TP3 : Quantifications et Modulations

Gilles Menez - UCA - UFR Sciences - Dépt. Informatique

---

1 Quantification

Jusqu'à présent vous avez échantillonné des signaux en les discrétisant en temps.

La quantification aborde **la discrétisation de l'amplitude** des signaux.

- ➢ C'est un procédé qui permet d'approcher un signal continu par les valeurs d'un ensemble discret de plus petite taille.
- ➢ Elle intervient nécessairement lors de la conversion analogique-numérique d'un signal réel puisque la puissance du continu ne peut pas être représentée par l'ordinateur.

Dans le cas présent, les échantillons étant déjà numériques c'est l'aspect "compression" que nous allons explorer. C'est un point crucial dans le contrôle du débit de la télécommunication, mais ça pourrait l'être tout autant si on s'intéressait au stockage de l'information (CDROM Audio, ...).

1.1 Quantificateur et Dictionnaire

Un quantificateur scalaire de taille n est une application Q, de $E = \mathbb{R}$ (dans le contexte d'un signal continu) dans un ensemble discret fini F, de dimension 1 et de taille n,

$$Q : E \to F = \{\hat{x_1} \dots \hat{x_n}\}$$

On note $\hat{x} = Q(x)$ la valeur quantifiée de x.

- ➢ L'ensemble F est généralement appelé **dictionnaire**.

Le but de la quantification est, à partir d'une valeur d'entrée donnée d'un espace E, de **déterminer la valeur la plus proche** dans l'ensemble F d'arrivée.

- ➢ Ainsi, on minimisera l'erreur commise par la quantification !

Pour l'instant,

- ✔ nous avons profité des capacités de représentation des nombres flottants de Python.

  Il s'agit d'une quantification (puisque le nombre de flottants n'est pas infini) mais cet effet de quantification induit par les inévitables arrondis est peu visible.

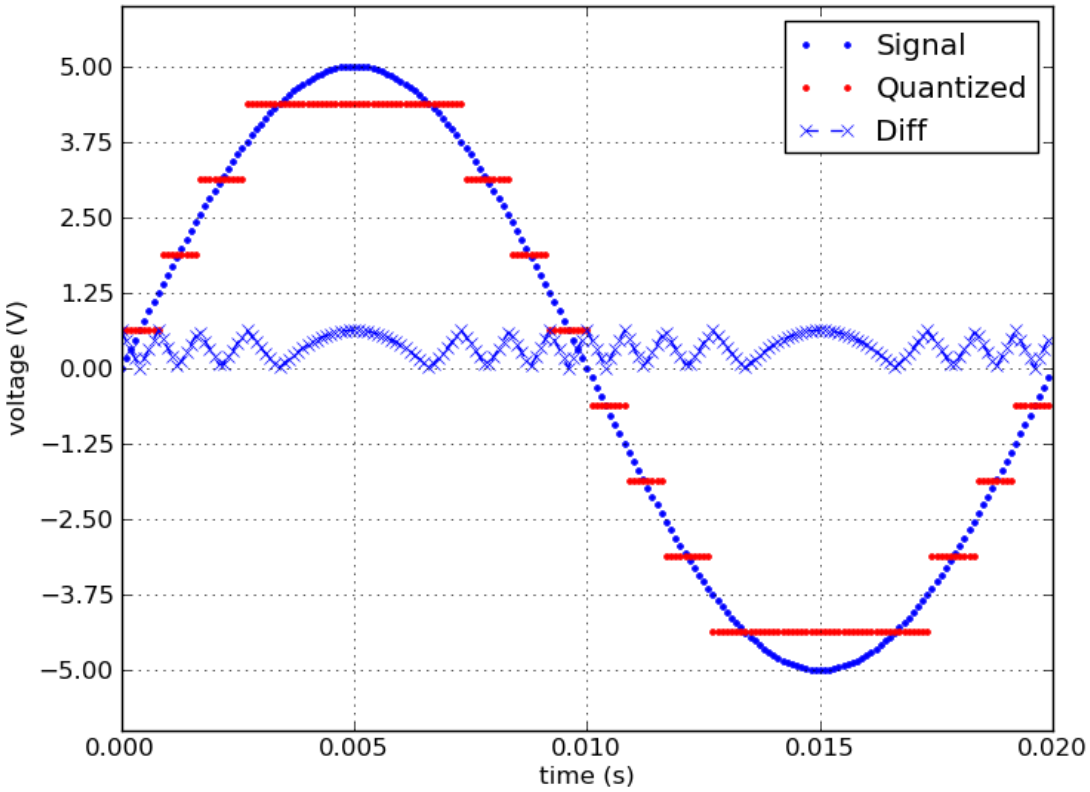
- ✔ nous aurions pu quantifié sur l'ensemble des entiers.

  Mais disposer de 32, 64 et même plus de bits pour représenter un échantillon est un luxe que les télécommunications ne peuvent pas encore offrir.

1.2 Quantificateur uniforme

Dans cet exercice, vous devez réaliser un quantificateur uniforme sur 3 bits d'un signal analogique (simulé par un de ceux que vous venez de coder) dont l'amplitude maximale est 5 Volt.

Cela devrait donner quelque chose comme ça :



Avant de réaliser le code Python qui produira cette courbe, il faut comprendre :

① Ce que représentent les courbes et notamment celle constituée de "palliers" rouge ?

② D'où vient ce step/pas constant entre les "palliers" rouges ?

③ Pourquoi y en a t'il 8 ?

Le squelette et l'enrobage graphique qui permettra d'obtenir ce graphe pourrait être :

```
#-*- coding: utf-8 *-*
'''
File tp1_q_vide.py  : Created on 28 feb 2012
@author : menez
Illustration de la quantification
'''
import math
import sys
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator
```

```
sys.path.append('/home/menez/EnseignementsCurrent/Cours_Telecom/TPs/Src')
import Signaux.tp1_0 as tp1

#-------------------------------------------
def make_dico(vmax, b):
    """
    Crée un dictionnaire uniforme de quantification sur "b" bits
    pour des signaux d'amplitude max "vmax"
    """
    pass

#-------------------------------------------
def QS(sig_s, vmax, b):
    """
    Quantificateur sur b bits d'un signal sig_s d'amplitude max vmax.
    Rend le signal quantifie et le bruit sur chaque echantillon.
    """

    # Creation du dictionnaire de quantification = une liste de ""codewords"
    D = make_dico(vmax, b)

    # Pour l'instant le signal quantifie est le meme que le signal original.
    Q = sig_s

    # Donc le signal de bruit est donc 0 puisqu'il n'y a pas de quantification.
    E = [0]*len(sig_s)

    return Q, E

#-------------------------------------------
def plot(inx, iny, leg, fmt='-bo', l=""):
    plt.plot(inx,iny,fmt,label=l)
    plt.xlabel('time (s)')
    plt.ylabel('voltage (V)')
    plt.ylim([-5.5, +5.5])

#===========================================
if __name__ == '__main__':
    np.set_printoptions(linewidth=250)
    np.set_printoptions(precision=3, suppress=True)
    a=5.0
    b=3

    # Signal à quantifier
    fe = 2000.0
    f = 50.0
    d = 0.04
    x,y=tp1.make_sin(a,0,f=f,fe=fe,d=d)

    # Signal quantifié et erreur
    z,err = QS(y,a,b)

    # Plot des signaux :
    fig = plt.figure(figsize=(12,12))
    ax = fig.add_subplot(1,1,1)
    step = 2*a/(2**b)
```

```
majorLocator = MultipleLocator(step) # Choisir la graduation en y
ax.yaxis.set_major_locator(majorLocator)
plot(x,y,"",'bo', l="Signal")
plot(x,z,"",'rs', l="Quantized")
plot(x,err,"",'--x', l="Diff")

title = "Sinusoide : $f_e={}, f={}, d={}$".format(fe,f,d)
tp1.plot_fig_decoration(title)

plt.show()
```

L'approche qui vous est proposée passe par la définition d'un dictionnaire explicite. C'est loin d'être optimal car une solution arithmétique pourrait certainement éviter d'avoir à construire ce dictionnaire. Il serait alors directement intégré dans le calcul.

Néanmoins, la présence d'un dictionnaire nécessite une recherche du plus proche "codeword" et ceci pourrait être fort utile si on passait à une quantification vectorielle ... mais c'est une autre histoire ;-)

(a) Votre solution doit opérer avec 3 bits **mais aussi 4, 5, ... bits** !

(b) Vous essayez ausis avec d'autres signaux que celui de la sinusoïde.

1.3 Bruit de quantification

Vous avez remarqué qu'une des courbes correspond au bruit de quantification.

➢ Soit pour chaque échantillon, la valeur absolue de la différence d'amplitude entre la valeur vraie et la valeur quantifiée correspondante.

De plus, dans ce contexte, en utilisant les formules vues en cours et que je rappelle ici :

① Calculer l'erreur carrée moyenne (MSE : Mean Square Error)

$$MSE = \sigma_q^2 = \frac{1}{N}\sum_i (x(i) - x_q(i))^2 \tag{1}$$

② Calculer le SNR (en linéaire et en dB) :

$$SNR(db) = 10 \times log_{10}(\sigma_x^2/\sigma_q^2) \tag{2}$$

avec N le nombre d'échantillons formant le signal, et σ_x^2 est la variance du signal original,

$$\sigma_x^2 = \frac{1}{N}\sum_i (x(i))^2 \tag{3}$$

③ En prenant soin d'utiliser une fréquence d'échantillonnage de l'ordre de $10000Hz$, augmenter le nombre de bits du quantificateur et montrer l'évolution du SNR.

1.4 Encodage sur un signal bivalent

A ce stade, chaque échantillon du signal a été projeté sur une valeur du dictionnaire.
Cette valeur est représentable sur b bits qu'il s'agit maintenant de sérialiser de façon à former une séquence de 1 et de 0.

➢ N'oublions pas non plus que ces valeurs sont signées !

Proposer une fonction d'encodage qui utilisera un code (”bit de signe” + valeur binaire) et renverra la chaîne de caractères (0 et 1 . . . d'où la notion de bivalence) correspondant au signal que l'on vient de quantifier :

➢ donc par exemple la valeur 7 sur 5 bits sera représentée par

00111

➢ et -7 toujours sur 5 bits par

10111

En Python, vous pourriez avoir besoin de ce qui suit pour fabriquer la représentation binaire d'un entier `v` sur `b` bits :

```
>>> b=5
>>> v=7
>>> print("{:0{}b}".format(v,b))
```

Vous obtenez finalement une séquence de 0 et de 1. Nous avons ici choisi d'en faire une liste de caractères.

```
menez ~/EnseignementsCurrent/Cours_Telecom/TPs/Src/Quantizer$ : python3 tp1_q.py
Step : 0.75
MSE : 0.052583457835335856
SNR : 85.57824428533529 en dB : 19.32363372297338
MSE : 0.052583457835335856
SNR : 85.57824428533529 en dB : 19.32363372297338
Step : 0.75
[ 0.     0.469  0.927  1.362  1.763  2.121  2.427  2.673  2.853  2.963  3.     2.963  2.853  2.673  2.427
2.121  1.763  1.362  0.927  0.469  0.    -0.469 -0.927 -1.362 -1.763 -2.121 -2.427 -2.673 -2.853 -2.963 -3.
    -2.963 -2.853 -2.673 -2.427
 -2.121 -1.763 -1.362 -0.927 -0.469]
[0.375, 0.375, 1.125, 1.125, 1.875, 1.875, 2.625, 2.625, 2.625, 2.625, 2.625, 2.625, 2.625, 2.625, 2.625, 1
.875, 1.875, 1.125, 1.125, 0.375, 0.375, -0.375, -1.125, -1.125, -1.875, -1.875, -2.625, -2.625, -2.625, -2
.625, -2.625, -2.625, -2.625, -2.625, -2.625, -1.875, -1.875, -1.125, -1.125, -0.375]
[0.0, 0.0, 1.0, 1.0, 2.0, 2.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 3.0, 2.0, 2.0, 1.0, 1.0, 0.0, 0.0, -
1.0, -2.0, -2.0, -3.0, -3.0, -4.0, -4.0, -4.0, -4.0, -4.0, -4.0, -4.0, -4.0, -4.0, -3.0, -3.0, -2.0, -2.0,
-1.0]
0000 0000 0001 0001 0010 0010 0011 0011 0011 0011 0011 0011 0011 0011 0011 0010 0010 0001 0001 0000 0000 10
01 1010 1010 1011 1011 1100 1100 1100 1100 1100 1100 1100 1100 1100 1100 1011 1011 1010 1010 1001
['0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '0' '1' '0' '0' '1' '0' '0' '0' '1' '0' '0' '0' '1
' '1' '0' '0' '1' '1' '0' '0' '1' '1' '0' '0' '1' '1' '0' '0' '1' '1' '0' '0' '1' '1' '0' '0' '1' '1' '0' '
0' '1' '1' '0' '0' '1' '1' '0' '0'
 '1' '0' '0' '0' '1' '0' '0' '0' '0' '1' '0' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '1' '1
' '0' '1' '0' '1' '0' '1' '0' '1' '0' '1' '1' '1' '0' '1' '1' '1' '1' '0' '0' '1' '1' '0' '0' '1' '1' '0' '
0' '1' '1' '0' '0' '1' '1' '0' '0'
 '1' '1' '0' '0' '1' '1' '0' '0' '1' '1' '0' '0' '1' '1' '0' '0' '1' '0' '1' '1' '1' '0' '1' '1' '1' '0' '1
' '0' '1' '0' '1' '0' '1' '0' '0' '1']
```

En reprenant les caractéristiques du quantificateur des questions précédentes, la console montre que chaque échantillon est mis en correspondance

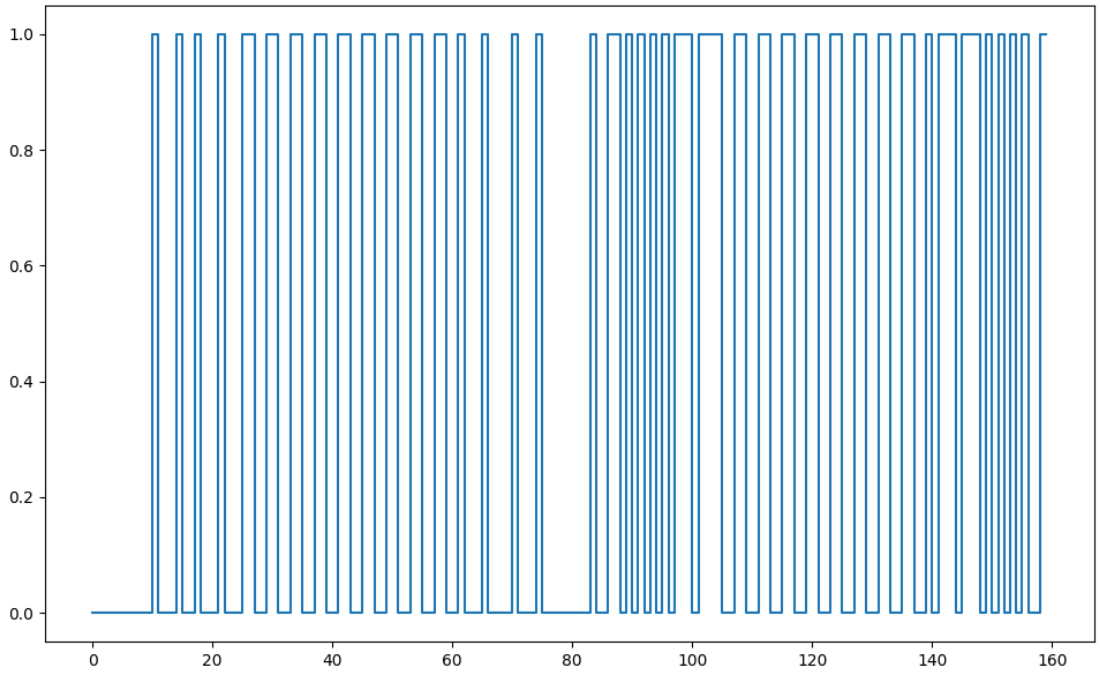
- avec une valeur quantifiée (donc un multiple du pas/step),
- l'index qui lui correspond ($[-4, +3]$ soit $8 = 2^3$ niveaux)
- et l'encodage binaire sur $1 + 3 = 4$ bits de cet index.
  (1 bit pour le signe et 3 bits pour l'amplitude)

Pour finir, on associe une durée de symbole (cf modulation) à chaque '1 ou '0'.
On peut simuler cela avec la fonction `step` de Matplotlib, en choisissant des abscisses entières.

Essayer ça, vous allez vite comprendre :

```
menez ~/EnseignementsCurrent/Cours_Telecom/TPs/TP2$ : python3
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3]
>>> y = [0,1,0]
>>> plt.step(x,y)
[<matplotlib.lines.Line2D object at 0x7f1dc27305f8>]
>>> plt.show()
```

TADAAAA : Ce signal est "prêt" à être transmis . . . comme une séquence de 0 et de 1.



Pour finir correctement, il faudrait opter pour une modulation (Manchester, MLT-3, . . .) . . .