

Transcription Automatisée de la Musique

Max Holemans - MP Option Informatique

TIPE session 2025

Table des matières

1	Cadre	3
1.1	Objectifs premiers	3
1.2	Outils utilisés	4
1.3	Cadre musical	4
2	Modélisation	4
2.1	Signal d'entrée	4
2.2	Modélisation à l'échelle tonale	4
2.3	Modélisation à l'échelle rythmique	5
3	Recherche des notes jouées	6
3.1	Fenêtrage du signal	6
3.2	Algorithme de somme spectrale	6
3.3	Transport sur l'échelle MIDI	7
4	Analyse rythmique	7
4.1	Détermination de l'amplitude $A(t)$	7
4.2	Construction de l'échelle temporelle musicale	8
5	Construction de partitions	9
6	Efficacité du programme	10
6.1	Complexité temporelle	10
6.2	Rapidité effective	10

Introduction

En musique, la transcription est la notation d’une œuvre musicale initialement non écrite, pour la conserver et permettre de la réinterpréter. Par la transcription, les musiciens convertissent les données musicales sonores en une représentation visuelle simplifiée, le plus souvent sous forme de partition musicale car c’est sous cette forme que la musique est le mieux étudiée et analysée.

Cependant, ce procédé peut se montrer très fastidieux et difficile pour une oreille peu expérimentée. C’est pour cela que la question de l’automatisation se pose : comment automatiquement convertir un signal sonore en données musicales visuellement représentables ?

Des outils tels que *Transcribe!* existent pour accompagner les musiciens dans leur transcription. Ils proposent des guides visuels en temps réel comme le spectre du signal, indiquent les notes potentiellement jouées à un instant donné, et avancent même un début d’analyse en théorie musicale d’un extrait sonore (en proposant les accords joués par exemple). Ces outils d’analyse sonore partagent certains procédés avec d’autres applications : la reconnaissance musicale de *Shazam* ou bien la transcription vocale.

1 Cadre

1.1 Objectifs premiers

Dans ce TIPE, nous cherchons à élaborer un algorithme qui convertit un signal sonore en données musicales visuelles. La donnée d’entrée sera un signal audio dans le format standardisé WAV représentant un extrait musical enregistré. Nous chercherons alors à automatiquement représenter cet extrait par un spectrogramme, puis par un « piano-roll » (représentation plus intuitive de la musique) et finalement par une partition musicale. Ces représentations (voir figure 1) correspondent à des niveaux d’abstraction différents des données de départ, et chaque étape d’abstraction sera construite à partir de la représentation précédente. Chaque forme a son utilité dans la recherche d’information musicale (comme expliqué dans [2] p.18), mais notre but sera ici d’aboutir à une partition.

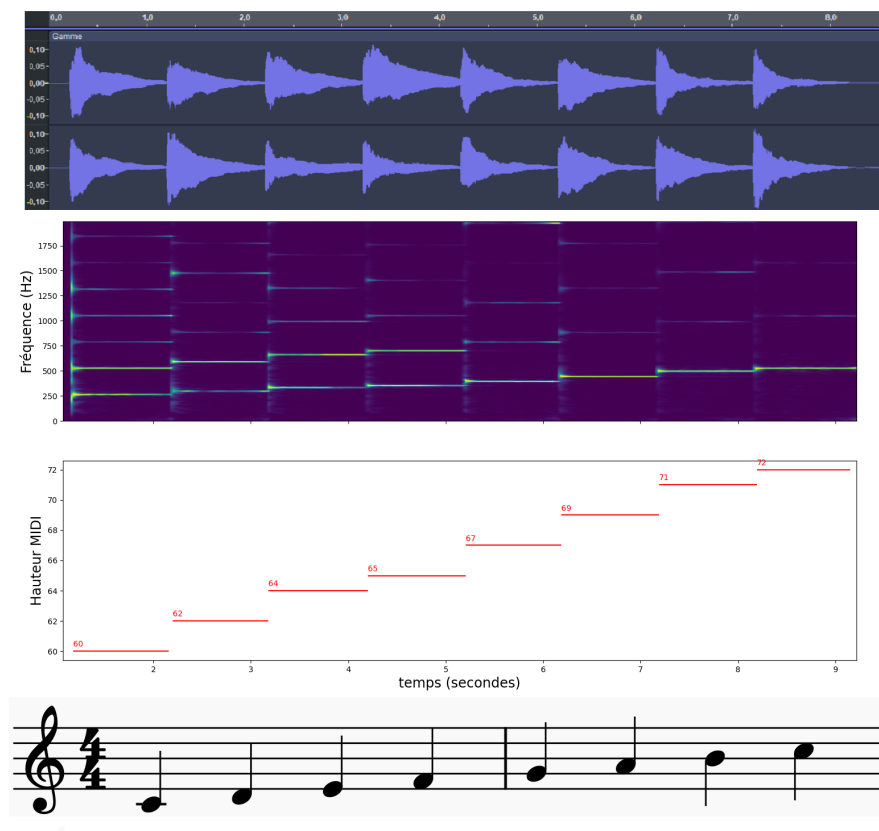


FIGURE 1 – Différentes représentations d’une gamme de Do majeur, ordonnées par niveau d’abstraction : enregistrement brut ; spectrogramme ; « piano-roll » ; partition.

1.2 Outils utilisés

L'implémentation de cet algorithme se fera avec le langage de programmation *Python 3*. Les bibliothèques *numpy* et *scipy* vont servir de base pour le traitement des signaux considérés. Les premières représentations visuelles seront générées avec le module *matplotlib.pyplot* mais c'est la bibliothèque *partitura* qui permettra de générer des partitions sous le format *MusicXML*; les fichiers sous ce format peuvent ensuite être exploitées avec un logiciel tiers tel que *Musescore*.

1.3 Cadre musical

La musique existe sous un nombre abondant de formes différentes, nous allons donc nous restreindre à un cadre musical défini et habituel dans la tradition de la musique occidentale. Notre étude se concentrera sur la musique tonale (les instruments sont joués avec une hauteur donnée) et l'on se placera dans le système de la gamme tempérée. La métrique (manière de segmenter les durées musicales) sera fixée par la signature rythmique 4/4, la plus commune dans ce cadre. On considérera que les notes sont jouées avec une attaque (ce qui est systématique avec des instruments comme la guitare, les claviers, la harpe et la contrebasse pincée). Ces caractéristiques de la musique seront explicitées dans l'étape de modélisation.

2 Modélisation

2.1 Signal d'entrée

L'extrait musical étudié sera représenté par une grandeur vibrante continue $s(t)$, échantillonnée en un enregistrement discret du signal avec une fréquence d'échantillonnage f_s . On note s la liste python représentant ce signal discret, et N_{signal} sa taille. Pour tout $i \in \llbracket N_{\text{signal}} - 1 \rrbracket$, on note $s[i]$ l'échantillon d'indice i de s . Dans la suite du rapport, la notation $s[n]$ désignera la grandeur correspondant à la liste s , c'est-à-dire l'analogue discrétisé de $s(t)$.

La grandeur $s(t)$ est donc unidimensionnelle (la musique stéréo sera préalablement convertie en mono). L'analyse de ce signal se fera sur deux échelles temporelles distinctes : l'échelle tonale en hautes fréquences qui représentera la hauteur (le *pitch*) des notes ainsi que leur timbre, et l'échelle rythmique en plus basses fréquences qui correspond aux mouvements, transitions et enchaînements de la musique.

2.2 Modélisation à l'échelle tonale

Par échelle tonale on entend l'échelle des hautes fréquences, dans le spectre audible : 20 Hz à 20 kHz. Par exemple, un piano standard de 88 touches joue des notes de 27,5 Hz à 4 186,01 Hz.

Considérons le son d'une note, tenue à une hauteur fixée. On peut d'abord considérer que ce signal est rigoureusement périodique, et on peut alors représenter son spectre d'amplitude constituée d'une composante continue (nulle en pratique), d'une composante fondamentale à la fréquence f_{fond} et de composantes harmoniques se situant à des fréquences multiples de f_{fond} . Ce modèle idéal est illustré dans la figure 3.

C'est cette fréquence fondamentale f_{fond} qui détermine la hauteur de la note : à chaque note de la gamme tempérée, on associe une fréquence, et un entier qui représente la note sur l'échelle MIDI, pratique pour la manipulation informatique des notes. On ramène ainsi l'échelle continue des fréquences jouées sur l'échelle discrète de la gamme tempérée, établie par convention.

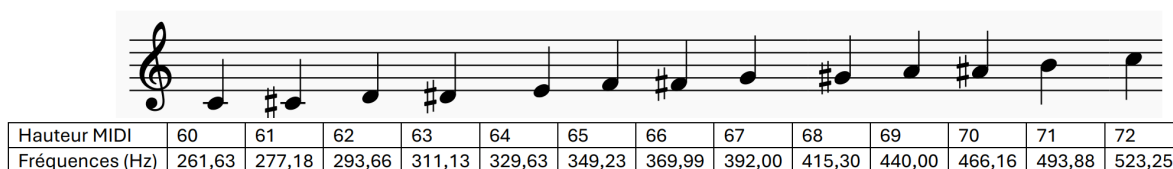


FIGURE 2 – Correspondance entre les hauteurs MIDI, les notes sur la portée et leur fréquence associée. On représente ici seulement les notes de la gamme chromatique de Do.

Les harmoniques caractérisent le timbre de la musique; l'enveloppe spectrale est ainsi propre à chaque instrument, indépendamment de la note jouée.

Cependant les notes réelles ne sont pas des signaux rigoureusement périodiques : un signal réel est limité dans l'espace et l'enregistrement présente un bruit de fond sonore, ce qui forme également du bruit dans le spectre du signal. De plus, les instruments réels ne sont pas parfaitement harmoniques (on parle de « quasi-harmonicité ») : les composantes harmoniques du spectre ne se trouvent pas exactement sur des multiples de la fréquence fondamentale.

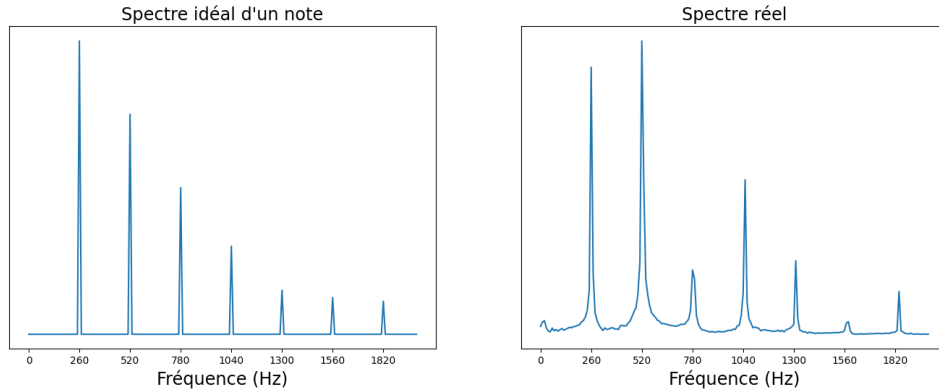


FIGURE 3 – Spectre idéal et spectre réel d'une note de fréquence fondamentale $f_{\text{fond}} = 260$ Hz.

2.3 Modélisation à l'échelle rythmique

Par échelle rythmique on entend l'échelle temporelle dans laquelle on ne perçoit pas des tons, mais des battements, des variations d'intensité sonore.

À l'échelle rythmique, la grandeur pertinente est l'amplitude efficace $A(t)$ (ou $A[n]$ en tant que signal discret). On peut alors modéliser un profil type d'amplitude pour une note de musique dans notre cadre. Une note commence par une phase d'attaque où l'amplitude augmente brutalement, puis la note est tenue pour une certaine durée, et enfin l'amplitude baisse lors de la fin de note et finit par laisser place au silence.

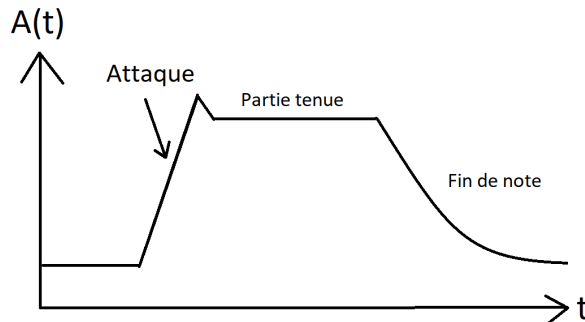


FIGURE 4 – Profil type d'amplitude pour une note. Ce profil est un modèle commun, notamment présenté dans [3].

Concernant l'analyse rythmique, nous allons également diviser l'échelle continue du temps en une échelle discrète grâce au tempo. Le tempo est la fréquence du battement régulier de la musique et est exprimée en battements par minute (bpm). Il représente le rythme régulier élémentaire de l'extrait. Ainsi avec un tempo et une signature rythmique (4/4 dans notre cas) donnés, on obtient des mesures divisées en 4 battements, et on peut également diviser les battements en parts égales (en pratique nous allons limiter notre précision rythmique à un battement ou un demi-battement pour des mélodies simples).

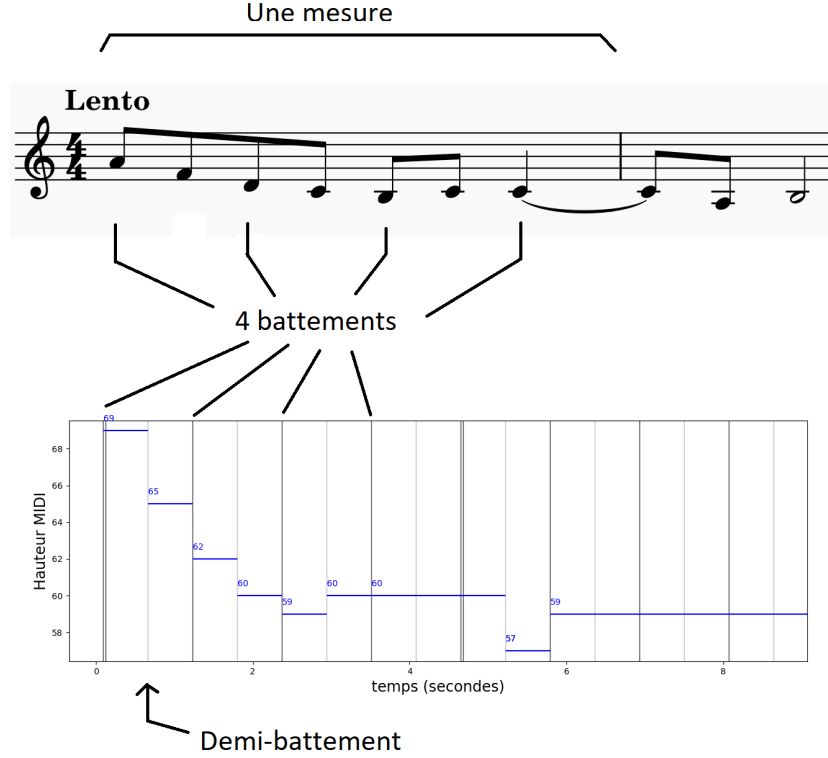


FIGURE 5 – Division rythmique dans la partition et dans le « piano-roll ».

3 Recherche des notes jouées

3.1 Fenêtrage du signal

L'objectif de cette section est de construire un algorithme qui détermine la note jouée à un instant donné t_0 du signal $s(t)$. L'information tonale d'une note se trouve dans le spectre de son signal, qui ne peut être déterminé à un instant unique. Notre analyse se portera donc sur un très court extrait de durée Δt de $s(t)$ centré en t_0 (la restriction de $s(t)$ à $[t_0 - \frac{\Delta t}{2}, t_0 + \frac{\Delta t}{2}]$). En pratique, l'indice $n_0 = t_0 f_s$ correspond à l'instant t_0 dans le signal discret et $N = \Delta t f_s$ est la taille de la fenêtre discrète considérée. En pratique, on note $n_0 = t_0 f_s$ l'indice correspondant à t_0 dans le signal discret et $N = \Delta t f_s$ la taille de la fenêtre discrète considérée :

$$s \left[n_0 - \frac{N}{2}, n_0 + \frac{N}{2} \right]$$

On multiplie cet extrait avec la fenêtre de Hann¹ w_{Hann} de taille N pour obtenir le signal discret fenêtré $s_w[n]$ défini sur $[n_0 - \frac{N}{2}, n_0 + \frac{N}{2}]$. Puis pour finalement obtenir le spectre $S[k]$ (où le paramètre entier k est lié aux fréquences) de la note jouée en n_0 , on applique la Transformée de Fourier Rapide sur l'extrait $s_w[n]$. Ces différentes fonctions sont données par la bibliothèque `scipy`.

Pour construire le spectrogramme du signal initial $s(t)$, il suffit de calculer les spectres $S[k]$ pour différentes valeurs de n_0 . On obtient ainsi un tableau de valeurs à deux dimensions $SG[n_0, k]$ représentable avec un gradient de couleurs comme le montre la figure 1.

Pour assurer une bonne résolution temporelle de notre analyse tonale, le calcul des spectres $S[k]$ se fait avec la méthode de la fenêtre glissante : on fixe un entier de saut h inférieur à N pour ne pas perdre en information et on augmente n_0 de h après chaque calcul de spectre, jusqu'à la fin du signal $s[n]$.

3.2 Algorithme de somme spectrale

Dans la littérature scientifique, différentes méthodes sont présentées pour déterminer la fréquence jouée à un instant donné : autocorrélation sur le signal $s(t)$, autocorrélation sur le spectre $S(t)$, étude

1. Puisqu'on va appliquer la Transformée de Fourier à un signal fini, cette étape de fenêtrage est nécessaire. Le fenêtrage plus standard de Hamming conviendrait également.

du cepstre², etc. Nous remarquons dans [6] que ces méthodes de recherche de fréquence fondamentale passent souvent par une phase de sélection de fréquences candidates.

Dans notre algorithme nous allons procéder par la sélection de fréquences candidates dans une plage de fréquences donnée, puis sélectionnerons le candidat correspondant le mieux à notre modèle de fréquence fondamentale.

Les fréquences candidates sont les fréquences des maxima locaux suffisamment proéminents dans le spectre $S[k]$. On associe ensuite à chaque candidat une valeur dite « somme spectrale » (présentée dans [5]) définie selon le pseudocode suivant :

Algorithme 1 : Pseudocode de la somme spectrale

Entrées : Le spectre S à l’instant considéré, une fréquence candidate f et un réel positif ε

Sorties : La somme spectrale associée à f à l’instant donné

somme $\leftarrow 0$

$k \leftarrow 1$

tant que kf est inférieur à la fréquence maximale du spectre S **faire**

 val $\leftarrow \max\{S(w) \mid w \in [f - \varepsilon, f + \varepsilon]\}$

 somme \leftarrow somme + val

$k \leftarrow k + 1$

fin

retourner somme

Ce score associé aux fréquences candidates représente l’intensité cumulée de la fréquence fondamentale f_{candidat} et des fréquences harmoniques $k.f_{\text{candidat}}$ présentes ou non. Le candidat qui maximise cette fonction de somme spectrale³ est donc la fréquence correspondant le mieux au rôle de fréquence fondamentale à cet instant du signal.

Pour tenir compte du caractère quasi-harmonique de la musique, la recherche de l’amplitude d’une harmonique est la recherche du maximum atteint dans un voisinage de la fréquence multiple $k.f_{\text{candidat}}$.

Cette méthode de somme spectrale peut être affinée pour prendre en compte le modèle psycho-acoustique de la perception humaine, comme cela est fait dans les pages 258 à 259 de [4].

3.3 Transport sur l’échelle MIDI

Grâce à l’algorithme de somme spectrale, nous pouvons déterminer la fréquence jouée pour chaque indice n_0 du spectrogramme $SG[n_0, k]$. Il faut alors ramener les fréquences trouvées sur l’échelle discrète de la gamme tempérée. À toute fréquence f , on associe une donnée d (sans dimension) par :

$$d = d_{\text{ref}} + 12 \log_2\left(\frac{f}{f_{\text{ref}}}\right)$$

où d_{ref} et f_{ref} sont des données de référence. Par convention, on utilisera la référence du La central : $d_{\text{ref}} = 69$ et $f_{\text{ref}} = 440$ Hz. La hauteur MIDI p associée à f est alors l’entier le plus proche de d .

Il est alors possible de tracer la hauteur MIDI en fonction du temps, c’est la représentation de la musique sous forme de « piano-roll ». Le passage de fréquences à hauteur MIDI est illustrée sur la figure 6.

4 Analyse rythmique

4.1 Détermination de l’amplitude $A(t)$

Comme énoncé à l’étape de modélisation, la grandeur pertinente pour l’analyse rythmique est l’amplitude efficace $A(t)$. Cette amplitude, en lien avec la valeur efficace du signal, épouse la forme de l’enveloppe de $s(t)$. Une première méthode pour estimer $A(t)$ utilise un filtre passe-bas numérique, appliqué au carré du signal. Le résultat de cette méthode est tracé en bleu sur la figure 7. Cette approche donne un signal $A(t)$ assez proche de l’enveloppe de $s(t)$ pour estimer le tempo de l’enregistrement musical⁴.

Cependant l’amplitude ainsi obtenue est inexploitable pour une analyse rythmique plus fine : le filtre passe-bas empêche la modélisation correcte de l’attaque (les sauts en amplitude ne sont plus assez raides) et laisse pourtant un bruit trop prononcé.

2. Ces deux dernières méthodes vont étudier les périodicités présentes dans le spectre d’un signal musical.

3. Notons qu’il existe des variantes de la somme spectrale : le produit spectral, et la somme spectrale des amplitudes carrées.

4. voir partie 4.2.

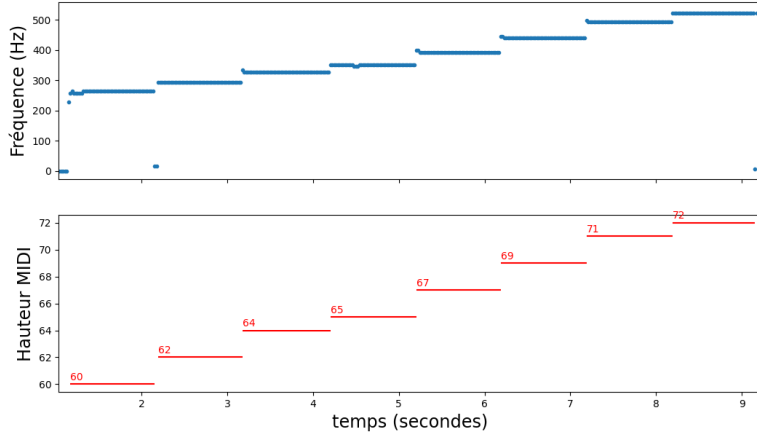


FIGURE 6 – Exemple de projection de fréquences sur l'échelle MIDI (gamme de Do majeur).

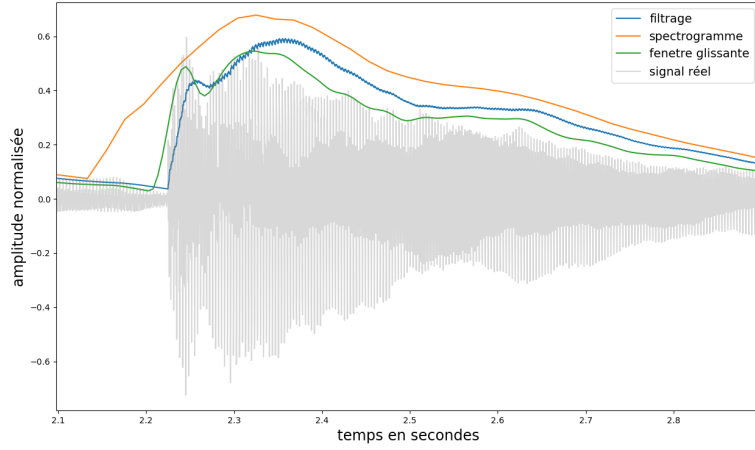


FIGURE 7 – Tracé de plusieurs méthodes de calcul de $A(t)$.

Plusieurs autres méthodes d'estimation de l'amplitude existent (dont une méthode exploitant le spectre $S[k]$, représentée en orange sur la figure 7), mais celle qui épouse le mieux l'enveloppe du signal parmi les méthodes explorées durant ce TIPE, et qui garde une attaque fidèle aux sauts d'intensité sonore de $s(t)$, est encore une méthode par fenêtre glissante :

On fixe N la taille de la fenêtre de sorte que $\Delta t = N/f_s$ soit de l'ordre de 50 ms. Soit un indice n_0 , on calcule l'amplitude efficace⁵ de $s[n]$ en n_0 par⁶ :

$$A[n_0] = \sqrt{\frac{1}{N+1} \sum_{i=0}^N \left| s \left[n_0 - \frac{N}{2} + i \right] \right|^2}$$

4.2 Construction de l'échelle temporelle musicale

L'élément le plus important dans l'analyse rythmique de la musique est le tempo. Celui-ci peut être déterminé avec une approximation grossière de l'amplitude efficace $A(t)$. Le tempo est une fréquence témoignant d'une allure périodique de $A(t)$. Il représente les battements de la musique indépendamment

5. Cette expression est l'équivalent discret de la valeur efficace locale : $A(t_0) = \sqrt{\langle |s(t)|^2 \rangle_{t, \text{local}}}$.

6. Cette formule est donnée dans [3] p.16.

des hauteur des notes jouées (on se place en basses fréquences ici, de l'ordre du Hz), mais on peut tout de même réappliquer des principes d'analyse tonale.

En calculant le spectre de l'amplitude efficace sur l'entièreté de l'enregistrement, par Transformée de Fourier Rapide, on peut trouver la fréquence de battement de la musique, un maximum atteint par le spectre entre 50 et 140 bpm⁷ (entre 0,8 et 2,3 Hz). Dans le cas d'un extrait musical assez régulier, on obtient une bonne estimation du tempo. Une fois le tempo déterminé, il est possible de segmenter l'échelle du temps en mesures comme décrit dans la phase de modélisation et illustré sur la figure 5, en imposant le début de la segmentation à l'instant du début de la première note.

Une estimation plus exacte de l'amplitude efficace $A(t)$ permet de pousser l'analyse rythmique plus loin. On peut identifier les silences comme étant les intervalles sur lesquels $A(t) < A_{\text{seuil}}$. On peut aussi dériver le signal $A(t)$ en un signal $D(t)$ dont les maxima sont atteints lors des attaques des notes, et on détermine ainsi les débuts des notes jouées, comme représenté sur la figure 8. L'analyse rythmique plus poussée que la recherche simple du tempo et le positionnement des notes peut être faite à partir de ces événements d'attaque, comme présenté dans [1].

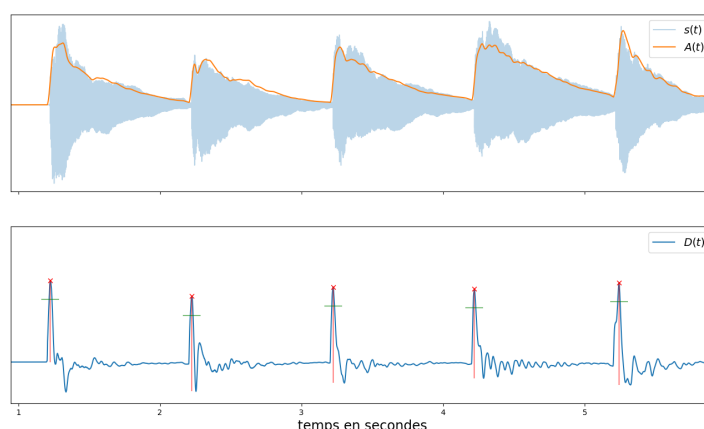


FIGURE 8 – Tracé de $A(t)$ calculée par méthode de fenêtre glissante, ainsi que de sa dérivée $D(t)$ dont les pics majeurs (correspondant aux attaques des notes) sont mis en valeur.

5 Construction de partitions

Jusqu'ici nous avons généré le spectrogramme et la représentation « piano-roll » de l'enregistrement musical donné. Mais l'outil le plus précieux pour l'analyse musicale est la partition écrite : les notes y sont représentées par des points sur une portée ; plus la note est placée haute sur celle-ci, plus son ton est haut ; les durées de notes sont représentées par leur forme (blanche, noire, croche...) ; les mesures sont séparées par des barres verticales sur la portée.

La bibliothèque **partitura** permet de construire une portée et de la remplir note par note. Au vu de l'utilisation de cette bibliothèque, une note sera représentée par une hauteur MIDI p , un instant de début de note t_i et un instant de fin de note t_f . Ces données peuvent être déterminées en recherchant les plateaux dépassant une longueur minimale dans la représentation « piano-roll » de $s(t)$ (cette même représentation est utilisée dans [2] p.19). Il suffit alors de projeter les instants t_i et t_f sur l'échelle temporelle discrète musicale déterminée lors de l'analyse rythmique, et d'ajouter la note à la portée de **partitura**.

On peut alors générer un fichier au format *MusicXML* qui représente la partition créée. Ce fichier peut ensuite être ouvert avec un logiciel tiers comme *MuseScore*, qui permet aux musiciens de modifier la partition résultante de l'algorithme.

7. battements par minute

6 Efficacité du programme

6.1 Complexité temporelle

Pour évaluer la complexité temporelle de l'algorithme, évaluons d'abord la complexité de l'analyse d'une fenêtre $s_w[n]$ de taille N décrite en partie 3.1. Le fenêtrage même est linéaire en N ; la Transformée de Fourier Rapide appliquée à $s_w[n]$ se fait en $O(N \log(N))$; la recherche de candidats est linéaire⁸ en N ; le calcul de la somme spectrale d'un candidat donné se fait au plus en $O(N)$ (car la taille du spectre construit $S[k]$ est proportionnelle à N). On peut de plus supposer que pour un signal assez épuré, le nombre de candidats reste petit, indépendamment de N . Dans ce cas, le calcul de toutes les sommes spectrales et la recherche du candidat idéal se fait en $O(N)$. On obtient ainsi une analyse du signal fenêtré de complexité quasi-linéaire.

Cependant nous utilisons la méthode de la fenêtre glissante⁹ avec une taille de fenêtre N fixée (avec $f_s = 44100$ Hz, $N = 2^{13}$ semble donner le meilleur rapport précision/coût) ainsi qu'un saut h fixé. Asymptotiquement, l'analyse du signal (jusqu'à la construction du « piano-roll ») est donc de complexité linéaire en le nombre de fenêtres considérées, donc linéaire en la taille N_{signal} du signal $s[n]$ de départ.

La construction de la partition se fait note par note, on retrouve ainsi également une complexité linéaire en N_{signal} pour cette étape.

Sous la forme actuelle de l'algorithme, le tempo est déterminé en $O(N_{\text{signal}} \log(N_{\text{signal}}))$ puisqu'on applique une Transformée de Fourier Rapide à $A[n]$ en entier. Cependant, lorsque l'on considère un enregistrement très long, il serait plus judicieux de déterminer le tempo localement, en appliquant la même étude sur une fenêtre de taille fixée. De cette façon, on trouve comme pour la méthode de la fenêtre glissante une complexité linéaire en N_{signal} .

Finalement l'étude complète de l'enregistrement se fait en complexité temporelle linéaire en la taille N_{signal} du signal de départ $s[n]$.

6.2 Rapidité effective

L'algorithme implémenté jusqu'à ce point du TIPE applique la méthode de la fenêtre glissante pour déterminer le spectrogramme de l'enregistrement, recherche les hauteurs jouées par somme spectrale, cherche le début et la fin de chaque note jouée, détermine le tempo avec une estimation de l'amplitude par filtre passe-bas, construit la portée `partitura` et le fichier `MusicXML` correspondant, puis affiche le spectrogramme et le « piano-roll » avec `matplotlib.pyplot`. Les coûts en temps de chaque étape de l'algorithme¹⁰ appliqué à deux extraits musicaux, sont mis en évidence dans la table 1.

étapes de l'algorithme	extr. 1	extr. 2
import de modules	1,03	1,04
import du fichier	0,01	0,04
spectrogramme	0,35	1,70
recherche hauteurs	0,46	5,23
détermination des t_i et t_f	< 0,01	0,01
estimation du tempo	0,22	1,09
construction de la partition	0,01	0,01
affichage	0,23	0,42
total	2,31	9,61

TABLE 1 – Temps de calcul en secondes des différentes étapes de l'algorithme appliqué à deux extraits : extr. 1 est la gamme de Do majeure et dure 10 secondes ; extr. 2 est la première minute de Adagio - Bach. Les deux extraits musicaux sont enregistrés sous le même format WAV et ont la même fréquence d'échantillonnage f_s .

Conclusion

Dans ce TIPE nous avons construit un algorithme permettant de représenter un enregistrement musical simple, comme une mélodie courte, sous différentes formes visuelles : le spectrogramme, la forme

8. On utilise ici la fonction `find_peaks` du module `scipy.signal`


9. voir partie 3.1

10. L'algorithme est exécuté avec Thonny

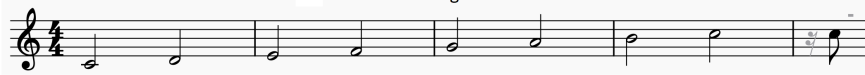
« piano-roll », et surtout la partition écrite. Notons que notre algorithme ne peut qu’analyser des mélodies¹¹, et il faudrait étendre notre algorithme de somme spectrale pour adapter notre étude aux sons polyphoniques.

En imposant une précision rythmique au battement ou au demi-battement près, les mélodies simples sont plutôt bien représentées par notre programme : la hauteur des notes est correctement transposée et le tempo correspond à l’original, à un facteur 2 ou 4 près, comme illustré sur la figure 9.

Partition originale : Gamme de Do majeur




Résultat de l'algorithme



Partition originale : Mélodie simple

Lento



Résultat de l'algorithme

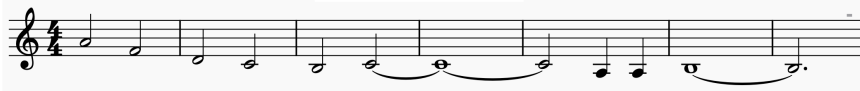



FIGURE 9 – Partitions résultantes de l’algorithme à partir de deux extraits simples.

Cependant lorsque la mélodie se complexifie, comme pour l’extrait étudié sur la figure 10, le simple calcul du tempo ne suffit plus à placer les notes sur les bons battements, il faudrait alors exploiter l’analyse rythmique plus fine présentée en fin de partie 4.2 pour déterminer plus précisément les débuts et fins de notes.

Partition originale : mélodie de Adagio - Bach



Résultats de l'algorithme




FIGURE 10 – Partition résultante de l’algorithme à partir d’une mélodie plus compliquée (première minute de Adagio -Bach).

De plus, notre analyse complète est assez rapide devant la durée du signal d’entrée (moins de 10 secondes de temps de calcul pour une minute d’enregistrement). Cette analyse pourrait donc être adaptée pour une transcription en temps réel d’un morceau enregistré, ce qui peut se montrer utile dans l’apprentissage en théorie musicale ou dans l’improvisation musicale.

11. Musique monophonique, à une voix, où les notes ne se superposent pas

Références

- [1] Simon DIXON. « Automatic Extraction of Tempo and Beat From Expressive Performances ». In : *Journal of New Music Research* 30.1 (2001), p. 39-58. URL : <https://www.tandfonline.com/doi/abs/10.1076/jnmr.30.1.39.7119>.
- [2] Jean-Louis DURRIEU. « Automatic transcription and separation of the main melody in polyphonic music signals ». Theses. Ecole nationale supérieure des telecommunications - ENST, mai 2010. URL : <https://theses.hal.science/tel-00560018>.
- [3] Dominique FOURER. « Approche informée pour l'analyse du son et de la musique ». Theses. Université Sciences et Technologies - Bordeaux I, déc. 2013. URL : <https://theses.hal.science/tel-00954729>.
- [4] Dik J. HERMES. « Measurement of pitch by subharmonic summation ». In : *The Journal of the Acoustical Society of America* (fév. 1988). URL : <https://www.researchgate.net/publication/19813760>.
- [5] RYX. *Reconnaissance de notes de musique*. Zeste de savoir. 2019. URL : <https://zestedesavoir.com/tutoriels/3013/reconnaissance-de-notes-de-musique/>.
- [6] David TALKIN. « Robust Algorithm for Pitch Tracking (RAPT) ». In : *Speech Coding and Synthesis*. Elsevier Science, 1995. Chap. 14.

Annexe

```

1  # Code principal
2
3  from time import time
4
5  timelog = {}
6  timelog["start"] = time()
7
8  from scipy.signal.windows import hann
9  from scipy.signal import find_peaks
10 from scipy.fft import rfft, rfftfreq
11 from scipy.io import wavfile
12 import matplotlib.pyplot as plt
13 import numpy as np
14
15 import partitura as pt
16
17 timelog["import de modules"] = time()
18
19 file_name = "../Matériel audio/Gamme.wav"
20
21 fs, raw_file = wavfile.read(file_name)
22 s = raw_file[:, 0] # conversion en mono
23
24 timelog["import du fichier"] = time()
25
26 N = len(s) # nombre de points
27 dt = 1/fs # intervalle de temps entre mesures
28 t = np.arange(N) * dt
29
30 tmin, tmax = 0, 10 # s
31 imin, imax = int(tmin/dt), int(tmax/dt)
32 N_fenetre = imax - imin
33 s_fenetre = s[imin:imax]
34 t_fenetre = t[imin:imax]
35
36 timelog["fenetre"] = time()
37
38 #####
39 # Determination du tempo #
40 #####
41
42 s_norm = s_fenetre/np.max(s_fenetre)
43 s_carre = s_norm * s_norm
44
45 fmin = 2
46 fc = 6 # en Hz, frequence de coupure du passe-bas
47
48 s_filtre = [0]
49 for n in range(N_fenetre - 1):
50     v = s_filtre[n] + 2*np.pi*fc/fs*(s_carre[n] - s_filtre[n])
51     s_filtre.append(v)
52
53 S_env = np.absolute(rfft(s_filtre))
54 freq_env = rfftfreq(N_fenetre, dt)
55
56 freq_fenetre_env = freq_env[int(fmin/freq_env[1]):int(fc/freq_env[1])]
57
58 bps = freq_fenetre_env[np.argmax(S_fenetre_env)]
59
60 timelog['filtrage / fft enveloppe'] = time()
61
62 #####
63 # Calcul du spectrogramme #
64 #####
65
66 N_win = 2**13 # Nombre de points de la fenetre
67 hop = 2**10 # taille du saut de fenetre (en nombre d'echantillons/points)
68
69 win = hann(N_win)
70 n = (N_fenetre - N_win)//hop + 1 # nombre d'analyses
71 p = N_win//2 + 1 # nombre de points d'une transformation
72
73 S = np.zeros((p, n))
74
75 T = np.zeros(n)
76
77 df = fs/N_win
78 freq = rfftfreq(N_win, dt)
79
80 for j in range(n):
81     # indices de début et de fin de la fenetre consideree
82     i_s = j*hop
83     i_e = i_s + N_win
84
85     # l'instant de l'analyse (milieu de fenetre)
86     _t = (i_s+i_e)/2 * dt
87
88     # l'extrait analyse et le calcul de sa transformee
89     _s = s_fenetre[i_s:i_e] * win
90     _S = np.absolute(rfft(_s))
91
92     # Ajustement Local ("normalisation")
93     m = np.max(_S)
94     if m > 0:
95         _S /= m
96
97     T[j] = _t
98     for i in range(p):
99         S[i, j] = _S[i]
100
101 timelog["spectrogramme"] = time()
102
103 #####
104 # Recherche de la note jouee #
105 #####
106
107 def somme_spect(S, f, eps):
108     somme, k, i = 0, 1, round(f/df)
109     while i < len(S):
110         # fenetre de taille ~2eps servant a compenser le caractere

```

FIGURE 11 – Programme principal, partie 1/3.

```

116 # quasi-harmonique de l'instrument
117 win = S[max(1-eps, 0):min(1+eps, len(S))]
118 somme += np.max(win)
119 k += 1
120 i = round(k*f/df)
121 return somme
122
123 Mel_freq = np.zeros(n)
124
125
126 prominence = 0.1 # prominence minimale des pics candidates
127
128 f_eps = 10 # taille spectrale de la fenetre pour la somme spectrale
129
130 i_eps = round(f_eps/df)
131 for i in range(n):
132     _S = S[i:, 1]
133     peaks, _ = find_peaks(_S, prominence=prominence)
134     f_cand = freq[peaks] # frequences candidates
135
136     # recherche du meilleur candidat (argmax de somme spectrale)
137     f_note, s_max = 0, 0
138     for f in f_cand:
139         somme = somme_spect(_S, f, i_eps)
140         if somme > s_max:
141             f_note = f
142             s_max = somme
143     Mel_freq[i] = f_note
144
145
146 def freq_to_note(f):
147     if f == 0:
148         return -1
149     N_ref, f_ref = 69, 440
150     N = N_ref + 12 * np.log2(f/f_ref)
151     return round(N)
152
153 Mel = np.array([freq_to_note(f) for f in Mel_freq])
154
155
156 timelog["recherche melodie"] = time()
157
158 #####
159 # Recherche des notes #
160 #####
161
162 notes = []
163
164 delta_t_min = 0.125 # s
165 note_val, note_debut = -1, T[0]
166
167 for i, t in enumerate(T):
168     if Mel[i] == note_val:
169         continue
170
171     if note_val != -1 and (t - note_debut >= delta_t_min) and note_val > 45:
172
173         notes.append((note_val, note_debut, t))
174
175         note_val = int(Mel[i])
176         note_debut = t
177
178     if note_val != -1 and (t - note_debut >= delta_t_min) and note_val > 45:
179         notes.append((note_val, note_debut, t))
180
181 timelog["recherche evenements"] = time()
182
183 #####
184 # Creation de la partition #
185 #####
186
187 #octave sous forme [(step,alter)] : ('C',1) -> C# et ('B',-2) -> B double flat
188 octave_notes = [(['C',0], ['C',1], ['D',0], ['D',1], ['E',0], ['E',1], ['F',0],
189                  ['F',1], ['G',0], ['G',1], ['A',0], ['A',1], ['B',0],
190                  ['C',0], ['C',1], ['D',0], ['D',1], ['E',0], ['E',1], ['F',0], ['F',1], ['G',0], ['G',1], ['A',0], ['A',1], ['B',0]]
191
192 len_octave = 12 #12 demi-tons dans la gamme tempérée standard
193 octave_shift = -1 #alignement midi - numéro d'octave
194
195 def number_to_pitch(n):
196     step, alter = octave_notes[n%len_octave]
197     octave = n//len_octave + octave_shift
198     return step, alter, octave
199
200 def number_to_aff(n):
201     step12 = octave_notes[n%len_octave]
202     octave = n//len_octave + octave_shift
203     return step12 + str(octave)
204
205 start_time = min([t0 for _, t0, _ in notes])
206
207 quarter_duration = 4 #division d'un batttement : 4 -> analyse à la double croche près
208 duration_length = 1 / bps / quarter_duration #durée d'une unité de temps
209
210 def time_to_duration(t):
211     return round((t-start_time)/duration_length)
212
213 def duration_to_time(d):
214     return start_time + d * duration_length
215
216 part = pt.score.Part(id='t00', part_name="test00", quarter_duration=quarter_duration)
217
218 for i, (n, t0, t1) in enumerate(notes):
219     step, alter, octave = number_to_pitch(n)
220     note = pt.score.Note(id=f"n{i}", step=step, octave=octave, alter=alter)
221
222     start, end = time_to_duration(t0), time_to_duration(t1)
223     part.add(note, start, end)
224
225 part.add(pt.score.TimeSignature(4, 4), start=0)
226 pt.score.add_measures(part)
227 pt.score.tie_notes(part)
228
229 pt.save_musicxml(part, "partition_game.xml")

```

FIGURE 12 – Programme principal, partie 2/3.

```

231 timelog["creation partition"] = time()
232
233
234 #####
235 # Affichage #
236 #####
237
238 fig, axs = plt.subplots(4, 1, sharex=True, figsize=(8, 6))
239
240 axs[0].plot(t_fenetre, s_fenetre)
241 axs[0].set_title("Extrait Audio")
242
243 axs[1].plot(T, Mel, ".")
244
245 for n, t0, t1 in notes:
246     axs[2].hlines(n, t0, t1, color="red")
247     axs[2].text(t0, n + 0.5, str(n), color="red")
248
249 for i in range(int((t_fenetre[-1] - start_time)/duration_length)):
250     t = start_time + i*duration_length
251     if i%quarter_duration == 0:
252         axs[3].axvline(t, color="grey")
253     else:
254         axs[3].axvline(t, color="lightgrey")
255
256 for n, t0, t1 in notes:
257     t_start = duration_to_time(time_to_duration(t0))
258     t_end = duration_to_time(time_to_duration(t1))
259     axs[3].hlines(n, t_start, t_end, color='blue')
260     axs[3].text(t0, n + 0.5, number_to_aff(n), color="blue")
261
262
263 timelog['display'] = time()
264
265 deltaTimelog = {n: int((t-timelog["start"])*100)/100
266                 for (n, t) in timelog.items()}
267
268 print(deltaTimelog)
269
270 plt.show()
271

```

FIGURE 13 – Programme principal, partie 3/3.

```

1  # Code secondaire : calcul de D(t)
2
3  from time import time
4
5  timelog = {}
6  timelog["start"] = time()
7
8  from scipy.signal.windows import hann
9  from scipy.signal import find_peaks
10 from scipy.io import wavfile
11 import matplotlib.pyplot as plt
12 import numpy as np
13
14 timelog["import de modules"] = time()
15
16 file_name = "../../Matériel audio/Gamme.wav"
17
18 fs, raw_file = wavfile.read(file_name)
19 s = raw_file[:, 0] # conv to mono
20
21 timelog["import du fichier"] = time()
22
23 N = len(s) # nombre de points
24 dt = 1/fs # intervalle de temps entre mesures
25 t = np.arange(N) * dt
26
27 tmin, tmax = 0, 10 # s
28 imin, imax = int(tmin/dt), int(tmax/dt)
29 N_fenetre = imax - imin
30 s_fenetre = s[imin:imax]
31 t_fenetre = t[imin:imax]
32
33 s_max = np.max(s_fenetre)
34 s_carre = (s_fenetre/s_max)**2
35
36 timelog["fenetrage"] = time()
37
38 #####
39 # Intensité par fenêtre glissante #
40 #####
41 N_win = 2*11
42 hop = 2*6
43 win = hann(N_win)
44
45 n = (N_fenetre - N_win)//hop + 1
46
47 I = np.zeros(n)
48 T = np.zeros(n)
49
50 for j in range(n):
51     i_s = j*hop
52     i_e = i_s + N_win
53
54
55
56
57 T[j] = _t
58 I[j] = np.sqrt(np.mean(s_carre[i_s:i_e] * win))
59
60 #I = np.log(1 + I)
61
62 I_max = np.max(I)
63
64 timelog["Calcul de l'intensité"] = time()
65
66 #####
67 # Derivation #
68 #####
69 D = [(I[i+1]-I[i])/(hop*dt) for i in range(n-1)]
70 T_D = [(T[i+1]+T[i])/2 for i in range(n-1)]
71
72 timelog["Calcul de derivate"] = time()
73
74 #####
75 # Recherche de pics #
76 #####
77 prominence = 2
78 dist_s = 1/8
79 dist = dist_s / (hop*dt)
80
81 peaks, peak_props = find_peaks(D, prominence=prominence, distance=dist)
82
83 #####
84 # Affichage #
85 #####
86 fig, axes = plt.subplots(2, 1, sharex=True)
87
88 axes[0].plot(t_fenetre, s_fenetre/s_max, alpha=0.3, label=r"$s(t)$")
89 axes[0].plot(T, I/I_max, label=r"$A(t)$")
90 axes[0].legend(fontsize="14")
91
92 axes[1].plot(T_D, D, label=r"$D(t)$")
93 for i, p in enumerate(peaks):
94     axes[1].plot(T_D[p], D[p], 'rx')
95
96 axes[1].vlines(T_D[p], ymax=D[p],
97               ymin=D[p]-peak_props['prominences'][i],
98               color='r', alpha=0.5)
99 axes[1].hlines(y=D[p]-prominence,
100               xmin=T_D[p]-dist_s/2, xmax=T_D[p]+dist_s/2,
101               color='g', alpha=0.5)
102 axes[1].legend(fontsize="14")
103 for ax in axes:
104     ax.tick_params(left=False, right=False, labelleft=False)
105 axes[1].set_xlabel("temps en secondes", size='xx-large')
106
107 timelog['display'] = time()
108 deltaTimelog = {n: int((t-timelog["start"])*100)/100
109                 for (n, t) in timelog.items()}
110 print(deltaTimelog)

```

FIGURE 14 – Programme du calcul de $D(t)$, correspondant à la figure 8.