Acquisition en mode permanent

1. Introduction

L'acquisition en mode permanent se fait sans utiliser la mémoire interne de la carte Sysam SP5. Les données sont transférées dans la mémoire du PC au fur et à mesure de leur acquisition, avec un tampon de seulement 256 valeurs. L'acquisition en mode permanent ne fonctionne bien que pour des fréquences d'échantillonnage inférieures à $100\,\mathrm{kHz}$ (environ). Lorsque la fréquence d'échantillonnage est de l'ordre du mégahertz, il se produit de temps en temps des coupures dans l'échantillonnage. En revanche, il permet deux modes de fonctionnement intéressants :

- ▷ L'acquisition d'un très grand nombre d'échantillons, plus grand que la capacité mémoire de la carte, par exemple pour un enregistrement audio de plusieurs secondes.
- ▷ L'acquisition en flux continu, avec filtrage et analyse en continu (comme sur un oscilloscope).

2. Acquisition d'un grand nombre d'échantillons

2.a. Principe

L'acquisition échantillonnée utilise la mémoire interne de la carte Sysam SP5. Lorsqu'une seule voie d'entrée est utilisée, et aucune sortie, la mémoire disponible permet de stocker $2^{18} = 262144$ échantillons. Pour certaines applications, ce nombre est insuffisant. Par exemple, s'il faut sur-échantillonner à $1\ kHz$ et maintenir l'acquisition pendant une heure, il faut pouvoir acquérir en continu 3,6 millions d'échantillons. Le mode permanent permet de faire cela.

On commence par ouvrir l'interface et configurer la ou les entrées que l'on veut utiliser :

```
import pycanum.main as pycan
sys=pycan.Sysam("SP5")
Umax = 2
sys.config_entrees([0],[Umax])
```

Supposons que l'on veuille faire l'acquisition de 1 million d'échantillons à la fréquence de $10\,\mathrm{kHz}$. On configure l'échantillonnage par :

```
fe = 10000.0
N=1000000
sys.config_echantillon_permanent(te*1e6,N)
```

L'espace mémoire nécessaire est réservé dans l'ordinateur (pas dans la carte). Avec Numpy, le traitement de tableaux de plusieurs millions d'éléments ne pose aucun problème (attention tout de même à certaines fonctions graphiques).

Pour faire l'acquisition, on peut utiliser :

```
sys.acquerir_permanent()
```

qui retourne lorsque l'acquisition est terminée (après $100\,\mathrm{s}$ dans cet exemple), ou bien lancer l'acquisition sur une tâche parallèle (appel non bloquant) par :

```
sys.lancer_permanent()
```

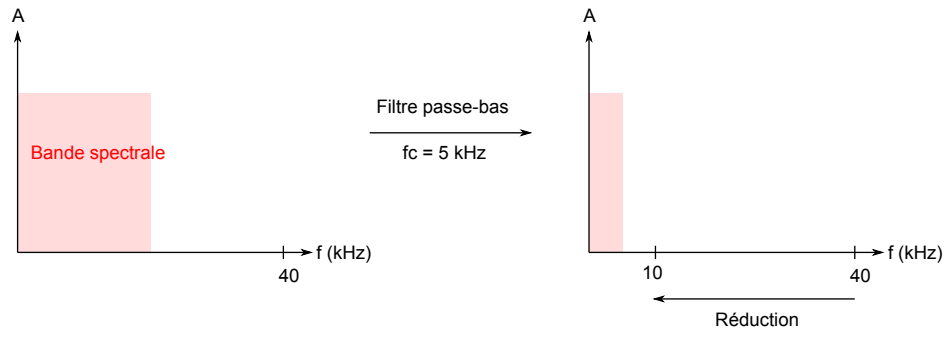
Cette fonction retourne dès que l'acquisition est lancée (voir exemple d'utilisation ci-dessous).

Lorsque l'acquisition est terminée, la récupération des données se fait comme pour l'acquisition en mode normal :

```
t = sys.temps()
u = sys.entrees()
t0 = t[0]
u0 = u[0]
```

2.b. Sur-échantillonnage et filtrage

Voici un exemple d'application du mode permanent. On fait l'acquisition d'un son à $40\,\mathrm{kHz}$ pendant plusieurs minutes. Cette fréquence d'échantillonnage permet d'échantillonner toutes les fréquences présentes dans le signal, et donc d'éviter le repliement de spectre. Cependant, il n'est pas toujours nécessaire de stocker tous ces échantillons pour en faire une analyse ultérieure, car la plupart des sons audibles ont un spectre qui ne s'étend pas au delà de $5\,\mathrm{kHz}$. Dans ce cas, on peut dire que l'acquisition est sur-échantillonnée par rapport au besoin final. Avant de réduire la fréquence d'échantillonnage, il faut néanmoins appliquer un filtrage numérique passe-bas au signal sur-échantillonné (filtre anti-repliement). Le principe de la méthode est montré sur la figure suivante, pour une réduction d'un facteur 4 :



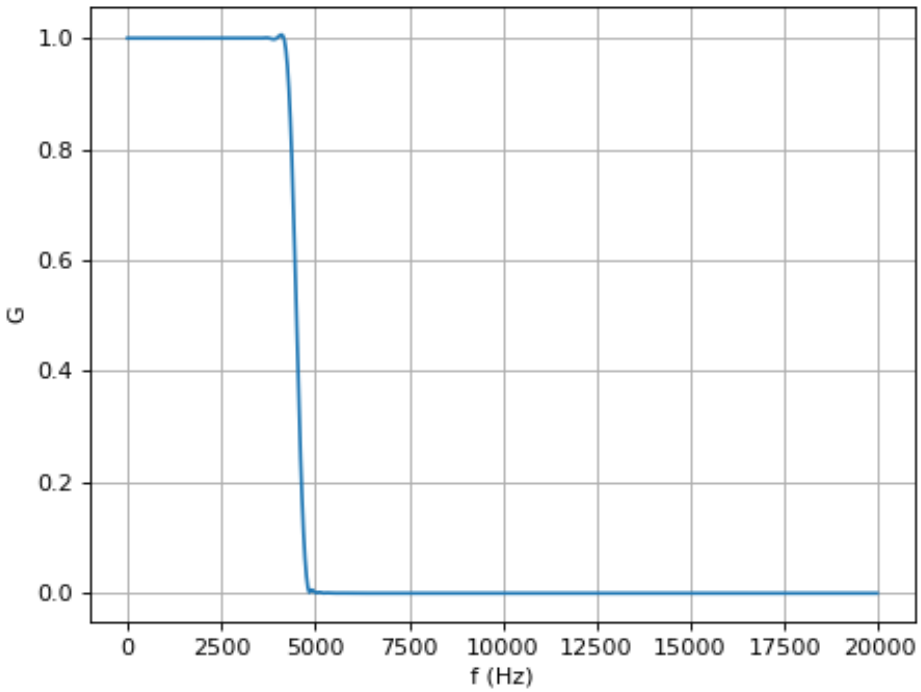
On définit un filtre RIF avec une fréquence de coupure inférieure à $5\ kHz$:

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

fe = 40000.0
fc = 4500
P = 200
b = scipy.signal.firwin(numtaps=P,cutoff=[fc/fe],nyq=0.5,window='hann')
```

On trace la réponse fréquentielle du filtre :

```
(w,h) = scipy.signal.freqz(b)
figure()
plot(w/(2*numpy.pi)*fe,numpy.absolute(h))
xlabel("f (Hz)")
ylabel("G")
grid()
```



On programme un échantillonnage pendant 1 minute :

```
sys=pycan.Sysam("SP5")
Umax = 2
sys.config_entrees([0],[Umax])
N = 2.4e6
te = 1.0/fe
sys.config_echantillon_permanent(te*1e6,N)
```

On peut faire l'acquisition, puis effectuer le filtrage et la réduction de la manière suivante :

```
sys.acquerir_permanent()
data = sys.entrees()
t = data[0]
u = data[1]
v = scipy.signal.convolve(u,mode='same')
y = v[0::4]
```

Il est aussi possible de faire le filtrage pendant l'acquisition, ce qui permet d'afficher le signal filtré au fur et à mesure. Pour cela, on définit le filtre par :

```
sys.config_filtre([1],b)
```

Après l'acquisition, les données sont récupérées par :

```
data = sys.entrees_filtrees(reduction=4)
```

2.c. Acquisition avec tracé du signal

Lorsque l'acquisition dure plusieurs minutes, il peut être utile de tracer le signal au fur et à mesure. Pour cela, il faut lancer l'acquisition en parallèle avec la commande `lancer_permanent()`, puis récupérer périodiquement les données acquises dans une boucle d'animation.

Le script suivant effectue les opérations suivantes :

- ▷ Choix d'une fréquence d'échantillonnage, d'une durée d'acquisition, d'une durée pour la fenêtre du tracé et d'un facteur de réduction de l'échantillonnage.
- ▷ Définition d'un filtre passe-bas anti-repliement (pour la réduction).
- ▷ Lancement de l'acquisition puis de la boucle d'animation.

La durée totale de l'acquisition est définie dans `duree_totale`. La durée de la fenêtre tracée est définie dans `duree`.

acquisitionPermanenteTrace.py

```
import pycanum.main as pycan
import math
from matplotlib.pyplot import *
import matplotlib.animation as animation
import numpy
import scipy.signal

sys=pycan.Sysam("SP5")
Umax = 2
sys.config_entrees([0],[Umax])
fe=40000.0 # fréquence de la numérisation
te=1.0/fe
duree_totale = 60
N = int(duree_totale*fe)
sys.config_echantillon_permanent(te*1e6,N)
reduction = 4 # réduction de la fréquence d'échantillonnage
fe_r = fe/reduction
te_r = te*reduction
duree = 0.1 # durée des blocs
longueur_bloc = int(duree/te_r) # taille des blocs traités
nombre_blocs = int(N*te/duree)
nombre_echant = nombre_blocs*longueur_bloc

#filtre passe-bas anti-repliement
fc = fe/reduction/2*0.8 # fréquence de coupure
b = scipy.signal.firwin(numtaps=40,cutoff=[fc/fe],nyq=0.5,window='hann')
```

```
sys.config_filtre([1],b)

sys.lancer_permanent()
n_tot = 0 # nombre d'échantillons acquis

fig,ax = subplots()
t = numpy.arange(longueur_bloc)*te_r
u = numpy.zeros(longueur_bloc)

line0, = ax.plot(t,u)
ax.grid()
ax.axis([0,duree,-Umax,Umax])
ax.set_xlabel("t (s)")
u = numpy.array([],dtype=numpy.float32)
t = numpy.array([],dtype=numpy.float32)

def animate(i):
    global ax,sys,t,u,line0,n_tot,longueur_bloc
    data = sys.paquet_filtrees(n_tot,reduction)
    t0 = data[0]
    u0 = data[1]
    n_tot += u0.size
    t = numpy.append(t,t0)
    u = numpy.append(u,u0)
    if n_tot >= nombre_echant:
        print(u"Acquisition terminée")
    i2 = n_tot-1
    if n_tot == 0:
        return
    if n_tot > longueur_bloc:
        i1 = i2-longueur_bloc
    else:
        i1 = 0
    line0.set_data(t[i1:i2],u[i1:i2])
    ax.axis([t[i1],t[i2],-Umax,Umax])

interval = 0.1 # intervalle de rafraichissement en s
ani = animation.FuncAnimation(fig,animate,frames=int(duree_totale/interval),repeat=False
show(block=True)
numpy.savetxt("data.txt",[t,u])
sys.fermer()
```

3. Acquisition sans fin en flux continu

Une acquisition sans fin en flux continu est lancée par :

```
sys.lancer_permanent(repetition=1)
```

Dans ce cas, les échantillons sont mémorisés dans un tampon circulaire. Le nombre de points définis par `config_echantillon_permanent` définit la taille des paquets lus par la fonctions `paquet`. Ce mode d'acquisition est utile lorsqu'on veut visualiser et analyser un signal en temps réel avec une fenêtre d'analyse de taille N, sans chercher à mémoriser tous les échantillons.

Le script suivant montre comment procéder pour fair une acquisition sans fin. Le tracé du signal, du signal filtré et de son spectre est fait dans l'animation lancée en parallèle. L'acquisition est faite avec un filtrage passe-bas, ce qui permet de voir en temps réel l'effet du filtre sur un signal délivré par un générateur de fonction, ou par la carte son de l'ordinateur.

La fonction `paquet`, appelée dans la boucle d'animation, renvoie le premier paquet disponible dans un tampon circulaire comportant 16 paquets. Si aucun paquet n'est disponible, la fonction renvoie un tableau vide. Il faut donc tester la taille de ce tableau pour savoir s'il y a des nouvelles données à traiter.

acquisitionSansFinFiltrage.py

```
import pycanum.main as pycan
import math
from matplotlib.pyplot import *
import matplotlib.animation as animation
import numpy
import scipy.signal
import time
import cmath


sys=pycan.Sysam("SP5")
Umax = 10.0
sys.config_entrees([0],[Umax])
fe=40000.0 # fréquence de la numérisation
te=1.0/fe
N = 1000 # nombre d'échantillons dans la liste circulaire (fenêtre d'analyse)
duree = N*te
print(u"Durée de la fenêtre = %f s"%(duree))
sys.config_echantillon_permanent(te*1e6,N)
reduction = 1 # réduction de la fréquence d'échantillonnage après filtrage
fe_r = fe/reduction
te_r = te*reduction
longueur_bloc = int(N/reduction)


#filtre passe-bas
fc = 1000 # fréquence de coupure
b = scipy.signal.firwin(numtaps=40,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()
plot(w/(2*math.pi)*fe,g)
xlabel("f (Hz)")
ylabel("G")
grid()
figure()
plot(w/(2*math.pi)*fe,phase)
xlabel("f (Hz)")
ylabel("phi")
grid()
show() # tracé de la réponse fréquentielle du filtre
sys.config_filtre([1],b)

sys.lancer_permanent(repetition=1) # lancement d'une acquisition sans fin
```

```
fig0,ax0 = subplots()
t = numpy.arange(longueur_bloc)*te_r
u = numpy.zeros(longueur_bloc)
line0, = ax0.plot(t,u,'r')
line1, = ax0.plot(t,u,'b')
ax0.grid()
ax0.axis([0,duree,-Umax,Umax])
ax0.set_xlabel("t (s)")

fig1,ax1 = subplots()
freq = numpy.arange(longueur_bloc)*1.0/duree
A = numpy.zeros(longueur_bloc)
line2, = ax1.plot(freq,A)
ax1.grid()
ax1.axis([0,fe/reduction,0,1.0])
ax1.set_xlabel("f (Hz)")

def animate0(i): # tracé du signal
    global sys,line0,data,u
    data = sys.paquet(-1,reduction)
    if data.size!=0:
        u = data[1]
        line0.set_ydata(u)
        line1.set_ydata(data[2])

def animate1(i): # tracé du spectre
    global line1,u
    if u.size==longueur_bloc:
        A = numpy.absolute(numpy.fft.fft(u))/longueur_bloc
        line2.set_ydata(A)

ani0 = animation.FuncAnimation(fig0,animate0,frames=100,repeat=True,interval=duree*1000)
ani1 = animation.FuncAnimation(fig1,animate1,frames=100,repeat=True,interval=duree*1000)
show(block=True)
sys.stopper_acquisition()
time.sleep(1)
sys.fermer()
```