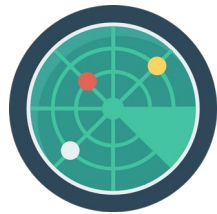


# Reconnaissance et séparation de notes et d'accords dans un fichier audio musical par analyse spectrale

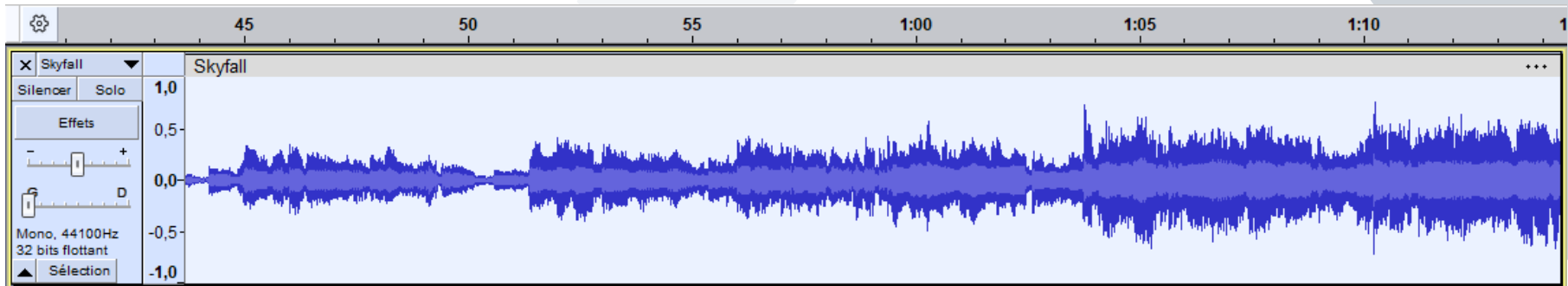


Florentin  
Noguès  
N°11968  
2023-2024

# Introduction

Abscisse : Temps [s]

Ordonnée : Amplitude [unité non communiquée]



**Figure 1.** Représentation temporelle de Skyfall – Adele dans le logiciel Audacity

Comment isoler et reconnaître des notes extraites d'un morceau musical ?

# Objectif

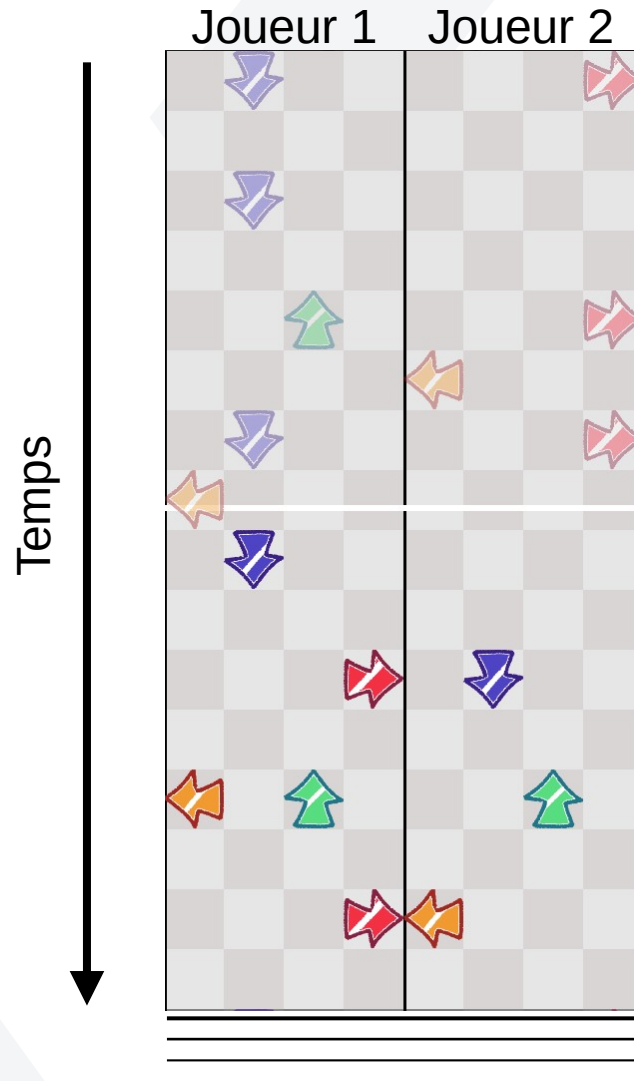
Génération d'une partition pour un jeu vidéo de rythme :  
Friday Night Funkin' (FnF)



**Figure 2.** Capture d'écran d'une partie de FnF

# Complexité du problème

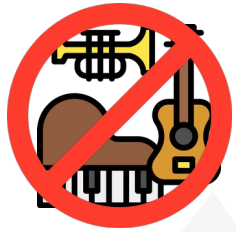
**Figure 3.** Partition d'une musique du jeu FnF



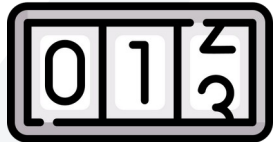
# Plan

1. Simplifications choisies
2. Une façon d'exploiter un fichier sonore
3. Conjecture
4. Mise en application de la conjecture
5. Analyse des résultats
6. Conclusion

# Simplifications choisies



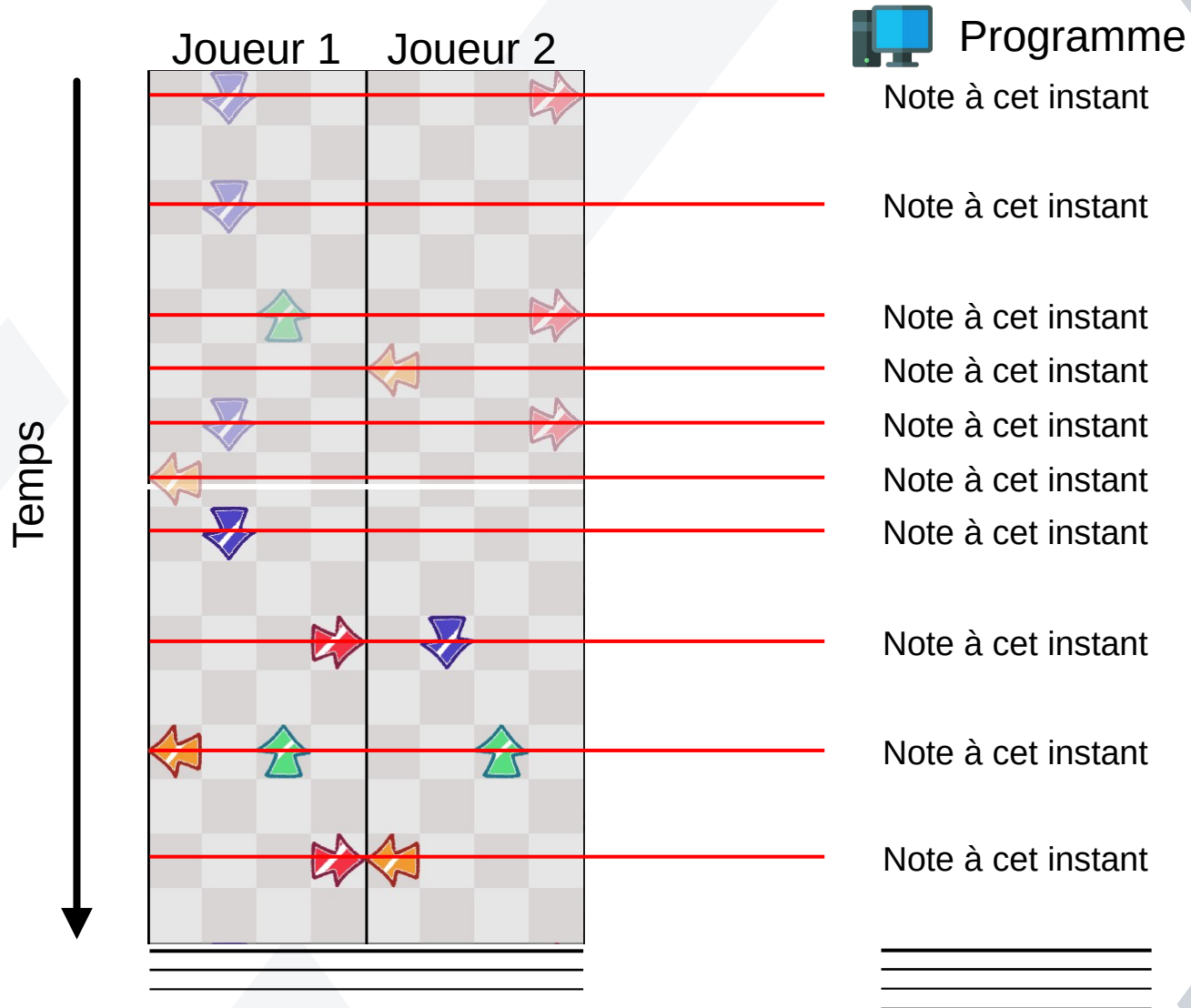
Fichier voix uniquement



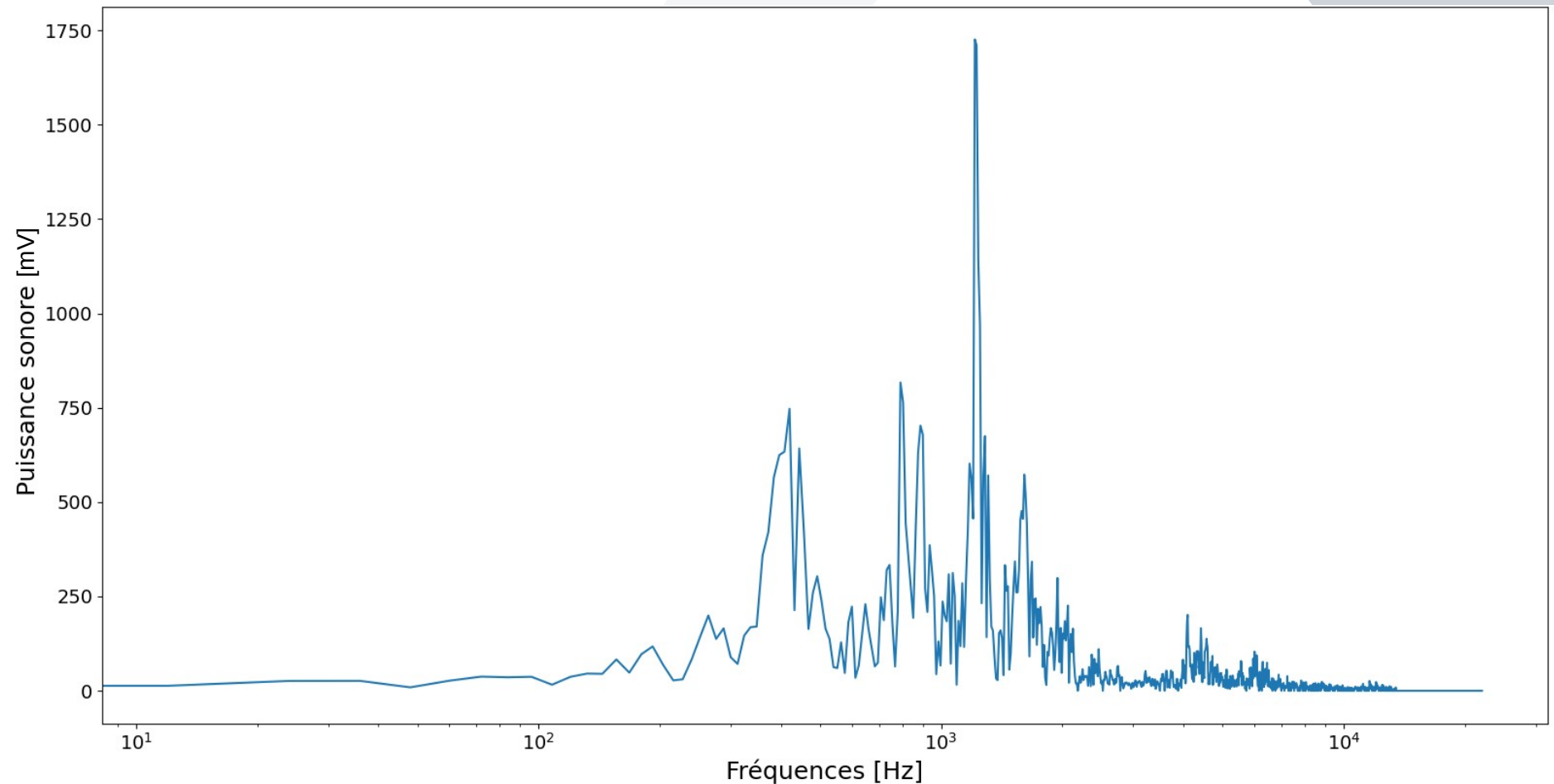
Détection d'occurrences de notes

# Simplifications choisies

Figure 4. Schématisation du problème



# Exploiter un fichier sonore



**Figure 5.** Représentation fréquentielle d'une note de Trapped – OurpleV3

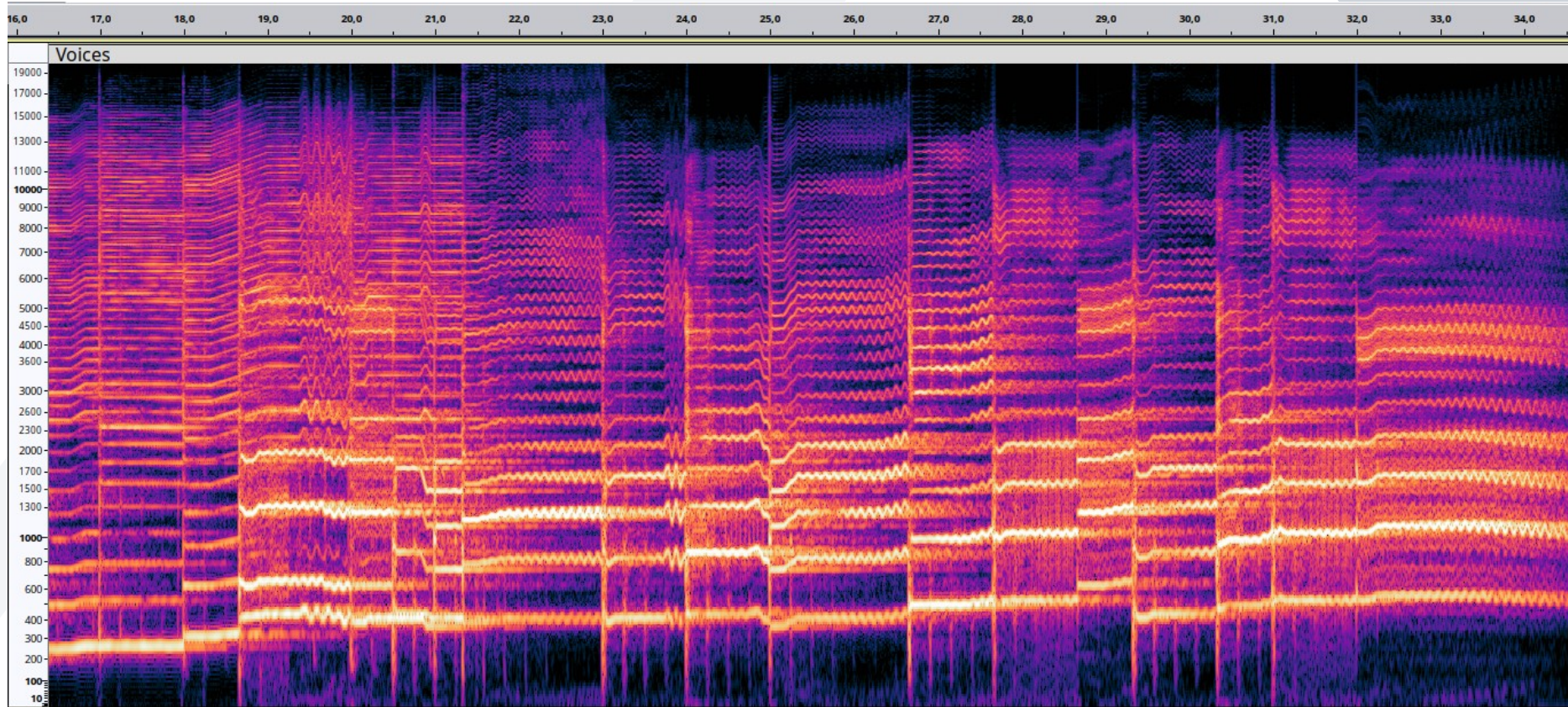


# Exploiter un fichier sonore

Abscisse : Temps [s]

Ordonnée : Fréquence [Hz]

Couleur : Amplitude [unité non communiquée]



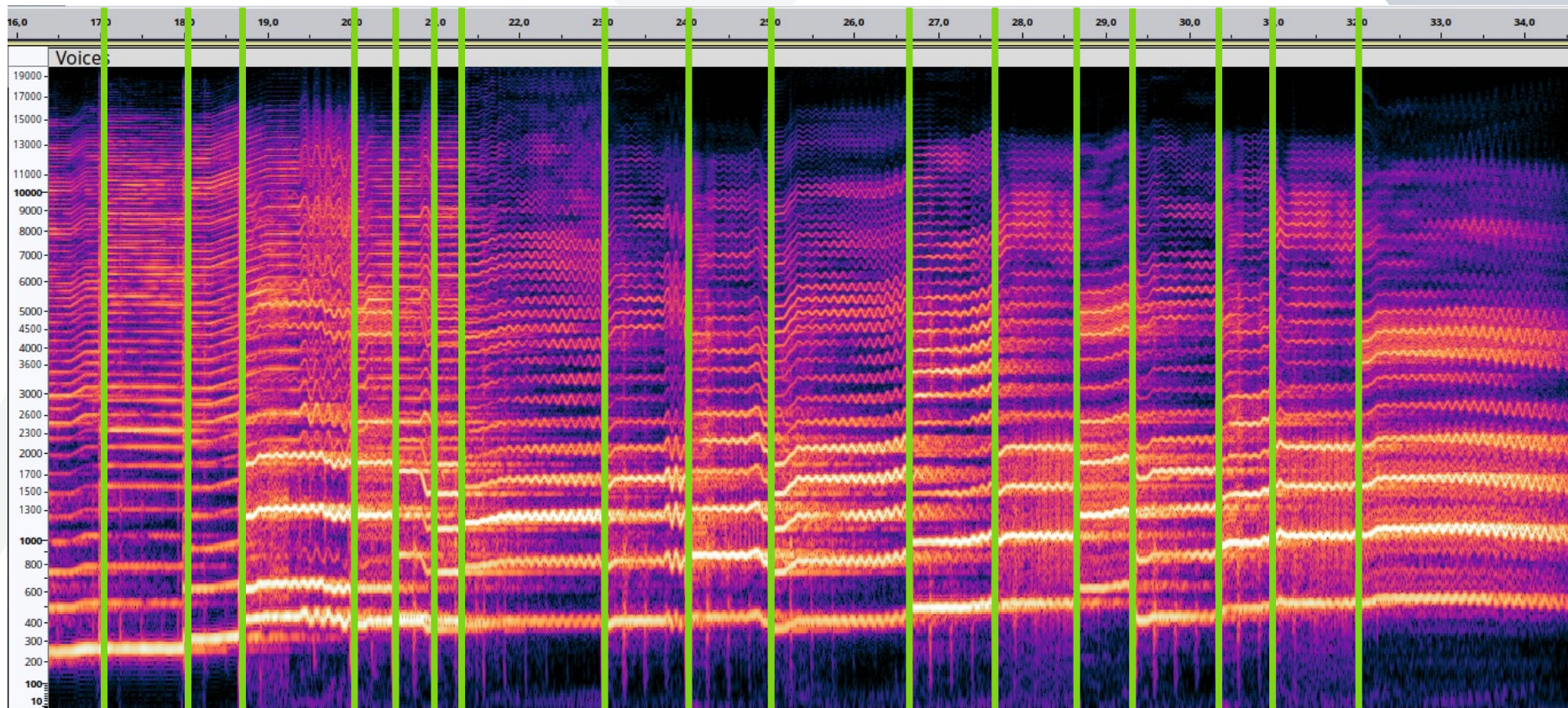
**Figure 6.** Sonogramme d'une partie du fichier Voix de Trapped – OurpleGuyV3 générée par Audacity

# Conjecture

Abscisse : Temps [s]

Ordonnée : Fréquence [Hz]

Couleur : Amplitude [unité non communiquée]



**Figure 7.** Sonogramme d'une partie du fichier Voix de Trapped – OurpleGuyV3 générée par Audacity

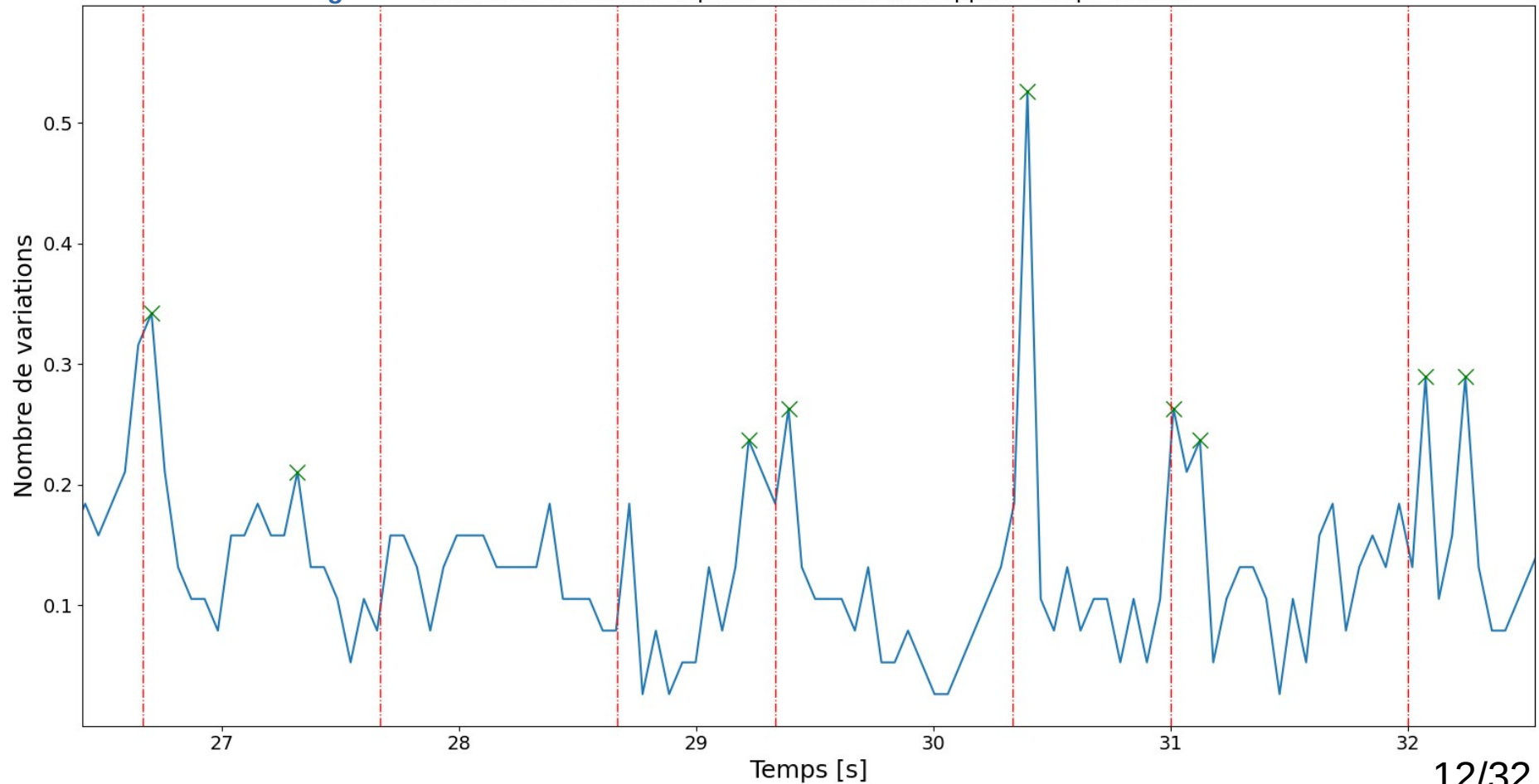
# Mise en application

- Utilisation de la fonction `fft_freq` de la bibliothèque `scipy`
- Création du sonagramme
- Dérivation de la puissance sonore à fréquence fixée en fonction du temps
- Comptage des dérivations à partir d'un certain seuil

# Mise en application

Seuil : 800 mV/s

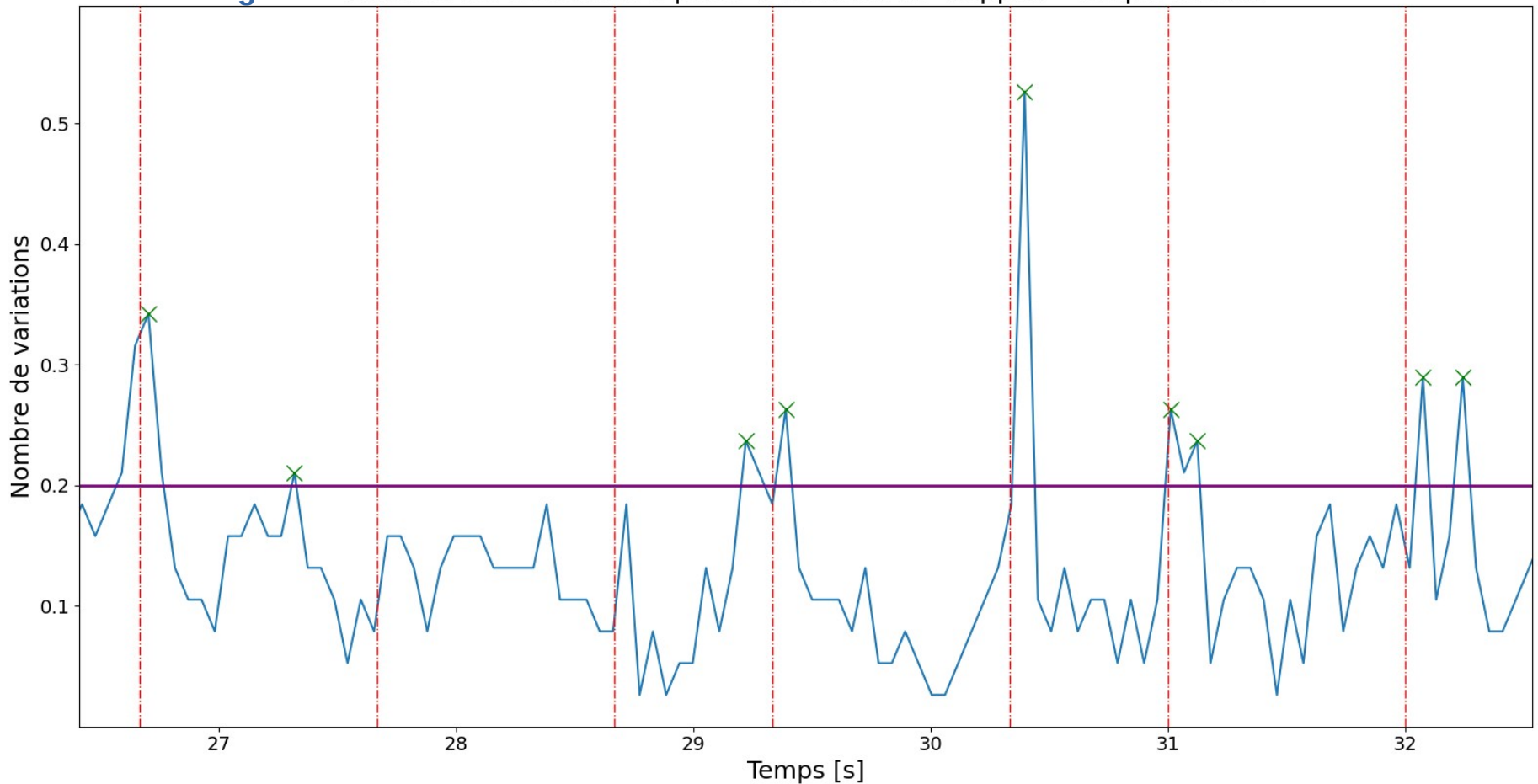
Figure 8. Nombre de variations à partir d'un seuil de Trapped - OurpleV3.wav



# Mise en application

Seuil : 800 mV/s Hauteur min : 0.2

Figure 9. Nombre de variations à partir d'un seuil de Trapped - OurpleV3.wav



# Analyse des résultats

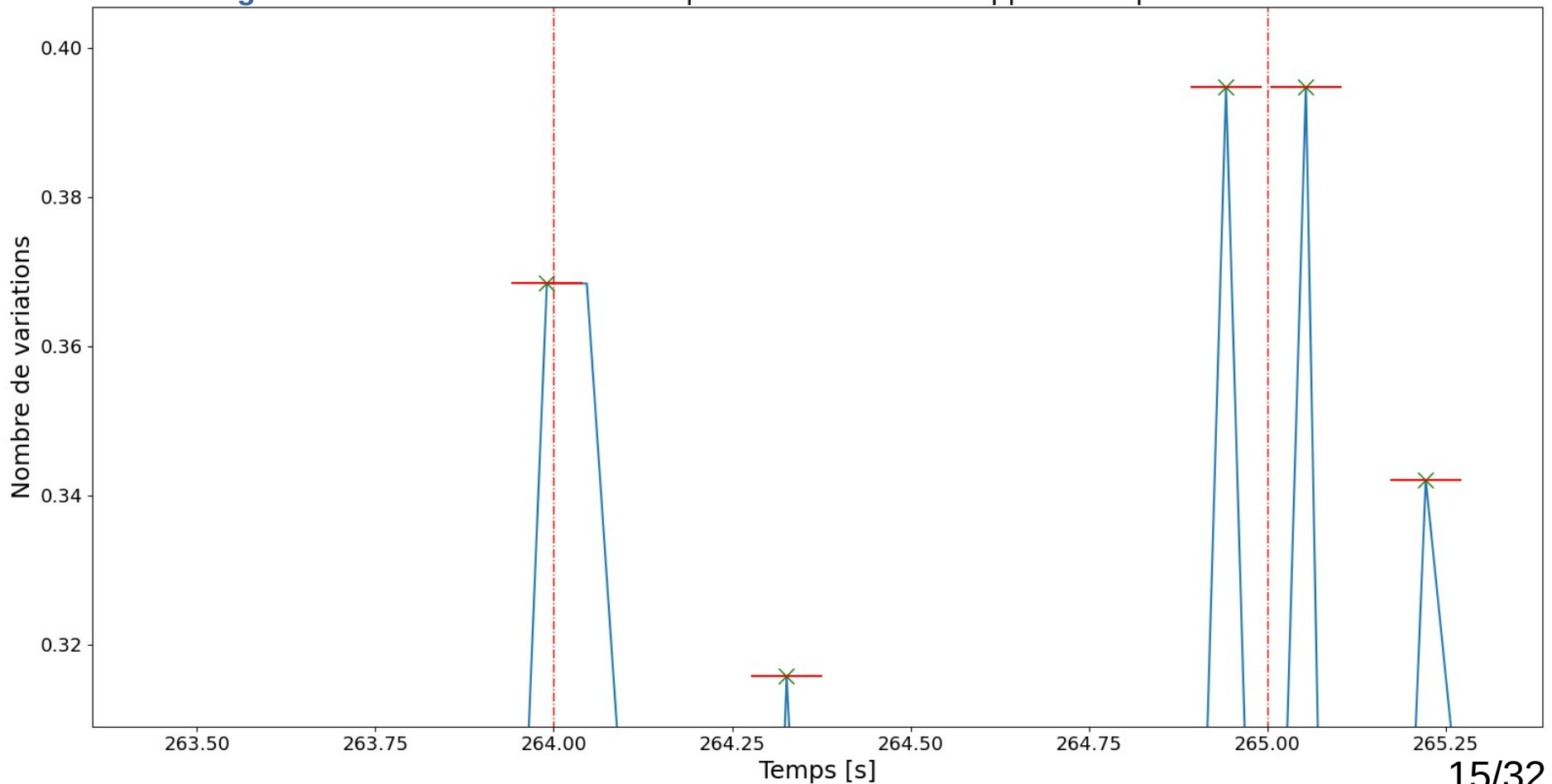
## Comment ?

- Algorithme testé sur différentes valeurs de seuils et de hauteurs
- Utilisation de partitions déjà créées par la communauté.
  - Notes générées bien placées
  - Nombre de notes générées
- Utilisation de deux quantités
  - Précision
  - Notes correctement générées / Notes originelles

# Analyse des résultats

Seuil : 800 mV/s Hauteur min : 0.2 Erreur : 0.05 s

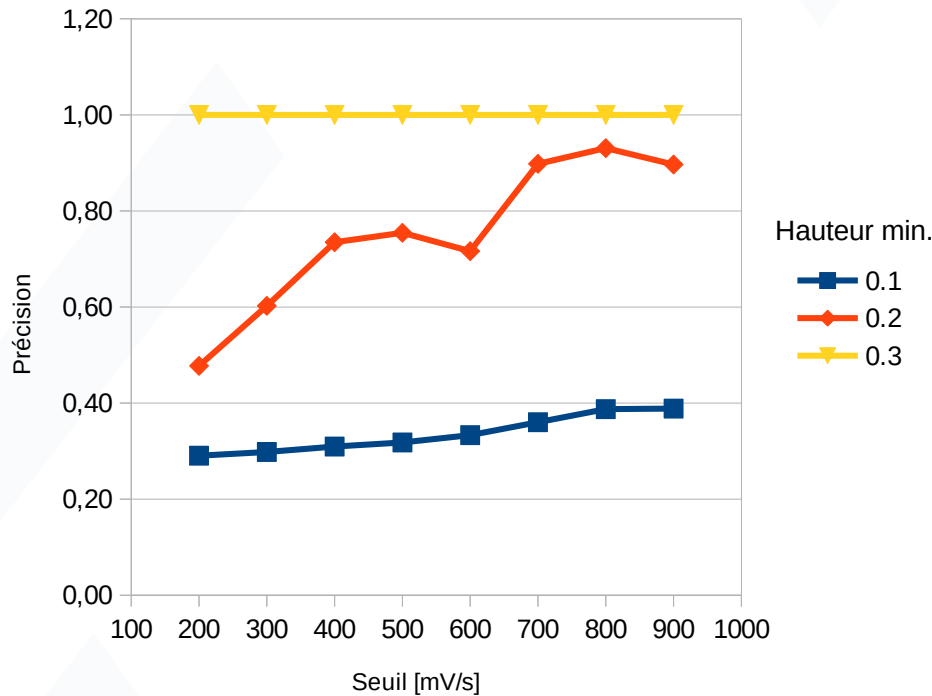
Figure 10. Nombre de variations à partir d'un seuil de Trapped - OurpleV3.wav



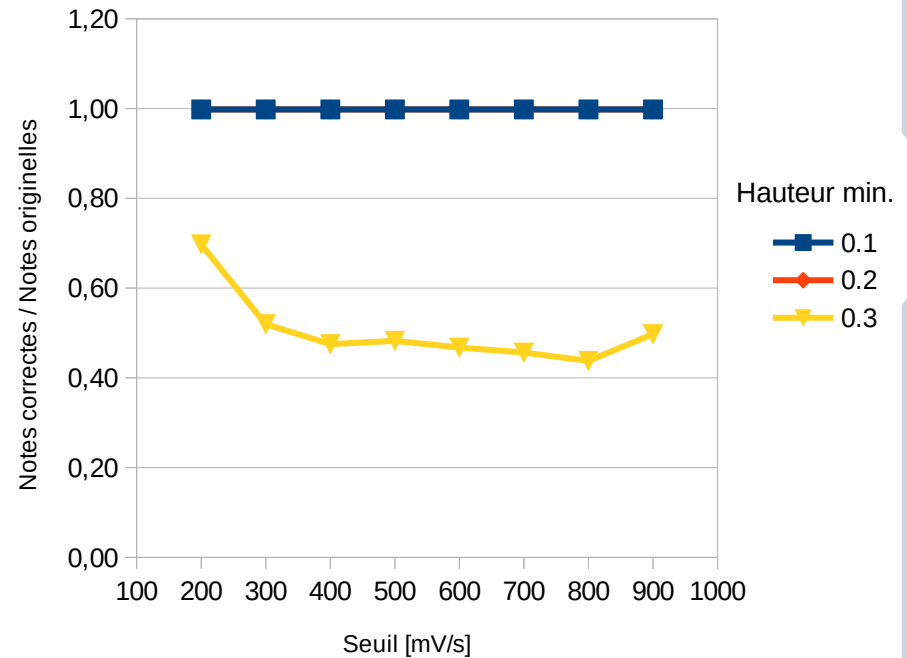
# Analyse des résultats

## *Restless – Ourple V3*

Précision en fonction du Seuil



Notes correctes / Notes originelles en fonction du Seuil

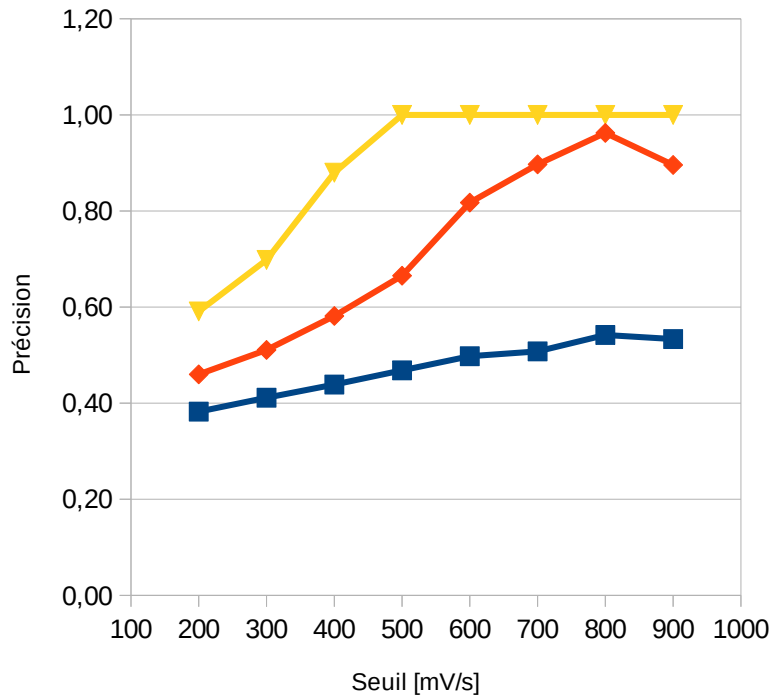




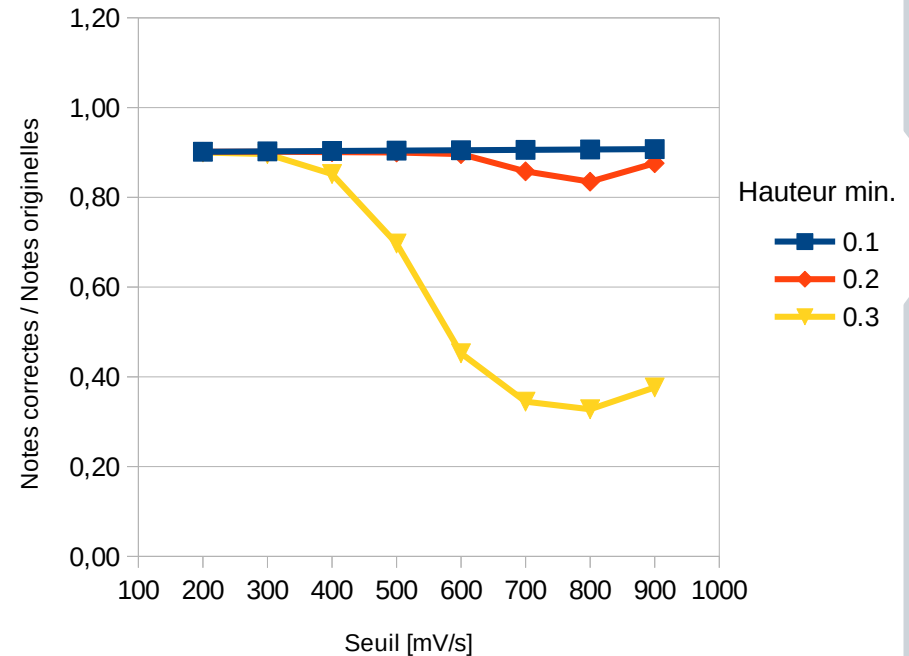
# Analyse des résultats

## *Impatience – Funkscop*

Précision en fonction du Seuil



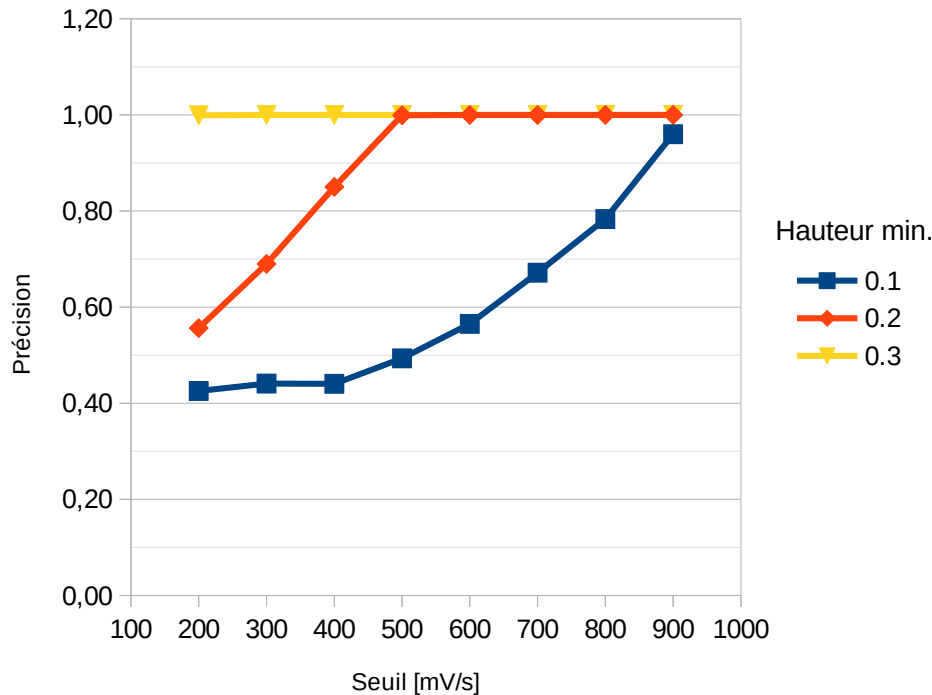
Notes correctes / Notes originelles en fonction du Seuil



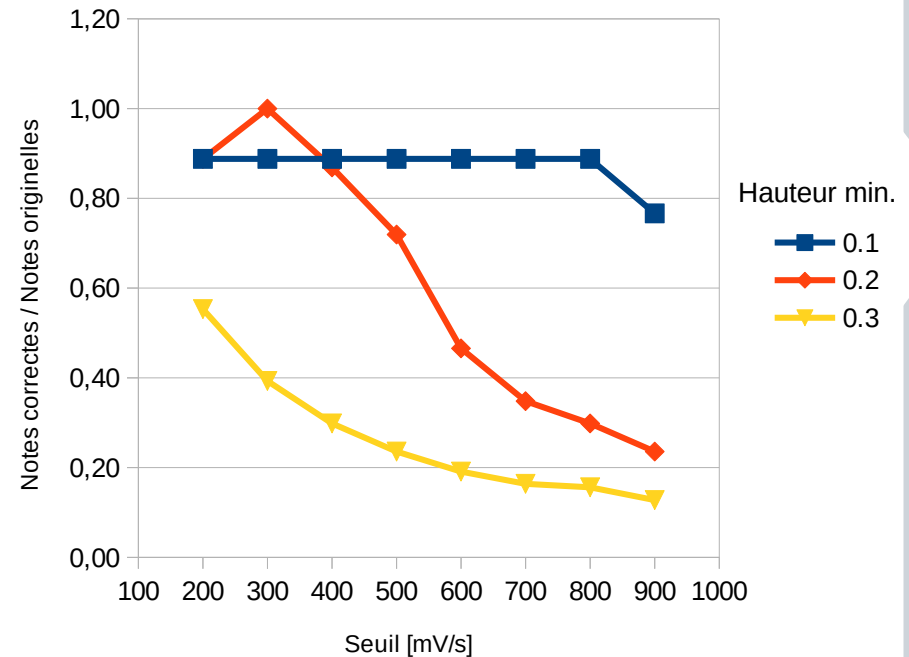
# Analyse des résultats

## *Followed – Ourple V3*



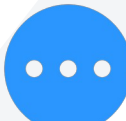
Précision en fonction du Seuil



Notes correctes / Notes originelles en fonction du Seuil



# Conclusion

-  La méthode fonctionne : les résultats sont bons
-  La méthode est incomplète
-  Résultats imparfaits toutefois : légitimité de l'utilisation de l'IA

# Annexe

## Code | main.py

```
1 from matplotlib import pyplot as plt
2 import numpy as np
3 from scipy.signal import find_peaks
4 import lib
5
6 musique = lib.Musique(134, "../../Musique/Voting Time - Impostor V4.wav", 32)
7 partition = lib.Partition("../../Data musique/voting-time.json")
8 nom_musique = "Voting Time"
9
10 for seuil_hauteur in np.arange(0.1, 0.31, 0.1):
11     for seuil_derive in range(200, 1000, 100):
12         print(str(seuil_hauteur) + str(seuil_derive) + " " + str(0.4 / 0.1 + 1000 /
13 100))
14         notes = musique.generate_notes()
15         lib.save_result(nom_musique, partition, seuil_hauteur, seuil_derive, notes)
```

# Annexe

## Code | lib.py > class Musique

```
1 class Musique:
2     bpm = 0
3     beat_snap = 64
4     path = ""
5     fe = 0
6     data = ""
7
8     def __init__(self, bpm, path, beat_snap = 64):
9         self.bpm = bpm
10        self.beat_snap = beat_snap
11        self.path = path
12        self.fe, self.data = scipy.io.wavfile.read(self.path)
13
14        """
15        Dérivation du spectrogramme en fonction du temps à fréquence fixée
16        """
17        def __derive_ftt(self, spectro):
18            taille_intensite = len(spectro[0])
19            deriver_liste = [[0 for _ in range(0, taille_intensite)]]
20
21            for i in range(1, len(spectro)):
22                deriver_liste.append([])
23                for j in range(0, taille_intensite):
24                    deriver_liste[i].append((np.abs(spectro[i][j]))-np.abs(spectro[i-
25                    1][j]))
26            return deriver_liste
```

# Annexe

## Code | lib.py > class Musique

```
1
2  """
3  Nombre de secondes par note dans une partition de FNF
4  """
5  def get_periode_note(self):
6      return (4*60)/(self.beat_snap*self.bpm)
7
8  """
9  Génère un spectrogramme d'un son
10 """
11 def generer_spectro(self):
12     fourrier_liste = []
13     left, right = self.data.T[0], self.data.T[1]
14     N = math.trunc(self.fe * self.get_periode_note())
15
16     maxi = math.ceil(len(self.data)/N)
17
18     for i in range(1, maxi):
19         fourrier = scipy.fft.fft(left[N*(i-1):N*i])
20         fourrier_liste.append(2.0/N * np.abs(fourrier[:N//2]))
21
22     return fourrier_liste
23
```

# Annexe

## Code | lib.py > class Musique

```
1
2
3 """
4 Compte le nombre de variations d'intensité de fréquence pour chaque intervalle de
temps
5 """
6 def count_diff(self, gap):
7     N = math.trunc(self.fe * self.get_periode_note())
8     deriver = self.deriver()
9     deriver_diff = [0]
10
11     for i in range(1, len(deriver)):
12         deriver_diff.append(0)
13         for j in range(0, len(deriver[0])):
14             if deriver[i-1][j] - deriver[i][j] > gap:
15                 deriver_diff[i] += 1
16
17     return deriver_diff
18
19
20 """
21 Dérivation du spectrogramme en fonction du temps à fréquence fixée
22 """
23 def deriver(self):
24     return self.__derive_ftt(self.generer_spectro())
```

# Annexe

## Code | lib.py > class Musique

```
1
2
3 """
4 Compte le nombre de variations d'intensité de fréquence pour chaque intervalle de
temps
5 """
6 def count_diff(self, gap):
7     N = math.trunc(self.fe * self.get_periode_note())
8     deriver = self.deriver()
9     deriver_diff = [0]
10
11     for i in range(1, len(deriver)):
12         deriver_diff.append(0)
13         for j in range(0, len(deriver[0])):
14             if deriver[i-1][j] - deriver[i][j] > gap:
15                 deriver_diff[i] += 1
16
17     return deriver_diff
18
19
20 """
21 Dérivation du spectrogramme en fonction du temps à fréquence fixée
22 """
23 def deriver(self):
24     return self.__derive_ftt(self.generer_spectro())
```



# Annexe

## Code | lib.py > class Musique

```
1
2  """
3  Affiche le spectrogramme du son
4  """
5  def plot_spectro(self):
6      N = math.trunc(self.fe * self.get_periode_note())
7
8      freq = scipy.fft.fftfreq(N, 1/self.fe)[:N//2]
9      temps = self.generate_time()
10
11     xfreq, xtemps = np.meshgrid(freq, temps)
12
13     np_fourrier = np.array(self.generer_spectro())
14
15     plt.xlabel("Temps [s]")
16     plt.ylabel("Fréquences [Hz]")
17     plt.title("Spectrogramme de " + os.path.basename(self.path))
18     plt.pcolormesh(xtemps, xfreq, np_fourrier, shading="nearest",
19     antialiased=True, cmap="nipy_spectral")
20     plt.colorbar(label="Puissance sonore", orientation="vertical", shrink=0.7)
21     plt.ylim(0, 10000)
```

# Annexe

## Code | lib.py > class Musique

```
1
2
3     """
4     Génère les notes
5     """
6
7     def generate_notes(self, hauteur, seuil):
8         temps = self.generate_time()
9         diffs = np.array(self.count_diff(seuil))
10
11         min_val = np.min(diffs)
12         max_val = np.max(diffs)
13
14         scaled_data = (diffs - min_val) / (max_val - min_val)
15
16         peaks = find_peaks(scaled_data, height=hauteur)
17
18         return temps[peaks[0]]
```

# Annexe

## Code | lib.py > class Partition

```
1 class Partition:
2     path = ""
3
4     def __init__(self, path):
5         self.path = path
6
7     """
8     Récupère le nombre de notes dans une partition
9     """
10    def nombreTotal(self):
11        json_data = json.load(open(self.path))
12        number = 0
13        doublon = 0
14
15        for section in json_data['song']['notes']:
16            already_calculated = {}
17            for note in section['sectionNotes']:
18                joueur = (note[1] <= 3) if 1 else 0
19                if not (note[0], (joueur)) in already_calculated:
20                    number += 1
21                    already_calculated[(note[0], (joueur))] = True
22            else:
23                doublon += 1
24        return number
```

# Annexe

## Code | lib.py > class Partition

```
1  """
2  Renvoie une liste de notes présentes dans la partition
3  """
4
5  def notes(self):
6      json_data = json.load(open(self.path))
7      notes = []
8      for section in json_data['song']['notes']:
9          for note in section['sectionNotes']:
10             if not (note[0] / 1000) in notes:
11                 notes.append(note[0] / 1000)
12         return sorted(notes)
13
14  """
15  Compte le nombre de notes correctement placées
16  """
17
18  def compare(self, notes_gen):
19      notes_bien_places = 0
20      notes = self.notes()
21      for i in range(0, len(notes_gen)):
22          index = dichotomie(notes_gen[i], notes, 0.05)
23          if index != -1:
24              notes.pop(index)
25              notes_bien_places += 1
26      return notes_bien_places
```

# Annexe

## Code | lib.py > Fonctions auxiliaires

```
1 """
2 Recherche par dichotomie d'une note à epsilon près
3 """
4 def dichotomie(value, array, epsilon):
5     if len(array)==1:
6         if (abs(value - array[0]) < epsilon):
7             return 0
8         else:
9             return -1
10    m = len(array)//2
11    if abs(value - array[m]) < epsilon:
12        return m
13    elif value < array[m]:
14        return dichotomie(value, array[:m], epsilon)
15    elif value > array[m]:
16        return m + dichotomie(value, array[m:], epsilon)
17
```

# Annexe

## Code | lib.py > Fonction auxiliaires

```
1 """
2 Enregistre les résultats dans un fichier au format JSON
3 """
4 def save_result(nom_musique, partition, seuil_hauteur, seuil_derive, notes):
5     data = json.load(open("./resultat.json"))
6     if(not nom_musique in data):
7         data[nom_musique] = {}
8         key = str(seuil_hauteur) + "," + str(seuil_derive)
9
10        if(not key in data[nom_musique]):
11            data[nom_musique][key] = {}
12
13            data[nom_musique][key]['note_gen'] = len(notes)
14            data[nom_musique][key]['note_correctes'] = partition.compare(notes)
15            data[nom_musique][key]['note_partitions'] = partition.nombreTotal()
16            data[nom_musique][key]['precision'] = data[nom_musique][key]
17            data[nom_musique][key]['notes_correctes_manquantes'] = data[nom_musique]
18            data[nom_musique][key]['note_partitions'] - data[nom_musique][key]['note_correctes']
19
20            with open('./resultat.json', 'w', encoding='utf-8') as f:
21                json.dump(data, f, ensure_ascii=True, indent=4)
```

# Annexe

## Transformée de Fourier Discrète

$$f_n = \sum_{k=0}^{N-1} c_k e^{-2i\pi \frac{kn}{N}}$$

$$f_n = \sum_{k=0}^{\frac{N}{2}-1} c_{2k} e^{-\frac{2ikn\pi}{N/2}} + e^{-\frac{2in\pi}{N}} \sum_{k=0}^{\frac{N}{2}-1} c_{2k+1} e^{-\frac{2ikn\pi}{N/2}}$$

$$f_{n+\frac{N}{2}} = \sum_{k=0}^{\frac{N}{2}-1} c_{2k} e^{-\frac{2ikn\pi}{N/2}} - e^{-\frac{2in\pi}{N}} \sum_{k=0}^{\frac{N}{2}-1} c_{2k+1} e^{-\frac{2ikn\pi}{N/2}}$$

# Annexe

## Algorithme de Cooley-Tukey (Fast Fourier Transform)

```
1 def fft(x):
2     N = len(x)
3     if N == 1:
4         return [x[0]]
5
6     X = [0] * N
7
8     paire = fft(x[:N:2])
9     impaire = fft(x[1:N:2])
10
11     for k in range(N//2):
12         w = math.e**(-2j*math.pi * k/N)
13         X[k] = paire[k] + w * impaire[k]
14         X[k + N//2] = paire[k] - w * impaire[k]
15         OP += 2
16
17     return X
```