Identification d’un morceau musical

BEAULIEU Antoine
N° 37754

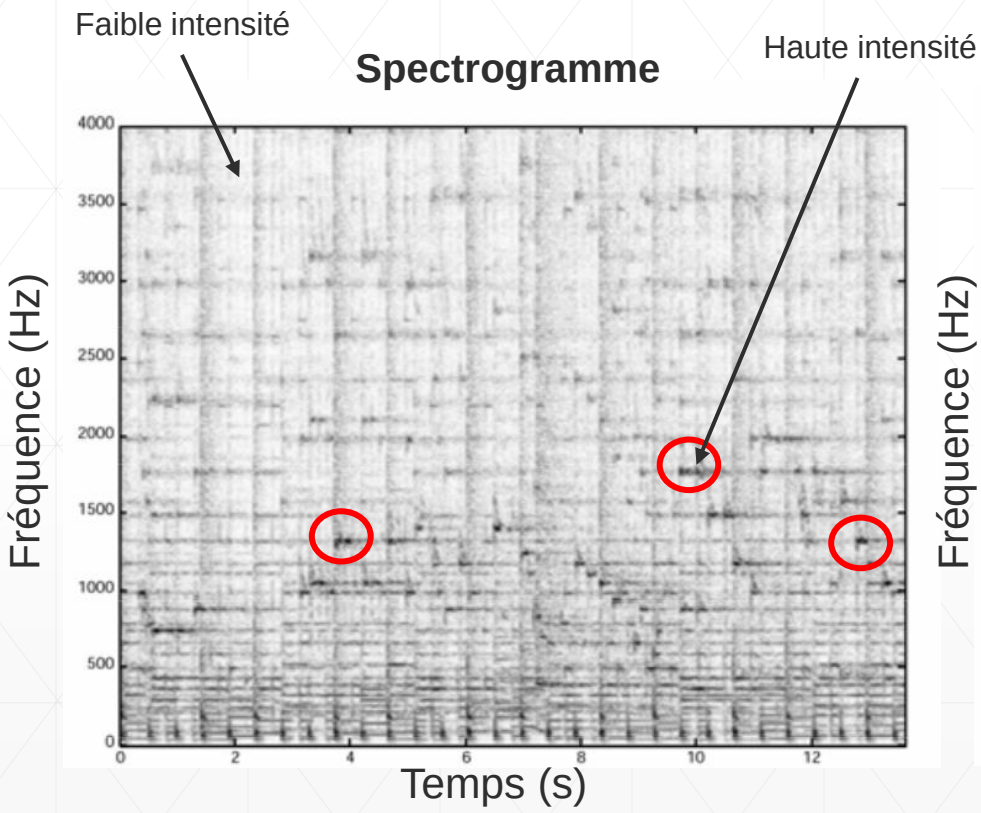
Sommaire

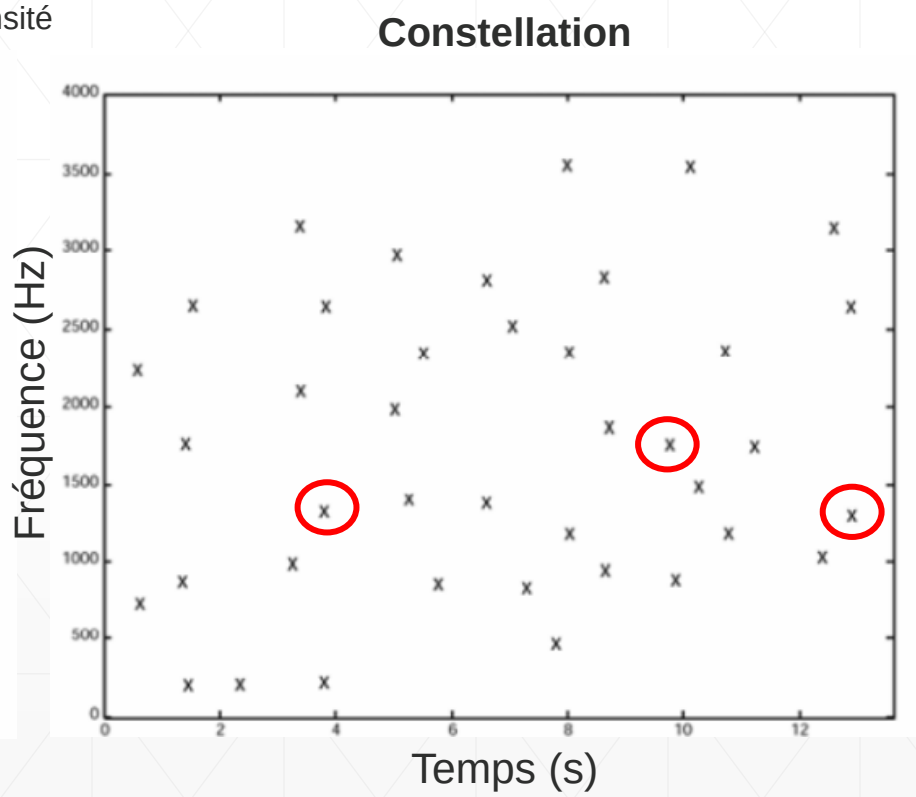
- Introduction
- Problématique
- Méthode mise en œuvre
- Plan d’expérience
- Conclusion

Introduction

Introduction

Introduction

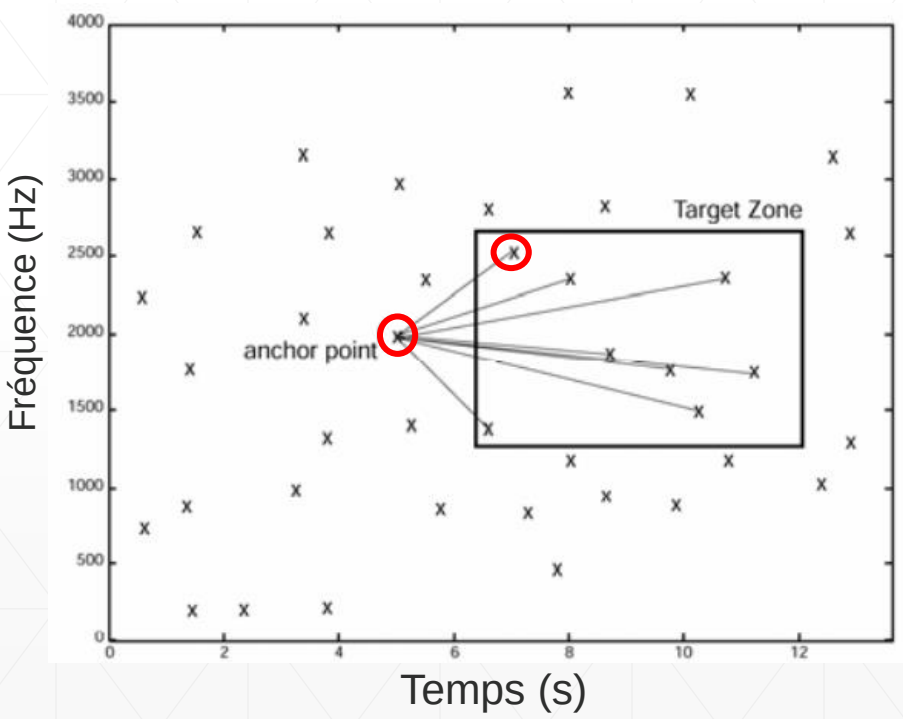




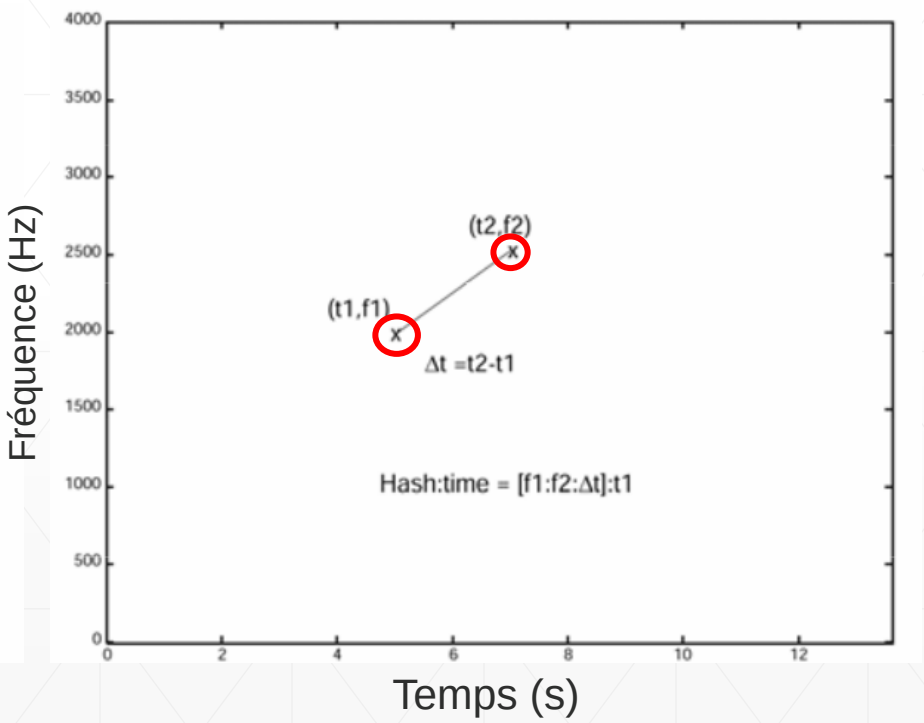
Source : https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf

Introduction

Constellation



Constellation



Source : https://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf

Problématique

Comment analyser un morceau de musique pour gagner plus facilement au blind-test ?

Méthode mise en œuvre

Stratégie

1. Analyse de l’échantillon

2. Construction de la base de données

3. Comparaison échantillon et base de données

Méthode mise en œuvre

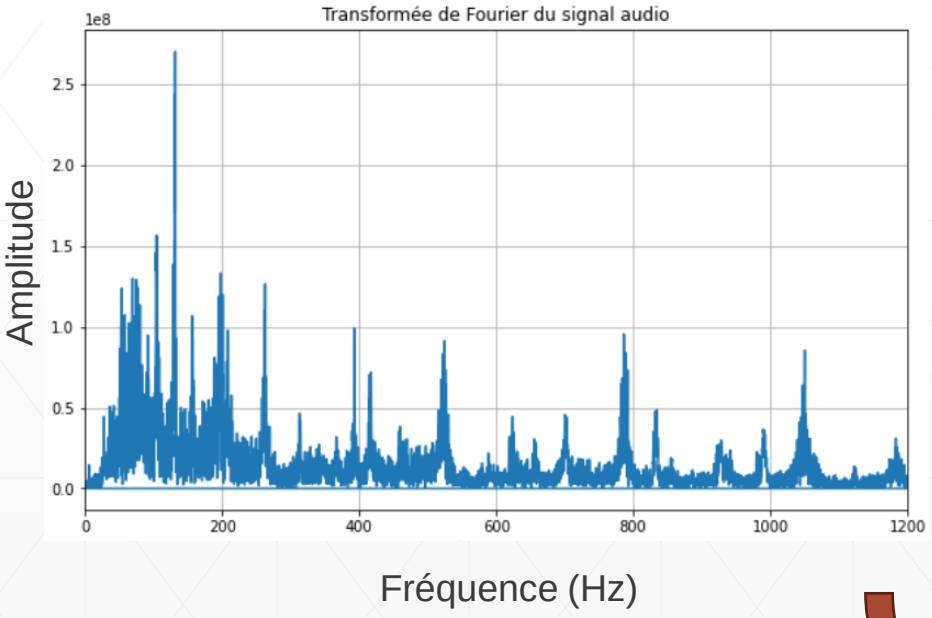
1 : Analyse de l’échantillon

Méthode mise en œuvre

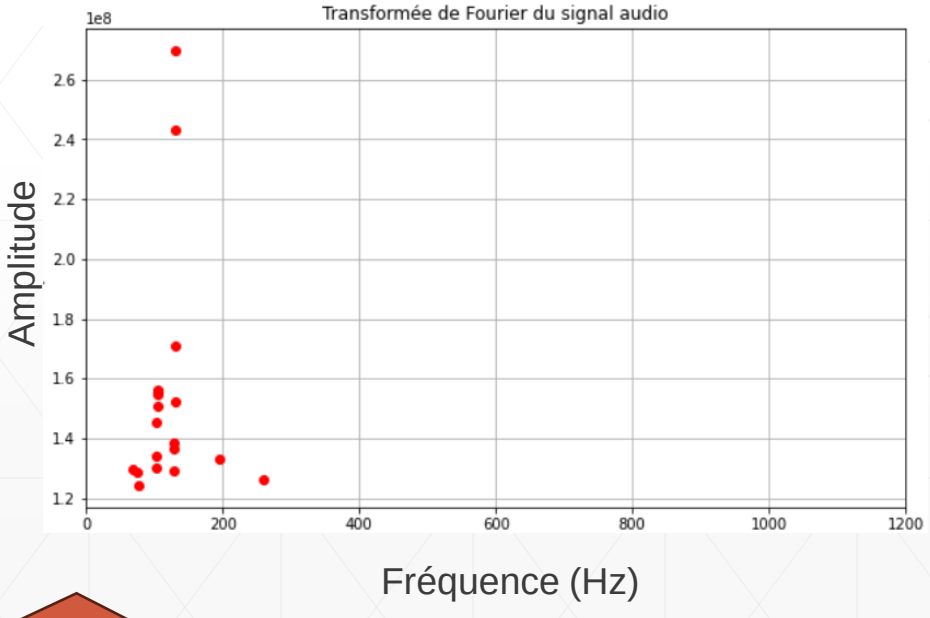
1 :Analyse de l'échantillon

Durée de l'échantillon choisie : **5 secondes**

Spectre



Spectre

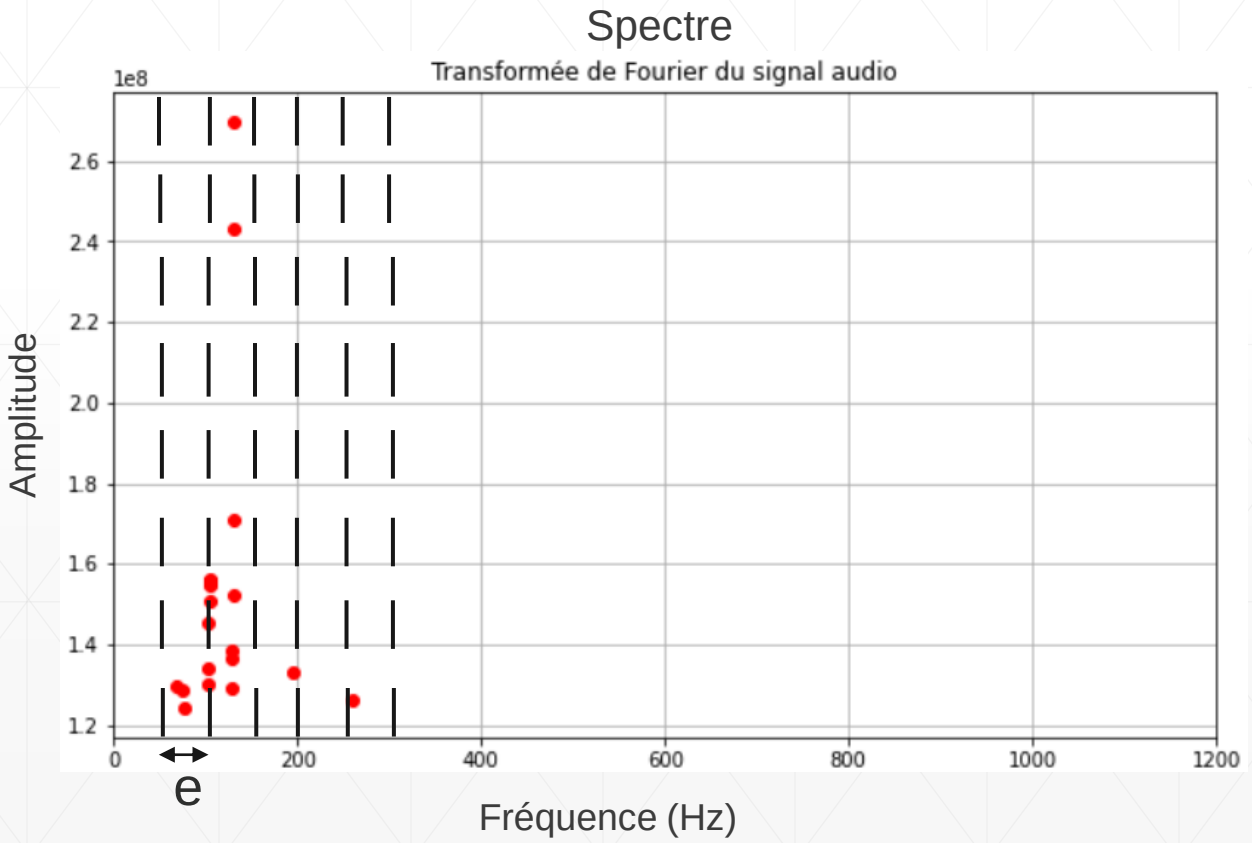


N=20 points

Méthode mise en œuvre

1 : Analyse de l’échantillon

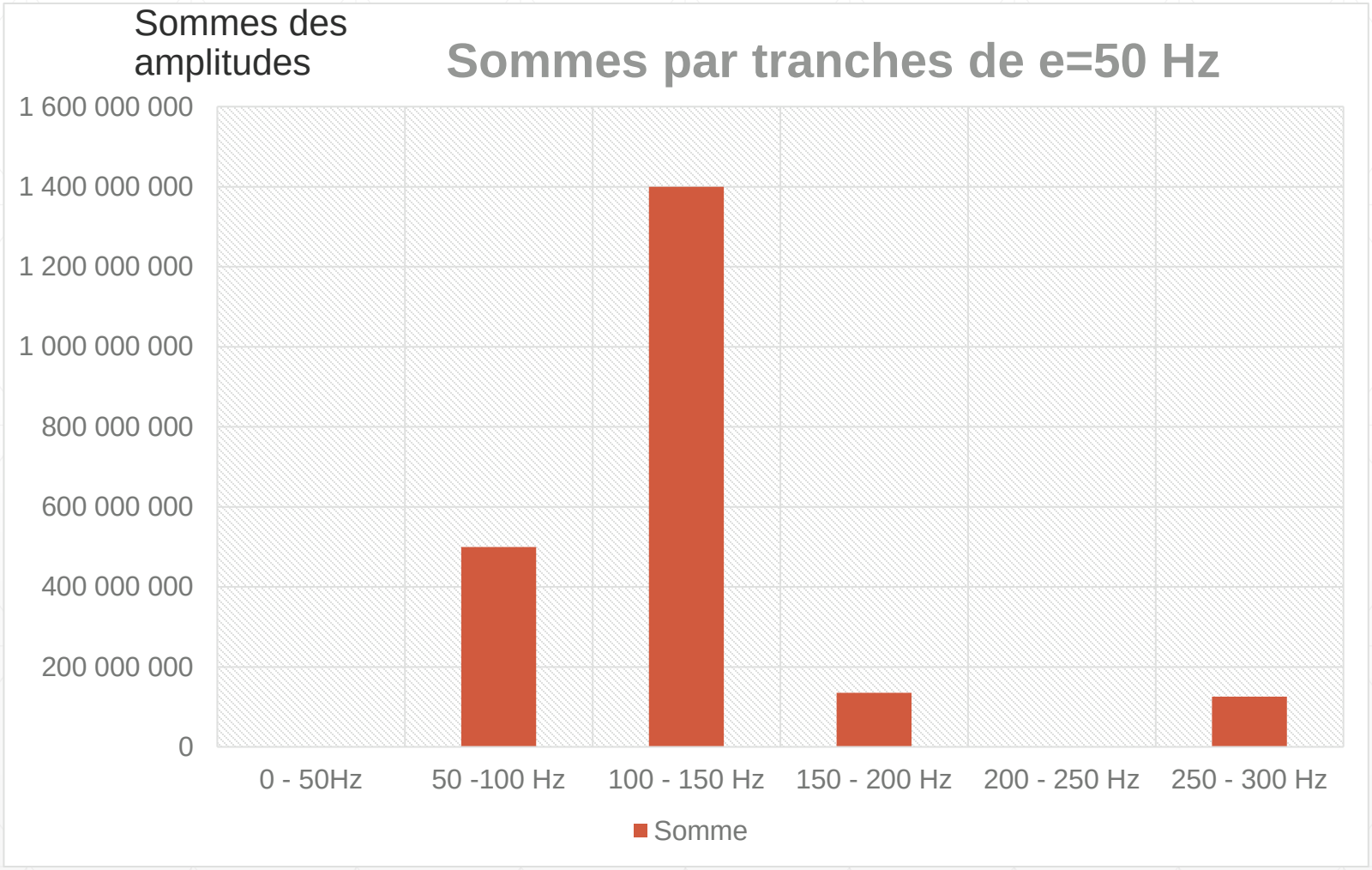
On somme les amplitudes par tranches de fréquences :



e = 50 Hertz

Méthode mise en œuvre

1 : Analyse de l'échantillon



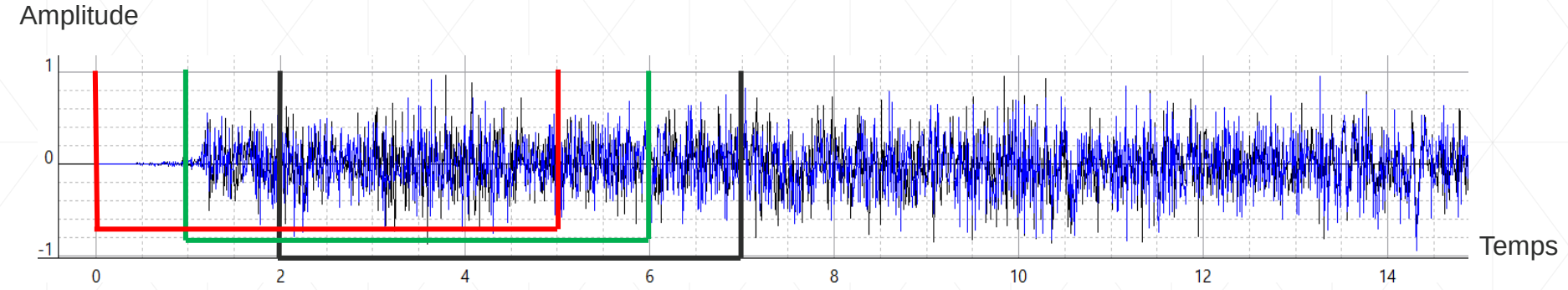
Méthode mise en œuvre

2 : Construction de la base de données

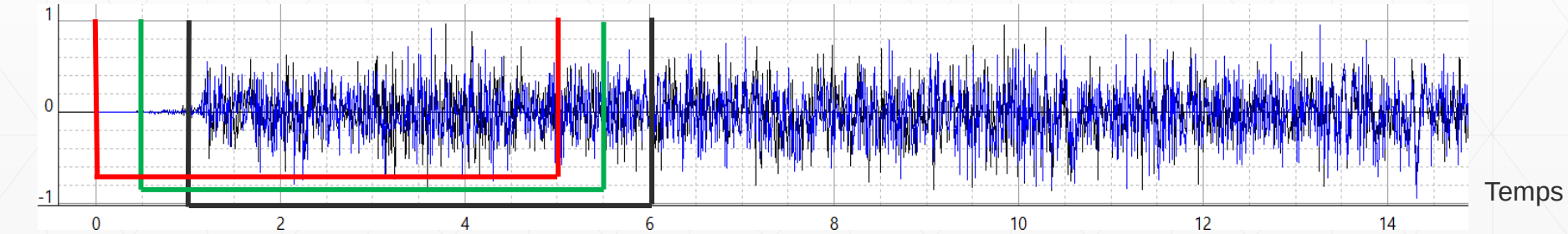
Méthode mise en œuvre

2 : Construction de la base de données

p= 1 seconde :

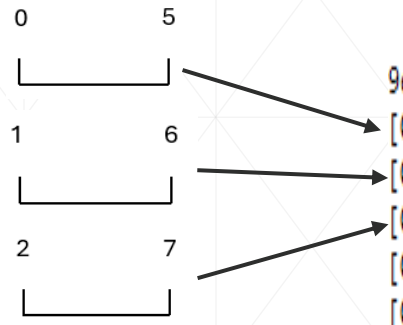


p= 0,5 secondes :



Méthode mise en œuvre

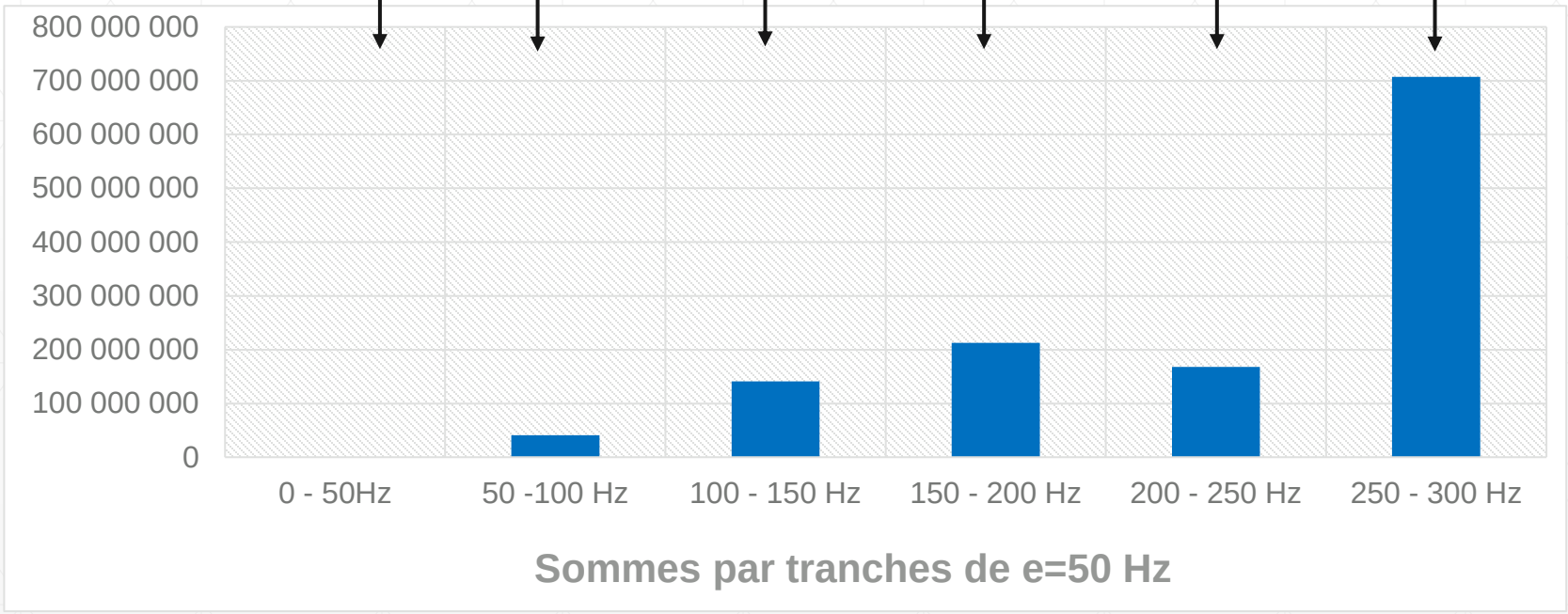
2 : Construction de la base de données



Si p= 1 seconde

9em_musique

[0, 146077846.7227971, 93918722.76673996, 389315636.2195614, 327366603.40471065, 0, 309777995.68599796, 0, 0,
[0, 97221712.47028235, 142297710.02028757, 314111274.54964375, 386256628.29924744, 0, 265575685.17002505, 0,
[0, 82659477.23281166, 103235360.73127005, 408088179.5003477, 288132745.25059485, 0, 222224614.47936156, 0, 0,
[0, 136994815.89218283, 148154618.81540996, 377793123.8930155, 191771116.12003648, 100180759.0182175, 1584482
[0, 100273751.43343213, 82216273.28984371, 322559415.00284463, 148984271.0032306, 289292690.19747937, 0, 8914
[0, 137789173.36469737, 130195306.71067137, 262775516.38033688, 0, 653176356.0529412, 0, 89200825.7140819, 0,
[0, 41358600.443409845, 141478835.59718004, 213020010.57464504, 168414377.05566105, 707881889.6508929, 0, 839



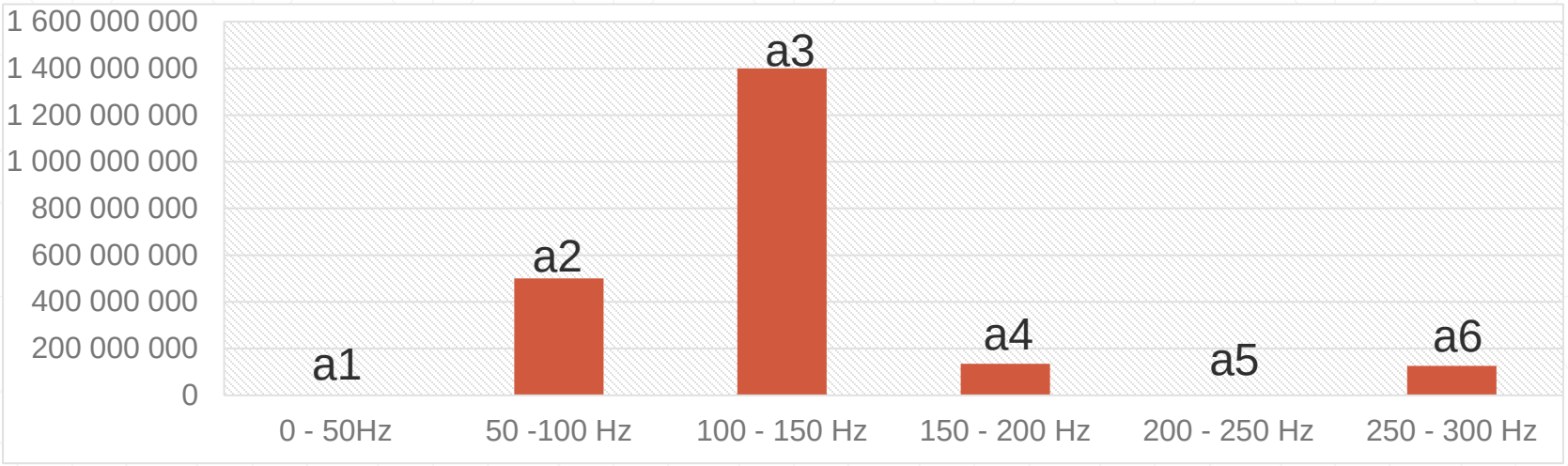
Méthode mise en œuvre

3 : Comparaison échantillon et base de données

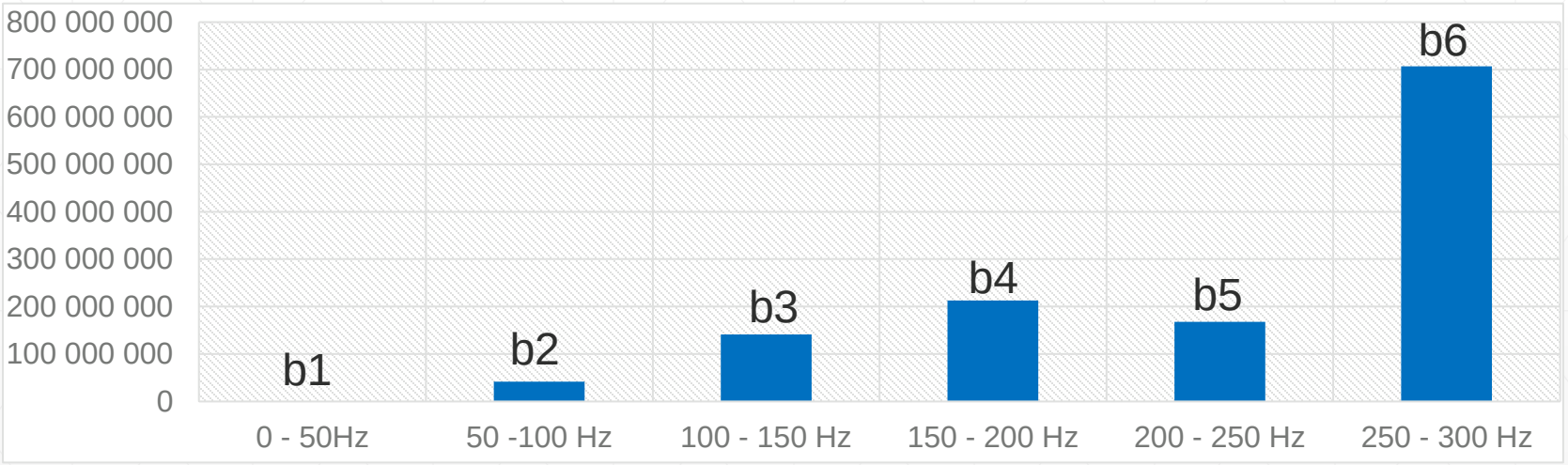
Méthode mise en œuvre

3 : Comparaison échantillon et base de données

Echantillon



Base De Données



$$Sk = |a1 - b1| + |a2 - b2| + |a3 - b3| + |a4 - b4| + |a5 - b5| + |a6 - b6| + \ldots$$

Méthode mise en œuvre

3 : Comparaison échantillon et base de données

Norme 1 : $$\|x - y\|_1 = \sum_i |x_i - y_i|$$

Norme 2 : $$\|x - y\|_2 = \sqrt{\sum_i |x_i - y_i|^2}$$

Méthode mise en œuvre

3 : Comparaison échantillon et base de données

On parcourt la base de données

↓

Liste_sommes_S = [S1, S2, S3, S4, S5, S6, …]

↓

m = min(Liste_sommes_S)

↓

i = Base_de_données.index(m) →

Base de données :

Adele - Someone like you

[0, 0, 13258915.30161498, 14492658.39782039, . . . , 0]

⋮

i → [0, 49986696.2830168, 15298473.815786073, 0, 0, …, 0]

⋮

[0, 0, 0, 0, 0, 0, 0, 30.33034113540441, 0, 0, 0, 0, . . . , 0]

Grover Washington - Just The Two of Us

[0, 875840767.9676921, 0, 0, 0, 0, 0, . . . , 0, 0]

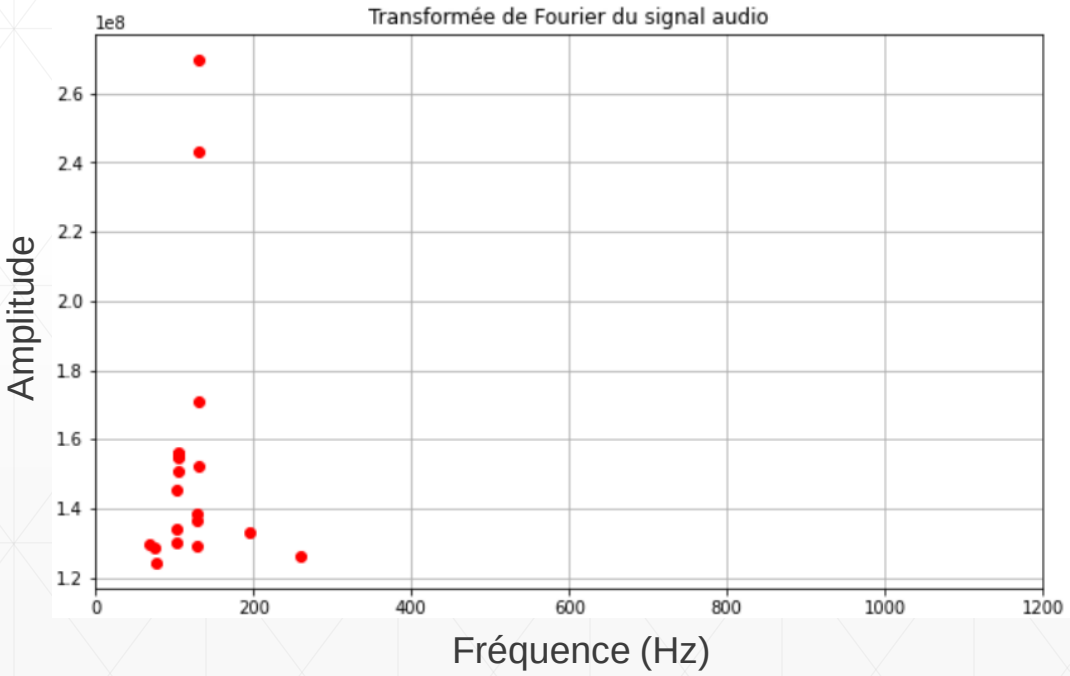
⋮

Plan d’expérience

Plan d’expérience

3 paramètres :

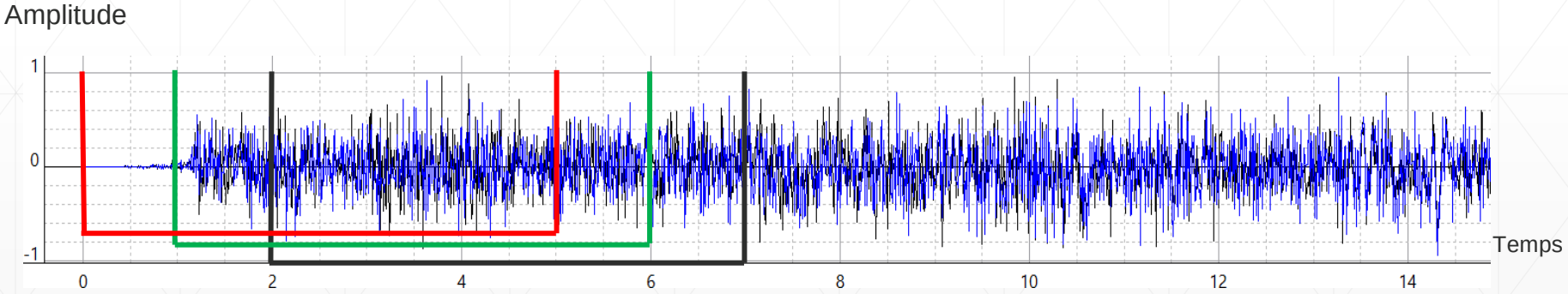
1) N : nombre de points sur un spectre



N = 20 points

Plan d’expérience

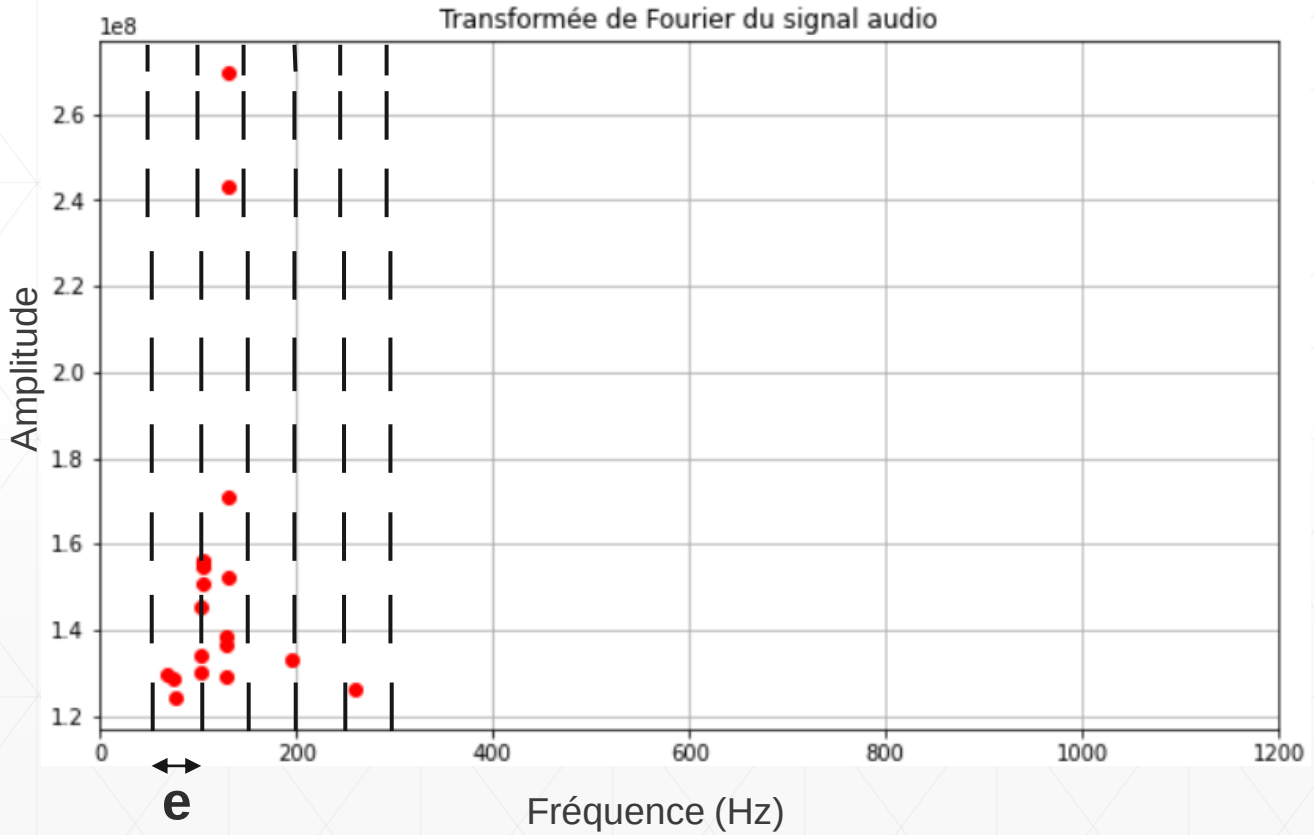
2) p : écart de temps entre deux spectre consécutifs dans la base de donnée



p= 1 seconde

Plan d'expérience

3) e : tranche de fréquences où on somme les amplitudes



e = 50 Hz

Plan d'expérience

p (s)	N	e (Hz)	Score (norme 1)	Temps (s) (norme 1)	Score (norme 2)	Temps (s) (norme 2)
2,5	50	200	84,0%	0,27	49,7%	0,24
2,5	50	50	98,4%	0,30	68,3%	0,26
2,5	20	200	79,4%	0,25	50,4%	0,22
2,5	20	50	95,9%	0,27	69,3%	0,24
1	50	200	89,8%	0,29	49,4%	0,25
1	50	50	99,6%	0,34	68,5%	0,30
1	20	200	82,9	0,27	56,3%	0,25
1	20	50	99,1%	0,32	72,3%	0,28

- **Score = $\frac{\text{nombre de succès}}{\text{nombre de tests}}$**

Succès : "le programme reconnait la bonne musique"

nombre de tests = 1000

- **Temps = $\frac{t1 + \ldots + t1000}{1000}$**

tk : "temps de compilation du k-ième test"

Annexe

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
import wave
from  random import randint
import contextlib
from random import random
import time

nb_mus=7 #nombre de musiques dans la bdd
var_musiques=["Robin Schulz & Topic ft. Oaks - One By One (Official Music Video)",
              "Adele - Someone Like You","9em_musique","Dmitri Shostakovich - Waltz No. 2",
              "Evanescence - Call Me When You're Sober","Green Day - Boulevard of Broken Dreams",
              "Grover Washington Jr. feat. Bill Withers - Just The Two of Us"]

#bdd = base de données

"""
paramètres
N : nombre de points dans le spectre
p : pas (Dans la base de donnée, pour chaque musique, on prend un spectre de 5s toutes les p secondes)
e : ecart de fréquence pour sommer au sein d'un spectre
"""

def reconnaitre_chanson(N,p,e):

    var_bdd=f"bdd N={N} p={p} e={e}"

    def choisir_chanson():
        return randint(0,nb_mus -1)
    var=var_musiques[choisir_chanson()]

    print(var)


    with contextlib.closing(wave.open(f"{var}.wav",'r')) as f:
        frames = f.getnframes()
        rate = f.getframerate()
        duration = frames / float(rate)
        #duration est la durée de la musique en secondes
```

Annexe

```
def choisir_extrait():
    start=randint(0,int(duration)-5-1)+random()
    """justification du -1 : si random() est supérieur à la partie décimale de
    duration, start+5 pourra etre en dehors de la chanson """
    return start
start=choisir_extrait()
end=start+5

# file to extract
with wave.open(f"{var}.wav", "rb") as infile:
    # get file data
    nchannels = infile.getnchannels()
    sampwidth = infile.getsampwidth()
    framerate = infile.getframerate()
    # set position in wave to start of segment
    infile.setpos(int(start * framerate))
    # extract data
    data = infile.readframes(int((end - start) * framerate))

# write the new file
with wave.open('exampleout.wav', 'w') as outfile:
    outfile.setnchannels(nchannels)
    outfile.setsampwidth(sampwidth)
    outfile.setframerate(framerate)
    outfile.setnframes(int(len(data) / sampwidth))
    outfile.writeframes(data)


#OUVRE CE FICHIER
sample_rate, audio_data = wavfile.read('exampleout.wav')


# Convertir le signal audio en mono en moyennant les canaux stéréo

if audio_data.ndim == 2:
    audio_data = np.mean(audio_data, axis=1)


# Calculer la transformée de Fourier

n = len(audio_data)
frequences = np.fft.fftfreq(n, 1 / sample_rate)
audio_fft = np.fft.fft(audio_data)
```

Annexe

```
"""

plt.figure(figsize=(10,6)) #Détermine la largeur et hauteur de votre graphique
plt.plot(frequences, np.abs(audio_fft))#prend le module de audio car peut avoir des valeurs complexes
plt.title('Transformée de Fourier du signal audio')
plt.xlabel('Fréquence (Hz)')
plt.ylabel('Amplitude')
plt.xlim(0, sample_rate/4)  # Afficher uniquement la partie positive partie des fréquences
plt.grid()
plt.show()

"""
plage_de_freq=2000
petites_freq=[]
def creation_petites_freq():
    for i in range(len(frequences)):
        if 0<=frequences[i]<=plage_de_freq:
            petites_freq.append(frequences[i])

creation_petites_freq()

l=len(petites_freq)
Amp=abs(audio_fft) #les amplitudes sont complexes
Amp_restreintes=[]
for i in range(l):
    Amp_restreintes.append(Amp[i])


freq_grandes_amp=[]
grandes_amp=[]

def creation_freq_grandes_amp_et_grandes_amp(N):
    for j in range(N):
        m=max(Amp_restreintes)
        ind=Amp_restreintes.index(m)
        grandes_amp.append(m)
        freq_grandes_amp.append(petites_freq[ind])
        Amp_restreintes.remove(m)
creation_freq_grandes_amp_et_grandes_amp(N)
```

Annexe

```
liste_des_s=[]


def créer_liste_des_s():
    for i in range (plage_de_freq//e):   #nombre de sommes
        a=i*e
        s=0
        for k in range(len(freq_grandes_amp)):
            if a<=freq_grandes_amp[k]<a+e:
                s=s+grandes_amp[freq_grandes_amp.index(freq_grandes_amp[k])]
        liste_des_s.append(s)

créer_liste_des_s()

bdd_liste=[]
liste_des_titres=[]
liste_des_S=[]
bdd=open(f"{var_bdd}.txt","r")
for k in bdd:
    bdd_liste.append(k)
bdd_liste.pop(0)


def comparaison():
    cpt=0
    for i in range(len(bdd_liste)): #parcours les lignes de la bdd
        S=0
        if bdd_liste[i][0]=='[': #si ce n'est pas un titre
            b=bdd_liste[i][1:len(bdd_liste[i])-2]
            """-2 pour enlever '\n' et ']' à la fin de la chaîne de caractères"""
            P=b.split(', ')
            """on obtient une liste dont les éléments sont de type str"""

            for j in range(int(plage_de_freq//e)): #plage de freq=2000
                S=S+abs(float(P[j])-liste_des_s[j])
                #sommes des valeurs absolues de la différence de chaque terme
                #float pour passer du type str à type float
            liste_des_S.append(S)
        else:
            titre=(bdd_liste[i][:len(bdd_liste[i])-1],bdd_liste.index(bdd_liste[i])-cpt)
            """titre est un couple composé du titre de la musique et de l'indice
            du titre de cette musique moins les lignes correspondants aux titres précédents """
            liste_des_titres.append(titre)
            cpt=cpt+1
    m=min(liste_des_S)
    ind=liste_des_S.index(m)
```

Annexe

```
    n=0
    B=True
    while B:
        B=True
        if (len(liste_des_titres)<=n+1) or (liste_des_titres[n][1]<=ind<liste_des_titres[n+1][1]):
            "premiere condition = condition d'arret pour eviter le out of range"
            B=False
        else :
            n=n+1
    return liste_des_titres[n][0]

print(comparaison())
```

Annexe

```
def créer_bdd(N,p,e):


    bdd=open(f"bdd N={N} p={p} e={e}.txt",'w')
    """'w' pour write: si le fichier n'existe pas -> il est créé, sinon -> les données
    préexistantes sont ecrasées et remplacées par les nouvelles"""
    bdd.close()
    bdd=open(f"bdd N={N} p={p} e={e}.txt","a")
    bdd.write(f'bdd N={N} p={p} e={e}')
    bdd.write('\n')
    bdd.close()

    var_musiques=["Robin Schulz & Topic ft. Oaks - One By One (Official Music Video)",
                  "Adele - Someone Like You","9em_musique","Dmitri Shostakovich - Waltz No. 2",
                  "Evanescence - Call Me When You're Sober","Green Day - Boulevard of Broken Dreams",
                  "Grover Washington Jr. feat. Bill Withers - Just The Two of Us"]

    for variable in range(len(var_musiques)):

        title=var_musiques[variable]
        var=var_musiques[variable]

        start=0
        T=5 #échantillon de 5 secondes


        with contextlib.closing(wave.open(f"{var}.wav",'r')) as f:
            frames = f.getnframes()
            rate = f.getframerate()
            duration = frames / float(rate)
            print(duration)

        end=int(duration)


        bdd=open(f"bdd N={N} p={p} e={e}.txt","a")
        bdd.write(title+'\n')
        bdd.close()
```

Annexe

```
# file to extract
with wave.open(f"{var}.wav", "rb") as infile:
    # get file data
    nchannels = infile.getnchannels()
    sampwidth = infile.getsampwidth()
    framerate = infile.getframerate()
    # set position in wave to start of segment
    infile.setpos(int(start * framerate))
    # extract data
    data = infile.readframes(int((end - start) * framerate))

# write the new file
with wave.open('exampleout.wav', 'w') as outfile:
    outfile.setnchannels(nchannels)
    outfile.setsampwidth(sampwidth)
    outfile.setframerate(framerate)
    outfile.setnframes(int(len(data) / sampwidth))
    outfile.writeframes(data)



#OUVRE CE FICHIER
sample_rate, audio_data = wavfile.read('exampleout.wav')

# Convertir le signal audio en mono en moyennant les canaux stéréo

if audio_data.ndim == 2:
    audio_data = np.mean(audio_data, axis=1)


# Calculer la transformée de Fourier

n = len(audio_data)
frequences = np.fft.fftfreq(n, 1 / sample_rate)
audio_fft = np.fft.fft(audio_data)


for k in range((end-T)//T):
    """end-T est nécéssaire, sinon on aurait été jusqu'à un temps de end+T-p dans la chanson,
    ce qui pourrait être en dehors de la chanson.
    On perd donc de l'information à la fin de la chanson sur un
    temps de p+(la partie décimale de duration)"""
```

Annexe

```
for c in range (int(T//p)):

    # extraction des sous fichiers
    start=T*k+p*c
    end=(k+1)*T+p*c
    with wave.open('exampleout.wav', "rb") as infile:
        # get file data
        nchannels = infile.getnchannels()
        sampwidth = infile.getsampwidth()
        framerate = infile.getframerate()
        # set position in wave to start of segment
        infile.setpos(int(start * framerate))
        # extract data
        data = infile.readframes(int((end - start) * framerate))

    # write the new file
    with wave.open('time'+str(c)+'.wav', 'w') as outfile:
        outfile.setnchannels(nchannels)
        outfile.setsampwidth(sampwidth)
        outfile.setframerate(framerate)
        outfile.setnframes(int(len(data) / sampwidth))
        outfile.writeframes(data)


    #OUVRE SOUS FICHIER
    sample_rate, audio_data = wavfile.read('time'+str(c)+'.wav')

    # Convertir le signal audio en mono en moyennant les canaux stéréo

    if audio_data.ndim == 2:
        audio_data = np.mean(audio_data, axis=1)

    # Calculer la transformée de Fourier

    n = len(audio_data)
    frequences = np.fft.fftfreq(n, 1 / sample_rate)
    audio_fft = np.fft.fft(audio_data)

    plage_de_freq=2000
    petites_freq=[]
    def creation_petites_freq():
        for i in range(len(frequences)):
            if 0<=frequences[i]<=2000:
                petites_freq.append(frequences[i])
```

Annexe

```
creation_petites_freq()

l=len(petites_freq)
Amp=abs(audio_fft)
Amp_restreintes=[]
for i in range(l):
    Amp_restreintes.append(Amp[i])



freq_grandes_amp=[]
grandes_amp=[]

def creation_freq_grandes_amp_et_grandes_amp(N):
    for j in range(N):
        m=max(Amp_restreintes)
        ind=Amp_restreintes.index(m)
        grandes_amp.append(m)
        freq_grandes_amp.append(petites_freq[ind])
        Amp_restreintes.remove(m)
creation_freq_grandes_amp_et_grandes_amp(N)



liste_des_sommes=[]


def créer_liste_des_sommes():
    for i in range (int(plage_de_freq//e)):
        a=i*e
        s=0
        for k in range(len(freq_grandes_amp)):
            if a<=freq_grandes_amp[k]<a+e:
                s=s+grandes_amp[freq_grandes_amp.index(freq_grandes_amp[k])]
        liste_des_sommes.append(s)

créer_liste_des_sommes()
bdd=open(f"bdd N={N} p={p} e={e}.txt","a")
x=str(liste_des_sommes)
bdd.write(x+'\n')
bdd.close()
```

Annexe

```
liste_N=[20,50]
liste_p=[1,2.5]
liste_e=[50,200]

def créer_8bdd():
    t1=time.process_time()
    for para_N in range(2):
        N=liste_N[para_N]
        for para_p in range(2):
            p=liste_p[para_p]
            for para_e in range(2):
                e=liste_e[para_e]

                créer_bdd(N,p,e)
                t2=time.process_time()
    print(t2-t1) #seule la difference entre 2 process time et valide
```