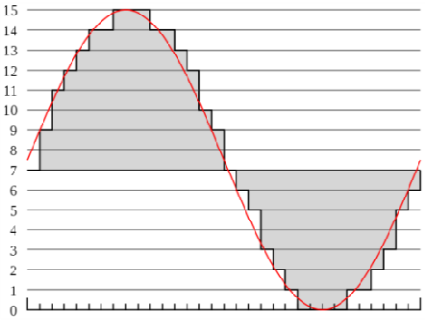
Université Nice Sophia Antipolis

POLYTECH

Département Sciences Informatiques :
3° année

Mini Projet de :
Son Signal et Image pour l'Informaticien

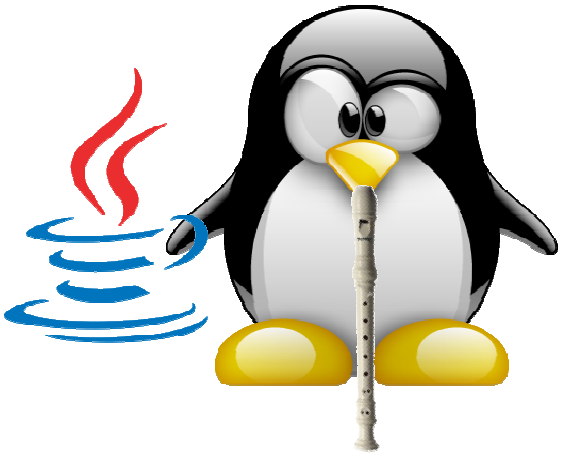


Réalisé par AUZIAS Maël et GENEVIER Sylvestre

De Décembre 2010 à Janvier 2011.

Durée : 3 semaines

Année scolaire 2010/2011



Introduction :

Présentation :

Ce compte rendu traite du MP[1]de SSII[2] réalisé dans le cadre de notre **enseignement** de **Sciences Informatiques**. Pour ce MP nous avons choisi de sujet numéro 2 dont le but est de **synthétiser** une note d'un instrument de musique à partir de nos **connaissances** en **son**, en **signal**, et en **programmation**.

Lors des TD réalisés en SSII nous avons **déjà réussi** à faire cela dans le langage du programme Matlab®. Nous savons donc quelles sont les méthodes à appliquer et la démarche à suivre.

Structure du compte rendu :

- Présentation du sujet choisi :
  Explications en détail du **sujet** et **brève présentation** du travail **réalisé**.
- Identification des modules et explication de la partie théorique :
  Explications de la méthode de travail pour **l'identification** des modules **à réaliser** et leur **importance**.
- Répartition des tâches :
  Explications de la méthode pour se **répartir les tâches** (programmation et rédaction) et un **tableau** pour la visualiser.
- Explication des modules :
  Explications en détail de **chaque méthode** et de ses **algorithmes**.
- Vérification des modules et de la somme des modules :
  Explications des **méthodes** utilisées **pour vérifier le bon fonctionnement** des modules **individuellement** et la **somme** de ceux-ci.
- Mise en œuvre du projet :
  Explications pour **utiliser**, **modifier**, **visualiser**, **tester** les classes.

Technologies choisies :

Pour ce projet nous avons utilisé :

- **Java** de l'entreprise Oracle® comme langage de programmation. Nous sommes en première année du cycle ingénieur et ce langage est notre **seul langage** de programmation **commun**. Cependant en prenant en compte que le langage Java est un langage orienté objet, nous sommes conscients que ce langage **n'est pas du tout optimisé** pour ce genre de projet.
- **Linux** est l'environnement dans lequel nous avons programmé en utilisant des **lignes de commandes** dans la console. Nous **n'avons pas** utilisé d'autre programme (ou EDI[3]) tel qu'Eclipse.
- **Shell** comme langage pour les scripts dans l'optique **d'automatiser** certaines commandes et les réaliser **plus rapidement**.

A noter qu'une **collaboration** avec le **groupe** de Jérémie Montiel et Mohammed BELGHITI ALAOUI a eu lieu lors d'un weekend dédié à la réalisation de ce projet. Ceci est dû aux **difficultés rencontrées** pour **l'amorce** du projet.

[1] MP : Mini Projet.
[2] SSII : Son, Signal et Image pour l'Informaticien.
[3] EDI : Environnement de Développement Intégré.

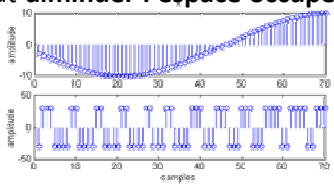
Sommaire :

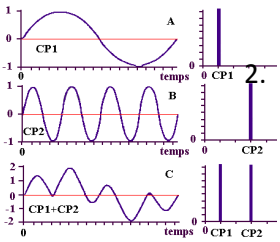
Présentation du sujet choisi :

Dans cette partie nous allons aborder les **sujets proposés** pour ce MP, donner les **spécifications** pour le **sujet** que nous avons **choisi** et enfin faire une **brève présentation** du **travail** que nous avons **réalisé**.

Sujets proposés :

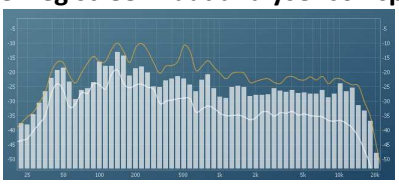
1. **Compression** d'un signal audio par **sous-échantillonnage** :
   En utilisant la fréquence d'échantillonnage il faut **diminuer l'espace occupé** par le fichier en dégradant **raisonnablement** la **qualité**.





2. **Synthèse** additive de **sons musicaux** :
   En utilisant les algorithmes déjà utilisés lors des TD il faut **recréer** les **notes** d'un **instrument** à partir d'aucun son.

3. **Synthèse** de son basée sur **l'analyse** fine des **harmoniques** :
   En utilisant une **note enregistrée** il faut **analyser** son **spectre** et **reproduire** le **son** le plus fidèlement possible.



Sujet choisi :

Le **sujet** que nous avons traité est le **sujet numéro 2**. Celui-ci est considéré comme « le plus simple » par l'enseignant et vu notre **niveau débutant** en **programmation** nous souhaitons ne pas nous lancer dans un projet qui serait **au dessus de nos moyens**. Au lieu de prendre le sujet 3 et de n'arriver qu'à réaliser la moitié, nous souhaitons réaliser le sujet 2 et **atteindre nos objectifs**.

Pour ce sujet nous allons utiliser des scripts Matlab® qui créent une **enveloppe** et une **synthèse**. Nous devrons **traduire** ces scripts **en** langage **java** et finir par jouer le son du **chronogramme** créé.

Travail réalisé :

Nous avons réussi à traduire les scripts en java de façon correcte. Nous créons le **temps discret**, la **somme** des **harmoniques**, une **enveloppe** et un **chronogramme correspondants** à nos attentes. Nous avons pu réaliser ces modules assez rapidement, ils ont, en effet, été codés en un week-end, même si il a ensuite fallu **améliorer** certains points). Nous les avons **testés** et **dessinés** en utilisant Gnuplot lorsque cela était possible.

Nous avons réussi ensuite à **jouer les notes**. Cependant lorsque nous souhaitons jouer plusieurs fois une même note, nous ne l'entendons qu'une seule et unique fois.

Identification des modules et explication de la partie théorique :

La création de la note de flute se fait en **plusieurs étapes**, et pour chacune d'entre elles, nous avons créé une **méthode** afin de reproduire le même comportement que pour la création d'une note sur Matlab.

Processus et identification des modules de la synthétisation d'une note :

1) Calcul des **fréquences harmoniques**,
2) Calcul du **temps discret** en fonction de la **fréquence d'échantillonnage**,
3) Calcul de la **somme** des **différents harmoniques** : $f(t_n) = \sum_{n=1}^{\infty}(a_n \sin(2\pi f_n t_i))$,
4) Calcul de **l'enveloppe**,
5) Calcul de la **multiplication** de **l'enveloppe** avec la **somme** des harmoniques,
6) Calcul des **valeurs normalisées** de la multiplication comprises entre -1 et 1,

Théorie de la synthétisation d'une note :

Théorème des séries de Fourrier :

Tout signal (sous certaines hypothèses de **périodicité**) peut se **décomposer** en **sommes** de **sinus** et cosinus.

La **fréquence** (f_n) des sinus correspond aux **différents harmoniques**. Les **coefficients** (a_n) de ces **harmoniques** varient d'un instrument à l'autre. Dans notre programme **il suffit** d'ailleurs **de changer ces valeurs** dans le tableau pour avoir l'impression **d'entendre un instrument différent**. Il **manquerait** de **créer** une **enveloppe différente** pour obtenir le son de cet instrument.

Le **temps discret** nous est **indispensable** pour pouvoir **calculer les sinus**. En **numérique** il n'y a **pas de temps analogique**, nous sommes donc obligé de **synthétiser le temps** aussi.

La **somme des harmoniques** est utile pour **avoir** une **note ressemblante** à une vraie note d'instrument et non une note pure (uni fréquentielle).

L'enveloppe, tout comme les coefficients des harmoniques, **varie** d'un instrument à l'autre. Nous devons donc la calculé pour être le plus fidèle possible au notes réelles. L'enveloppe d'un signal sonore est **l'évolution de son amplitude au cours du temps**. Un segment **croissant** correspond à une **attaque** (le volume **augmente**), un segment **constant** correspond à une phase de **maintient** (le volume reste au **même niveau**), un segment **décroissant** correspond à une **chute** (le son **s'atténue**).

La **multiplication** de l'enveloppe et de la somme des harmoniques **créée un chronogramme** qui sera par la suite **utilisé pour jouer le son**. Les valeurs d'un son joué doivent être comprises entre [-1 ; 1], pour cela nous cherchons la plus grande valeur et nous divisons toutes les valeurs de la multiplication par celle-ci.

Les autres méthodes du code correspondent à la **création de fichiers** ou la **lecture du son** créé et nous ont été données auparavant car nous n'avons actuellement pas les connaissances nécessaires pour les écrire nous même.

Répartition des tâches :

La répartition des tâches a été faite en **favorisant le travail de groupe** plutôt que le travail individuel. Ainsi, nous avons dans un premier temps « décortiqué » et **analysé le sujet** à deux pour avoir la **meilleure vision possible du travail à accomplir**. Nous avons ensuite commencé à codé individuellement certaines méthode pour gagner un peu de temps (notamment, Auzias a réalisé dans un premier temps la méthode synth() et Genevier la méthode correspondant à l'enveloppe), mais nous sommes ensuite très vite revenu à un travail à deux pour plus de cohérence et aussi car nous avons décidé de grouper toutes les **méthodes essentielles à la création du son dans une même classe**. Ainsi, nous avons travaillé très efficacement un week-end et finalement produit une première version du code que nous avons ensuite amélioré lorsque nous le pouvions.

Programmation :

Tableau 1 : Répartition de la programmation

Module :	Réalisé par (auteur principal) :
FreqHarmonique	Sylvestre
TempsDiscret	Sylvestre
Synth	Maël
Enveloppe	Maël
Chronogramme	Maël
Filewrite, tobytearray, play	Maël et Sylvestre
FFT	Sylvestre
Debug (la classe java)	Maël
Test	Sylvestre

Ce tableau indique l'auteur principal d'un module mais la plupart du temps nous les avons travaillés tous les deux.

Ainsi, nous avons décidé qu'il était peut être préférable de **perdre un peu de temps en travaillant systématiquement à deux**, sachant que cela **profiterai à la cohérence de notre classe principale** et finalement à un **gain de temps ensuite** car il n'était pas nécessaire de mettre en relation deux codes créés séparément.

Rédaction :

Tableau 2 : Répartition de la rédaction

Module :	Réalisé par (auteur principal) :
Introduction	Maël
Présentation du sujet choisi	Maël
Identification des modules et explication théorique	Sylvestre et Maël
Répartition des tâches	Sylvestre
Explication des modules	Sylvestre
Vérification des modules et de la somme des modules	Maël
Mise en œuvre du projet	Maël
Conclusion	Sylvestre

Explication des modules :

Nous allons dans cette partie du rapport essayer de **clarifier le rôle des méthodes les plus importantes** de la classe note. Nous mettrons de coté les méthodes qui correspondent par exemple à l'écriture d'un fichier sur le disque, car elles ne sont pas intégralement le fruit de notre travail, et surtout présentent moins d'intérêt.

Méthode freqHarmoniques() :

Méthode qui crée un **tableau** de la taille du nombre de coefficients des harmoniques et qui **contient** les **fréquences pour chaque harmoniques**, essentielles à la création d'un signal (en effet, la fréquence de chaque harmonique a une importance cruciale pour la définition du signal).

Méthode tempsDiscret() :

Méthode qui crée un **tableau contenant** les valeurs des **temps discrets**, c'est-à-dire un tableau de la forme 0, 1/FE, 2/FE, 3/FE…
Ce tableau sera ensuite essentiel pour synthétiser le signal pour chaque valeur de ces temps discrets.

Méthode synth() :

Méthode qui **synthétise** le **signal pour chaque case** du tableau **de temps discret**, à partir du tableau des **fréquences harmoniques**. Le résultat est stocké dans un tableau contenant donc FE+1 valeurs.

Méthode env() :

Méthode qui crée **l'enveloppe** correspondant à la note d'une flute. On stockera le résultat dans un tableau également de taille FE+1, on pourra ainsi plus tard le **multiplier** au **signal synthétisé** à l'aide de la méthode suivante. A noter que pour réaliser cette méthode, nous avions deux possibilités : suivre le script Matlab et le retranscrire en java ou adopter une enveloppe préexistante, créée par Philippe Guillaume, expert en reconstitution de tels signaux. Nous avons adopté la deuxième méthode, validée par M. Leroux, car celle-ci est plus clair, plus simple a réalisé, et plus compréhensible. Pour cela, nous avons **calculé les ordonnées** de la courbe **de l'enveloppe** (trouvée sur Internet) **dans un tableau**.

Méthode chronogramme() :

Méthode qui **crée le signal final**, elle **applique l'enveloppe** au **signal synthétisé**.

Méthode play() :

Méthode qui **permet de jouer le son** créé sur les haut-parleurs.

Vérification des modules et de la somme des modules :

Lors de la programmation il arrive, plus ou moins fréquemment, que le programme ne donne **pas le résultat souhaité**. Cela nous est arrivé plusieurs fois et nous avons donc eu recours à **différentes méthodes de débogage**.

A noter que les débogages de la classe Note sont particuliers. En effet nous avons utilisé beaucoup de tableaux ayant une grande taille (44100 environ). Les algorithmes les remplissant sont plus ou moins lents, ce qui nous a poussés à **optimiser la rapidité** d'exécution de notre code.

Rappel : le constructeur de la classe Note fonctionne comme suit :

- Remplissage du **tableau des harmoniques**,
- Remplissage du **tableau de temps discret**,
- Remplissage du **tableau du signal synthétisé** : $f(t_n) = \sum_{n=1}^{\infty}(a_n \sin(2\pi\, f_n\, t_i))$, avec f_n la fréquence de l'harmonique au rang n et t_i le temps discret,
- Remplissage du **tableau de l'enveloppe**,
- Remplissage du **tableau du chronogramme** en multipliant les tableaux d'enveloppe et du signal synthétisé case par case.

Le constructeur représente la plus grande partie de notre travail et du travail de débogage. Les 5 parties ci-dessus sont chacune des méthodes (pour plus de visibilité) de la classe. Nous allons voir dans le détail les méthodes de vérifications.

Remplissage du tableau des harmoniques :

Ce tableau est assez petit puisqu'il a la taille du tableau des coefficients harmoniques et nous en avons dans notre cas moins de 10. Le **tableau** est donc **facilement affichable** et lisible.

Pour vérifier son fonctionnement il nous a fallu juste **afficher toutes les cases** à l'écran **après** son **remplissage**. Nous avons pu ainsi voir que le tableau de case comprise entre [0,n] est rempli de la façon suivante :

T[i] = (i+1) * f, où i (compris entre 0 et n) est l'entier d'itération et le numéro de la case.

Remplissage du tableau de temps discret :

Ce tableau a 44101 cases. Il n'est **pas envisageable de l'afficher** dans la console. La première case du tableau contient le nombre 1/FE, où FE est la fréquence d'échantillonnage. La case suivante a la somme de la case précédente et de la première.

Pour vérifier son fonctionnement nous avons **enregistré le tableau dans un fichier** puis, de façon aléatoire, calculé le nombre que nous voulions pour certaine case. Nous avons immédiatement vu que la fonction pouvait marcher car les nombres sont croissants et le dernier s'approche de 1 (le dernier est : 0.9999773).

Une méthode différente possible aurait été de fixer une **marge d'erreur** et de **soustraire la case n avec la case n-1** et d'afficher le nombre de cas où la différence sort de la marge d'erreur. Nous aurions pu avoir un **retour sur la précision des chiffres** et la **justesse** de notre **vecteur temps**.

Remplissage du tableau du signal intermédiaire :

Ce tableau et cette méthode sont ceux qui ont **le plus été travaillé**. En effet le son en milieu de projet n'étant pas ressemblant à une flute nous avons vérifié cette méthode **beaucoup de fois** avec à chaque fois la même conclusion : notre **algorithme** est **valide**, et le même résultat : le **son** est **mauvais**. C'est pourtant sur cette méthode et celle où le son est joué que les erreurs ont le plus de risque d'être produites. La méthode jouant le son est juste une utilisation basique d'une classe (qui nous est inconnue) et de l'application d'une méthode. Nous nous sommes rendus compte tout à la fin du projet que c'était une des valeurs du constructeur qui était mauvaise.

Pour déboguer cette méthode nous avons créé une classe **spécialement conçue pour corriger** cette fonction. Nous avons créé une « note » de fréquence 1 Hz (**fréquence inaudible**, mais la **lisibilité** sur le chronogramme se trouve **grandement améliorée**). Nous avons modifié le nombre et les valeurs des coefficients harmoniques. Nous avons **affiché** avec Gnuplot **chaque calcul d'harmonique** sur la durée du signal final **puis les sommes** des deux premiers harmoniques, puis les trois premiers, puis les quatre premiers…jusqu'à la somme de la totalité des harmoniques :
Note : le coefficient de l'harmonique de rang n est n-1 : $a_0 = 1$, $a_1 = 2$, $a_2 = 3$…

Tableau 3 : Graphiques des harmoniques de débogage :

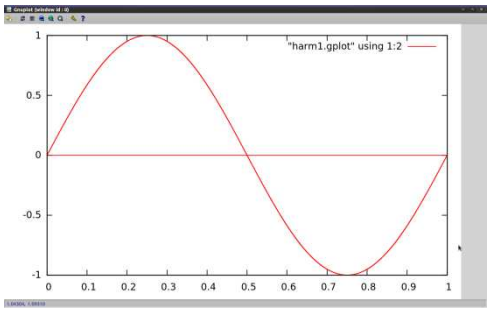


Figure 1 : f1 = 1*sin(2*pi*1*t)

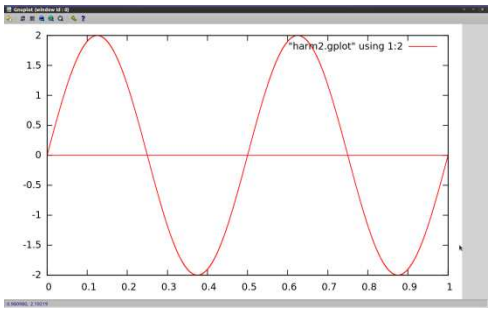


Figure 2 : f2 = 2*sin(2*pi*2*t)

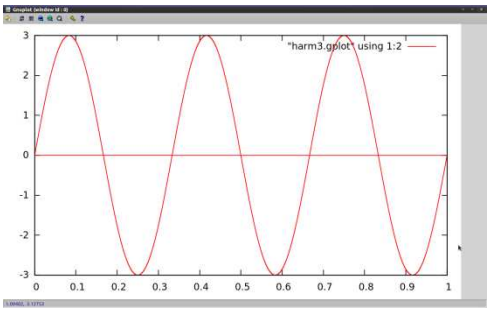


Figure 3 : f3 = 3*sin(2*pi*3*t)

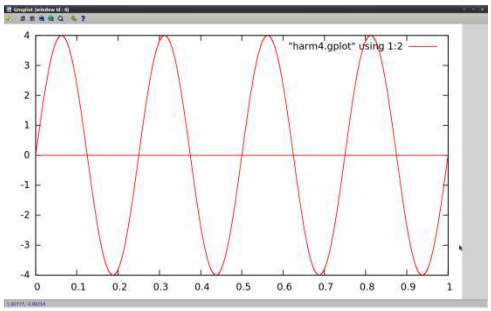


Figure 4 : f4 = 4*sin(2*pi*4*t)

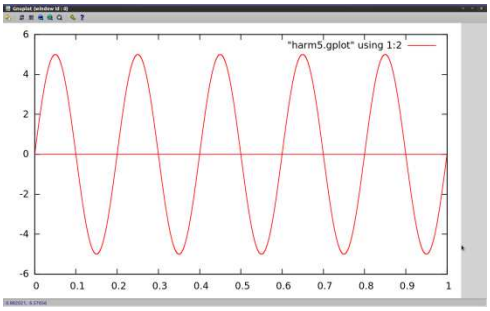


Figure 5 : f5 = 5*sin(2*pi*5*t)

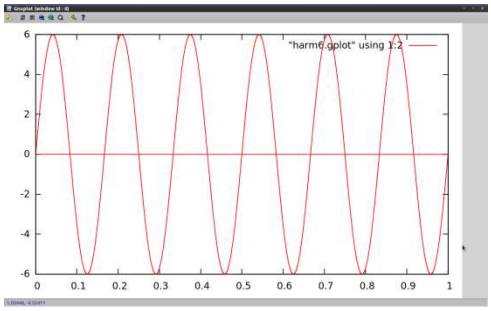


Figure 6 : f6 = 6*sin(2*pi*6*t)

Tableau 4 : Graphique des sommes des signaux fn

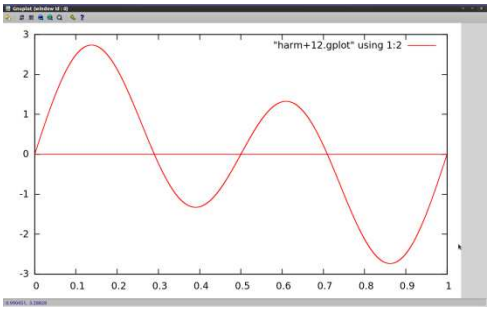


Figure 7 : Somme des signaux f1,f2

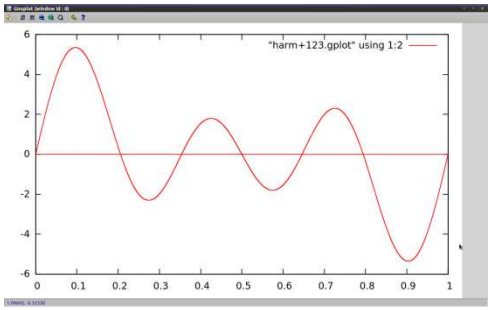


Figure 8 : Somme des signaux f1,f2,f3

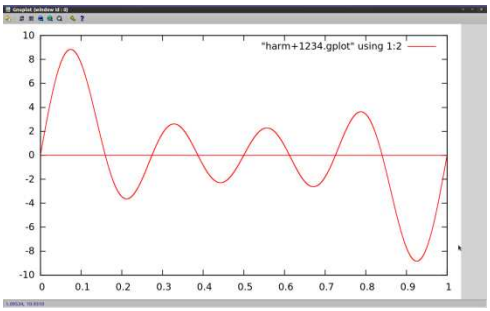


Figure 9 : Somme des signaux f1,f2,f3,f4

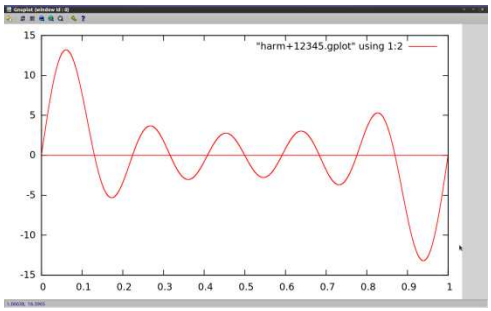


Figure 10 : Somme des signaux f1,f2,f3,f4,f5

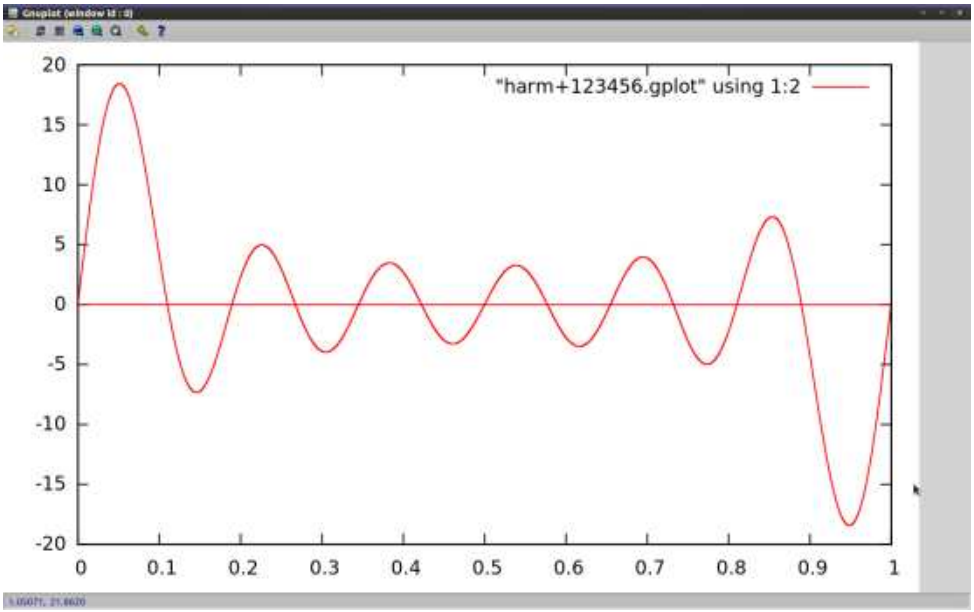


Figure 11 : Somme total des signaux fn

Nous obtenons des **graphiques** qui **correspondent** à nos **attentes**, et concluons qu'il n'y a **pas d'erreur** dans l'algorithme de remplissage.

Remplissage du tableau de l'enveloppe :

Cette méthode a été **très facile** à déboguer. Son bon fonctionnement et sa vérification a confirmé une fois de plus que le tableau de temps discret est bien rempli. Pour la vérification il a suffit de **visualiser** avec Gnuplot le **résultat obtenu**. Nous connaissions la forme de l'enveloppe que nous voulions obtenir et en visualisant les deux enveloppe nous avons facilement déduis que la méthode était bonne.

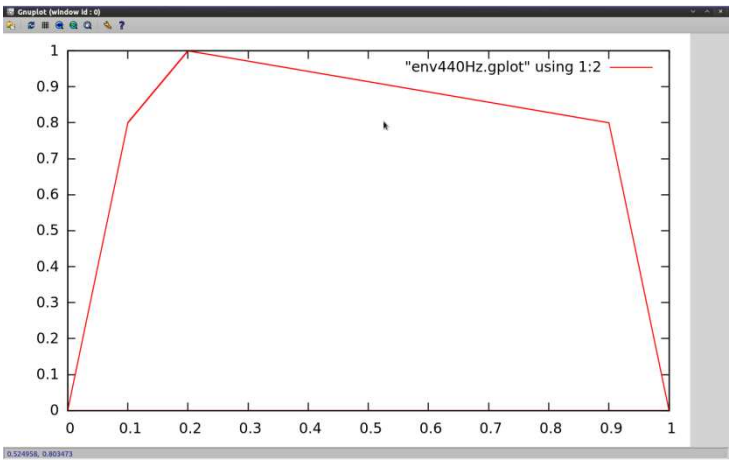


Figure 12 : Graphique de l'enveloppe

Remplissage du tableau du chronogramme :

La vérification de cette méthode a été tout aussi rapide que la vérification de la méthode précédente. Cette méthode ne fait que **multiplier** case par case **les tableaux** du signal intermédiaire et l'enveloppe. Nous avons vu ci-dessus a quoi correspond l'enveloppe, et pour le signal suivant :

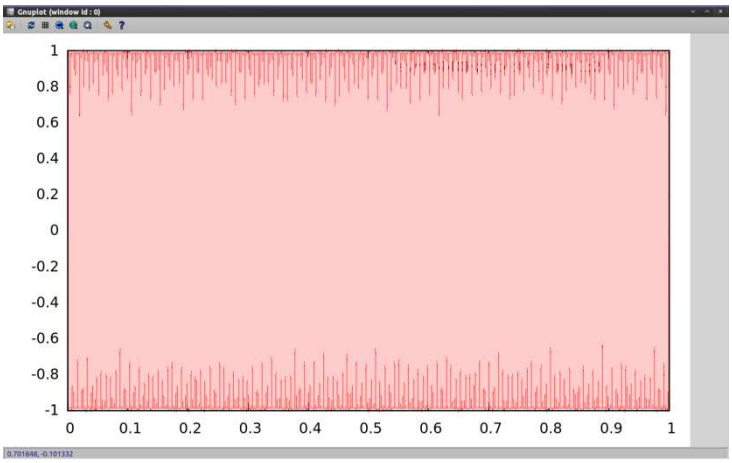


Figure 13 : Chronogramme sans enveloppe d'un la3 de flute

Nous obtenons le chronogramme suivant :

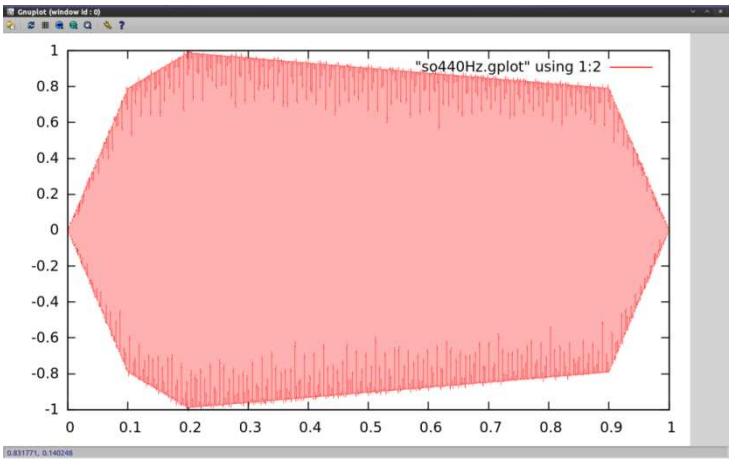


Figure 14 : Chronogramme d'un la3 d'une flute

En **visualisant** les **deux graphiques** qui servent de **ressources** à la méthode et le **graphique produit** on peut facilement en déduire que **l'algorithme** et le **fonctionnement** sont **bons**.

L'ensemble des fonctions :

Toutes les méthodes que nous avons vues sont validées de façon individuelle et du point de vu de l'ensemble du programme. Les **méthodes non évoquées** sont les **méthodes** qui nous ont été **fournies** par les enseignants. Par exemple celle qui permet d'écrire dans un fichier qui **fonctionne parfaitement**.

Mise en œuvre du projet :

Ce projet a été réalisé sous GNU/Linux. Vous pouvez aisément :

Action	Répertoire
Lire la documentation	doc
Modifier les fichiers sources	src
Exécuter les fichiers binaires	bin

Pour savoir comment faire il suffit de lire le paragraphe ci-dessus qui vous intéresse :

Documentation :

La **documentation** est présente pour **toutes nos méthodes**. En effet elles ont été **correctement commentées** de façon à pouvoir être réutilisée. Pour lire cette documentation il vous suffit d'ouvrir le **fichier** « **index.html** » avec un explorateur internet.

Modification :

La **modification** d'une **de nos classes** est **simple**, il suffit d'**éditer** le fichier **code source** java avec un éditeur de texte et de réaliser les modifications souhaitées. Pour compiler et exécuter reportez-vous aux paragraphes suivant.

Compilation :

La **compilation** des fichiers sources se fait à partir d'une **console sous GNU/Linux**. Pour cela allez dans le répertoire « src » puis tapez :

```
javac –d ../bin nomDeLaClasse.java
```

Ainsi le fichier nomDeLaClasse.class sera créé dans le répertoire bin et la classe Test pourra s'en servir pour s'exécuter.

Exécution :

L'exécution des **fichiers binaires** se fait, elle aussi, à partir d'une console sous GNU/Linux. Pour cela aller dans le répertoire « bin » puis tapez :

```
java Test
```

Ainsi le fichier Test.class s'exécutera et réalisera ce qui est programmé.

Affichage des graphiques :

L'affichage des graphiques se fait à l'aide de **Gnuplot**. Vérifier s'il est bien installé sur votre machine. Rendez-vous sur http://www.gnuplot.info/ pour l'obtenir. Aller ensuite dans le dossier « bin ». Exécuter le fichier Test.class (cf ci-dessus). Celui-ci va créer différent fichier « *.gplot ». Vous pouvez afficher leur valeur en utilisant le script que j'ai créé. Pour exécuter ce script tapez :

```
./gplot fichiersAAfficher.gplot
```

Ainsi les graphiques apparaîtront sur l'interface graphique de votre environnement.

Note : vous pouvez remplacer « fichiersAAfficher.gplot » par « *.gplot » pour afficher tous les graphiques générés et présents dans le répertoire.

Conclusion :

Finalement, ce projet nous a permis de **découvrir** ou **approfondir** certaines **notions** dans **deux matières** bien distinctes, en **programmation** et en **SSII**, mais surtout nous a appris à **combiner** nos **savoirs** respectifs dans ces deux matières. En effet, on se rend d'avantage compte de leur utilité en **appliquant** ainsi **nos connaissances** sur **un même** projet. Ainsi, nous avons découvert de nouvelles méthodes en programmation tout en réutilisant nos connaissances acquises lors des TD de SSII sur Matlab. En se basant sur les **scripts Matlab** déjà créés, nous sommes parvenus à **créer** un **son** en **java** de **flûte** dont la fréquence est modifiable.

Il faut tout de même préciser que pour **atteindre ce résultat**, nous avons dû surmonter **différentes difficultés**. Tout d'abord il nous a fallu **comprendre les codes** mis à notre disposition, ceux en java mais également les scripts Matlab. Ensuite, la traduction d'un langage vers l'autre n'a pas tout le temps été simple, et il a parfois fallu « ruser ». Enfin, nous avons eu un **problème majeur** afin d'obtenir un son, et nous avons réussi à résoudre ce problème uniquement **la dernière semaine** avant le rendu, on peut donc aussi mentionner le **respect** des **contraintes de temps comme** une éventuelle **difficulté**, bien que nous n'avons pas eu trop de problèmes à ce niveau là.

Pour finir, nous gardons bien à l'esprit que notre projet peut être amélioré. En effet, on pourrait rajouter la possibilité de **choisir l'instrument à jouer**. Nous avions cependant dès le début décidé de ne **pas** se fixer **d'objectifs trop hauts** par rapport à notre niveau – nous sommes effectivement débutants en programmation – pour être **sûr de bien pouvoir approfondir** et **comprendre chaque étape** de notre projet.

Table des illustrations :

Figure — Page

Tableau — Page