Cachier des charges : Traitement du signal

Détection de notes par analyse Fréquentielle

Alan Courtel
Paul Dufour
Guillaume Senderens

Encadrants : Nelly Barrau et Lucie Desplat

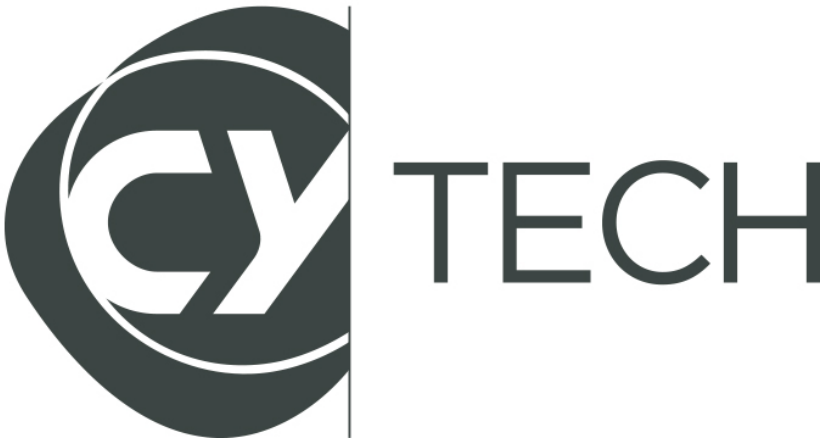


Table des matières

1 Introduction

Dans le cadre du projet de Traitement du Signal, nous avons décidé de nous pencher sur un problème de détection de notes à partir d'un instrument avec un ou plusieurs accords.
Le projet de détection de notes et d'accords est utilisé dans l'industrie musicale. L'application shazam leader dans la détection de musique a lancé son application de détection de morceaux. Son application permet de détecter une musique jouée pour connaître le nom de l'artiste et le nom du morceau parmi une base de données de plus de 8 millions de titres (chiffres publiés en 2021).
Notre projet est plus avancé, nous voulons offrir un produit aux musiciens amateurs et professionnels qui permettrait d'établir une partition musicale à partir d'une musique à un seul instrument d'abord, mais peut-être avec plusieurs instruments à l'avenir.

2 Objectifs

L'objectif de notre projet porte sur la détection de notes à partir d'un instrument avec un ou plusieurs accords. Pour cela nous allons utiliser la transformée de Fourier. L'objectif ultime serait de détecter les notes de manière à générer une partition de musique.

3 Aspect fonctionnels et techniques

3.1 Les Outils Mathématiques

La théorie de Fourier, aussi appelée analyse spectrale, permet fondamentalement de passer d'un signal temporel à sa représentation spectrale. D'un point de vue mathématique, la théorie de Fourier transforme une fonction intégrable sur R en une fonction décrivant son spectre fréquentiel.

Théorème de Fourier : Toute fonction continue périodique peut être décomposée en somme (série) de fonctions sinus et cosinus.

$$f(t) = \frac{a_0}{2} + \sum_{n=-1}^{+\infty} (a_n \cos(n\omega t) + b_n \sin(n\omega t)) \tag{1}$$

avec $\mathrm{a}_n = \frac{2}{T} \int_{-\frac{T}{2}}^{+\frac{T}{2}} f(t) \cos(n\omega t)\,\mathrm{d}t$ *et* $b_n = \frac{2}{T} \int_{-\frac{T}{2}}^{+\frac{T}{2}} f(t) \sin(n\omega t)\,\mathrm{d}t$

On peut généraliser ce théorème aux fonctions continues non périodiques en utilisant la transformée de Fourier :

$$\mathrm{F(w)} = \frac{1}{2\pi} \int_R f(t) \exp(-iwt)\,\mathrm{d}t$$

On utilise l'outil mathématique DFT (Transformée de Fourier Discrète) en python pour discrétiser la fonction et l'échantillonner. L'algorithme FFT (Transformée de Fourier Rapide) est une manière efficace de calculer la DFT d'une fonction en utilisant les symétries dans les termes de la fonction. La DFT ne fonctionne en théorie qu'avec des signaux périodiques. Sa définition mathématique pour un signal s de N échantillons est la suivante :

$$\mathrm{S(k)} = \sum_{n=0}^{N-1} s(n) e^{\frac{-2i\pi kn}{N}}$$

Conditions pour utiliser l'algorithme FFT :
D'après le théorème de Shannon, la fréquence de décomposition des harmoniques doit être au plus égale à la moitié de la fréquence d'échantillonnage du signal. De plus, en pratique, les signaux enregistrés sont issus d'une acquisition électronique via le logiciel Audacity, ils sont donc non périodiques. La DFT commence d'abord par répéter plusieurs fois le signal de manière à obtenir un signal périodique.

En python on emploie en pratique deux fonctions du module numpy :
numpy.fft.fft : calcule la transformée de Fourier discrète unidimensionnelle.
numpy.fft.fftfreq : Renvoie les fréquences d'échantillonnage de la transformée discrète de Fourier
analyse de gradient pour déterminer les pics d'amplitude dans la méthode complexe (plusieurs notes joué) :

Définition du gradient : Le gradient est la généralisation à plusieurs variables de la dérivée d'une fonction d'une seule variable. Il est défini en tout point où la fonction est différentiable. C'est un vecteur qui caractérise la variabilité de cette fonction au voisinage de ce point.

3.2 Préparation des données

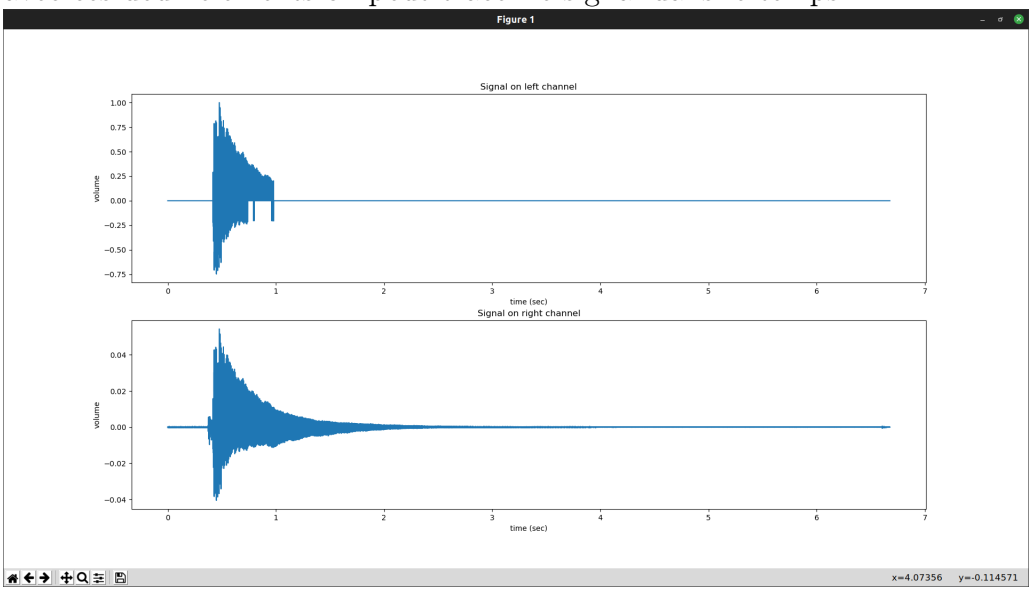
L'enregistrement des fichiers audio de test est fait de la façon suivante :

- Un micro est placé au dessus d'un piano
- On enregistre une note, un accord ou une suite de notes/accords avec Audacity
- On exporte l'enregistrement au format .wav pour analyse

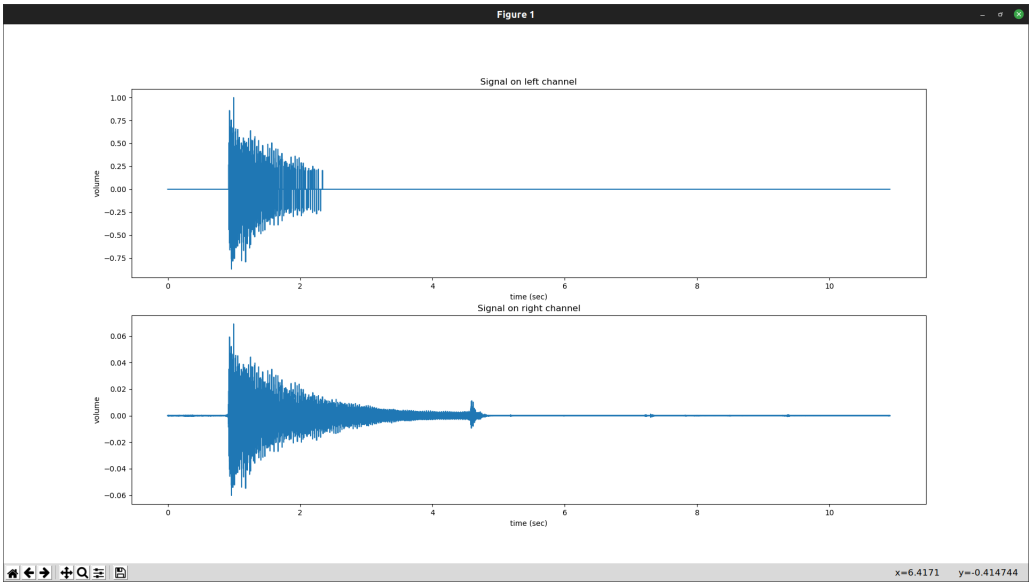
Le reste du projet est réalisé en python sur ces fichiers audio.

3.3 Méthode de détection simple : détection d'une note joué par un instrument

Pour lire les fichiers audio au format wav on utilise la librairie soundfile de python, qui nous retourne un numpy array et la fréquence d'échantillonnage, avec ces deux éléments on peut tracer le signal dans le temps.

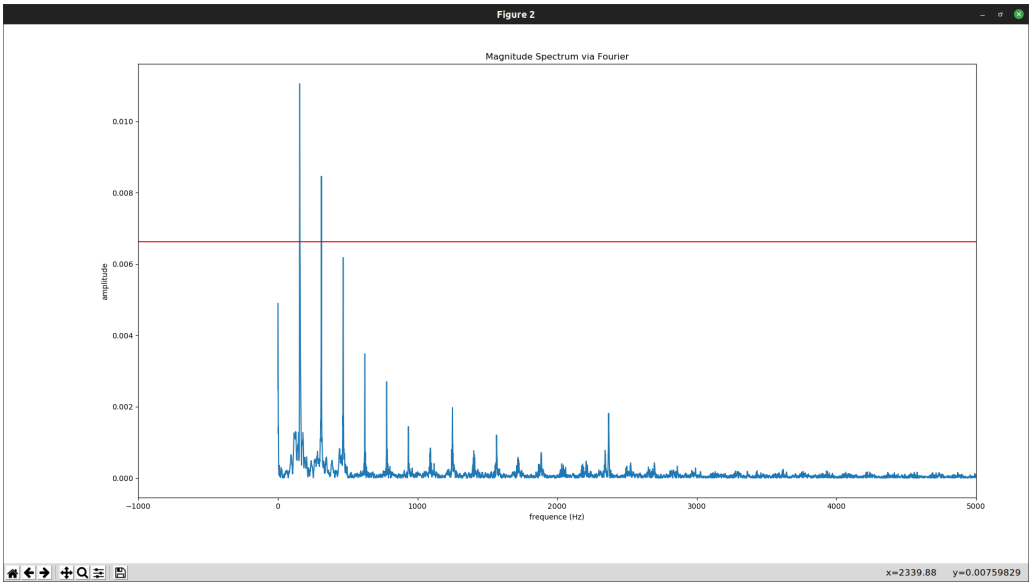


Signal de la note D#3 joué au piano

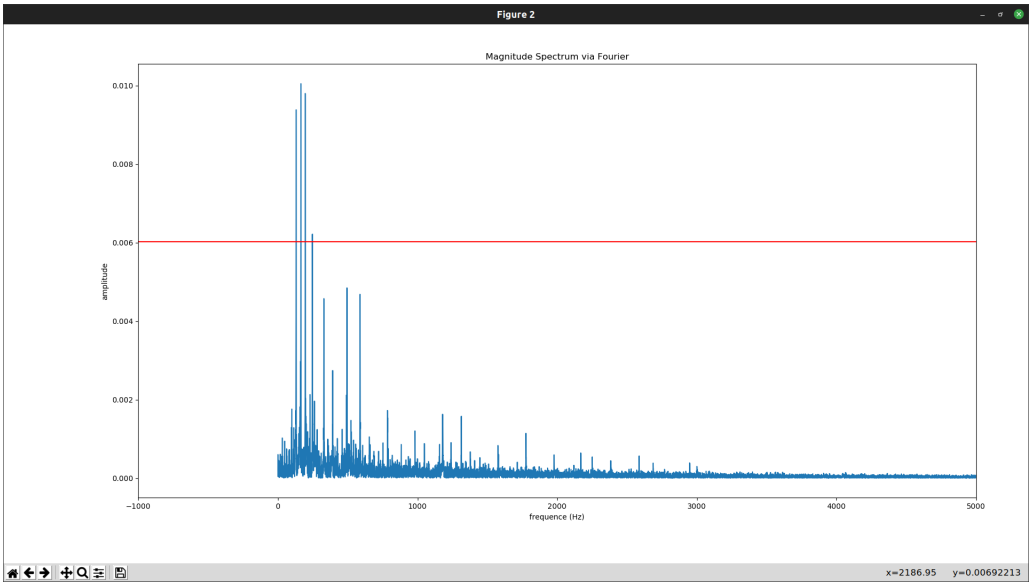


Signal de l'accord C3maj7 au piano

Le vecteur de données retourner par la fonction read de soundfile nous confie deux canaux audio mais nous utiliserons uniquement le canal correspondant à l'oreille gauche, la coupur bizarre sur l'image correspond à une fonction qui met les valeurs à 0 lorsqu'elle en dessous d'un certain seuil, on s'en rendu compte que l'analyse spectrale était plus clair en faisant cela.
On va ensuite pouvoir tracer le spectre en amplitude à l'aide de la librairie numpy.fft et des fonctions fft et fftfreq de cette librairie, ces deux fonctions nous permettront d'obtenir respectivement le vecteur des amplitudes du signal et le vecteur des fréquences correspondant à ces amplitudes. Il suffit ensuite de tracer les graphiques :



Spectre en amplitude note D#3



Spectre en amplitude accord C3maj7

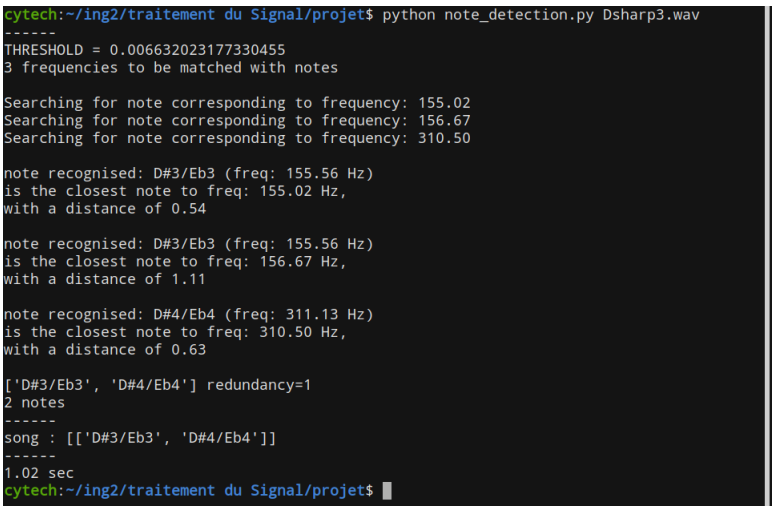
La droite rouge sur ces graphiques représente le seuil en amplitude pour être considéré comme une note que l'on défini de la façon suivante :
SEUIL = AMPLITUDEMAX * MULTIPLICATEUR. Le multiplicateur est fixé à 70% après des tests manuels. On a aussi pensé à prendre un facteur de l'amplitude moyenne plutôt que max mais le problème c'est qu'elle est

extrêmement proche de 0 et une variation énorme selon le signal. Pour récupérer les fréquences d'amplitude haute on parcours le vecteur (array) qui correspond aux amplitudes, si l'amplitude est au-dessus du seuil on récupère la fréquence correspondante. A cette étape, on a remarqué que les fréquences était capturé de nombreuses pour un seul pic, ce qui est redondant, pour éviter cela nous avons instauré un saut de fréquence qui fonctionne de la manière suivante : si on trouve un valeur au dessus du seuil on saute X valeur dans le parcours de vecteur, après des tests on a établit ce saut de fréquence à FJUMP = 15 par défaut. On a maintenant un vecteur qui contient toutes les fréquences censées correspondre à des notes. Pour trouver les notes correspondantes, on utilise le tableau des notes-fréquences :

NOTE	OCTAVE 0	OCTAVE 1	OCTAVE 2	OCTAVE 3	OCTAVE 4	OCTAVE 5	OCTAVE 6	OCTAVE 7	OCTAVE 8
C	16.35	32.7	65.41	130.81	261.63	523.25	1046.5	2093	4186.01
C#/Db	17.32	34.65	69.3	138.59	277.18	554.37	1108.73	2217.46	4434.92
D	18.35	36.71	73.42	146.83	293.66	587.33	1174.66	2349.32	4698.63
D#/Eb	19.45	38.89	77.78	155.56	311.13	622.25	1244.51	2489.02	4978.03
E	20.6	41.2	82.41	164.81	329.63	659.25	1318.51	2637.02	5274.04
F	21.83	43.65	87.31	174.61	349.23	698.46	1396.91	2793.83	5587.65
F#/Gb	23.12	46.25	92.5	185	369.99	739.99	1479.98	2959.96	5919.91
G	24.5	49	98	196	392	783.99	1567.98	3135.96	6271.93
G#/Ab	25.96	51.91	103.83	207.65	415.3	830.61	1661.22	3322.44	6644.88
A	27.5	55	110	220	440	880	1760	3520	7040
A#/Bb	29.14	58.27	116.54	233.08	466.16	932.33	1864.66	3729.31	7458.62
B	30.87	61.74	123.47	246.94	493.88	987.77	1975.53	3951.07	7902.13

Tableau de correspondance note-fréquence

(source : `https://studioguru.co/producer-tools/note-frequency-chart/`)
On compare ensuite les fréquences capturées précédemment et on calcule la distance à chaque note avec la norme 1 (valeur absolue de l'écart en fréquence), et on associe chaque fréquence à la note qui a la distance la plus petite. L'analyse est finie on retourne donc la liste des notes reconnues :



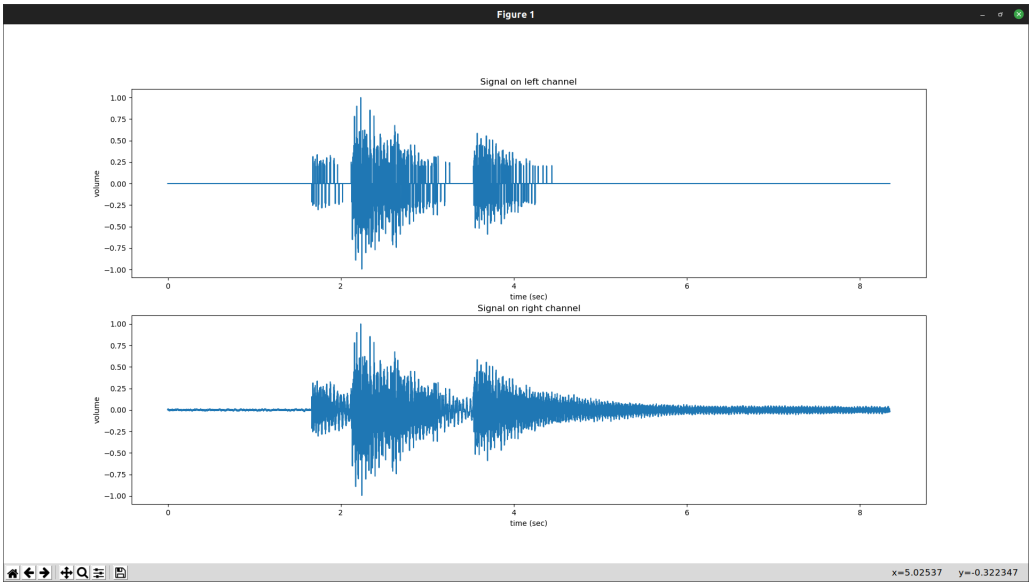
Sortie du terminal pour l'analyse simple

Cette capture d'écran montre deux limitations du projet :

- le terme "redundancy=1" signifie qu'on a capturé des fréquences correspondant à la même note, ça rallonge un peu le temps de calcul c'est donc à éviter en augmentant le pas en fréquence (FJUMP) par exemple.
- la note D4 est considérée comme étant joué mais il s'agit en fait d'une harmonique de la note qui fut réellement joué D3, il n'existe pas de méthode pratique pour différencier les deux.

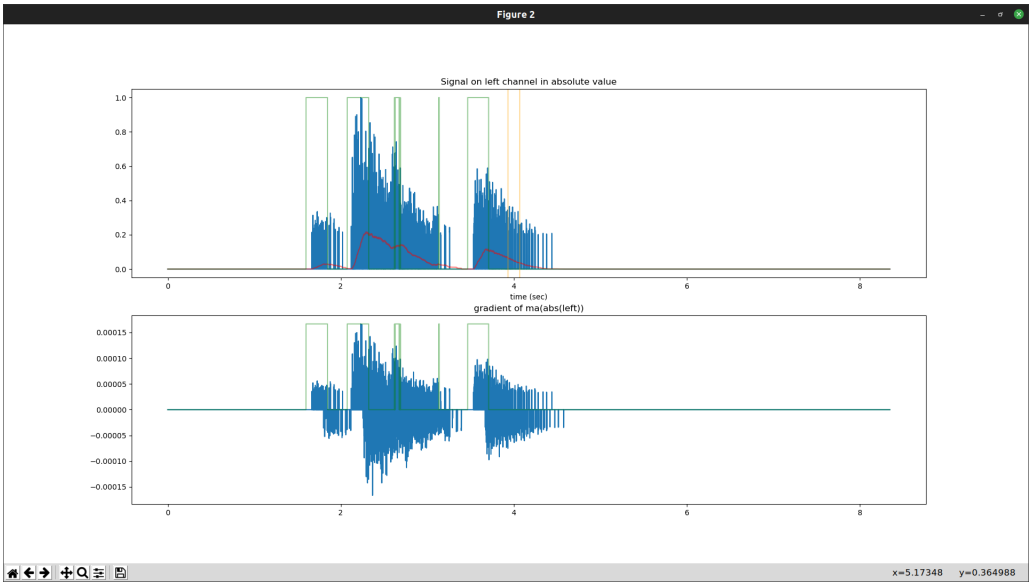
3.4 Méthode complexe : plusieurs notes jouées dans le temps

L'analyse complexe constitue la deuxième partie de notre travail, notre but est de pouvoir reconnaître une suite de notes/accords réparties dans le temps. En premier lieu le but est donc de déterminer à quel moment chaque accord est joué, voici le signal sur lequel nous avons travaillé :



Signal de 4 notes/accords joués

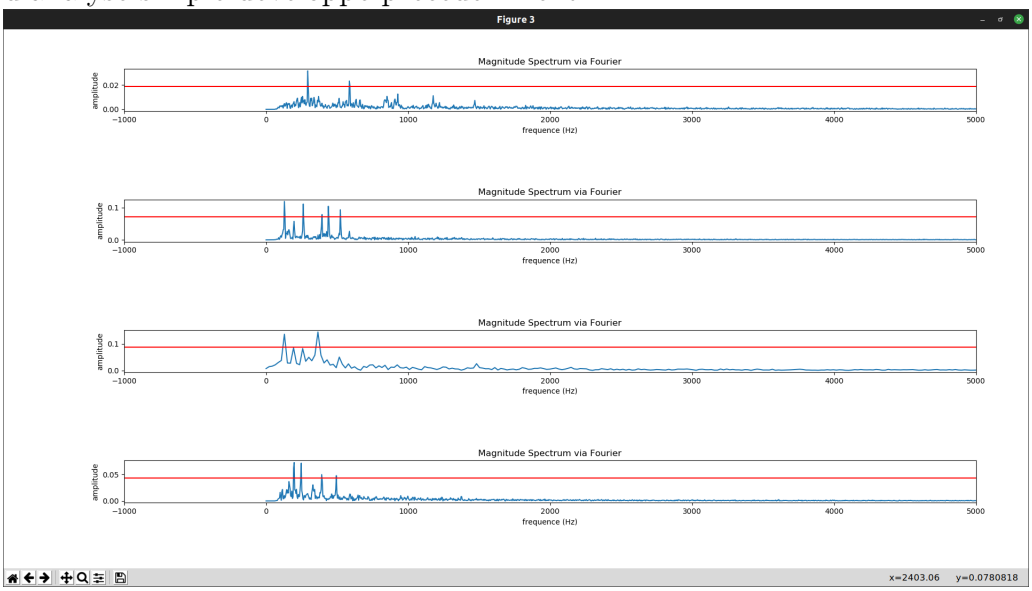
Dans cet audio on a joué : 1 note, 1 accord (4 notes), 1 note, 1 accord (5 notes) On a gardé le même filtre supprimeur de volume, ici appliqué sur le canal gauche (canal de travail). Une première idée était de simplement trouver les grands changements de volume, cependant cette méthode est ne fonctionne uniquement sur des audios ou les notes sont bien distinctes, mais dans notre exemple, il y a une note qui est joué sur la partie du milieu caché dans le bruit créé par l'accord joué avant (à environ 2s). Il nous fallait donc une méthode plus subtile et précise, pour ce faire nous étudierons la positivité du gradient de ce signal. L'image qui suit représente le reste du travail :



Analyse complexe du signal

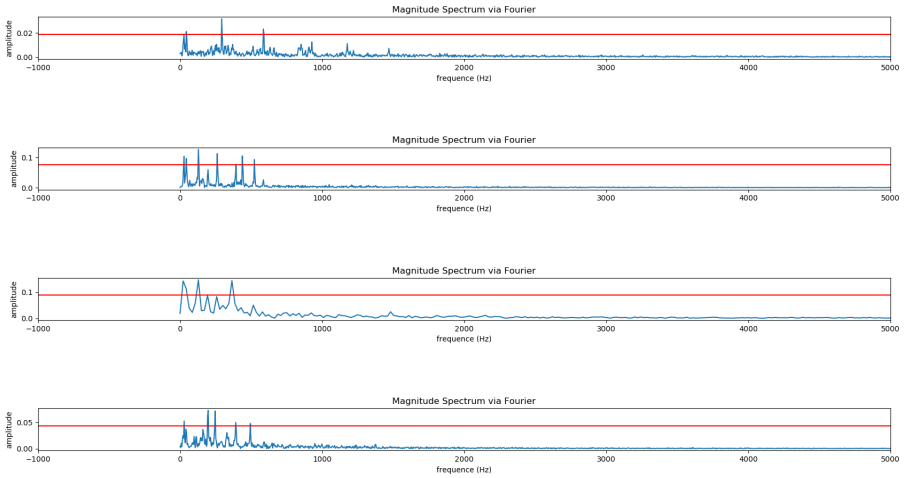
Commençons par le graphique du haut, celui-ci représente le signal en valeur absolue, qui permet une analyse plus fiable car tout ce qui nous intéresse sont les augmentations de volume. Pour continuer l'analyse nous avons ensuite appliquer un filtre moyenneur d'ordre 6000, c'est-à-dire que chaque point de donnée devient la moyenne des 6000 derniers points, ce signal moyenné correspond à la courbe rouge du premier graphique, l'idée est que les variations de volume seront plus visibles et clairs sur un telle signal. Ensuite c'est là qu'intervient le gradient, nous avons appliqué la fonction grad() de la librairie numpy pour donner une idée de la pente du signal à chaque instant, bien sûr la gradient (figure du dessous) oscille énormément puisque le signal, même moyenné, comporte des oscillations, mais on a quand même pu se rendre compte qu'une pente croissante (augmentation de volume) correspond à des valeur plutôt positive du gradient autour d'un point donné. Nous avons donc développé une fonction indicatrice qui parcourt le gradient, autour de chaque point on somme les valeurs à droite et à gauche, si cette somme est positive on écrit 1 dans un vecteur de même que celui du gradient sinon on écrit 0. Cette fonction permet donc d'établir les zones de volumes croissant, c'est la courbe verte sur les deux graphiques. Les 4 plus grands rectangles ont bien reconnu les zones où les notes sont jouées, on peut apercevoir un dernier rectangle très fin, il correspond au bruit de main sur le clavier lors de l'enregistrement, c'est donc une erreur mais la finesse du rectangle nous permettra

de le filtrer.
Nous utilisons ensuite une fonction qui va à partir de cette indicatrice et du signal original nous retourner des échantillons audios qui correspondent à chaque note/accords, ces échantillons sont ensuite envoyées dans la fonction d'analyse simple développé précédemment :



Spectre en amplitude des différentes notes/accords

Petite distinction c'est qu'est sur cet exemple, nous avons dû utiliser un filtre passe haut butterworth, d'ordre 5 et fréquence de coupure 90 Hertz car il y avait un pic énorme dans les très basse fréquences qui correspondait au bruit du clavier, c'est donc un défaut d'enregistrement mais il fallait le résoudre. Voici à quoi ressemblait les spectres avant le filtre :



Spectre en amplitude avant filtre Butterworth

Les résultats de cette analyse multiple sont ensuite recombinés et retournés ce qui finit notre programme.

```
note recognised: C3 (freq: 130.81 Hz)
is the closest note to freq: 128.88 Hz,
with a distance of 1.93

note recognised: F#4/Gb4 (freq: 369.99 Hz)
is the closest note to freq: 365.17 Hz,
with a distance of 4.82

['C3', 'F#4/Gb4'] redundancy=0
2 notes


==================
THRESHOLD = 0.04347696103979455
4
4 frequencies to be matched with notes

Searching for note corresponding to frequency: 193.86
Searching for note corresponding to frequency: 247.47
Searching for note corresponding to frequency: 391.84
Searching for note corresponding to frequency: 494.95

note recognised: G3 (freq: 196.00 Hz)
is the closest note to freq: 193.86 Hz,
with a distance of 2.14

note recognised: B3 (freq: 246.94 Hz)
is the closest note to freq: 247.47 Hz,
with a distance of 0.53

note recognised: G4 (freq: 392.00 Hz)
is the closest note to freq: 391.84 Hz,
with a distance of 0.16

note recognised: B4 (freq: 493.88 Hz)
is the closest note to freq: 494.95 Hz,
with a distance of 1.07

['G3', 'B3', 'G4', 'B4'] redundancy=0
4 notes
------
song : [['D4', 'D5'], ['C3', 'C4', 'G4', 'A4', 'C5'], ['C3', 'F#4/Gb4'], ['G3', 'B3',
'G4', 'B4']]
------
3.50 sec
cytech:~/ing2/traitement du Signal/projet$
```

Sortie du terminal après analyse complexe

les vrais notes/accords joués étaient les suivants :

- D4
- C3,G3,C4,A4
- F4

— G2,E3,G3,B3,E4

On peut voir qu'il y a beaucoup d'harmoniques qui apparaissent et que certaines n'ont pas été détectées, on peut expliquer cela par la limitation de l'analyse du spectre en amplitude avec un seuil, certaines notes donnait des pics beaucoup trop faible pour être reconnues, et les notes jouées les plus fortes ont créés des pics d'harmoniques très important également.

4 Axe d'amélioration

On a voulu améliorer notre programme pour qu'il puisse détecter une suites de notes dans le temps d'un instrument.
Lors du parcours du spectre en amplitude, pour éviter la redondance de capture de fréquence, c'est-à-dire les fréquences qui correspondent au même pic d'amplitudes. On a mis en place un "saut de fréquence", on saute plusieurs points de données, dès qu'un pic est détecté.
Pour les 2 méthodes : On a utilisé un filtre passe haut avant d'appliquer la méthode de fourier pour enlever le bruit lié au fait qu'on a enregistrer avec un micro plutôt que de directement enregistrer la sortie du piano. point d'amélioration.
Nous pourrions améliorer le seuil en amplitude pour détecter une note ou non. Nous pourrions mieux détecter le début de la note à l'aide du gradient par exemple. Nous n'avons pas encore testé sur un audio avec plusieurs instruments, dont on ne peut garantir le résultat et l'optimisation du résultat. Par ailleurs, il est difficile de différencier une harmonique d'une note qui a été réellement jouée.

5 Limites et Contraintes

Les fichiers audio nécessitent une bonne qualité audio. De plus, le traitement de la détection est long : l'analyse dure 3 secondes pour un audio d'une durée de 8 secondes. Sur une partition de plusieurs minutes le traitement peut durer plusieurs dizaines de secondes voire minutes.

6 Les apports des sources externes

Avant de commencer à travailler sur le projet, nous avons effectué des simples recherches Google pour voir si un travail similaire de détection de note à déjà été effectué. Les quelques liens que nous avons trouvés n'ont pas été très utile dès le départ car il était difficile de comprendre la structure des programmes et de suivre le raisonnement des codeurs sans avoir commencer à travailler sur le projet. Ce n'est qu'une fois après avoir bien avancé sur le projet qu'on s'est rendu compte que la structure de notre travail coïncidait avec celle de ces autres codeurs. Nous avons emprunter un aspect technique de ce travail :
`https://github.com/Amagnum/Music-notes-detection`
Cette article explique l'idée d'un saut de fréquence pour éviter la redondance lors de la capture des fréquences les plus pertinentes. Les autres liens ne nous ont pas aidé sur le plan technique, en revanche on a remarqué que ces travaux étaient ajustés à des données trop parfaites, c'est-à-dire sans bruit et sans résonance des notes.
`https://master-studios.com/how-to-program-a-music-note-detector-in-python3-in-27-`
`https://medium.com/@ianvonseggern/note-recognition-in-python-c2020d0dae24/`

7 Conclusion

Ce projet a été très enrichissant, nous avons mieux compris la notion de détection de notes, tout en approfondissant nos connaissances sur les notions de Fourrier, domaines spectral et fréquences. Nous avons mis à dispositions des compétences mathématiques vu dans d'autres matières. Par ailleurs, ce code a une vraie utilité, ayant des musiciens dans le groupe le code développé nous permet de potentiellement créer des partitions de piano facilement, même s' il y a des limites lié aux harmoniques ou encore l'aspect temporelle entre chaque note.