Question 1 :

Non, l'énergie des morceaux ne se répartit pas uniformément sur tout le spectogramme. Dans le test ci-dessous, on constate que 90% de l'énergie est contenue dans moins de 2% des coefficients.

Input sur *algorithm.py* :

```python
if __name__ == '__main__':

    encoder = Encoding(128, 120)
    fs, s = read('samples/Dark Alley Deals - Aaron Kenny.wav')
    encoder.process(fs, s)
    encoder.nb_min_coeff()
```

Output :

```
(base) leonard@air-de-leonard Shazam % python algorithm.py
Il suffit de 1788397 coefficients sur 105320020 (soit 0.016980598750361044) pour
 obtenir 90% de l'énergie
```

Question 2 :

Par tâtonnement (lors des questions 5 et 6), nous avons choisi un *peak_local_max* de 1000, un *nperseg* de 128 et un *noverlap* de 120

Question 3 :

Oui, la représentation sous forme de hash est invariante par transation (temporelle), car l'information temporelle retenue n'est que l'écart entre l'ancre et la cible

Question 4 :

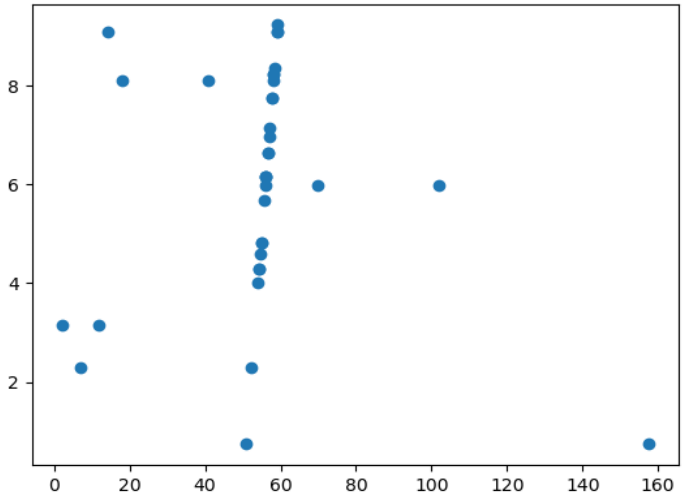
Lorsque le morceau et l'extrait correspondent, une corrélation apparaît (le nuage de point forme une droite). Lorsqu'ils ne correspondent pas, les points semblent être répartis uniformément sur le graph. Ce phénomène est illustré par les deux exemples ci-dessous.

Premier cas : l'extrait correspond au morceau

Input sur *algorithm.py* :

```python
if __name__ == '__main__':

    encoder = Encoding(128, 120)
    encoder_2 = Encoding(128, 120)

    fs, s = read('samples/Dark Alley Deals - Aaron Kenny.wav')

    encoder.process(fs, s)
    encoder_2.process(fs, s[50*fs: 60*fs])
    matching = Matching(encoder.signature, encoder_2.signature)
    matching.display_scatterplot()
```

Output :

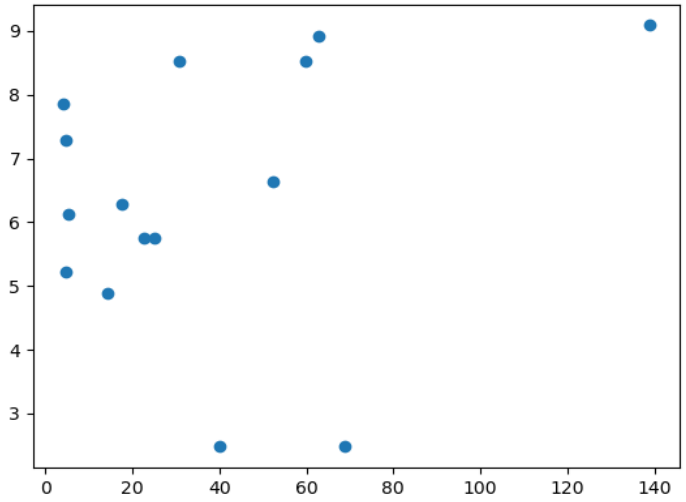


Second cas : l'extrait ne correspond pas au morceau

Input sur *algorithm.py* :

```
if __name__ == '__main__':

    encoder = Encoding(128, 120)
    encoder_2 = Encoding(128, 120)

    fs, s = read('samples/Dark Alley Deals - Aaron Kenny.wav')
    encoder.process(fs, s)

    fs, s = read('samples/Jal - Edge of Water - Aakash Gandhi.wav')
    encoder_2.process(fs, s[50*fs: 60*fs])

    matching = Matching(encoder.signature, encoder_2.signature)
    matching.display_scatterplot()
```

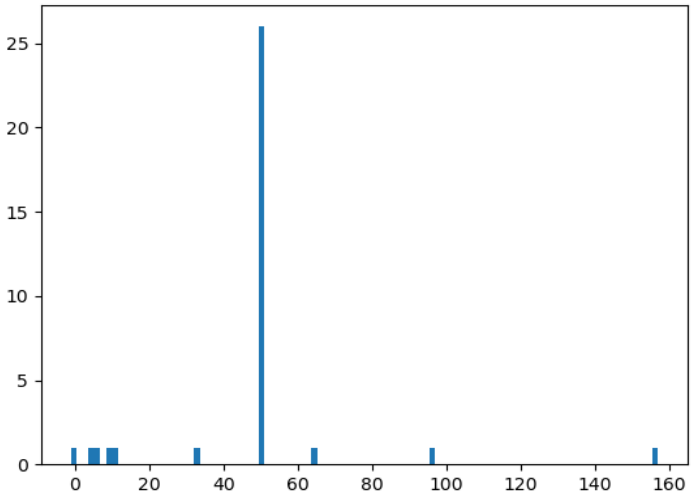
Output :



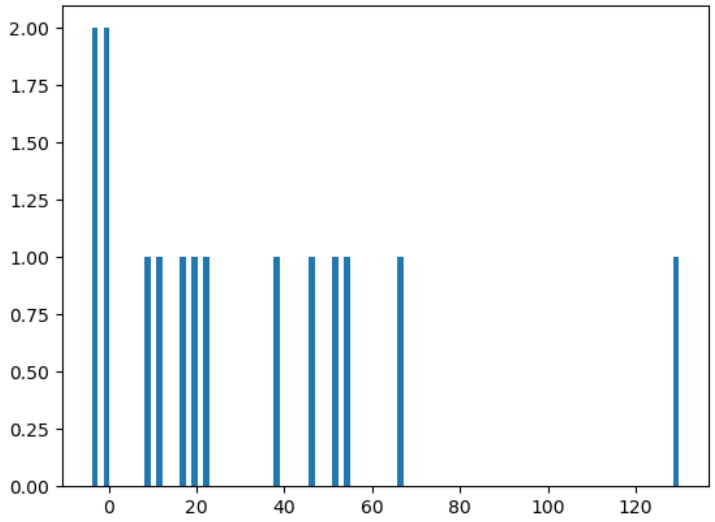
Question 6 :

Les histogrammes présentent un pic très net lorsque l'extrait correspond au morceau. Pour illustrer ce phénomène, nous utilisons les deux exemples précédents (mêmes inputs) :

Premier cas : l'extrait correspond au morceau



Second cas : l'extrait ne correspond pas au morceau



Question 7 :

Nous avons choisi le critère suivant : nous calculons le pic le plus haut de l'histogramme et le second pic le plus haut (distinct du premier). Si la hauteur du premier est plus de 3 fois supérieure au second, on considère que l'extrait correspond au morceau. Nous avons aussi fixé comme condition que le nombre de points doit dépasser 10, pour éviter les "faux positifs" en cas d'échantillon très faible.
Cela se traduit par le morceau de code suivant, de la classe *matching* :

```
def does_it_match(self):
    ys, xs, z = plt.hist(self.difference, bins=100)
    first_peak = 0
    second_peak = 0
    for peak in ys:
        if peak > first_peak:
            second_peak = first_peak
            first_peak = peak
        elif peak > second_peak:
            second_peak = peak

    if first_peak > 3 * second_peak and len(self.time_1) > 10:
        return True
    else:
        return False
```

Question 8 :

Nous avons rempli le fichier demo.py et vérifié que tous les extraits tirés aléatoirement (commencant à 50s et durant 10s) étaient reconnus.
Pour un test plus systématique, nous avons créé le fichier test_all.py , comportant la fonction *test_all*, qui pour un temps de début et un durée donnée, vérifie que pour tous les morceaux de la base de donnée, un extrait ayant ces caractéristiques puisse être reconnu par l'algorithme.
Nous avons observé que *test_all* fonctionne pour une durée des extraits supérieure ou égale à 10s. Pour une durée inférieure, la réussite du test n'est plus garantie. Plus l'extrait est

long, plus le calcul de sa signature va demander du temps, donc il semble que la durée de 10s est optimale pour notre algorithme.