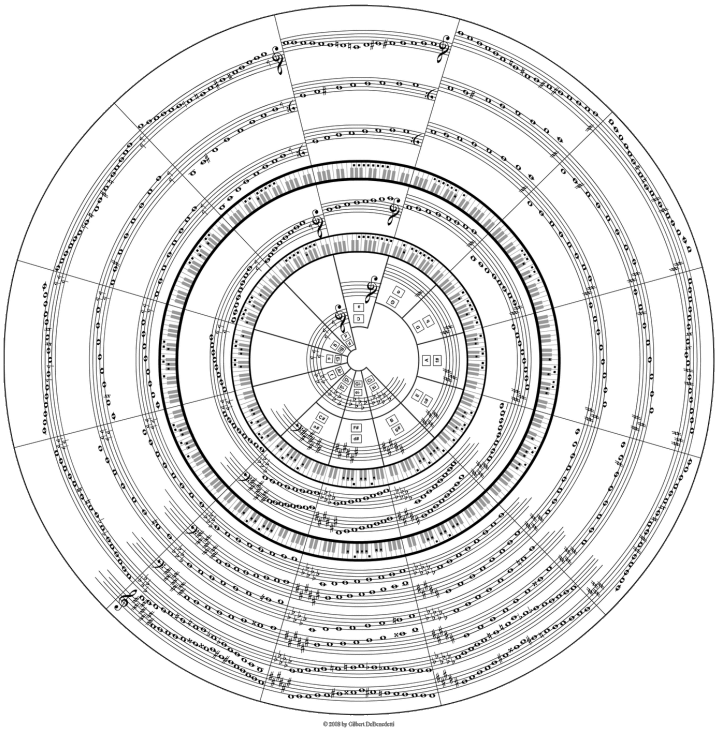
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN NORMANDIE

Département Génie Mathématique

Projet de transcription de musique

Rapport d'avancement du projet semestriel



Rand ASSWAD
Ergi DIBRA
Yuge SUN

A l'attention de :
Mme. Natalie FORTIER

Contents

1 Introduction

1.1 Idée du projet

Le but du projet est de créer un logiciel de transcription automatique de morceaux de musique, dans le cadre du projet semestriel on vise créer un version capable de traiter des morceaux monophoniques.

1.2 Idée de traitement

On part du principe qu'un signal sonore est une série harmonique, ce qui sera expliqué en détails dans le rapport final du projet.

A partir du signal harmonique on extrait la fréqence fondamentale en fonction du temps, par la suite on transforme les fréqences en notes. Par la suite, on analyse les notes pour obtenir une suite de notes associées avec des durées.

L'étape finale fait appelle à la théorie de musique, elle consiste à extraire le *tempo* à partir des durées des notes, et de reconnaitre la *gamme* du morceau à partir des notes obtenues.

2 Le traitement effectué

2.1 Cas initial

2.1.1 Partie théorique

Le signal obtenu en entrée est un signal harmonique. C'est-à-dire qu'une note simple de durée T secondes est donnée par un signal

$$u(t) = \sum_{k\in\mathbb{N}} A_k \cdot \cos(2\pi k f_0 t), \forall t \in [0, T]$$

où f_0 est la fréquence fondamentale du signal sur $[0, T]$.

Un morceau musical peut être vu comme une suite de notes où chaque note est caractérisée par sa fréquence et sa durée. En divisant le morceau en N parties, on définit la suite $(t_i)_{i=0,\dots,N-1}$ telle que $t_i = i \cdot \frac{T}{N}$, le signal devient

$$x(t) = \sum_{i=0}^{N-1} \underbrace{\sum_{k\in\mathbb{N}} A_{k,i} \cdot \cos(2\pi k f_{0,i} t)}_{u_i(t)} \cdot \mathbb{1}_{[t_i, t_{i+1}[}(t)$$

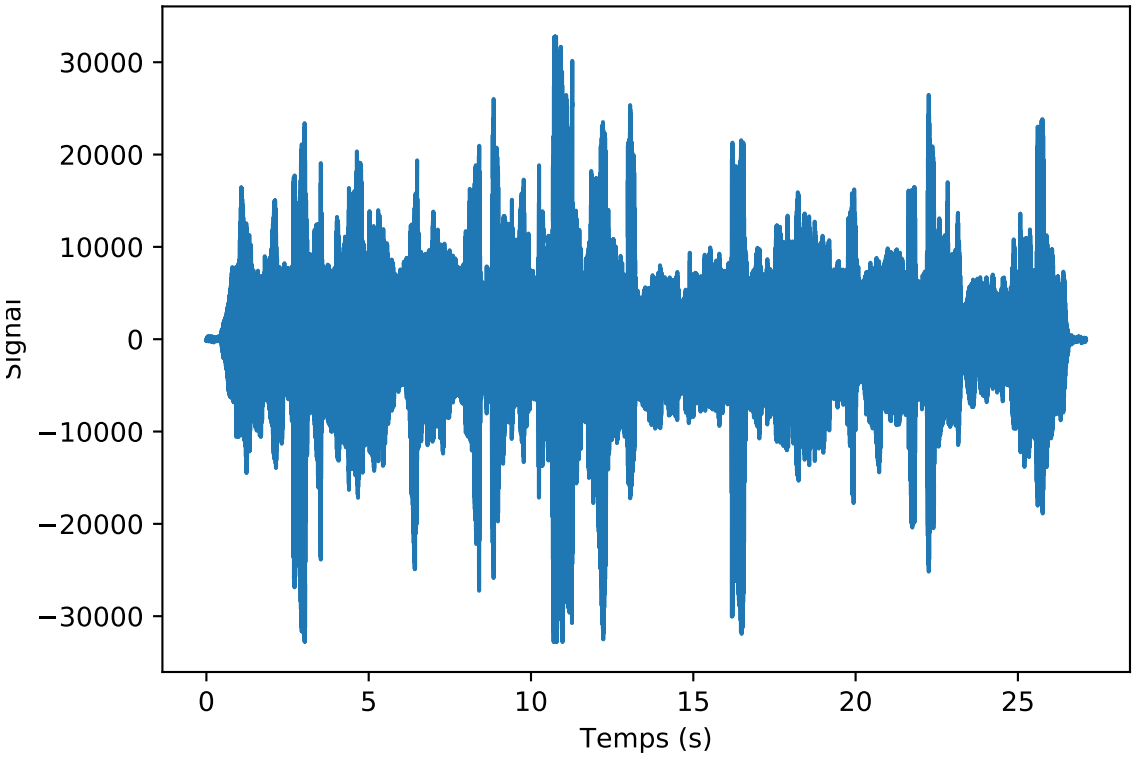
2.1.2 Partie appliquée

Le stockage de signaux sonore consiste à prélever des valeurs à intervalles définis. En général, l'échantillonnage est régulier, souvent fixé à 44100 Hz (échantillons par seconde).

```
from scipy.io import wavfile
from matplotlib import pyplot as plt
import numpy as np
inputFile = project_path + 'sounds/reiding-cut.wav'
# morceau exemple
fs, x = wavfile.read(inputFile)
# fs = échantillinage
# x = signal discrétisé
```

```
## /usr/lib/python3.6/site-packages/scipy/io/wavfile.py:273: WavFileWarning: Chunk (non-data) not unde
##   WavFileWarning)
```

```
t = np.arange(x.size) / float(fs)
# t = temps discrétisé
plt.plot(t, x)
```



2.2 Détection des fréquences fondamentales

Ils existent plusieurs algorithmes de détection de fréqences fondamentales, il y'en a deux types : applications sur le domaine temporel et sur le domaine fréquenciel. Chaque type présente des avantages et des inconvénients, dans le cas d'un signal monophone les algorithmes temporels sont privilégiés.

Après de nombreuses recherches on a décidé d'appliquer l'algorithme de **YIN** *(Kawahara et de Cheveigné, 2002)* car il est rapide, efficace et produit des erreurs moins importantes comparé avec d'autres algorithmes. Son principe se base sur la fonction d'autocorrélation.

Les étapes de l'algorithme :

1. La méthode d'autocorrelation
2. La fonction de différences
3. La fonction de la moyenne normalisée cumulée
4. Le seuil absolu
5. L'interpolation parabolique
6. La meilleure estimation locale

On expliquera la méthode en détails dans le rapport final du projet, et on étudiera l'erreur de cette méthode.

Voyons une l'application de l'algorithme sur l'exemple précédent.

```
from src.pitch import YIN
time, pitch = YIN(fs, x)
plt.plot(time, pitch)
```

```
## /usr/lib/python3.6/site-packages/scipy/io/wavfile.py:273: WavFileWarning: Chunk (non-data) not unde
##   WavFileWarning)
```

2.3 Analyse de notes

2.3.1 Introduction du problème

L'espace de notes est un espace linéaire discrèt, mais l'espace de fréquences est continue non-linéaire. Le problème consiste à trouver une fonction qui associe les fréquences fondamentales obtenues avec des valeurs entières.

2.3.2 Gammes et intervalles

En acoustique, un **intervalle** désigne le rapport de fréquences de deux sons. Or, chaque intervalle est caractéristique d'une échelle musicale, elle-même varie selon le type de musique.

En musique, une **gamme** (*en:* **scale**) est une suite de notes conjointes où la fréquence de la dernière est le double de celle de la première. Une gamme se caractérise par sa première note et la suite d'intervalles qui séparent les notes conjointes.

L'**armure** — ou l'**armature** (*en:* **key signature**) — est un ensemble d'altérations réunies à la clé. Elle est composée soit exclusivement de dièses, soit exclusivement de bémols — en dehors du cas particulier constitué par le changement d'armure. Ces altérations correspondent à la tonalité principale des mesures suivant la clé.

À chaque tonalité majeure est associée une tonalité en mode mineur, présentant la même armure de clef et appelée relative mineure.

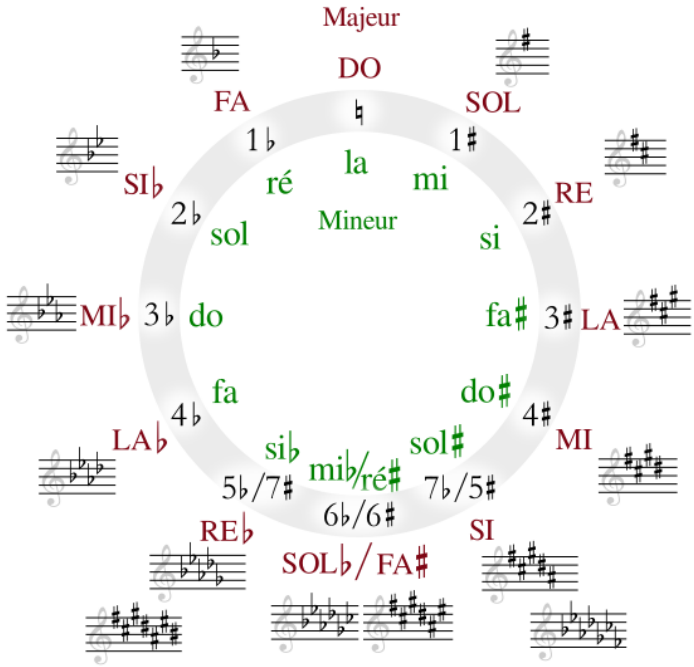


Figure 1: La cycle des quintes

Pour simplifier, on va considérer la théorie de la musique occidentale basée sur l'accord tempéré (depuis le XVIII^{e} siècle). Dans ce cas, l'intervalle séparant la première et la dernière note d'une gamme est dite *octave*, une octave se divise en 12 écarts égales appelés *demi-tons*. La dernière note porte le même nom de la première dans la gamme.

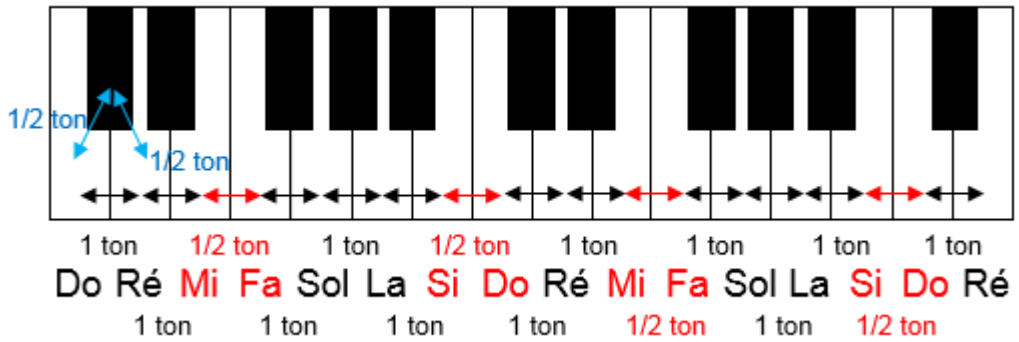


Figure 2: Les intervalles sur un piano

2.3.3 Nomenclature

Ils existent plusieurs systèmes de nomenclature de notes de musique. Le système utilisé en France adopte les noms en termes de *Do-Ré-Mi-Fa-Sol-La-Si*. De plus, il existe un système basé sur l'alphabet latin : *C-D-E-F-G-A-B*. Les deux systèmes sont très utilisés, dans ce projet on utilisera le dernière pour simplifier.

Vu que les noms des notes se répètent au bout d'un octave, il faut distinguer une note LA de fréquence $440Hz$ d'une autre de fréquence $220Hz$ ou $880Hz$.

Le système de notation scientifique **Scientific Pitch Notation** identifie une note par sont nom alphabetique avec un nombre identifiant l'octave dans laquelle elle se situe, où l'octave commence par une note C. Par exemple la fréquence $440Hz$ représente A_4 sans ambiguité, et les fréquences $220Hz$ et $880Hz$ représentent les notes A_3, A_5 respectivement.

Dans le protocole **MIDI**, le notes sont représentées par un nombre entier, il permet de coder plus de 10 octave en partant de la note C_{-1}.

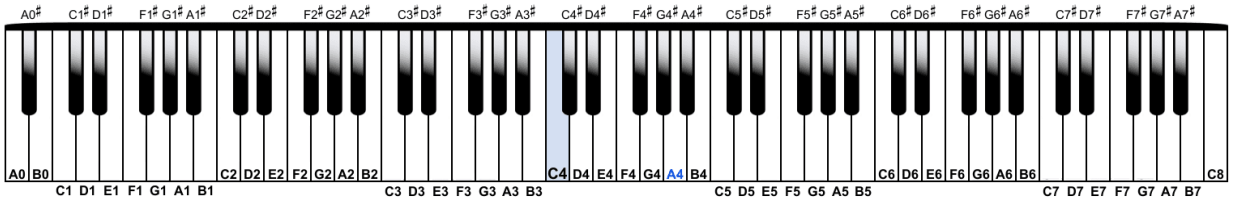


Figure 3: La notation scientifique des notes sur un piano

2.3.4 Reconnaissance des notes

Un demi-ton est l'écart entre deux touches voisines sur un piano. On voudrais savoir le rapport r de fréquences associé à un demi-ton, sachant que l'octave double la fréquence on peut conclure facilement :

$$r^{12} = 2 \Rightarrow r = 2^{1/12}$$

On souhaite ramener l'espace de fréquences $(\mathbb{R}, \times)$ à l'espace $(\mathbb{N}, +)$ tel que $\boxed{\text{demi-ton} \equiv 1}$. On définit donc une bijection

$$\forall f \in]0, \infty[, f \mapsto 12 \log_2 f$$

En arrondissant le résultat à la valeur entière la plus proche, on obtient un espace linéaire discrèt correspondant aux notes.

Il sera convenient d'obtenir les mêmes notes du protocole **MIDI** vu qu'il est très bien établi et très utilisé. Pour cela, on effectue une petite translation, en partant de la note de référence $A4 \equiv 69_{\text{MIDI}} \equiv 440Hz$.

$$\begin{cases} \varphi : f \mapsto 12 \log_2 f + c_{\text{ref}} \\ \varphi(440) = 69 \end{cases} \Rightarrow c_{\text{ref}} = 69 - 12 \log_2 440$$

Par conséquent, la bijection φ est définit par :

$$\varphi :]0, \infty[\to \mathbb{R} : f \mapsto 12 \log_2 f + c_{\text{ref}} \quad \text{avec } c_{\text{ref}} = 69 - 12 \log_2 440$$

On note $\bar{\varphi}$ la fonction définit par $\bar{\varphi}(f) = \lfloor \varphi(f) \rceil \in \mathbb{Z}$ où $\lfloor \cdot \rceil$ est la fonction d'arrondissement à l'entier le plus proche.

On peut donc obtenir les nombres MIDI de notes à partir des fréquences fondamentales grâce à la fonction $\bar{\varphi}$.

```
def pitch_to_midi(frequency, A4=440, silence_threshold=10):
    if frequency < silence_threshold:
        return -1
    else:
        return round(12*log2(frequency) + 69 - 12*log2(A4))
def midi_to_note(nb_midi):
    if nb_midi==-1:
        octave = None
        note = nb_midi
    else:
        octave = nb_midi // 12
        note = nb_midi % 12
    return note, octave
notes_midi = [pitch_to_midi(f) for f in pitch]
plt.plot(time, notes_midi)
```

Néanmoins, le nombre MIDI n'est pas suffisant pour identifier une note, car certaines notes ont la même fréquence en accord tempéré et donc le même nombre midi (i.e. la même touche sur un piano), par exemple MIDI($C\#$) = MIDI($D\flat$). Pour distinguer ces notes il est nécessaire de trouver la gamme du morceau.

2.3.5 Reconnaissance de la gamme/l'armature

Dans cette étude, on ne s'intéressera aux notes dans une octave. On introduit donc la fonction ψ :

$$\psi :]0, \infty[\to [0, 12[: f \mapsto \psi(f) \mod 12$$

De même, on définit la fonction $\bar{\psi}$ telle que $\bar{\psi}(f) = \lfloor \psi(f) \rceil$. On voit que $\text{Im}(\bar{\psi}) = \mathbb{Z}/12\mathbb{Z}$

Note	C		B		E	F		G		A		B
$\bar{\psi}(f)$	0	1	2	3	4	5	6	7	8	9	10	11

En musique classique, ils existent 4 types de gammes, on ne s'intéressera qu'à un : *la gamme majeure*. Comme on l'a déjà dit, une gamme est caractérisée par sa première note et la suite des intervalles. Dans la gamme majeure, les intérvalles en fonction du ton sont : 1–1–½–1–1–1–½.

La gamme *Do/C Majeur* contient donc les notes {0, 2, 4, 5, 7, 9, 11}.

De même, la gamme *Sol/G Majeur* contient les notes {7, 9, 11, 0, 2, 4, 6}. Ces gammes diffèrent par une note, la note 5≡F est remplacée par la note 6 qui correspond à F# ou G♭. Dans le contexte du Sol Majeur, on sait que 6≡F# car la gamme contient déjà 7≡G.

On voit bien que l'identification de la gamme est *nécessaire* pour la distinction entre certaines notes.

Une gamme peut être alors identifié par son ensemble de notes qu'on notera G tel que $G \subset \mathbb{Z}/12\mathbb{Z}, |G| = 7$. On définit le vecteur $g \in \{0, 1\}^{12}$ associé à G tel que

$$g_i = \mathbb{1}_G(i) = \begin{cases} 1 & \text{si } i \in G \\ 0 & \text{sinon} \end{cases}$$

On définit donc E l'ensemble de gammes majeures.

Soit F l'ensemble de fréquences fondamentales obtenues par l'algorithme de YIN, soit $S = \bar{\psi}(F) \subset \mathbb{Z}/12\mathbb{Z}$, soit $p : Z/12\mathbb{Z} \to \mathbb{N} : n \mapsto$ le nombre d'occurances de n dans le morceau. On note $p_{\max} = \max_{n \in S} p(n)$. On définit le vecteur $x \in [0,1]^{12}$ tel que $x_i = \frac{p(i)}{p_{\max}}$.

La gamme du morceau est alors la solution du problème d'optimisation

$$\min_{g \in E} \quad \|g - x\|$$

En musique classique, $|E| = 12$ donc le problème d'optimisation ne nécessite pas une résolution mathémtique avancée.

2.3.6 Implémentation

On définit la classe `Scale` qui définit une type d'objets *gamme majeure*. Une instance de `Scale` contient l'indice de sa première note `Scale.base_note`, et le vecteur $g \in 0, 1^{12}$ associé `Scale.vect`.

```
class Scale(object):
    def __init__(self, note_index):
        # Constructeur de gamme majeur à partir d'une note
        self.base_note = note_index
        self.vect = np.array(shift_array([1,0,1,0,1,1,0,1,0,1,0,1], note_index))
```

On extrait le vecteur x associé au morceau.

```
scale_vect = [0] * 12
for nb in notes_midi:
    note, octave = midi_to_note(nb)
    if note > -1:
        scale_vect[note] += 1
scale_vect = scale_vect/max(scale_vect)
```

On définit une fonction qui extrait la gamme en résolvant le problème d'optimisation.

```
def find_scale(scale_vect):
    # Fonction qui prend le vecteur x associé au morceau
    # retourne un objet de classe Scale
    E = [Scale(i) for i in range(12)]
    # liste de gammes majeurs
    d = [norm(scale_vect - g.vect) for g in E]
    # liste de distances entre le vecteur x et les gammes de E
    index_min = argmin(d)
    # l'indexe de la gamme la plus proche à x
    return E[index_min]
```

On applique cette fonction au morceau étudié, Concerto Op. 35 en Si mineur pour violon par Oskar Rieding. La gamme *Si mineur* possède deux dièses à l'armure (cf. cycle des quintes), on espère obtenir le même résultat:

```
scale = find_scale(scale_vect)
print(scale)
```

```
## Key signature: 2 #
```

Ayant identifié l'armure du morceau, on peut obtenir les notes exactes à partir du nombre midi.

3 Le traitement en cours de développement

3.1 Post-traitement

Les résultats obtenus sont certainement intéressants. Or, il y a des notes courtes enregistrés qui ne sont que du bruit capté lors des changements de notes. On a essayé d'éliminer ces notes en supprimant les notes courtes, mais on a perdu certaines notes importantes.

Nous avons observé une relation entre les maximums locaux de l'envelope du signal et le changement de notes, ce qui nous a poussé à trouver une solution qui permet d'identifier ces changements correctement. Nous avons donc cherché une méthode mathématique perméttant d'extraire telle fonction. Par conséquent, on a trouvé plusieurs méthodes intéressantes dites en anglais **Onset Detection**.

Notre idée de base ne constitue qu'une partie de la solution. En effet, l'identification des *Onsets* ou *Débuts* s'effectue en deux étapes:

1. Calculer une fonction dite **Onset Detection Function** ou parfois **Onset Strength Signal (OSS)** dont les maximums locaux correspondent aux *Onsets*.
2. **Peak Selection** : il s'agit d'une méthode de sélection de maximums locaux.

Ils existent plusieurs fonctions OSS utilisées dans le domaine temporel, fréquentiel, ou complexe. Qui permettent de quantifier l'énergie du signal et/ou les changements dans la fréquence fondamentale.

Pour la deuxième étape, on introduit souvent un seuil afin d'éliminer les points avec un OSS petit. En général, le seuil n'est pas constant, il se calcul à partir de la fonction OSS par des méthodes statistiques comme la moyenne mobile.

On choisira après les points de l'OSS supérieurs au seuil et maximals dans un voisinage petit qu'on fixe.

3.2 La reconnaissance du rythme

Sans le rythme, la partition ne peut être écrite. Vu qu'on n'a pas encore fini le traitement, nous ne pouvons pas passer à cette étape, le projet se restreint à un outil de conversion en format MIDI.

Si vous pouvez nous guider dans cette étape, ça sera encore mieux pour nous.

3.3 Etape finale

Une fois le *tempo* indentifié, les résultats se transforment facilement en format MusicXML qui peut être extrait facilement en format PDF grâce aux libraries existantes.

Références

[1] Alain de Cheveigné et Hideki Kawahara.
YIN, a fundamental frequency estimator for speech and music, 2002.

[2] Carlos Rosão, Ricardo Ribeiro, et David Martins de Matos.
Influence of peak selection methods on onset detection, 2012.

[3] Chris Duxbury, Juan Pablo Bello, Mike Davies and Mark Sandler.
Complex domain onset detection for musical signals, 2003.