Université Paris-Saclay
ENS Paris-Saclay
M1 E3A

TP 1
2023 – 2024

—

Théorie de l'Information - Codage de Source

—

TP 1 – Entropie – Quantification

Objectifs du TP

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`. Ils pourront également être réalisés à l'aide de `Matlab`.

1 Première partie - 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. Comment évaluer la fréquence d'apparition de chaque symbole de `x` ?
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`.
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`.
4. Charger le fichier `Declaration1789.txt` et donner l'alphabet de cette source. Vous pourrez vous servir de

```
with open("Declaration1789.txt", "r") as file:
    texte = file.read()
```

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

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

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

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.
10. Comparer et commenter les résultats obtenus.
11. Reprenez les résultats précédents avec le fichier `Alarm05.wav`. Commenter.

2 Seconde partie - 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.

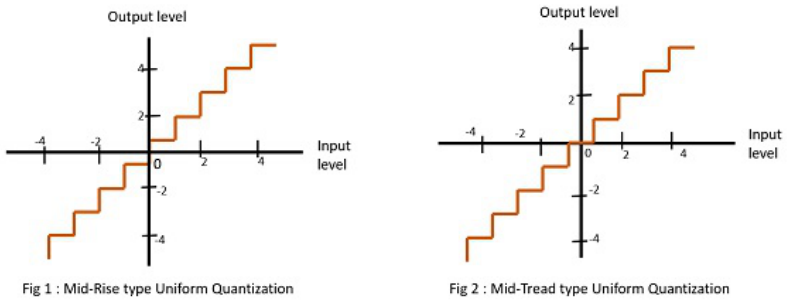

FIGURE 1 – Caractéristiques entrée-sortie des quantificateurs de type *mid-rise* et *mid-tread*

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 caractéristique entrée-sortie de ce quantificateur est donnée sur la figure 1 (a). 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. Vous pourrez utiliser la fonction `floor` de la bibliothèque `numpy`.
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`. Pour le tracé, vous pourrez vous inspirer du code suivant

```
import matplotlib.pyplot as plt
plt.plot(x,qx,'-r')
plt.legend(['mid_rise'])
plt.xlabel('Entrée')
plt.ylabel('Sortie')
plt.grid()
plt.show()
```

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`.
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`.

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.
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 caractéristique entrée-sortie de ce quantificateur est donnée sur la figure 1 (b). 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.
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.

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.

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.

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.
10. Pour chaque valeur de M, tracer la distorsion en fonction de l'entropie des index de quantification paramétrée en `Delta`.
11. Quelles sont les valeurs de `Delta` optimales lorsque la variance de la source générant `x` est 2 ou 4 ?
12. Répéter les expériences précédentes avec une source Laplacienne de moyenne nulle et de variance 1 puis 2.

Nous allons maintenant considérer des signaux audio.

13. Ouvrir le fichier `Alarm05.wav` par exemple à l'aide de

```
from scipy.io import wavfile
import scipy.io
filename = 'Alarm05.wav'
samplerate, data = wavfile.read(filename)
```

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 ?
15. Représenter chacune des voix du signal `Alarm05.wav`. Attention à la graduation temporelle de l'axe des abscisses.
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`. Pour cela, enregistrer les données quantifiées aux format `wav` en vous inspirant du code ci-dessous

```
from scipy.io.wavfile import write
import numpy as np
samplerate = 44100; fs = 100
t = np.linspace(0., 1., samplerate)
amplitude = np.iinfo(np.int16).max
data = amplitude * np.sin(2. * np.pi * fs * t)
write("example.wav", samplerate, data.astype(np.int16))
```

Il est ensuite possible d'écouter le contenu du fichier en utilisant la bibliothèque `simpleaudio`

```
import simpleaudio as sa
import wave
wave_read = wave.open(path_to_file, 'rb')
wave_obj = sa.WaveObject.from_wave_read(wave_read)
play_obj = wave_obj.play()
play_obj.wait_done()
```