TP6 - Filtre numérique anti-repliement

1. Introduction

On se propose de mettre au point un logiciel d'analyse spectrale pour un microcontrôleur (de type Arduino). Il s'agit de déterminer la fréquence d'un signal (issu d'un capteur) dont la valeur est de l'ordre de quelques dizaines de Hertz mais inférieure à $100\,\mathrm{Hz}$. En raison de la très faible quantité de mémoire RAM disponible sur le microcontrôleur, le nombre d'échantillons qui sera utilisé pour le calcul du spectre est fixé à $N_e = 256$.

Afin d'assurer une bonne précision de la fréquence obtenue, la durée $T = N_e T_e$ de la fenêtre analysée doit être grande, ce qui pose problème compte tenu de la faible valeur de N_e. Il faut donc opérer à une fréquence d'échantillonnage faible mais convenable pour analyser des signaux de fréquence inférieure à $100\,\mathrm{Hz}$. Afin de respecter la condition de Nyquit-Shannon, on choisit $f_e = 200\,\mathrm{Hz}$. L'inconvénient d'une fréquence d'échantillonnage aussi faible est le risque de repliement si le signal possède des composantes de fréquences supérieures à $100\,\mathrm{Hz}$, en particulier si le signal comporte un bruit important (ce qui n'est pas rare dans les signaux provenant des capteurs).

Ce problème est résolu par une technique consistant à sur-échantillonner à une fréquence 5 fois plus grande, soit $f'_e = 1000\,\mathrm{Hz}$, puis à appliquer un filtre passe-bas numérique (appelé filtre anti-repliement) avant de réduire la fréquence d'échantillonnage à $f_e = 200\,\mathrm{Hz}$ pour le stockage dans un tableau de taille $N_e = 256$ et le calcul de la transformée de Fourier discrète.

Matériel :

- ▷ Plaque d'essai.
- ▷ ALI TL081.
- ▷ Fils avec emboûts dénudés.
- ▷ 3 résistances de $10\,\mathrm{k}\Omega$ ou 3 résistances d'une autre valeur supérieure.
- ▷ Générateur de signaux.
- ▷ Oscilloscope.
- ▷ Carte d'acquisition Sysam SP5.

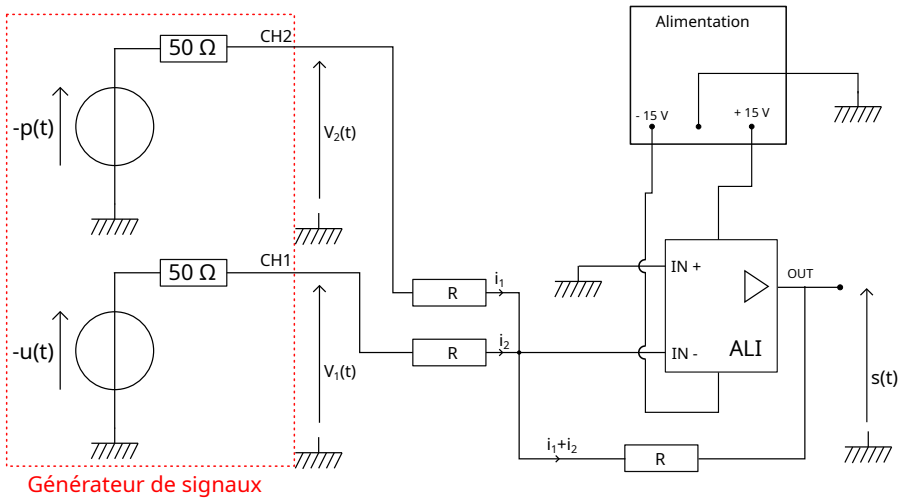
2. Réalisation du signal

On souhaite numériser et analyser un signal comportant une partie utile $u(t)$, dont le spectre s'étend au maximum jusqu'à $100\,\mathrm{Hz}$ et une perturbation $p(t)$, dont le spectre contient des composantes de fréquences supérieures à $100\,\mathrm{Hz}$:

$$s(t) = u(t) + p(t) \tag{1}$$

Le signal utile $u(t)$ sera délivré par la voie 1 du générateur de signaux, la perturbation $p(t)$ sera délivrée par la voie 2 du même générateur.

Pour réaliser l'addition des deux signaux, on utilise le circuit actif suivant, qui comporte un amplificateur linéaire intégré (ALI) fonctionnant en régime linéaire :



Les trois résistances notés R sont égales (aux incertitudes près) et sont supérieures à $10\,\mathrm{k\Omega}$. Le générateur délivre $-u(t)$ sur la voie CH1 et $-p(t)$ sur CH2.

Pour étudier le fonctionnement de ce circuit, on utile le modèle d'ALI idéal : les courants entrant dans les entrées IN+ et IN- sont négligeables et on a $V_+ = V_-$.

[1] Justifier que $i_1(t) \approx -\frac{u(t)}{R}$ et $i_2(t) \approx -\frac{p(t)}{R}$.

[2] Démontrer la relation (1).

[3] Réaliser le circuit. Visualiser $s(t)$ sur la voie 1 de l'oscilloscope.

[4] Tester le circuit avec $u(t)$ sinusoïdal de fréquence $27{,}15\,\mathrm{Hz}$ et d'amplitude $5\,\mathrm{V}$, et $p(t)$ sinusoïdal de fréquence $150\,\mathrm{Hz}$ et d'amplitude $0{,}5\,\mathrm{V}$.

3. Conception du filtre numérique

[5] La fréquence d'échantillonnage finale étant $f_e = 200\,\mathrm{Hz}$, quelle doit être la fréquence de coupure du filtre numérique anti-repliement ?

Le filtre passe-bas est un filtre de convolution. Les coefficients b_k du filtre sont calculés avec la fonction `scipy.signal.firwin` de la manière suivante

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import firwin,freqz
fep = # à compléter
fc = # à compléter
P=40
b = firwin(numtaps=2*P+1,cutoff=[fc/fep],window='hamming',nyq=0.5)
```

La réponse fréquentielle est obtenue avec la fonction `scipy.signal.freqz` :

```
w,h=freqz(b)

plt.figure()
plt.subplot(211)
plt.plot(w/(2*np.pi)*fe,np.absolute(h))
plt.ylabel("G")
```

```
plt.grid()
plt.subplot(212)
plt.plot(w/(2*np.pi)*fe,np.unwrap(np.angle(h)))
plt.xlabel("f (Hz)")
plt.ylabel("déphasage")
plt.grid()
```

[6] Sur le Jupyter Hub, créer un notebook intitulé `FiltreAntiRepliement` dans le dossier `TP/TP6`.

[7] Calculer ce filtre. Tracer sa réponse impulsionnelle avec la fonction `stem`. Tracer sa réponse fréquentielle et vérifier qu'elle correspond bien aux spécifications recherchées.

4. Numérisation et filtrage

Le signal utilise $u(t)$ est sinusoïdal de fréquence $27{,}15\,\text{Hz}$ et d'amplitude $5\,\text{V}$. On étudiera trois perturbations :

- ▷ Signal 1 : $p(t)$ sinusoïdal de fréquence $150\,\text{Hz}$ et d'amplitude $0{,}5\,\text{V}$.
- ▷ Signal 2 : $p(t)$ en dents de scie de fréquence $150\,\text{Hz}$ et d'amplitude $0{,}5\,\text{V}$.
- ▷ Signal 3 : $p(t)$ bruit blanc d'amplitude $0{,}5\,\text{V}$.

Le traitement et l'analyse spectrale de ces trois signaux seront faits dans un notebook Python sur le Jupyter Hub.

Pour numériser le signal $s(t)$ et l'enregistrer dans un fichier NPY, on utilise le script suivant (à exécuter sur le PC) :

acquisition.py

```
import numpy as np
import matplotlib.pyplot as plt
import pycanum.main as pycan
can = pycan.Sysam("SP5")
Vmax = 10.0
can.config_entrees([0],[Vmax])
fe = 1000
te = 1/fe
N = 256*5
can.config_echantillon(te*1e6,N)
can.acquerir()
t=can.temps()[0]
u0=can.entrees()[0]
can.fermer()
# modifier le nom du fichier fichier pour chaque expérience
np.save('signal-1.npy',np.array([t,u0]))
plt.figure()
plt.plot(t,u0)
plt.grid()
plt.xlabel('t (s)')
plt.ylabel('u (V)')
plt.xlim(0,1e-1)
plt.show()
```

[8] Exporter le fichier NPY vers le Jupyter Hub. Dans le notebook, récupérer le tableau du temps et le signal échantillonné par : `[t,x]=np.load('signal-1.npy')`.

[9] Réaliser le filtrage au moyen de la fonction `filtrage_convolution` mise au point au TP précédent. Tracer sur la même figure le signal `x` et le signal filtré `y`.

La réduction de la fréquence d'échantillonnage d'un facteur 5 consiste à garder un échantillon sur 5, avec la syntaxe de tranche suivante :

```
x_reduit = x[::5]
y_reduit = y[::5]
t_reduit = t[::5]
```

[10] Réaliser la réduction de la fréquence d'échantillonnage sur le signal filtré `y` et sur le signal non filtré `x`. Comparer les deux signaux après réduction.

5. Analyse spectrale

[11] Déterminer la résolution fréquentielle de l'analyse spectrale.

[12] Calculer la transformée de Fourier discrète de `x_reduit` et de `y_reduit` avec la fonction `numpy.fft.fft`. Générer le tableau des fréquences (`freq`) puis tracer les spectres avec la fonction `plot`, avec une échelle d'amplitude en décibel (tracer les spectres dans deux figures différentes mais à la même échelle).

[13] Pour le signal 1, comparer les deux spectres. Le filtre anti-repliement remplit-il sa fonction ?

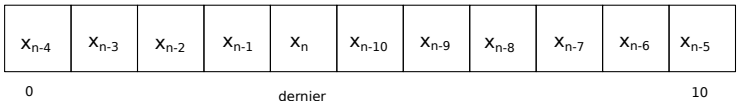
[14] L'analyse spectrale confirme-t-elle la fréquence du signal $u(t)$ programmée sur le générateur ?

[15] Faire le filtrage et l'analyse pour les signaux 2 et 3. On présentera l'analyse des trois signaux dans le notebook les unes à la suite des autres. Pour chaque signal, commenter le spectre du signal non filtré et celui du signal filtré.

6. Filtrage et réduction simultanés

Le microcontrôleur de l'Arduino ne possède pas assez de mémoire pour stocker le signal échantillonné à 1000 Hz. Il comporte seulement la mémoire nécessaire pour stocker 256 échantillons et faire le calcul de la TFD. En conséquence, il faut effectuer en même temps le filtrage et la réduction de la fréquence d'échantillonnage.

Le filtrage nécessite de garder en mémoire les $2P+1$ derniers échantillons. On utilise pour cela une liste circulaire. Il s'agit d'un tableau à $2P+1$ éléments que l'on remplit avec les échantillons au fur et à mesure de leur numérisation. Lorsque la liste est pleine, on revient au début de la liste. En conséquence, le dernier échantillon (x_n) se trouve à un indice variable noté `dernier`. Lorsque `dernier` atteint la valeur $2P+1$, on doit le remettre à zéro. Lors du calcul de la convolution, il faut parcourir la liste par indice décroissant, en commençant par `dernier`, sans oublier de passer à $2P$ lorsqu'on parvient à la valeur -1. La figure suivante représente une liste circulaire à 11 éléments ($P=5$), avec le dernier échantillons à l'indice `dernier=4` :



Lorsque l'échantillon suivant arrive, il prend la place de x_{n-10} et `dernier` devient égal à 5.

L'algorithme de filtrage et réduction est le suivant :

- ▷ Obtention des échantillons à la fréquence d'échantillonnage de $1000\,\mathrm{Hz}$ et ajout de ces échantillons dans la liste circulaire au fur et à mesure de leur numérisation.
- ▷ Tous les 5 échantillons, calcul de y_n par la formule de convolution :

$$y_n = \sum_{k=0}^{2P} b_k x_{n-k}$$

- ▷ Stockage du résultat dans le tableau de taille $N_e = 256$.

[16] Écrire une fonction `filtrage(liste_circulaire,b,dernier)` qui calcule et renvoie le produit de convolution à partir des échantillons présents dans la liste circulaire.

Afin de simuler le fonctionnement du logiciel qui sera implanté sur l'Arduino, on écrit un code qui lit séquentiellement les échantillons présents dans le tableau `x`, les place dans la liste circulaire et, tous les 5 échantillons, appelle la fonction `filtrage`. La valeur renvoyée par cette fonction est placée dans le tableau final de taille $N_e = 256$. Lorsque ce tableau est plein, on calcule la TFD puis on extrait les fréquences des sinusoïdes présentes dans le signal.

[17] Écrire ce code et le tester.