Travaux pratiques de filtrage numérique.

Bourget-Vecchio Emery | Traitement de signal | 08/01/2021

UNIVERSITÉ
DE MONTPELLIER

Introduction

Ce compte rendu est une synthèse de ce que j'ai pu réaliser pendant les séances de TP de traitement de signal.
Le sujet du TP se trouve ici :
https://www.lirmm.fr/~strauss/MasterInfo/TravauxPratiquesTS.pdf
Les documents annexes sont eux ici :
https://www.lirmm.fr/~strauss/MasterInfo/TravauxPratiquesHMIN109.html
Nous avons comme base un programme en C++ et qui nous permet de générer des fichiers en .wav.
Tous les fichiers sonores présentés ici seront dans un dossier annexe.
Les extraits de code présent dans ce résumé représentent des fonctions qui me semblent importantes/inédites.

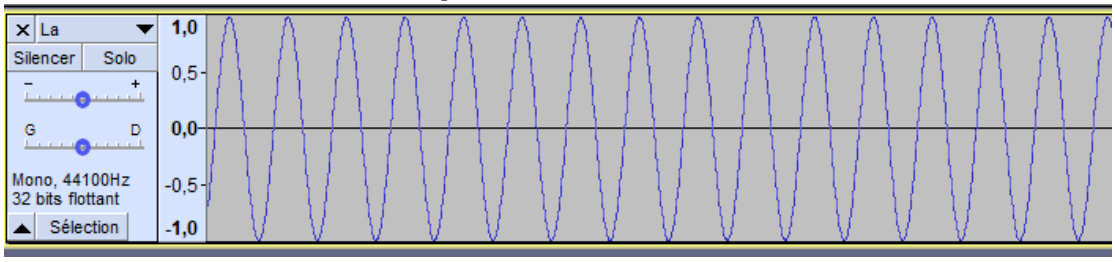
Table des matières

I. Création de notes

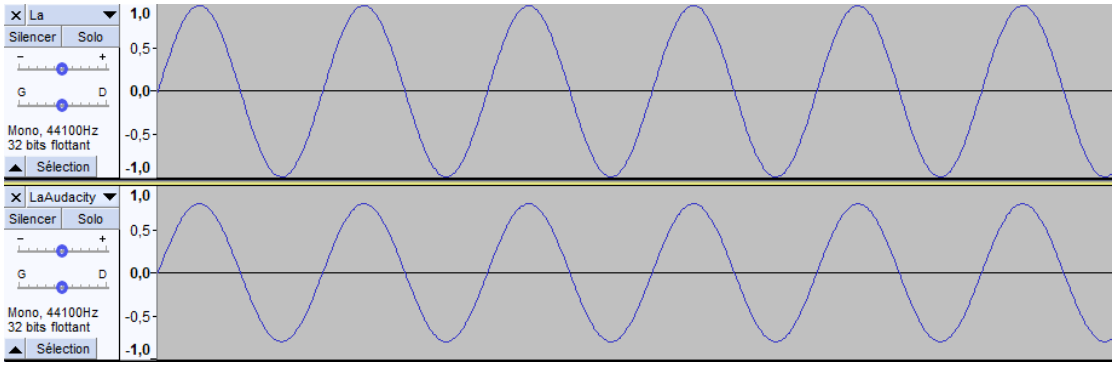
Pour répondre à la première question du TP, il m'a fallu créer la note LA qui a une fréquence de 440 Hz dans le format .wav en mono pendant 6 secondes.
Pour cela, j'ai dans un premier temps créé un tableau de unsigned char avec comme taille prédéfinie la multiplication de la fréquence d'échantillonnage avec la durée voulue.
Pour éviter un sous-échantillonnage et nous placer dans les conditions de Shannon, nous devons échantillonner à au moins le double de la fréquence de notre signal étudié.
L'oreille humaine ne peut pas en générale entendre les fréquences supérieures à 20000 Hz, donc si l'on veut respecter les conditions de Shannon, nous devons prendre une fréquence d'au moins 40000 Hz. C'est pour cela que pour la plupart des logiciels traitant des fréquences utilisent 44100 Hz comme fréquence d'échantillonnage.
C'est cette fréquence multipliée par le temps (donc 6) que nous allions utiliser pour initialiser notre tableau de unsigned char.
Ensuite pour chaque élément de ce tableau, nous allons convertir en période pour avoir une forme sinusoïdale en appliquant la formule ci-dessous.

$$data8[i] = \left(\sin\left(i/rapport_{note/échantillonnage}/2\pi\right) + 1\right) \cdot 127.5$$

Avec :
data8[i]: tableau de unsigned char a l'indice i.
$rapport_{note/échantillonnage}$: rapport entre la note étudiée et la fréquence d'échantillonnage
Le sinus à un domaine de définition compris entre -1 et 1, pour le convertir en type unsigned Char, il nous changer le domaine de définition pour qu'il soit compris entre 0 et 255. Pour cela, il suffit de rajouter 1 à la valeur du sinus et de multiplier le tout par 127.5. Nous pouvons retrouver cela à la fin de notre formule ci-dessus.
Pour faire la conversion inverse, il suffit de diviser notre nombre par 127.5 et de lui retirer -1.
Une fois que chaque élément du tableau a été traité, il nous suffit d'appeler le constructeur de la classe Wave et d'ensuite appeler la fonction write pour générer notre fichier wav.
Cela nous donne une note de cet aspect :



Pour vérifier ma note, j'ai généré à l'aide de Audacity la note La pour comparer les deux courbes et les deux tonalités.

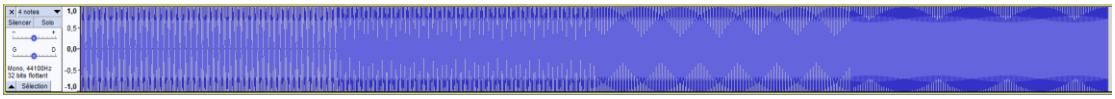


Ici nous pouvons voir que les deux courbes coïncident énormément ce qui montre que note La est correct.

J'ai par la suite amélioré les fonctions pour prendre en compte plusieurs notes d'affilée. Par exemple si j'ai un son qui contient 4 notes, il m'a fallu diviser par 4 la taille de data8 afin d'appliquer de façon égale chacune des notes. Cela se représente par la formule ci-dessous :

$$data8[i + \frac{n \cdot data8Size}{nbrNote}] = \left(\sin\left(i/rapport_{note/échantillonnage}/2\pi\right) + 1\right) \cdot 127.5$$

Voici ci-dessous un exemple avec les notes DO(261.63) ; MI(329.63) ; FA(349.23) ; LA (440)



Pour rendre l'utilisation des tableaux plus simple, j'ai préféré par la suite stocker mon signal dans un tableau de double vector<double> .

II. Transformée de Fourrier

A) DFT ET IDFT

La transformé de fourrier permet de passer d'un signal en fonction du temps à un signal en fonction de la fréquence. Cela nous permettra par la suite de pouvoir appliquer des filtres sur notre signal.
J'ai codé deux fonctions dans mon programme, la première se nommant DFT qui permet d'appliquer la transformer de fourrier et la deuxième étant IDFT qui permet de faire son inverse.
Les deux méthodes ont pour but d'appliquer les formules ci-dessous (cf. sujet de TP pour plus de détails) :

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

Figure 1 : Méthode appliquée dans la fonction DFT

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

Figure 2 : Méthode appliquée pour calculer la partie réelle de l'IDFT

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

Figure 3 : Méthode appliquée pour calculer la partie imaginaire de L'IDFT

Pour ces deux méthodes, j'ai dû préalablement créer deux tableaux vides qui permettra de stocker la partie réelle et imaginaire de notre transformé de fourrier.
Une fois la transformé de fourrier terminé, il m'a fallu créer une fonction pour visualiser mes résultats en normalisant entre [-1, 1] mes résultats. J'ai alors créé la fonction **DFT visualize**.
Cette fonction a pour premier objectif de calculer la norme de notre signal à l'aide de cette formule :

$$norme[i] = \sqrt{signal[i]^2_{real} + signal[i]^2_{imaginary}}$$

Figure 1: Calcule de norme au rang i

En appliquant cette formule à chaque élément du signal, on obtient une nouvelle liste de double de même taille que notre partie réelle. Ainsi chacun des éléments de la liste norme contient la norme de notre signal d'origine.
Nous allons également récupérer la norme maximale de notre signal et la stocker dans une valeur double se nommant « Max ».

$$norme\,[i] \;= \left(2.0 \cdot \frac{norme[i]}{\max}\right) - 1.0$$

Figure 2: Normalisation de la liste norme pour qu'elle soit comprise entre -1 et 1

Nous pouvons ainsi dès à présent normaliser noter liste norme pour qu'elle soit contenue entre -1 et 1 à l'aide de la formule suivante :
Par la suite, il nous faut convertir notre liste de < double > en liste de unsigned char afin d'être traité par la classe Wave.cpp du programme.
Pour chaque élément de notre liste, nous allons ainsi appliquer la conversion que nous avons vue dans la première partie.
Enfin, pour terminer, nous pouvons appeler la fonction write de wave afin d'écrire notre fichier sonore et d'avoir une représentation fréquentielle de notre signal.

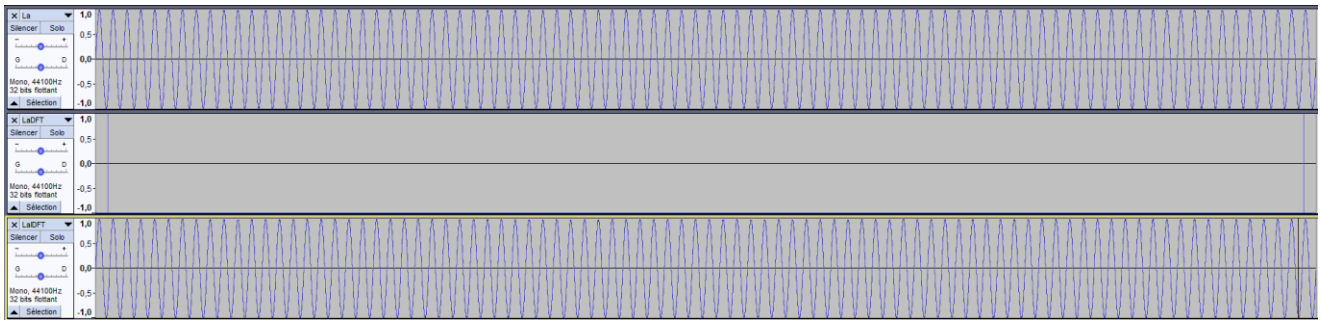


Figure 3: Note avec l'application de la DFT et de L'IDFT

La première courbe représente la note LA en fonction du temps et en dessous, cela représente également la note LA, mais en fonction de la fréquence.
Nous pouvons distinguer 2 pics qui se trouvent au début et à la fin de notre **LaDFT.**
Les deux représentent notre note LA, cependant il y en a un deuxième, car notre courbe se répète.
Pour vérifier que notre DFT est juste, j'ai appliqué l'IDFT et vérifier qu'on récupère bel et bien la note LA, ce qui est bien le cas comme on peut le voir sur la 3e courbe.

B) FFT

Nous avons vu précédemment une première approche de la transformée de Fourrier, les fonctions de transformation ont été codées par moi-même en utilisant des algorithmes relativement simples avec des algorithmes relativement lourds. J'ai ainsi dû faire mes traitements de signal sur des fichiers de moins de 0.3 seconde sinon quoi le programme prenait beaucoup trop de temps.
En annexe, nous pouvions trouver ce lien : http://paulbourke.net/miscellaneous/dft/
Ce dernier nous ramène vers une fonction FFT qui est une version optimisée de notre méthode DFT. Cela va ainsi nous permettre de pouvoir de traiter des fichiers beaucoup plus volumineux dans un temps réduit.
Cependant pour appliquer cet algorithme, il faut que la taille de notre signal soit comprise entre $2^{m-1} < n$ (taille du signal) $<= 2^m$.
Pour trouver la taille adéquate, j'ai créé 2 fonctions, une qui va prendre la taille du signal en paramètre et une qui va incrémenter n jusqu'à 2^n soit supérieur à la taille du signal. Je pourrai ainsi récupérer cette valeur et l'utiliser pour définir les tailles initiales de ma variable partie_imaginaire et partie_réelle. Une fois cela effectuer, il me suffit d'appliquer les mêmes étapes que pour la DFT.
On peut retrouver les résultats ci-dessous :

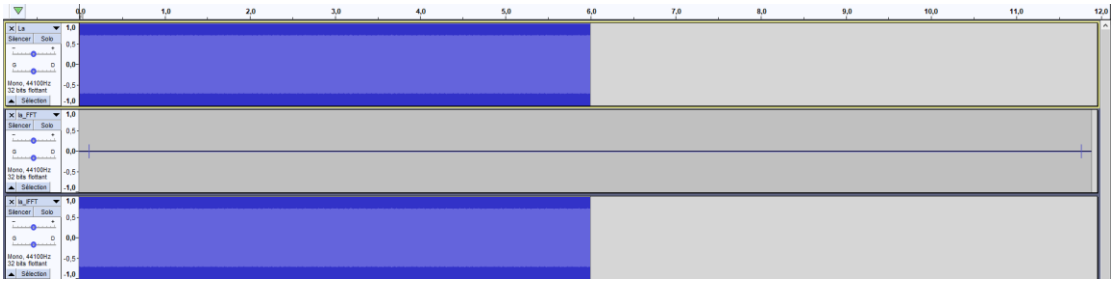


Figure 4: FFT et IFFT sur la note LA

Avec cette capture d'écran, nous pouvons remarquer deux choses :

- la première est la longueur du fichier qui est 6 secondes pour la piste 1 et 3. Cela aurait demandé énormément de temps avec la méthode classique alors qu'avec la méthode FFT, l'opération a quasiment été instantanée.

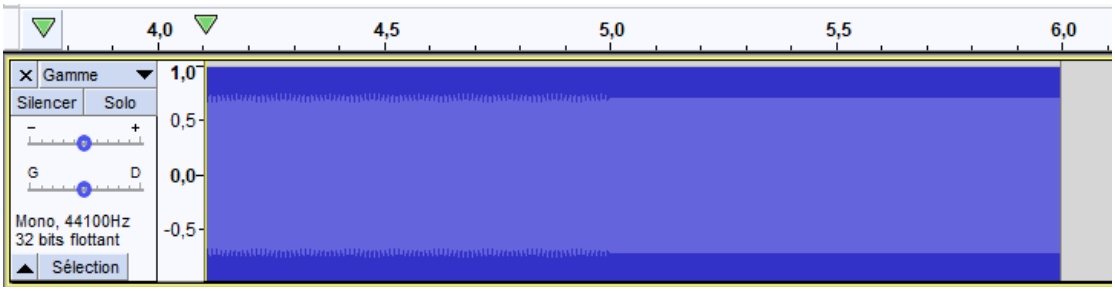
- La deuxième est que pour la piste 2, le temps est de 12 secondes, montrant ainsi le bon fonctionnement de notre algorithme vu ci-dessus.

III. Filtres simples

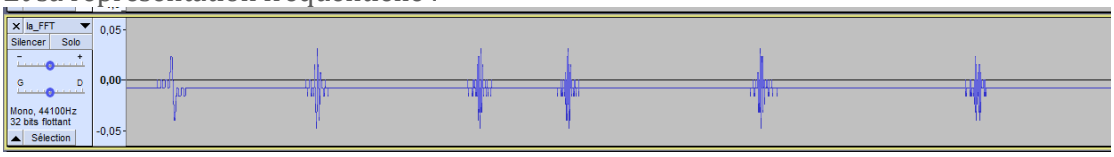
La transformé de fourrier nous a permis d'avoir un signal défini sur le spectre des fréquences. Cette représentation va nous permettre de pouvoir appliquer efficacement des filtres pour filtre différent type des signaux.

Pour réaliser mes tests, j'ai créé une gamme de 6 notes composée des notes DO (261.63) RE (293.66), MI (329.63), FA 349.23), SOL (392.00) et LA (440).

Voici sa représentation temporelle :



Et sa représentation fréquentielle :



A) Filtre passe-haut

L'objectif du filtre passe-haut et de laisser passer les hautes fréquences et de supprimer les fréquences sous un certain seuil fc.

Ici nous allons dans un premier temps de supprimer la note DO, nous allons donc filtrer toute fréquence qui serait en dessous de la note RE soit 293.66.
J'ai donc créé une fonction qui prend en paramètre la partie réelle et la partie imaginaire de note signal, la taille du signal et le seuil fc (ici 293.66).
J'ai dû par la suite trouver le bon facteur multiplicatif pour fc.
Pour cela j'ai multiplié la fréquence par le rapport entre la taille du signal et la fréquence d'échantillonnage.
Dans notre cas, le signal mesure 524288, ce qui divisé par 44100 nous donne environ 11.88. Ce chiffre multiplié par fc nous donne 3491.211.

Notre algorithme va parcourir le signal et supprimer (c'est-à-dire mettre à 0 la valeur) les premiers 3492 termes, mais également les 3492 derniers termes (à cause de la symétrie) de la partie réelle et de la partie imaginaire pour supprimer la note DO.

Cet algorithme m'a permis d'avoir la représentation fréquentielle ci-dessous :

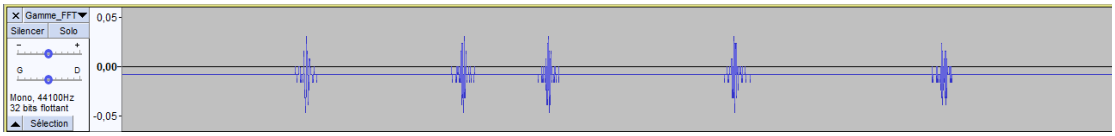


Figure 5: Représentation fréquentielle d'une gamme filtrée pour enlever le Do

Si l'on compare cette représentation avec celle de base présente plus haut, nous pouvons remarquer que la première fréquence a disparu, cette dernière représentait la note DO.

Cela nous a permis d'avoir la représentation temporelle suivante :

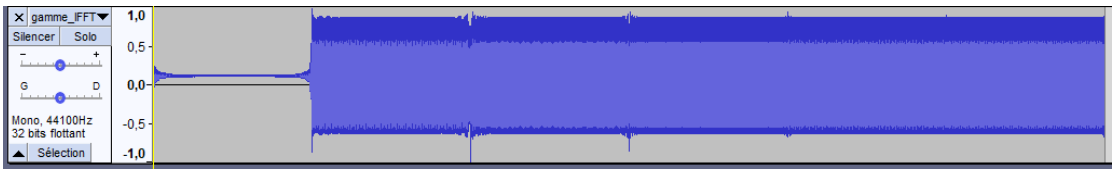
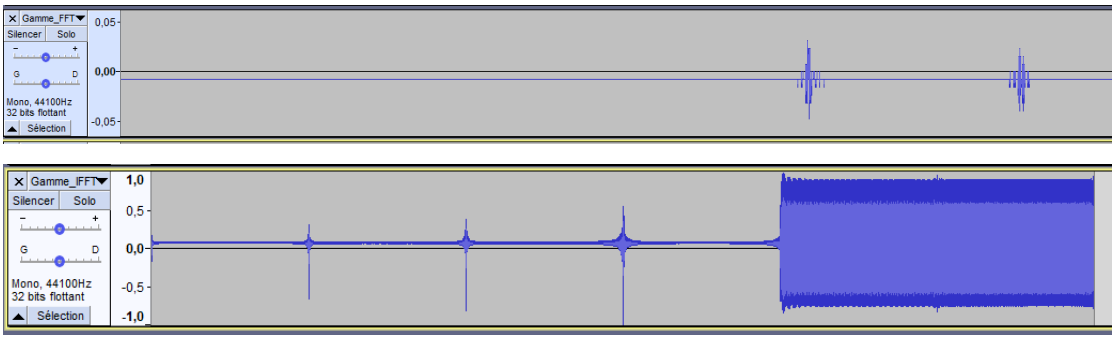


Figure 6 : Représentation temporelle d'une gamme filtrée pour enlever le Do

À l'écoute de ce fichier, la note Do à complètement disparu, cependant on remarque également qu'on a perdu en qualité sur les notes restantes, cela vient surement de mon algo qui n'est pas parfait.

Voici un second exemple avec un fc à la note du Sol, ce qui ainsi filtrer toutes les notes en dessous de Fa (Fa compris).



B) Filtre passe-bas

Le filtre passe-bas est l'inverse du filtre passe-haut, et fonctionne quasiment avec le même principe, nous allons avoir une fréquence de seuil fc mais cette fois-ci, nous allons retirer toutes les fréquences qui sont au-dessus de ce dernier.
Nous allons ici essayer d'enlever la note LA, notre fc sera donc sa fréquence soit 440 Hz.
Le facteur multiplicateur se calcule de la même façon que dans le filtre passe-haut, ce qui

Figure 7 : Représentation fréquentielle et temporelle d'une gamme filtrée pour enlever les notes en dessous de Fa

nous donne440* 11.88 = 5227.2.
Notre algorithme va ici parcourir les signaux réel et imaginaire et supprimer les termes qui sont compris entre les valeurs 5228 et 524288(taille du signal) – 5228 termes.

Cela nous donne ainsi :

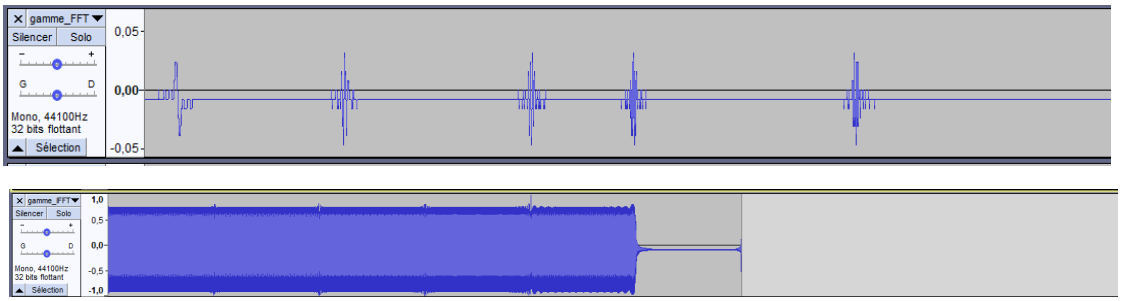


Figure 8: Représentation fréquentielle et temporelle d'une gamme filtrée pour enlever les notes au-dessus de la note LA

On peut facilement constater qu'en comparant avec la gamme sans filtrage, la fréquence du La a ici été enlever, ce qui nous permit d'avoir une gamme sans la note LA.

Note : Je ne sais pas pourquoi le son est ici beaucoup moins endommagé que quand je fais un filtrage passe-bas.

C) Filtre passe-bande

L'objectif du filtre passe-bande va être de conserver uniquement un intervalle de fréquence, il va ainsi nous falloir 2 seuils, un que l'on va appeler fc1 qui va supprimer toutes les fréquences qui lui serait inférieures, et un autre que l'on va appeler fc2 qui va supprimer toutes les fréquences qui lui serait supérieures.
Pour programmer ce filtre, j'ai simplement succinctement appliqué un filtre passe-bas et un filtre passe haute.
Par exemple, j'ai pris comme fc1 la note RE(ce qui va donc supprimer la note DO) et pour f2 la note LA(ce qui va supprimer la note LA).

Voici mes résultats :

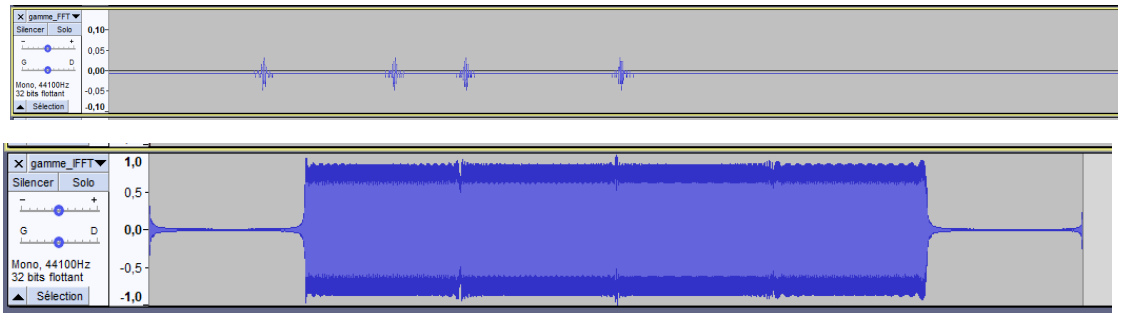


Figure 9: Représentation fréquentielle et temporelle d'une gamme filtrée en coupe-bande pour ne garder que les notes entre le Re et Sol

On remarque encore une fois ici que la première et la dernière fréquence ont été supprimées, ce qui donne à l'écoute les notes RE ;MI,FA,SOL.

D) Filtre coupe-bande

Le filtre coupe est quant à lui l'exact inverse du filtre passe bande.
Il coupe un intervalle de fréquence.
Pour programmer cela, j'ai pris deux fréquence seuil, fc_1 et fc_2 et mon programme enlever toutes les fréquences comprises entre fc1 et fc2 et toutes les fréquences comprises entre N(la taille de mon signal)-fc2 et N-fc2.
Pour le tester, j'ai enlevé la note RE de notre gamme.

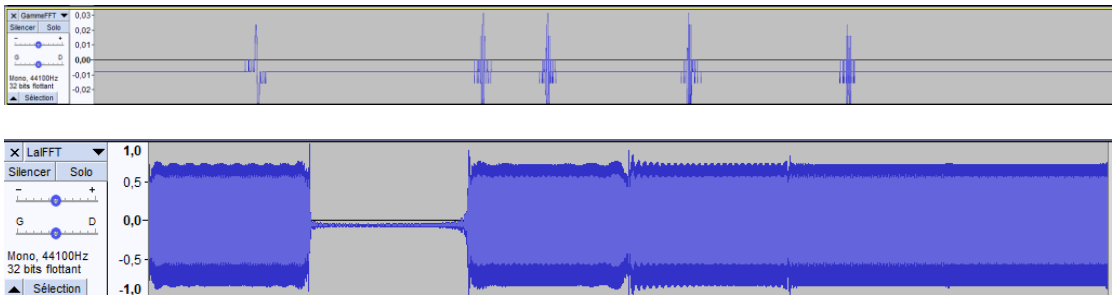
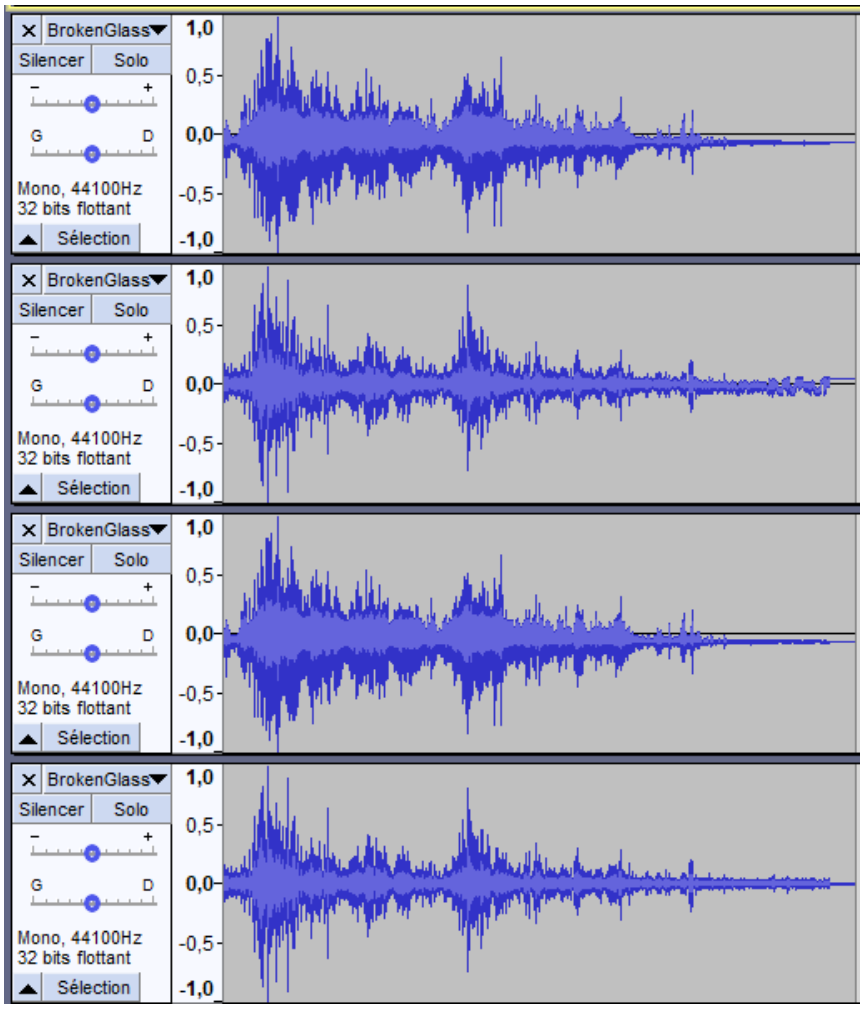


Figure 10 : Représentation fréquentielle et temporelle d'une gamme filtrée en passe -bande pour enlever la note Ré.

On constate ici facilement que notre programme a bel et bien effacé la note Ré.

E) Broken glass

J'ai sur ce son appliqué les 3 filtrages possibles.
Voici ci-dessous, les résultats dès que j'ai obtenus :



Son original

Son avec filtre passe-bas à 2500Hz

Son avec filtre passe-haut à 7500Hz

Son avec filtre passe-bande de 2500 à 7500Hz

J'ai choisi les fréquences arbitrairement pour obtenir un résultat qui met en évidence les filtres sans trop estomper le signal original.
Le son est très strident est tend vers les aigus, ainsi notre filtre passe-bas est grandement efficace, on peut facilement voir la différence entre le signal original est celui mettant en place le filtre.
Pour le filtre passe-haut, il est plus difficile de remarquer des différences, cependant à l'oreille cela est plus simple.

IV. Filtre de Butterworth

I. Équation récursive

Le filtre de Butterworth est un filtre linéaire, conçu pour posséder un gain aussi constant que possible dans sa bande possible.
Il s'applique sur un signal sous forme temporelle.
Les données du TP nous permettent d'avoir le filtre discret du filtre de Butterworth à l'ordre 3.

$$\frac{\alpha^3\left(z^{-3}+3z^{-2}+3z^{-1}+1\right)}{A(\alpha)+z^{-1}\cdot B(\alpha)+z^{-2}\cdot C(\alpha)+z^{-3}\cdot D(\alpha)} = \frac{Y(z)}{X(z)}$$

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 $D(\alpha) = (-1+2\alpha-2\alpha^2+\alpha^3)$.

La relation entre X(z) la transformée en Z de l'entrée de ce filtre et de Y(z) la sortie de ce filtre est :

$$F(z) = \frac{Y(z)}{X(z)}$$

Nous allons ainsi grâce à cette relation trouver la relation récurrente existante entre les échantillons (y_k) de la sortie du filtre et les échantillons (x_k) de l'entrée du filtre.

$$\alpha^3\left(X(z)\cdot z^{-3}+X(z)\cdot 3z^{-2}+X(z)\cdot 3z^{-1}+X(z)\right) = Y(z)\cdot A(\alpha)+z^{-1}\cdot B(\alpha)+Y(z)\cdot z^{-2}\cdot C(\alpha)+Y(z)\cdot z^{-3}\cdot D(\alpha)$$

$$\alpha^3\left(Z\{x_{n-3}\}+Z\{3x_{n-2}\}+Z\{3x_{n-1}\}+Z\{x_n\}\right) = Z\{y_n\}\cdot A(\alpha)+Z\{y_{n-1}\}\cdot B(\alpha)+Z\{y_{n-2}\}\cdot C(\alpha)+Z\{y_{n-3}\}\cdot D(\alpha)$$

$$\alpha^3\left(Z\{x_{n-3}+3x_{n-2}+3x_{n-1}+x_n\}\right) = Z\{y_n\cdot A(\alpha)+y_{n-1}\cdot B(\alpha)+y_{n-2}\cdot C(\alpha)+y_{n-3}\cdot D(\alpha)\}$$

$$\alpha^3\left(x_{n-3}+3x_{n-2}+3x_{n-1}+x_n\right) = y_n\cdot A(\alpha)+y_{n-1}\cdot B(\alpha)+y_{n-2}\cdot C(\alpha)+y_{n-3}\cdot D(\alpha)$$

$$\frac{\alpha^3(x_{n-3}+3x_{n-2}+3x_{n-1}+x_n)-y_{n-1}\cdot B(\alpha)-y_{n-2}\cdot C(\alpha)-y_{n-3}\cdot D(\alpha)}{A(\alpha)} = y_n$$

II. Programmation du filtre

Cette formule va nous permettre d'appliquer un filtre sur le domaine temporel. L'implémentation a été faite comme suit :

```
void filtre_Butterworth(double *Input, double *Output, int N, double fc){
    double alpha =  M_PI*(fc / 44100);
    double A= 1 + alpha*(2  + 2*alpha + alpha*alpha);
    double B= -3 + alpha*(-2  + 2*alpha + 3*alpha*alpha);
    double C= 3 + alpha*(-2  -2*alpha + 3*alpha*alpha);
    double D= -1 + alpha*(2 -2*alpha + alpha*alpha);

    for(int i = 0; i < N; i++){
        if(i<=2){
            Output[i] = Input[i];
        }
        else {
            Output[i] = (alpha * alpha * alpha * (Input[i - 3] + 3 * Input[i - 2] + 3 * Input[i - 1] + Input[i]) -
                        Output[i - 1] * B - Output[i - 2] * C - Output[i - 3] * D) / A;
        }
    }

}
```

III. Test du filtre

Si l'on reprendre notre gamme (DO à LA) que nous avons utilisé pour les filtrages, en passant sa version TEMPORELLE à la fonction, c'est-à-dire sans avoir appliquer la transformée de Fourrier, et en passant un filtre fc (qui représente ici notre seuil) à des fréquences de la gamme, on obtient le résultat suivant :

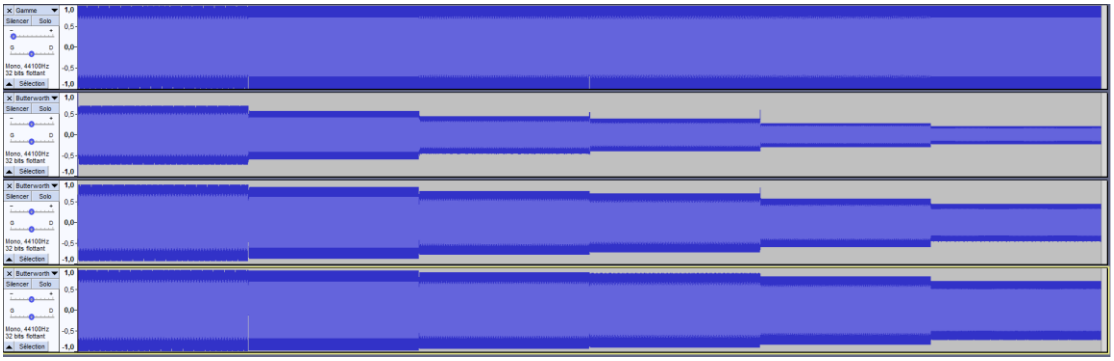


Figure 11 : Représentation temporelle d'une gamme (1er = sans filtre ; 2eme= filtre de BTW en Do : 3eme = filtre de BTW en FA ; 4eme= Filtre de BTW en LA)

On peut clairement voire ici l'impact du filtre sur notre signal, toutes les fréquences supérieures à notre fréquence de seuil seront atténuées jusqu'à complément disparaitre. Si l'on compare ce filtre avec le filtre passe-bas, on remarque que leurs effets sont à peu près les mêmes, mis à part qu'il y en a qui atténue complément les signaux qui dépassent un certain seuil et l'autre qui les atténuent progressivement.

IV. Filtre passe-haut temporelle

Dans cette dernière partie, nous allons réétudier les filtres passe-haut, mais cette fois-ci avec une approche temporelle.

Si l'on va sur ce lien : https://fr.wikipedia.org/wiki/Filtre_passe-haut, on obtient la fonction de transfert du filtre passe-haut.

$$H(j\omega) = \frac{v_o}{v_i} = \frac{Kj\frac{\omega}{\omega_c}}{1 + j\frac{\omega}{\omega_c}}$$

où

$$\omega = 2\pi f$$
$$\omega_c = 2\pi f_c$$

En remplacement

$$\frac{p}{\omega_c} = \left(\frac{1}{\alpha}\right)\frac{z-1}{z+1}$$

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

On obtient :

$$H(p) = \frac{K\left(\frac{1}{\alpha}\right)\frac{z-1}{z+1}}{1+\left(\frac{1}{\alpha}\right)\frac{z-1}{z+1}}$$

En développant le calcul, on a :

$$H(z) = \frac{K\alpha^2\left((z-1)^2(z+1)+(z-1)(z+1)^2\right)}{\alpha^2(z+1)^2(z-1)}$$

Pour obtenir une forme récursive, il faut passer H (z) en factorisant z^{-1}

$$H(z) = \frac{K\alpha^2\left((1-z^{-1})^2(1+z^{-1})+(1-z^{-1})(1+z^{-1})^2\right)}{\alpha^2(1+z^{-1})^2(1-z^{-1})}$$

En redéveloppant on obtient :

$$H(z) = \frac{K\alpha^2\left((1-z^{-2})+(-z^{-3}-z^{-2}+z^{-1}+1)\right)}{\alpha^2(-z^{-3}-z^{-2}+z^{-1}+1)}$$

En considérant :

$$H(z) = \frac{Y(z)}{X(z)}$$

Ainsi :

$$K\alpha^2 \left(X(z)\left(1 - z^{-2}\right) + X(z)\left(-z^{-3} - z^{-2} + z^{-1} + 1\right)\right) = \alpha^2 Y(z)\left(-z^{-3} - z^{-2} + z^{-1} + 1\right)$$

$$K\alpha^2 \left(-Z\{x_{n-3}\} - 2Z\{x_{n-2}\} + Z\{x_{n-1}\} + 2Z\{x_n\}\right) = \alpha^2 \left(-Z\{y_{n-3}\} - Z\{y_{n-2}\} + Z\{y_{n-1}\} + Z\{y_n\}\right)$$

$$K\alpha^2 \left(Z\{-x_{n-3} - 2x_{n-2} + x_{n-1} + 2x_n\}\right) = \alpha^2 \left(Z\{-y_{n-3} - y_{n-2} + y_{n-1} + y_n\}\right)$$

$$K\alpha^2 \left(-x_{n-3} - 2x_{n-2} + x_{n-1} + 2x_n\right) = \alpha^2 \left(-y_{n-3} - y_{n-2} + y_{n-1} + y_n\right)$$

$$y_n = \frac{K\alpha^2(-x_{n-3} - 2x_{n-2} + x_{n-1} + 2x_n)}{\alpha^2} + y_{n-3} + y_{n-2} - y_{n-1}$$

Avec :

K est le gain du filtre, de formule :

$$G_{dB}(\omega) = 20 \cdot \log|H(j\omega)| = 20 \cdot \log\left(\frac{\omega}{\omega_c}\right) - 10 \cdot \log\left(1 + \left(\frac{\omega}{\omega_c}\right)^2\right)$$

Et $\alpha = \pi \frac{\omega_c}{\omega_e}$

J'ai essayé d'implémenter cela dans mon code, mais malheureusement, cela n'a pas fonctionné et me donne des résultats inexploitables.

Conclusion

La réalisation de ce TP a été un très enrichissant. Il a permis de mettre en pratique les notions vues en cours et d'approfondir mes connaissances. Même si le TP n'est pas entièrement terminé, je suis globalement très satisfait du travail que j'ai pu réaliser. Faire ce rapport m'a permis de familiariser avec des concepts tels que la transformée en Z, de Fourier, les notions de filtrages et domaine temporel et de fréquentiel. Ces notions me seront d'une aide précieuse lors de mes futurs de traitement d'image.