

Audio Processing

Résumé

Résumé réalisé par : Anthony Rouneau

Aidé par : Maximilien Charlier

Section : 2^{ème} Bloc Master en Sciences Informatiques

Images : Proviennent du cours de M. Dutoit, M. Dupont et M. D'Alessandro.

Contents

I Dutoit – Analyse bas niveau	2
1 Cell phone	2
1.1 Analyse du fichier audio	2
1.2 Prédiction linéaire (LP) sur un segment	3
1.3 Prédiction linéaire sur le son entier (5400bits/s)	3
1.4 CELP (14kbits/s)	4
2 MP3	4
2.1 Principe général	4
2.2 Analyse d'erreur par bande de fréquence	4
2.3 Encodage perceptuel	6
3 Watermarking	7
3.1 Principe général	7
3.2 Spread spectrum	7
3.3 Minimiser l'erreur	8
3.4 Modèle sycho-acoustique	9
4 Dictation machine	10
4.1 Principe général	10
4.2 Approche statique	10
4.3 Approche séquentielle	11
II Dupont – Reconnaissance automatique	12
1 Introduction	12
1.1 Analyse / Reconnaissance	12
1.2 Domaines d'application	12
1.3 Structure musicale	12
1.4 Dimensions musicales	13
2 Analyse musicale	13
2.1 Reconnaissance locale	13
2.1.1 Timbre	13
2.1.2 Harmonie	13
2.1.3 Rythme	14
2.1.4 En bref	16
2.2 Débuts de segment (onset)	16
2.2.1 Variation d'énergie	17
2.2.2 Variations spectrales	18

2.2.3 Sélection d'onset	19
2.2.4 Évaluation des performances	19
2.3 Tempo	20
2.3.1 Méthode initiale	20
2.3.2 Approche d'auto-corrélation	20
2.4 Beats	21
2.5 Conclusion	21
3 Identification de musique	22
3.1 Analyse par frame	22
3.2 Analyse par repères (landmarks)	23
4 Réseaux de neurones (profonds)	25
4.1 Neurone	26
4.2 Réseau	26
4.2.1 Réseaux de neurones convolutionnels (CNN)	27
4.3 Reconnaissance audio	28
4.3.1 Amélioration 1	28
4.3.2 Amélioration 2	28
4.3.3 Amélioration 3	29
4.4 Conclusion	29
III D'Alessandro – Synthèse musicale et audio temps-réel	30
1 Question 1	30
1.1 Question	30
1.2 Éléments de réponse	30
2 Question 2	31
2.1 Question	31
2.2 Éléments de réponse	31
3 Question 3	32
3.1 Question	32
3.2 Éléments de réponse	32
4 Question 4	33
4.1 Question	33
4.2 Éléments de réponse	33
5 Question 5	34
5.1 Question	34
5.2 Éléments de réponse	34

6 Question 6	35
6.1 Question	35
6.2 Éléments de réponse	35
7 Question 7	36
7.1 Question	36
7.2 Éléments de réponse	36

Partie I

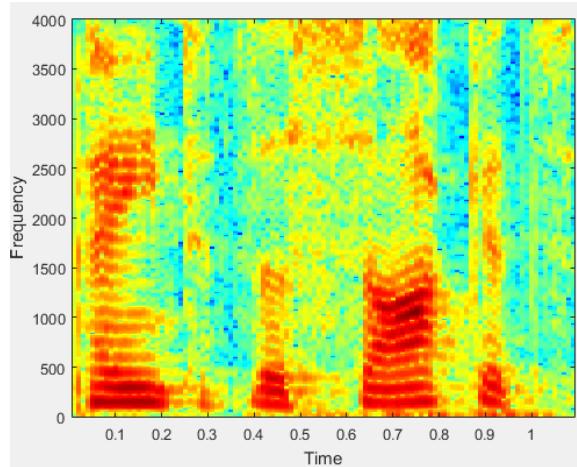
Dutoit – Analyse bas niveau

1 Cell phone

1.1 Analyse du fichier audio

Spectrogramme

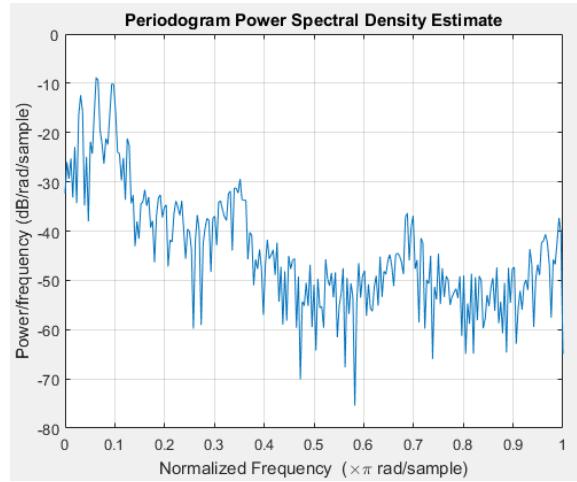
On utilise une **fenêtre de pondération** pour afficher un spectrogramme. Au plus la taille de cette fenêtre est élevée, au plus on a de détails. Inversement, si on prend une petite fenêtre, on aura une vision globale, plus générale. **Les différentes tailles ont chacune leurs avantages et inconvénients.** Un spectrogramme met en évidence les **formants**, qui peuvent être lu et reconnus par des spécialistes. Les fréquences sont sur l'axe vertical, le temps sur l'axe horizontal, et l'intensité en couleur. Exemple de spectrogramme détaillé (plus précis sur les couleurs) :



On affiche souvent un spectrogramme sur le fichier audio complet, ou en tout cas sur plusieurs mots.

Périodogramme

Donne la **densité spectrale de puissance**. L'axe horizontal affiche les fréquences (souvent normalisée pour représenter les fréquences entre 0 et $F_e/2$), tandis que l'axe vertical donne l'amplitude. Le périodogramme s'applique généralement sur une lettre (voyelle) afin d'en identifier les formants. Dans l'exemple suivant, $F_e = 8\text{kHz}$. On a donc $1 = 4\text{kHz}$. On identifie la **fondamentale** à 125Hz (premier pic), et ses **formants** (pics de plus haute amplitude) en 300, 1400 et 2700Hz.



1.2 Prédiction linéaire (LP) sur un segment

Coarticulation

La **coarticulation** est un problème venant du fait que la **prononciation d'un son** dépend du son qui le précède et du son qui le suit. Ça vient du fait que les cordes vocales sont "mécaniques" et présentent une inertie.

Son voisé (~voyelle)

On va chercher 10 coefficients (A_z) qui représente les positions de formants dans la parole. On va ensuite pouvoir retrouver cette voix (la générer) grâce à un filtre tous-pôles ($\frac{1}{A_z}$)¹. Si on passe le signal de base par le filtre² inverse de génération (A_z , qui est l'inverse de $\frac{1}{A_z}$), on obtient le **résidu du signal**, c'est à dire l'**excitation sonore** (indépendante de la lettre formée) qui va définir la **position des harmoniques** et le **pitch du son**.

Son non-voisé (~consonne)

Dans ce cas-ci, on va préférer utiliser un périodogramme moyené (pwelch) afin d'avoir une meilleure estimée de la densité spectrale de puissance. L'excitation peut être générée à partir de données aléatoires (bruit blanc).

1.3 Prédiction linéaire sur le son entier (5400bits/s)

Pitch constant

Le truc ici est de générer une excitation qui restera la même pour tout segment de **10ms de voix**. On obtient alors un son très métallique, très peu naturel. Pour bien faire, on doit prendre en compte l'itération d'avant (retard z) dans le filtrage afin que toutes les frames soient

¹freqz permet d'obtenir la réponse en fréquence d'un filtre et son amplitude à partir de coefficients A_z

²filter permet d'appliquer un filtre à des échantillons.

correctement liées. De plus, on doit faire attention à ce que les excitations (impulsions de Dirac) se rejoignent correctement (décalage nécessaire dans le cas où le nombre de segment choisi n'est pas un multiple de la fréquence d'échantillonnage).

Pitch adaptatif

Pour chaque frame, on va analyser le pitch et générer une excitation qui y correspondra. On doit faire attention au décalage qui est nécessaire pour lier les différentes excitations tout en gardant des fenêtres de 10ms.

LPC10 (2400bits/s)

Le même principe, mais avec des fenêtres plus grandes : 22.5ms.

1.4 CELP (14kbits/s)

Le principe ici est de générer aléatoirement un dictionnaire d'excitations, que l'on va filtrer par les coefficients de l'analyse LPC afin d'avoir des excitations plus riches que de simples impulsions de Dirac.

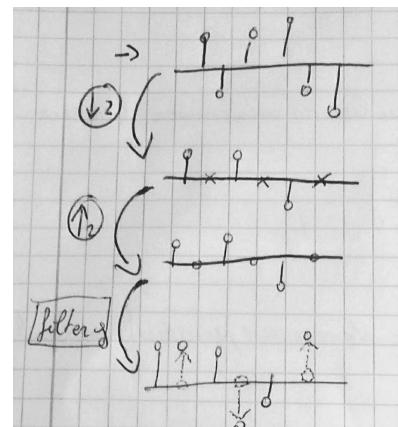
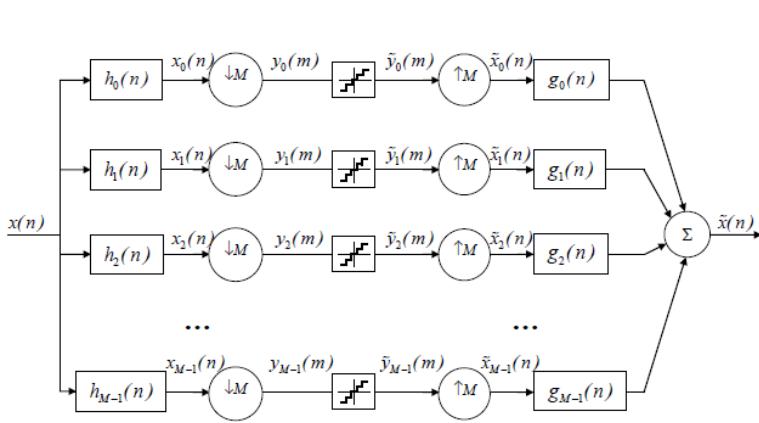
2 MP3

2.1 Principe général

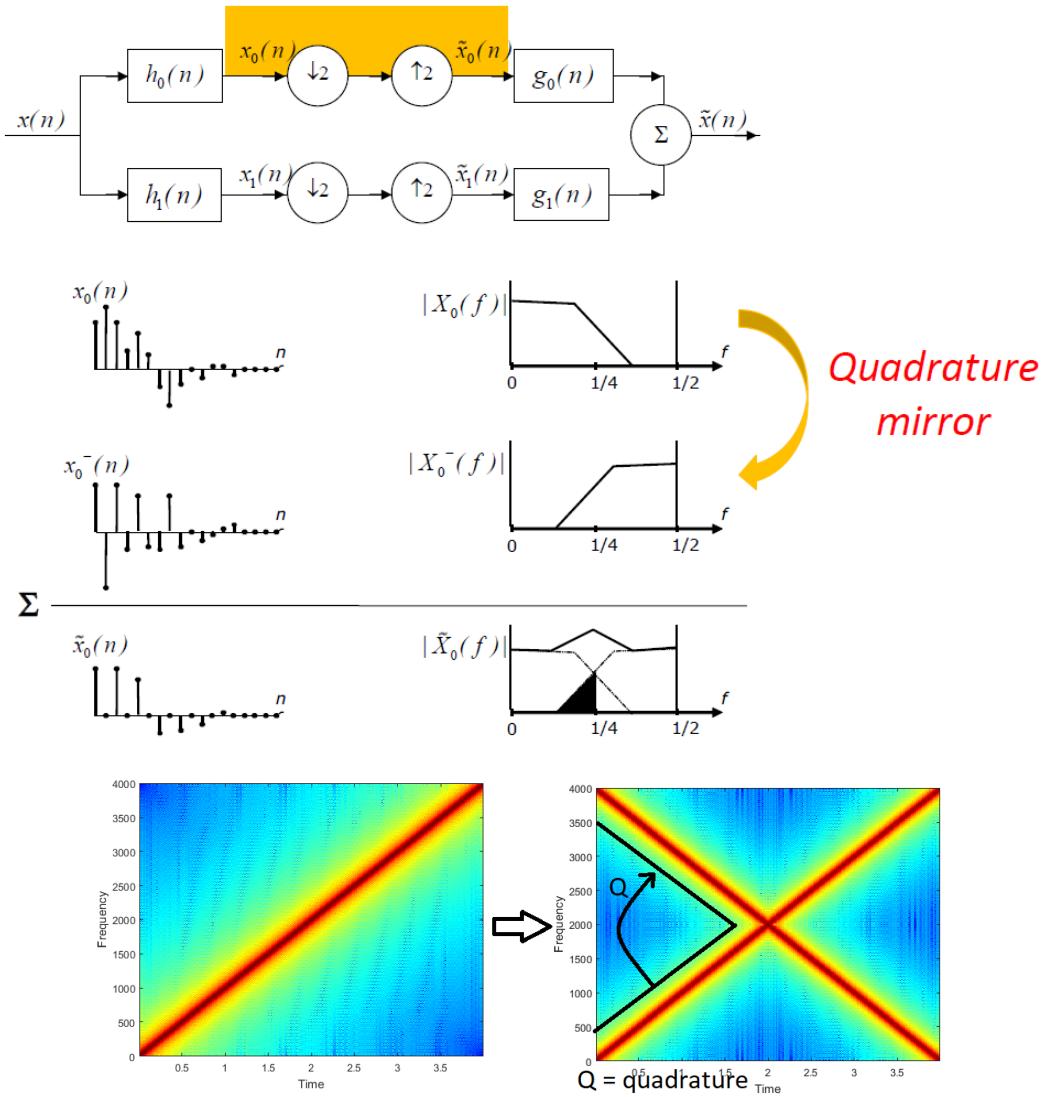
Le principe général de la compression MP3 est d'ajuster la quantization du son en fonction de ce qui est audible ou pas. En effet, on peut augmenter le ratio signal sur bruit (S/N) en augmentant le bit rate, mais ce n'est pas nécessaire de le faire sur toutes les bandes de fréquence de la même façon. C'est pourquoi on va utiliser un modèle de perception qui va nous guider pour savoir quelles bandes de fréquence améliorer et quelles bandes de fréquence il faut 'laisser à l'abandon'.

2.2 Analyse d'erreur par bande de fréquence

Pour réaliser la compression, on a besoin d'ajuster chaque bande de fréquence. C'est pourquoi on aura également besoin de sous-échantillonner les signaux pour pouvoir reconstruire le signal final. En effet, si on divise le signal original en M bandes de fréquences, on aura M fois plus d'échantillons que le signal original. Ensuite, on peut le passer dans un filtre "escalier" afin de le digitaliser. Si l'on veut retrouver le même nombre d'échantillons qu'avant le scaling, on va devoir sur-échantillonner. C'est ce qu'on appelle le sub-band filtering. Le principe est donc de retirer des échantillons, de les remplacer par des zeros, et ensuite de passer le tout dans un filtre qui va donner de la "couleur" à ces zéros.



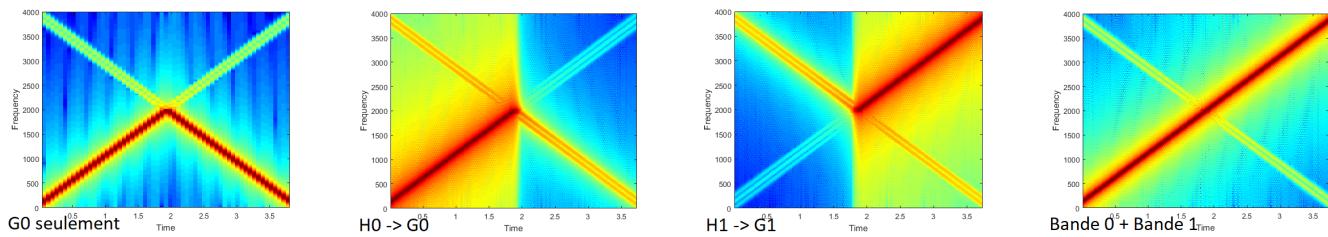
En prenant un exemple, si on effectue cette opération sur un 'chirp', on aura un recouvrement total du spectre. En effet, les filtres utilisés avant et après la digitalisation ne sont pas "idéaux", et ajoutent de l'erreur. On a alors un effet de 'miroir en quadrature' :



On va alors faire en sorte que les filtres g_0 et g_1 produisent des **recouvrements spectraux qui s'annulent** réciproquement (Esteban & Galland). Il y a plusieurs méthodes pour générer ces filtres. **MP3 est un mélange de PQMF et de MDCT**. En particulier, le MP3 effectue 32 fois moins de calcul que le MDCT car il calcule les cas particuliers seulement.

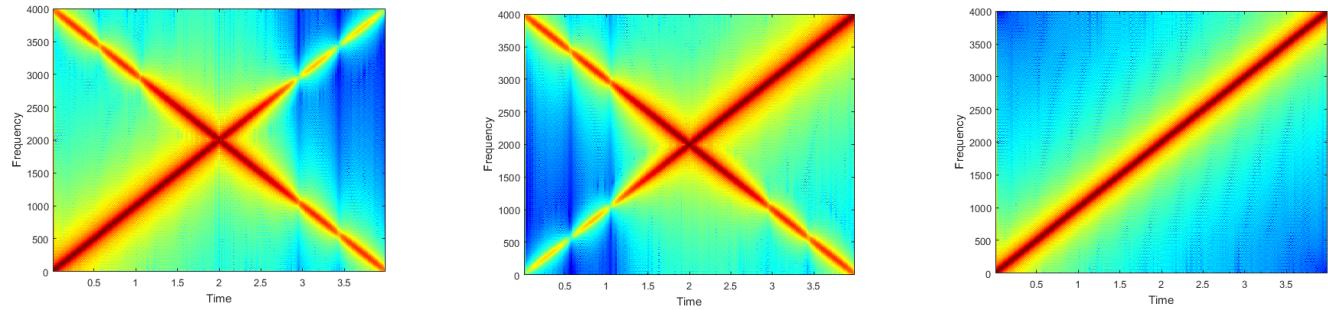
Deux filtres simples

Dans l'exemple avec le chirp (On prend $H_0 = G_0$ et $H_1 = G_1$), on a $M = 2$. On a donc deux bandes : une passe-bas et une passe-haut. Les filtres H sont appelés filtres "quarter-band" et permettent de supprimer l'aliasing.



Banque de filtres QMF

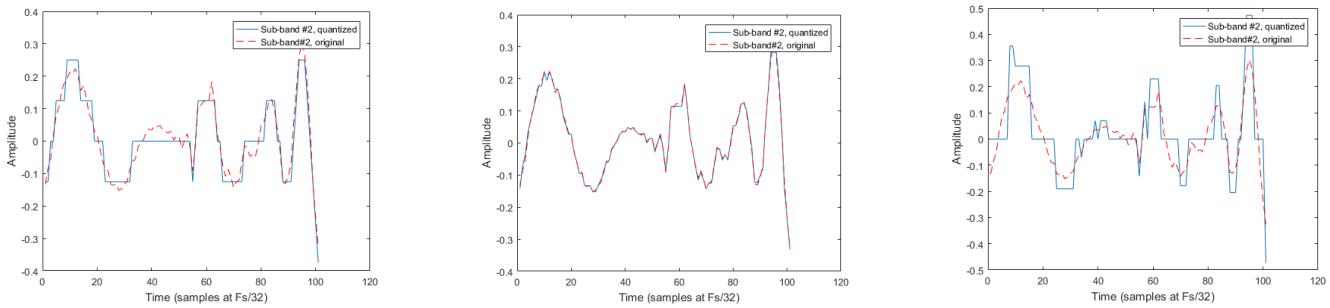
Ces filtres garantissent une reconstruction parfaite, tout en acceptant de l'aliasing dans les bandes. le fait d'additionner l'aliasing de chaque bande à la fin permet une annulation complète de l'aliasing.



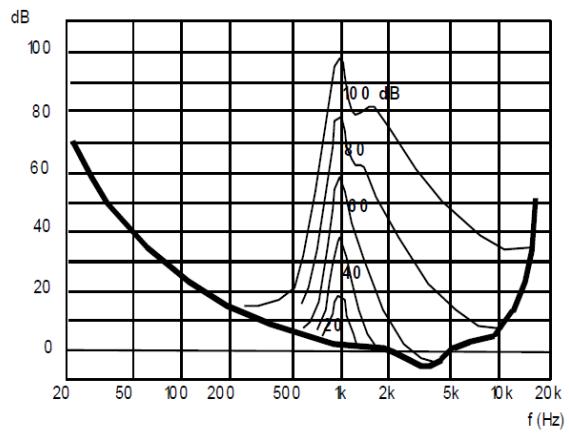
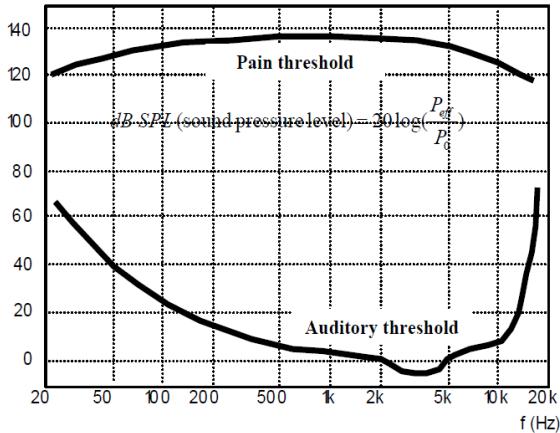
2.3 Encodage perceptuel

Lorsqu'on encode un signal sur un certain nombre de bit, l'erreur d'encodage peut être élevé. On peut baisser cette erreur en mettant des poids sur les sous-bandes de fréquence. En effet, on va utiliser l'amplitude (en valeur absolue) maximum afin de trouver le 'niveau de précision' nécessaire à l'encodage de cette bande de fréquence. On retrouve dans les exemples suivants : 1) un encodage typique sur 4 bits, 2) un encodage pondéré sur 4 bits, 3) un encodage pondéré sur 4 bits où les poids ont été multipliés par 10.

Si on divisait les poids par x , on augmenterait la précision de l'encodage, mais on couperait les hautes amplitudes (on considèrerait que le max des amplitudes est lui aussi divisé par x).



Le principe du MP3 est d'appliquer un tel encodage pondéré, en utilisant un modèle perceptuel calculé selon l'oreille humaine. On va donc accentuer la précision sur les zones où l'oreille humaine est sensible et diminuer la précision dans les zones que l'oreille humaine n'entend pas. On dit donc que le signal sur bruit (SNR) doit rester au dessus du SMR (seuil défini par le modèle auditif).



3 Watermarking

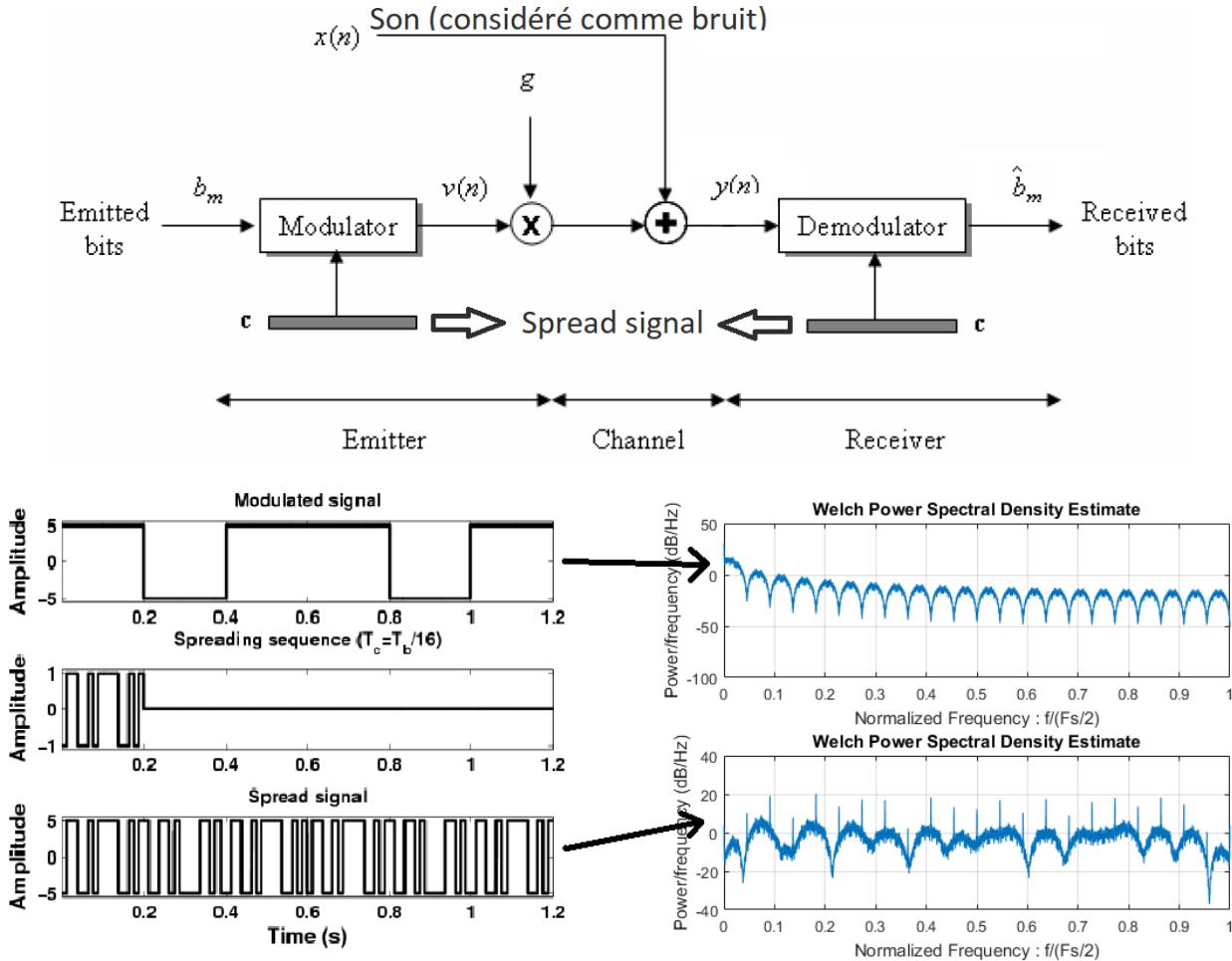
3.1 Principe général

Le principe du watermarking va être d'utiliser une encodage en **spread spectrum** classique, comme celui de l'ADSL sur le signal téléphonique, afin de cacher une information ou un message dans un signal audio. Pour ce faire, on va considérer le signal audio comme étant le bruit, et la watermark comme étant le signal d'origine.

3.2 Spread spectrum

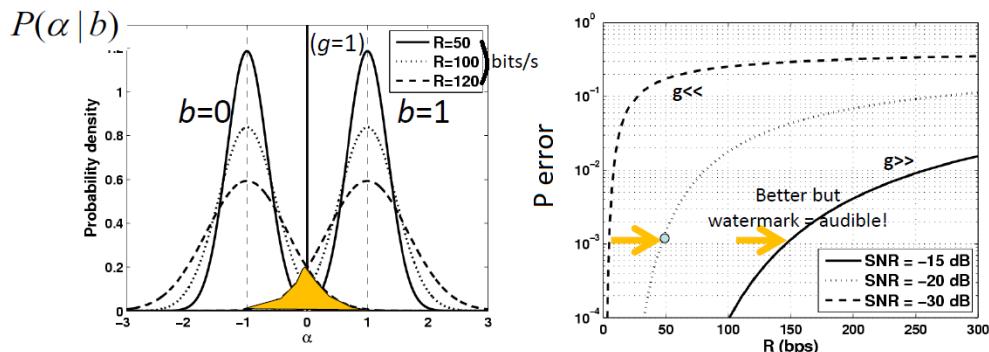
On commence donc par encoder un message en bits. Ensuite, on va générer un "spread signal" qu'on utilisera pour encoder le message dans le signal audio (qui est considéré comme le bruit). En effet, ce faisant, on peut rendre la densité spectrale de puissance plus plate, et donc mieux faire passer 'incognito' la watermark dans le son. Il reste alors à additionner le signal audio avec la watermark pour donner le son "watermarké". Le nombre d'échantillons pris par

le spread signal est calculé grâce à $\frac{F_s}{\text{debit}}$). Pour démoduler, il faut alors multiplier les frames (autant de fois qu'il y a de bits dans le message) du son watermarké par le spread signal.

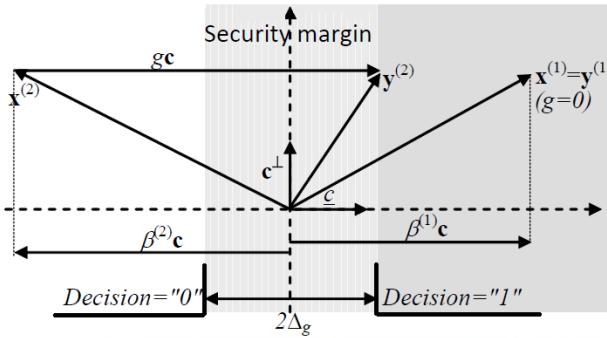


3.3 Minimiser l'erreur

En augmentant le gain (et donc le SNR), on réduit l'erreur, mais on augmente aussi l' chance d'entendre la watermark. De plus, en diminuant le débit, on peut également réduire l'erreur (réduire la zone de confusion entre les gaussiennes).

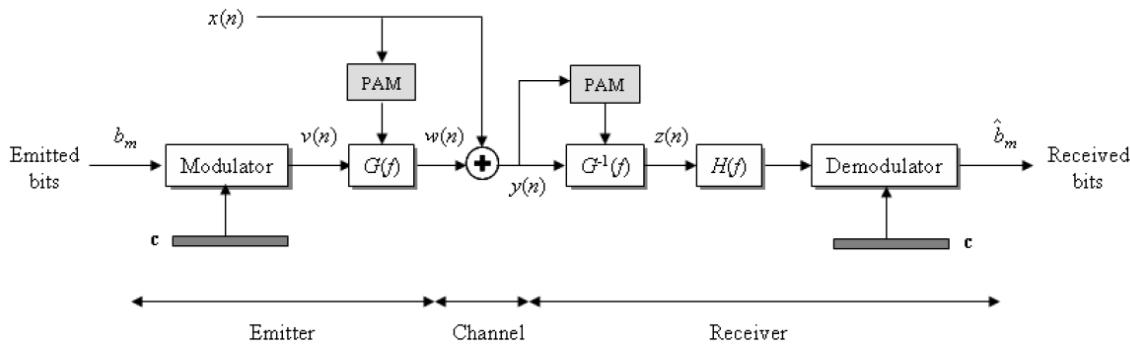


L'idée est donc de réduire le débit autant que possible, en poussant le gain un minimum. En particulier, on peut augmenter le gain "juste ce qu'il faut" pour ne pas avoir d'erreur de détection. Mais de cette manière, on risque très probablement d'entendre la watermark au dessus de l'audio...



3.4 Modèle psycho-acoustique

On va donc utiliser le modèle psycho-acoustique (PAM) pour augmenter le gain sur certaines bandes de fréquences inaudibles. On va donc pousser le gain par bande de fréquence, jusqu'à atteindre la limite audible. Ceci se fait grâce à un "shaping filter" (G) qui va faire coller la watermark au maximum au modèle auditif. Enfin, on a besoin d'un filtre de Wiener (H) qui va extraire la watermark du signal encodé. H est un filtre tous zéros (non-récuratif), et ses coefficients sont symétriques (filtre non-déphasant).



Pour calculer H , on aurait idéalement besoin de connaître v , mais si on connaît v , ça ne sert à rien de démoduler le signal... On va donc utiliser le spread spectrum pour estimer Φ_{vv} .

$$\begin{bmatrix} \phi_{zz}(0) & \phi_{zz}(1) & \dots & \phi_{zz}(2p) \\ \phi_{zz}(-1) & \phi_{zz}(0) & \dots & \phi_{zz}(2p-1) \\ \dots & \dots & \dots & \dots \\ \phi_{zz}(-2p) & \phi_{zz}(-2p+1) & \dots & \phi_{zz}(0) \end{bmatrix} \begin{bmatrix} h_{-p} \\ h_{-p+1} \\ \vdots \\ h_p \end{bmatrix} = \begin{bmatrix} \phi_{vv}(-p) \\ \phi_{vv}(-p+1) \\ \vdots \\ \phi_{vv}(p) \end{bmatrix}$$

H

4 Dictation machine

4.1 Principe général

Le principe ici est de calculer les probabilités d'obtenir des séquences de lettres. Formalisé, ça donne ceci :

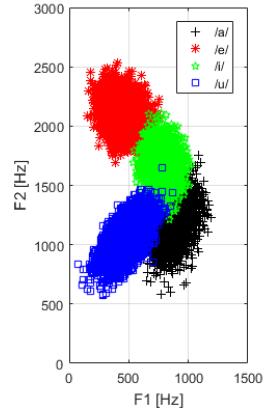
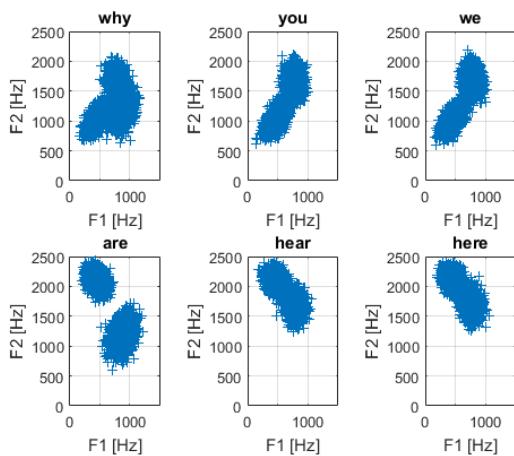
$$M^* = \arg \max_{M_i} P(M_i | X, \Theta)$$

Bayes \rightarrow $P(M_i | X, \Theta) = \frac{P(X | M_i, \Theta) P(M_i | \Theta)}{P(X | \Theta)}$
 Lettre \downarrow $P(X | M_i, \Theta)$
 Observation \downarrow $P(M_i | \Theta)$
 Langage \downarrow $P(X | \Theta)$
Hidden Markov Model \rightarrow $P(X | M_i, \Theta)$
Acoustic model likelihood \rightarrow $P(X | M_i, \Theta)$
Language model prior \rightarrow $P(M_i | \Theta)$
Markov model (n-gram) \rightarrow $P(M_i | \Theta)$

On peut alors essayer de retrouver la lettre en fonction des fréquences détectées (et donc par leurs formants) et éventuellement en fonction des lettres précédentes. On fonctionne généralement avec le logarithme des probabilités afin d'avoir des nombres plus grands et plus comparables

4.2 Approche statique

Cette première approche suppose que l'ordre temporel des lettres n'a pas d'importance. C'est évidemment faux, mais cela peut servir à introduire le sujet. On commence alors par calculer les probabilités **a priori** ($P(X)$) des lettres (probabilités de tomber sur la lettre X) en fonction du jeu de données. Ensuite, il reste à calculer les 'likelihood' de chaque observation en fonction de chaque lettre. Ceci peut se faire grâce à un modèle gaussien.



Comme chaque lettre peut être représentée par une gaussienne, des mots peuvent être représentés par un modèle de mélange de gaussiennes. Ce genre de modèle posera problème lorsque l'ordre des lettres à de l'importance. Dans l'exemple, les mots 'you' (/iu) et 'we' (/ui) auront exactement le même modèle, et on ne pourra donc pas les distinguer avec une telle approche.

On peut estimer un tel modèle grâce à un clustering (il faut prendre des valeurs réelles,

où proche de celles-ci pour initialiser k-means). Le problème est que k-means n'a qu'un sens géométrique. On va faire appel à l'algorithme d'Estimation-Maximisation (E-M) avec les valeurs obtenues par k-means comme valeurs initiales. Estimation = On calcule les probabilité pour une donnée d'appartenir à un modèle (cluster). Maximisation = On recalcule les moyennes/variances pour maximiser la séparation entre les classes.

4.3 Approche séquentielle

HMMs

On va utiliser ici des modèles de Markov cachés. Ces modèles sont des machines à états qui ont des probabilités d'émission et de transition. Les probabilités de transitions sont nécessaires pour passer d'un état à un autre. On va alors chercher le "best-path" parmi les modèles de Markov disponibles en utilisant l'algorithme de Viterbi.

N-grams

On va utiliser des probabilités que deux mots puissent se suivre. Par exemple, pour séparer 'hear' (/ir) et 'here' (/ir), qui ont la même prononciation, on peut dire que le bigram ('we', 'here') est très peu probable. On peut calculer ces probabilités sur de grandes bases de données de phrases.

Partie II

Dupont – Reconnaissance automatique

1 Introduction

1.1 Analyse / Reconnaissance

Il y a deux étapes dans la reconnaissance automatique de sons :

- **Analyse** – Extraction d'**attributs** (features) et d'**information haut-niveau depuis un signal brut**.
- **Reconnaissance** – **Utilisation des attributs** extraits par l'analyse pour **reconnaitre/classer** un son, une musique.

Depuis l'arrivée des réseaux de neurones profonds, ces étapes tendent à se confondre (à cause des CNNs qui analysent et classent). Si on regarde la musique comme un signal temporel uniquement, c'est un **signal très désordonné** et est un peu un fouillis. Le but de l'analyse/reconnaissance c'est d'arriver à **analyser les sons et les musiques comme le cerveau humain** le ferait : isoler des instruments, isoler le rythme et le tempo, imaginer une partition musicale permettant de jouer la musique écoutée, ...

1.2 Domaines d'application

- **Identification de musiques** – Reconnaître artiste et titre d'une musique.
- **Transcription de musiques** – Extraire la partition d'une musique. Les machines en sont actuellement incapable, mais la recherche se focalise sur des sous-problèmes comme identifier les instruments joués, le rythme, etc...
- **Recommandation de musiques** – Trouver des similitudes entre les musiques pour proposer des playlist.
- **Production de musiques** – Modifier des sons pour améliorer la qualité de la musique ou générer des effets.

1.3 Structure musicale

Au niveau de l'analyse, on peut établir une hiérarchie concernant les parties de musiques analysées. La **plus petite partie** analysable d'un fichier audio est une **trame (frame) audio**. A partir de celles-ci, on peut établir une hiérarchie allant du plus gros agglomérat à la plus petite partie audio :

Sections ⇒ Mesures ⇒ Beats ⇒ Segments ⇒ Trames

Ce qui veut dire qu'on peut former un segment à partir de plusieurs trames, un beat à partir de plusieurs segment, etc... On peut noter qu'un **segment** représente une note, de son début à sa fin.

1.4 Dimensions musicales

Trois dimensions principales peuvent être isolées en parlant de musiques (avec les outils permettant de les analyser) :

- Le timbre du son – MFCCs.
- La mélodie, l'harmonie, les accords – Chroma
- Le rythme et le tempo – Rythmogramme

Ces trois dimensions seulement ne suffisent pas pour décrire les musiques les plus complexes.

2 Analyse musicale

2.1 Reconnaissance locale

2.1.1 Timbre

Le **timbre** représente tout ce qui rend le son particulier. Par exemple la différence entre une note de piano et la même note jouée à la guitare est le timbre du son. On parle aussi de timbre pour des combinaisons d'effets différents. On utilise des Mel-frequency cepstral coefficients (MFCCs) sur chaque frame comme attributs du son pour isoler le timbre. En effet, les MFCCs sont utilisés car ils ne prennent pas en compte le "pitch", et donc la note jouée. En pratique, on utilise un **timbrogramme**, qui est une séquences de MFCCs affichée de manière similaire à un spectrogramme. Ils sont une représentation compacte de l'enveloppe spectrale de puissance. L'idée est que chaque coefficient représente une partie du timbre.

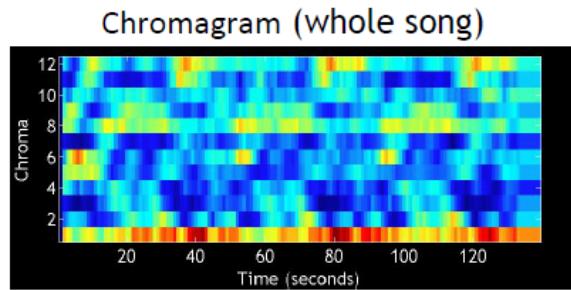
Les MFCCs sont bons pour détecter les instruments utilisés, mais aussi pour détecter l'artiste et le genre. En effet, les effets utilisés peuvent aider pour classifier le genre, et les instruments utilisés peuvent aider à reconnaître l'artiste.

2.1.2 Harmonie

Pour détecter les harmonies et les notes en général, on va utiliser des chromagrammes sur des frames de musique (30ms). Ils vont tenter de détecter des suites de notes en ignorant l'octave de ces dernières. De cette manière, on peut identifier les accords joués (ainsi que leurs variantes: mineurs, 7ème, ...). Le principe est qu'une banque de filtres et utilisée afin d'isoler chaque note. Pour chaque note, un filtre $B(x)$ est utilisé et retourne 1 si le modulo 12 vaut 0 (si on se trouve à une octave de la note du filtre). On peut noter que dans la formule, un paramètre k_0 permet d'accorder le chroma à sa guise (pour varier du La à 440Hz par exemple).

$$C(b) = \sum_{k=0}^{N_M} B(12 \log_2(k/k_0) - b) W(k) |X[k]|$$

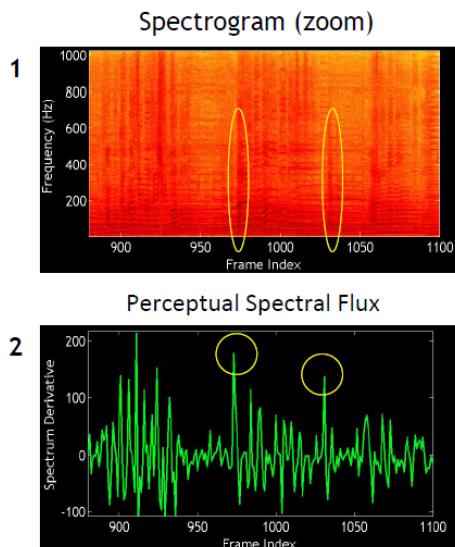
o $W(k)$ is weighting, $B(b)$ selects every $\sim \text{mod } 12$



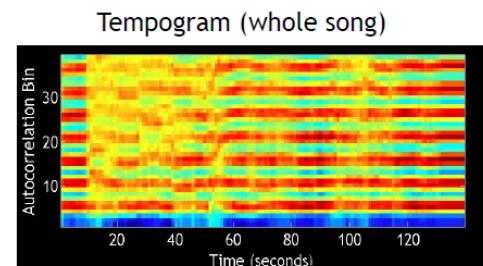
Le chromatogramme tel quel a un problème : il dépend de la précision de la FFT utilisée. En effet, les fréquences (discrètes) détectées par la FFT peuvent être un peu décalées par rapport aux fréquences des notes de musique. Le chromatogramme peut être utilisé pour reconnaître les accords, la clé de la musique, les émotions données par la musique, ou encore pour analyser les musiques qui vont bien ensemble (utilisent la même suite d'accord).

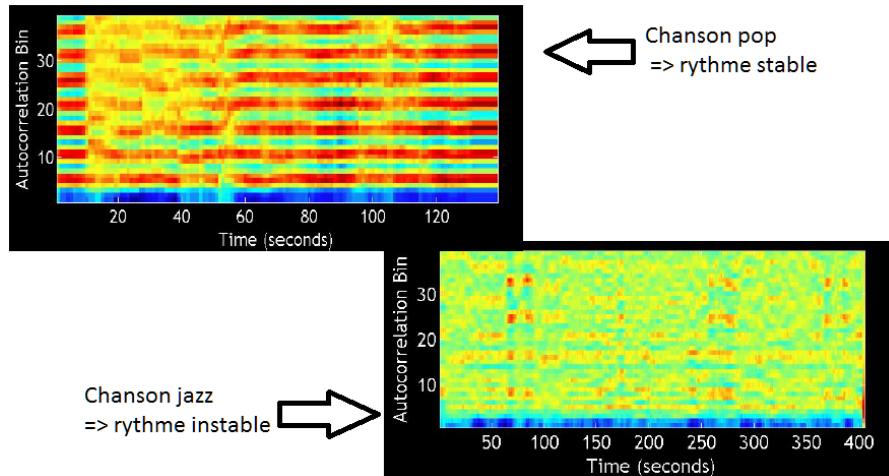
2.1.3 Rythme

Contrairement aux deux précédentes dimensions, le rythme s'analyse sur plusieurs secondes de musique, un beat n'a de sens qu'avec les beats qui l'entourent pour finalement former un rythme. C'est pourquoi on va analyser des mesures.

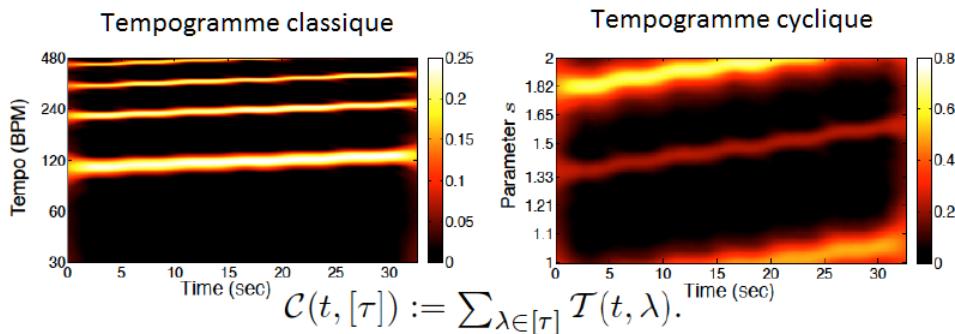


3 Periodicity Computation by Autocorrelation Function

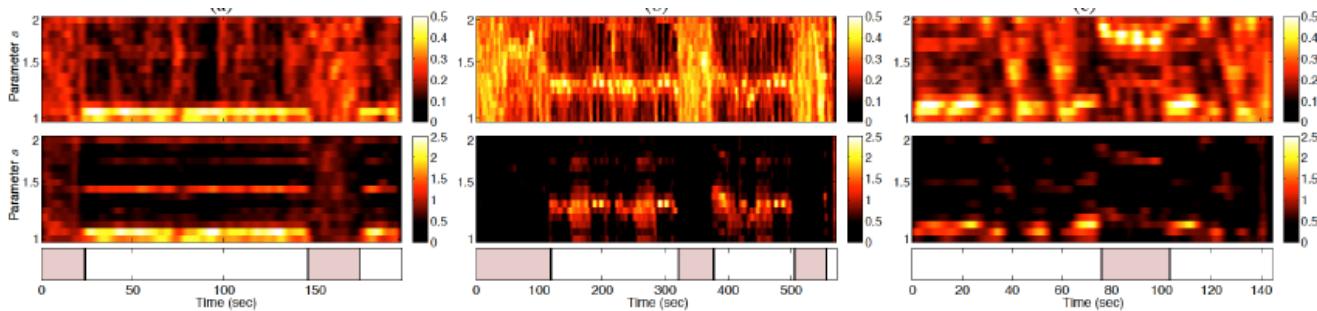




Le **flux spectral perceptuel** représente un graphe de changement local. A chaque pic, on dit qu'il y a un **nouvel "event"**. C'est pourquoi les parties de chanson avec une batterie forte présentent des pics réguliers et donc un tempogramme régulier. Il y a plusieurs lignes à cause des multiples de la période (genre $2\pi, 4\pi, \dots$). Pour rectifier le tir, on va utiliser un **tempogramme cyclique** dont les ordonnées $\in [1, 2]$ car on retient tout ce qui se trouve entre le tempo de base (1) et ses multiples (2).

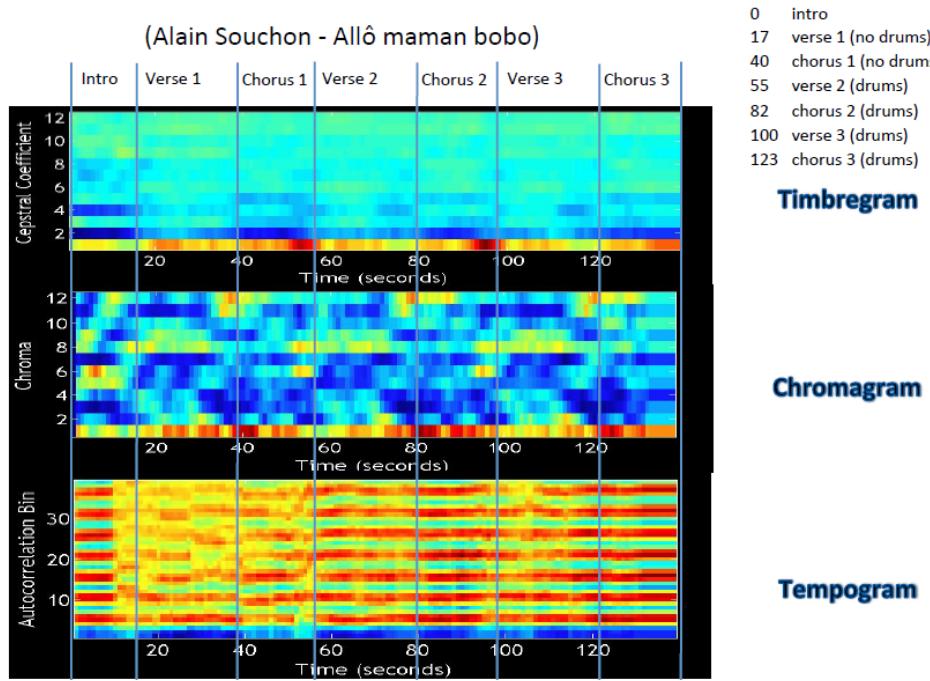


On peut utiliser les tempogrammes pour **segmenter les musiques** en différentes parties. On peut imaginer qu'à chaque changement de rythme, une nouvelle partie de la musique commence



2.1.4 En bref

Grâce au timbre (timbrogramme), aux notes (chromagramme) et au rythme (tempogramme), on peut essayer de classer des musiques selon leur genre et selon les émotions qu'elles évoquent.



2.2 Débuts de segment (onset)

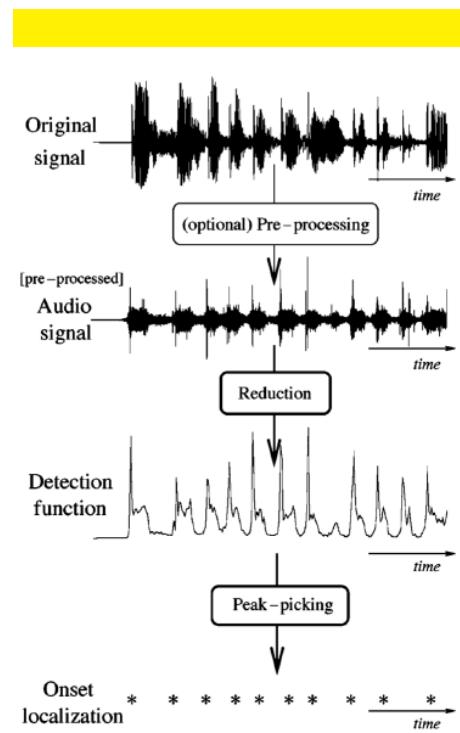
Cette partie est consacrée au calcul du flux spectral perceptuel. En effet, on a besoin de déte^{cte}re le début de chaque note jouée pour pouvoir estimer le début d'un nouvel "event". Cela pose quelques problèmes à résoudre :

- Séparer les notes de plusieurs instruments qui jouent en même temps.
- Combler le manque de démarcation claire (note qui change sans impulsion claire).
- Ne pas considérer un arrêt soudain d'une note comme un nouvel "event".

La création de la fonction de détection d'onset (onset strength function) s'appelle aussi la réduction car on part d'un signal audio complet pour obtenir un signal réduit ne comprenant que des probabilités d'avoir un onset à un moment donné (car on va avoir une valeur toutes les 10ms environs, donnant une fréquence de 100Hz).

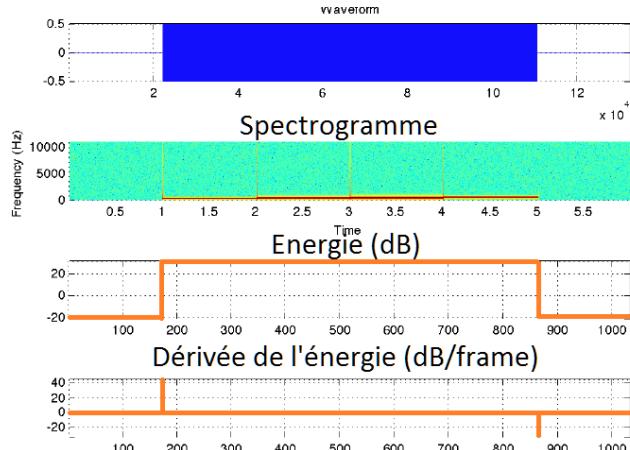
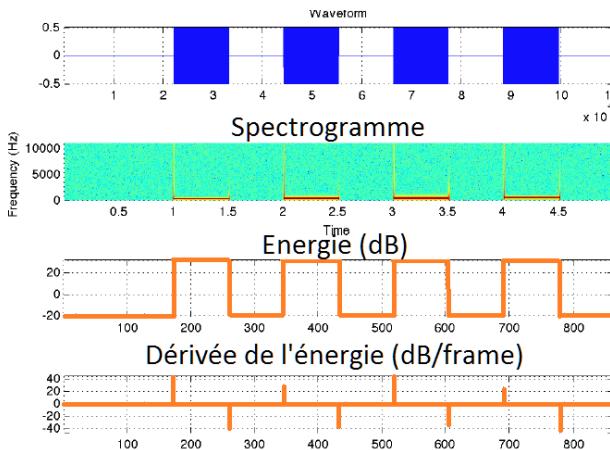
Il y a trois chose que l'on peut faire pour la partie "réduction".

- Déetecter les changements temporel (saut d'énergie soudain).
- Déetecter les changements spectraux (changements nets dans des fft sur de courtes périodes).
- Utiliser des modèles probabilistes.



2.2.1 Variation d'énergie

On ne peut pas toujours détecter les onset par la dérivée de l'énergie car les notes sont parfois toutes connectées



Pour remédier à ce problème, on peut amplifier l'amplitude des hautes fréquences dans le spectre afin de marquer les changements abrupts. En effet, un changement soudain implique une montée d'énergie dans les hautes fréquences. C'est ce qu'on a fait dans l'exemple suivant;

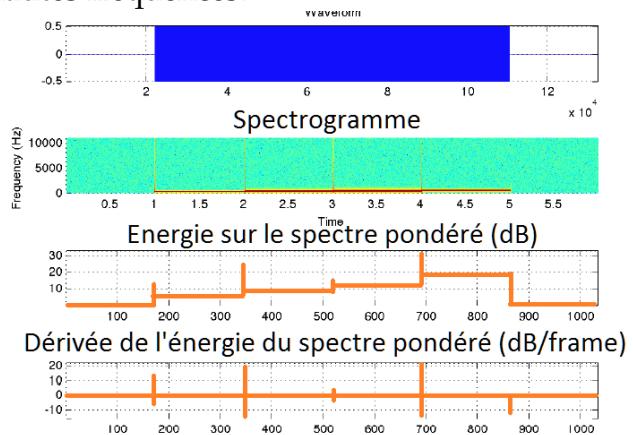
on a calculé l'énergie sur le spectre amplifié en hautes fréquences.

$$\text{Base} \quad \tilde{E}(n) = \frac{1}{N} \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} W_k |X_k(n)|^2$$

W_k Poids qui sera élevé sur les hautes fréquences (pour les mettre en évidence).

$$\text{Avancé} \quad o(t) = \sum_f W(f) \max(0, \frac{|X(f, t)|}{|X(f, t - 1)|} - 1)$$

$W(f)$ fonctions spécifiques à certaines bandes de fréquences

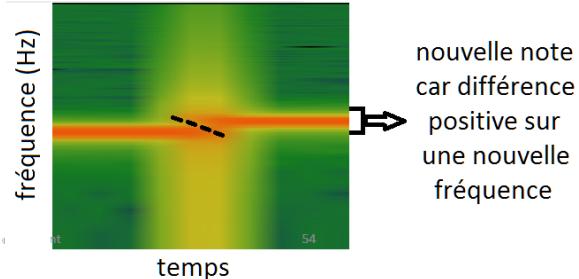


2.2.2 Variations spectrales

On va ici regarder les changements entre des FFT consécutives. On va donc faire la différences entre plusieurs FFT. On ne garde que les différences positives (et on retourne 0 sinon) car on cherche de nouvelles fréquences (si nouvelle - ancienne > 0 alors il est possible qu'une nouvelle note soit jouée). On finit par trouver des changements de fréquence fondamentale en accumulant ces différences pour chaque bande de fréquence.

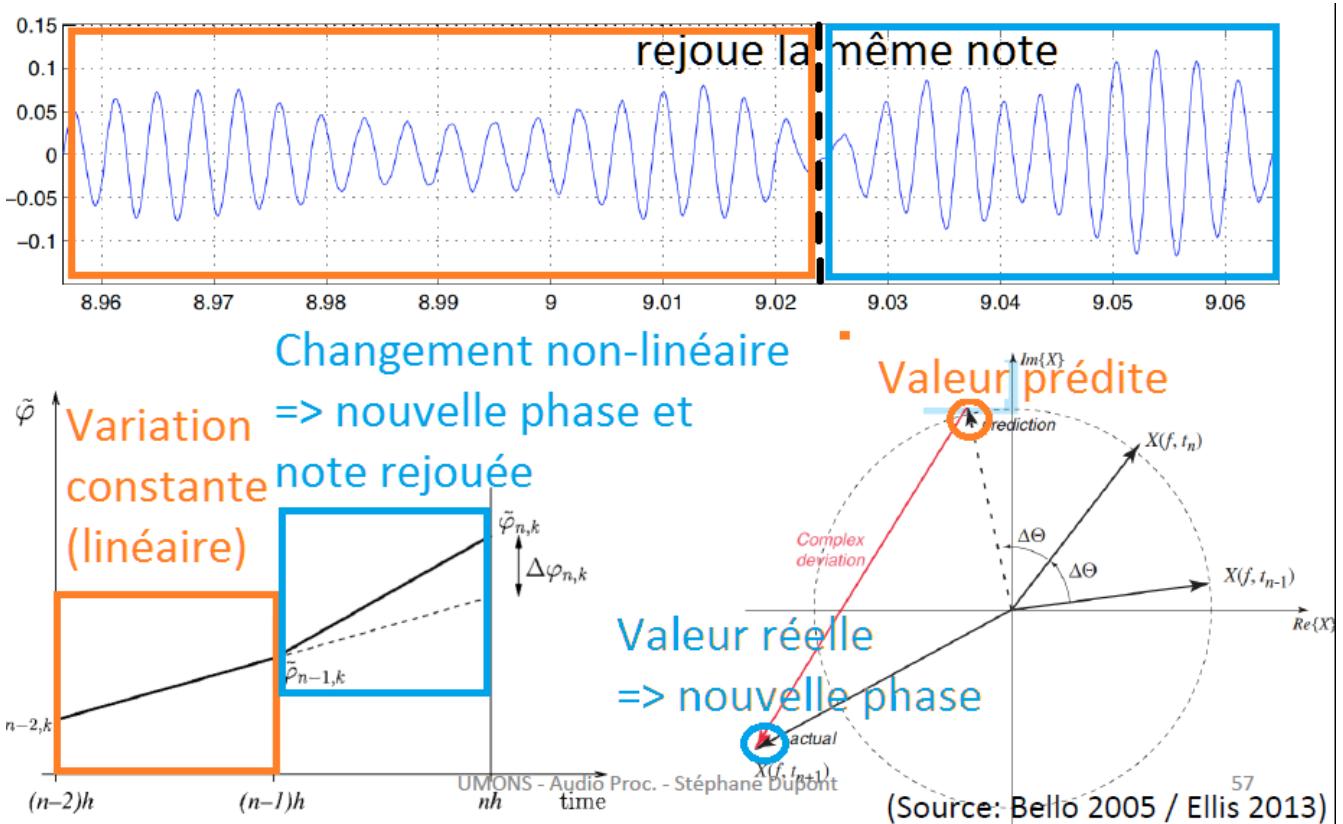
$$SD(n) = \sum_{k=\frac{N}{2}}^{\frac{N}{2}-1} \{H(|X_k(n)| - |X_k(n-1)|)\}^2$$

: $H(x) = (x + |x|)/2$, On ne garde la valeur que si ça augmente



On a encore un problème à ce niveau-ci : Si un joueur de violon rejoue une même note, on doit pouvoir détecter la nouvelle note, mais la différence spectrale n'y verra rien car ce seront presque les mêmes composantes en fréquences qui seront jouées. Par contre, on aura un changement de phase car on recommence à jouer la note.

Une analyse trame par trame implique que la phase va changer à chaque analyse. Cependant, on peut prédire la phase de la prochaine analyse car elle augmente de manière linéaire jusqu'à retomber à zéro (évolution cyclique). On va donc estimer qu'il y a une nouvelle note lorsqu'il y a un changement abrupt (non-linéaire, différent de la valeur prédictive) de phase. Tout ceci est illustré dans l'exemple suivant.



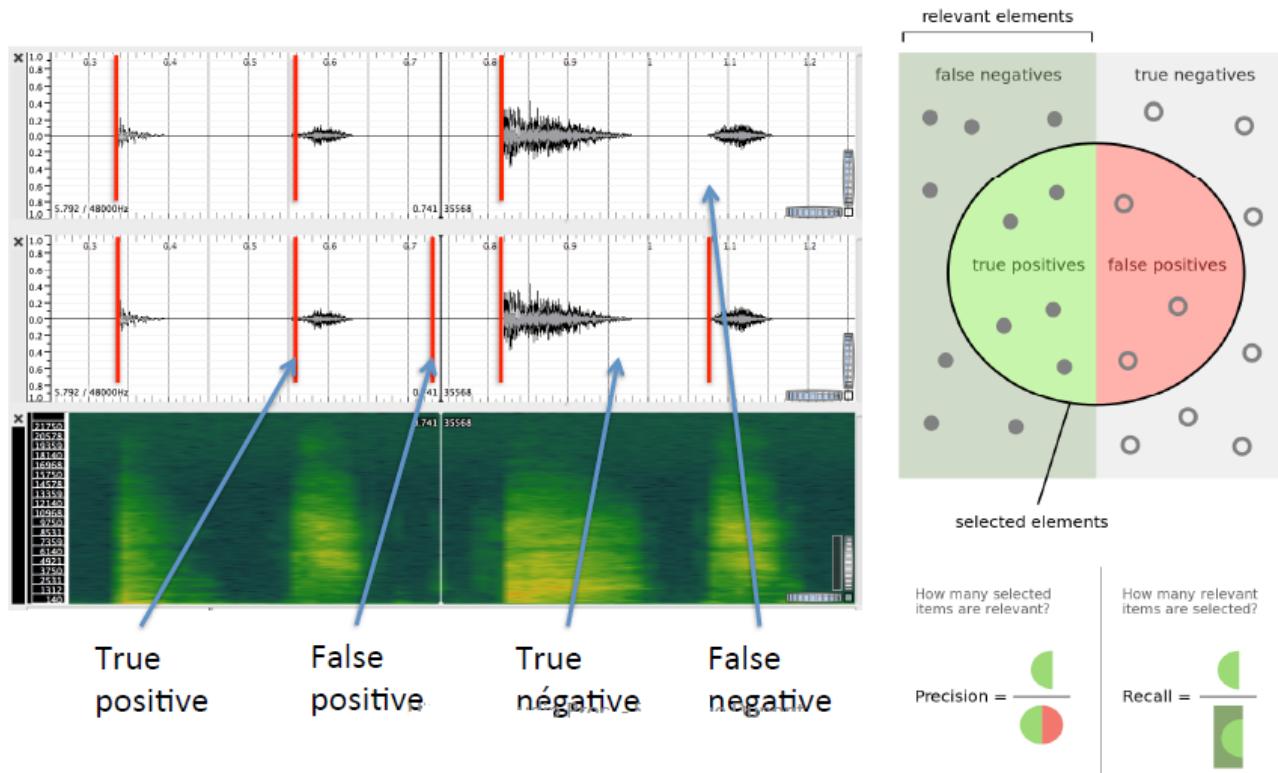
En pratique on a besoin de faire une analyse complexe de spectre dans la différence de fréquence et analyse de phase car sinon, la phase pourrait être trop sensible au bruit.

2.2.3 Sélection d'onset

Une fois que la fonction de détection d'onset est établie, il faut sélectionner les pics qui seront considérés comme onset, et donc comme début de segment. Ça peut se faire par une technique de **sélection par seuil** ou par **sélection intelligente de pic**. Cependant, un traitement supplémentaire est nécessaire avant de pouvoir faire cela. Un **post-traitement de la fonction de détection** est nécessaire afin de palier au changement d'orchestration de la chanson qui pourrait influencer la qualité des pics. On peut y appliquer un filtre passe-bas pour la lisser, un filtre passe haut pour mettre en évidence les pics, on peut aussi la normaliser par l'écart-type, ...

2.2.4 Évaluation des performances

Deux méthodes sont utilisées en général pour évaluer les performances d'un détecteur : **Précision/Rappel** et la **F-mesure**. La F-mesure valant $\frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$



2.3 Tempo

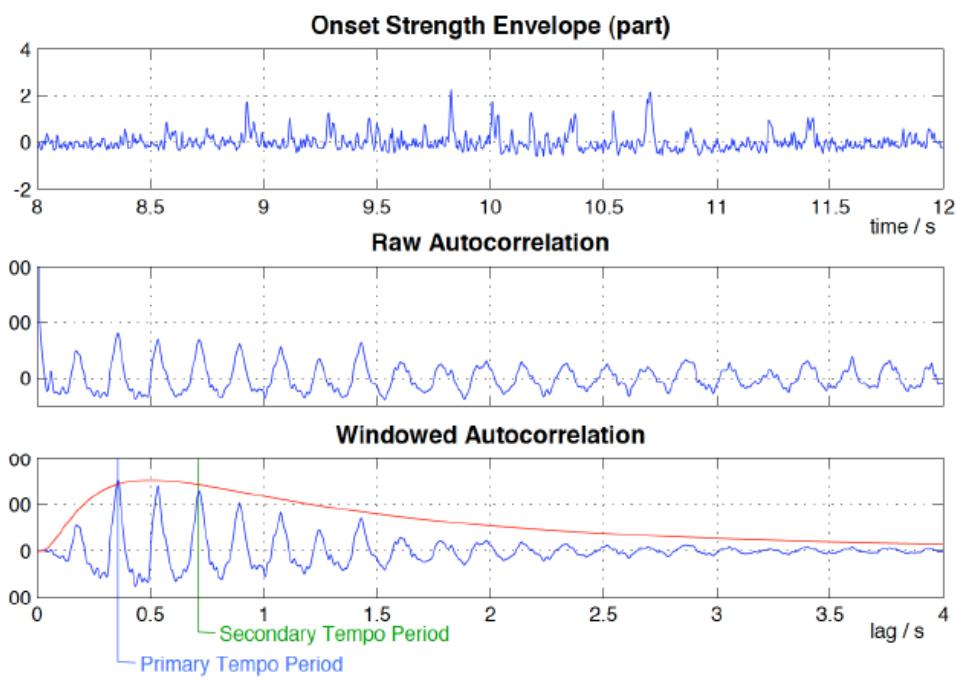
Jusqu'à présent, on s'est intéressé au rythme de manière locale, mais on voudrait maintenant avoir le **tempo global** de la chanson (e.g. 90BPM). Pour ce faire il faut détecter les onset. De plus, il faut s'intéresser aux "tatum", qui sont des onsets entre les beats importants pour la perception. Typiquement, le **tempo** représente la périodicité entre les onsets de grande intensité.

2.3.1 Méthode initiale

A la base, la recherche du tempo se faisait par **recherche de fréquence**. On soumettait les onsets à des **résonateurs**, qui cherchaient à quelle fréquence les résonateurs résonnaient le mieux.

2.3.2 Approche d'auto-corrélation

On va chercher la **corrélation entre les onsets** en faisant glisser le graphe avec un certain délai à faire varier (on superpose un onset avec tous les autres, mais décalés, pour connaître sa corrélation avec ceux là). Le résultat donne les **tempo possibles**, mais avec les "octaves" possibles. C'est pourquoi on va pondérer le tout selon les tempo les plus utilisés en musique ou pour le genre musical en question (**introduction de connaissance musicale**).



2.4 Beats

Une fois le tempo détecté, on peut s'attaquer à la détection des **beats** qui ne sont finalement que les **pics principaux d'onset séparés par des périodes de tempo**. Il faut donc chercher à **maximiser la fonction**

$$C(\{t_i\}) = \sum_{i=1}^N O(t_i) + \alpha \sum_{i=2}^N F(t_i - t_{i-1}, \tau_p) = \text{onsets} + \alpha \cdot \text{régularité du tempo}$$

α étant un facteur d'échelle, permettant de l'importance de la régularité du tempo. Le problème est que cette fonction demande un temps exponentiel à maximiser. On va donc avoir recours à de la **programmation dynamique** (stocker les résultats intermédiaires parce qu'ils se répètent dans les formules) afin de résoudre le problème de manière itérative. Une autre technique utilisée est de faire la corrélation des onsets avec un modèle collant au tempo, avec une certaine tolérance à l'erreur.

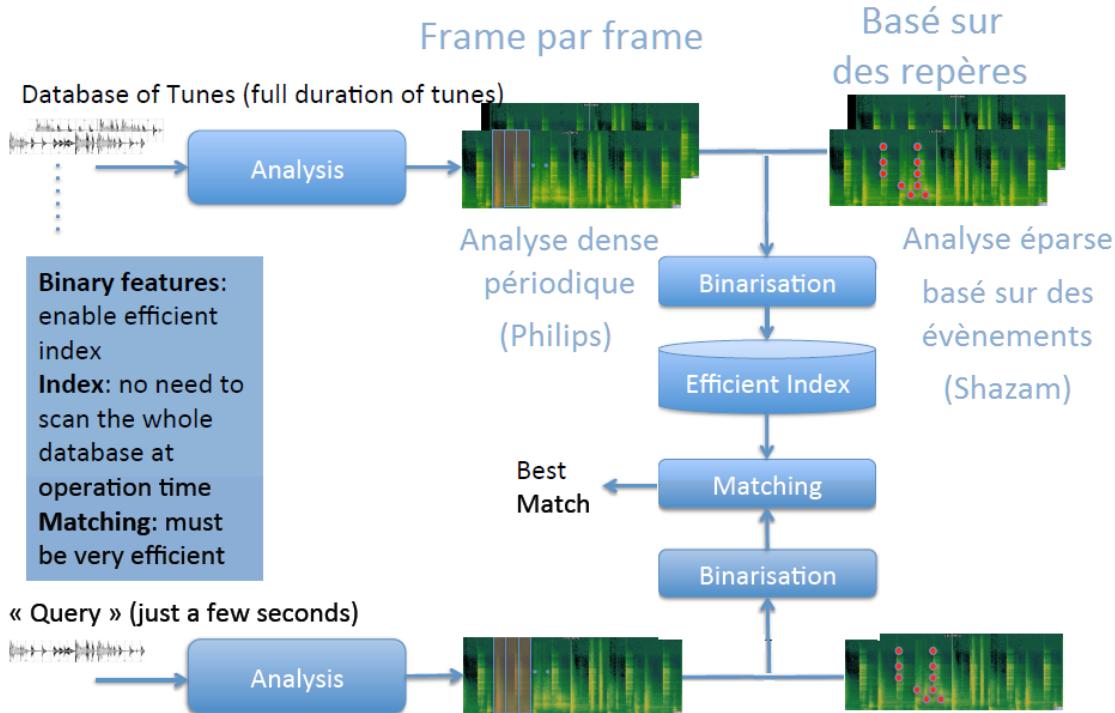
Une chose plus compliquée à faire est de détecter les **downbeats**, qui sont les **beats qui démarrent une mesure**. Pour ce faire, on a besoin de plus de connaissances musicales car sa position varie selon les genres musicaux. Des fonctions de signatures temporelles seraient nécessaires.

2.5 Conclusion

Note générale sur l'analyse musicale: Il y a beaucoup plus de techniques que celles vues au cours pour extraire des informations d'une musique.

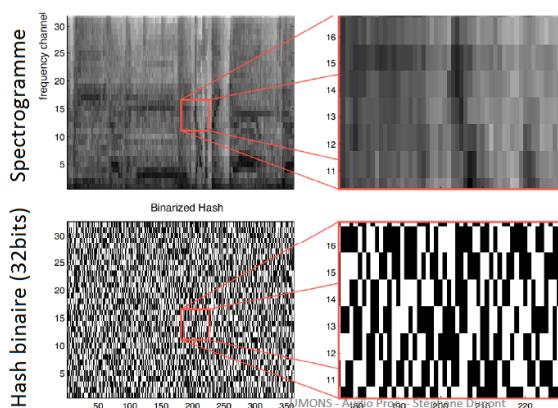
3 Identification de musique

Principe : reconnaître une musique par son titre et artiste en n'écoulant que quelques secondes. Pour ce faire on va chercher une "empreinte digitale" unique pour la musique, qui pourra être reconnue en écoutant une partie de la musique.

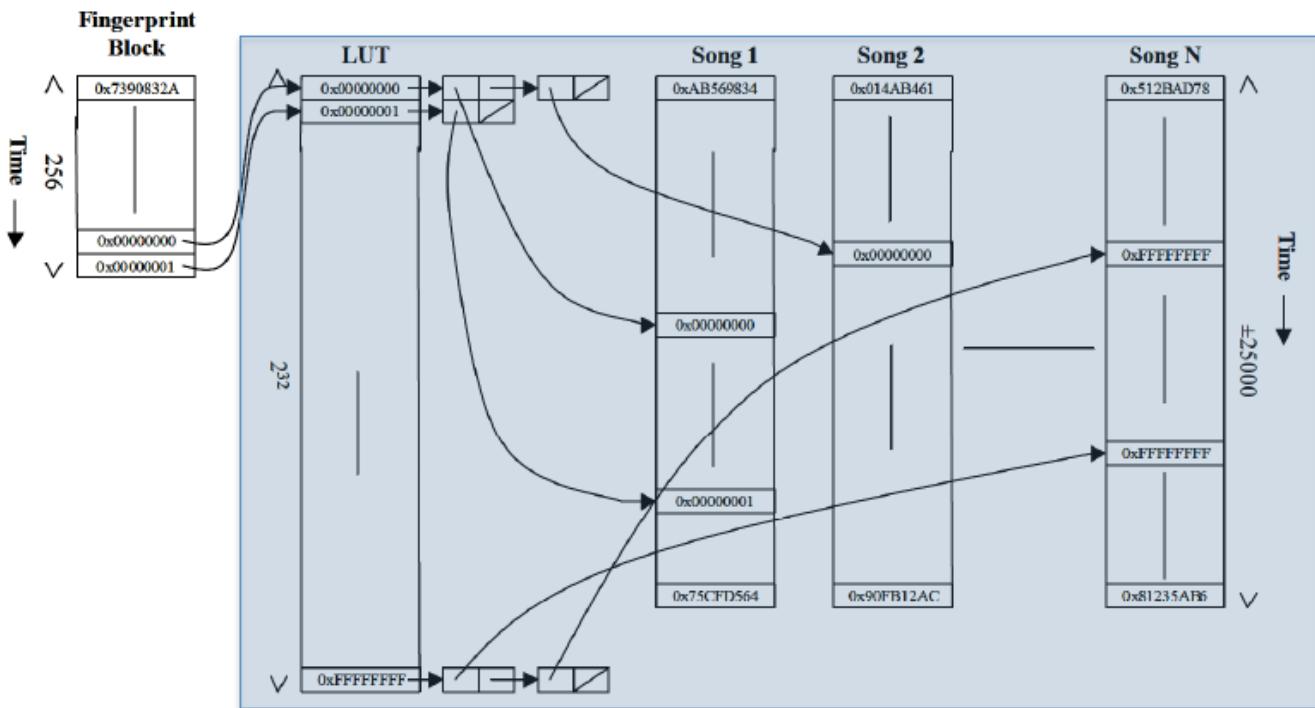


3.1 Analyse par frame

La méthode initiale consiste à encoder chaque frame sur 32 bits. Les 32 bits représentent chacun une **bande de fréquence d'une FFT**. Pour chaque bande, s'il y a un **changement local** conséquent par rapport à la FFT précédente, le bit correspondant est mis à 1. Cette analyse se fait sur des frames plus longues que d'habitude ($\sim 100\text{ms}$) afin d'être plus **robustes**.



On utilise ensuite ces 32 bits par frame pour retrouver la musique. La méthode principalement utilisée est une "lookup table" (LUT) qui a une table d'entrée contenant toutes les combinaisons possibles d'empreintes 32 bits (2^{32} possibilités). On a ensuite accès à n'importe quel endroit de n'importe quelle chanson et on peut commencer à chercher un match.



En effet, utiliser une recherche en force brute est débile et prendrait trop de temps. De plus, grâce à la LUT, on peut facilement contourner les erreurs à 1 bit près. En inversant un par un les bits d'une séquence de 32 bits, on obtient 32 nouvelles séquences que l'on peut entrer dans la LUT. Néanmoins, le bruit peut poser problème pour retrouver correctement ces vecteurs 32 bits.

3.2 Analyse par repères (landmarks)

Le principe est de prendre un spectrogramme (fig. 1A), d'en extraire les pics, et de se servir de chaque pic comme " ancre" (anchor) (fig. 1B). On va ensuite relier chaque ancre avec les autres pics (fig. 1C) et former des triplets (fréquence de démarrage, fréquence de fin, différence de temps) (fig. 1D). Ce sont ces triplets qui vont former nos repères dans la reconnaissance de musique. En pratique, on a pas besoin de trouver beaucoup de landmarks communs pour identifier une musique.

Ces repères sont robustes car ils se reposent sur les pics du spectrogramme qui sont les fréquences principales et les harmoniques de celles-ci, et sont donc les composantes les plus fortes en énergie dans le spectre. Les valeurs des triplets sont discrétisées selon 256 bandes de fréquences et 64 marquages temporels.

On a alors 22 bits par repère (fréquence, fréquence, temps) \rightarrow (8 bits, 8 bits, 6 bits).

Hash = (start frequency, end frequency, time difference)

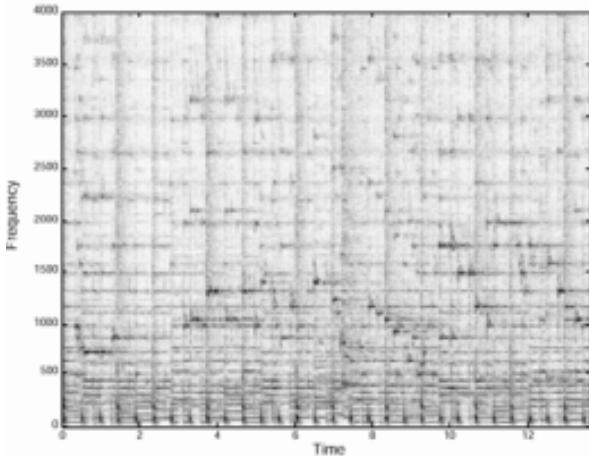


Fig. 1A - Spectrogram

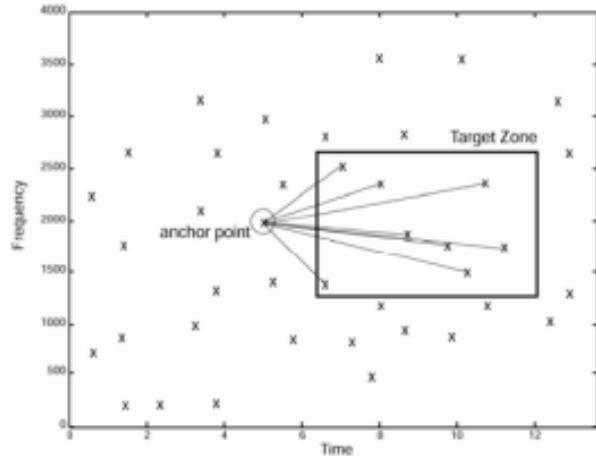


Fig. 1C - Combinatorial Hash Generation

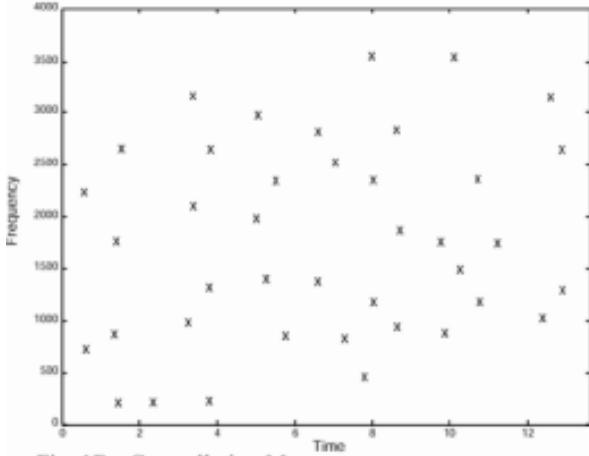


Fig. 1B - Constellation Map

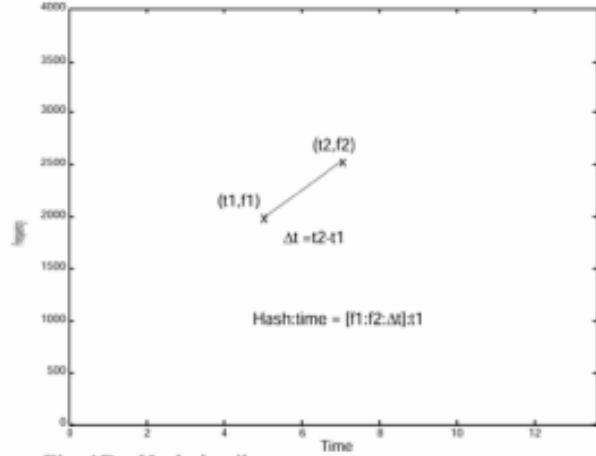
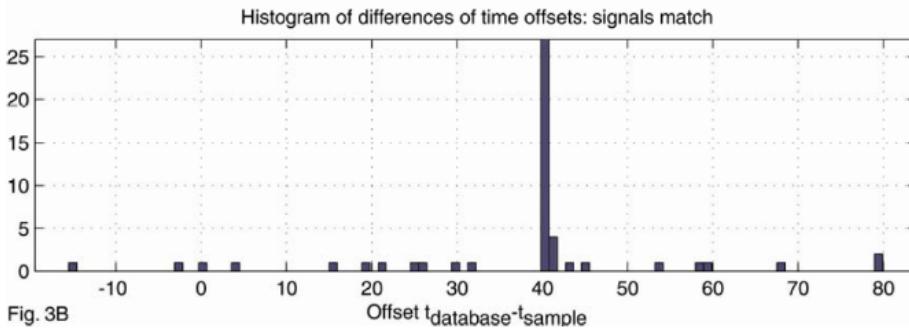
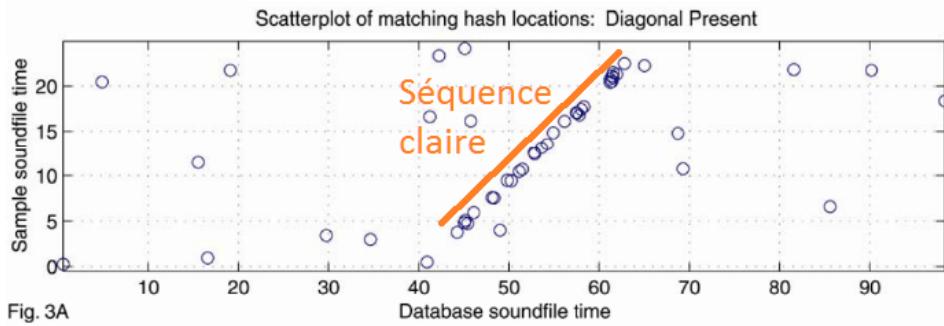


Fig. 1D - Hash details

La détection de pics dans le spectrogramme peut être ajustée en réglant des paramètres afin de détecter moins ou plus de pics, et ce afin de limiter le nombre de repères par chanson. Si de tels paramètres sont utilisés pour les chansons dans la base de données, alors ces paramètres doivent être appliqués également aux chansons écoutées (qui doivent être reconnues).

Ensuite, les repères sont présentés à une LUT qui va encore une fois retourner toutes les chansons qui contiennent le repère donné. Cependant, pour reconnaître la musique, il va falloir faire appel au temps dans la chanson auquel l'ancre du repère a été détecté. En effet, on va faire une comparaison temporelle entre le moment où le (l'ancre du) repère a été détecté et le moment où le repère se trouve dans la chanson de la base de données. La comparaison devrait donner une séquence claire, permettant d'avoir un grand nombre de match sur une courte période, et donc un grand nombre de match dans un histogramme comptant le nombre de matchs par fenêtre de temps.



Cette technique permet d'obtenir de très bon résultat, malgré un bruit élevé (SNR de 3dB) et une compression élevée. Par contre, la technique est sensible à la modification de la vitesse de la musique originale (car le repère dépend du temps de par sa 3^{eme} composante) ou si on modifie le pitch de la musique (car le repère dépend fortement des fréquences fondamentales).

4 Réseaux de neurones (profonds)

Pour cette partie, c'est la reconnaissance vocale (ASR) qui va être étudiée. La technique de base pour faire de l'ASR consiste à suivre 3 étapes :

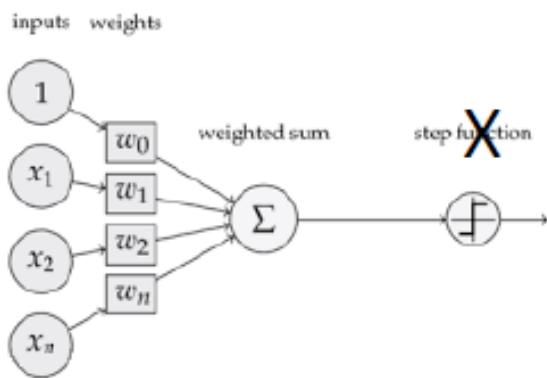
1. Extraire des informations utiles des fichiers audio (FFT, Spectrogramme, ...).
2. Calculer les probabilités d'obtenir tel ou tel phonèmes à partir des informations extraites.
3. Calculer les probabilités d'obtenir un tel mot ou une telle phrase selon une suite de phonèmes.

Les deux principales faiblesses de cette technique est (1) qu'il faut évaluer à la main l'utilité des informations à extraire de l'audio pour chaque nouvelle tâche et (2) qu'il n'y a pas d'optimisation globale de ces 3 étapes, il n'y a qu'une optimisation locale pour chaque étape. De plus, les performances n'égalent pas les performances humaines. Pour remédier à ces problèmes, on va faire appel à des réseaux de neurones profonds qui vont effectuer les 3 étapes eux-même, permettant une optimisation globale et une facilité d'utilisation (car pas d'analyse des features intéressantes). Ce faisant, on se débarrasse de la partie analyse du signal, mais aussi des suppositions faites sur les modèles probabilistes et sur les features qui pourraient être intéressantes.

4.1 Neurone

Avant de créer un réseau, on va commencer par définir ce qu'est un neurone. Deux fonctions intéressaient les chercheurs à la base : (1) **le symbolisme**, permettant de représenter les données et la connaissance humaine, et (2) **le connectionisme**, permettant à des cellules virtuelles d'interagir et leur capacité d'apprentissage.

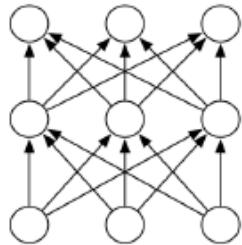
Le principe d'un **neurone** est de faire une **somme pondérée** de toutes les dimensions de la donnée entrée, d'y ajouter éventuellement un **bias**, et ensuite d'y **appliquer une fonction** permettant de classer la donnée comme appartenant (1) ou n'appartenant pas (0) à une classe spécifique. On a donc un **classificateur binaire**. On peut noter que si les données ne sont pas séparable linéairement, on peut toujours trouver une **transformation de l'espace** (en ajoutant des dimensions par exemple) pour rendre les données séparables linéairement dans ce nouvel espace.



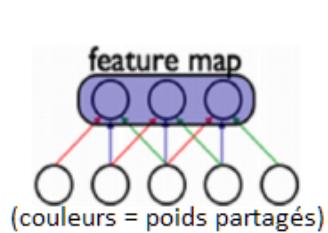
Entraîner un tel réseau consiste à **ajuster les poids** de la somme. On peut noter que si un ensemble de poids classifie parfaitement un ensemble de donnée, ces même poids multipliés par une constante classifiera les données tout aussi parfaitement. Il n'y a donc **pas de moyen de connaître les poids réellement optimaux** (car bases de donnée finies et incomplètes par rapport aux données réelles possibles). Les techniques modernes cherchent à éviter ce problème au maximum.

4.2 Réseau

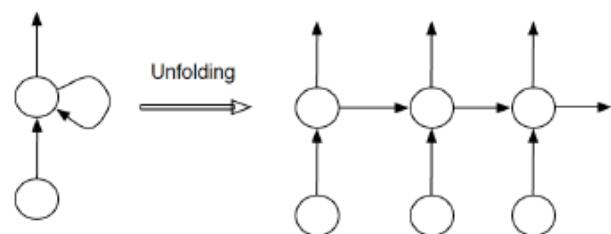
Il existe plusieurs architectures connues :



Feed-forward neural network



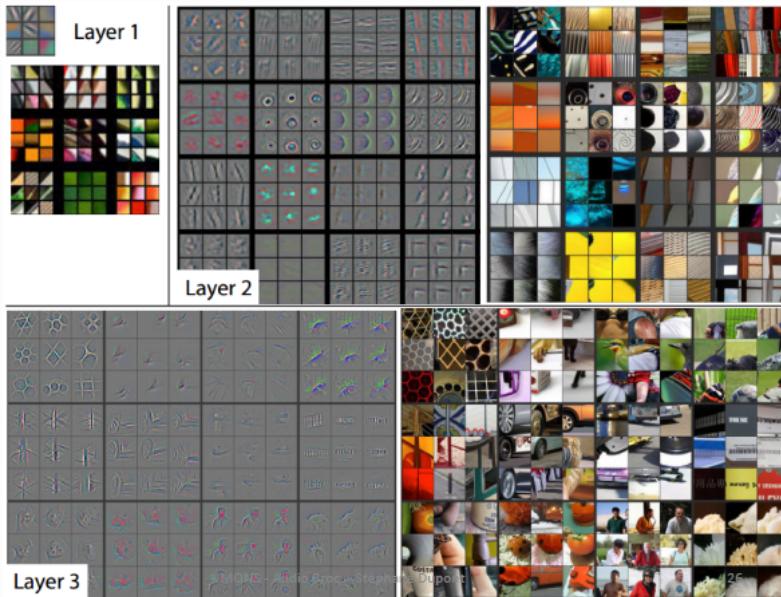
Convolutional neural network



Recurrent neural network

Lorsque beaucoup de neurones et de couches de neurones sont utilisées, il devient impossible d'ajuster les poids à la main. On va utiliser un algorithme de back-propagation calculant des gradients entre tous les poids et bias afin de les ajuster de manière automatisée dans une phase d'apprentissage. tout cela permettra d'optimiser une fonction de coût utilisant ces paramètres.

On peut voir un neurone comme étant un filtre audio, défini par ses poids, calculant sa résonance avec l'entrée (par un calcul de cross-corrélation). Il en ressort une "puissance de résonance", qu'on peut appartenir aux probabilités en sortie du neurone. C'est par cette comparaison que l'on peut observer les résultats des couches intermédiaires d'un CNN. (On multiplie les pixels par les poids des neurones). On peut donc voir les réseaux de neurones comme une manière de construire (durant la phase d'entraînement) d'appliquer une cascade, une banque de filtres.



Perceptron

$$f(x) = h \left(\sum_{i=1}^M w_i x_i + w_0 \right)$$

FIR linear digital filter

$$y(n) = \sum_{i=1}^N b_i x(n-i)$$

Cross-correlation

$$(f * g)(n) = \sum_{m=-N}^N f(m) g(m-n)$$

Linear transformation

4.2.1 Réseaux de neurones convolutionnels (CNN)

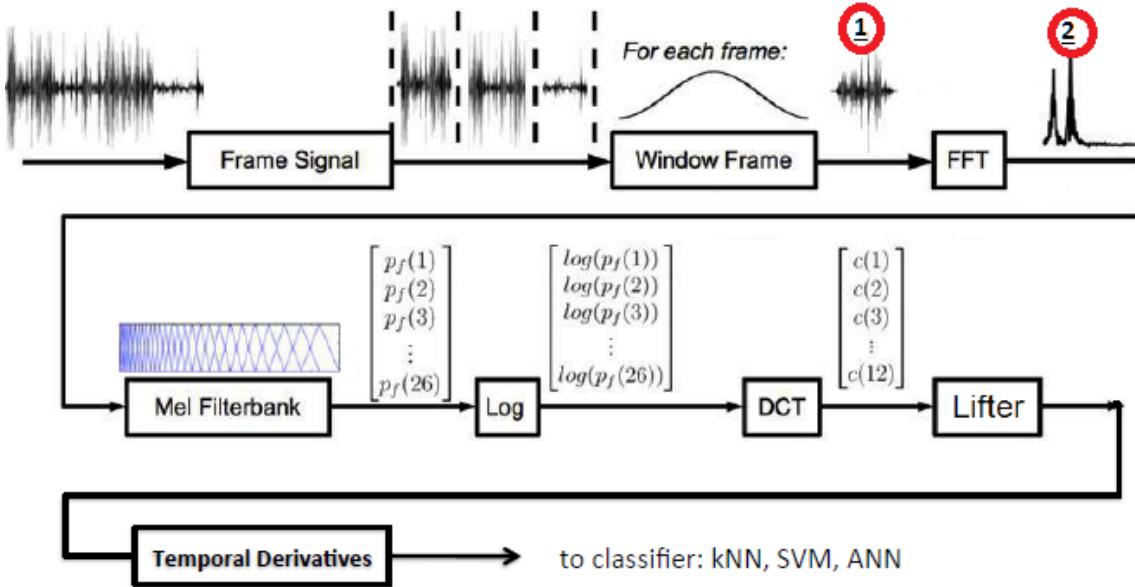
Les réseaux de neurones convolutionnels permettent de prendre des entrées avec énormément de dimensions. En effet, celui-ci :

- Regroupe les poids entre certains neurones (pour simuler une convolution => appliquer un même filtre à une zone de l'image). Ceci est plus ou moins équivalent à faire parcourir un seul neurone sur une zone de l'entrée (e.g. une zone d'image).
- Mets certains poids à 0 afin d'ignorer quelques dimensions inutiles.
- A des couches de "pooling" permettant de faire une moyenne des sorties de la couches précédentes et donc de réduire la taille du réseau progressivement (~downsampling...). Pour ce faire, ces couches de pooling possèdent des neurones qui possèdent tous le même poids (parce que faire la moyenne c'est faire une somme pondérée avec un poids constant).

4.3 Reconnaissance audio

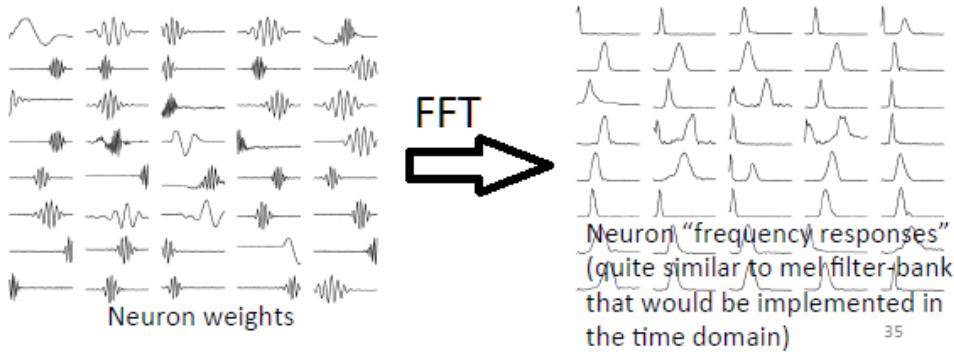
La technique de base utilisée est de calculer les MFCCs. Cette technique est déjà inspirée par le biologique (résolution des basses fréquences plus élevée par les filtres de Mel).

L'image suivante illustre les étapes parcourue pour trouver ces coefficients :



4.3.1 Amélioration 1

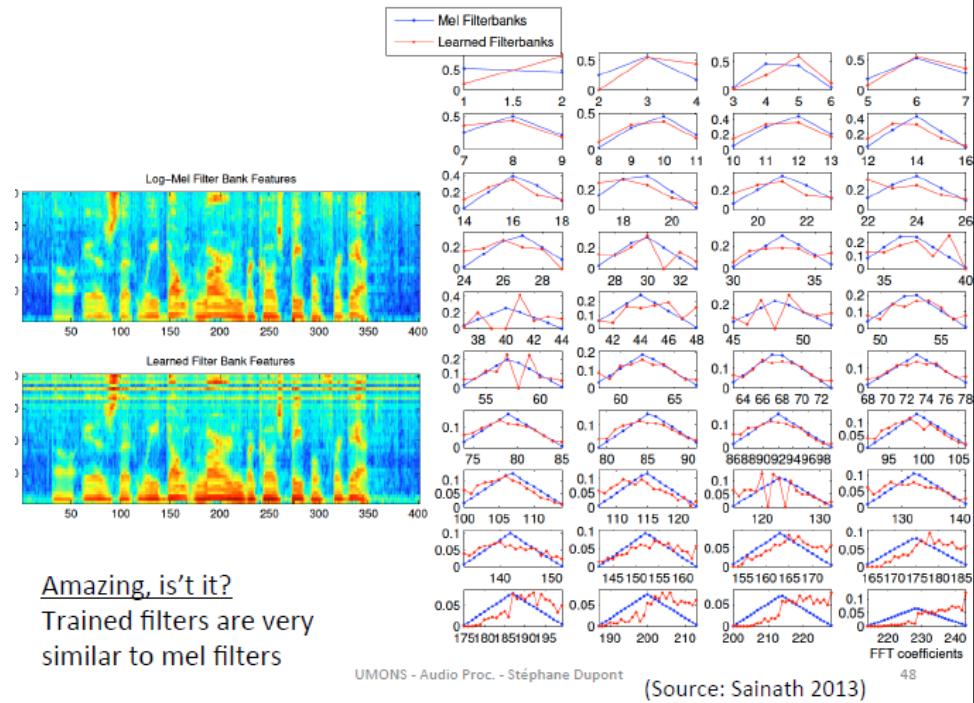
La première amélioration (cf. (1) dans l'image précédente) vise à mieux analyser le signal audio en utilisant une Restricted Boltzmann Machine (RBM, une architecture de réseaux de neurones permettant un apprentissage non-supervisé). Cette analyse va permettre de creuser les détails de plusieurs bandes de fréquences en même temps. En effet, les phonèmes peuvent être détectés grâce à leurs deux fréquences de bases (formants), et les grouper déjà à ce niveau-ci permet une analyse plus poussée.



4.3.2 Amélioration 2

Une autre amélioration (cf. (2) de l'image sur les MFCCs) serait de ne pas analyser le signal audio brut, mais d'analyser la sortie de la FFT en la donnant en entrée à un réseau de neu-

rones profond. Le but étant de remplacer la banque de filtre de Mel par des "filtres" calculés par un réseau de neurones. On peut voir dans l'exemple suivant que les filtres obtenus sont très proches des filtres de Mel, et ces derniers sont le résultats d'années de recherche dans le domaine. On a donc une structure qui permet de trouver les filtres optimaux (ce qui n'était pas le cas pour les filtres de Mel) automatiquement, sans avoir besoin d'années de recherche, juste de l'entraînement.



Appliquer cette technique en sortie de la fenêtre de pondération (au (1) de l'image MFCC) également permettrait d'améliorer encore les résultats.

4.3.3 Amélioration 3

Appliquer une RBM à la sortie de la FFT permet d'avoir des résultats assez haut-niveau en sortie. (analyse fréquence par fréquence).

4.4 Conclusion

Les réseaux de neurones profonds permettent d'entraîner automatiquement des systèmes complexes "All-In-One", mais demande énormément de données afin d'être précis, peuvent vite devenir des boîtes noires, et les étudier demande beaucoup de temps car les architectures sont nombreuses et encore en phase de tests.

Partie III

D'Alessandro – Synthèse musicale et audio temps-réel

1 Question 1

1.1 Question

Décrivez les notions de “temps-réel” vues des points de vue du système informatique et de l’utilisateur. À l'aide d'exemples concrets, illustrez brièvement comment ces deux perspectives affectent la latence acceptable d'un système audio temps-réel.

1.2 Éléments de réponse

Temps-réel ordinateur (hard real-time)

Le principe est que l'ordinateur garantisse une réponse dans des contraintes de temps spécifiques. C'est à dire que l'ordinateur a Δt secondes pour fournir une réponse, sans quoi la tâche échoue. L'algorithme peut éventuellement soumettre une réponse sous-optimale juste pour être dans les temps.

Priority Inversion Lorsque l'on veut faire du hard real-time, il faut éviter au maximum les tâches qui sont plus lentes que votre tâche. C'est ce qu'on appelle l'inversion de priorité. Ces tâches dépendent souvent d'interruptions systèmes qu'elles doivent attendre, ou encore de contraintes physiques comme l'attente d'un disque dur, etc... Exemple :

- Allocation mémoire (malloc)
- mutex / semaphores
- Accès disque dur
- Appel sur la GUI
- Accès réseau.

Toutes ces tâches sont dépendantes d'autres tâches pour être complétées, c'est pourquoi on ne peut pas garantir un moment auquel elles seront terminées.

Impact sur le feeling humain Au plus une machine va se rapprocher du hard-realtime, au moins elle sera 'human-friendly'. En effet, on va éviter au maximum les interfaces graphiques, les interactions avec l'utilisateur, etc...

Temps-réel humain (HCI real-time)

Cette définition de temps réel s'applique à la sensation utilisateur. Typiquement, on va faire en sorte que l'ordinateur ou que la tâche réponde dans un laps de temps acceptable pour un humain. Cette fenêtre de temps est définie par la notion de retard ou de confusion temporelle de l'humain, et ce situe entre 1 et 100ms.

Perception L'être humain utilise plus que 5 sens pour interagir avec le monde, on peut citer par exemple le retour visuel, le retour auditif, la conscience de la position de ses membres (proprioception), le retour haptique, le modèle balistique (trajectoire emprunté par ses membres pour effectuer une action), etc... Parler d'instantanéité revient alors à trouver un juste équilibre entre tous ces sens.

Retour audio En particulier, les décalages audio pourraient être entendus par un homme, c'est pourquoi on va chercher à éviter de lancer des tâches dont on ne peut mesurer le temps d'exécution dans le thread audio. Garder ce thread audio aussi rapide et fluide permettra un sentiment d'instantanéité à l'oreille humaine.

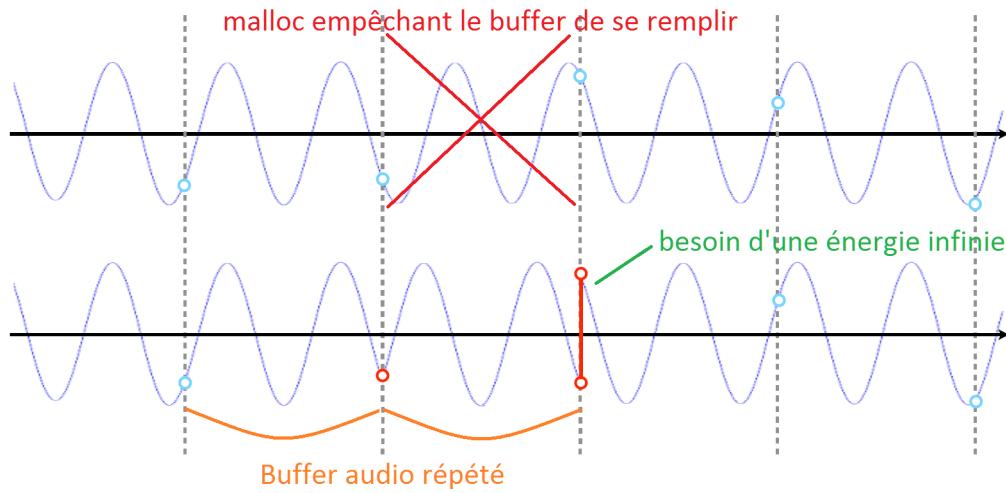
2 Question 2

2.1 Question

Vous êtes dans la situation où, pour produire un son en temps-réel, vous devriez allouer une ressource mémoire importante. On vous a dit, en passant, qu'il "fallait éviter les malloc(), car ça faisait glitcher l'audio". Veuillez re-contextualiser cette mise en garde: Qu'est-ce qu'un audio glitch? Quel est le lien entre allocation de mémoire et audio glitch? Comment éviter cela?

2.2 Éléments de réponse

L'utilisation du malloc dans le thread audio représente une inversion de priorité. En effet, le thread audio se veut temps-réel (haute priorité) au niveau de la perception humaine, et malloc prend un temps indéterminé (faible priorité) à s'exécuter. Comme le thread audio attend un certain laps de temps avant de prendre des entrées et sorties audio, on risque de 'rater le bus' et de ne pas envoyer la sortie à temps (frame drop). Le buffer audio va alors répéter la dernière séquence audio, provoquant un glitch, car le décalage provoqué dans la sinusoïdal demanderait théoriquement une énergie infinie dans les hautes fréquences, ce qui provoque un son dérangeant et audible. De plus, l'humain est sensible aux changements de phase (mais pas à la phase en elle-même).



Pour arranger ce problème, on peut simplement ne pas utiliser de malloc pendant l'exécution du thread audio, mais sur un autre thread, ou avant que le thread audio ne se lance. Si ça a du sens, on peut aussi utiliser des structures de données ne demandant pas ce genre d'allocation mémoire, comme un ring buffer (il faut alors que toutes les données ne soient pas à garder).

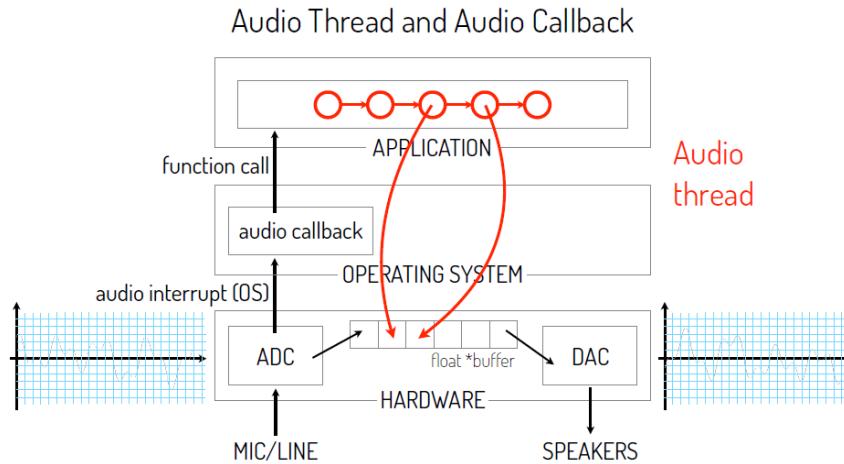
3 Question 3

3.1 Question

Comment, sur un système d'exploitation moderne, les applications gèrent-elles leur accès aux ressources audio? Expliquer, dans son contexte, ce qu'est le "callback audio", ce qu'il fait et son lien avec les entrées/sorties audio d'un ordinateur typique.

3.2 Éléments de réponse

Dans un ordinateur modernes, énormément de processus fonctionnent grâce aux interruptions système. On pourrait même dire qu'à part le "main" de l'application, tout le reste est fait d'interruptions. C'est également comme ça que le thread audio fonctionne. Le callback audio est une fonction qui est déclenchée par l'interruption "audio ready" du driver audio. Le buffer est envoyé vers les haut-parleurs à intervalle régulier, et n'attend pas que les applications ait fini de le remplir.



En pratique, le buffer envoie 64 échantillons à la fois, ce qui représente $\frac{64}{44100} = 0.0015s$ de décalage entre deux sorties audio. On se trouve bien dans la fenêtre idéale pour l'impression humaine d'instantané.

4 Question 4

4.1 Question

Dans mon application, j'ai inséré un filtre numérique très complexe et le calcul de ses coefficients prend, sur une machine typique, 20 ms. Je souhaite bien évidemment que le taux de rafraîchissement de ces coefficients soit le plus proche possible du minimum, soit 50 Hz. Décrivez deux approches différentes permettant de réaliser ce filtrage sans aucun glitch audio.

4.2 Éléments de réponse

Méthode 1 La première méthode est d'utiliser un thread de calcul séparé du thread audio. Ce thread de calcul sera relié au callback audio par un ring buffer. Il faut alors faire attention à ce que la tête d'écriture du buffer ait toujours de l'avance sur la tête de lecture. Pour ce faire, il faudra faire attention à ce que le filtre traite assez de données en même temps pour pouvoir avoir assez de données "en avance" que pour remplir le buffer à temps à chaque callback. On peut également retarder la première sortie audio en remplaçant autant que possible le ring buffer, et ensuite démarrer la sortie audio afin de "prendre de l'avance" et ne pas droper de frames. Pour que les calculs se rapprochent le plus possible du temps-réel, on va utiliser des types atomiques. Et bien sûr, ne jamais faire d'inversion de priorité.

Méthode 2 Dans cette deuxième méthode, on suppose avoir le contrôle sur la taille du buffer audio. Dans ce cas, on va chercher à avoir un buffer assez grand que pour attendre les données traitées par le filtre. La taille idéale de ce dernier peut se calculer facilement :

$$\frac{\text{nb_samples}}{\text{sampling_freq}} = 0.020s$$

Vu qu'on connaît en principe la fréquence d'échantillonnage, on peut calculer la taille du buffer à fixer. On peut éventuellement prendre une taille un peu plus grande pour accorder un peu plus de temps que 20ms. Par exemple, si appliquer le filtre à l'audio prend 1ms, on va plutôt chercher un buffer de 21ms. Tout ceci en ajoutant les bonnes pratiques énoncés dans la méthode 1 (types atomiques, ...).

5 Question 5

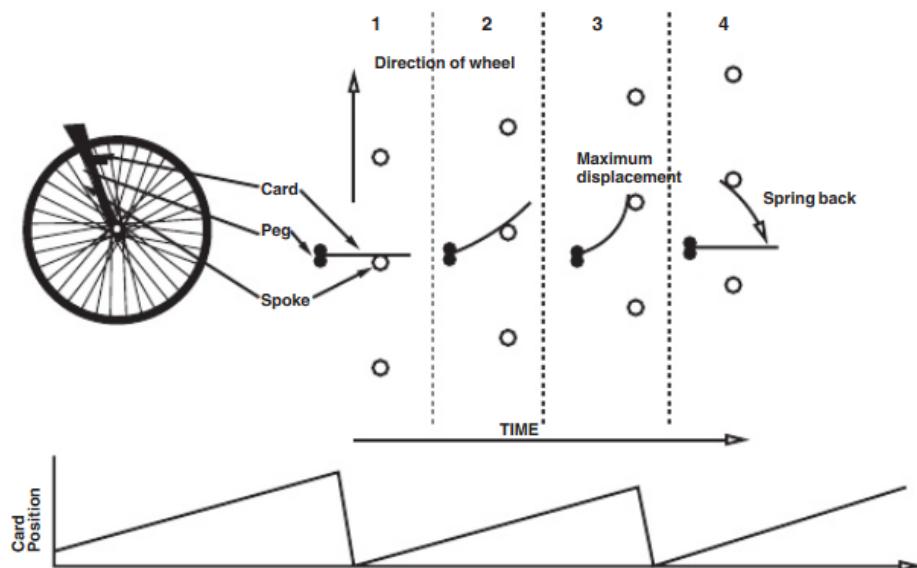
5.1 Question

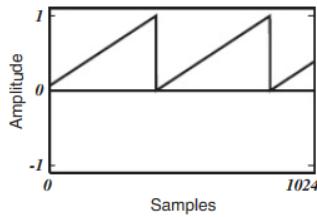
Décrivez en quoi la synthèse additive est un exemple simple de "phaseur + table d'onde" (en anglais: "phasor + wavetable"). À quoi le phaseur est-il associé en synthèse additive et pourquoi est-ce si important de piloter la synthèse par ce phaseur?

5.2 Éléments de réponse

La synthèse additive consiste à additionner des signaux pour en former des nouveaux. Par exemple, avec la lumière si l'on additionne du rouge et du vert on obtient du jaune. En audio, on peut reconstruire n'importe quel signal par l'addition de sinusoïde. Par exemple, un signal (presque) carré peut s'obtenir en additionnant une sinusoïde avec ses harmoniques (il faudrait les sommer à l'infini pour obtenir un vrai signal carré). En audio on utilise extrêmement souvent des sinusoïdes. Étant donné leur usage courant plutôt que de calculer un sinus "from scratch" on utilise des tables contenant les valeurs de la sinusoïde entre 0 et 2pi. C'est ce qu'on appelle "wavetable". Les valeurs vont de -1 et 1. Plus le système sera "puissant" plus l'on aura une une "wavetable" de grande taille (cela augmente la précision).

Un phaseur permet de créer un signal en dent de scie. Celui-ci oscille entre 0 et 1 de façon cyclique.





En changeant son amplitude, on peut le faire osciller entre 0 et 2π pour l'utiliser comme index pour la "wavetable". En changeant la fréquence du phasor, on peut changer la fréquence du sinus. Exemple: Si l'on prend un phaser avec une fréquence de 2Hz et que l'on veut additionner deux sinus d'une fréquence respective de 2 et 4 Hz, il suffira de faire wavetable(phaser(x)) + wavetable(phaser(2x)).

Il est donc possible de créer des signaux complexes avec ces deux outils combinés. De plus le phasor permet de fournir implicitement une base de temps pour les sinus. Un exemple visuel est donné [ici³](#).

6 Question 6

6.1 Question

Qu'est-ce qu'un buffer circulaire (ring buffer)? Illustriez son fonctionnement et son utilité dans deux exemples typiques: a) la visualisation de longs segments audio à l'écran; b) la synthèse de longues réponses impulsionales hors du thread audio.

6.2 Éléments de réponse

Un buffer circulaire est un buffer à indice circulaire. C'est-à-dire qu'à la place de dépasser la taille du buffer en incrémentant les indices d'accès au buffer, on va revenir au début de celui-ci. De plus, deux indices sont stockés en mémoire : la tête de lecture et la tête d'écriture. Ils sont beaucoup utilisés en audio lorsque l'on veut séparer écriture et lecture.

- a) Pour la visualisation de longs segments audio, le principe est d'écrire les signaux audio reçu ou envoyés au buffer audio dans le buffer circulaire, et d'utiliser la tête de lecture dans le thread visuel. L'avantage est qu'on peut régler la fréquence de rafraîchissement vidéo afin de ne pas avoir de ralentissement. On peut essayer de faire en sorte que le thread visuel se rafraîchisse en même temps que l'on reçoit l'audio à visualiser. On va laisser alors un peu d'avance à la tête d'écriture avant d'afficher les signaux audio au niveau visuel. Le tout sans jamais à avoir à allouer plus d'espace mémoire afin de ne pas avoir de ralentissement sur la visualisation.

³<http://www.animations.physics.unsw.edu.au/jw/phasor-addition.html>

- b) Même principe que la réponse à la question 5 – On va chercher à stocker les résultats des calculs juste un peu plus vite qu'on ne les lit dans le thread audio. Ceci permet de ne jamais avoir à attendre ces calculs et ainsi éviter les glitches audio.

7 Question 7

7.1 Question

Décrivez le principe de fonctionnement d'un guide d'ondes (en anglais: waveguide) de type Karplus-Strong. Pourquoi sonne-t-il comme une corde pincée? Quel mécanisme lui donne sa fréquence fondamentale? Comment est-elle calculée?

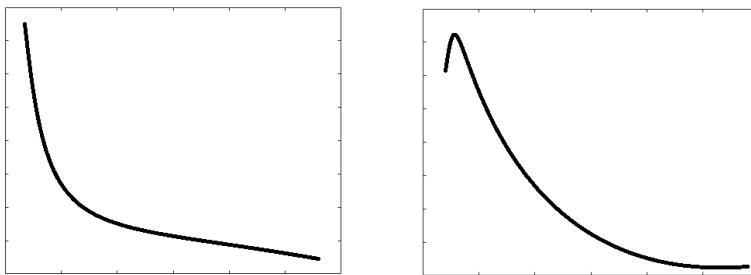
7.2 Éléments de réponse

Le principe d'un guide d'onde est de reproduire le son que pourrait produire un instrument à cordes. Pour ce faire, on utilise 3 éléments principaux :

- Un gain – Résonance.
- Un délai – Fréquence fondamentale de la note jouée.
- Un filtre – Timbre.

Ces trois éléments font parti d'une boucle mettant à jour les valeurs du buffer audio.

Gain Le gain est ce qui va faire durer la note (résonner). En pratique, le gain $\in [0, 0.999999\dots]$ car s'il valait 1, la note ne s'arrêterait jamais et s'il valait plus que 1, le son finirait par saturer car son amplitude ne ferait que croître. Ceci va donner une décroissance continue (image de gauche), or, un vrai instrument à cordes donne plutôt une impulsion avant de décroître progressivement (image de droite). Appliquer cela est un point d'amélioration du modèle de base des guides d'ondes.



Délai C'est ce qui va définir la fréquence fondamentale du son. En pratique, c'est la longueur du buffer contenant les segments audio (bruit blanc) qui va définir la note jouée. En effet, il y a l'idée de répéter un même son (principe de sinusoïde) et au plus le buffer sera petit,

au plus la note sera aigüe car on va répéter le buffer plus de fois (pour une même fréquence d'échantillonnage). Typiquement, on a :

$$\frac{\text{sampling_rate}}{\text{frequency}} = \text{buffer_size}$$
$$\Leftrightarrow$$
$$\frac{\text{sampling_rate}}{\text{buffer_size}} = \text{frequency}$$

Filtre Typiquement, on va utiliser un filtre de type "biquad" (equalizer) pour influencer le timbre du son. De cette manière, on peut affecter la manière dont le son 'sonne' (type guitare, type mandoline, type piano, ...). Le type de filtre utilisé peut donc influencer la qualité du son.