error of the LP prediction method is good comparing to the constant, linear, and mirror prediction methods, but the maximum error is comparable.

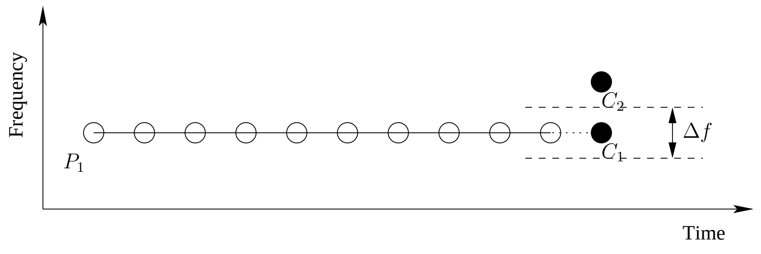
Using the Burg method makes then the peak choosing step of the sinusoidal analysis much more accurate. Figure 3.24 shows the expected results for two types of partials using Burg extrapolation. We can see also that we keep the Δf frequency interval to allow estimation errors in the prediction.

This better estimation of the peaks makes the partial tracking algorithm work better, especially when it comes to sounds with vibrato and/or tremolo.

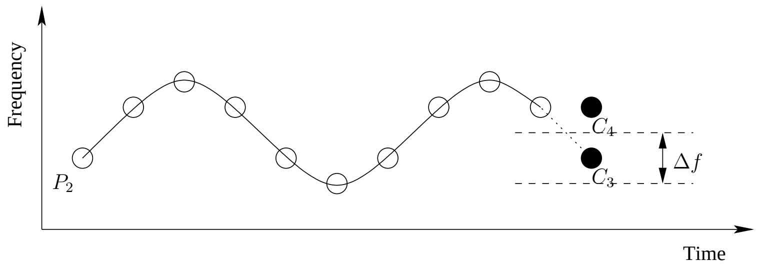
3.4 Synthesis of sounds from the sinusoidal model

The simple synthesis of sounds from sinusoidal model is actually quite straightforward, provided that computation time is not a concern. The main idea is to synthesise the sound from the sum of sinusoids at a given sample rate. This has been written earlier as:

$$s(t) = \sum_{p=1}^{P} a_p(t) \cos(\phi(t))$$



(a) Partial tracking on a constant trajectory



(b) Partial tracking on a sinusoidal trajectory

Figure 3.24: *Performing partial tracking with the Burg based method. Provided a sufficiently high order, it performs well for linear frequency trajectories and well for sinusoidal frequency trajectories. Contrarily to Figure 3.22 on page 48, on the sinusoidal track, with the Burg method, the peak C_3 is chosen, and thus the sinusoidal frequency trajectory of the partial is preserved.*

The parameters of the sinusoidal model however are not known under a continuous form. Indeed, the parameters were found using a STFT. Hence the known parameters are sampled at a sample rate of

$$F_s' = \frac{F_s}{h}$$

where h is the hop-size in samples and F_s is the sample rate the discrete-time form of s.

Now, in order to synthesise s from the discrete-time sinusoidal parameters, we have to find a way to obtain a continuous-time of the parameters (at least to compute $s[n]$ we need to go back to F_s).

To solve this issue, a first degree polynomial can be used for amplitude interpolation and third degree polynomial can be used for phase interpolation. This gives satisfactory results, and can hardly be enhanced by increasing the polynomial degrees [GMdM+03].

A problem might arise if the phases of the sinusoids of the signal are not known. In this case, we can incrementally recompute the phase of each partial p at each sound sample by a discrete approximation of Equation 3.23 on page 37, for n being any (positive) sample index:

$$\phi_p\left((n+1)T_s\right) = \phi_p\left(nT_s\right) + 2\pi f_p\left(nT_s\right)T_s$$

so that the relations between phases and frequencies are maintained, and then compute the complete sound by using Equation 3.4 on the facing page. This synthesis method however does not keep the waveform of the original signal if the $\phi_p(0)$ values are not known (even if the waveform has only little influence on the human perception of sound).

Other synthesis methods exist. One major interest in the field of synthesis is speed. In order to be able to synthesise many sounds in real time, it is important to have fast-synthesis methods. Such fast-synthesis methods have been recently studied by Robine *et al.*[RSM06].

3.5 Time-scaling with the classic sinusoidal model

Most sinusoidal synthesis methods can also be used to perform time-scaling. The method we found and present here uses resampling techniques.

Indeed, series of tests for several polynomial phase models can be found in the work of Girin *et al.* [GMdM+03], the McAulay-Quatieri method described above being the order-3 polynomial of this study. Among these tests, the third synthetic example shows sinusoidal evolutions for the frequencies (vibrato). The vibrato (sinusoidal variation of the fundamental frequency) is tested with and without tremolo (sinusoidal variation of the amplitude). It is clear that these sinusoidal evolutions cannot be perfectly approximated by polynomials of finite degrees like the order-3 polynomial phase model of the McAulay-Quatieri method. But provided that the frequency of a sinusoidal evolution remains below the Nyquist frequency, this evolution can be – in theory – perfectly reconstructed using the cardinal sine reconstructor (see Equation 3.46). That is the reason why we could expect an improvement in quality when using the resampling technique.

In order to achieve the goal of time scaling using the sinusoidal model, one solution is thus to resample the control signals present in the partials.

3.5.1 Resampling the parameters

Let $s(t)$ be a continuous signal, and $s[n] = s(\frac{n}{F_s})$ be the sampled version of this signal, with F_s being the sampling frequency of the signal. Moreover, $s(t)$ is band-limited to $\frac{F_s}{2}$. According to the Shannon-Nyquist theorem, $s(t)$ can be reconstructed from its sampled form $s[n]$ by convolving the discrete signal by a reconstructor – a windowed sinc function – in practice using an algorithm similar to the one proposed by Smith in [SG84, Smi00], except that we chose to use the Hann window instead of the family of Kaiser windows.

Reconstructing the parameters

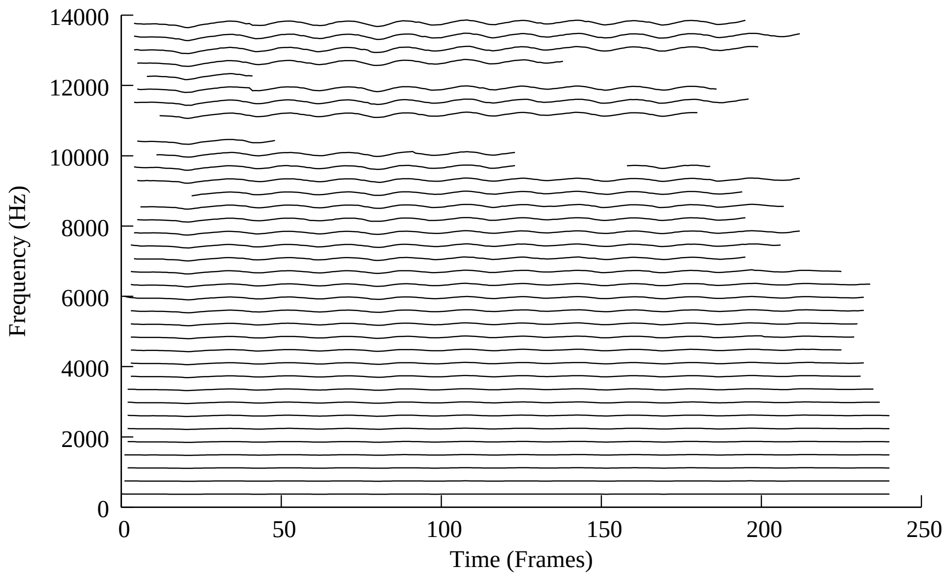
In theory, we consider the impulse train made of the samples of the discrete signal where they are known – at times multiple of the sampling period – and 0 (zero) elsewhere. According to the Shannon-Nyquist theorem, the continuous version of the signal s is reconstructed simply by convolving this impulse train by the ideal reconstructor, the *cardinal sine* – noted sinc – function[2] defined by:

$$\mathrm{sinc}(t) = \begin{cases} \frac{\sin(\pi t)}{\pi t} & \text{for } t \neq 0 \\ 1 & \text{for } t = 0 \end{cases} \tag{3.46}$$

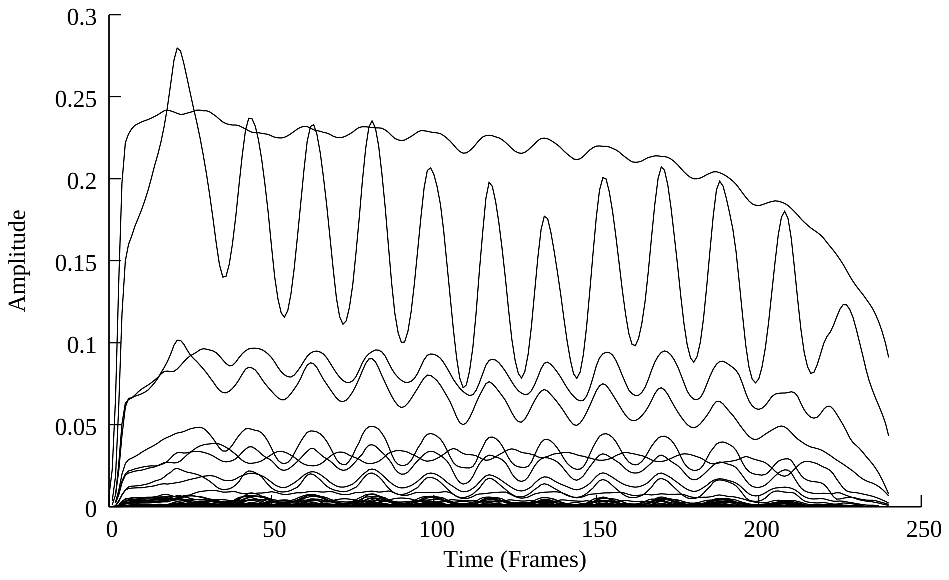
and

$$r_i = \mathrm{sinc}(F_s t) \tag{3.47}$$

[2]We could also multiply its spectrum by the box (rectangular) function.



(a) Frequencies



(b) Amplitudes

Figure 3.25: *Frequencies (a) and amplitudes (b) of the partials of an alto saxophone as functions of time (during approximately 2.9 s).*

Thus

$$s(t) = \sum_{n=-\infty}^{+\infty} s\left(\frac{n}{F_s}\right) r_i\left(t - \frac{n}{F_s}\right) \tag{3.48}$$

that is

$$s(t) = (s * r_i)(t) \tag{3.49}$$

In fact, the ideal reconstructor cannot be used because of its infinite time support, and we need instead a reconstructor of finite support. In the remainder of this section, let us denote by N this finite size expressed in samples. This size allows us to tune the trade-off of reconstruction quality versus computation time in the resampling process. We obtain this practical reconstructor by multiplying the ideal reconstructor by some window of finite support. We chose a symmetric Hann window of size N noted w_N.

The practical reconstructor is then given by:

$$r_p(t) = w_N\left(\frac{1}{2} + \frac{F_s}{N}t\right) \cdot r_i(t) \tag{3.50}$$

for $t \in \left[-\frac{N}{F_s}, \frac{N}{F_s}\right[$, $r_p(t) = 0$ elsewhere.

Using Equation 3.49, it is easy to obtain the continuous signal $s(t)$ from its sampled version. Once, we have the continuous function, upsampling by a factor u ($u \geq 1$) the signal s is straightforward since we can compute this function at any time, all the more at multiples of the new sampling period $\frac{T_s}{u}$. Upsampling is like considering the $s(\frac{t}{u})$ function. Downsampling s by a factor d ($d \geq 1$) is slightly more complicated, since high frequencies have to be filtered out in order to fulfil the Nyquist condition. Considering $F_m = \min(F_s, F_s')t$, this is done by defining the new reconstructor function as

$$r_i(t) = \operatorname{sinc}(F_m t)$$

and

$$r_p(t) = w_N\left(\frac{1}{2} + \frac{F_m}{N}t\right) \cdot r_i(t)$$

for $t \in \left[-\frac{N}{F_m}, \frac{N}{F_m}\right[$, $r_p(t) = 0$ elsewhere.

By resampling the control signals of the sound, we would have an easy way to modify the length of the sound without modifying the timbre of the sound.

Indeed, the timbre of an harmonic sound is given by the relative amplitudes of the harmonics of the sound (hence of the relative amplitude of the partials, derived from the spectral envelope), and thus resampling keeps this relation between amplitudes, since we resample all the partials by the same ratio.

However, a problem arises at the boundaries of a discrete signal. Indeed, this signal is of finite support and, during the convolution, we need some of its values before its beginning and after its end. This is usually zero for temporal signal, since a natural sound generally starts from silence and returns to silence, and that is not offset. However, for control signals, they might have to be extrapolated. One solution is to use the classic reflection method (*i.e.* for samples before the beginning, that is for any sample index i which is a negative integer, we define $s(i) = 2s(0) - s(-i)$). This ensures the continuity of both the signal and its first derivative. However, this is not the best way for extrapolating the signal. For the amplitudes (of the audio signal a, of the partials a_p, etc.), extrapolating

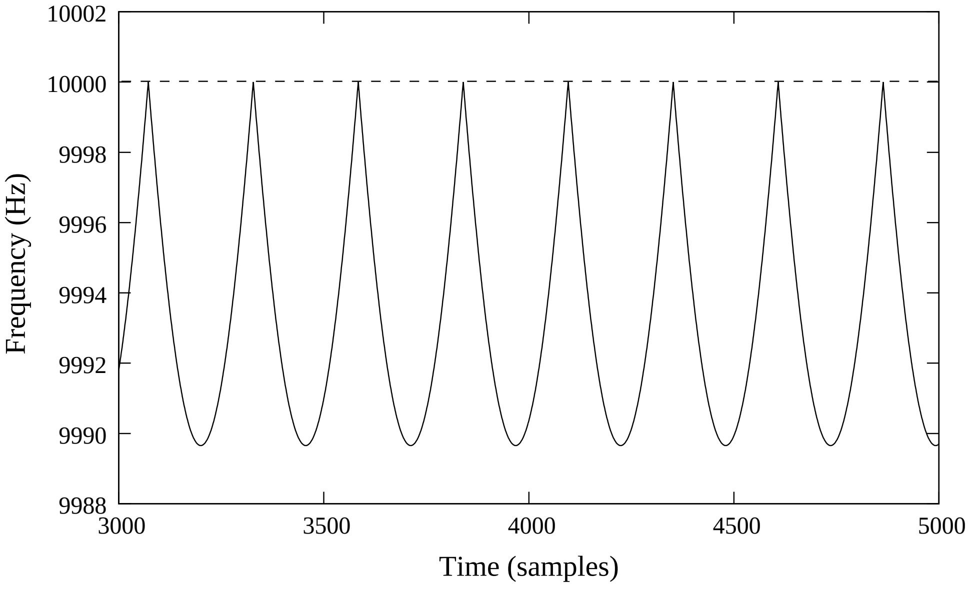


Figure 3.26: *Resampling using a finite reconstructor is not adapted for the resampling of non-zero-centred signals. Here is an example of the resampling of a constant signal centred on 10000 Hz (represented by a dashed line). The resampling ratio is 256. The result of this resampling is given as a solid line.*

the signal with 0 (zero) values seems to be a natural choice, when those signals fade in and out from and to zero. However this will smooth the attack and this zero-padding technique cannot be used for other kinds of signals such as the frequencies f_p or the phases ϕ_p of the partials. For this reason, we use the extrapolation with the Burg method as proposed in section 3.3.2 on page 47.

In theory resampling is working on every kind of signal. However, in practice, only careful resampling can give good results. Resampling, as explained earlier, is based on a windowed sinc function in order to be finite. With this finite version of the cardinal sine, artifacts may appear if the signal to resample is not zero-centred. Indeed, samples of the signal that are left out by the finite version of the cardinal sine. And if those samples are far from zero, their influence on the final resampled signal is not negligible. In fact, while the ideal (infinite) cardinal sine reconstructor considers the values of the entire signal to compute new samples, the practical (finite) reconstructor uses only the values closest to a given new sample. Hence, the influence of the signal samples that are not included for the reconstruction is neglected. Of course, the shape of the cardinal sine function makes the centre sample far more important, and the further from the centre the samples, the less significance and influence they have for the newly computed sample. When the signal to resample is zero-centred, the left out samples can indeed be neglected. However, as shown on Figure 3.26, for signals with a large offset, we may not find this influence negligible. In the latter case, if we added a sinusoidal modulation with an amplitude of 5 Hz on the signal, the resampling would have a disastrous effect, completely loosing the modulation in the artifacts of the resampling.

Control signals of the sound fall into this category of signals that are not zero-centred.

This implies that the resampling of control parameters will not be perfect.

In order to enhance the resampling, a simple trick is to subtract the mean of the signal to the signal itself. This way, the resulting signal is (locally) zero-centred. This will work for signals that are more or less stationary in time, such as the frequencies of the partials of a stationary note.

However, we noticed earlier that the frequency signal of a sound is the derivative of the phase signal. Hence, in the case where the frequency is stationary, the phase will be linearly increasing. Indeed, we consider in our study only positive frequencies. This is possible since the analysed signal is real, and thus its amplitude spectrum is symmetric around zero.

This linear term makes it impossible to have the phase signal close to zero at all points. Thus the problem mentioned earlier might reappear and create errors for the phase signal. Another trick would be to perform local zero-centring of the signal around the position of the signal we want to resample. The comparison between standard resampling and this local zero-centred resampling is shown on Figures 3.27 on the next page and 3.28 on the facing page.

According to these figures, the local zero-centering resampler performs a lot better than the traditional resampling technique. However, we have said before that the phase of the partials we want to resample are linear if the frequency is constant. Hence, in order to go even further in enhancement of the resampling, we have tried to center the signal around zero by substracting polynomials to the signal prior to resampling. This way, we perform the resampling only on the modulations of the signal, not on the envelope. The polynomials we have removed are linear (first degree), quadratic (second degree) and cubic (third degree). The comparison of the performances of all these resampling techniques is showed in Figures 3.29 on page 60 and 3.30 on page 61. It has to be noted that the error of the resampling technique using higher degree polynomial is lower only for low frequency signals, while being almost identical for higher frequency signals.

This enhancement is significant for our purpose of resampling the control signals since they contain mainly low frequencies.

One last point about resampling is about the windowing of the reconstructor function. As said earlier, the sinc reconstructor has to be windowed in order to be applicable in the finite case. Thus, we have compared the Kaiser-windowed sinc that is used in Matlab against the Hann-windowed sinc we use in our version of the resampler. The results are shown in the Figure 3.33 on page 64. For our purpose (resampling low frequency signals), the Hann-window reconstructor performs best. It has to be noted that the maximal error values seems to be describing the Kaiser and the Hann windows respectively, and that the width of the bumps seems to be related to the size of the window.

Resampling the frequency

Let us now denote by T_s the sampling period of the sound to be synthesised. The easiest way is to incrementally recompute the phase of each partial p at each sound sample by a discrete approximation of the frequency synthesis Equation 3.23 on page 37, for k being any (positive) sample index:

$$\phi_p((k+1)T_s) = \phi_p(kT_s) + 2\pi f_p(kT_s)T_s \qquad (3.51)$$

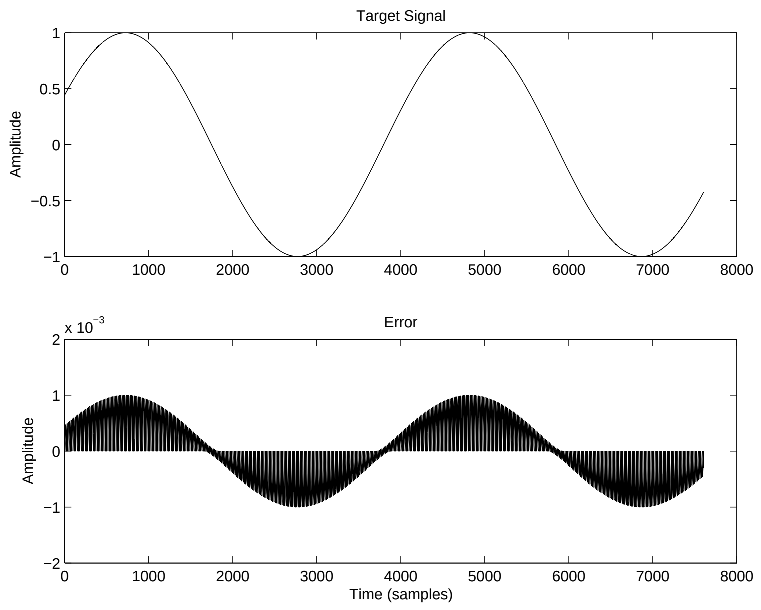


Figure 3.27: *Resampling a simple low-frequency sine function. The target signal was down-sampled, then resampled to the correct length. The resampling ratio does not influence the maximal error signal. The error depends on the amplitude of the signal to resample. The resampling errors at the boundaries have not been shown on this figure. The reconstructor has a size of 20 samples.*

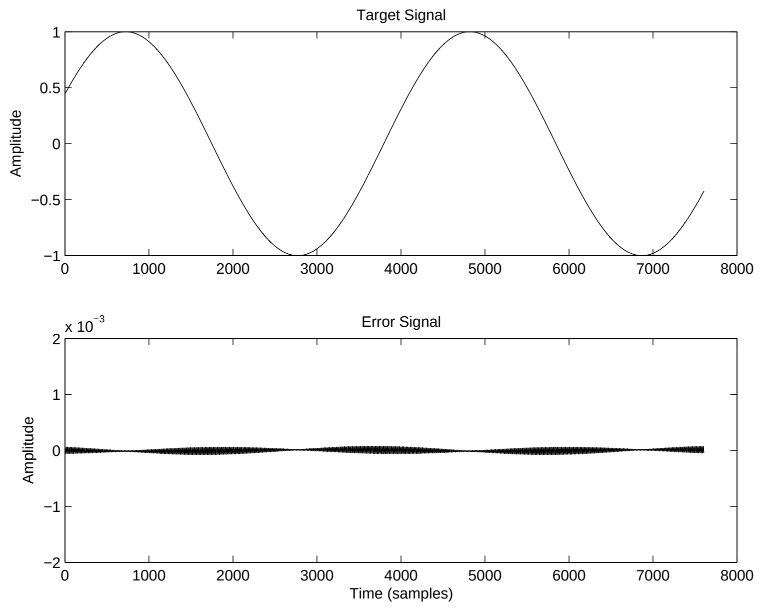


Figure 3.28: *Resampling a simple low-frequency sine function using local zero-centring. The target signal was under-sampled, then resampled to the correct length. The resampling ratio does not influence the error signal. The error depends on the amplitude of the signal after the removal of the local mean. The resampling errors at the boundaries have not been shown on this figure. The reconstructor has a size of 20 samples.*

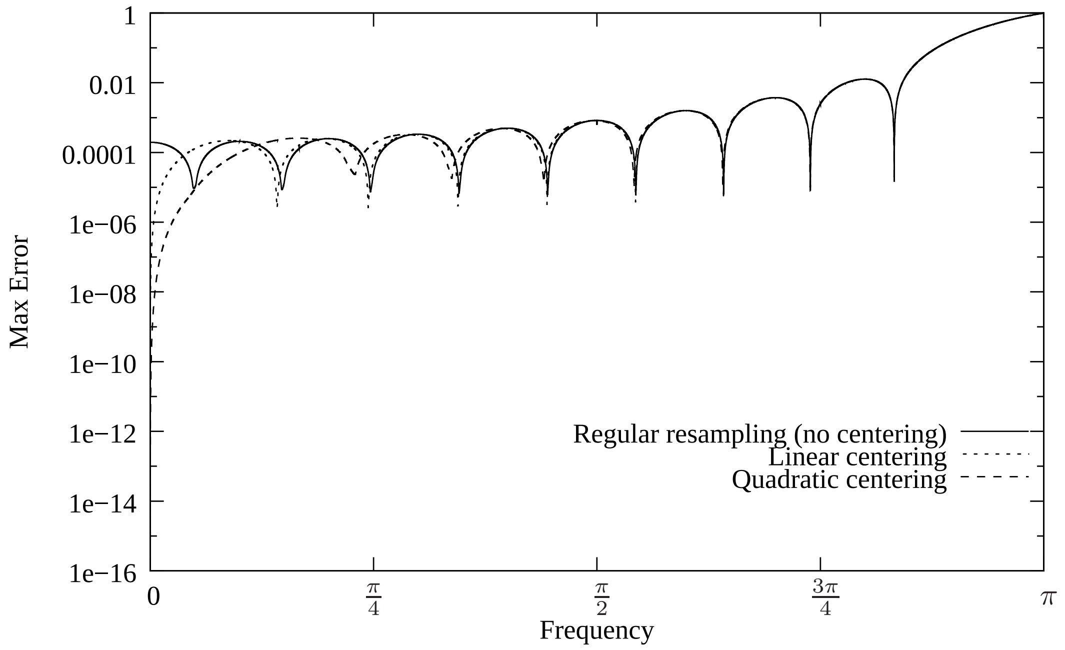


Figure 3.29: *Maximal resampling error using various resampling techniques plotted against the frequency of the sinusoid they have been applied on. The resampled signal is already zero-centred.*

so that the relations between phases and frequencies are maintained, and then compute the complete sound by using Equation 3.22 on page 37.

Since we need, for each partial p, the values of its frequency f_p and amplitude a_p at each sound sample, we have to up-sample these parameters using the technique previously described.

Indeed, in order to perform time scaling by a factor k during the synthesis, the frequency f_p and amplitude a_p of each partial can simply be resampled according to this k factor prior to the synthesis algorithm itself, to match the targeted length.

Since this technique does not take the measured values of the phase into account, except $\phi_p(0)$ at the time origin, the resulting sound has not the same shape as the original sound (see Figure 3.34(b) on page 65), although this makes no audible difference for most sounds.

On the contrary, using the phase for synthesis ensures a shape-invariant synthesis.

Resampling the unwrapped phase

In the same fashion that we resampled the frequency, we can resample the unwrapped phase. Indeed, if we consider the unwrapped phase as being a continuous signal, we can apply a cardinal sine function for resampling.

However, two problems arise. The first has been presented previously: Since the unwrapped phase has at least a linear component (since the frequency is its derivative), we cannot centre the phase around zero to minimise the effect of the finite (truncated) cardinal sine function. Hence, slight artifacts might occur. Another solution is to use the

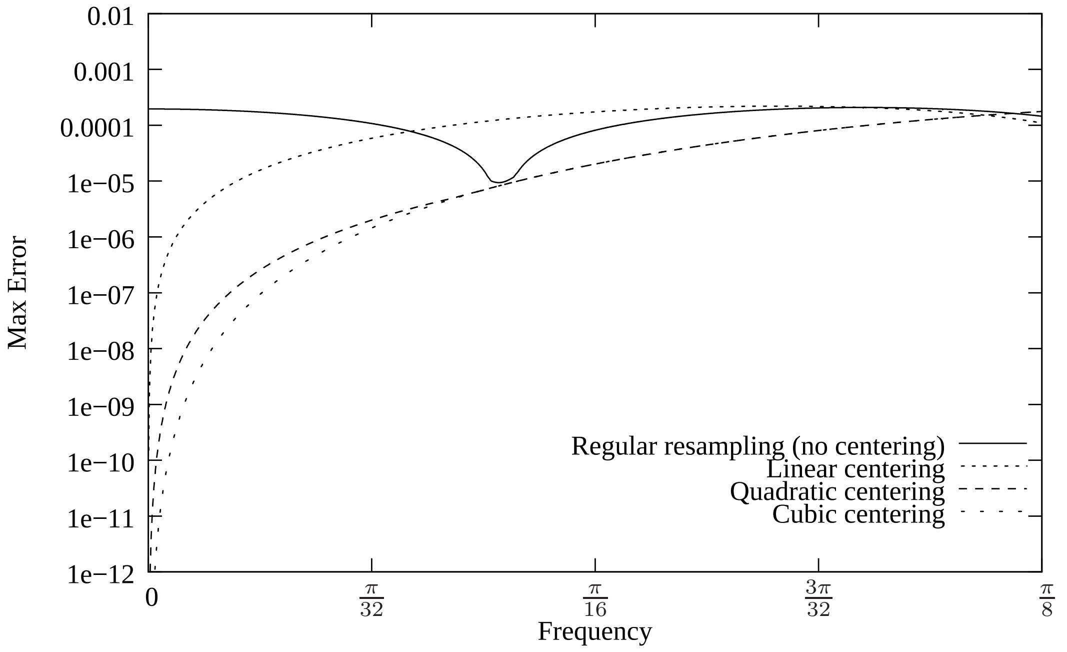


Figure 3.30: *Maximal resampling error using various resampling techniques plotted against the frequency of the sinusoid they have been applied on. We have zoomed on the lower frequency to show that higher order polynomials help enhancing the resampling. Mean-centering and linear-centering are almost identical, thus only mean-centering has been plotted here.*

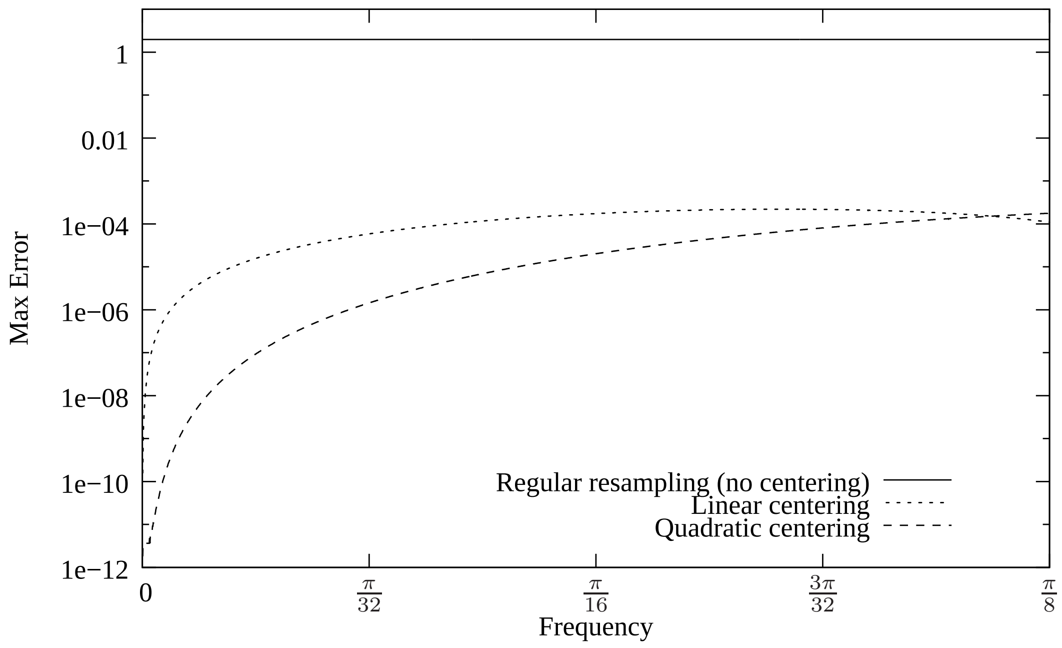


Figure 3.31: *Maximal resampling error using various resampling techniques plotted against the frequency of the sinusoid they have been applied on. The test signal is a sinusoid of given frequency shifted by a constant.*

resampling technique presented earlier which first substracts a polynomial to the signal prior to performing the resampling.

The same problem of linear components might appear also for the resampling of the amplitude, during the attack and decay of the sound, since these pieces of the control signal are not stationary.

The second problem is still about the linear component of the phase. Indeed, since the frequency is the derivative of the phase, resampling the linear term of the phase will change the gradient of the linear term, thus the frequency. Hence, considering that the phase is indeed close to linear (that is, that the frequency is close to constant), after the phase has been resampled, it is necessary to correct the trajectory of the phase. This correction is in fact just a multiplication of the new values of the phase by the scaling factor.

However, even if these tricks lower the error of resampling on phase trajectories, the problem is not yet solved. A solution would be to have a model for the phase trajectories in order to find a better way to resample.

Hence we will try to find a more satisfactory method to resample the partials. By finding a suitable model for the partial signals, we could use the model to resample the partials.

In the work of Girin *et al.*[GMdM+03], some synthesis methods are compared using the the Signal-to-Noise Ratio (SNR). The SNR measures the energy ratio between the original signal and the residual part (noise) obtained by subtracting the resynthesis from the original. The greater is the SNR, the more accurate is the phase model for the

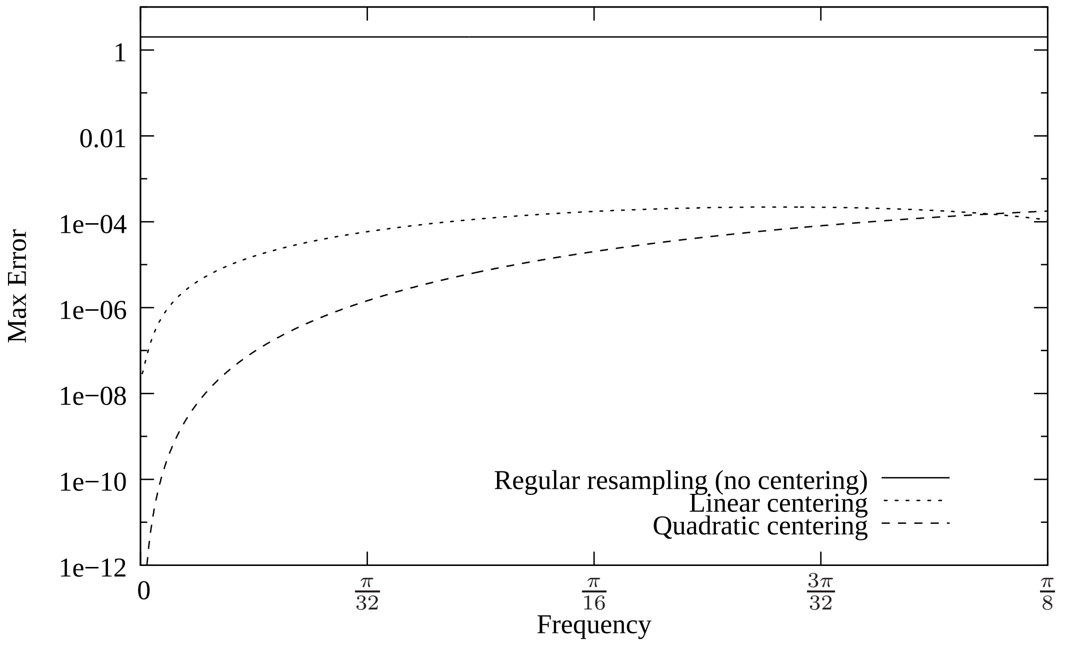


Figure 3.32: *Maximal resampling error using various resampling techniques plotted against the frequency of the sinusoid they have been applied on. The test signal is sinusoid of given frequency added to a second degree polynomial. Here the linear-centering method is not as good as on the previous figure, while the quadratic method still performs very well (in low frequencies).*

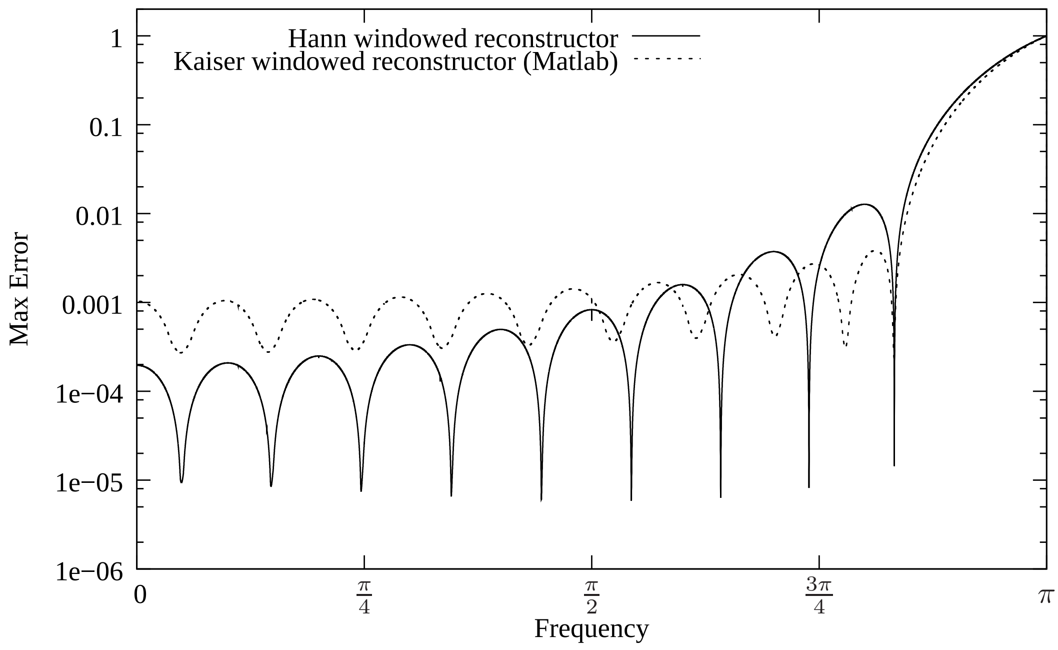


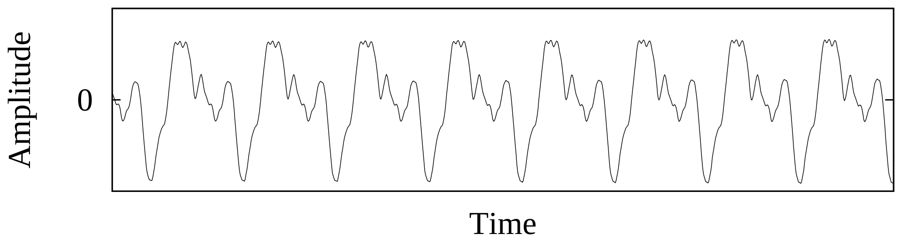
Figure 3.33: *Maximal resampling error using various resampling techniques. Here, Matlab's resampling technique using a Kaiser-windowed ($\beta = 5$) cardinal sine as reconstructor is compared to the technique using a Hann-windowed reconstructor over single sinusoids of varying frequencies. The reconstructor contains 10 wings (lobes) on each side.*

considered example. For the third – vibrato + tremolo – example, the SNR for the McAulay and Quatieri method is 76.21. For the phase-resampling method using Equation 3.50 with $k = 32$, we compute a SNR of only 21.04, but with $k = 256$ the SNR is then 76.61 and thus outperforms all the polynomial phase models listed in the work of Girin *et al.*[GMdM+03]. The SNR is a growing function of k, and quality is at the expense of computation time. However, large values of k are needed only for the phase. Indeed, the resampling of the amplitude parameter is not problematic: by resampling only the amplitude with $k = 32$ and using the ideal phase, the SNR would have been 110.32. The problem is with the imprecision on the reconstructed phase (without local centring), because of the resampling of its linear term (with a finite cardinal sine). Although the error remains very small in percentage, the large values of the unwrapped phase at the end of the partials lead to error in the magnitude of π (which, for phase values that are known modulo 2π, can be quite problematic). The synthesised signal is then from time to time put out of phase from the original one, thus the SNR is poor, but the perceived quality is still high. The SNR is indeed an imperfect perceptual metric.

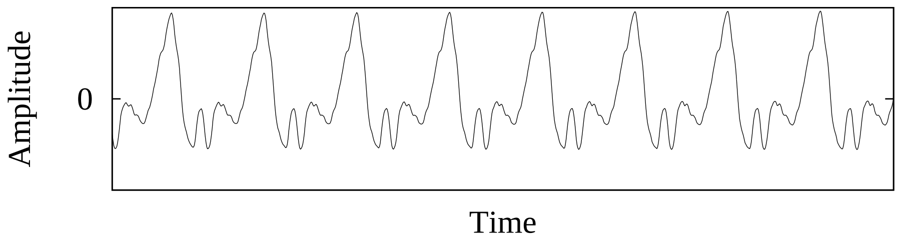
Using the resampling technique presented before, it might be however possible to solve this problem.

It has to be noted that the main advantage of phase resampling over frequency resampling is that the synthesis is then shape invariant, if we consider that the phase reconstruction is flawless.

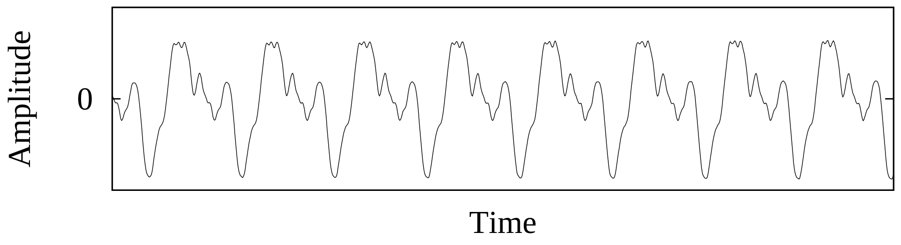
The results of such resampling for time-stretching is shown on Figure 3.35. We can see



(a) Original Signal

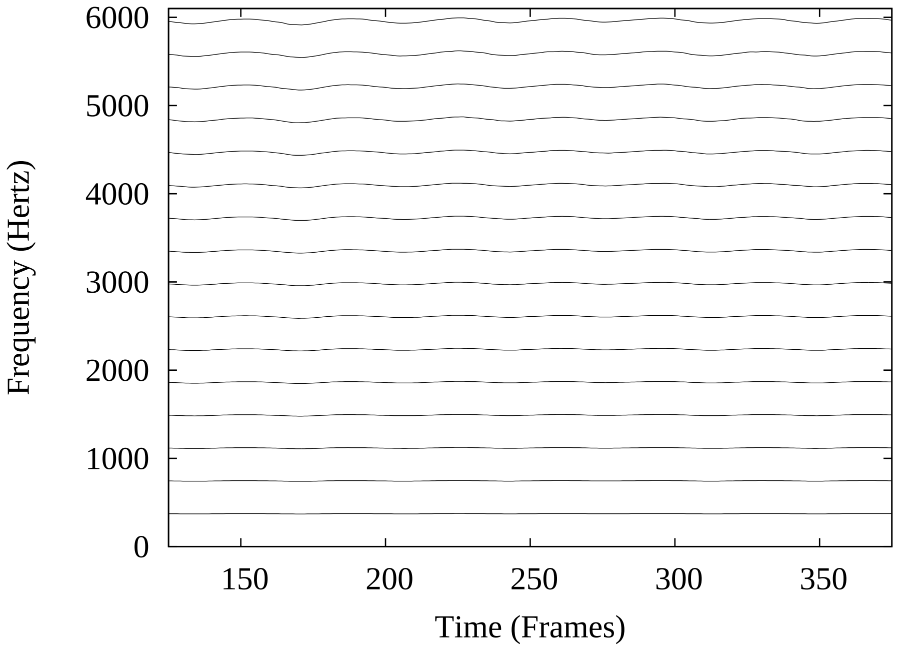


(b) Raw Synthesis

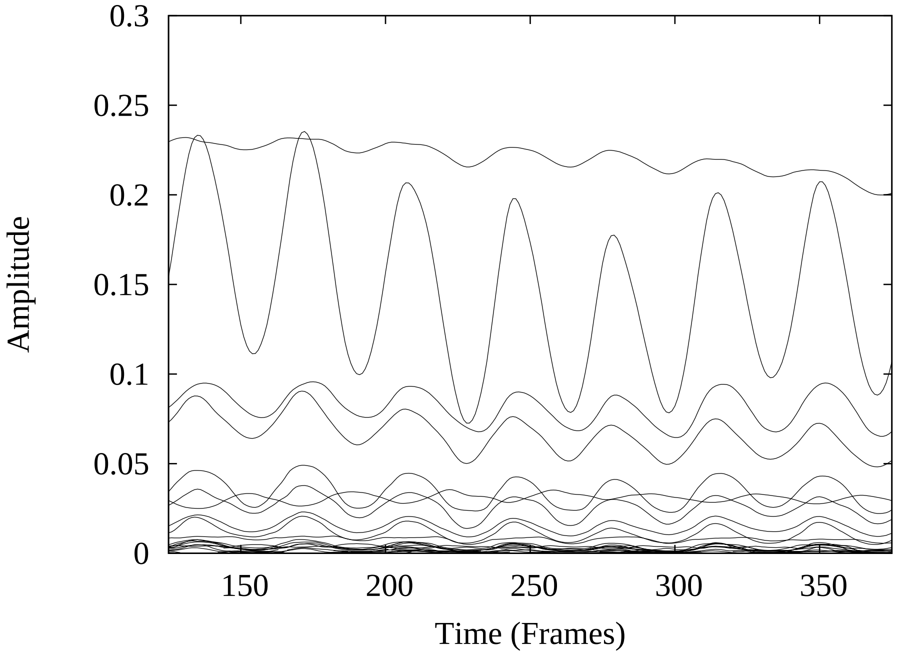


(c) Shape-Invariant Synthesis

Figure 3.34: *Raw (b) versus shape-invariant (c) synthesis methods. The latter is very close to the original signal (a).*



(a) Frequencies



(b) Amplitudes

Figure 3.35: *Normal time-stretching (order 1) of the saxophone sound, by a factor 2. The frequencies (a) and amplitudes (b) of the partials are shown at the middle of the resulting sound. The rates of the vibrato and tremolo have changed (see Figure 3.25 on page 55).*

that in comparison to Figure 3.25 on page 55 the modulations (vibrato and tremolo) have been slowed down, due to the stretching of the sound. Hence the vibrato and tremolo are indeed altered by time-stretching in the sinusoidal domain, even if the respective frequency of the partials has been properly conserved, as well as the amplitudes.

3.5.2 Non-uniform time scaling

The application we aimed for in this thesis is time-scaling. Using the techniques presented in this work, we have shown how to perform time scaling, conserving pitch and timbre. However, this time scaling is simply performed off-line and with a fixed rate all over the sound.

Several time scaling methods using sinusoidal models have existed for long now and are being enhanced by the addition of transient modelling in the sound [RÖ3]. However an interesting feature that has recently been developed in this domain is non-uniform time scaling [RSB05, BRR04].

In the works of Bogaards *et al.*[BRR04, Bog05], the presented software (Audiosculpt 2, made at the IRCAM, Paris) has a time-scaling method that allows non-uniform playing of the sound. Indeed, using a mouse as input device, it is possible to listen at any piece of the sound at any speed by moving the mouse over the spectrograms representing the sound. Doing so, it is even possible to “freeze” the sound. Hence, if the mouse is not moving, then the same small part of the sound is played in a stationary manner, only available if dealing with sinusoidal models.

These features are available with simple modifications to the synthesis methods of the sinusoidal model. In the case of the freeze feature, the resynthesis of the sound from sinusoidal parameters is simply based on a couple of frequency and amplitude parameters. Indeed, using the Equation 3.4 on page 52, it is easy to see that if the amplitudes and the frequencies stay constant, then the amplitude spectrum stays identical over time. Hence the sound stays the same for an infinite amount of time. This could be viewed as a sound-freeze indeed. However the modulations of the sound are not taken into account. If this freeze is applied to a sound containing vibrato and tremolo, the modulations will not be heard during the freeze state.

If freezing is possible, then it is also possible to time scale the sound in a non-uniform manner, based on the local speed of the mouse. The first solution would be to resample the sinusoidal parameters of the sound using the models presented earlier (see Section 3.5 on page 54) for each new local rate. The whole parameters do not have to be resampled, just the part that is being played. Indeed, the instructions from the input device being totally unpredictable, the only way to achieve good speed is to consider the parameters of the sound locally and resample them locally. Otherwise, it would take too much computation time to resample the whole signal at each input device event.

Another application of non-uniform time-scaling is the scaling preserving the attack of the sound. Indeed, after having identified the attack part of the sound, it might be possible to scale the sustain part of the sound in order to obtain a consistent result, with no artifacts on the attack.

This however would have the same drawback as the freeze, that is that the modulations of the sound (such as vibrato and tremolo) are not kept at the same rate.

In order to keep the modulations at a correct rate, it will be necessary to use the higher order sinusoidal model we will present in the next chapters.

3.6 Conclusion

We have presented in this chapter the basics of the sinusoidal model. We have also introduced some enhancements to the partial tracking technique. The linear prediction part has been done during my Masters research project in collaboration with Mathieu Lagrange, Sylvain Marchand, and Jean-Bernard Rault. This technique enhances significantly the partial tracking of vibrated signals.

Moreover we have developed enhancements to the resampling technique allowing low-frequency or uncentered signals to be resampled without too much error.

This technique is then used as a means to perform time-scaling using the sinusoidal model. Resampling the partials gives indeed good results for low-noise sounds. However, sounds containing vibrato are affected by this technique. The scaling indeed modifies the vibrato rate accordingly.

Hence, the next step is to find a solution for time-scaling sounds without scaling the modulations of the sound too.

Chapter 4

Order-2 Spectral Modelling

In the preceding chapter, we have shown the use of considering the control parameters of the sound as discrete-time signals, in particular for time-scaling without alteration of the pitch and timbre of the sound.

This approach allowed us to use standard signal tools such as resampling on the control parameters of the sound.

Let us now recenter the objectives of our work: We want an artifact-free time scaling-method. This implies no change in pitch nor timbre, as we have done in the preceding chapter. However, the resampling of the control parameters does not keep the modulations of the sound intact. For instance the vibrato of the sound is slowed down whenever the sound is stretched, and this can become noticeable if the stretching factor becomes too important. This shortcoming can however be dealt with.

Indeed, we show that these modulations of the sound can be identified and worked on.

The first step, covered in this chapter, is to model the partials. In the same way that we treated the classic spectral model by describing the spectral composition of the sound, we will here show how the control parameters of the sound can be decomposed (modelled). Hence, considering that the control parameters of the sound are discrete-time signals, we can use the usual modelling tools. We suppose that the partials are deterministic, meaning that they are represented by mathematical functions containing no random part. The idea is thus to find the mathematical functions (there are three sets of data in each partial: the frequency, amplitude, and phase trajectories) representing the partial, in other words we have to find a model for partials. Thus, we look at different models for the partials. The first one is a polynomial model. It was originally proposed by McAulay and Quatieri as part of the sinusoidal model. The second one, proposed with Laurent Girin and Sylvain Marchand, is based on sums of sinusoids. However, since both of the preceding models have advantages and limitations, we proposed a third model based on both polynomials and sums of sines. We then present the estimation of the parameters of this model and give some insight on the limitations of this model.

This estimation is based on the Fourier transform. Thus, we call this part of the analysis of the partials the *order-2 spectral model* since its principle is close to the classical spectral modelling, and since it is applied to higher-order (higher-level) sound parameters.

Determining the mathematical function behind the control parameters of the sound has interesting perspectives, both on partial tracking and on partial classification.

By the way, this process is the first step towards a sinusoidal modelling of the control parameters of the sound that we will introduce in the next chapter.

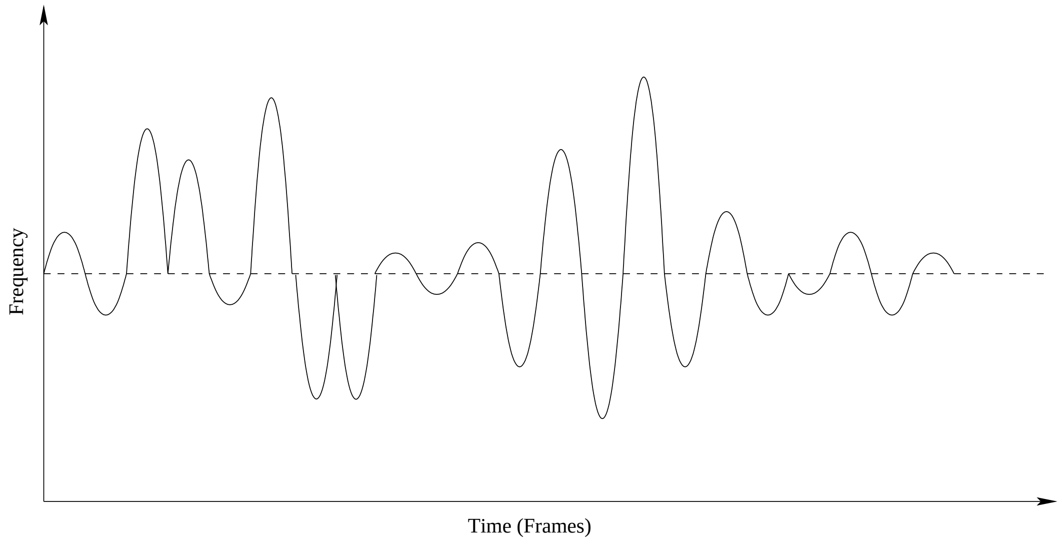


Figure 4.36: *Frequency track obtained using a cubic phase interpolation. Oscillations on the frequency track are created.*

4.1 Modelling the parameters

From the observation of the control parameters of the sound in its sinusoidal form, it is possible to assume that these control signals are deterministic. From the definition of a deterministic signal, we can say that there must be a randomless mathematical definition of this signal. The objective of modelling the control parameters of the sound is to find the mathematical function behind these signals.

Moreover, one of the aimed applications of this work is time scaling with minimal alteration of the timbre and the modulations in the generated sound. The way to perform time scaling using the sinusoidal model is to take each parameter signal of the partials and extend them to the desired length. This cannot be done correctly unless there exists a model of these parameters.

4.1.1 Modelling the parameters as polynomials

McAulay & Quatieri [MQ86] suggested that the phase trajectories of the partials would be modelled as third degree polynomials. Indeed, they used cubic interpolation between phase intervals, since four constraints are known. These constraints are the phases at the two successive frames and the frequencies at the two frames. Equations 3.24 on page 38 and 3.27 on page 38 show how the constraints are included in the calculation of the third-degree polynomial. If these four constraints are fulfilled, the phase and the frequency trajectories of the partials will both be continuous.

This third-degree polynomial interpolation gives good results in most cases. Indeed, when this third degree polynomial is applied to short pieces of signal, the polynomial fits well to the given phase trajectory. This is generally the case for sinusoidal analysis, and thus this method is widely used nowadays.

However, this cubic phase interpolation has a problem presented by Ding & Qian

[DQ97]. Indeed, constraining a third degree polynomial between two values and their derivatives can create an oscillation problem in frequency (derivative) trajectories (Figure 4.36 on the facing page). They argue that frequency and phase measurements are in general not consistent, and that constraining a third degree polynomial into corrupted data can create artifacts. In order to find a solution to this problem, Ding & Qian proposed to replace the third degree polynomial interpolation by a quadratic-spline- based interpolation. The estimation of the parameters of this model is done using a least-square error minimisation. They also integrate a confidence factor for phase and frequency measurements.

Hence we suppose that the phase function at the nth frame is:

$$\phi_n(t) = a_n + b_n t + c_n t^2 \tag{4.52}$$

This function is a piecewise second degree polynomial, and thus can be expressed by a combination of quadratic basis spline functions (B-splines), such that:

$$\phi(t) = \sum_{m=-2}^{N-1} \alpha_m B_m(t) \tag{4.53}$$

where $B_n(t)$ is the nth quadratic B-spline function. The coefficients α_n are then found by minimising the squared error:

$$\epsilon = \lambda \sum_{n=0}^{N} \left(\phi(t_n) - \hat{\phi}_n\right)^2 + (1-\lambda) \sum_{n=0}^{N} \left(\omega(t_n) - \hat{\omega}_n\right)^2 T^2 \tag{4.54}$$

where $\hat{\omega}_n$ and $\hat{\phi}_n$ are respectively the frequency and the phase estimates at frame n.

The weighting factor λ can be viewed as the relative importance of the phase mismatch to the frequency mismatch. Also,

$$\begin{aligned} \phi(t_n) &= \alpha_{n-2} B_{n-2}(t_n) + \alpha_{n-1} B_{n-1}(t_n) && (4.55) \\ &= \frac{1}{2}(\alpha_{n-2} + \alpha_{n-1}) && (4.56) \end{aligned}$$

and

$$\begin{aligned} \omega(t_n) &= \alpha_{n-2} B'_{n-2}(t_n) + \alpha_{n-1} B'_{n-1}(t_n) && (4.57) \\ &= \frac{1}{T}(-\alpha_{n-2} + \alpha_{n-1}) && (4.58) \end{aligned}$$

Hence, the squared error can be rewritten as

$$\epsilon = \lambda \sum_{n=0}^{N} \left(\frac{1}{2}(\alpha_{n-2} + \alpha_{n-1}) - \hat{\phi}_n\right)^2 + (1-\lambda) \sum_{n=0}^{N} \left((-\alpha_{n-2} + \alpha_{n-1}) - \hat{\omega}_n T\right)^2 \tag{4.59}$$

and can then be found using the classical least-square method.

This gives better results than the McAulay & Quatieri method. However, this method does not eliminate completely the oscillations in the frequency trajectories.

*
* *

Another method for avoiding the strange behaviour of the frequency track is to take into account the derivative of the frequency. Girin *et al.*[GMdM+03] tried to include the derivative of the frequency track in the estimation of the interpolating polynomial. This raises the polynomial up to the fifth degree:

$$\phi(n) = \phi_k + \omega_k n + \frac{\psi_k}{2}n^2 + \alpha n^3 + \beta n^4 + \gamma n^5 \tag{4.60}$$

As in equations 3.24 on page 38 and 3.27 on page 38, ϕ_k and ω_k represent the phase and frequency at $n = 0$. ψ is then the derivative of the frequency ω, thus it is the second derivative of the phase ϕ. ψ_k is its value at $n = 0$.

There are then six constraints to fulfil:

$$\begin{aligned}
\phi(N) = &\quad \phi_k + \omega_k N + \tfrac{\psi_k}{2}N^2 + \alpha N^3 + \beta N^4 + \gamma N^5 &= \phi_{k+1} + 2\pi M \qquad (4.61)\\
\omega(N) = &\quad \omega_k + \psi_k N + 3\alpha N^2 + 4\beta N^3 + 5\gamma N^4 &= \omega_{k+1} \qquad (4.62)\\
\psi(N) = &\quad \psi_k + 6\alpha N + 12\beta N^2 + 20\gamma N^3 &= \psi_{k+1} \qquad (4.63)
\end{aligned}$$

Solving these constraints gives:

$$\begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} = \begin{bmatrix} \frac{10}{N^3} & \frac{-4}{N^2} & \frac{1}{2N} \\ \frac{-15}{N^4} & \frac{7}{N^3} & \frac{-1}{N^2} \\ \frac{6}{N^5} & \frac{-3}{N^4} & \frac{1}{2N^3} \end{bmatrix} \begin{bmatrix} \phi_{k+1} - \phi_k - \omega_k N - \frac{\psi_k}{2}N^2 + 2\pi M \\ \omega_{k+1} - \omega_k - \psi_k N \\ \psi_{k+1} - \psi_k \end{bmatrix} \tag{4.64}$$

As in the case of the third-degree polynomial, the unwrapping factor M is chosen to obtain a "maximally smooth" frequency trajectory. Hence

$$M = i\left[\frac{1}{2\pi}\left((\phi_k - \phi_{k+1}) + (\omega_k + \omega_{k+1})\frac{N}{2} + (\psi_k - \psi_{k+1})\frac{N^2}{40}\right)\right] \tag{4.65}$$

However, this increase in the order of the polynomial, even though taking one more constraint into account, does not make a big difference on the Signal-to-Noise Ratio (SNR) of the resynthesised signal. Indeed, a fifth-degree polynomial does only add 2.35 dB to the SNR at most compared to the third degree polynomial. These results were obtained with natural sounds, such as human voices and instruments.

*
* *

On longer pieces of signal, the fitting of a polynomial can be much more of a problem. As long as the sound is stationary, the polynomial will fit well, but in cases of non-stationarity of the sound, it does not fit at all. For example, a sound containing vibrato will have an oscillating phase trajectory. This oscillation can lead to dramatic fitting problems for the third-degree polynomial. Indeed, third-degree polynomials cannot approximate an oscillating function such as the sine function for more than a period. Hence, if we want to model the entire phase trajectory of a sound, the polynomials cannot be used safely, at least not at the third degree.

Moreover, polynomial interpolation has another drawback that might be of importance. Even if the fitting with the phase curve gave good results, polynomials are not suited for extrapolation since they can produce out-of-range values.

Hence, even though the polynomial fitting gives good results on short pieces of phase trajectories, it is not correct and does not work on longer pieces of partial trajectories. Thus, it is not a good idea to model the control signals of the sound as polynomials.

4.1.2 Modelling the parameters as sums of sines

From the observation of Figure 3.20 on page 42, we can also decide to model the partial trajectories based on the oscillating phenomenon and the mean of the control signal. The estimation of the parameters for the oscillation and the mean of the control signal can be given by performing a Fourier transform on the partial trajectories. Indeed, the first bin of the Fourier transform gives the amplitude of a "flat cosine" function (constant function) in the signal, which would be used to compute the mean of the signal over the Fourier analysed part of the signal. The other bins of the Fourier transform give amplitudes, frequencies, and phases of the control signal.

Such an approach was presented by Girin *et al.*[GFM04, FGM05] for phase modelling of voiced speech. They consider the whole voiced speech partial as a basis for their modelling. This is what they call "long-term modelling". The proposed model is the following:

$$\phi(n) = \sqrt{\frac{2}{N}} \sum_{p=0}^{P-1} c_p w(p) \cos\left(\left(n + \frac{1}{2}\right)\frac{p\pi}{N}\right) + c_P n \tag{4.66}$$

This model is called the Linear + Discrete Cosine Model (LDCM). Indeed, equation 4.66 is composed of two parts. The first part of the equation is a sum of discrete cosine functions and the second part is a linear term. The linear term was introduced due to the fact that the mean frequencies of the signal are considered constant and that the phases thus have a linear component (since the frequencies are the derivatives of the phases).

After the measurements of the phase are performed, using a Fourier transform, the unwrapping of the phase values is done in the same fashion as in section 3.1 on page 37.

Then, in order to compute the c_p coefficients of the model, a standard MMSE minimisation is used. With

$$C = (c_0, c_1, \ldots, c_P)^T$$

being the coefficients of the model,

$$\phi = (\phi_1, \phi_2, \ldots, \phi_K)^T$$

being the unwrapped phase measurements, defining

$$A = \sqrt{\frac{2}{N}} \begin{bmatrix} \frac{1}{\sqrt{2}}\cos\left(\left(n_1 + \frac{1}{2}\right)\frac{\pi}{N}\right) & \frac{1}{\sqrt{2}}\cos\left(\left(n_1 + \frac{1}{2}\right)\frac{2\pi}{N}\right) & \cdots & \frac{1}{\sqrt{2}}\cos\left(\left(n_1 + \frac{1}{2}\right)\frac{P\pi}{N}\right) & n_1 \\ \frac{1}{\sqrt{2}}\cos\left(\left(n_2 + \frac{1}{2}\right)\frac{\pi}{N}\right) & \frac{1}{\sqrt{2}}\cos\left(\left(n_2 + \frac{1}{2}\right)\frac{2\pi}{N}\right) & \cdots & \frac{1}{\sqrt{2}}\cos\left(\left(n_2 + \frac{1}{2}\right)\frac{P\pi}{N}\right) & n_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{\sqrt{2}}\cos\left(\left(n_K + \frac{1}{2}\right)\frac{\pi}{N}\right) & \frac{1}{\sqrt{2}}\cos\left(\left(n_K + \frac{1}{2}\right)\frac{2\pi}{N}\right) & \cdots & \frac{1}{\sqrt{2}}\cos\left(\left(n_K + \frac{1}{2}\right)\frac{P\pi}{N}\right) & n_K \end{bmatrix}$$

as the matrix of DCT terms evaluated at the components of N, the mean-square error minimisation induces the following equation:

$$C = \left(A^T A\right)^{-1} A^T \phi \tag{4.67}$$

Hence, solving this equation gives the coefficients of the model.

This model yields good results, according to some SNR measurements. The order of the model has great effect on the accuracy of the modelling, and an order of 20 seems to give satisfactory results for partials of half a second.

However good this model is, it has to be reminded that our work is focused on musical signal, while this LDCT model is focused on voice signals. Indeed, voice signals have different frequency modulation than musical signals might have.

Moreover, the problem of the glissando is not addressed in the LDCT model. Indeed, the glissando is an increase of the frequency of a signal over time that can happen in some musical phrases. Hence, the frequency does not have a constant mean anymore, and thus the linear term of the LDCT model will not fit anymore for the phases of a signal containing a glissando.

Another problem of a model based on sines is that it might not be suited for amplitude trajectories of the partials of a signal. Even though on the sustained part of signal it would be possible to model the amplitude trajectories with sines, the attack, decay, and release of the sound might be problematic.

Hence, it is possible to imagine a more general model suited to partial trajectories.

4.1.3 Modelling the parameters using sums of sines and polynomials

As seen in the previous section, using a sum of sines to model partial trajectories will not be generic enough. The polynomial modelling also has its drawbacks. Indeed, the polynomials are not good to estimate sinusoidal trajectories of partials. Moreover, interpolating phase trajectories with polynomials might create an unpleasant oscillating behaviour.

Hence, one solution would be to use both models at once to solve the genericity problems. Indeed, in the case of an oscillating frequency, the sinusoidal model would fit perfectly (as seen on Figure 4.37 on the facing page, and in the case of a glissando, the polynomial model will perform very well. Thus, both are needed in the model in order to be able to handle every case. Then, we can describe a new Poly-Sin model as:

$$s(t) = \Pi(t) + \sum_{p=1}^{P} a_p(t)\cos(\phi_p(t)) \tag{4.68}$$

where $\phi_p(t)$ is given in equation 3.23 on page 37 and $\Pi(t)$ is a polynomial.

In order to extract $\Pi(t)$, we use a least-square error minimisation. The aim of the least squares method is to minimise the squared error of the polynomial approximation. So let N points located at positions $(t_k, s(t_k))$ with $k = 0 \ldots N-1$ being the sampling of the function s. We wish to find a globally-defined function $\widetilde{s}(t_k)$ that approximates the given values $s(t_k)$ at points t_k in a least-square sense (with an error $\overline{s}$), that is

$$\overline{s} = \min_{\widetilde{s}\in\Pi_m} \sum_{k=0}^{N-1} \left(\widetilde{s}(t_k) - s(t_k)\right)^2 \tag{4.69}$$

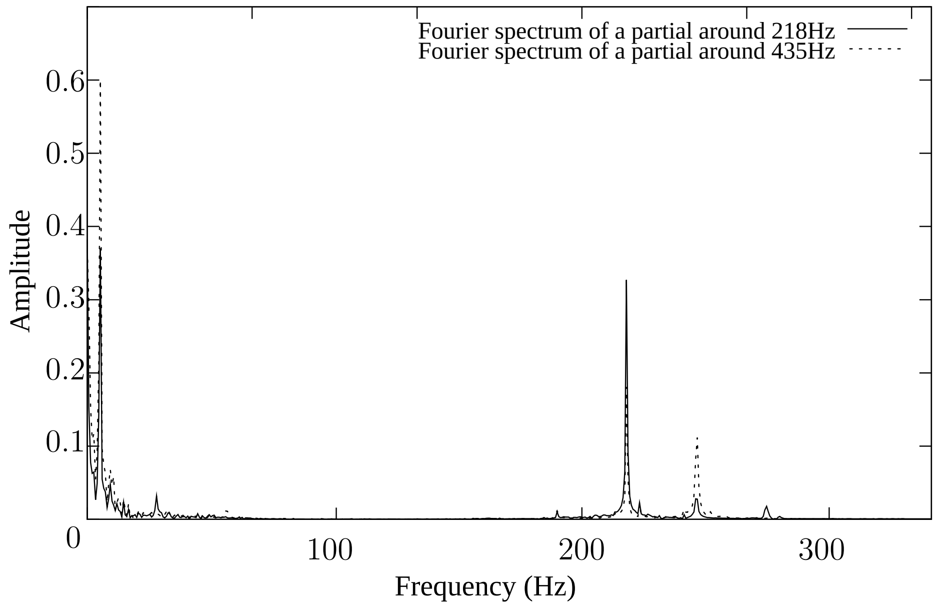


Figure 4.37: *Fourier spectra of the frequency signals of two partials of a same sound (with vibrato). It is clear that sinusoids in the sound can be detected from the spectral peaks that are displayed.*

where Π_m is the set of polynomials of total degree m and $\widetilde{s}$ can be written as

$$\widetilde{s}(t_k) = \pi(t_k)^T A \tag{4.70}$$

where $\pi(t_k) = \left[1, t_k, t_k^2, \ldots, t_k^p\right]^T$ and $A = [a_0, a_1, \ldots, a_p]^T$ is the vector containing the coefficients of the polynomial we are looking for, and p is the degree of the approximating polynomial.
In other terms,

$$\widetilde{s}(t_k) = a_0 + a_1 t_k + a_2 t_k^2 + \ldots + a_p t_k^p \tag{4.71}$$

so, the function to minimise is

$$\overline{s}(a_0, a_1, a_2, \ldots, a_p) = \sum_{k=0}^{N-1} \Big(\pi(t_k)^T A - s(t_k)\Big)^2 \tag{4.72}$$

A necessary – but not sufficient – condition to identify the minimum in a N-dimension space is

$$\nabla \overline{s} = 0 \tag{4.73}$$

In other words

$$\frac{\partial \overline{s}}{\partial a_0} = \frac{\partial \overline{s}}{\partial a_1} = \frac{\partial \overline{s}}{\partial a_2} = \ldots = \frac{\partial \overline{s}}{\partial a_p} = 0 \tag{4.74}$$

Then, the equation to solve is

$$MA = B \tag{4.75}$$

with $M = \sum_k \pi(t_k)\pi(t_k)^T$ and $B = \sum_k \pi(t_k)s(t_k)$.

For further details, see for example [Nea04].

A Fourier transform of the signal was performed on the residual after the least-square estimation in order to determine the sinusoidal part of the signal.

Now that we have at our disposal two estimation methods, we can perform the short-term analysis of our signal to obtain time-dependent parameters of our new model. The corresponding analysis will be windowed so that the locality of the analysis is kept. For each window the analysis will be twofold. First we find the best-fitting third-degree polynomial[1] using the least-square regression discussed earlier. The solving of the matrix system (equation 4.75 on the preceding page) gives us the coefficients of our polynomial. We then subtract this polynomial from our signal and proceed to the second step of our analysis, which consists of the Fourier analysis of the residual.

The method has to be a fine tuned, considering that modulations have a frequency of less than 2 or 3 Hz are polynomials, while the modulations of higher frequency are considered as sinusoids. This separation has to be done manually since the "base" composed of both sinusoids and third degree polynomials is overcomplete. From this assumption, we can deduce that the window size should be long enough to allow a Fourier transform to find the modulations (thus containing at least 1 period of the modulations), but it has to stay short enough to remain a short-term analysis.

This twofold estimation method works even better when applied several times in a row over the same piece of signal. Indeed, by re-evaluating the polynomial after the sinusoidal part has been estimated and removed from the original signal, the polynomial is getting more precise. Repeating this operation four or five times gives very good results.

This separation of the envelope and the modulation of partial parameters is somewhat similar to the decomposition of Arfib and Delprat [AD98, AD99] consisting of a pitch curve and a modulation component. This work will be discussed in more detail in Section 5.1 on page 107.

One major drawback of this estimation method is the poor resolution of the Fourier transform due to the time-frequency tradeoff. Hence, we explored the perspective of High-Resolution (HR) methods, which perform better in this prospect.

4.1.4 High-resolution estimation method

Decomposing the signal as a sum of sinusoids and a third order polynomial is not always reliable. Since the sine functions can be used as a base for signals, adding third degree polynomials is making the "base" overcomplete. Indeed, we can use either the polynomial or the sines to describe slow varying signals.

Hence a choice has to be made. We decided that the slow varying envelope of the signal would be described as a polynomial, and that the higher frequency modulation would be described as a sum of sines.

It has been decided that everything falling under 3 Hz would be considered as the envelope of the control signal. Then, we have to find the right tool to separate the signal in these two parts: envelope and modulations.

Until now, we have used a global least-square method to find the envelope and a Fourier transform to find the modulations, without really caring about the 3 Hz limit. The ideal

[1]We consider that the phase trajectory of a sound without modulation can be correctly modelled using third-degree polynomials.

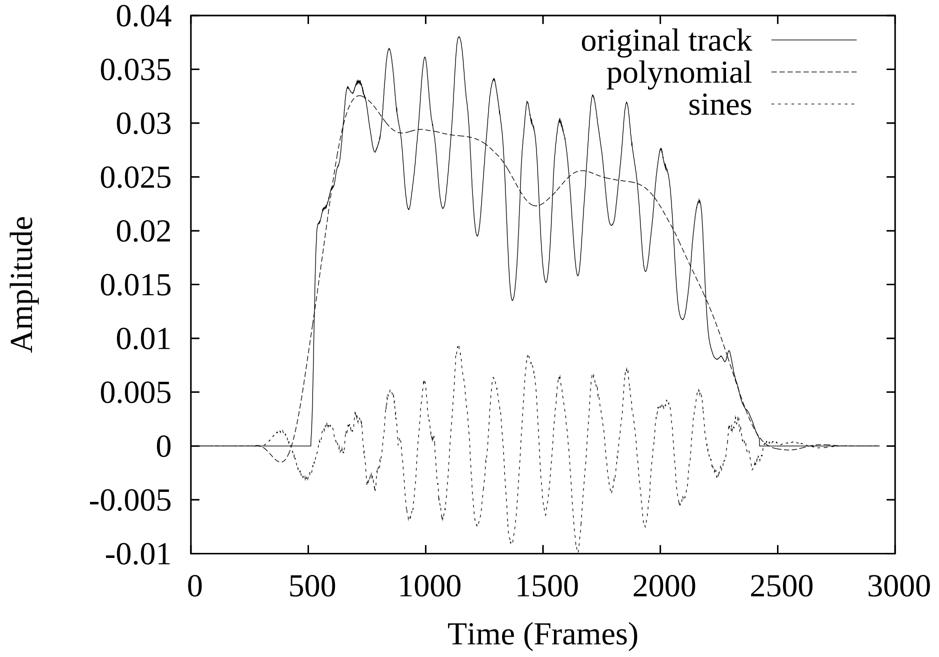


Figure 4.38: *Decomposition of the amplitude of a partial (control signal) showing the two steps of an analysis using a polynomial and sinusoids. The (order-2) frames are estimated at each sample of the evolution of the (order-1) partial, using 256-sample windows.*

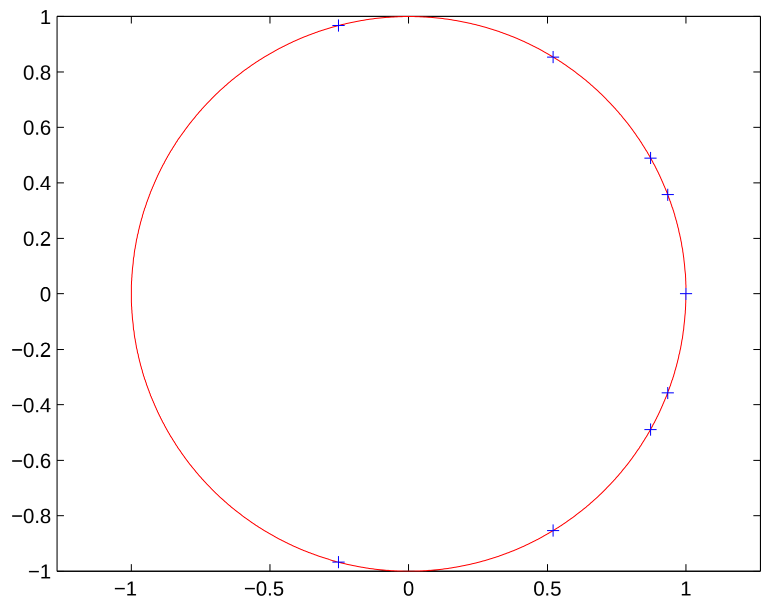
solution would be to have a method that estimates both sines and polynomials of a signal at the same time and that we could decide which is which, based on the frequency of the coefficients we found.

Such a solution is available through the High-Resolution (HR) methods (see [Bad05]). Indeed, the method finds a given number of complex exponentials in the signal and the frequencies of these exponentials are known. The complex exponentials can either express sine functions or polynomials. This comes in very handy in our case, since we decide that the exponentials under 3 Hz belong to polynomials, while the other belong to sinusoids. It is then possible to compute the coefficients of both the sines and the polynomial.

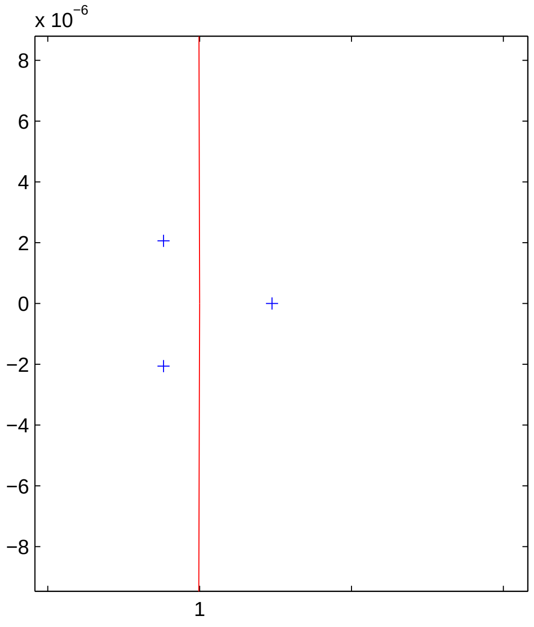
Such an approach is ideal for our problem. However, the High-Resolution method has a major drawback, which is the prior knowledge of the number of complex exponentials in the signal. Since this is not known for natural measured signals, we need an estimation method for the number of exponentials in the sound.

In earlier stages of the research, while working on synthetic examples, the estimation was not a problem, since we generated these signals, we knew how many exponentials would be contained in the signal. However, this is not the case for general signals.

Some estimation methods have been proposed, but none seem to work in our case. Hence we arbitrarily decided that the signal would contain a third degree polynomial and four sinusoids, and this would then give twelve exponentials to find in the signal. Indeed, each complex exponential (corresponding to a sinusoid) has two poles on the unit circle, symmetrical about the x-axis, and the polynomial has $n+1$ poles around the null frequency point $(1 + 0 \times i)$, where n is the degree of the polynomial.



(a) The entire complex circle



(b) Zooming close to the null frequency, the point $1 + 0 \times i$

Figure 4.39: *Displaying the frequencies of the complex exponentials of a synthetic signal on the complex circle. The first figure shows all the poles of the signal on the complex circle. One sinusoid corresponds to two complex exponentials, thus to two poles symmetrical about the x-axis. The polynomials have their poles close to the* $1+0\times i$ *point (which corresponds to the null frequency), in a number equal to the degree of the polynomial present in the signal plus one, and seem to form a regular polygon.*

The HR method we used here is the ESPRIT method [RPK86], which has been fully described in the PhD of Badeau [Bad05].

The results given by this methods were very promising at first. Indeed, on synthetic signals, the results were very good. Moreover, the HR methods have the big advantage of being far more accurate than a Fourier transform over fewer samples. On the synthetic example given on Figures 4.40 on the following page, the resulting error is indeed very small.

However, on real control signals, the result is not as good. On the beginning and on the end of the control signals, it has been found that the model does not fit correctly to the signal. Indeed, on Figures 4.41 on page 81 and 4.43 on page 83, we can see that the error signal is of the same order of amplitude as the original signal. This is a consequence of another drawback of the HR methods: if the analysed signal does not fit with the model, the parameters estimated by the model may be really incorrect. Thus the HR method is not really reliable.

Though, on stationary parts of the control signal, the model fits quite well to the reality. Hence, as showed in Figure 4.42 on page 82, the estimation of the parameters performs better, giving smaller error signals.

Thus, even though the HR methods have a better resolution than the Fourier transform, they have several drawbacks that seems to make them unusable for our purpose. Hence, for the remainder of the document, we will use the preceding combination of the Fourier transform and least-square polynomial estimation.

4.2 Predicting partial tracks using the new model

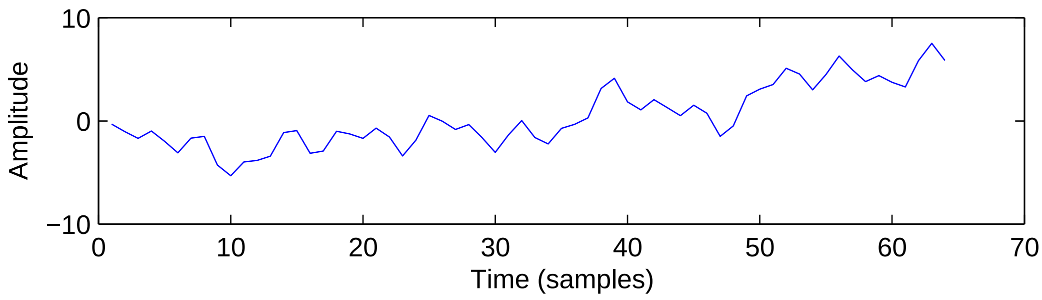
During the process of partial tracking with the McAulay-Quatieri algorithm [MQ86], the peak selection algorithm is very important to follow the right tracks. Indeed, choosing the wrong peak during partial tracking can be quite disastrous (as shown in [LMR05b]). To enhance the peak selection, various methods have been investigated.

Indeed, the algorithm we use is based on the prediction of the future of a partial from its past peaks. To choose the best peak candidate in the ones available in the next frame, the past peaks of the currently-considered partial are used to compute a virtual – predicted – peak. Then, we take the closest peak candidate among the measured peaks.

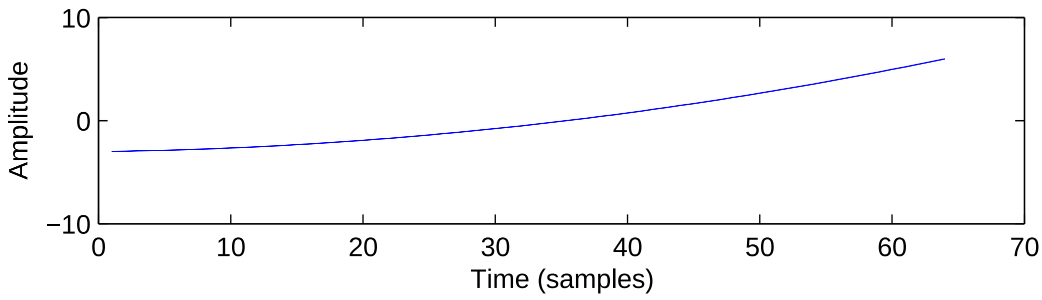
As we said earlier, the Burg method proved to work best for partial tracking. However, the Burg method tends to minimise prediction errors at the expense of spectra which are not well suited for sinusoidal analysis (see [KKZS03] for more details). One alternative would be to use a Fourier transform to compute the spectrum of the partial parameters, and extend the sinusoids of this spectrum to predict future peaks.

Here, we took this other approach: We take advantage of the spectral analysis done on the partials to propose a prediction method based on spectral extrapolation. Using the values for the sinusoids we found using the Fourier analysis, it is easy to extrapolate these sinusoids by direct sine function evaluation.

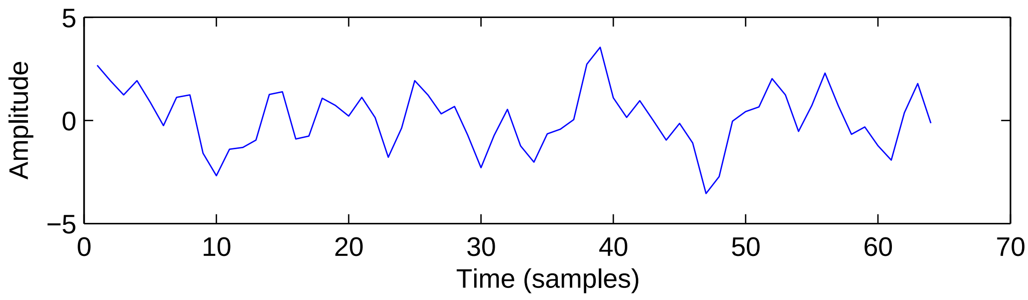
As for the spectral extrapolation, the polynomial part is in turn extrapolated in order to ensure the global envelope. This makes our extrapolation method well-suited for peak-selection in partial tracking, since no artifact is added to the result.



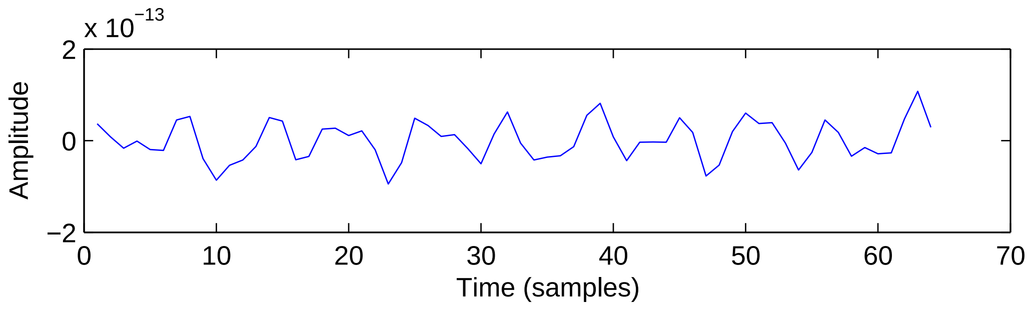
(a) The original synthetic signal



(b) The polynomial part of the synthetic signal

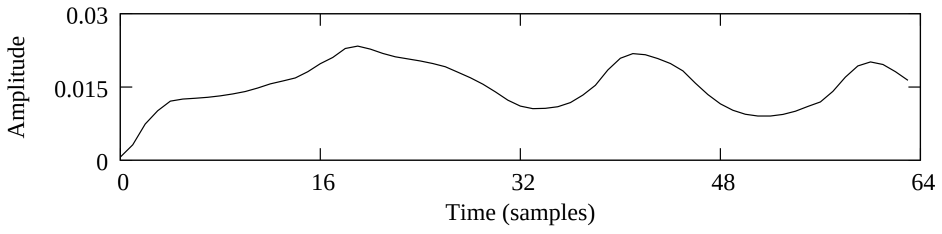


(c) The sinusoidal part of the synthetic signal

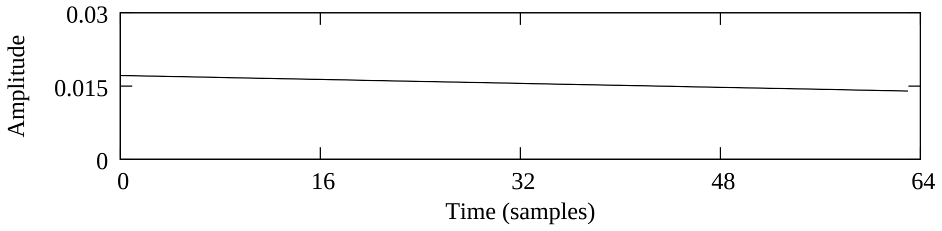


(d) The residual error signal

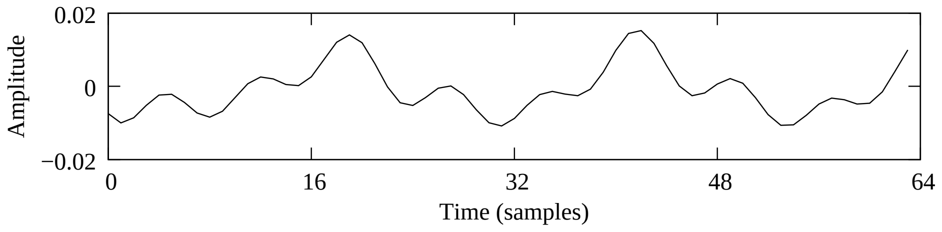
Figure 4.40: *Decomposing a synthetic signal using the esprit high-resolution method.*



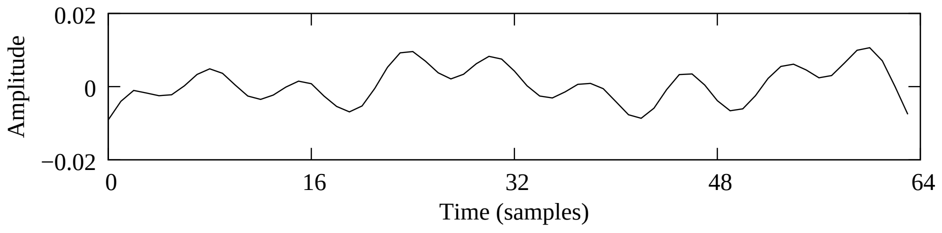
(a) The beginning of a real-amplitude signal



(b) The polynomial part of the beginning of the real-amplitude signal

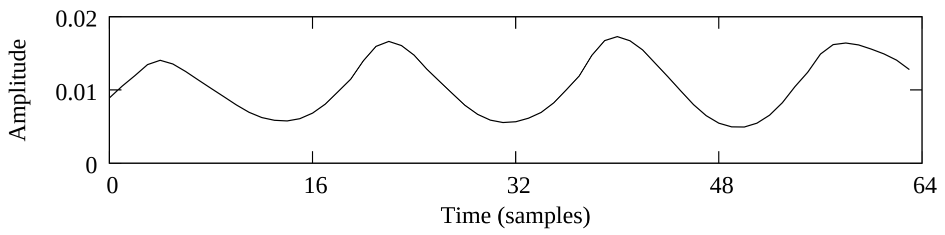


(c) The sinusoidal part of the beginning of the real-amplitude signal

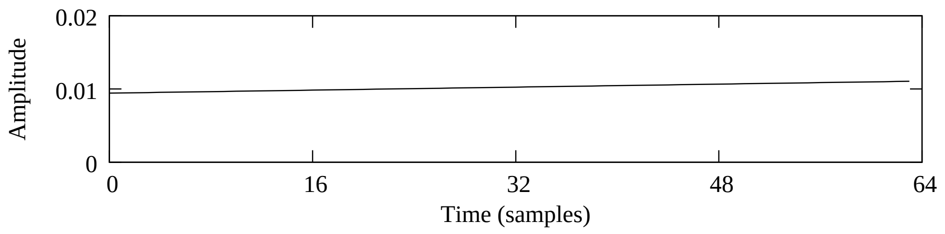


(d) The residual error signal

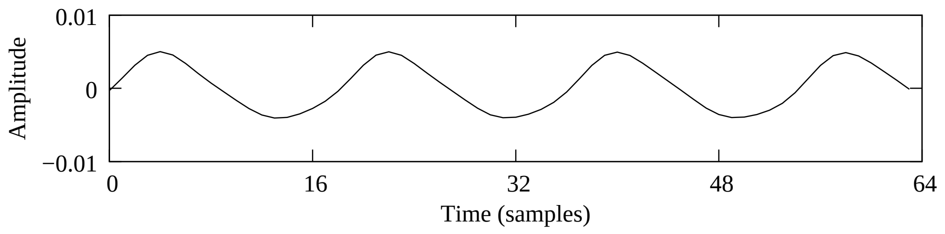
Figure 4.41: *Decomposing the beginning of a real-amplitude signal using the ESPRIT high-resolution method. The residual error signal has about the same amplitude as the original signal. This shows that the estimation did not work well on this part of the signal.*



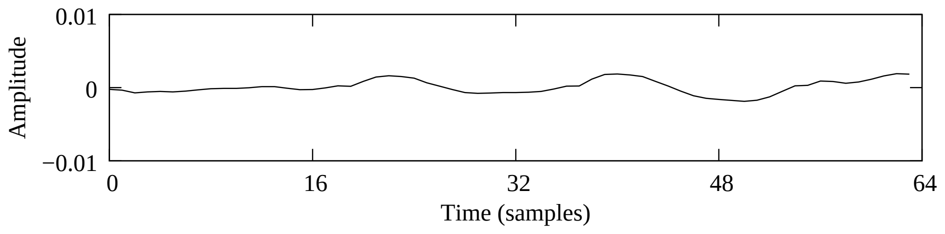
(a) The middle part of a real-amplitude signal



(b) The polynomial part of the middle part of the real-amplitude signal

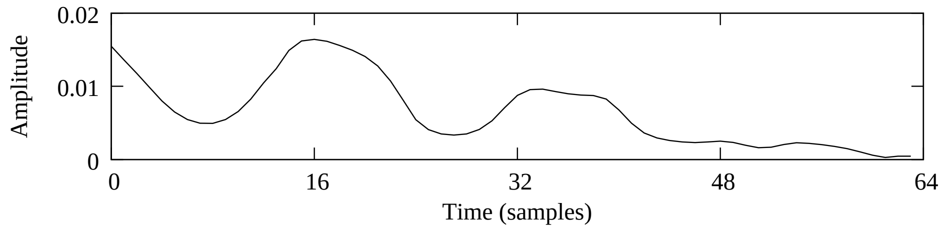


(c) The sinusoidal part of the middle part of the real-amplitude signal

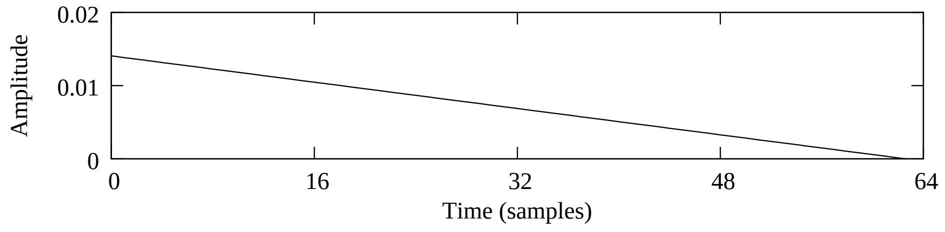


(d) The residual error signal

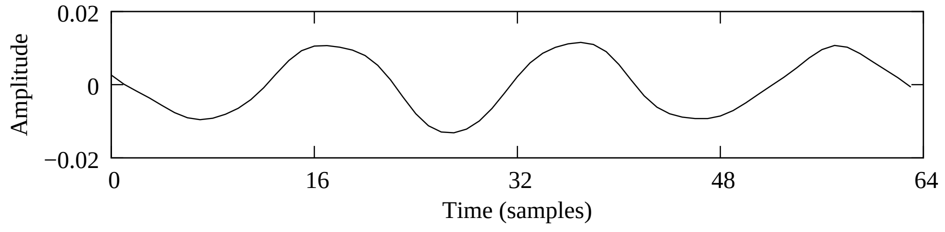
Figure 4.42: *Decomposing the middle part of a real-amplitude signal using the ESPRIT high-resolution method. Here the residual error signal is quite low. On this stationary part, this method works well.*



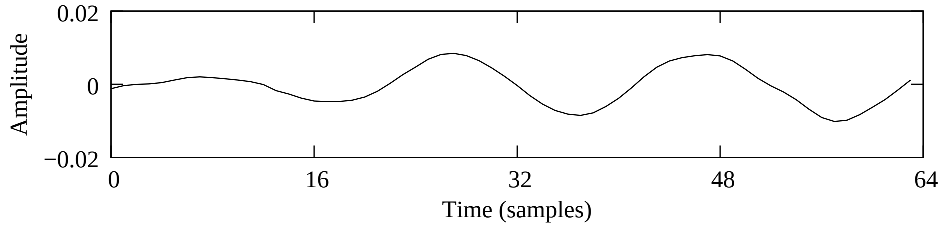
(a) The end of a real-amplitude signal



(b) The polynomial part of the end of the real-amplitude signal



(c) The sinusoidal part of the end of the real-amplitude signal



(d) The residual error signal

Figure 4.43: *Decomposing the end of a real-amplitude signal using the ESPRIT high-resolution method. The problem here is the same as for the beginning of the signal, that is that the residual error signal is strong. It is clear that this methods performs well only on the stationary parts of the control signals.*

The way we perform the prediction is quite simple. The first step is to find the parameters of our model on the past samples used for the prediction. Once those parameters are found, we just consider that the parameters of the sinusoids will be constant over the predicted samples, and we compute the next values of the polynomial using the coefficients we found. Adding the two parts of the computation gives us the predicted samples.

Figure 4.44 on the facing page shows some results of 1-sample predictions on a simple sinusoid. We call 1-sample predictions the prediction of a single sample at each iteration from the past samples. Figure 4.44(a) on the next page shows the predictions on a pure sinusoid. Burg performs better, but our method shows promising results. The small deviations are due to the polynomial estimation method which is not perfect. Figure 4.44(b) on the facing page shows the predictions on a pure sinusoid where we displaced a sample to simulate a local error. Our method performs better in that matter since it is not disturbed by the noise, whereas the Burg method deviated a bit from the sinusoidal trajectory.

On longer predictions however (extrapolating more than one sample), our method might not be as good. Indeed, the polynomial is not guaranteed not to diverge outside the analysis window. Thus diverging may occur under certain conditions, among which a very small number of periods of the sinusoidal part of the signal. A solution would be to lower the degree of the polynomial (the lower the degree, the more stable the polynomial), but that would mean possibly lowering the quality of our estimation. This is a trade-off between approximation quality and long-term prediction stability.

As for short signals, our method does not perform very well until more than two periods of the modulation are present in the analysis frame. This is illustrated by Figure 4.45, where we see that the predictions are diverging at the extrema of the signal. This can be explained by the fact that the third degree polynomial is still not “flattened” until the oscillation of the modulation is clearly established in the signal. In that case, the polynomial is trying to approximate the sinusoid, which it is obviously not fit for.

However, for very short signals, the Poly-Sin method is equivalent to polynomial extrapolation which works quite well on signals evolving very slowly. This is illustrated by the 20 first samples in Figure 4.45 on page 86.

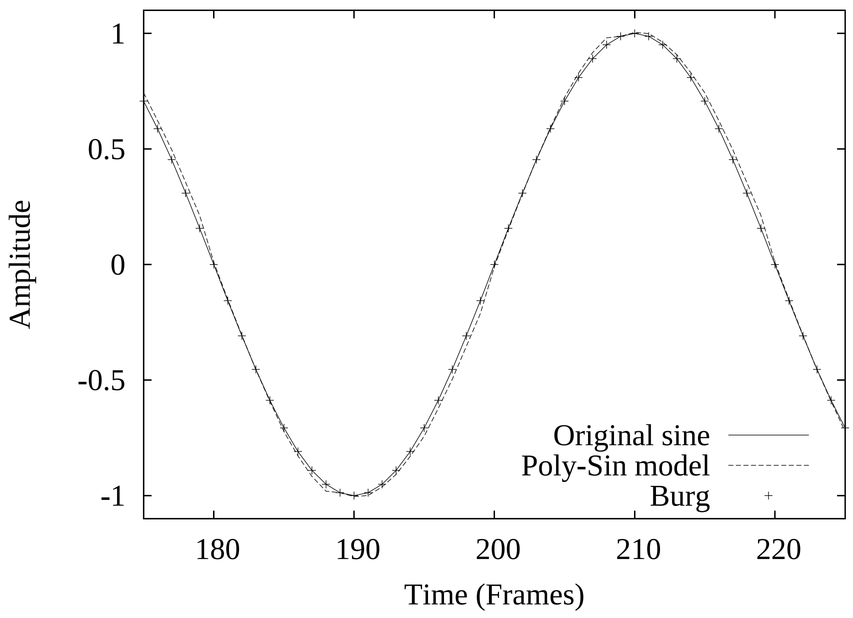
Another advantage of the method is that it is self-adapted to the number of samples available from the past. Indeed, the estimation over a small number of samples is possible an favours the use of polynomials, and even one sample is enough for (constant) extrapolation. The estimation over a large number of samples on the other hand is also well performed by favouring the Fourier transform.

4.3 Inaudible control signals

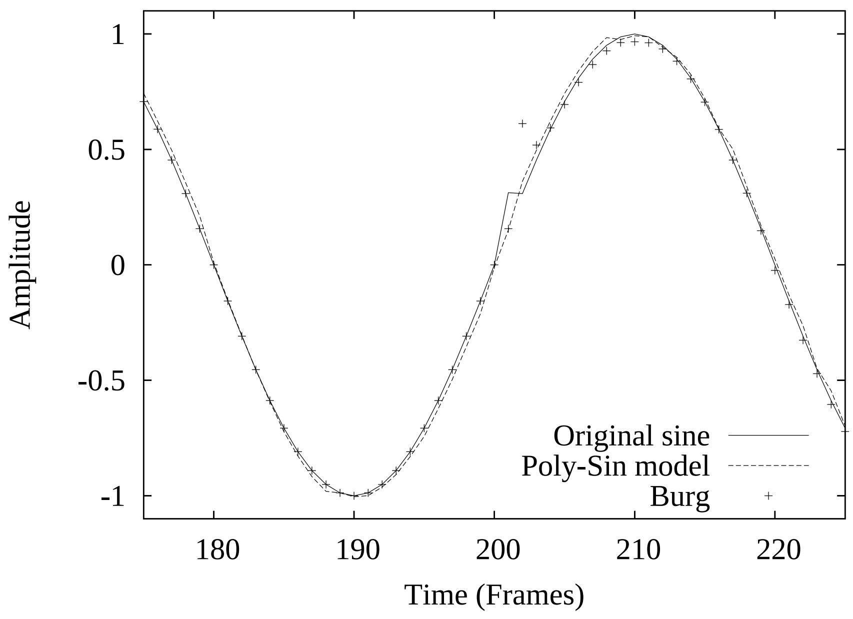
The order-2 spectral model as we have defined it can be used for the prediction step in partial tracking. The partial tracking can be enhanced further by the use of the Poly-Sin model.

Indeed, the Fourier transform of the predicted partial trajectories can show some valuable information. In the context of long-term peak prediction, we use the Burg method since the possible instability of the polynomial makes the Poly-Sin model not well-suited for long-term prediction.

This long term prediction is useful in order to find different possible tracks for a single



(a) Pure Sinusoid



(b) Altered Sinusoid

Figure 4.44: *Predicting one sample at the time on a pure sinusoid (a) and an altered sinusoid (b). The window-size for Poly-Sin prediction is 128 samples, the order for Burg prediction is 8.*

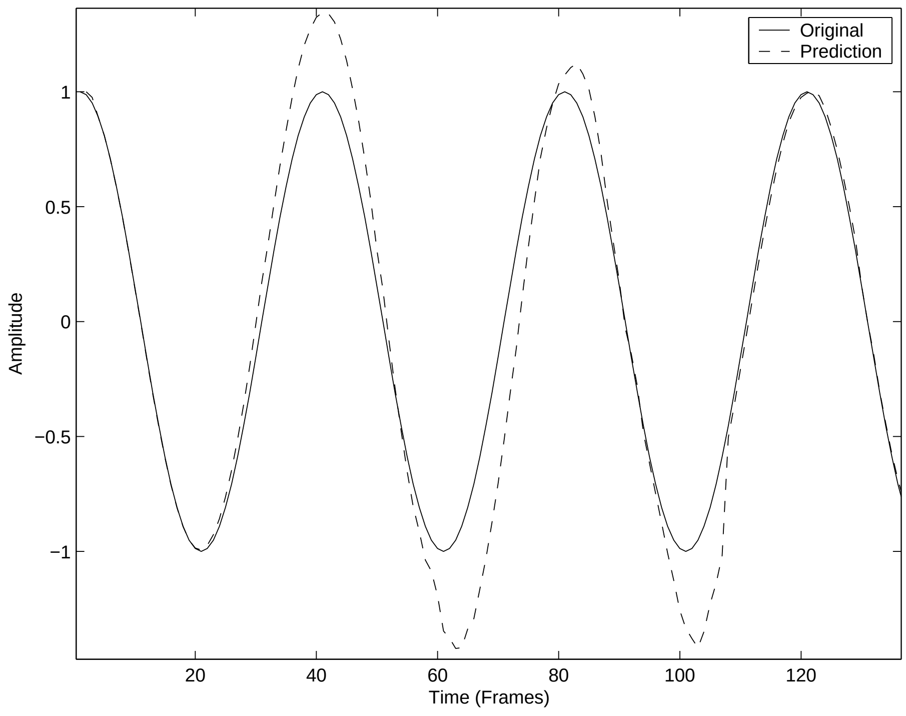


Figure 4.45: *Prediction using the Poly-Sin model, using a third degree polynomial. The solid line represents the original signal (that in fact contains no polynomial), the dashed line represents the prediction signal. The prediction signal is computed sample by sample using each time all the past original samples. For example, prediction of sample 40 is computed using original samples 0 through 39. We can see that the prediction works well in the beginning (when no modulation can yet be identified) and in the end (when the modulation is well established), but performs badly between a half period of signal and two and a half periods of the signal.*

partial and then keep the best fitted peaks. In order to determine the best fitting predicted peaks, Lagrange *et al.*try to minimise the high frequency content of a partial [LMR05c]. This high-frequency content can be revealed through the use of the Fourier transform of the trajectory as energy in the higher frequencies. In other words, they want to keep the peaks forming the smoothest partial-track.

Indeed, the sinusoidal model is based on the hypothesis that the control parameters of the sound (partial) are slow-varying. Playing the control parameters directly, as if they were sound signals, those signals would be inaudible, since they are not supposed to contain frequency component of more than 20 Hz. Hence, we say that slow-varying control parameters of the sound are inaudible. By adding high frequency content in these partials, one could add strong amplitude and frequency modulations to the sound, and thus question the perceptive consistency of the model (one frequency heard for one partial).

A good method to detect high frequency content in the partials is to apply a high-pass filter to the frequency trajectories of the partial and its different possible computed trajectories (made of measured and estimated peaks). The trajectory which would induce the least energy in the high frequencies of the control parameters would then be chosen to be the continuation of the partial.

By observing the “smoothness” of frequency trajectories, McAulay & Quatieri developed a way to unwrap the phase trajectories of the partials. On the other hand, in order to stay smooth, we can define a minimal level of energy in audible frequency bands (above 20 Hz). This could then be applied to frequency trajectories of partials, as shown by Lagrange *et al.*[LMR07].

Frequency and amplitude modulations of higher frequency can be considered as audible. It is then suitable to avoid such modulations in order to have a noiseless sinusoidal sound. Hence we can apply a high pass filter to detect higher frequency or amplitude modulation (those above 20 Hz in fact).

Knowing that we generally perform the sinusoidal analysis of a 44100 Hz sampled signal with 2048-sample windows and a 512-sample hop size, the sampling frequency of the resulting partial is about 86 Hz. The Nyquist frequency is then about 43 Hz and thus we need a high-pass filter that cuts half of the spectrum away (thus about 21 Hz). The filter used is a low-delay elliptic infinite response (IIR) high-pass filter. The coefficients of the filter are:

$$c = \begin{pmatrix} 1 \\ 0.7358 \\ 1.0762 \\ 0.5540 \\ 0.2346 \end{pmatrix}$$

$$d = \begin{pmatrix} 0.06 \\ -0.2274 \\ 0.335 \\ -0.2274 \\ 0.06 \end{pmatrix}$$

c and d corresponding respectively to the pole and zero parts of the IIR filter. The magnitude response of the filter is given in Figure 4.46 on the following page and some

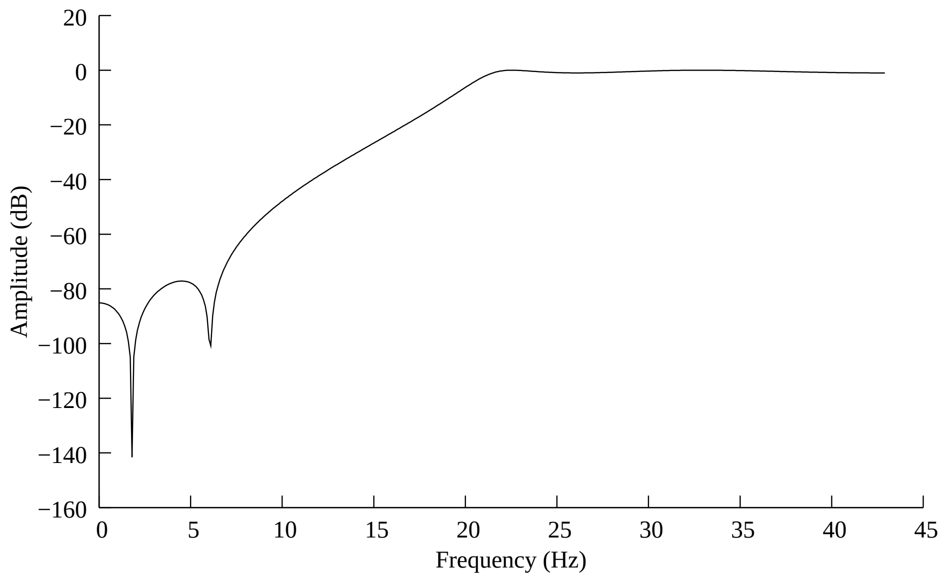


Figure 4.46: *Magnitude response of the filter used to detect high-frequency content in partial trajectories.*

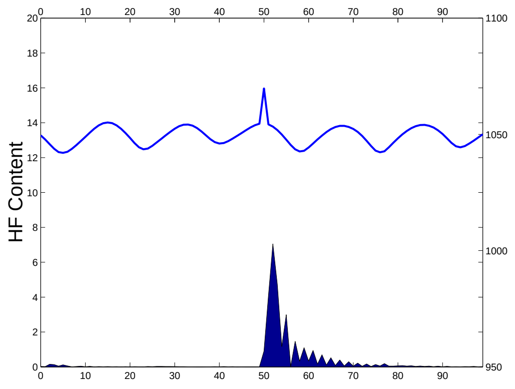
examples of its output is given on Figure 4.47 on the next page.

This high frequency (more than 20 Hz detection) can be used for partial tracking. Indeed, it is possible to apply the high content filtering to the fitness measurement of a peak to a trajectory. By considering the high frequency content introduced by a peak candidate that would be added to a partial, it is possible to decide whether a peak is suitable or not.

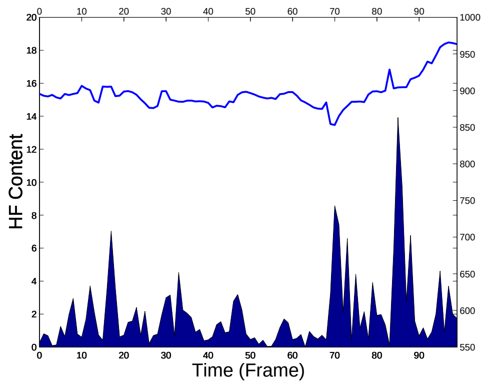
The chosen trajectory should then contain the highest possible number of extracted peaks while maintaining a small HFC in both frequency and amplitude. To each trajectory is associated a cost function that considers the HFC both in frequency and amplitude. Additionally, the cost function is divided by a factor $\Gamma \in]0, 1]$ each time an interpolated peak is used. If the Γ factor is different from 1, then adding a interpolated peak changes the cost of the trajectory, since Γ is elevated at the power of n_p, n_p being the number of interpolated peaks in the trajectory number p.

$$\Pi_p = \left(\frac{1}{\Gamma}\right)^{n_p} \cdot \frac{\sum_{i=0}^{N_f} |\tilde{a}_i^k|^2}{K_a} \cdot \frac{\sum_{i=0}^{N_f} |\tilde{f}_i^k|^2}{K_f} \tag{4.76}$$

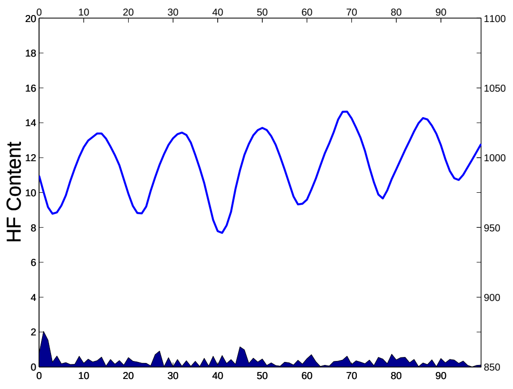
where $\tilde{a}_i^k$ and $\tilde{f}_i^k$ are, respectively, the high-frequency filtered amplitude and frequency of the k-th peak in the frame i. This filtering is done using memories of the filters associated to the current partial. K_a and K_f are normalising constants. The choice of the best trajectory leads to constraints on the relative order between costs and not on their absolute values, so that K_a and K_f can be safely set to 1. A good value for Γ has been found to be 0.9. The selected peak is then the first peak of the trajectory with the smallest Π_p.



(a) Example 1



(b) Example 2



(c) Example 3

Figure 4.47: *Output of the high pass filter over 3 partial trajectories. The first signal is sinusoid with a burst on one sample, the second signal is noise, and the third is a real world vibrato trajectory. The trajectories without noise have the lowest filter output.*

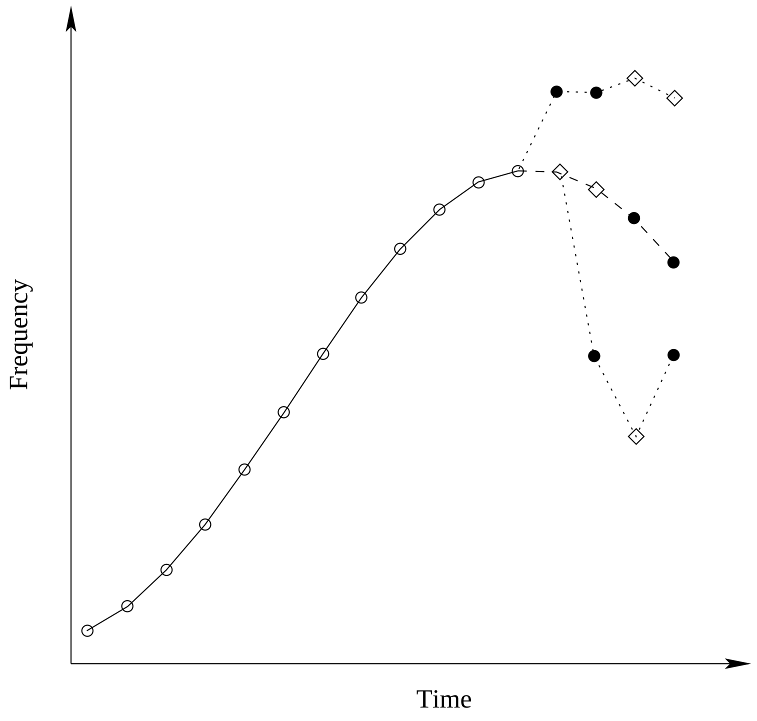


Figure 4.48: *The possible outcomes of partial tracking. Using high-frequency content filtering, it is possible to prefer a predicted peak (black dots) over an extracted peak (diamonds) in order to keep the trajectory smooth.*

4.4 Classification of partials

The order-2 spectral model can be bent to yet another application: the classification of the partials by sound. Indeed, it may be interesting to study the ability to automatically gather spectral components (partials) of the same sound by considering the modulations of their frequency and amplitude parameters over time.

4.4.1 Context

The extraction of high level information from monophonic recordings have been widely studied (see [Mar99, ALP03, KS04] for example) concerning the problem of instrument recognition. In the case of weak polyphonies, *i.e.* solo playing with low amplitude accompaniment, has been addressed (in [EB04]) by considering only the prominent peaks of the spectrum. The extraction of high level information from strong polyphony, *i.e.* identifying several instruments playing simultaneously at an equivalent loudness level, is still a challenging issue.

When dealing with complex sound mixtures, one would like a mid-level representation of the polyphony [ER95, BP05] that could ease further processing. Ideally, this representation should describe polyphonic sounds as a set of coherent spectral regions, where each set can be considered as monophonic.

We present here a new metric for spectral components gathering that can be used to build such a mid-level representation. We consider the case of several sources playing at the same time, and that the partials of each source have already been tracked. Thus, the challenge is to gather the already identified partials to obtain the mid-level representation mentioned above.

4.4.2 Mid-level representation of polyphonic sounds

For various applications, one needs a representation of polyphonic sounds where the partial information as well as their evolutions with respect to time of each sound source can easily be extracted. In this section, we discuss the fact that the well-known sinusoidal model can be a basis for such a representation.

The sinusoidal model represents pseudo-periodic sounds as sums of sinusoidal components – so-called partials – controlled by parameters that evolve slowly with time [MQ86, Ser97]:

$$P_k(m) = (F_k(m), A_k(m), \Phi_k(m)) \tag{4.77}$$

where $F_k(m)$, $A_k(m)$, and $\Phi_k(m)$ are respectively the frequency, amplitude, and phase of the partial P_k at time index m. These parameters are valid for all $m \in [b_k, \cdots, b_k + l_k - 1]$, where the b_k and l_k are respectively the starting index and the length of the partial.

These partials are part of a more perceptually coherent entity that will be called here an sound entity (after Lagrange [Lag04]).

Thus, this can be written as:

$$\mathcal{S} = \bigcup_{n=1}^{N} E_n \tag{4.78}$$

with $\mathcal{S}$ being the mid-level representation of the sound, E being an acoustical entity and N the total number of entities in the sound. Hence each entity is made of a group of partials:

$$E_n = \bigcup_{k=1}^{M_n} P_k^n \tag{4.79}$$

where M_n is the total number of partials P_k^n in the entity.

The partials can be extracted from polyphonic sounds with dedicated tracking algorithms (see section 3.1 on page 37 and [LMR04, LMR05a]). However, in strong polyphony, the identification of all the partials is made difficult due to a common problem in spectral audio processing called spectral bin contamination. As noted by Ellis and Rosenthal [ER95], some spectral components of different tones may be represented as a unique partial. This problem will not be addressed here, since we only consider mixtures of already tracked entities.

To extract these entities from a sinusoidal representation of a sound, similarities between partials should be considered in order to gather the ones belonging to the same acoustical entity. From the perceptual point of view, some partials belong to the same entity if they are perceived by the human auditory system as a unique sound. There are several cues that lead to this perceptual fusion: the common onset, the harmonic relation of the frequencies, the correlated evolutions of the parameters and the spatial location [Bre90].

The earliest attempts at acoustical entity identification and separation consider harmonicity as the sole cue for group formation. Some rely on a prior detection of the fundamental frequency [Gro96, FCQ98] and others consider only the harmonic relation of the frequencies of the partials [Kla02, VK00, RG04]. Yet, many musical instruments are not perfectly harmonic.

According to the work of McAdams [McA89], a group of partials is perceived as a unique acoustical entity only if the variations of these partials are correlated. Therefore, the correlated evolutions of the parameters of the partials is a generic cue since it can be observed with any vibrating instruments.

In order to define a dissimilarity metric that considers the common variation cue, we will study in the next section the physical properties of the evolutions of the frequency and amplitude parameters of the partials.

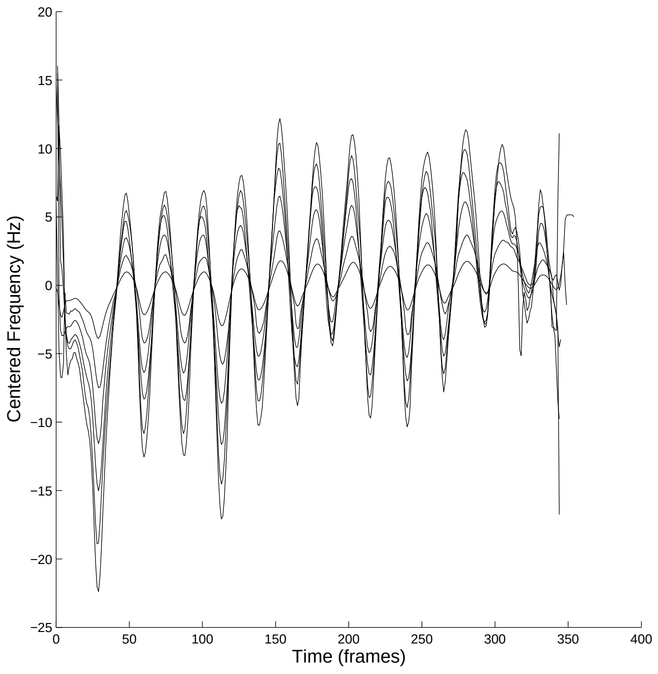
4.4.3 The common variation cue

Let us consider an harmonic tone modulated by a vibrato of given depth and rate. All the harmonics are modulated at the same rate and phase but their respective depth is scaled by a factor equal to their harmonic rank (see Figure 4.49(a) on the next page). It is then important to consider a metric which is scale-invariant.

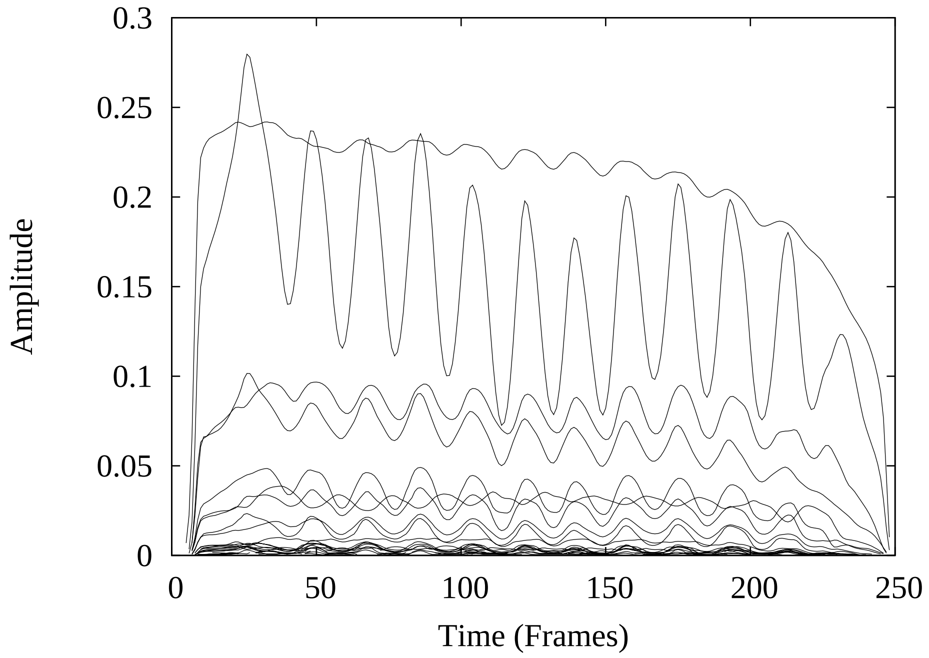
Cooke uses a distance [Coo93] equivalent to the cosine dissimilarity d_c, also known as *intercorrelation*:

$$d_c(X_1, X_2) = 1 - \frac{c(X_1, X_2)}{\sqrt{c(X_1, X_1)}\sqrt{c(X_2, X_2)}} \tag{4.80}$$

$$c(X_1, X_2) = \sum_{i=1}^{N} X_1(i)\, X_2(i) \tag{4.81}$$



(a) Frequencies



(b) Amplitudes

Figure 4.49: *Mean-centred frequencies and amplitudes of some partials of a saxophone tone with vibrato. The frequencies seem to be identical if it was not for the scale.*

where X_1 and X_2 are real vectors of size N. Here, X_n will be the in turn frequency and amplitude of a partial over time. This dissimilarity is scale-invariant.

Virtanen *et al.* proposed (in [VK00]) to use the mean-square error between the vectors first normalised by their average values:

$$d_v(X_1, X_2) = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{X_1(i)}{\bar{X}_1} - \frac{X_2(i)}{\bar{X}_2}\right)^2 \tag{4.82}$$

where X_1 and X_2 are vectors of size N and $\bar{X}$ denotes the mean of X. This normalisation is particularly relevant while considering the frequencies since the ratio between the mean frequency of a given harmonic and the one of the fundamental is equal to its harmonic rank.

Lagrange proposed (in [Lag05]) to consider the Auto-Regressive (AR) model as a scale-invariant metric that considers only the predictable part of the evolutions of the parameters:

$$X_l(n) = \sum_{i=1}^{k} k_l(i) X_l(n-i) \tag{4.83}$$

where the $k_l(i)$ are the AR coefficients. Since the direct comparison of the AR coefficients computed from the two vectors X_1 and X_2 is not relevant, the spectrum of these coefficients is compared as proposed by Itakura [Ita75]:

$$d_{\mathrm{AR}}(X_1, X_2) = \log \int_{-\pi}^{\pi} \frac{|K_1(\omega)|}{|K_2(\omega)|}\frac{d\omega}{2\pi} \tag{4.84}$$

where

$$K_l(\omega) = 1 + \sum_{i=1}^{k} k_l(i) e^{-ji\omega} \tag{4.85}$$

When considering the amplitudes of the partials, a scale-invariant metric is also important. In this context, the normalisation proposed by Virtanen is no longer motivated since the relative amplitudes of the harmonics depend on the envelope of the sound. For example, on Figure 4.49(b), the topmost curve (with small modulations) represents the amplitudes of the fundamental partial, while the second to the top curve with broad oscillation represents the first harmonic.

Moreover the envelope is globally decreasing as the frequency grows, but it can appear that the amplitude of the envelope is also ascending due to the specific shape of the envelope around formants. Therefore, when the frequency of a partial is modulated, the amplitude may be modulated with a phase shift, see the bottom curve of Figure 4.49(b) on the preceding page. Therefore, a metric that is phase-invariant should be considered.

The amplitude evolution of a partial is composed of a temporal envelope and some periodic modulations. Since the envelope of the amplitude of the partials can be very different from partials to partials of the same entity it may be useful to consider only the periodic modulations while computing their similarities.

The metric introduced in the next section will cope with these issues.

4.4.4 Proposed metric

The aim of proposing a new metric is to go beyond temporal domain by taking the parameters to the spectral domain. There was already an attempt at this, using AR models (see equation 4.84 on the preceding page).

Since the Fourier transform is based on the fact that the input signal is periodic, using a spectrum of the evolutions of the partials might show common periodicities of the partials. This will be handy for the modulations of the partials created by vibrato and tremolo, since we can assimilate these modulations to sinusoidal ones over a short period of time (see [MW00, MR04] and Figure 4.51). It can be also interesting for micro-modulations such as the ones produced by vibrating strings such as the strings of a piano (see Figure 4.50). Hence, the spectrum of the evolutions in frequency and amplitude of the sound are relevant from the point of view of the correlation of evolutions.

Let us see then how we compute the correlation of evolutions in order to obtain our new metric, first for the frequency parameters of the sound, second for the amplitude parameters of the sound (since two slightly different methods are used).

Using the frequencies of the partials

The first step in the calculation of our new metric is to correlate the evolutions of the frequencies of the partials. As we said before, a good description of these evolutions is given by the spectra of these evolutions.

The way to compute the spectra of the frequency evolutions of the signal from a partial is to take off the mean value of this frequency and then compute the Fourier transform of the resulting signal. This way, the first bin of the Fourier spectra of the frequency evolutions will be null, and thus only the modulations of the control signal will be captured. Indeed, in order to have a clean spectrum relevant to the evolutions, it is necessary to have these evolutions centred around zero.

Then, we apply the previously exposed process to the frequencies of all the partials from which we want to measure evolution correlation. Once we have these frequencies expressed in terms of spectra, the way to compute the distance between two partial signals is to intercorrelate their spectra (see equation 4.80 on page 92). This gives

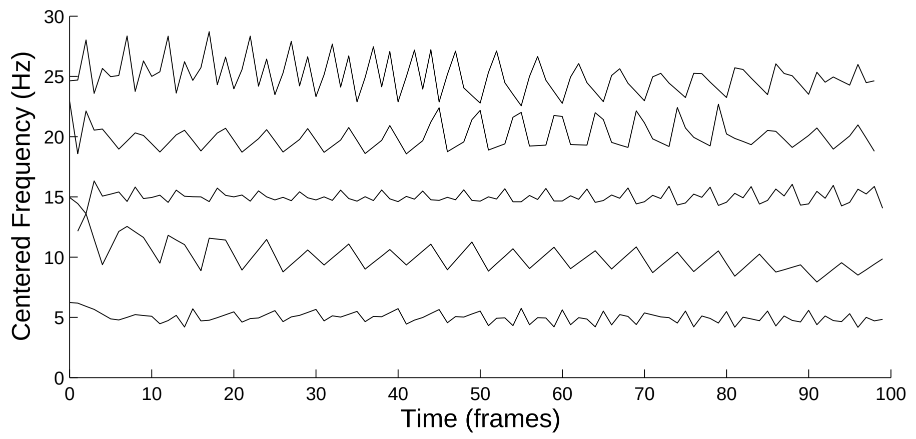
$$d_s(f_1, f_2) = d_c(|F_1|, |F_2|) \tag{4.86}$$

where f_1 and f_2 are the zero-centred (we subtracted an constant polynomial) frequency vectors of two partials P_1 and P_2 and F_1 and F_2 are the Fourier spectra of f_1 et f_2.

This distance is phase-invariant since we use the amplitude spectra.

Using the amplitudes of the partials

In the case of the amplitudes of the partials, the problem is slightly more complicated. Indeed, in order to centre the oscillating part of the signal around zero, subtracting the mean will not be sufficient. As presented in other works [RMG05], subtracting a polynomial is sufficient to centre the oscillations around zero. The idea behind this polynomial subtraction is that the envelope of a sound (seen as attack, decay, sustain, and release) can be roughly locally approximated by a polynomial.



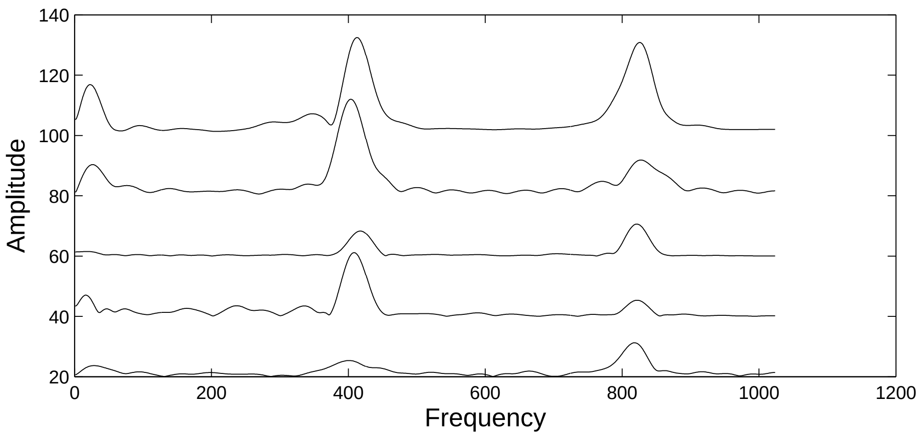


Figure 4.50: *Centred frequencies (top) of a piano note and their corresponding spectra (bottom). Each curve is shifted and the spectra are smoothed using zero-padding for clarity sake.*

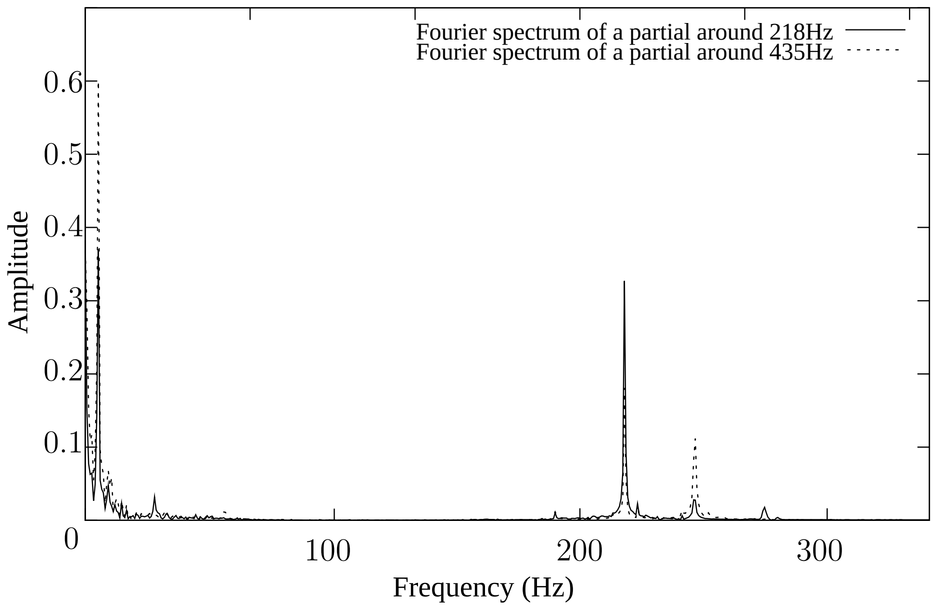


Figure 4.51: *Fourier spectra of the frequency signals of two partials of a same sound (with vibrato). The spectra are clearly correlated, showing similar peaks at the same frequencies.*

This gives us the distance d_{sp}:

$$d_{sp}(a_1, a_2) = d_c(|\widetilde{A_1}|, |\widetilde{A_2}|) \tag{4.87}$$

where $\widetilde{A_k}$ is the Fourier spectrum of $\widetilde{a_k}$ with

$$\widetilde{a_k} = a_k - \Pi(a_k)$$

where a_1 and a_2 are the amplitudes of two partials, $\Pi(x)$ is the envelope polynomial computed from signal x using a simple least-square method ([Nea04]).

Metric combination

In order to exploit both the frequency and amplitude parameters, we need a way to combine the measures of amplitude and frequency distances.

Virtanen *et al.* proposed to combine frequency and amplitude parameters distances by means of adding the two distance measures while considering an harmonicity factor. In this work [VK00], they weighted each distances before performing the addition. Based on summing also, we consider the following distance, even though since the weights are not supplied and no harmonicity information is available it does not correspond to the original distance.

$$d_{v+v}(P_1, P_2) = \frac{d_v(f_1, f_2) + d_v(a_1, a_2)}{2} \tag{4.88}$$

where f_k and a_k are respectively the frequency and amplitude of partial P_k.

Since our proposed distances d_s and d_{sp} are normalised, if we want to give the same weight to the two distances, we can combine the frequency and amplitude distances by performing a simple mean. This would then yield :

$$d_+(P_1, P_2) = \frac{d_s(f_1, f_2) + d_{sp}(a_1, a_2)}{2} \tag{4.89}$$

In order to take into account the best result on part of one of the measures, a method would be to take the minimum of the two distances:

$$d_m(P_1, P_2) = \min(d_s(f_1, f_2), d_{sp}(a_1, a_2)) \tag{4.90}$$

However, better results are achieved when we multiply amplitude and frequency parameter distances, as it will be presented in section 4.4.6 on page 101. This combination, however less robust to errors, seems to take better account of the performance of each distance measure independently. In order to keep the results in linear scale, a square root is applied to the combination:

$$d_\times(P_1, P_2) = \sqrt{d_s(f_1, f_2) d_{sp}(a_1, a_2)} \tag{4.91}$$

4.4.5 Evaluation

We will present here the methodology used for evaluating the performance of the different metrics reviewed in section 4.4.3 on page 92 and proposed in section 4.4.4 on page 95. The evaluation database is first described. Next, several criteria are presented, each one evaluating a specific property of the evaluated metric.

Database

In this study, we focus on a subset of musical instruments that produce pseudo-periodic sounds and model them as a sum of partials. The instruments of the Iowa database [iow] globally fit to this condition even though some samples have to be removed.

The evaluation database is created as follows. Each file of the Iowa database is split into a series of audio files, each containing only one tone. The partials are then extracted for each tone using common partials tracking algorithms [MQ86, Ser97, LMR04]. Since we consider only the prominent partials of a given tone, only the extracted partials lasting for at least 1 second are retained.

At last, 2800 acoustical entities, *i.e.* group of partials extracted from the same musical tone are available. The *pizzicato* tones, *i.e.* plucked-string tones with strong attack and weak resonating phase as well as the *pianissimo* tones, *i.e.* tones with very low amplitude, are discarded.

Criteria

Once the evaluation database is defined, one need some criteria to evaluate the capability of a given metric to determine that two partials are “close” if they actually belong to the same acoustical entity and “far” otherwise.

Fisher criterion A relevant dissimilarity metric between two partials is a metric which is low for partials of the same entity – the class from the statistical point of view – and high for partials that do not belong to the same entity. The intra-class dissimilarity should then be minimal and the inter-class dissimilarity as high as possible. Let U be the set of elements of cardinal $\# U$ and E_n the entity of index i among a total of N different entities. An estimation of the relevance of a given dissimilarity $d(x,y)$ for a given acoustical entity is:

$$\text{intra}(E_n) = \sum_{P_i^n \in E_n} \sum_{P_j^n} d(P_i^n, P_j^n) \quad (4.92)$$

$$\text{inter}(E_n) = \sum_{P_i^n \in E_n} \sum_{\overline{P}_j^n \in \overline{E}_n} d(P_i^n, \overline{P}_j^n) \quad (4.93)$$

$$\mathcal{F}(E_n) = \frac{\text{inter}(E_n)}{\text{intra}(E_n)} \quad (4.94)$$

where $\overline{E}_n = U \backslash E_n$. The overall quality $\mathcal{F}(U)$ is then defined as:

$$\mathcal{F}(U) = \frac{\sum_{n=1}^{N} \text{inter}(E_n)}{\sum_{n=1}^{N} \text{intra}(E_n)} \quad (4.95)$$

This last criterion $\mathcal{F}(U)$ is based on the Fisher discriminant commonly used in statistical analysis. It provides a first evaluation of the discrimination quality of a given metric. It can however be noticed that this criterion is dependent of the scale of the studied dissimilarity metric.

Density criterion Dissimilarity-vector based classification involves calculating a dissimilarity metric between pair-wise combinations of elements and grouping together those for which the dissimilarity metric is small according to a given classification algorithm.

The density criterion $\mathcal{D}$ intends to evaluate a property of the tested metric that should be fulfilled in order to be relevantly used in combination with common classification algorithms such as hierarchical clustering or K-means [JMJ99, Mac67]. Indeed, many classification algorithms iteratively cluster partials which relative distance is the smallest one. The density criterion verifies that these two partials actually belong to the same acoustical entity.

More formally, given a set of elements X, $\zeta(X)$ is defined as the ratio of couples (a,b) so that b is closest to a and a and b belong to the same acoustical entity.

Given a function named cl defined as:

$$cl: \begin{array}{ccc} X & \rightarrow & \mathbb{N} \\ a & \mapsto & i \end{array}$$

where i is the index of the class of a. We get:

$$\mathcal{D}(X) = \frac{1}{\# X} \# \left\{(a,b) \,|\, d(a,b) = \min_{c \in X} d(a,c) \wedge E(a) = E(b)\right\} \quad (4.96)$$

where X can be either an acoustical entity E_n or the universe U and $\# x$ denotes the cardinal of x.

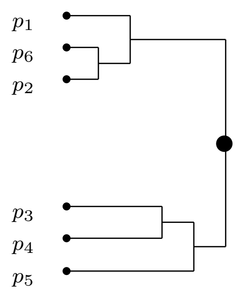


Figure 4.52: *Dendrogram representing the hierarchy obtained using the Agglomerative Hierarchical Clustering (AHC) algorithm with 6 partials. The cut at the highest level of the hierarchy represented by a dot identify two acoustical entities $C_1 = \{p_1, p_6, p_2\}$ and $C_2 = \{p_3, p_4, p_5\}$.*

Classification criterion For this criterion, the quality of the tested metric is evaluated by considering the quality of a classification done using the tested metric and a classification algorithm.

We consider an agglomerative hierarchical clustering procedure [Joh67]. This algorithm produces a series of partitions of the partials: $(G_n, G_{n-1}, \ldots, G_1)$.

The first partition G_n consists of n singletons and the last partition G_1 consists of a single class containing all the partials. At each stage, the method joins together the two clusters of partials which are most similar according to the chosen dissimilarity metric. At the first stage, of course, this ends in joining together the two partials that are closest together, since at the initial stage each cluster has only one partial. At each stage, the dissimilarity between the new cluster and the other ones is computed using the method proposed by Ward [War63].

Hierarchical clustering may be represented by a two-dimensional diagram known as *dendrogram* which illustrates the fusions made at each successive stage of clustering, see Figure 4.52 where the length of the vertical bar that links two classes is calculated according to the distance between the two joined clusters.

The acoustical entities can then be found by “cutting” the dendrogram at relevant levels. Here, for the classification criterion, the acoustical entities are identified by simply cutting the dendrogram at the highest levels to achieve the desired number of entities. If the desired number of entities is 2, only the highest level is cut (see Figure 4.52).

The classification criterion $\mathcal{H}$ is then defined as the number of partials correctly classified versus the number of partials classified:

$$\mathcal{H}(X) = \frac{1}{\#\ X} \#\ \{a | a \in \hat{E}_n \wedge E(a) = i\} \tag{4.97}$$

where $\hat{E}_n$ is an acoustical entity extracted from the hierarchy.

Methodology

To compare the metrics proposed in section 4.4.4 on page 95 and those reviewed in section 4.4.3 on page 92, we use the following methodology to compute the three evaluation

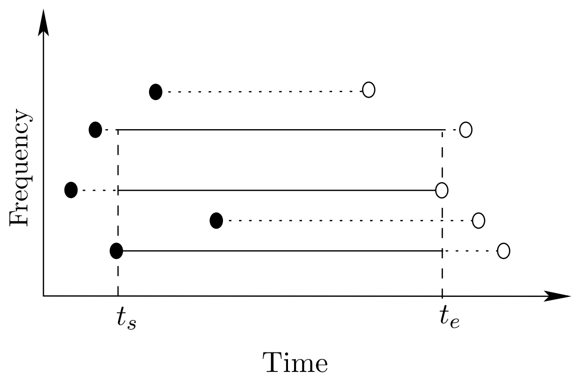


Figure 4.53: *Selection of the common parts of the partials of the two acoustical entity. A partial start is represented with a black filled dot and its end with a white filled dot. Only the partials existing before and after t_s and t_e are kept, represented with solid lines. These indices delimit the common part of all the partials.*

criteria. First, a number of acoustical entities is randomly selected in the database. Then, for each couple of entities between this selection, the following procedure is operated.

For the two entities of the considered couple (E_i, E_j), we compute t_s and t_e, the median values of the starting/ending time index of the partials. Only the partials existing before $t_s + \epsilon_s$ and after $t_e - \epsilon_e$ are kept. The values ϵ_s and ϵ_e are arbitrarily small constants (see Figure 4.53).

Then, the partials of the two entities are gathered to obtain the tested sinusoidal representation of the mixture $S = E_i + E_j$. Only the common part defined as the time interval where all the partials are active is considered to evaluate the tested metric.

For example, the common part of the partials represented in Figure 4.53 is between t_s and t_e.

4.4.6 Results

All the distance reviewed in section 4.4.3 on page 92 and proposed in section 4.4.4 on page 95 are now compared using the evaluation methodology described in the last section. The correlation distance d_c of equation 4.80 on page 92 and the distance d_v proposed by Virtanen (see equation 4.82 on page 94) requires no parametrisation.

The distance based on AR modelling d_{ar} considers AR vectors of 4 coefficients computed with the Burg method [Bur67]. The distance d_s of equation 4.86 on page 95 considers spectra computed with the Fast Fourier Transform (FFT) using vectors windowed by the periodic Hann window. The computation of the distance d_{sp} (see equation 4.87 on page 97) is similar except that the envelope (approximated with polynomials) is removed before the FFT computation.

In order to achieve reasonable computation time, a subset of 300 acoustical entities were randomly extracted from the database first and used for all the experiments detailed in the remainder of this section.

The results are presented as mean values for criterion, and the values in parentheses are the standard deviations (not shown for $\mathcal{F}$ since the value is already normalised).

Frequency parameter

	$\mathcal{F}$	$\mathcal{D}$	$\mathcal{H}$
d_c	2.909	0.938 (0.216)	0.929 (0.137)
d_v	1.763	0.929 (0.230)	0.881 (0.172)
d_{ar}	1.863	0.712 (0.326)	0.757 (0.166)
d_s	**3.488**	**0.944** (0.210)	**0.940** (0.130)
d_{sp}	2.909	0.936 (0.219)	0.931 (0.133)

Table 4.3: *Three criteria (Fisher $\mathcal{F}$, density $\mathcal{D}$, hierarchical classification $\mathcal{H}$) results for the five distances considered, applied on the frequencies of the partials. The values in parentheses are the standard deviations. The density and hierarchical criteria (the two last columns) are presented as scores between 0 and 1, 1 being a perfect result.*

The distances between partials based on the frequency parameter is shown on Table 4.3. The d_s distance we proposed gives the best results for the three criteria. It should be noted that the correlation distance (d_c) gives also good results for the two last criteria. We can also see that removing the polynomial from the frequencies of the partials does not contribute to the quality of the metric since frequencies of the partials of the sounds in the Iowa database are quasi-stationary. The performance is even worse because of the modulations that the polynomial might take away from the frequency evolutions.

Amplitude parameter

	$\mathcal{F}$	$\mathcal{D}$	$\mathcal{H}$
d_c	1.304	**0.818** (0.300)	0.786 (0.162)
d_v	1.298	0.784 (0.316)	0.773 (0.159)
d_{ar}	**1.938**	0.664 (0.331)	0.733 (0.156)
d_s	1.452	0.778 (0.301)	0.781 (0.163)
d_{sp}	1.366	0.796 (0.297)	**0.803** (0.171)

Table 4.4: *Three criteria (Fisher $\mathcal{F}$, density $\mathcal{D}$, hierarchical classification $\mathcal{H}$) results for the five distances considered, applied on the amplitudes of the partials. The values in parentheses are the standard deviations. The density and hierarchical criteria (the two last columns) are presented as scores between 0 and 1, 1 being a perfect result.*

As presented on Table 4.4, the performance of the distance measures for the amplitude parameter are globally worse than those obtained for the frequency parameter, lowering from 94% to 80% correct classifications at best. However, the polynomial removal slightly enhances the results.

The metric d_c performs best for the density criterion since it is generally very low for very similar partials. The metric d_{ar} gives a good result for the Fisher criterion, while it performs badly for the two other criteria. This metric was tested by Lagrange [Lag05], but only on a very limited database. On a larger database such as the one of the Iowa, we can see that this metric does not seem very stable on the three criteria. In this matter, the spectral metrics d_s and d_{sp} perform best.

Combination of amplitude and frequency parameters

	$\mathcal{F}$	$\mathcal{D}$	$\mathcal{H}$
d_{v+v}	1.298	0.784 (0.316)	0.773 (0.159)
d_+	2.040	0.923 (0.230)	0.928 (0.137)
d_m	**3.303**	0.934 (0.216)	0.943 (0.122)
$d_\times$	2.702	**0.937** (0.217)	**0.951** (0.116)

Table 4.5: *Three criteria (Fisher $\mathcal{F}$, density $\mathcal{D}$, hierarchical classification $\mathcal{H}$) results for the four combined distances we defined. The values in parentheses are the standard deviations. The density and hierarchical criteria (the two last columns) are presented as scores between 0 and 1, 1 being a perfect result.*

For combinations of dissimilarity metrics we computed all possible combinations of preceding metrics (d_c, d_v, d_{ar}, d_s, d_{sp}) with the three operators we proposed (+, ×, min), but we kept only the most relevant ones for clarity sake. Table 4.5 presents the results for the selected combinations.

Perhaps surprisingly, the metric d_m is given best for the Fisher criterion, while the metric $d_\times$ shows best results for both density and hierarchical classification criteria (the classification performance is enhanced by 1% over the obtained results with the frequency cue only). This tends to show that the combination of the frequency and amplitude parameters are correlated, and that while working with the micro-modulations of the sound, the fact that the frequency and amplitude parameters of the sound are linked (see section 4.5 on the next page) enhances the results.

4.4.7 A new tool to gather partials?

Considering the correlation of the spectrum of these evolutions leads to more reliable results than the ones obtained with the AR modelling approach proposed in previous works [Lag05]. According to the experiments, the modulations of the frequency appear to be the most relevant cue. However, the modulations of the amplitude can also be considered as relevant especially when the amplitude envelope of the partial is removed. We also demonstrated that considering the combination of metrics of frequencies and the amplitudes enhanced the classification results by about 1%.

This new metric may be used for the classification of partials into sound entities. It has to be noted that the hierarchical classification used as a quality criterion in our study, even though very naive, yields to very good results, about 95% of correct classifications. The use of more sophisticated classification methods will certainly lead to better performance. It would also be of interest to cope with the problem of contaminated partials when dealing with the more realistic case of sound entities mixed in the time domain.

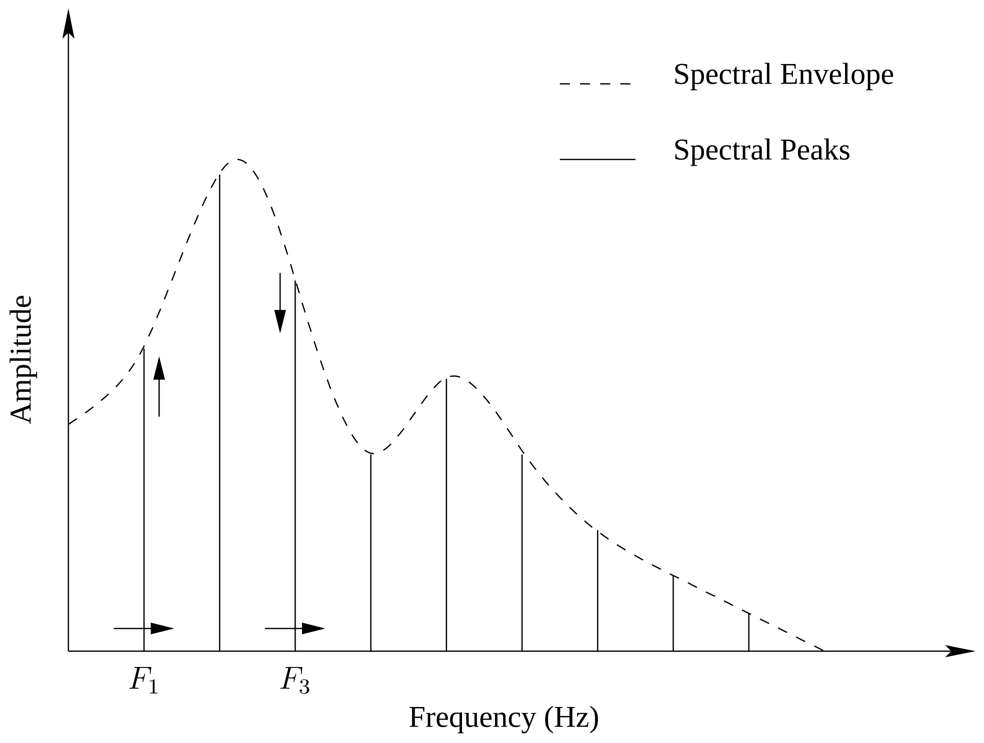


Figure 4.54: *Amplitude function of the frequency. Application of a harmonic frequency comb to a spectral envelope. During vibratos, the peaks at F_1 and F_3 have opposed phase tremolos.*

4.5 Considerations on the link of tremolo and vibrato

Arfib & Delprat (in [AD98]) consider that tremolo modulations are linked to vibrato modulations. Hence, when modifying the vibrato of a sound, they also alter the tremolo of the sound accordingly.

This is made possible by the way they decompose the signal. Indeed, each analysed piece of the sound signal is first separated into a source and a resonator. This source-resonance model [Opp69] is mainly used for sounds which are the result of a source filtered by a set of resonators.

The result of this operation gives the source spectrum and the resonance. The resonance is actually the smoothed spectral envelope of the Fourier spectrum of the sound signal, and the source can be seen as a harmonic comb (depending on the instrument). Hence, the modification of the vibrato is in fact a modification of the evolution of the harmonic comb.

As seen on Figure 4.54, the application of a vibrato (modulation of the frequency) ends up in modulation of the amplitude, the tremolo. The peaks placed on F_1 and F_3 have then opposed phase tremolos, since the spectral envelope have opposed gradients. This is visible on real examples (see Figure 4.55 on the next page for example). This implies that there is a strong link between the vibrato, the tremolo and the spectral envelope.

This vibrato and tremolo link could be used in our multi-level model. Indeed, the

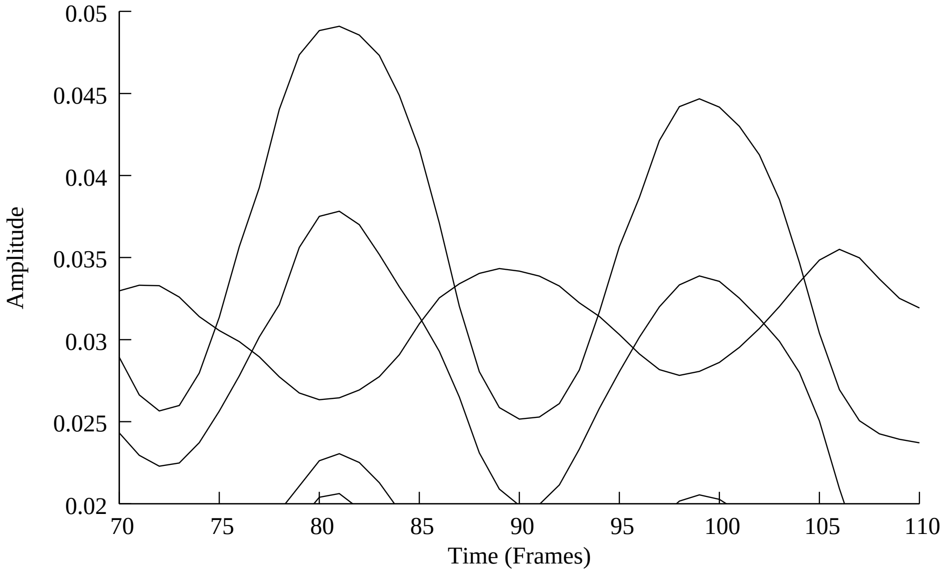


Figure 4.55: *Zoom on the amplitude tracks of the alto saxophone from Figure 3.25(b) on page 55. The tracks represented here are clearly either in phase, or opposed phase. This confirms the spectral envelope hypothesis.*

vibrato and the tremolo evolution could be detected jointly, once the spectral envelope have been estimated (some methods are presented in [RR05]). This would then lead to more precise estimations of the modulations of the sound.

Of course, it might happen that the spectral envelope changes with the change in frequencies, but as for the vibrato, we can consider that the frequency modulation has a relatively small amplitude, and thus that the spectral envelope does not change significantly.

However, Verfaille *et al.*[VGD05] have suggested that the vibrato is indeed linked to the tremolo but that is also modifies the brightness of the sound. Hence, the spectral envelope might not be neglectable after all.

4.6 Conclusion

In this chapter, we have dealt with the analysis of the partials of the sound themselves in the time-frequency domain, and one of its application. We have designed a new model for the frequency, amplitude and phase tracks of the partials consisting of a sum of sinusoids and polynomials. The methods we present to find the parameters of our model seem promising, eventhough some fine tuning might still be necessary.

Also, we present another enhancement to the partial tracking algorithm based on high-frequency filtering of the peak estimates in order to validate the inclusion of a peak in a partial.

We also have experimented the use of this model for the gathering of partials by source. Eventhough preliminary, the tests we have conducted seem very encouraging about the possibility of comparing the partials using our new model.

Chapter 5

Order-2 Sinusoidal Modelling

In the preceding chapter, we have tried to model the control parameters. We will now study and extract the evolution of these control parameters over time.

Indeed, in the case of vibrato and tremolo, the parameters of the sinusoids present oscillations over time. These oscillations represent the fluctuations of the frequency and the amplitude over time that characterise the vibrato and tremolo. Hence, by resampling the parameters of the partials, the oscillations of the amplitude and frequency are not preserved. Indeed, the frequency of the oscillations of the frequency and amplitude parameters of a sinusoid is decreasing as the sound gets longer and increasing as the sound gets shorter.

These alterations of the frequency of the tremolo and vibrato that happen for important scaling factors are not acceptable for an artifact-free time-scaling algorithm. Hence we had to find a way to keep the frequencies of the tremolo and vibrato at realistic values.

In the order-2 spectral model, the time-scaling would preserve the vibrato and the tremolo of a sound, but it would also suffer from the same shortcomings as spectral modelling applied to the sound. In order to overcome this, an equivalent to sinusoidal modelling for control parameters will be presented here.

Time-scaling of sounds without alteration of the vibrato has already been studied, and this work will be presented in this chapter, along with the limitations of this algorithm.

Then, we present the research Sylvain Marchand and I have carried out on the preservation of tremolo and vibrato during time-scaling. This work is based on an order-2 sinusoidal model, that is a set of parameters (order-2 partials) that control the control parameters (well-known partials) of the sound.

These order-2 partials allow us to work directly on the control parameters of the modulations (vibrato and tremolo) of the partials (now called order-1 partials). Thus we achieve time-scaling without alteration of the vibrato and the tremolo.

5.1 Previous work on vibrato and tremolo

The issue of vibrato and tremolo alteration has been addressed by Arfib & Delprat [AD98, AD99] for the case of monophonic and harmonic sounds. This work is based on time-frequency analysis of the sounds. Indeed, decomposing the sound with Gabor grains and then cepstra, an overall control shape of the pitch of the sound is found. From this control shape (called pitch curve), the modelling of the vibrato curve is then performed. It has been assumed in the work of Arfib & Delprat that the vibrato is quasi-sinusoidal, and that the shape of the vibrato is not of special interest. Hence, it is possible to describe the pitch curve as:

$$F(t) = F_0(t) + DF(t) * \sin(2\pi t \cdot F_v(t))$$

where DF is the amplitude of the vibrato, F_v is the frequency of the vibrato and $F_0(t)$ is the pitch curve. This model is thus describing the vibrato as a single sinusoid.

The tremolo is also indirectly taken into account in this approach of Arfib & Delprat, since it is considered to be strongly linked to the pitch curve through the formants of the sound. Indeed, the resonance describes the harmonic structure of the sound (based on the spectral envelope of the sound), thus the link between the vibrato and the tremolo. Hence, only the vibrato needs to be manipulated in order to have the corresponding tremolo track. This hypothesis simplifies the model a lot and seems to give good results.

From this point, it is possible either to suppress the vibrato and tremolo of the sound by conserving only the pitch curve, or to synthesise a new vibrato and tremolo by adding a new sinusoid to the pitch curve.

These two operations are needed for time scaling. Indeed, the first step of the time-scaling process is to remove the tremolo and vibrato of the sound, then it is necessary to scale the pitch curve, and finally apply a new vibrato (and tremolo) to the scaled pitch curve. From this modified pitch curve it is possible to resynthesise a scaled sound that has a vibrato with a reasonable rate.

5.2 Hierarchical modelling

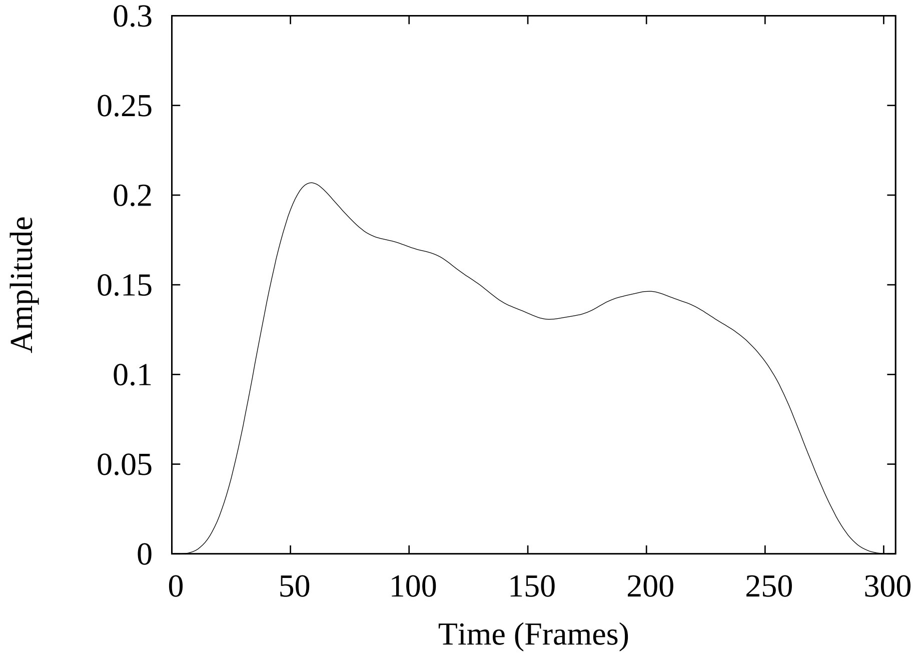
The vibrato manipulating method presented above (the work of Arfib & Delprat) uses the time-frequency representation of the sound. Even though it does not use the same process as ours (they use a source-resonator model, we use a sinusoidal model), it is not incompatible. Indeed, we have successfully applied their algorithm – vibrato removal and replacement – to frequency and amplitude tracks of the sinusoidal model we are using.

The main problem with the previously presented vibrato and tremolo synthesis however, is that the vibrato and tremolo rates and amplitudes are fixed and succinctly described. Hence, the vibrato and tremolo do not have any evolution in frequency and amplitude, and are represented by only one sinusoid, giving the vibrato and tremolo a sinusoidal shape. These approximations give good results, but they sound too perfect. Another problem is that the sound is supposed to be harmonic, that is, it supposes a certain relation between the different partials of the sound and monophonic. This is certainly not the case for most sounds.

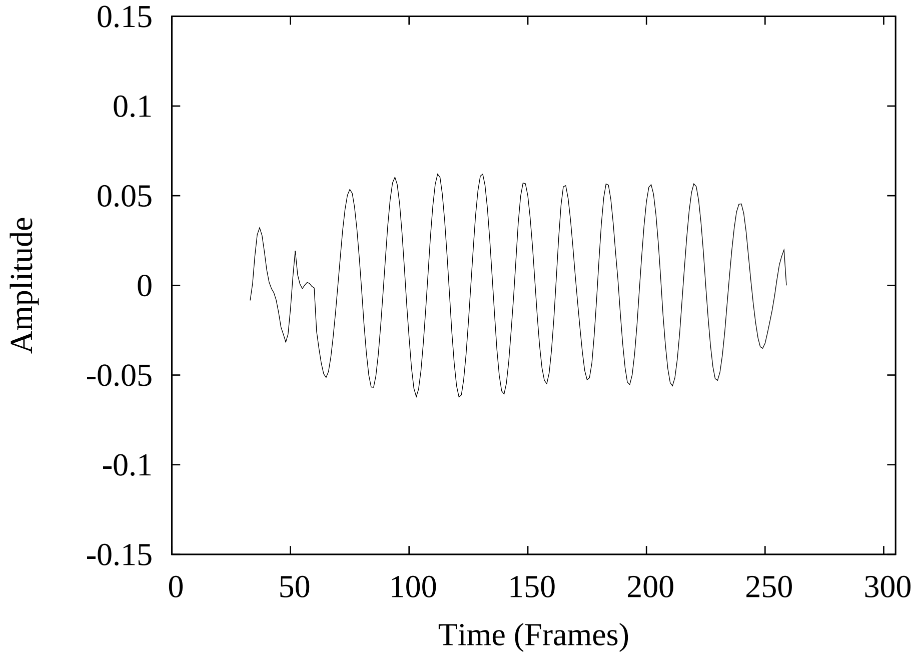
Then, how can we make the synthesised sound more natural? The solution we found is to use the original evolutions in frequency and amplitude of the vibrato and tremolo in the synthesised sound. In the context of time scaling, the idea is to also scale the evolutions of the vibrato and tremolo of the sound accordingly.

Hence, we have to gather information about these evolutions. Since the vibrato and the tremolo can be viewed as periodic signals in the parameters of the partials of the sound, it is possible to extract them to obtain the frequency and the amplitude of both the vibrato and the tremolo. Such extraction methods have been presented in Section 4.1.3 on page 74.

The main idea is to consider each control parameter track as a signal containing an envelope and an oscillatory modulation (see Figure 5.56 on the next page).



(a) Amplitude Envelope



(b) Tremolo

Figure 5.56: *Envelope (a) plus modulation (b) decomposition of the amplitude of the second partial of the saxophone. The envelope is the low-pass filtered version of the amplitude of the partial. The modulation is obtained by subtracting this envelope from the original amplitude.*

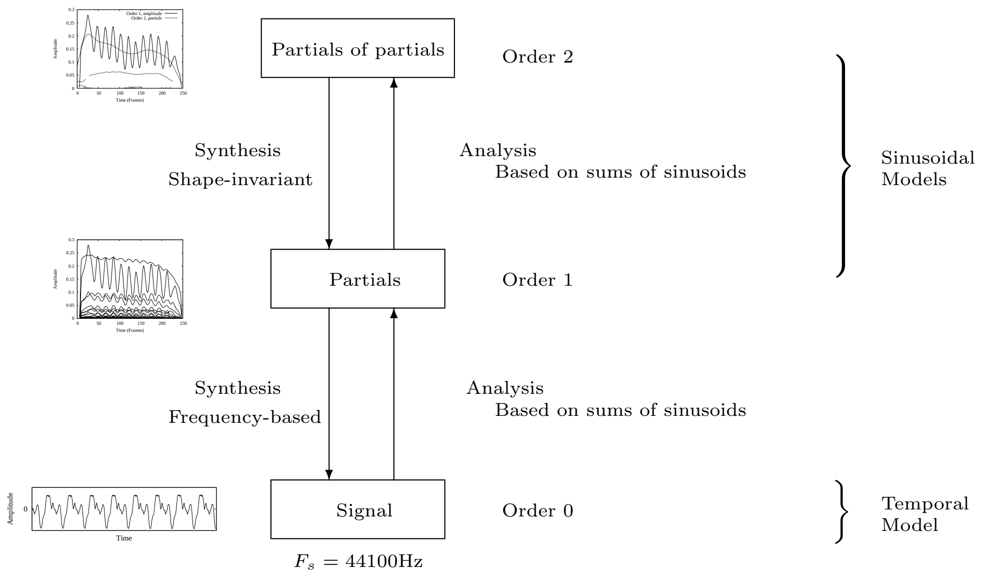


Figure 5.57: *Showing the process of analysis and synthesis with the hierarchical model using frequency and amplitude control parameters. Analysing only on frequency and amplitude components allows to work with the classic sinusoidal analysis.*

In a first time we have studied the frequency and amplitude control parameters in order to analyse the vibrato an tremolo of the sound. In a second time, we studied the phase and amplitude control parameters to obtain the vibrato and tremolo.

5.2.1 Work on frequency and amplitude control parameters and enhanced time scaling

In order to track the evolutions of the modulations of the frequency and amplitude parameters, it is important to understand that we can consider the frequency and amplitude control parameters as discrete signals. Indeed the control parameters are conveying information over time, that is to say, the frequency and the amplitude of a given sinusoid present in the sound at a given moment. Moreover, since the parameters of this sinusoid has been tracked using a short-time Fourier transform over a discrete signal, it is easy to understand that the control parameters are also discrete signals.

Hence, in order to investigate the evolutions of the periodicities (modulations) of the frequency and amplitude parameters, we can use the same tools as we were using over the sound as discrete signal. Thus, we can use the short-time Fourier transform over the tracks of a given partial. This way, we obtain phase, frequency, and amplitude parameters of the sinusoids detected in a given amplitude or frequency track, and we can build partials of this given amplitude or frequency track.

We call these partials of control parameters order-2 partials, because they are partials of partials. In the same fashion that we call order 2 the second level of sinusoidal analysis, we call order 1 the classic first level of sinusoidal analysis and order 0 the temporal level.

Analysis

In order to obtain the vibrato and tremolo parameters from the frequency and amplitude control parameters, it is necessary to analyse these control parameters. As seen in Section 4.1.2 on page 73, one solution is to model the control parameters as a sum on sine functions. This model gives us the partials for the oscillating part. If the window of signal to be analysed is not zero-centred, then the first bin of the Fourier transform gives the mean of the signal over that given analysis window.

In order to capture the evolution of the vibrato and tremolo, we are using a sliding STFT with the purpose of using the sinusoidal model over the control signal.

Performing a sinusoidal analysis over each piece of the control signal allows also to separate the envelope of the control signal from the modulations. Indeed, since we use a Fourier transform over each window, we can extract the mean value of the control signal over each window. This mean value is in fact given by the first bin of the Fourier transform. Hence, by gathering all the mean values (0 degree polynomial) of the pieces of the signal, we actually perform a low-pass filtering on the control signal. Thus, we obtain the envelope of the signal. The rest is then the modulation.

An example of this decomposition is shown on Figure 5.56 on page 109.

During this sliding window analysis of the discrete control signal, we perform a Fourier transform on each piece of the signal. Apart from the first bin of the Fourier transform that is kept for the envelope, the sinusoidal analysis is performed as usual, with peak extractions and the formation of partials. Those partials, called order-2 partials, describe the evolution of the modulations of the control signal of the sound, that is the vibrato and tremolo.

Order-2 partials are shown on Figures 5.58 on the following page, 5.59 on page 113, and 5.60 on page 114. The chaotic behaviour of the small order-2 harmonics can be explained by two reasons. First, the time scale has been dramatically reduced due to the shift from order-1 to order-2 sinusoidal analysis. Indeed, the sample-rate shifted from 44100 Hz to 86 Hz, and thus we have a lot less data to work with. The second reason is that the order-2 harmonics have very low amplitudes, which might contribute to the capture of noisy peaks by the order-2 tracking algorithm.

Synthesis

The synthesis within the order-2 model consists of two levels. First, the frequency and amplitude parameters of the (order-1) partials are reconstructed – synthesised – from the order-2 parameters (phase and amplitude only). This method is shape-invariant, and thus the vibrato and tremolo are kept as close to the original as possible. We obtain partial trajectories almost identical to the original, except for the transients though.

Second, the (order-0) audio signal is synthesised from these synthetic order-1 partials (using their frequency and amplitude) by the technique described in section 3.5.1 on page 58. Note that this technique is not shape-invariant, but for now we only use the frequencies and amplitudes (thus not the phases) of the synthetic partials to generate the result. However, as mentioned before, this makes almost no audible difference for most sounds.

Using the classic (order-1) sinusoidal parameters, the time-scaling operation would consist in resampling these parameters to the desired length. This was explained in section

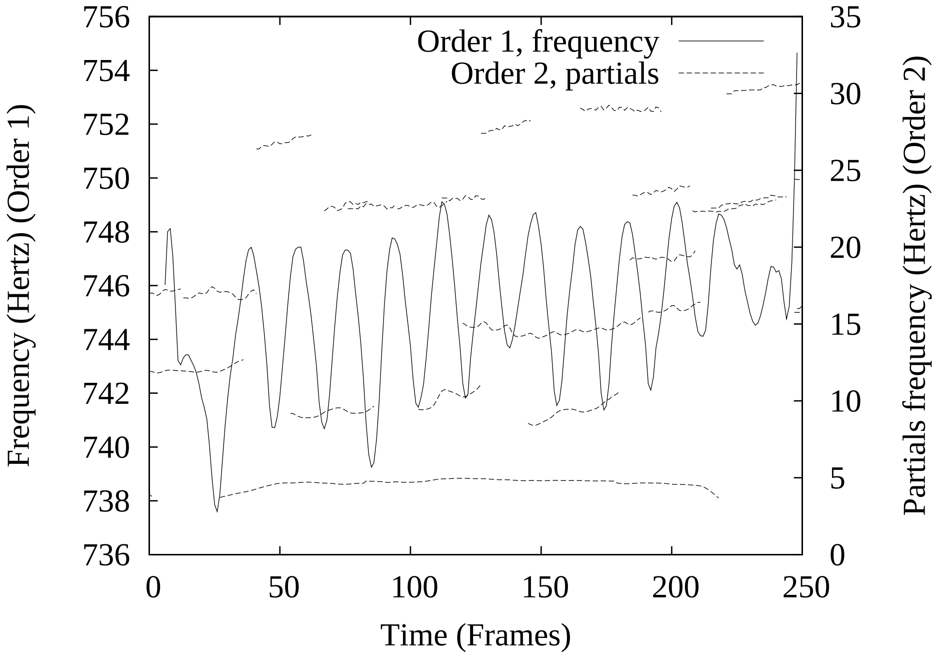


Figure 5.58: *Frequencies of the order-2 partials (dashed) for the frequency of the second partial (plain) of the saxophone. We can see on this figure that there is a main order-2 partial that has an almost constant frequency trajectory at 5 Hz. Indeed, it corresponds to the rate of the vibrato that we can see as a solid line. We can also see that there are smaller order-2 frequency trajectories. Indeed, they seem to be harmonics of the modulation frequency.*

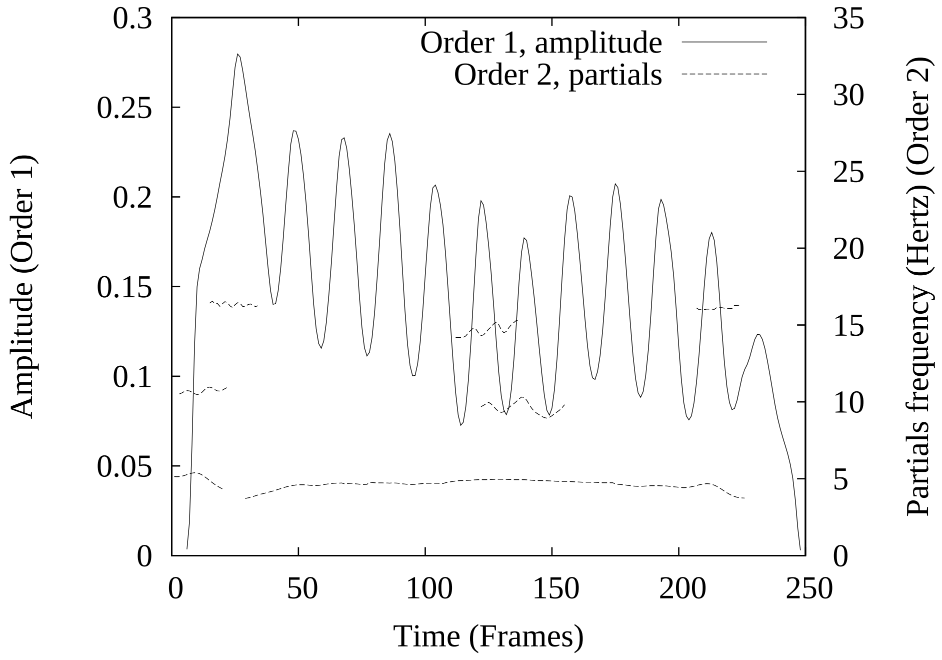


Figure 5.59: *Frequencies of the order-2 partials (dashed) for the amplitude of the second partial (plain) of the saxophone. On this figure too, it is clear that there is a tremolo modulation at an almost constant rate of 5 Hz, and which looks very much like the one on the vibrato on the preceding figure. Hence, the belief that tremolo and vibrato are linked seems to be confirmed. The amplitude curve is smoother than the frequency curve presented on the previous figure, and hence this amplitude curve has less harmonics.*

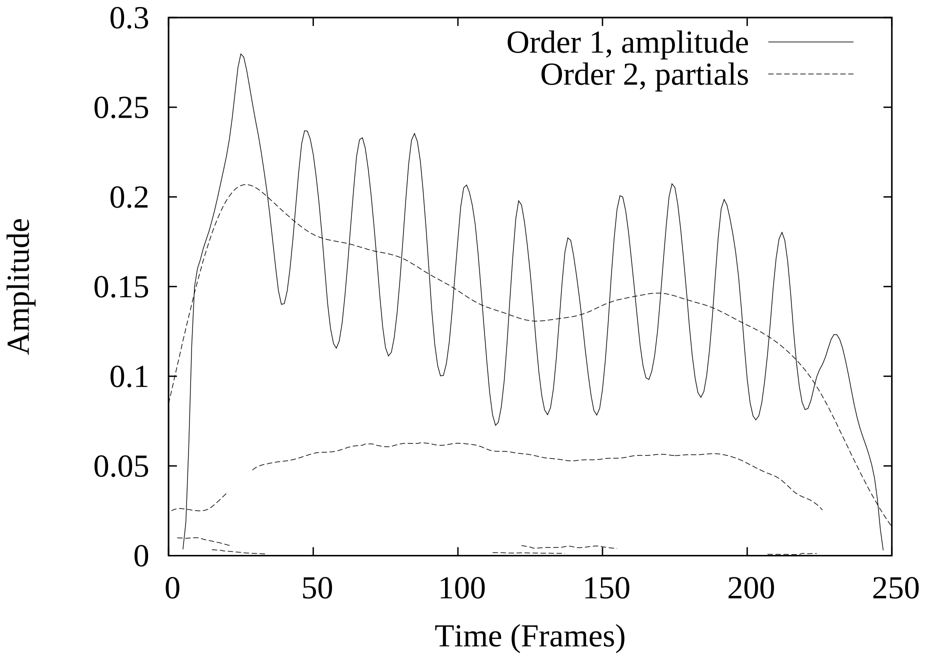


Figure 5.60: *Amplitudes of the order-2 partials (dashed) for the amplitude of the second partial (plain) of the saxophone. On this figure, we recognise the envelope that follows the amplitude curve, but with smoothed attack, which modifies the final sound. The main order-2 partial is also present, at a reasonable level. The harmonics on the other hand are all very close to zero in amplitude. Thus, this might explain the chaotic trajectories they showed on the previous figures.*

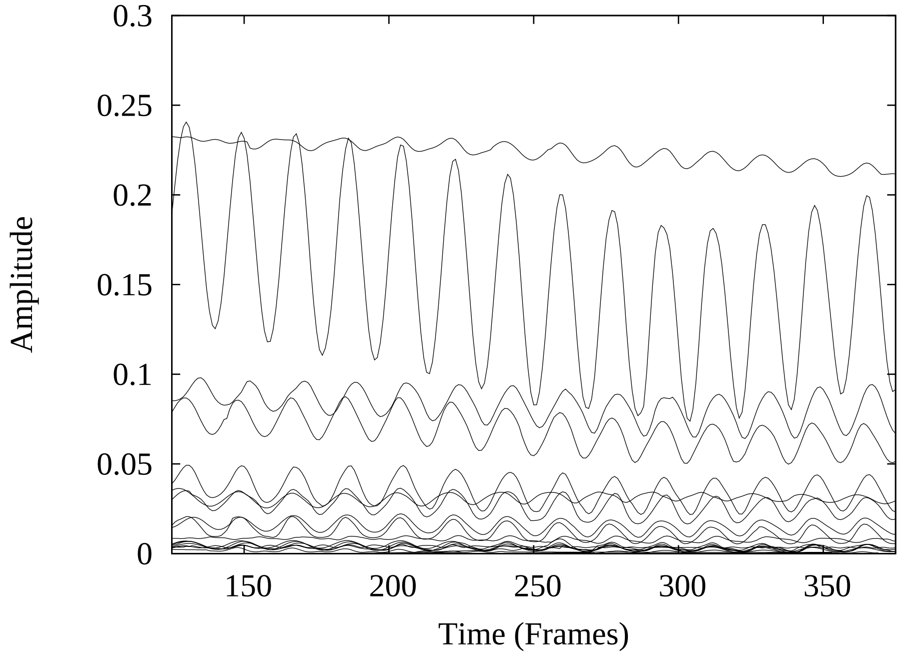


Figure 5.61: *Enhanced time-stretching (order 2) of the saxophone sound, by a factor 2. The amplitudes of the partials are shown at the middle of the resulting sound. The shape and rate of the tremolo are preserved (see Figure 3.25(b) on page 55 for a comparison).*

3.5, and we concluded that vibrato and tremolo were not conserved. We use the same technique for our enhanced time-scaling, however applied to order-2 partials (*i.e.* partials of partials) instead of order-1 partials. By resampling order-2 partials to the desired length, we obtain after the synthesis a scaled sound with the original vibrato and tremolo rate. Moreover, since we use a shape-invariant method to synthesise the extended vibrato and tremolo, they are very similar to the original in shape too.

Figure 5.61 is an example of this enhanced time-stretching on the saxophone sound shown on Figure 3.25(b) on page 55. We can see that the tremolo shape and rate on the first figure are very close to the ones of the second, but that the evolution of this tremolo was slowed down, as the global envelope was scaled.

However, the main drawback of this method is the drawback of sinusoidal modelling in general, that is the smoothing of sudden changes such as the attack. Moreover, the original waveform of the temporal (order-0) sound is not kept, since the last synthesis is not shape-invariant (see Figure 5.57 on page 110).

This order-2 model has been compared with other vibrato modelling techniques in the work of Verfaille *et al.*[VGD05]. It seems that it performs very well in regard to the other methods, and appears to be perceptually sound.

5.2.2 Work on phase and amplitude control parameters

In order to solve the second problem, thus to make the last synthesis shape-invariant, it is necessary to synthesise the temporal sound using phase parameters instead of frequency

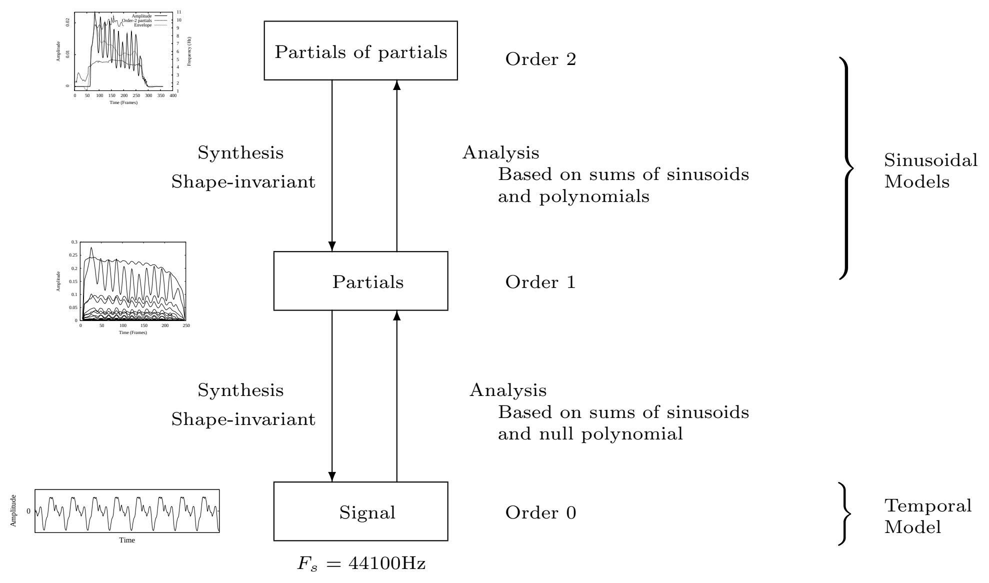


Figure 5.62: *Showing the process of analysis and synthesis with the hierarchical model using phase and amplitude control parameters. The sinusoidal analysis are performed using the sums of sinusoids and polynomials, according to newly presented model (see section 4.1.3 on page 74). This new model allows all synthesis steps to be shape invariant, since the phase can be modelled better.*

parameters.

Then, it is necessary to analyse the phase trajectories of the partials in order to estimate their control parameters. The major issue with this is that phase parameters are not usually constant, as opposed to the frequency parameters that can be considered as quasi constant over a short period of time. Hence, if we perform a sinusoidal analysis of a phase partial trajectory, it can be problematic if we consider it as a sum of sine functions. Indeed, the linear term of the phase trajectories could considerably alter the Fourier analysis. Hence, the analysis method presented in section 4.1.2 on page 73 may not apply on the phase parameters.

Instead, we have designed the Poly-Sin model (see section 4.1.3 on page 74). This model represents the signal as a combination of a polynomial (of a maximum degree of 3) and a sum of sinusoids. This model was designed for phase trajectories. Indeed, since the frequency is the derivative of the phase, considering the frequency to be at least constant, the phase is at least linear, that is a polynomial of first degree. But in case of a glissando, we can imagine the frequency to be at least linear, if not quadratic. This would make the phase to be of third degree at most.

The idea behind our new Poly-Sin model is that we need order-2 partials to be based on phase rather than frequency tracks, so that we can explore even higher levels without waveform modification at the different levels, which could turn out to have disastrous effects. Indeed, with the frequency analysis only, considering the fact that the synthesis is not shape invariant, the vibrato and the tremolo would have been incorrect. We needed the phase. This phenomenon would be happening also if we wanted to go to order-3

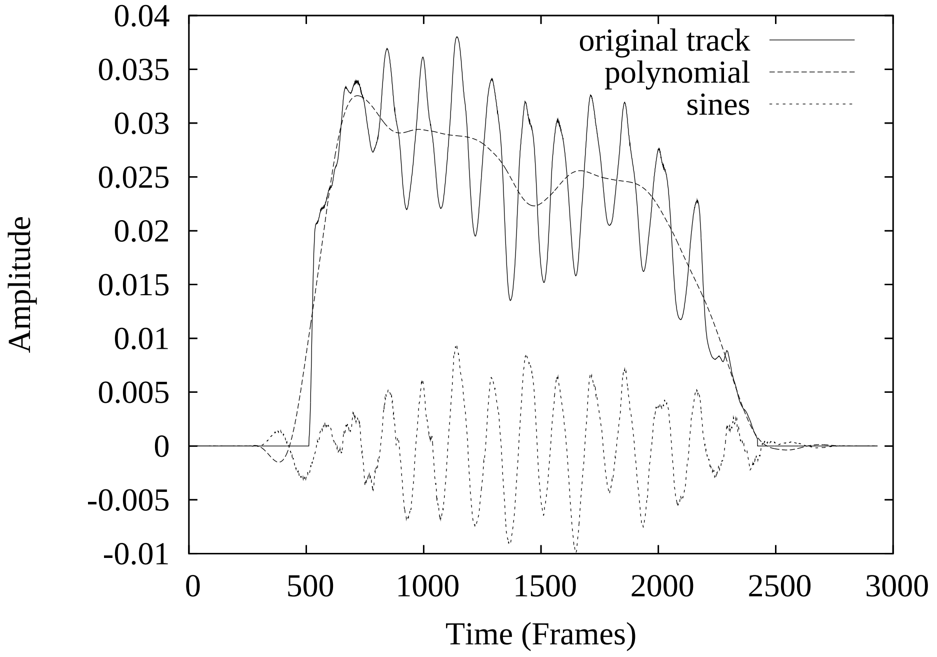


Figure 5.63: *Decomposition of the amplitude of a partial (control signal) showing the two steps of an analysis using a polynomial and sinusoids. The (order-2) frames are estimated at each sample of the evolution of the (order-1) partial, using 256-sample windows. The envelope here has been extracted using polynomial estimations, as opposed to the previous low pass-filtering performed by the first bin of the Fourier transform we used earlier.*

(partials of partials of partials) or more. Actually, each new level captures modulations of the preceding level and diminishes the sampling rate. At the highest level, we expect the control signal to be constant, which would be very easy to time-scale. Thus, the use of a polynomial estimation would allow us to correctly analyse the phase tracks of our partials.

Indeed, we assume that the phase and amplitude tracks of our partials are in fact composed of sums of sinusoids and polynomials. Though one could argue that polynomials could be approximated by sinusoids and *vice-versa*, in our analysis procedure, the polynomial is gathering the global envelope of the signal while the sinusoids are gathering the oscillations of the signal, thus separating our signal in the two required components: envelope and sinusoids. This is done using the method presented in section 4.1 on page 70.

An illustration of this decomposition, obtained after the re-analysis of an order-1 partial, is shown on Figure 5.63. The sinusoidal part is the result of the re-synthesis of order-2 partials.

Figures 5.64 on the next page and 5.65 on page 119 show an amplitude track together with its order-2 partials and envelope. We can see that the order-2 partials are mainly present when the modulation is mostly active.

These order-2 partials are obtained by the re-analysis of the partial, again using an analysis window. A suitable window size is when the analysis window contains at least two periods of the oscillations, if any. In our context, these oscillations represent musical parameters of the sound such as vibrato and tremolo. We have considered in the scope of

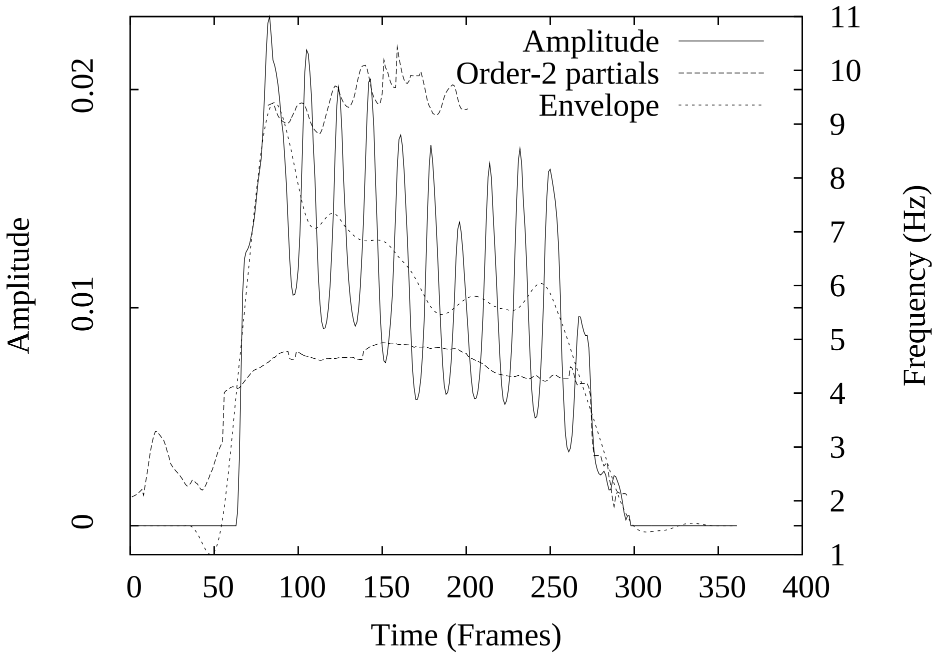


Figure 5.64: *Order-2 frequency partials and envelope of an amplitude track. The solid line represents the original amplitude track, the dashed lines represent the order-2 frequency partials, and the dotted line represents the polynomial envelope of the amplitude track. On this order-2 analysis, there is only one harmonic. This leads us to think that the polynomial envelope has captured the main curve much better, to leave an almost pure modulation signal.*

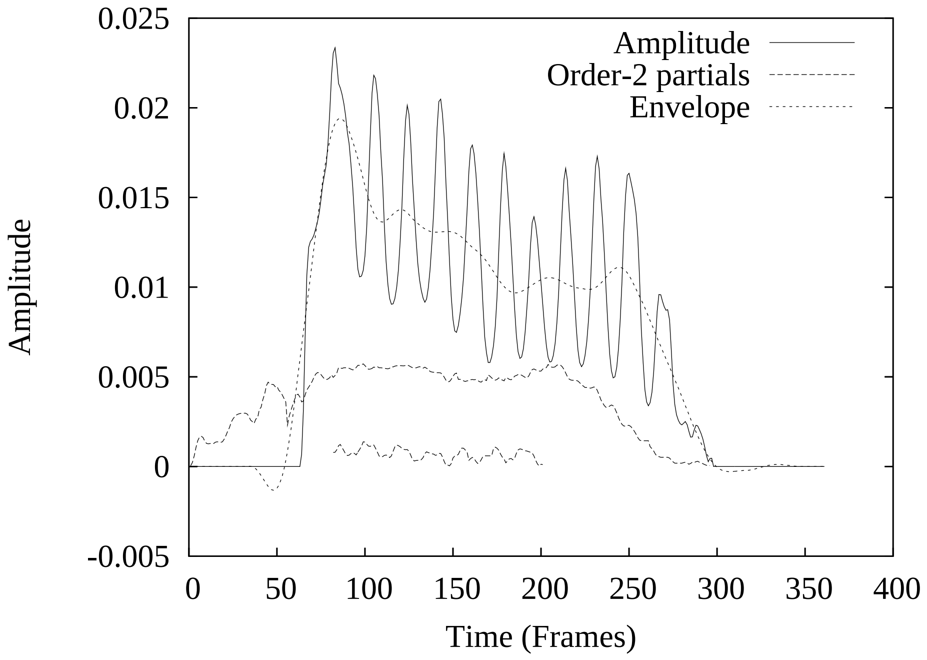


Figure 5.65: *Order-2 amplitude partials and envelope of an amplitude track. The solid line represents the original amplitude track, the dashed lines represent the order-2 amplitude partials, and the dotted line represents the polynomial envelope of the amplitude track. The lonely harmonic has a very small amplitude, which confirms the close-to-sinusoidal shape of the modulation.*

our study that the minimal vibrato and tremolo rates are around 3 Hz. This estimation leads to an easy computation of the minimal window size (to have at least two periods of the vibrato in the analysis window), which is two times the sampling frequency divided by the minimal vibrato and tremolo rates.

5.3 Non-uniform time scaling at order 2

Using order-2 sinusoidal sound modelling is then a solution for non-uniform time scaling conserving vibrato and tremolo. Indeed, if the partials of the sound are modelled using a Poly-Sin model for example (presented earlier in section 4.1.3 on page 74), it is then possible to obtain a freeze function without alteration of the vibrato and tremolo.

The way this time-scaling would be performed is actually very close to the way it happens at order 1. Indeed, it is based on a simple resampling of the order-2 partials. Once the resampling has been done (locally) at the given local rate, it is possible to compute the values of the order-1 sinusoidal parameters. These order-1 partials are in turn resynthesised using the classic method.

This application on order-2 time scaling has not been implemented yet however. Surely, such an application would yield satisfactory results, even though it would have the attack problems that are characteristic of the higher-level sinusoidal models. Thus, maybe the use of the transient identification techniques presented in the works of Röbel [RÖ3] might be a solution for the attack problem.

5.4 Conclusion

We have presented an application of sinusoidal modelling to control parameters of the sound. Based on the analysis method described in the previous chapter, we successfully constructed order-2 partials describing the evolutions of the modulations of the partials.

Using these order-2 partial, it is now possible to time-scale sounds without altering the timbre, pitch nor vibrato/tremolo rates.

The next step is then to apply this technique recursively to obtain even higher level time-scaling methods.

Chapter 6

Higher Order Modelling

Considering the hierarchical modelling we have just created with order-2 partials on top of the classic partials, we imagined we could go even further. Hence, after order-2, we explored the possibility of extracting the modulations of the melody of a tune for example. This would be based on the same principles, that is finding periodicities in the melody, at least for very repetitive music. However, this turns out to be much more difficult than we expected, so we present this exploratory work and leave it to be finished.

6.1 Musical modelling

One of the major objectives of this thesis is time-scaling. Since we started to build a hierarchy over the two sinusoidal of order 1 and order 2, we start to think that maybe a hierarchical time-scaling is possible.

Hence, one might imagine that even higher order models (than order 2) might exist. Since we do not know if this hypothesis is correct and since we do not know how these models might evolve when we reach each time a higher level, we decide for now to have a look directly at the musical level, without exploring the transition between order-2 level and the musical level.

What we call the musical level is the level where musical patterns are heard, and thus where melodic repetitions might be detected.

The search for musical patterns is manifold. For example, one method based on networks and combinatorics has been presented by Lartillot-Nakamura ([LN04]). This seems to mimic the listening pattern of humans.

Another example is the Continuator, presented by François Pachet [Pac02]. This system is a hybrid between interactive music systems, that can be limited by their ability to generate coherent content and music imitation systems, which are not interactive. The purpose of the Continuator is to allow musicians to extend their playing with stylistically consitent, automatically learnt material. The system is able to learn and generate music from any style, either in standalone mode, as continuations of the musician's input, of as interactive improvisation backup.

The Continuator is based on a Markov model of musical styles, but it also takes into account musical issues such as rhythm, beat, harmony and imprecision. The learning scheme is based on an indexing method which represents all the subsequences found in the corpus input by the musician, which are then in turn used to feed the continuation mechanism.

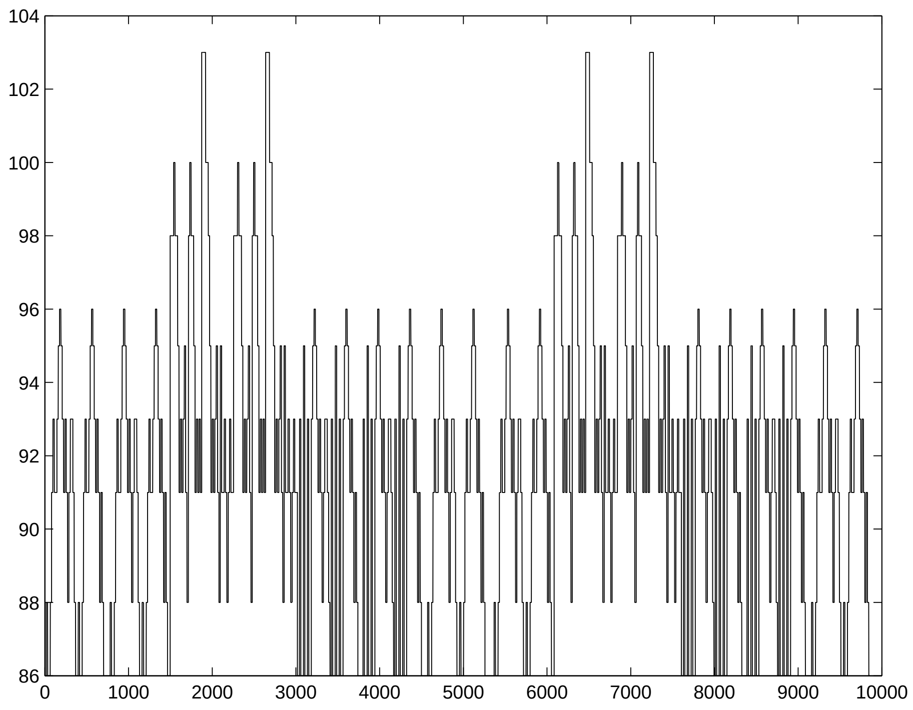


Figure 6.66: *The representation of a melody as a signal of the discrete MIDI pitch in function of time.*

Our approach however is totally different. It is based on a signal approach of the music. Indeed, we transform melody as set of pitch values ordered by time. Figure 6.66 shows such a signal. In our experiments, such melodies were taken from MIDI files, hence the pitch value corresponds to the discrete MIDI pitch value.

The next step is to find the periodicities of the signal. Our idea is to use the Fourier transform to find the periodicities of the signal. It has to be clear that in order to experiment the detection of periodicities when the melody is represented as a signal, only very repetitive musical melodies have been chosen.

Figure 6.67 on the facing page shows the Fourier spectrogram of the flute melody of an Irish tune represented in Figure 6.66. Even though there are clearly structures in the spectrogram, it seems quite difficult to computationally find the patterns in this spectrogram. Indeed, since the melody signal is not continuous, the Fourier transform creates harmonic combs spanning over the whole spectrum (since the spectrum of a square signal is such a comb). Hence the sinusoidal function used for the Fourier decomposition might not be particularly suited for the transformation of this signal.

Thus, we decided to use a square function basis instead of a sinusoidal basis (such as the one used for the Fourier transform).

6.2 Finding base functions

The Walsh functions [Wal23] form such a basis (Figure 6.68 on page 124). However, these functions might not be of more use than the sinusoidal function. Indeed, we want to have

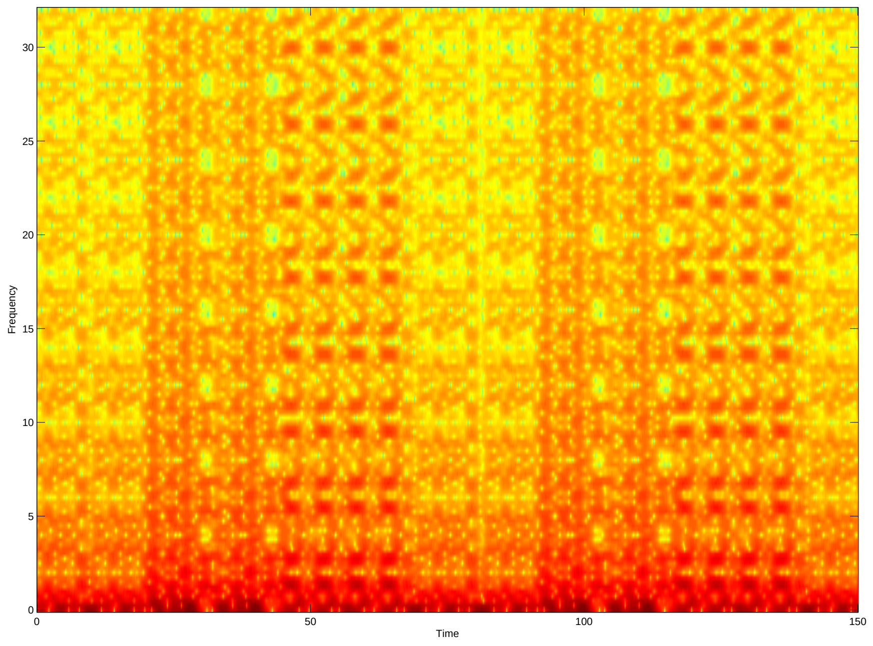


Figure 6.67: *Spectrogram of a repetitive flute melody from an Irish tune. The song is composed of three verses and two choruses.*

a mathematical transform that would give exploitable results. By exploitable we mean that the resulting spectrum is sparse and easily readable, such as the Fourier spectrum of simple sinusoidal signals. For example, it is a requirement that slow variations and quick variations are well distinguished. With the Walsh functions it is clear that such distinction cannot be made, since some of the function mix slow and quick variations.

Therefore, we propose a new set of functions. The idea is to produce box train functions of increasing frequency, in order to mimic with square functions the behaviour that the Fourier transform has with sinusoids.

The transform of such functions would require to fulfil certain properties, actually corresponding to some musical properties. Among them are:

- The offset invariance, which represents a pitch shift in the music domain.

- The shift invariance, which represents a change in the timing of the notes.

- The regularity, meaning that two identical sequences of notes would have the same transform.

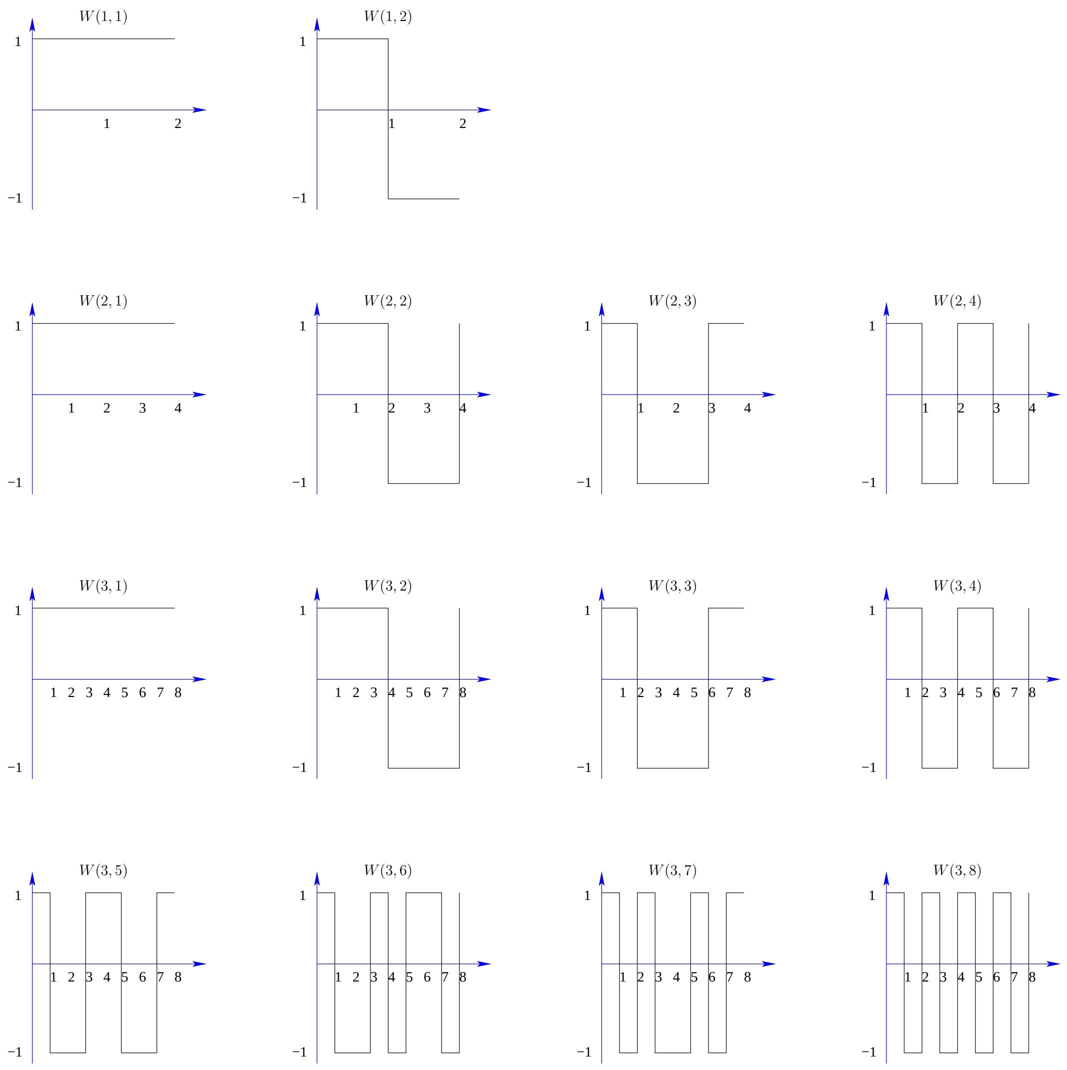


Figure 6.68: *The Walsh functions, forming a basis of 2, 4 or 8 square functions.*

6.2.1 Box functions

Let b_M (M integer, $0 < M$) be the box function defined by:

$$b_M[n] = \begin{cases} 1 & \forall n \in [0; M[\\ 0 & \text{elsewhere} \end{cases} \tag{6.98}$$

In order to find a basis for the MIDI signals, we will check if any spectrum can be found by a combination of box train function spectra. Hence we need to determine the discrete spectrum of a box function.

The discrete spectrum of size $N \geq M$ of a box function b_M is given by:

$$B_{M,N}[m] = \sum_{n=0}^{M-1} e^{-i2\pi \frac{mn}{N}} \tag{6.99}$$

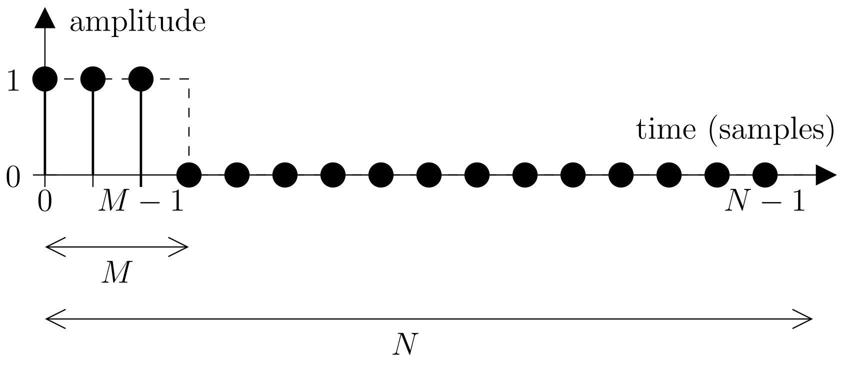


Figure 6.69: *Box function b_3.*

By defining

$$R_M(x) = \sum_{n=0}^{M-1} e^{-i2\pi nx}$$

Then, we have

$$B_{M,N}[m] = R_M\left(\frac{m}{N}\right)$$

When x is an integer, $e^{-i2\pi x} = 1$, then $R_M(x) = M$ since all the terms of the sum equal 1. Otherwise, if x is not an integer, then $e^{-i2\pi x} \neq 1$ and we recognise the sum of the M terms of a geometric series, thus:

$$\begin{aligned} R_M(x) &= \frac{1-e^{-i2\pi Mx}}{1-e^{-i2\pi x}} \\ &= \frac{e^{-i\pi Mx}}{e^{-i\pi x}} \cdot \frac{e^{+i\pi Mx} - e^{-i\pi Mx}}{2} \cdot \frac{2}{e^{+i\pi x} - e^{-i\pi x}} \\ &= e^{-i\pi(M-1)x} \cdot \frac{\sin(\pi Mx)}{\sin(\pi x)} \\ &= e^{-i\pi(M-1)x} \cdot S_M(x) \end{aligned}$$

where

$$\begin{cases} S_M(x) = \frac{\sin(\pi Mx)}{\sin(\pi x)} & \forall x \neq 0 \\ S_M(0) = \lim_{x\to 0} \frac{\sin(\pi Mx)}{\sin(\pi x)} = M \end{cases} \tag{6.100}$$

Thus, the phase spectrum of the box function is:

$$\angle R_M(x) = -\pi(M-1)x + \pi\delta(S_M(x)) \tag{6.101}$$

where

$$\delta(x) = \begin{cases} 1 & \text{when } x < 0 \\ 0 & \text{otherwise} \end{cases}$$

and the amplitude spectrum is:

$$|R_M(x)| = |S_M(x)| \tag{6.102}$$

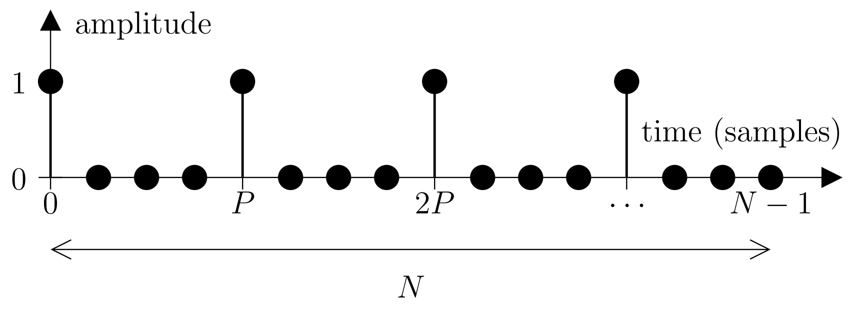


Figure 6.70: *Impulse train function $\diamond i_{P,N}$ (here for $P = 4$, $N = 16$).*

In the case where $M = N$, then the box function is in fact a constant function of 1, and the phase is undefined.

Now that the box function has been studied, let us have a look at the impulse train function.

6.2.2 Impulse train functions

An impulse train function is a train of Dirac impulses that are regularly spaced (see Figure 6.70). Let $\diamond i_{P,N}$ be an impulse train function of length N and of periodicity P.

$$\diamond i_{P,N}[n] = \begin{cases} 1 & \text{if } n \mod [P] = 0 \\ 0 & \text{elsewhere} \end{cases} \tag{6.103}$$

where P is a divisor of N ($N \mod [P] = 0$).

The Fourier spectrum of this function is then:

$$\begin{aligned} \Diamond I_{P,N}[m] &= \sum_{n=0}^{\frac{N}{P}-1} e^{-i2\pi \frac{mnP}{N}} & (6.104) \\ &= R_{\frac{N}{P}}\left(\frac{mP}{N}\right) & (6.105) \end{aligned}$$

$$\Diamond I_{P,N} = \frac{N}{P} \diamond i_{\frac{N}{P},N} \tag{6.106}$$

Hence the spectrum of an impulse train function is an impulse train function.

6.2.3 Box train functions

Now that both the impulse train and the box functions have been studied, we can go on to the box train functions. These are simply the convolution of an impulse train function with a box function. An example of such a function is shown on Figure 6.71 on the facing page.

$$\diamond b_{P,M,N} = b_{M,N} * \diamond i_{P,N} \tag{6.107}$$

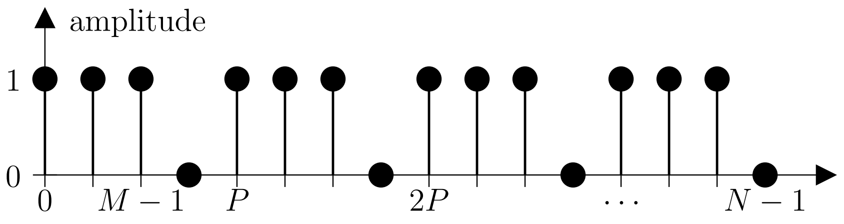


Figure 6.71: *Box train function $\diamond b_{M,P,N}$ (here for $M = 3$, $P = 4$, $N = 16$).*

The spectrum is then:

$$\begin{aligned}\Diamond B_{P,M,N} &= B_{M,N} \cdot \Diamond I_{P,N} \\ &= R_M\left(\frac{m}{N}\right) \cdot R_{\frac{N}{P}}\left(\frac{mP}{N}\right)\end{aligned}$$

Hence the amplitude spectrum is:

$$\begin{aligned}|B_{P,M,N}| &= \left|R_M\left(\frac{m}{N}\right)\right| \cdot \left|R_{\frac{N}{P}}\left(\frac{mP}{N}\right)\right| && (6.108) \\ &= \left|S_M\left(\frac{m}{N}\right) \cdot S_{\frac{N}{P}}\left(\frac{mP}{N}\right)\right| && (6.109)\end{aligned}$$

and the phase spectrum is:

$$\begin{aligned}\angle B_{P,M,N} &= \angle R_M\left(\frac{m}{N}\right) + \angle R_{\frac{N}{P}}\left(\frac{mP}{N}\right) && (6.110) \\ &= -\pi\left(\frac{M+N-(P+1)}{N}\right)m + \pi\delta\left(S_M\left(\frac{m}{N}\right) \cdot S_{\frac{N}{P}}\left(\frac{mP}{N}\right)\right) && (6.111)\end{aligned}$$

6.2.4 Linearly independent system

The aim is now to find a basis for square functions. Let us suppose that the length of the vectors of our basis is $N = 2^L$. We define the functions $v_{k,L}$ such that:

$$\begin{aligned}v_{0,L} &= b_{2^L,2^L} = \diamond b_{2^L,2^L,2^L} \\ v_{k,L} &= \diamond b_{2^{L-k},2^{L-k+1},2^L} \quad \forall k \in]0; L]\end{aligned} \tag{6.112}$$

The spectrum of each of these functions is simply the spectrum of the corresponding box train functions:

$$V_{k,L} = \Diamond B_{2^{L-k+1},2^{L-k},2^L} \tag{6.113}$$

Figure 6.74 on page 130 shows the spectra of a set of the linearly independent vectors we have defined. They show interesting properties in spectral form, but they are not sufficient to describe the entire space of the square functions. Indeed, the space described by the square functions is in fact the same space as the one described by sine functions, that is the entire space of signals. Hence, we need 2^L vectors to describe the discrete space of signals of length 2^L.

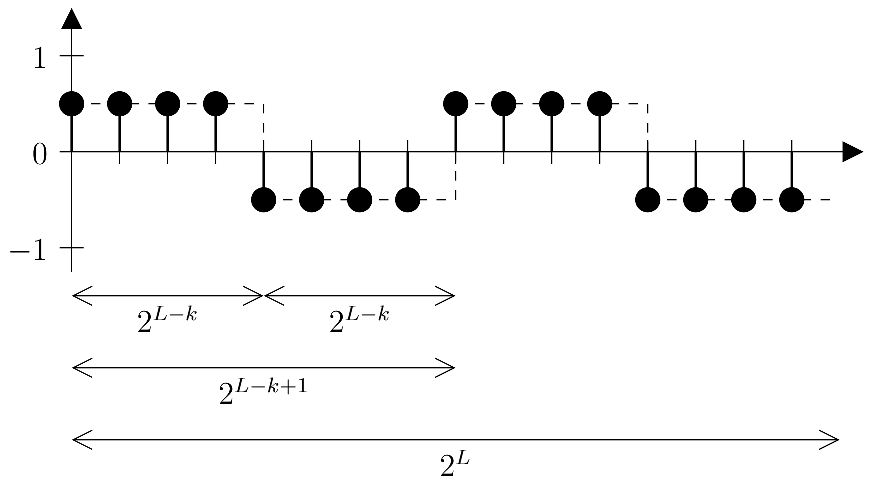


Figure 6.72: *Vector $v_{k>0,L}$ (here $k = 2$, $L = 2$).*

With the vectors we have defined we have $L+1$ vectors of a new basis. Hence we need $2^L - (L+1)$ more vectors.

One possible way to find the the missing vector is to start from a comb spectrum, that would be a subset of the spectrum of another function. For example, on Figure 6.74, we could try to find a square function which amplitude spectrum would be a subset of the amplitude spectrum of $v_{1,4}$ for example.

The functions we have defined can also be found as the Rademacher functions. There also exists a Rademacher transform based on these functions. The base is indeed made complete by adding the time-translated versions of the Rademacher functions.

Once the base vectors are found, they will allow the detection of periodicities in square functions. As for computer music, many applications would possibly arise from these base vectors. An obvious application is time-scaling for musical melodies. If we can find patterns in the music, we can then repeat those patterns in order to lengthen the music. Another application would be music summarising. If the patterns are found, then it is easy to grab all the different patterns and make a music summary. Surely, there would also be many other application to other domains of computer science.

6.3 Conclusion

In this chapter, we had a look at the possibilities of using a Fourier-like transform to melodies. This does not seems possible using the standard sinusoidal functions, even though the phase invariance property Fourier transform seems interesting for our purpose.

Moreover, more constraints would have to be defined in order to have a basis consistent with music. The definition of the musical constraints would be done by identifying suitable musical transformations and distance between musical patterns. However, this would lead us to the domain of musicology which is beyond the scope of this thesis.

The Rademacher transform presented earlier does not fulfil one musical constraint, which is the shift-invariance. Indeed, whatever the respective times of two pattern, if

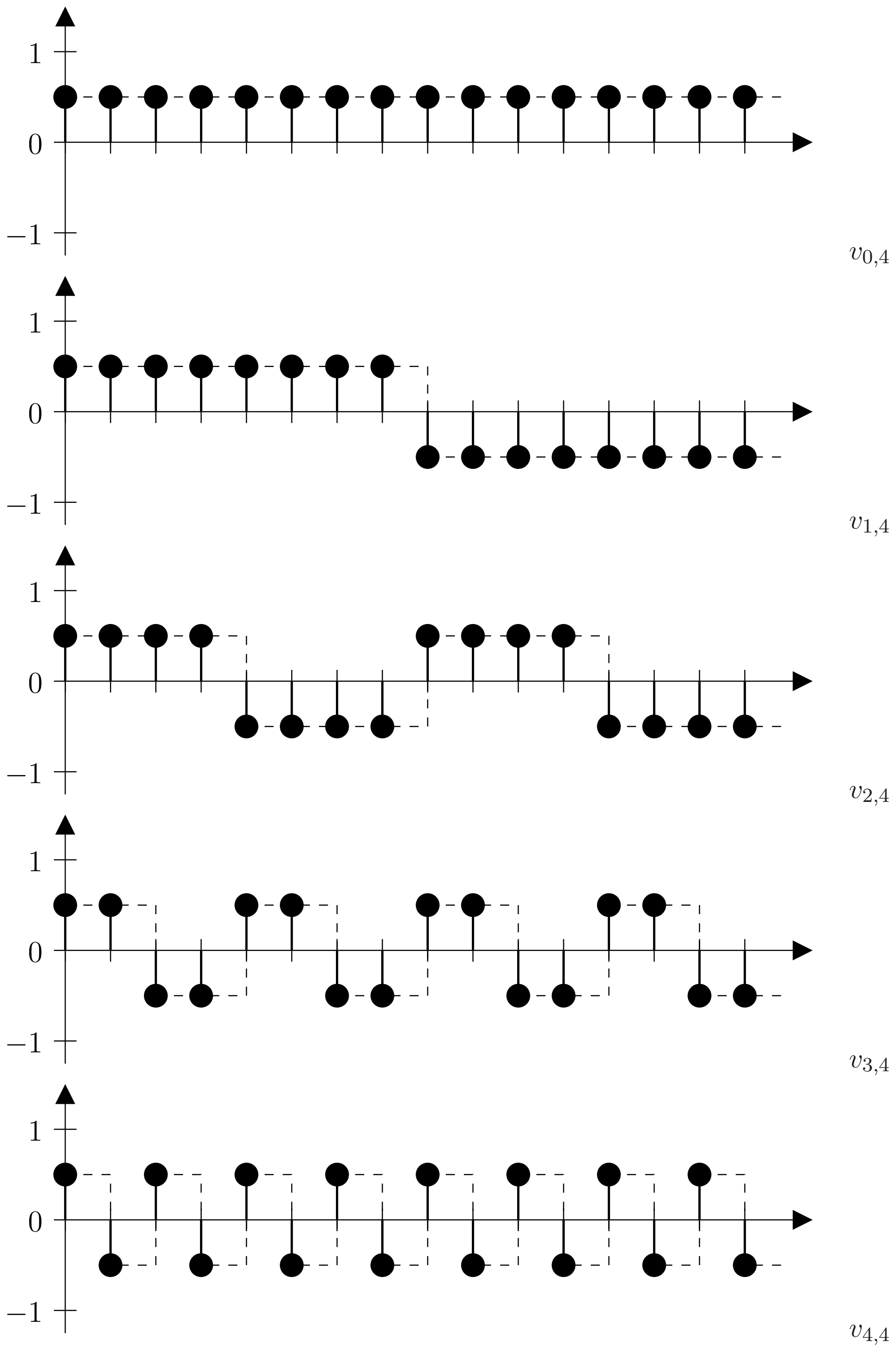


Figure 6.73: *Linearly independent system (here for $L = 4$, thus $N = 2^L = 16$).*

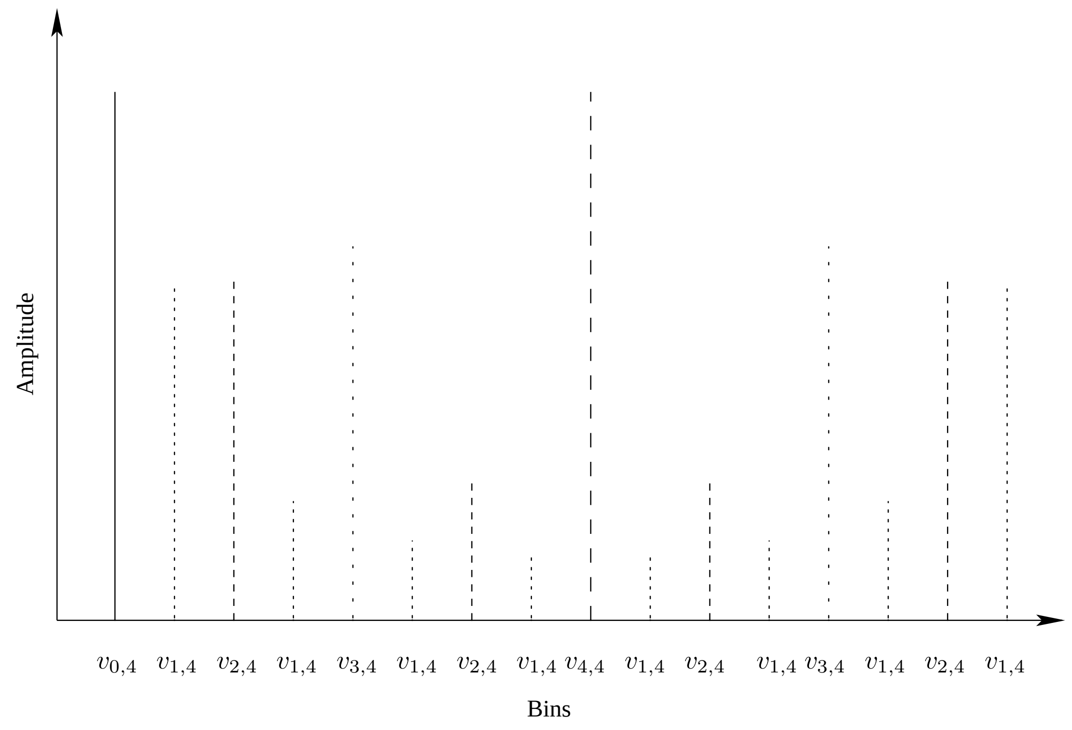


Figure 6.74: *Fourier spectra of the $v_{0,4}$, $v_{1,4}$, $v_{2,4}$, $v_{3,4}$, and $v_{4,4}$ vectors. This vectors are the new base vectors we created. They fill all the available bins. Each bin has been given the name of the vector that fills it, thus forming a linearly independent family. However, they are not forming a basis, since it is not possible to obtain all spectra from a combination of the spectra of the proposed vectors.*

their rhythms and intervals are identical, then they should be considered as identical. Moreover, the basis not being orthogonal, the a base function might show as multiple peaks in the resulting spectrum. This would impair the readability of the spectrum.

Based on the musical constraints we would then be able to perform an exhaustive basis research to find a suitable basis for the previously define musical transformations and pattern distances.

Our approach has been to try reuse the same tools (sinusoidal modelling) as in the preceding chapter. However, the musical (symbolic) domain being really different from the signal domain, we were not able to apply these tools here. This is still a open issue that we will surely investigate in the future, after having identified the various musical constraints.

Chapter 7

The Clicks Software

Clicks[1] is the name of a computer software I have developed in order to assist me during this Ph.D. work. It is implemented in Common Lisp and contains the implementation of most of the concepts presented in the previous chapters.

7.1 Constraints and general choices

The work done in this Ph.D. thesis mainly deals with signal processing, and partial tracking. The Clicks software was thus designed for this purpose, adding several features which makes it expendable and usable for a wider field of application. The ultimate goal would be to make Clicks a prototyping platform for signal and sound processing algorithms.

The first objective of this software was thus to be an experimenting platform for the concepts presented in this thesis. Most of the work in this thesis is based on the Fourier transform, hence one of the constraints is to have an efficient Fourier transform algorithm. It would also need some signal processing capabilities. In order for the program to be prototyping software, a command-line evaluation facility is necessary.

Mathwork's Matlab would have been a solution if it was at our disposal. However it was not, and the free alternatives did not seem sufficiently complete for my purpose.

Hence, I decided to code my own software, named Clicks, and I could add some more objectives to the task. First, with the generalisation of multicore processors, threading capabilities would be nice. Then, it would have to be a free and open-source software, more particularly it has to be released under the GNU General Public License.

Clicks should then be able to handle complex data structures without too much trouble. Hence a high-level language with object programing is necessary.

The choice was made to use Common Lisp as the language to implement our program. This language is inherently based on a read-eval-print loop, which makes prototyping a lot easier than with batch languages. The interested reader have a look at [Sei04, hyp] for more information on Common Lisp.

Indeed, in Common Lisp, in order to have a command-line interactor, one simple instruction is needed :

```
(eval (read))
```

This, by the way implies that any standalone Common Lisp program includes all the Common Lisp functions available to the user. This gives the possibility to expand the program (and the language) as needed by the user.

[1] Available at `http://www.labri.fr/perso/mraspaud/Clicks`

The interactive loop is thus a very good thing for prototyping. Indeed, the user can build his/her program by small increments, checking each time that the code is valid. This make development much easier than with batch languages. Moreover, it is easy in Common Lisp to add new types, classes, operators, functions and more to the language. Defining such things on the command-line makes them available for the rest of the session to the user, and makes it easy to program bottom-up, by trial and error, and thus correcting most of the bugs before actually implementing the core of any application.

Common Lisp also has the advantage of being a high-level language. This feature makes it easy for the programmer to be creative and productive by avoiding technicalities of low-level languages. It is easy to manipulate high-level concepts. In Clicks, for example, we very simply manipulate high-level objects such as peaks and partials.

This high level, combined with dynamic typing, makes the Common Lisp code easily readable and concise (the version of Clicks presented here is no more than 6000 lines of code). The end result is that well-written Common Lisp software is easily maintainable and expandable.

However, low-level programming languages such as C are usually chosen in order to produce fast and efficient software. Even if speed is not our main priority (since we are not looking for real time analysis at the moment), using Common Lisp does not reflect the fact that we do not care about speed and efficiency. On the contrary, a fast program would make it even more attractive and usable.

Indeed, it has been shown recently by D. Verna [Ver06a, Ver06b] that Common Lisp is usually at least as fast C (C being usually known as the fastest language). This comes from the fact that nowadays, Common Lisp compilers are very efficient, and that when needed, Common Lisp code can be statically typed in order to gain speed.

The primary objective of this software program being to perform order-2 sinusoidal analysis, we have then worked on the program to achieve this goal. However, in the design of the program, we try to take into account the wider range of applications it should be usable as, namely as signal/sound processing prototyping software.

This chapter is organised as follows. First, the organisation of the program directory is shortly presented, followed by quick instructions to get started. Then starts the listing of features. First the basic operations such as vector and matrix arithmetics are presented, then the signal processing tools (least-square method and linear prediction) are listed. Then the Fourier transform functions are shown, with an insight on the details of its implementations. After this, the window functions and resampling are presented, followed by the data structures and protocols regarding sound (in temporal, spectral and sinusoidal models). I finish this listing by presenting the functions for spectral-peak estimation, partial tracking and time-scaling.

Finally, I present the tools that allows multithreading to be used, the GUI, and the binary I/O tool I wrote.

7.2 Organisation of the program directory

The Clicks file tree is organised as follows:

- Root directory – Contains the necessary `clicks.asd` ASDF file and `package.lisp` which contains the information about the Clicks package.

- `maths` directory – Contains files for basic mathematical functions, mainly vector and matrix operations.
- `sp` directory – Contains files for signal processing functions, including the least-square method, lpc computation, FFT, resampling and analysis window computations.
- `io` directory – Contains files for input and output to sound files (wav, aiff), and the more general sound protocol.
- `gui` directory – Contains the GUI file.
- `lockable-buffer` directory – Contains the files for multithreading functions.
- `app/sinusoidal-model` directory – Contains the files for the order-1 and order-2 sinusoidal analysis.

7.3 How to install and launch Clicks

In order to install Clicks, get it on `http://www.labri.fr/perso/mraspaud/Clicks`, and decompress it. Clicks depends on iterate, sb-simple-audio and mcclim, so asdf-install them if you do not have them.

Then, within the Clicks directory, do

```
CL-USER> (load "clicks.asd")
T
CL-USER> (load-clicks-with-gui)
; loading system definition from
[...]
; compilation unit finished
;   printed 9 notes
#<PACKAGE "CLICKS">
CLICKS>
```

Clicks is loaded ! Have fun !

7.4 Basic tools

7.4.1 Vector operations

In order to handle signals, it is important to have functions to handle arrays (also called vectors in Common Lisp).

The first functions that are available are the arithmetic element-by-element operations. These operations can also handle single numbers.

- **v+** *(&rest args)* Adds elements of the vectors. For example:

  ```
  CLICKS> (v+ #(6 7 8 9) #(1 1 1 1))
  #(7 8 9 10)
  CLICKS> (v+ #(1 2 3 4) 5 #(1 1 1 1))
  #(7 8 9 10)
  ```

- **v-** *(arg1 &rest args)* Subtracts elements of the vectors.
- **v*** *(&rest args)* Multiplies elements of the vectors. This is element-by-element multiplication. Example:

  ```
  CLICKS> (v* #(1 2 3 4) 5 #(2 2 2 2))
  #(10 20 30 40)
  ```

- **v/** *(arg1 &rest args)* Divides elements of the vectors.

* * *

Less trivial operations are also provided:

- **convolution** *(s h)* Computes the convolution of two vectors.

  ```
  CLICKS> (convolution #(1 2 3 4) #(2 3 2 3))
  #(2 7 14 24 24 17 12)
  ```

- **scalar-product** *(a b)* Computes the scalar product (dot product) of two vectors.
- **sum** *(s)* Computes the sum of the elements of an array.
- **mean** *(s)* Computes the mean value of the elements of an array.
- **variance** *(s)* Computes the variance of an array.
- **diff** *(s &key (pad nil))* Computes the differences between adjacent elements of an array s. The key argument *:pad* indicates if the result should have the same number of elements as the input argument. It is nil by default.

  ```
  CLICKS> (diff #(1 2 3 4) :pad t)
  #(1 1 1 0)
  ```

- **v-abs** *(s)* Computes the abs value of the elements of an array.

  ```
  CLICKS> (v-abs #(1 2 -3 #c(0 4) #c(3 4)))
  #(1 2 3 4.0 5.0)
  ```

- **v-max** *(s)* Computes the maximum value of an array.
- **v-min** *(s)* Computes the minimum value of an array.

* * *

I also implemented more general array handling functions.

- **v-make** *(beg end &key (by 1) (element-type t))* Generates an array that contains increasing or decreasing numbers, specifying the starting and ending numbers, and as a key argument *:by* the increment (by default 1). It is also possible to specify the element type using *:element-type* (*t* by default).

  ```
  CLICKS> (v-make 1 5 :by 2)
  #(1 3 5)
  CLICKS> (v-make 5 1 :by -2)
  #(5 3 1)
  ```

- **v-fun** *(fun a)* Applies a function to an array.

  ```
  CLICKS> (v-fun #'sin (v-make 1 5 :by 2))
  #(0.84147096 0.14112 -0.9589243)
  ```

- **pad-array** *(array &key (before 0) (after 0) (pad-value 0))* Adds elements to an array, either before (using the key *:before*), after (using the key *:after*) or both (using both keys). The value of the elements to add is given by *:pad-value* (defaults to 0).

  ```
  CLICKS> (pad-array (v-make 1 5 :by 2) :before 1 :after 3 :pad-value 6)
  #(6 1 3 5 6 6 6)
  ```

- **swap-array** *(a)* Rotates an array by half its size.

  ```
  CLICKS> (swap-array #(1 2 3 4))
  #(3 4 1 2)
  ```

7.4.2 Matrix operations

Signal processing algorithms often include matrix operations. The matrices are defined as two-dimensional arrays.

I defined some basic matrix operations at first.

- **m+** *(&rest args)* Adds matrices and numbers.

- **m-** *(arg1 &rest args)* Subtracts matrices and numbers.

- **m*** *(&rest args)* Multiplies matrices and numbers.

  ```
  CLICKS> (m* #2A((1 0) (1 1))
              #2A((2 3) (4 5))
              2)
  #2A((4 6) (12 16))
  ```

- **m.*** *(&rest args)* Multiplies identically-sized matrices element by element, or matrices with numbers.

```
CLICKS> (m.* #2A((1 0) (1 1))
             #2A((2 3) (4 5))
             2)
#2A((4 0) (8 10))
```

- **transpose** *(M)* Computes the transposition of a matrix.
- **identity-matrix** *(n &optional m)* Generates an identity matrix of size its argument. If two arguments m and n are provided, this functions creates a $m - by - n$ matrix that has 1 on its diagonal and 0 elsewhere.

```
CLICKS> (identity-matrix 2 3)
#2A((1 0 0) (0 1 0))
```

- **identity-matrix-p** *(M)* Checks if a matrix has ones on its main diagonal and zero elsewhere.
- **symmetric-matrix-p** *(M)* Checks if the matrix is symmetric.
- **toeplitz** *(col &optional row)* Creates a toeplitz matrix. Accepts one or two arguments: the column vector, and optionally the row vector used to generate the matrix.
- **hankel** *(col &optional row)* Creates a hankel matrix. Accepts one or two arguments: the column vector, and optionally the row vector used to generate the matrix.
- **flip** *(M &optional (direction :horizontally))* Flips a matrix either *:horizontally* (default) or *:vertically.*
- **inverse-matrix** *(M)* Computes the inverse of a matrix.

7.4.3 Linear prediction

In order to enhance the partial tracking algorithm, we implemented the Burg method and some auxiliary functions.

- **arburg** *(s order)* Computes the linear prediction coefficients of a signal using the Burg method at a given order.
- **burg-extrapolate-1** *(s order)* Extrapolates a signal at a given order using the Burg method.
- **burg-reverse-extrapolate-1** *(s order)* Backward extrapolates a signal at a given order using the Burg method.
- **burg-extrapolate-n** *(s order n)* Extrapolates a signal at a given order using the Burg method n times.
- **burg-reverse-extrapolate-n** *(s order n)* Backward extrapolates a signal at a given order using the Burg method n times.

- **extrapolate-array** *(array &key (before 0) (after 0) (max-order 8))* Extrapolates the values of a array using the Burg method, giving *:before* sample before the array and *:after* after. The key arguments *:max-order* specifies the maximum order for the Burg method.

These linear prediction functions where used in section 3.3.2 on page 47.

7.4.4 Least-square estimation

For the PolySin model, we need a way to estimate the polynomial within a signal. One solution is the least-square estimation method.

- **least-squares-estimation** *(s &key (order 3))* Finds the coefficients of the closest polynomial of a given order by a least-square estimation. The coefficients of the higher order are given first.
- **polynomial-value** *(x coeffs &key (element-type t))* Computes the polynomials value(s) at position(s) x from the list of its coefficients, higher order coefficients first. x can be either a single number or an array of positions. For example computing the values of the polynomial $x^2 - x$ at position 1, 2, 3, and 4 is done in the following way:

  ```
  CLICKS> (polynomial-value #(1 2 3 4) '(1 -1 0))
  #(0 2 6 12)
  ```

- **n-length-polynomial** *(coeffs n &key (element-type t))* Computes the n first elements of a polynomial defined by *coeffs*.
- **closest-polynomial** *(s &key (order 3))* Makes a polynomial approximation of *s*.
- **remove-polynomial** *(s &key (order 3))* Removes a polynomial approximation of *s*. Returns both the residual signal and the coefficients of the approximating polynomial.

These are the functions we used to estimate polynomials for our order-2 model (section 4.1.3 on page 74) and for the resampling 3.5.1 on page 54.

7.4.5 Fast Fourier transform

Figures 7.75 on the next page and 7.76 on page 141 show the implementation (by Robert Strandh) of the Fourier transform in Common Lisp. This function is central in our software and to our work, meaning that it is called very often and consumes a lot of computing power, and thus a lot of time. It is then very important to make this function efficient. The code presented here shows the use of static type declaration through the use the `declare` function. Moreover, this function also can be used to set the optimisation level of the compiler. Here, the options used are `(speed 3)` and `(safety 0)`, which respectively mean that this part of the code has to be optimised for speed and that it has to avoid all type checks (and thus lower the safety of this part of the code). The use of `simple-arrays` and bit shifting for power-of-two multiplications, along with other programming tricks, makes

```
(defun fft-common (instance source dest)
  (let ((inter (slot-value instance 'inter))
        (coeffs (slot-value instance 'coeffs))
        (size (slot-value instance 'size)))
    (declare (type complex-sample-array
                   source dest inter coeffs))
    (declare (type fixnum size))
    (assert (= size (length source)))
    (assert (= size (length dest)))
    (labels ((aux (n starts steps startd)
               (declare (optimize (speed 3) (safety 0))
                        (type fixnum starts steps startd))
               (if (= n 2)
                   (let ((a (aref source starts))
                         (b (aref source (+ starts steps))))
                     (declare (type (complex double-float) a b))
                     (setf (aref dest startd) (+ a b)
                           (aref dest (1+ startd)) (- a b))
                     nil)
                   (let ((2*steps (ash steps 1))
                         (n/2 (ash n -1)))
                     (declare (type fixnum 2*steps n/2))
                     (rotatef dest inter)
                     (aux n/2 starts 2*steps startd)
                     (aux n/2
                          (+ starts steps)
                          2*steps
                          (+ startd n/2))
                     (rotatef dest inter)
                     (loop for i of-type fixnum
                           from (the fixnum (+ startd n/2))
                           for c of-type fixnum from 0 by steps
                           for dummy of-type fixnum from 0 below n/2
                           do (setf (aref inter i)
                                    (* (aref coeffs c)
                                       (aref inter i))))
                     (loop for i of-type fixnum from startd
                           for j of-type fixnum
                           from (the fixnum (+ startd n/2))
                           for dummy of-type fixnum from 0 below n/2
                           do (let ((a (aref inter i))
                                    (b (aref inter j)))
                                (declare (type (complex double-float)
                                               a b))
                                (setf (aref dest i) (+ a b)
                                      (aref dest j) (- a b))))
                     nil))))
      (aux size 0 1 0)
      dest)))
```

Figure 7.75: *The Common Lisp code of the Fourier transform.*

```
(defclass fft-instance ()
  ((size :initarg :size)
   (inter :initarg :inter)
   (coeffs :initarg :coeffs)))


(deftype real-sample () `double-float)

(deftype complex-sample () `(complex double-float))

(deftype real-sample-array ()  `(simple-array real-sample (*)))

(deftype complex-sample-array ()  `(simple-array complex-sample (*)))


(defun make-fourier-instance (size direction)
  ;; direction 1 direct, -1 inverse
  ;; check here that size is a power of two
  (let ((inter (make-array size
                           :element-type 'complex-sample))
        (coeffs (make-array (ash size -1)
                            :element-type 'complex-sample))
        (coeff #c(1d0 0d0))
        (factor (exp (/ (* direction -2 pi #c(0 1)) size))))
    (loop for i of-type fixnum from 0 below (ash size -1)
          do (setf (aref coeffs i) coeff)
          do (setf coeff (* coeff factor)))
    (make-instance 'fft-instance
                   :size size
                   :inter inter
                   :coeffs coeffs)))
```

Figure 7.76: *The Common Lisp code of the Fourier transform auxiliary functions.*

this function extremely efficient and makes it comparable in speed to a Fourier transform developed in C.

As a comparison, computing a thousand Fourier transforms of 1024 samples on a Intel(R) Celeron(R) 2.60GHz takes 0.308 seconds with our implementation, and 0.055 seconds using the processor optimised FFTW [FJ05].

Hence, even if the Lisp FFT is 6 times slower, it is concise, readable and above all portable.

The following functions available to the user.

- **sfft** *(source)* Standard fast Fourier transform. Takes an array of any type and size as input. If the size is not a power of two, the array is padded to the next power-of-two size.

  ```
  CLICKS> (sfft #(1 2 3 4))
  #(#C(10.0d0 0.0d0) #C(-2.0d0 2.0d0) #C(-2.0d0 0.0d0)
  #C(-1.9999999999999998d0 -2.0d0))
  ```

- **fft** *(source)* Fast Fourier transform. Takes a simple-array of complex double-float. The size of the array must be a power-of-two size.

- **fft!** *(source destination)* Fast Fourier transform, destructive version (places the result in *destination*).

- **fft-common** *(instance source destination)* This is the core function of the Fourier transform. It is the same as **fft!**, except the *instance* arguments that provides information necessary to the Fourier transform.

- **make-fourier-instance** *(size direction)* Creates an fourier-instance object, that performs precomputation for the Fourier transform given a *size* and a *direction* (1 for fft, -1 for ifft).

7.4.6 Resampling

The resampling of a signal is very useful for time-scaling applications. We have applied here our research on resampling (section 3.5.1 on page 54), that is the subtraction of a polynomial before the actual resampling of a signal.

- **sinc** *(x)* Computes the value of the cardinal sine function at position x.

- **reconstructor-value** *(x wings &key (window :hann))* Computes the value of the reconstructor at a given position, using the given window. The *wings* argument specifies the length of the reconstructor, by counting the number of wings (lobes) of the sinc function on each side of zero.

- **resample** *(s source-rate dest-rate &key (wings 10) (centering-fun #'no-centering) (offset 0))* Computes the resampling of a array *s*. The centring function (*centering-fun*) allows to decide which function is used to recenter the signal around zero. The offset allows to displace the signal.

```
CLICKS> (resample #(1 2 3 4) 2 5)
#(1 1.6137735853089887d0 1.9491546012888756d0 2.0258769361194124d0
  2.2604037565434085d0 3 3.959010672417362d0 4.2918622884833875d0
  3.392617837207366d0 1.60400781810801d0)
CLICKS> (resample #(1 2 3 4) 2 5 :centering-fun #'poly-centering)
#(1 1.4d0 1.8d0 2.2d0 2.6d0 3 3.4d0 3.8d0 4.2d0 4.6d0)
CLICKS> (resample #(1 2 3 4) 2 5 :centering-fun #'poly-centering
                                 :offset 1)
#(1.2d0 1.6d0 2 2.4d0 2.8d0 3.2d0 3.6d0 4 4.4d0 4.8d0)
```

The available centring functions are:

- **no-centering** *(seq x)* This function does not do anything.
- **mean-centering** *(seq x)* This function subtracts the mean of the sequence *seq*.
- **poly-centering** *(seq x &optional (order 3))* This function subtract a polynomial of order *order* from the sequence.

In order to create a new function to compute the centring function, the two only required arguments of the new functions must be the sequence to process and the position x where the function must be computed. The return values must be the modified sequence and the value that has been subtracted. Hence, we could define a new function that subtract 1 to values of the sequence if x is odd in this manner:

```
(defun odd-centering (seq x)
  (if (oddp x)
      (values (v- seq (make-array (length seq) :initial-element 1))
              1)
      (values seq 0)))
```

- **analog-value** *(s x &key (wings 10) (centering-fun #'no-centering))* This functions compute the analog value of a discrete signal s at the arbitrary position x, using a reconstructor having *wings* wings, after applying the *centering-fun*.

```
CLICKS> (analog-value #(1 2 3 4) 1/4 :centering-fun #'poly-centering)
1.25d0
```

7.4.7 Windows

We have defined a few window functions, since we did not need much more than the Hann window. Of course, the user can and is encouraged to implement more of them if necessary. The available windows are rectangular, Hann, Hamming, Blackman, and Barlett windows.

The simple functions to compute the windows and their values are the following:

- **make-window** *(type size)* Creates a window of given type and size. Available type are: *:rectangular*, *:hann*, *:hanning*, *:hamming*, *:blackman* , *:barlett*, and *:triangular*.

- **window-value** *(x type)* Returns the value of the window of the given type at position x. This value should be zero outside $[0, 1[$.

- **rectangular** *(N)* Creates the discrete rectangular window of size N.
- **hann** *(N)* Creates the discrete Hann window of size N.
- **hanning** *(N)* Creates the discrete Hann window of size N.
- **hamming** *(N)* Creates the discrete Hamming window of size N.
- **blackman** *(N)* Creates the discrete Blackman window of size N.
- **barlett** *(N)* Creates the discrete triangular (Barlett) window of size N.
- **triangular** *(N)* Creates the discrete triangular (Barlett) window of size N.

In order to have a single value of a window at a given x position, the functions ***window*-value** *(x)* are provided. Outside $[0,1[$, they return zero. ***window*** can be any of **rectangular**, **hann**, **hanning**, **hamming**, **blackman**, **barlett**, **triangular**.

Then, the functions that apply windows to given arrays are the following:

- **n-fill-window** *(in-array type)* Fills a given array destructively with a window of the given type.
- **n-apply-window** *(in-array type)* Multiplies a given array destructively by a window of the given type.
- **apply-window** *(in-array type)* Multiplies a given array by a window of the given type.
- **apply-window!** *(in-array out-array type)* Multiplies *in-array* by a given window type and destructively places the output in *out-array*.

The last four functions were optimised with memoization [Mic68] of the generated window. Indeed, in the case of sinusoidal analysis, it is quite common to perform the same transformation to the signal over and over again. Hence, not having to compute the shape a the window each type brings a significant improvement in performance.

Finally, the window-sum can be retrieved with the function **window-sum** *(type length)*.

7.5 Sound modelling

The core of this thesis is about sound modelling. Hence, particular attention has been given to this part of the implementation.

7.5.1 General sound interface

The general sound interface is based on the definition of a sound. In its most generic form, the useful parameters to a sound are the number of its channels and its sample rate. Thus we define the *sound* class as:

```
(defclass sound ()
  ((channels :initarg :channels :accessor channels)
   (sample-rate :initarg :sample-rate :accessor sample-rate)))
```

Then, the methods that should be available for all sounds are:

- **load-from-file** *(snd file)* Opens a file containing sound data and fills the given sound instance.
- **save-to-file** *(snd file)* Save a sound instance to a file.
- **play** *(snd)* Plays a sound.
- **record** *(snd)* Records a sound.

The last two functions would imply real-time analysis and synthesis, especially for higher order sound models, which, at the moment, is not yet available in this implementation. However, we have good hope that it will be developed in the future.

A dispatcher function has also been created in order to load sound files with specific extensions to the right sound model.
load-sound-file *(file)*. This function is thus more advisable than the former.

7.5.2 PCM sounds

The *pcm-sound* class implements the first three methods described in the previous subsection.

In order to be output as a sound to the speakers of the computer, the *sb-simple-audio* Common Lisp library has been used.

7.5.3 Spectral peaks

A spectral peak is the manifestation of a sinusoid in the spectrum of a signal. It is defined by three parameters at least: its amplitude, its frequency, and its phase. I also added a fourth parameter that we will be useful in the future of this software program: its position (azimuthal for now).

Hence, a spectral peak is defined as an array of four double-precision floating number.

```
(deftype peak () '(simple-array double-float (4)))
```

The creation of a peak is done using the dedicated function:
make-peak *(&key (frequency 0.0d0) (amplitude 0.0d0) (phase 0.0d0) (position 0.0d0))*
The accessor functions are the following:

- **peak-frequency** *(peak)*
- **peak-amplitude** *(peak)*
- **peak-phase** *(peak)*
- **peak-position** *(peak)*

for retrieving the values, and

- **(setf peak-frequency)** *(value peak)*
- **(setf peak-amplitude)** *(value peak)*

- **(setf peak-phase)** *(value peak)*
- **(setf peak-position)** *(value peak)*

for setting those values. All these accessor functions have been inlined for optimised performance.

7.5.4 Spectral sounds

What we call a spectral sound is a collection of peaks defining a sound. This spectral sound is the first step towards partial tracking.

The spectral sound can be created with the following function:
create-peak-sound *(&key (channels 1) (sample-rate +default-sample-rate+))*

The spectral sounds instance holds information about the sample rate, the number of channels it has, the peak information and the envelope information in case of non zero-centred sounds.

The peaks contained in such a sound are gathered in *frames*, since they are found at the same time index.

Hence in order to deal with the peaks and envelope information, the following functions were created:

- **with-peak-sound** *((peak-output envelope-output peak-sound) &body body)* This macro opens the peak sound for writing. The two following functions can then be used to write data.
- **store-peaks** *(peak-output peaks)* This function writes a peak frame into *peak-output*, as defined in **with-peak-sound**.
- **store-envelope** *(peak-output peaks)* This function writes envelope coefficients into *envelope-output*, as defined in **with-peak-sound**.
- **with-elements-from-peak-sound** *(p-sound peak-iterator &body body)* This macro allows iteration over the peak frames contained in the peak sound. It is based on the **iterate** macro, and thus all the **iterate** keywords and constructions are allowed.
- **with-elements-from-peak-sound-envelope** *(p-sound peak-iterator &body body)* This macro allows iteration over the envelope coefficients contained in the peak sound. It is based on the **iterate** macro, and thus all the **iterate** keywords and constructions are allowed.
- **envelope** *((p-sound peak-sound) &key (sample-rate +default-sample-rate+) (f-cut 2))* This method reconstructs the envelope of the peak sound at a given sample rate and cutting the higher frequencies of the envelope at *f-cut*.

The implementation of the spectral sound is based on files. Indeed, in case of large data, it is best not to overload the memory, since heavy calculations might take place. Moreover, files make it easy to reuse computed data for different purposes.

7.5.5 Spectral peak estimation

The spectral peak estimation is performed using the Marchand - Desainte-Catherine method [DCM00], explained in section 3.2 on page 39.

- **short-time-spectral-analysis** *((snd pcm-sound) &key (hop-size 512) (frame-length 2048) (window-type :hann) (with-envelope nil))* This method computes the peak frames of a given pcm sound and returns the corresponding peak sound.
- **peak-extraction-without-envelope** *(data index &key (frame-length 2048) (window-type :hann) (sample-rate 44100))* Performs a spectral peaks extraction over a piece of signal.
- **peak-extraction-with-envelope** *(data index &key (frame-length 2048) (window-type :hann) (sample-rate 44100) (iterations 10) (order 3))* Performs a spectral peaks extraction over a piece of signal, computing the envelope of that piece of signal. The process can be iterated for better results.
- **mdc-compute-peak-frame** *((spectrum diff-spectrum &key (sample-rate 44100) (frame-length 2048) (window-type :hann))* Computes the peak frame from a spectrum and its derivative using the Marchand - Desainte-Catherine method.

In order to avoid massive memory allocations, and thus enhance the performance of the peak estimation, the implementation of the peak-extraction functions always use the same arrays in a destructive manner.

7.5.6 Partials

In the sinusoidal model, sounds are represented by a collection of partials. Each partial is itself a collection of peaks that fit well together. The partials are also ordered in time. Hence, the *partial* class is defined as:

```
(defclass partial ()
  ((sample-rate :initform nil
                :initarg :sample-rate
                :accessor sample-rate)
   (peak-list :initform nil
              :initarg :peak-list
              :accessor peak-list)
   (date-of-birth :initform 0
                  :initarg :date-of-birth
                  :accessor date-of-birth)
   (date-of-death :initform 0
                  :initarg :date-of-death
                  :accessor date-of-death)))
```

The fact that the peaks are stored as a list can be a problem for very long partials. However, it is the most convenient, since operation of removal and adding of peaks is quite common in partial tracking.

While tracking the partials, it also important to gather some peaks called "ghost peaks" since they are not present in the peak frames, but that seem to be plausible. Hence another class called *track* is defined as:

```
(defclass track (partial)
  ((state :initarg :state
          :initform :alive
          :accessor state)
   (ghost-peaks :initarg :ghost-peaks
                :initform 0
                :accessor ghost-peaks)))
```

The state of the track can be either *:alive*, *:zombie*, *:embalmed*, or *:rotten*. This means respectively that a track has found proper peaks to gather, that the track did not find a proper peak recently, but is still allowed to look for other possibilities, that the track is not gathering anymore peaks, but that it is worth keeping, and that the track is not gathering anymore peaks, and that it is not worth keeping.

The *ghost-peaks* slot is the number of extrapolated peaks at the end of the track. If this number increases too much, the track is then not allowed to continue the peak hunt and is put to the states *:embalmed* if it is worth keeping, *:rotten* if not.

Then, in order to be able to save the partials to a file, a macro character had to be defined (#T), along with the corresponding print-object method.

The following functions are then provided to handle the partials.

- **create-track** *(&key (sample-rate +default-sample-rate+) (date-of-birth 0))* Creates a track.
- **close-track** *(track)* Performs all the operations to turn the track into a partial.
- **deadp** *(track)* Checks if a track is dead (rotten or embalmed).
- **track-length** *(partial)* Returns the number of peaks in the partial.
- **nth-peak** *(n track &key (from-end nil))*
- **newest-peak** *(track)* Returns the most recent peak from the track.
- **strip-peaks** *(track n &key (key #'identity) (from-end nil))* Applies the function *key* to the *n* first (or last if *from-end* is *t*) peaks of *track*.
- **append-peak** *(peak track &key (ghost nil))* Append a peak to a track. If *ghost* is *t*, set is as a ghost peak.
- **validate-ghost-peaks** *(track)* Sets all the ghost peaks as real.
- **remove-last-peak** *(track)* Removes the newest peak from the track.

The partials also have to be synthesised.

- **synthesize-partial** *(partial destination-sample-rate)* Synthesises the partial at the given sample rate.

- **mix-partial-into-array** *(partial samples &key (destination-sample-rate +default-sample-rate+))* Synthesises a partial and adds it to a signal (represented as an array).
- **mix-partial-into-pcm** *(partial pcm-instance)* Synthesises a partial and adds it to an instance of a pcm sound.

7.5.7 Sinusoidal sounds

A sinusoidal sound is actually a collection of partials. The corresponding class is *sin-sound*. We have implemented some functions to handle this collection.

- **create-sin-sound** *(&key (channels 1) (sample-rate +default-sample-rate+))* Creates a sinusoidal sound.
- **store-partial** *(partial sin-sound)* Adds a partial to *sin-sound*.
- **add-track** *(track sin-sound)* Converts a track to a partials and add it to *sin-sound*.
- **with-elements-from-sin-sound** *((s-sound partial-iterator) &body body)* This macro allows iteration over the partials contained in the sinusoidal sound. It is based on the **iterate** macro, and thus all the **iterate** keywords and constructions are allowed.
- **nth-partial** *(n sin-sound)* Returns the nth partial of *sin-sound*.
- **envelope** *(sound &key (sample-rate +default-sample-rate+))* Returns the envelope of a sinusoidal sound, if it has one.
- **(setf envelope)** *(envelope sin-sound)* Sets the envelope of a sinusoidal sound.

The synthesis of the sinusoidal sound is also implemented.

- **synthesize** *((sin-sound sin-sound) &optional (destination-sample-rate +default-sample-rate+))* Returns an array of the pcm signal after the synthesis and the mixing of all the partials.
- **synthesize-to-pcm** *(sound &optional (destination-sample-rate +default-sample-rate+)* Synthesises the sinusoidal sound and puts it into an instance of a pcm sound.

7.5.8 Partial tracking

Using all the functions presented earlier, it is quite easy the perform the tracking of partials. Hence, only a few functions had to be implemented for this task. The partial tracking principles are explained in section 3.1 on page 37.

- **estimate-next-peak** *(track)* Estimates the next peak of the track, using the Burg method if there are enough peaks in the past of the track, or the mirror method otherwise.
- **find-next-peak-for** *(track peak-frame)* Tries to match a peak from the peak frame to the track.

- **track-partial** *(sound &key hop-size frame-length window-type)* Tracks partials from *sound* which can be either a peak sound or a pcm sound. In the latter case, a peak-extraction algorithm is first applied to generate a peak sound.

7.5.9 Time-scaling of sounds

The time-scaling techniques presented in this thesis are of course implemented. They are explained in sections 3.5 on page 54 and 5.2 on page 108.

The following functions are available to the user.

- **stretch** *(pcm-sound factor order)* Dispatch function to let the user choose easily the type of scaling to perform.
- **stretch-order-2** *(sin-sound factor)* Performs time-scaling at order 2 of the sound, that is at order 1 of the partials.
- **stretch-order-1** *(sin-sound factor)* Performs time-scaling at order 1 of the sound, that is resampling the partial trajectories.
- **stretch-order-0** *(pcm-sound factor)* Performs time-scaling by resampling the pcm signal.
- **stretch-partial-order-1** *(partial factor)* Performs time-scaling on the partials by resampling the partials of partials (order-2 partials) and the envelope of the partials.
- **stretch-partial-order-0** *(partial factor)* Performs time-scaling on the partials by resampling them.
- **track-partials** *((p partial) &key (hop-size 1) (frame-length 64) (window-type :hann) (iterations 10))* Taking a partial as parameter, this method performs order-2 partial-tracking on a partials. Hence, it computes partials of partials.
- **short-time-spectral-analysis** *((p partial) &key (window-type :hann) (frame-length 64) (hop-size 1) (with-envelope t) (iterations 10))* Finds spectral peaks in partial trajectories.

7.6 Design and other features

In order to produce software that implements a sound analysis/transformation/synthesis chain we need to have an adapted chain design. For example, a McAulay and Quatieri analysis chain would be composed of three elements: a sound reading element, then a spectral peak analysis element, and finally a partial tracking element. This partial tracking element needs the output data of the peak analysis element, which in turn needs the output data of the sound reading element. This chain of elements is depicted in Figure 7.77 on the facing page.

Hence, I have decided to implement multithreading support, so that these elements of the chain can be run in parallel when possible and the buffer protocol which ensure the data flow between elements of the chain. The latter is a generalised implementation of the writer/reader paradigm, such that the functions corresponding to a given flow can be created at runtime if needed.

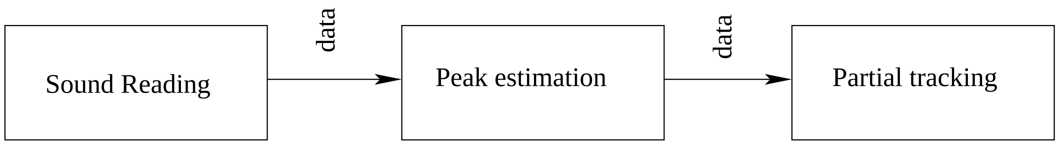


Figure 7.77: *The chaining of tasks for the McAulay and Quatieri analysis. The two last elements of the chain need as input data the output data of the element preceding them.*

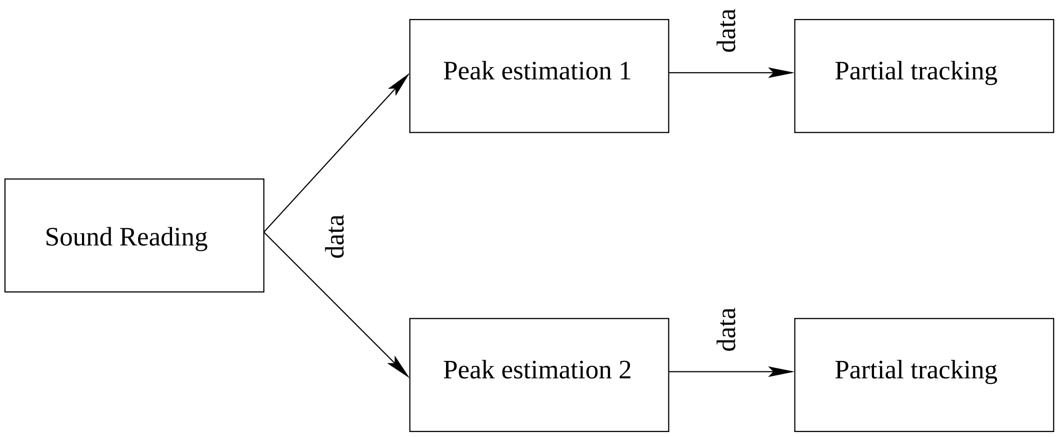


Figure 7.78: *The chaining of tasks for the McAulay and Quatieri analysis in the case of multiple algorithms for the peak estimation step. The output of the sound-reading task is used by both peak analysis tasks, and the algorithms on each branch can be performed independently. Hence, the interest of multithreading.*

7.6.1 Multithreading support

The chain architecture, in more complicated cases, would thus greatly benefit from a multithreaded architecture.

One of the constraints defined here is the chained structure of the software. Indeed, peak estimation cannot be performed before that the samples of the sound are read, for example. Hence we could define several tasks that will be executed one after the other. However this approach would not take any advantage of a possible parallel architecture (like hyperthreaded processors or smp architectures). In order to take advantage of this, the solution we chose is to use threads.

Each task will thus be launched in a thread. Indeed, a task needs only a part of the data of the preceding task (in the chain), and thus, instead of waiting for a task until it is terminated, it is possible to start only slightly later. This way, the different tasks are executed in parallel.

The user might also want to be able to change the algorithms used for a given task easily, or compare two or more methods for a given part of the McAulay and Quatieri analysis method. Hence the simple chain presented earlier might become a little bit more complicated (see Figure 7.78 on the previous page). It is clear from these configurations that our software needs a modular architecture. That is, we need a set of objects that will process data and a way to connect them to each other in order to have the flow of data going through them. One could then imagine the prototype of three functions needed to build such chains. For example, here is what the building of the simple McAulay and Quatieri analysis chain should look like:

```
(defprocess sound-reader #'sound-reading-function "saxo.wav")
(defprocess peak-analysis #'peak-analysis-function)
(defprocess partial-tracking #'partial-tracking-function)

(connect sound-reader peak-analysis)
(connect peak-analysis partial-tracking)

(run sound-reader peak-analysis partial-tracking)
```

In this example, we can distinguish three parts. The first part, which consists of three calls to `defprocess` creates the different elements of the chain (herein called *tasks*) with the associated function and optionally the parameters needed by the functions (for example, `sound-reader-function` needs a sound file to read).

The second part of the above example consists of two calls to `connect`. This function "connects" the output of the first task to the input of the second task, so that the tasks share the data.

The third part starts the different task threads (call to `run`).

The connection of the different threads needs to be explained a bit further. This connection consists of the linking of the output of a first thread to the input of a second thread. Thus we have to decide how the data will be transmitted from one thread to another. The two options are either to pass the data as a parameter to the functions or to save the data on the secondary storage. The first option has the advantage of being fast (no need to store the possibly large data on a physical support), but the data is non persistent. That is, once the threads are terminated, the data is lost. On the other hand storing the data on the secondary storage can be slow (though this problem will tend to disappear with the wider use of persistent primary storage such as M-RAM drives) but the advantage is that the data is persistent. Hence, once some data is stored, other threads can use it anytime, would it be after the death of the producing thread or during another session of the software. Moreover, the data can then be easily examined manually (if it is stored in a readable way).

This is why the buffer protocol has been created.

Buffer protocol

In order to easily add threads to Clicks, the buffer protocol has been implemented and made available to the user.

Figures 7.79 on the following page and 7.80 on page 155 show the implementation of the data buffer used for the inter-process communication. The data buffer is in fact written to disk. The access to the file is strictly controlled by the use of a mutex. This means that the reading and writing processes cannot write and read the file at the same time. This ensures that pieces of data are entirely written before anyone can read them, and hence avoid bugs involving incomplete data.

With this protocol, the reader-writer program can be simply written as follows:

```
(defun writer (buffer)
  (as-buffer-writer buffer
   ...
   (buffer-push buffer elt)))

(defun reader (buffer)
  (with-elements-from-buffer buffer elt
   ...
   (do-something-with elt)))
```

7.6.2 Plotting and interacting

The plotting interface

In order to be able to visualise the data, we have implemented a minimal plotter. This plotter was called “The Clickographer” and was implemented using McClim [mcc].

The first possibility offered by the Clickographer is the plotting of simple arrays. It is possible to draw in different colours, and more than one curve at once.

For example, the following command had to be input to generate Figure 7.81 on page 156:

```
CLICKS> (plot #(1 2 1 2 1 2 1 2) (list #(2 2.5 2 2.5 2 2.5 2 2.5) clim::+red+))
```

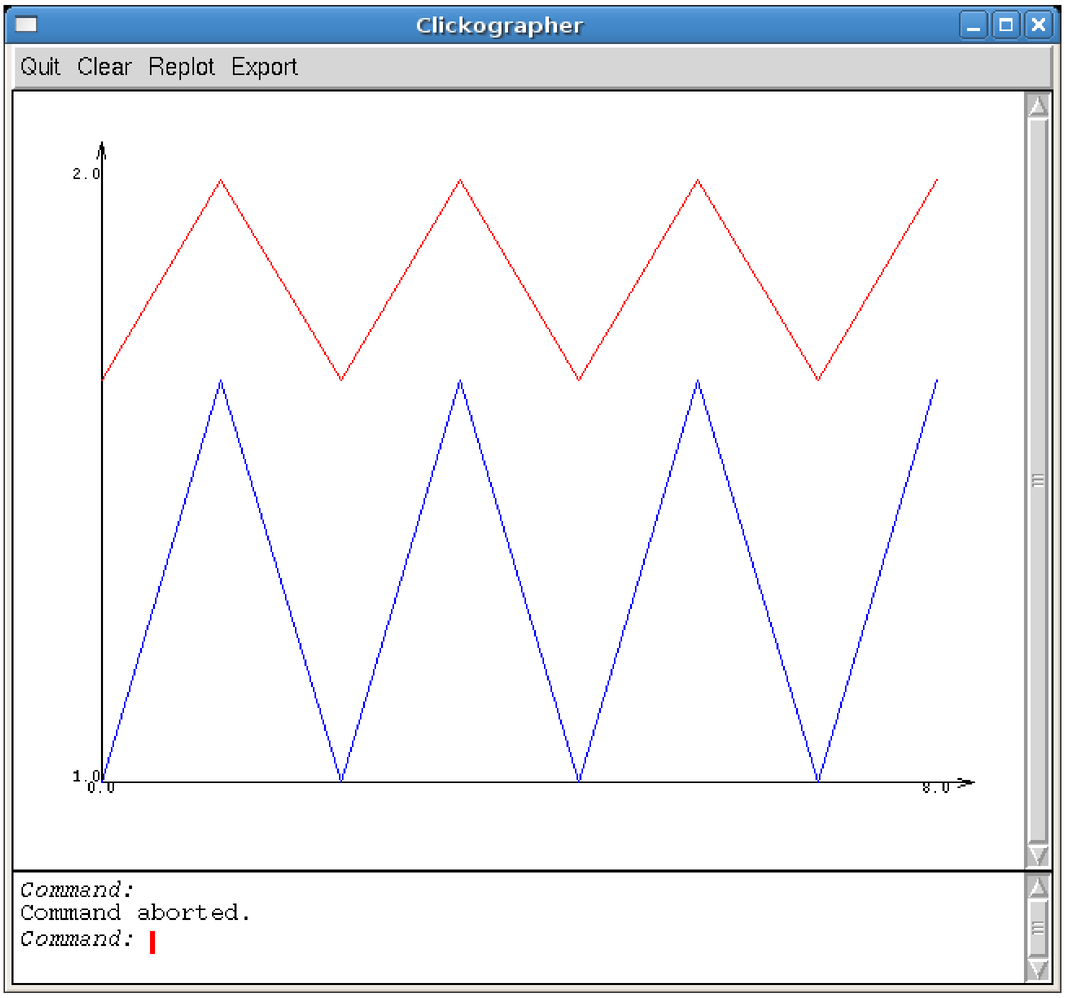
It is also possible to draw sinusoidal sounds, and interact with it. For now the interaction is quite simple, since it is possible only to select a partial and plot it alone. However, we plan to add the possibility to add more features such as opening a new window displaying information about the curve selected or plot the partial on this new window.

```
(defclass buffer ()
  ((buffer :initarg buffer :initform nil :accessor buffer)
   (buffer-name :initform (format nil "buffer-~a" (incf buffer-count))
                :reader buffer-name)
   (buffer-stream :initarg buffer-stream :initform nil
                  :accessor buffer-stream)
   (buffer-queue :initarg buffer-queue :initform (make-waitqueue)
                 :accessor buffer-queue)
   (buffer-lock :initarg buffer-lock
                :initform (make-mutex :name "buffer lock")
                :accessor buffer-lock)
   (buffer-writer-thread :initarg buffer-writer-thread :initform nil
                         :accessor buffer-writer-thread)))

(defun buffer-writer-start (buffer)
  (setf (buffer-writer-thread buffer) sb-thread:*current-thread*))

(defun buffer-writer-alive-p (buffer)
    (and (buffer-writer-thread buffer)
         (sb-thread:thread-alive-p (buffer-writer-thread buffer))))

(defun buffer-push (object buffer)
  (unless (buffer-writer-thread buffer) (buffer-writer-start buffer))
  (with-mutex ((buffer-lock buffer))
    (format (buffer-stream buffer) "~s~%" object)
    (force-output (buffer-stream buffer))
    (condition-broadcast (buffer-queue buffer))))

(defun end-writing-in-buffer (buffer)
  (setf (buffer-writer-thread buffer) nil)
  (condition-notify (buffer-queue buffer)))
```

Figure 7.79: *The buffer protocol, part one. The buffer implementation uses files and is synchronised using a mutex.*

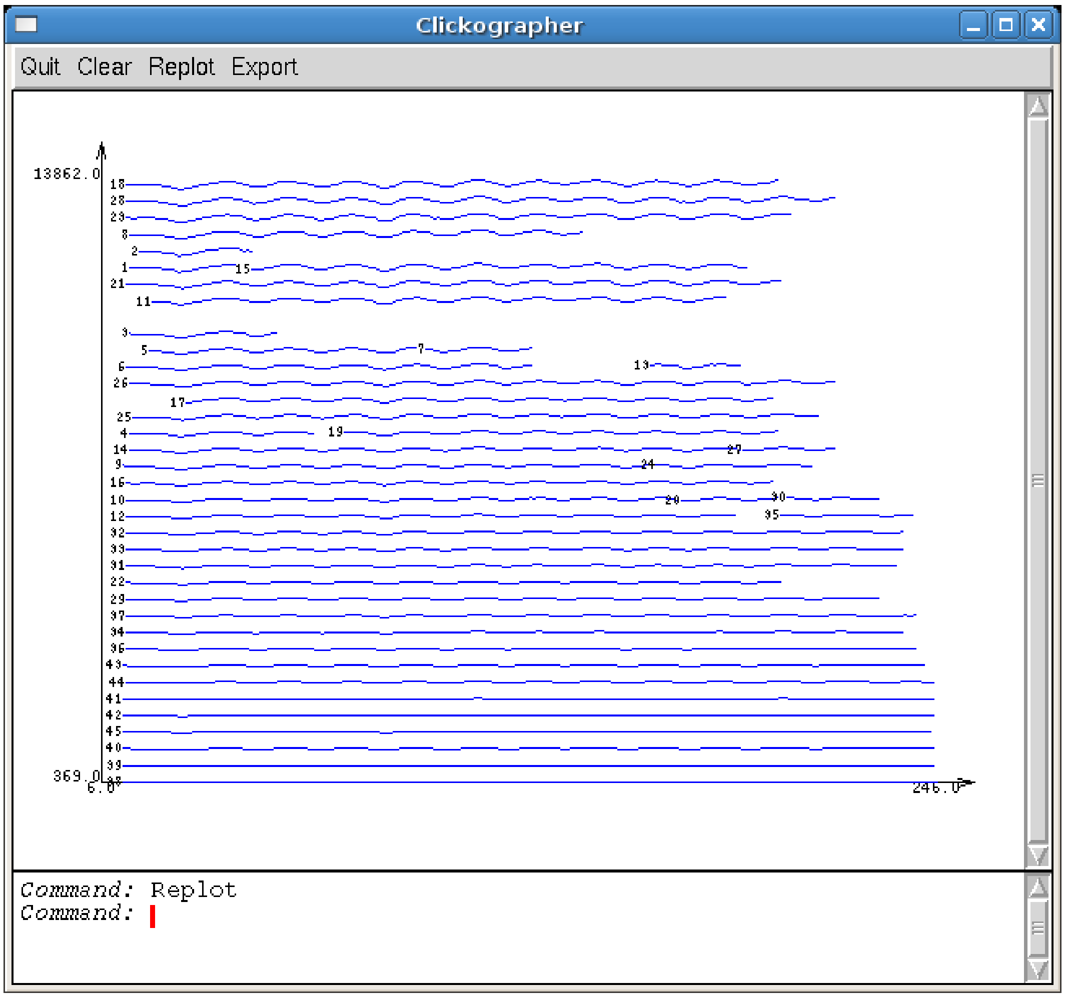
```
(defmacro as-buffer-writer (buffer &body body)
  (let ((s (gensym)))
    `(progn
       (buffer-writer-start ,buffer)
       (let ((,s (buffer-stream ,buffer)))
         (with-open-file
             (,s (buffer-name ,buffer) :direction :output
                 :if-exists :supersede)
           ,@body))
       (end-writing-in-buffer ,buffer))))

(defmacro with-elements-from-buffer (buffer single-element &body body)
  `(let ((,single-element nil))
     (with-open-file
         ((buffer-stream ,buffer) (buffer-name ,buffer) :direction :input
          :if-does-not-exist :create)
       (iterate
        (while
            (or (buffer-writer-alive-p ,buffer)
                (not (eq (peek-char nil (buffer-stream ,buffer)
                                    nil 'eof nil)
                         'eof))))
        (with-mutex ((buffer-lock ,buffer) :wait-p t)
          (iterate (while (setf ,single-element
                                (read (buffer-stream ,buffer)
                                      nil nil)))
                   (release-mutex (buffer-lock ,buffer))
                   ,@body
                   (get-mutex (buffer-lock ,buffer) :wait-p t))
          (when (buffer-writer-alive-p ,buffer)
            (condition-wait (buffer-queue ,buffer)
                            (buffer-lock ,buffer)))))))
```

Figure 7.80: *The buffer protocol, part two. The buffer implementation uses files and is synchronised using a mutex.*



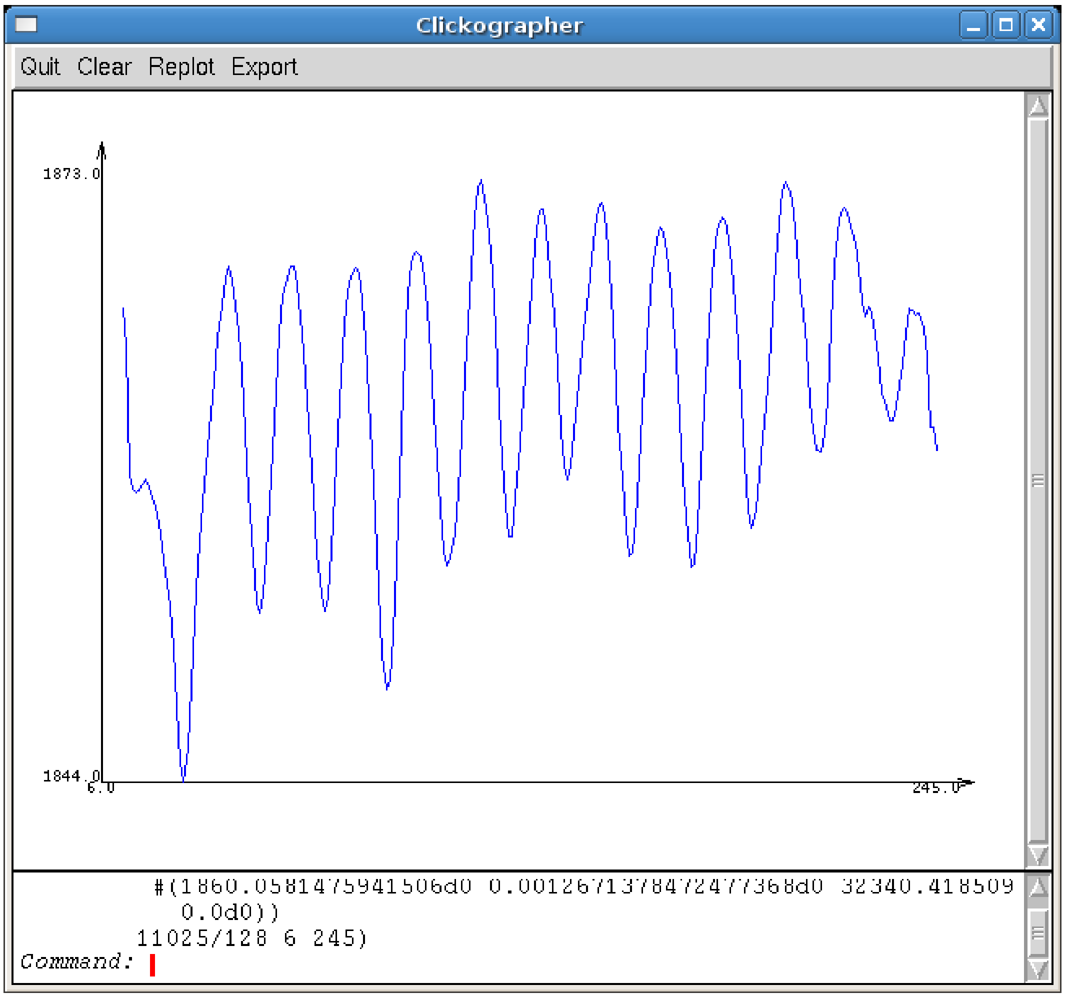
```
Command:
Command aborted.
Command:
```

Figure 7.81: *The Clickographer can plot simple arrays, and can plot in different colours.*



```
Command: Replot
Command:
```

Figure 7.82: *Plotting a sinusoidal sound. The partials are numbered to ease their retrieval from the sinusoidal sound by the user.*



```
               #(1860.0581475941506d0 0.0012671378472477368d0 32340.418509
                 0.0d0))
             11025/128 6 245)
Command:
```

Figure 7.83: *A single partial after clicking on the same partial in the sinusoidal sound representation shown in Figure 7.82 on the preceding page*

Figure 7.82 on page 157 was produced using:

```
CLICKS> (plot (track-partials (load-sound-file "saxo.wav")))
```

This feedback about what is drawn on screen is made easy thanks to the wonderful *presentation* feature of Clim.

However, one of the most interesting features of the Clickographer is that it is available from the Lisp REPL. Indeed, as the user works with Clicks on the REPL, it was a necessary condition that the plotting could be done directly from there.

Hence, it is possible to plot something using the **plot** *(thing &rest other-things)* function.

thing can either be an array, a partial, a sinusoidal sound or a list. If it is a list, it is either a list formed by the object to plot and its colour, or the object to plot, its colour, and its starting position.

Finally, the Clickographer can export figures in encapsulated postscript (eps) format, easily included in LaTeX documents for example.

This plotter is still in its very early stages, since it was just a debugging tool at first. However, it is now planned that we develop more features (such as zooming, multiple y-axis plotting, adding labels) such that it will be a full feature plotter usable to generate professional graphics.

Binary I/O

The binary input and output is necessary in order write and read binary files and thus to exchange data with other programs for example. Hence, I developed a library to do such input and output.

A simple example of how to write and read binary files is the following:

```
CL-USER> (with-open-file (s "test.bin"
                         :direction :output
                         :element-type '(unsigned-byte 8)
                         :if-exists :supersede)
           (define-endianess :little-endian)
           (define-output-stream s)
           (funcall (get-IEEE-float-writer 'double-float) 56.3332563322d0))
NIL
CL-USER> (with-open-file (s "test.bin"
                         :direction :input
                         :element-type '(unsigned-byte 8)
                         :if-exists :supersede)
           (define-endianess :little-endian)
           (define-input-stream s)
           (funcall (get-IEEE-float-reader 'double-float)))
56.3332563322d0
```

The function **define-endianess** *(endianess)* accepts either *:little-endian* or *:big-endian*. The **get-IEEE-float-writer** *(type)* and **get-IEEE-float-reader** *(type)* look up for the appropriate functions to write and read IEEE single floats in binary format.

It is also possible to write and read signed and unsigned integers using **write-unsigned-integer** *(number nb-bytes)*, **write-signed-integer** *(number nb-bytes)*, **read-unsigned-integer** *(nb-bytes)*, and **read-signed-integer** *(nb-bytes)*.

Finally, it is possible to generate readers and writers for any kind of floating number, given the size of the sign, exponent, mantissa and the bias, and associate it to a type.

For example, in order to generate the previously presented readers and writers, we called:

```
(define-IEEE-float IEEE-double-float
                   :sign-bits 1
                   :exponent-bits 11
                   :mantissa-bits 52
                   :bias 1023
                   :associated-type 'double-float)
```

7.7 Conclusion

I have developed the Clicks software program over the past three years as a work platform for this thesis, and thus it is only logical that it implements most of the work presented in this thesis. However, I developed it keeping in mind the fact that it should be usable by others for other purposes, and hence I tried to design an easily maintainable and expandable piece of software.

This implementation has been a lonely work, except for the Fourier transform which was implemented by Robert Strandh and Sylvain Marchand.

It is with great enthusiasm that the implementation of Clicks will continue, adding more features in order to make it a nice prototyping software program for audio signal processing.

Conclusion

During the three years of this PhD thesis, we approached many concepts and examined many ideas, a lot of them not being either feasible or correct. However, we have surely advanced on the way to the futuristic dream of controlling the way music is being played.

We first presented the different sound models.

The temporal model being a simple capture of the pressure of the air produced by a sound is thus very low-level. Hence, it not possible to easily produce sophisticated effects in this model. However, we have presented the OLA method and some derivatives that can give satisfying results for time-scaling if the scaling factor remains close to 1.

The spectral model we presented afterwards opens new possibilities regarding sounds effects. The vocoder allows for example time-scaling without modification of the pitch.

At a even higher level, we presented the sinusoidal model. This model was designed by McAulay & Quatieri and gives the possibility to work on sounds as a set of partials, that is tuples of amplitude, frequency, and phase values of the sinusoids present in the sound over time. Together with Mathieu Lagrange and Sylvain Marchand, we sought enhancements to the sinusoidal analysis algorithm in order to facilitate further processing of the sinusoidal data. Progress was made in this matter, with for example the use of linear prediction for partial tracking.

Time-scaling in the sinusoidal model is also more robust than in the other models we presented earlier. We also presented a new method for performing this time-scaling in the sinusoidal model, based on the resampling of the sinusoidal parameters, first by resampling the frequency, then by resampling the phase. These two time-scaling methods, even though preserving the timbre of the sound, do not keep the modulations of the control parameters, such as the tremolo and the vibrato.

Hence, our idea was that this sinusoidal model could be a basis for the more evolved models we need to perform complex time-scaling.

Thus we could go on to the next step that was the modelling of the sinusoidal parameters, considered as control signals of the sound.

This concept of modelling parameters of the partials (of the first model) is called order-2 modelling. The first and widely accepted model is a model based on polynomials since McAulay and Quatieri modelled the phase trajectories of the partials as 3rd-degree polynomials. This model however is not really correct as soon as more than the human perception comes into account.

We then found a new model based on sums of sinusoids. The control parameters such as amplitude and frequency of the sinusoids were then well modelled using sums of sinusoids, and thus we considered that the control parameters contained periodic micro-modulations. This assumption however was not perfectly valid, mainly for the phase parameter, and in the case where the frequency was not constant, since the modulations were not centred around a fixed position.

Thus, together with Laurent Girin and Sylvain Marchand, we designed a new model, based on both the sums of sinusoids and on the polynomials. This new model seems to be the most accurate. However, using the Fourier analysis and a least-square polynomial analysis we do not obtain very precise estimations of the parameters of the control signals. Even using High-Resolution methods does not give satisfying results yet. The model was then used for partial prediction, among other things.

An application to partial classification was developed with Mathieu Lagrange, and gives interesting results regarding the gathering of partials produced by the same note.

We also discussed the possibility of enhancing the estimators for our new models by using the strong link that seems to be present between tremolo and vibrato.

Thus, having this order-2 model of the partials, it is possible to perform partial-tracking on the order-2 parameters.

Keeping the modulations of the sound is then possible, since a first solution was presented by Arfib and Delprat. This solution however made simplifying assumptions on the nature of the modulations. Thus we researched a new way to describe these modulations.

In order to fulfil we designed the so-called *hierarchical modelling*, which consists in the reanalysis of the results of a first sinusoidal analysis, first based on the frequency and amplitude trajectories of the partials of the sound, then based on the phase and amplitude trajectories. We then performed enhanced time-scaling, that is time-scaling without alteration of neither the pitch nor the vibrato and tremolo.

Then, we looked at even higher levels by trying to detect the periodicities in the melodies of songs. The melodies are considered to be discrete and repetitive in the first place. However, this periodicities detection needs a brand new transform based on a set of square vectors. This set has yet to be defined, even though we found some linearly independent vectors for a start. Indeed, the Rademacher transform, that would be the closest to our need does not fulfil the shift invariance constraint we think is necessary for such a transform in the musical domain. By the way, we need to define musical properties that need to be observed with our pattern detection transform. This would require research in musicology, which is beyond the scope of this thesis. Moreover, we would need to perform an exhaustive research of the base vectors that would comply to these constraints. This is why we did not investigate much further in this domain.

Finally, we presented the piece of software we wrote to support our research. This application is aimed at being a multi-threaded audio-processing platform that would allow me to experiment with the concepts presented in this thesis.

Such a work we presented in this document shows many new tracks that we would follow in our future work.

The first one would be the enhancement of the estimators for our newly designed hierarchical models. Then, the link between vibrato and tremolo can be explored and maybe new facts about spectral envelopes might arise.

The enhanced time-scaling also suffers from the attack problem that is typical of the sinusoidal modelling. Hence, designing a hierarchical model integrating transients might lead to better enhance time-scaling techniques.

On another level, the musical modelling needs heavy research as for the base functions need for a transform detecting repetitive patterns in music. Once these patterns have been found, we need to study more in detail the properties of their repetitions and see if performing time-scaling on the parameters of this new transform can lead to interesting results, both theoretically but also perceptually.

In fact, many new research directions arise from this work, including applications to source separation or instrument recognition, and probably others that we did not think of.

Bibliography

[AD98] Daniel Arfib and Nathalie Delprat. Selective Transformations of Sounds Using Time-Frequency Representations: An Application to the Vibrato Modification. In *104th Convention of the AES*, Amsterdam, the Netherlands, May 1998. AES. Preprint 4652 (P5-2).

[AD99] Daniel Arfib and Nathalie Delprat. Alteration of the Vibrato of a Recorded Voice. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 186–189, Beijing, China, October 1999. International Computer Music Association (ICMA).

[AF95] François Auger and Patrick Flandrin. Improving the readability of time-frequency and time-scale representations by the reassignment method. *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 43:1068–1089, May 1995.

[AKZ02] Daniel Arfib, Florian Keiler, and Udo Zölzer. *DAFx – Digital Audio Effects*. John Wiley & Sons, 2002.

[All77] Jont B. Allen. Short-Term Spectral Analysis, Synthesis, and Modification by the Discrete Fourier Transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 25(3):235–238, 1977.

[ALP03] Giulio Agostini, Maurizio Longari, and Emanuele Pollastri. Musical instrument timbres classification with spectral features. *EURASIP Journal on Applied Signal Processing*, 1(11), 2003.

[Bad05] Roland Badeau. *Méthodes à haute résolution pour l'estimation et le suivi de sinusoïdes modulées. Application aux signaux de musique.* PhD thesis, École doctorale d'informatique, télécommunications et électronique de Paris, 2005.

[Bog05] Niels Bogaards. Analysis-Assisted Sound Processing with Audiosculpt. In *Proceedings of the Digital Audio Effects (DAFx) Conference*, pages 269–272. Universidad Politécnica de Madrid, Spain, 2005.

[BP05] Juan P. Bello and Jeremy Pickens. A Robust Mid-level Representation for Harmonic Content in Music Signals. In *International Conference on Music Information Retrieval*, October 2005.

[Bre90] Albert S. Bregman. *Auditory Scene Analysis: The Perceptual Organization of Sound*. The MIT Press, 1990.

[BRR04] Niels Bogaards, Axel Röbel, and Xavier Rodet. Sound Analysis and Processing with Audiosculpt 2. In *Proceedings of the International Computer Music Conference (ICMC)*, 2004.

[Bur67] John P. Burg. Maximum Entropy Spectral Analysis. In *Proc. 37th Meeting Soc. Exploration Geophys.*, 1967.

[Coo93] Martin Cooke. *Modelling Auditory Processing and Organization*. Cambridge University Press, New York, 1993.

[cso] Csound official website. World Wide Web. Online. `http://www.csounds.com`.

[DCM00] Myriam Desainte-Catherine and Sylvain Marchand. High Precision Fourier Analysis of Sounds Using Signal Derivatives. *Journal of the Audio Engineering Society*, 48(7/8):654–667, July/August 2000.

[DGR93] Philippe Depalle, Guillermo Garcia, and Xavier Rodet. Analysis of Sound for Additive Synthesis: Tracking of Partials Using Hidden Markov Models. In *Proceedings of the International Computer Music Conference (ICMC)*, San Francisco, 1993. International Computer Music Association (ICMA).

[Dol86] Mark Dolson. The phase vocoder: A tutorial. *Computer Music Journal*, 10(4):14–27, Winter 1986.

[DQ97] Yinong Ding and Xiaoshu Qian. Processing of Musical Tones Using a Combined Quadratic Polynomial-Phase Sinusoid and Residual (QUASAR) Signal Model. *Journal of the Audio Engineering Society*, 45(7/8):571–584, July/August 1997.

[Dud39] Homer Dudley. The Vocoder. *Bell Labs*, 18:122–126, 1939.

[EB04] Jana Eggink and Guy J. Brown. Instrument Recognition in Accompanied Sonoatas and Concertos. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, November 2004.

[ER95] Dan Ellis and David Rosenthal. Mid-level Representations for Computational Auditory Scene Analysis. In *International Joint Conference on Artificial Intelligence - Workshop on Computational Auditory Scene Analysis*, August 1995.

[FCQ98] Pablo Fernandez and Javier Casajus-Quiros. Multi-Pitch Estimation for Polyphonic Musical Signals. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 3565–3568, April 1998.

[FEJ54] Grant Fairbanks, W.L. Everitt, and R.P. Jaeger. Method for Time of Frequency Compression-Expansion of Speech. *IEEE Transactions on Audio and Electroacoustics*, AU-2:7–12, January 1954.

[FG66] J.L. Flanagan and R.M. Golden. Phase Vocoder. *Bell System Technical Journal*, 45:1493–1509, November 1966.

[FGM05] Mohammad Firouzmand, Laurent Girin, and Sylvain Marchand. Comparing Several Models for Perceptual Long-Term Modeling of Amplitudes and Phase Trajectories of Sinusoidal Speech. In *Proceedings of the INTERSPEECH - EUROSPEECH Conference*, Lisboa, Portugal, September 2005.

[FJ05] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. special issue on "Program Generation, Optimization, and Platform Adaptation".

[GFM04] Laurent Girin, Mohammad Firouzmand, and Sylvain Marchand. Long Term Modeling of Phase Trajectories within the Speech Sinusoidal Model Framework. In *Proceedings of the INTERSPEECH – 8th International Conference on Spoken Language Processing (ICSLP'04)*, Jeju Island, Korea, October 2004.

[GMdM+03] Laurent Girin, Sylvain Marchand, Joseph di Martino, Axel Röbel, and Geoffroy Peeters. Comparing the Order of a Polynomial Phase Model for the Synthesis of Quasi-Harmonic Audio Signals. In *Proc. Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, New York, USA, October 2003. IEEE.

[Gro96] Stephen Grossberg. *Pitch Based Streaming in Auditory Perception.* Cambridge MA, Mit Press, 1996.

[GS97] Bryan E. George and Mark J. T. Smith. Speech Analysis/Synthesis and Modification Using an Analysis-by-Synthesis/Overlap-Add Sinusoidal Model. *IEEE Transactions on Speech and Audio Processing*, 5(5):389–406, September 1997.

[Har78] Fredric J. Harris. On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform. In *Proceedings of the IEEE*, volume 66, pages 51–83, 1978.

[HDC01] Pierre Hanna and Myriam Desainte-Catherine. Influence of Frequency Distribution On Intensity Fluctuation of Noise. In *Proceedings of the Digital Audio Effects (DAFx) Conference*, pages 120–124. University of Limerick and COST (European Cooperation in the Field of Scientific and Technical Research), December 2001.

[hyp] The hyperspec. World Wide Web. Online. `http://www.lispworks.com/documentation/HyperSpec/Front/index.htm`.

[iow] The Iowa Music Instrument Samples. Online. `http://theremin.music.uiowa.edu`.

[Ita75] Fumitada Itakura. Minimum Prediction Residual Principle Applied to Speech Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):67–72, February 1975.

[JMJ99] Anil K. Jain, M. Narasimha Murty, and Flynn Patrick J. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[Joh67] S. C. Johnson. Hierarchical Clustering Schemes. *Psychometrika*, (2):241–254, 1967.

[KKS01] Ismo Kauppinen, Jyrki Kauppinen, and Pekka Saarinen. A Method for Long Extrapolation of Audio Signals. *Journal of the Audio Engineering Society*, 49(12):1167–1180, December 2001.

[KKZS03] Florian Keiler, Can Karadogan, Udo Zölzer, and Albrecht Schneider. Analysis of Transient Musical Sounds by Auto-Regressive Modeling. In *Proceedings of the Digital Audio Effects (DAFx) Conference*, pages 301–304, Queen Mary, University of London, United Kingdom, September 2003.

[Kla02] Anssi Klapuri. Separation of Harmonic Sounds Using Linear Models for the Overtone Series. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 6, pages 3089 – 3092, 2002.

[KM02] Florian Keiler and Sylvain Marchand. Survey on Extraction of Sinusoids in Stationary Sounds. In *Proceedings of the Digital Audio Effects (DAFx) Conference*, pages 51–58. University of the Federal Armed Forces, Hamburg, Germany, September 2002.

[KR02] Ismo Kauppinen and Kari Roth. Audio Signal Extrapolation – Theory and Applications. In *Proceedings of the Digital Audio Effects (DAFx) Conference*, pages 105–110. University of the Federal Armed Forces, Hamburg, Germany, September 2002.

[KR05] Ismo Kauppinen and Kari Roth. Improved Noise Reduction in Audio Signals using Spectral Resolution Enhancement with Time-Domain Signal Extrapolation. *IEEE Transactions on Speech and Audio Processing*, pages 1210–1216, November 2005.

[KS04] A.G. Krishna and Thippur V. Sreenivas. Music Instrument recognition : from Isolated Notes to Solo Phrases. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Montreal, Canada, May 2004.

[Lag04] Mathieu Lagrange. *Modélisation sinusoïdale des sons polyphoniques*. PhD thesis, Université Bordeaux 1, 2004.

[Lag05] Mathieu Lagrange. A New Dissimilarity Metric For The Clustering Of Partials Using The Common Variation Cue. In *Proceedings of the International Computer Music Conference (ICMC)*, Barcelona, Spain, September 2005. International Computer Music Association (ICMA).

[LD97] Jean Laroche and Mark Dolson. Phase-vocoder: About this Phasiness Business. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, NY, USA, October 1997.

[LMR04] Mathieu Lagrange, Sylvain Marchand, and Jean-Bernard Rault. Using Linear Prediction to Enhance the Tracking of Partials. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 4, pages 241–244, May 2004.

[LMR05a] Mathieu Lagrange, Sylvain Marchand, and Jean-Bernard Rault. Improving the Tracking of Partials for the Sinusoidal Modeling of Polyphonic Sounds. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 4, pages 241–244, March 2005.

[LMR05b] Mathieu Lagrange, Sylvain Marchand, and Jean-Bernard Rault. Long Interpolation of Audio Signals Using Linear Prediction in Sinusoidal Modeling. *Journal of the Audio Engineering Society*, 53(10):891–905, October 2005.

[LMR05c] Mathieu Lagrange, Sylvain Marchand, and Jean-Bernard Rault. Tracking Partials for the Sinusoidal Modeling of Polyphonic Sounds. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Philadelphia, USA, March 2005.

[LMR07] Mathieu Lagrange, Sylvain Marchand, and Jean-Bernard Rault. Enhancing the Tracking of Partials for the Sinusoidal Modeling of Polyphonic Sounds. *IEEE Transactions on Speech, Audio, and Language Processing*, 2007. to be published.

[LMRR03] Mathieu Lagrange, Sylvain Marchand, Martin Raspaud, and Jean-Bernard Rault. Enhanced Partial Tracking Using Linear Prediction. In *Proceedings of the Digital Audio Effects (DAFx) Conference*, pages 141–146. Queen Mary University of London, United Kingdom, September 2003.

[LN04] Olivier Lartillot-Nakamura. *Fondements d'un Système d'Analyse Musicale Computationnelle suivant une Modélisation Cognitiviste de l'Écoute.* PhD thesis, Université Paris 6, February 2004.

[MA86] J. Marques and L. Almeida. A Background for Sinusoid Based Representation of the Voiced speech. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pages 1233–1236, Tokyo, 1986.

[Mac67] J.B. MacQueen. Some Methods for Classification and Analysis of Multivariate Observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, Berkeley, California, USA, 1967. University of California Press.

[Mak75] John Makhoul. Linear Prediction : A Tutorial Review. *Proceedings of the IEEE*, 63(4), April 1975.

[Mar99] Keith Dana Martin. *Sound-Source Recognition : A Theory and Computational Model.* PhD thesis, Massachusets Institute of Technology, 1999.

[Mar00] Sylvain Marchand. *Sound Models for Computer Music (analysis, transformation, synthesis).* PhD thesis, University of Bordeaux 1, LaBRI, December 2000.

[Mas96] Paul Masri. *Computer Modeling of Sound for Transformation and Synthesis of Musical Signals*. PhD thesis, University of Bristol, 1996.

[MC97] Michael W. Macon and Mark A. Clements. Sinusoidal Modeling and Modification of Unvoiced Speech. *IEEE Transactions on Speech and Audio Processing*, 5(6):557–560, 1997.

[McA89] Stephen McAdams. Segregation of Concurrents Sounds : Effects of Frequency Modulation Coherence. *Journal of the Audio Engineering Society*, 86(6):2148–2159, 1989.

[mcc] Mcclim web site. World Wide Web. Online. `http://common-lisp.net/project/mcclim/`.

[Mic68] Donald Michie. Memo Functions and Machine Learning. *Nature*, 218:19–22, 1968.

[ML95] E. Moulines and J. Laroche. Non-Parametric Techniques for Pitch-Scale and Time-Scale Modification of Speech. *Speech Communication*, 16:175–206, 1995.

[ML03] Aaron S. Master and Yi-Wen Liu. Robust Chirp Parameter Estimation for Hann Windowed Signals. In *IEEE International Conference on Multimedia and Exposition (ICME)*, 2003.

[ML06] Sylvain Marchand and Mathieu Lagrange. On the Equivalence of Phase-Based Methods for the Estimation of Instantaneous Frequency. In *Proceedings of the 14th European Conference on Signal Processing (EUSIPCO'2006)*, Florence, Italy, September 2006.

[MQ86] Robert J. McAulay and Thomas F. Quatieri. Speech Analysis/Synthesis Based on a Sinusoidal Representation. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(4):744–754, 1986.

[MR04] Sylvain Marchand and Martin Raspaud. Enhanced Time-Stretching Using Order-2 Sinusoidal Modeling. In *Proceedings of the Digital Audio Effects (DAFx) Conference*, pages 76–82. Federico II University of Naples, Italy, October 2004.

[MW00] Maureen Mellody and Gregory H. Wakefield. The Time-Frequency Characteristic of Violin Vibrato: Modal Distribution Analysis and Synthesis. *Journal of the Acoustical Society of America*, 107:598–611, 2000.

[Nea04] Andrew Nealen. An As-Short-As-Possible Introduction to the Least Squares, Weighted Least Squares and Moving Least Squares Methods for Scattered Data Approximation and Interpolation. Online. `http://www.nealen.com/projects/`, May 2004.

[NLVS99] Scot Nathan Levine, Tony S. Verma, and Julius O. Smith. Multiresolution Sinusoidal Modeling for Wideband Audio with Modifications. *IEEE International Conference on Acoustics, Speech and Signal Processing*, 6:3585–3588, March 1999.

[Opp69] Alan V. Oppenheim. A Speech Analysis-Synthesis System Based on Homomorphic Filtering. *Journal of the Acoustical Society of America*, (45):458–465, February 1969.

[OS99] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing.* Prenctice Hall, 1999. Second Edition.

[Pac02] François Pachet. The Continuator: Musical Interaction with Style. In *Proceedings of ICMC*, pages 211–218. ICMA, September 2002. Best paper award.

[Por76] Michael R. Portnoff. Implementation of the Digital Phase Vocoder Using the Fast Fourier Transform. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 24(3):243–248, 1976.

[Puc95] Miller Puckette. Phase-locked Vocoder. In *Proceedings of the IEEE Conference on Applications of Signal Processing to Audio and Acoustics*, New Paltz, New York, USA, 1995.

[RÖ3] Axel Röbel. Transient Detection and Preservation in the Phase Vocoder. In *Proceedings of the International Computer Music Conference (ICMC)*, pages 247–250, 2003.

[RG04] Julie Rosier and Yves Grenier. Unsupervised Classification Techniques for Multipitch Estimation. In *116th Convention of the Audio Engineering Society*, volume 3, pages 201–204. AES, May 2004.

[RMG05] Martin Raspaud, Sylvain Marchand, and Laurent Girin. A Generalized Polynomial and Sinusoidal Model for Partial Tracking and Time Stretching. In *Proceedings of the Digital Audio Effects (DAFx) Conference*, pages 24–29. Universidad Politécnica de Madrid, Spain, October 2005.

[RPK86] R. Roy, A. Paulraj, and T. Kailath. ESPRIT — A Subspace Rotation Approach to Estimation of Parameters of Cisoids in Noise. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(5):1340—-1342, October 1986.

[RR05] Axel Röbel and Xavier Rodet. Efficient Spectral Envelope Estimation and its Application to Pitch Shifting and Envelope Preservation. In *Proceedings of the Digital Audio Effects (DAFx) Conference*, pages 30–35. Universidad Politécnica de Madrid, Spain, October 2005.

[RSB05] Emmanuel Ravelli, Mark Sandler, and Juan P. Bello. Fast Implementation for Non-Linear Time-Scaling of Stereo Signals. In *Proceedings of the Digital Audio Effects (DAFx) Conference*, pages 182–185. Universidad Politécnica de Madrid, Spain, 2005.

[RSM06] Matthias Robine, Robert Strandh, and Sylvain Marchand. Fast Additive Sound Synthesis Using Polynomials. In *Proceedings of the Digital Audio Effects (DAFx) Conference*, pages 181–186. McGill University, Montreal, Canada, September 2006.

[RW85] SS Roucos and A.M. Wilgus. High Quality Time-Scale Modification of Speech. In *IEEE Proceedings of the International Conference on Acoustics, Speech and Signal Processing*, Tampa, Florida, USA, March 1985.

[Sei04] Peter Seibel. *Practical Common Lisp*. Apress, September 2004. Online. `http://www.gigamonkeys.com/book/`.

[Ser89] Xavier Serra. *A System for Sound Analysis / Transformation / Synthesis Based on a Deterministic plus Stochastic Decomposition*. PhD thesis, CCRMA, Department of Music, Stanford University, 1989.

[Ser97] Xavier Serra. *Musical Signal Processing*, chapter Musical Sound Modeling with Sinusoids plus Noise, pages 91–122. Studies on New Music Research. Swets & Zeitlinger, Lisse, the Netherlands, 1997.

[SG84] Julius O. Smith and Phil Gossett. A Flexible Sampling-Rate Conversion Method. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages 19.4.1–19.4.2, San Diego, 1984. IEEE.

[Smi00] Julius O. Smith. Digital Audio Resampling Home Page. Technical report, CCRMA, 2000. Online. `http://www-ccrma.stanford.edu/~jos/resample/resample.html`.

[Soc40] Societé Montyon et Franklin. *Portraits et histoire des hommes utiles, collection de cinquante portraits*. 1839–1840.

[Ver06a] Didier Verna. Beating C in Scientific Computing Applications – On the Behavior and Performance of Lisp, Part I. In *Third European LISP Workshop at ECOOP*, Nantes, France, 2006.

[Ver06b] Didier Verna. How to Make Lisp Go Faster than C. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, Hong Kong, 2006. International Association of Engineers.

[VGD05] Vincent Verfaille, Catherine Guastavino, and Philippe Depalle. Perceptual evaluation of vibrato models. In *Actes du Colloque Interdisciplinaire de Musicologie*, Montreal, Canada, March 2005.

[VK00] Tuomas Virtanen and Anssi Klapuri. Separation of Harmonic Sound Sources Using Sinusoidal Modeling. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 2, pages 765–768, April 2000.

[VVHK99] Koen Vos, Renat Vafin, Richard Heusden, and Bastiaan Kleijn. High-Quality Consistent Analysis-Synthesis in Sinusoidal Coding. In *AES 17th International Conference on High-Quality Audio Coding*, September 1999.

[Wal23] Joseph Leonard Walsh. A Closed Set of Normal Orthogonal Functions. *American Journal of Mathematics*, (45):5–24, 1923.

[War63] Joe H. Ward. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58:238 – 244, 1963.

List of Figures

List of Tables

Modèles spectraux hiérachiques pour les sons et applications

Résumé : Dans ce mémoire, nous nous intéressons à l'étirement temporel du son sans artefact. Nous nous plaçons dans le cadre des sons quasi-harmoniques à faible niveau de bruit.

Nous présentons ici une nouvelle approche hiérarchique de la modélisation du son, en nous basant sur la transformée de Fourier et le modèle sinusoïdal appliqués aux paramètres de contrôle du son. Nous introduisons donc la modélisation des signaux de contrôle du son ainsi que ses applications (incluant des améliorations de l'algorithme de suivi de partiels), puis nous ouvrons la voie vers la modélisation au niveau musical en nous basant sur la recherche de périodicités dans la mélodie.

À chaque niveau, nous proposons une technique d'étirement temporel du son. En manipulant les paramètres des signaux de contrôle du son, nous arrivons ainsi à un étirement du son préservant la hauteur, le timbre et les modulations.

Nous montrons également le travail de développement logiciel effectué pour valider les résultats présentées dans ce document.

Discipline : Informatique

Mots-Clefs : Traitement du signal sonore, transformée de Fourier, modèle sinusoïdal, étirement temporel du son.

Hierarchical spectral models for sounds and applications

Abstract : In this thesis, we focus on artifact-free time scaling of sounds. We work here on quasi-harmonic sounds with low noise levels.

We present here a new hierarchical approach of sound models based on the Fourier transform and the sinusoidal model applied to the control parameters of sound. Thus, we introduce the modelling of sound control signals with its applications (including enhancements to the partial-tracking algorithm). Then, we open the way to musical modelling, based on research of periodicities in the melody.

At each level, we propose a technique for time-scaling of sound. Working on the parameters of the control signals of sound, we successfully time-scale sounds with preservation of pitch, timbre and modulations.

We also show the software development work performed in order to validate the results presented in this document.

Discipline : Computer Science

Keywords : Audio signal processing, Fourier transform, sinusoidal model, time-scaling of sound.

LaBRI,
Université Bordeaux 1,
351 cours de la Libération,
33405 Talence Cedex (FRANCE).