Filtrage d'un signal audio en temps réel

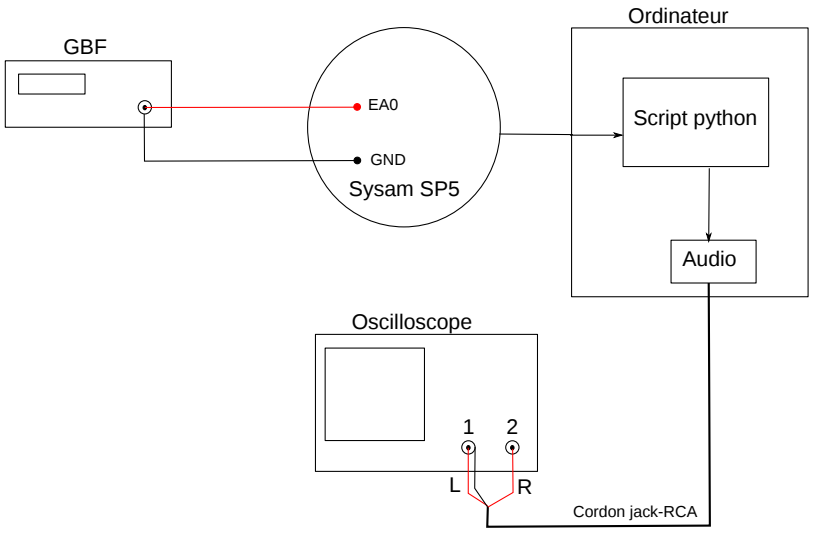
1. Introduction

Ce chapitre montre comment faire l'acquisition d'un signal audio avec la carte SysamSP5 tout en faisant un filtrage en temps réel. Le signal audio initial et le signal filtré sont redirigés vers la sortie audio pour une observation à l'oscilloscope.

Les signaux sont traités par paquets de N échantillons (de 100 à 1000), ce qui conduit à un délai entre l'entrée et la sortie égal environ à N multiplié par la période d'échantillonnage. Il ne s'agit donc pas d'un véritable filtrage en temps réel (de type DSP), dont le délai de calcul est égal à la période d'échantillonnage.

2. Montage expérimental

Pour la mise au point du filtre, on utilise le montage suivant :



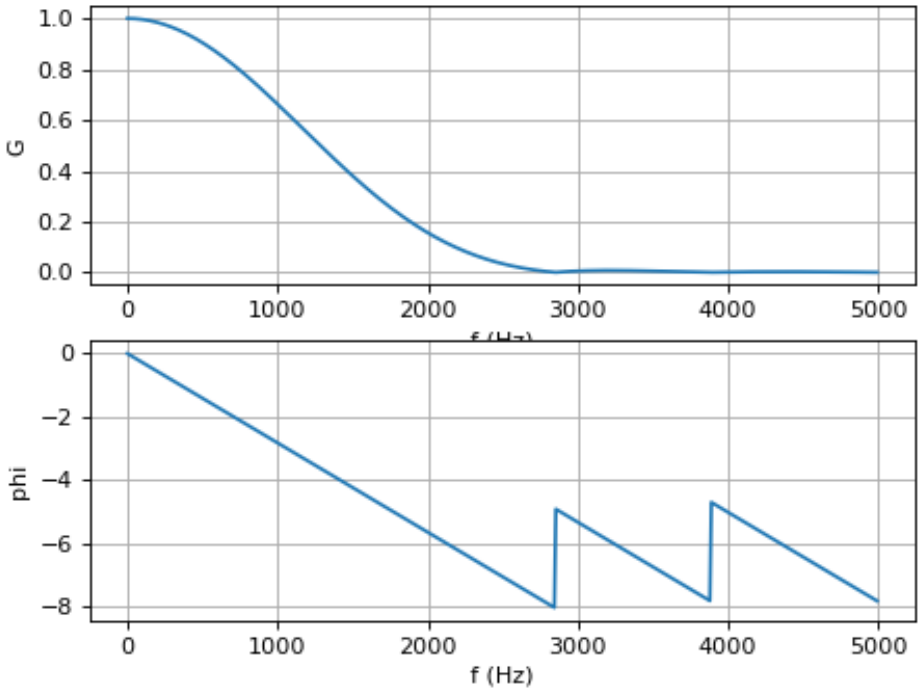
Un générateur de fonctions (GBF) délivre une sinusoïde dont l'acquisition est faite sur la voie EA0 de la carte SysamSP5. Le script python effectue le filtrage en temps réel. Le signal non filtré et le signal filtré sont envoyés sur les deux canaux (R et L) de la sortie audio, pour une observation à l'oscilloscope. On pourra ainsi comparer le signal non filtré et le signal filtré tels qu'ils se présentent dans un filtrage en temps réel réalisé par un circuit DSP.

Une application est le filtrage en temps réel du signal délivré par un microphone (avec amplificateur), ou par les capteurs d'une guitare électrique. Dans ce cas, on pourra écouter le résultat du filtrage avec un haut-parleur (en mono sur le canal R).

3. Filtrage RIF

Comme premier exemple, on considère un filtrage RIF passe-bas travaillant à une fréquence d'échantillonnage de 10 kHz. On commence par calculer les coefficients du filtre et tracer sa réponse fréquentielle :

```
import numpy
from matplotlib.pyplot import *
import scipy.signal

fe=10000.0
fc = 1000

b = scipy.signal.firwin(numtaps=10,cutoff=[fc/fe],nyq=0.5,window='hann')
w,h =scipy.signal.freqz(b)
g = numpy.absolute(h)
phase = numpy.unwrap(numpy.angle(h))
figure()
subplot(211)
plot(w/(2*numpy.pi)*fe,g)
xlabel("f (Hz)")
ylabel("G")
grid()
subplot(212)
plot(w/(2*numpy.pi)*fe,phase)
xlabel("f (Hz)")
ylabel("phi")
grid()
```



L'acquisition doit être faite en mode *permanent* et en flux continu. Pour cela, il faut configurer l'échantillonnage avec la fonction `config_echantillon_permanent(te,N)`, où te

est la période d'échantillonnage et N la taille des paquets. L'acquisition doit être lancée avec la fonction `lancer_permanent(repetition=1)`. La récupération du dernier paquet se fait avec `paquet(-1)` (qui renvoie une liste vide si aucun nouveau paquet n'est disponible).

Dans ce premier exemple, on utilisera le filtrage intégré au module `pycan`, qui se programme avec la fonction `config_filtre`.

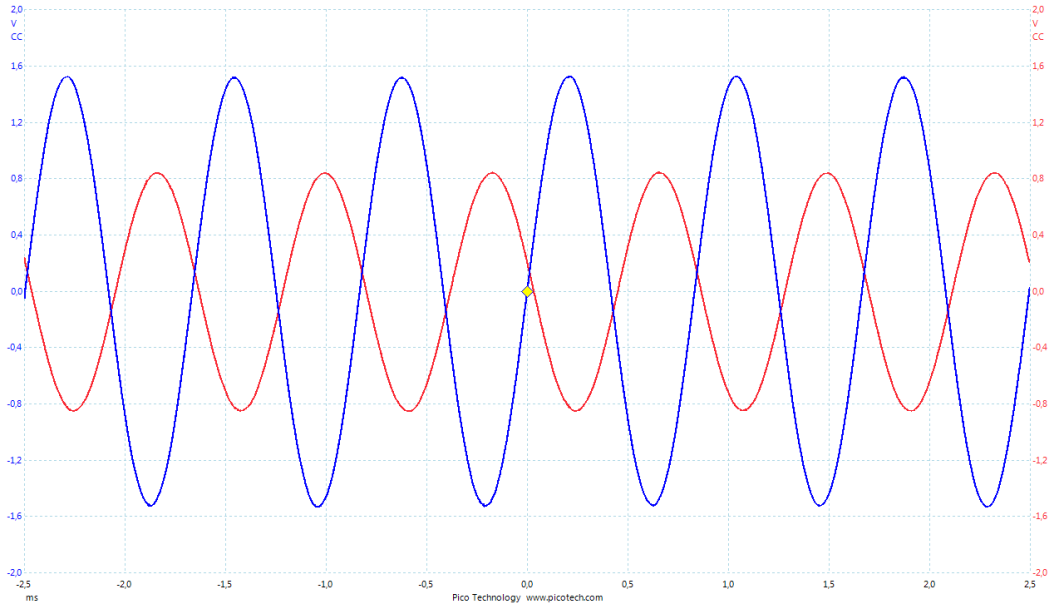
Pour utiliser la sortie audio, il faut tout d'abord créer deux tampons circulaires `RingBuffer` (un pour chaque voie de la sortie). La fonction `output_stream_start` permet de déclencher le flux audio de sortie avec une fréquence d'échantillonnage (identique à celle de la numérisation).

Voici le script complet, qui effectue le filtrage d'un nombre déterminé de paquets. Le signal non filtré et le signal filtré sont redirigés vers les deux voies R et L de la sortie audio.

filtrageRIFSortieAudio.py

```
import pycanum.main as pycan
import scipy.signal
import time

fe=10000.0
fc = 1000
b = scipy.signal.firwin(numtaps=10,cutoff=[fc/fe],nyq=0.5,window='hann')

sys=pycan.Sysam("SP5")
Umax = 2.0
sys.config_entrees([0],[Umax])
te=1.0/fe
N = 200 # taille des paquets
tampon_x = pycan.RingBuffer(6,N)
tampon_y = pycan.RingBuffer(6,N)
duree = N*te
sys.config_echantillon_permanent(te*1e6,N)
sys.config_filtre([1],b)
sys.lancer_permanent(repetition=1) # lancement d'une acquisition sans fin
pycan.output_stream_start(fe,tampon_x,tampon_y)

k = 0
while k<3000:
    data = sys.paquet(-1,reduction=1)
    if data.size!=0:
        tampon_x.write(data[1])
        tampon_y.write(data[2])
        k+=1
    time.sleep(duree*0.2)

pycan.output_stream_stop(0)
sys.stopper_acquisition()
time.sleep(1)
sys.fermer()
```

La figure suivante montre les oscillographes obtenus pour une sinusoïde de 1200 Hz avec une amplitude de crête à crête de 2 V.Le volume de la sortie audio est au maximum.



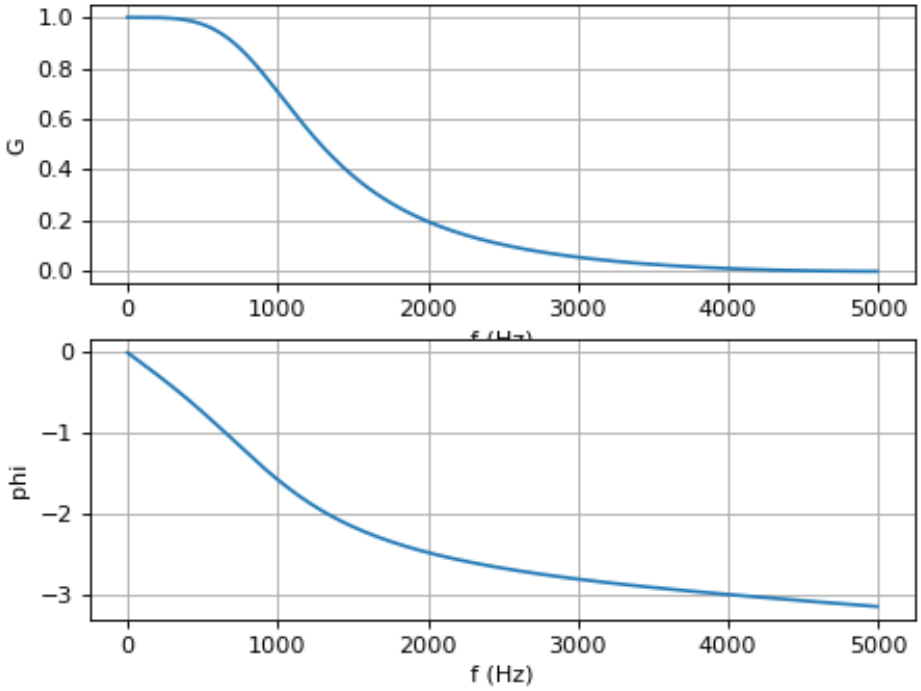
On voit qu'une valeur de 1 envoyée sur la sortie audio donne une tension d'environ $1{,}6\,\mathrm{V}$. Ce gain dépend bien sûr du réglage du volume. Pour une amplitude en entrée supérieure à $2{,}1\,\mathrm{V}$ (crête à crête), il se produit un écrêtage.

On remarque la qualité du lissage effectué par le circuit audio, alors que la fréquence d'échantillonnage est seulement 8 fois plus grande que la fréquence du signal.

4. Filtre récursif biquadratique

Les filtres récursifs biquadratiques sont très utilisés en traitement du son, en raison de leur grande efficacité pour seulement 5 coefficients. Voici par exemple un filtre passe-bas :

```
fe=10000.0
fc = 1000

b,a = scipy.signal.iirfilter(N=2,Wn=[fc/fe*2],btype="lowpass",ftype="butter")
w,h =scipy.signal.freqz(b,a)
g = numpy.absolute(h)
phase = numpy.unwrap(numpy.angle(h))
figure()
subplot(211)
plot(w/(2*numpy.pi)*fe,g)
xlabel("f (Hz)")
ylabel("G")
grid()
subplot(212)
plot(w/(2*numpy.pi)*fe,phase)
xlabel("f (Hz)")
ylabel("phi")
grid()
```




```
print(a)
--> array([ 1.        , -1.1429805,  0.4128016])

print(b)
--> array([0.06745527, 0.13491055, 0.06745527])
```

Le script de filtrage ci-dessous effectue le filtrage avec la fonction `scipy.signal.lfilter`. Le reste est identique au script précédent.

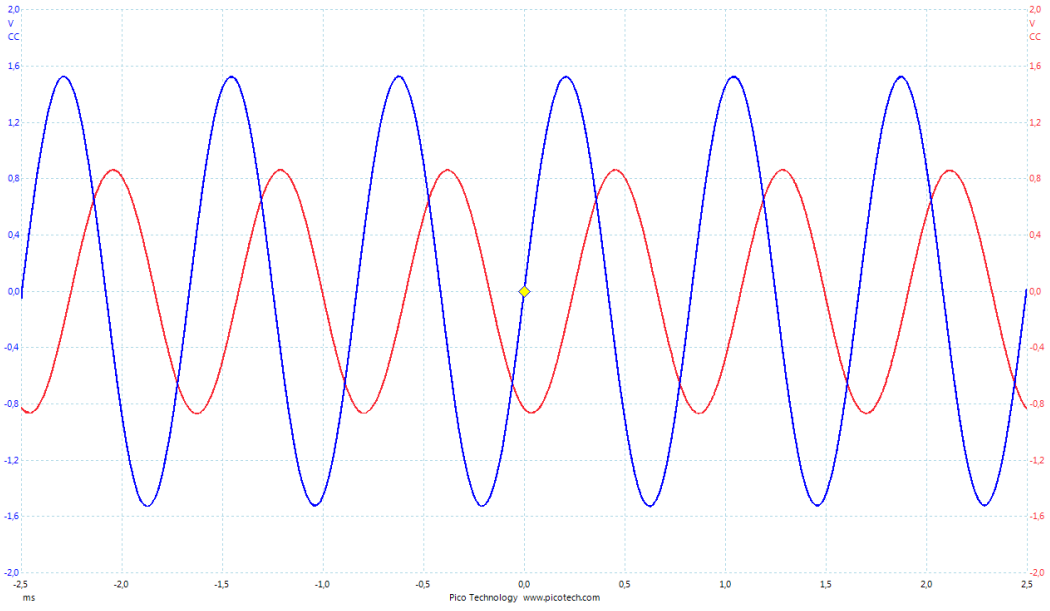
filtrageBIQUADSortieAudio.py

```
import pycanum.main as pycan
import scipy.signal
import time

fe=10000.0
fc = 1000
b,a = scipy.signal.iirfilter(N=2,Wn=[fc/fe*2],btype="lowpass",ftype="butter")
zi = scipy.signal.lfiltic(b,a,y=[0,0],x=[0,0]) # condition initiale

sys=pycan.Sysam("SP5")
Umax = 2.0
sys.config_entrees([0],[Umax])
te=1.0/fe
N = 200 # taille des paquets
tampon_x = pycan.RingBuffer(6,N)
tampon_y = pycan.RingBuffer(6,N)
duree = N*te
sys.config_echantillon_permanent(te*1e6,N)
sys.lancer_permanent(repetition=1) # lancement d'une acquisition sans fin
pycan.output_stream_start(fe,tampon_x,tampon_y)
```

```
k = 0
while k<3000:
    data = sys.paquet(-1,reduction=1)
    if data.size!=0:
        x = data[1]
        [y,zi] = scipy.signal.lfilter(b,a,x,zi=zi)
        tampon_x.write(x)
        tampon_y.write(y)
        k+=1
    time.sleep(duree*0.2)

pycan.output_stream_stop(0)
sys.stopper_acquisition()
time.sleep(1)
sys.fermer()
```

On remarque l'argument `zi` de la fonction `scipy.signal.lfilter`, qui permet de transmettre les variables d'état. La fonction `scipy.signal.lfiltic` permet de calculer l'état initial.

La figure suivante montre les oscillographes obtenus pour une sinusoïde de $1200\,\mathrm{Hz}$ avec une amplitude de crête à crête de $2\,\mathrm{V}$. Le volume de la sortie audio est au maximum.



Le délai entre le signal délivré par le GBF et le signal sur la sortie audio est environ égal à la période d'échantillonnage multipliée par la taille N des paquets. Si l'on augmente la fréquence d'échantillonnage, on peut aussi augmenter proportionnellement la taille des paquets. Par exemple, une fréquence d'échantillonnage de $40\,\mathrm{kHz}$ avec $N = 1000$ fonctionne bien. Le délai est alors de $25\,\mathrm{ms}$.