Liens entre musique et mathématiques

Brice Vittet & Lucas Bolbènes

15 juillet 2022

Tous les programmes évoqués dans ce rapport sont accessibles sur le dépôt github suivant :
https://github.com/lucasbolbenes/2022_StageExcellenceUGA

Les mots en italique sont des liens cliquables.

Table des matières

Chapitre 1

Contexte du stage

1.1 Présentation du laboratoire

Nous avons été accueillis par l'Institut Fourier, un des laboratoires de Mathématiques de l'Université Grenoble Alpes. Ce laboratoire est associé au CNRS depuis sa création en 1966. Les thèmes de recherche sont variés et classés selon les catégories suivantes :

- Algèbre et géométrie
- Combinatoire et didactique
- Géométrie et topologie
- Physique mathématique
- Probabilités
- Théorie des nombres

Nous avons été accompagné par M. Romain Joly, maître de conférences à l'Institut Fourier. Ses recherches sont incluses dans la catégorie « Physique mathématique ». M. Joly s'intéresse aux équations aux dérivées partielles. Son travail consiste notamment à étudier des modèles issus d'observations ou d'intuitions, par exemple en physique ou en biologie, et d'en valider l'aspect mathématique.

FIGURE 1.1 – Institut Fourier

1.2 Présentation des stagiaires

Nous sommes deux étudiants de L2 en Mathématiques et Informatique, Brice Vittet et Lucas Bolbènes respectivement associés au campus de Grenoble et à l'antenne de Valence. Nous avons effectué notre stage dans le cadre des stages d'excellence UGA. Nous l'avons réalisé en binôme ce qui explique que notre rapport soit commun. Nous allons tout deux intégrer une école d'ingénieur informatique pour l'année 2022-2023.

1.3 Historique du stage

Le premier sujet abordé durant notre stage était la création des gammes musicales. Pourquoi utilise-t-on une gamme de 12 notes en Occident ? Derrière ce choix se cache des raisons mathématiques que nous avons étudiées. Dans un second temps nous avons cherché des modules Python permettant de créer des sons synthétiques, d'écouter des sons, ainsi que de traiter des fichiers wav (nous avons notamment utilisé les modules winsound et PyAudio). Ces modules nous ont permis de produire nos premiers scripts, notamment *entendreNombre.py*. Nous avons également étudié des phénomènes liés à la musique comme par exemple le phénomène de battement ou encore l'illusion de Shepard. Durant notre stage nous avons pu assister à une présentation de notre tuteur au lycée Vaucanson et ainsi en apprendre plus sur les travaux de Joseph Fourier. Nous avons également pu participer à une présentation entre stagiaires où nous avons échangé et présenté successivement nos travaux. La dernière partie de notre stage fut la conception et l'implémentation de filtres et de traitements sur des fichiers wav.

Chapitre 2

Construction des gammes

2.1 Gammes pythagoriciennes

2.1.1 Une construction par la quinte

Le cycle de la quinte

La construction de la gamme de Pythagore repose sur une construction par la quinte. Pythagore partait du principe qu'une note quelconque était toujours consonante avec sa quinte, mais aussi avec la quinte de sa quinte. Nous voulons donc une gamme de notes comprises entre la fréquence de la fondamentale et son octave. Sachant cela, l'algorithme de construction d'une gamme par la quinte est le suivant :

Soit f_0 la fréquence de la note fondamentale.
Soit $(U_k)_{k\in\mathbb{N}}$ une suite telle que U_k soit la fréquence de la k-ième note.
Soit N le nombre de notes que l'on choisit de manière arbitraire.

$$\begin{cases} U_0 = f_0 \\ U_{k+1} = \begin{cases} \frac{U_k \cdot \frac{3}{2}}{2}, & \text{si } U_k \cdot \frac{3}{2} > 2 \\ U_k \cdot \frac{3}{2}, & \text{sinon} \end{cases} \end{cases}$$

Une fois que les N notes sont construites, on les trie dans l'ordre croissant.

La première condition pour le choix de U_{k+1} est due au fait que multiplier U_k par $\frac{3}{2}$ peut nous faire sortir de l'intervalle $[f_0, 2f_0]$. Pour éviter cela on divise la note obtenue par 2, ce qui nous permet d'avoir la note dont elle est l'octave. Cet algorithme ne s'arrête pas de lui même, nous le démontrerons juste après. Il est donc nécessaire de choisir de manière arbitraire un nombre de notes pour arrêter la construction de la gamme. Une fois que la gamme est construite, il faut ranger les éléments dans l'ordre croissant. Les gammes pythagoriciennes correspondent à l'ensemble des gammes construites à partir de cette méthode.

Un cycle infini

Afin de démontrer que le cycle est infini nous avons besoin d'introduire un théorème que l'on considérera comme classique et dont on ne fera pas la démonstration ici.

Théorème 2.1 (Unicité de la décomposition en facteurs premiers) *Tout entier strictement positif peut être écrit comme un produit de nombres premiers d'une unique façon, à l'ordre près des facteurs.*

Le cycle de la quinte n'est pas réellement un cycle dans le sens où on ne pourra jamais retomber avec exactitude sur une octave de la note fondamentale. En effet, on constate que tous les intervalles sont séparés deux à deux par une puissance de $\frac{3}{2}$.

— Quelle que soit la note que l'on construit avec notre algorithme, on peut écrire sa fréquence de la manière suivante : $f_0 \cdot \frac{(\frac{3}{2})^p}{2^r}$

— Quelle que soit l'octave de la note fondamentale, sa fréquence peut être écrite de la manière suivante : $f_0 \cdot 2^q$

Pour qu'une de nos notes ait une fréquence égale à celle d'une de nos octaves, il faudrait que ces deux expressions puissent être égales. Soient $q \in \mathbb{N}^*$, $p \in \mathbb{N}^*$ et $r \in \mathbb{N}$ supposons :

$$f_0 \cdot \frac{(\frac{3}{2})^p}{2^r} = f_0 \cdot 2^q$$

On aurait alors :

$$\begin{aligned} \frac{(\frac{3}{2})^p}{2^r} &= 2^q \\ (\frac{3^p}{2^p}) &= 2^q \cdot 2^r \\ 3^p &= 2^q \cdot 2^r \cdot 2^p \end{aligned}$$

Et donc :

$$3^p = 2^{q+r+p}$$

On aurait donc une puissance de 2 égale à une puissance de 3. D'après le théorème 2.1, cette égalité est impossible. On a alors $3^p \neq 2^{q+r+p}$ ce qui démontre bien que le cycle est infini.

2.1.2 Défauts des gammes construites avec le cycle de la quinte

Les gammes construites avec avec le cycle de la quinte ont un défaut majeur. Comme expliqué précédemment, le cycle de la quinte est infini. Seulement lorsque l'on se ramène à un nombre fini de notes pour former une gamme, on doit faire en sorte que la gamme puisse boucler. Concrètement si on prend la gamme de 12 notes construite avec l'algorithme précédent, l'écart entre la fréquence de la dernière note de cette gamme, et la fréquence de l'octave supérieure ne sera pas une quinte parfaitement juste. Si on multiplie la fréquence de cette dernière note par $\frac{3}{2}$ on ne retombera pas sur $2 \cdot f_0$. Par conséquent, les 11 premières quintes sont parfaitement justes, mais la dernière est légèrement trop courte, c'est la « quinte du loup ». L'écart entre la fréquence de la dernière note de la gamme pythagoricienne à 12 notes et la fréquence du premier octave est appelé comma pythagoricien ou comma diatonique.

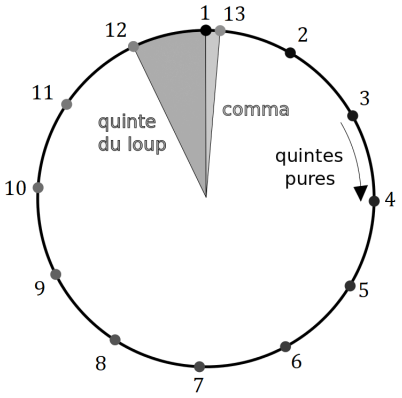

FIGURE 2.1 – Cycle de la quinte avec 12 notes

La quinte du loup devait être prise en compte lors de la création de partitions. Elle a longtemps été cachée dans des intervalles peu utilisés. Elle était une limite à la créativité en rendant notamment la transposition compliquée. Tous ces problèmes furent réglés avec l'apparition des gammes tempérées.

2.2 Gammes tempérées

2.2.1 Construction de la gamme tempérée

Principe

Le principe de la gamme tempérée est de découper l'octave en notes de telles sorte que le facteur séparant deux notes soit le même pour toutes les notes. Cela implique d'avoir toutes les quintes légèrement fausses, mais cette construction supprime les problèmes de transposition et facilite la pratique de la musique orchestrale. Définissons un certain α tel que la relation entre la fréquence U_k d'une note et la fréquence U_{k+1} de la note suivante soit : $U_{k+1} = U_k \cdot \alpha$. Cette construction correspond en fait à une suite géométrique de raison α.

Un peu de mathématiques

Définissons un intervalle allant de f_0 à $2 \cdot f_0$. On considèrera que f_0 correspond à la fréquence de la note fondamentale, et que $2 \cdot f_0$ correspond à la fréquence de l'octave supérieure. Soient $(U_k)_{k \in \mathbb{N}}$ une suite telle que U_k soit la fréquence de la k-ième note et N soit le nombre de notes que l'on choisit de manière arbitraire. L'objectif est de découper cette intervalle en N sous-intervalles de telle sorte que :

— $U_0 = f_0$
— $U_N = 2 \cdot f_0$
— $\forall k, U_k = f_0 \cdot \alpha^k$

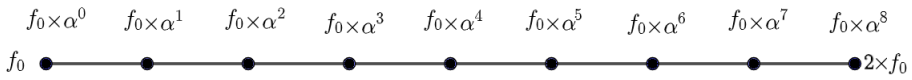

FIGURE 2.2 – Construction d'une gamme tempérée pour N = 8

Déterminons α en fonction de N, on veut que $2 \cdot f_0 = \alpha^N \cdot f_0$ donc :

$$\begin{aligned} 2 \cdot f_0 &= \alpha^N \cdot f_0 \\ 2 &= \alpha^N \\ \ln 2 &= \ln(\alpha^N) \\ \ln 2 &= N \cdot \ln \alpha \\ \frac{\ln 2}{N} &= \ln \alpha \\ \alpha &= e^{\frac{\ln 2}{N}} \end{aligned}$$

Algorithme

Maintenant que l'on a déterminé α nous pouvons construire notre gamme à l'aide de l'algorithme suivant.

Soit f_0 la fréquence de la note fondamentale.
Soit $(U_k)_{k\in\mathbb{N}}$ une suite telle que U_k soit la fréquence de la k-ième note.
Soit N le nombre de notes que l'on choisit de manière arbitraire.

$$\begin{cases} U_0 = f_0 \\ \\ U_k = e^{\frac{\ln 2}{N} \cdot k} \end{cases}$$

2.2.2 Choix de la valeur associée à N

Une idée pour choisir

Nous disposons désormais d'un algorithme qui permet de construire une gamme de N notes où le passage d'une note à sa suivante se fait en multipliant par une même valeur. On pourrait maintenant se demander si ce nombre de notes à une influence sur l'harmonie de notre gamme. Il est légitime de penser que la présence de la quinte, la tierce ou la quarte sont des éléments qui favorisent une meilleure harmonie. L'algorithme utilisée ne permet pas d'obtenir avec exactitude ces notes dans notre gamme. Mais il permet d'obtenir des notes qui en sont plus ou moins proches. L'idée, pour savoir si une gamme est meilleure qu'une autre est de déterminer dans une première gamme la note ayant la fréquence la plus proche de la fréquence de la quinte, d'en faire le rapport avec la quinte, puis de regarder la différence en valeur absolue entre ce rapport et 1 avant de refaire le même procédé avec une seconde gamme et de comparer les deux résultats. Plus une note est proche de la quinte, plus le rapport entre ces deux fréquences est proche de 1. C'est pour cette raison que nous avons décidé de calculer la différence en valeur absolue entre le rapport obtenue et 1, pour avoir un résultat qui tend vers 0 lorsque les notes sont de plus en plus proches. La gamme qui engendrera la plus petite valeur sera celle qui contient une note qui est plus proche de la quinte, et donc qui aura plus de chance d'avoir une bonne harmonie. On peut répéter ces procédés avec la tierce majeure, mineure et aussi la quarte.

Création d'un script et interprétation des résultats

Afin d'appliquer l'idée précédente, nous avons créé le script *etudeErreurGammes.py*. Le résultat est présenté sous forme de tableau dont chaque ligne représente une gamme ainsi que le résultat du procédé précédent appliqué pour la quinte, la tierce mineure, la tierce majeure et la quarte sur cette gamme. La première colonne représente le nombre de notes de la gamme courante. Voici les résultats obtenus :

n	Quinte	Tierce Maj	Tierce Min	Quarte
2	5.7E-02	1.3E-01	1.7E-01	6.1E-02
3	5.8E-02	7.9E-03	5.0E-02	5.5E-02
4	5.7E-02	4.9E-02	9.0E-03	6.1E-02
5	1.0E-02	5.6E-02	4.3E-02	1.0E-02
6	5.7E-02	7.9E-03	5.0E-02	5.5E-02
7	9.3E-03	2.5E-02	1.6E-02	9.4E-03
8	2.8E-02	3.7E-02	9.0E-03	2.7E-02
9	2.0E-02	7.9E-03	2.8E-02	2.1E-02
10	1.0E-02	1.5E-02	2.6E-02	1.0E-02
11	2.7E-02	2.9E-02	6.7E-03	2.8E-02
12	1.1E-03	7.9E-03	9.0E-03	1.1E-03
13	2.1E-02	9.8E-03	2.2E-02	2.1E-02
14	9.3E-03	2.5E-02	1.6E-02	9.4E-03

FIGURE 2.3 – Résultats pour les gammes allant de 2 à 14 notes

On remarque très clairement que l'erreur vis à vis de la quinte est minimale pour la gamme à 12 notes dans cet intervalle. On peut en déduire que ce nombre de notes est un très bon choix. De plus la prochaine gamme ayant une erreur inférieure à 12 pour la quinte est atteinte à 29 notes. Seulement ce nombre de notes commence à être trop grand. La gamme tempérée à 12 notes correspond à un excellent équilibre entre la faible valeur de l'erreur et la faible quantité de notes.

Chapitre 3

Le phénomène de battement

3.1 Introduction

Vie Courante

Le battement est un phénomène sonore que l'on peut notamment entendre lors de l'accordage d'un instrument à corde. Il en ressort alors un son dont l'intensité oscille en fonction du temps.

Explication

Ce phénomène est dû à l'interférence de deux ondes sonores périodiques de fréquences initiales très proches mais distinctes. Leur superposition peut conduire à une amplitude maximale ou nulle, ce qui entraine une variation de l'intensité par rapport le temps.

FIGURE 3.1 – Une personne en train d'accorder une guitare

On va alors chercher à visualiser cet effet sonore.

3.2 Visualisation

Graphique

Soient f_1, f_2 deux fonctions telles que :

$$\begin{cases} f_1(t) = c_1 \ \sin(\omega_1 2\pi t) \\ f_2(t) = c_2 \ \sin(\omega_2 2\pi t) \end{cases}$$

On choisit deux fonctions sinusoïdales de fréquences ω_1 et ω_2 différentes. En faisant la somme de ces deux fonctions avec $\omega_1 = 440$ Hz et $\omega_2 = 450$ Hz, et avec les amplitudes c_1, c_2 égales, nous obtenons la figure ci dessous :

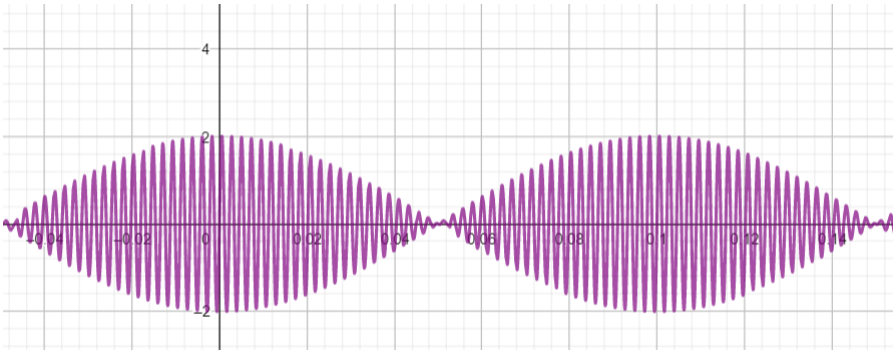

FIGURE 3.2 – somme de $f_1 + f_2$

Interprétation

On voit alors sur le graphe que l'amplitude sera maximale en 0 et minimale en 0.05 et que la fonction obtenue aura une période de 0.1 qui correspond à la distance entre les deux sommets. Le son alors produit sera alors plus fort quand l'amplitude est au maximum, et plus faible quand l'ampliutde est proche de zéro, ce qui produit l'effet de battement/variation.

On va alors essayer de comprendre mathématiquement ce phénomène.

3.3 Formulation

Formule

Étudions en détail la somme de deux sinus ayant une même amplitude c. Soit $f(t)$ la somme de ces deux fonctions sinus.

$$f(t) = f_1(t) + f_2(t)$$

En remplaçant on obtient :

$$f(t) = c \cdot sin(\omega_1 2\pi t) + c \cdot sin(\omega_2 2\pi t)$$

Or d'après la formule trigonométrique suivante :

Proposition 1

$$\sin(a) + \sin(b) = 2\sin\left(\frac{a+b}{2}\right) \cdot cos\left(\frac{a-b}{2}\right)$$

On a :

$$f(t) = c \cdot 2\sin(t2\pi \cdot \left(\frac{\omega_1 + \omega_2}{2}\right)) \cdot \cos(t2\pi \cdot \left(\frac{\omega_1 - \omega_2}{2}\right))$$

Interprétation

On voit alors que la somme de deux fonctions sinus est égale à un produit de fonctions sinus, dont la première a pour fréquence la moyenne des fréquences des fonctions initiales et la seconde a pour fréquence la demi-différence des fréquences initiales. Plus précisément, la première sinusoïde définira la période de la fonction f (ce qui correspond à la période du battement). Sa fréquence sera alors :

$$\left(\frac{\omega_1 + \omega_2}{2}\right)$$

La deuxième sinusoïde définira quant à elle l'amplitude de la fonction f. Sa fréquence sera alors :

$$\left(\frac{\omega_1 - \omega_2}{2}\right)$$

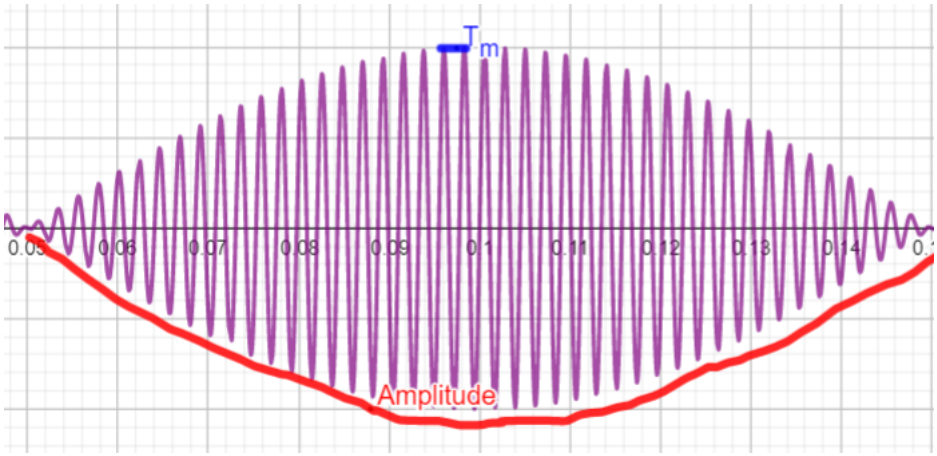

FIGURE 3.3 – L'action de la première sinusoïde en bleu, et la seconde en rouge

Remarque

On a pu entendre que la durée d'un battement en seconde était égal au décalage entre les deux fréquences, ce qu'on entend alors n'est pas la demi-différence des fréquences des deux sons mais seulement la différence.

Chapitre 4

Création de filtres grâce à la FFT

La FFT (Fast Fourier Transform) est un algorithme permettant de calculer la transformée de Fourier discrète d'une fonction. De manière intuitive, appliquer la transformée de Fourier à une fonction représentant un son (une amplitude variante par rapport au temps) permet d'obtenir une fonction représentant les fréquences appartenant au son d'origine. Il est important de comprendre que nous ne travaillons par réellement avec une fonction lorsque nous faisons du traitement numérique. En effet, étant donné que nous sommes dans l'univers discret, notre donnée d'origine est un tableau de valeurs. Ces valeurs correspondent aux valeurs de la fonction récupérées à un intervalle de temps régulier. D'ailleurs l'échantillonage correspond au nombre de valeur récupéré chaque seconde. À l'aide de cet algorithme ainsi que de certains modules Python, nous avons pu réaliser des modifications sur des fichiers wav. Le code derrière ces différents filtres sont tous disponibles dans la classe *Signal.py*.

4.1 Filtre passe-bas et filtre passe-haut

Pour appliquer un filtre passe-haut nous avons eu l'idée suivante. Tout d'abord on applique la FFT sur notre tableau. Ensuite, on supprime toutes les fréquences inférieures à une certaine valeur. Puis on applique la IFFT (Inverse Fast Fourier Transform) pour récupérer le son filtré.

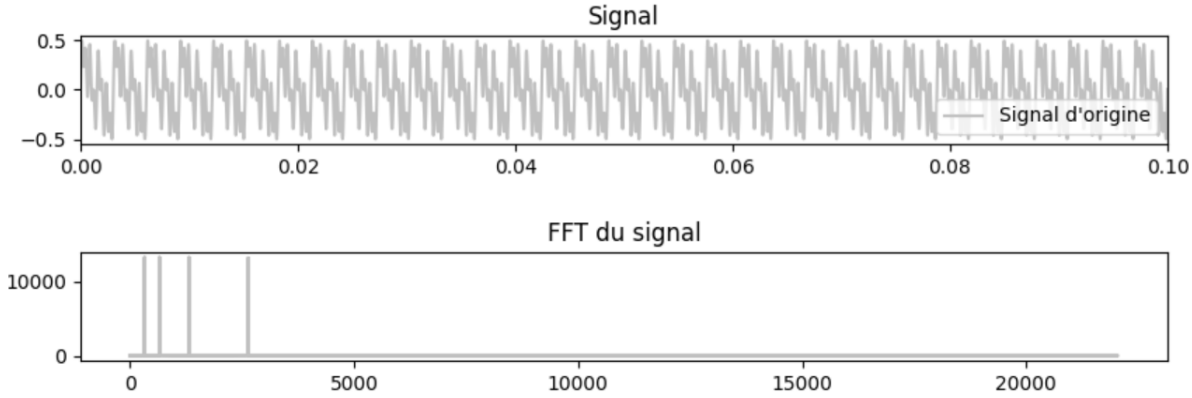

FIGURE 4.1 – Graphique du son d'origine

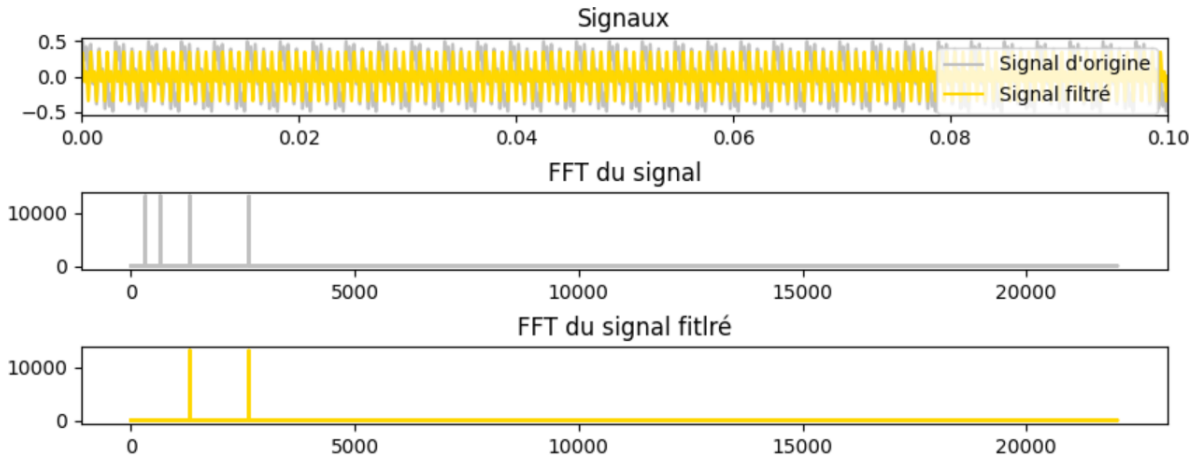

FIGURE 4.2 – Graphique du son filtré avec un filtre pass-haut de 1000 Hz

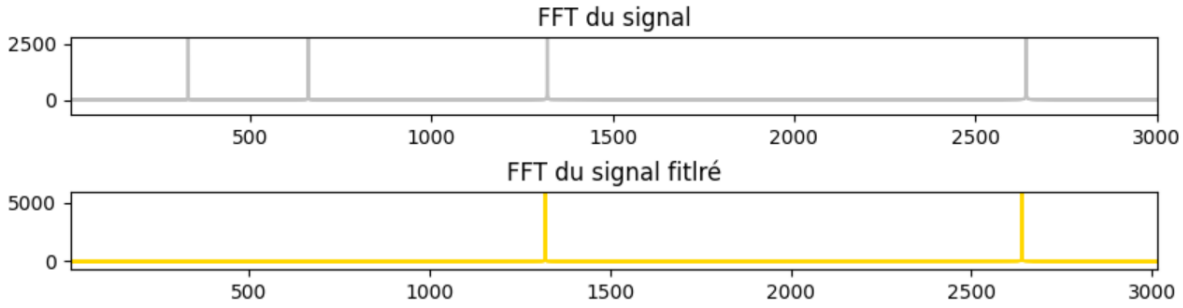

FIGURE 4.3 – Zoom sur le graphique du son filtré avec un filtre passe-haut de 1000 Hz

C'est exactement le même principe pour le filtre passe-bas mais en enlevant cette fois ci toutes les fréquences supérieures à une certaine valeur. Pour tester vous même ce programme lancer le programme *app.py*.

4.2 Modifier la tonalité d'un son

Pour baisser la hauteur d'un son, l'idée est la suivante. Tout d'abord on applique la FFT sur notre tableau. Ensuite on décale les fréquences vers la gauche un certain nombre de fois. Puis on applique la IFFT pour récupérer le son filtré. C'est exactement le même principe pour augmenter la tonalité du son mais en décalant cette fois-ci vers la droite. Pour tester vous même ce programme lancer le programme *app.py*. Les fonctions associées à la diminution et à l'augmentation de la tonalité sont respectivement nommées shift_left et shift_right dans le fichier *Signal.py*.

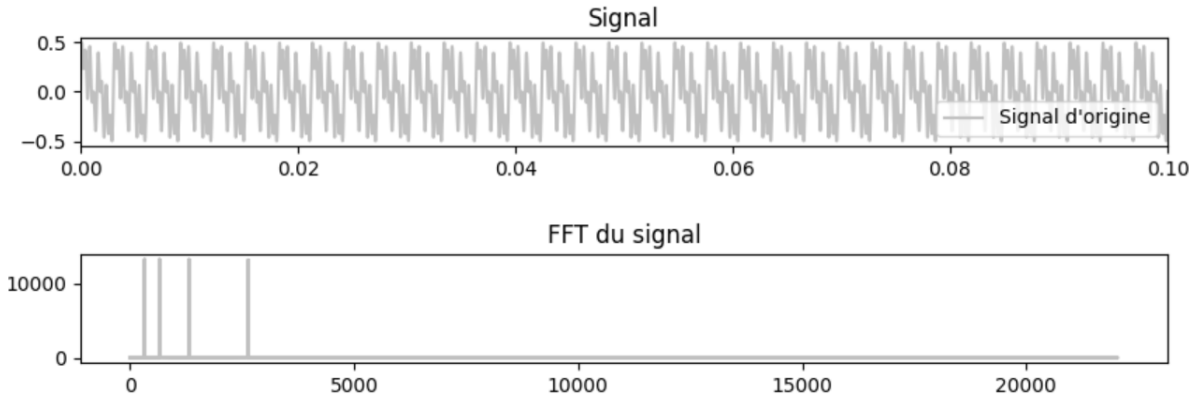

FIGURE 4.4 – Graphique du son d'origine

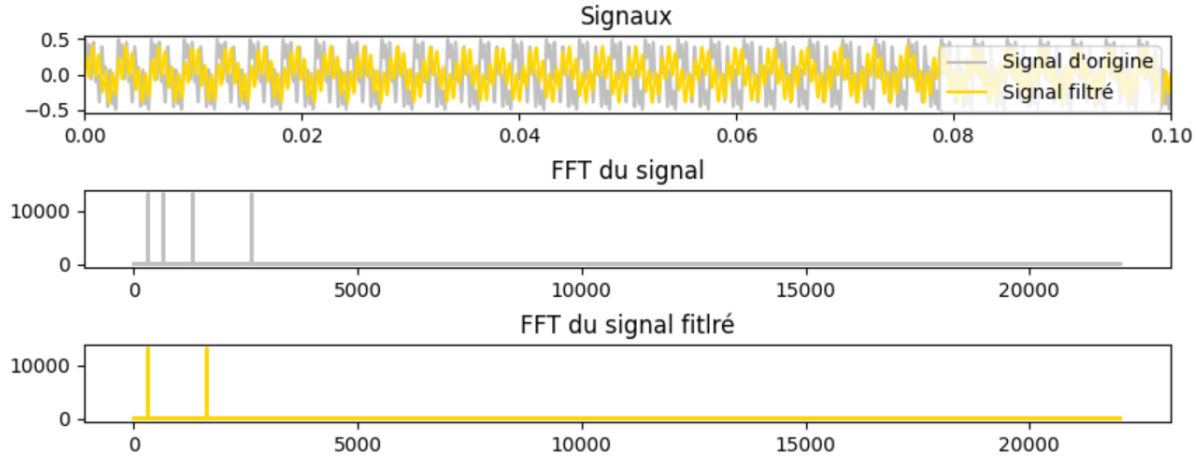

FIGURE 4.5 – Graphique du son avec un décalage des fréquences de 1000 Hz vers la gauche

4.3 Auto-tune

Notre dernier objectif fut de créer un auto-tune. L'algorithme imaginé pour réaliser ce traitement est un peu plus complexe que les précédents. Tout d'abord il faut définir une gamme, on mettra par défaut la gamme tempérée à 12 notes. Ensuite l'idée est la suivante. On subdivise le tableau d'origine, pour chaque divisions, on applique la FFT et on repère la fréquence fondamentale du son. Si celle ci n'est pas égale à l'une des notes de la gamme définit précédemment, il faut la décaler pour la rendre « juste ». Pour se faire on décale notre fréquence fondamentale jusqu'à atteindre la fréquence de la note dont elle est la plus proche dans la gamme en utilisant la fonction shift_left ou shift_right. Notre implémentation fonctionne très bien pour des sons purs ou des enregistrements qui sont des sons purs successifs. Cependant elle fonctionne très mal pour les sons complexes car le repérage de la fréquence fondamentale est une tâche compliqué dont l'implémentation peut être améliorée.

Chapitre 5

Exercices de programmation

Nous avons précédemment présenté la création et l'utilisation de scripts Python permettant d'observer et de comprendre certains liens entre les mathématiques et la musique. Mais, durant notre stage, nous avons également créé des scripts dans l'objectif de nous améliorer en programmation.

5.1 Conception d'une interface graphique

Un de nos objectifs était de concevoir une interface graphique permettant d'utiliser la majorité des scripts créés. Ce programme principal se nomme *app.py*. Sa spécification est disponible dans le fichier *README.md* disponible dans notre dépôt github.

FIGURE 5.1 – Capture d'écran du programme principal

5.2 Entendre les nombres

L'objectif de ce programme est de nous permettre « d'entendre » les nombres. La première étape est de définir une gamme de N notes. Ensuite notre nombre, dépourvu de sa virgule, est converti en base N avant que ses chiffres ne soient joués à la suite dans la gamme définie précédemment. Par exemple si j'ai un nombre qui donne 1245A lorsqu'il est dépourvu de sa virgule, le programme joue à la suite la première note, puis la seconde, la quatrième, la cinquième note de la gamme et enfin la dixième note de la gamme. En effet lorsque le nombre de note de la gamme est supérieur à 9 on utilise l'alphabet pour représenter les notes suivantes. Une partie intéressante du script est la fonction baseTenToX qui comme son nom l'indique prend un nombre décimal en paramètre et le convertit en base X. Le script se nomme *entendreNombre.py* mais vous pouvez l'utiliser depuis le programme *app.py*.

5.3 Création d'un synthétiseur très basique

Pourquoi est ce qu'un La4 joué au piano semble différent d'un La4 joué à la guitare sachant que ces deux notes ont la même hauteur ? La différence entre ces deux notes réside dans une propriété du son : le timbre. Ce qui va principalement faire varier le timbre d'un son est la présence ou l'absence des harmoniques de la fréquence fondamentale, ainsi que l'intensité de ces dernières. Notre programme illustre cette idée. Grâce au module Tkinter, nous avons réalisé une interface graphique permettant à l'utilisateur de régler l'intensité des 10 premières harmoniques d'une note fondamentale choisie préalablement. Cela permet alors de comprendre une partie du fonctionnement des synthétiseurs. Le programme se nomme *harmonique.py*.

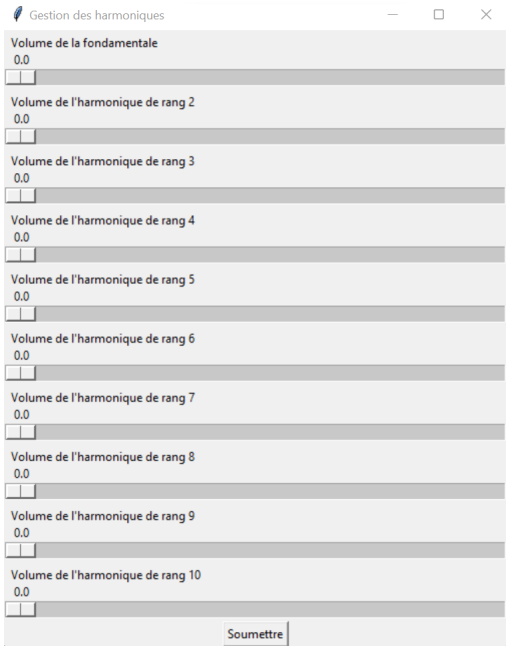

FIGURE 5.2 – Capture d'écran du programme permettant de gérer le volume des 10 premières harmoniques

Chapitre 6

Conclusion

Il est important de noter que ce rapport ne contient pas tout ce qui a pu être abordé durant ce stage. Nous tenons à remercier l'Institut Fourier pour son accueil.

6.1 Lucas Bolbènes

Ce stage m'a beaucoup appris, que ce soit sur le contenu scientifique mais également sur le monde de la recherche. J'ai conscience que nous avons seulement abordé une infime partie des sujets possibles avec ce thème et je serai ravi de pouvoir retravailler sur celui-ci un jour. La musique et l'informatique étant deux grandes passions il serait très intéressant de pouvoir mêler les deux en travaillant par exemple sur la conception de synthétiseurs, et j'ai conscience que les mathématiques ont un rôle majeur dans ce genre de projet. Actuellement je me dirige plus vers une carrière d'ingénieur que de chercheur. Mais j'ai conscience que le monde de la recherche existe et qu'il a une importance capitale. Je tiens à remercier l'Institut Fourier pour son accueil et en particulier M. Joly qui a su nous guider tout en nous laissant explorer nos propres pistes.

6.2 Brice Vittet

Ce stage fut vraiment très enrichissant dans la découverte du monde de la recherche. Nous avons côtoyé beaucoup d'enseignants chercheurs mais je ne savais pas vraiment ce qu'ils faisaient, là j'ai pu voir quelques brides. La tournure du stage fut différente de mes attentes. Je pensais qu'on allait avoir des instructions et des indications sur quelque chose à faire de précis, mais en fait, Romain Joly nous donnait seulement des pistes de réflexion a chercher, et c'était nous seuls qui devions les comprendre et les étudier à notre façon. Au début ce n'était pas facile et très fatiguant, puis au fil des semaines nous avons réussi à trouver un rythme qui nous convenait. Ce système de stage sans objectif final précis, et sans guide, est très intéressant car nous étions libres de partir là où on le souhaitait, mais aussi perturbant, car nous n'étions pas habitué à ce genre de travail. Pour finir, ce thème ma permis d'en apprendre plus sur Fourier, hors de ce que l'on avait vu en cours, et alors j'ai découvert la vrai utilité de son théorème et ses applications.