UE 455 TP 1 Théorie de l'Information - Codage de Source

SHI Wenli

16 Février 2024

Table des matières

0 Introduction

L'objectif de ce TP est d'estimer l'entropie d'une source, par exemple d'un texte, d'une image, ou d'un fichier audio. Une seconde partie est consacrée à l'étude de deux type de quantificateurs scalaires et d'évaluer le compromis en débit et distorsion lors d'une quantification scalaire.

Ces TP seront réalisés de préférence en `python`. L'environnement de développement conseillé est `Spyder`.

1 Partie 1 - Entropie

On considère une liste `x` contenant une suite d'entiers. Dans un premier temps, nous allons construire une fonction d'évaluation de l'entropie de la source X ayant généré `x`, en faisant l'hypothèse que cette source est sans mémoire.

1.1 Entropie sans mémoire

Manipulation: 1 Comment évaluer la fréquence d'apparition de chaque symbole de `x` ?

Dans ce problème, on doit compter la fréquence d'apparition de chaque symbole dans la source `x`, où `x` est une série d'entiers. La méthode de calcul est la suivante :

$$p(x) = \frac{n(x)}{\texttt{len}(x)}$$

où $n(x)$ est le nombre d'occurrences de chaque symbole `x`.

En `python`, on peut d'abord initialiser un dictionnaire pour stocker le nombre d'occurrences de chaque symbole. De plus, on peut utiliser la méthode `set()` pour obtenir l'alphabet de la source et la fonction `count()` pour compter le nombre d'occurrences de chaque symbole. Ensuite, utilisez la fonction `len()` pour obtenir la longueur totale de la source, puis on divise le nombre d'occurrences de chaque élément de la liste des fréquences par la longueur totale. L'implémentation du code sera montrée dans Manipulation 3.

Manipulation: 2 Donner l'expression d'une estimée de l'entropie de la source ayant généré `x` à partir des fréquences d'apparition des symboles de `x`.

L'expression d'une estimée de l'entropie de la source `x` est :

$$H(x) = -\sum_{x \in X} p(x) \log_2 p(x)$$

où $p(x)$ est la fréquence d'apparition de symbole de `x`.

Manipulation: 3 Construire d'une fonction dont l'en-tête est `def entropy(x):` et prenant `x` en entrée et renvoyant une estimée de l'entropie de la source ayant généré `x`.

D'après les réponses aux deux premières manipulations, on peut implémenter cette fonction en `python`. Le code d'implémentation est le suivant :

Listing 1 – Fonction - entropy()

```
1  import numpy as np
2
3  def entropy(x):
4      """ Fonction pour calcule l'entropie sans memoire de la source x """
5      # Initialiser un dictionnaire pour stocker
6      frequences = {}
7      longueur = len(x)
8
9      symboles_set = set(x)
10     print(f"Alphabet de x est \n {symboles_set}")
11     frequences = {symbole: x.count(symbole) for symbole in symboles_set}
12
13     entropie = 0
```

```
    for frequence in frequences.values():
        probabilite = frequence / longueur
        entropie -= probabilite * np.log2(probabilite)

    return entropie
```

Manipulation: 4 Charger le fichier `Declaration1789.txt` et donner l'alphabet de cette source.

On utilise le code suivant pour lire un fichier texte, et utilise la méthode `set()` et la fonction `print()` pour donner l'alphabet. On met le format d'encodage du fichier sur `utf-8`.

Listing 2 – Lecture de texte

```
# Lecture de 'Declaration1789.txt'
with open('Declaration1789.txt', 'r', encoding='utf-8') as file:
    declaration_text = file.read()
```

```
Alphabet de x est
 {']', 'j', '0', 'd', 'N', 's', 'ç', 'E', '4', 'P', 'Ê', 'p', '.', '7', 'ê', 'à',
'5', '-', ';', "'", 'x', 'b', ',', 'L', 'î', 'T', '2', '\n', '1', 'm', 'g', 'a', 'i',
'3', 't', 'y', 'C', '8', 'e', '6', ' ', 'l', 'è', 'é', 'n', 'v', 'û', 'h', 'u', 'r',
'[', 'f', ':', '9', 'c', 'A', 'q', 'o'}
```

FIGURE 1.1.1 – Output de Console - Alphabet.

Manipulation: 5 Estimer l'entropie de la source ayant généré ce texte.

En appelant la fonction `entropy()`, on peut estimer l'entropie de `x`.

Listing 3 – Output de Console - Entropie

```

Entropie estimee du texte est : 4.33 bits/symbole.
```

1.2 Entropie de Markov d'ordre 1

La source ayant généré `x` n'est plus considérée sans mémoire.

Manipulation: 6 Evaluer la fréquence d'apparition de chaque paire distincte de symboles de `x`.

Pour le modèle de Markov d'ordre 1, on doit considérer la fréquence d'apparition de différentes paires de symboles et construire une matrice de probabilité de transition $\mathbf{P}$. Pour cette matrice, sa taille est $n \times n$, où `n` est le nombre total de symboles dans l'alphabet de la source. $\mathbf{P}_{i,j}$ dans la matrice est exprimé comme la probabilité que le prochain symbole x_j apparaisse lorsque le symbole est x_i. Dans cette matrice, la somme de tous les éléments de chaque ligne est 1. Cela peut être exprimé par la formule suivante :

$$\mathbf{P}_{i,j} = \texttt{Pr}(x_n = a_j | x_{n-1} = a_i)$$

La valeur de chaque élément de la matrice peut être implémentée en Python à l'aide de `Numpy`.

On utilise la même méthode que précédemment pour obtenir l'alphabet, puis initialise un vecteur de probabilité. Le vecteur de probabilité $\mathbf{p}$ peut être exprimé par la formule suivante :

$$\mathbf{p} = (\texttt{Pr}(x_{n-1} = a_1), \ldots, \texttt{Pr}(x_{n-1} = a_J))^T$$

où J est le nombre total de symboles dans l'alphabet de la source `x`.

Ensuite, on initialise une matrice de probabilité de transition. On parcourt toute la source et ajoute le nombre d'occurrences de chaque paire de symboles à la matrice. Enfin, chaque ligne de la matrice doit être normalisée. Le code d'implémentation est présenté dans la Manipulation 8.

Manipulation: 7 Donner l'expression d'une estimée de l'entropie de la source ayant généré `x` en supposant que cette source est décrite par un modèle de Markov d'ordre 1.

L'expression d'une estimée de l'entropie de Markov d'ordre 1 de la source `x` est

$$\begin{aligned} H(x) &= H(x_2|x_1) \\ &= -\sum_{(x_1,x_2)\in X^2} \texttt{Pr}(X_2 = x_2|X_1 = x_1) \cdot \texttt{Pr}(X_1 = x_1) \cdot \log_2 \texttt{Pr}(X_2 = x_2|X_1 = x_1) \end{aligned}$$

Manipulation: 8 Construire d'une fonction dont l'en-tête est `def entropy1(x):` et prenant `x` en entrée et renvoyant une estimée de l'entropie de la source ayant généré `x` en supposant que cette source est décrite par un modèle de Markov d'ordre 1.

D'après les réponses aux deux premières manipulations, on peut implémenter cette fonction en `python`. Le code d'implémentation est le suivant :

Listing 4 – Fonction - entropy2()

```
def entropy2(x):
    """ Fonction pour calcule l'entropie de Markov d'ordre 1 de la source x """
    longueur_p = len(x)   # Longueur de la source

    symboles_set_mar = set(x)
    longueur_set = len(symboles_set_mar)    # Nombre d'alphabet
    # Frequence d'apparition
    frequences_p = {symbole: x.count(symbole) / longueur_p for symbole in
        symboles_set_mar}

    # Vecteur de probabilite
    symbole_array = np.array(list(frequences_p.values()))
    print(f"Le vecteur de probabilite est \n {symbole_array}")

    symbole_index = {symbole:index for index, symbole in enumerate(symboles_set_mar)}

    # Pour strocker les frequences des paire
    transition_matrice = np.zeros((longueur_set, longueur_set))

    for i in range(longueur_p - 1):
        symbole_current = x[i]
        symbole_next = x[i+1]

        current_index = symbole_index[symbole_current]
        next_index = symbole_index[symbole_next]

        # Stocker les paires dans la matrice de transition
        transition_matrice[current_index, next_index] += 1

    # Normalisation de la matrice de transition (matrice de probabilite)
    transition_matrice /= transition_matrice.sum(axis=1, keepdims=True)

    # Affichage de la matrice de transition
    #print(f"La matrice de transition est \n {transition_matrice}")

    entropie2 = 0

    # Calcul de l'entropie du modele de Markov d'ordre 1
    for i in range(longueur_set):
        entropie2 += -sum(transition_matrice[i,:] * symbole_array[i] * np.log2(
            transition_matrice[i,:] + 1e-10))

    return entropie2
```

Manipulation: 9 Evaluer l'entropie de la source ayant généré le contenu de `Declaration1789.txt` en supposant que cette source est décrite par un modèle de Markov d'ordre 1.

En appelant la fonction `entropy2()`, on peut estimer l'entropie de Markov d'ordre de `x`.

Listing 5 – Output de Console - Entropie2

```
Entropie de Markov du texte: 3.16 bits/symbole
```

Manipulation: 10 Comparer et commenter les résultats obtenus.

Listing 6 – Output de Console - Entropie et Entropie2

```
Entropie estimee du texte est : 4.33 bits/symbole.
Entropie de Markov du texte: 3.16 bits/symbole
```

En comparant les résultats de l'estimation de l'entropie des sources text `Declaration1789.txt` entre deux modèles différents : on peut constater que l'entropie de l'information obtenue à l'aide du modèle de Markov du premier ordre est plus petite.

$$H_{Markov} < H_{sm}$$

L'entropie de l'information est une mesure de l'incertitude de l'information. Pour la source x du modèle sans mémoire, l'entropie H_{sm} indique que chaque symbole est indépendant, sans tenir compte de la corrélation entre les symboles adjacents. Pour le modèle de Markov d'ordre 1, l'entropie H_{Markov} prend en compte la corrélation entre les symboles adjacents, c'est-à-dire que la probabilité d'occurrence de chaque symbole sera affectée par son symbole précédent.

De ce point de vue, le modèle de Markov d'ordre 1 considère plus d'informations, donc l'entropie obtenue par ce modèle est plus petite que celle du modèle sans mémoire.

Manipulation: 11 Reprenez les résultats précédents avec le fichier `Alarm05.wav`. Commenter.

On prend un voix dans le signal audio pour calculer l'entropie. Pour la fonction de calcul d'entropie précédente, puisque `ndarray` n'a pas de méthode `set()`, on écrit une nouvelle fonction de calcul d'entropie appliquée au type `ndarray`.

D'après les résultats de sortie, on peut voir que l'entropie de signau non quantifié est relativement grande.

Listing 7 – Output de Console - Entropie d'audio

```
Entropie d'audio: 11.08 bits/symbole
```

Listing 8 – Fonction - entropy3()

```
def entropy3(x):
    """ Fonction pour calcule l'entropie sans memoire de la source x """

    alphabet = []
    occurrences = []
    # Pour ndarray
    x=list(x)
    for i in range(0,len(x)):
        if x[i] not in alphabet:
            alphabet.append(x[i])
            occurrences.append(x.count(x[i]))

    freq = np.array(occurrences)/len(x)
    entropie3 = -sum(freq*np.log2(freq))

    return entropie3
```

2 Partie 2 - Quantification

L'objectif de cette seconde partie du TP est de comparer les performances de deux type de quantificateurs uniformes, le quantificateur de type *mid-rise* et le quantificateur de type *mid-tread*. Dans un premier temps, nous allons considérer des réalisation d'une source Gaussienne de moyenne nulle et de variance unité.

2.1 Quantificateur de type *mid-rise*

Manipulation: 1 Construire une fonction dont l'en-tête est `def quant_midrise(x,Delta):` réalisant une quantification de type *mid-rise* de `x` avec un pas `Delta`. La sortie de cette fonction est un tuple comprenant une liste `qx` des valeurs quantifiées de `x` et une liste `idx` des index de quantification obtenus. Les index de quantification ne sont pas bornés, ainsi `idx` peut prendre toute valeur entière positive ou négative.

Pour un quantificateur de type *mid-rise*, pour la source `x` avec un pas `Delta`, la valeur quantifiée `qx` et l'indice de quantification `idx` peuvent être exprimés par les formule suivantes :

$$\texttt{idx} = \left\lfloor \frac{x}{\texttt{Delta}} \right\rfloor$$

$$\texttt{qx} = \texttt{Delta} \times \texttt{idx} + \frac{\texttt{Delta}}{2}$$

D'après les relations ci-dessus, on peut implémenter cette fonction.

Listing 9 – Fonction - quant_midrise()

```
1 import numpy as np
2
3 def quant_midrise(x, Delta = 1):
4     """ Fonction pour la quantification de type Mid-rise """
5     idx = np.floor(x / Delta)
6     qx = idx * Delta + Delta / 2.0
7
8     return idx, qx
```

Manipulation: 2 Tracer la caractéristique entrée-sortie du quantificateur *mid-rise*. Pour cela, mettre à l'entrée du quantificateur une liste de nombres allant de -10 à 10. Vous pourrez utiliser la fonction arange de la bibliothèque `numpy`.

La sortie de ce quantificateur peut être tracée en appelant la fonction précédente et en utilisant le code suivant. Le résultat est présenté dans la figure.

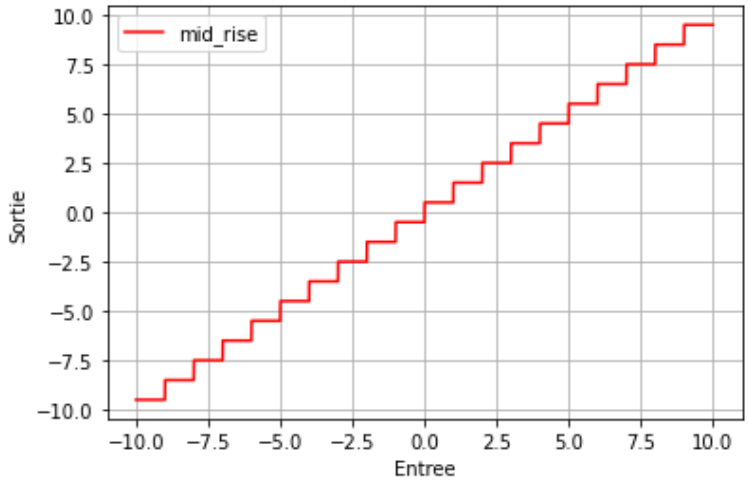

FIGURE 2.1.1 – Trace de Mid_rise.

Listing 10 – Trace - quant_midrise()

```
import numpy as np
import matplotlib.pyplot as plt

x1 = np.arange(-10, 10, 0.0001) # Generer une liste de nombres de -10 a 10
Delta = 1                       # Pas de quantification

idx1, qx1 = quant_midrise(x1, Delta)

plt.plot(x1, qx1, '-r')
plt.legend(['mid_rise'])
plt.xlabel('Entree')
plt.ylabel('Sortie')
plt.grid()
plt.show()
```

Manipulation: 3 Générer un tableau `x` contenant $N = 10000$ réalisations de variables indépendantes et identiquement distribuées suivant une loi Gaussienne de moyenne nulle et de variance unité. Vous pourrez utiliser la fonction `random.normal` de la bibliothèque `numpy`.

On peut utiliser le code suivant pour construire un tableau comme décrit dans la question. Ce tableau contient 10 000 paramètres indépendants et distribués de manière identique qui obéissent à une distribution gaussienne.

Listing 11 – Tableau x

```
# Generer un tableau de la variable gaussienne
N = 10000
x = np.random.normal(loc=0, scale=1, size=N)
```

Manipulation: 4 Pour différentes valeurs de `Delta`, évaluer la distorsion introduite par le quantificateur, en considérant une mesure de distorsion quadratique

$$D = \frac{1}{N}\sum_{i=1}^{N}(x_i - q_i)^2$$

où q_i est la valeur quantifiée de x_i. Evaluer aussi l'entropie de `idx`.

Selon la formule de calcul donnée de l'erreur quadratique moyenne, on peut écrire une fonction pour calculer l'erreur quadratique moyenne. Pour le calcul de l'entropie de l'information, on peut appeler la fonction `entropy()` dans la section précédente.

On crée une liste pour stocker différentes valeurs `Delta` et calcule la valeur de MSE et l'entropie sur `idx` pour différentes valeurs `Delta`. Le résultat est le suivant :

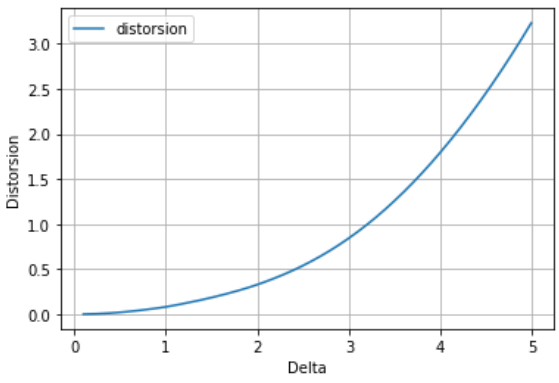

FIGURE 2.1.2 – Evaluation de distorsion.

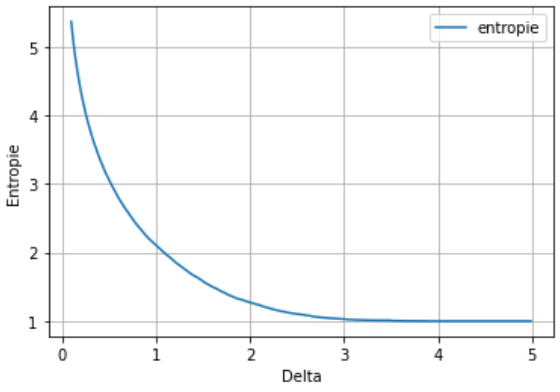

FIGURE 2.1.3 – Evaluation d'entropie.

Lorsque on observe les résultats de sortie, on peut voir que lorsque la valeur de `Delta` augmente, la valeur de l'entropie diminue progressivement car les informations quantifiées sont réduites. Pour la valeur de MSE, lorsque `Delta` est plus petit, la valeur de distorsion est plus petite, car la quantification est plus précise.

L'implémentation du code de cette partie est la suivante :

Listing 12 – Calcul de distorsion et entropy

```
def distorsion_quad(x, qx):
    """ Fonction pour calculer la distorsion quadratique """
    dis_quad = sum((x - qx)**2) / 10000

    return dis_quad

# Differentes valeurs de Delta a evaluer
valeurs_delta = np.arange(0.1, 5, 0.01)

distorsions = []
entropies = []

# Pour chaque valeur de Delta
for Delta in valeurs_delta:
    # Quantifier x avec le Delta actuel
    idx, qx = quant_midrise(x, Delta)

    # Calculer la distorsion
    distorsion_ac = distorsion(x, qx)
    distorsions.append(distorsion_ac)

    # Calculer l'entropie de idx
    entropie_ac = entropy(idx)
    entropies.append(entropie_ac)
```

Manipulation: 5 Tracer la distorsion en fonction de l'entropie des index de quantification obtenus à la sortie du quantificateur de type *mid-rise*. Commenter la courbe obtenue.

La courbe de distorsion en fonction de l'entropie est présentée ci-dessous :

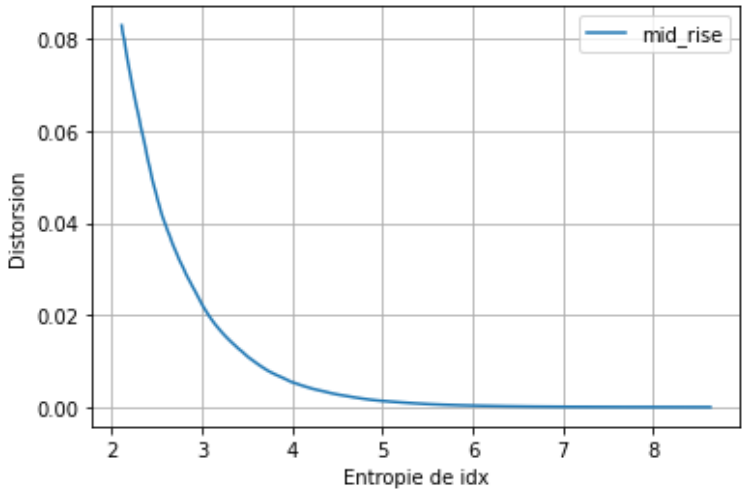

FIGURE 2.1.4 – Trace de distorsion sur entropie de Mid_rise.

En observant cette courbe, on peut constater que lorsque l'entropie de l'information quantifiée est plus petite, la distorsion après quantification devrait être plus grande. Cependant, plus l'entropie de l'information est grande, plus la quantité d'informations est importante, ce qui signifie plus la complexité du traitement. Par conséquent, on doit équilibrer le choix entre distorsion et entropie.

2.2 Quantificateur de type *mid-tread*

Manipulation: 6 Construire une fonction dont l'en-tête est `def quant_midtread(x,Delta):` réalisant une quantification de type *mid-tread* de `x` avec un pas `Delta`. La sortie de cette fonction est un tuple comprenant une liste `qx` des valeurs quantifiées de `x` et une liste `idx` des index de quantification obtenus. Les index de quantification ne sont pas bornés, ainsi `idx` peut prendre toute valeur entière positive ou négative.

Pour un quantificateur de type *mid-tread*, pour la source `x` avec un pas `Delta`, la valeur quantifiée `qx` et l'indice de quantification `idx` peuvent être exprimés par les formule suivantes :

$$\texttt{idx} = \left\lfloor \frac{x}{\texttt{Delta}} + \frac{1}{2} \right\rfloor$$
$$\texttt{qx} = \texttt{Delta} \times \texttt{idx}$$

D'après les relations ci-dessus, on peut implémenter cette fonction.

Listing 13 – Fonction - quant_midtread()

```
1 import numpy as np
2
3 def quant_midtread(x, Delta = 1):
4     """ Fonction pour la quantification de type Mid-tread """
5     idx = np.floor((x + Delta / 2.0 ) / Delta)
6     qx = idx * Delta
7
8     return idx, qx
```

La sortie de ce quantificateur peut être tracée en appelant la fonction précédente et en utilisant le code précédent. Le résultat est présenté dans la figure.

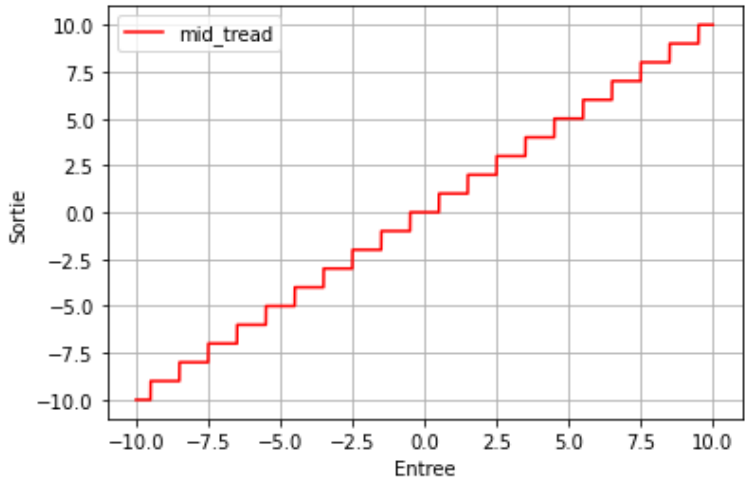

FIGURE 2.2.1 – Trace de Mid_tread.

Manipulation: 7 Tracer la distorsion en fonction de l'entropie des index de quantification obtenus à la sortie du quantificateur de type *mid-tread*. Commenter la courbe obtenue.

La courbe de distorsion en fonction de l'entropie est présentée ci-dessous.

En observant cette courbe, on peut constater que lorsque l'entropie de l'information quantifiée est plus petite, la distorsion après quantification devrait être plus grande. Cependant, plus l'entropie de l'information est grande, plus la quantité d'informations est importante, ce qui signifie plus la complexité du traitement. Par conséquent, on doit équilibrer le choix entre distorsion et entropie.

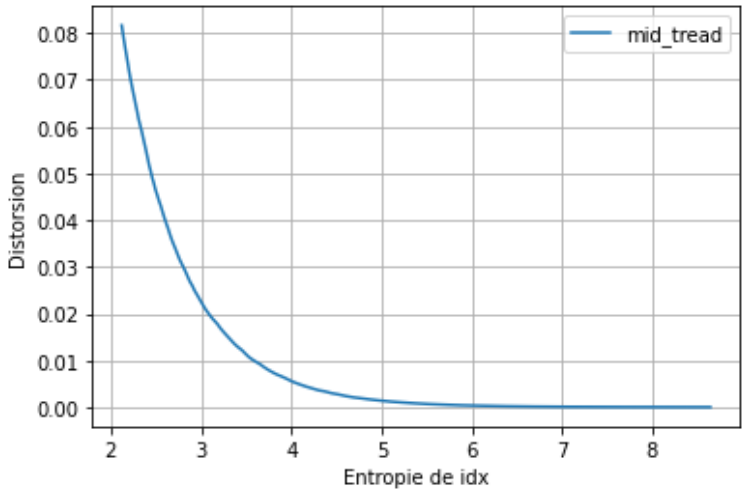

FIGURE 2.2.2 – Trace de distorsion sur entropie de Mid_tread.

2.3 Quantificateurs

Dans ce qui suit, nous allons considérer une contrainte sur le nombre de niveaux de sortie de chaque quantificateur. On considère que les index de quantification doivent être représentés sur R bits, auquel cas seulement $M = 2^R$ index différents peuvent être représentés.

Manipulation: 8 Adapter les fonctions précédentes en considérant les en-têtes suivants :

```
def quant_midrise(x,Delta,M = np.inf):
```

et

```
def quant_midtread(x,Delta,M = np.inf):
```

afin d'implanter des fonctions de quantification *mid-rise* ou *mid-tread* de `x` avec un pas `Delta`, lorsque seulement M index de sortie différents peuvent être représentés. Par défaut, M est infini et on obtient alors l'implantation initiale des quantificateurs.

En utilisant la fonction `np.clip()`, on peut limiter toutes les valeurs d'un `array` à une certaine plage. Par conséquent, il suffit d'appeler la fonction `np.clip()` sur `idx` dans la fonction et de définir la plage sur $-M/2 \sim M/2 - 1$ pour réaliser cette fonction. Le code d'implémentation est le suivant :

Listing 14 – Fonction - Adaptation

```
1  def quant_midrise(x, Delta = 1, M = np.inf):
2      """ Fonction pour la quantification de type Mid-rise """
3      idx = np.floor(x / Delta)
4
5      idx = np.clip(idx, -M/2, M/2 - 1)    # Limiter les index
6
7      qx = idx * Delta + Delta / 2.0
8
9      return idx, qx
10
11 def quant_midtread(x, Delta = 1, M = np.inf):
12     """ Fonction pour la quantification de type Mid-tread """
13     idx = np.floor((x + Delta / 2.0 ) / Delta)
14
15     idx = np.clip(idx, -M/2, M/2 - 1)    # Limiter les index
16
17     qx = idx * Delta
18
19     return idx, qx
```

Dans les manipulations suivantes, on prend comme exemple le quantificateur de type *mid_rise*.

Manipulation: 9 Pour les valeurs de $M = 2$, $M = 4$, $M = 8$, et $M = 16$, tracer la distorsion obtenue par la quantification de `x` avec différentes valeurs de `Delta`. Montrer que pour chaque valeur de M, il existe une valeur de `Delta` minimisant la distorsion. Interpréter ce résultat.

On trace les courbes correspondant aux différentes valeurs de M dans une même figure pour faciliter l'observation. A partir de la figure, on peut Montrer que pour chaque valeur de M, il existe une valeur de `Delta` minimisant la distorsion.

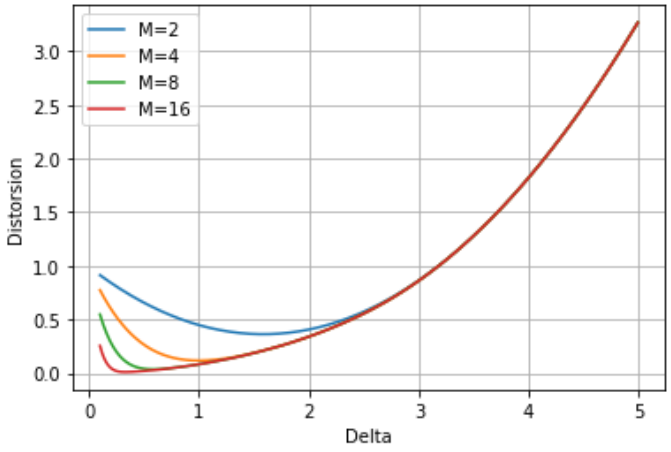

FIGURE 2.3.1 – Trace de distorsion pour différentes M.

En même temps, on montre également la valeur de `Delta` qui minimise la distorsion sous différentes valeurs de M. On peut voir que lorsque M est grand, la valeur minimale de distorsion pouvant être obtenue sera plus petite.

Listing 15 – Output de Console - Distorsion avec M

```
Pour M=2, quand Delta = 1.58, la distorsion est minimale: 0.36
Pour M=4, quand Delta = 0.99, la distorsion est minimale: 0.12
Pour M=8, quand Delta = 0.58, la distorsion est minimale: 0.04
Pour M=16, quand Delta = 0.33, la distorsion est minimale: 0.01
```

Manipulation: 10 Pour chaque valeur de M, tracer la distorsion en fonction de l'entropie des index de quantification paramétrée en `Delta`.

On trace la distorsion en fonction de l'entropie de `idx` pour différentes valeurs de M. La courbe est représentée sur la figure.

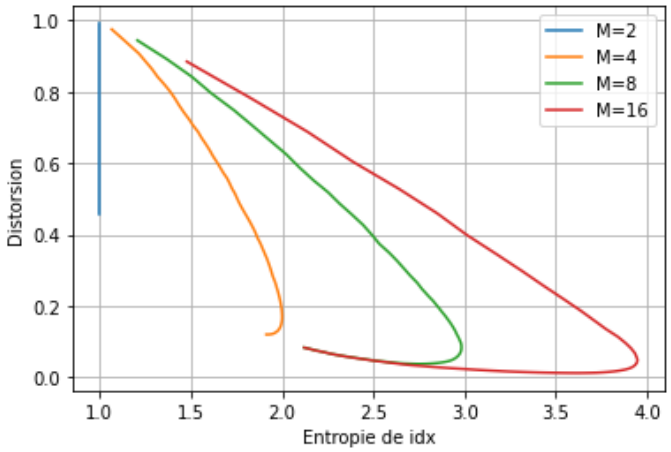

FIGURE 2.3.2 – Trace de distorsion sur entropie pour différentes M.

En prenant $M = 8$ comme exemple, $M = 8 = 2^3$, c'est-à-dire $R = 3$. Par conséquent, lorsque l'entropie s'approche de la valeur critique R, la direction de la courbe change.

Manipulation: 11 Quelles sont les valeurs de `Delta` optimales lorsque la variance de la source générant `x` est 2 ou 4 ?

On compare la valeur `Delta` optimale et la valeur de distorsion minimale lorsque la variance de la source est respectivement de 1, 2 et 4.

Listing 16 – Output de Console - Distorsion avec M Variance

```

 La variance = 2
Pour M=2, quand Delta = 3.21, la distorsion est minimale: 1.48
Pour M=4, quand Delta = 2.00, la distorsion est minimale: 0.48
Pour M=8, quand Delta = 1.17, la distorsion est minimale: 0.15
Pour M=16, quand Delta = 0.68, la distorsion est minimale: 0.05

 La variance = 4
Pour M=2, quand Delta = 4.99, la distorsion est minimale: 6.24
Pour M=4, quand Delta = 3.95, la distorsion est minimale: 1.89
Pour M=8, quand Delta = 2.36, la distorsion est minimale: 0.59
Pour M=16, quand Delta = 1.32, la distorsion est minimale: 0.18
```

À partir des résultats de sortie, on peut voir que la valeur du `Delta` optimal est proportionnelle à la valeur de la variance source et que la valeur de distorsion minimale obtenue est une relation quadratique.

Manipulation: 12 Répéter les expériences précédentes avec une source Laplacienne de moyenne nulle et de variance 1 puis 2.

Pour une source laplacienne, ses variables sont conformes à la fonction de distribution de probabilité suivante :

$$f(x|\mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right)$$

On peut utiliser les instructions suivantes pour obtenir une source laplacienne `x` avec une moyenne de 0, une variance de 1 ou 2 et un nombre de variables de 10 000.

Listing 17 – Tableau x Laplacien

```
# Generer un tableau de la variable Laplacienne
N = 10000
x4 = np.random.laplace(loc=0, scale=1, size=N)
x5 = np.random.laplace(loc=0, scale=2, size=N)
```

Pour la source laplacienne, les courbes de sa distorsion par rapport à différentes valeurs `Delta` sont représentées sur la figure.

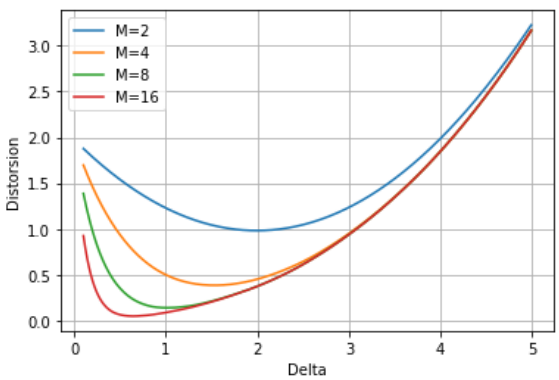

FIGURE 2.3.3 – Distorsion pour variance = 1.

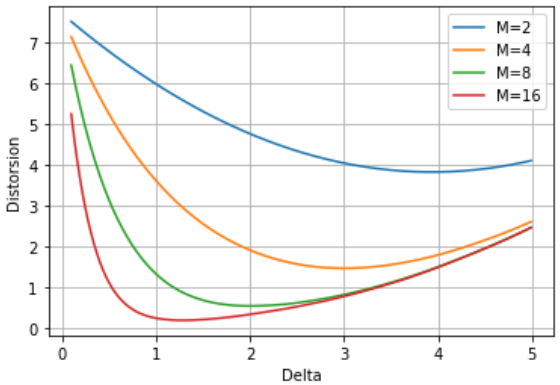

FIGURE 2.3.4 – Distorsion pour variance = 2.

2.4 Application des signaux audio

Nous allons maintenant considérer des signaux audio.

Manipulation: 13 Ouvrir le fichier `Alarm05.wav`.

On peut ouvrier le fichier à l'aide de code suivant :

Listing 18 – Lecture du signal

```
1 import scipy.io
2 from scipy.io import wavfile
3
4 filename = 'Alarm05.wav'
5 samplerate, data = wavfile.read(filename)
```

Manipulation: 14 Le fichier `Alarm05.wav` contient-il un signal mono ou stéréo ? Quelle est la fréquence d'échantillonnage ? Sur combien de bit les échantillons de `Alarm05.wav` sont-ils représentés ?

En examinant la dimension et le type `data`, on peut savoir si il est stéréo et combien de bits d'échantillonnage il contient. On peut connaître la valeur de la fréquence d'échantillonnage en regardant la valeur de `samplerate`.

Listing 19 – Output de Console - Audio

```
1  Le signal audio
2 La dimmension du signal : 2
3 Sample Rate est : 22050
4 Bits des echatillons est : int16
```

Par conséquent, on peut constater que le signal est un signal stéréo, avec une fréquence d'échantillonnage de 22050 Hz et un bit d'échantillonnage de 16 bits. Pour un signal stéréo, il comprend principalement un voix gauche et un voix droit. La durée de l'audio peut être exprimée comme `temps = len(data)/samplerate`.

Manipulation: 15 Représenter chacune des voix du signal `Alarm05.wav`. Attention à la graduation temporelle de l'axe des abscisses.

On représente les deux sons comme indiqué sur la figure.

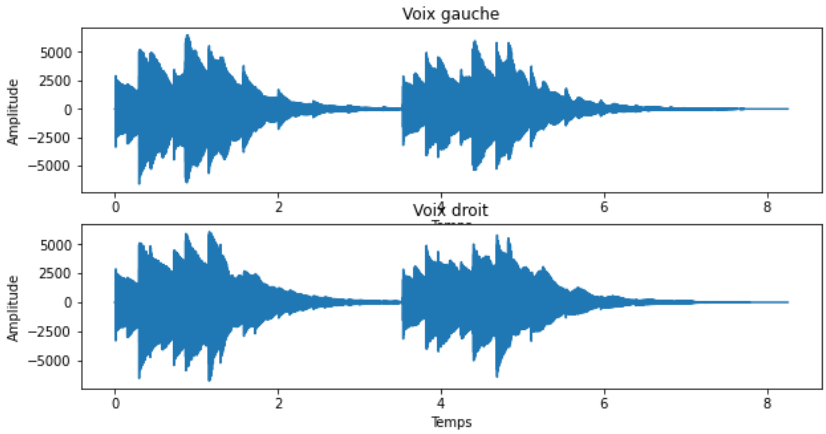

FIGURE 2.4.1 – Signal audio stéréo.

Manipulation: 16 Réaliser la quantification du tableau data pour différentes valeurs de $M = 2$, $M = 4$, $M = 8$, et $M = 16$. Tracer la distorsion en fonction de `Delta`. Montrer que pour chaque valeur de M, il existe une valeur de `Delta` minimisant la distorsion. Ecouter le signal audio quantifié pour cette valeur de `Delta`.

On prend un voix gauche dans le signal audio pour cette manipulation.

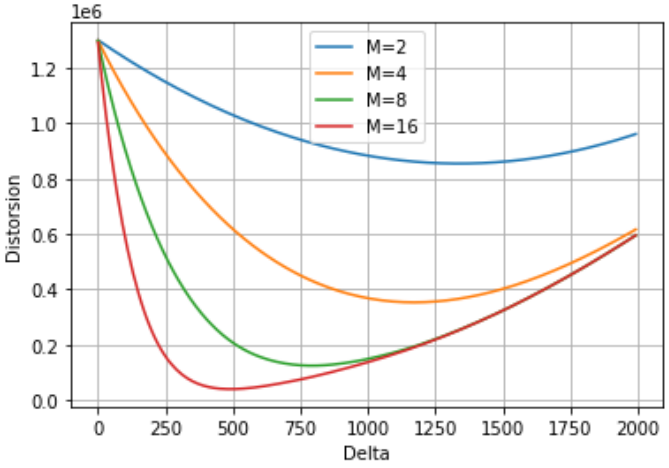

FIGURE 2.4.2 – Trace de distorsion d'audio.

D'après la courbe tracée, on peut montrer que pour chaque valeur de M, il existe une valeur de `Delta` minimisant la distorsion.

3 Conclusion

Pendant ce TP, on a fait
— estimé l'entropie d'une source, par exemple d'un texte, d'une image, ou d'un fichier audio.
— consacré à l'étude de deux type de quantificateurs scalaires.
— évalueé le compromis en débit et distorsion lors d'une quantification scalaire.
— appliqué aux signaux audio.

[1]

Références

[1] M. Kieffer. *UE 455 – Théorie de l'Information - Codage de Source.* Université Paris-Saclay, 2023-2024.