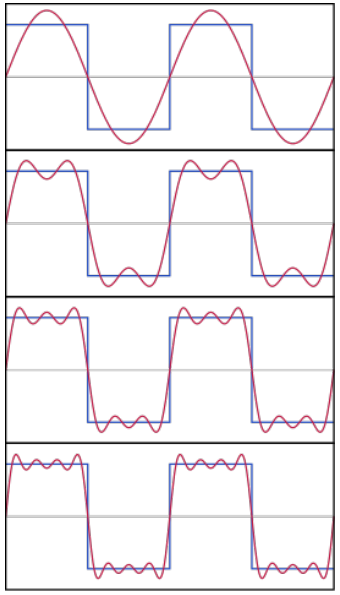
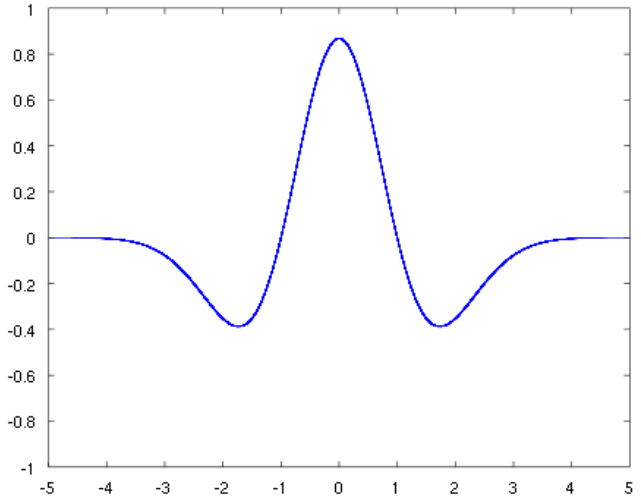
Algorithmes de production et d'identification d'empreintes de sons musicaux





Outils analytiques utilisés

Théorie de Fourier

Théorie des ondelettes

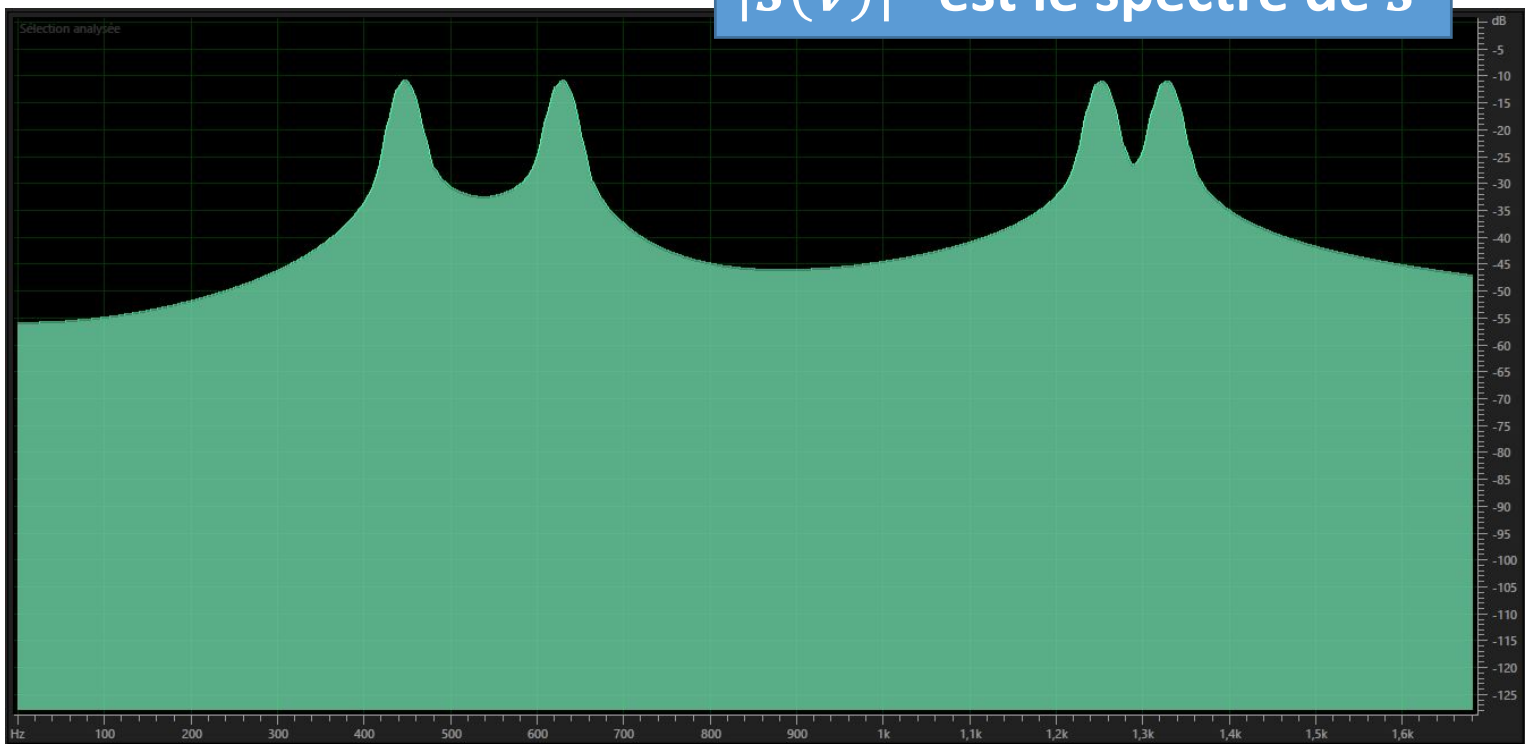
Transformation de Fourier

- $\forall f, g \in L^2(\mathbb{C}), < f, g > = \int_{-\infty}^{+\infty} f(t)\overline{g(t)}\, dt$
- **Transformée de Fourier de s(t) :**

$$\hat{s} : \nu \longmapsto < s, e^{i2\pi\nu t} >$$

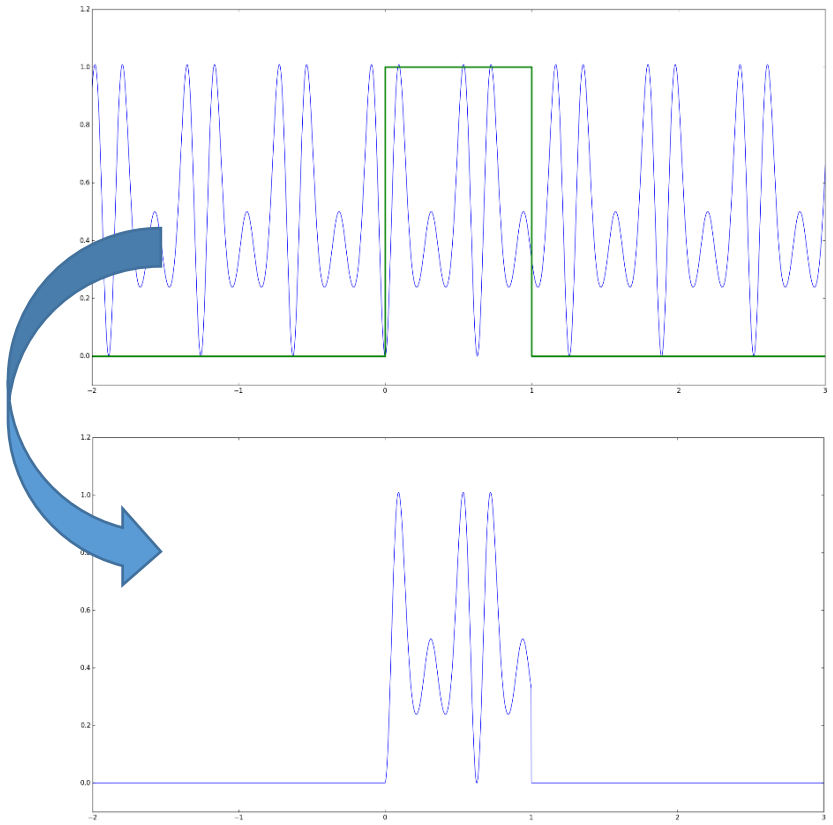
i.e. $\hat{s} : \nu \to \int_{-\infty}^{+\infty} s(t)e^{-i2\pi\nu t}dt$

$|\hat{s}(\nu)|^2$ est le spectre de s



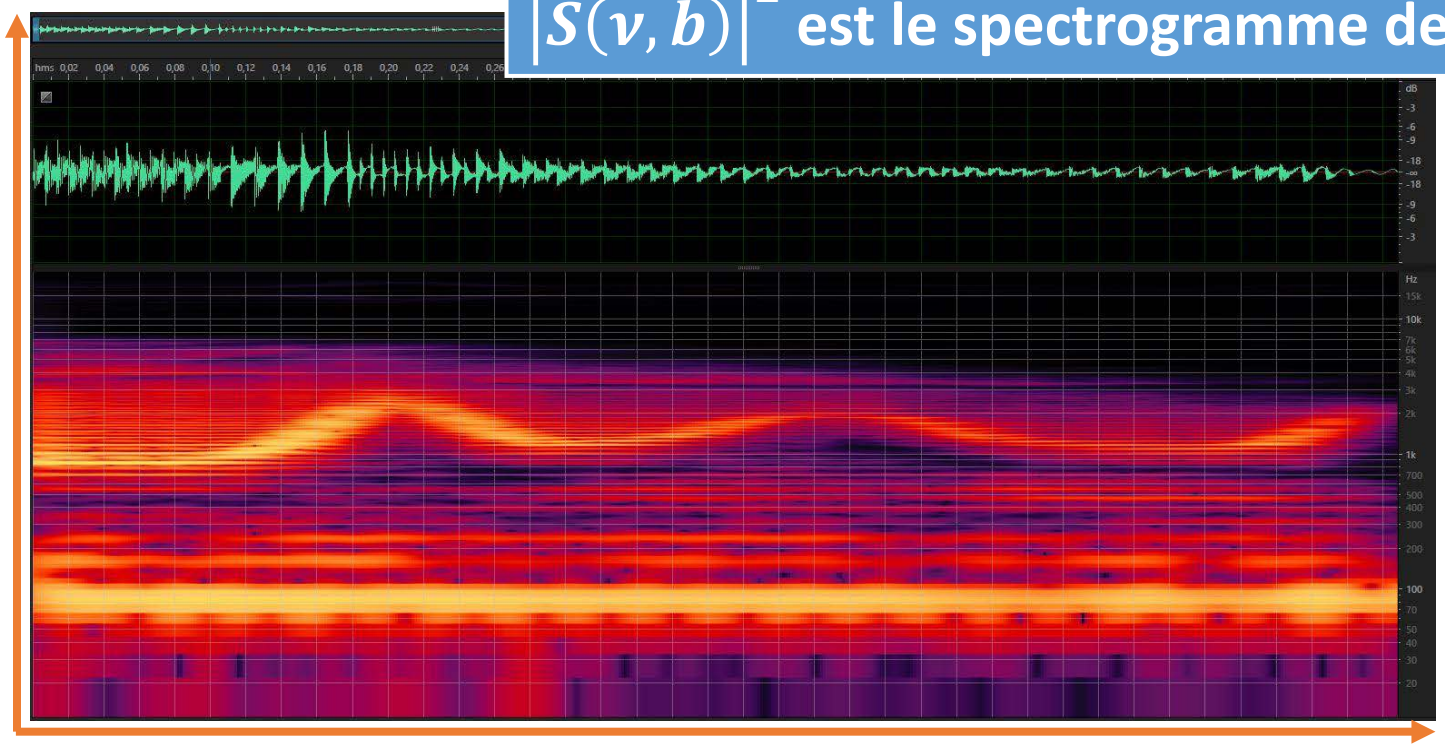
Transformation de Fourier à fenêtre glissante

- $S: (\nu, b) \longmapsto < s(t)g(t-b), e^{i2\pi\nu t} > = \int_{-\infty}^{+\infty} s(t)g(t-b)e^{-i2\pi\nu t}dt$
- **Fenêtre rectangulaire :** $g(t) = \begin{cases} 1 \; si \; t \in [0, T] \\ 0 \; sinon \end{cases}$



ν

$|\widehat{S}(\nu, b)|^2$ est le spectrogramme de s



b

Analyse multirésolution par ondelettes (AMR)

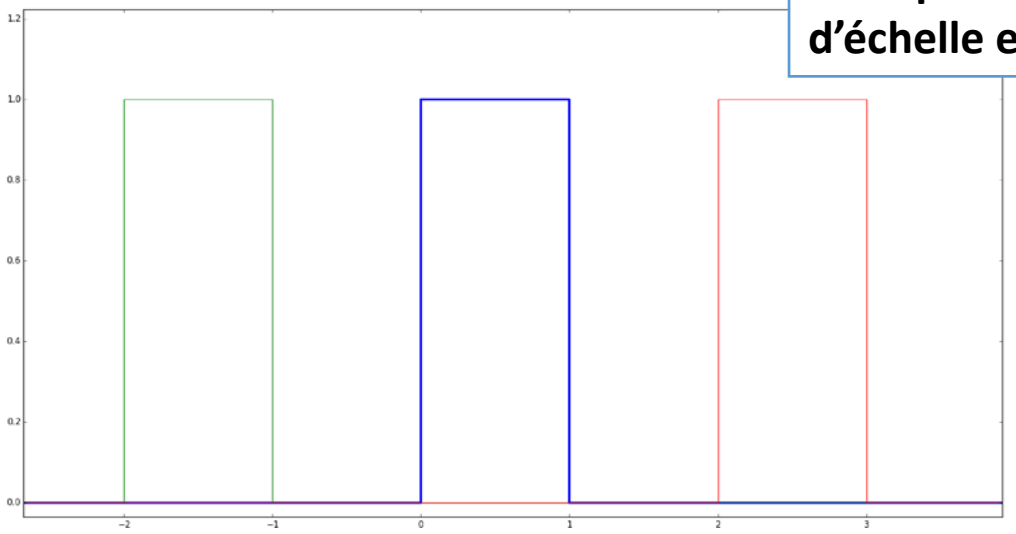
- **Une AMR est une suite de sous-espaces vectoriels fermés de $L^2(\mathbb{R})$ telle que :**
  - **(1)** $\forall \mathbf{j} \in \mathbb{Z}, \mathbf{V_{j+1}} \subset \mathbf{V_j}$
  - **(2)** $\lim_{j \to +\infty} V_j = \{0\}$ **et** $\overline{\lim_{j \to -\infty} V_j} = L^2(\mathbb{R})$
  - **(3)** $\forall \mathbf{f(t)} \in \mathbf{L^2}(\mathbb{R}), f(t) \in V_j \Leftrightarrow \mathbf{f}\left(\frac{\mathbf{t}}{\mathbf{2}}\right) \in V_{j+1}$
  - **(4)** $\exists \varphi(t) \in L^2(\mathbb{R}); \{\varphi(t-k), k \in \mathbb{Z}\}$ ***est une base orthogonale de*** V_0

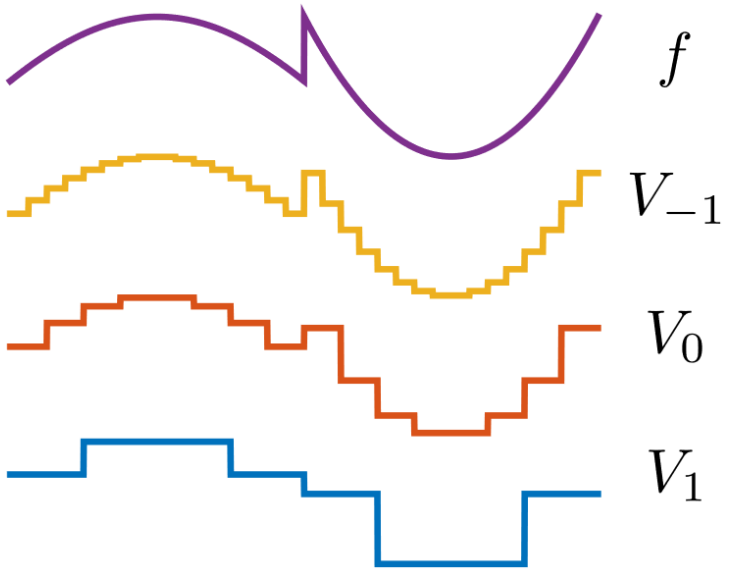
fonction d'échelle
ondelette père

Les espaces V_j sont définis par

$V_j = \{f, f \text{ constante sur } [k2^j, (k+1)2^j]\}$

Exemple où la fonction d'échelle est de Haar



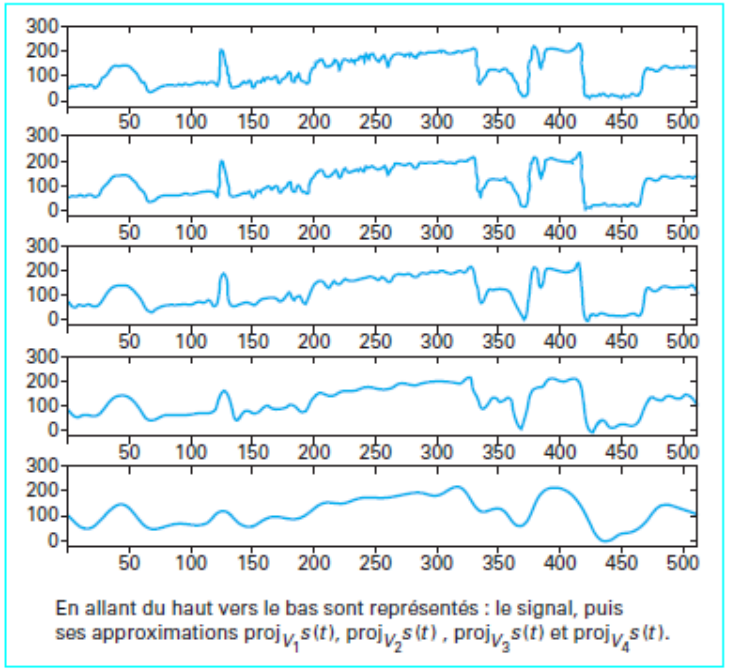


$$V_{j-1} = V_j \oplus^{\perp} W_j$$

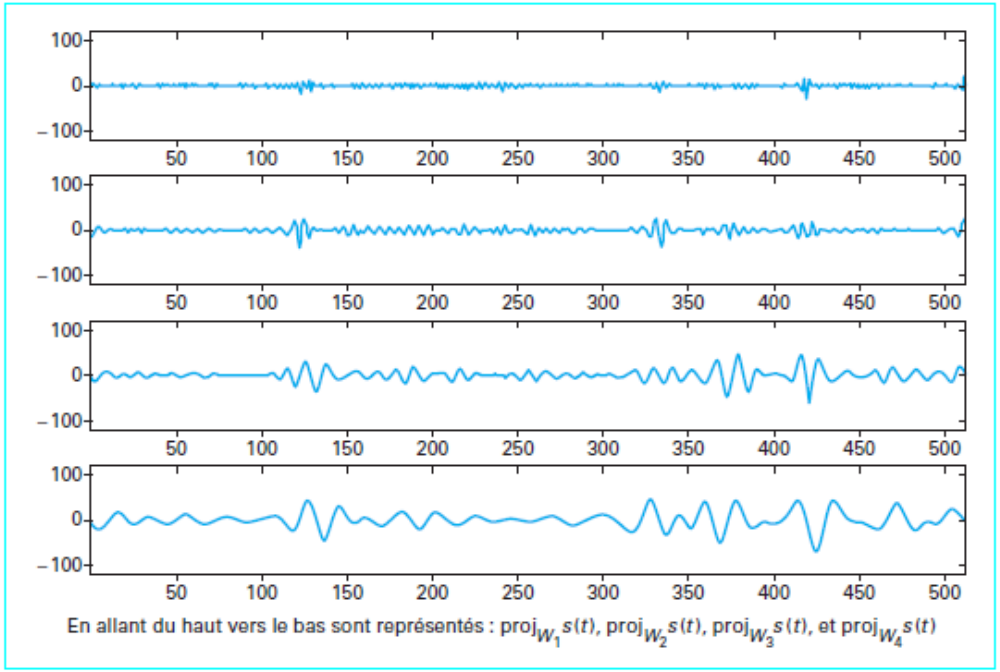
$$proj_{V_{j-1}} s(t) = proj_{V_j} s(t) + proj_{W_j} s(t)$$

coefficients d'ondelettes

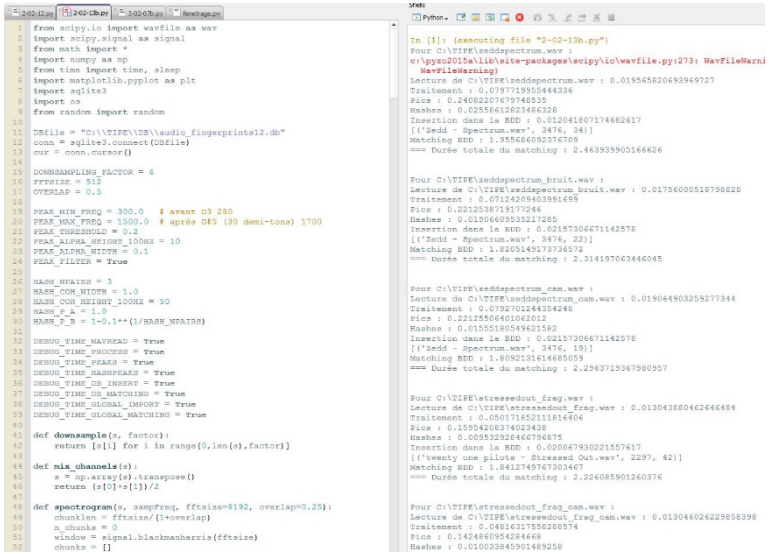
Exemple d'AMR



En allant du haut vers le bas sont représentés : le signal, puis ses approximations $\text{proj}_{V_1} s(t)$, $\text{proj}_{V_2} s(t)$, $\text{proj}_{V_3} s(t)$ et $\text{proj}_{V_4} s(t)$.



En allant du haut vers le bas sont représentés : $\text{proj}_{W_1} s(t)$, $\text{proj}_{W_2} s(t)$, $\text{proj}_{W_3} s(t)$, et $\text{proj}_{W_4} s(t)$



```
from scipy.io import wavfile as wav
import scipy.signal as signal
from math import *
import numpy as np
from time import time, sleep
import matplotlib.pyplot as plt
import sqlite3
import os
from random import random

DBfile = "C:\\TIPE\\DB\\audio_fingerprints12.db"
conn = sqlite3.connect(DBfile)
cur = conn.cursor()

DOWNSAMPLING_FACTOR = 6
FFTSIZE = 512
OVERLAP = 0.5

PEAK_MIN_FREQ = 300.0   # avant D3 280
PEAK_MAX_FREQ = 1500.0  # après G#5 (30 demi-tons) 1700
PEAK_THRESHOLD = 0.2
PEAK_ALPHA_HEIGHT_100Hz = 10
PEAK_ALPHA_WIDTH = 0.1
PEAK_FILTER = True

HASH_NPAIRS = 3
HASH_CON_WIDTH = 1.0
HASH_CON_HEIGHT_100Hz = 50
HASH_F_A = 1.0
HASH_F_B = 1-0.1**(1/HASH_NPAIRS)

DEBUG_TIME_WAVREAD = True
DEBUG_TIME_PROCESS = True
DEBUG_TIME_PEAKS = True
DEBUG_TIME_HASHPEAKS = True
DEBUG_TIME_DB_INSERT = True
DEBUG_TIME_DB_MATCHING = True
DEBUG_TIME_GLOBAL_IMPORT = True
DEBUG_TIME_GLOBAL_MATCHING = True

def downsample(s, factor):
    return [s[i] for i in range(0,len(s),factor)]

def mix_channels(s):
    s = np.array(s).transpose()
    return (s[0]+s[1])/2

def spectrogram(s, sampFreq, fftsize=8192, overlap=0.25):
    chunklen = fftsize/(1+overlap)
    n_chunks = 0
    window = signal.blackmanharris(fftsize)
    chunks = []
```

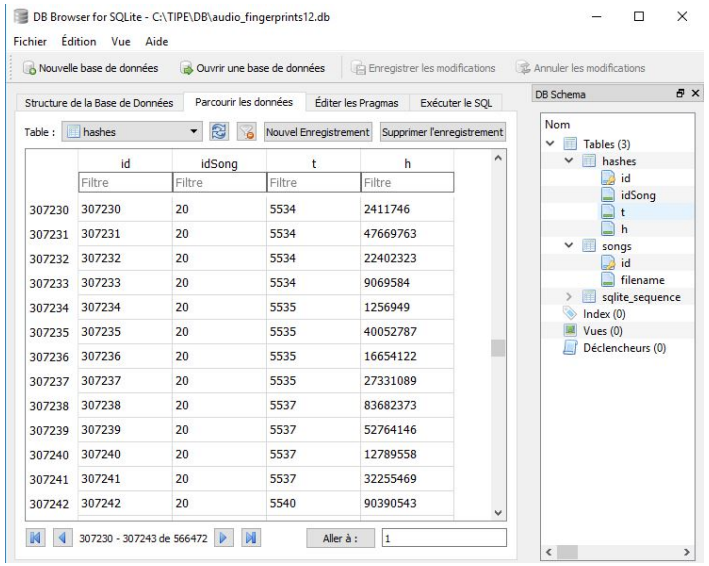
```
In [1]: (executing file "2-02-13b.py")
Pour C:\TIPE\zeddspectrum.wav :
c:\pysz2015a\lib\site-packages\scipy\io\wavfile.py:273: WavFileWarni
  WavFileWarning)
Lecture de C:\TIPE\zeddspectrum.wav : 0.019565820693969727
Traitement : 0.07977199555444336
Pics : 0.2408220767979486535
Hashes : 0.025586126234863326
Insertion dans la BDD : 0.012041807174682617
[('Zedd - Spectrum.wav', 3476, 34)]
Matching BDD : 1.955686092376709
=== Durée totale du matching : 2.463939905166626


Pour C:\TIPE\zeddspectrum_bruit.wav :
Lecture de C:\TIPE\zeddspectrum_bruit.wav : 0.017560005187988828
Traitement : 0.07124209403991699
Pics : 0.2212538719177246
Hashes : 0.013066005325317285
Insertion dans la BDD : 0.021573066711425781
[('Zedd - Spectrum.wav', 3476, 22)]
Matching BDD : 1.8205149173736572
=== Durée totale du matching : 2.3141970634460645


Pour C:\TIPE\zeddspectrum_cam.wav :
Lecture de C:\TIPE\zeddspectrum_cam.wav : 0.019064903259277344
Traitement : 0.07927012443542480
Pics : 0.2212550640106201212
Hashes : 0.015551805496215820
Insertion dans la BDD : 0.021573066711425781
[('Zedd - Spectrum.wav', 3476, 19)]
Matching BDD : 1.8092135620117188
=== Durée totale du matching : 2.2963719367980957


Pour C:\TIPE\stressedout_frag.wav :
Lecture de C:\TIPE\stressedout_frag.wav : 0.013043880462646484
Traitement : 0.05017185211181640625
Pics : 0.1155420827865600586
Hashes : 0.009532928466796875
Insertion dans la BDD : 0.020067930221557617
[('Twenty one pilots - Stressed Out.wav', 2297, 42)]
Matching BDD : 1.8412749767303467
=== Durée totale du matching : 2.226085901260376


Pour C:\TIPE\stressedout_frag_cam.wav :
Lecture de C:\TIPE\stressedout_frag_cam.wav : 0.013046026229858398
Traitement : 0.04816317558288574
Pics : 0.1428604125976562
Hashes : 0.010033845901489258
```



Réalisation des algorithmes

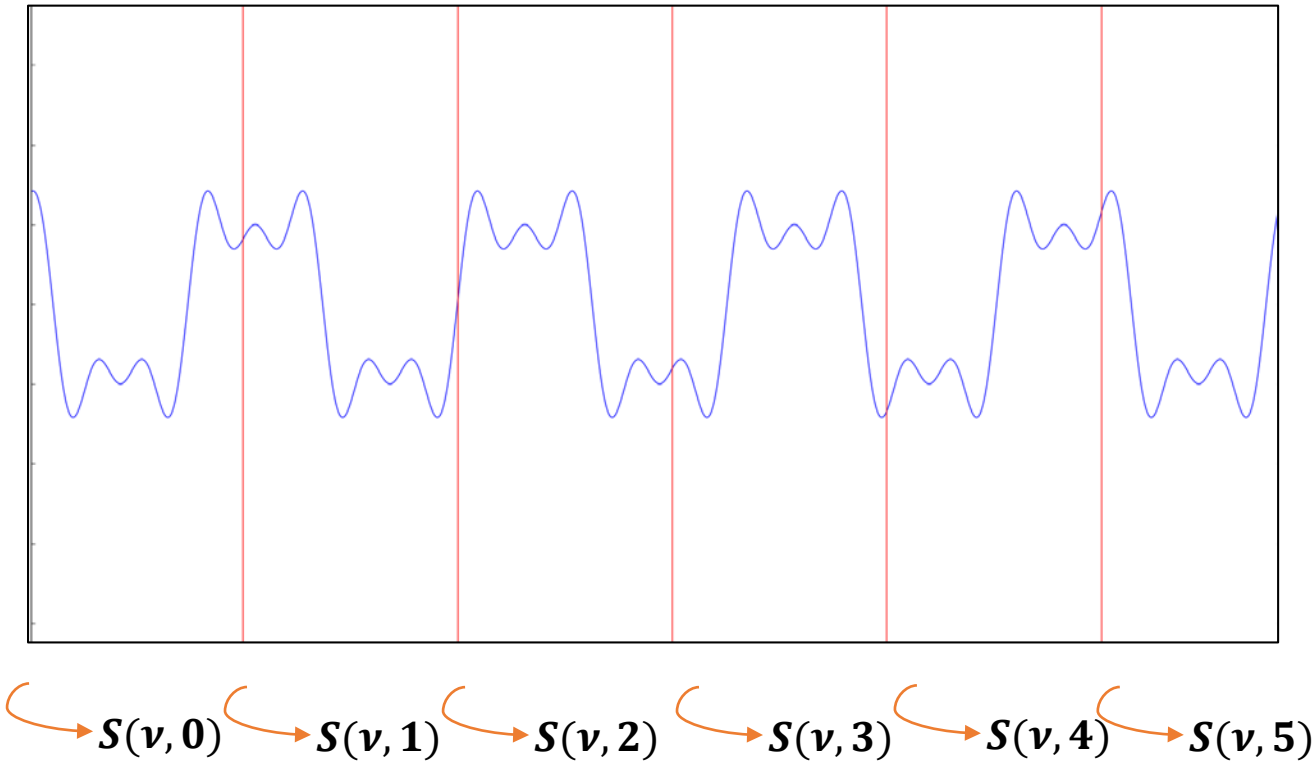
3 approches : 2 via la TFFG, 1 via l'AMR

Principe

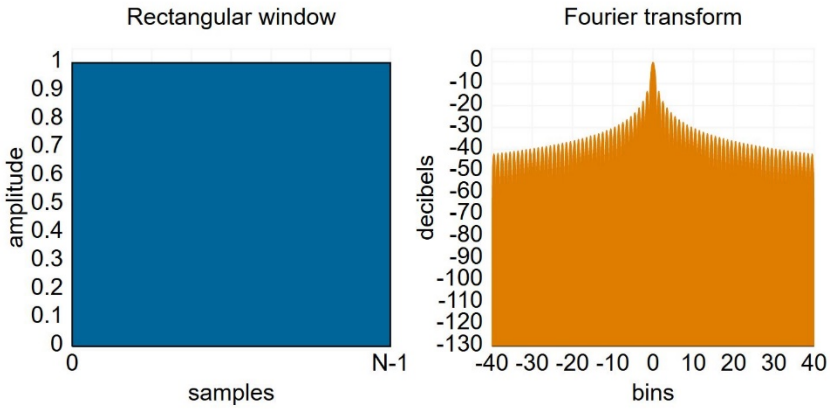
1) **Extraire l'information utile des signaux (les *features*)**
2) **Adapter cette information de sorte à ce qu'elle soit stockée dans une base de données : étape de hachage**
3) **Enregistrer l'information / l'utiliser pour identifier un extrait, à l'aide de requêtes SQL**

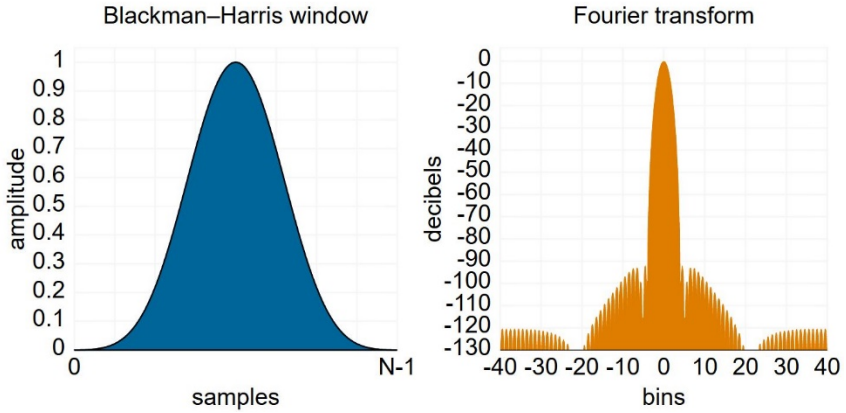
Calcul du spectrogramme

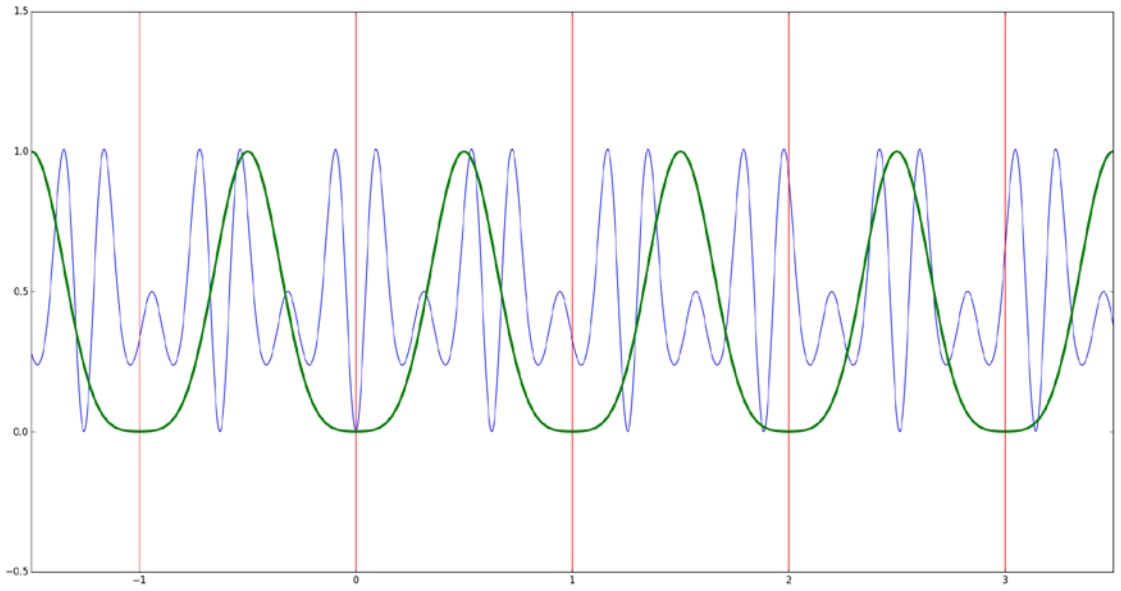
- **On découpe le signal en subdivisions régulières**
- **On applique la transformée de Fourier à chaque subdivision**

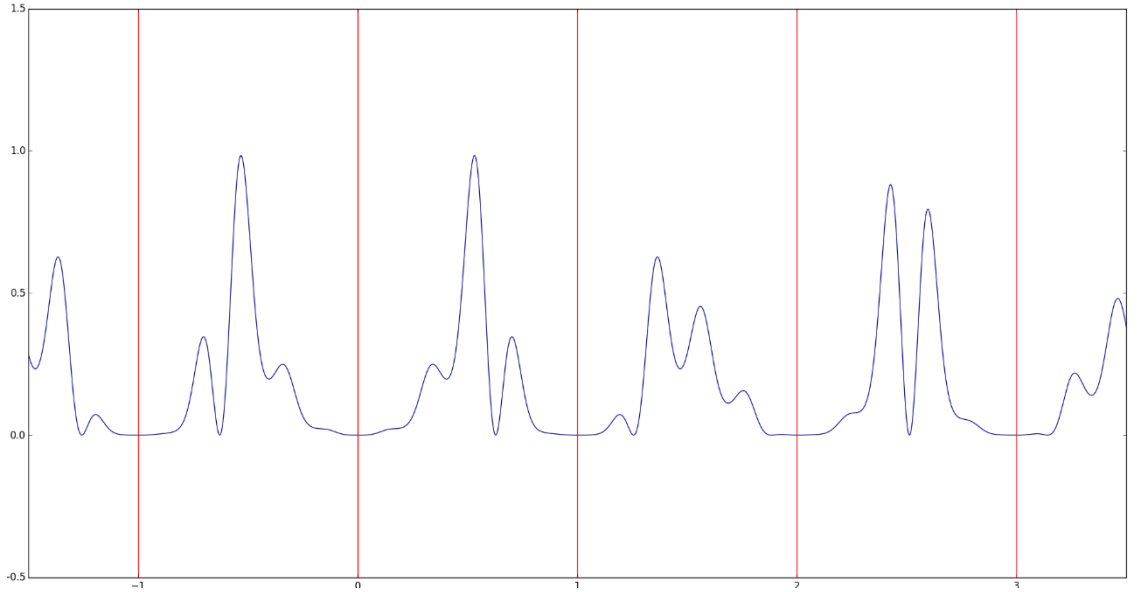


- **On applique à chaque subdivision une fenêtre de Blackmann-Harris**



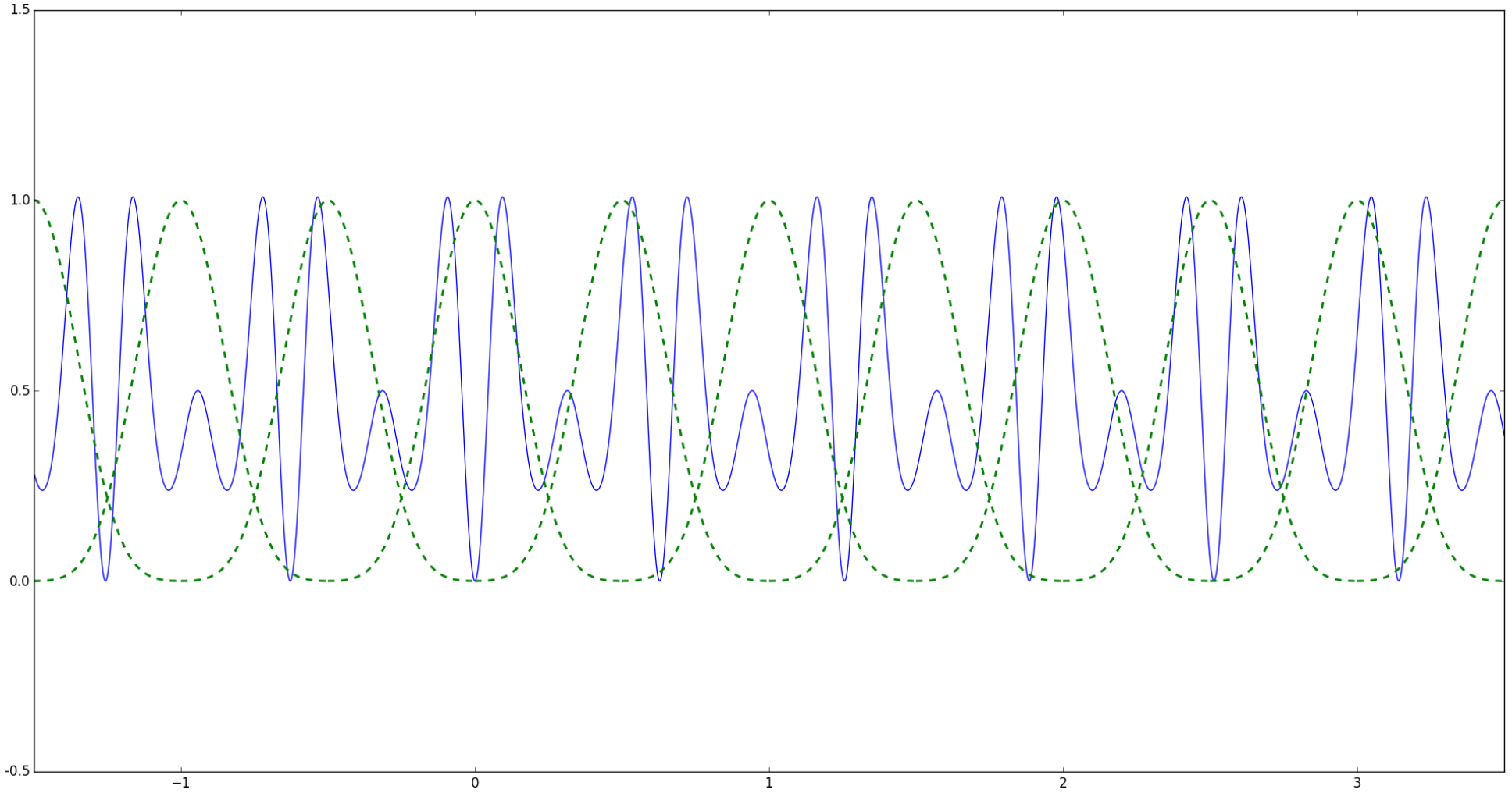






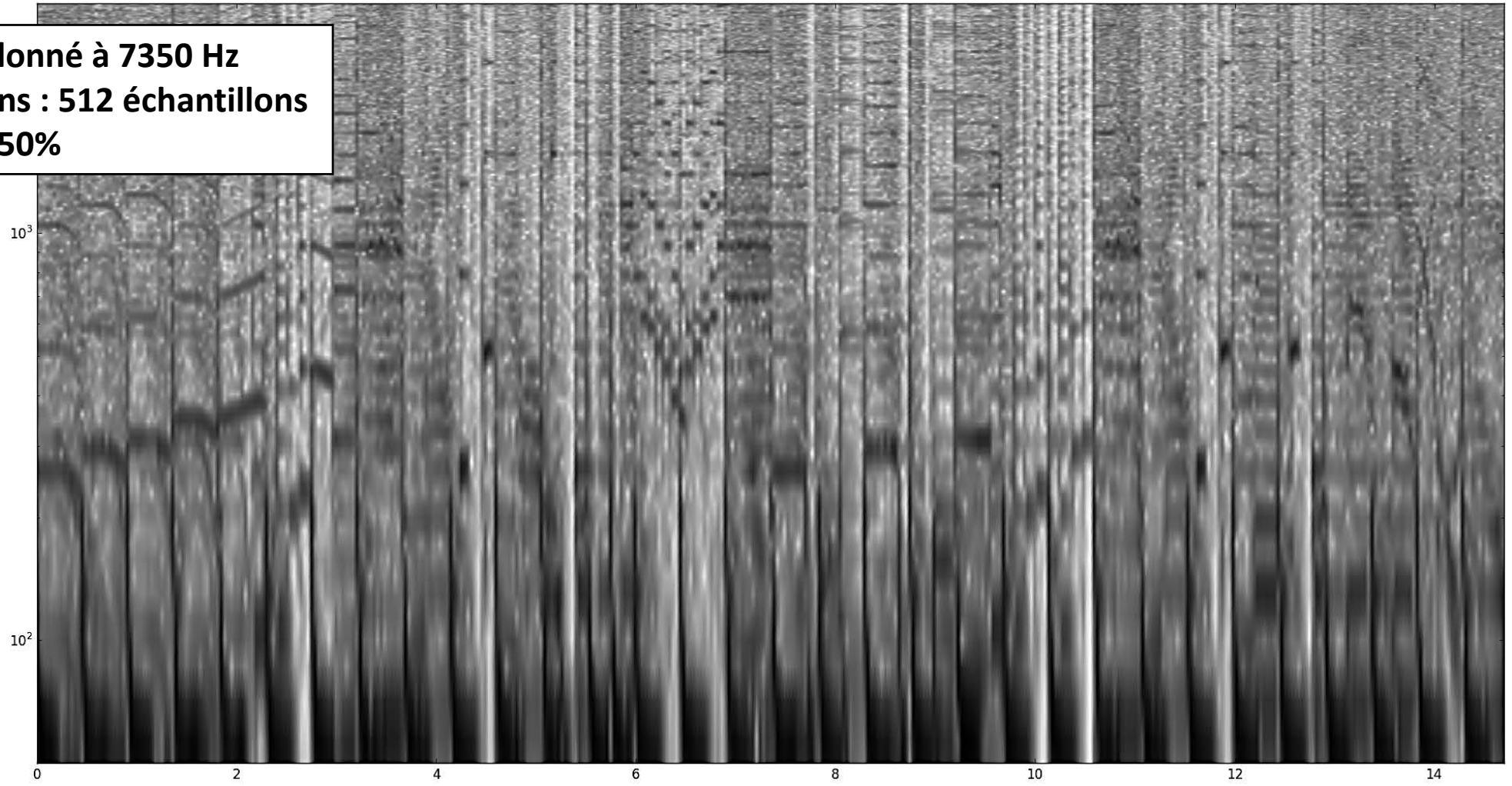
$S(\nu, 0)$ $S(\nu, 1)$ $S(\nu, 2)$ $S(\nu, 3)$ $S(\nu, 4)$ $S(\nu, 5)$

- **On fait se chevaucher les subdivisions (on en rajoute)**



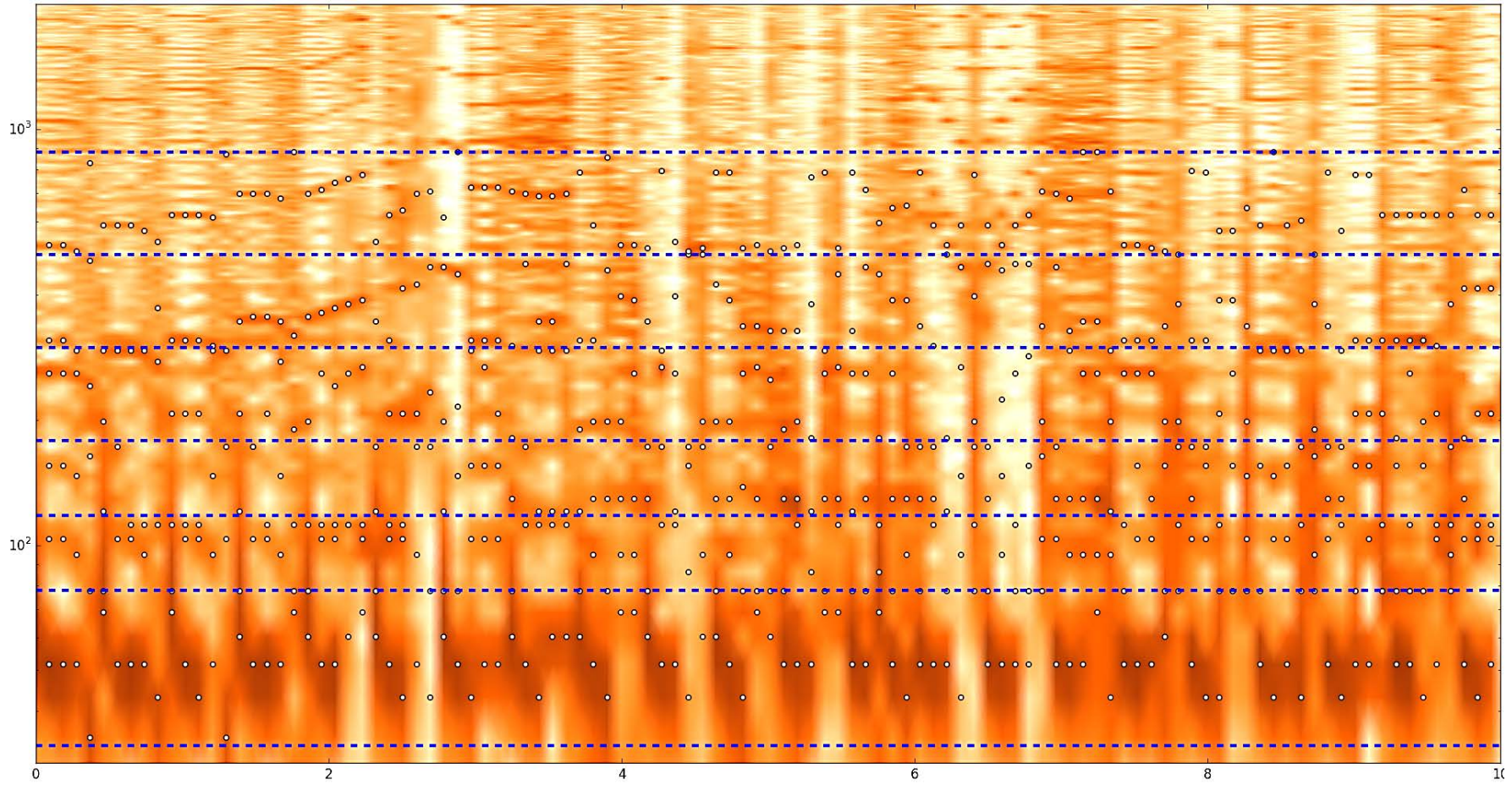
Exemple de spectrogramme obtenu

Signal sous-échantillonné à 7350 Hz
Taille des subdivisions : 512 échantillons
Chevauchement de 50%

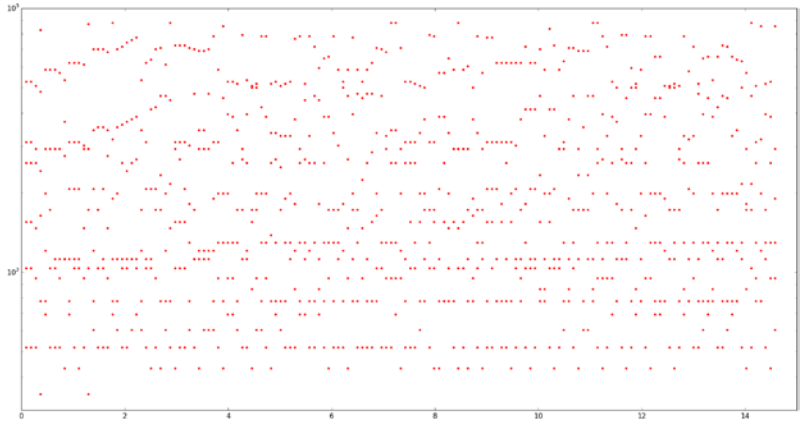


Première approche

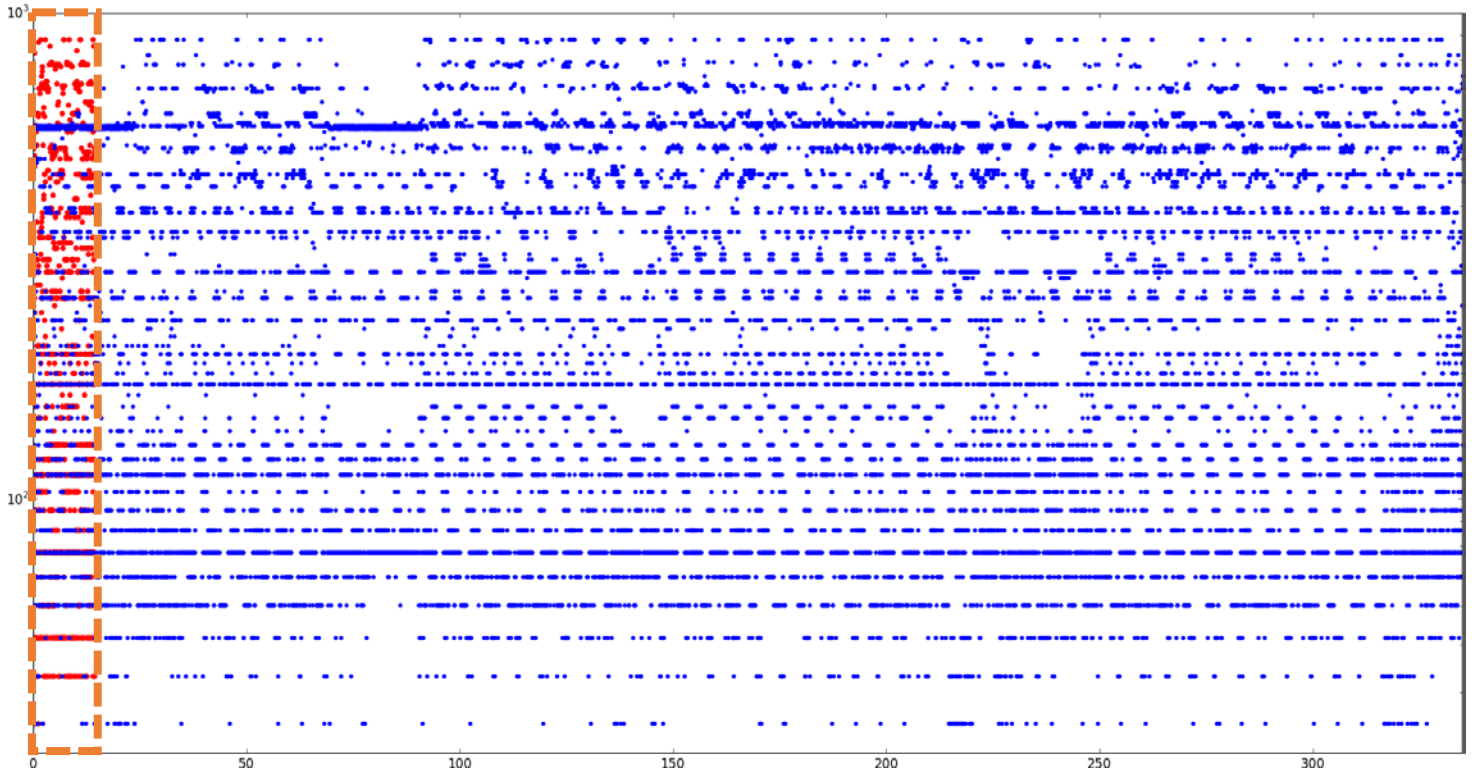
- **On découpe le spectrogramme en 6 bandes de fréquences, et à chaque instant, on cherche le point de chaque bande de fréquences où l'intensité est maximale à cet instant.**
- **Si ce maximum est supérieur à 60%, on le considère comme un pic d'amplitude.**



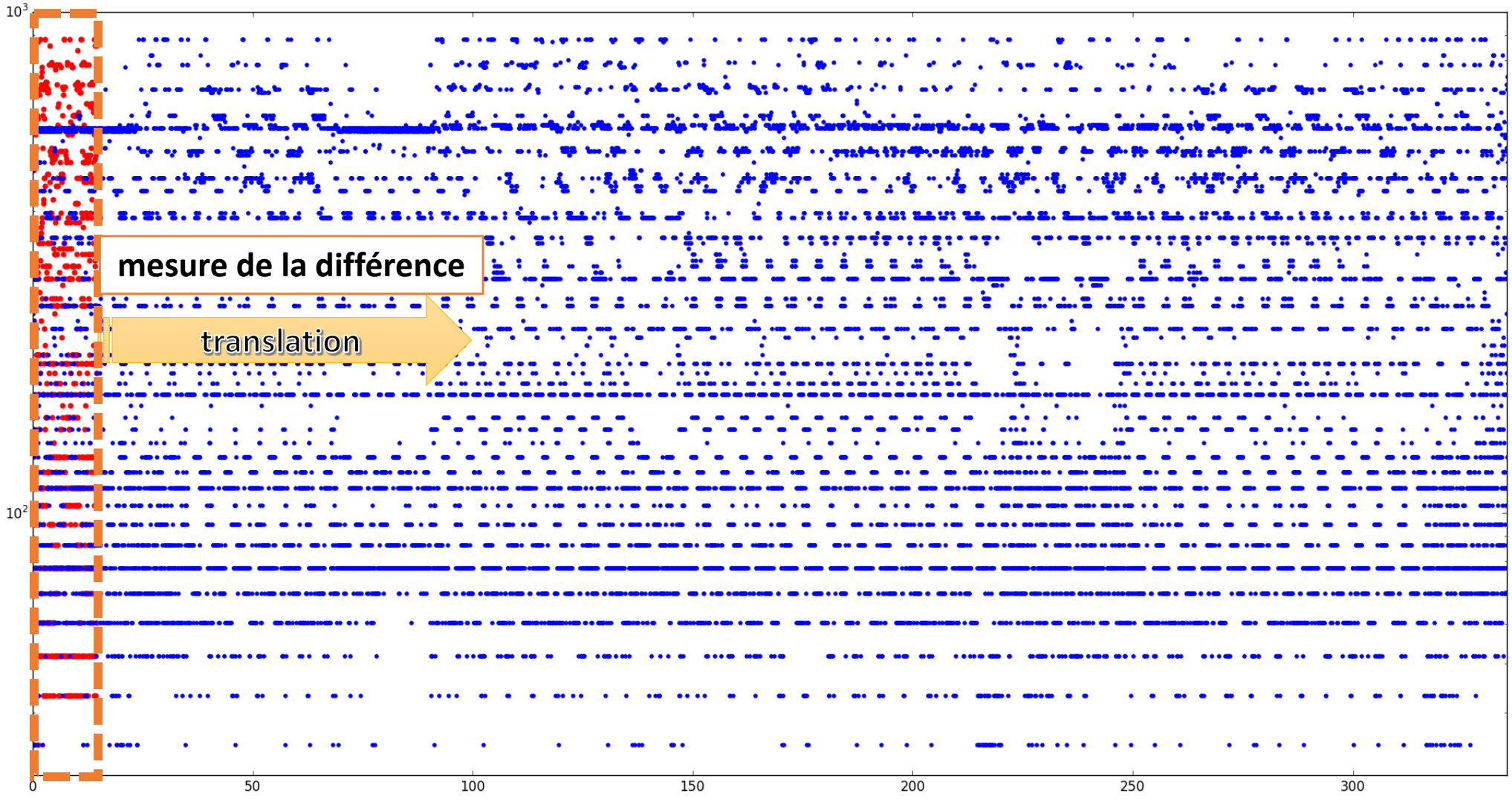
- **On obtient donc un nuage de points**

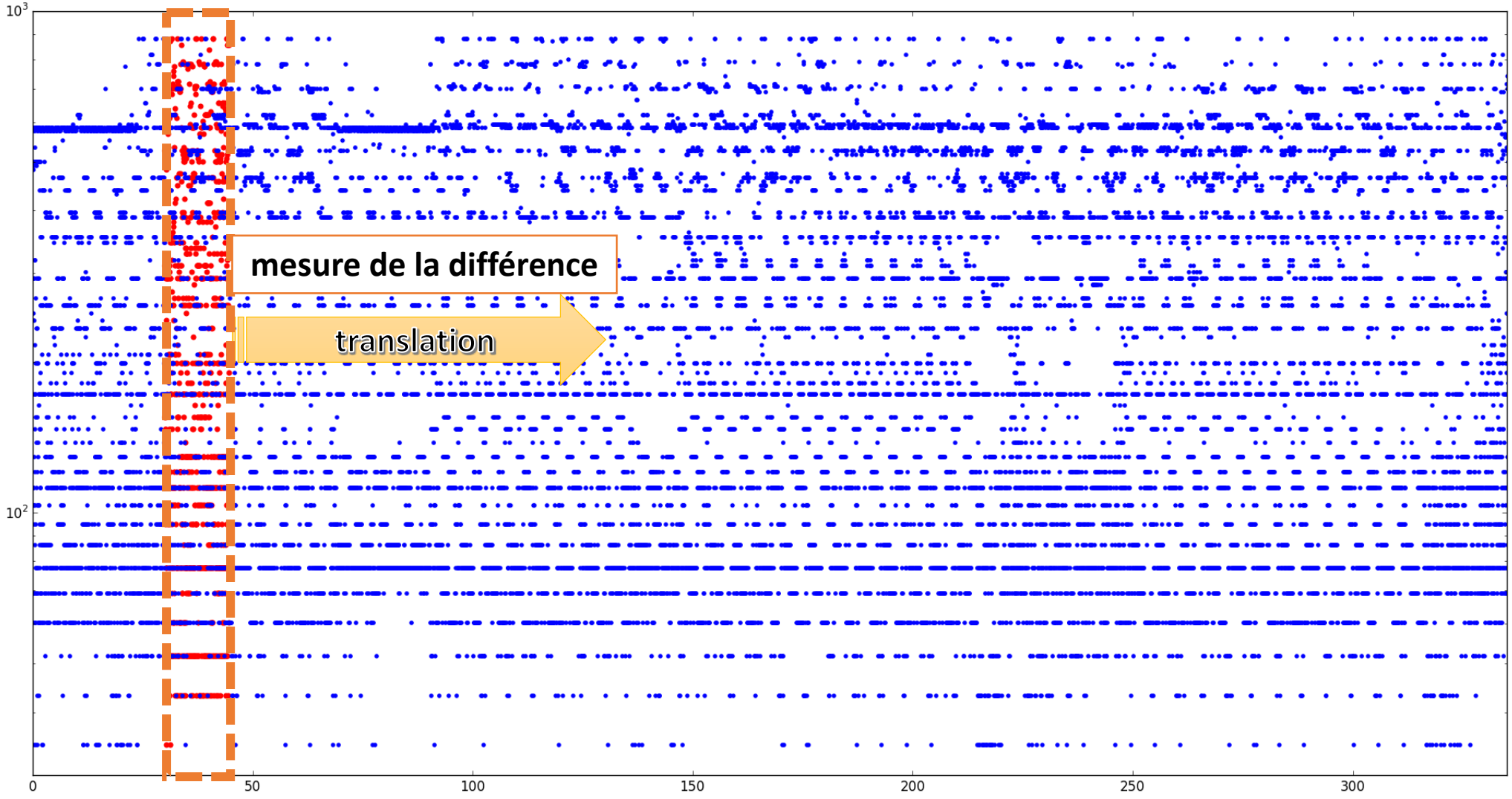


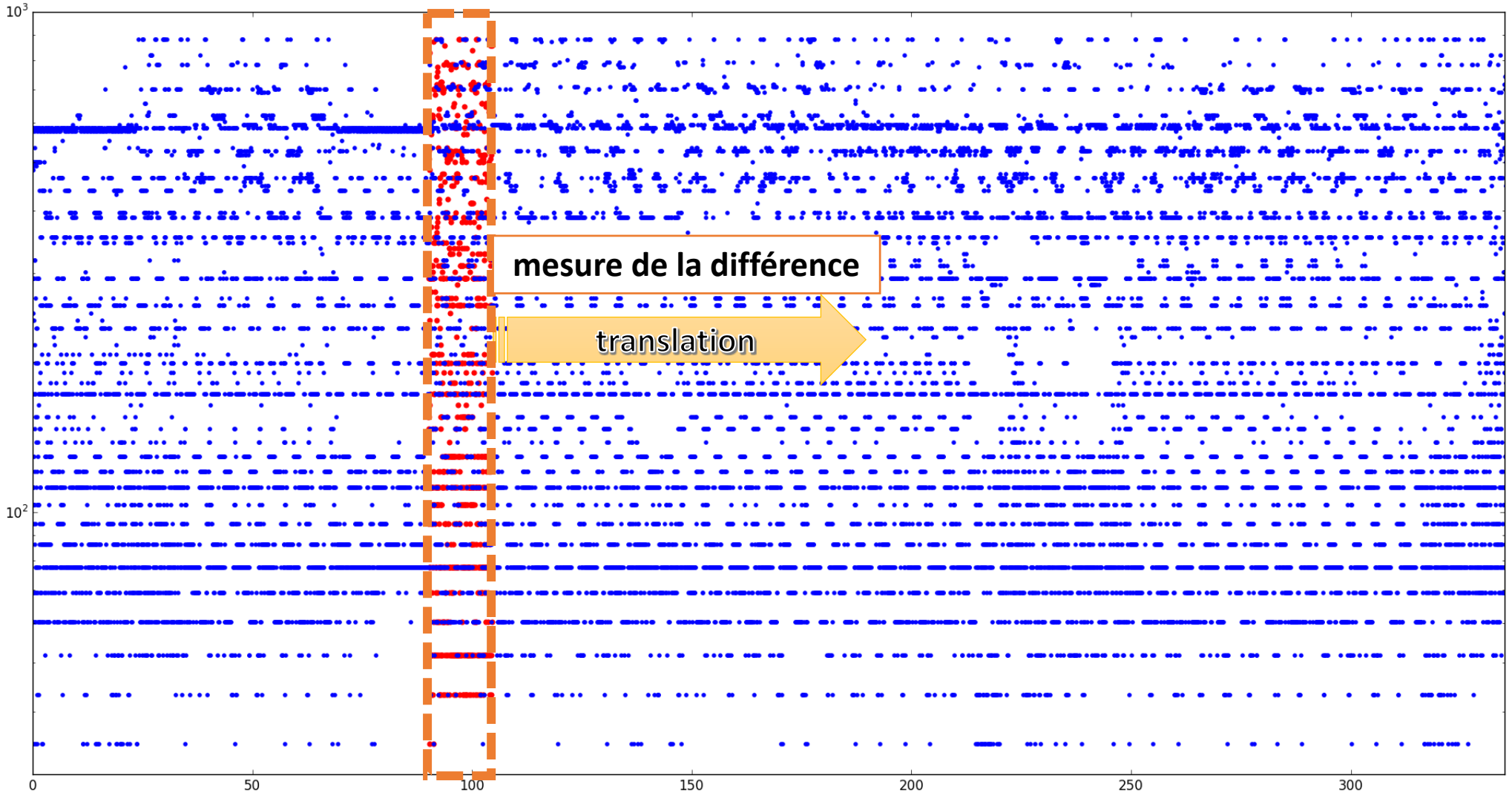
nuage de pics de l'extrait analysé



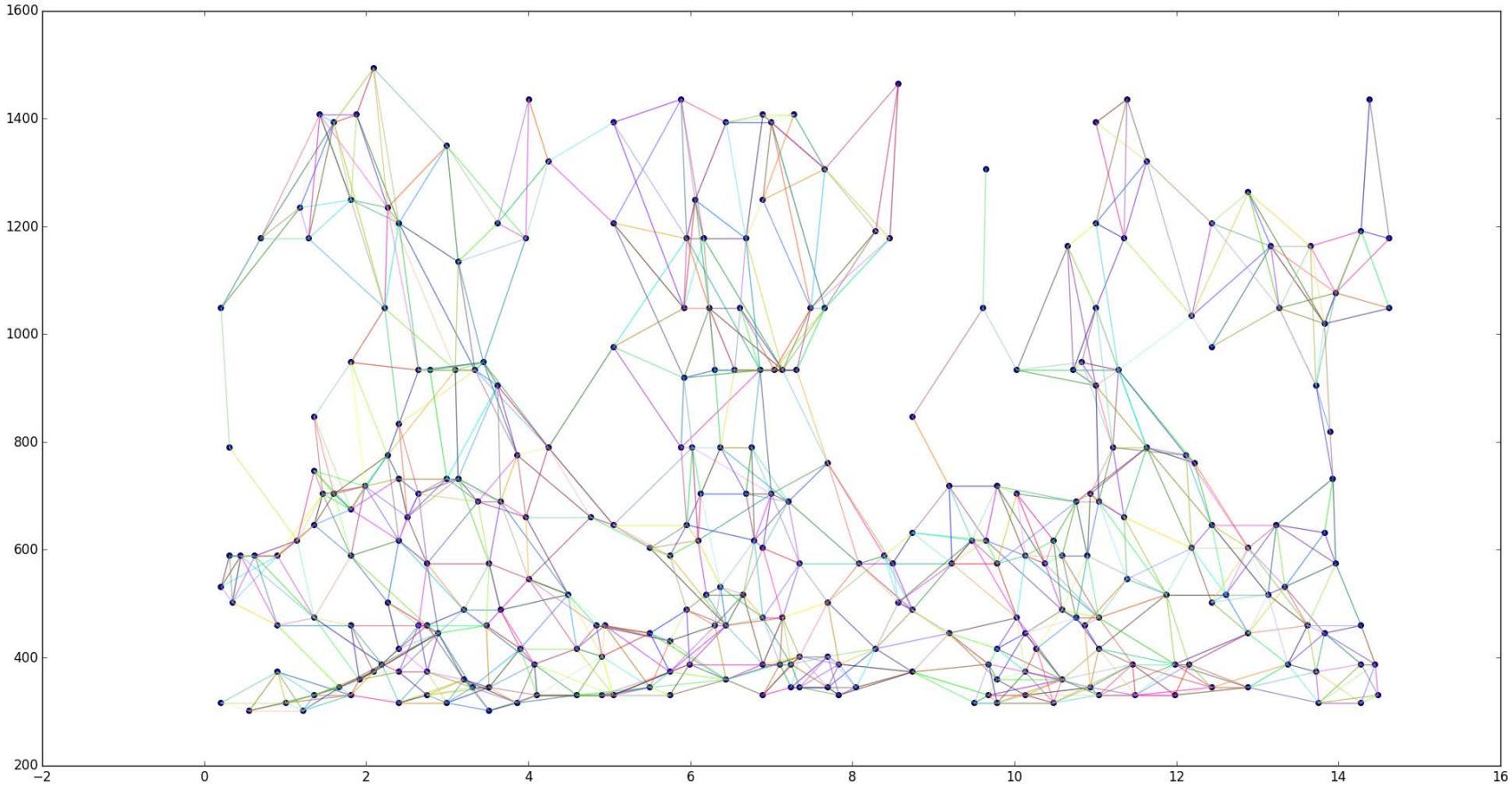
nuage de pics d'une musique de la base de données


10^3
10^2
mesure de la différence
translation
0
50
100
150
200
250
300


10^3
10^2
mesure de la différence
translation
0
50
100
150
200
250
300

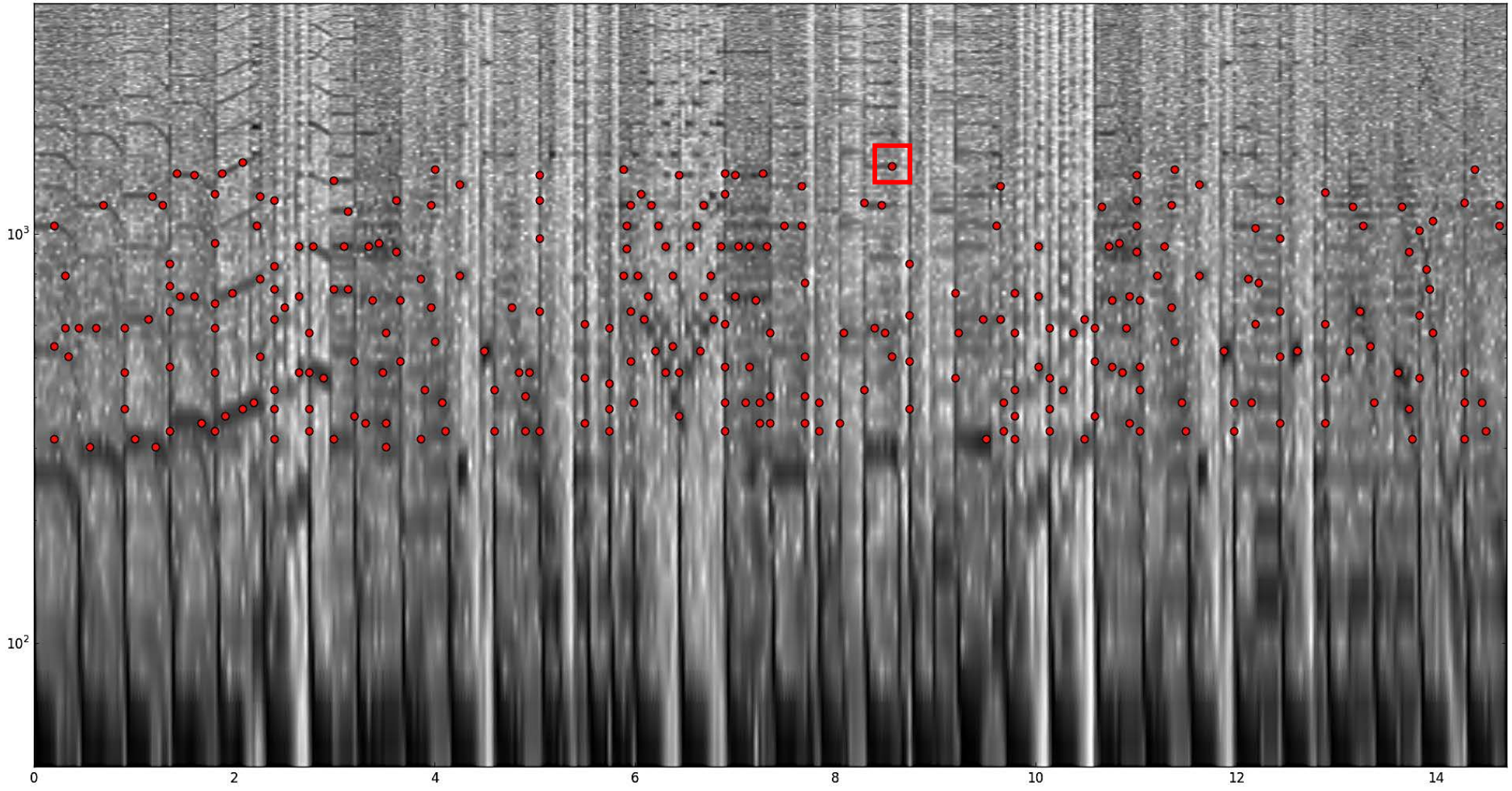

10^3
10^2
0
50
100
150
200
250
300
mesure de la différence
translation

Deuxième programme

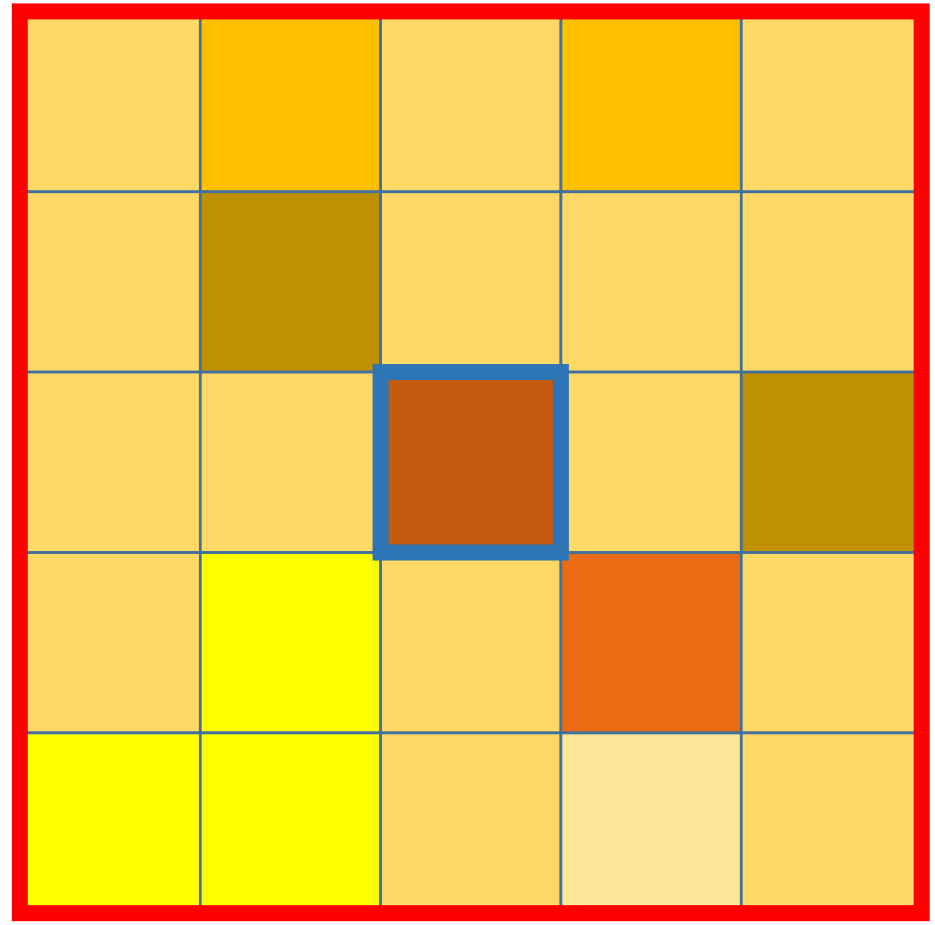


Extraction des features

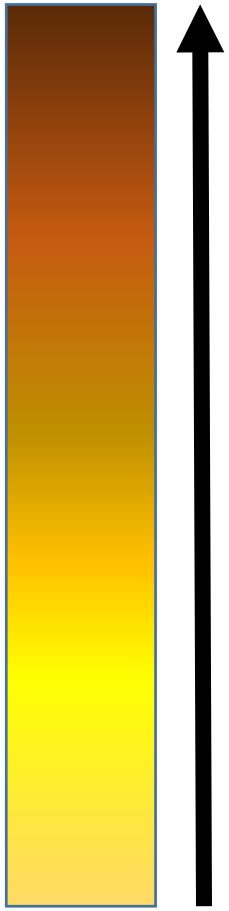
- **Recherche des maximums locaux d'amplitude**



10 Hz lorsque f = 100 Hz

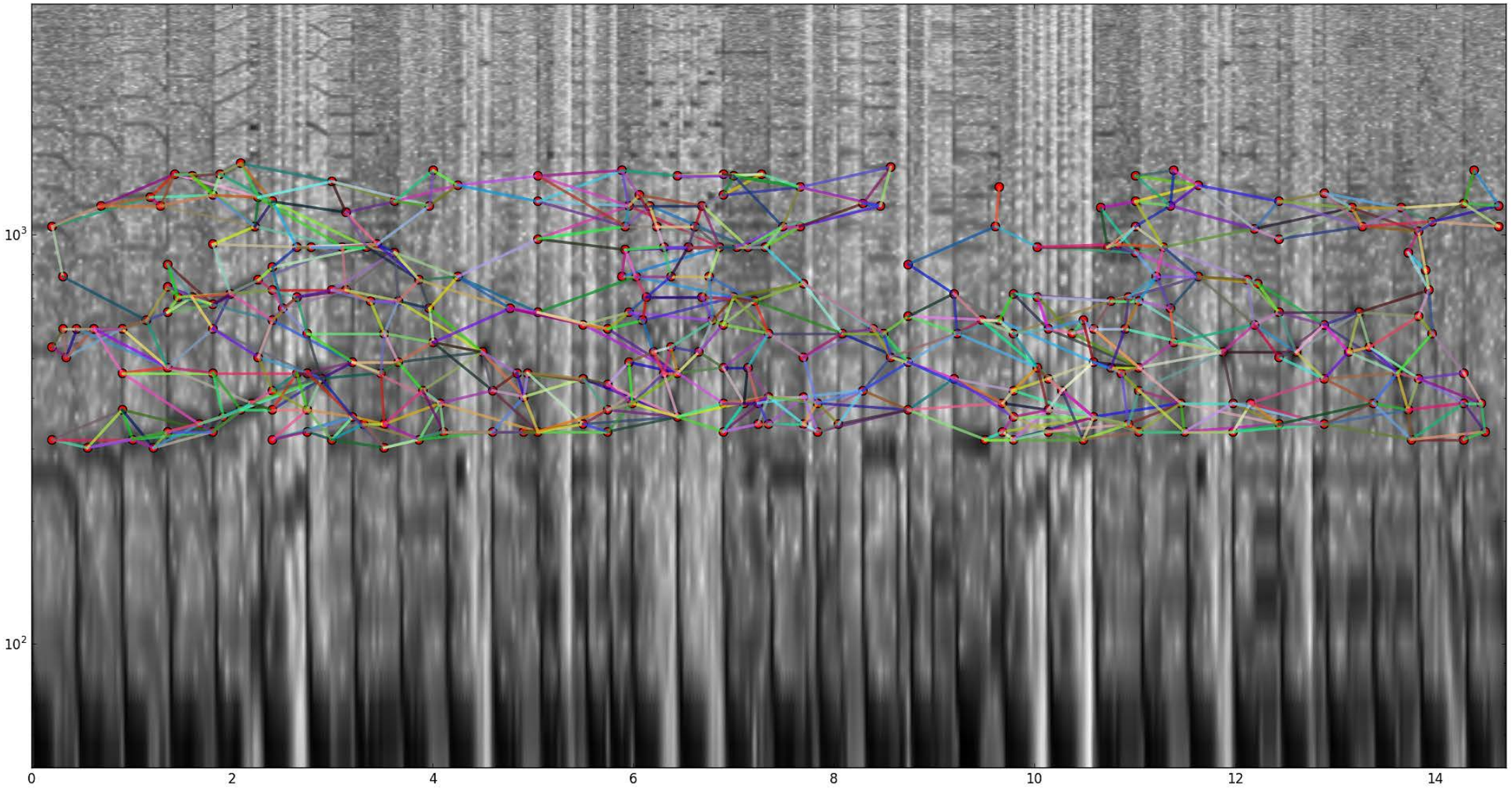


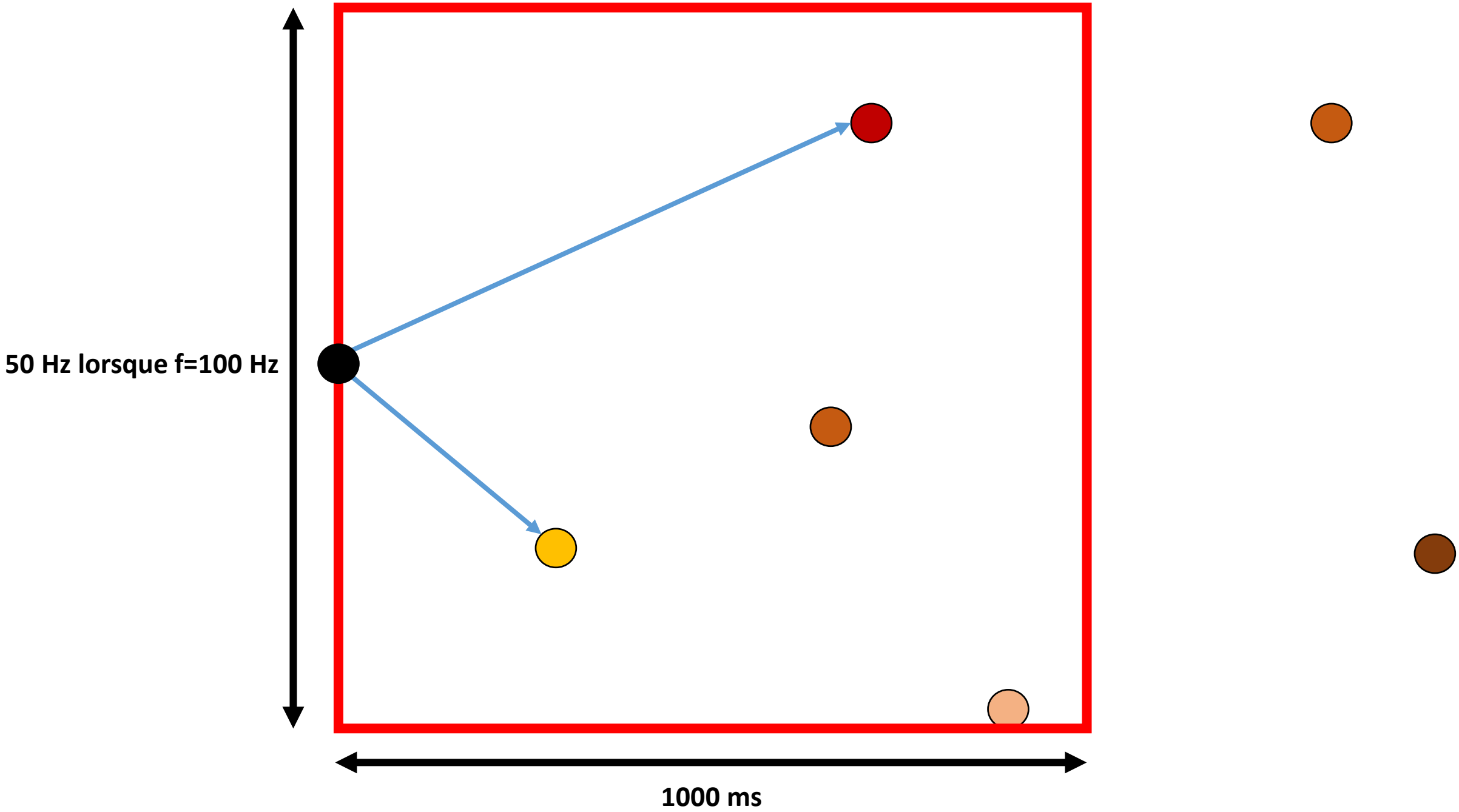
100 ms

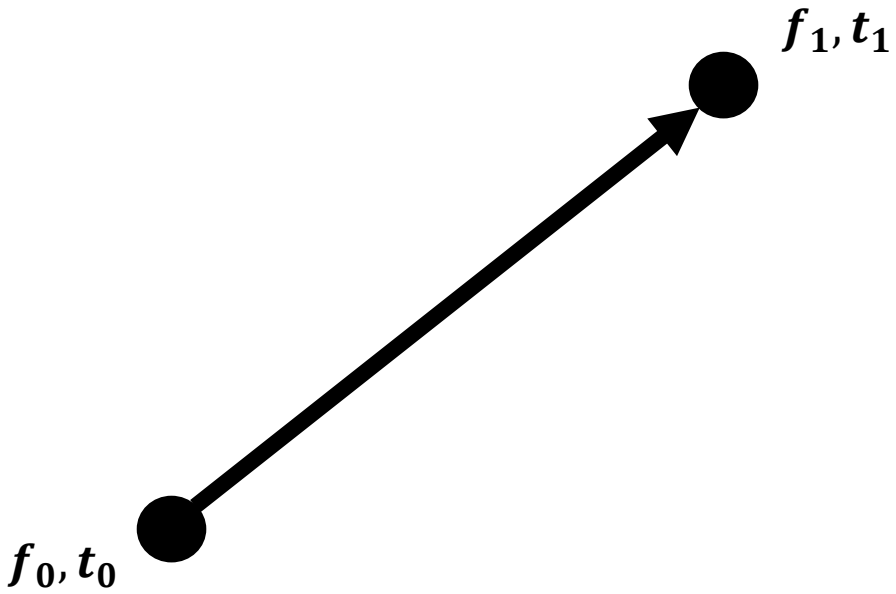


intensité

- **Appairage des points**




50 Hz lorsque f=100 Hz
1000 ms



$(t_1 - t_0, f_0, f_1)$

fonction de hachage

2411746 (par exemple)

On stocke une paire dans la base de données sous la forme (t_0, h)

Structure de la base de données

Table *songs*

id	filename
ENTIER	TEXTE
3	Emancipator - Anthem.wav

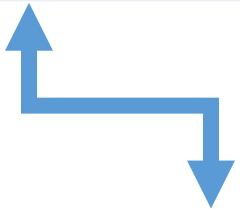


Table *hashes*

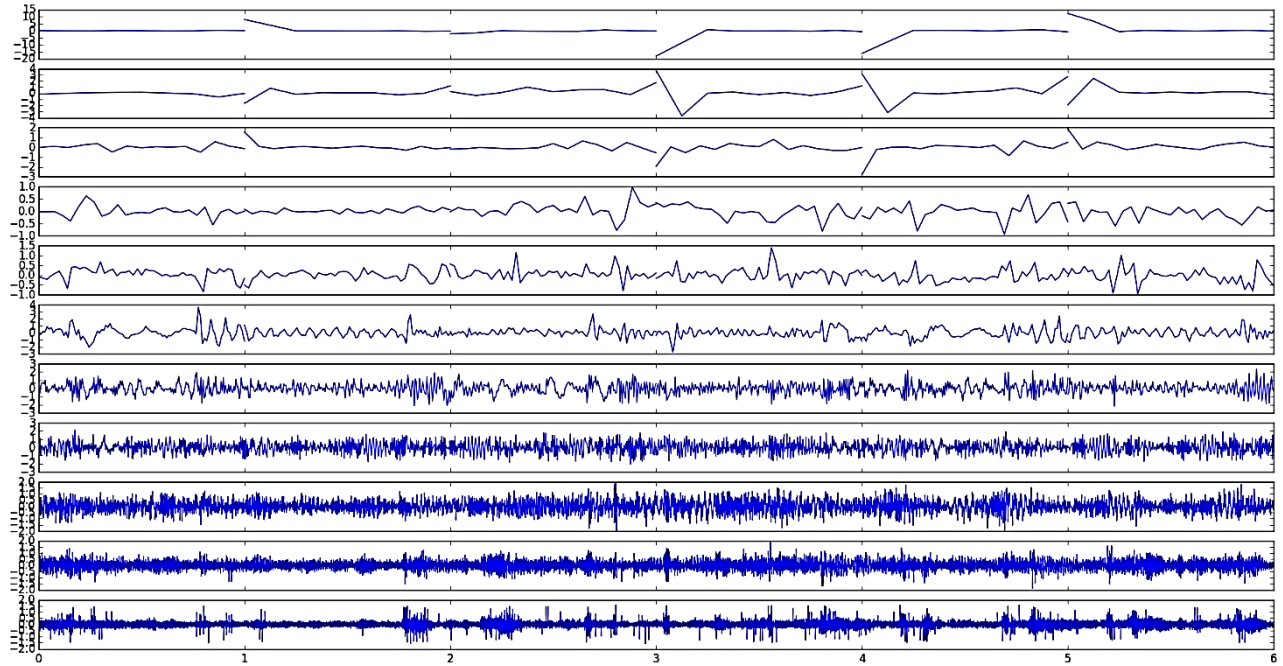
id	idSong	t	h
ENTIER	ENTIER	ENTIER	ENTIER
47062	3	2269	45931033

Identification

On insère les paires hachées de l'extrait dans la table hashes, avec idSong=0.

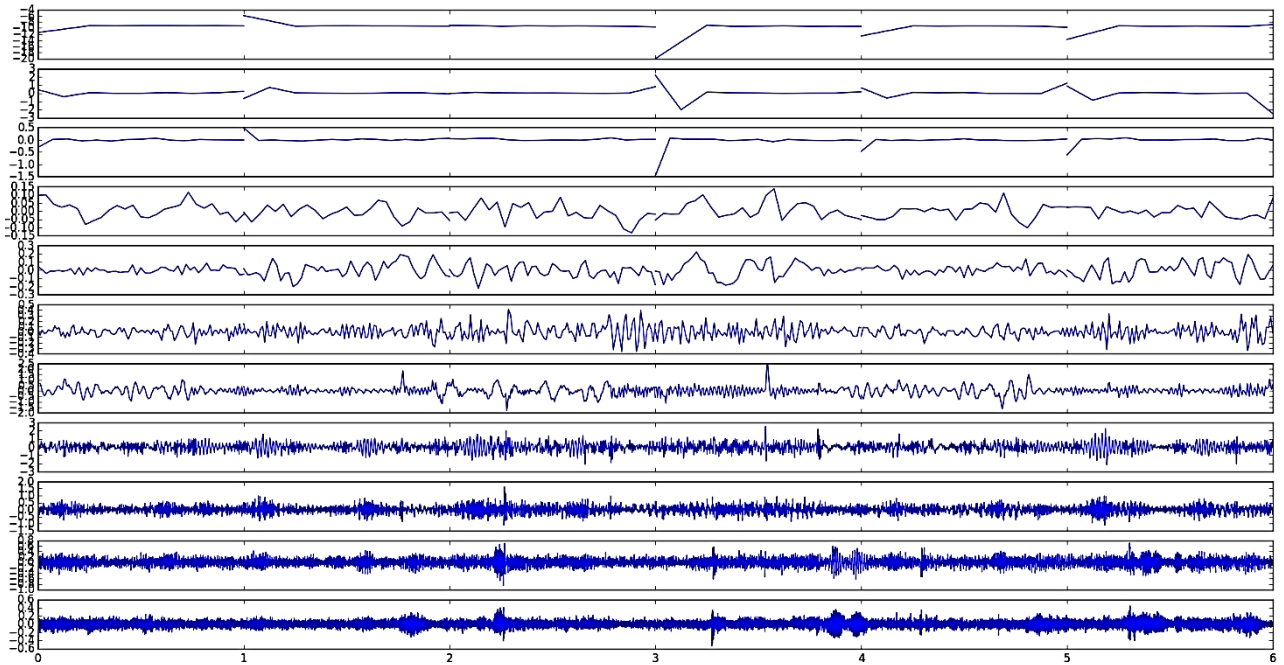
```
SELECT b.idSong
FROM hashes a, hashes b
WHERE a.idSong=0 AND b.idSong!=0 AND b.t >= a.t AND b.h=a.h
GROUP BY b.t-a.t, b.idSong
ORDER BY COUNT(*) DESC LIMIT 1
```

Troisième programme : analyse multirésolution par ondelettes (AMR)



Projections sur les espaces de détails W_j

extrait de qualité optimale



extrait enregistré par une webcam

Bilan

Résultats, améliorations possibles

Résultats obtenus

- Testé avec 432 musiques dans la base de données
- Premier algorithme naïf :
  - 1h30 de temps d'importation
  - 5 minutes pour identifier un extrait de 10 secondes
  - Identifie correctement des extraits très dégradés
- Deuxième algorithme :
  - 30 minutes de temps d'importation (3 fois plus rapide)
  - 27 secondes pour identifier un extrait de 10 secondes (10 fois plus rapide)
  - Identifie correctement des extraits moyennement dégradés
- Troisième algorithme (estimations) :
  - Temps d'importation semblable
  - 4 fois moins de paires : identification 4 fois plus rapide