École Centrale de Nantes

SCIMUS
Rapport

Sonification de micro-organismes

Élève :
Vincent Gallot

Encadrants :
Jean-François Petiot
Olivier Grovel

26 mars 2023

Table des matières

1 Présentation du projet

1.1 Objectifs du projet

Le projet de sonification de micro-organismes a pour objectif de lier sciences biologiques et sciences musicales en donnant une identité sonore à différentes espèces de champignons en fonction de leur composition moléculaire. La difficulté du projet vient de l'équilibre à trouver entre la qualité du son produit et à quel point ce son représente des vérités physiques. Ce projet n'a pas vocation à améliorer l'analyse de la structure moléculaire des champignons, mais plus à apporter un côté ludique à la visualisation des données et de permettre à des personnes non expertes de découvrir la diversité du vivant par la musique.

Le travail réalisé peut être trouvé sur GitHub :
https://github.com/VincentEnOptionRV/somicor

1.2 Obtention des données brutes

Les données que nous cherchons à sonifier dans ce projet proviennent de différentes espèces de champignons terrestres et aquatiques. Ces champignons sont analysés par spectrométrie de masse. La spectrométrie de masse est une technique d'analyse qui permet la détermination des masses moléculaires des composés analysés ainsi que leur identification et leur quantification. Son principe réside dans la séparation en phase gazeuse d'ions en fonction de leur rapport masse/charge (m/z). Un spectromètre de masse comporte une source d'ionisation suivie d'un ou plusieurs analyseurs qui séparent les ions produits selon leur rapport m/z, d'un détecteur qui compte les ions et amplifie le signal, et enfin d'un système informatique pour traiter le signal. Le résultat obtenu est une matrice qui représente dans le temps l'abondance relative de ces ions pour chaque rapport m/z, qui a la forme suivante :

$$\begin{pmatrix} a_{00} & \cdots & a_{0n} \\ \vdots & \ddots & \vdots \\ a_{p0} & \cdots & a_{pn} \end{pmatrix}$$

où a_{ij} représente l'abondance de l'ion de rapport $(m/z)_j$ au temps $t = i * \delta_t$, avec δ_t la résolution temporelle du spectromètre de masse de telle sorte que $n * \delta_t$ soit la durée totale de l'expérience et $(m/z)_0$ (resp. $(m/z)_p$) soit le plus petit (resp. grand) rapport m/z que le spectromètre ait enregistré.
Dans notre cas, on a les valeurs suivantes :

$\delta_t \approx 0.28$ s
$n = 5250$
$n * \delta_t = 35.00$ min

$(m/z)_0 = 100.110350$
$(m/z)_p = 978.594623$
$p = 824$

Ces données sont stockées dans un fichier csv qui, une fois importé dans un tableur, a la forme suivante :

	0	0.0083	0.015	0.0217	0.0283	0.035
100.110349745977	0	0	0	0	0	0
102.010985056559	0	0	0	0	0	0
102.032212189705	0	0	0	0	0	0
102.535130092076	0	0	0	0	0	0
109.012079753573	0	0	0	0	0	0
110.01796305807	0	0	0	0	0	0
110.522608757019	0	0	0	0	0	0
111.019336700439	0	0	0	0	0	0
111.087585449219	0	0	0	0	0	0
111.521388583713	0	0	0	0	0	0
112.015762329102	0	0	0	0	0	0
112.085187639509	0	0	0	0	0	0

FIGURE 1 – Données sous format "matrice".
Sur les lignes : les valeurs $(m/z)_j$ et sur les colonnes : les valeurs $t = i * \delta_t$.
Sur cette capture, tous les a_{ij} visibles sont nuls.

Afin de réduire la taille des données, on peut remplacer cette matrice par un vecteur qui, au lieu de détailler l'abondance à tout instant de l'expérience, ne va expliciter que l'abondance maximale qu'atteint un composé lors de l'expérience, ce qui donne un fichier de cette forme :

1,110.0203mz,33.73min	4.2378336693192 29E8
2,610.1809mz,31.62min	6.4338899483418785E7
3,536.1630mz,29.90min	3.386854857488172E7
4,110.0203mz,0.08min	1.16149323462431 25E8
5,612.1816mz,31.56min	2.6624892804937005E7
6,111.0205mz,33.40min	3.4199632638439 84E7
7,102.0343mz,33.76min	2.684549065744685E8
8,182.9852mz,33.66min	6.398189874599744E7
9,131.5335mz,33.40min	2.434192421914481E7
10,135.1016mz,0.97min	1.8132376817084417E7
11,136.1120mz,34.41min	2.2015390026075023E8
12,226.9507mz,33.27min	1.0567302963366274E7
13,164.1431mz,34.67min	1.817101935581247 5E8
14,119.0857mz,0.26min	8.133589593461065E7
15,538.1614mz,29.90min	1.1832913865853835E7
16,158.9640mz,33.27min	9597751.248339877

FIGURE 2 – Données sous format "vecteur".
On associe à chaque composé de rapport $(m/z)_j$ une abondance maximale a_j ainsi qu'un temps t_j auquel ce max est atteint.

1.3 Outils utilisés

Le projet a été réalisé en langage Python à l'aide de notebooks Jupyter. Comme les données sont des matrices sous format CSV, les bibliothèques standards de traitement et analyse des données ont été utilisées (`numpy, matplotlib, pandas`). À ces bibliothèques s'ajoutent des bibliothèques de synthèse et analyse sonore : `librosa, torchsynth, pysynth`.

2 Les différentes itérations du projet

2.1 Expérimentations avec les données

Au commencement du projet, les données que j'avais à ma disposition pour un échantillon consistaient uniquement en la version "vecteur" : on a donc pour chaque molécule détectée par le spectromètre de masse une abondance maximale et un temps associé. Ainsi, si on demande à `pandas` une description simple des données, on obtient le résultat suivant :

	Masse (mz)	Temps (min)	2905.mzML
count	953.000000	953.000000	2.570000e+02
mean	401.237266	12.775593	2.848970e+07
std	182.765170	9.082042	1.508230e+08
min	100.076200	0.070000	1.153957e+03
25%	251.090900	3.450000	6.486626e+03
50%	385.221100	12.790000	6.871151e+04
75%	529.268100	17.630000	3.114525e+06
max	799.563300	33.730000	1.377023e+09

FIGURE 3 – Description statistique des données brutes.
NB : Le nom de la 3e colonne correspond au nom de l'échantillon étudié.

Il y a plusieurs éléments notables dans cette description :

— En regardant la ligne *count*, on voit 953 masses différentes détectées, et 953 position temporelle de pics, mais il n'y a que 257 valeurs d'abondance. Cela s'explique par le fait qu'en réalité, les données de plusieurs échantillons sont agrégées, et on a donc des masses qui n'apparaissent pas dans cet échantillon, mais qui apparaissent dans d'autres. Ces masses ont des valeurs d'abondances valant `NaN` dans notre tableau, et il sera important de ne pas prendre en compte ces données dans notre sonification, au risque de réaliser des opérations interdites avec des `NaN`.

— En observant les différents quartiles des valeurs d'abondance, on se rend compte que l'échelle logarithmique conviendrait beaucoup mieux que l'échelle linéaire pour représenter ces données.

En appliquant un *log* à nos valeurs d'abondance, on se retrouve avec le tableau descriptif suivant :

	Masse (mz)	Temps (min)	2905.mzML
count	953.000000	953.000000	257.000000
mean	401.237266	12.775593	11.926353
std	182.765170	9.082042	3.511650
min	100.076200	0.070000	7.050952
25%	251.090900	3.450000	8.777498
50%	385.221100	12.790000	11.137672
75%	529.268100	17.630000	14.951587
max	799.563300	33.730000	21.043190

FIGURE 4 – Description statistique des données après application de *log* aux valeurs d'abondance.

Une fois qu'on a ces données, on peut essayer de les sonifier. Reste à savoir quelle méthode utiliser. S'il est assez logique de garder la correspondance entre le temps dans la spectrométrie et le temps dans le son généré, il faut choisir les grandeurs audio à associer à nos grandeurs physiques que sont le rapport m/z et l'abondance. Après quelques essais et discussions avec mes encadrants, il a été décidé que le son serait généré de la manière suivante : chaque masse détectée correspond à une note de musique, tel que la fréquence de la note est proportionnelle au rapport m/z, et l'amplitude sonore est proportionnelle à la valeur d'abondance. Cela revient à dire que l'on souhaite obtenir un spectrogramme qui est identique au graphe des rapport m/z en fonction du temps.

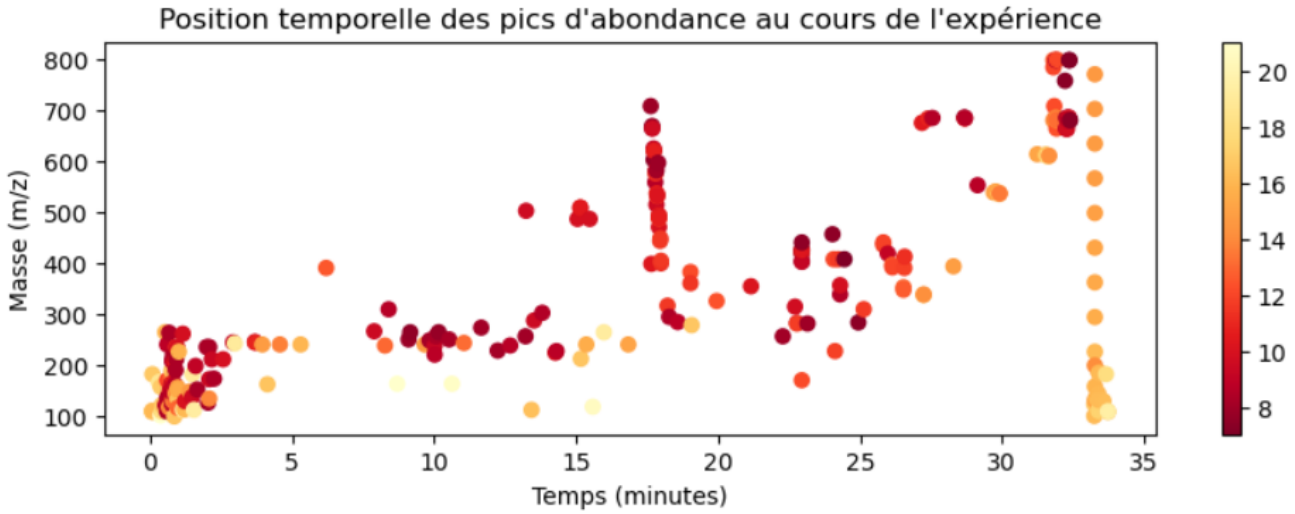

FIGURE 5 – Le graphe en question : un point correspond à un couple $((m/z)_j,t_j)$, et sa couleur est définie proportionnellement à $log(a_j)$

2.2 Fonction Python de sonification

Afin de rendre l'analyse du son plus simple, nous nous contenterons pour le moment de sons purs, c'est-à-dire que le son produit par une molécule est une sinusoïde de fréquence fixe. Cependant, avec le format de données que nous avons, il nous manque une information avant de pouvoir générer du son : on ne sait pas combien de temps la note doit durer. Puisque nous avons décidé plus tôt que l'amplitude du son est fonction de l'abondance de la molécule, la logique voudrait que l'amplitude varie selon l'abondance à chaque instant, mais ce n'est pas une information à notre disposition, car nous n'avons que les données des pics. Pour avoir ces données, il faudrait utiliser le format matriciel explicité plus haut, mais n'y ayant pas encore accès, j'ai décidé de considérer en première approximation que toutes les molécules suivent la même distribution normale.

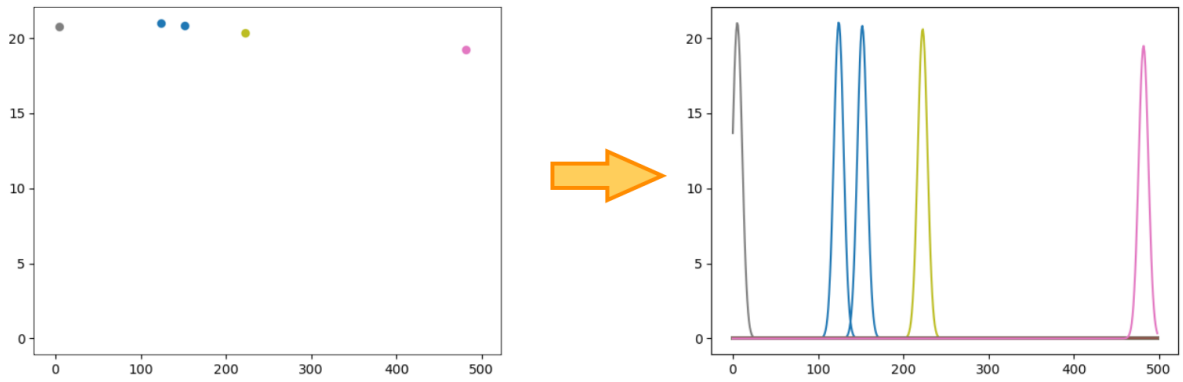

FIGURE 6 – Illustration du procédé d'approximation normale de l'abondance. L'abscisse représente le temps et l'ordonnée l'amplitude.

Si on applique ce procédé à toutes nos molécules on obtient une "matrice d'abondance", qui représente l'abondance de chaque molécule en fonction du temps.

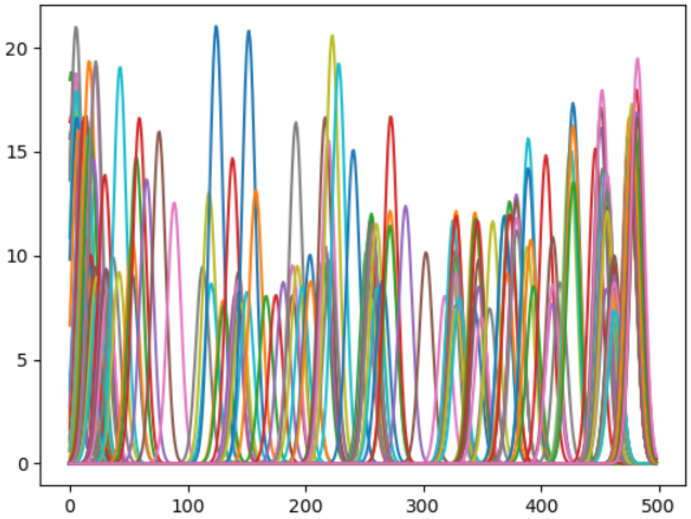

FIGURE 7 – Graphe de la matrice d'abondance d'un échantillon.

Forts de cette matrice d'abondance, il est possible de commencer à générer du son. Pour ce faire, on va prendre individuellement chaque molécule et lui associer une fréquence en fonction de sa masse, puis générer une sinusoïde de la forme $a_{tj}sin(\pi f_j t)$ avec j l'indice de la colonne de la molécule dans la matrice, a_{tj} l'abondance de cette molécule à l'instant t, et f_j la fréquence associée à la molécule. Ce faisant, on obtient autant de vecteurs qu'il y a de molécules dans l'échantillon, et on peut ensuite faire la moyenne de ces vecteurs pour obtenir le vecteur "son", qui contient des valeurs de pression acoustique. On peut ensuite utiliser la fonction `Audio` de la bibliothèque `IPython.Display` pour transformer ce dernier en fichier audio. Le code pour accomplir cela est le suivant :

```
sr = 11000 # sampling rate (frequence d'echantillonage)
duration = 10 # secondes
mass_vector = data["Masse (mz)"].loc[data["2905.mzML"] > 0].to_numpy()


# Structure d'entree attendue : vecteur taille m et matrice taille n*m avec :
# n = duree_experience / resolution_temporelle = nombre de releves pendant l'experience
# m = nombre de molecules distinctes (Rapport m/z distinct)
def son(mass_vector, abundance_matrix, sr, duration):
    n = len(abundance_matrix)
    m = len(mass_vector)
    durationOfOneSample = duration*sr/n

    # On ramene l'amplitude dans ]0,1]
    amplitude_matrix = abundance_matrix / np.max(abundance_matrix)

    # Chaque colonne est une molecule differente, chaque ligne est une unite temporelle
    sounds = np.zeros((duration*sr,m))

    # Parcours par temps croissant (on suppose la matrice d'abondance triee)
    for i in tqdm.tqdm(range(duration*sr)):
        sampleNumber = int(i/durationOfOneSample) # padding sans interpolation
        t = i/sr # temps

        # Parcours de toutes les molecules pour un instant donne
        for j in range(m):
            freq = mass_vector[j] # frequence
            amp = abundance_matrix[sampleNumber][j] # amplitude
            sounds[i][j] = amp * np.sin(np.pi*freq*t)

    # Fusion des channels de toutes les molecules
    return np.mean(sounds,axis=1)


audio = son(mass_vector, abundance_matrix, sr, duration)
display(ipd.Audio(audio, rate=sr))
```

Pour cette version, on a choisi la fréquence associé à une molécule exactement égale à son ratio m/z, qui est compris entre 100 et 900 pour notre échantillon.

Remarque - Il est à noter que le code précédent ainsi que les codes qui suivront dans ce rapport sont très peu efficaces. Par exemple, pour générer un son de 10 secondes, l'exécution de ce code prend 50 secondes sur un processeur Intel Core i5-8400. Ce code a été rédigé dans un but explicatif, et pour chaque version de la fonction de sonification qui sera présentée dans ce document il existe une version optimisée à l'aide de la bibliothèque `numpy` (mais beaucoup moins lisible). Pour cet exemple, sur la même machine, la version optimisée de cette fonction ne prend que 500 ms d'exécution.

Si on observe la forme du son généré avec cette fonction, on remarque que cette dernière suit la courbe de moyenne des colonnes de la matrice d'abondance. C'est bon signe, puisqu'on souhaite que l'amplitude sonore soit fonction de l'abondance.

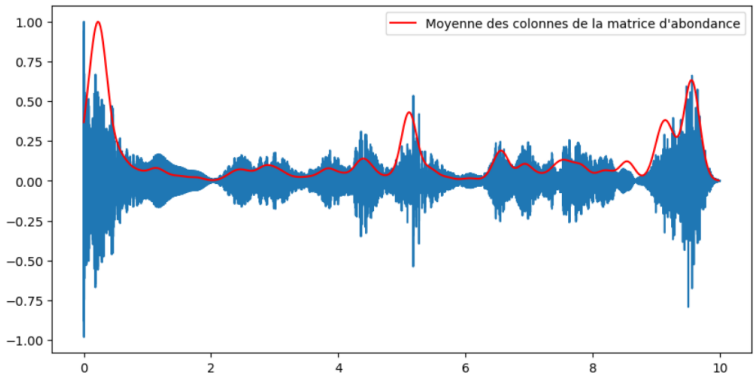

FIGURE 8 – Profil de pression du son généré

En effectuant une analyse spectrale du son, on observe qu'on a bien ce qu'on souhaitait initialement : un spectre identique au graphe des pics en fonction du temps (voir Figure 5 pour la comparaison).

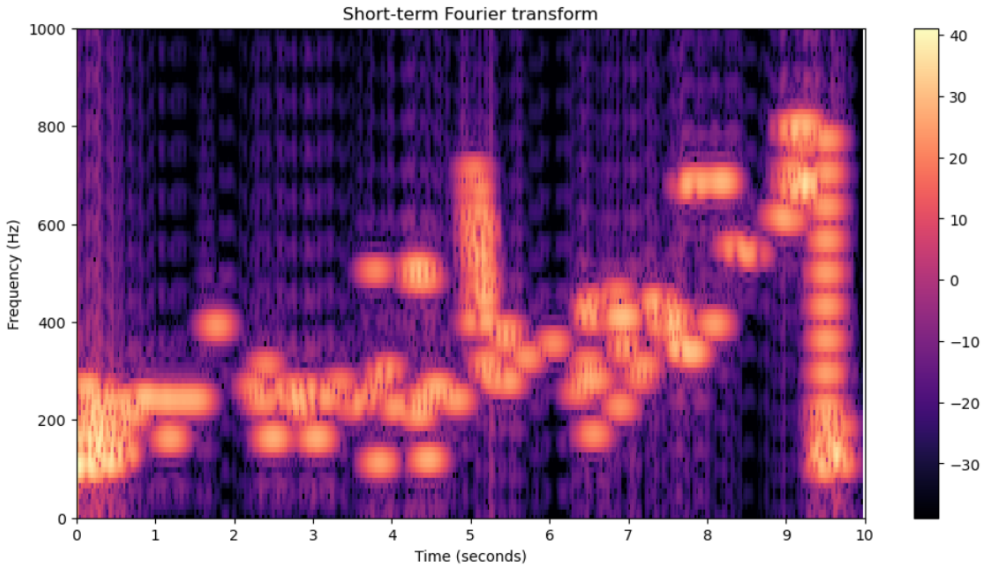

FIGURE 9 – Spectrogramme du son généré.

Cependant, on remarque également la présence d'un bruit important. Ce bruit est une conséquence de la ligne `23` du code : sans interpolation, il y a existence de "marches" d'amplitude sonore, que l'on peut observer si on effectue un zoom sur le profil de pression.

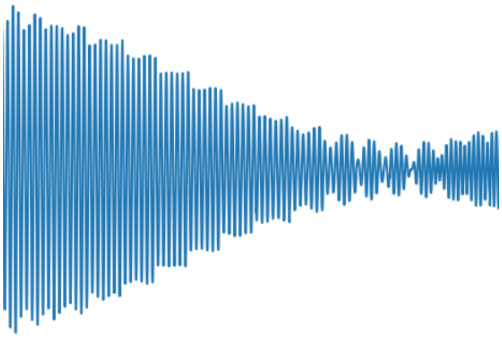

FIGURE 10 – Un exemple des "marches" d'amplitude, conséquence de l'absence d'interpolation.

Autre problème de ce son : Il contient trop d'informations. Sur cet exemple, l'échantillon contient 257 molécules distinctes, et le son généré dure 10 secondes. Cela fait presque 26 notes par seconde, sachant que chacune des 257 notes est jouée à une fréquence distincte.

Ainsi, il a fallu mettre en place une seconde version de cette fonction de sonification. Cette deuxième fonction implémente une interpolation afin d'éliminer le bruit sur le spectrogramme, mais ne résout pas par elle même le problème de la surcharge d'information. Ce dernier est résolu par la fonction qui génère la matrice d'abondance à partir des pics : en ajoutant un argument supplémentaire `n` à la fonction, on va pouvoir limiter le nombre de pics aux `n` pics les plus importants.

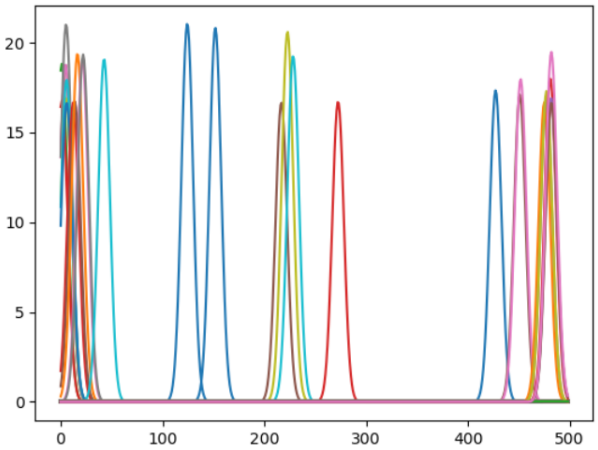

FIGURE 11 – Exemple d'une matrice d'abondance limitée aux 30 plus grands pics.

On obtient le code suivant :

```
def son(mass_vector, abundance_matrix, sr, duration):
    n = len(abundance_matrix)
    m = len(mass_vector)
    durationOfOneSample = duration*sr/n

    # On ramene l'amplitude dans ]0,1]
    amplitude_matrix = abundance_matrix / np.max(abundance_matrix)

    # Chaque colonne est une molecule differente, chaque ligne est une unite temporelle
    sounds = np.zeros((duration*sr,m))

    # Parcours par temps croissant (on suppose la matrice d'abondance triee)
    for i in tqdm.tqdm(range(duration*sr)):

        ### Mise a jour de la fonction : Interpolation ###
        sampleNumber = int(i/durationOfOneSample)
        if sampleNumber < len(abundance_matrix) - 1:
            sampleProgress = i%durationOfOneSample
            sampleCurrent = abundance_matrix[sampleNumber]
            sampleNext = abundance_matrix[sampleNumber+1]
            sampleInterpolation =
                (sampleNext-sampleCurrent)/durationOfOneSample
                * sampleProgress + sampleCurrent
        else:
            sampleInterpolation = sampleCurrent
        ### Fin de la mise a jour ###

        t = i/sr # temps

        # Parcours de toutes les molecules pour un instant donne
        for j in range(m):
            freq = mass_vector[j] # frequence
            amp = abundance_matrix[sampleNumber][j] # amplitude
            sounds[i][j] = amp * np.sin(np.pi*freq*t)

    # Fusion des channels de toutes les molecules
    return np.mean(sounds,axis=1)
```

Une inconsistance supplémentaire a été corrigée dans cette deuxième version : on a "inversé" les masses. En effet, dans la première version, les molécules avec une masse élevée se voyaient attribuer une fréquence élevée, mais il est plus naturel que cela soit l'inverse.

Avec le même échantillon qu'auparavant, mais en utilisant cette nouvelle fonction ainsi qu'en limitant à 30 pics la matrice d'abondance, on obtient un nouveau spectrogramme, qui n'est plus bruité et sur lequel on peut observer les effets de la limitation du nombre de pics.

FIGURE 12 – Spectrogramme obtenu après génération du son par la deuxième fonction.

A l'issue de cette étape, nous avons donc un moyen de sonifier les données, mais le résultat n'est pas parfait : nous n'utilisons que des sons purs, ce qui ne donne pas un résultat très agréable, et nous avons fait une *très* grossière approximation sur la distribution des molécules au cours de l'expérience.

Afin de corriger ce second point, j'ai eu accès à la version matricielle des données du spectromètre de masse, afin de pouvoir obtenir une matrice d'abondance qui représente mieux la réalité chimique des échantillons.

2.3 Changement du format de données

La version matricielle des données contient les donnés brute du spectromètre. À l'idée de travailler avec ces données, j'imaginais qu'il suffirait simplement d'envoyer ces données en tant que matrice d'abondance dans la fonction de sonification, mais je me suis heurté à un problème : les données sont trop précises pour mon utilisation. En effet, le spectromètre permet de différentier plusieurs isotopes de la même molécule, et est donc extrêmement précis dans sa mesure de rapport m/z. Ainsi, si on observe les courbes d'abondance pour différentes valeurs de m/z, on obtient le résultat suivant :

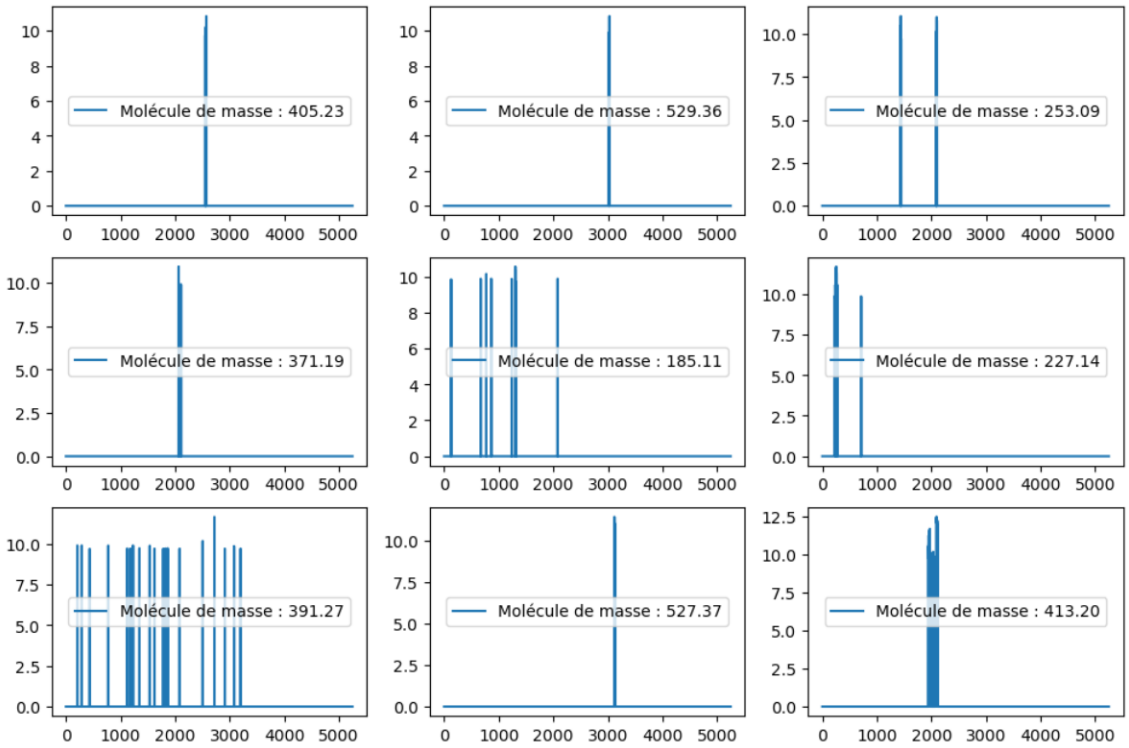

FIGURE 13 – Courbes d'abondances pour certaines valeurs de m/z.

En fait, avec une telle précision, la même molécule correspond à plusieurs valeurs de m/z. Il faut donc effectuer un traitement préalable des données afin de regrouper les masses correspondant à la même molécule. Entre nos données brutes et la matrice d'abondance, il y a trois étapes de traitement :

- Regrouper les valeurs de m/z. Pour ce faire, il suffit de choisir une valeur `n` et d'additionner les masses par groupes de taille `n`. Nous avons choisi `n = 10`.
- Adoucir les données en effectuant une convolution entre nos données et une fenêtre pour permettre un moyennage. J'ai fais le choix de convoluer les données avec une fenêtre de Blackman (disponnible dans la bibliothèque `numpy`) à 200 points. C'est une valeur très importante qui aura pour effet de beaucoup étirer nos données, mais cela est nécessaire si on veut que la note jouée dure plus de quelques dixièmes de seconde.
- Ne sélectionner qu'un certain nombre de molécules, pour éviter une surcharge d'information. À cette occasion, la sélection a été raffinée, et il est possible de choisir selon différents ordres : "plus abondantes", "plus lourdes", et "plus légères".

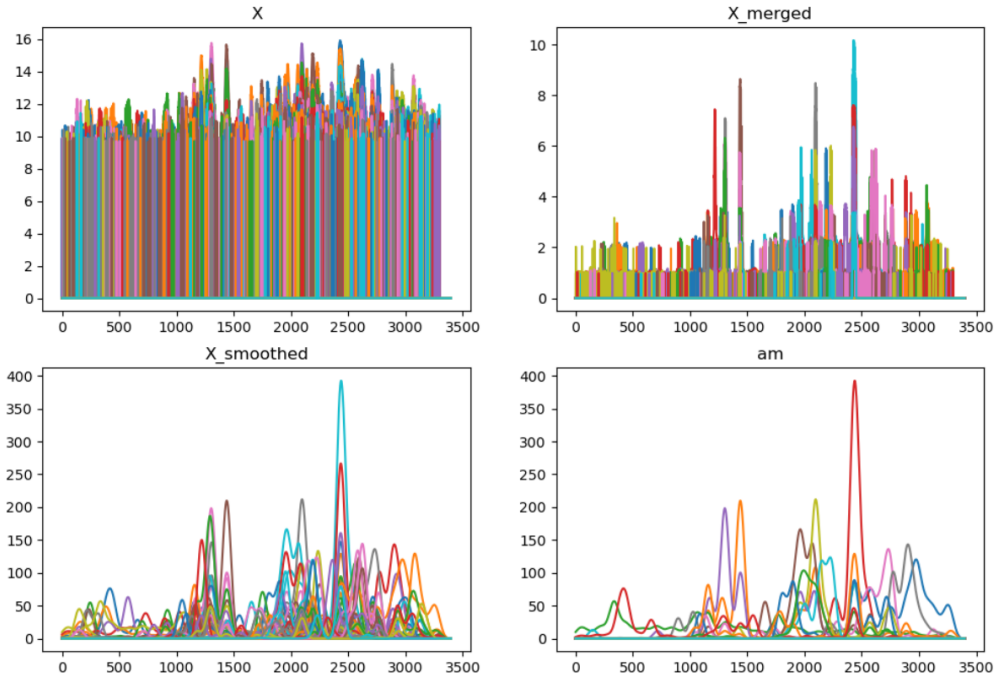

FIGURE 14 – De gauche à droite, du haut vers le bas : données brutes, données après fusion par groupes de 10, données après adoucissement, sélection des 14 molécules les plus abondantes

De plus, dans un premier pas vers un son plus agréable, la sélection de la fréquence ne se fait plus en prenant l'exacte valeur de la masse de la molécule, mais en créant une correspondance entre les masses et les fréquences des notes de piano.

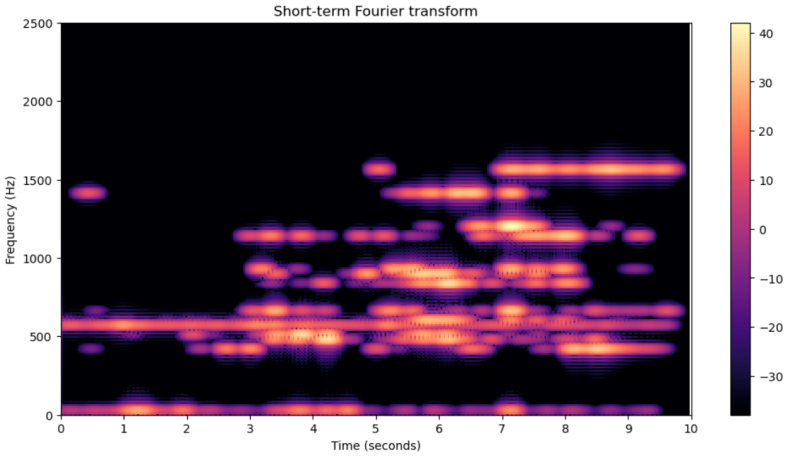

FIGURE 15 – Spectrogramme du son généré.

2.4 Amélioration du son produit

Jusqu'ici, on a uniquement travaillé sur des sons purs. Mais l'idéal serait d'entendre le son d'un instrument. Il y a alors deux possibilités : effectuer le travail de synthèse sonore pas soi-même ou se tourner vers des bibliothèques Python existantes. La deuxième solution est bien souvent plus simple, mais encore faut-il trouver la bibliothèque adaptée. Après en avoir testé quelques unes (`Torchsynth`, `Pysynth`, `Tones`), il est apparu que je n'aurais pas le temps d'adapter ma structure de données aux structures de ces bibliothèques, et j'ai décidé de le faire par moi-même.

Ainsi, pour améliorer le son, j'ai changé de stratégie de génération de la matrice d'abondance. Au lieu d'utiliser directement la matrice obtenue dans la partie précédente, on va effectuer un seuillage de nos valeurs : on va considérer chaque molécule comme une note, qui est pressée si la valeur d'abondance est au dessus du seuil, et relâchée sinon. On associe la force de presse de la note à l'amplitude maximale du pic considéré. Avec cette méthode, il est possible de "jouer" de n'importe quel instrument à partir de nos données, puisqu'on sait maintenant quand jouer et relâcher les notes.

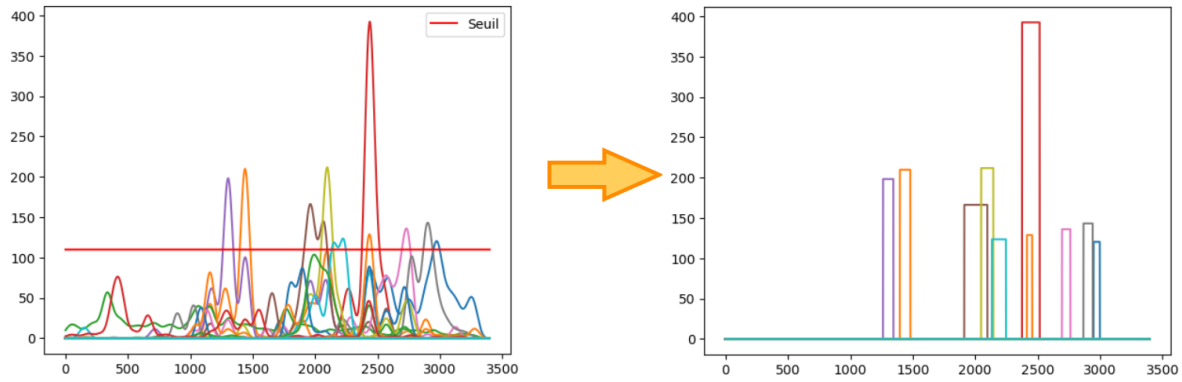

FIGURE 16 – Application du processus de seuillage avec un seuil à 30%

Afin de simuler un véritable instrument, il faut transformer cette enveloppe brute en une enveloppe ADSR (Attack, Decay, Sustain, Release). Selon les valeurs que l'on donnera à cette enveloppe, on pourra simuler différents instruments.

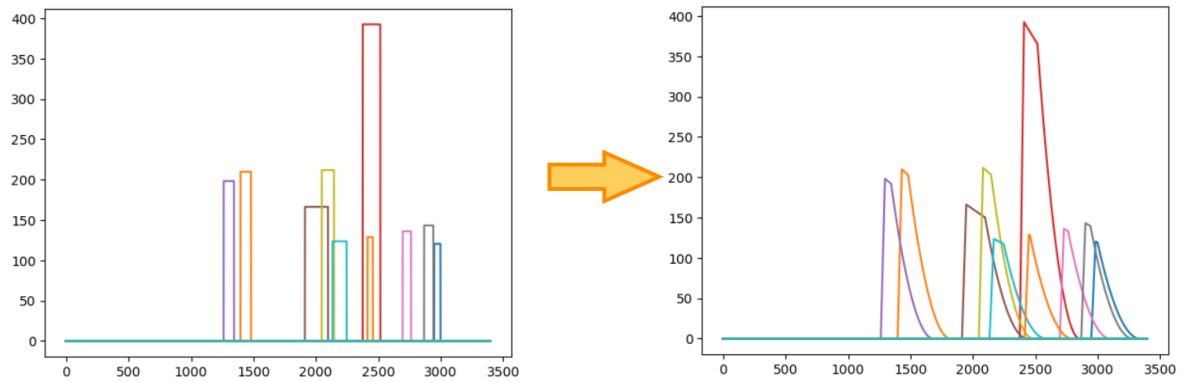

FIGURE 17 – Transformation en une enveloppe ADSR pour chaque note. Ici, on considère un son de 10s avec un temps d'attaque de 0.01s, un temps de decay de 9 secondes, une valeur de sustain à 0 et un temps de release de 1s. On se rapproche avec ces valeurs d'une enveloppe ADSR de piano.

Cette matrice des enveloppes est ce qui va remplacer notre matrice d'abondance dans la fonction de sonification, puisque c'est elle qui va donner l'amplitude de notre son

En plus de l'enveloppe ADSR, nous allons donner plus d'harmoniques à notre son pur. Il y a deux manières de procéder. La première consiste à rajouter "à la main" ces harmoniques supplémentaires, avec une fonction d'atténuation de ces harmoniques à définir. On peut simplement définir une atténuation en $\frac{1}{x}$ ou $\frac{1}{x^2}$, mais si on souhaite être plus précis, on peut aussi prendre en compte le fait que les harmoniques sont atténuées différemment selon l'intensité de la note, de telle sorte que les notes jouées plus fortes sont plus brillantes. La deuxième méthode consiste à utiliser une autre fonction de base que la sinusoïde pour faire notre synthèse, et d'utiliser par exemple une fonction carrée ou triangle.

Reste à choisir les valeurs : celles de l'enveloppe ADSR et celles de l'atténuation des harmoniques (dans le cas de la première méthode). Ici, cela revient à de l'essai et erreur pour trouver une combinaison de valeurs qui "sonne bien", car il a été surprenamment difficile de trouver des ressources à ce sujet. Par manque de temps, les résultats de cette phase d'essai et erreur sont assez moyens, mais le gros du travail est fait, et il suffit de régler ces paramètres pour obtenir un meilleur son.

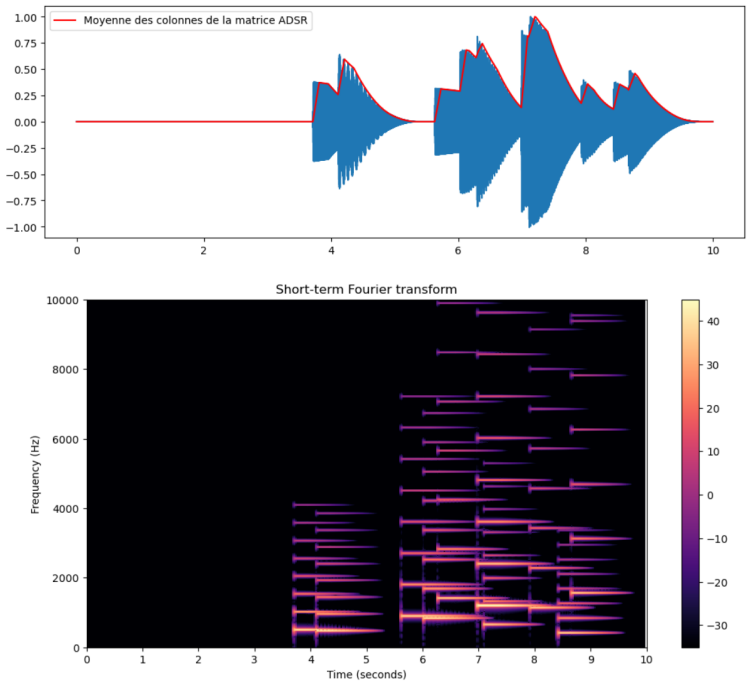

FIGURE 18 – Profil de pression et spectrogramme du son synthétisé à partir de la matrice ADSR obtenue Figure 17. Ici, on a choisi d'ajouter 10 harmoniques avec une atténuation en $\frac{1}{x^2}$.

Note sur la donnée temporelle - Pour que le son soit plus agréable, ainsi que pour faciliter l'utilisation des bibliothèques de synthèse sonore, il aurait été possible de transformer nos données en partition pour envoyer cette dernière vers une des-dites bibliothèques. Cependant, la précision temporelle des données serait perdue, car on devrait s'aligner sur la temporalité d'une partition. Cela nous ramène à l'aspect principal de ce projet : trouver un équilibre entre la réalité physique et la représentation sonore.

3 Conclusion et perspectives

Ce projet a été centré autour de la mise en place d'une synthèse sonore dans le but de sonifier des données obtenues dans le cadre d'études biochimiques. En ce sens, le projet est une réussite, car il permet de sonifier de différentes manières, avec différents types de données et d'approximations. De plus, il s'agit d'un projet autonome qui ne nécessite pas de bibliothèques tierce autre que les ultra-classiques de la Data Science. Le projet a évidemment ses limites, qui sont assez clairement définies par le fait que le son synthétisé n'est pas aussi agréable que ce qu'on aurait pu espérer. Il s'agit là de la piste d'amélioration principale qu'il serait possible d'apporter au projet.

Pour ce qui est de mon expérience personnelle avec ce projet, je l'ai trouvé très intéressant dans la liberté des approches qu'il offre : il n'y a pas de bonne ou mauvaise réponse quand on cherche à sonifier des données, seulement des approches différentes. Les problématiques d'équilibre entre réalité et représentation m'ont toujours intéressé, et si j'ai pu explorer ces problématiques de manière visuelle en option Réalité Virtuelle l'an dernier, j'ai pu cette année explorer le côté auditif de ces questions.

Comme souvent, le manque de temps et/ou la mauvaise gestion de ce dernier s'est fait sentir en fin de projet, mais je considère que la partie principale a été réalisée avec succès, et nous avons pu écouter ces champignons, même si leur voix n'était pas la plus plaisante.