Travaux pratiques de filtrage numérique.

Master I, Informatique.

1• Objet du projet

Ce projet vise à expérimenter le filtrage numérique sous deux de ses formes : le filtrage dans l'espace fréquentiel (de Fourier) et le filtrage dans l'espace temporel. Considérez ce travail plus comme un projet que comme des travaux pratiques. Vous êtes libres de changer les paramètres de ce projet tant que vous réalisez du filtrage fréquentiel.
Pour réaliser ce travail, vous disposez de plusieurs fichiers son "BrokenGlass.wav", "Whistle.wav", "GammeFlute.wav" et "GammePiano.wav", et de deux fichiers source Wave.cpp et Wave.hpp contenant des fonctions permettant de lire et stocker de fichiers sons sous le format wav. Les fichiers son sont enregistrés en mono-voie pour simplifier le travail. Vous pouvez bien sûr tester cette procédure sur d'autres fichiers sons charges sur internet.
L'idée est de vous familiariser avec le traitement du signal sur cet exemple simple et facile d'accès. Vous essayerez de réaliser des filtres passe-haut, passe bas, basse-bandes, réjecteur de bande. Les fichier "Gamme" vous permettront de contrôler à l'oreille la qualité du filtrage obtenu (peut on retirer le "sol" de la gamme ?).
Pensez à vous munir d'un casque audio pour éviter une trop grande cacophonie dans la salle.

2• Rappels sur le son numérique.

En théorie l'oreille humaine perçois les fréquences de vibrations de l'air comprises entre 20 Hz et 20 000 Hz. En réalité cette perception varie d'un individu à l'autre.
Pour enregistrer un son et se placer dans les conditions de Shannon, on doit donc échantillonner (au minimum) à une fréquence double de la fréquence maximale qu'on souhaite représenter et pré-filtrer le signal pour éviter d'enregistrer des fréquences supérieures (éviter ce qu'on appelle le repliement de spectre). En standard, un fichier son est échantillonné à 44100 Hz (c'est à dire un peu plus que 2×20000 Hz). Pour la suite du TP, il sera important de repérer les fréquences des différentes notes de la gamme.
Pour un fichier .wav codé sur 8 bits, la valeur 127 correspond au 0 tandis que les valeur 0 et 255 correspondent aux maxima d'amplitude du son codé (voir annexe).

3• Transformée de Fourier numérique

En théorie, la transformée de Fourier d'un signal échantillonnée est continue. On montre cependant qu'il est possible de définir une transformation de Fourier échantillonnée d'un signal échantillonné qui est une transformation au sens où, à un signal échantillonné correspond une transformée unique et vice-versa. Cette transformée discrète n'est autre que la discrétisation de la transformée continue théorique.

Dans la pratique, soit un signal réel discret représenté par la suite de ses N échantillons (x_n) $(n = 0 \ldots N-1)$, sa transformée est la suite de valeurs complexes (X_k) $(k = 0 \ldots N-1)$ définies par : $X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi\frac{kn}{N}}$, et si on pose $X_k = a_k + ib_k$, on en déduit :

$a_k = \sum_{n=0}^{N-1} x_n \cos\left(2\pi\frac{kn}{N}\right)$ et $b_k = -\sum_{n=0}^{N-1} x_n \sin\left(2\pi\frac{kn}{N}\right)$ car les (x_n) sont réels.

La valeur de la transformée pour k=0 représente la part continue du signal discret (x_k). La k^{ieme} valeur de la transformée correspond à la composante du signal pour la fréquence $\frac{k}{N}f_e$, f_e étant la fréquence d'échantillonnage. Le phénomène de repliement du spectre fait qu'il existe une symétrie de la transformée du signal par rapport à la valeur $\frac{f_e}{2}$ (appelée aussi fréquence de Shannon), c'est à dire que $a_k = a_{N-k}$ et $b_k = b_{N-k}$. Cette symétrie vient du fait que le signal est réel. En fait, les deux valeurs complexes X_k et X_{N-k} représentent la composante fréquentielle du signal discret (x_k) pour la fréquence $\frac{k}{N}f_e$. Donc, lorsqu'on veux modifier le signal pour sa composante $\frac{k}{N}f_e$, il faut modifier de façon symétrique X_k et X_{N-k}.

Comme la transformée de Fourier discrète est une transformation, elle est réversible, son inversion s'écrit :

$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k e^{i2\pi\frac{kn}{N}}$. Comme (X_k) est complexe, la transformation inverse doit être un peu plus décomposée. A priori elle devrait donner deux composantes, l'une réelle et l'autre imaginaire :

$x_n = \frac{1}{N}\sum_{k=0}^{N-1} (a_k + ib_k)\left(\cos\left(2\pi\frac{kn}{N}\right) + i\sin\left(2\pi\frac{kn}{N}\right)\right)$

$x_n = \frac{1}{N}\sum_{k=0}^{N-1} a_k \cos\left(2\pi\frac{kn}{N}\right) - b_k \sin\left(2\pi\frac{kn}{N}\right)$ est la partie réelle de cette transformation, tandis que la partie imaginaire (qui devrait être nulle dans le cas présent) s'écrit :

$y_n = \frac{1}{N}\sum_{k=0}^{N-1} a_k \sin\left(2\pi\frac{kn}{N}\right) + b_k \cos\left(2\pi\frac{kn}{N}\right)$.

4• Filtre de Butterworth

4.1• Introduction

Le filtre de Butterworth est un filtre linéaire, conçu pour posséder un gain aussi constant que possible dans sa bande passante.
L'ordre du filtre détermine l'acuité de la coupure entre bande passante et bande de rejet.
Dans ce TP nous nous intéresserons à un filtre de Butterworth d'ordre 3.
Soit f_c la fréquence de coupure du filtre, et $\omega_c = 2\pi f_c$ la pulsation de coupure.

4.2• Equation du filtre continu

Le filtre en continu à comme fonction de transfert :

$$F(p) = \frac{1}{\left(\frac{p}{\omega_c}\right)^3 + 2\left(\frac{p}{\omega_c}\right)^2 + 2\left(\frac{p}{\omega_c}\right) + 1}$$

avec $p = j\omega$, et $\omega = 2\pi f$, f la fréquence.

4.3• Equation du filtre discret

Pour obtenir le filtre discret, la méthode la plus simple consiste à remplacer p par $F\left(\frac{2}{T}\frac{z-1}{z+1}\right)$ (où T est la période d'échantillonnage = $\frac{1}{f_e}$) pour obtenir F(z).

Posons $\alpha = \frac{\omega_c T}{2} = \pi\frac{f_c}{f_e} = \pi\frac{\omega_c}{\omega_e}$, donc $\frac{p}{\omega_c} = \left(\frac{2}{T\omega_c}\right)\frac{z-1}{z+1} = \left(\frac{1}{\alpha}\right)\frac{z-1}{z+1}$

En développant le calcul on a :

$$F(z) = \frac{\alpha^3(z+1)^3}{(z-1)^3 + 2\alpha(z-1)^2(z+1) + 2\alpha^2(z-1)(z+1)^2 + \alpha^3(z+1)^3}$$

On voit que cette expression dépend uniquement de z et α qui est (à π près) le rapport entre la fréquence d'échantillonnage $f_e = \frac{1}{T}$ et la fréquence de coupure du filtre $f_c = \frac{\omega_c}{2\pi}$. On remarque que, comme on doit avoir $f_c < \frac{f_e}{2}$, $\alpha < \frac{\pi}{2}$. Donc on règle la fréquence de coupure du filtre en faisant varier α dans l'intervalle $[0, \frac{\pi}{2}]$.

Pour obtenir une forme recursive, il faut passer F(z) en factorisation de z^{-1} :

$$F(z) = \frac{\alpha^3(1+z^{-1})^3}{(1-z^{-1})^3 + 2\alpha(1-z^{-1})^2(1+z^{-1}) + 2\alpha^2(1-z^{-1})(1+z^{-1})^2 + \alpha^3(1+z^{-1})^3}.$$

ou encore

$$F(z) = \frac{\alpha^3(z^{-3} + 3z^{-2} + 3z^{-1} + 1)}{A(\alpha) + z^{-1}B(\alpha) + z^{-2}C(\alpha) + z^{-3}D(\alpha)}$$

avec $A(\alpha) = (1 + 2\alpha + 2\alpha^2 + \alpha^3)$, $B(\alpha) = (-3 - 2\alpha + 2\alpha^2 + 3\alpha^3)$,
$C(\alpha) = (3 - 2\alpha - 2\alpha^2 + 3\alpha^3)$ et $D(\alpha) = (-1 + 2\alpha - 2\alpha^2 + \alpha^3)$.

5• TRAVAIL.

5.1• Création d'un fichier .wav

Vous allez créer un fichier La.wav en y synthétisant la note "la" (440 Hz) en mono pendant 6 secondes. Ecoutez le fichier obtenu et vérifiez la hauteur de la note et la durée. Vous pouvez modifier les amplitudes et changer de note.

5.2• Création de deux fonctions : transformations de Fourier discrètes directe et inverse.

Vous allez créer deux fonctions dont les prototypes pourraient être les suivant :
*void DFT(double *signal, double *partie_reelle, double *partie_imaginaire, int N) ;*
*void IDFT(double *signal, double *partie_reelle, double *partie_imaginaire, int N) ;*
Vous pouvez bien sur modifier ces prototypes.
Utilisez ces fonctions pour créer la gamme chromatique de la (la-si-do-re-mi-fa-sol). Faites des essais avec les deux méthodes pour créer un accord de do-majeur (do-mi-sol) et de la mineur (la-do-mi). Vous pouvez aussi essayer de créer des sons nouveaux (attention, il faut que le nombre d'échantillons soit suffisant, 5 sec. minimum).
Conseil : les calculs se font en format *double* tandis que l'enregistrement se fait en format *unsigned char*. Pensez à faire une fonction transformant un signal codé sur 8bits (entre 0 et 255) en un signal codé en réel flottant (sur un intervalle [-1, 1] par exemple) et vice-versa.

5.3• Faites (ou récupérez) des fonction qui permettent d'afficher les fichiers son et regardez les. Testez vos transformations directes et inverses sur un des fichiers son a disposition (regardez si le son est identique à l'original après avoir subit les deux transformations). ATTENTION A LA QUANTIFICATION !

5.4• Utilisez les deux fonctions *DFT* et *IDFT* pour réaliser des filtrages fréquentiels sur vos sons de gamme. Essayez de supprimer le début de la gamme, le milieu de la gamme ou la fin de la gamme et essayez vos filtrages sur le fichier de bris de verre (ou d'autres fichiers son). Faites une fonction de filtrage passe-bas dans laquelle seule la fréquence de coupure est un paramètre (attention la fréquence de coupure numérique est un entier dépendant de la fréquence de coupure, du nombre d'échantillons et de la fréquence d'échantillonnage).

5.5• Equation récursive.

Ecrivez l'équation récursive associée à la fonction de transfert échantillonnée du filtre de Butterworth.

5.6• Algorithme

Ecrivez l'algorithme associé et faites en une fonction filtrage dont le prototype sera :
*int FiltreButterworth(double *Input, double *Output, int N, double alpha) ;*

où N est le nombre de données, Input et Output sont deux tableaux alloués de N valeurs. Vous supposerez toujours que les premières valeurs du signal filtré sont obtenues en recopiant les valeurs du signal entrant.
Vous pouvez simplifier le paramétrage de votre fonction en créant une fonction calculant α en fonction de la fréquence d'échantillonnage et de la fréquence de coupure désirée.

5.7• Utilisez cette fonction dans votre programme principal précédent et testez son aptitude à filtrer les sons de basse et haute fréquence.

5.8• Essayez de réaliser **très simplement** un filtre ne laissant passer que les hautes fréquences et testez-le. Essayez ensuite de retirer une note sélectivement dans la gamme. Constatez la sélectivité de votre filtre. Vous pouvez essayer d'utiliser d'autres filtres (tchebychev, Bessel, Elliptique, ...) dont vous trouverez les expressions continues sur internet. Pouvez-vous atténuer tous les sons ayant une fréquence donnée ?

5.9• Pour ceux d'entre vous qui seraient rapides, vous pouvez essayer de créer une fonction de transposition - il faut bien réfléchir à comment faire - c'est à dire remontant ou baissant globalement un fichier musical dans la gamme.

Pour votre rapport, il vous faut comparer les méthodes de filtrage, la sélectivité que vous obtenez, la déformation des sons par telle ou telle méthode, votre aptitude à modifier l'espace temporel grâce à l'espace fréquentiel (par exemple prenez un fichier son que vous connaissez et modifiez sélectivement quelques raies de l'espace de Fourier - en faisant attention à la symétrie de cet espace). Voyez entre autre l'influence de la taille d'un fichier sur le potentiel de modification.

6• ANNEXE : Le format wav

Source de la notice : http://subo.developpez.com/index.htm

Header d'un fichier wav :

```
// CHUNK TYPE
char file_type[4];       // (4 octets) : Constante "RIFF" i.e id du format
int file_size;          // (4 octets) : file_size est nb d'octets restant à lire
char file_id[4];         // (4 octets) : Identifiant "WAVE"
// CHUNK FORMAT
char chunk_id[4];        // (4 octets) : Identifiant "fmt "
int chunk_size;         // (4 octets) : Nombre d'octets pour définir le chunk
short format;            // (2 octets) : Format de fichier (1: PCM, ...)
short channels_nb;      // (2 octets) : Nombre de canaux (1 = mono, 2 = stéréo)
int sampling_freq;      // (4 octets) : Fréquence d'échantillonnage (en Hertz)
int bytes_per_second;   // (4 octets) : Nombre d'octets par seconde de musique
short bytes_per_sample; // (2 octets) : Nombre d'octets par échantillon
short depth;             // (2 octets) : Nombre de bits par donnée (8 ou 16)
// CHUNK DONNéES
char data_id[4];         // (4 octets) : Constante "data"
int data_size;           // (4 octets) : nombre d'octets restant
```

```
// DATA
unsigned char* data8;    // Tableau de données (cas données 8 bits)
short* data16;           // Tableau de données (cas 16 bits)
long int data_nb;        // Nombre de données
```

L'amplitude : En 8 bits, l'amplitude possède une résolution de 256 valeurs. En 16 bits, 65536 valeurs, etc... Il existe aussi le 24 et 32 bits.
La fréquence : La fréquence d'échantillonnage, correspond au nombre d'échantillons (sample) pour une seconde d'enregistrement. Ainsi 44100 Hz signifie 44100 échantillons pour une seconde de son mémorisé. Plus la fréquence d'échantillonnage est élevée, meilleure est la qualité du signal sonore, plus les données digitales sont proches de l'original.

Le débit : On peut calculer le "débit" (ko/s) d'un signal avec les paramètres: depth (8 ou 16 bits), channels_nb (mono ou stéréo) et la fréquence 'sampling_freq'. Par exemple, en 8 bits mono 44100Hz, cela nous donne pour une seconde d'enregistrement: 44100 octets (1 octet=8 bits). En stéréo, c'est le double. En 16 bits stéréo, c'est le quadruple. Ainsi, avec la qualité du CD audio (16 bits stéréo 44,1 kHz), on obtient 176400 octets par seconde.

Le format PCM : C'est le format de fichier "standard" pour les signaux sonores, car les données sont brutes, c'est-à-dire qu'elles ne sont ni modifiées, ni compressées. Le fichier possède une en-tête de 44 octets, permettant de connaître le type du signal: son format, sa fréquence, le nombre de voies, etc... En 8 bits, la valeur de l'amplitude est non signée, et en 16 bits, l'amplitude est signée.

L'ordre des données : Après les 44 octets de l'en-tête, viennent les données. Les données ont un ordre bien défini. Dans le cas d'un signal 8 bits mono, c'est 1 seul octet par sample, donc les données se suivent normalement. Pour un signal 8 bits stéréo, ce sont 2 octets par sample, l'octet de la voie de gauche, puis l'octet de la voie de droite: L,R, L,R, L,R, etc... Dans le cas d'un signal 16 bits mono, ce sont 2 octets par sample, l'octet de poids faible, puis l'octet de poids fort dans le cas d'une représentation en mémoire « little endian ». Dans le cas d'un signal 16 bits stéréo, ce sont 4 octets par sample; Deux octets pour la voie de gauche, et deux pour la voie de droite : L,L,R,R, L,L,R,R, L,L,R,R, etc...

Le format des données : Le signal 8 bits (mono ou stéréo) possède des données non-signées, c'est-à-dire que le point (l'amplitude) le plus bas vaut zéro, le point du milieu vaut 127, et le point le plus haut vaut 255. Avec un signal 16 bits, les choses sont différentes, les données sont signées : le point le plus bas vaut -32268, le point du milieu vaut zéro, et le point le plus au vaut 32767.

Attention! Toutes ces explications ne sont valables que pour le format PCM, car il existe d'autres types de compression pour les signaux WAV. Cependant, le format PCM est

le plus utilisé pour le traitement des samples.