Dossier TIPE
Reconnaissance des notes de musique

Armand GONTHIER & François PECH

Abstract

L'écriture d'une partition de musique est une tâche manuelle qui se fait soit à la main, soit à la souris via un ordinateur. C'est une tâche qui demande beaucoup de concentration : il est en effet difficile pour un musicien de se souvenir de toutes les mélodies imaginées ou au préalable jouées, ce qui limite cette pratique seulement aux virtuoses.

Il serait ainsi intéressant d'obtenir les notes retranscrites sur une partition immédiatement après les avoir jouées laissant au musicien, même novice, la possibilité de se reprendre par la suite.

Mais est-il possible de passer d'une mélodie enregistrée à la partition exacte de ce qui a été joué ?

Durant l'étude, nous nous sommes concentrés sur la transcription de la flûte à bec, possédant des propriétés facilement exploitables. Pour réaliser ce projet, nous nous sommes principalement appuyés sur la décomposition de Fourier, un outil permettant de décomposer le signal en une somme de fréquences.

Le programme final est en mesure de suivre les notes jouées avec de rares erreurs. En optimisant davantage le programme, on peut envisager voir le nombre d'erreurs diminuer. Le résultat de l'étude étant concluant, il reste à l'adapter à un gamme d'instruments plus large.

The writing of a musique sheet is a manual task that is either done by hand, or with a mouse via a computer. It is a task that requires a lot of concentration : it is in fact hard for a musician to remember all of the melodies he imagined or he previously performed. This limits this practice only to the virtuosos.

It would be interesting to obtain the notes directly on a musique sheet after playing them. This would give the possibility, even to novice players, to correct himself afterwards.

But is it possible to go from a melody recorded to an exact musique sheet of what has been played?

During the study, we focused on the transcription of the flute since it possesses easily exploitable properties. In order to realise this project, we used the decomposition of Fourier, a tool that allows us to decompose the signal in a sum of frequencies.

The final programme is able to spot the notes that are being played with very few mistakes. By optimising the programme further, we are able to consider that the number of mistakes would decrease. The result of the study being decisive, we only need to adapt it to a wider range of instruments.

SOMMAIRE

I. Présentation du projet

Dans le cadre de notre TIPE, nous avons choisi d'étudier la reconnaissance des notes de musiques jouées à l'aide d'une flûte.

Nous nous sommes fixé pour objectif de permettre la reconnaissance en temps réel des notes jouées, puis de les afficher sur une partition, au bon moment avec la bonne longueur, de la même façon que si la partition avait été rédigée à la main. Cela doit donc prendre en compte les changements ainsi que les répétitions de notes, et ce pour toute flûte. Le projet se présentera sous la forme d'un programme Python composé d'une interface.

Nous avons pour cela décomposé notre travail en plusieurs études:

- Celle d'une seule note pré-enregistrée
- Celle de plusieurs notes pré-enregistrées, différentes ou non, et jouée avec un temps de pause entre chaque
- Celle de notes pré-enregistrées et jouées avec un temps de pause infime entre chaque
- Celle de notes jouées en temps réel

Notre principale motivation est de pouvoir ensuite construire un produit hardware, basé sur cet algorithme de reconnaissance musicale, afin de faciliter l'apprentissage de la musique et des morceaux d'artistes confirmés. Le prétexte du TIPE nous est ainsi apparu idéal pour concilier le travail de recherche avec nos études.

Au cours du projet, nous avons travaillé avec notre code Python, bien sûr, mais également avec le logiciel Audacity qui nous a permis d'obtenir des représentations précises des sons étudiées.

II. Analyse de Fourier

A. La Transformée de Fourier

Afin de pouvoir analyser un son, nous avons besoin d'en extraire la fréquence. Cette règle vaut également pour les notes de musique.
Pour cela, on utilise une décomposition du temps en fréquence qui s'appelle la transformée de Fourier.

Celle-ci s'appuie sur le théorème de Fourier qui s'énonce ainsi :

Toute fonction périodique, à valeur réelles, et ne présentant qu'un nombre fini de discontinuités (au sens de dénombrable) peut se décomposer en une série de Fourier de la forme

$$f(t) = a_0 + \sum_{n=1}^{\infty} a_n cos(n\omega t) + b_n sin(n\omega t) \quad \text{où } \omega = \frac{2\pi}{T}$$

et où $a_n = \frac{2}{T}\int_{t_0}^{t_0+T} f(t)cos(n\omega t)dt$, $b_n = \frac{2}{T}\int_{t_0}^{t_0+T} f(t)sin(n\omega t)dt$, $a_0 = \frac{1}{T}\int_{t_0}^{t_0+T} f(t)dt$

On préférera utiliser une formulation complexe de cette décomposition car beaucoup plus maniable dans les calculs :

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{in\omega t} \text{ avec } c_n = \frac{1}{T}\int_{t_0}^{t_0+T} f(t)\, e^{-in\omega t}$$

Cette formule nous permet donc de transformer n'importe quel signal T-périodique en une somme de sinus et de cosinus et d'en extraire ensuite un spectre de fréquences. Toutefois, la Transformée de Fourier se base sur l'hypothèse que le son dure depuis la nuit des temps et qu'il sera constant à l'avenir.

B. La Transformée de Fourier Discrète et FFT

Pour effectuer la même transformation sur un signal fini et non périodique, on utilise son équivalent discret, la **Transformée de Fourier Discrète**. Il en existe de nombreuses implémentations numériques, la plus célèbre, et celle que nous utiliserons, s'appelle la FFT (pour **F**ast **F**ourier **T**ransformation)

La FFT est un algorithme fondamental d'analyse de données numériques et de traitement du signal. Il permet le calcul rapide de la Transformée de Fourier Discrète (abrégée TFD) dont le principal défaut est justement sa lenteur : reprenons la formule de la transformée de Fourier pour un signal fini de longueur N. On limite le calcul des n de la TFD à N valeurs :

$$f(t) = \sum_{n=0}^{N-1} c_n \, e^{i\omega nt} \text{ où } \omega = \frac{2\pi}{N}$$

Pour chaque valeur de t, le calcul de f implique N multiplications ainsi que N -1 additions. Ainsi, pour calculer les N valeurs de la TFD, il faudra effectuer N² multiplication et N² - N additions. Par exemple, pour N = 10^3 , le nombre d'opération nécessaires sera de 2N² - N = $2 \times 10^6 - 10^3 = 1\,999\,000$ opérations.

C'est un calcul très inefficace puisqu'il n'utilise pas les propriétés de symétrie et de périodicité de l'exponentielle complexe :

Symétrie : $e^{i\omega(k+\frac{T}{2})} = -e^{i\omega k}$

Périodicité : $e^{i\omega(k+T)} = e^{i\omega k}$

L'algorithme de la FFT prend lui en compte ces propriétés et se base sur l'approche du "Diviser pour régner".

Si l'on sépare notre somme en somme d'indice pair et d'indice impair, il vient :

$$\sum_{n=0}^{N-1} c_n \, e^{i\omega nt} = \sum_{r=0}^{\frac{N}{2}-1} c_{2r} e^{i\omega 2rt} + \sum_{r=0}^{\frac{N}{2}-1} c_{2r+1} e^{i\omega(2r+1)t}$$

$$= \sum_{r=0}^{\frac{N}{2}-1} c_{2r} e^{i\omega 2rt} + e^{i\omega t} \sum_{r=0}^{\frac{N}{2}-1} c_{2r+1} e^{i\omega 2rt}$$

On remarque que $e^{i\omega 2rt} = e^{i\omega \frac{2\pi}{N} 2rt} = e^{i\omega \frac{2\pi}{\frac{N}{2}} rt}$

Ainsi

$$\sum_{n=0}^{N-1} c_n e^{i\omega nt} = \sum_{r=0}^{\frac{N}{2}-1} c_{2r} e^{i\omega \frac{2\pi}{\frac{N}{2}} rt} + e^{i\omega t} \sum_{r=0}^{\frac{N}{2}-1} c_{2r+1} e^{i\omega \frac{2\pi}{\frac{N}{2}} rt}$$

Notre TFD initiale peut s'écrire sous la forme de deux TFD de plus petite longueur.

L'intérêt est que cela réduit sensiblement le nombre d'opérations nécessaires au calcul de la TFD : pour chaque t, il y aura $\frac{N}{2}$ - 1 + $\frac{N}{2}$ - 1 + 1 = N - 1 opérations, donc pour un signal de longueur N, il y aura N² - N.

Ainsi pour N = 10^3, il y aura $10^6 - 10^3 = 999\,000$ opérations. De plus, avec l'hypothèse que N est une puissance de 2, on peut répéter le processus jusqu'à obtenir une TFD de base 2.

C. Conditions d'applicabilité de l'algorithme de la FFT

Plusieurs règles doivent être respectées pour que la transformée de Fourier puisse fonctionner correctement :

1) Le nombre d'échantillons de données en entrée doit être égal au nombre de fréquence en sortie.

2) La fréquence maximale pouvant être représentée est la fréquence de Nyquist. Elle vaut, selon le théorème de Nyquist-Shannon, la moitié de la fréquence d'échantillonnage. En pratique, puisque les fréquences audibles par l'oreille humaine sont comprises en moyenne entre 20 et 20 000 Hz, on utilisera une fréquence d'échantillonnage de 44.100Hz avec une certaine marge pour que les fréquences ne se coupent pas et permettre ainsi que le son soit plus naturel.

   Dans ce cas, la fréquence de Nyquist vaut 22 050 Hz.

3) Lors de la Transformée de Fourier Discrète, l'ordre des fréquences est comme suit : 0Hz - Fréquence Positives - Fréquence de Nyquist - Fréquences Négatives

4) Enfin, on sait, d'après le théorème de Parseval, que l'énergie totale est égale à la somme des contributions des différents harmoniques du signal. L'égalité qui en découle est la suivante :

$$\sum_{n=-\infty}^{\infty} \left|c_n\right|^2 = \frac{1}{T}\int_0^T |f(t)|^2\, dt$$

Avec $\sum_{-\infty}^{\infty} \left|c_n\right|^2 = \left|a_0\right|^2 + \frac{1}{2}\left(\sum_{1}^{\infty} |a_k(t)|^2 + |b_k(t)|^2\right)$

Ainsi, pour un signal donné, les données récupérées après transformée de Fourier doivent être ajustées au nombre d'échantillons pour que le niveau d'énergie soit équivalent

D. Transformée de Fourier à court terme

En pratique, sur des sons pré-enregistré, nous devrons étudier des intervalles de temps très court sur l'ensemble du signal afin de déterminer si oui ou non une note est jouée à tel instant, quelle note…

Pour cela, on utilise la **Transformée de Fourier à court terme** (TFCT), où le principe est d'appliquer la FFT mais sur un segment de temps très court (de l'ordre de 10^{-1} seconde) et faire "glisser" ce segment sur tout le signal. Le problème qui se pose en segmentant ainsi notre signal est que ses bords ont une transition "perçante" qui n'existait avant. Cette transition n'apparaît pas sur les représentation numérique du signal comme sur Audacity.

Il n'existe pas d'application de TFCT satisfaisantes dans les bibliothèques standards de Python. Nous avons essayé nous même des implémentations plus ou moins efficaces de cet algorithme avant d'obtenir un algorithme satisfaisant sur le site de Kevin Nelson.

Celui-ci fonctionne de la manière suivante :
D'abord, il faut sélectionner le segment de signal à étudier. On utilisera une fenêtre glissante pour visualiser quelle portion du signal est étudié. La taille de la fenêtre ne variera pas au cours du traitement du signal, seul son point de départ varie.

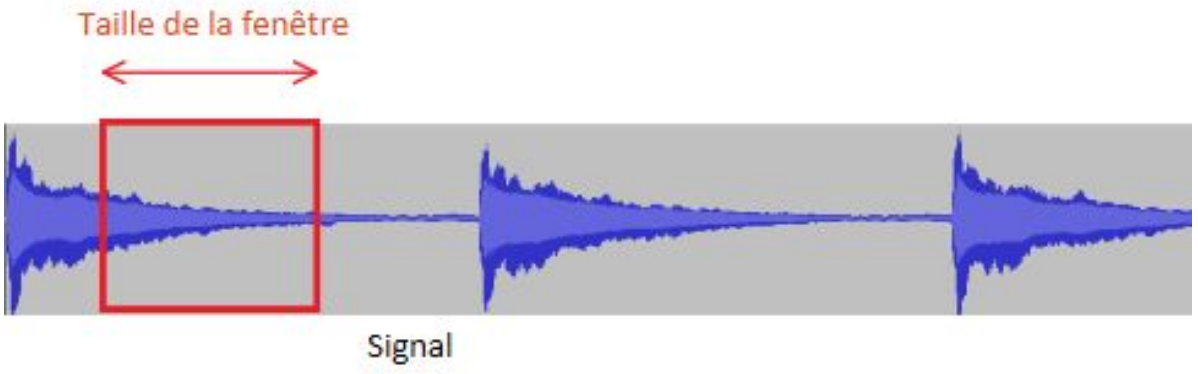
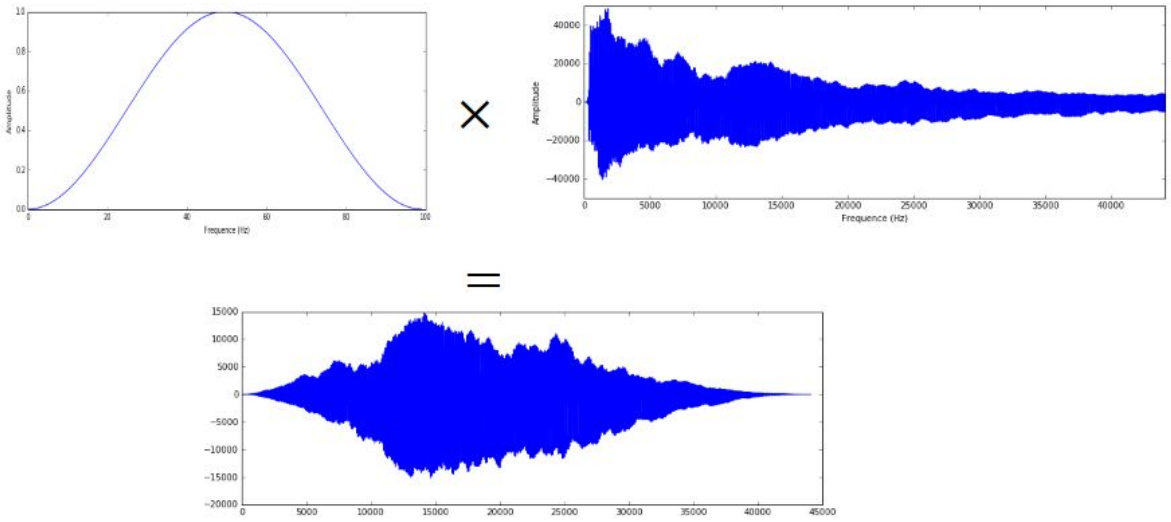


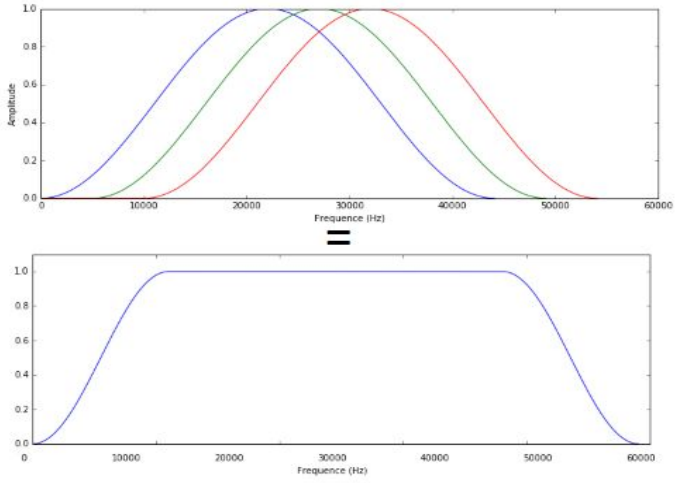
Illustration du concept de segment glissant

Nous avons évoqué un effet de bords “perçants” lors d’une telle segmentation du signal. Pour y remédier, nous multiplions notre segment par un demi-cosinus (ou fenêtre de Hann) pour permettre au signal segmenté d’être fondu en entrée et en sortie. Ainsi, les bords seront atténués, l’effet disparaîtra et les transitions n'affectent plus la transformée de Fourier du signal.



Multiplication du signal par une fonction de Hann

Il y a un autre avantage à l’utilisation d’une fenêtre de Hann : si la fenêtre chevauche la dernière d’au moins la moitié, la somme des fenêtres aura une amplitude inchangée. Multiplier notre signal par cette fonction ne modifiera donc pas les données pour une éventuelle représentation.



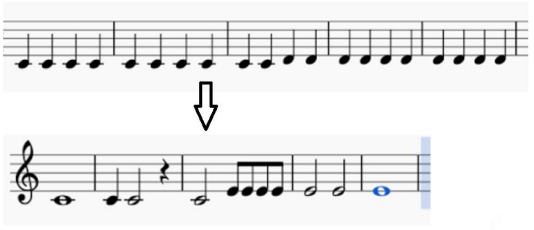
Superposition de cosinus

Et enfin, on peut appliquer la FFT sur le signal fenêtré et en extraire la fréquence.

III. Caractéristique du son

Nous avons pour but de détecter les notes jouées par l'utilisateur. Nous avons vu au préalable que cela se traduisait par la détection de la fréquence la plus présente dans le son joué, grâce à la transformée de Fourier.

Cependant, une partition de musique est plus complexe qu'une suite de note à jouer. Elle est composée de notes dont on doit préciser la longueur mais aussi de silence.



Il est donc important de comprendre les différentes caractéristiques du son, de sa création jusqu'à l'arrivé dans un micro ou dans nos oreilles.

A. Le son

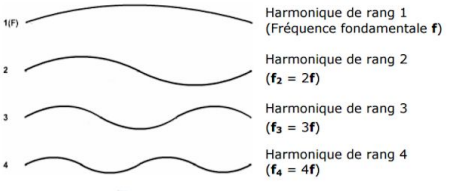
Un son est défini selon plusieurs caractéristiques:

- **Son intensité**, appelée aussi volume. Elle correspond à l'amplitude de l'onde, cette amplitude est donnée par l'écart maximal de la grandeur caractérisant l'onde. Dans notre cas, le son est une onde de compression, cette grandeur est la pression. l'intensité du son se mesure en décibels (dB).

- **Sa hauteur tonale ou fréquence** qui dépend de la vitesse à laquelle l'objet vibre. On exprime alors en hertz le nombre de vibrations par seconde. Plus le nombre de vibrations est grand, plus le son sera aigu. Certaine de ces fréquences correspondent à des notes, par exemple le LA3 à 440Hz.

- Le rapport de fréquence entre deux notes détermine l'intervalle musicale entre elles. L'échelle occidentale divise les fréquences audibles en **octaves** elle même divisé en 12 sous-intervalles égaux, dont chacun correspond donc à un rapport de $2^{1/12}$. on appelle cet intervalle un demi-ton

- Un son est rarement constitué d'une seule fréquence, mais plus souvent d'un ensemble de fréquence plus ou moins présente donnant le **timbre** de l'instrument. Lorsque l'on joue un la3 au piano le son est constitué de sa fondamentale à 440Hz mais aussi d'harmoniques: l'ensemble des fréquences multiples de la fondamentale.



Ainsi 2 instruments jouant la même note auront la même fondamentale mais 2 timbres différents.

Cas particulier:

- **La vibration transversale** d'une corde est donnée par l'équation de d'Alembert:

$$\frac{\partial y^2}{\partial x^2} = \frac{\mu}{T_0}\frac{\partial^2 y}{\partial t^2}$$

avec y l'élongation de la corde, μ la masse linéique de la corde, l'abscisse x et le temps t.

Au final y suit cette équation dans le cas général.

$$\forall x,t: y_n(x,t) = \sin(\mathrm{k_{nx}})\left(\gamma_n \cos(\omega_n t) + \frac{L}{n\pi c}\delta_n \sin(\omega_n t)\right)$$

$$avec \quad k_n = \frac{n\pi}{L} \quad et \quad \omega_n = \frac{n\pi c}{L}$$

Cependant dans le cas des instruments à cordes pincées, les coefficient de Fourier valent $b_n = 0$ et $a_n = -\frac{2L^2h}{bn^2\pi^2(b-L)}sin(\frac{bn\pi}{L})$

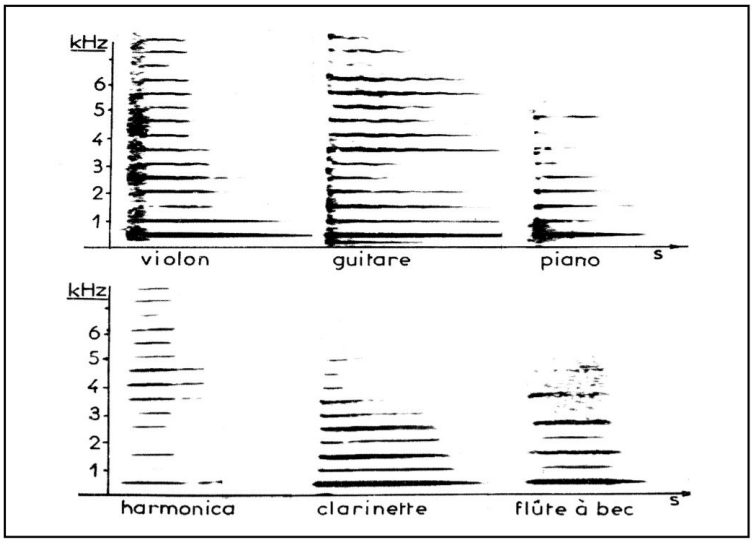
Et à l'inverse, pour les instruments à cordes frappées $a_n = 0$ et $b_n = \frac{4uL}{n^2\pi^2c}sin(\frac{n\pi}{2L}(2a+e))sin(\frac{n\pi e}{2L})$.

Il existe aussi, dans le cas d'une corde en oscillation entretenues telle que le violon, des $\alpha \, et \, \beta$ sont complexes à calculer et approximatif

- **L’inharmonicité.** Il arrive parfois lorsque l’on trace le spectre d’un signal que l’on constate que les raies ne sont pas équidistantes, il n’y a pas d’harmonie. Cela est visible sur un son de piano à cause de ses cordes en acier. On modifie alors l’équation de d’Alembert en lui ajoutant un terme

$$\mu \frac{\partial^2 y}{\partial t^2} - T_0 \frac{\partial^2 y}{\partial x^2} - ESK^2 \frac{\partial^4 y}{\partial x^4}$$

- **Les transitoires** sont la variations de l’amplitude ainsi que son spectre au cours du temps. Sans celle-ci il y aurait une moins grande diversité d’instruments. Expérimentalement si on répète en boucle un petit groupe de période d’un son de piano, le son sera alors très similaire au son d’une flûte



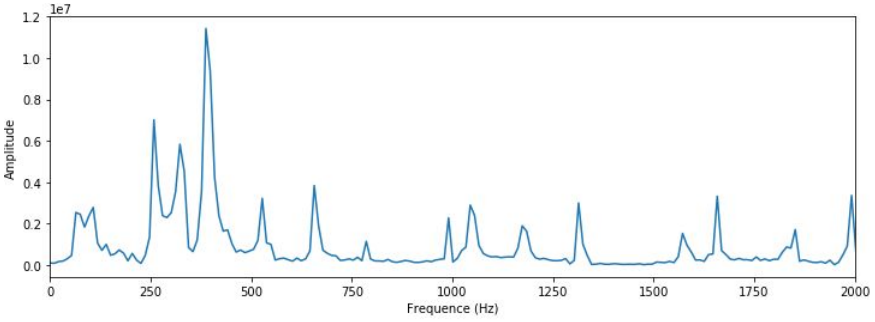
Sonagrammes de quelques instruments

.

Afin de retranscrire toutes les altérations contenue dans le son analysé, il nous est nécessaire de s’appuyer sur des caractéristiques du son, or il est difficile de trouver une méthode pour pouvant supporter tous les instruments à la fois.
Nous avons donc choisis comme première approche de nous concentrer sur la flûte à bec.

B. La flûte à bec

Nous avons choisis la flûte à bec car son étude est assez facile.
Tout d'abord il est possible de jouer une seul note à la fois, ainsi il n'est pas nécessaire de s'intéresser aux accords(augmentant la complexité du code). Un accord est l'action de jouer plusieurs notes au même instant, le son est alors constitué d'un assemblage de plusieurs notes jouées. Il est difficile de trouver les fréquences fondamental de chacune d'elle grâce à son spectre.

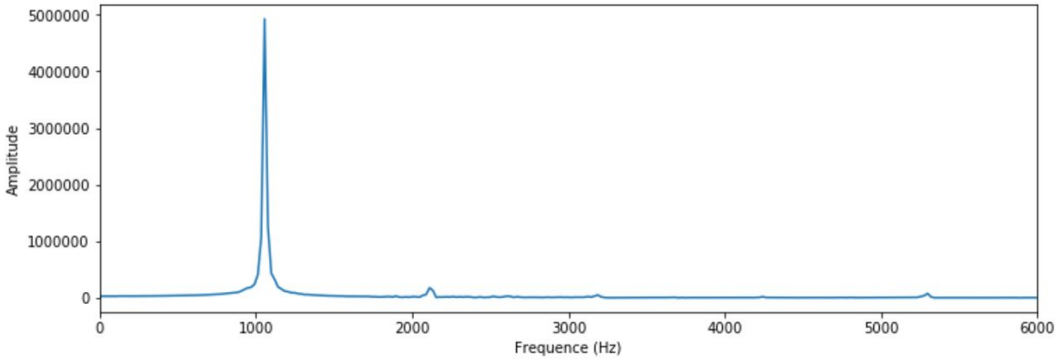


Spectre d'un accord do majeur (do mi sol) joué au piano

En plus de ne pouvoir jouer qu'une seule note à la fois, à la flûte, on est obligé de marquer un silence entre deux notes, aussi petit qu'il soit.

De plus, l'instrument est accordé par rapport au LA3 à 440Hz et ne nécessite pas d'être accordé par l'utilisateur comme une guitare ou un piano. Ainsi il n'y a pas besoin d'étalonner notre programme, puisque les fréquences restent les même d'une flûte à l'autre.

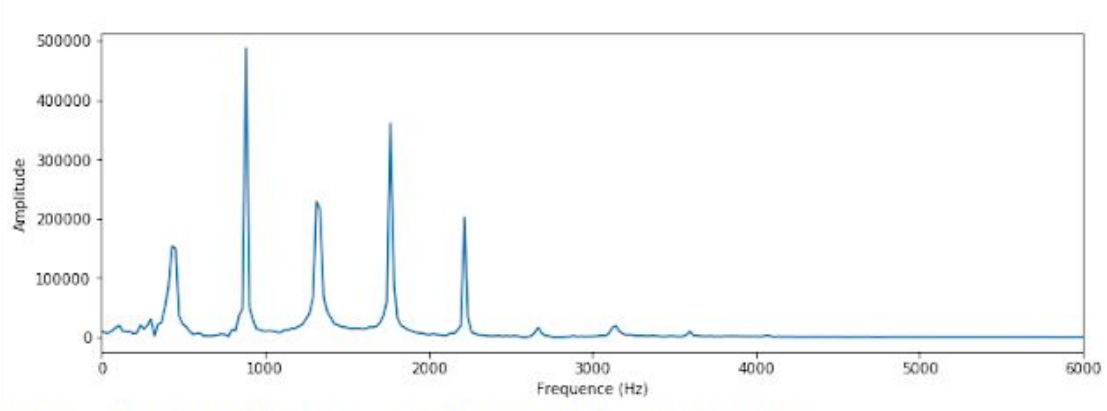
Enfin le son de la flûte est simple et proche d'un son pur: un son composé seulement d'une seule fréquence.



Spectre d'un DO5 à la flute à bec

Comme on peut le voir sur le spectre d'un do5 joué a la flute, les amplitudes des harmoniques sont très faible par rapport à la fondamentale. Il est donc facile de récupérer la fondamentale correspondant à la fréquence avec l'amplitude maximale.

On peut comparer ce résultat au spectre d'un la3 au piano



On observe qu'ici l'amplitude de la 2ième harmonique et de la 4 ieme ont une plus grande amplitude que la fondamentale, il faudrait donc suivre un ensemble de test sur le spectre pour que le programme puisse donner la bonne note correspondant à la fondamentale. (En partant de l'hypothèse qu'une seul note est jouée)

IV. Réalisation

A. La théorie

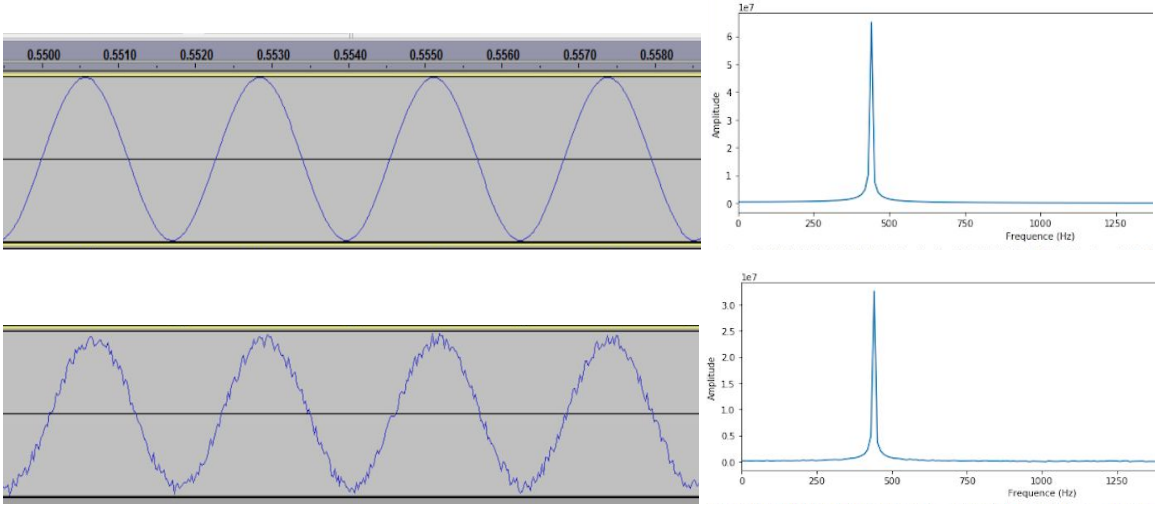
1. Acquisition du son:

Pour acquérir notre son, nous avons besoin d'un micro capable de capter l'ensemble des fréquences audibles.

La précision du microphones nécessaire pour notre programme étant impossible à déterminer, nous avons choisi de faire une approche expérimentale nous permettant dans un premier temps de tester notre analyse de Fourier sur un signal créer informatiquement et composé d'une seule sinusoïde.

Nous avons ensuite comparé ce résultat avec la même sinusoïde à laquelle on a ajouté du bruit. Le bruit qui s'ajoute à la sinusoïde est un bruit gaussien, c'est-à-dire un bruit qui ne privilégie aucune fréquence. Les imprécisions du micros sont similaire à un bruit gaussien..

Exemple d'une sinusoïde à 440Hz avec son spectre et d'une sinusoïde à 440Hz par dessus laquelle on a rajouté du bruit gaussien:



On remarque donc que le bruit n'affecte pas l'information qui nous intéresse. On retrouve toujours une forte amplitude pour une fréquence de 440hz.

2. Transformée de Fourier

Nous allons découper le son enregistré en plusieurs fenêtres de même longueur. Le but est d'obtenir, grâce à la FFT, le spectre de chacune de ces fenêtres et de relever la fréquence avec la plus grande amplitude. Cette fréquence correspond à la note jouée à cet instant. Ainsi sur chaque fenêtre on aura la note la plus présente avec la valeurs de son amplitude qui nous servira pour la 3ieme partie du code.

3. Interprétation

Nous avons vu que l'écriture d'une partition de musique ne se réduisait pas seulement à écrire la note jouée à chaque instant t, mais d'indiquer également les silences et la longueur de chaque note. Il faut donc interpréter les résultats de la 2ieme partie du code. Pour cela nous nous appuierons sur l'amplitude de ces résultats.

L'interprétation va donc comparer les notes et les amplitudes des fenêtres qui se suivent. Différents cas seront à considérer :

- Déterminer si la note contenue dans la fenêtre est un résultat cohérent avec l'instrument. Pour cela on regardera si la fréquence est comprise dans l'intervalle 250Hz-2000Hz et si l'amplitude est suffisamment grande.
- Longueur de la note. Si plusieurs fenêtres à la suite comportent la même note, traduire cela par une fenêtre plus grande avec cette note.
- Changement de note. Il suffit de regarder entre 2 fenêtre si la note change
- Répétition de note. Plus difficile à mettre en place, la fréquence restant la même. Nous savons qu'il est impossible de jouer 2 fois la même note à la flûte sans laisser un silence même infime entre les 2 sons. Il est donc possible de détecter cette répétition avec l'apparition de fenêtre ayant une faible amplitude comparé à celle précédente.

En définitive, on aura seulement des fenêtres, plus ou moins grande, avec la note correspondante

4. Ecriture de la partition:

L'écriture d'une partition étant pourvue de nombreuse règle et de style d'écriture, nous utiliserons un logiciel déjà existant: Lilypond, rendant l'écriture automatique. Il suffit de donner la liste des notes avec leurs durée les unes après les autres, ainsi que le tempo auquel on joue.

B.Algorithme - Explication du Code

Nous avons codé notre programme en Python et avons utilisé la bibliothèque PyQt5 pour l'interface graphique. Cette dernière présente l'avantage non négligeable d'être incluse avec un éditeur WYSIWYG (What You See Is What You Get) dans les packages standards de Python ce qui simplifie considérablement le codage de l'UI, allège notre code et nous permet de nous concentrer sur les fonctions essentielles de notre programme.

Au 11 Mai 2017, notre programme est composé de deux parties : un créateur de son analogique et un analyseur de son pré-enregistré. Le code n'affiche pas encore de lui-même le recomposition de notre son sur une partition,nous utilisons pour le moment un logiciel, Lilypod, qui se charge de faire cela pour nous. Notre son est précisément analysé, voire même trop, nous avons donc beaucoup de données à traduire c'est pourquoi, avant de penser à les afficher, nous nous concentrons d'abord sur cette partie traduction.

1. Création de Son

Les fichiers créés seront au format .wav pour des raisons de compressibilité. Les format tels que le .mp4 compressent et modifient les signaux pouvant fausser les analyses de Fourier. Pour visualiser et écouter nos signaux, nous pouvons utiliser le logiciel Audacity.

A l'aide de cet onglet, nous pouvons créer des signaux composés d'une à cinq fréquences ou alors un signal composé uniquement de bruit. Pour pouvoir travailler avec des fichiers audio, les lire, modifier et ré-écrire dessus, nous utiliserons le module wave, inclus dans les packages standards de Python car simple d'utilisation.

La création d'un bruit est plutôt basique : on sélectionne une valeur entre -1 et 1 au hasard pour chaque point de notre signal et on l'écrit sur le fichier. En audio-numérique, les amplitudes varient entre 32767 (pour 1) et -32767 (pour -1)

```
for i in range(int(self.dureeText.text()) * int( self.frequenceEchText.text() ) ):
          value = random.uniform(-32767.0, 32767)
          packed_value = struct.pack('<h', value)
          wavef.writeframes(packed_value)
```

dureeText correspond à la durée du signal souhaitée par l'utilisateur et frequenceEchText la fréquence d'échantillonnage. wavef correspond à notre signal en écriture.

La création d'un signal sinusoïdale est à peine plus technique : plutôt que de travailler avec une valeur aléatoire, on écrira une valeur calculée à l'aide des différentes fréquences renseignées par l'utilisateur.

Il y a donc deux boucles for : l'une pour parcourir chaque puis du signal (durée * nombre de point par seconde), la deuxième pour parcourir une liste contenant les fréquences renseignées par l'utilisateur.

On sait que les signaux sont sinusoïdaux, on utilise la fonction sin du module math pour calculer la valeur en chaque point.

Le seul petit point technique ici est de ne pas oublier, après avoir parcouru tous nos sin et les avoir sommé, de diviser la valeur à incorporer par le nombre de fréquence : avant d'incorporer le résultat en un point, nous le multiplions par 32767 (car c'est ainsi qu'est codée l'amplitude). Or en sommant les sinus des différentes fréquences, il est possible que la somme soit supérieure à 1. En essayant d'incorporer une valeur supérieure à 32767 à notre signal, le programme retournera vraisemblablement une erreur.

```
for i in range(int(self.dureeText.text()) * int(self.frequenceEchText.text())):
              courbeAuPointi = 0
              for _ in range(0, self.nbFrequence):
                     courbeAuPointi += math.sin( 2*int(
self.tableauFrequenceText[_].text())*math.pi*float(i)/float(self.frequenceEchText.text()
))
              courbeAuPointi = int(32767.0 / self.nbFrequence * courbeAuPointi)
              wavef.writeframesraw( struct.pack('<h', courbeAuPointi ) )
```

Code de la fonction :

```
def creationSon(self):

       wavef = wave.open('{}.wav'.format(self.nomSonText.text()),'w') #Nouveau fichier
(Création / écrasement)
       wavef.setnchannels(1) # Son mono-canal
       wavef.setsampwidth(2) #La taille d'un échantillon en bytes
       wavef.setframerate(int(self.frequenceEchText.text())) #Le nombre de fenêtre

       if self.bruit.isChecked() :
              for i in range(int(self.dureeText.text()) * int(
```

```
self.frequenceEchText.text() ) ):
                value = random.uniform(-32767.0, 32767)
                packed_value = struct.pack('<h', value)
                wavef.writeframes(packed_value)
        else:
                for i in range(int(self.dureeText.text()) *
int(self.frequenceEchText.text())): #range le nombre d'échantillons totaux
                courbeAuPointi = 0
                for _ in range(0, self.nbFrequence):
                        courbeAuPointi +=
math.sin(2*int(self.tableauFrequenceText[_].text())*math.pi*float(i)/float(self.frequenc
eEchText.text()))
                courbeAuPointi = int(32767.0 / self.nbFrequence * courbeAuPointi)
                wavef.writeframesraw( struct.pack('<h', courbeAuPointi ) )

        wavef.writeframes(str.encode('')) #Convertion en bytes
        wavef.close()
```

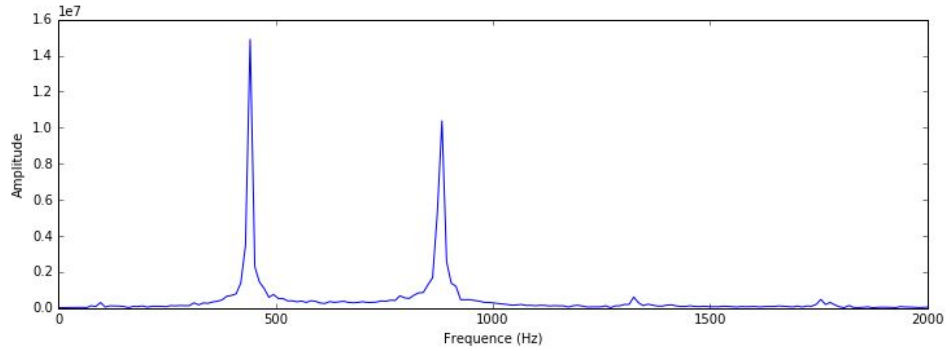
2. Analyse de son pré-enregistré :

Comme expliqué dans le I) D), nous avons deux algorithmes pour analyser les signaux dans la durée à l'aide de la Transformée de Fourier à court terme : l'un est celui que nous avons tenté d'implémenter car nous ne savions également pas qu'un tel algorithme, qui étudie le signal en totalité, avait un nom et existait. L'autre en grande partie inspiré du travail de Kevin Nelson, un blogger indépendant, et offrant des résultats tout à fait satisfaisant.

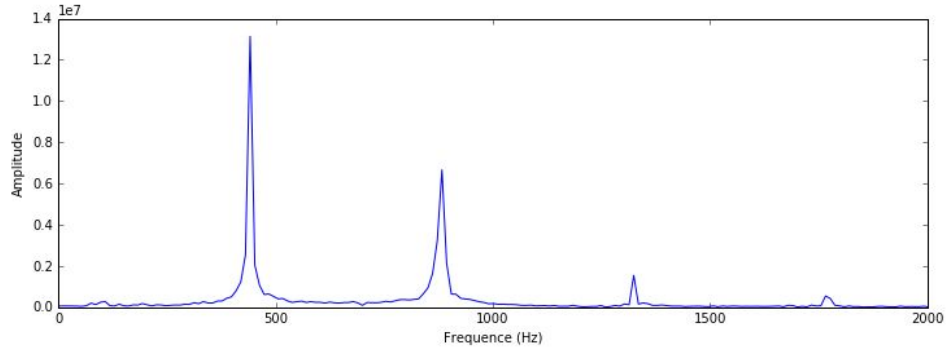
Nous allons expliquer les deux algorithmes.

Première version :

L'idée était d'appliquer la FFT sur des fenêtres qui glissaient dans le temps et de traduire les fréquences obtenues. Cependant, la taille des fenêtres étaient exactement celle de la fréquence d'échantillonnage (44.100 points dans la majorité de nos tests) Nous pouvions ainsi analyser un silence avec seulement les 10 premiers points d'une nouvelle note, l'analyse de Fourier nous donnerait pour fréquence celle de la note jouée car l'amplitude du spectre y serait plus importante. Si nous utilisions une autre fréquence, les fenêtres étaient certes réduites, mais le son analysé moins riche et pouvait offrir des absurdités. Nous avions également pensé à utiliser une variable correction, qui ignorait les sons en dessous d'une certaine amplitude mais le problème restait le même.



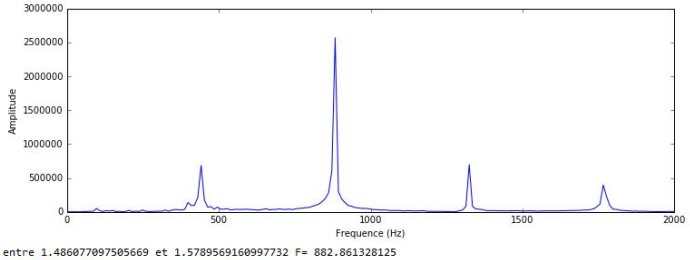

```
entre 0.37151927437641724 et 0.46439909297052157 F= 441.430664063
Note MIDI : A4
Note : LA4
```



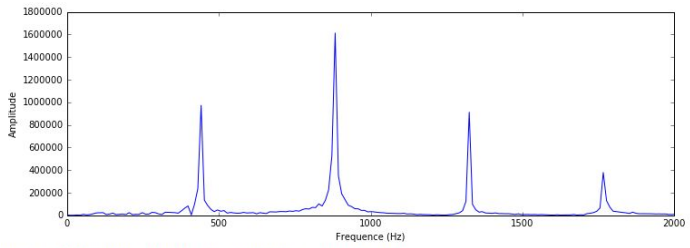

```
entre 0.46439909297052157 et 0.5572789115646258 F= 441.430664063
Note MIDI : A4
Note : LA4
```

Ci-dessus un exemple de ce que nous obtenions après analyse avec la version Bêta de l'algorithme. La note jouée est un La4 au piano, il nous restitue la bonne fréquence lorsque l'amplitude est haute. A noter que les différentes fenêtres ne se superposaient pas.

A plus basse amplitude, lorsque le son du piano allait decrescendo, la fréquence "majoritaire" était un octave plus élevée. Ceci peut en effet se remarquer à l'oreille, le son ayant tendance à devenir plus grave lorsqu'il est de faible amplitude.




```
entre 1.486077097505669 et 1.5789569160997732 F= 882.861328125
Note MIDI : A5
Note : LA5
```




```
entre 1.5789569160997732 et 1.6718367346938776 F= 882.861328125
Note MIDI : A5
Note : LA5
```

Enfin, le code écrit pour cette fonction :

```
for i in range(int(len(signal)/fenetre_FFT)):#Déplacement de la fenêtre FFT
        #Calcul de la TFD
        s = signal[fenetre_FFT*i:fenetre_FFT*(i+1)]
        signal_FFT = abs(fft(s)) #On ne récupère les composantes réelles

        #Récupération du domaine fréquentiel en passant la période d'échantillonnage
        Freq_FFT = fftfreq(s.size, dt)

        #Extraction des valeurs réelles de la FFT et du domaine fréquentiel
        signal_FFT = signal_FFT[0:len(signal_FFT)//2]
        Freq_FFT = Freq_FFT[0:len(Freq_FFT)//2]

        note = frequence_Midi(Freq_FFT[np.argmax(signal_FFT)])

        #Affichage du spectre du signal
        plt.figure(figsize=(12,4))
        plt.xlabel('Frequence (Hz)'); plt.ylabel('Amplitude')
        Freq_min = 0
        Freq_max = 2000
        plt.xlim(Freq_min,Freq_max)
        plt.plot(Freq_FFT,signal_FFT)
        plt.show()
```

Ici signal_FFT et Freq_FFT sont divisés pour deux car au delà de leur moitié, les valeurs sont respectivement complexes et négatives. Puisqu'on ne s'intéresse qu'aux valeurs réelles, on les ignores.

La fonction fréquence_Midi permet la conversion de la fréquence en une note MIDI. Les notes MIDI sont les notations musicales anglo saxonnes, plus pratiques à traiter dans un premier temps car la longueur de leur écriture ne varie pas d'une note à l'autre, tandis qu'un Do a un caractère de moins qu'un Sol par exemple, un Si commence par un 'S' tout comme Sol… Il est plus simple de travailler avec du MIDI, qui s'écrit A, B, … jusqu'à F#.

Version Satisfaisante :

Cette version de l'algorithme reprends celui expliqué en I) D). On ajoute une étape à cet algorithme initial, celle d'agrandir artificiellement le signal avec des zeros à l'aide du padded. Le dernier segment de notre signal ne peut dépasser la fin des données de plus d'une fois la taille de la fenêtre d'étude.

```
rate, self.signal = read(self.fileNameText.text())
        if len(self.signal.shape) != 1:
                self.signal = np.array(self.signal.sum(axis=1), dtype='float64')

        rééchantillonage = 10
        freqenceEchantillonage = eval(self.FFTsizeText.text())

        self.signal = sig.decimate(self.signal, rééchantillonage, ftype='fir')
        rate = rate // rééchantillonage
        dt = 1./rate
```

```
        taille_fenêtre = 1000
        taille_decalage = np.int32(np.floor(taille_fenêtre *
(1-float(self.overlapText.text())))
        total_segments = np.int32(np.ceil(len(self.signal) /
np.float32(taille_decalage)))
        t_max = len(self.signal) / np.float32(rate)
        fenetreHanning = np.hanning(taille_fenêtre) * float(self.overlapText.text()) * 2
        inner_pad = np.zeros((freqenceEchantillonage * 2) - taille_fenêtre)

        proc = np.concatenate((self.signal, np.zeros(freqenceEchantillonage)))
        result = np.empty((total_segments, freqenceEchantillonage), dtype=np.float32)

        for i in range(total_segments):
                décalage = taille_decalage * i
                t = (décalage + taille_fenêtre)/rate
                segment = proc[décalage:décalage + taille_fenêtre]
                windowed = segment * fenetreHanning
                padded = np.append(windowed, inner_pad)
                spectrum = np.fft.fft(padded) / freqenceEchantillonage
                autopower = np.abs(spectrum * np.conj(spectrum))
                result[i] = autopower[:freqenceEchantillonage]
                freqSpectrum = np.fft.fftfreq(spectrum.size, dt)

                note = self.frequence_Midi( abs(freqSpectrum[np.argmax(abs(spectrum))]))
```

Expliquons l'utilité de chaque variable :

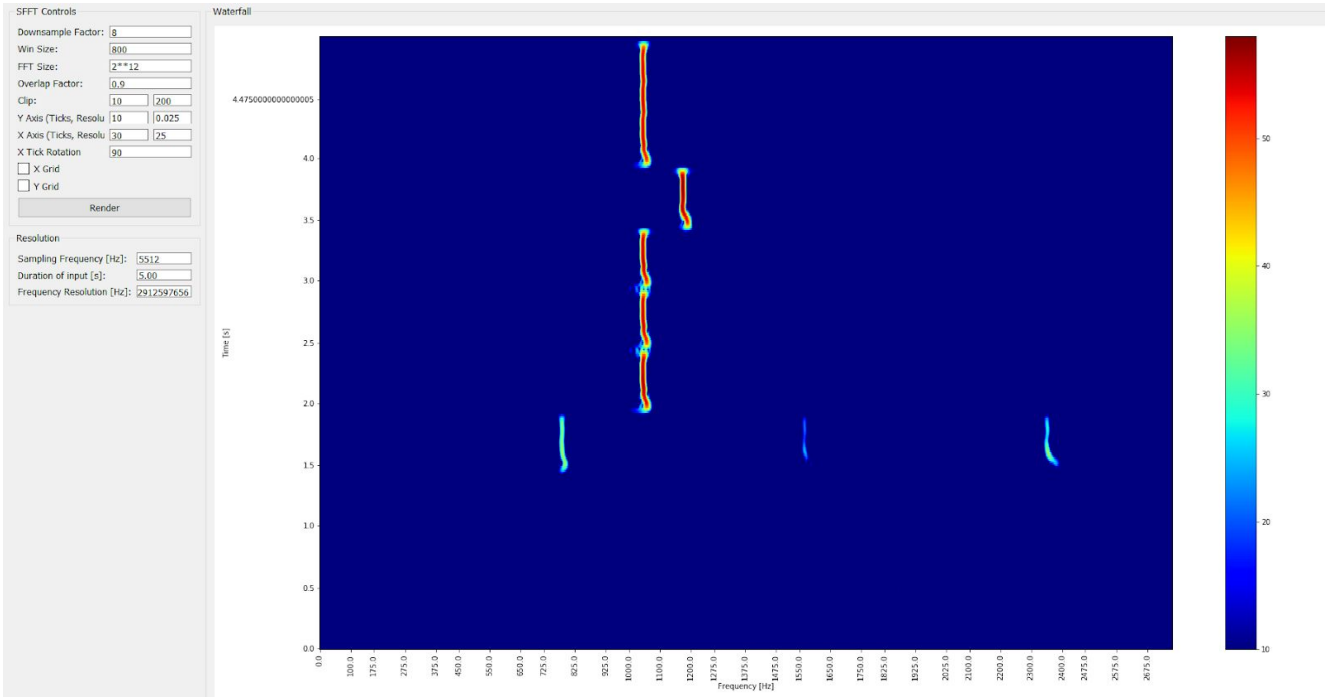
- rate : la fréquence à laquelle notre signal étudié est échantillonné
- signal : les données du sons étudié. Si il est Stéréo, on le convertit en mono dans le if
- rééchantillone : le facteur de rééchantillonage, on rééchantillone notre signal pour ne le réduire qu'aux fréquences qui nous intéresses dans le cas de la flûte afin d'augmenter la vitesse de calcul (car moins de points). Généralement, si on part de l'hypothèse que les sons sont échantillonnés à 44.100 Hz, les fréquences possibles jouée par une flûtes son celle de la 4ème et 5ème octave, donc de 250Hz à 2000Hz en comptant les harmoniques. On peut ainsi rééchantillonné notre son à 4410Hz, il sera certes moins riches en fréquences parasites, mais il gardera toujours la bonne fréquence.
- taille_decalage : il s'agit du nombre de points que l'on déplacera la fenêtre de Hanning. On incrémente i fois à chaque tour de boucle
- taille_fenêtre : Nombre de points auxquels on applique la FFT
- t_max : Longueur du signal étudié
- total_segments : Nombre de segments (=fenêtres) nécessaires pour parcourir le son en intégralité. Ne sert que pour la boucle for.

- window : Il s'agit de la fenêtre de Hanning. Il est de la taille de la fenêtre d'étude puisque qu'on étudie des segment de cette taille.

La liste autopower, elle, nous permet par la suite d'afficher notre signal à l'aide d'un spectrogramme, bien plus pratique à lire. Il ne contient que des valeurs réelles, qui l'on peut ensuite convertir en décibels, bien plus représentatifs pour une personne lambda qu'une amplitude d'ordre 10^7 .

Puisque la liste ne contient pas de valeurs complexes, elle est divisée par deux par rapport à la taille du spectrum qui lui, contient les résultats de la TFD sur notre signal fenêtré. (Réels donc, et complexes) C'est pourquoi, pour garder toujours le même nombre d'échantillon de fréquence, nous avons doublé artificiellement la taille du segment.

Cet algorithme permet par la suite de représenter notre son en son intégralité, et non plus seulement par petit bout, à l'aide d'un spectrogramme. Plus le décibel pour une fréquence est élevé, plus l'ordonnée au cours du temps est rouge :



On a ici représenté l'évolution au cours du temps d'un morceau de Papa Noël joué à la flûte (Sol - Do - Do - Do - Ré- Do)

En prenant ainsi une taille de fenêtre suffisamment petite, il nous sera possible de détecter les blancs entre deux notes puisque, visuellement, on peut se les représenter.

Le code pour lancer ce programme est disponible sur le GitHub de Kevin Nelson : https://github.com/KevinNJ/Projects/tree/master/Short%20Time%20Fourier%20Transform et est écrit pour PyQt4.

Conclusion

Au cours de l'année nous avons tenté de créer un logiciel capable, à partir d'une mélodie de flûte à bec, de retranscrire le son sur une partition de musique.
Le programme actuel respecte les critères attendus, cependant il reste à améliorer une partie du code: la traduction. En effet, nous avons vu avec la première approche il nous manquait des données pour l'effectuer. Cependant, avec la seconde approche, nous récupérons énormément de données, la traduction est donc plus difficile à mettre en oeuvre mais réalisable.
Nous savons qu'avec plus de temps, il nous sera possible de pallier à ce problème. Une fois cela fait, nous pourrons alors commencer à étudier d'autres types d'instrument comme le piano ou la guitare, le but final étant de créer un logiciel commercialisable destiné aux novices ainsi qu'aux musiciens confirmés.

Annexe

Version du code au 11 Mai 2017 :

- Création de son/bruit
- Analyse signal pré-enregistré

```
# -*- coding: utf-8 -*-
"""
Created on Fri November  4 18:05:07 2016

@author: Armand&Pech
"""

#Importation des librairies nécéssaires au fonctionnement de Qt
import sys, os
from PyQt5 import uic #Pour extraire l'UI du code XML généré avec Qt Designer

from PyQt5.QtWidgets import QApplication, QWidget
from PyQt5 import QtWidgets
import wave, math, struct, random
from PyQt5.QtGui import QPen
from PyQt5.QtCore import Qt

import numpy as np
from scipy.io.wavfile import read
import scipy.signal as sig

#Notes de musique en notation anglosaxone (C = Do, D = Ré, ...)
NOTE_MIDI = ['C','C#','D','D#','E','F','F#','G','G#','A','A#','B']
#Note de musique dans la notation courante
NOTE = {"A":"LA",
        "B":"SI",
        "C":"DO",
        "D":"Ré",
        "E":"MI",
        "F":"FA",
        "G":"SOL",
        "C#":"DO#",
        "D#":"Ré#",
        "F#":"FA#",
        "G#":"SOL#",
        "A#":"LA#",
        }


qtCreatedFile ="C:\\Users\\Armand\\Desktop\\software.ui" #Il s'agit de l'Ui créée sur Qt
Designer
Ui_Window, QtBaseClass = uic.loadUiType(qtCreatedFile) #On récupère l'Ui dans
Ui_MainWindow

class software(QtBaseClass, Ui_Window, QWidget): #Hérite de l'Ui (pour le design) créée
et de sa classe (pour les propriétés)
        """ Classe lançant l'interface de notre programme, composée de trois onglets :
                -Le premier concerne la création de sons
                -Le deuxième, l'écriture en partition d'un son pré-enregistré
                -Le troisième, l'écriture d'une partition d'un son joué en direct
```

```
        ### CREATION DE SON ###
        Cet onglet permet de produire des sons au format .wav, à une fréquence donnée.
        On peut définir la durée du son (par défaut 1s), la fréquence d'échantillonage
(par défaut 44.100Hz),
        le nombre de sinusoïde composant le son ainsi que la fréquence de chacun.
        Il permet alors de créer des sons simples, à une fréquence, comme des sons
complexes, jusqu'à 5 fréquences.
        Une fois le son créé, on affiche un aperçu de la sinusoïde créé à l'aide de la
classe Sinusoide.

        ### ECRITURE SON PRE-ENREGISTRE ###
        Cet onglet permet d'analyser notre son pré-enregistré à l'aide de la transformée
de Fourrier.
        On peut alors choisir le fichier à analyser, la taille de la fenêtre de Fourrier
        ainsi que l'amplitude minimal à considérer (=Correction).
        La taille de la fenêtre de Fourrier doit être une puissance de 2, écrit sous la
forme
        2**X

        """

        def __init__(self):
        super().__init__()
        self.setupUi(self)

        self.tabSoftware.currentChanged.connect(lambda tab : self.selectionTab(tab))

        self.initTabCreationSon()
        self.initTabAnalysePreSaved()


        def selectionTab(self, tab):
        """Méthode qui nous permettra de redimensionner la fenêtre à chaque fois que l'on
change
        de tab, afin d'obtenir un résultat propre entre chaque tab.
        tab vaut 0 si la tab sélectionné est celle de Création de son, 1, si c'est celle
        de l'analyse de son pré-enregistré et 2 si il s'agit de la tab d'analyse de son
        en direct.
        """
        if tab == 0:
                self.setMinimumHeight(200 + (self.nbFrequence-1)*20)
                self.setMinimumWidth(410)
                self.resize(self.minimumWidth(), self.minimumHeight())
                self.tabSoftware.resize(390, 170 + (self.nbFrequence-1)*20)
        elif tab == 1:
                self.tabSoftware.resize(430, 140)
                self.setMinimumHeight(175)
                self.setMinimumWidth(450)
                self.resize(self.minimumWidth(), self.minimumHeight())

        def initTabCreationSon(self):
        """Méthode initialisant le premier onglet de notre interface QT
        avec les réglages et signaux souhaités. """
        self.nomSonText.setText("Sound")
        self.dureeText.setText("1")
        self.frequenceEchText.setText("44100")
        self.nbSinus1.setChecked(True)
        self.nbFrequence = 1


        """Le nombre de fréquences à afficher sera envoyé à la méthode nbFrequence
        lorsque l'on change la valeur du radioButton"""
        self.nbSinus1.toggled.connect(lambda : self.affichageFrequence(1))
        self.nbSinus2.toggled.connect(lambda : self.affichageFrequence(2))
```

```
        self.nbSinus3.toggled.connect(lambda : self.affichageFrequence(3))
        self.nbSinus4.toggled.connect(lambda : self.affichageFrequence(4))
        self.nbSinus5.toggled.connect(lambda : self.affichageFrequence(5))

        """ Le signal du boutton "Créer"""
        self.creerButton.clicked.connect(self.creationSon)

        """ Le signal de la checkBox"""
        self.bruit.stateChanged.connect(lambda state: self.bruitChecked(state))

        """
        On créé deux attributs de liste qui contiendront les fréquences n°X ainsi
        que les entrées correspondantes.
        L'utilité est de pouvoir ensuite parcourir ces deux tableaux afin d'afficher ou
        de cacher chaque fréquence et entrée selon le radio boutton sélectionné.
        L'ordre dans lequel les fréquences sont ajouté à la liste est important puisqu'il
        corrspond à l'odre où les fréquences sont affichés à l'écran.
        """
        self.tableauFrequence = []
        self.tableauFrequenceText = []

        self.tableauFrequence.append(self.frequence1)
        self.tableauFrequence.append(self.frequence2)
        self.tableauFrequence.append(self.frequence3)
        self.tableauFrequence.append(self.frequence4)
        self.tableauFrequence.append(self.frequence5)

        self.tableauFrequenceText.append(self.frequence1Text)
        self.tableauFrequenceText.append(self.frequence2Text)
        self.tableauFrequenceText.append(self.frequence3Text)
        self.tableauFrequenceText.append(self.frequence4Text)
        self.tableauFrequenceText.append(self.frequence5Text)


        self.affichageFrequence() #Un appel initiatique

        def affichageFrequence(self, nbFrequence=1):
        """Cette méthode affichera autant de couple label:entrée que vaut
        nbFréquence : cette variable correspond au nombre de sinusoïde que
        contiendra le signal créé.
        Par défaut, nbFrequence = 1 car le radio boutton n°1 est selectionné
        par défaut (donc par défaut, on ne créera un son composé que d'une fréquence)
        On redimensionne également la fenêtre selon le nombre de fréquence affichés.
        """
        self.nbFrequence = nbFrequence

        for i in range(0,5):
               self.tableauFrequence[i].setVisible(False)
               self.tableauFrequenceText[i].setVisible(False)
        for i in range(0,self.nbFrequence):
               self.tableauFrequence[i].setVisible(True)
               self.tableauFrequenceText[i].setVisible(True)

        self.tabSoftware.resize(390, 170 + (self.nbFrequence-1)*20)
        self.setMinimumHeight(200 + (self.nbFrequence-1)*20)
        self.setMinimumWidth(410)
        self.resize(self.minimumWidth(), self.minimumHeight())

        def creationSon(self):
        """Cette méthode s'occupe de la création de notre fichier .wav, créé dans le
        même dossier que notre executable.
        Si un fichier avec le nom spécifié existe déjà, il sera écrasé.
        32767 correspond à une amplitude de 1, -32767 à une amplitude de -1

        ### BRUIT ###
```

```
        Si la check box "Bruit" est cochée, le son générée sera aléatoire et non
sinusoïdal
        Ceci est réalisé indépendament de ou des fréquence(s) renseignée(s)

        ### SINUSOIDE ###
        Le son créé sera composé des fréquences renseignées. Le nombre de point dépends
        de la fréquence d'échantillonage indiquée, par défaut 44.100 points.
        On aura en sortie une sinusoïde approximée mais suffisament précise pour
        produire un son parfait.
        courbeAuPointi vaut, en sortie de la deuxième boucle for, au maximum le nombre
        de fréquence (sin est compris entre -1 et 1, et puisqu'on peut additionner 4 fois
dans la boucle for)
        c'est pourquoi avant d'écrire dans notre fichier audio à l'aide de
.writeframesraw, on
        divise par nbFrequence afin d'obtenir une valeur d'amplitude entre -1 et 1.
        """
        self.statusbar.setStyleSheet("color : blue")
        self.statusbar.showMessage('Processing ...')

        wavef = wave.open('{}.wav'.format(self.nomSonText.text()),'w') #Nouveau fichier
(Création / écrasement)
        wavef.setnchannels(1) # Son mono-canal
        wavef.setsampwidth(2) #La taille d'un échantillon en bytes
        wavef.setframerate(int(self.frequenceEchText.text())) #Le nombre de fenêtre

        if self.bruit.isChecked() :
                for i in range(int(self.dureeText.text()) *
int(self.frequenceEchText.text())):
                value = random.uniform(-32767.0, 32767)
                packed_value = struct.pack('<h', value)
                wavef.writeframes(packed_value)
        else:
                for i in range(int(self.dureeText.text()) *
int(self.frequenceEchText.text())): #range le nombre d'échantillons totaux
                courbeAuPointi = 0
                for _ in range(0, self.nbFrequence):
                        courbeAuPointi +=
math.sin(2*int(self.tableauFrequenceText[_].text())*math.pi*float(i)/float(self.frequenc
eEchText.text()))
                courbeAuPointi = int(32767.0 / self.nbFrequence * courbeAuPointi)
                wavef.writeframesraw( struct.pack('<h', courbeAuPointi ) ) #On incorpore
le point dans le signal
                self.sinusoide() #On affiche la sinusoïde obtenue

        wavef.writeframes(str.encode('')) #Convertion en bytes
        wavef.close()

        self.statusbar.showMessage("Done", 4000)
        self.statusbar.setStyleSheet("color : green; font : italic")



        def bruitChecked(self, state):
        """ Cette méthode correspond au signal de la CheckBox "Bruit"
        Si la checkBox est cochée, on va rendre inutilisable toutes les entrées
        de fréquences puisque nous n'en auront pas besoin pour générer notre fichier .wav
        A l'inverse, si la checkbox est décochée, on active toutes les entrées afin
        que l'utilisateur puisse y entrer des fréquences pour produire le son.
        state peut prendre les valeurs 0 et 2 : 0 signife que la checkBox est décochée
        et 2 signifie qu'elle est cochée.
        """
        if state == 0:
                for _ in range(len(self.tableauFrequenceText)):
                self.tableauFrequenceText[_].setEnabled(True)
        elif state == 2:
```

```
                for _ in range(len(self.tableauFrequenceText)):
                self.tableauFrequenceText[_].setEnabled(False)

        def sinusoide(self):
        """Cette méthode affiche la sinusoïde obtenue après création dans une
        QGraphicView. Cette QGraphicsView affichera la sinusoide que l'on dessinera
        dans sceneGraphique, notre QGraphicsScene.
        L'algorithme de calcul du point est le même. La différence
        réside dans le fait que l'on calcule le point i et le point i+1 afin
        de tracer une ligne reliant les deux points.
        courbeAuPointi correspond à notre ordonnée. A chaque tout de boucle on définit
        le niveau 0 à la moitiée de notre GraphicScene.
        On multiplie les incrémentation par 50 pour qu'elle puisse être visible à
l'écran,
        on divise par le nombre de fréquence pour ne pas sortir du cadre de la
        QGraphicView au cas où courbeAuPointi est plus grand que 1 après la deuxième
boucle
        for.
        """
        self.sceneGraphique = QtWidgets.QGraphicsScene(0, 0, 1000, 200)
        size = self.sceneGraphique.sceneRect()

        for i in range(int(size.width())*int(self.dureeText.text())):
                courbeAuPointi = int(size.height()//2)
                courbeAuPointi1 = courbeAuPointi
                for _ in range(0, self.nbFrequence):
                courbeAuPointi +=
math.sin(2*int(self.tableauFrequenceText[_].text())*math.pi*float(i)/float(self.frequenc
eEchText.text()))*100//self.nbFrequence
                courbeAuPointi1 +=
math.sin(2*int(self.tableauFrequenceText[_].text())*math.pi*float(i+1)/float(self.freque
nceEchText.text()))*100//self.nbFrequence
                self.sceneGraphique.addLine(i, courbeAuPointi, i+1, courbeAuPointi1,
QPen(Qt.red))

        self.graphique.setScene(self.sceneGraphique)

        self.legendeGraphique.setText("Aperçu de {}.wav".format(self.nomSonText.text()))
        self.tabSoftware.resize(700, 300)
        self.setMinimumWidth(720)
        self.setMinimumHeight(330)

        def initTabAnalysePreSaved(self):
        """Méthode initialisant le deuxième onglet de notre interface QT
        avec les réglages et signaux souhaités. """
        self.FFTsizeText.setText("2**15")

        """Signal du bouton "Parcourir..." """
        self.parcourir.clicked.connect(self.selectionnerFichier)

        """Signal du bouton "Analyser" """
        self.analyserButton.clicked.connect(self.analyserSon)

        def selectionnerFichier(self):
        """Méthode appelée lorsque l'on clique sur le bouton "Parcourir..."
        Elle ouvre un explorateur de fichiers, par défaut sur le bureau de
l'utilisateur"""
        self.fichierChoisi = QtWidgets.QFileDialog().getOpenFileName(self, "Fichier audio
à analyser", os.path.expanduser("~\\Desktop"), "Fichier .wav (*.wav)")
        self.fileNameText.setText(self.fichierChoisi[0])

        def analyserSon(self):
        """Cette méthode s'occupe de traiter de l'analyse de  notre son :
        elle utilise l'agorithme de la STFT pour décomposer notre signal.
        On analyse des fenêtre de Fourrier réduite, et non le son en intégralité.
```

```
        Ainsi, on peut déterminer quelle note est jouée des des intervalles
        de temps très court.

        ### EXPLICATION DES VARIABLES ###
        - rate : la fréquence d'échantillonage de notre signal
        - signal : les données du sons étudié. Si il est Stéréo, on le reconvertit en
mono
        - rééchantillone : le facteur de rééchantillonage, on rééchantillone notre signal
        pour ne le réduire qu'aux fréquences qui nous intéresses dans le cas de la flûte
        afin d'augmenter le taux de calcul (car moins de points). Généralement, si on
        part de l'hypothèse que les sons sont échantillonés à 44.100 Hz, les fréquences
possibles
        jouée par une flûtes son celle de la 4ème et 5ème octave, donc de 250Hz à 1000Hz.
        On peut ainsi rééchantillonné notre son à 4410Hz, il sera certes moins riches en
        fréquences parasites, mais il gardera toujours la bonne fréquence.
        - taille_decalage : il s'agit du nombre de points que l'on déplacera la fenetre
de Hanning
        - taille_fenêtre : Nombre de points auxquels on applique la FFT
        - t_max : Longueur du signal étudié
        - total_segments : Nombre de segments (=fenêtres) nécessaires pour parcourir le
son en
        intégralité

        """
        rate, self.signal = read(self.fileNameText.text())
        if len(self.signal.shape) != 1:
                self.signal = np.array(self.signal.sum(axis=1), dtype='float64')


        rééchantillonage = 10
        freqenceEchantillonage = eval(self.FFTsizeText.text())

        self.signal = sig.decimate(self.signal, rééchantillonage, ftype='fir')
        rate = rate // rééchantillonage
        dt = 1./rate

        taille_fenêtre = 1000
        taille_decalage = np.int32(np.floor(taille_fenêtre *
(1-float(self.overlapText.text()))))
        total_segments = np.int32(np.ceil(len(self.signal) /
np.float32(taille_decalage)))
        t_max = len(self.signal) / np.float32(rate)
        fenetreHanning = np.hanning(taille_fenêtre) * float(self.overlapText.text()) * 2
        inner_pad = np.zeros((freqenceEchantillonage * 2) - taille_fenêtre)

        proc = np.concatenate((self.signal, np.zeros(freqenceEchantillonage)))
        result = np.empty((total_segments, freqenceEchantillonage), dtype=np.float32)

        self.reconstitution = []
        self.textTraduction.setPlainText("")


        for i in range(total_segments):
                décalage = taille_decalage * i
                t = (décalage + taille_fenêtre)/rate
                segment = proc[décalage:décalage + taille_fenêtre]
                windowed = segment * fenetreHanning
                padded = np.append(windowed, inner_pad)
                spectrum = np.fft.fft(padded) / freqenceEchantillonage
                autopower = np.abs(spectrum * np.conj(spectrum))
                result[i] = autopower[:freqenceEchantillonage]
                freqSpectrum = np.fft.fftfreq(spectrum.size, dt)

                note = self.frequence_Midi( abs(freqSpectrum[np.argmax(abs(spectrum))]))
```

```
                self.textTraduction.setPlainText(self.textTraduction.toPlainText() +
"Entre {}s et {}s : F = {}\n".format( round(décalage/rate, 3), round(t, 3),
round(abs(freqSpectrum[np.argmax(abs(spectrum))]), 0) ) )
                self.textTraduction.setPlainText(self.textTraduction.toPlainText() + "Note
MIDI : {} - ".format(note))
                self.textTraduction.setPlainText(self.textTraduction.toPlainText() + "Note
: {}\n\n".format(self.Midi_Note(note)))

                self.reconstitution.append( [ ['%.3f' % (décalage/rate), '%.3f' % t if t
<t_max else t_max], [note, max(result[i])] ] )


        for _ in range(len(self.reconstitution)):
                self.reconstitution[_][1][1] =
round(20*np.log10(self.reconstitution[_][1][1]), 0)

        self.tabSoftware.resize(700, self.tabSoftware.height() )
        self.setMinimumWidth(720)
        self.setMinimumHeight(175)

        self.traduction = self.traduction_reconstitution()
        print("\n{}".format( self.traduction) )

        def frequence_Midi(self, frequence):
        """Méthode qui renvoit la conversion en MIDI d'une fréquence passée en paramètre
        La formule qu'on affecte à note correspond à celle de la conversion frequence ->
MIDI
        Elle retourne la noe et l'octave
        Elle s'aide de la liste NOTE_MIDI définie en début de code"""
        note=int(12*(np.log2(np.atleast_1d(frequence))-np.log2(440))+57)
        return NOTE_MIDI[note%12]+str(note//12)

        def Midi_Note(self, note):
        """Méthode qui renvoit la conversion en note conventionnelle d'une note MIDI
passée en paramètre
        Elle s'aide de note liste NOTE défini en début de code"""
        return "{}".format(NOTE[note[0]])+"{}".format(note[-1])


        def traduction_reconstitution(self):
        """Méthode qui renvoit une "traduction" de la reconstitution :
        On stocke les notes jouées en ignorant les decrescendos et les silences
        avec leur début et leur fin dans un tableau de tuples de la forme ((début_note,
fin_note), note)"""

        temps_ecoule = 0
        traduction = []
        for indice in range(len(self.reconstitution) - 1):
                if self.reconstitution[indice][1][0] !=
self.reconstitution[indice+1][1][0]:
                traduction.append(((temps_ecoule,self.reconstitution[indice][0][1]),
self.reconstitution[indice][1]))
                temps_ecoule = self.reconstitution[indice][0][1]
        return traduction

if __name__ == '__main__': #Si l'executable est lancé directement

        app = QApplication(sys.argv) #On démarre l'interface Qt
        app.aboutToQuit.connect(app.deleteLater) #Ligne nécéssaire pour éviter les
erreurs de kernel

        Soft = software() #On démarre notre programme à l'aide de cette commande
        Soft.show()
        app.exec_() #Pour éviter les Warnings
```

Code Interface Graphique :

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file
'C:\Users\Armand\Desktop\software.ui'
#
# Created by: PyQt5 UI code generator 5.6
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_mainWindow(object):
        def setupUi(self, mainWindow):
        mainWindow.setObjectName("mainWindow")
        mainWindow.resize(860, 469)
        mainWindow.setMinimumSize(QtCore.QSize(410, 0))
        self.centralWidget = QtWidgets.QWidget(mainWindow)
        self.centralWidget.setMinimumSize(QtCore.QSize(0, 0))
        self.centralWidget.setObjectName("centralWidget")
        self.tabSoftware = QtWidgets.QTabWidget(self.centralWidget)
        self.tabSoftware.setGeometry(QtCore.QRect(10, 10, 701, 181))
        self.tabSoftware.setMinimumSize(QtCore.QSize(0, 0))
        self.tabSoftware.setUsesScrollButtons(True)
        self.tabSoftware.setObjectName("tabSoftware")
        self.tabCreateurSon = QtWidgets.QWidget()
        self.tabCreateurSon.setMinimumSize(QtCore.QSize(0, 0))
        self.tabCreateurSon.setObjectName("tabCreateurSon")
        self.horizontalLayoutWidget = QtWidgets.QWidget(self.tabCreateurSon)
        self.horizontalLayoutWidget.setGeometry(QtCore.QRect(10, 10, 668, 261))
        self.horizontalLayoutWidget.setObjectName("horizontalLayoutWidget")
        self.creationSonLayout = QtWidgets.QHBoxLayout(self.horizontalLayoutWidget)
        self.creationSonLayout.setContentsMargins(0, 0, 0, 0)
        self.creationSonLayout.setSpacing(10)
        self.creationSonLayout.setObjectName("creationSonLayout")
        self.parametreLayout = QtWidgets.QFormLayout()

self.parametreLayout.setFieldGrowthPolicy(QtWidgets.QFormLayout.FieldsStayAtSizeHint)

self.parametreLayout.setLabelAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCo
re.Qt.AlignVCenter)
        self.parametreLayout.setHorizontalSpacing(6)
        self.parametreLayout.setVerticalSpacing(1)
        self.parametreLayout.setObjectName("parametreLayout")
        self.nomSonLabel = QtWidgets.QLabel(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.nomSonLabel.sizePolicy().hasHeightForWidth())
        self.nomSonLabel.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setFamily("MS Shell Dlg 2")
        font.setPointSize(10)
        self.nomSonLabel.setFont(font)
        self.nomSonLabel.setLayoutDirection(QtCore.Qt.RightToLeft)
        self.nomSonLabel.setAlignment(QtCore.Qt.AlignCenter)
        self.nomSonLabel.setObjectName("nomSonLabel")
        self.parametreLayout.setWidget(0, QtWidgets.QFormLayout.LabelRole,
self.nomSonLabel)
```

```
        self.nomSonText = QtWidgets.QLineEdit(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.nomSonText.sizePolicy().hasHeightForWidth())
        self.nomSonText.setSizePolicy(sizePolicy)
        self.nomSonText.setMinimumSize(QtCore.QSize(133, 0))
        font = QtGui.QFont()
        font.setFamily("MS Sans Serif")
        self.nomSonText.setFont(font)
        self.nomSonText.setAcceptDrops(False)
        self.nomSonText.setText("")
        self.nomSonText.setObjectName("nomSonText")
        self.parametreLayout.setWidget(0, QtWidgets.QFormLayout.FieldRole,
self.nomSonText)
        self.dureeLabel = QtWidgets.QLabel(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.dureeLabel.sizePolicy().hasHeightForWidth())
        self.dureeLabel.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setFamily("MS Shell Dlg 2")
        font.setPointSize(10)
        self.dureeLabel.setFont(font)
        self.dureeLabel.setObjectName("dureeLabel")
        self.parametreLayout.setWidget(1, QtWidgets.QFormLayout.LabelRole,
self.dureeLabel)
        self.dureeText = QtWidgets.QLineEdit(self.horizontalLayoutWidget)
        self.dureeText.setEnabled(True)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.dureeText.sizePolicy().hasHeightForWidth())
        self.dureeText.setSizePolicy(sizePolicy)
        self.dureeText.setObjectName("dureeText")
        self.parametreLayout.setWidget(1, QtWidgets.QFormLayout.FieldRole,
self.dureeText)
        self.frequencEchLabel = QtWidgets.QLabel(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.frequencEchLabel.sizePolicy().hasHeightForWidth())
        self.frequencEchLabel.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setFamily("MS Shell Dlg 2")
        font.setPointSize(10)
        self.frequencEchLabel.setFont(font)
        self.frequencEchLabel.setObjectName("frequencEchLabel")
        self.parametreLayout.setWidget(2, QtWidgets.QFormLayout.LabelRole,
self.frequencEchLabel)
        self.frequenceEchText = QtWidgets.QLineEdit(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.frequenceEchText.sizePolicy().hasHeightForWidth())
        self.frequenceEchText.setSizePolicy(sizePolicy)
        self.frequenceEchText.setObjectName("frequenceEchText")
```

```
        self.parametreLayout.setWidget(2, QtWidgets.QFormLayout.FieldRole,
self.frequenceEchText)
        self.nbSinusLabel = QtWidgets.QLabel(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.nbSinusLabel.sizePolicy().hasHeightForWidth())
        self.nbSinusLabel.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setFamily("MS Shell Dlg 2")
        font.setPointSize(10)
        self.nbSinusLabel.setFont(font)
        self.nbSinusLabel.setObjectName("nbSinusLabel")
        self.parametreLayout.setWidget(4, QtWidgets.QFormLayout.LabelRole,
self.nbSinusLabel)
        self.sinusLayout = QtWidgets.QHBoxLayout()
        self.sinusLayout.setObjectName("sinusLayout")
        self.nbSinus1 = QtWidgets.QRadioButton(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.nbSinus1.sizePolicy().hasHeightForWidth())
        self.nbSinus1.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setFamily("MS Shell Dlg 2")
        font.setPointSize(10)
        self.nbSinus1.setFont(font)
        self.nbSinus1.setObjectName("nbSinus1")
        self.sinusLayout.addWidget(self.nbSinus1)
        self.nbSinus2 = QtWidgets.QRadioButton(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.nbSinus2.sizePolicy().hasHeightForWidth())
        self.nbSinus2.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setFamily("MS Shell Dlg 2")
        font.setPointSize(10)
        self.nbSinus2.setFont(font)
        self.nbSinus2.setObjectName("nbSinus2")
        self.sinusLayout.addWidget(self.nbSinus2)
        self.nbSinus3 = QtWidgets.QRadioButton(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.nbSinus3.sizePolicy().hasHeightForWidth())
        self.nbSinus3.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setFamily("MS Shell Dlg 2")
        font.setPointSize(10)
        self.nbSinus3.setFont(font)
        self.nbSinus3.setObjectName("nbSinus3")
        self.sinusLayout.addWidget(self.nbSinus3)
        self.nbSinus4 = QtWidgets.QRadioButton(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.nbSinus4.sizePolicy().hasHeightForWidth())
        self.nbSinus4.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
```

```
        font.setFamily("MS Shell Dlg 2")
        font.setPointSize(10)
        self.nbSinus4.setFont(font)
        self.nbSinus4.setObjectName("nbSinus4")
        self.sinusLayout.addWidget(self.nbSinus4)
        self.nbSinus5 = QtWidgets.QRadioButton(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.nbSinus5.sizePolicy().hasHeightForWidth())
        self.nbSinus5.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setFamily("MS Shell Dlg 2")
        font.setPointSize(10)
        self.nbSinus5.setFont(font)
        self.nbSinus5.setObjectName("nbSinus5")
        self.sinusLayout.addWidget(self.nbSinus5)
        self.parametreLayout.setLayout(4, QtWidgets.QFormLayout.FieldRole,
self.sinusLayout)
        self.frequence1 = QtWidgets.QLabel(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.frequence1.sizePolicy().hasHeightForWidth())
        self.frequence1.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setPointSize(10)
        self.frequence1.setFont(font)
        self.frequence1.setMouseTracking(False)
        self.frequence1.setLayoutDirection(QtCore.Qt.RightToLeft)
        self.frequence1.setAutoFillBackground(False)
        self.frequence1.setInputMethodHints(QtCore.Qt.ImhEmailCharactersOnly)
        self.frequence1.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.frequence1.setTextFormat(QtCore.Qt.PlainText)
        self.frequence1.setScaledContents(False)

self.frequence1.setAlignment(QtCore.Qt.AlignRight|QtCore.Qt.AlignTrailing|QtCore.Qt.Alig
nVCenter)
        self.frequence1.setWordWrap(False)
        self.frequence1.setIndent(8)
        self.frequence1.setObjectName("frequence1")
        self.parametreLayout.setWidget(5, QtWidgets.QFormLayout.LabelRole,
self.frequence1)
        self.frequence1Text = QtWidgets.QLineEdit(self.horizontalLayoutWidget)
        self.frequence1Text.setObjectName("frequence1Text")
        self.parametreLayout.setWidget(5, QtWidgets.QFormLayout.FieldRole,
self.frequence1Text)
        self.frequence2 = QtWidgets.QLabel(self.horizontalLayoutWidget)
        font = QtGui.QFont()
        font.setPointSize(10)
        self.frequence2.setFont(font)
        self.frequence2.setObjectName("frequence2")
        self.parametreLayout.setWidget(6, QtWidgets.QFormLayout.LabelRole,
self.frequence2)
        self.frequence2Text = QtWidgets.QLineEdit(self.horizontalLayoutWidget)
        self.frequence2Text.setObjectName("frequence2Text")
        self.parametreLayout.setWidget(6, QtWidgets.QFormLayout.FieldRole,
self.frequence2Text)
        self.frequence3 = QtWidgets.QLabel(self.horizontalLayoutWidget)
        font = QtGui.QFont()
        font.setPointSize(10)
        self.frequence3.setFont(font)
        self.frequence3.setObjectName("frequence3")
```

```
        self.parametreLayout.setWidget(7, QtWidgets.QFormLayout.LabelRole,
self.frequence3)
        self.frequence3Text = QtWidgets.QLineEdit(self.horizontalLayoutWidget)
        self.frequence3Text.setObjectName("frequence3Text")
        self.parametreLayout.setWidget(7, QtWidgets.QFormLayout.FieldRole,
self.frequence3Text)
        self.frequence4 = QtWidgets.QLabel(self.horizontalLayoutWidget)
        font = QtGui.QFont()
        font.setPointSize(10)
        self.frequence4.setFont(font)
        self.frequence4.setObjectName("frequence4")
        self.parametreLayout.setWidget(8, QtWidgets.QFormLayout.LabelRole,
self.frequence4)
        self.frequence4Text = QtWidgets.QLineEdit(self.horizontalLayoutWidget)
        self.frequence4Text.setObjectName("frequence4Text")
        self.parametreLayout.setWidget(8, QtWidgets.QFormLayout.FieldRole,
self.frequence4Text)
        self.frequence5 = QtWidgets.QLabel(self.horizontalLayoutWidget)
        font = QtGui.QFont()
        font.setPointSize(10)
        self.frequence5.setFont(font)
        self.frequence5.setObjectName("frequence5")
        self.parametreLayout.setWidget(9, QtWidgets.QFormLayout.LabelRole,
self.frequence5)
        self.frequence5Text = QtWidgets.QLineEdit(self.horizontalLayoutWidget)
        self.frequence5Text.setObjectName("frequence5Text")
        self.parametreLayout.setWidget(9, QtWidgets.QFormLayout.FieldRole,
self.frequence5Text)
        self.creerButton = QtWidgets.QPushButton(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.creerButton.sizePolicy().hasHeightForWidth())
        self.creerButton.setSizePolicy(sizePolicy)
        self.creerButton.setObjectName("creerButton")
        self.parametreLayout.setWidget(10, QtWidgets.QFormLayout.LabelRole,
self.creerButton)
        self.bruit = QtWidgets.QCheckBox(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.bruit.sizePolicy().hasHeightForWidth())
        self.bruit.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setPointSize(8)
        self.bruit.setFont(font)
        self.bruit.setAutoFillBackground(False)
        self.bruit.setChecked(False)
        self.bruit.setAutoRepeat(False)
        self.bruit.setAutoExclusive(False)
        self.bruit.setTristate(False)
        self.bruit.setObjectName("bruit")
        self.parametreLayout.setWidget(10, QtWidgets.QFormLayout.FieldRole, self.bruit)
        self.creationSonLayout.addLayout(self.parametreLayout)
        self.graphiqueLayout = QtWidgets.QVBoxLayout()
        self.graphiqueLayout.setSpacing(4)
        self.graphiqueLayout.setObjectName("graphiqueLayout")
        self.graphique = QtWidgets.QGraphicsView(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,
QtWidgets.QSizePolicy.Expanding)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.graphique.sizePolicy().hasHeightForWidth())
```

```
        self.graphique.setSizePolicy(sizePolicy)
        self.graphique.setObjectName("graphique")
        self.graphiqueLayout.addWidget(self.graphique)
        self.legendeGraphique = QtWidgets.QLabel(self.horizontalLayoutWidget)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Preferred,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.legendeGraphique.sizePolicy().hasHeightForWidth())
        self.legendeGraphique.setSizePolicy(sizePolicy)
        self.legendeGraphique.setLayoutDirection(QtCore.Qt.LeftToRight)
        self.legendeGraphique.setLineWidth(11)
        self.legendeGraphique.setAlignment(QtCore.Qt.AlignCenter)
        self.legendeGraphique.setObjectName("legendeGraphique")
        self.graphiqueLayout.addWidget(self.legendeGraphique)
        self.creationSonLayout.addLayout(self.graphiqueLayout)
        self.tabSoftware.addTab(self.tabCreateurSon, "")
        self.analysePreEnregistre = QtWidgets.QWidget()
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,
QtWidgets.QSizePolicy.Preferred)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.analysePreEnregistre.sizePolicy().hasHeightForWidth())
        self.analysePreEnregistre.setSizePolicy(sizePolicy)
        font = QtGui.QFont()
        font.setPointSize(10)
        self.analysePreEnregistre.setFont(font)
        self.analysePreEnregistre.setObjectName("analysePreEnregistre")
        self.horizontalLayoutWidget_3 = QtWidgets.QWidget(self.analysePreEnregistre)
        self.horizontalLayoutWidget_3.setGeometry(QtCore.QRect(10, 10, 671, 102))
        self.horizontalLayoutWidget_3.setObjectName("horizontalLayoutWidget_3")
        self.horizontalLayout_2 = QtWidgets.QHBoxLayout(self.horizontalLayoutWidget_3)
        self.horizontalLayout_2.setContentsMargins(0, 0, 0, 0)
        self.horizontalLayout_2.setObjectName("horizontalLayout_2")
        self.analysePresavedLayout = QtWidgets.QFormLayout()

self.analysePresavedLayout.setLabelAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|
QtCore.Qt.AlignVCenter)

self.analysePresavedLayout.setFormAlignment(QtCore.Qt.AlignLeading|QtCore.Qt.AlignLeft|Q
tCore.Qt.AlignTop)
        self.analysePresavedLayout.setVerticalSpacing(2)
        self.analysePresavedLayout.setObjectName("analysePresavedLayout")
        self.fileLabel = QtWidgets.QLabel(self.horizontalLayoutWidget_3)
        font = QtGui.QFont()
        font.setPointSize(10)
        self.fileLabel.setFont(font)
        self.fileLabel.setObjectName("fileLabel")
        self.analysePresavedLayout.setWidget(0, QtWidgets.QFormLayout.LabelRole,
self.fileLabel)
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.fileNameText = QtWidgets.QLineEdit(self.horizontalLayoutWidget_3)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.fileNameText.sizePolicy().hasHeightForWidth())
        self.fileNameText.setSizePolicy(sizePolicy)
        self.fileNameText.setObjectName("fileNameText")
        self.horizontalLayout.addWidget(self.fileNameText)
        self.parcourir = QtWidgets.QToolButton(self.horizontalLayoutWidget_3)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,
```

```
QtWidgets.QSizePolicy.Maximum)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.parcourir.sizePolicy().hasHeightForWidth())
        self.parcourir.setSizePolicy(sizePolicy)
        self.parcourir.setObjectName("parcourir")
        self.horizontalLayout.addWidget(self.parcourir)
        self.analysePresavedLayout.setLayout(0, QtWidgets.QFormLayout.FieldRole,
self.horizontalLayout)
        self.FFTsizeLabel = QtWidgets.QLabel(self.horizontalLayoutWidget_3)
        font = QtGui.QFont()
        font.setPointSize(10)
        self.FFTsizeLabel.setFont(font)
        self.FFTsizeLabel.setObjectName("FFTsizeLabel")
        self.analysePresavedLayout.setWidget(1, QtWidgets.QFormLayout.LabelRole,
self.FFTsizeLabel)
        self.FFTsizeText = QtWidgets.QLineEdit(self.horizontalLayoutWidget_3)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.FFTsizeText.sizePolicy().hasHeightForWidth())
        self.FFTsizeText.setSizePolicy(sizePolicy)
        self.FFTsizeText.setObjectName("FFTsizeText")
        self.analysePresavedLayout.setWidget(1, QtWidgets.QFormLayout.FieldRole,
self.FFTsizeText)
        self.overlapLabel = QtWidgets.QLabel(self.horizontalLayoutWidget_3)
        font = QtGui.QFont()
        font.setPointSize(10)
        font.setBold(False)
        font.setWeight(50)
        font.setKerning(False)
        self.overlapLabel.setFont(font)
        self.overlapLabel.setObjectName("overlapLabel")
        self.analysePresavedLayout.setWidget(2, QtWidgets.QFormLayout.LabelRole,
self.overlapLabel)
        self.overlapText = QtWidgets.QLineEdit(self.horizontalLayoutWidget_3)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,
QtWidgets.QSizePolicy.Fixed)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)
        sizePolicy.setHeightForWidth(self.overlapText.sizePolicy().hasHeightForWidth())
        self.overlapText.setSizePolicy(sizePolicy)
        self.overlapText.setObjectName("overlapText")
        self.analysePresavedLayout.setWidget(2, QtWidgets.QFormLayout.FieldRole,
self.overlapText)
        self.analyserButton = QtWidgets.QPushButton(self.horizontalLayoutWidget_3)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,
QtWidgets.QSizePolicy.Minimum)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.analyserButton.sizePolicy().hasHeightForWidth())
        self.analyserButton.setSizePolicy(sizePolicy)
        self.analyserButton.setObjectName("analyserButton")
        self.analysePresavedLayout.setWidget(3, QtWidgets.QFormLayout.FieldRole,
self.analyserButton)
        self.horizontalLayout_2.addLayout(self.analysePresavedLayout)
        self.textTraduction = QtWidgets.QTextEdit(self.horizontalLayoutWidget_3)
        sizePolicy = QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Maximum,
QtWidgets.QSizePolicy.Maximum)
        sizePolicy.setHorizontalStretch(0)
        sizePolicy.setVerticalStretch(0)

sizePolicy.setHeightForWidth(self.textTraduction.sizePolicy().hasHeightForWidth())
```

```
        self.textTraduction.setSizePolicy(sizePolicy)
        self.textTraduction.setDocumentTitle("")
        self.textTraduction.setReadOnly(True)
        self.textTraduction.setTextInteractionFlags(QtCore.Qt.NoTextInteraction)
        self.textTraduction.setObjectName("textTraduction")
        self.horizontalLayout_2.addWidget(self.textTraduction)
        self.tabSoftware.addTab(self.analysePreEnregistre, "")
        mainWindow.setCentralWidget(self.centralWidget)
        self.statusbar = QtWidgets.QStatusBar(mainWindow)
        self.statusbar.setObjectName("statusbar")
        mainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(mainWindow)
        self.tabSoftware.setCurrentIndex(0)
        QtCore.QMetaObject.connectSlotsByName(mainWindow)

        def retranslateUi(self, mainWindow):
        _translate = QtCore.QCoreApplication.translate
        mainWindow.setWindowTitle(_translate("mainWindow", "Reconnaissance des notes de
musique"))
        self.nomSonLabel.setText(_translate("mainWindow", "Nom du son :"))
        self.dureeLabel.setText(_translate("mainWindow", "Durée : (en s)"))
        self.frequencEchLabel.setText(_translate("mainWindow", "Fréquence
d\'échantillonage : (en Hz)"))
        self.nbSinusLabel.setText(_translate("mainWindow", "Nombre de sinusoïdes :"))
        self.nbSinus1.setText(_translate("mainWindow", "1"))
        self.nbSinus2.setText(_translate("mainWindow", "2"))
        self.nbSinus3.setText(_translate("mainWindow", "3"))
        self.nbSinus4.setText(_translate("mainWindow", "4"))
        self.nbSinus5.setText(_translate("mainWindow", "5"))
        self.frequence1.setText(_translate("mainWindow", "Fréquence 1 :"))
        self.frequence2.setText(_translate("mainWindow", "Fréquence 2 :"))
        self.frequence3.setText(_translate("mainWindow", "Fréquence 3 :"))
        self.frequence4.setText(_translate("mainWindow", "Fréquence 4 :"))
        self.frequence5.setText(_translate("mainWindow", "Fréquence 5 :"))
        self.creerButton.setText(_translate("mainWindow", "Créer"))
        self.bruit.setText(_translate("mainWindow", "Bruit"))
        self.legendeGraphique.setText(_translate("mainWindow", "Aperçu de nomSonText"))
        self.tabSoftware.setTabText(self.tabSoftware.indexOf(self.tabCreateurSon),
_translate("mainWindow", "Créateur de sons"))
        self.fileLabel.setText(_translate("mainWindow", "Fichier audio à analyser :"))
        self.parcourir.setText(_translate("mainWindow", "Parcourir..."))
        self.FFTsizeLabel.setText(_translate("mainWindow", "Taille de la fenêtre de
Fourrier :"))
        self.overlapLabel.setText(_translate("mainWindow", "Pourcentage de chevauchement
:"))
        self.overlapText.setText(_translate("mainWindow", "0.9"))
        self.analyserButton.setText(_translate("mainWindow", "Analyser"))
        self.tabSoftware.setTabText(self.tabSoftware.indexOf(self.analysePreEnregistre),
_translate("mainWindow", "Analyse de sons pré-enregistré"))
```

Bibliographie

---

transformée de Fourier:

- Transformées de Fourier avec Numpy : https://docs.scipy.org/doc/numpy/reference/routines.fft.html#standard-ffts (Consulté le 11/05)
- Transformée de Fourier Wikipédia : https://fr.wikipedia.org/wiki/Transformation_de_Fourier#Transformation_de_Fourier_pour_les_fonctions_int.C3.A9grables (Consulté le 11/05)
- Différentes applications de la FFT : http://www.tangentex.com/PythonTFD.htm (Consulté le 11/05)
- Explcation algorithme FFT : http://herve.boeglen.free.fr/Tsignal/chapitre5/chapitre5.htm (Consulté le 11/05)
- Différente analyse de spectre : http://acoustique.e-monsite.com/pages/partie-iii/analyse-des-sons-musicaux.html (Consulté le 11/05)

Son:

- Généralité sur le son : http://kljh.pagesperso-orange.fr/Vision/Muse/ (Consulté le 11/05)
- Comment lire une partition : http://www.apprendrelesolfege.com/ (Consulté le 11/05)
- Créer des sinusoides en Python : http://blog.acipo.com/wave-generation-in-python/ (Consulté le 11/05)
- Note de musique Wikipedia : https://fr.wikipedia.org/wiki/Note_de_musique (Consulté le 11/05)
- Conversion fréquence/MIDI : https://www.seventhstring.com/resources/notefrequencies.html (Consulté le 11/05)
- Conversion fréquence/MIDI en ligne :http://newt.phys.unsw.edu.au/music/note/
- Créer du bruit en Python : https://soledadpenades.com/2009/10/29/fastest-way-to-generate-wav-files-in-python-using-the-wave-module/ (Consulté le 11/05)
- analyse spectrale (consulté le 11/05) http://www.ta-formation.com/acrobat-cours/spectre.pdf

Bibliothèque python:

- Interface graphique avec Tkinter : http://apprendre-python.com/page-tkinter-interface-graphique-python-tutoriel (Consulté le 11/05)
- Tangentex : http://www.tangentex.com/index.htm (Consulté le 11/05)
- Doc Qt5 : http://doc.qt.io/qt-5/ (Consulté le 11/05)
- Documentation Matplotlib : http://matplotlib.org/ (Consulté le 11/05)
- Caractéristique de la flute: http://blog.u-bourgogne.fr/etiennesafa/?p=160 (Consulté le 11/05)

Autre:

- AudioFingerprint (Shazam) : http://willdrevo.com/fingerprinting-and-audio-recognition-with-python/ (Consulté le 11/05)
- Kevin Nelson TFCT : https://kevinsprojects.wordpress.com/2014/12/13/short-time-fourier-transform-using-python-and-numpy/ (Consulté le 11/05)
- Audacity : http://audacity.fr/ (Consulté le 11/05)