TP1 : Signaux Numériques

Gilles Menez - UNSA - UFR Sciences - Dépt. Informatique

Les objectifs de ce premier TP sont :

① Vous placer dans un environnement de programmation (Python + Eclipse (PyDev)),
`PyDev` est le plugin Python sous Eclipse. Cet environnement servira durant les TPs et peut être après? Ceci étant, vous pouvez choisir de travailler dans un autre environnement !

② Faire un programme Python,
Si vous ne connaissez pas Python ? ... aucun soucis ... on va aller doucement !
Si vous connaissez Python, alors montrez-le et soignez vos programmes !
L'idée est d'**utiliser** Python comme une calculatrice dotée d'un écran graphique et donc capable d'afficher des courbes.

③ Appréhender les signaux numériques qui constituent les supports des informations.

---

1 Installer et s'installer dans l'environnement ...

➟ **A la maison**, si vous travaillez (au moins) sous Ubuntu, alors à grands coups de gestionnaire de paquets vous installez Eclipse, Python, Matplotlib ... en 15 minutes et tout (notamment la liaison entre exécutables et lib) est "quasi automatique" !

➟ **Sous Windows ou MacOsX** c'est aussi rapide.

- ➢ Eclipse
- ➢ et `http://www.enthought.com/products/epd.php`
- ➢ ou `http://continuum.io/downloads`.

➟ **Dans les salles CRIPS**
Normalement, `eclipse` est déjà installé ainsi que quelques packages indispensables à vos TPs :

```
apt-get install python-matplotlib
apt-get install python-numpy
apt-get install python-scipy
```

... merci aux "roots" !

Mais il manque peut être le plugin PyDev sous Eclipse que vous devez installer.

➠ Donc, une fois la fenêtre d'eclipse ouverte :

Help− >Install New Software ...
puis téléchargement de l'environnement PyDev via le site : `http://pydev.org/updates`

1.1 Prise en main de Python sous Eclipse

On va faire des programmes et même si on commence par faire simple, on ne perd de vue l'objectif de donner une organisation à nos réalisations.

- ➢ Donc lorsqu'on écrit du code, on le fait proprement : on définit des constantes, on factorise le code et on évite les ré-écritures, on essaye de faire de l'objet, ...
- ➢ d'autant que votre note de **contrôle continu** est basée sur cela !

1.1.1 Créer la configuration et expérimenter

Grâce à PyDev, dans le projet (`TPTelecom`),

① Créer un package : `Signaux`

② Créer un module : `tp1_0.py`

Je sais bien que vous êtes des "fans" du copier-coller. Mais prenez le temps de taper ce code et de comprendre ce qu'il signifie.

De plus, le copier-coller introduit des codes de caractères qui ne sont pas corrects pour l'interpréteur Python ... **vous allez perdre du temps**!

```
#!usr/bin/python
#-*- coding : utf-8 *-*
'''
File : tp1_0.py
Created on 28 fevr. 2012
@author: menez
Creer un signal numerique et l'afficher
On utilise ici le B,A,BA de Python
    => Donc ca doit etre compris !!!!
'''
import math
import matplotlib.pyplot as plt
# Peut etre rajouter /usr/lib/python2.7 dans le PYTHONPATH/External Libraries du projet ?
import numpy as np

def make_sin(a=1.0, ph=0, f=440.0, fe=8000.0, nT=1):
    """
    Create a synthetic 'sine wave'
    First version : use classic Python lists
    """
    omega = 2*math.pi*f
    N = int(fe/f)
    te = 1.0/fe
    sig_t = []
    sig_s = []
    for i in range(N*nT):
        t = te*i
        sig_t.append(t)
        sig_s.append(a*math.sin((omega*t)+ph))

    return sig_t, sig_s


def plot(inx, iny, leg, format='-bo'):
    plt.plot(inx,iny,format)
    plt.xlabel('time (s)')
    plt.ylabel('voltage (V)')
    plt.title(leg)
    #plt.ylim([-1.2, +1.2])
    plt.grid(True)


if __name__ == '__main__':
    x,y=make_sin(2,f=50.0,fe=1000.0,nT=2)
    plot(x,y,"Une sinusoide ...")

    plt.show()
```

③ Au niveau des commentaires que vous pourrez/devez ajouter, n'hésitez pas à préciser ce que représentent les variables : a, f, fe, nT, N ...

④ Il faut aussi comprendre les quelques manipulations sur les "tableaux" Python.

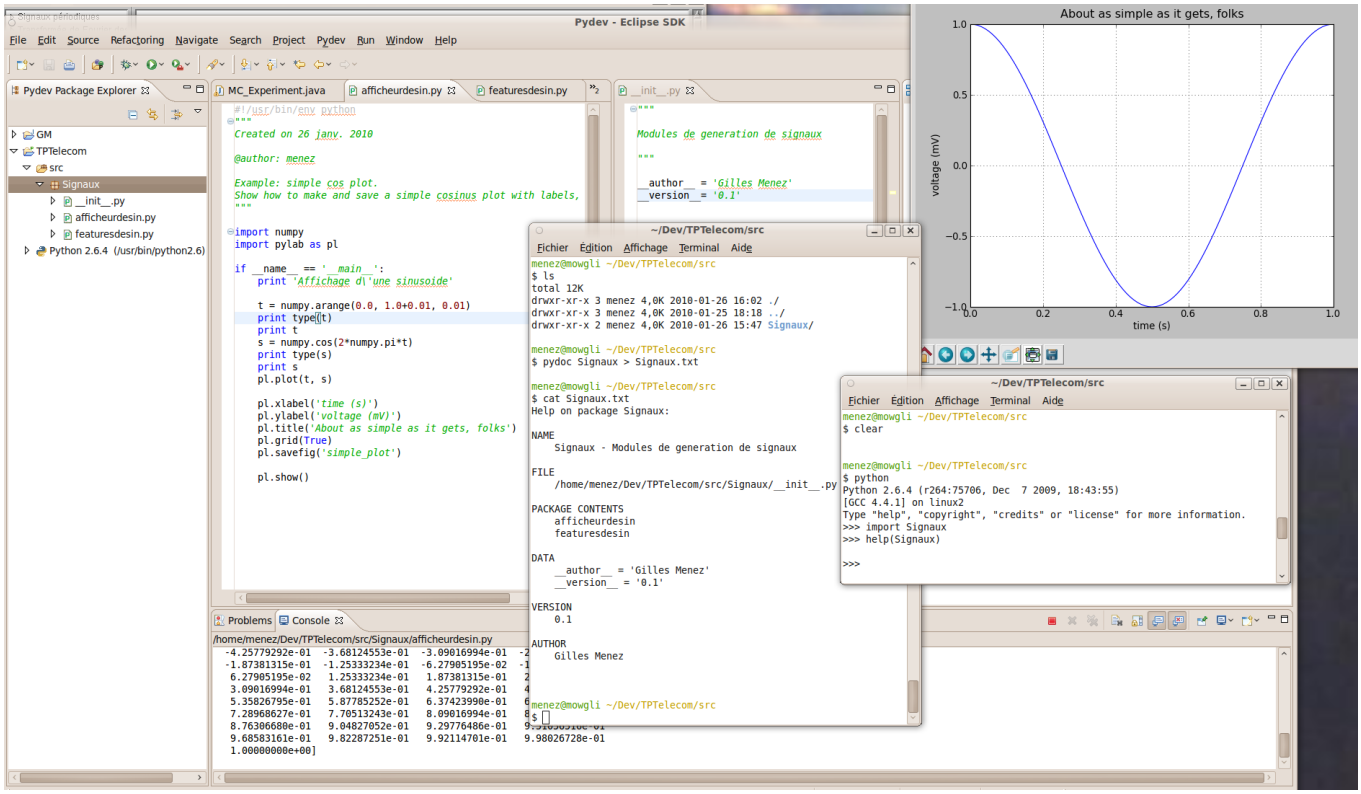
⑤ Vous notez aussi comment changer le format du tracé :

`http://matplotlib.sourceforge.net/api/pyplot_api.html#matplotlib.pyplot.plot`

⑥ Exécuter pour obtenir le tracé et faire valider.

1.1.2 Votre environnement ...

Vous devriez travailler dans un environnement qui ressemble à celui là :

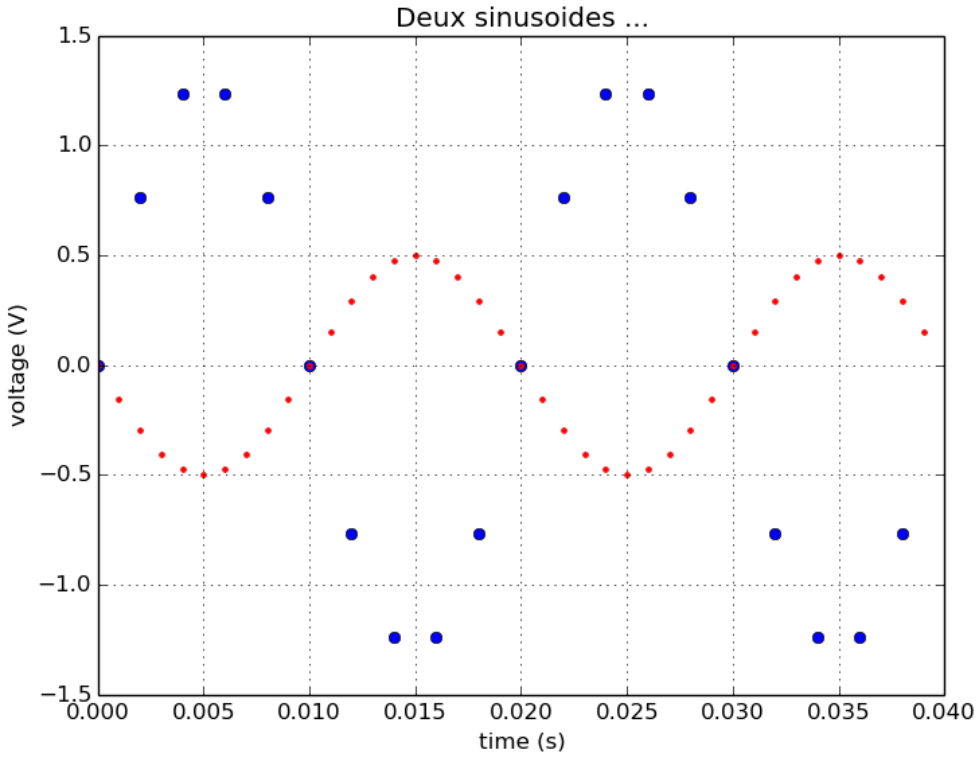


- ➢ Choisir la bonne (i.e PyDev) perspective Eclipse,
- ➢ L'affichage de la console Eclipse,
- ➢ Un terminal

Vous devez prendre le temps de comprendre **"complètement"** cet écran. On en discute si besoin !

1.1.3 Vous essayez seul . . .

Dans un nouveau module `tp1_1.py` , créer un code capable d'afficher la "même" fenêtre que celle-ci :



Il s'agit du tracé de deux sinus : s_1, s_2

(a) Quantifiez en quoi ils sont différents ?
- ➢ amplitudes (a_1, a_2) ?
- ➢ fréquences $(freq_1, freq_2)$?
- ➢ phases à l'origine $(ph1, ph2)$?

(b) La fréquence d'échantillonage est-elle la même pour les deux signaux ?

(c) Faire valider.

Petite aide :

On utilise toujours les fonctionnalités offertes par la plateforme Python numérique.
`http://matplotlib.sourceforge.net/api/index.html`

L'affichage de deux courbes sur la même fenêtre peut être fait grâce à l'appel :

```
pl.plot(t, s1 , "r:", t , s2, "b.")
```

ou

```
pl.plot(t, s1 , "r:", hold=True)
pl.plot(t, s2, "b.")
```

Mais attention !

- ➢ Lequel de ces deux formats d'affichage (" :" et ".") interpole la courbe réelle et donne **l'illusion d'une courbe continue** ?

2 Génération de signaux

Ce que l'on vient de faire dans la première partie est la **numérisation** d'un signal continu dont l'équation est :

$$s(t) = asin(\omega\ t + \phi) \tag{1}$$

avec $\omega = 2 \times \pi \times f$ la pulsation en rad/s et f la fréquence du signal $s(t)$ en H_z.

Pour l'instant nous nous sommes essentiellement intéressé à la **discrétisation en temps** puisque nous avons calculé $s(t)$ pour $t = i \times \frac{1}{f_e}$.

Par contre toutes les amplitudes sont "autorisées"(/potentiellement présentes) et on n'a donc pas **discrétisé en amplitude** !

2.1 Signal carré

Sur la base des équations en temps continu fournie par l'url suivante :

`http://fr.wikipedia.org/wiki/Signal_carré`

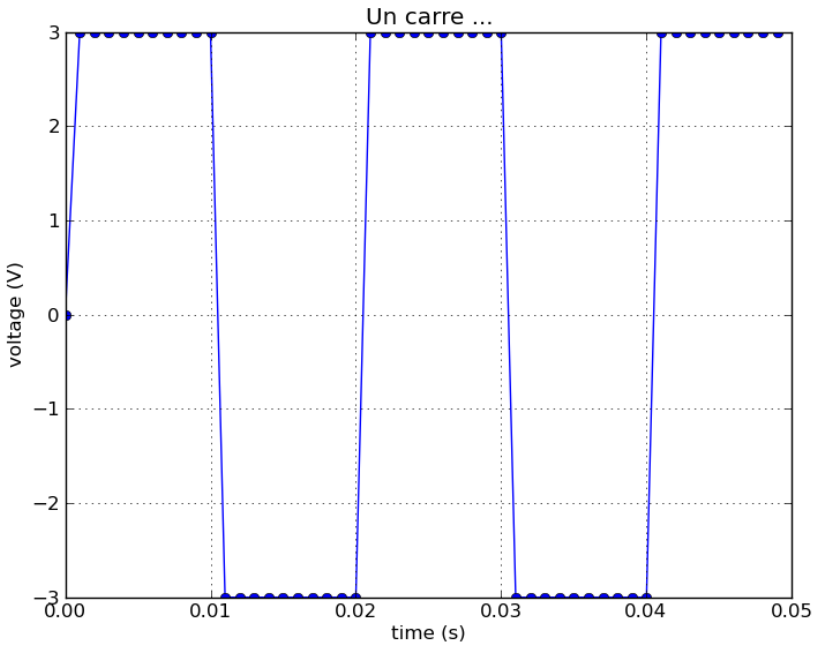
et notamment l'équation :

$$x(t) = a * sgn(sin(t)) \tag{2}$$

avec $sgn(x)$, la fonction "signe" qui est égale à 1 quand x est positif, 0 quand x est nul et -1 quand x est négatif.

(a) Dans un module `tp1_2.py` , réaliser une fonction générant un signal carré.

Vous devriez obtenir, si $a = 3, f = 50.0, fe = 1000.0, nT = 3$:



(b) Faire valider

(c) Cette première définition d'un signal carré donne des résultats peu satisfaisants lors de l'implémentation du fait du test de 0 effectué dans la fonction "sgn".

En effet, le nombre d'échantillons au niveau positif peut différer du nombre d'échantillons au niveau négatif de plus de un échantillons.

Deux raisons à cela :

① Si on commet une petite erreur d'arrondi sur le calcul de l'angle on obtient un sinus qui au lieu de valoir 0 va valoir 0.0001 ou -0.0001.

La fonction "sgn" accroit de facon disproportionnée les conséquences de cette erreur d'arrondi. Ce qui aurait du être 0 vaut finalement -1 ou $+1$.

② Si au lieu de tester 0 on s'autorise un $+/-epsilon$ le problème est alors que le $sin(0)$ devrait donner $+1$ et le $sin(\pi/2)$ devrait donner -1 pour équilibrer le signal.

La question est : "Est-ce que la méthode Heavyside fourni dans wikipedia, permet d'obtenir de meilleur résultats ?"

`http://fr.wikipedia.org/wiki/Signal_Carré`

(d) Faire valider

2.2 Signal Dent de scie

Sur la base de l'équation en temps continue : `http://en.wikipedia.org/wiki/Sawtooth_wave`

$$x(t) = 2 \times \left(\frac{t}{T} - \left\lfloor \frac{t}{T} - \frac{1}{2} \right\rfloor \right) \tag{3}$$

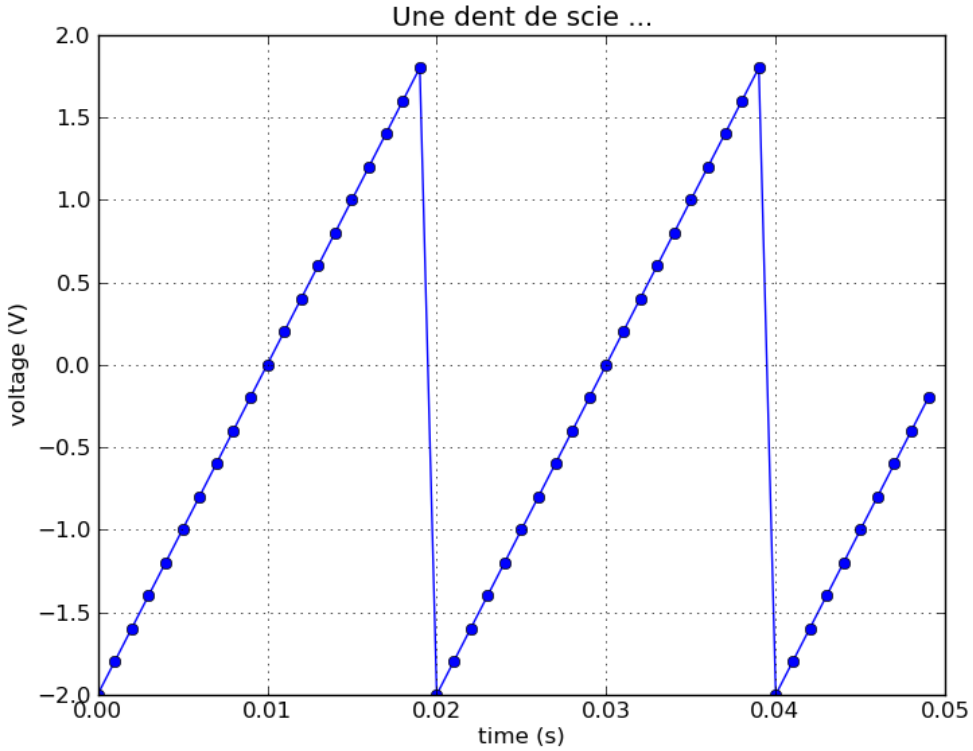
$$= 2 \times \left(\frac{t}{T} - \text{floor}\left(\frac{t}{T} - \frac{1}{2} \right) \right) \tag{4}$$

Mais à mon avis, l'équation fournie par wikipédia est fausse ... je vous propose :

$$x(t) = 2 \times a \times \left(\frac{t}{T} - \left\lfloor \frac{t}{T} \right\rfloor - \frac{1}{2} \right) \tag{5}$$

(a) Proposer une fonction permettant d'obtenir un signal "dent de scie".

Vous devriez obtenir, si $a = 2, f = 50.0, fe = 1000.0, nT = 2$:



Attention !
Cette figure est fausse car l'amplitude du signal n'est pas 2.0 (en 0.02 sec) ... pourquoi ? peut-on faire mieux !

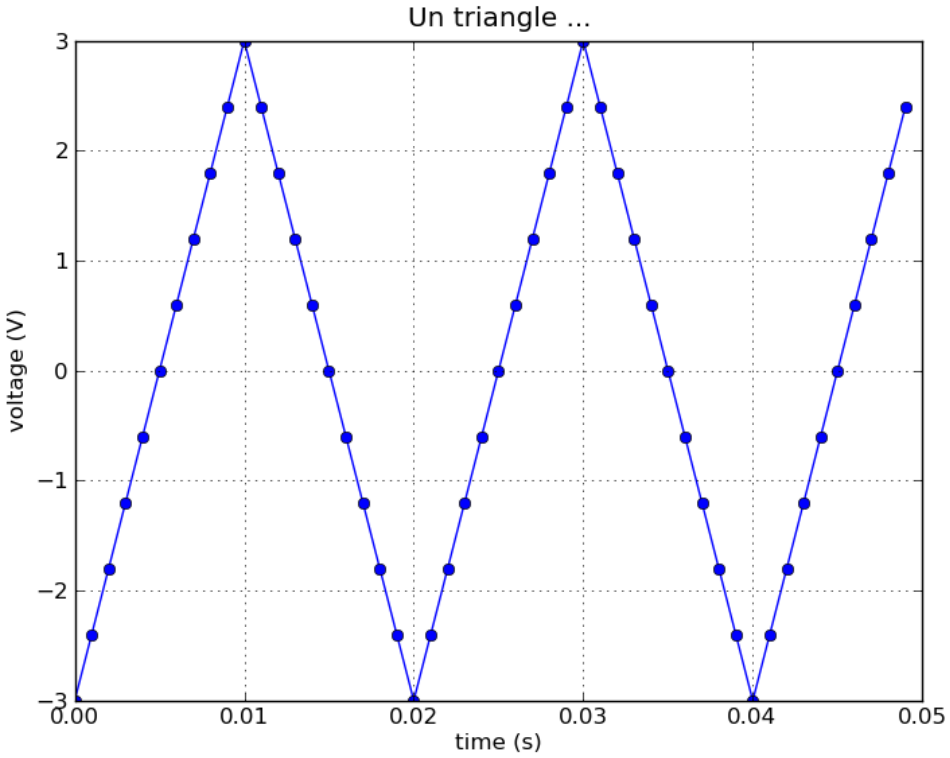
(b) Faire valider.

2.3 Signal Triangle

$$x(t) = a \times \left(4 \times \left(\left|\frac{t}{T} - \left\lfloor \frac{t}{T} + \frac{1}{2} \right\rfloor\right|\right) - 1.0\right) \tag{6}$$

(a) Proposer une fonction (dans le même module) permettant d'obtenir un signal "triangle".

Vous devriez obtenir, si $a = 3, f = 50.0, fe = 1000.0, nT = 2$:



(b) Faire valider.

3 Evolutions du code

3.1 Utilisation des bibliothèques

Faire afficher un carré, une dent de scie et un triangle à $50Hz$ en utilisant les fonctions de la bibliothèque `Scipy` :
`http://docs.scipy.org/doc/scipy-0.13.0/reference/signal.html`

3.2 Paradigme "Objet"

Jusqu'à présent nous avons exploité le paradigme procédural pour écrire le code de génération des signaux :

- ➢ carré,
- ➢ sinusoïde,
- ➢ dent de scie,
- ➢ triangle, ...

Considérons que ces codes était des prototypes, vous devez faire évoluer votre code pour mettre à disposition des **classes** : carré, triangle, ...

Pour vous aider :

`http://docs.python.org/2/tutorial/classes.html`

(a) Créer un module `tp1_usingclasses.py` et faire valider.

- ➢ L'obtention d'une factorisation "maximale" du code est un des critères d'évaluation.
- ➢ Il y a aussi la richesse des méthodes proposées : "plot", "pickle" ?

3.3 Vectorisation du code

L'utilisation de la bibliothèque `numpy` permet de vectoriser assez facilement ce type de code.

Sauf cas spécial (à expliquer et à défendre), l'instruction "for" ne doit plus apparaître dans votre module `tp1_usingclasses.py`

- ➢ Le `math.sin` est remplacé par le `numpy.sin`, etc.

(a) Faire valider.

4 Pour certains ...

4.1 Tonalités DTMF

Ecrire un script qui réconstitue le numéro de téléphone à partir du signal sonore DTMF.

4.2 Les images

Les images sont aussi des signaux ... avec deux dimensions !

L'image numérique est un signal 2D : $f(x, y)$, la valeur du pixel.

$$f : \quad \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N} \tag{7}$$

$$x, y \longrightarrow f(x, y) \tag{8}$$

4.2.1 La notion de fréquence pour une image

Sur la base d'une image de taille $(2^n \times 2^n)$ et $n = 6$ vous allez générer une sinusoïde verticale en vous inspirant du code suivant :

```
'''
Created on 24 janv. 2011
@author: menez
Ref :
    http://www.academictutorials.com/graphics/graphics-2d-fourier-transform.asp
    http://local.wasp.uwa.edu.au/~pbourke/miscellaneous/dft/
'''
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import numpy as np
import math

if __name__ == '__main__':
    SZ = 64
    im = np.zeros((SZ,SZ), np.uint8) # Image comme un tableau
    X = range(-SZ/2,SZ/2)
    Y = range(-SZ/2,SZ/2)

    # fp periodes sur une image
    fp = 4.0    # 1.0; 2.0; 4.0; 8.0; 16.0
    fr = fp/SZ # frequence reduite
    for i in X:
        for j in Y:
            im[i,j]= 128 + 128*math.sin(2*3.14*fr*i)

    plt.figure(1)
    plt.clf()
    plt.imshow(im,cmap=plt.cm.gray)

    # Plot de l'image comme une surface en 3D
    # Generation d'une meshgrid :
    # XG = tableau 2D des x sur le plan forme par "X,Y",
    # XG[i,j] = X[j] et idem pour YG
    XG, YG = np.meshgrid(X, Y)
    fig = plt.figure(2)
    plt.clf()
    ax = Axes3D(fig)
    ax.plot_surface(XG, YG, im, rstride=1, cstride=1, cmap=cm.jet)
    plt.show()
```

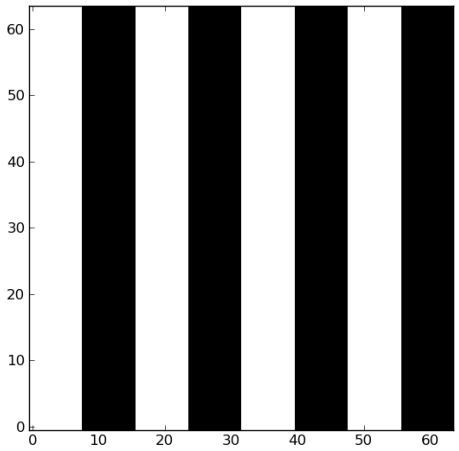
Ce code produit 2 figures :

① L'image issue de la matrice de pixels ainsi formée,

② La représentation 3D de la sinusoïde formant l'image.

4.2.2 Synthèse d'images

Dans cet exercice, vous générez des images, ou plutôt des mires, (64×64) composées de lignes verticales comme celle-ci :



① Si on fait un parallèle entre signal 1D et signal 2D (image), à quel signal 1D correspond cette mire ?

② Refaire ce travail sur un damier (sinusoïde et mire).

5 Du son !

A partir des signaux que vous êtes désormais capable de générer, on va essayer de produire un fichier au format audio WAV ... pour écouter.

J'ai trouvé sur Internet : `http://fsincere.free.fr/isn/python/cours_python_ch9.php` un morceau de code qui permet de créer un fichier au format WAV, que l'on pourra ensuite écouter.

```
#-*- coding : utf-8 *-*
# Python version 2.7
# (C) Fabrice Sincere Modified by Gilles MENEZ
import wave
import math

print 'Creation d\'un fichier audio au format WAV (PCM 8 bits stereo 44100 Hz)'
NomFichier = 'son.wav'
fson = wave.open(NomFichier,'w') # instanciation de l'objet fson
nbCanal = 2     # stereo
nbOctet = 1     # taille d'un echantillon : 1 octet = 8
fech = 44100    # frequence d'echantillonnage

frequenceG = float(raw_input('Frequence du son du canal de gauche (Hz) ? '))
frequenceD = float(raw_input('Frequence du son du canal de droite (Hz) ? '))
niveauG = float(raw_input('Niveau du son du canal de gauche (0 Ã  1) ? '))
niveauD = float(raw_input('Niveau du son du canal de droite (0 Ã  1) ? '))
duree = float(raw_input('Duree (en secondes) ? '))

nbEchantillon = int(duree*fech)
print 'Nombre d\'echantillons :',nbEchantillon

parametres = (nbCanal,nbOctet,fech,nbEchantillon,'NONE','not compressed') # tuple
fson.setparams(parametres)     # creation de l'en-tete (44 octets)

# niveau max dans l'onde positive : +1 -> 255 (0xFF)
# niveau max dans l'onde negative : -1 -> 0 (0x00)
# niveau sonore nul : 0 -> 127.5 (0x80 en valeur arrondi)

amplitudeG = 127.5*niveauG
amplitudeD = 127.5*niveauD

print 'Generation d\'un son sinusoidale sur chaque canal ...'
for i in range(0,nbEchantillon):
    # canal gauche
    # 127.5 + 0.5 pour arrondir Ã  l'entier le plus proche
    valG = chr(int(128.0 + amplitudeG*math.sin(2.0*math.pi*frequenceG*i/fech)))
    # canal droit
    valD = chr(int(128.0 + amplitudeD*math.sin(2.0*math.pi*frequenceD*i/fech)))
    fson.writeframes(valG + valD) # ecriture frame

fson.close()
```

(a) Comprendre et adapter ce code pour en faire une méthode de classe et permettre une utilisation qui pourrait ressembler à ce qui suit :

```
niv = 0.8
s = Sinusoide(127.5*niv, f=440.0)
s.make(fe=44100.0, nT=1000)
s.write_as_wav("son.wav")
```

(b) Ecouter ce "la 440" et faire valider.

(c) Générer un fichier qui contient un "do ré mi fa sol la si do".
Les fréquences correspondantes sont :

```
freq=[264,297,330,352,396,440,495,528]
```

(d) La génération du fichier WAV nécessite une quantification opérée par l'utilisation de la fonction `int()`.

Montrer sur un plot les deux signaux : l'original et le quantifié.

Faire valider

6 Quantification

Dans le cadre de cet exercice, on souhaite développer une méthode de quantification "plus paramétrable" que la fonction **int** et qui permette notamment de choisir le nombre de bits utilisés pour représenter chaque échantillons.

6.1 Quantificateur uniforme

Un signal numérique est un signal discrétisé en temps/abscisse **et** en valeurs/ordonnées.

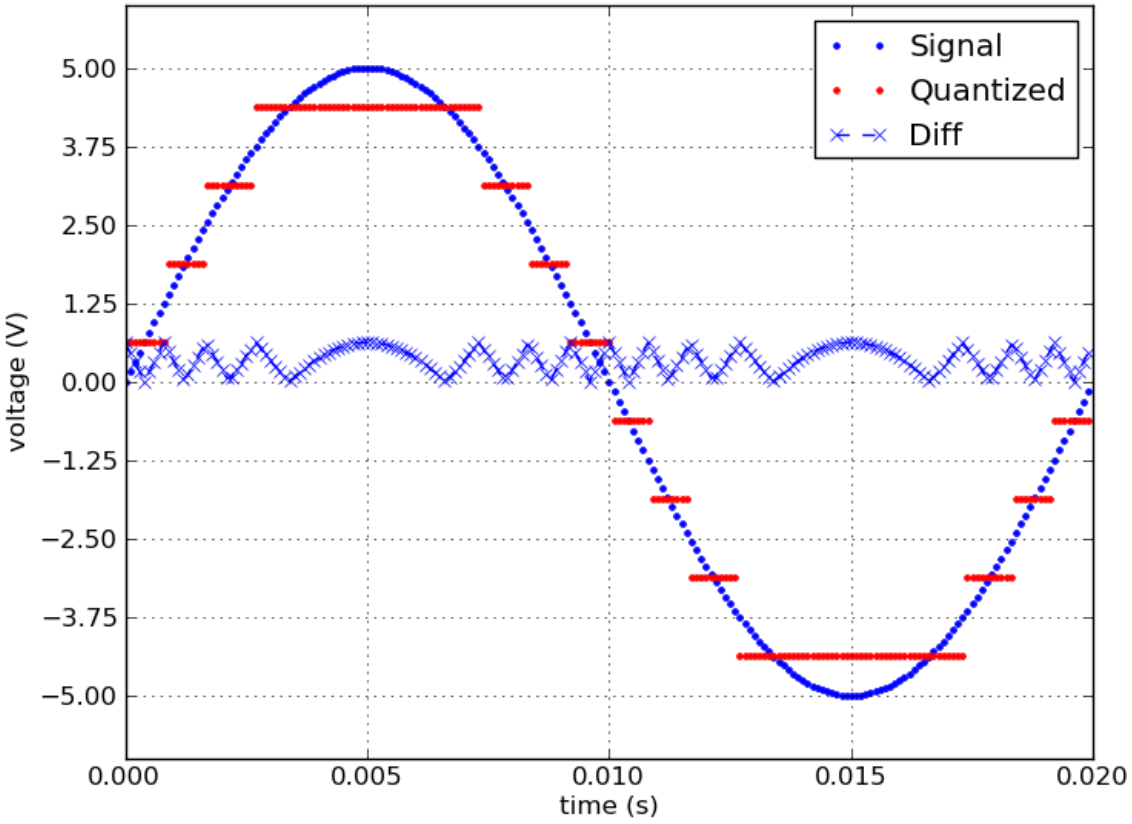
Pour l'instant,

- ✔ soit 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.
- ✔ soit nous avons quantifié sur l'ensemble des entiers. L'erreur commise est alors visible.

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 dans le module `tp1_q.py` , il faut comprendre :

① Ce que représentent chaque courbe et notamment celle constituée de "palliers" rouge ?

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

③ Pourquoi y en a t'il 8 ?

Pour arriver à vos fins, vous devriez sans doute aussi vous documenter sur les fonctions Python : "round", "math.floor", "math.ceil" ...
L'enrobage graphique qui permet d'obtenir ce graphe est :

```
#!usr/bin/python
#-*- coding : utf-8 *-*
'''
File tp1_q_vide.py  : Created on 28 feb 2012
@author: menez
Illustration de la quantification
'''
import math
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator
import tp1_0

def QS(sig_s, vmax, b):
    """
    Quantificateur sur b bits d'un signal sig_s
    d'amplitude max vmax
    Cette fonction rend le signal quantifie
    et le bruit sur chaque echantillon.
    """
    # pour l'instant le signal quantifie est le meme que le
    # signal original.
    # le signal de bruit est donc 0 puisqu'il n'y a pas de
    # quantification.
    return sig_s, [0]*len(sig_s)

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

if __name__ == '__main__':
    np.set_printoptions(linewidth=250)
    np.set_printoptions(precision=3, suppress=True)
    a=5.0
    b=3
    step = 2*a/(2**b)

    fe = 2000.0
    f = 50.0
    nT=2
    x,y=tp1_0.make_sin(a,0,f=f,fe=fe,nT=nT)

    #print y
    #y = np.array(y)+a
    z,err = QS(y,a,b)

    # plot du signal quantifie
    fig = plt.figure(figsize=(12,12))
    ax = fig.add_subplot(1,1,1)
    majorLocator = MultipleLocator(step)
    ax.yaxis.set_major_locator(majorLocator)
    plot(x,y,"",'bo', l="Signal")
    plot(x,z,"",'rs', l="Quantized")
    plot(x,err,"",'--x', l="Diff")
    plt.title("Sinusoide : $f_e$ =" + str(fe) + ", $f$ = " + str(f) + ", $d$ = " + str(nT) )
    plt.legend()
    plt.show()
```

(a) Faire valider.
Si possible, vous intégrez cette méthode à votre démarche "objet".
Vous montrez que votre algorithme permet de génerer des quantificateur sur 3 bits **mais aussi 4 et 5 bits** !

(b) Essayer de faire écouter le résultat dans un WAV.

6.2 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)

$$MSE = \sigma_q^2 = \frac{1}{N}\sum_i (x(i) - x_q(i))^2 \tag{9}$$

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

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

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

$$\sigma_x^2 = \frac{1}{N}\sum_i (x(i))^2 \tag{11}$$

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

(a) Faire valider.

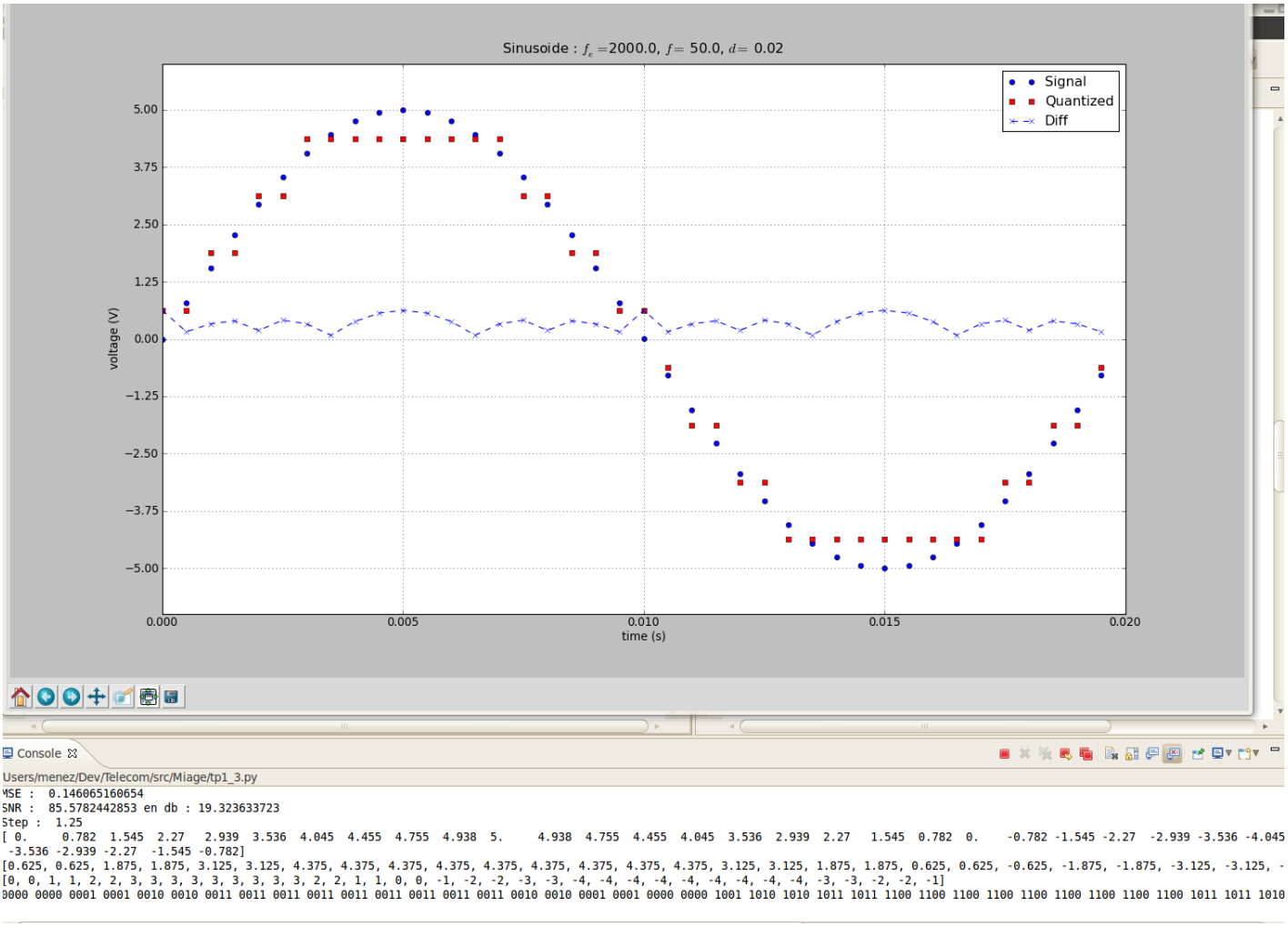
6.3 Encodage sur un signal bivalent

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 :

Vous pourriez avoir besoin de ça :

`http://www.daniweb.com/software-development/python/code/216539`

Pour obtenir quelquechose qui ressemble à ça :



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)

(a) Faire valider.