

## Projet SY16

---

Identification automatique de fichiers audio.

# Table des matières

<b>1</b>	<b>Bibliographie</b>	<b>4</b>
1.1	Modèles auto régressif . . . . .	4
1.1.1	Calcul des modèles AR . . . . .	4
1.1.2	Avantages/Défauts de cette approche . . . . .	5
1.1.3	Essais effectués . . . . .	5
1.2	Analyse rythmique . . . . .	6
1.2.1	Inconvénient de la méthode . . . . .	7
1.2.2	Intérêt de la méthode . . . . .	7
1.3	Estimation de la fréquence fondamentale . . . . .	8
1.3.1	Avantages/Inconvénients . . . . .	8
1.4	Estimation spectrale . . . . .	8
1.4.1	Avantages - Inconvénients . . . . .	8
<b>2</b>	<b>Mise en place de l'analyse automatique</b>	<b>10</b>
2.1	Synoptique générale . . . . .	10
2.2	Extraction d'une empreinte caractéristique . . . . .	10
2.2.1	Rééchantillonnage . . . . .	11
2.2.2	Spectrogramme . . . . .	11
2.2.3	Détection de la fréquence dominante . . . . .	12
2.3	Construction d'une base de données de musiques . . . . .	13
2.4	Reconnaissance d'une chanson dans la base de données . . . . .	14
2.4.1	Méthode . . . . .	14
2.4.2	Intercorrélation . . . . .	14
2.4.3	Ordre . . . . .	15
2.4.4	Extraction du maximum . . . . .	15
2.5	Tests de robustesse . . . . .	16
2.5.1	Ajout d'un bruit . . . . .	16
2.5.2	Test avec deux musiques de deux sources différentes . . . . .	17
2.6	Ouvertures quant à la complexité et l'algorithmique . . . . .	18
2.6.1	Complexité . . . . .	18
2.6.2	Optimisations potentielles . . . . .	19
<b>3</b>	<b>Annexes</b>	<b>20</b>
3.1	Code matlab du programme principal . . . . .	20
3.2	Code matlab de la fonction get_freq . . . . .	21
<b>4</b>	<b>Bibliographie</b>	<b>23</b>

# Table des figures

1.1	Vecteurs de coefficients AR . . . . .	5
1.2	Extraction de la rythmique d'un morceau . . . . .	7
1.3	Exemple de spectrogramme . . . . .	9
2.1	Spectrogramme d'une chanson . . . . .	12
2.2	Spectrogramme et hauteur de note . . . . .	13
2.3	Plusieurs extraits musicaux représentés . . . . .	14
2.4	Intercorrélations entre signaux de la bdd . . . . .	15
2.5	Comparaison signal bruité/ Non bruité . . . . .	17
2.6	Diverse intercorrelations dans la bdd . . . . .	18

# Liste des algorithmes

1	Construction du vecteur de coefficients AR . . . . .	5
2	Ajout de bruit . . . . .	6
3	Code matlab pour le filtrage . . . . .	7
4	Passe bas et rééchantillonnage sous matlab . . . . .	11
5	Spectrogramme sous matlab . . . . .	11
6	Extraction du maximum fréquentiel . . . . .	12
7	Extraction du maximum . . . . .	16
8	Ajout de bruit . . . . .	16

## Problématique - définition des objectifs

L'objectif de ce projet est de mettre en place un système de reconnaissance automatique de chansons par rapport à leur contenu. Il s'agit plus précisément de trouver un moyen d'effectuer une empreinte d'un fichier audio indépendamment de la fréquence d'échantillonnage, de la qualité, et du format de compression de cet dernier.

# Chapitre 1

## Bibliographie

Nous allons dans cette partie aborder les domaines que nous avons explorés avant de finalement choisir une solution viable.

### 1.1 Modèles auto régressif

Dans une première approche, nous nous sommes orientés vers des estimations de coefficients auto régressifs. L'idée était en effet d'estimer des coefficients AR sur des fenêtres de taille fixe.

#### 1.1.1 Calcul des modèles AR

Le modèle AR(p) est donné par l'équation :

$$X_t = \sum_{i=1}^p \varphi_i X_{t-i} + \varepsilon_t.$$

Il est basé sur les paramètres  $\varphi_i$  où  $i = 1, \dots, p$ . Ces paramètres peuvent être estimés en utilisant une méthode des moindres carrés ou bien par les équations de Yule-Walker

It is based on parameters where  $i = 1, \dots, p$ . Those parameters may be calculated using least squares regression or the Yule-Walker equations:

$$\gamma_m = \sum_{k=1}^p \varphi_k \gamma_{m-k} + \sigma_\varepsilon^2 \delta_m$$

où  $m = 0, \dots, p$ , ce qui donne  $p + 1$  equations.

$\gamma_m$  est l'autocorrélation de la fonction  $X$ ,  $\sigma_\varepsilon$  est l'écart type du bruit, et  $\delta_m$  est la fonction delta de Kronecker.

Etant donné que la dernière partie de l'équation est non nulle si et seulement si  $m=0$ , l'équation est habituellement résolue grâce à la matrice pour  $m>0$

$$\begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \vdots \end{bmatrix} = \begin{bmatrix} \gamma_0 & \gamma_{-1} & \gamma_{-2} & \dots \\ \gamma_1 & \gamma_0 & \gamma_{-1} & \dots \\ \gamma_2 & \gamma_1 & \gamma_0 & \dots \\ \dots & \dots & \dots & \dots \end{bmatrix} \begin{bmatrix} \varphi_1 \\ \varphi_2 \\ \varphi_3 \\ \vdots \end{bmatrix}$$

qui résout tous les  $\varphi$ .

Pour  $m=0$ , on obtient :

$$\gamma_0 = \sum_{k=1}^p \varphi_k \gamma_{-k} + \sigma_\varepsilon^2$$

ce qui nous permet de déterminer  $\sigma_\epsilon^2$ .

### 1.1.2 Avantages/Défauts de cette approche

L'avantage de cette méthode est incontestablement sa simplicité algorithmique par rapport aux méthodes temps/fréquence que nous aborderons plus loin dans cette bibliographie. En effet, les équations de Yule Walker permettent de trouver très simplement les coefficients AR sur des petites fenêtres.

A l'inverse, le gros défaut de cette méthode réside dans le fait qu'évidemment nos signaux ne sont pas stationnaires et que l'approximation de dire que notre signal sera stationnaire sur une fenêtre n'est bien évidemment pas vraie dans une écrasante majorité des cas. De plus, les tests montrent que cette méthode est très sensible aux bruits. On peut voir dans le graphique 1.1.3

### 1.1.3 Essais effectués

L'approche retenue pour les essais avec les coefficients AR était de former une empreinte avec les coefficients d'un extrait de musique d'une dizaine de secondes. Les coefficients ont été estimés grâce à la méthode de Yule Walker sur des fenêtres successives qui ne se chevauchent pas d'une durée de 100ms.

Les coefficients ainsi estimés forment un vecteur de dimension N qu'il est aisé de comparer avec une base de données d'autres coefficients. Il suffit en effet de calculer l'erreur quadratique et de choisir la chanson dans la base de données qui minimise cette erreur quadratique.

---

#### Algorithm 1 Construction du vecteur de coefficients AR

---

```

1 [sig,Fe] = lire_fichier('audio.wav');
2 Ws = 0.1; Taille de la fenetre en secondes
3 Ncoefs = 20 Nombre de coefficients AR à estimer
4 coefs = 0; Initialisation du vecteur de valeurs
5 for i=300:400
6   concater_coefs(coefs,estime_coefs_ar(sig(floor(i*Ws*Fe:(i+1)*Ws*Fe)),Ncoefs));
7 end

```

---

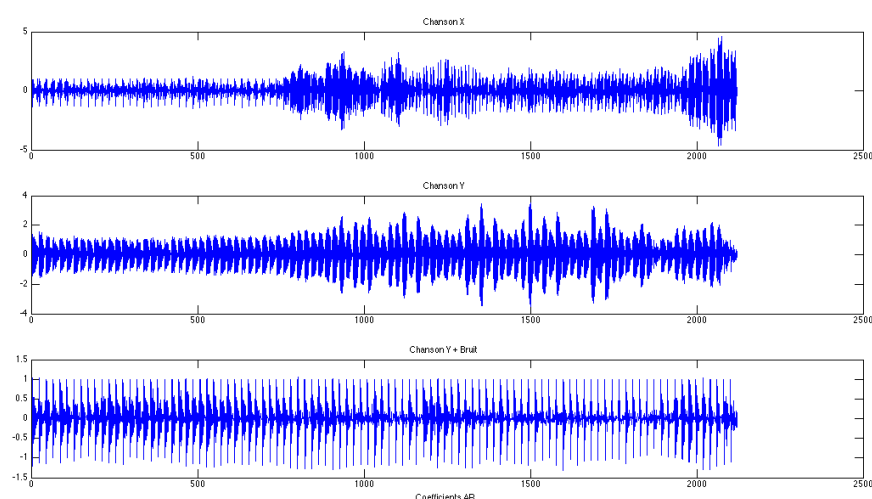


FIG. 1.1 – Vecteurs de coefficients AR

### Ajout d'un bruit

En vue de faire des tests de bon fonctionnement on ajoutera un bruit blanc centré à notre signal audio original. Modifier la puissance de ce bruit nous permet de tester la robustesse de notre estimation et ainsi de simuler une dégradation liée à une recompression ou autres.

---

#### Algorithm 2 Ajout de bruit

---

```

1 Pnoise = 0.01;
2 y = wgn(1,length(sig),0.001).*PNoise;
3 sig_bruit = sig + y'; % Ajout du bruit au signal original

```

---

### Comparaison des vecteurs

Une fois les vecteurs obtenus, il faut trouver un critère de comparaison entre ceux ci. Comme dit précédemment, il s'agit ici d'une façon assez simple de minimiser l'erreur quadratique de toutes les composantes du vecteur. Ainsi, si l'on a deux vecteurs  $X$  et  $Y$ , de même dimension  $1 \times n$  on aura :

$$e = \sum_{i=1}^n (X[n] - Y[n])^2 = (X - Y) \cdot (X - Y)^T$$

Une simple opération de tri peut nous permettre de sélectionner la valeur de  $e$  la plus petite et ainsi de nous donner la chanson la plus proche dans une base de données

## 1.2 Analyse rythmique

Un de nos axes de recherche était la reconnaissance d'instruments en contexte polyphonique, et plus précisément la reconnaissance de signaux de batterie en présence d'instruments harmoniques. La plupart des instruments de musique produisent des sons harmoniques, dits non-percussifs, à l'inverse des instruments de percussions, qui produisent des sons à fréquence élevée sur un intervalle de temps très court. Les instruments harmoniques ont de plus des harmoniques qui n'excèdent pas 6kHz. Il s'agit donc de détecter les impulsions provenant d'une batterie, en conservant dans le signal la plage de fréquences occupée par les sons de grosse caisse, caisse claire, tom, etc. La méthode utilisée pour extraire la source percussive du signal polyphonique est tout d'abord de réaliser un filtrage passe-haut à 7000Hz avec une fenêtre de Hanning.

A l'écoute du signal ainsi filtré on entend clairement tous les sons de la batterie, de ceux provenant de la caisse claire à ceux de la grosse caisse, d'intensité plus forte. Quelques sons harmoniques subsistent, il s'agit des amorces de note du signal de la voix humaine. L'intensité des signaux est également diminuée. Pour remédier à ce problème, on divise les valeurs du signal filtré par leur variance, afin d'en extraire les pics les plus courts sur l'échelle temporelle. En passant au logarithme les valeurs ainsi obtenues, on accentue encore le phénomène.

Les sons d'intensité courte et de fréquence élevée sont identifiés par les valeurs maximale sur ce nouveau signal. On peut alors réaliser le vecteur des instants pour lesquels on a une impulsion en provenance de l'instrument percussif.

**Algorithm 3** Code matlab pour le filtrage

---

```

1 %ESSAI GIMMICK A 22050HZ
2 %lecture du signal d'origine
3 clear all;
4 [y,Fe,Nbits]=wavread('gimmick-22050Hz.wav');
5 figure(1),plot(y), axis tight;
6 title('signal d''origine');
7
8 %lecture et affichage du signal filtré
9 load export;
10 figure(1),hold on,plot(gimmickF4.data,'g');
11 title('signal filtré(vert) comparé au signal d''origine');
12 %Ecriture du fichier .wav permettant d'écouter le signal filtré
13 wavwrite(gimmickF4.data,Fe,Nbits,'Filtrage4-gimmick.wav');

```

---

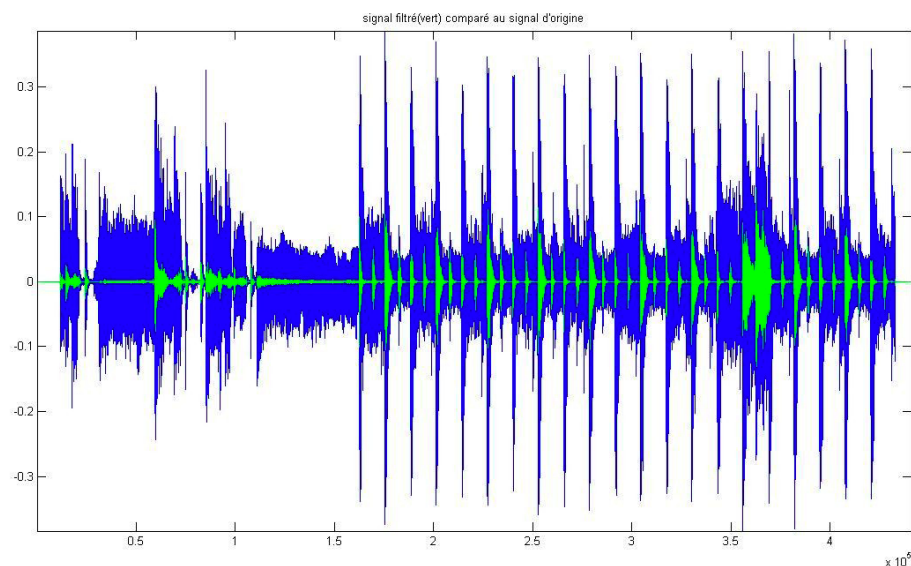


FIG. 1.2 – Extraction de la rythmique d'un morceau

**1.2.1 Inconvénient de la méthode**

Cette méthode ne tient pas compte du fait que la voix humaine peut également produire des harmoniques à fréquences élevées, tout particulièrement dans le cas où le chanteur est une femme. Elle fonctionnerait très bien pour une musique essentiellement instrumentale. La méthode est uniquement applicable dans les cas où un instrument percussif est présent dans la chanson. Le signal doit être finement échantillonné, de façon à prendre en compte les impulsions de batterie sur des intervalles de temps assez faibles, ce qui mène rapidement à des traitements lourds.

**1.2.2 Intérêt de la méthode**

Possibilité de produire une empreinte sur toute la durée de la chanson (vecteur  $1 * n$ )



## 1.3 Estimation de la fréquence fondamentale

Une grande partie de la bibliographie s'est faite à la bibliothèque de l'IRCAM<sup>1</sup>. Le but était de trouver les articles les plus pertinents au niveau de l'estimation de fréquence fondamentale.

D'après les chercheurs de l'Ircam, la démarche la plus pertinente dans notre cas aurait été d'utiliser un modèle quasiharmonique vu dans [MIREX7] utilisé dans de la forme suivante :

$$y[n] = \left\{ \sum_{m=1}^M \sum_{h_m=1}^{H_m} a_{m,h_m}[n] \cos((1 + \delta_{m,h_m})h_m\omega_m n + \phi_m[n]) \right\} + v[n]$$

où  $n$  est l'index temporel,  $M$  le nombre de sources,  $H_m$  le nombre d'harmoniques pour la  $m$ ème source,  $\omega_m$  représente la fondamentale de la source  $m$  et  $\phi_m[n]$  dénote la phase.  $v[n]$  représente le bruit. Ce dernier est considéré assez petit pour permettre de détecter la sinusoïde.

### 1.3.1 Avantages/Inconvénients

Après réflexion, cette méthode a l'avantage de bien coller à la réalité du modèle physique d'un fichier audio, mais présente le gros désavantage de devoir procéder à l'estimation de beaucoup de paramètres comme le nombre de sources, le nombre d'harmoniques etc.

De plus ce modèle nécessiterait un très long travail de précision pour être générique. En plus, ce modèle est sensible au bruit, ainsi qu'à certains formats de compression comme le MP3, qui détruisent certaines fréquences inaudibles, mais essentielles pour l'estimation de certaines harmoniques élevées.

## 1.4 Estimation spectrale

On peut faire ici une application directe du cours de SY16. Le spectrogramme est en effet la plus usuelle des représentations temps fréquence. On a en effet le temps en abscisse et la fréquence en ordonnée, ce qui permet littéralement de "voir les fréquences" grâce à un astucieux système de couleurs, on peut savoir directement où sont représentées les fréquences les plus intenses.

Le paramètre le plus sensible dans l'estimation spectrale est la taille de la fenêtre sur laquelle la transformée de Fourier sera effectuée. En effet ce paramètre devra être choisi de sorte que l'information privilégiée (temps ou fréquence) soit la plus pertinente.

### 1.4.1 Avantages - Inconvénients

L'avantage de cette technique est incontestablement qu'elle est proche de ce qu'entend l'oreille humaine. Elle va donc contenir les informations pertinentes d'un point de vue psycho-acoustique. De plus si l'on s'y prend bien, il est possible de s'affranchir des problèmes liés à l'encodage (surtout si l'on regarde les basses fréquences, étant donné que les formats comme le MP3 ou le ogg-vorbis compressent en supprimant les composantes fréquentielles inaudibles).

Le gros inconvénient lié à cette technique provient surtout de la complexité algorithmique liée au spectrogramme. Il est cependant possible de diminuer cette complexité en faisant un ré-échantillonnage du signal en amont. Ce que nous verrons dans la partie 2.

---

<sup>1</sup>IRCAM : Institut de Recherche et Coordination Acoustique/Musique

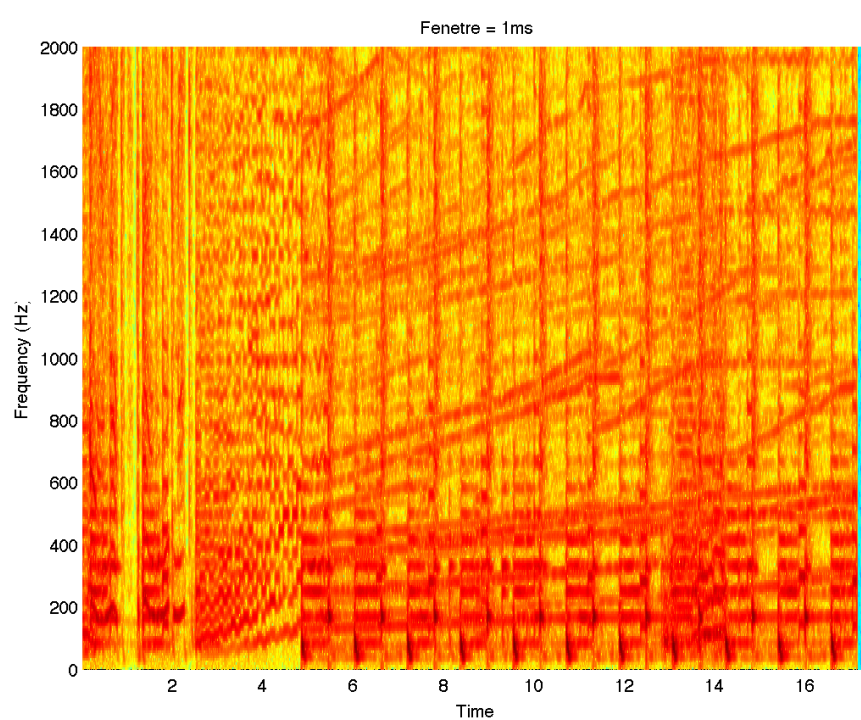


FIG. 1.3 – Exemple de spectrogramme

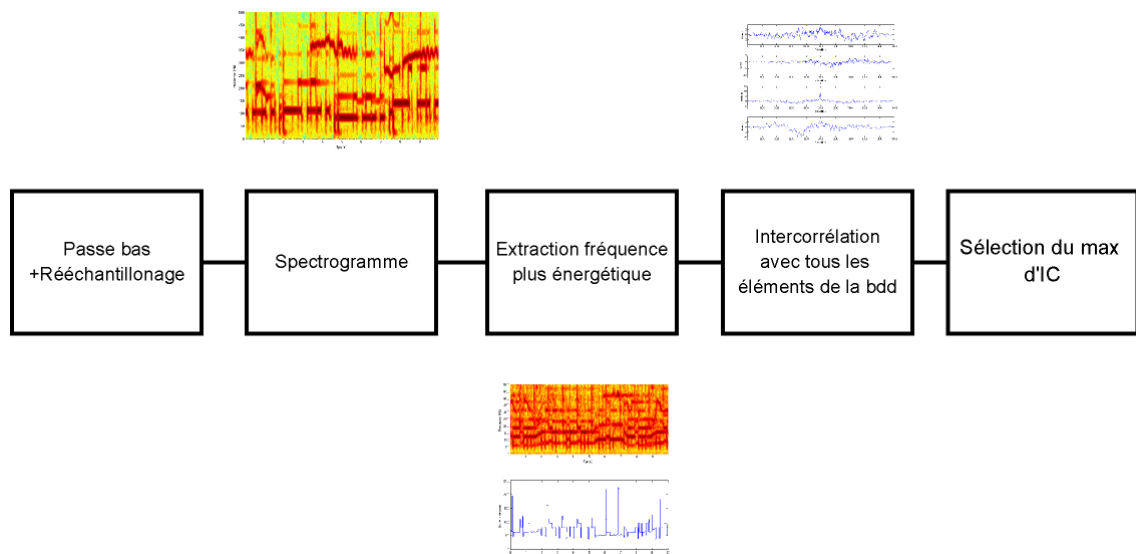
## Chapitre 2

# Mise en place de l'analyse automatique

Nous verrons dans ce chapitre comment précisément nous avons procédé pour l'analyse automatique des signaux

### 2.1 Synoptique générale

Dans ce schéma on retrouve les principales étapes de l'algorithme d'extraction d'empreinte et de reconnaissance dans la base de donnée. Nous détaillerons toutes les étapes dans les sections suivantes.



### 2.2 Extraction d'une empreinte caractéristique

Nous allons voir ici les techniques permettant l'extraction d'une empreinte unique par morceau de musique.

### 2.2.1 Rééchantillonnage

Après avoir fait quelques expériences, ils s'avère qu'un rééchantillonnage à basse fréquence (1KHz) donne de très bons résultats pour plusieurs raisons :

- Réduction de la complexité algorithmique : en effet ce rééchantillonnage fait en sorte que l'algorithme de FFT utilisé pour calculer le spectrogramme doit travailler sur beaucoup moins de valeurs.
- Plus grande stabilité de l'algorithme d'extraction de fréquence la plus intense : l'empreinte étant réalisé par extraction de la fréquence la plus intense sur le spectrogramme, le fait d'appliquer un passe bas à 500 Hz va nous permettre d'extraire avec plus de précision une fréquence plus proche de la fondamentale. On aura ainsi beaucoup moins de chances de sauter d'une harmonique à l'autre.
- Suppression des bruits hautes fréquences et indépendance du format de compression choisi. En effet les formats de compression usuels ont pour habitude de couper les fréquences inaudibles. Ainsi le fait d'effectuer un passe bas en amont de l'analyse rendra celle ci plus robuste au bruit.

---

**Algorithm 4** Passe bas et rééchantillonnage sous matlab

---

```
1 x=resample(y,Fe_new,Fe);
```

---

### 2.2.2 Spectrogramme

Une fois le passe bas effectué, on va créer un spectrogramme d'un extrait du morceau. On prendra pour cela un échantillon d'une dizaine de secondes au milieu de la musique dont on veut réaliser l'empreinte sur lequel on effectuera le spectrogramme.

---

**Algorithm 5** Spectrogramme sous matlab

---

```
1 [s,f,t]=spectrogram(x(10000:50000),Nw,fix(Nw*.98),512,Fe_new);
```

---

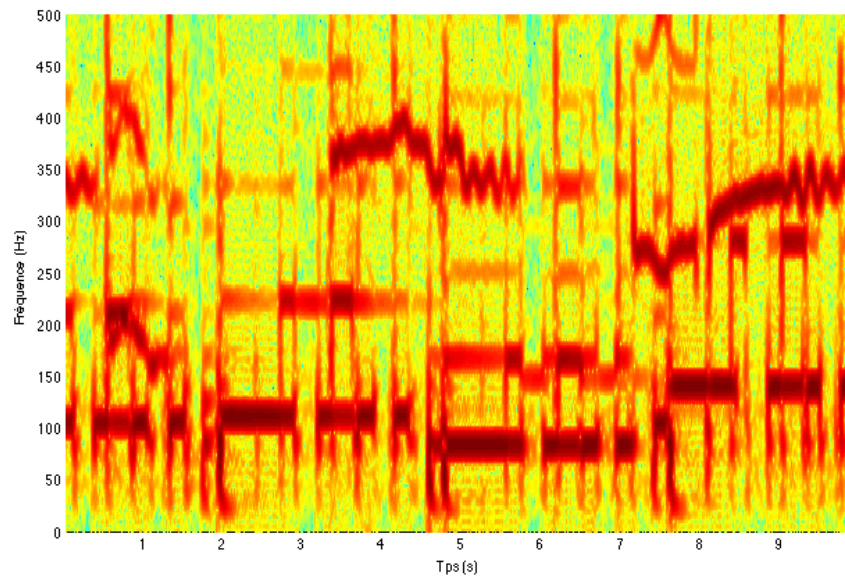


FIG. 2.1 – Spectrogramme d'une chanson

On voit clairement sur la figure du spectrogramme 2.2.2 qu'on peut facilement détecter la fondamentale ou au moins la fréquence la plus importante entre 0 et 500Hz.

### 2.2.3 Détection de la fréquence dominante

Après avoir extrait le spectrogramme d'un échantillon d'un morceau de musique, il faut maintenant extraire une empreinte correspondant aux fréquences les plus intenses en fonction du temps. Pour ça, une simple détection de maximum nous permet d'obtenir un vecteur contenant cette information

---

**Algorithm 6** Extraction du maximum fréquentiel

---

1     $[a, c] = \text{max}(\text{abs}(s)) ;$

---

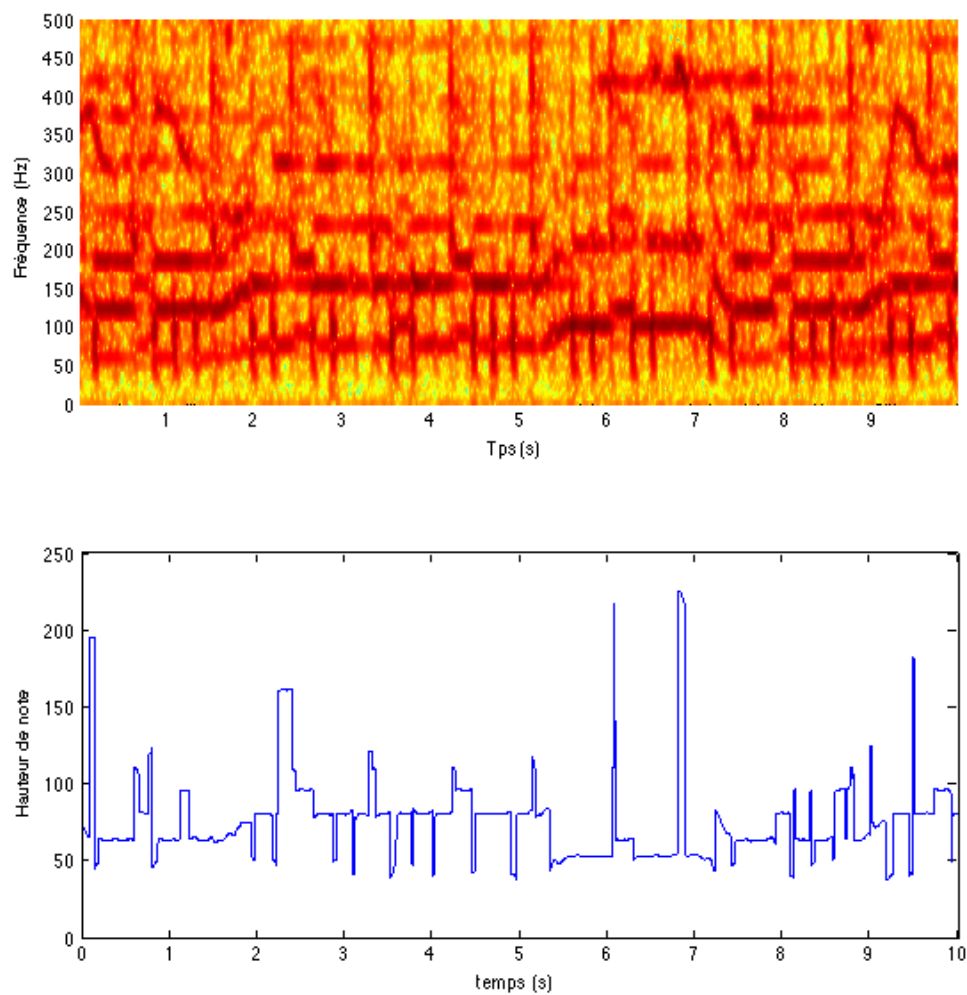


FIG. 2.2 – Spectrogramme et hauteur de note

## 2.3 Construction d'une base de données de musiques

Maintenant que nous avons réussi à extraire une empreinte digne de ce nom pour une musique donnée, il s'agit de créer une mini base de données des oeuvres pour pouvoir faire la reconnaissance d'une parmi les autres. Pour cela, on va simplement créer une matrice contenant les vecteurs empreintes.

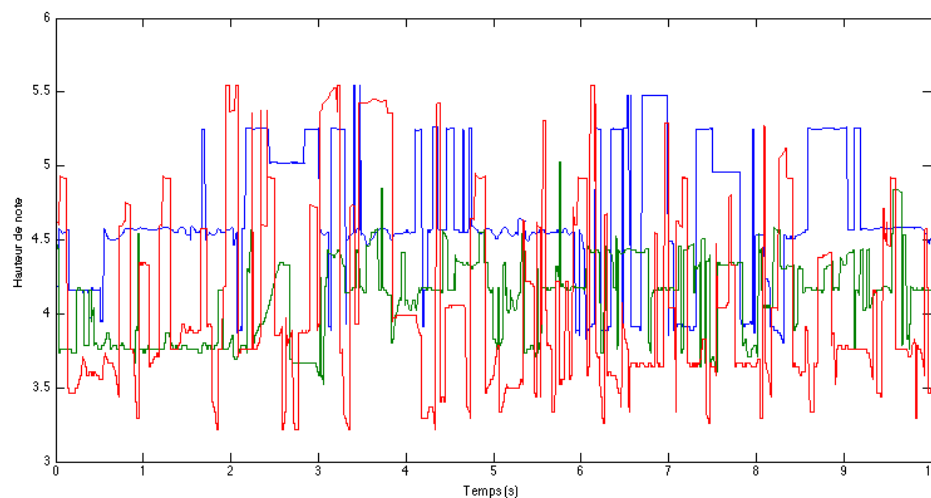


FIG. 2.3 – Plusieurs extraits musicaux représentés

On voit bien sur le graphique représentant les hauteurs de notes qu'elles sont toutes bien différentes et qu'on peut difficilement les confondre

## 2.4 Reconnaissance d'une chanson dans la base de données

### 2.4.1 Méthode

On voit ici la méthode générale utilisée pour repérer la chanson qui concorde le mieux dans la base de données. Tout d'abord une intercorrélation puis un maximum sera effectué.

### 2.4.2 Intercorrélation

Une méthode qui résiste bien aux décalages temporels sera d'effectuer l'intercorrélation entre l'extrait à tester et tous les éléments de notre base de données. Une fois normalisées (on divisera ici par l'écart type du signal pour cela) on devrait obtenir un graphe qui aura l'allure d'une auto-corrélation, le théorème de cauchy schwartz nous assure que le résultat de notre intercorrélation présentera un maximum en zéro (avec un potentiel décalage au cas où les enregistrements soient décalés).

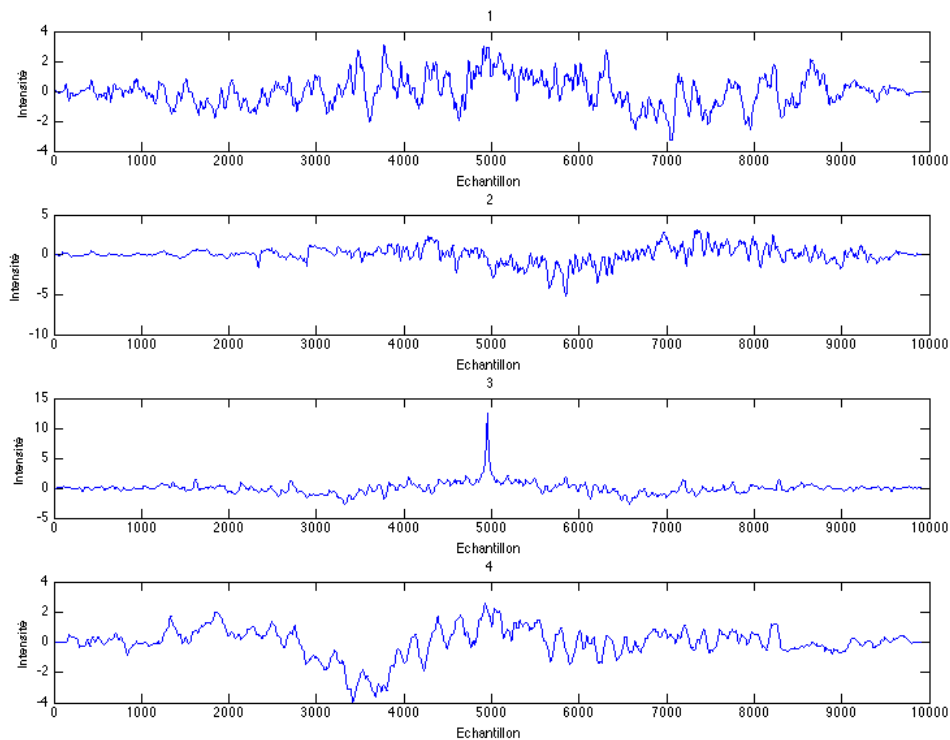


FIG. 2.4 – Intercorrélations entre signaux de la bdd

### 2.4.3 Ordre

Une fois ce maximum d'intercorrélation effectué pour chaque élément de la base, on obtient un critère qui nous permet de créer notre relation d'ordre.

### 2.4.4 Extraction du maximum

Une simple extraction de maximum nous permet de détecter quelle chanson sera la plus pertinente dans notre cas.



---

**Algorithm 7** Extraction du maximum

---

```
1 for i=1:Nmusic
2     % Extraction de la corrélation
3     y=xcorr(cb-mean(cb),c(i,:)-mean(c(i,:)));
4     % Normalisation du vecteur
5     sd = sqrt(var(y));
6     y=y./sd;
7     % Extraction du maximum yv = valeur et ypos la pos
8     [yv,ypos]=max(y);
9     if(yv > vmax)
10         vmax = yv;
11         num = i;
12         % Détection de l'offset de notre chanson
13         % Par rapport à celle dans la BDD
14         offset = ypos-length(y)/2
15         offset = offset*2/Fs;
16     end
17 end
```

---

Cerise sur le gateau, notre algorithme est en mesure de dire le décalage de la musique analysée par rapport à la musique dans la BDD. Ceci grace à la position du pic d'intercorrélation.

## 2.5 Tests de robustesse

### 2.5.1 Ajout d'un bruit

Le premier test de robustesse de robustesse a été l'ajout d'un bruit blanc en vue de mesurer l'impact de celui ci sur la mesure

---

**Algorithm 8** Ajout de bruit

---

```
1 y = wgn(length(sig),1,0.001).*PNoise;
2 sig_bruit = sig + y;
```

---

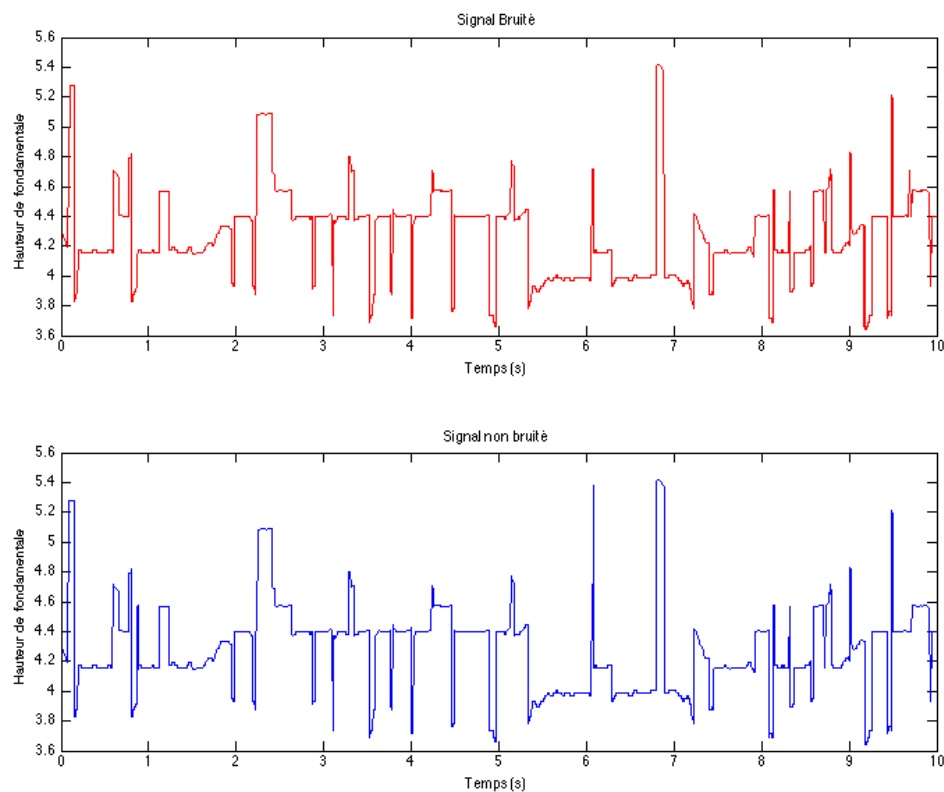


FIG. 2.5 – Comparaison signal bruité/ Non bruité

On voit clairement sur la figure 2.5 que même après ajout d'un bruit d'une puissance non négligeable (audible), on reste toujours dans la plupart des cas très proche de l'empreinte dans la base de données.

### 2.5.2 Test avec deux musiques de deux sources différentes

Le test ultime de notre système consiste à comparer une musique trouvée sur internet sur un site présentant des vidéos de qualité médiocre avec une bibliothèque, sachant que notre chanson à tester est dans la base de données.

Ce test réalisé a donné de très bons résultats avec notre méthode, en effet, le bon fichier a été trouvé dans la base de données, de plus le bon décalage a été trouvé.

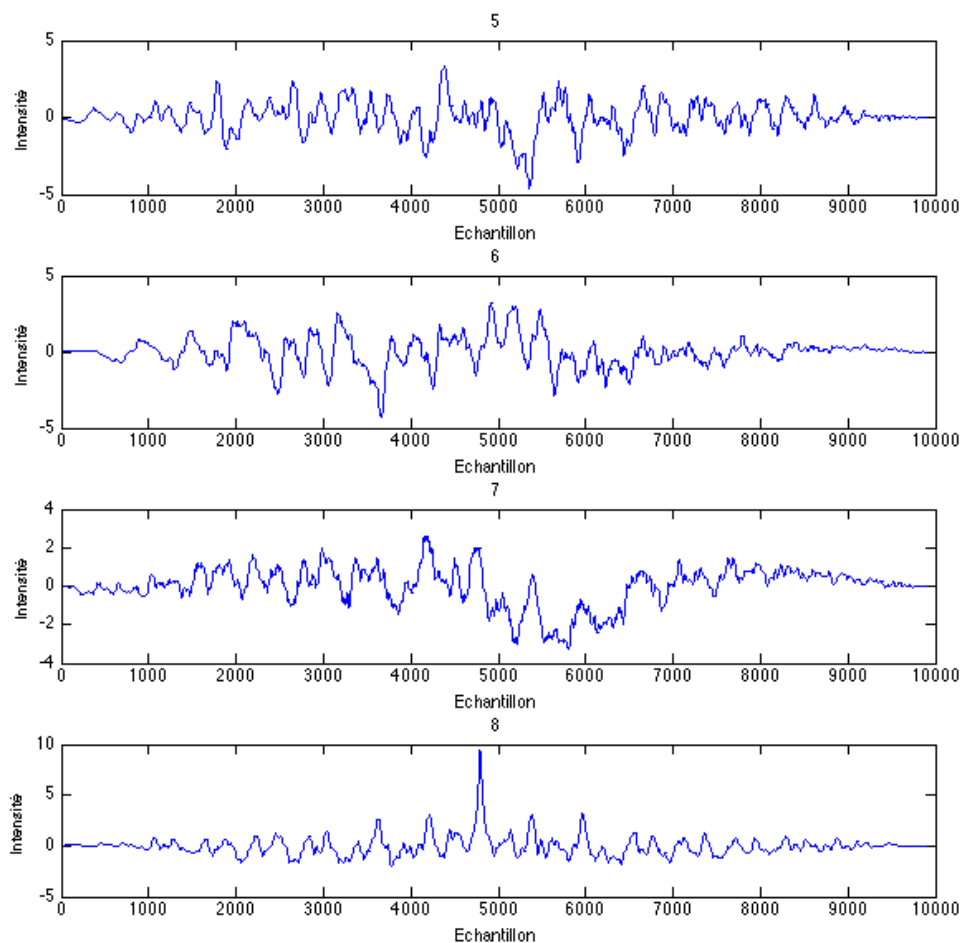


FIG. 2.6 – Diverse intercorrelations dans la bdd

Comme on le voit dans la figure 2.6 un pic se dégage nettement pour le signal 8, ce qui correspond exactement au signal en question dans la base de données.

## 2.6 Ouvertures quant à la complexité et l'algorithmique

### 2.6.1 Complexité

Une première analyse quant à la complexité de nos algorithmes peut se présenter sous la forme suivante :

1. La complexité mémoire est assez intéressante. En effet on se retrouve à stocker une empreinte d'environ 5ko par morceau de musique, et on obtient quelque chose qui n'explose pas en fonction du nombre de musiques dans la base de données, on reste donc en  $O(n)$  au niveau mémoire.
2. La complexité en temps est un peu plus complexe à analyser. En effet, pour chaque morceau de musique, on doit effectuer une intercorrélations avec le morceau à analyser, ce qui

fait que dans tous les cas nous sommes amenés à comparer tous nos morceaux. D'une part l'intercorrélation présente une complexité  $O(n^2)$  par rapport à la taille de l'échantillon ( $5 \cdot 10^3 \cdot 5 \cdot 10^3 = 25 \cdot 10^6$  opérations). De plus nous sommes en  $O(n)$  par rapport au nombre de musiques dans la bibliothèque. On se retrouve donc à terme avec une complexité  $O(nm^2)$  avec  $n$  le nombre de morceaux dans la base de données et  $m$  la taille de l'échantillon. Cette complexité est à la fois la complexité dans le cas moyen et la complexité dans le pire des cas, les deux étant confondues dans ce cas précis étant donné qu'il faut parcourir toute la base.

### 2.6.2 Optimisations potentielles

En vue d'optimiser la recherche dans notre base de données, il faudrait pouvoir passer à une complexité logarithmique. La seule façon pour cela est de trouver un critère de tri sur notre base de données de sorte à ce que la base de données soit conservée triée et que les accès à une chanson précise soit faite en  $\log(n)$ .

## Chapitre 3

# Annexes

### 3.1 Code matlab du programme principal

```

1  %% Fichier de test pour projet SY16
2  f1 = 'samples/president.wav';
3  f2 = 'samples/bach_toccatav.wav';
4  f3 = 'samples/nyd.wav';
5  f4 = 'samples/radiohead_calm.wav';
6  f5 = 'samples/guitar.wav';
7  f6 = 'samples/music.wav';
8  f7 = 'samples/radiohead.wav';
9  f8 = 'samples/cake_orig.wav';
10 fb = 'samples/cake.wav';
11
12 T=1000;
13 Ws = 0.1;
14 tstart = 55;
15 tend = 65;
16 Fs = 1000;
17
18 %
19 % Reading songs :
20 % [sig,fe] = wavread(f1);
21 % sig = sig(:,1);
22 % c(1,:) = log(get_freq(sig,fe,Ws,tstart,tend,Fs));
23 % 1
24 %
25 % [sig,fe] = wavread(f2);
26 % sig = sig(:,1); % c(2, :) = log(get_freq(sig,fe,Ws,tstart,tend,Fs));
27 % 2
28 %
29 % [sig,fe] = wavread(f3);
30 % sig = sig(:,1);
31 % c(3, :) = log(get_freq(sig,fe,Ws,tstart,tend,Fs));
32
33 % 3 % % [sig,fe] = wavread(f4);
34 % sig = sig(:,1);
35 % c(4, :) = log(get_freq(sig,fe,Ws,tstart,tend,Fs));
36
37 % 4 % % [sig,fe] = wavread(f5);
38 % sig = sig(:,1);
39 % c(5, :) = log(get_freq(sig,fe,Ws,tstart,tend,Fs));

```

```

40
41 % 5 % % [sig,fe] = wavread(f6);
42 % sig = sig(:,1);
43 % c(6, :) = log(get_freq(sig,fe,Ws,tstart,tend,Fs));
44
45 % 6
46 %
47 % [sig,fe] = wavread(f7);
48 % sig = sig(:,1);
49 % c(7, :) = log(get_freq(sig,fe,Ws,tstart,tend,Fs));
50 % 7
51
52 %
53 % [sig,fe] = wavread(f8);
54 % sig = sig(:,1);
55 % c(8, :) = log(get_freq(sig,fe,Ws,tstart,tend,Fs));
56 % 8
57 %
58
59
60
61 % Bruitage
62 [sig,fe] = wavread(fb);
63 sig = sig(:,1);
64 PNoise = 0.01;
65 %y = wgn(length(sig),1,0.001).*PNoise;
66 sig_bruit = sig; % + y; cb = log(get_freq(sig_bruit,fe,Ws,tstart,tend,Fs));
67 %
68 % Affichage de toutes les courbes figure(2) for i=1:8 subplot(3,3,i);
69 y=xcorr(cb-mean(cb),c(i,:)-mean(c(i,:)));
70 sd = sqrt(var(y)); y=y./sd; plot(y); title(i) xlabel('Echantillon');
71 ylabel('Intensité'); end
72 vmax = 0; sc = size(c);
73 Nmusic = sc(1);
74
75 for i=1:Nmusic
76 y=xcorr(cb-mean(cb),c(i,:)-mean(c(i,:)));
77 vy = var(y);
78 y=y./sqrt(vy);
79 [yv,ypos]=max(y);
80
81 if(yv > vmax)
82     vmax = yv;
83     num = i;
84     length(y)
85     offset = ypos-length(y)/2
86     offset = offset*2/Fs;
87 end
88
89 end
90 disp('La chanson la plus proche est la numéro');
91 disp(num);
92 disp('Offset temporel :');
93 disp(offset);

```

## 3.2 Code matlab de la fonction get\_freq

```

1 function [c] = get_freq(y,Fe,Ws,tbegin,tend,fe_n)

```

```
2 % fonction permettant de convertir une chanson au format wav en frquences
3 % y: signal
4 % Fe : fréquence d'échantillonnage de y
5 % Ws : taille de la fenetre du temps fréquence
6 % tbegin : temps de départ pour l'analyse
7 % tend : temps de fin pour l'analyse
8 % fe_n : nouvelle fréquence d'échantillonnage
9 % les plus dominantes
10 % Conversion signal .wav sous Matlab
11 Te=1/Fe;
12 N=length(y);
13 t1=0:Te:(N-1)*Te;
14 x=resample(y,fe_n,Fe);
15
16 %Analyse temps frequence du signal audio
17 % Nb d'échantillons dans la fenetre
18 Nw=fix(Ws*fe_n);
19 % Calcul des valeurs du spectrogramme
20 [s,f,t]=spectrogram(x(tbegin*fe_n:tend*fe_n),Nw,fix(Nw*.98),512,fe_n);
21 subplot(211);
22 surf(t,f,log(abs(s)),'EdgeColor','none');
23 %axis xy;
24 %axis tight;
25 %colormap(jet);
26 %view(0,90);
27 %ylabel('Fréquence (Hz)');
28 %xlabel('Tps (s)');
29 %subplot(212);
30 %plot(c);
31 %ylabel('Hauteur de note');
32 %xlabel('echantillon (ech)')
33 [a,c]=max(abs(s));
```

## Chapitre 4

# Bibliographie

[MIREX07] Chunghsin Yeh , Niels Bogaards and Axel Röbel: Synthesized polyphonic music database with verifiable ground truth for multiple F0 estimation, 8th International Conference on Music Information Retrieval (ISMIR'07), Vienna, Austria, 2007.

[ICASSP05] Chunghsin Yeh, Axel Röbel and Xavier Rodet: Multiple fundamental frequency estimation of polyphonic music signals, Proc. IEEE, International Conference on Acoustics, Speech and Signal Processing Philadelphia, 2005.