Notes de TP: Traitement du signal (HMIN109)

Question 1 :

Objectif : Nous voulons générer un signal pour une note précise pendant un temps précis.

Données :

- une note (une fréquence en Hz) : f,
- un temps (en secondes) : t_t,
- un taux d'échantillonnage (= fréquence d'échantillonnage) : f_e

Ressources :

⇒ On connaît la formule d'une sinusoïde en fonction du temps et d'une fréquence :

$$s(t) = A \cdot \sin(\omega t + \varphi)$$
$$\omega = 2\pi \cdot f = \frac{2\pi}{T}$$

$A = Amplitude\,(\text{à } 1.0 \text{ pour un signal de taille maximale})\,(\text{pas d'unité})$
$t_t = Temps\ total\ d'enregistrement$
$i = index\ dans\ tableau\ d'échantillons$
$t = Indice\ de\ temps = i * \frac{1}{f_e}\,(secondes)$
$\omega = La\ pulsation$
$\varphi = Décalage\ de\ phase\,(état\ de\ départ)\,(secondes)$
$T = \frac{1}{f} = La\ période\,(secondes)$

En partant de là, on sait N, le nombre d'échantillons : $N = t_t * f_e$
Ce qui donne la taille du tableau qui représente le signal.
À noter qu'on travaille sur un signal non-discrétisé, c'est-à-dire dans [-1 ; 1].

Ce signal sera à :

- normaliser (s'assurer que toutes les valeurs sont dans la bonne range)
- discrétiser (passer toutes les valeurs dans [0 ; 255])

avant de le passer au créateur de fichier.

Résolution : On peut maintenant créer le signal voulu à l'aide d'une boucle for(...){} sur un tableau de flottants de taille N, qui représente les échantillons.

Question 2 :

Gamme: *Suite de notes jouées à la suite, distinctement les unes des autres.*
Accord: *Ensemble de notes jouées simultanément sur un temps donné.*
3ème octave:

A3 → 220 Hz (la)
B3 → 247 Hz (si)
C3 → 131 Hz (do)
D3 → 147 Hz (ré)
E3 → 165 Hz (mi)
F3 → 176 Hz (fa)
G3 → 196 Hz (sol)

FFT *(Fast Fourier Transform)*: Algorithme calculant à moindre coût le même résultat que notre version non-optimisée de la transformée de Fourier. *(Notre version n'a pour but que de connaître la théorie mais ne sera jamais utilisée car BEAUCOUP trop lente)*

L'utilisation de la transformation de Fourier nécessite des opérations de normalisation et de cast dès qu'on veut accéder à des fichiers. Les réponses suivantes sont centrées sur la question, les opérations de normalisation et cast ne sont qu'évoquées.
Pour voir comment adapter les données pour les écrire dans un fichier, voir "io.cpp"

Objectifs :

On veut :

1 - Écrire la transformée de Fourier discrète directe (et son inverse).
2 - Utiliser ces fonctions pour générer la gamme au 3ème octave.
3 - Créer un accord de do-majeur & la-mineur.

Ressources OBJECTIF 1 :

→ Notre signal réel est la suite de nombres réels : $x_n,\ n \in [0;N[$
= Donnée d'entrée de la transformée de Fourier
= Ce que calcule la transformée inverse

→ Notre signal transformé est la suite de nombres complexes : $X_k,\ k \in [0;N[$
= Donnée d'entrée de la transformée de Fourier inverse
= Ce que calcule la transformée directe

$\mathcal{F}$ = Transformée de Fourier
$\mathcal{F}^{-1}$ = Transformée de Fourier inverse

Un nombre complexe s'écrit : $a+ib$ où a et b sont des réels.
On aura donc pour $\mathcal{F}$ (et l'inverse pour $\mathcal{F}^{-1}$):

- 1 tableau en donnée:

```
double[N] signal;
```

- 2 tableaux à calculer (à implémenter avec des vector<double>):

```
double[N] reelsFourier;
double[N] imaginairesFourier;
```

Le tableau *reelsFourier* contient le coefficient a (partie réelle) pour chaque X_k.
Le tableau *imaginairesFourier* contient la partie imaginaire (b) pour chaque X_k.

Nous connaissons les formules pour calculer chaque parties réelles et imaginaires :

$$a_k = \sum_{n=0}^{N-1} \left(x_n \cdot \cos\left(2\pi \frac{k \cdot n}{N}\right)\right)$$

$$b_k = -1 \cdot \sum_{n=0}^{N-1} \left(x_n \cdot \sin\left(2\pi \frac{k \cdot n}{N}\right)\right)$$

On parcourt le tableau *imaginairesFourier* que dans les N premiers éléments. Cependant, l'algorithme a besoin d'un tableau dont la taille est une puissance de 2. *(C'est surtout vrai pour la FFT qu'on utilisera plus tard au lieu de notre version)* On cherche la première puissance de 2 plus grande que le nombre d'échantillons. Ce n'est qu'après qu'on fait l'allocation en mémoire.

$\Rightarrow$ n *tel que* $N \leqslant M = 2^n$
$M =$ *taille allouée*

Résolution OBJECTIF 1 :

On utilise une simple boucle for(...){} pour remplir les tableaux :

```cpp
#include <cmath>

//imaginairesF est en réalité de taille M, non N, mais ça ne change pas le calcul

void fourierTransform(double* signal, size_t N, double* reelsF, double* imaginairesF){
      for(size_t k=0 ; k<N ; k++){ //boucle dans le tableau des complexes
          double tempR = 0.0;
          double tempI = 0.0;

          for(size_t n=0 ; n<N ; n++){ // boucle dans le signal
              tempR += (signal[n] * cos(2.0 * M_PI * ((double)(k*n)/N)));
              tempI += (signal[n] * sin(2.0 * M_PI * ((double)(k*n)/N)));
          }

          reelsF = tempR;
          imaginairesF = -1.0 * tempI;
      }
}
```

Pour faire la transformée inverse, on applique la formule:

$$signal[n] = \frac{1}{N} \cdot \sum_{k=0}^{N-1} \left(a_k \cdot \cos\left(2\pi \frac{k \cdot n}{N}\right) - b_k \cdot \sin\left(2\pi \frac{k \cdot n}{N}\right)\right)$$

À PARTIR DE MAINTENANT, ON UTILISE LA **FFT**

Ressources OBJECTIF 2 :

On prend *reelsFourier[k]* et *imaginairesFourier[k]*, avec $k \in [0; M[$

Ceux sont les valeurs de la transformée pour la fréquence $f = \frac{k}{M} \cdot f_e$

On en déduit la formule pour trouver k_1.

k_1 = index de la transformée à accéder pour modifier la fréquence f

$k_1 = M \cdot \frac{f}{f_e}$

À cause du repliement du spectre, le signal transformé est symétrique.

L'axe de symétrie est la valeur : $f_{Shannon} = \frac{f_e}{2}$

Cette symétrie nous oblige à trouver k_2, l'index symétrique de k_1.

$\Rightarrow \quad k_2 = M - k_1$

Résolution OBJECTIF 2 : *genereGamme.cpp/genereGamme();*

Le principe est de construire chaque note individuellement.

On ne fait la fusion des tableaux (signaux) qu'à la fin.

On détermine le temps de chaque note et la fréquence d'échantillonnage.

Ici :

- t_t=5s
- f_e=44100 Hz

On peut déjà calculer que $N = t_t * f_e$.

On déclare en mémoire *signalFinal* qui est un vector<double> vide pour la fusion.

M, la taille des segments de la transformée n *tel que* $N \leqslant M = 2^n$
M = *taille allouée*

Pour chaque fréquence à ajouter à la gamme (dans un for):

1. On alloue:

- 1 segment de taille N (*signalTemp*)
- 2 segments de taille M (*reelsFourier* & *imaginairesFourier*)

Ne pas oublier d'initialiser le contenu des segments à 0.0:

```
vector<double> segment = vector<double>(taille, 0.0);
```

2. Calculer les valeurs de k_1 et k_2 en fonction de:

- la note en cours ($= f$)
- la fréquence d'échantillonnage passée en paramètre ($= f_e$)
- M qu'on a calculé au début

3. Dans *reelsFourier* et *imaginairesFourier* changer les valeurs d'index k_1 & k_2

Peu importe la valeur tant qu'elle est plus grande que 0.0

C'est grâce au fait que le signal sera normalisé à la fin.

4. On passe nos tableaux à la transformée de Fourier inverse.
FFT(−1, *M*, *reelsFourier*, *imaginairesFourier*);
Le -1 en paramètre déclenche la transformée inverse
! Les tableaux en paramètres sont écrasés par le résultat !

5. Recopier les N premières valeurs de *reelsFourier* dans *signalTemp*

6. Concaténer *signalFinal* et *signalTemp*

À la fin du for, le contenu de *signalFinal* peut être envoyé au créateur de fichier
Ne pas oublier la normalisation et le cast

Résolution OBJECTIF 3 : *genereGamme.cpp/genereAccord();*
Il faut faire exactement la même chose que dans l'objectif 2.
La différence est qu'au lieu d'allouer des nouveaux segments à chaque itération, on modifie la même paire de segment *reelsFourier* & *imaginairesFourier* et on passe cet unique couple à la transformée inverse qu'une fois.

Le résultat peut être envoyé vers le créateur de fichier.

Question 3 :

Objectif : Pouvoir visualiser le résultat de nos fonctions

Ressource : Audacity

Résolution :
Audacity > Analyse > Tracer le spectre
→ Permet de tracer le spectre de fréquences d'un signal sur une sélection donnée

Question 4 :

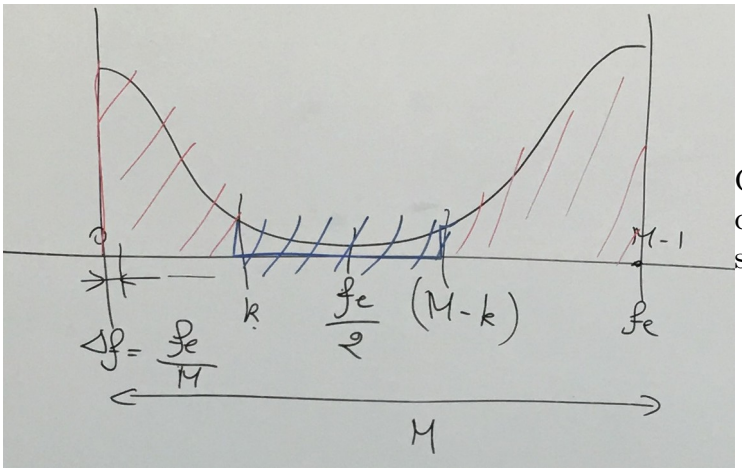
Objectifs :
- Écrire un filtre passe-haut avec la transformée de Fourier
- Écrire un filtre passe-bas avec la transformée de Fourier
- Écrire un filtre passe-bande avec la transformée de Fourier

Ressources :
Aucune ressource en plus n'est nécéssaire à la conception de ces filtres.

Résolution :
On commence par créer les tableaux reelsFourier et imaginairesFourier.
Leur taille doit évidemment être déduite du signal d'entrée, ne pas oublier d'init.
On recopie notre signal d'entrée de taille N dans les N premiers slots de reelsFourier
Les valeurs de imaginairesFourier restent toutes à 0.0

On peut maintenant donner ces 2 tableaux à la FFT(1, ...);

On sait déjà calculer les valeurs de k_1 et k_2, on les calcule donc pour f_c
f_c est la fréquence de coupure voulue.
La fréquence de coupure numérique est juste l'arrondi par troncature vers un int de f_c.



Comme on le voit sur le schéma ci-contre, une fois k_1 et k_2 calculé, ont sait que :

- Pour un filtre passe-bas, il faut faire un for de k_1 à k_2 et passer tous les indexes rencontrés à 0.0 dans reelsFourier ET dans imaginairesFourier.

- Pour un filtre passe-haut, il faut faire un for de 0 à k_1 et un autre de k_2 à M, dans les quels on passera tous les indexes rencontrés à 0.0 dans reelsFourier ET dans imaginairesFourier.

(Le filtre passe-bande est facilement implémentable en suivant la même logique)

Une fois cette opération terminée, on peut faire la transformée inverse, normaliser, quantifier et écrire les N premiers indexes de reelsFourier dans un fichier.

Question 5, 6 & 7 :

Objectifs :

- Écrire l'équation récursive du filtre de Butterworth du 3ème ordre.
- L'implémenter.

Ressources :

Le document *"Notes Filtre de Butterworth.pdf"* joint dans le même répertoire donne le détail pour déduire l'expression récursive à partir de la transformée en Z de la fonction de transformation du filtre donnée dans l'énoncé.

Pour passer à l'implémentation, seule la dernière expression encadrée en rouge est intéressante.

Résolution:

Chacune de ces fonctions est implémentée dans *butterworth.cpp*

Dans la suite:

$e_k = e(k) = e[k]$ = le k^{ième} élément du signal d'entrée.

$s_k = s(k) = s[k]$ = le k^{ième} élément du signal de sortie.

Avant de commencer, il est utile de préciser pour ne pas s'embrouiller que:
- Le filtre de Butterworth est récursif, au sens physique, pas informatique.
- L'implémentation ne comprendra aucune fonction s'appelant elle-même, elle se fera de façon linéaire.

Le terme "récursif" signifie simplement ici que pour calculer s[k],
on va utiliser non seulement e[k], mais aussi des e[k-i] et des s[k-i].
Sachant que le filtre est du 3ème ordre, on peut ajouter que: i <= 3.

De plus, les signaux que nous étudions sont causaux. Cela signifie que:
- Chaque e[k] est indépendant des autres e[...].
- Le signal e[j]=0.0 si j<0.
  ⇒ Cette information est utile car dans les premières itérations du tableau, il serait compliqué de demander la lecture d'un index de tableau négatif.

Implémentation:

1. On commence par écrire un fonction qui calcule alpha: double process_alpha(...)
Son expression est donnée explicitement dans l'énoncé: $\alpha = \pi \cdot \frac{f_c}{f_e}$

2. On a ensuite besoin de créer 4 fonctions:
- double process_A(double alpha),
- double process_B(double alpha),
- double process_C(double alpha),
- double process_D(double alpha)

Une fois de plus, les expressions de A(α), B(α), C(α), D(α) sont dans le sujet.

3. On a besoin de 7 nouvelles fonctions:
- double process_a(double alpha),
- double process_b(double alpha),
- double process_c(double alpha),
- double process_d(double alpha),
- double process_e(double alpha),
- double process_f(double alpha),
- double process_g(double alpha),

Ces fonctions sont à implémenter selon l'expression qui en est donnée dans le document "*Notes Filtre de Butterworth.pdf*" dans le cadre rouge.
Elles utilisent les 4 fonctions implémentées précédemment.

4. On écrit une fonction: double get(vector<double>& signal, long int index)
Cette fonction renvoie 0.0 si *index* est strictement inférieur à 0.
Elle renvoie signal[index] sinon.

5. On peut maintenant écrire notre fonction butterworth principale:
void butterworth(int fc, int fe, vector<double>& e, string outputName)

Dans cette fonction, on commencera par créer un vector de double s.
Ce vector doit faire la même taille que e et être init avec des 0.0
On calcule alpha et tous les coefs de a à g.
À partir de là, le signal final se calcule par une simple boucle:

```
for(long int k = 0 ; k < (long int)e.size() ; k++){
    s[k] = (coeff_a * get(e, k-3)) +
           (coeff_b * get(e, k-2)) +
           (coeff_c * get(e, k-1)) +
           (coeff_d * get(e, k)) +
           (coeff_e * get(s, k-1)) +
           (coeff_f * get(s, k-2)) +
           (coeff_g * get(s, k-3));
}
```

Le signal s peut être envoyé vers le créateur de fichier.

Question 8 :

Filtre passe bande inversé, implémentable avec Fourier

Question 9 :

Facultative.

Le compte rendu

COMPTE RENDU de TP-projet: À rendre le **06/01/2020** à **23h59** dernier délai

Il doit faire entre **3 et 5 pages** sans compter les figures (captures Audacity, ...).

IL PEUT ÊTRE RENDU SOIT:

- En ligne: doit être rendu en **PDF** et **aucun autre format**

- En papier: au **secrétariat du bâtiment info** dans une enveloppe. Il faut demander à ce que l'enveloppe soit **cachetée de la date de remise**.

QUE METTRE DANS LE RAPPORT ?

Il doit comprendre les **observations** et **conclusions** qu'on a faites pendant ces TP-projets.

Ex: Les différences observées entre un signal filtré ou non.

Si on a utilisé des **algos trouvés** sur internet (*ex:* FFT), il faut les **mentionner** et dire ce qu'il nous ont **apporté**. *(leur défauts, qualités, pourquoi, ...)*

Il y a la possibilité de **joindre des fichiers sons** pour appuyer son explication.

Il est impératif de bien citer ses sources et de les mettre en italique dans le rapport.

QUE NE PAS METTRE DANS LE RAPPORT ?

Il **ne** faut **pas** mettre la **façon dont on a implémenté** nos filtres, ni de code.

Il **ne** faut **pas** mettre les **réponses aux questions** qui sont **posées** dans le TP pour nous faire implémenter les filtres

Il **ne** faut **pas** faire une **recopie de la partie explicative** du sujet de TP