Stage de Recherche

(magistère EEA, deuxième année)

—

Inharmonicité des instruments à cordes libres : Etude et programmation d'un logiciel d'aide à la facture.

—

stage encadré par Bertrand DAVID au laboratoire TSI*
à l'Ecole Nationale Supérieure
des Télécomunications de Paris

Nicolas STURMEL

22 août 2004

*TSI : Traitement du signal et de l'image

Table des matières

1 Introduction

Ce stage a été effectué dans le cadre de ma deuxième année de magistère EEA. Afin de me fournir une expérience dans le domaine de la recherche.

Etant donné mon grand intérêt pour le traitement du son appliqué à la musique, je me suis naturellement tourné vers les laboratoires exeçcant des activités dans ce domaine. Et c'est à la suite d'une demande auprès du directeur du laboratoire de Traitement des Signaux et des Images (TSI) de l'Ecole Nationale Superieur des Télécomunications de Paris (ENST) que j'ai eu l'occasion de rencontrer M. Bertrand DAVID (par l'intermédiaire de M. Gaël Richard), chercheur dans le groupe Audio, Acoustique et Ondes (AAO) du même laboratoire. A l'issue de l'entretien, le présent stage m'a été proposé.

1.1 Présentation du laboratoire

L'ENST n'a pas seulement une vocation de formation d'ingénieurs, les laboratoires qu'elle acceuille ont acquis une renommée mondiale dans le domaine, notamment, des télécommunications.

Le laboratoire TSI possède des locaux dans les deux bâtiments de l'ENST : ceux de la rue Dareau et ceux de la rue Barrault, dans le 13ème arondissement de Paris. C'est à la rue Dareau que j'ai eu la possibilité de mener mon travail de recherche. Les domaines de recherches couverts par le laboratoire sont très vastes : de la reconnaissance vocale à la compression de données en passant par l'acoustique et la synthèse. C'est pourquoi il est divisé en 5 groupes :

- Un groupe "Traitement et Interprétation des Images" (TII)
- Un groupe "Traitements Statistiques et Applications aux Communications" (TSAC)
- Un groupe "Perception, Apprentissage et Modélisation" (PAM)
- Un groupe "Codage"
- Un groupe "Audio, Acoustique et Ondes" (AA0)

C'est dans ce dernier groupe que se sont concentrées mes activités.

1.2 Contexte du stage

Le travail présenté dans ce rapport est inclus dans le domaine de la représentation et l'estimation des signaux musicaux. Il utilise comme outil principal d'analyse et de traitement des signaux dits à haute résolution (et plus spécifiquement les méthodes dites sous-espace). Grâce à ces outils, le cas de signaux de musique présentant des battements entre deux fréquences très proche peut être appréhendé. Ces méthodes permettent en effet de dépasser le classique compromis entre les précisions fréquentielles et temporelles auquel sont soumises les méthodes à base de transformée de Fourier.

L'implémentation choisie pour le travail est une interface homme machine, programmée dans un premier temps à partir des outils Matlb^{TM} . Elle doit proposer une représentation à la fois accessible et complète des données et proposer plusieurs méthodes d'analyses et paramètres afin d'adapter le comportement de l'interface au mieux en rapport avec les données analysées. Le but final est de proposer cet outil à une large gamme d'utilisateurs potentiels, qu'ils soient professionnels de la musique ou chercheur ; donc familier ou non avec les techniques et le vocabulaire du traitement du signal ainsi que l'environement Matlab.

Cet outil pourrait, par exemple, servir à tout chercheur en acoustique musicale qui a souvent la nécéssité de mesurer rapidement les fréquences et les amortissement associés qui composent un son.

2 Objectifs du stage

L'équipe disposait alors d'une interface Matlab^{TM}pour représenter les données et estimer les résultats des analyses sus-décrites à l'aide de la méthode Matrix Penciel [**1**]. Les objectifs de ce stage se sont donc inscrits dans la continuité du travail déjà effectué.

Dans un premier temps il s'agissait d'étudier l'integrabilité de la méthode hautre résolution ESPRIT à l'interface éxistante. Notamment via un processus de préaccentuation du signal dont l'intérêt sera vu par la suite.
Dans un second temps, il a fallu permettre l'estimation du coefficient d'inharmonicité[1] de la corde analysée. Finalement, optimiser l'interface homme-machine pour la rendre simple et robuste afin d'en accélérer l'exécution et de pouvoir la diffuser.

[1] Voir plus loin

3 Problèmes posés et traitement du signal

3.1 Problème

Dans un premier temps il est nécéssaire de résoudre le problème de l'estimation des signaux. Soit une note de piano :

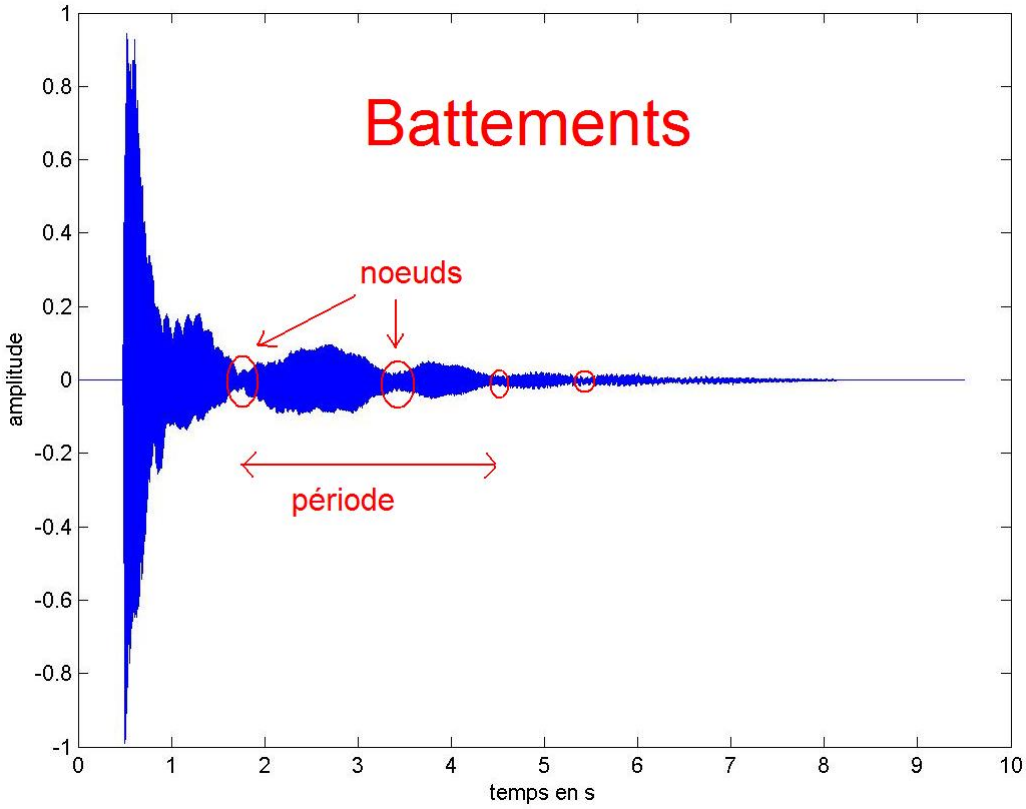

Fig 3.1 Note de piano

Nous voyons, sur la représentation, des battements qui traduisent la présence de composantes fréquencielles très proches. L'estimation de ces composantes et de leurs paramètres associés nécéssite donc des méthodes possédant une meilleure résolution que les méthodes traditionnelles à base de transformée de Fourier.

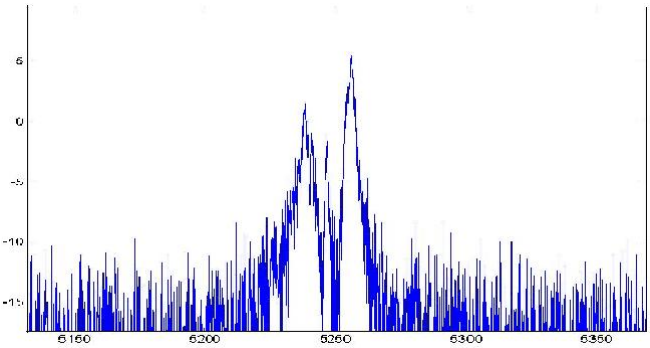

Fig 3.2 Limites de TF

On choisit donc d'appliquer une méthode dite "sous-espace", qui est une méthode haute résolution dont l'idée principale a été émise par Prony **[6]**. Nous allons donc voir comment

on peut appliquer une méthode haute résolution (en l'occurence, la méthode ESPRIT[2]) à des signaux musicaux pour arriver à en estimer les différentes composantes. La méthode ESPRIT est particulièrement intéréssante car, basée sur l'invariance rotationelle du sous espace signal, elle se révèle nettement moins complexe que l'algorithme Matrix Pencil déjà en place dans l'interface.

3.2 Théorie

Dans cette section, on expose l'aspect théorique qui nous donne la démarche pour appliquer la méthode haute résolution ESPRIT à un signal musical.

3.2.1 Modèle du signal

On considère un signal composé uniquement de sinusoïdes amorties (appellées partiels). Sa représentation en temps discret est la suivante :

$$x[t] = \sum_{k=0}^{K-1} a_k e^{((\alpha_k + i2\pi f_k)t + \Phi_k)}$$

- Les paramètres $a_k > 0$, $\alpha_k > 0$ et $f_k \in [-\frac{1}{2}, \frac{1}{2}[$ sont fixés et inconnus
- Φ_k sont des variables aléatoires et uniformes sur $[-\pi, \pi[$.

Où les a_k représentent les amplitudes complexes associées à chaque pôle $z_k = e^{i2\pi f_k + \alpha_k}$ du signal. Ainsi, on a aussi :

$$x[t] = \sum_{k=0}^{K-1} a_k e^{i\Phi_k} z_k^t$$

Le but de la méthode ESPRIT est donc de déterminer les paramètres z_k et $a_k e^{i\Phi_k}$ du signal que l'on considère bruité par un bruit blanc gaussien complexe de variance σ^2 :

$$s[t] = x[t] + b[t]$$

3.2.2 Formalisme mathématique, propriété de la matrice de covariance

On peut vérifier que $s[t]$ est un processus centré, stationnaire au second ordre au sens large, de fonction d'autocovariance :

$$r_{ss}[m] = \mathbb{E}[s^*[t]s[t+m]] = \sum_{k=0}^{K-1} a_k^2 e^{(i2\pi f_k + \alpha_k)m} + \sigma^2\delta[m]$$

De même si on considère n tel que $K < n \leqslant N - K + 1$ et $\mathbf{x}(t) = [x(t), \ldots, x(t+n-1)]^t$, $\mathbf{b}(t) = [b(t), \ldots, b(t+n-1)]$, $\mathbf{s}(t) = [s(t), \ldots, s(t+n-1)]$, et $\mathbf{v}(t) = [1, z, \ldots, z^{n-1}]$: on peut estimer la matrice de covariance $\mathbf{R}_{ss} \triangleq \mathbb{E}[\mathbf{s}(t), \mathbf{s}(t)^H]$ à l'aide de l'estimateur biaisé :

$$\widehat{\mathbf{R}}_{ss} = \frac{1}{N-n+1} \sum_{t=0}^{N-n} \mathbf{s}(t)\mathbf{s}(t)^H$$

Etant donné que $x[t]$ et $b[t]$ sont indépendants, $\mathbf{R}_{ss}$ est la somme de $\mathbf{R}_{xx} \triangleq \mathbb{E}[\mathbf{x}(t)\mathbf{x}(t)^H]$ et de $\mathbf{R}_{bb} = \sigma^2\mathbf{I}_n$:

$$\mathbf{R}_{ss} = \mathbf{R}_{xx} + \sigma^2\mathbf{I}_n$$

[2]ESPRIT : Estimation of Signal Parameters via Rotational Invariance Technique, ou encore : estimation des paramètres du signal via invariance rotationelle

La matrice $\mathbf{R}_{xx}$ est naturellement de rang égal à K. Ses valeurs propres λ_k sont donc nulles pour $k > K$. Ainsi, les valeurs propres $\bar{\lambda}_m$ de $\mathbf{R}_{ss}$ seront telles que :

$$\bar{\lambda}_m = \begin{cases} \lambda_m + \sigma^2 & \forall m \in \{0, \ldots, K-1\} \\ \sigma^2 & \forall m \in \{K, \ldots, n-1\} \end{cases}$$

Soit $\{\mathbf{w}_m\}_{m=0\ldots n-1}$ la famille composée des vecteurs propres de la matrice $\mathbf{R}_{xx}$; cette famille est aussi une base de la matrice $\mathbf{R}_{ss}$. On appelle l'espace de dimension K généré par la famille $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{K-1}]$ le sous espace **signal**. Celui engendré par les $n - K$ derniers vecteurs est appelé le sous espace bruit.

C'est en travaillant sur cet espace signal que nous allons arriver à déterminer les paramètres du signal, notamment en utilisant les propriétés d'invariance par rotation.

3.2.3 La matrice de Vandermonde

soit $\mathbf{x} = [x[0], \ldots, x[N-1]]^T$ et $\mathbf{v}(z) = [1, z^2, z^3, \ldots, z^{N-1}]$ alors on peut remarquer que :

$$\mathbf{x} = \sum_{k=0}^{K-1} \alpha_k \mathbf{v}(z_k)$$

On peut encore plus condenser cette écriture en introduisant la matrice de Vandermonde de dimension N par K $\mathbf{V} = [\mathbf{v}(z_0), \ldots, \mathbf{v}(z_{K-1}]$ et $\alpha = [\alpha_0, \ldots, \alpha_{K-1}]^T$:

$$\mathbf{x} = \mathbf{V}\alpha$$

La matrice $\mathbf{V}$ est appellée matrice de Vandermonde. Il peut être intéréssant de remarquer que cette matrice de Vandermonde est directement liée aux pôles du signal.

3.2.4 La méthode esprit plus en détails

Il s'agit de remarquer une propriété du sous espace signal et de la matrice de Vandermonde : Si on définit $\mathbf{V}_\uparrow$ comme la matrice des $n-1$ dernières lignes de la matrice $\mathbf{V}$ génératrice de ce qui a été défini comme l'espace signal ; et que nous définissons de la même manière $\mathbf{V}_\downarrow$ la matrice contenant les $n-1$ premières lignes de $\mathbf{V}$; alors :

$$\mathbf{V}_\uparrow = \mathbf{V}_\downarrow \mathbf{D}$$

Où la matrice $\mathbf{D}$ est la matrice diagonale telle que $\mathbf{D} = diag(z_0, \ldots, z_{K-1})$

De même, en transposant cette définition à la matrice $\mathbf{W}$, alors on obtient :

$$\mathbf{W}_\uparrow = \mathbf{W}_\downarrow \mathbf{\Phi}$$

Où les valeurs propres de la matrice $\mathbf{\Phi}$ sont les pôles z_p estimés du signal à analyser. En effet,$\mathbf{V}$ et $\mathbf{W}$ sont deux matrices dans le même sous espace signal défini précédemment, on peut donc en conclure qu'il existe une matrice de passage C telle que :

$$\mathbf{W}C = \mathbf{V}$$

et donc que $\mathbf{\Phi} = C\mathbf{D}C^{-1}$. Et que par conséquent, V et W possèdent les mêmes valeurs propres.

On va donc, dans l'algorithme ESPRIT, chercher à estimer puis diagonaliser la matrice $\mathbf{\Phi}$

3.2.5 Algorithme

principe : Le principe de l'algorithme tient en quatre points :
- Déterminer et diagonaliser la matrice de covariance du signal
- Garder le sous espace signal de cette matrice
- Calculer la matrice $\mathbf{\Phi}$
- Diagonaliser $\mathbf{\Phi}$ pour déterminer ses valeurs propres : les pôles du signal.

Pour déterminer la matrice de covariance du signal on peut utiliser l'estimateur défini plus tôt dans le document (voir annexes) :

$$\widehat{\mathbf{R}}_{ss} = \frac{1}{N-n+1} \sum_{t=0}^{N-n} \mathbf{s}(t)\mathbf{s}(t)^H$$

De plus, pour déterminer plus facilement l'espace signal, on utilisera la fonction *svd* de Matlab^{TM}qui va classer les vecteurs du sous espace dans l'ordre décroissant des valeurs propres : la sélection des K premiers vecteurs nous donnera donc le sous espace signal.

code Matlab^{TM}

```
function [freq,amort]=esprit(S,n,k,Fe)
% Determination des frequences qui constituent S par la méthode haute
% résolution \textsc{esprit} (Estimation of Signal Parameters via Rotational
% Invariance Techniques)
%
%   usage :   [freq,amort]=ESPRIT(S,n,K,Fe)

% -------------- Etape 1 : calcul de la covariance

SS=covariance(S,max(size(Sbis)),n);

% -------------- Etape 2 : Determination du sous espace signal

[L,B,U]=svd(SS) ;
Us = U(:,1:k);

% -------------- Etape 3 : invariance par rotation

U1 = Us(1:(n-1),:); U2 = Us(2:n,:);
Psi = pinv(U1)*U2;

% -------------- Etape 4 : Determination des paramètres

phi = eig(Psi);

freq = angle(phi)/(2*pi);
amort = -log(abs(phi)) ;
```

3.2.6 Détermination des amplitudes complexes

L'algorithme ESPRIT tel qu'il est décrit pour l'instant ne permet pas de déterminer tous les paramètres du signal. Il est nécéssaire d'approcher les valeurs des amplitudes complexes α du signal. La méthode retenue est une estimation des moindres carrés linéaires qui est suffisante ici pour résoudre le problème. Il s'agit donc de construire la matrice de Vandermonde $\mathbf{V}$ du signal grâce aux pôles déterminés par la méthode haute résolution.

On a alors $\alpha \mathbf{V} = \tilde{S}(t)$ où $\tilde{S}(t)$ est le signal estimé. Appliquer la méthode des moindres carrés revient donc à minimiser l'erreur $\|S(t) - \alpha \mathbf{V}\|$. En calculant la pseudo inverse de la matrice de Vandermonde, on peut estimer α :

$$\alpha = (\mathbf{V}^H \mathbf{V})^{-1} \mathbf{V}^H S(t)$$

3.3 Application à des signaux réels

Dans cette section, nous voyons comment adapter l'algorithme défini ci dessus pour l'analyse de signaux musicaux réels.

3.3.1 Modifications dans le domaine temporel

Dans le respect du modèle, il est nécéssaire de modifier temporellement le signal. En effet, tout signal musical issu d'un instrument à corde libre ne possède pas seulement une phase de décroissance, mais aussi une phase dite "d'attaque" dont le modèle décrit en début de la partie précédente ne tient pas compte.

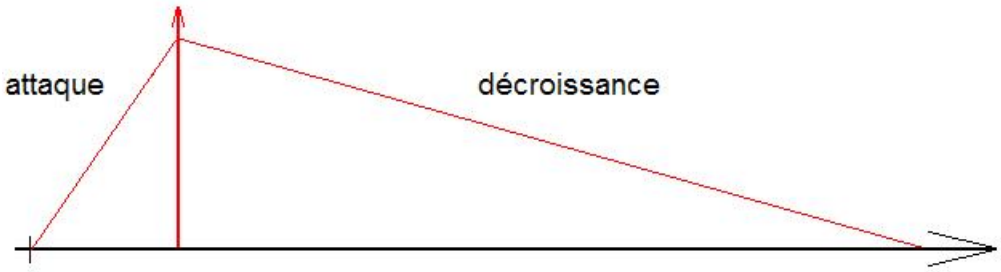

Fig 3.3 Enveloppe d'un signal musical issu d'un instrument à corde libre.

Pour que l'estimation des paramètres du modèle soit la plus juste possible, il convient d'avoir un signal correspondant à ce modèle, et donc, de supprimer l'attaque du signal.

On constate qu'à la fin de l'attaque, le signal atteint son maximum en amplitude. Il est donc facile de supprimer l'attaque en coupant la partie du signal se situant avant le maximum en amplitude.

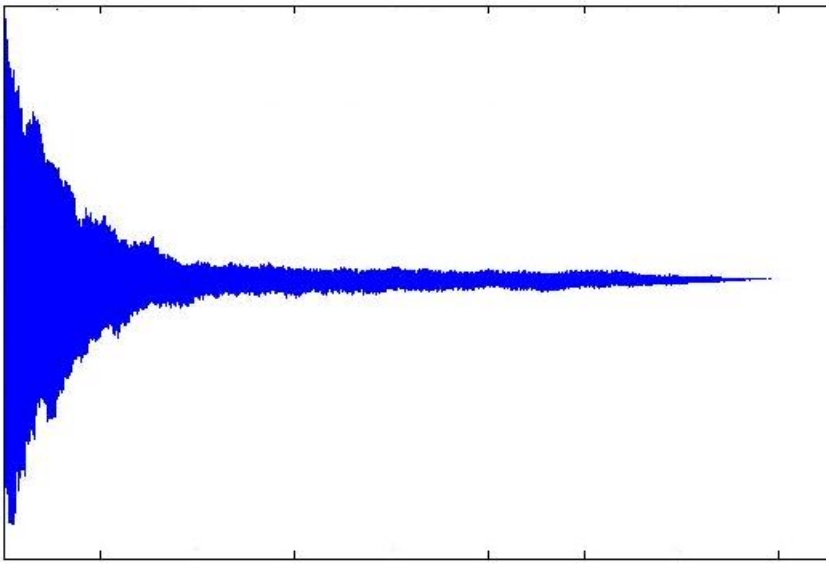

Fig 3.4 Signal retenu.

3.3.2 Problème dans le domaine fréquenciel / préaccentuation

Soit le spectre d'une note de piano, relativement typique du spectre d'une note quelconque jouée sur un instrument à corde libre :

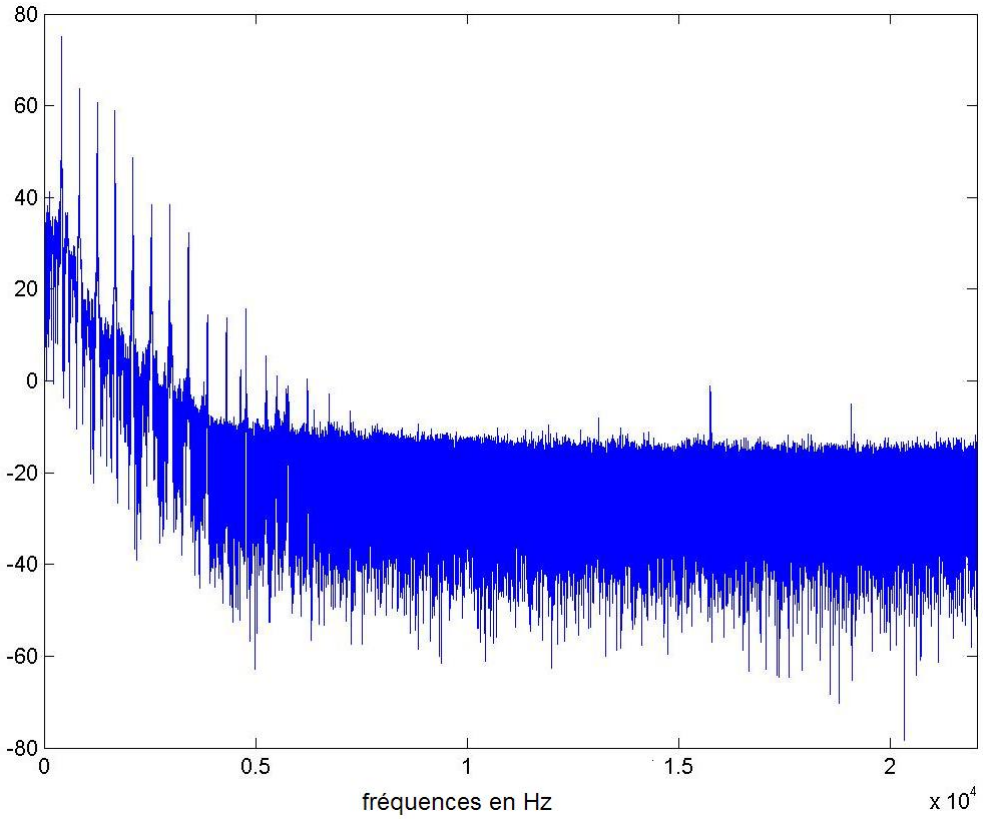

Fig 3.5 Note de piano.

Comme on peut s'y attendre pour un signal musical, ce spectre est à première vue harmonique. Mais nous constatons aussi que le niveau du bruit n'est pas homogène pour toutes les fréquences. Ceci est dû à plusieurs raisons :

- l'étalement spectral des pics (qui sont des sinusoïdes amorties) est tel que lorsque que les pics sont rapprochés, pour les notes graves par exemple, il suffit seul à créer un bruit de fond.
- l'absorbtion d'énergie lors de la frappe ou du pincement de la corde s'accompagne typiquement d'un bruit coloré qui, pour le piano, possède un maximum d'énergie autour de 200 Hz.

Nous voilà bien loin du modèle signal+bruit blanc que nous avons défini pour appppliquer la méthode esprit. Il est donc nécéssaire de tenter de blanchir le bruit, en touchant le moins possible au contenu spectral utile : les pics. C'est ce que nous appellerons la phase de préaccentuation du signal.

3.3.3 Blanchiment de tout le spectre par méthode LPC

La première idée est d'effectuer une estimation de l'enveloppe spectrale du signal par le biais de la méthode LPC (voir [**7**]). L'algorithme va estimer un filtre AR[3] qu'il suffira d'inverser pour

[3]Modèle autorégressif : $y[z] = a_1 y[z-1] + \ldots + a_n y[z-n]$ pour un filtre d'ordre n

tenter de blanchir le spectre.

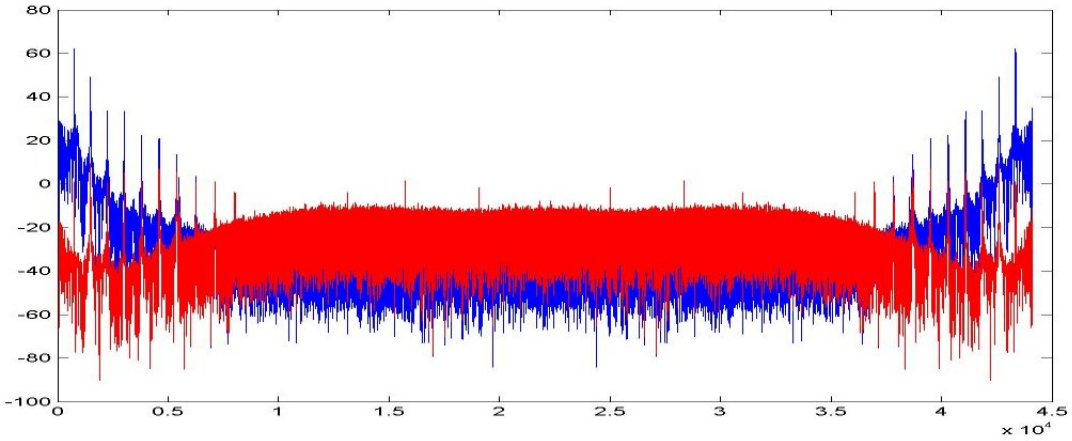

Fig 3.2 En bleu : spectre original, en rouge : spectre préaccentué par lpc

Malheureusement cette méthode présente des limites évidentes pour les signaux à bande spectrale limitée. En l'occurence, nous pouvons constater ici un surélèvement des hautes fréquences qui va amener l'algorithme ESPRIT à des eurreurs d'estimation.

3.3.4 Blanchiment du bruit seul

Pour palier au problème précédent on cherche une méthode consistant à estimer l'enveloppe du bruit seul. Pour celà, nous cherchons à filtrer la transformée de Fourier d'une partie du signal pour ne garder que le bruit. Le filtrage en question est réalisé avec un filtre médiant (code en annexes).

Le bruit étant considéré gaussien, donc à spectre réel, le filtrage est opéré sur la norme de la transformée de Fourier.

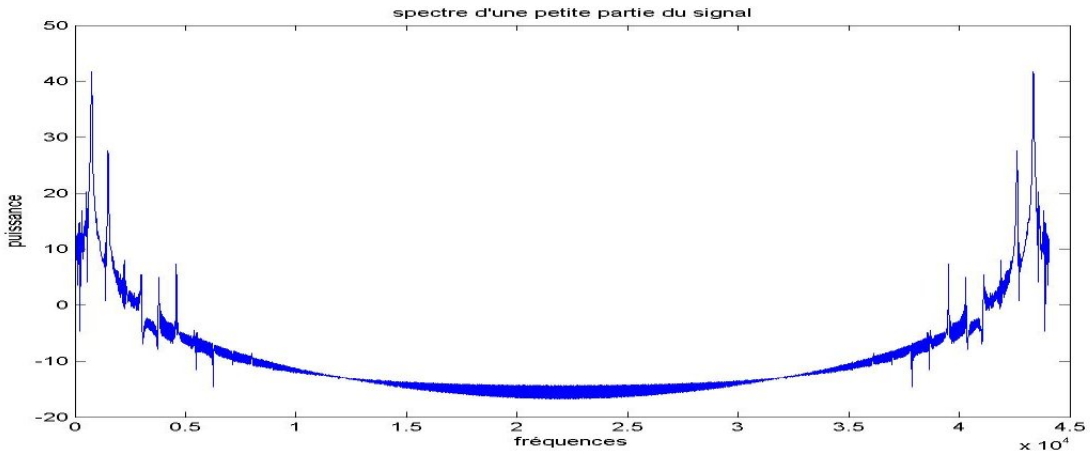

Fig 3.3 Spectre à court terme du signal

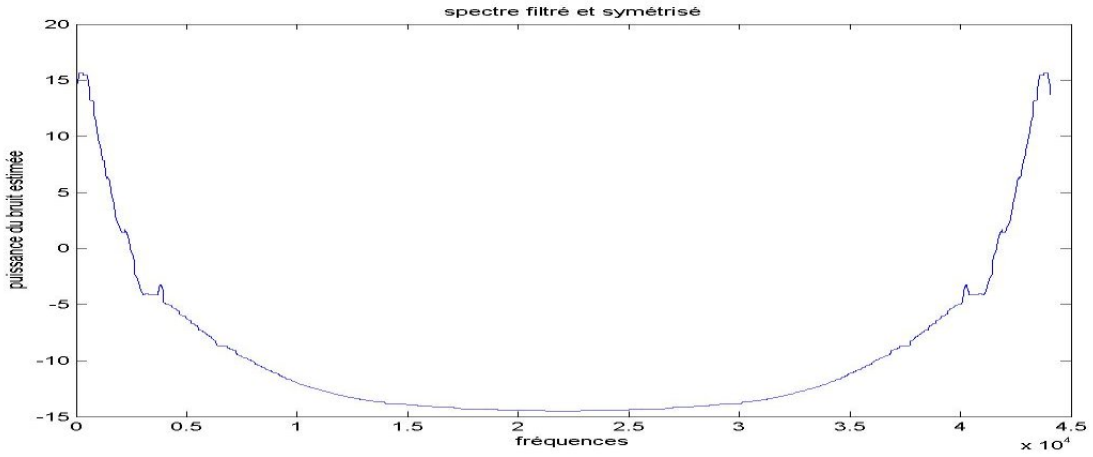

Fig 3.4 Spectre filtré médian.

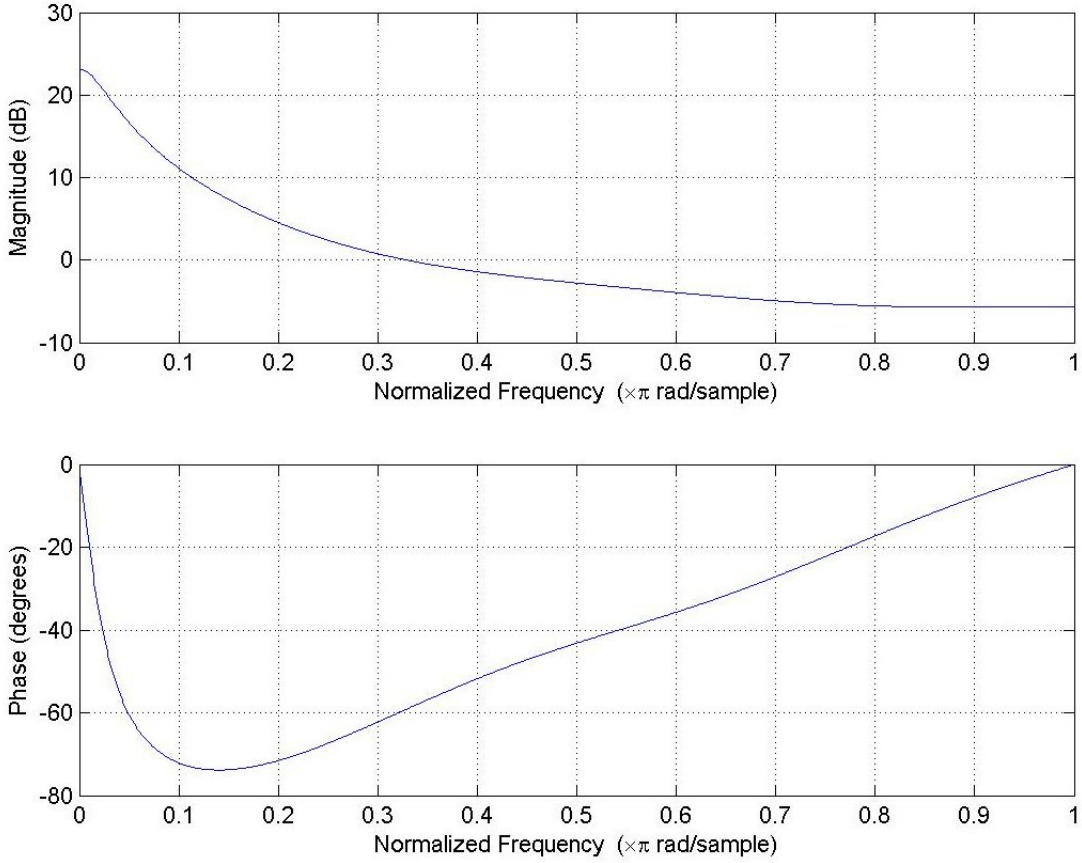

Fig 3.5 Filtre AR estimé

De plus, pour conserver la symétrie hermitienne et diminuer les calculs lors du filtrage en question, il est nécéssaire de reconstruire la partie correspondant aux fréquences réduites comprises entre 0.5 et 1.

Il est donc relativement simple à présent d'estimer un filtre AR correspondant à l'enveloppe spectrale obtenue, notamment en utilisant à nouveau l'algorithme de Levinson, et de blanchir le bruit contenu dans le signal original.

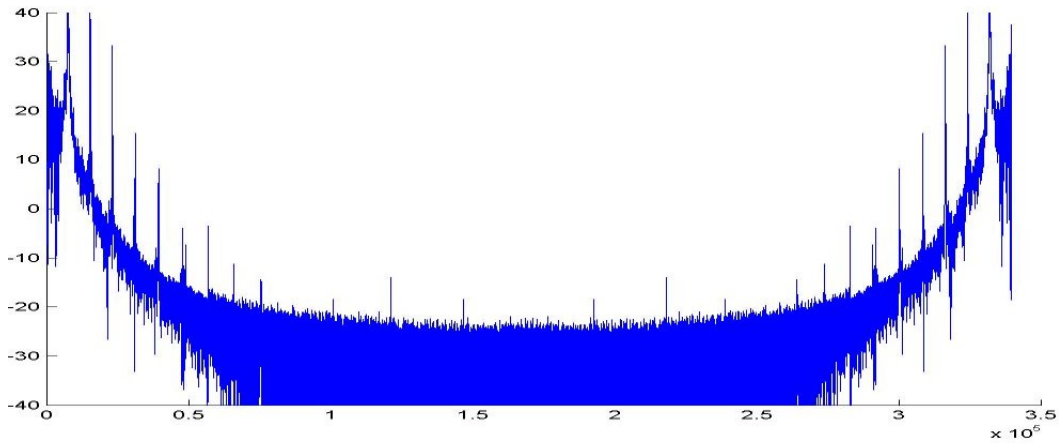

Fig 3.6 Sepctre du signal original.

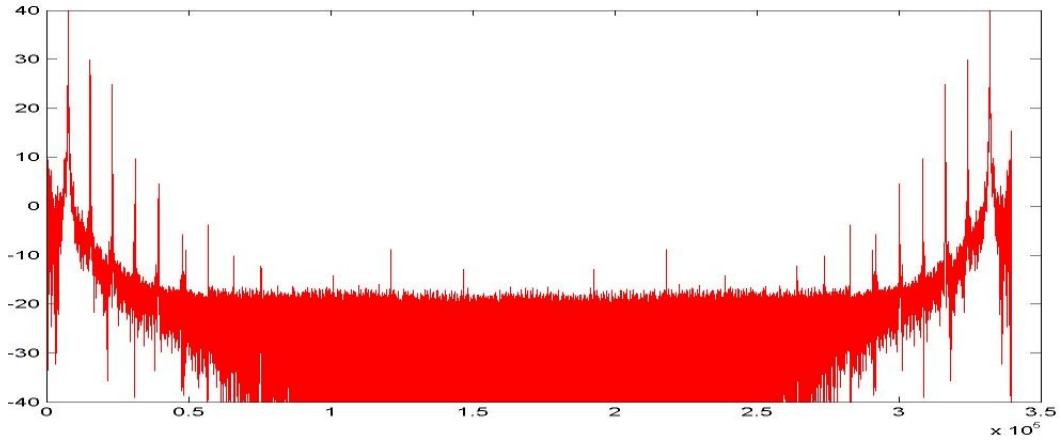

Fig 3.7 Spectre après tentative de blanchiment du bruit.

A présent, on constate que le filtre obtenu par cette méthode ne possède pas une profondeur d'action satisfaisante. Le bruit du signal préaccentué n'est pas blanc, mais la différence entre les basses fréquences et les hautes fréquences a tout de même été réduit ici de 20 dB.

3.3.5 Vitesse de Calcul

Dans le respect du cahier des charges imposé à l'interface, mais aussi pour le confort de l'utilisateur, il est nécéssaire d'avoir l'analyse la plus rapide possible. C'est pourquoi :

- la fonction de calcul de la matrice de covariance a été programmée en C et interfacée avec Matlab^{TM}via l'utilisation d'un MEX-file (voir Annexes),
- la détermination des amplitudes complexes ne se fait que sur la première seconde du signal. De cette manière les matrices manipulées restes raisonnables, comme de toute façon, les paramètres que nous cherchons alors à estimer ne dépendent que des conditions initiales du signal.

3.4 Premier résultat

Voici les premiers résultats obtenus par application de la méthode ESPRIT au signal présenté au début de la section (signal présentant des battements) :

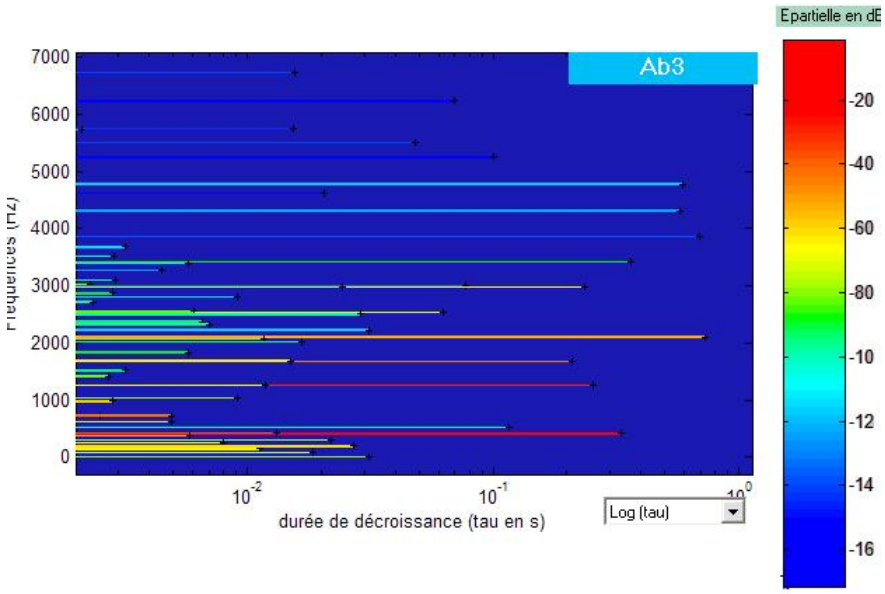

Fig 3.8 Premier résultat.

Ce résultat est obtenu pour une estimation de 100 pôles (50 partiels) à l'aide d'un matrice de covariance calculée sur une fenêtre d'estimation de 800 points.

4 Etude des cordes libres

4.1 Caractéristique du signal produit

4.1.1 Présentation du problème

On décrit la corde libre comme une corde homogène de masse linéique μ constante,fixe à ses deux extrémités tendues horizontalement le long de l'axe X par une force T_0. Nous négligeons son poids ainsi que toute cause d'amortissement devant les forces de tension. Au repos, la corde se confond avec l'axe Ox.

On étudie les petits mouvements de cette corde dans le plan xOy de part et d'autre de la position d'équilibre ; en admettant qu'un élément de corde au repos reste à la même abscisse pendant le mouvement. Pour ce faire, on applique une faible élongation à un point quelconque de la corde suivant l'axe Oy, pour enfin libèrer la corde.

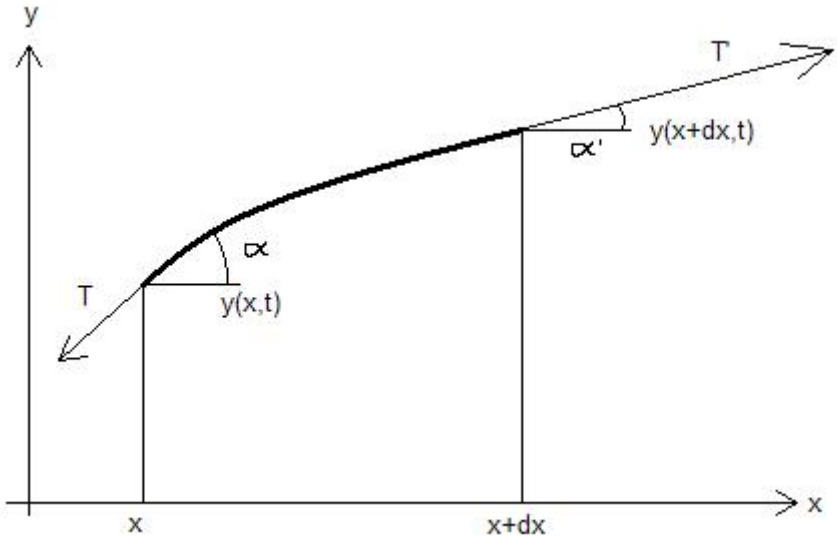

Fig 7.1 Portion de corde comprise entre x et x+dx

Soit la déformation transversale dans la direction y décrite à l'instant t pour le point d'abscisse x : y(x,t). Comme on fait subir une élongation faible, on considère que la pente de la corde, soit $\frac{\partial y}{\partial x} << 1$. On définit par l'angle $\alpha(x,t)$ l'inclinaison locale de la corde par rapport à l'axe x.

4.1.2 Résolution mathématique

Soit $\mathbf{T}(x)$ la tension au point d'abscisse x exercée par la partie de droite sur la partie de gauche. (sollicitation en extension). Avec les notations de la figure ci dessus, on a : $\vec{T} = -\mathbf{T}(x)$ et $\vec{T'}(x) = \mathbf{T}(x+dx)$.

En appliquant la relation fondamentale de la dynamique[4] à l'élément de masse μdx compris entre x et x+dx, on trouve :

$$\mu dx\vec{a} = \vec{T'} - \mathbf{T}(x+dx) - \mathbf{T}(x) = \frac{\partial \mathbf{T}}{\partial x}dx \quad soit \quad \mu\vec{a} = \frac{\partial \mathbf{T}}{\partial x}$$

[4] $\sum forces\_\vec{appliquees} = \vec{0}$

En projection sur Ox Il n'y a pas de déplacement selon x, donc $a_x = 0$.

$$0 = \frac{\partial T_x}{\partial x} \Rightarrow T_x = Tcos(\alpha) = Cte$$

Comme alpha est petit on a donc : $T = |\mathbf{T}| = Cte = T_0$.

En projection sur Oy :

$$\mu\frac{\partial^2 y}{\partial t^2} = \frac{\partial T_y}{\partial x} = \frac{\partial(Tsin(\alpha))}{\partial x} = T_0\frac{\partial^2 y}{\partial x^2}\ car\,\alpha\ petit,\ alors\,tan(\alpha) = \frac{\partial y}{\partial x}$$

D'où l'équation finale :

$$\frac{\partial^2 y}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 y}{\partial t^2}\ avec\,c = \sqrt{\frac{T_0}{\mu}}$$

En supposant les variables x et t séparables on trouve :

$$y(x,t) = F(x)G(t)$$

Ce qui nous donne, avec comme conditions aux limites F(x=0)=0 et F(x=L)=0 (corde fixe aux extrémités) :

$$G(t) = A_n cos(\omega_n t) + B_n sin(\omega_n t) \quad et \quad F(x) = D_n sin(\frac{\omega_n}{c}x)$$

où $\omega_n = \frac{n\pi c}{L}, n \in \mathbb{N}^*$. A chaque valeur de n correspond un mode ω_n ; ces modes sont les modes propres de la corde. Ainsi :

$$y_n(x,t) = (a_n cos\frac{n\pi ct}{L} + b_n sin\frac{n\pi ct}{L})sin\frac{n\pi x}{L}$$

En posant $c_n = \sqrt{a_n^2 + b_n^2}$ et $\phi_n = arctan\frac{a_n}{b_n}$, le mode n s'écrit sous la forme :

$$y_n(x,t) = (c_n sin\frac{\omega_n}{c}x)sin(\omega_n t + \phi_n)$$

On vient donc de démontrer que le spectre de vibration d'une corde était un spectre harmonique. On peut aussi démontrer que l'amplitude de chaque harmonique dépend principalement des conditions initiales, mais l'intérêt est limité dans le cas du présent rapport.

4.1.3 Inharmonicité

Le modèle utilisé pour les calculs ci-dessus est bien sur simplifié. Il ne donne qu'un aspect basique de la corde libre. En réalité la corde est rarement fixe à ses deux extrémités, et possède une raideur.

Le premier point se traduit par admittance mécanique au niveau des points d'ancrage.

Le deuxième point est propre aux craractéristiques propres de la corde dont on peut notamment tirer une impédance caractéristique de transmission de l'onde.

Ce sont là les deux principales causes de l'inharmonicité des cordes libres. Par calcul **[4]**, on peut démontrer que les fréquences propres ne sont plus harmoniques mais de la forme :

$$f_n = nf_0(1+\epsilon)\ \ avec\ \ \epsilon = Bn^2 - \frac{Z_c Im(Y(L))}{\pi n}$$

Où Y(L) est l'admittance mécanique, Z_c est l'impédance caractéristique et B une constante propre à la corde.

La détermination précise des fréquences de ces composantes se fait donc par récurence, en tenant compte de l'erreur précédente. Ainsi, pour déterminer la composante de rang n+1, on cherche un partiel autour de la fréquence $2f_n - f_{n-1}$.

4.2 Detection du fondamental

Nous venons de démontrer que les sons produits par les instruments à corde libre étaient pseudo-harmonique ; on peut donc déterminer facilement le fondamental d'un son analysé à partir de la transformée de Fourier en calculant le *produit spectral*.

Pour celà, il suffit de calculer le produit des amplitudes à des fréquences données, ces fréquences étant multiples entre elles, autrement : des fréquences harmoniques d'un fondamental arbitraire. Nous considérons ici que la largeur des pics sur la transformée de Fourier est supérieure au décalage dû à l'inharmonicité.

Soit X le spectre du signal N la profondeur de calcul et f la fréquence fondamentale arbitraire :

$$P(f) = \prod_{n=1}^{N} |X(e^{j2\pi nf})|^2$$

En faisant glisser ce fondamental sur une plage fréquencielle prédéterminée, il suffit de chercher la fréquence donnant le produit maximum pour en déduire le fondamental effectif du signal analysé.

Exemple, pour une note de piano de fondamental 1000 Hz :

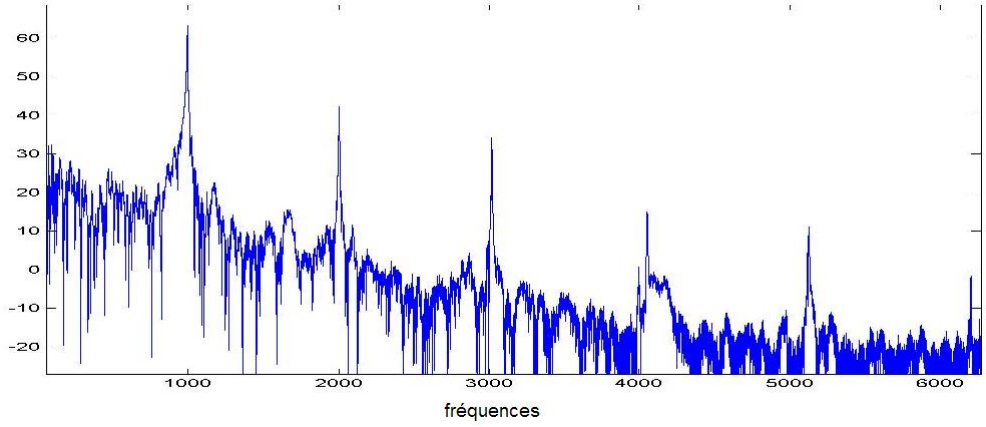

Fig 7.2 Spectre du signal

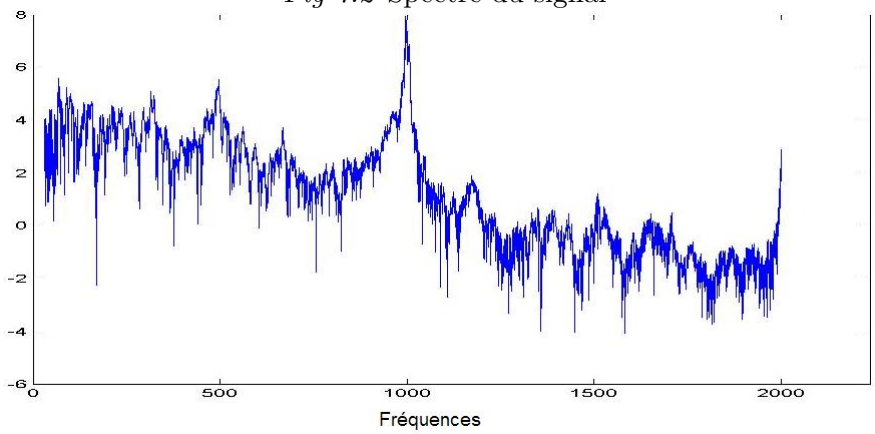

Fig 7.3 Résultat pour un fondamental recherché entre 50 et 2000 Hz. Pour N=5.

Malheureusement, nous ne pouvons pas avoir d'a priori sur le signal analysé. Il est donc nécéssaire de rechercher le fondamental sur un très large plage de fréquence. L'estimation du

fondamentale peut donc être faussée, c'est pourquoi il est important d'ajuster finement la valeur N qui donne la profondeur du produit spectral.

C'est la valeur 5 qui donne les meilleurs résultats, mais quelques erreurs tout de même. Ansi, il serait bon de prévoir un ajustement de cette valeur directement depuis l'interface.

Le script correspondant est disponible en annexes.

4.3 Calcul de l'inharmonicité

Dans l'optique d'une aide à la facture ou à la modélisation des instruments à cordes libres (comme pour le piano, que nous verrons par la suite), il est intéressant d'utiliser les résultats de l'analyse pour déterminer le coefficient d'inharminicité de la corde correspondante.

Pour cela, on représente les fréquences des harmoniques determinées divisées par la fréquence théorique de l'harmonique, en fonction du rang au carré du l'harmonique, soit :
$\frac{\hat{f}}{f}$ en fonction de n^2 où $f = n.f0$ avec $f0$ fréquence fondamentale de l'harmonique et n le rang de l'harmonique.

Pour déterminer la fréquence de l'harmonique de rang n, il faut tenir compte des harmoniques de rang inféreur. On cherche donc l'harmonique de rang n+1 sur une plage de fréquences comprise entre $[2f_{n-1} - f_n - \Delta_f; 2f_{n-1} - f_n + \Delta_f]$.

Enfin, pour déterminer le coefficient d'inharmonicité à partir des points, on utilise la fonction `polyfit()` qui va nous donner les coefficients C de la fonction $y = C(1)x + C(2)$ et donc nous donner le coefficient d'inharmonicité $C(1)$.

Le script correspondant est disponible en annexes.

5 Interface homme-machine

5.1 Interface de départ

La base de départ du stage est une interface Matlab^{TM}originalement développée lors du stage de DEA de Vincent Julien **[2]** et destinée à l'aide à la facture de cloches. Elle permet de charger un fichier audio (l'acquisition d'une note par exemple), de représenter les données sous différentes formes ainsi que de procéder à l'analyse et à la présentation des résultats.

L'atout principal de cette interface réside dans la visualisation des signaux : on visualise la progression de chaque partiel[5] dans le temps, ainsi que son énergie (visualisation en 3 dimensions). La visualisation est nettement plus claire par rapport, notamment, au spectrogramme, car l'analyse préalable tire uniquement les informations utiles du signal et permet un résolution à la fois spatiale et temporelle accrue. Les méthodes de Fourier n'offrant pas assez de résolution, ce programme s'appuie sur les méthodes haute résolution qui permettent de s'affranchir de la limite en rapport avec l'horizon d'observation propre à la transformé de Fourier. Les instruments à cordes libres possédant le plus souvent des modes multiples à des fréquences très proches, les méthodes hautes résolutions permettent d'arriver à les séparer.

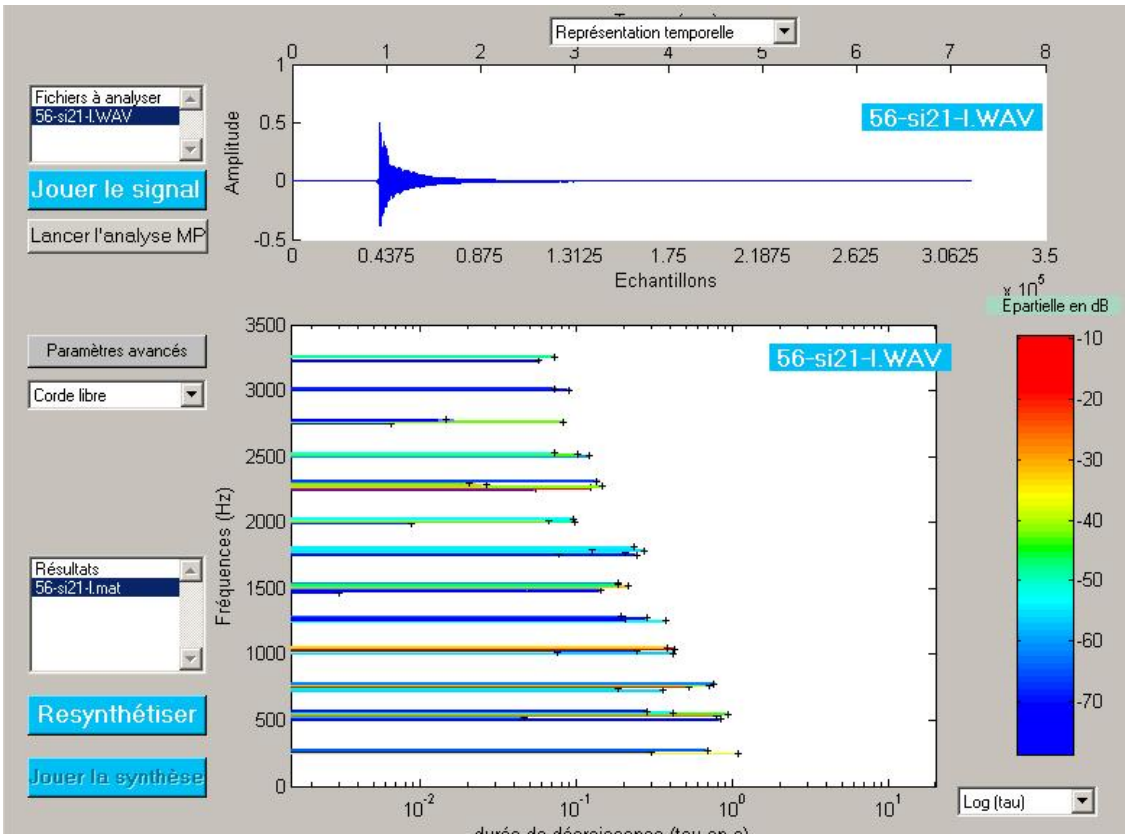

Fig2.1 Interface de départ

5.2 Fonctionnement sommaire du programme

Le programme est destiné à l'analyse de sons issus d'instruments à cordes libres. Il utilise la méthode d'analyse haute résolution Matrix Pencil **[1]** qui n'a pas été étudiée pendant ce stage (car remplacé par la méthode ESPRIT) pour déterminer les partiels composant la note. Cette méthode de calcul étant particulièrement gourmande (algorithme en $O(n^3)$, un système de

[5] Un partiel est une combinaison d'une fréquence, amplitude et d'une décroissance : $A_n e(-\alpha_n t) sin(2\pi f_n t)$

filtrage du signal est réalisé autour de pics préalablement déterminés par critère de forme sur la transformée de Fourier du signal. Il est possible de choisir entre trois méthodes de détermination des pics :

- Une méthode se basant sur le modèle de la corde libre : chaque pic est déterminé de manière récursive en tenant compte des harmoniques précédents. Cette méthode permet aussi d'atteindre l'inharmonicité de la corde jouée.
- Une méthode se basant sur le modèle d'un son percussif, programmée par Bertrand David, qui recherche les pics les plus énergétiques.
- Une dernière méthode sans a priori sur le signal, qui sélectionne donc les pics les plus énergétiques à l'aide d'une analyse multirésolution.

Une fois les pics déterminés, chacun d'entre eux est isolé par filtrage. Le signal obtenu est décimé afin de réduire au minimum le nombre de points sur lesquels est appliqué l'algorithme Matrix Pencil.

5.3 Architecture du programme

Malheureusement, le programme était inutilisable dans la forme dans laquelle il m'a été présenté, en particulier pour des raisons de portabilité d'un ordinateur à l'autre ainsi que sa robustesse. Il a donc été nécéssaire, dans un premier temps, de determiner précisément l'architecture du programme ; à savoir :
L'importance, pour chaque fonction, des variables globales.
Les interactions entre l'interface et les fonctions (rôles des boutons, événements)

5.3.1 Principe d'une GUI Matlab

L'outil GUI[6] Matlab^{TM} offre la possibilité de programmer des interfaces utilisateur dont le developpement est relativement simpliste. Chaque objet (boutons, graphiques, listes) est défini par un certains nombre de fonction et de propriétés.

Les fonctions sont utilisées pour exécuter la construction de l'objet, ainsi que les évenements le concernant (appui sur un bouton, choix dans une liste...). La fonction la plus intéressante est la fonction de *Callback* qui est appelée lors d'une action de l'utilisateur sur l'objet.

Les propriétés de l'objet sont stockées dans une structure de données. On y accède avec les fonctions `set()` et `get()`

.

Afin de passer toutes ces propriétés de fonction en fonction, mais aussi de pouvoir définir et stocker des variables globales, une super structure appelée *handles* est utilisée dans le programme.

Ainsi pour accéder aux bornes de l'axe X de la fenêtre graphique appelée graph_resultats il faudra appeler la fonction : get(handles.graph_resultats,'Xlim')

L'intégralité de la structure du programme est disponible en annexes.

5.4 Debuggage du programme

5.4.1 Révision de la structure de sauvegarde du fichier

Au cours des différents appels de fonctions, un certain nombre de données sont chargées à partir du fichier de sauvegarde des résultats. La fonction de chargement du fichier n'était pas

[6]Graphical User Interface : Interface Graphique

assez robuste (incompatibilité avec les chemins de fichiers comportant des caractères spéciaux). De plus, il n'était absolument pas possible de changer le chemin de sauvegarde, ce qui posait de gros problèmes lors d'un accès en lecture seule au fichier d'analyse. Finalement, lorsque le fichier de résultats éxistait déjà, les données étaient cumulées et menaient donc à des résultats aberrants.

Un nouveau bouton a été rajouté, il permet via une fenètre pop-up, de selectionner le dossier de destination du fichier résultat.

La procédure de sauvegarde a été modifiée : si le fichier de sauvegarde par défaut existe, une série de pop-up oriente l'utilisation vers l'éventuel écrasement du fichier, ou un renommage du futur fichier résultat.

La fonction de sauvegarde a été sécurisée pour garantir la prise en compte de tous les caractères spéciaux qui peuvent être échappés par Matlab^{TM} en utilisant l'apostrophe.

La gestion des fichiers résultats via le menu de sélection a été modifiée pour qu'elle prenne en compte les multiples chemins possibles. Cependant un bug persiste encore : si deux fichiers du même nom sont chargés à partir de deux repertoires différents, il est impossible d'accéder au premier fichier : seul le deuxième répertoire est gardé en mémoire. Pour contourner le bug, il suffit de renommer les fichiers résultats quand on change de répertoire.

5.4.2 Bugs d'affichage

Afin d'être correctement affichées dans la fenêtre de résultats, les données devaient etre triées en ordre croissant. L'algorithme d'affichage a été modifié pour être plus stable vis-à-vis de données non triées.
L'affichage des fréquences estimées sur la transformée de Fourier originale correspondait toujours au dernier signal analysé et non au fichier de résultat couramment affiché : ceci a été modifié.
La synthése, de la même manière, n'était pas possible lors du chargement seul du fichier résultat. Il adonc été décidé de sauvegarder certaines informations supplémentaires dans le fichier résultat pour pouvoir se dissocier totalement du fichier source.

A tous ces problèmes, il y a bien sûr d'autres problèmes mineurs qui ont été résolus afin de rendre l'application plus robuste, notamment vis à vis des erreurs de manipulation. (mais un travail plus important reste à faire).

5.5 Modifications apportées au programme

En sus de l'implémentation d'une nouvelle méthode résolution, il a aussi été nécéssaire de revoir l'interface graphique du programme et notamment d'ajouter de nouveaux moyens de visualisation et de comparaison pour comprendre et valider les résultats.

5.5.1 Paramètres avancés

Pour chaque méthode d'analyse, une fenètre de paramètres avancés a été ajoutée. Ainsi, on peut regler au mieux chaque analyse en rapport avec le fichier analysé :

- Le nombre de pics à rechercher
- Le nombre de partiels retenus
- La largeur de la fenetre d'estimation
- Ou encore, pour la méthode ESPRIT la méthode de préaccentuation ainsi que le nombre de pôles estimés.

5.5.2 Visualisation avancée des partiels et des résultats

Afin d'atteindre plus facilement les valeurs numériques des résultats de l'analyse, un bouton de visualisation a été ajouté. Il ajoute, à côté de chaque partiel visible dans la fenètre de résultat, les valeurs numériques des fréquences, amortissements,amplitudes, phases et énergies estimées. De plus, pour faciliter la visualisation des résultats, une entrée a été ajoutée pour n'afficher que les partiels dont l'énergie est supérieure à la valeur indiquée ; et le fond de la fenêtre de visualisation a été coloré de la couleur des partiels d'énergie la plus basse (bleu marine).

5.5.3 Fenetre synthèse

Pour s'assurer de la pertinence de la resynthèse du signal à partir des résultats de l'analyse, une fenêtre supplémentaire a été ajoutée à l'interface de départ. Il est donc à présent possible de visualiser le signal resynthétisé dans le domaine temporel, fréquenciel ; mais aussi de visualiser l'erreur d'estimation par les moindres carrés.

5.6 Interface finale

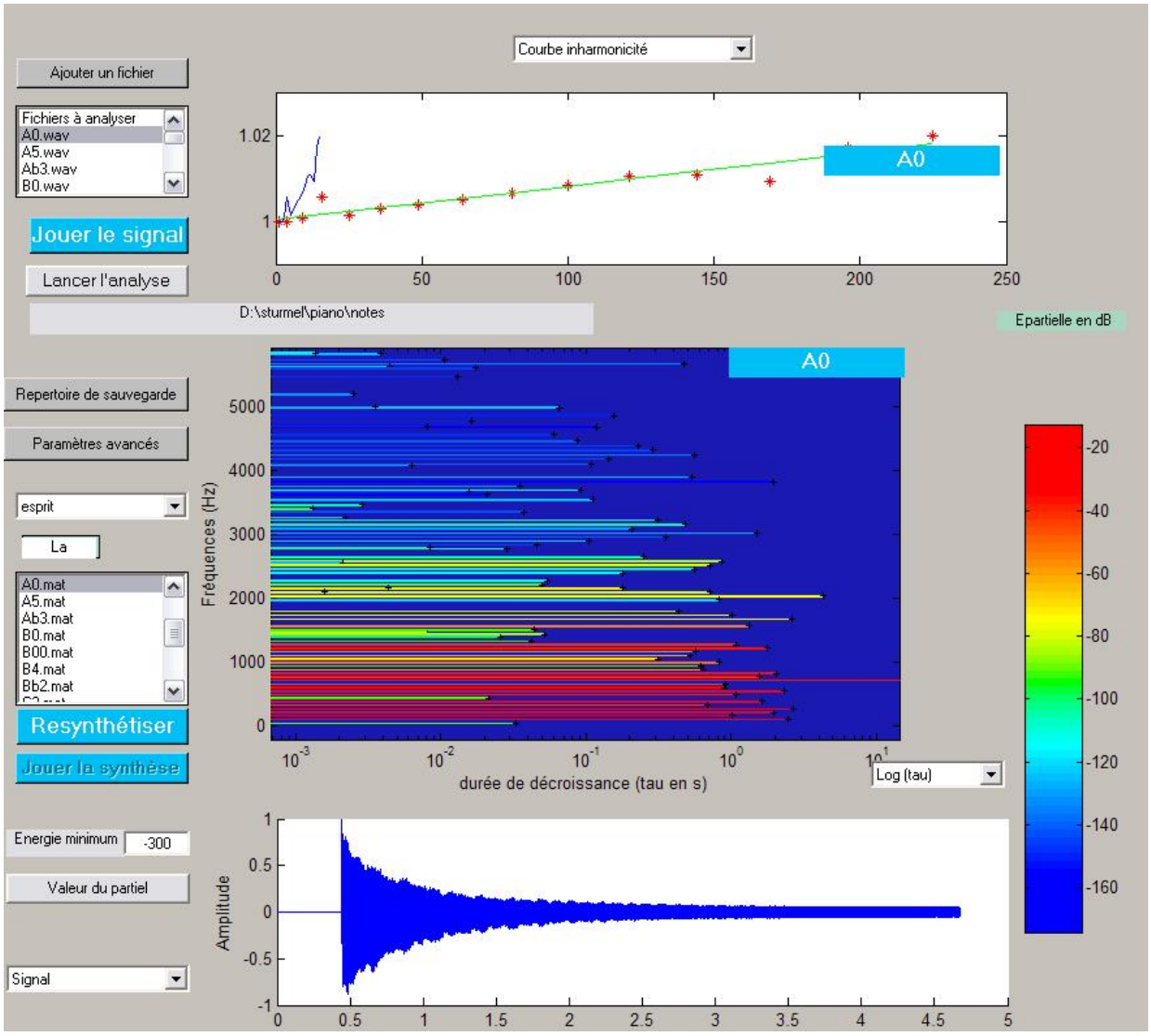

Fig 4.1 Interface finale du programme.

6 Nouvelles voies à explorer

Après un peu plus de deux mois de travail, l'interface et le programme sont maintenant assez robustes pour des analyses massives. On pourrait encore envisager beaucoup de fonctionalités supplémentaires comme du suivi de partiels, ou une visualisation en hacherogramme notamment.

6.1 Validation du programme : modélisation de l'inharmonicité d'un piano

Pour valider une utilisation massive du programme, il a été question de savoir si on pouvait modéliser fidèlement l'inharmonicité des cordes de pianos (dans le but d'optimiser les procédés de synthèse de l'instrument). Notamment en représentant le coefficient d'inharmonicité en fonction du rang de la note. Le programme a donc été utilisé pour analyser une quinzaine d'échantillons de piano, et construire la courbe suivante qui représente le logarithme de l'inharmonicité du logarithme du fondamental.

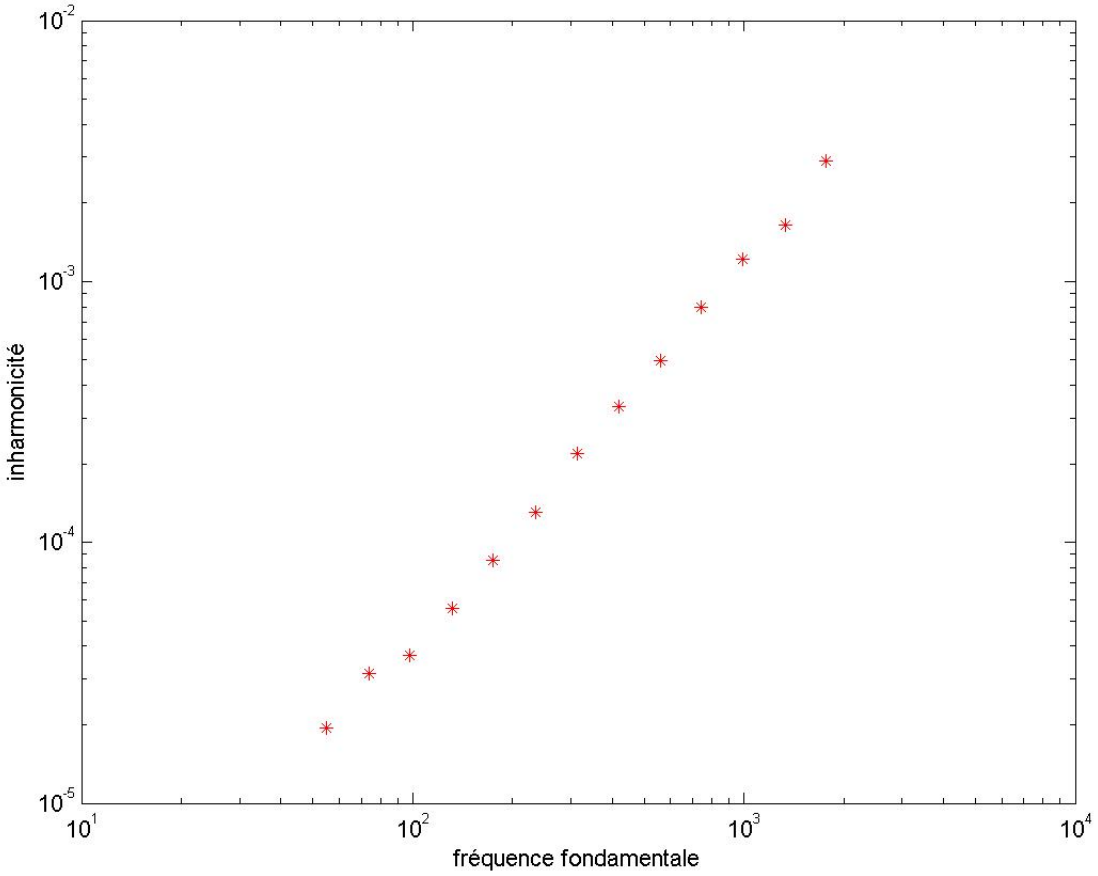

Fig 5.1 Inharmonicité d'un piano.

Les résultats donnés pour les notes du bas du spectre (octaves -1 et 0) n'ont pas été retenus car :

- la structure des cordes pour ces notes n'est pas la même que pour les suivantes, donc les constantes mécaniques (et a fotciori les constantes d'inharmonicité) changent.
- la détermination du coefficient d'inharmonicité était corrompue par une mauvaise estimation du fondamental (le problème a été évoqué auparavant)

La courbe obtenue est modèlisable par une droite, le logarithme du fondamental est équivalent au numéro de la note correspondante. Et on constate que l'inharmonicité est une fonction d'une puissance de la fréquence du fondamental de la note. Ce résultat, qui reprend un étude déjà réalisée par le passé, permet de valider le fonctionnement de l'interface. Mais permet aussi de déterminer avec précision le modèle de l'inharmonicité du piano qui pourrait, à terme, permettre de se passer de l'utilisation de banques de sons pour cet instrument comme c'est actuellement l'usage en production musicale.

7 Conclusion

7.1 Aboutissement du programme

L'interface est maintenant robuste et utilisable par une personne n'ayant pas une grande expérience de l'environement Matlab^{TM} ni de l'utilisation des GUI.

L'implémentation de la méthode esprit s'est révélée concluante et globalement plus rapide que les méthode déjà existantes, tout en étant raltivement plus simple au niveau de l'algorithme.

Les différents mode de visualisation ajoutés permettent de comprendre plus facilement les résultats obtenus et de vérifier simplement leur fidelité vis à vis du signal de départ par la comparaison avec le sepctre du signal resynthétisé.

La détermination de l'inharmonicité a été éprouvée sur un exemple concret.

7.2 Diffusion du programme

Ce programme peut donc s'avérer utile dans de nombreux domaines qui couvrent autant la lutherie que la synthèse ou la recherche et la modélisation acoustique. Il serait donc intéréssant de figer une version et de la programmer en C/C++ en utilisant la bilbiothèque mathématique libre GSL[7] afin de pouvoir la diffuser librement.

De plus, le gain en puissance du au passage du language Matlab^{TM} au language C diviserait le temps de calcul par deux ou trois selon leur complexité.

7.3 Bénéfice personnel

En plus de la manipulation des GUI de Matlab^{TM} , ce stage m'aura permis de mettre en application dans le domaine du traitement des signaux musicaux une grande partie du cours de signal et image de Maitrise et Magistère EEA. J'ai pu notamment me rendre compte qu'il n'est pas facile de concilier la théorie à la pratique, et que les solutions recherchées sont souvent plus simple que ce à quoi on peut s'attendre.

Je tiens à remercier M. Bertrand DAVID pour m'avoir permis de travailler sur ce projet, ainsi que tout le personnel du département TSI de l'ENST.

[7] GNU Scientific Library : www.gnu.org/software/gsl/

8 Références bibliographiques

[1] : Y. HUA, T. SARKAR, **Matrix Pencil Method for Estimating Parameters of Eponentialy Damped/Undamped Sinusoid in Noise**, *IEEE Mai 1990.*

[2] : Vincent JULIEN, **Estimation et représentation des signaux musicaux dans la perspective d'une aide à la facture**, *Mémoire de DEA ATIAM 2001-2002.*

[3] : Bertrand DAVID, **Caractéristiques acoustiques de structures vibrantes par mise en atmosphère raréfiée**, *thèse de doctorat 1999.*

[4] : B. KOZLOV, S. PUONG, **Acoustique de la guitare**, *Enseignement d'approfondissement acoustique X2001.*

[5] : Pablo QUEIXALOS, **Analyse acoustique d'un Clavecin historique**, *Rapport de stage IUT GEII Juin 2004.*

[6] : Roland BADEAU, **Estimation de sinusoïdes**, *Cours de la brique TSMAP ENST.*

[7] : H. MARINCHIO, Y. RENAULT, N. STURMEL, **Vocoder à synthèse vocale**, *Rapport de TER Avril 2004.*

9 Annexes

Sommaire

9.1 Calcul de la matrice de covariance

Le calcul de la matrice de covariance en utilisant l'estimateur défini ci avant peut être très grourmand en temps si son écriture n'est pas optimisé. Un MEX-file a donc été écrit et comilé afin d'optimiser au mieux le calcul, et ainsi de gagner un maximum de temps.

On somme les matrices obtenues par le produit $\mathbf{S}(t)\mathbf{S}(t)^H$ où $\mathbf{S} = [s(t), \ldots, s(t+n-1)]$. Mais cette analyse de la matrice de covariance n'est pas suffisante, il faut remarquer que les éléments de la matrice sont :

- Symétriques : ainsi, pour une matrice carré d'ordre n, on ne calculera que $\frac{n^2}{2}$ éléments.
- Se suivent selon une diagonale selon la formule : $C(i,j) = C(i+1, j+1) + s(i)s(i-k)$ avec $C(n,j) = \sum_{i=n}^{N} s(i)s(i-k)$ avec k dépendant de la diagonale en question.

Ainsi il est possible de construire la matrice, non pas en exécutant $L.n^2$ multiplications, mais $L.n$. L'algorithme en O(n) obtenu, conjugué à l'efficacité de la compilation en C permet de gagner un temps précieux. On passe par exemple, pour le calcul d'une matrice de covariance d'ordre 200 d'un temps de calcul de 107.25 secondes en utilisant le language matlab, à un temps de 3,95 secondes avec la fonction ci après.

```
#include <mex.h>
#include <stdio.h>
/*
 *                          COVARIANCE version 3.0-rc1
 *
 *  comme le calcul sous forme matricielle est assez gigantesque, on décide de le
 *  faire en utilisant les Mex-files. Cet algorithme est la troisième optimisation
 *  qui tient compte toute la structure de la matrice de covariance.
 *  Documentation sur les MEX-files : http://www.mathworks.com/support/tech-notes/1600/1605.html
 */

void mexFunction(int nlhs, mxArray *plhs[ ],int nrhs, const mxArray *prhs[ ]) {
/*
 * Les variables sont affectées comme suis :
 *         - plhs[0] : matrice de sortie, de taille n*n
 *         - prhs[0] : signal d'entrée
 *         - prhs[1] : nombre entier, taille du signal d'entrée
 *         - prhs[2] : nombre n, taille des blocs
 */

        // Déclaration des variables
        int i,j,k ;
        double * signal;
        int taille_signal,taille_block;
        double *tsignal,*block, *Out;
        double temp;

        // Initialisation des variable (MEX-spécifique)
        tsignal=mxGetPr(prhs[1]);       // On recoit le pointeur de l'argument 2
        block=mxGetPr(prhs[2]);         // On recoit le pointeur de l'argument 3
        taille_signal=(int)*tsignal;    // Conversion en entier (une ligne pour rien)
        taille_block=(int)*block;       // ^^^^^^^^^^^^^^^^^^ îdem ^^^^^^^^^^^^^^^^^^
        plhs[0] = mxCreateDoubleMatrix(taille_block, taille_block, mxREAL);
                                  // ^^ création de la matrice de sortie
        Out=mxGetPr(plhs[0]);           // Conversion des variables
        signal=mxGetPr(prhs[0]);        //  ^^^^^^^ idem ^^^^^^^^
```

```
        // Initialisation de la matrice solution (que des zéros)
        for(i=0;i<taille_block;i++)
                for(j=0;j<taille_block;j++)
                    Out[i*taille_block+j]=0;
        // Algorithme en lui même :
        for(k=0;k<taille_block;k++) {
                temp=0;
                for(i=taille_signal;i>=taille_block-k;i--)
                        if(i-k>0)
                            temp+=signal[i]*signal[i-k];
                for(i;i>=0;i--) {
                        Out[(i)*taille_block+i+k]=temp;
                        if(i-k > 0)
                            temp+=signal[i]*signal[i-k];
                }
}
/*
 * Symétricité de la matrice
 */
for(i=0;i<taille_block;i++)
            for(j=0;j<=i;j++)  {
                Out[i*taille_block+j]=Out[j*taille_block+i];
        }
}
```

9.2 Structure initiale du programme

L'interface comporte entre autres les fonctionnalités suivantes :
- Une fenêtre "signal" qui permet de visualiser le signal. Un menu déroulant situé au dessus de la fenêtre permet de choisir entre une représentation temporelle, fréquence, ou un spectrogramme notamment.
- Deux boîtes ou *list boxes*, qui listent les fichiers sur lesquels agit le programme : les fichiers sources et les fichiers résultats.
- Une partie analyse qui consiste en un bouton de lancement et un menu déroulant pour choisir une méthode d'analyse.
- Une dernière fenêtre qui permet la visualisation des résultats de l'analyse sous différentes formes via un menu déroulant sur le coté.

9.2.1 Gestion des événements

Avant même de s'intéresser au fonctionnement intime du programme, il est tout d'abord nécessaire de règler tous les problèmes liés à l'interface proprement dits. En particulier, il peut s'avérer intéressant de visualiser le diagramme et la succession des événements lors d'une action sur l'interface :

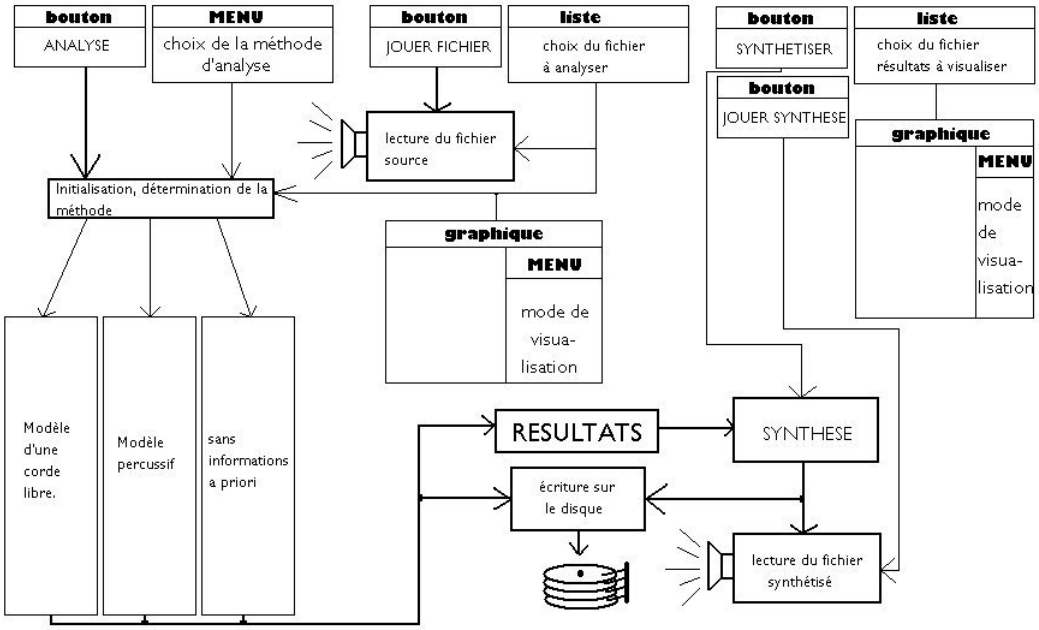

Fig2.2 Diagramme des évènements

9.2.2 Organigramme de l'analyse

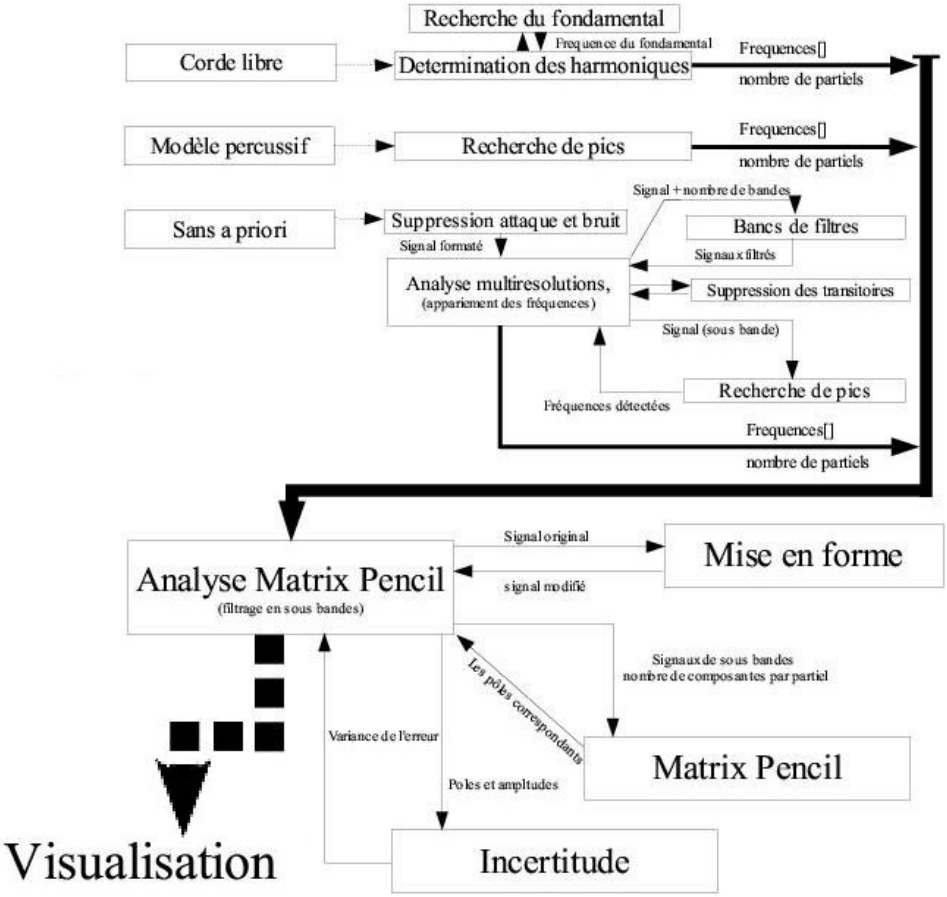

Fig2.3 Organigramme de l'analyse

9.2.3 Listing des fonctions

Enfin, il est aussi nécéssaire de répertorier toutes les fonctions importantes appelées lors de l'analyse d'un fichier.

Fonction initmp7(...)
- Création d'un répertoire pour les résultats.
- Changement de répertoire vers le répertoire 'résultat'.
- Création du fichier de résultats.
- Initialisation des variables contenues dans le fichier de résultats
- Sauvergarde des variables, fin de la fonction.

Fonction harmoniques(s,fichres)
- Formatage du signal (troncature du début et de la fin)
- Recherche du fondamental par appel de la fonction (*Fondamental3(s1,FS)*)
- Initialisation des paramètres pour la recherche d'harmoniques
- Recherche des harmoniques et des Δ_f.

Fonction harmoniques(s,fichres)
- Formatage du signal (troncature du début et de la fin)

- Recherche du fondamental par appel de la fonction (*Fondamental3(s1,FS)*)
- Initialisation des paramètres pour la recherche d'harmoniques
- Recherche des harmoniques et des Δ_f.

Fonction fondamental3(s,Fs,K)
- Determination par produit spectral
- Elaboration de la plage des fréquences de recherche
- Determination du produit specral pour chaque fréquence
- Recherche du maximum
- Renvoi de la fréquence associée au maximum

Fonction pics2(...)
- Recherche des pics par critère de forme sur la transformée de Fourier

Fonction remove_onset2(s)
- Calcul de l'enveloppe.
- Suppression de l'attaque du signal.
- Suppression de la fin du signal si on trouve qu'à partir d'un moment la décroissance arrive en dessou de -20 dB.
- Conversion du signal de sortie 'out' en vecteur colonne.

Fonction multiresolution_tracking(...)
- Synthèse du banc de filtres, et filtrage du signal : *banc_de_filtre(flipud(s),N)*
- Suppression des transitoires dans chaque sous bande avec utilisation de la fonction *transient_lgth()*.
- calcul des fréquences de coupure réduites pour chaque bande.
- Découpage en trames
- Recherche de pics avec *find_freq()*.
- Appariement des pics et sauvegarde des résultats.
- Synthèse de chaque filtre d'ordre 128
- Filtre du signal, et affectation aux variables de sorties

Fonction transient_lgth(...)
- Donne la longueur des transitoires pour chaque filtre utilisé

Fonction find_freq(...)
- Initialisation des paramètres utilisés.
- Calcul du spectre moyenné.
- Determine le spectre de la fenêtre d'analyse, puis garde uniquement son le lobe principal.
- Calcul et normalisation de la corrélation entre les deux spectres.
- Tri : les valeurs gardées ne sont que celles supérieures au seuil.
- Recherche des maximums locaux et des fréquences réduites associées

Fonction analyse_mp33(...)
- Ouverture fichier resultats
- Suppression de l'attaque du signal : remove_onset(s)
- Pour chaque partiel :
  - transposition autour de la fréquence nulle (démodulation),

- filrage Passe-bas,
- décimation
- troncature du bruit après la fin de la décroissance
- analyse MP par la fonction : MatrixPencil(sdec,composantes)
- moindres carrés (estimation amplitude et phase des composantes du partiel)
- calcul de l'erreur (signal synth % signal original)
- calcul d'incertitudes (approche de perturbation -cf.HUA-) : incmp(...)
- sauvegarde des résultats

Fonction remove_onset(s)

- Suppression de l'attaque par détection d'enveloppe (relativement semblable à la fonction remove_onset2(...)

Fonction MatrixPencil(sdec,composantes)

- Application directe de l'algorithme Matrix Pencil. La fonction renvoie les pôles solutions

Fonction incmp(sdec,composantes)

- Calcul de l'incertitude sur les pôles retournés par Matrix Pencil.

Il existe bien sûr d'autres fonctions, qui s'occupent notamment du formatage et de l'affichage des données. Leur fonctinonnement est relativement trivial ; nous en avons vu quelques unes au cours des modifications apportées au programme.

9.3 Scripts Matlab™

9.3.1 Interface pour la méthode esprit

```
function w=methode_esprit4(s,taille_fenetre,taille_sous_espace,fichres)
% --------------------------------------------------------------------------
%
%       w=methode_esprit4(s,taille_fenetre,taille_sous_espace,fichres)
%
%  Fonction d'application de la méthode esprit pour supplanter Matrix
%  Pencil. Ici on utilise la technique de loupe spectrale.
%
%--------------------------------------------------------------------------

% NS 9/6/2004

my_global;                        % Variables globales
eval(['load ',39,fichres,39]);  % Chargement du fichier resultat
Fe=handles.Fs/nombre_bandes;

% |- - - - - - - - - - - - - -|
% |---- algorithme ESPRIT ----|
% |- - - - - - - - - - - - - -|
[signal,ampbis,phasebis,freqbis,amortbis,err_amp]=esprit12(s,taille_fenetre,taille_sous_espace,Fe)
freq=freqbis(:);
phase=phasebis(:);
amort=amortbis(:);
amp=ampbis(:);

w = waitbar(0.8,'formatage des resultats');

RES=[];
FR=freq(:);
npartiels = length(FR);

% Sauvegarde des résultats
RES(:,1)=FR;
RES(:,2)=zeros(length(FR),1);
RES(:,3)=handles.Fs/nombre_bandes*amort(:);
RES(:,4)=zeros(length(FR),1);
RES(:,5)=zeros(length(FR),1);
RES(:,6)=amp(:);
RES(:,7)=phase(:)
RES(:,8)=err_amp(:);

% On élimine les amplitude et les phases NaN
RES=RES((find(RES(:,6)<10^9)),:);
RES=RES((find(RES(:,7)<10^9)),:);

% On met à jour le vecteur FR utilisé pour l'affichage des points rouges
% sur la fft
FR=RES(:,1);

eval(['save ',39,fichres,39,' chems Fs Lsig indmax N_b SNR ',...
'npartiels FR nfaits df RES HISTO REStchar HISTOchar']);
```

```
toc
```

9.3.2 Préaccentuation

```
function [LPC,Sbis]=m_en_forme(S,Fe)
% Fonction de mise en forme du signal
% usage :   Sbis = M_EN_FORME(S,Fe)
%
%   - Detection de la partie utile
%   - préaccentuation

[Y,I]=max(S);
my_global;

% Decoupage de la partie utile du signal
I2=find(S>seuil);
I3=find(S==0);
[Y,I]=min(abs(I3-I(1)))
Ster=S(I3(I):end);

% Préactentuation du signal par blanchiment de spectre selon
% le paramètre coeff_lpc.
% Si supérieur à 1 :
%       --> application du fitre inverse de celui obtenu par LPC
% Si égal à 0 :
%       --> On ne fait rien
% Si égal à -1 :
%       --> Préaccentuation par blanchiment du bruit

if handles.coeff_lpc==-1
    % Utilisation d'un blanchiment de spectre capilo tracté, mais qui
    % marche pas mal.
    Sbis=blanchi(Ster,3,10);
    LPC=1;
elseif handles.coeff_lpc~=0
    LPC=lpc(Ster,handles.coeff_lpc)
    Sbis=filter(LPC,1,Ster);
else
    Sbis=Ster;
    LPC=1;
end

% normalisation du signal
Sbis=Sbis/max(abs(Sbis));
```

9.3.3 Méthode esprit

```
function [signal,amp,phase,freq,amort,Err_amp]=esprit(S,n,k,Fe)
% Determination des fréquences qui constituent S par la méthode haute
% résolution ESPRIT (Estimation of Signal Parameters via Rotational
% Invariance Techniques)
%
%   usage :   [signal,amp,freq,amort]=ESPRIT(S,n,K,Fe)
%
```

```
%             V  : vecteur des valeurs propres qui donennt les fréquences
%             recherchées
%             S  : Signal d'entrée
%             n  : ordre d'estimation de la covariance du signal S
%             K  : dimension de l'espace signal
%             Fe : fréquence d'échantillonnage du signal
%
% ------ la fonction réalise les étapes suivantes :
% Etape 1 : calcul de la covariance  (matrice carrée de dimension n)
% Etape 2 : determiner la matrice qui contient le vecteur propres de R
% Etape 3 : en déduire les matrice Wup et Wdown qui constituent les base du
%           sous espace signal (non ?)
% Etape 4 :
% Etape 5 :
% ------------------------------------------------------------------------

% Modifié le 08/06/2004 par NS, créé le 03/06/2004 par NS aussi :p
% ------------------------------------------------------------------------
% -------------- Initialisation : troncature du signal

% ---- on supprime le signal inférieur à -40 dB. Pour affiner la recherche
% ---- on utilise un filtrage médiant, pour supprimer les pics _très_
% ---- locaux du signal. ( très est sinonyme de : localisé sur 1 seul
% ---- échantilon )

my_global;

w = waitbar(0,'mise en forme');
[LPC,Sbis]=m_en_forme(S,Fe); % On en profite pour récuprer les coéfficients de la LPC
size(Sbis)
close(w);
w = waitbar(0.1,'matrice de covariance');

% ------------------------------------------------------------------------
% -------------- Etape 1 : calcul de la covariance

SS=covariance3(Sbis,max(size(Sbis)),n); % Calcul de la covariance optimisé en o(n)

close(w);w = waitbar(0.2,'Detremination du sous espace signal');

% -------------- Etape 2 : Determination du sous espace signal

[L,B,U]=svd(SS,0) ;
Us = U(:,1:k);
close(w);w = waitbar(0.4,'Calcul des poles');

% -------------- Etape 3 : invariance par rotation

U1 = Us(1:(n-1),:);
U2 = Us(2:n,:);

% -------------- Etape 4 : Determination des valeurs propres

Psi = ((U1'*U1)^-1)*(U1'*U2);
phi = eig(Psi);
```

```
phi(find(imag(phi)==0))=[]
close(w);w = waitbar(0.5,'Determination des amplitudes complexes');

% -------------- Etape 4bis : Methode des moindre carrés pour determiner
% --------------              amplitude et phase


% --- recherche des amplitudes et des phases (moindres carrés linéaires) ---

    % -- On formate les pole en ne gardant que les poles ayant une
    % signification physique :

phi=phi(find(abs(phi)<1));
Sbis=S(max(S):(min(max(S)+1.5*handles.Fs,end)));
N=length(Sbis);

disp('calcul des amplitudes');
Vdm = vandermonde(phi,N);
b = pinv(Vdm)*Sbis(:); % amplitudes complexes
Erreur=20*log10((Sbis(:)-Vdm*b)./(Sbis(:)));
b=b(:);
ampbis=abs(b);
phasebis=angle(b);

% --- On se débarrasse des poles abhérents

phi=flipud(sort(phi));
phi=phi(find(angle(phi)>0))
phi=phi(find(angle(phi)<pi))
f_estime = angle(phi)/(2*pi);
alpha = -log(abs(phi)) ;

% --- Calcul de l'energie du signal d'erreur :
%                        (d'après B. David)

disp('calcul de l''erreur sur l''amplitude');
        ss = Vdm*b; %signal synthétique généré avec les param issus de l'analyse
        Edb = sum(  abs(Sbis(:)-ss(:)) .^2) / sum(abs(Sbis(:)).^2 );
        Edb = 10*log10( Edb )*[1;zeros(length(f_estime)-1,1)]; % dimensionné pour remplir le fichier

% -------------- Etape 5 : mise en forme : On selectionne les
% --------------           partiels cohérents

close(w);w = waitbar(0.6,'elimination des poles abhérents');
freq=[];
amort=[];
amp=[];
phase=[];
Err_amp=[];
j=1;

if min(size(f_estime))>0
    for i=1:max(size(f_estime))
        if f_estime(i)>0 && f_estime(i)<0.5  && alpha(i)>0
            freq(j)=f_estime(i)*Fe;
```

```
            amort(j)=alpha(i);
            amp(j)=ampbis(i);
            phase(j)=phasebis(i);
            Err_amp(j)=Edb(i);
            j=j+1;
        end
    end
end


close(w);
handles.Erreur_amp=Erreur;

% --------------- fin du script
```

9.3.4 Calcul de l'inharmmonicité

```
freq = handles.freq;
        E=handles.E_partielle;
        Energie=0;
        E_temp=0;

        %%% Determination du fondamental
        Fondam=fondamental3(diff(handles.signalEntree),handles.Fs,5);
        n=[];
        harmos=[];
        for i=1:15
            Delta=10^9;
            pointeur=0;
            for f=1:length(freq)
                if length(harmos)>1
                    %%% Si on peut appliquer la récurence :
                    if n(end)==i-1 && n(end-1)==i-2
                        Delta_bis=abs(freq(f)-(2*harmos(end)-harmos(end-1)))
                    else
                        Delta_bis=abs(freq(f)-i*Fondam);
                    end
                else
                    Delta_bis=abs(freq(f)-i*Fondam);
                end

                    %%% Conditions pour qu'un partiel soit considéré comme potentiellement
                    %%% harmonique de la note.
                if  Delta>Delta_bis && Delta_bis<Fondam/4 && handles.amort(f)<100
                    pointeur=f;
                    Delta=Delta_bis;
                end
            end

            %%%% Si on a trouvé un partiel valide :
            if pointeur~=0
                n=[n;i];
                harmos=[harmos;freq(pointeur)];
                freq(pointeur)=[];
                Energie=E_temp ;
```

```
            end
        end

        %%%% Gestion de l'affichage, determination des coefficients, etc..
        harmos/Fondam
        axes(handles.graph_signal_entree);
        cla;
        reset(gca);
        plot(n.^2,harmos./(n*Fondam),'r*');
        hold on;
        plot(n,harmos./(n*Fondam));
        P=polyfit(n(:).^2,harmos(:)./(n*Fondam),1);
        plot(n.^2,polyval(P,n.^2),'g');
        hold off;
        msgbox({['Coefficient directeur : ',num2str(P(1)),''],['Valeur du fondamental : ',num2str(Fo
        handles.harmos=harmos;
        xlabel('n^2');
        ylabel('f/n');
        hold off;
```

9.3.5 Produit spectral

```
function fond = fondamental3(s,Fs,K)
% ---------------- Recherche du fondamental par produit spectral --------
%
%             usage :     fond = FONDAMENTAL3(s,Fs,K)
%
%           s : signal à analyser
%           Fs : fréquence d'échantillonnage de s
%           K : ordre de la recherche du produit spectral
%
%   On procède à la determination du fondamental par produit spectral : on
%   choisit arbitrairement une plage de fréquences qu'on considère comme
%   potentielement fondamentale. En fesant le produit des valeurs de la fft
%   aux fréquences multiples de chacune des fréquences de la plage
%   supposée ; il suffit de choisir la fréquence qui donne le maximum de
%   puissance : celle ci est le fondamental.
% ------------------------------------------------- NS 06/2004 --------
taille=2^nextpow2(max(size(s)));
Fmax=2000;
Fmin=30;
if K*Fmax>Fs/2
    Fmax=Fs/(2*K);
end
NFmax=ceil(Fmax*taille/Fs);
NFmin=ceil(Fmin*taille/Fs);

S=fft(s,taille);
Echant=repmat((1:K),[NFmax-NFmin+1 1]).*repmat((NFmin:NFmax)',[1 K]);
Echant=[(NFmin:NFmax)' Echant]; % Ici, on augmente l'effet du premier échantillon sur le produit spe
                                % de cette manière, on a moins de chance de choisir un sous harmoniq
                                % fondamental. (cas où le nombre d'harmoniques est inférieur à K).
                                % En contre partie, l'amplitude du fondamental doit être élevée, c'e
                                % généralement le cas.

PS=prod(S(Echant),2);                % Calcul le produit
```

```
frequences=(NFmin:NFmax)*Fs/taille;   % Axe des fréquences
[V,I]=max(PS);          % Recherche du maximum
fond=frequences(I);     % Identification de la fréquence associée
if fond<50     % Protection
    fond=0;
end
```