

INSA STRASBOURG

SIGNAL S8

Compte Rendu Final du Mini-Projet

Cacher une image dans une chanson

Auteurs :

CARTIER MILLON Damien
DOPPLER Luc

Encadrant :

M. CHEVEREAU Guillaume

18 avril 2021

Sommaire

0	Préambule	3
1	Introduction	3
2	État de l'art	4
2.1	Utilisation des spectrogrammes	4
2.2	Logiciels grand public permettant de réaliser un spectrogramme	5
2.3	Type de fenêtrage	5
2.3.1	Pondération Gaussienne	5
2.3.2	Pondération de Bartlett	6
2.3.3	Pondération de Hann	6
2.3.4	Autres pondérations	6
3	Objectifs du projet	7
3.1	Livrables	7
3.2	Livrables bonus	7
4	Travail réalisé	8
4.1	Méthode de travail	8
4.1.1	Hiérarchisation des objectifs chronologie du projet	8
4.1.2	Politique de test	8
4.1.3	Utilisation de Git et Github	8
4.1.4	Commentaires et fichier README.md	8
4.2	Livrés	8
4.3	Schémas de principe	9
4.3.1	Réalisation d'un spectrogramme sautant	9
4.3.2	Réalisation d'un spectrogramme glissant	9
4.3.3	Incrustation d'une image en nuances de gris	9
4.3.4	Incrustation d'une image en couleurs	10
5	Résultats, analyse des performances et analyse critique	11
5.1	Comparaison des différents spectrogrammes obtenables	11
5.1.1	Influence de la pondération	11
5.1.2	Influence de la méthode utilisée	13
5.1.3	Influence de la fenêtre utilisée pour la transformée de Gabor	14
5.2	Incrustation de l'image dans le signal	15
5.2.1	Concept	15
5.2.2	Vérification de l'incrastation	15
5.3	Incrustation d'une image RGB	16

6	Limites et améliorations possibles	19
6.1	Limites du projet	19
6.2	Améliorations d'ergonomie	19
6.3	Amélioration de la pondération	19
6.4	Améliorations liées à des aspects de programmation	19
7	Conclusion	21

0 Préambule

L'ensemble de notre répertoire de projet est disponible au lien suivant :

<https://github.com/DamienCM/Projet-S8-Signal/>

1 Introduction

Le but de ce mini-projet est l'incrustation d'une image dans une chanson. On peut également voir cela comme la dissimulation d'une image dans un fichier audio.

Ce projet va nous permettre de mettre en oeuvre les outils et compétences acquis au cours de la première partie du cours de signal. Nous utiliserons principalement les transformées de Fourier et les transformées de Gabor (transformées de Fourier glissantes).

Ayant tous deux déjà des expériences avec Python, nous avons décidé de privilégier Python à Matlab.

Parmis les diverses fonctionnalités à implémenter, nous devrons convertir un fichier audio en un signal, analyser ce signal à l'aide d'une transformée de Fourier, convertir une image en un signal (une matrice à 2 dimensions), intégrer cette image dans un signal audio et représenter le spectrogramme d'un signal audio contenant ou non une image ajoutée par nos soins.

Les fichiers, librairies, scripts, fonctions ... utilisés sont décrits dans le fichier *README.md*. Les arguments en entrée des fonctions et les sorties des fonctions sont détaillés dans les fichiers Python lors de leur déclaration.

2 État de l'art

2.1 Utilisation des spectrogrammes

Contrairement à la transformée de Fourier qui donne une représentation sur toute la durée du signal (donc une représentation uniquement fréquentielle), le spectrogramme utilise la transformée de Gabor ou transformée de Fourier glissante et permet d'obtenir une représentation temps-fréquence. Cela permet de connaître les composantes fréquentielles d'un signal en fonction du temps.

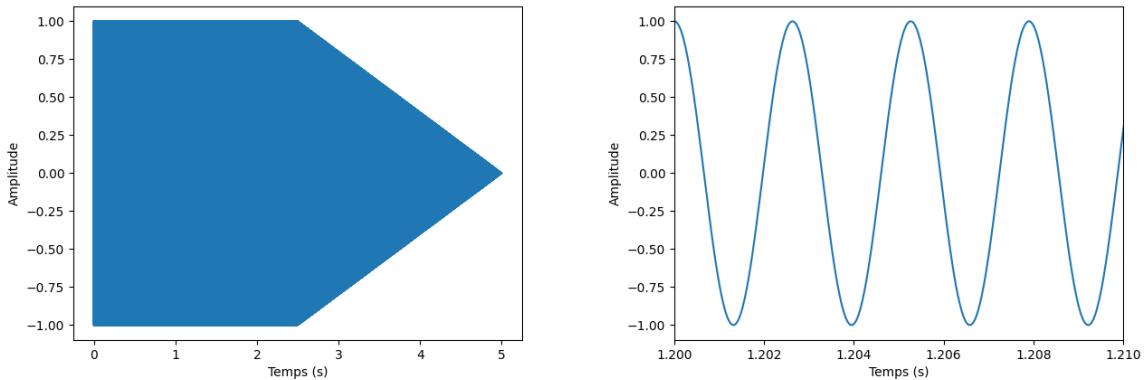


Figure 1: Représentation temporelle du Signal

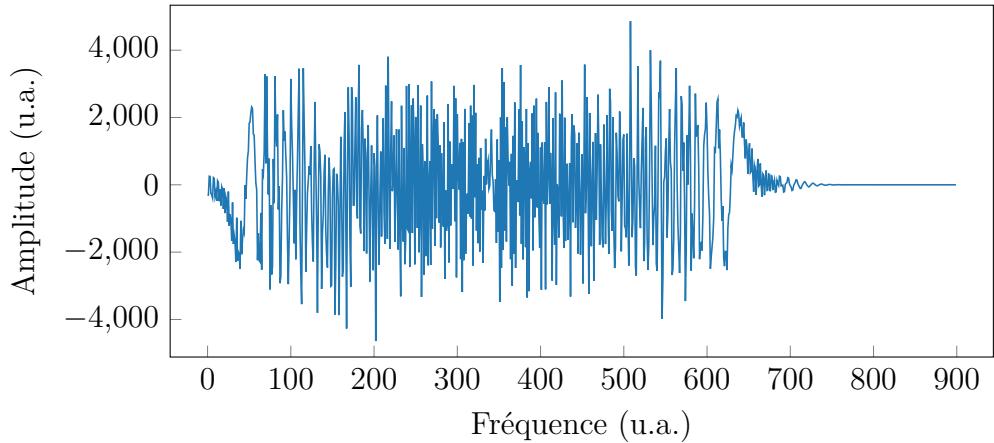


Figure 2: Représentation fréquentielle

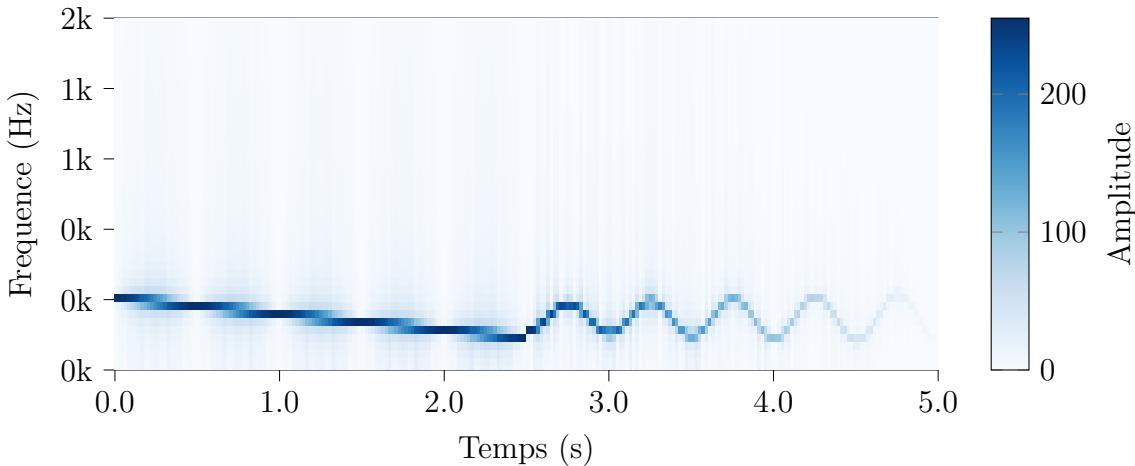


Figure 3: Spectrogramme du son (Représentation temps-fréquence)

2.2 Logiciels grand public permettant de réaliser un spectrogramme

Bien qu'il soit aisément possible de réaliser un spectrogramme avec MATLAB, Octave ou en Python, ces langages ne sont pas maîtrisés par le grand public et ils ne disposent pas d'une interface graphique.

Si une personne lambda souhaite réaliser un spectrogramme, que ce soit pour vérifier qu'elle chante juste, pour visualiser les *Demon Face* dans les morceaux d'*Aphex Twins* ou tout simplement pour le plaisir il est possible de le faire à l'aide de logiciels spécialisés comme Audacity ou Adobe Audition.

L'utilisation d'Audacity nous permettra de vérifier la présence d'une image dans un son et donc le bon fonctionnement de notre code.

2.3 Type de fenêtrage

La transformée de Gabor utilise une fenêtre glissant sur le signal. Les caractéristiques de cette fenêtre ont un impact direct sur la qualité du spectrogramme.

L'utilisation d'une fenêtre rectangulaire (*i.e.* une porte) génère des artefacts graphiques qui ne sont pas souhaités. Afin de limiter ces artefacts, il est possible de pondérer le signal sur la fenêtre de temps.

Pour cela, le signal à étudier est fenêtré à l'aide d'un porte puis pondéré à l'aide d'une fonction de pondération. Plusieurs types de pondérations sont possibles.

2.3.1 Pondération Gaussienne

C'est une pondération effectuée à l'aide d'une distribution Gaussienne.

2.3.2 Pondération de Bartlett

Aussi appelée fenêtre triangulaire.

$$h(t) = \begin{cases} \frac{2t}{T} & \text{si } t \in [0, \frac{T}{2}[\\ \frac{2(T-t)}{T} & \text{si } t \in [\frac{T}{2}, T[\\ 0 & \text{sinon} \end{cases} \quad (1)$$

2.3.3 Pondération de Hann

$$h(t) = \begin{cases} \frac{1}{2} - \frac{1}{2} \cos\left(2\pi \frac{t}{T}\right) & \text{si } t \in [0, T[\\ 0 & \text{sinon} \end{cases} \quad (2)$$

2.3.4 Autres pondérations

Il existe de nombreuses autres pondérations comme celles de Hamming, Blackman, Kaiser, Flat Top, en cosinus relevé,...

3 Objectifs du projet

3.1 Livrables

Les livrables définis en début de semestre sur Moodle sont les suivants:

- Un script qui prendra en entrée une image grayscale et un fichier musical et qui renvoie un fichier musical avec l'image incrustée dans le spectrogramme
- Un outil de visualisation du spectrogramme qui met en évidence l'image cachée dans le fichier musical
- Une analyse de l'intérêt et des limites de la méthode utilisée
- Des propositions d'améliorations

Les livrables bonus sont:

- Encoder une image couleur ou un film
- Proposer une façon d'encoder l'image sans qu'elle soit directement visible dans le spectrogramme

3.2 Livrables bonus

Afin d'ajouter un peu de saveurs et de difficultés au projet, nous avons décidé de réaliser l'un des livrables bonus, à savoir l'encodage d'une image RGB dans le spectrogramme.

4 Travail réalisé

4.1 Méthode de travail

4.1.1 Hiérarchisation des objectifs chronologie du projet

Jusqu'au premier rendu notre objectif était la programmation des fonctions permettant de récupérer le son, de le traiter puis de tracer le spectrogramme du son. Nous avions également débuté le travail de mise en forme des images comme la transformation en nuances de gris, le redimensionnement, ...

Une fois ces outils programmés et testés, nous nous sommes concentrés sur le cœur de notre projet, à savoir l'incrustation d'une image dans le spectrogramme d'un fichier audio.

Finalement, les objectifs plus poussés comme le spectrogramme glissant et l'utilisation d'images en couleurs ont été réalisés.

4.1.2 Politique de test

Afin de contrôler le bon fonctionnement de nos algorithmes tout au long du projet, nous avons réalisé des tests unitaires réguliers lors de la conception de chacune de nos fonctions. De plus, une fois les fonctions fonctionnelles, nous avons réalisé des tests d'intégration pour vérifier le bon fonctionnement des fonctions entre elles.

De plus, pour limiter les temps de test et pour avoir un signal dont le spectrogramme nous était connu, nous avons principalement utilisé le son *filsDuVoisin.wav* pour les premiers tests.

4.1.3 Utilisation de Git et Github

L'utilisation de Git et Github nous a permis de travailler à distance et en parallèle sur le code. De plus, la gestion des versions permet de garder un historique de l'ensemble du projet.

4.1.4 Commentaires et fichier README.md

Toutes les fonctions ont été documentées sous leur définition. Sont indiqués les paramètres d'entrée, les valeurs en sortie et le rôle de la fonction dans le projet.

Le fichier README.md donne les informations sur le rôle de chacun des fichiers ainsi que les fonctions qu'il contient. Pour le détail d'une fonction, il est nécessaire de se référer à la documentation réalisée lors de la définition de cette dernière.

4.2 Livrés

Les éléments livrés à la fin de ce mini-projet sont les suivants:

- un script python *spectrogram.py*: la librairie principale du projet dans laquelle sont contenues les fonctions utilisées
- un script *demo_incrustation.py* incrustant une image en nuances de gris dans un son. Ce dernier est sauvegardé dans le dossier audio.
- un script *demo_rgb.py* incrustant une image en couleur dans un son. Ce dernier est également sauvegardé dans le dossier audio.
- un script *comparaison_sautant_glissant.py* qui affiche le spectrogramme sautant et le spectrogramme glissant d'un son afin de comparer les performances.

De plus, des sons de test, des exemples de son reconstitués, des images et des spectrogrammes sont sauvegardés dans les dossiers *images*, *plots* et *audio*.

4.3 Schémas de principe

4.3.1 Réalisation d'un spectrogramme sautant

Pour réaliser un spectrogramme sautant, une fenêtre (porte) se déplace sur le signal sans recouvrement. Pour chaque fenêtre, la transformée de Fourier discrète est calculée à l'aide de la fonction FFT. Toutes ces transformées de Fourier sont concaténées en une matrice qui, une fois affichée correspond au spectrogramme sautant.

Une pondération peut être effectuée sur la fenêtre afin de limiter les artefacts présents sur le spectrogramme.

$$\text{Audio} \xrightarrow[\text{sautante}]{\text{Porte}} \xrightarrow{\text{Pondration}} \xrightarrow{\text{FFT}} \text{Matrice} \xrightarrow[\text{Norme}]{\text{Plot}} \text{Spectrogramme}$$

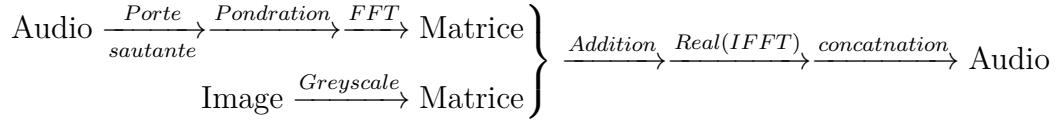
4.3.2 Réalisation d'un spectrogramme glissant

Pour réaliser le spectrogramme glissant, le principe est similaire. Mais la porte glisse sur le signal et les fenêtres se recouvrent. Comme pour le spectrogramme sautant, les transformées de Fourier sont calculées, concaténées puis affichées pour avoir le spectrogramme glissant.

$$\text{Audio} \xrightarrow[\text{glissante}]{\text{Porte}} \xrightarrow{\text{Pondration}} \xrightarrow{\text{FFT}} \text{Matrice} \xrightarrow[\text{Norme}]{\text{Plot}} \text{Spectrogramme}$$

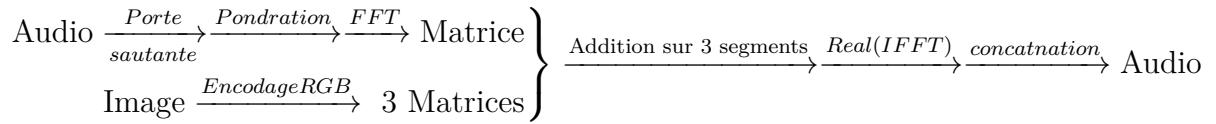
4.3.3 Incrustation d'une image en nuances de gris

Pour incruster une image dans le spectrogramme d'un son, il faut incruster l'image dans la matrice des FFT glissantes/sautantes. Pour cela, on transforme l'image en une matrice de nuances de gris. Cette dernière sera ajoutée à la matrice des FFT. Finalement pour chaque colonne de cette matrice, il faudra concaténer la partie réelle de la transformée de Fourier inverse. Grâce à cela on obtient un nouveau son dont le spectrogramme contient l'image.



4.3.4 Incrustation d'une image en couleurs

L'incrustation d'une image en couleur est différente mais suit le même principe. Dans un premier temps, l'image est décomposée en 3 matrices correspondant aux couleurs RGB (Red, Green, Blue). Ces 3 matrices sont ajoutées à la suite, chacune sur un tiers du signal.



5 Résultats, analyse des performances et analyse critique

5.1 Comparaison des différents spectrogrammes obtenables

5.1.1 Influence de la pondération

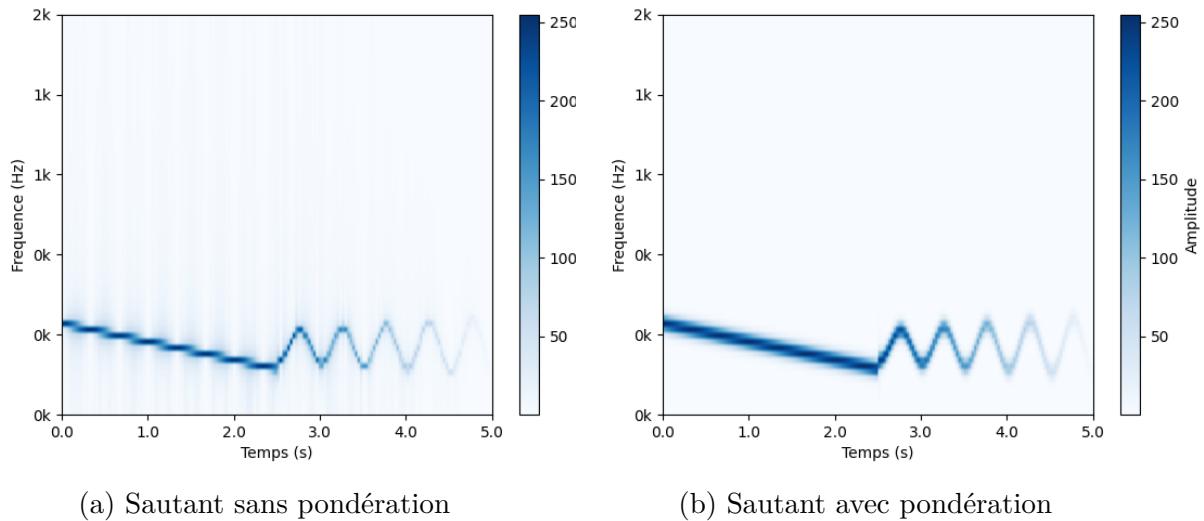


Figure 4: Comparaison des spectrogrammes pour un pas de $t = 0.03\text{ s}$

L'utilisation de la pondération permet de limiter voire supprimer les artefacts présents sur le spectrogramme. Comme il s'agit d'une simple multiplication à effectuer, le temps de calcul n'est que très peu impacté par l'application d'une pondération.

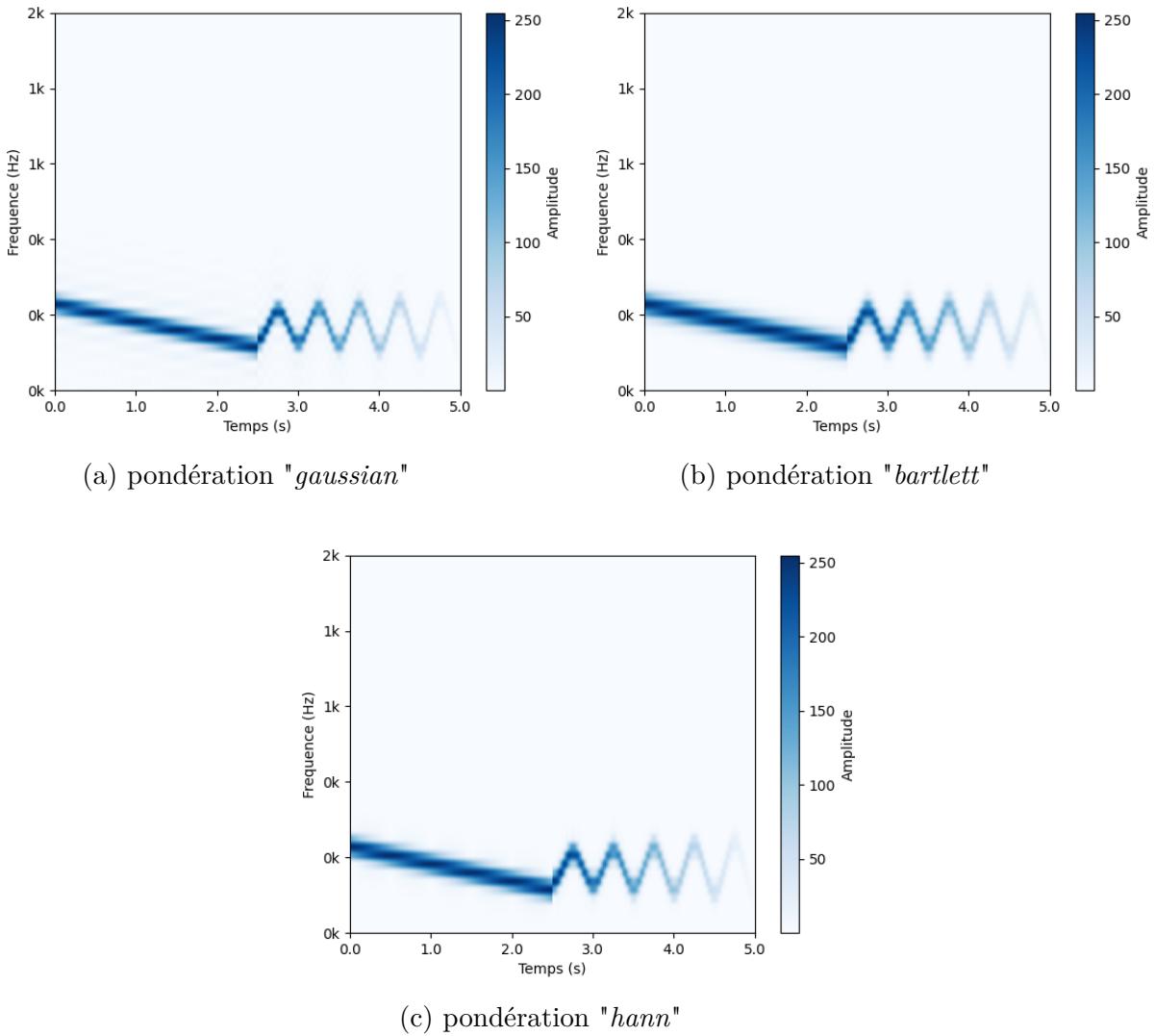


Figure 5: Comparaison des différentes pondération utilisables pour un pas de temps de $t = 0.02\text{ s}$

Nous pouvons constater qu'il n'y a pas de différence significative entre les 3 types de pondération. Ainsi dans la suite nous utiliserons par défaut la pondération "gaussian".

5.1.2 Influence de la méthode utilisée

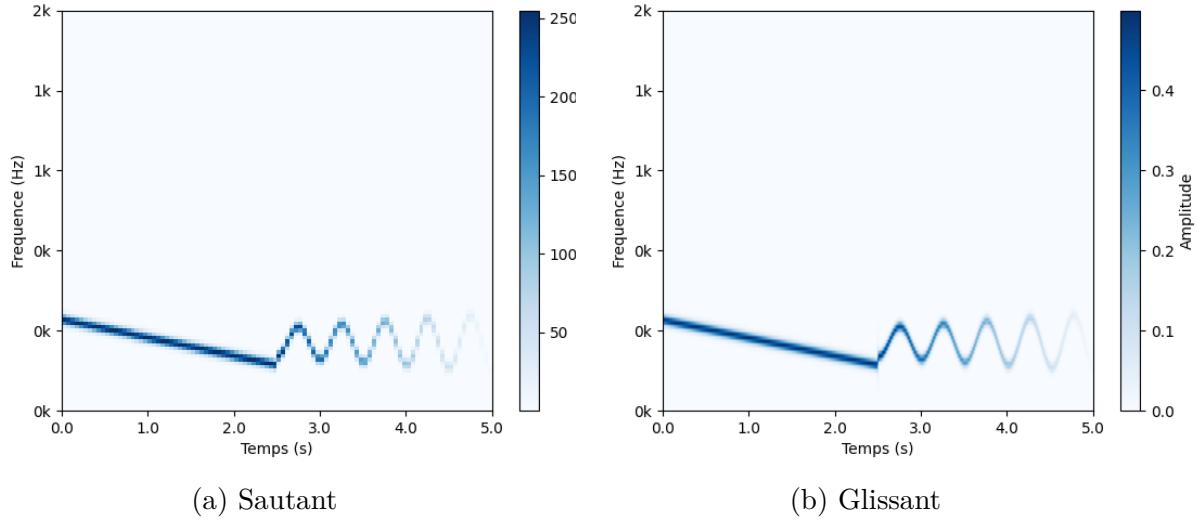


Figure 6: Comparaison de la méthode utilisée $t = 0.05\text{ s}$

Nous pouvons remarquer que la méthode du spectrogramme glissant, permet d’obtenir des résultats plus propres que celle du sautant.

Cependant la méthode glissant prend beaucoup plus de temps à exécuter. De plus, plus le pas de temps choisi est petit, moins la différence semble visible.

En fait, la limite du spectrogramme sautant est que l’on a une matrice des FFT de taille constante, on ne peut pas augmenter la taille de cette matrice, juste changer les dimensions en conservant la même taille. Tandis qu’avec la méthode glissante on peut augmenter cette taille, au dépends du temps de calcul.

5.1.3 Influence de la fenêtre utilisée pour la transformée de Gabor

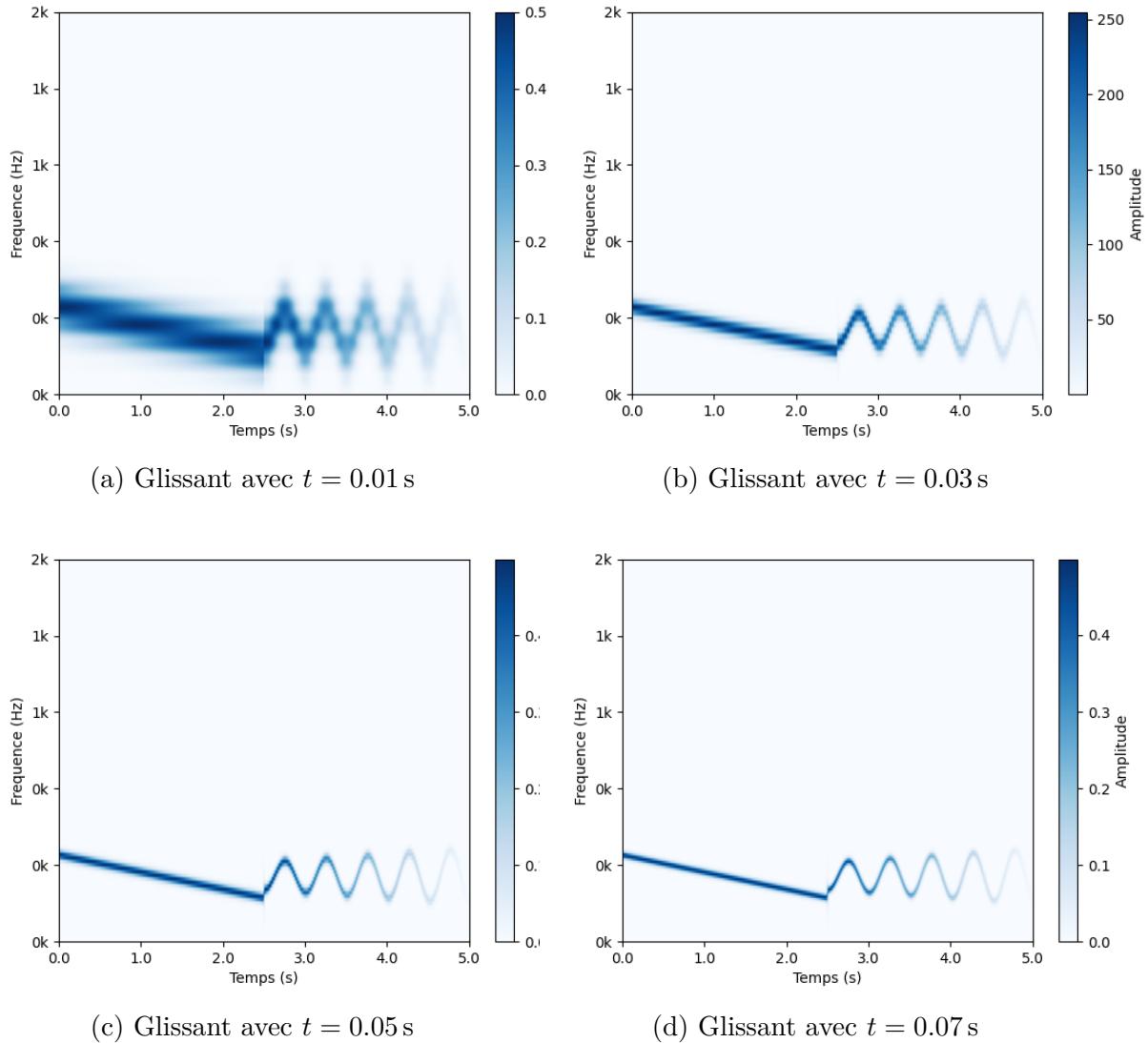


Figure 7: Comparaison des spectrogrammes glissant pour un pas de temps variable

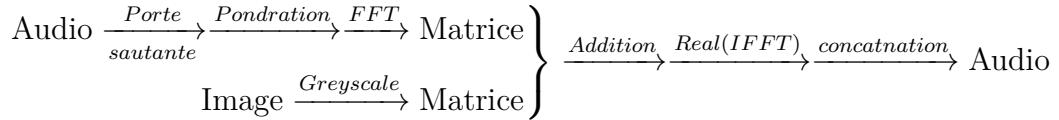
Ainsi nous pouvons remarquer qu'augmenter la fenêtre sur laquelle on effectue les FFT améliore la qualité des spectrogrammes obtenus.

Cependant il devient de plus en plus long d'exécuter le code. Nous n'avons pas réussi à exécuter le code avec des pas de temps supérieurs à 0.07 s, car nous obtenions systématiquement des erreurs systèmes (erreur SIGKILL sous Ubuntu et crash sous Windows, problème de mémoire saturée).

5.2 Incrustation de l'image dans le signal

5.2.1 Concept

Comme expliqué précédemment pour ajouter notre image dans le fichier sonore, nous allons procéder en ajoutant directement notre image en nuances de gris à la matrice des FFT sautantes.



En pratique il faut faire attention aux conversions entre les types : la matrice des FFT est une matrice de Complexes alors que l'image est une matrice de Réels.

De plus il pour pouvoir utiliser une addition standard il est nécessaire que les deux matrices aient la même taille (largeur et hauteur).

De plus il nous paraissait indispensable de pouvoir placer et dimensionner l'image dans le spectrogramme. Ainsi, nous avons inclus les paramètres `x_scale`, `y_scale`, `x_shift`, `y_shift` placer à peu près où l'on veut notre image.

En pratique ces paramètres fonctionnent en entourant notre image de bandes blanches.

5.2.2 Vérification de l'incrustation

Afin de vérifier que notre image a bien été incrustée dans le spectrogramme nous allons dans un premier temps vérifier avec notre fonction. Puis dans un second temps nous allons vérifier à l'aide d'un logiciel externe (*Audacity* dans ce cas) afin de pouvoir confirmer de manière certaine qu'il y a bien l'image dans le spectrogramme.

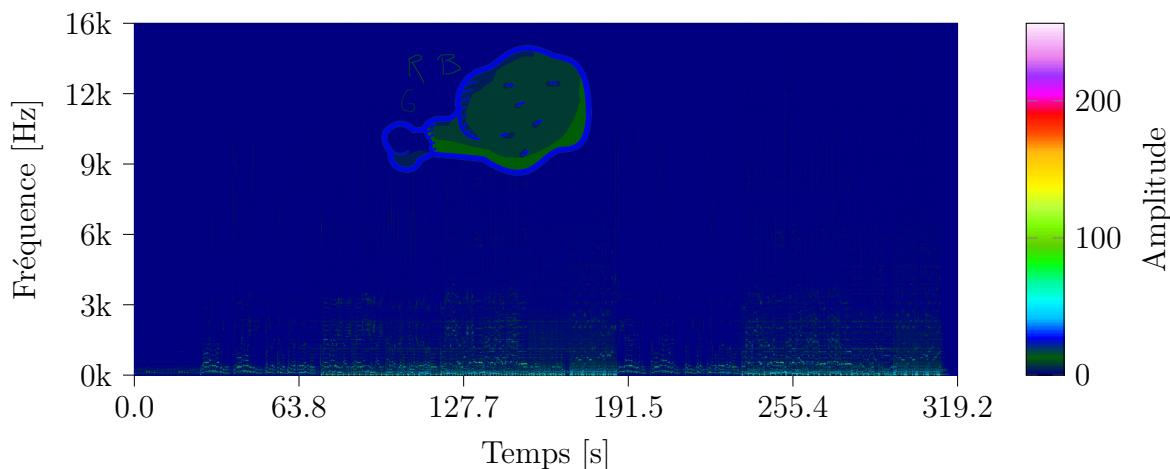


Figure 8: Visualisation de l'incrustation d'une image en nuance de gris avec notre fonction

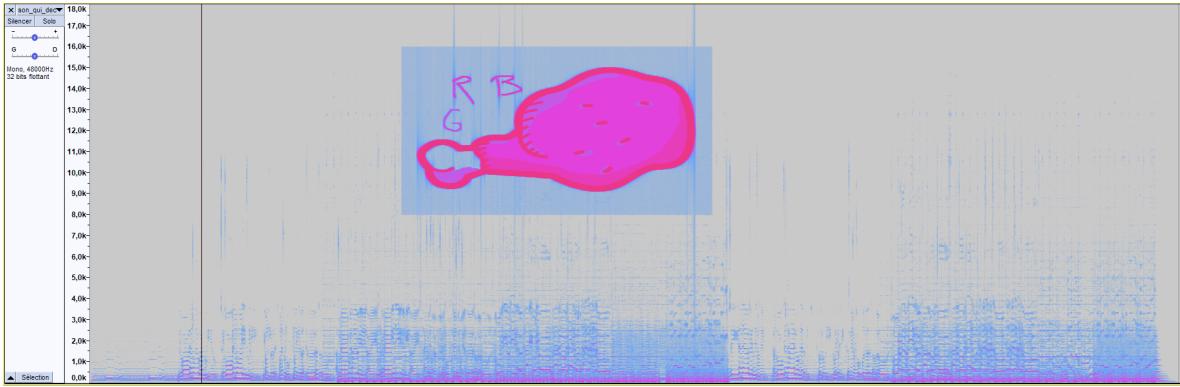


Figure 9: Visualisation de l'incrustation d'une image en nuance de gris avec Audacity

5.3 Incrustation d'une image RGB

Comme expliqué précédemment pour incruster une image RGB dans le spectrogramme, nous avons choisi d'incruster en réalité 3 images grayscale : une comportant la partie R, une comportant la partie G, et une pour le B. Ensuite en combinant les différentes composantes on peut récupérer l'image RGB.

$$\begin{array}{c} \text{Audio} \xrightarrow[\text{sautante}]{\text{Porte}} \xrightarrow{\text{Pondration}} \xrightarrow{\text{FFT}} \text{Matrice} \\ \text{Image} \xrightarrow{\text{EncodageRGB}} 3 \text{ Matrices} \end{array} \left. \right\} \xrightarrow{\text{Addition sur 3 segments}} \xrightarrow{\text{Real(IFFT)}} \xrightarrow{\text{concatnation}} \text{Audio}$$

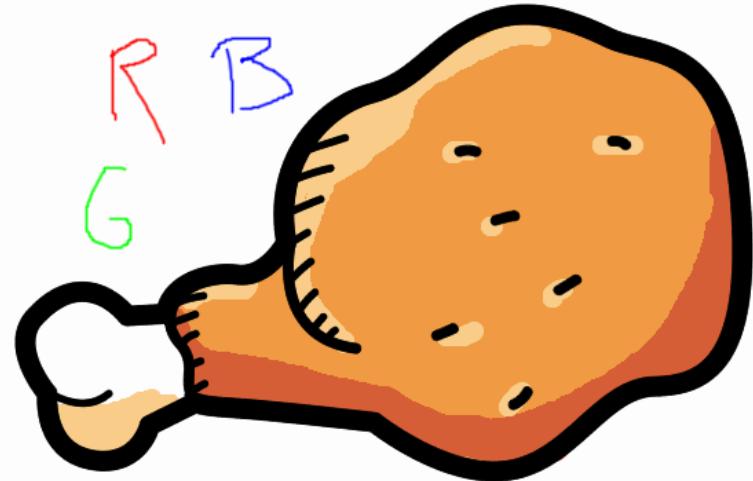


Figure 10: Image à incruster

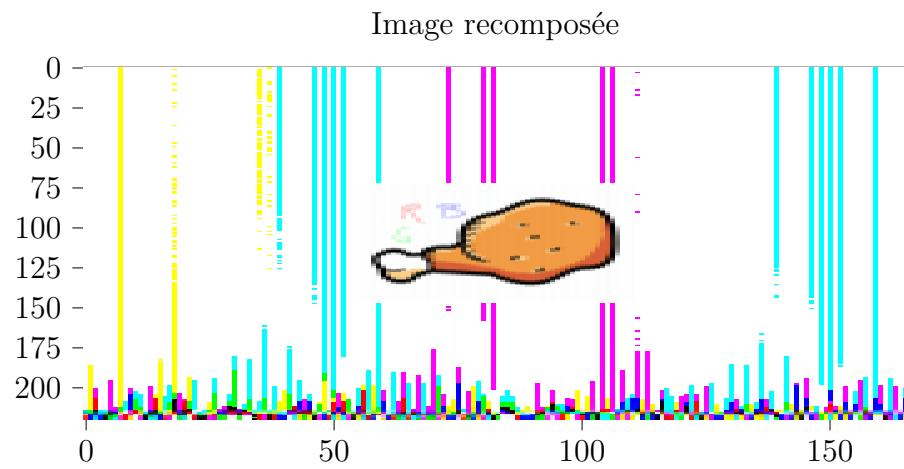


Figure 11: Pour un calcul de matrices sautantes avec $t = 0.01\text{ s}$ (échelle = Pixels)

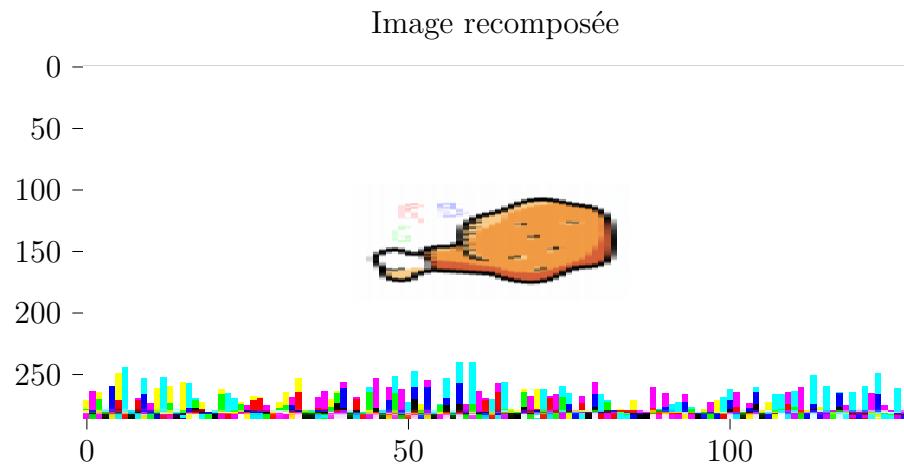


Figure 12: Pour un calcul de matrices sautantes avec $t = 0.013\text{ s}$ (échelle = Pixels)

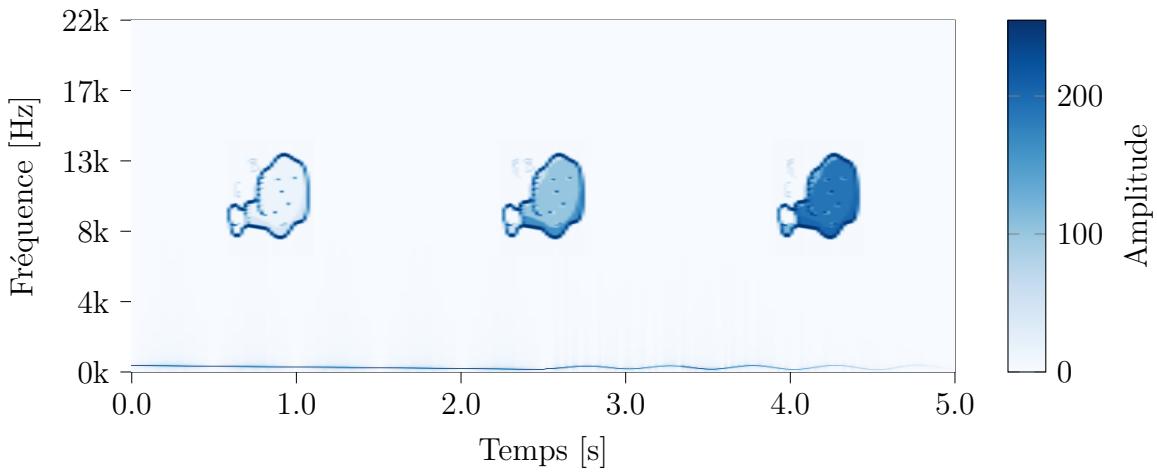


Figure 13: Spectrogramme du son contenant notre image RGB

Les 3 images R, G et B dans le spectrogramme sont bien visibles, chacune sur un tiers de la durée totale.

De plus on remarque des artefacts dans l'image décodée pour un pas de temps $t < 0.013\text{ s}$. Cependant bien que ces artefacts soient présents autour de l'image, celle-ci semble être préservée (Figure 14). C'est à dire qu'en rognant l'image correctement (fonction *crop* de *PIL*) il doit être possible d'en extraire l'image pure.

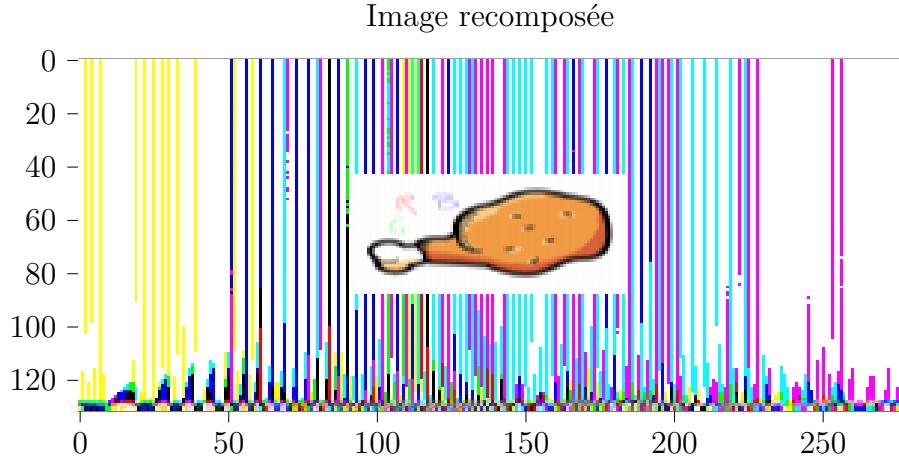


Figure 14: Pour un pas de temps $t = 0.06\text{ s}$

6 Limites et améliorations possibles

6.1 Limites du projet

Lors de l'analyse des résultats, nous avons mis en évidence une limite informatique à notre projet. En effet il nous était impossible d'utiliser un pas de temps supérieur à 0.07 secondes. Bien que les matrices soient calculées, l'erreur survient au moment du plot. Ainsi, il semblerait qu'il faille trouver une manière différente de tracer le spectrogramme pour dépasser cette limite.

On pourrait alors essayer d'utiliser la librairie PIL pour enregistrer directement la matrice en tant qu'image, sans avoir à utiliser l'interface de matplotlib qui sature si la matrice est trop grande.

À pas de temps fixe, plus le fichier audio est long (ou plus sa fréquence d'échantillonage est élevée), plus le nombre de points à traiter est grand. Cela est également une limite de notre algorithme actuel car il traite le signal dans son intégralité et a donc des besoins en mémoire de plus en plus importants. Pour limiter le temps de calcul et les temps d'allocation de la mémoire, il serait possible par exemple de diviser le signal en plusieurs morceaux, de les traiter à la suite puis de les fusionner pour l'affichage. Mais cela pose des difficultés d'implémentation au niveau des raccords entre les différents morceaux. D'autre part cela impliquerait aussi de modifier (presque) toutes les fonctions du programme afin de les adapter à ce nouveau mode de fonctionnement.

6.2 Améliorations d'ergonomie

Une amélioration souhaitable est la création d'une interface graphique permettant de choisir aisément le fichier audio, le type de transformée (sautante ou glissante) ainsi que d'afficher le spectrogramme.

6.3 Amélioration de la pondération

Une amélioration intéressante est de changer la pondération de la fenêtre. Dans le projet, nous comparons 3 types de pondérations différentes pour le spectrogramme glissant: la pondération Gaussienne, la pondération de Bartlett et la pondération de Hann. Ce sont 3 types de pondérations connues mais ce ne sont pas les seules qui existent. Ainsi, il serait intéressant de comparer différents types de pondération en fonction de différents types de fichiers audio pour déterminer quelle pondération correspond le mieux à quel type de son. De plus, il serait également intéressant d'appliquer les pondérations de Bartlett et de Hann aux spectrogrammes glissants.

6.4 Améliorations liées à des aspects de programmation

Dans un premier temps il faudrait adapter notre fonction d'affichage : *spectro_plotting* pour que l'axe du temps s'affiche avec les bonnes valeurs. Pour cela, il faut rajouter un

paramètre pour préciser si l'on plot une matrice sautante au glissante, et dans le cas de la glissante il faut redéfinir des nouvelle limites d'axes.

La mémoire utilisée par le projet est très importante. En effet, les matrices utilisées ont des dimensions très importantes, surtout dans lors de l'utilisation de transformées glissantes. Afin de limiter les temps de calculs liés à des matrices volumineuses, la librairie *numpy* est utilisée.

D'autre part le principal problème de notre programme est la gestion de la mémoire, pour cela une amélioration possible est d'utiliser des fonctions *generator* python (grâce à l'instruction *yield*) et d'itérer au travers de ce *generator* pour garder une petite partie des données en mémoire, de travailler dessus puis enfin de les sauvegarder. L'idée est d'itérer sur toutes les données du fichier de cette manière au travers d'une boucle *for*.

7 Conclusion

Ce projet nous a permis de mettre en pratique certains outils de traitement du signal découvert lors de cet enseignement que sont la transformée de Fourier et la transformée de Gabor. Ces outils nous ont permis d'analyser un signal puis de le représenter à l'aide de son spectrogramme. Finalement, nous avons modifié le signal de tel sorte qu'une image apparaisse sur son spectrogramme.

Nous avons pu observer de manière très concrète l'influence de paramètres tels la taille de la fenêtre ou la pondération sur le résultat final. Faire les bons choix est crucial afin d'avoir un résultat exploitable.

Nous avons également touché du doigt les limites physiques de nos ordinateurs, en raison de la quantité de mémoire nécessaire à certains calculs avec des fenêtres glissantes. Cela met en évidence l'importance de programmer efficacement nos algorithmes pour obtenir nos résultats de la façon la plus rapide et la moins gourmande en puissance de calcul possible.

References

- [1] Guillaume Cheverau. *Analysez les signaux 1D*. 2021. URL: <https://openclassrooms.com/fr/courses/4500266-analysez-les-signaux-1d>.
- [2] Guillaume Cheverau. *Signal GE4-MIQ4 (Cours Moodle)*. 2021.
- [3] Wikipedia. *Window Function*. 2021. URL: https://en.wikipedia.org/wiki/Window_function.