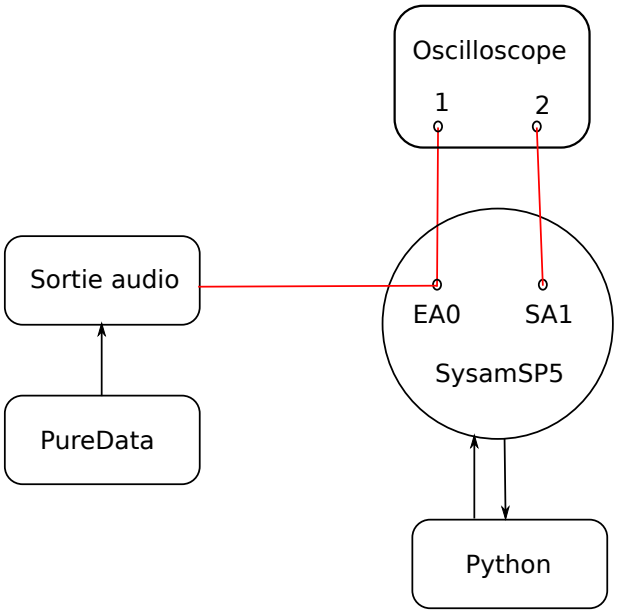
Filtrage numérique d'un signal

1. Introduction

Ce chapitre montre comment effectuer un filtrage numérique d'un signal numérisé par la carte SysamSP5. Le signal à filtrer est fourni sur la sortie audio de l'ordinateur au moyen du logiciel de synthèse de son puredata. Le programme puredata syntheseHarmonique.pd permet de faire la synthèse d'un signal périodique comportant jusqu'à 4 harmoniques, avec possibilité d'ajouter du bruit.

La sortie audio de l'ordinateur (canal 0) est reliée à l'entrée EA0 de la carte SysamSP5. Elle est également reliée à la voie 1 d'un oscilloscope. Le signal filtré numériquement est envoyé sur la sortie SA1 de la carte et visualisé sur la voie 2 de l'oscilloscope.



2. Conversion analogique-numérique

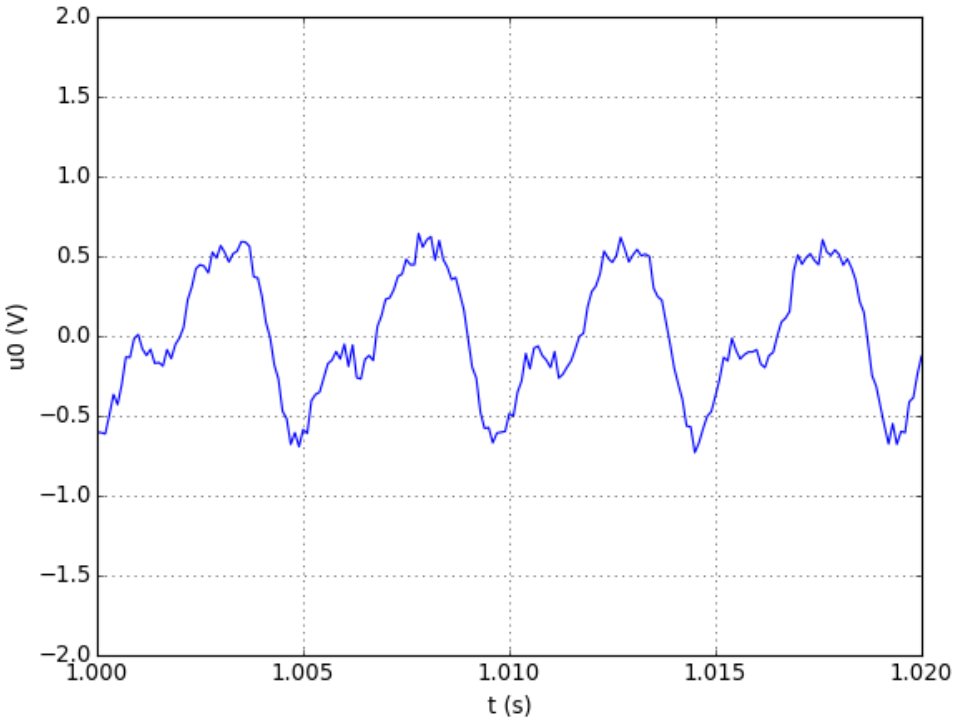
Le signal périodique, de fréquence 207 Hz, comporte 3 harmoniques. On fait une acquisition d'une durée de 2 s avec une fréquence d'échantillonnage de 10 kHz.

```
from matplotlib.pyplot import *
import numpy
import math
import cmath
import numpy.fft
import scipy.signal


import pycanum.main as pycan
```

```
nom="207Hz"
can = pycan.Sysam("SP5")
# configuration de l'entrée 0 avec un calibre 2.0 V
can.config_entrees([0],[2.0])
fe=10000.0
te=1.0/fe
T=2.0
N = int(T/te)
can.config_echantillon(te*10**6,N)
can.acquerir()
t0=can.temps()[0]
u0=can.entrees()[0]
numpy.savetxt("%s.txt"%nom,[t0,u0])
```

```
[t0,u0] = numpy.loadtxt("207Hz.txt")
figure()
plot(t0,u0)
xlabel('t (s)')
ylabel('u0 (V)')
grid()
axis([1,1.02,-2,2])
```



Le signal de la sortie audio a une amplitude maximale de 1 V. On voit bien l'effet du bruit sur le signal échantillonné.

3. Calcul du filtre passe-bas

Pour réduire le bruit, on définit un filtre passe-bas à réponse impulsionnelle finie.

Pour définir un filtre passe-bas, il faut tout d'abord calculer le rapport de la fréquence de coupure sur la fréquence d'échantillonnage (qui doit être inférieur à $1/2$) :

$$a = \frac{f_c}{f_e} \tag{1}$$

On appelle filtre idéal un filtre qui laisse passer sans atténuation toutes les fréquences inférieures à la fréquence de coupure, et qui annule les fréquences supérieures à la coupure. Son déphasage doit varier linéairement avec la fréquence dans la bande passante. Malheureusement, la réponse impulsionnelle d'un filtre passe-bas idéal est infinie. Elle est donnée par :

$$g(k) = 2a\frac{\sin(2\pi\, ka)}{2\pi\, ka} \tag{2}$$

Cette réponse impulsionnelle est rendue finie par troncature. Soit P l'indice de troncature. La réponse impulsionnelle finie comporte alors $N = 2P + 1$ termes. Elle est définie par :

$$h_k = g(k - P) \tag{3}$$

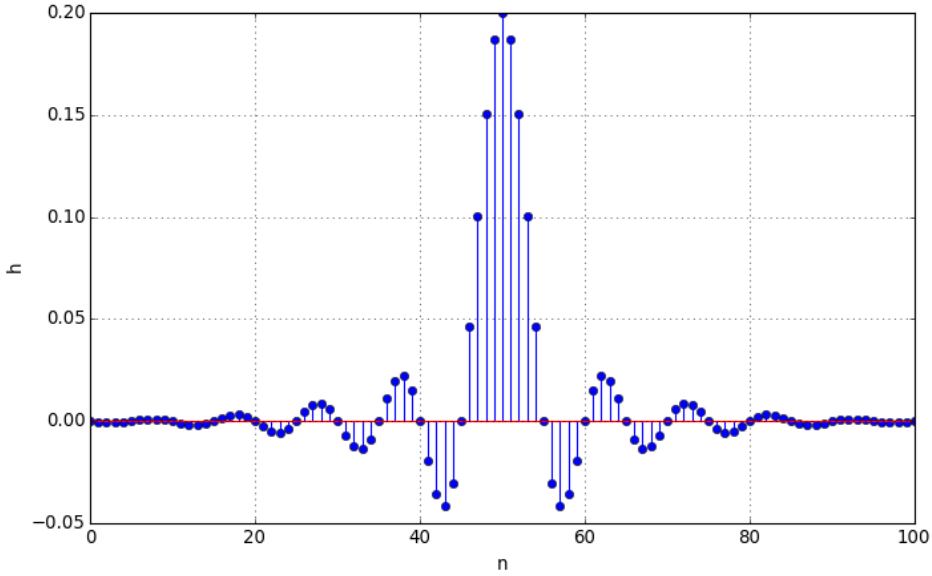
l'indice k variant de 0 à $2P$.

L'indice de troncature devra être d'autant plus grand que a est faible. Il faut donc éviter une fréquence d'échantillonnage trop grande par rapport à la fréquence de coupure (le sur-échantillonnage n'est pas toujours souhaitable en filtrage numérique). Un critère simple est $Pa > 1$. Pour un coefficient a donné, la sélectivité du filtre augmente avec P. À la limite $P \to \infty$, on obtient un filtre idéal.

Voici le calcul de la réponse impulsionnelle, pour une fréquence de coupure de $1000\ Hz$. Pour éliminer les ondulations dans la bande passante, on multiplie la réponse impulsionnelle par une fenêtre de Hamming.

```
fc1 = 1000.0
a = fc1/fe
P=50
h = numpy.zeros(2*P+1)
def sinc(u):
    if u==0:
        return 1.0
    else:
        return math.sin(math.pi*u)/(math.pi*u)
for k in range(2*P+1):
    h[k] = 2*a*sinc(2*(k-P)*a)
h = h*scipy.signal.get_window("hamming",h.size)

figure(figsize=(10,6))
stem(h)
xlabel("n")
ylabel("h")
grid()
```



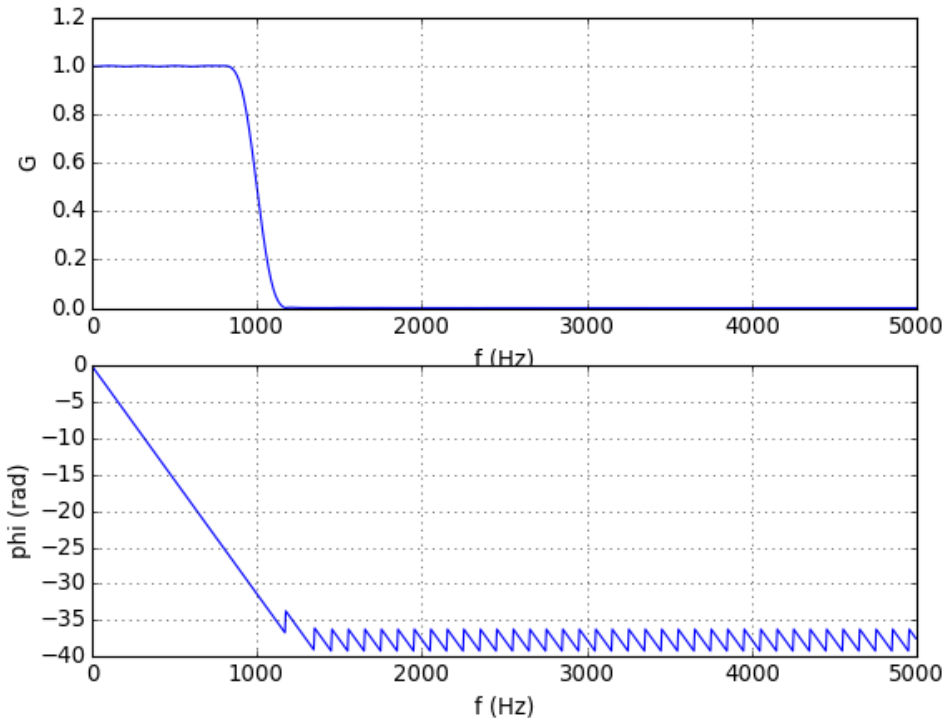
La réponse fréquentielle du filtre est donnée par la fonction suivante :

$$H_f(f) = \sum_{k=0}^{N-1} h_k \exp(-i2\pi\, k\, f\, T_e) \tag{4}$$

Voici le tracé de la réponse fréquentielle du filtre :

```
def reponseFreq(h,nf):
    f = numpy.arange(0.0,0.5,0.5/nf)
    g = numpy.zeros(f.size)
    phi = numpy.zeros(f.size)
    for m in range(f.size):
        H = 0.0
        for k in range(h.size):
            H += h[k]*cmath.exp(-1j*2*math.pi*k*f[m])
        g[m] = abs(H)
        phi[m] = cmath.phase(H)
    phase = numpy.unwrap(phi)
    return (f,g,phase)

(f,g,phase) = reponseFreq(h,1000)
figure()
subplot(211)
plot(f*fe,g)
xlabel('f (Hz)')
ylabel('G')
grid()
subplot(212)
plot(f*fe,phase)
xlabel('f (Hz)')
ylabel('phi (rad)')
grid()
```



Le gain est égal à 1 jusqu'à 800 Hz et la phase varie linéairement avec la fréquence. Les trois harmoniques du signal (fréquence 207 Hz) sont dans la bande passante.

4. Filtrage numérique

On effectue le filtrage des échantillons par convolution :

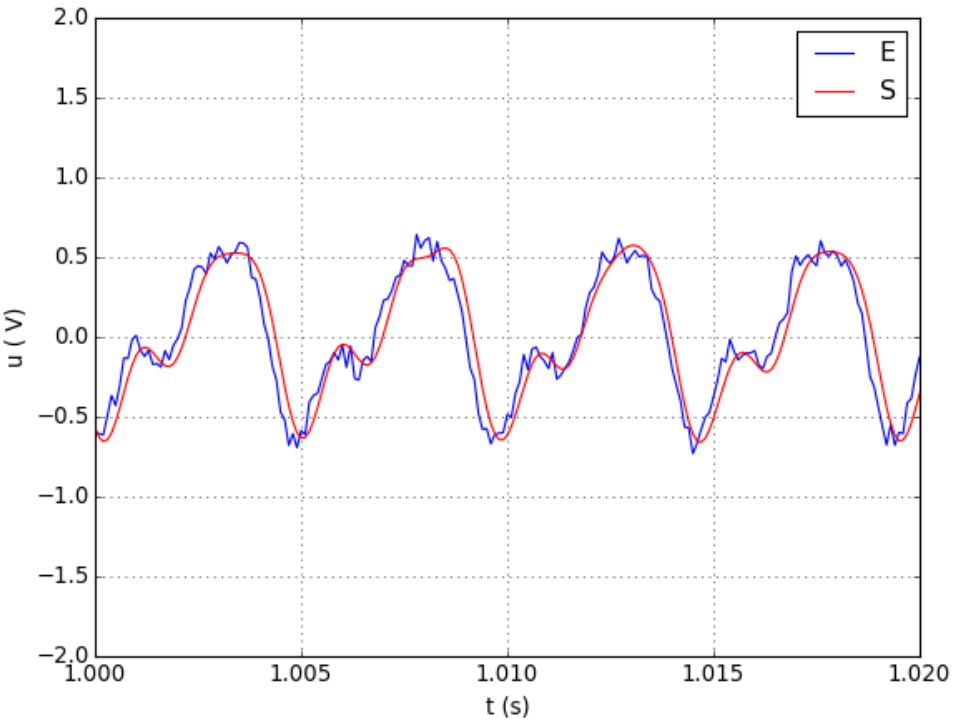
```
y = scipy.signal.convolve(u0,h,mode='same')
```

L'option `mode='same'` permet de garder le même nombre d'échantillons. Pour simuler le retard qui apparaît dans un filtre qui fonctionne en temps réel, on doit effectuer un décalage des échantillons :

```
y = numpy.roll(y,P)
```

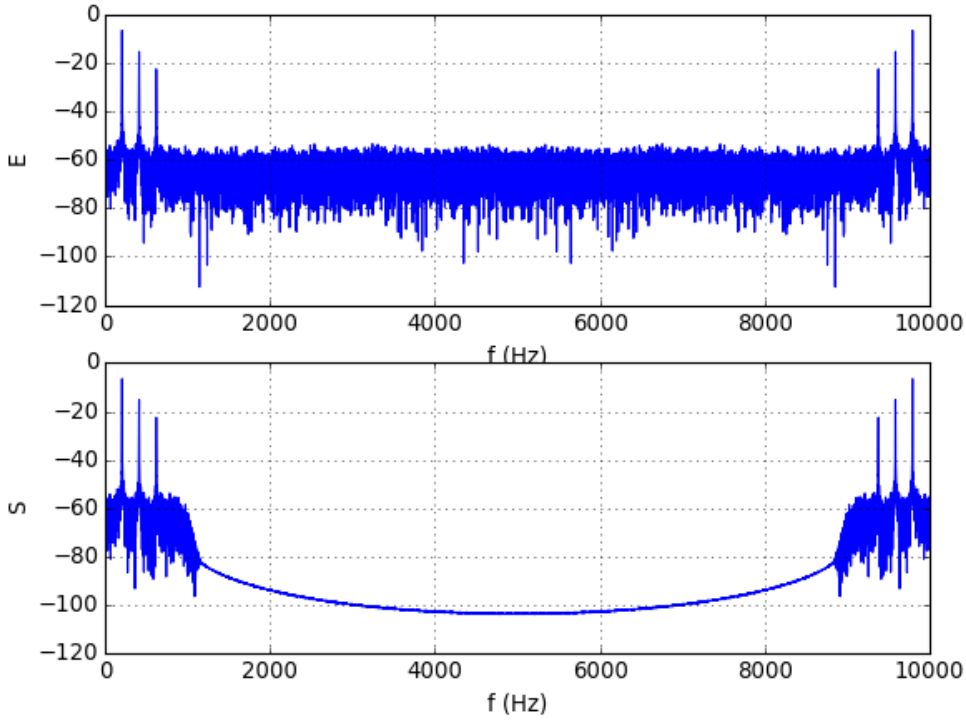
On trace les échantillons du signal et ceux du signal filtré :

```
figure()
plot(t0,u0,'b',label='E')
plot(t0,y,'r',label='S')
xlabel("t (s)")
ylabel("u ( V)")
axis([1,1.02,-2,2])
legend(loc='upper right')
grid()
```



On calcule aussi la transformée de Fourier discrète des deux signaux afin de tracer leur spectre en échelle décibel :

```
freq = numpy.arange(N)*1.0/T
tfd = numpy.fft.fft(u0)
spectre_entree = 20.0*numpy.log10(numpy.abs(tfd)*2.0/N)
tfd = numpy.fft.fft(y)
spectre_sortie = 20.0*numpy.log10(numpy.abs(tfd)*2.0/N)
figure()
subplot(211)
plot(freq,spectre_entree)
xlabel('f (Hz)')
ylabel('E')
grid()
subplot(212)
plot(freq,spectre_sortie)
xlabel('f (Hz)')
ylabel('S')
grid()
```



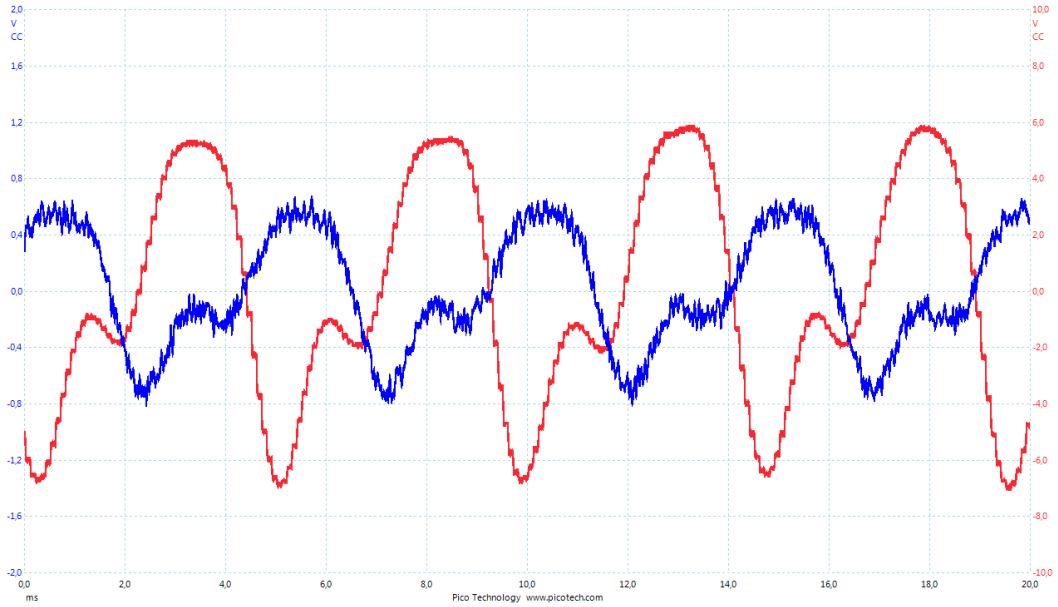
5. Conversion numérique-analogique

On programme la sortie SA1 de la carte SysamSP5 avec les échantillons du signal filtré. On multiplie les valeurs par 10 de manière à profiter de la gamme de tension de sortie $[-10, 10]\ V$. Le convertisseur N/A ayant une résolution de 12 bits, il est important de faire cette multiplication pour limiter au maximum le bruit de quantification sur le signal analogique de sortie.

```
can.config_sortie(1,te*10**6,y*10,-1)
can.declencher_sorties(1,0)
show(block=True)
can.stopper_sorties(1,0)
```

La fonction `show(block=True)` est bloquante. Elle permet d'afficher les graphiques précédents et de stopper l'exécution. Le dernier argument de la fonction `can.config_sortie` égal à -1 permet de répéter les échantillons sans fin.

Voyons les signaux obtenus sur l'oscilloscope. La voie 1 (bleu) est le signal analogique de départ (sortie audio non filtrée). La voie 2 (rouge) est la sortie SA1 de la carte SysamSP5, qui comporte le signal filtré converti en analogique.



Il faut remarquer que les deux signaux ne sont pas synchrones. Le décalage apparaissant sur cette figure n'a donc aucune signification.

Le signal analogique obtenu en sortie ne comporte plus le bruit analogique du signal d'origine. En revanche, il comporte un bruit de quantification dû à la fréquence d'échantillonnage de $10\,\mathrm{kHz}$. La dernière étape est un lissage du signal pour éliminer ce bruit de quantification. On peut utiliser pour cela un filtre analogique. Nous allons plutôt utiliser une méthode numérique.

6. Lissage numérique

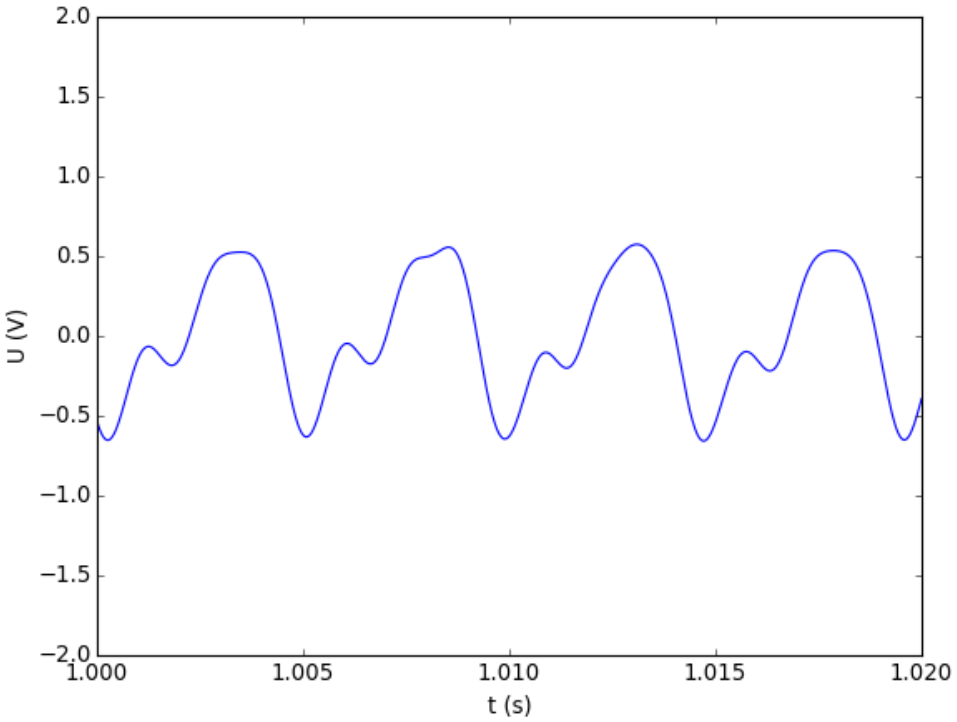
Cette technique consiste à augmenter la fréquence d'échantillonnage du signal numérique, avant de lui appliquer un lissage par interpolation. C'est la méthode utilisée dans les lecteurs de CD audio.

On choisit d'augmenter la fréquence d'échantillonnage d'un facteur $n = 5$. On doit créer un nouveau tableau d'échantillons en répliquant chaque échantillon n fois.

```
n=5
fe2 = fe*n
te2 = 1.0/fe2
y1 = numpy.zeros(0)
t1 = numpy.zeros(0)
i=0
t=0.0
while i<y.size:
    for j in range(n):
        y1 = numpy.append(y1,y[i])
        t1 = numpy.append(t1,t)
        t += te2
    i += 1
```

Le lissage peut être effectué par un filtre RIF passe-bas :

```
P = 10
h = scipy.signal.firwin(numtaps=2*P+1,cutoff=[0.5/n],nyq=0.5,window='hann')
y2 = scipy.signal.convolve(y1,h,mode='same')
figure()
plot(t1,y2)
xlabel('t (s)')
ylabel('U (V)')
axis([1,1.02,-2,2])
```

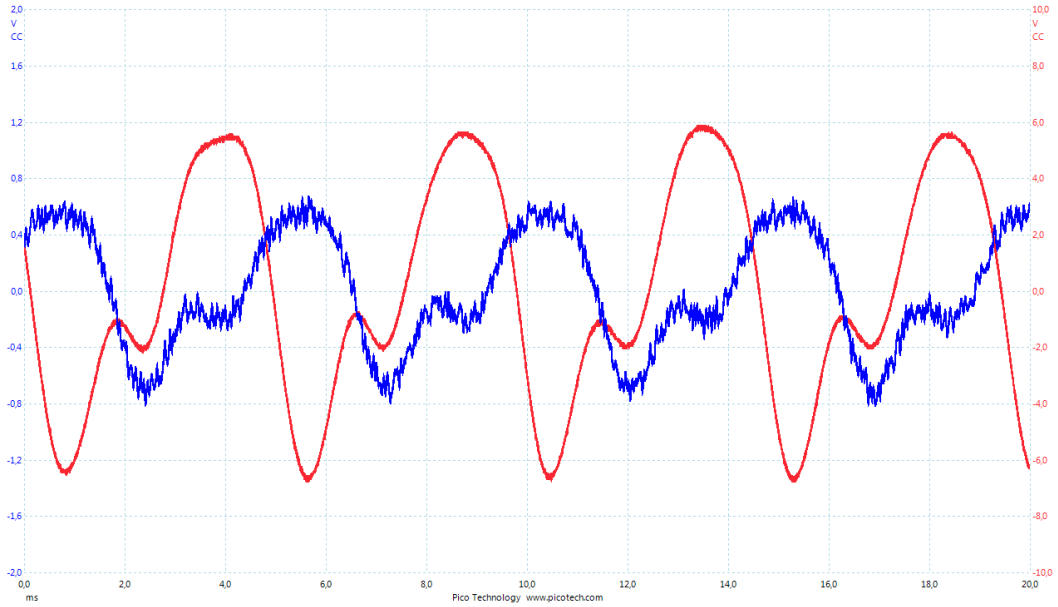


Enfin on fait la conversion numérique-analogique avec la nouvelle fréquence d'échantillonnage :

```
can.config_sortie(1,te2*10**6,y2*10,-1)
can.declencher_sorties(1,0)
show(block=True)
can.stopper_sorties(1,0)

can.fermer()
```

Voici le résultat à l'oscilloscope :


2,0
V
CC
1,6
1,2
0,8
0,4
0,0
-0,4
-0,8
-1,2
-1,6
-2,0
10,0
V
CC
8,0
6,0
4,0
2,0
0,0
-2,0
-4,0
-6,0
-8,0
-10,0
0,0
2,0
4,0
6,0
8,0
10,0
12,0
14,0
16,0
18,0
20,0
ms
Pico Technology www.picotech.com