Interface Python pour convertisseurs A/N Eurosmart

Frédéric Legrand

13 mars 2022

Table des matières

1 Installation et fonctions de l'interface **3**
1.1. Installation . . . 4
1.2. Fonctions de l'interface . . . 6
1.2.a. Ouverture d'une liaison avec le CAN . . . 6
1.2.b. Configuration des entrées analogiques . . . 6
1.2.c. Utilisation des entrées analogiques avec échantillonnage . . . 7
1.2.d. Utilisation des sorties analogiques avec échantillonnage . . . 9
1.2.e. Utilisation simultanée des entrées et sorties . . . 10
1.2.f. Lecture des entrées sans échantillonnage . . . 10
1.2.g. Écriture directe sur les sorties . . . 10
1.2.h. Acquisition en mode parallèle (SysamSP5) . . . 11
1.2.i. Filtrage numérique (SysamSP5) . . . 12
1.2.j. Acquisition en mode permanent (Sysam SP5) . . . 12
1.2.k. Ports logiques (Sysam SP5) . . . 13
1.2.l. Compteur et chronomètre (Sysam SP5) . . . 15
1.2.m. Synthèse de signaux périodiques sur la sortie audio . . . 16
1.2.n. Sortie audio d'un signal . . . 17

2 Enregistrement d'un signal **19**
2.1. Dispositif expérimental . . . 20
2.2. Acquisition . . . 20
2.3. Tracé du signal . . . 21

3 Déclenchement de l'acquisition **22**
3.1. Déclenchement par une des voies acquises . . . 23
3.2. Déclenchement par un signal externe TTL . . . 24

4 Acquisition point par point **26**
4.1. Introduction . . . 27
4.2. Lecture directe . . . 27
4.3. Lecture par paquet échantillonné . . . 29

5 Utilisation des sorties **32**
5.1. Génération d'un signal échantillonné sur une sortie . . . 33
5.2. Utilisation des entrées et sorties (SysamSP5) . . . 34
5.3. Utilisation simultanée des entrées et sorties . . . 35

6 Diagramme de Bode **37**
6.1. Introduction . . . 38
6.2. Dispositif expérimental . . . 38
6.3. Méthodes numériques . . . 38
6.3.a. Génération de la tension d'entrée . . . 38
6.3.b. Traitement des signaux . . . 40
6.4. Programme complet . . . 41
6.5. Exemple . . . 45

7 Caractéristique d'un dipôle **48**
7.1. Obtention d'une caractéristique à faible courant . . . 49
7.2. Obtention d'une caractéristique à courant moyen . . . 52

8 Filtrage numérique d'un signal **53**
8.1. Introduction . . . 54
8.2. Conversion analogique-numérique . . . 54
8.3. Calcul du filtre passe-bas . . . 56
8.4. Filtrage numérique . . . 58
8.5. Conversion numérique-analogique . . . 60
8.6. Lissage numérique . . . 61

9 Acquisition et traitement parallèles **64**
9.1. Introduction . . . 65
9.2. Configuration des entrées . . . 65
9.3. Tracé des signaux en mode parallèle . . . 65
9.4. Filtrage en mode parallèle . . . 66

10 Acquisition en mode permanent **68**
10.1. Introduction . . . 69
10.2. Acquisition d'un grand nombre d'échantillons . . . 69
10.2.a. Principe . . . 69
10.2.b. Sur-échantillonnage et filtrage . . . 70
10.2.c. Acquisition avec tracé du signal . . . 72
10.3. Acquisition sans fin en flux continu . . . 74

11 Filtrage en temps réel d'un signal audio **77**
11.1. Introduction . . . 78
11.2. Montage expérimental . . . 78
11.3. Filtrage RIF . . . 78
11.4. Filtre récursif biquadratique . . . 81

Chapitre 1

Installation et fonctions de l'interface

1.1. Installation

Ce document présente une interface des convertisseurs analogique/numérique (CAN) SysamSP5 (port USB) et SysamPCI (carte PCI) de Eurosmart pour Python.

L'interface fonctionne sous Windows (XP,7,8 et 10) avec Python 32 bits. Il n'y a pas de version pour Python 64 bits car les pilotes fournis par Eurosmart sont en 32 bits.

Elle est distribuée avec le code source sous licence CeCILL.

Distribution par PyPI (pypi.org/project/pycanum/) pour python (win32) 3.5, 3.6, 3.7 et 3.8. Pour l'installer, ouvrir une console et exécuter les commandes suivantes dans le répertoire Scripts de la distribution (après avoir exécuté `activate.bat`) :

```
pip install --upgrade pip
pip install pycanum
```

Pour faire une mise à jour :

```
pip install --upgrade pycanum
```

Distribution recommandée pour Python : Anaconda 3 32 bits. Exécuter `pip install pycanum` dans le répertoire `C:\Anaconda3\Scripts`.

Exécutables d'installation pour python win32 :

- ▷ pycanum-4.2.7.win32-py2.7.exe
- ▷ pycanum-4.2.7.win32-py3.5.exe
- ▷ pycanum-4.2.7.win32-py3.6.exe
- ▷ pycanum-4.2.7.win32-py3.7.exe
- ▷ pycanum-4.2.7.win32-py3.8.exe

Fichiers wheel pour installation par `pip install fichier.whl` :

- ▷ pycanum-4.2.7-cp35-cp35m-win32.whl
- ▷ pycanum-4.2.7-cp36-cp36m-win32.whl
- ▷ pycanum-4.2.7-cp37-cp37m-win32.whl
- ▷ pycanum-4.2.7-cp38-cp38-win32.whl

Code source : pycanum-4.2.7.zip

Version 4.2.7

Correction d'un bug dans la configuration du trigger.

Version 4.2.6 :

Correction d'une erreur sur la configuration des voies en différentiel pour les calibres 1 et 0.2.

Ajout d'une option pour réduire la profondeur de numérisation.

Version 4.2.4 :

Envoi de deux signaux numériques sur la sortie audio.
Scripts de démonstration :

- Envoi d'un signal numérique sur la sortie audio
- Acquisition en continu avec filtrage et sortie audio (filtre RIF)
- Acquisition en continu avec filtrage et sortie audio (filtre récursif biquad)

Version 4.2

Prise en compte des données de calibrage de la carte SysamSP5. Cette mise à jour est importante pour obtenir les conversions analogique-numérique et numérique-analogique les plus précises (LSB en 12 bits). En contre-partie, seuls les calibes 10, 5, 1 et 0.2 volts sont accessibles pour les entrées analogiques.

Version 4.1 :

- Fonctions de génération de signaux périodiques sur la sortie audio (2 voies, synthèse directe par table avec accumulateur de phase).

Version 4.0

- Acquisition en mode permanent, permettant de dépasser les limites de la mémoire de la carte.
- Acquisition en mode permament en boucle, pour l'analyse et le traitement en temps réel.
- Filtrage numérique pendant l'acquisition (filtres RIF et RII).
- Réduction de la fréquence d'échantillonnage lors de la récupération des tensions.
- Compteur et chronomètre.

Scripts de démonstration (pour Sysam SP5) :

- Acquisition sur une voie et analyse spectrale
- Acquisition sur deux voies et analyse spectrale
- Acquisition avec déclenchement
- Synthèse d'un signal périodique sur une sortie
- Acquisition avec utilisation simultanée d'une sortie
- Acquisition d'un signal, filtrage puis restitution en sortie
- Boucle de filtrage d'un signal
- Acquisition et tracé en parallèle
- Acquisition en mode permanent avec tracé du signal
- Acquisition en continu avec tracé du signal et de son spectre
- Génération de deux signaux périodiques sur la sortie audio et acquisition

- ▷ Génération de deux signaux périodiques sur la sortie audio (définis par leurs harmoniques)

Pages de documentation avec exemples :

- ▷ Enregistrement d'un signal
- ▷ Déclenchement de l'acquisition
- ▷ Acquisition point par point
- ▷ Utilisation des sorties
- ▷ Mesure de fréquence
- ▷ Diagramme de Bode
- ▷ Caractéristique d'un dipôle
- ▷ Filtrage numérique d'un signal
- ▷ Acquisition et traitement en parallèle
- ▷ Acquisition en mode permanent
- ▷ Filtrage d'un signal audio en temps réel

Pour installer le pilote de la centrale SysamSP5, télécharger le fichier zip sur Eurosmart et exécuter le programme d'installation. Remarque : sous windows 10 64 bits, on exécute DPInst64.exe mais le pilote installé est en 32 bits. La DLL SP5.dll se trouve dans le dossier `c:\Windows\SysWOW64`, qui contient les DLL 32 bits sur un windows 64 bits.

1.2. Fonctions de l'interface

1.2.a. Ouverture d'une liaison avec le CAN

L'interface se trouve dans le module `pycanum.main`. Elle est constituée d'une classe `Sysam`.

L'ouverture d'une liaison avec un convertisseur se fait en construisant un objet de la classe `Sysam`.

```
sys = Sysam(nom)
```

Constructeur : ouverture de la liaison avec le CAN

- ▷ `nom (string)` : nom du convertisseur (SP5 ou PCI)

```
Sysam.fermer()
```

Fermeture de la liaison

Attention : un seul objet de la classe Sysam peut être ouvert dans un programme Python. Il faut donc fermer la liaison avant de créer un nouvel objet Sysam.

1.2.b. Configuration des entrées analogiques

```
Sysam.config_entrees(voies,calibres,diff=[])
```

Sélection des entrées analogiques et configuration du calibre (gain de l'amplificateur d'entrée).

- ▷ `voies (list)` : liste des entrées analogiques à sélectionner, numérotées de 0 à 7.
- ▷ `calibres (list)` : liste des valeurs absolues maximales des tensions (en volts), une pour chaque voie sélectionnée.
- ▷ `diff (list)` : argument optionnel, liste des voies en mode différentiel.

Sur la carte SysamPCI, la fréquence d'échantillonnage maximale diminue lorsqu'on augmente le nombre de voies sélectionnées. Sur la centrale SysamSP5, il faut sélectionner les entrées 0,1,2 et 3 pour bénéficier de la fréquence d'échantillonnage maximale (10 MHz).

Sur SysamPCI, la sélection d'une seule voie en mode différentiel les place toutes en mode différentiel. Les tensions sont alors mesurées entre les entrées 0 et 4, 1 et 5, 2 et 6, 3 et 7. Sur SysamSP5, chaque canal (0-4,1-5,2-6,3-7) peut être placé en mode différentiel indépendamment des autres. Pour placer le premier et le deuxième canal en mode différentiel, il faut affecter [0,1] au dernier argument (diff).

1.2.c. Utilisation des entrées analogiques avec échantillonnage

Les fonctions décrites ci-dessous permettent d'effectuer des acquisitions échantillonnées sur les entrées analogiques.

```
Sysam.config_echantillon(techant,nbpoints)
```

Configuration de la période d'échantillonnage et du nombre de points à acquérir.

- ▷ `techant` : période d'échantillonnage en microsecondes
- ▷ `nbpoints` : nombre de points à acquérir

La période d'échantillonnage effectivement utilisée est multiple de la période d'horloge du convertisseur. Le nombre de points est limité par la capacité mémoire de la carte.

Le convertisseur de la carte Sysam effectue une numérisation en 12 bits mais l'interface permet de réduire le nombre de bits utilisés pour le calcul de la tension (les bits les moins significatifs sont éliminés par un décalage à droite) :

```
Sysam.config_quantification(nbits)
```

Configuration du nombre de bits de la quantification.

- ▷ `nbits` : nombre de bits utilisés pour la quantification (inférieur ou égal à 12).

```
Sysam.config_trigger(voie,seuil,montant=1,pretrigger=0,pretriggerSouple=0,hyste
```

Configuration d'un déclenchement sur une des voies de l'acquisition.

- ▷ `voie (integer)` : voie sur laquelle se fait le déclenchement (une des voies configurées).
- ▷ `seuil (float)` : seuil du déclenchement en volts.
- ▷ `montant (integer)` : 1 pour un déclenchement sur front montant (par défaut), 0 pour un déclenchement sur front descendant.
- ▷ `pretrigger (integer)` : option pour SysamSP5, nombre de points enregistrés avant la condition de déclenchement.
- ▷ `pretriggerSouple (integer)` : option pour SysamSP5. 1 si le déclenchement doit se faire lorsque le nombre de points de pretrigger n'est pas atteint.
- ▷ `hystereris (integer)` : option pour SysamSP5. Hystérésis du déclenchement.

Attention : la valeur `pretrigger=0` ne conduit pas au déclenchement attendu. Il faut choisir au minimum `pretrigger=1`.

```
Sysam.config_trigger_externe(pretrigger=0,pretriggerSouple=0)
```

Configuration d'un déclenchement sur une voie externe TTL (entrée SYNCHRO EXT).

- ▷ `pretrigger (integer)` : option pour SysamSP5, nombre de points enregistrés avant la condition de déclenchement.
- ▷ `pretriggerSouple (integer)` : option pour SysamSP5. 1 si le déclenchement doit se faire lorsque le nombre de points de pretrigger n'est pas atteint.

```
Sysam.desactiver_trigger()
```

Désactivation du déclenchement par un signal (voie acquise ou externe).

```
Sysam.acquerir()
```

Acquisition des tensions sur les entrées configurées. La fonction retourne lorsque l'acquisition est terminée.

```
temps = Sysam.temps(reduction=1)
```

Récupération des temps de la dernière acquisisition.

- ▷ `reduction (integer)` : facteur de réduction de la fréquence d'échantillonnage.

- ▷ `temps (ndarray)` : tableau ndarray (numpy). Chaque ligne du tableau fournit les temps (en s) échantillonnés de la voie correspondante.

Remarque : sur SysamPCI, les temps des différentes voies sont décalés.

```
tensions = Sysam.entrees(reduction=1)
```

Récupération des tensions de la dernière acquisition.

- ▷ `reduction (integer)` : facteur de réduction de la fréquence d'échantillonnage.
- ▷ `tensions (ndarray)` : tableau ndaray (numpy). Chaque ligne du tableau fournit les tensions (en V) de la voie correspondante.

Un facteur de réduction égal à 2 permet d'obtenir un échantillon sur 2.
Les temps et les tensions de la première voie sélectionnée sont `temps[0]` et `tensions[0]`.

1.2.d. Utilisation des sorties analogiques avec échantillonnage

Les fonctions décrites ci-dessous permettent d'émettre un signal échantillonné sur la sortie SA1 (SysamPCI), ou sur les deux sorties SA1 et SA2 (SysamSP5). Pour une utilisation simultanée des entrées, voir le paragraphe suivant.

```
Sysam.config_sortie(nsortie,techant,tensions,repetition=0)
```

Configuration d'une sortie

- ▷ `nsortie (integer)` : numéro de la sortie (1 ou 2)
- ▷ `techant (float)` : période d'échantillonnage en microsecondes
- ▷ `tensions (ndarray)` : tableau ndarray (numpy) fournissant le signal échantillonné à appliquer sur la sortie (valeurs en volts).
- ▷ `repetition (integer)` : nombre de répétitions du signal, -1 pour une répétition périodique.

Sur la carte SysamSP5, l'argument `repetition` est égal à -1 ou 0. Pour répéter 2 fois ou plus un signal donné, il faut donc programmer cette répétition dans le signal échantilonné fourni.

```
Sysam.declencher_sorties(s1,s2)
```

Déclenchement d'une ou deux sorties. Les deux sorties (SysamSP5) sont déclenchées simultanément. Pour qu'elles soient synchrones, il faut néanmoins que les périodes d'échantillonnage soient identiques. Sur SysamSP5, la fonction retourne immédiatement. Sur SysamPCI, elle retourne lorsque l'émission est terminée ou lorsque l'utilisateur presse la touche ESC.

- ▷ `s1 (integer)` : 1 pour déclencher la sortie 1, 0 sinon.
- ▷ `s2 (integer)` : 1 pour déclencher la sortie 2, 0 sinon.

Remarque : sur SysamSP5, la fonction retourne immédiatement ; il faut donc éventuellement prévoir une mise en attente avant d'effectuer d'autres opérations.

```
Sysam.stopper_sorties(s1,s2)
```

Stopper une émission périodique sur SysamSP5. Sur SysamPCI, il faut appuyer sur la touche ESC pour stopper l'émission.

- ▷ `s1 (integer)` : 1 pour stopper la sortie 1, 0 sinon.
- ▷ `s2 (integer)` : 1 pour stopper la sortie 2, 0 sinon.

Sur SysamSP5, il faut stopper les sorties avant de les configurer (si elles sont déjà en fonctionnement). Ne pas le faire peut provoquer des erreurs dans le remplissage de la mémoire du convertisseur numérique-analogique.

Sur SysamSP5, il est possible de déclencher une acquisition lorsque les sorties sont en fonctionnement. Par exemple, si les sorties sont en fonctionnement périodique, une acquisition peut être déclenchée à tout moment. Dans ce cas, il faudra toutefois que le nombre de points total utilisé par les entrées et les sorties n'excède par 0x3FFFF, soit 262142. Dans le cas contraire, une erreur est déclenchée à l'appel de la fonction `Sysam.acquerir`.

1.2.e. Utilisation simultanée des entrées et sorties

Sur SysamSP5, il est possible de déclencher les sorties puis l'acquisition après (mais pas l'inverse). Sur SysamPCI, cela est impossible avec les fonctions décrites plus haut. La fonction suivante permet d'effectuer simultanément une acquisition et une émission des sorties :

```
Sysam.acquerir_avec_sorties(tensions1,tensions2)
```

Effectuer l'acquisition avec une utilisation simultanée et synchrone des sorties. La période d'échantillonnage des sorties est la même que pour les entrées.

- ▷ `tensions1 (ndarray)` : tableau ndarray (numpy) fournissant le signal échantillonné à appliquer sur la sortie 1 (en volts).
- ▷ `tensions2 (ndarray)` : tableau ndarray (numpy) fournissant le signal échantillonné à appliquer sur la sortie 2 (en volts).

Remarque : le nombre de points total utilisable pour les entrées et sorties est limité par la mémoire RAM du SysamSP5. Il est de 0x3FFFF, soit 262142.

1.2.f. Lecture des entrées sans échantillonnage

La lecture directe des entrées (sans passer par la mémoire du CAN) peut être utilisée lorsque la cadence d'acquisition est très lente (moins de un échantillon par seconde). Dans ce cas, on utilisera l'horloge interne du PC pour déclencher des lectures périodiques (fonction time.sleep).

```
Sysam.activer_lecture(voies)
```

Activer la lecture directe sur une ou plusieurs voies, qui doivent avoir été configurées au préalable avec `config_entrees`.

▷ `voies (list)` : liste des voies.

```
tensions = Sysam.lire()
```

Lecture directe des tensions sur les entrées.

▷ `tensions (list)` : liste des tensions lues.

```
Sysam.desactiver_lecture()
```

Désactivation de la lecture directe.

1.2.g. Écriture directe sur les sorties

```
Sysam.ecrire(s1,tension1,s2,tension2)
```

Appliquer une tension constante sur une ou deux sorties

▷ `s1 (integer)` : 1 pour changer la tension sur la sortie 1, 0 sinon.

▷ `tension1 (float)` : tension à appliquer sur la sortie 1 en volts.

▷ `s2 (integer)` : 1 pour changer la tension sur la sortie 2, 0 sinon.

▷ `tension2 (float)` : tension à appliquer sur la sortie 2 en volts.

1.2.h. Acquisition en mode parallèle (SysamSP5)

Lorsque le temps d'acquisition est long (supérieur à quelques secondes), il peut être utile de l'effectuer sur une tâche d'exécution parallèle (thread) afin d'effectuer un traitement des données simultanément. Les fonctions suivantes permettent d'effectuer une acquisition en mode parallèle, sur le CAN SysamSP5 seulement. Pour le CAN SysamPCI, le pilote ne permet pas ce mode d'acquisition.

```
Sysam.lancer()
```

Lancer l'acquisition en mode parallèle. La fonction retourne dès que l'acquisition est lancée.

```
Sysam.lancer_avec_sorties(tensions1,tensions2)
```

Lancer l'acquisition en mode parallèle, avec utilisation simultanée des sorties.

- ▷ `tensions1 (ndarray)` : tableau ndarray (numpy) fournissant le signal échantillonné à appliquer sur la sortie 1 (en volts).
- ▷ `tensions2 (ndarray)` : tableau ndarray (numpy) fournissant le signal échantillonné à appliquer sur la sortie 2 (en volts).

```
A = Sysam.paquet(premier,reduction=1)
```

Lire le paquet de points déjà acquis lors d'une acquisition en mode parallèle.

- ▷ `premier (integer)` : indice du premier point à récupérer, ou -1 pour obtenir un paquet lors d'une acquisition permanente en boucle.
- ▷ `reduction (integer)` : facteur de réduction de la fréquence d'échantillonnage.
- ▷ `A (ndarray)` : tableau ndarray (numpy) contenant les données. Si N est le nombre de voies acquises, les N premières lignes sont les instants de ces N voies, les N lignes suivantes sont les tensions numérisées, et les N dernières lignes sont les tensions numérisées et filtrées.

1.2.i. Filtrage numérique (SysamSP5)

Les tensions lues sur les entrées analogiques sont filtrées pendant l'acquisition. La relation de récurrence appliquée est la suivante :

$$a_0 y_n = \sum_{k=0}^{N-1} b_k x_{n-k} - \sum_{k=1}^{M-1} a_k y_{n-k} \quad (1.1)$$

Pour définir un filtre à réponse impulsionnelle finie (RIF), on doit définir les coefficients a_k. Pour définir un filtre à réponse impulsionnelle infinie (RII), il faut aussi définir les coefficients b_k.

```
Sysam.config_filtre(a,b)
```

Définition des coefficients de filtrage

- ▷ `a (list)` : liste (ou tableau ndarray) définissant les coefficients a.
- ▷ `b (list)` : liste (ou tableau ndarray) définissant les coefficients b.

Si cette fonction n'est pas appelée, le filtre par défaut est défini par $a_0 = 1, b_0 = 1$ (filtre identité).

Pour obtenir les valeurs des tensions filtrées, on utilise la fonction suivante :

```
tensions = Sysam.entrees_filtrees(reduction=1)
```

Récupération des tensions filtrées de la dernière acquisition.

- ▷ `reduction (integer)` : facteur de réduction de la fréquence d'échantillonnage.
- ▷ `tensions (ndarray)` : tableau ndaray (numpy). Chaque ligne du tableau fournit les tensions (en V) de la voie correspondante.

Les tensions non filtrées sont toujours accessibles avec la fonction `Sysam.entrees`.

La fonction `Sysam.paquet` décrite ci-dessus permet de récupérer les signaux filtrés avec les signaux non filtrés.

Une application typique de ces fonctions est le filtre anti-repliement numérique : on numérise avec un suréchantillonnage pour effectuer le filtrage numérique, puis on réduit la fréquence d'échantillonnage au moment de récupérer les tensions.

1.2.j. Acquisition en mode permanent (Sysam SP5)

Dans l'acquisition échantillonnée en mode permanent, la mémoire interne de la carte Sysam SP5 n'est pas utilisée. Les valeurs des tensions sont transmises au fur et à mesure de l'acquisition (avec un tampon interne à la carte de seulement 256 valeurs).

L'intérêt du mode permanent est de permettre l'acquisition d'un nombre de points supérieur à la capacité mémoire de la carte, par exemple plusieurs millions de points. En revanche, ce mode d'acquisition ne fonctionne pas si la fréquence d'échantillonnage est trop élevée (le maximum est entre 500 kHz et 1 MHz). Il faut faire des tests avec des signaux déterminés pour évaluer la limite exacte.

Le mode permanent permet aussi d'obtenir un flux continu d'échantillons, par exemple pour traiter un signal audio en temps réel.

```
Sysam.config_echantillon_permanent(techant,nbpoints)
```

Configuration de la période d'échantillonnage et du nombre de points à acquérir.

- ▷ `techant` : période d'échantillonnage en microsecondes
- ▷ `nbpoints` : nombre de points à acquérir (ou taille des paquets)

```
Sysam.acquerir_permanent()
```

Acquisition des tensions sur les entrées configurées, en mode permament. La fonction retourne lorsque l'acquisition est terminée.

```
Sysam.lancer_permanent(repetition=0)
```

Lancer l'acquisition en mode parallèle et mode permanent. La fonction retourne dès que l'acquisition est lancée.

- ▷ `repetition (integer)` : 1 pour que l'acquisition soit répétée dans fin.

Lorsque l'argument `repetition=1` est adopté, l'acquisition se fait en boucle et les paquets lus ont la taille définie par `config_echantillon_permanent`. Les paquets sont récupérés avec la fonction `paquet` en choisissant `premier=-1`. Les paquets contiennent les instants, les tensions non filtrées, et les tensions filtrées. Les paquets sont en fait stockés dans un tampon circulaire comportant 16 paquets. Lorsqu'aucun paquet n'est disponible, la fonction `paquet` renvoie un tableau vide. Une application de ce mode d'acquisition est le traitement d'un flux audio.

1.2.k. Ports logiques (Sysam SP5)

La centrale Sysam SP5 possède deux ports logiques de 8 bits chacun, accessibles par les connecteurs Sub-D.

Le port B fonctionne en TTL (tensions de 0 à 5 V). Il est constitué de deux paquets de 4 bits, chacun pouvant être programmé en entrée ou en sortie. Ce port bénéficie d'une protection contre les surtensions (max 12V). Le boitier BOLOGIC vendu par Eurosmart permet d'y accéder avec des douilles bananes. La fonction suivante permet de configurer le port B.

```
Sysam.portB_config(bits,etat)
```

Configuration d'un des paquets de 4 bits du port B en entrée ou en sortie.

- ▷ `bits (integer)` : 0 pour configurer les bits 0 à 4, 1 pour configurer les bits 4 à 7.
- ▷ `etat (integer)` : 0 pour configurer en entrée, 1 en sortie.

La fonction suivante permet d'écrire sur un bit configuré en écriture :

```
Sysam.portB_ecrire(bit,etat)
```

Écriture sur un bit du port B

- ▷ `bit (integer)` : numéro du bit (de 0 à 7).
- ▷ `etat (integer)` : 0 ou 1

La fonction suivante permet de lire sur un bit configuré en lecture :

```
b=Sysam.portB_lire(bit)
```

Lire un bit du port B

- ▷ `bit (integer)` : numéro du bit (de 0 à 7).
- ▷ `b (integer)` : valeur du bit (0 ou 1).

Remarque importante : les bits configurés en sortie se mettent automatiquement au niveau haut lorsqu'on ferme l'interface. Pour éviter cela, il suffit de configurer les bits en entrée avant de fermer l'interface.

Le port C fonctionne en CMOS (tensions de 0 à 3.3 V) et ne possède pas de protection contre les surtensions. Son utilisation est donc réservée à l'interfaçage avec des circuits CMOS. Chacun des 8 bits est programmable en entrée ou en sortie.

La fonction suivante configure le port C

```
Sysam.portC_config(bit,etat)
```

Configuration d'un des 8 bits du port C en entrée ou en sortie.

- ▷ `bit (integer)` : numéro du bit à configurer (0 à 7).
- ▷ `etat (integer)` : 0 pour configurer en entrée, 1 en sortie.

La fonction suivante permet d'écrire sur un bit configuré en écriture :

```
Sysam.portC_ecrire(bit,etat)
```

Écriture sur un bit du port C

- ▷ `bit (integer)` : numéro du bit (de 0 à 7).
- ▷ `etat (integer)` : 0 ou 1

La fonction suivante permet de lire sur un bit configuré en lecture :

```
b=Sysam.portC_lire(bit)
```

Lire un bit du port C

- ▷ `bit (integer)` : numéro du bit (de 0 à 7).
- ▷ `b (integer)` : valeur du bit (0 ou 1).

1.2.l. Compteur et chronomètre (Sysam SP5)

La centrale Sysam SP5 possède un compteur 48 bits avec une horloge de période 10 nanosecondes. L'entrée CHRONO peut être utilisée pour cela, ou bien une des entrées du port B accessibles par le connecteur PORT LOGIQUE. Ces entrées doivent recevoir un signal TTL (0/5 V).

L'utilisation en compteur consiste à compter le nombre de fronts sur une durée déterminée. Une application est la détermination de la fréquence d'un signal TTL.

```
Sysam.config_compteur(entree,front_montant,front_descend,hysteresis,duree)
```

Configuration du compteur

- ▷ `entree (integer)` : Numéro de l'entrée utilisée (0 à 7 pour le port B), ou `pycan.ENTREE_CHRONO` (valeur 16) pour l'entrée CHRONO.
- ▷ `front_montant (integer)` : 1 si les fronts montants sont comptés, 0 sinon.
- ▷ `front_descend (integer)` : 1 si les fronts descendants sont comptés, 0 sinon.
- ▷ `hysteresis (float)` : durée de l'hystérésis en microsecondes. Après un front compté, le compteur reste inactif pendant cette durée.
- ▷ `duree (float)` : durée de fonctionnement du compteur, en microsecondes.

La période de base de l'horloge étant de 10 ns, les durées effectives sont multiples de 10 ns.

```
Sysam.compteur()
```

Déclenchement du compteur.

```
n=Sysam.lire_compteur()
```

Attente de la fin du comptage et renvoi du nombre de fronts comptés.

- `n (integer)` : nombre de fronts comptés sous forme un entier 64 bits `numpy.uint64`.

L'utilisation en chronomètre consiste à compter le nombre de tops d'horloge entre deux fronts du signal TTL.

```
Sysam.config_chrono(entree,front_montant,front_descend,hysteresis)
```

Configuration du chronomètre

- `entree (integer)` : Numéro de l'entrée utilisée (0 à 7 pour le port B), ou `pycan.ENTREE_CHRONO` (valeur 16) pour l'entrée CHRONO.
- `front_debut (integer)` : front déclenchant le chronomètre (0 pour descendant, 1 pour montant).
- `front_fin (integer)` : front d'arrêt du chronomètre, (0 pour descendant, 1 pour montant).
- `hysteresis (float)` : durée de l'hystérésis en microsecondes. Après le front de démarrage, le détecteur de front reste inactif pendant cette durée.

```
Sysam.chrono()
```

Déclenchement du chronomètre.

```
n=Sysam.lire_chrono()
```

Attente de la fin du chronométrage et renvoi du nombre de tops d'horloge.

- `n (integer)` : nombre de tops d'horloges (10 ns) sous forme un entier 64 bits `numpy.uint64`.

Attention : l'entrée CHRONO ne fonctionne que sur front montant. Pour déclencher aussi sur front descendant, il faut utiliser une des entrées du port logique B.

1.2.m. Synthèse de signaux périodiques sur la sortie audio

Ces fonctions complémentaires permettent de générer deux signaux périodiques sur la sortie audio stéréo. La synthèse d'un signal se fait avec une table de 256 échantillons, par la méthode de l'accumulateur de phase. La fréquence d'échantillonnage par défaut est 44 kHz, mais il est possible de l'augmenter jusqu'à environ 300 kHz.

La méthode de synthèse directe par table (DDS) permet d'ajuster la fréquence avec une très grande précision. La précision en fréquence est :

$$\Delta f = \frac{f_e}{2^{32}}$$

Par exemple, pour une fréquence d'échantillonnage de 44 kHz, la précision est de 10^{-5} Hz.

L'amplitude de la sortie audio lorsque le volume est au maximum est d'environ 1,6 V.

```
audio_table_start(ch1,ch2,f1,f2,fs=44000,fpb=256)
```

Démarrage de la génération de deux signaux périodiques sur la sortie audio

- ▷ `ch1 (ndarray)` : table de 256 flottants compris entre -1.0 et 1.0 pour la voie 1
- ▷ `ch2 (ndarray)` : table de 256 flottants compris entre -1.0 et 1.0 pour la voie 2
- ▷ `f1 (float)` : fréquence pour la voie 1
- ▷ `f2 (float)` : fréquence pour la voie 2
- ▷ `fs (int)` : fréquence d'échantillonnage en Hz
- ▷ `fpb (int)` : nombre d'échantillons envoyés au contrôleur audio par trame

Les valeurs des tables supérieures à 1.0 ou inférieures à -1.0 sont écrêtées. La synchronisation des deux voies n'est garantie que si les deux fréquences sont identiques.

```
audio_table_stop(seconds)
```

Arrêt de la génération des signaux.

- ▷ `seconds` : délai en secondes avant l'arrêt

```
(table_1,table_2) = Sysam.audio_harmonics_start(a1,p1,a2,p2,f1,f2,norm=True,gai
```

Génération de deux signaux périodiques définis par des harmoniques

- ▷ `a1 (float list)` : amplitudes des harmoniques pour la voie 1
- ▷ `p1 (float list)` : phases des harmoniques pour la voie 1
- ▷ `a2 (float list)` : amplitudes des harmoniques pour la voie 2
- ▷ `p2 (float list)` : phases des harmoniques pour la voie 2
- ▷ `f1 (float)` : fréquence pour la voie 1
- ▷ `f2 (float)` : fréquence pour la voie 2

- ▷ `norm (boolean)` : True pour normaliser les tables (valeur max ou min ramenée à 1 ou -1)
- ▷ `gain (float)` : coefficient multiplicateur appliqué aux tables (à utiliser avec norm=True)
- ▷ `fs (int)` : fréquence d'échantillonnage en Hz
- ▷ `fpb (int)` : nombre d'échantillons envoyés au contrôleur audio par trame
- ▷ `table_1 (ndarray)` : table de la voie 1
- ▷ `table_2 (ndarray)` : table de la voie 2

1.2.n. Sortie audio d'un signal

Les fonctions suivantes permettent de diriger un ou deux signaux numériques vers la sortie audio.

Les échantillons à envoyer sur une des deux voies de la sortie audio sont stockés dans un tampon circulaire. Les données sont écrites dans le tampon par blocs. La classe `RingBuffer` contient ce tampon circulaire.

```
buffer=RingBuffer(nblocks_power,samples_per_block)
```

Création d'un tampon.

- ▷ `nblocks_power (int)` : puissance de 2 du nombre de blocs, par exemple 6 pour 32 blocs.
- ▷ `samples_per_block (int)` : nombre d'échantillons par bloc.

Le tampon doit être détruit à la fin du script, afin de libérer l'espace mémoire associé :

```
RingBuffer.delete()
```

Destruction du tampon.

La fonction suivante permet d'écrire un bloc de données dans le tampon :

```
RingBuffer.write(data)
```

Écriture d'un bloc d'échantillons dans le tampon.

- ▷ `data (ndarray)` : tableau contenant le bloc d'échantillons, qui doit avoir la longueur `sample_per_block` défini au moment de la création du tampon.

La fonction suivante déclenche l'écriture d'un flux sur la sortie audio :

```
output_stream_start(sample_rate,ring_buffer_1,ring_buffer_2)
```

Configuration de la sortie audio.

- ▷ `sample_rate (int)` : fréquence d'échantillonnage en Hz.

- ▷ `ring_buffer_1 (RingBuffer)` : tampon circulaire pour la voie 1.
- ▷ `ring_buffer_2 (RingBuffer)` : tampon circulaire pour la voie 2 ou `None`.

Si l'on veut diriger le même flux de données sur les deux voies, il faut attribuer `None` à `ring_buffer_2`. Il ne faut pas attribuer le même tampon aux deux voies, car un tampon ne peut subir qu'une seule lecture.

La fonction suivante permet de stopper le flux audio :

```
output_stream_stop(seconds)
```

- ▷ `seconds (int)` : nombre de secondes avant l'arrêt.

Remarque importante : l'écriture des blocs dans le tampon doit se faire à la même fréquence que la sortie audio. Si N est la taille d'un bloc et f_e la fréquence d'échantillonnage, il faut faire cette écriture environ toutes les N/f_e secondes. Le tampon circulaire a néanmoins la faculté d'absorber des variations dans le rythme d'écriture, d'autant mieux que le nombre blocs est élevé.

Chapitre 2

Enregistrement d'un signal

2.1. Dispositif expérimental

Dans cet exemple, un capteur de force est placé sur la voie EA0 de la centrale SysamSP5. Un pendule à deux masses et trois ressorts est suspendu au capteur de force. Il s'agit d'enregistrer le signal fourni par le capteur de force au cours de l'oscillation du pendule.

2.2. Acquisition

On commence par importer tous les modules nécessaires :

```
import pycanum.main as pycan
import matplotlib.pyplot as plt
import numpy
import numpy.fft
```

On ouvre l'interface avec la centrale Sysam SP5 et on configure l'entrée 0 (EA0) avec une tension maximale de 5 V :

```
sys = pycan.Sysam("SP5")
sys.config_entrees([0],[5])
```

Choix de la période d'échantillonnage et du nombre d'échantillons :

```
te=0.01 # en secondes
fe=1/te
ne=10000
duree=ne*te
sys.config_echantillon(te*10**6,ne)
```

Le pendule est lancé puis on lance l'acquisition :

```
sys.acquerir()
```

Enfin on récupère les temps et les tensions puis on ferme l'interface :

```
t=sys.temps()
u=sys.entrees()
sys.fermer()
```

2.3. Tracé du signal

Les temps et les tensions de la première voie (la seule dans le cas présent) :

```
u1=u[0]
t1=t[0]
```

Les données doivent être enregistrées dans un fichier texte afin de les traiter ultérieurement :

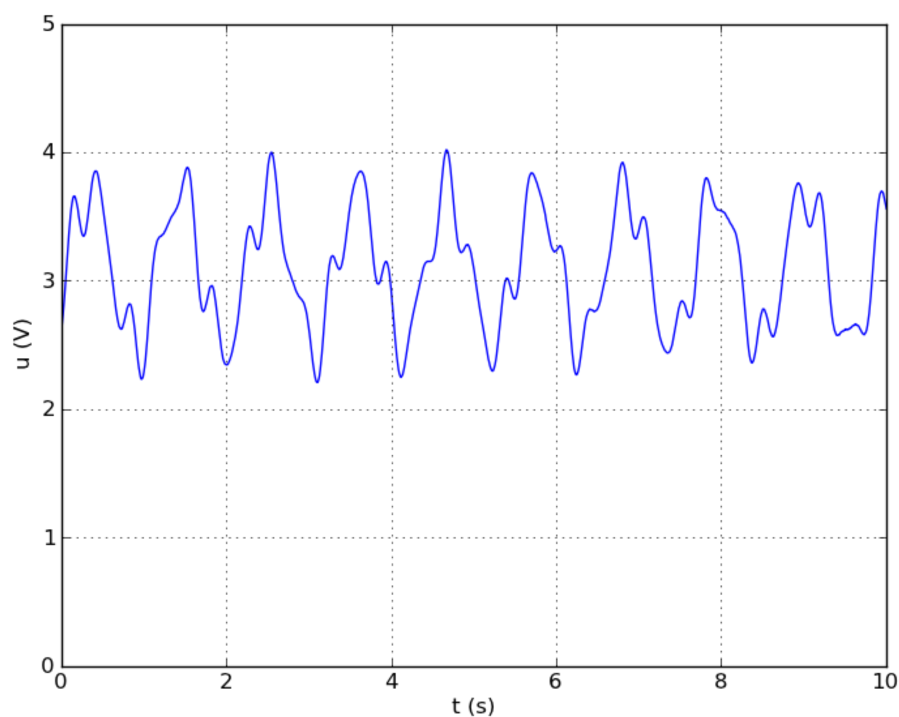
```
numpy.savetxt('pendule-4.txt',numpy.array([t1,u1]).T)
```

La lecture du fichier se fait de la manière suivante :

```
[t1,u1] = numpy.loadtxt('pendule-4.txt',unpack=True)
```

On trace le signal :

```
plt.figure()
plt.plot(t1,u1)
pylab.axis([0.0,10.0,0.0,5.0])
plt.grid()
plt.xlabel("t (s)")
plt.ylabel("u (V)")
```



Chapitre 3

Déclenchement de l'acquisition

3.1. Déclenchement par une des voies acquises

Une tension sinusoïdale de fréquence 1000 Hz est appliquée sur l'entrée EA0 de la centrale SysamSP5.

Ouverture de l'interface et configuration de l'entrée :

```
import pycanum.main as pycan
import matplotlib.pyplot as plt
import numpy
sys = pycan.Sysam("SP5")
sys.config_entrees([0],[10])
```

Configuration de l'échantillonnage :

```
te=1e-5
ne=100
tmax=ne*te
sys.config_echantillon(te*1e6,ne)
```

On choisit de déclencher l'acquisition lorsque la tension sur la voie EA0 dépasse le seuil de 0.0 V, sur un front montant. L'argument `pretrigger` précise le nombre de points qui doivent être stockés avant la condition de déclenchement (option pour SysamSP5).

```
sys.config_trigger(0,0.0,montant=1,pretrigger=10)
```

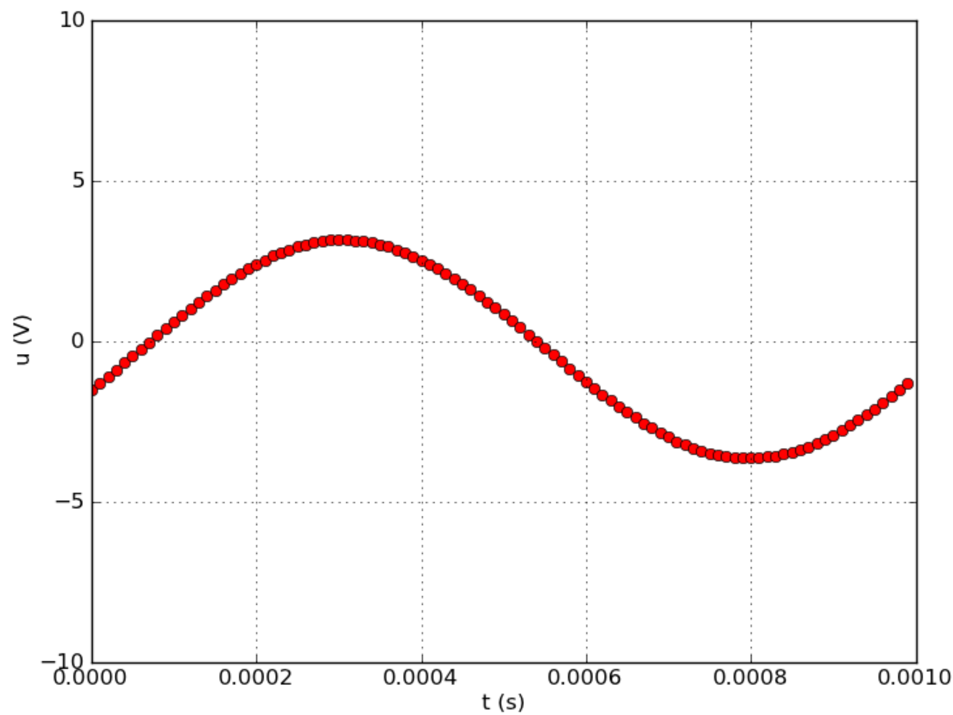
Attention : la valeur `pretrigger=0` ne conduit pas au déclenchement attendu. Il faut choisir au minimum `pretrigger=1`.

On déclenche l'acquisition et on récupère les temps et les tensions :

```
sys.acquerir()
t=sys.temps()
u=sys.entrees()
```

Tracé du signal :

```
plt.figure()
plt.plot(t[0],u[0],'ro')
plt.axis([0.0,tmax,-10.0,10.0])
plt.xlabel('t (s)')
plt.ylabel('u (V)')
plt.grid()
```

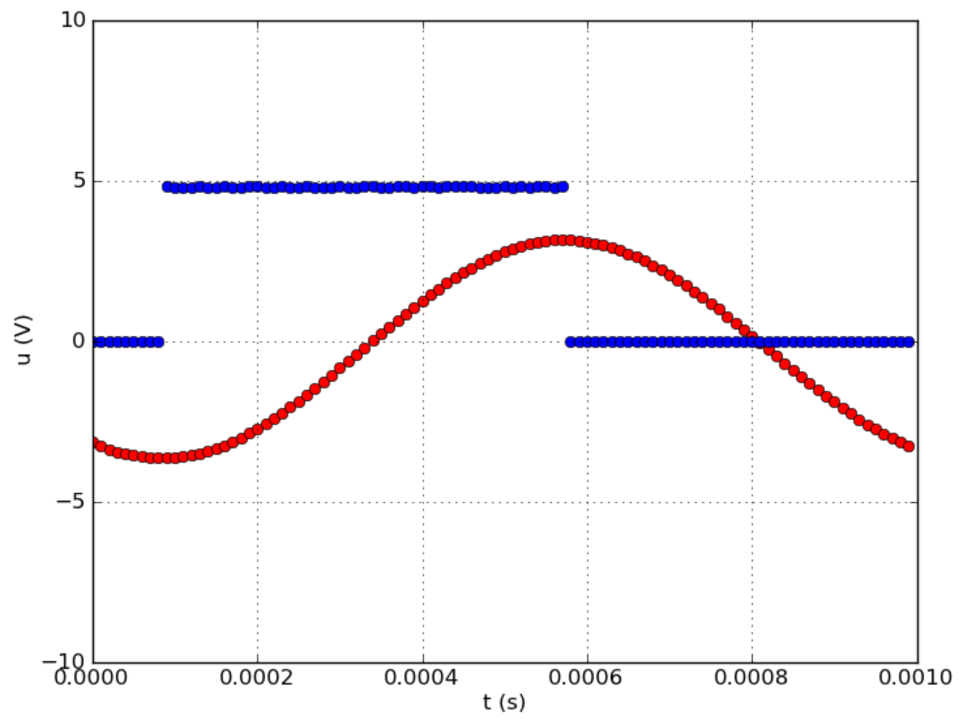


3.2. Déclenchement par un signal externe TTL

Un signal TTL (0-5 V) peut être utilisé pour effectuer le déclenchement. Il doit être envoyé sur l'entrée SYNCHRO EXT.

La tension sinusoïdale délivrée par le GBF est envoyée sur l'entrée EA0, le signal TTL délivré par le GBF est envoyé sur l'entrée EA1 et sur l'entrée SYNCHRO EXT.

```
sys.config_entrees([0,1],[10,10])
sys.config_echantillon(te*1e6,ne)
sys.config_trigger_externe(pretrigger=10)
sys.acquerir()
t=sys.temps()
u=sys.entrees()
plt.figure()
plt.plot(t[0],u[0],'ro')
plt.plot(t[1],u[1],'bo')
plt.xlabel('t (s)')
plt.ylabel('u (V)')
plt.axis([0.0,tmax,-10.0,10.0])
plt.grid()
sys.fermer()
plt.show()
```


10
5
0
−5
−10
u (V)
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
t (s)

Chapitre 4

Acquisition point par point

4.1. Introduction

Dans certaines expériences, l'acquisition peut se faire point par point, c'est-à-dire sans utiliser l'échantillonneur du CAN pour cadencer les mesures. L'acquisition peut être déclenchée par l'utilisateur, ou bien à intervalle de temps régulier (intervalle de l'ordre de la seconde ou plus).

Dans l'exemple ci-dessous, un pont diviseur constitué d'une résistance de 47 K et d'une thermistance de 47 K est alimenté par l'alimentation +5 V du CAN SysamSP5. La tension aux bornes de la résistance est lue sur l'entrée EA0. Pendant l'acquisition, la thermistance est légèrement chauffée (à la main) pour faire varier la tension mesurée.

4.2. Lecture directe

La première méthode consiste à utiliser la lecture directe des entrées. Les mesures sont faites de manière régulière, en effectuant un échantillonnage (approximatif) avec l'horloge de l'ordinateur. On commence par ouvrir l'interface et configurer l'entrée EA0 :

```
import pycan.main as pycan
import matplotlib.pyplot as plt
import numpy
import time

sys = pycan.Sysam("SP5")
sys.config_entrees([0],[10])
```

Choix de la période d'échantillonnage, du nombre de points et création des tableaux :

```
te = 5.0
ne = 30
listeEA0 = numpy.zeros(ne,dtype=float)
listeTemps = numpy.zeros(ne,dtype=float)
```

La commande suivante permet d'activer la lecture directe de l'entrée EA0 :

```
sys.activer_lecture([0])
```

Boucle d'acquisition :

```
for k in range(ne):
    tensions=sys.lire() # lecture directe
    listeTemps[k] = k*te # instant voulu
    listeEA0[k] = tensions[0]
    print("t = %f, u0 = %f"%(listeTemps[k],listeEA0[k]))
    t_clock_new = time.clock()
    t_reel = t_reel + t_clock_new-t_clock # instant réel
```

```
    t_clock = t_clock_new
    print("t reel = %f"%t_reel)
    time.sleep((k+1)*te-t_reel) # calage sur le temps réel à chaque itération
```

Remarquer la fonction `time.sleep`, qui effectue une attente de manière à effectuer l'acquisition suivante à l'instant voulu (à peu près). Dans cet exemple, le temps voulu est stocké, mais on pourrait aussi stocker le temps réel. On vérifie avec les valeurs affichées que le décalage entre le temps voulu et le temps réel reste très faible. Cependant, l'ordinateur n'est pas un système temps réel et donc la régularité de l'échantillonnage ne peut être garantie, bien qu'elle soit très probable pour des périodes aussi grandes. En conséquence, il est préférable de stocker le temps réel calculé dans la boucle.

Enfin on désactive la lecture directe puis on ferme l'interface :

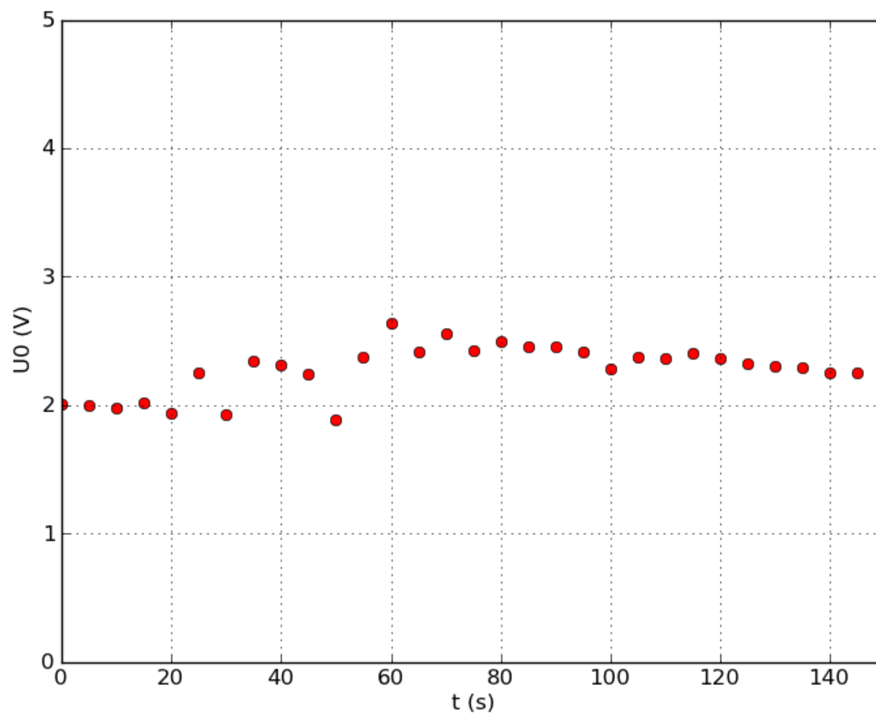
```
sys.desactiver_lecture()
sys.fermer()
```

Sauvegarde des données pour une utilisation ultérieure :

```
numpy.savetxt('entreeDirecte-data-1.txt',[listeTemps,listeEA0])
```

Tracé de la tension en fonction du temps :

```
plt.figure()
plt.plot(listeTemps,listeEA0,'ro')
plt.axis([0,ne*te,0.0,5.0])
plt.xlabel("t (s)")
plt.ylabel("U0 (V)")
plt.grid()
plt.show()
```



Les points ne semblent pas suivre une courbe régulière. Cela est dû au fait que la tension mesurée est très fluctuante. Dans ce cas, la lecture directe n'est pas une bonne méthode.

4.3. Lecture par paquet échantillonné

Dans le cas où la tension mesurée fluctue fortement, il faut effectuer l'acquisition de paquets échantillonnés afin de calculer la valeur moyenne. Cette méthode offre en plus l'avantage de fournir un écart type, c'est-à-dire une incertidude pour les mesures (incertitude liée aux fluctuations).

Le début est identique à l'exemple ci-dessus :

```
te = 5.0 # période d'échantillonnage en secondes
ne = 40 # nombre d'échantillons
listeEA0 = numpy.zeros(ne,dtype=float)
listeTemps = numpy.zeros(ne,dtype=float)
listeEcarts = numpy.zeros(ne,dtype=float)
sys = pycan.Sysam("SP5")
sys.config_entrees([0],[10])
```

On fixe la durée des paquets d'acquisition et le nombre de points :

```
ta = te/10.0 # durée de l'acquisition
na = 1000 # nombre de points de l'acquisition
tea = ta/na # période d'échantillonnage de l'acquisition
```

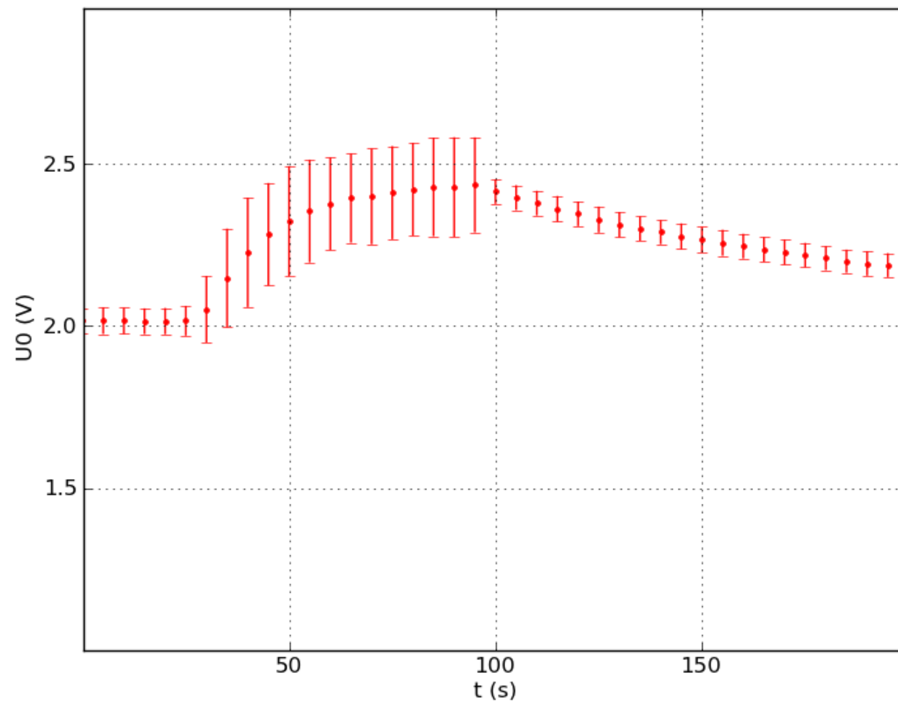
```
sys.config_echantillon(tea*1e6,na)
```

Dans la boucle de mesure, on calcule la moyenne des valeurs du paquet et l'écart type :

```
t_clock = time.clock()
t_reel = 0.0
for k in range(ne):
    sys.acquerir()
    tensions=sys.entrees()
    moyenne = numpy.mean(tensions[0])
    ecart = numpy.std(tensions[0])
    listeTemps[k] = k*te
    listeEA0[k] = moyenne
    listeEcarts[k] = ecart
    print("t = %f, u0 = %f, ecart = %f"%(listeTemps[k],listeEA0[k],listeEcarts[k]))
    t_clock_new = time.clock()
    t_reel = t_reel + t_clock_new-t_clock
    t_clock = t_clock_new
    print("t reel = %f"%t_reel)
    time.sleep((k+1)*te-t_reel)
sys.fermer()
numpy.savetxt('entreePointParPoint-data-1.txt',[listeTemps,listeEA0,listeEcarts])
```

On trace la tension en fonction du temps avec des barres d'incertitude :

```
plt.figure()
plt.plot(listeTemps,listeEA0,marker='.',linestyle='',color='r')
plt.errorbar(listeTemps,listeEA0,yerr=listeEcarts,xerr=None,fmt=None,ecolor='r')
plt.axis([0,ne*te,1.0,3.0])
plt.xlabel("t (s)")
plt.ylabel("U0 (V)")
plt.grid()
plt.show()
```



On voit à présent que la tension est très fluctuante lorsque la thermistance est tenue entre les doigts pour la chauffer.

Chapitre 5

Utilisation des sorties

5.1. Génération d'un signal échantillonné sur une sortie

La carte SysamPCI possède un convertisseur N/A (sortie SA1). La centrale SysamSP5 possède deux convertisseurs N/A (sorties SA1 et SA2).

On commence par charger les modules et ouvrir l'interface :

```
import pycanum.main as pycan
import matplotlib.pyplot as plt
import numpy
import math
import time

sys = pycan.Sysam("SP5")
```

Pour configurer une sortie, il faut tout d'abord créer un tableau numpy (ndarray) contenant les valeurs des échantillons en volts. Dans le cas d'un signal périodique, on peut se contenter d'échantillonner une période (voir le document Synthèse numérique d'un signal périodique pour l'inconvénient de cette méthode). Voici par exemple la génération d'une sinusoïde d'amplitude 5 volts :

```
ns = 1000 # nombre de points
v1 = 5.0*numpy.sin(2*numpy.pi*numpy.arange(ns)/ns)
```

La configuration d'une sortie se fait en précisant la période d'échantillonnage en microsecondes, les échantillons, et le nombre de répétitions des échantillons fournis. Si l'on veut une répétition périodique sans fin, on choisit ce nombre à -1 :

```
tes = 1.0e-5 # période d'échantillonnage en secondes
sys.config_sortie(1,tes*1e6,v1,-1)
```

On déclenche la sortie SA1 au moment souhaité :

```
sys.declencher_sorties(1,0)
```

Sur SysamPCI, la fonction précédente est bloquante et l'émission de la sortie périodique est stoppée par un appui sur la touche ESC. Avec la SysamSP5, la fonction précédente retourne immédiatement. Il faut placer une fonction d'attente avant de stopper la sortie :

```
time.sleep(10) # on laisse la sortie émettre pendant 10 secondes
sys.stopper_sorties(1,0)
sys.fermer()
```

On peut aussi utiliser la fonction `matplotlib.pyplot.show` pour bloquer l'exécution, ce qui laisse l'émission se faire jusqu'à fermeture de la fenêtre.

Remarque : la période d'échantillonnage effective est nécessairement multiple de la période d'échantillonnage de base, qui est de 0.2 μs sur SysamSP5. Pour générer un signal de fréquence arbitraire, il faut l'échantillonner sur plusieurs périodes, comme expliqué dans le document Synthèse numérique d'un signal périodique.

5.2. Utilisation des entrées et sorties (SysamSP5)

Sur la SysamSP5, il est possible de déclencher les sorties, puis une acquisition.

Dans l'exemple suivant, les sorties génèrent deux sinusoïdes déphasées avec une période de 100 ms. Les tensions des deux sorties sont acquises sur les voies EA0 et EA1. Un signal TTL de période 10 ms est acquis sur la voie EA2. Les périodes d'échantillonnage des entrées et sorties sont choisies identiques (10 microsecondes).

Configuration des entrées :

```
sys=pycan.Sysam("SP5")
sys.config_entrees([0,1,2],[10,10,10])
ne=20000
te=1e-5
sys.config_echantillon(te*1e6,ne)
```

Calcul des échantillons pour les sorties et configuration des sorties :

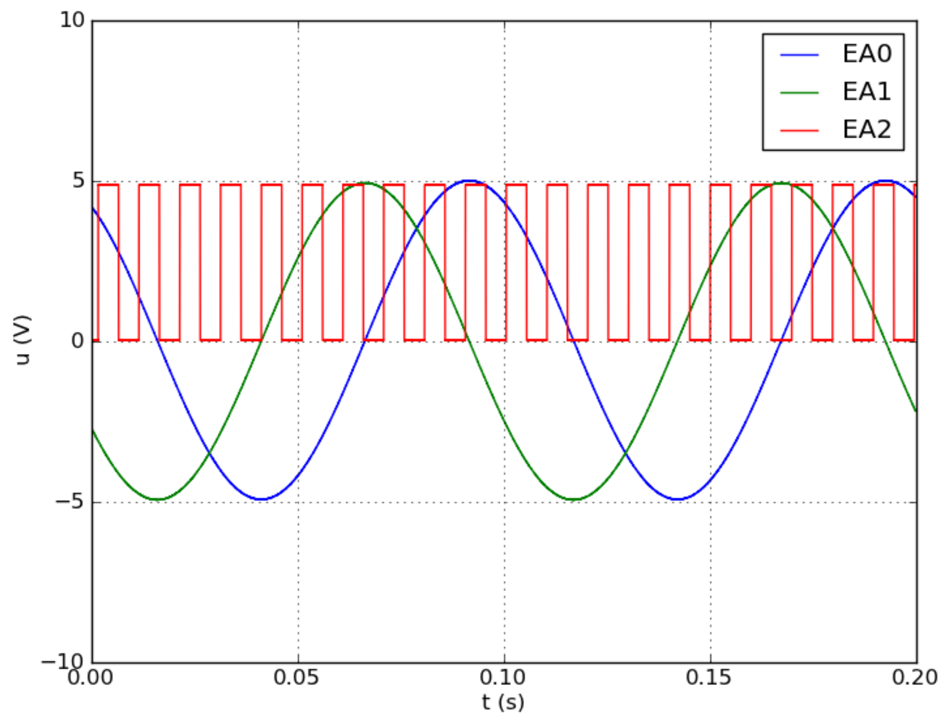
```
ns=10000
tes=te
v1 = 5.0*numpy.cos(2*numpy.pi*numpy.arange(ns)/ns)
v2 = 5.0*numpy.cos(2*numpy.pi*numpy.arange(ns)/ns + numpy.pi/2)
sys.config_sortie(1,tes*1e6,v1,-1)
sys.config_sortie(2,tes*1e6,v2,-1)
```

On déclenche les sorties, puis on attend 0.5 secondes avant de déclencher l'acquisition des entrées :

```
sys.declencher_sorties(1,1)
time.sleep(0.5)
sys.acquerir()
```

Récupération des données et tracé des courbes :

```
t=sys.temps()
u=sys.entrees()
sys.fermer()
plt.figure()
plt.plot(t[0],u[0],label="EA0")
plt.plot(t[1],u[1],label="EA1")
plt.plot(t[2],u[2],label="EA2")
plt.axis([0,te*ne,-10.0,10.0])
plt.xlabel("t (s)")
plt.ylabel("u (V)")
plt.grid()
plt.legend()
plt.show()
```



5.3. Utilisation simultanée des entrées et sorties

Sur SysamPCI, la méthode précédente est impossible car l'appel de la fonction `declencher_sorties(` est bloquant.

On peut faire une utilisation simultanée des entrées et sorties avec la fonction `acquerir_avec_sortie` qui fonctionne aussi sur SysamSP5.

Dans l'exemple suivant, une rampe est générée sur chaque sortie. Les tensions délivrées par les deux sorties sont acquises sur les voies EA0 et EA1. Un signal TTL de période 10 ms est acquis sur la voie EA2.

On génère les tableaux des échantillons pour les sorties :

```
sys=pycan.Sysam("SP5")
T=0.1
ne=10000
te=T/ne
v1 = numpy.zeros(ne,numpy.double)
v2 = numpy.zeros(ne,numpy.double)
for k in range(ne-2): # rampe  de 0 à 5.0 V, montante pour SA1, descendante pour SA2
    v1[k] = k*5.0/(ne-1)
    v2[k] = 5.0-v1[k]
```

Les deux dernières valeurs des tableaux sont laissées nulles de manière à annuler les tensions des sorties à la fin de l'acquisition.

On configure l'acquisition :

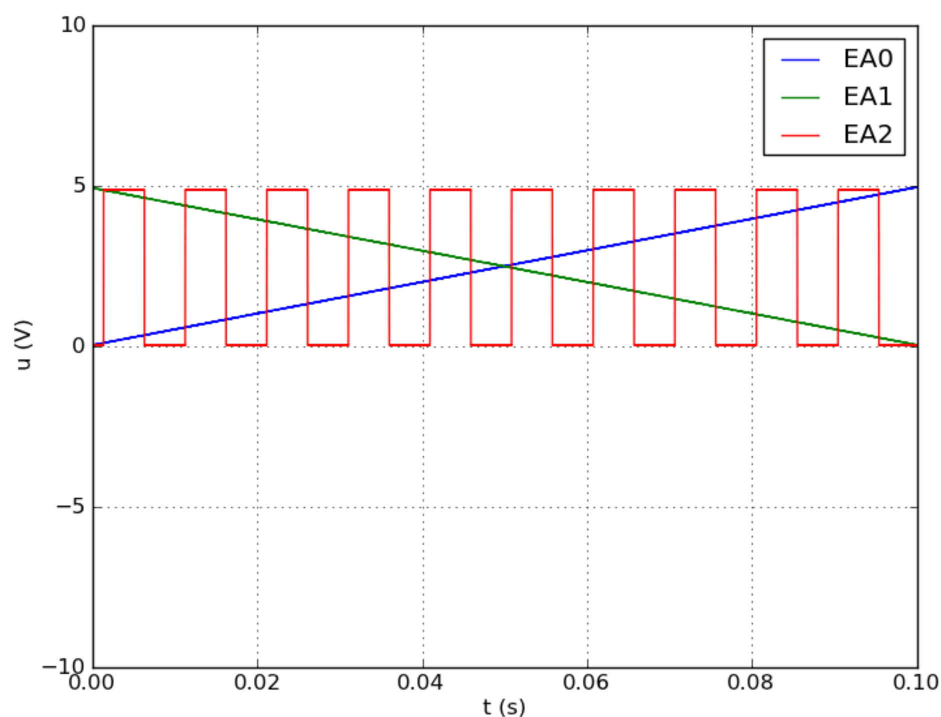
```
sys.config_entrees([0,1,2],[10,10,10])
sys.config_echantillon(te*1e6,ne)
```

Puis on déclenche l'acquisition et les sorties simultanément. La période d'échantillonnage des sorties est identique à celle des entrées, de manière à assurer la synchronisation.

```
sys.acquerir_avec_sorties(v1,v2)
```

Enfin on récupère les données et on trace les courbes :

```
t=sys.temps()
u=sys.entrees()
sys.fermer()
plt.figure()
plt.plot(t[0],u[0],label="EA0")
plt.plot(t[1],u[1],label="EA1")
plt.plot(t[2],u[2],label="EA2")
plt.xlabel("t (s)")
plt.ylabel("u (V)")
plt.axis([0,ne*te,-10.0,10.0])
plt.grid()
plt.legend()
plt.show()
```



Chapitre 6

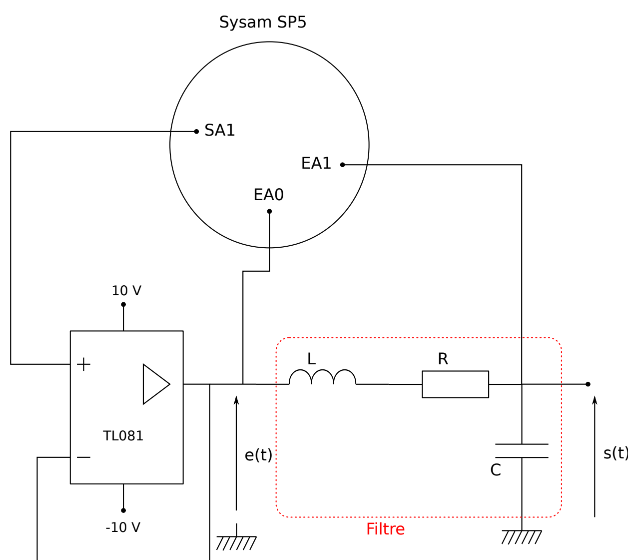
Diagramme de Bode

6.1. Introduction

Cette partie montre comment obtenir la réponse fréquentielle d'un filtre au moyen d'une carte Sysam SP5. La tension appliquée à l'entrée du filtre est générée par la carte et le script Python présenté permet de faire un balayage de la fréquence.

6.2. Dispositif expérimental

Le dispositif permet d'étudier la réponse d'un filtre à un signal délivré par la sortie SA1 de la carte Sysam SP5.



La tension d'entrée du filtre est $e(t)$, sa tension de sortie est $s(t)$. À titre d'exemple, nous étudions un filtre constitué d'une bobine et d'un condensateur placés en série, la tension de sortie étant prise aux bornes de ce dernier. Lorsque l'impédance d'entrée du filtre est faible, comme c'est le cas ici au voisinage de la fréquence de résonance, il est nécessaire de placer un amplificateur suiveur entre la sortie SA1 de la carte et l'entrée du filtre.

Le tension $e(t)$ est numérisée sur la voie EA0, la tension $s(t)$ est numérisée du la voie EA1.

6.3. Méthodes numériques

6.3.a. Génération de la tension d'entrée

La tension appliquée est sinusoïdale donc les deux signaux ont la forme suivante :

$$e(t) = E\cos(\omega t)$$
$$s(t) = S\cos(\omega t + \phi)$$

Le convertisseur numérique-analogique de la sortie SA1 fonctionne à la même fréquence d'échantillonnage que le convertisseur analogique-numérique. La fréquence d'échantillonnage maximale pour une utilisation simultanée des entrée et d'une sortie est $1\,MHz$. Nous pouvons donc en principe générer une sinusoïde jusqu'à une fréquence de $499\,kHz$ en respectant la condition de Nyquist-Shannon.

Notons e_n et s_n les signaux échantillonnés, pour $n = 0, 1, \dots N-1$.

Voyons tout d'abord comment les échantillons e_n sont générés. Afin de pouvoir choisir la fréquence f précisément, les N échantillons comportent un nombre variable P de cycles. La fréquence est donc :

$$f = \frac{P}{NT_e} \tag{6.1}$$

où T_e est la période d'échantillonnage. Le nombre d'échantillons par période est :

$$N_p = \frac{N}{P} \tag{6.2}$$

Ce nombre est en général non entier, ce qui permet justement de faire varier finement la fréquence alors que la période d'échantillonnage ne peut prendre que des valeurs multiples d'une période de base, que nous fixons à 1 microseconde. Dans un premier temps, une valeur approximative de N_p est choisie, par exemple 100, mais cette valeur peut être abaissée lorsque la fréquence est trop grande.

Supposons que l'on ait fixé `N` et que l'on souhaite une valeur approximative `Np` et une fréquence `freq`. On commence par préciser la valeur de la période d'échantillonnage minimale :

```
teMin=1e-6
```

Lorsque le nombre `Np` d'échantillons par période est supérieur au rapport de la période par la période d'échantillonnage minimale, on doit corriger sa valeur :

```
Np = min(Np,1/(teMin*freq))
```

La période d'échantillonnage adoptée est un multiple de `teMin` qui permet d'obtenir environ la fréquence `freq` avec `Np` échantillons par période :

```
techant = int(1/(Np*freq*teMin))*teMin
if techant==0:
        techant = teMin
```

On calcule la valeur de P, le nombre de périodes pour les N échantillons :

```
P = int(freq*N*techant)
```

Pour finir, on calcule la fréquence effective, qui peut être légèrement différente de la fréquence demandée :

```
freq = P/(N*techant)
```

et la valeur effective du nombre de points par périodes :

```
Np=N/P
```

Cette valeur est différente de la valeur demandée car elle est en général non entière.

Voici comment se fait la génération des N échantillons de tension pour la sortie SA1 :

```
e1 = amp*np.cos(2*np.pi*P/N*np.arange(N))
```

Si l'on veut que la fréquence effective soit toujours très proche de la fréquence demandée, il faut que P soit grand et donc il faut que N soit très grand. La valeur $N = 50000$ donne de très bons résultats. La quantité de mémoire interne de la carte Sysam SP5 qu'il faut utiliser est $3N$ mots de 12 bits, ce qui est largement inférieure à la valeur maximale (262144).

6.3.b. Traitement des signaux

Soient $\overline{e}$ et $\overline{s}$ les valeurs moyennes des tensions d'entrée et de sortie du filtre. La première est en principe nulle mais la seconde peut avoir une valeur non négligeable par rapport à l'amplitude du signal. Ces valeurs moyennes sont évaluées en calculant la moyenne arithmétique du signal échantillonné :

$$\overline{e} = \frac{1}{N}\sum_{n=0}^{N-2} e_n$$

$$\overline{s} = \frac{1}{N}\sum_{n=0}^{N-2} s_n$$

La fonction `numpy.mean` permet de calculer la valeur moyenne.

L'amplitude efficace du signal retranchée de sa valeur moyenne est l'écart-type des N échantillons :

$$E_{eff} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1}(e_n - \overline{e})^2}$$

$$S_{eff} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1}(s_n - \overline{s})^2}$$

La fonction `numpy.std` permet de calculer l'écart-type.

Le gain du filtre est le rapport des amplitudes efficaces :

$$G = \frac{S_{eff}}{E_{eff}} \tag{6.3}$$

Afin de calculer le déphasage entre la sortie et l'entrée, considérons le produit suivant :

$$\begin{aligned} z(t) &= s(t)(e(t) - je(t - \frac{T}{4})) \\ &= ES\cos(\omega t + \phi)(\cos(\omega t) - j\sin(\omega t)) \\ &= \frac{ES}{2}\left(e^{j\phi} + e^{-j(2\omega t+\phi)}\right) \end{aligned}$$

Sa valeur moyenne est :

$$\overline{z} = \frac{ES}{2}e^{j\phi} = E_{eff}S_{eff}e^{j\phi} \tag{6.4}$$

Le déphasage cherché est donc l'argument de $\overline{z}$. On peut aussi calculer la valeur de la fonction de transfert :

$$\underline{H} = \frac{\overline{z}}{E_{eff}^2} \tag{6.5}$$

Le calcul des échantillons z_n nécessite de disposer du signal $e(t-\frac{T}{4})$, c'est-à-dire le signal $e(t)$ en avance d'un quart de période. Le nombre d'échantillons de décalage correspondant est la partie entière de $N_p/4$. Cependant, lorsque N_p est faible, l'obtention précise du signal en quadrature pose problème, à l'exception du cas peu probable où N_p est un nombre entier divisible par 4. Pour résoudre cette difficulté, nous effectuons une augmentation du nombre d'échantillons en procédant à une interpolation par transformée de Fourier. Cette méthode d'interpolation (d'un signal périodique) consiste à calculer la transformée de Fourier discrète du signal (TFD) à N echantillons, à séparer les deux parties conjuguées de taille $N/2$ puis à ajouter kN zéros entre ces deux parties. On obtient ainsi la TFD d'un signal qui comporte (k+1)N échantillons et qui est obtenu par la transformée de Fourier discrète inverse. Le nombre d'échantillons ajoutés par interpolation entre deux échantillons du signal initial est k.

6.4. Programme complet

analyseFrequentielle.py

```
import numpy as np
import matplotlib.pyplot as plt
import pycanum.main as pycan
import numpy.fft
```

La fonction `interpol` ci-dessous effectue l'interpolation par transformée de Fourier discrète. `x` est un tableau contenant le signal échantillonné. Le nombre d'échantillons est multiplié par `ninter+1` (`ninter` contient le paramètre noté k ci-dessus).

```
def interpol(x,ninter):
    N = len(x)
    tfd = numpy.fft.fft(x)
    N1 = N//2
```

```
    tfd2 = np.concatenate((tfd[0:N1],np.zeros(N*ninter),tfd[N1:N]))
    y = np.real(numpy.fft.ifft(tfd2))*(ninter+1)
    return y
```

La fonction `mesure` présentée ci-dessous effectue l'acquisition et les calculs pour une fréquence donnée. Ses paramètres sont :

- ▷ `can` : objet de la classe Sysam. Les deux entrées utilisées doivent être configurées au préalable.
- ▷ `freq` : fréquence souhaitée, en hertz.
- ▷ `amp` : amplitude de $e(t)$, en volts.
- ▷ `delai` : délai en secondes avant de commencer l'analyse des signaux (pour le régime transitoire).
- ▷ `N` : nombre d'échantillons des signaux.
- ▷ `Np` : nombre approximatif d'échantillons par période (maximal).
- ▷ `ninter` : nombre d'échantillons ajoutés par interpolation.
- ▷ `plot` : si `True`, les signaux $e(t)$ et $s(t)$ sont tracés.

Les valeurs renvoyées sont :

- ▷ `freq` : fréquence effective du signal.
- ▷ `G` : gain du filtre.
- ▷ `phi` : déphasage du filtre, en radians.
- ▷ `H` : valeur de la fonction de transfert.
- ▷ `techant` : période d'échantillonnage, en secondes.
- ▷ `Np` : nombre d'échantillons par période.

```
def mesure(can,freq,amp,delai,N=50000,Np=100,ninter=4,plot=False):
    teMin = 1e-6
    Np = min(Np,1/(teMin*freq))
    techant = int(1/(Np*freq*teMin))*teMin
    if techant==0:
        techant = teMin
    P = int(freq*N*techant)
    freq = P/(N*techant)
    Np = N/P
    e1 = amp*np.cos(2*np.pi*P/N*np.arange(N))
    can.config_echantillon(techant*1e6,N)
    can.acquerir_avec_sorties(e1,0)
    t=can.temps()[0]
    signaux = can.entrees()
    u=signaux[0]
    s=signaux[1]
    N = len(t)
```

```
    n1 = int(delai/techant)
    t=t[n1:N]
    t=t-t[0]
    e=u[n1:N]
    s=s[n1:N]
    N=len(e)
    r = ninter+1
    if ninter>0:
        e = interpol(e,ninter)
        s = interpol(s,ninter)
        t = np.arange(len(e))*t[len(t)-1]/len(e)
    d = int(Np*r/4)
    E=e.std()
    S=s.std()
    G=S/E
    z=s[d:N]*(e[d:N]-1j*e[0:N-d])
    Z = z.mean()
    phi = np.angle(Z)
    H = Z/E**2
    if plot:
        plt.figure()
        plt.plot(t,e,'b')
        plt.plot(t,s,'r')
        plt.grid()
        plt.ylim(-10,10)
        plt.xlim(0,10/freq)
        plt.show()
    return(freq,G,phi,H,techant,Np)
```

Voici le script qui permet d'acquérir la réponse fréquentielle du filtre. On commence par ouvrir l'interface avec la carte Sysam SP5 :

```
can = pycan.Sysam("SP5")
```

puis on initialise les listes qui serviront à les mesures et on définit le tableau des fréquences :

```
liste_f = []
liste_G = []
liste_phi = []
frequences = np.logspace(2,4,300)
```

L'amplitude initiale doit être inférieure à 10 V et doit être telle que l'amplitude de la sortie pour la plus faible fréquence soit inférieure à 10 V. Dans le cas du circuit étudié ici,

l'amplitude initiale est choisie égale à 2 volts car le montage suiveur avec le circuit LC ne fonctionne pas de manière linéaire pour une amplitude plus grande (à cause de la limite du courant de sortie de l'ALI).

```
amp0 = 2.0
amp = amp0
delai = 2e-2
ninter=0
```

Dans la boucle ci-dessous, la mesure pour une fréquence donnée se fait deux fois. La première mesure est faite avec le calibre maximal pour les deux entrées (10 V). Elle permet d'estimer le gain du filtre. Pour la seconde mesure, on ajuste l'amplitude de telle sorte que l'amplitude en sortie ne dépasse par 2 volts (pour d'autres filtres analysés, d'autres stratégies sont possibles). Le calibre des entrées est choisi afin d'offrir le maximum de précision puis la seconde mesure est effectuée, qui donne un résultat plus précis que la première. L'amplitude du signal d'entrée est conservée pour la mesure suivante, ce qui permet d'adapter cette amplitude aux variations du gain. Par exemple pour le circuit LC, l'amplitude diminue à l'approche de la résonance pour s'adapter à la très forte augmentation du gain.

```
for f in frequences:
    can.config_entrees([0,1],[10.0,10.0])
    (f,G,phi,H,te,Np) = mesure(can,f,amp,delai,ninter=ninter)
    amp = min(amp0,amp0/G)
    can.config_entrees([0,1],[amp*1.1,amp*G*1.1])
    (f,G,phi,H,te,Np) = mesure(can,f,amp,delai,ninter=ninter,plot=False)
    print("f = %f Hz, G = %f, phi = %f, te = %f, Np = %f"%(f,G,phi,te,Np))
    liste_f.append(f)
    liste_G.append(G)
    liste_phi.append(phi)
```

On ferme l'interface et on enregistre les mesures dans un fichier :

```
can.fermer()
np.savetxt('filtreLC-1.txt',np.array([liste_f,liste_G,liste_phi]).T,header='f\t G\t
```

Le gain en décibel est calculé et on utilise la fonction `numpy.unwrap` pour éliminer les sauts de phase de 2π. Pour finir, on trace le gain en décibel et le déphasage en fonction de la fréquence, en échelle logarithmique :

```
GdB = 20*np.log10(liste_G)
liste_phi = np.unwrap(liste_phi)
plt.figure()
plt.plot(liste_f,GdB,'b-')
plt.xscale('log')
```

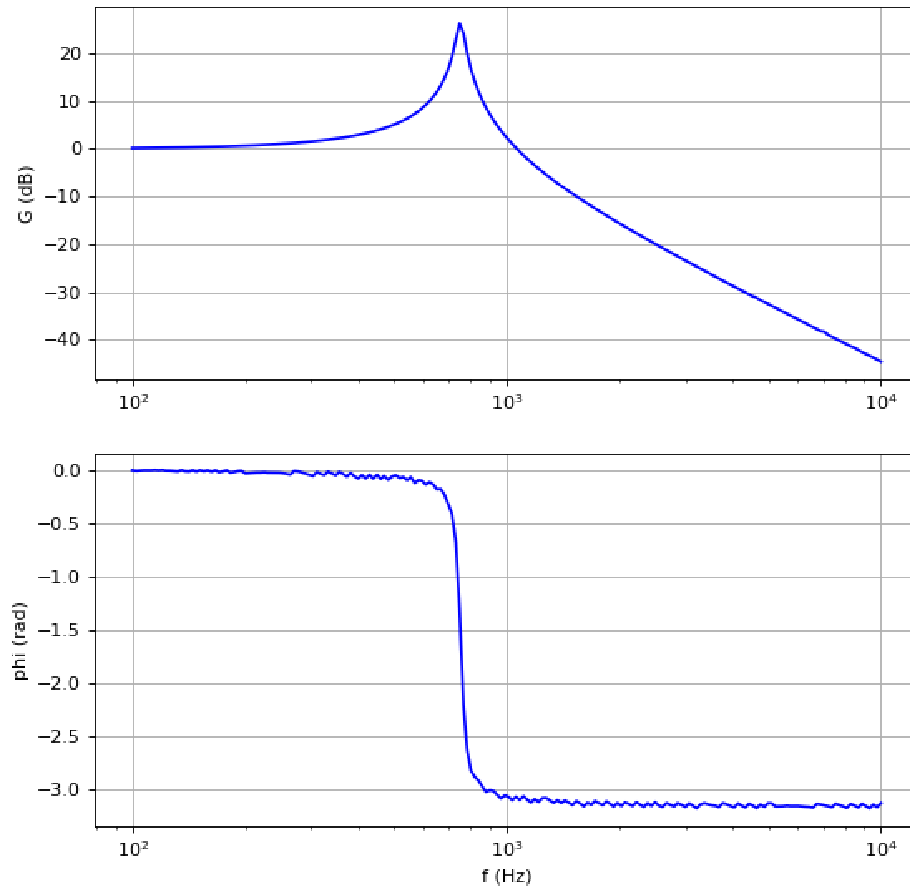
```
plt.grid()
plt.xlabel('f (Hz)')
plt.ylabel('G (dB)')
plt.figure()
plt.plot(liste_f,liste_phi,'b-')
plt.xscale('log')
plt.xlabel('f (Hz)')
plt.ylabel('phi (rad)')
plt.grid()
plt.show()
```

6.5. Exemple

On présente ici les résultats pour le filtre LC (bobine d'auto-inductance 45 mH et de résistance 12 Ω, condensateur de 1 μF).

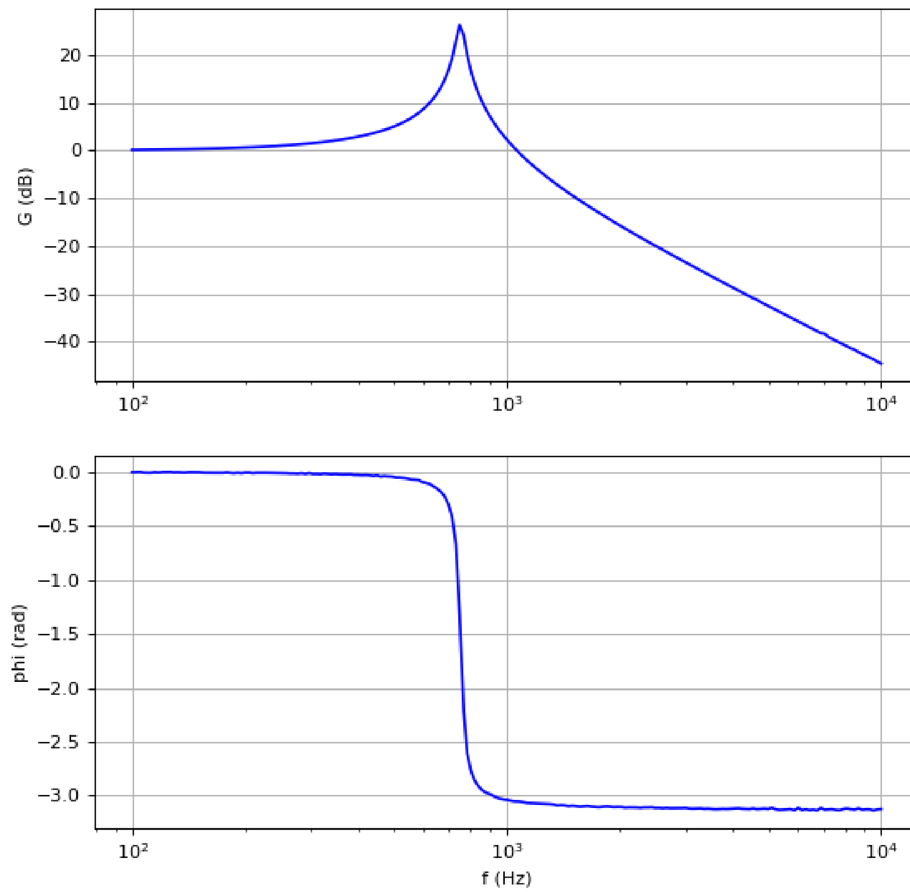
Voici le diagramme de Bode obtenu sans interpolation des signaux :

```
import numpy as np
from matplotlib.pyplot import *

[liste_f,liste_G,liste_phi] = np.loadtxt('filtreLC-1.txt',unpack=True, skiprows=1)
GdB = 20*np.log10(liste_G)
liste_phi = np.unwrap(liste_phi)

figure(figsize=(8,8))
subplot(211)
plot(liste_f,GdB,'b-')
xscale('log')
grid()
ylabel('G (dB)')
subplot(212)
plot(liste_f,liste_phi,'b-')
xscale('log')
xlabel('f (Hz)')
ylabel('phi (rad)')
grid()
```



Voici le diagramme de Bode obtenu avec `ninter=8` :

```
[liste_f,liste_G,liste_phi] = np.loadtxt('filtreLC-2.txt',unpack=True, skiprows=1)
GdB = 20*np.log10(liste_G)
liste_phi = np.unwrap(liste_phi)

figure(figsize=(8,8))
subplot(211)
plot(liste_f,GdB,'b-')
xscale('log')
grid()
ylabel('G (dB)')
subplot(212)
plot(liste_f,liste_phi,'b-')
xscale('log')
xlabel('f (Hz)')
ylabel('phi (rad)')
grid()
```



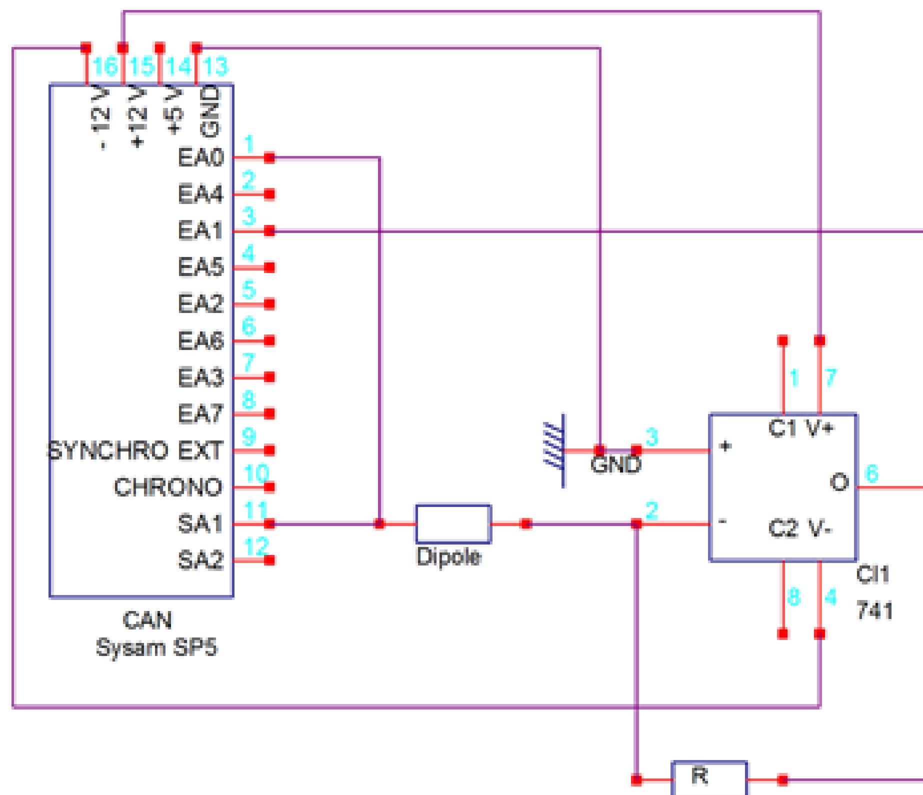
L'interpolation permet d'améliorer la précision de la mesure du déphasage.

Chapitre 7

Caractéristique d’un dipôle

7.1. Obtention d'une caractéristique à faible courant

Pour des courants ne dépassant pas 20 mA, la caractéristique d'un dipôle peut être obtenue avec le montage suivant :



Le dipôle est alimenté directement par la sortie SA1 du CAN. Un convertisseur courant-tension permet de lire le courant sur la voie EA1, alors que la tension aux bornes du dipôle est lue sur la voie EA0. La résistance R effectue la conversion courant-tension (i=u/R). La résistance R est choisie en fonction du courant maximal supporté par le dipôle sous une tension de 10 V. L'amplificateur est alimenté par les bornes +/- 12 V du CAN.

Dans l'exemple ci-dessous, le dipôle est une thermistance d'environ 1 $k\Omega$. La résistance R est choisie égale à 1 $k\Omega$.

```
import pycanum.main as pycan
import matplotlib.pyplot as plt
import numpy
import time
```

Pour acquérir un point de la caractéristique, il est préférable d'effectuer une acquisition échantillonnée que d'effectuer une mesure unique. Les valeurs de tension et de courant sont obtenues par moyennage des échantillons. L'écart-type permet d'obtenir les incertitudes dues aux fluctuations aléatoires. La fonction suivante effectue ces opérations. Une

pause de 10 secondes est ici introduite pour laisser le temps à la thermistance d'atteindre un état thermique stationnaire.

```
def acquerirPoint(r,sys,tension):
    print("u = %f\n"%tension)
    sys.ecrire(1,tension,0,0)
    time.sleep(10)
    ne=1000
    te=1.0e-4
    sys.config_echantillon(te*1e6,ne)
    sys.acquerir()
    temps=sys.temps()
    tensions=sys.entrees()
    u = numpy.mean(tensions[0])
    du = numpy.std(tensions[0])
    i = -numpy.mean(tensions[1])/r
    di = numpy.std(tensions[1])/r
    print("u = %f, i= %f\n"%(u,i))
    return [u,du,i,di]
```

Lorsque le CAN SysamSP5 est branché et son câble USB connecté, les sorties analogiques SA1 et SA2 prennent une valeur de saturation +12 V ou -12 V. On commence donc par appliquer une tension nulle pour laisser la thermistance se refroidir :

```
sys=pycan.Sysam("SP5")
sys.ecrire(1,0.0,0,0)
time.sleep(30)
```

On configure les deux entrées utilisées, avec une tension maximale de 10 V :

```
sys.config_entrees([0,1],[10,10])
```

On donne les valeurs de la résistance R, du nombre de points à acquérir, des tensions minimale et maximale, puis on génère les différentes listes :

```
r=0.1 # résistance du convertisseur courant-tension en kOhm
np = 100
umin = 0
umax = -10
listeU = numpy.linspace(start=umin,stop=umax,num=np)
liste_u = numpy.zeros(np,dtype=numpy.float32)
liste_du = numpy.zeros(np,dtype=numpy.float32)
liste_i = numpy.zeros(np,dtype=numpy.float32)
liste_di = numpy.zeros(np,dtype=numpy.float32)
```

La boucle d'acquisition :

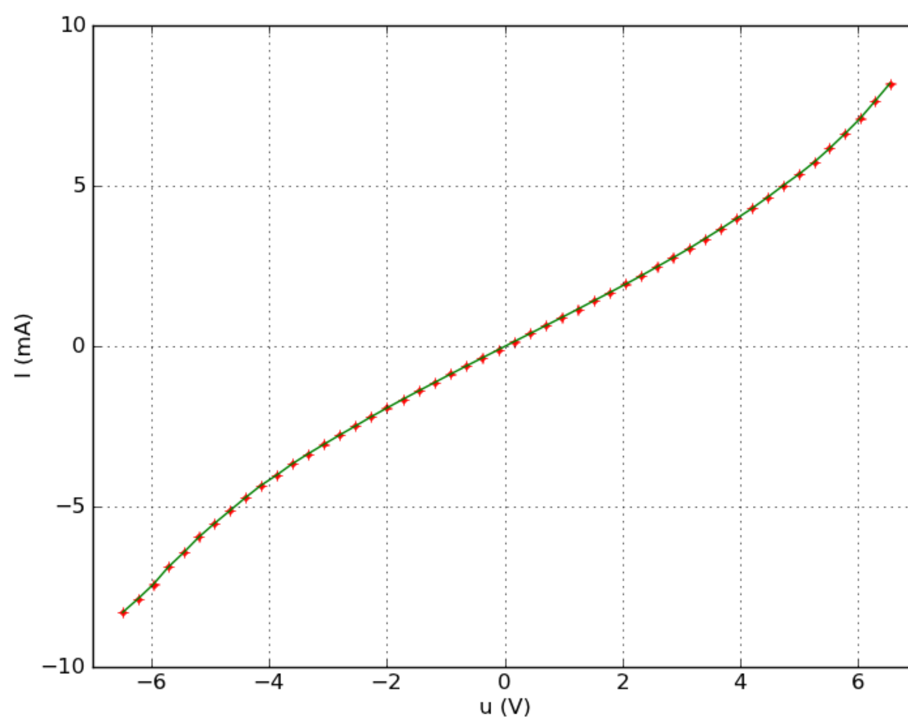
```
for k in range(np):
    [liste_u[k],liste_du[k],liste_i[k],liste_di[k]] = acquerirPoint(r,sys,listeU[k]
```

On ferme l'interface puis on sauvegarde les données :

```
sys.fermer()
numpy.savetxt('thermistance-10sec.txt',[liste_u,liste_du,liste_i,liste_di])
```

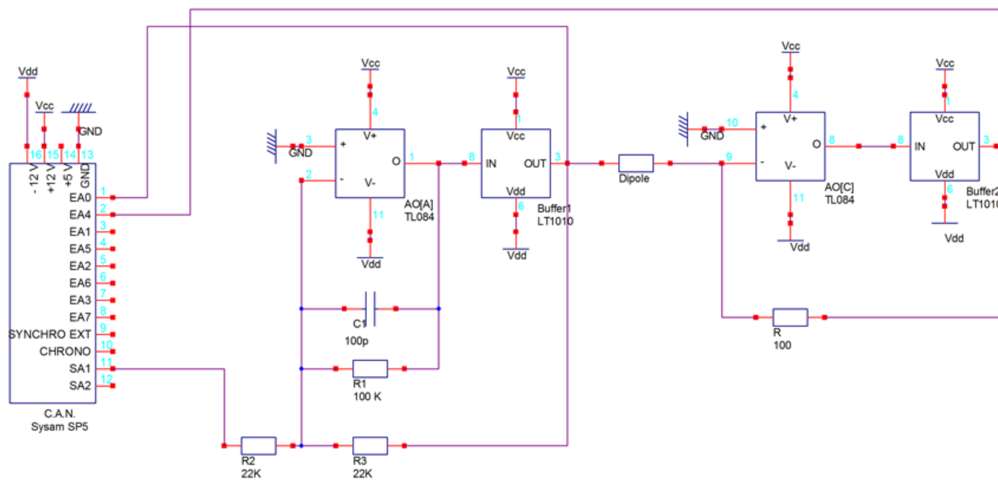
On effectue enfin le tracé de la caractéristique avec les barres d'incertitude :

```
plt.figure()
plt.plot(liste_u,liste_i,marker='.',linestyle='',color='r')
plt.errorbar(liste_u,liste_i,xerr=liste_du,yerr=liste_di,ecolor='r')
plt.axis([0,10,-10/r,10/r])
plt.xlabel("u (V)")
plt.ylabel("I (mA)")
plt.grid()
plt.show()
```

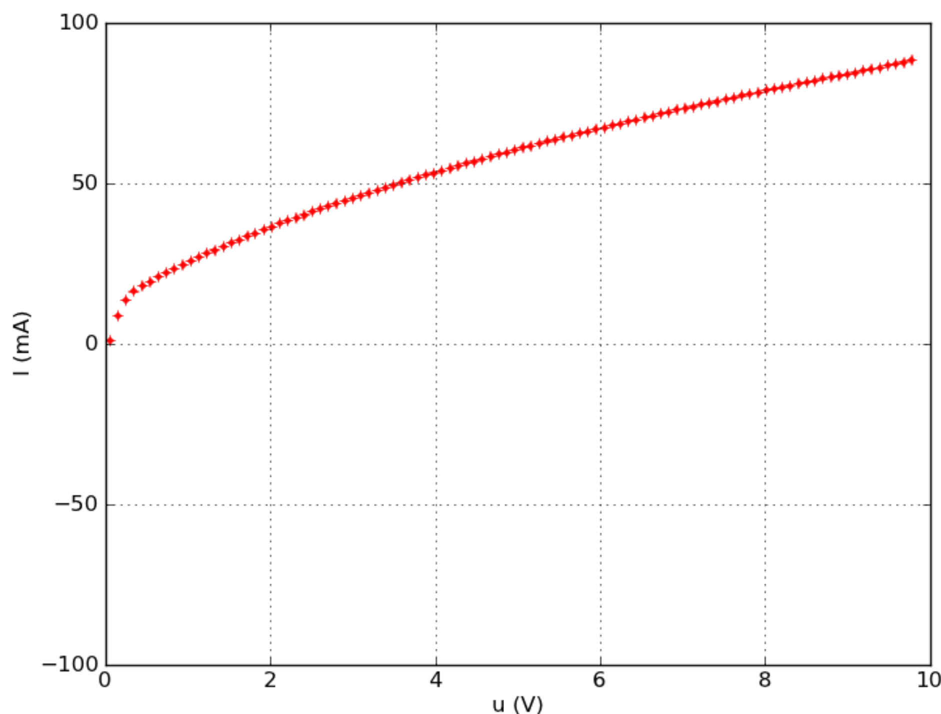


7.2. Obtention d'une caractéristique à courant moyen

Pour des courants jusqu'à 150 mA, un buffer de puissance LT1010 peut-être utilisé. Il en faut un pour amplifier la sortie du CAN et un second pour le convertisseur courant-tension. Dans les deux cas, le buffer est placé dans la boucle de rétroaction de l'amplificateur.



Dans l'exemple ci-dessous, le dipôle est une lampe à incandescence 12 V 100 mA. La résistance du convertisseur courant-tension est de 100 Ohm (2 Watt). À chaque point, le temps de pause avant la mesure est de 10 secondes. Voici la caractéristique obtenue :



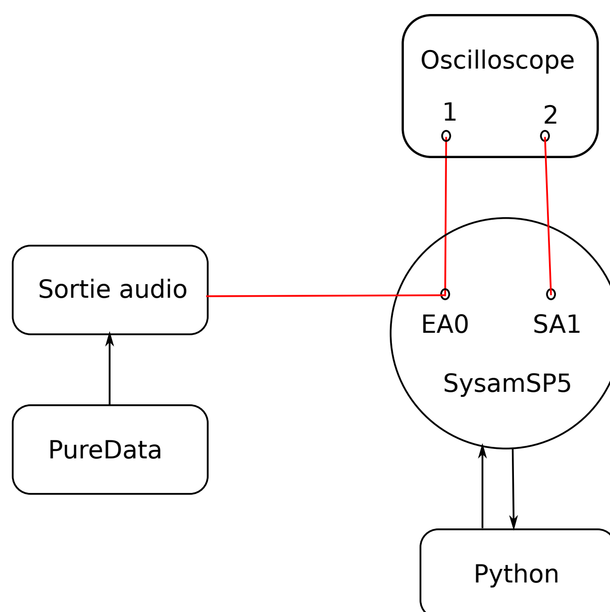
Chapitre 8

Filtrage numérique d’un signal

8.1. 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.



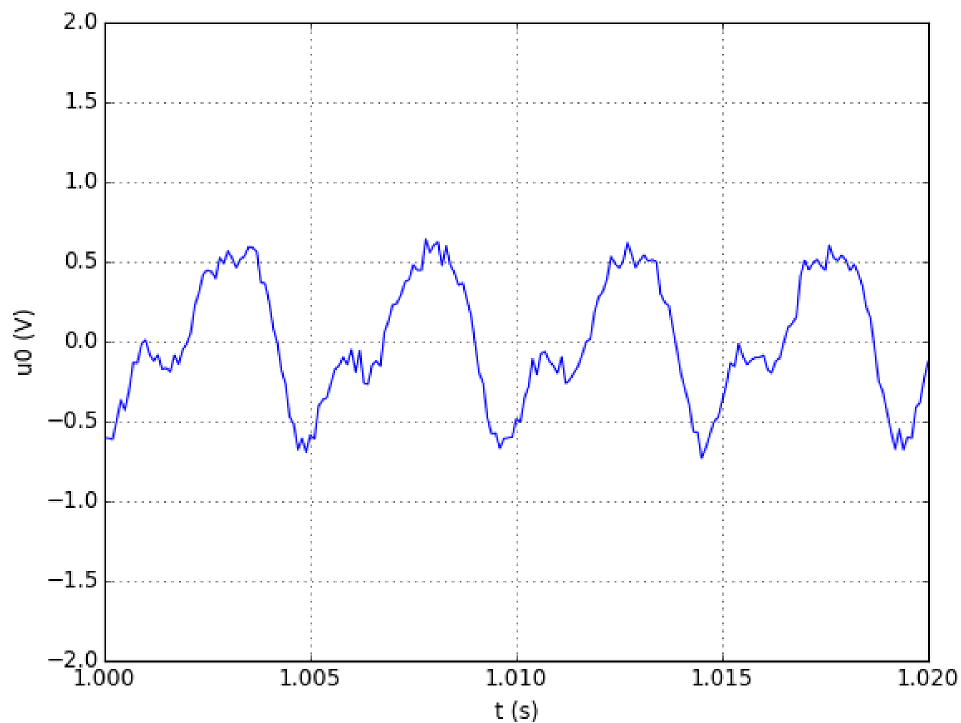
8.2. Conversion analogique-numérique

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.

8.3. 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) :

$$a = \frac{f_c}{f_e} \tag{8.1}$$

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 :

$$g(k) = 2a\frac{\sin(2\pi ka)}{2\pi ka} \tag{8.2}$$

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 :

$$h_k = g(k - P) \tag{8.3}$$

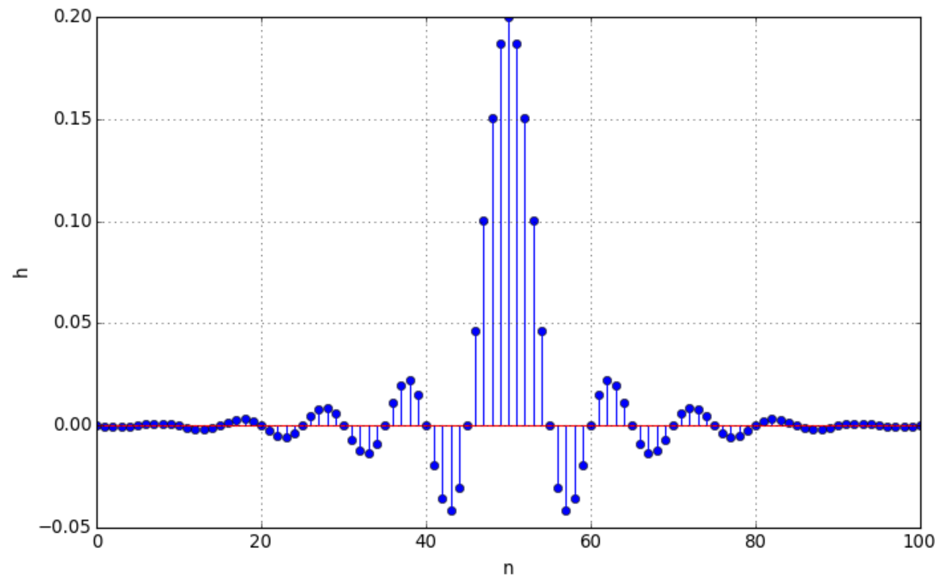
l'indice k variant de 0 à $2P$.

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()
```



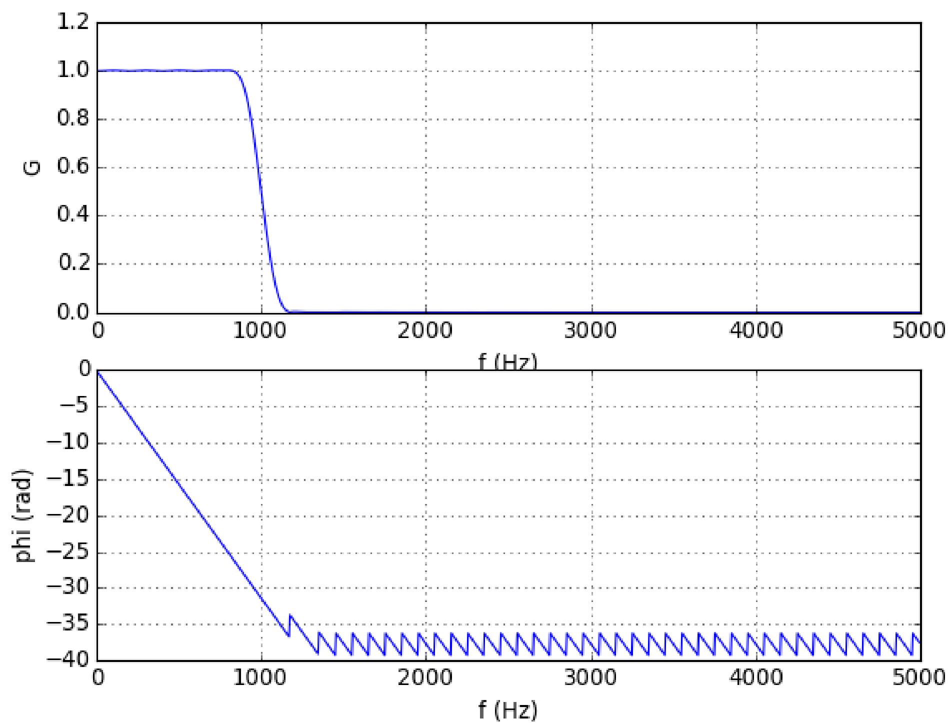
La réponse fréquentielle du filtre est donnée par la fonction suivante :

$$H_f(f) = \sum_{k=0}^{N-1} h_k \exp(-i2\pi\, k\, f\, T_e) \qquad (8.4)$$

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.

8.4. Filtrage numérique

On effectue le filtrage des échantillons par convolution :

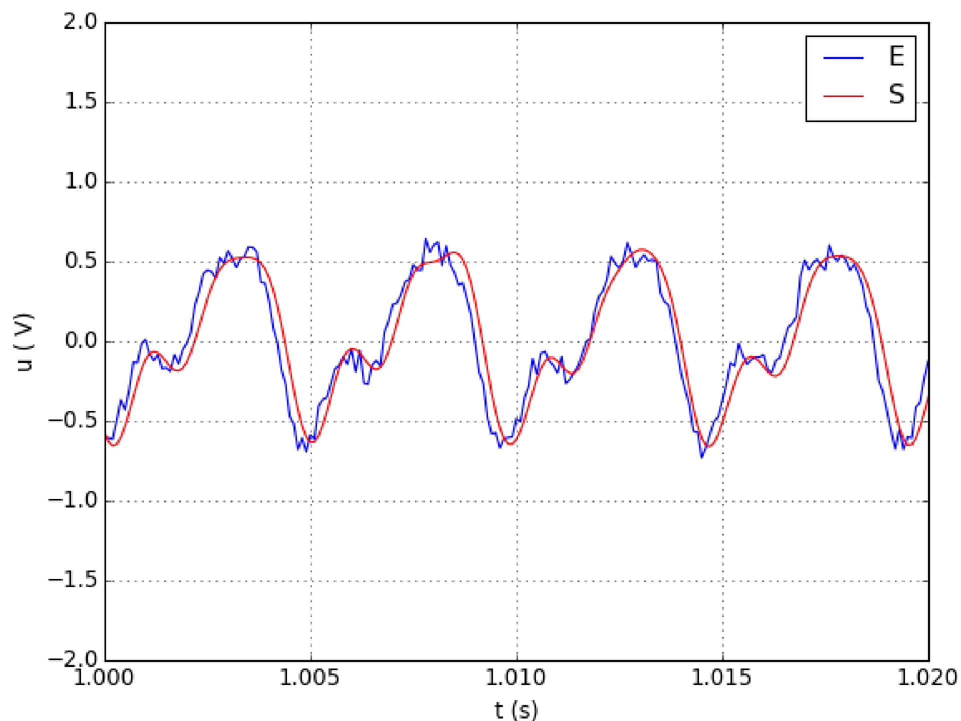
```
y = scipy.signal.convolve(u0,h,mode='same')
```

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 :

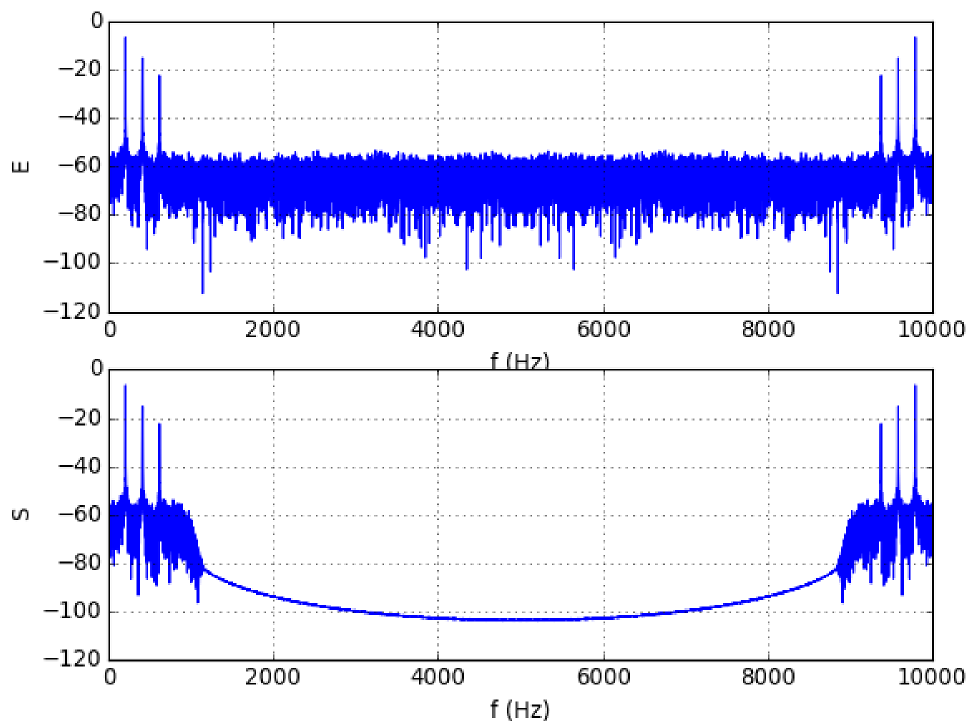
```
figure()
plot(t0,u0,'b',label='E')
plot(t0,y,'r',label='S')
xlabel("t (s)")
ylabel("u ( V)")
axis([1,1.02,-2,2])
legend(loc='upper right')
grid()
```



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()
```



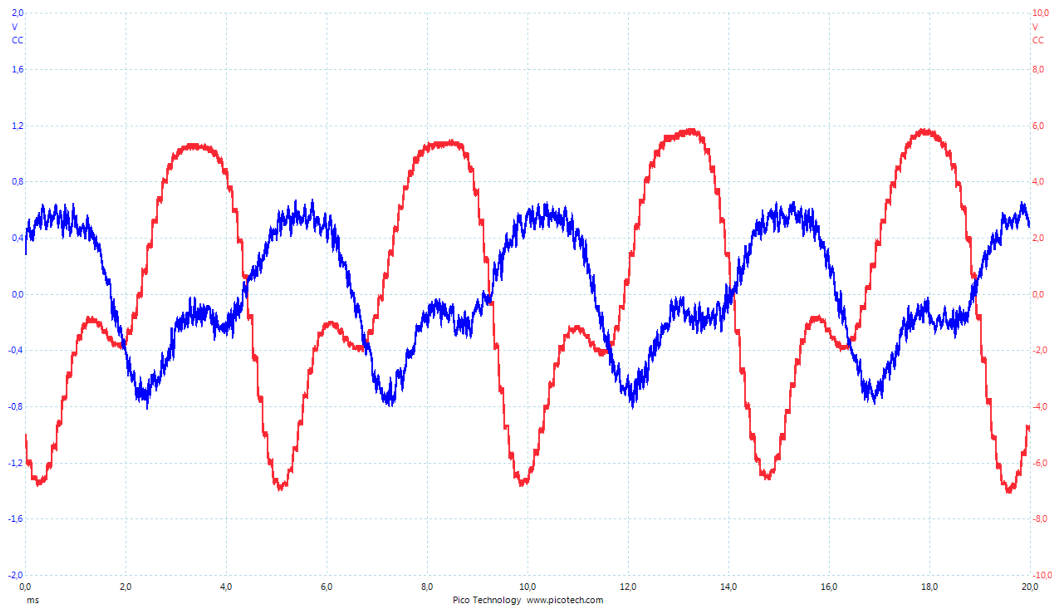
8.5. Conversion numérique-analogique

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 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.

8.6. Lissage numérique

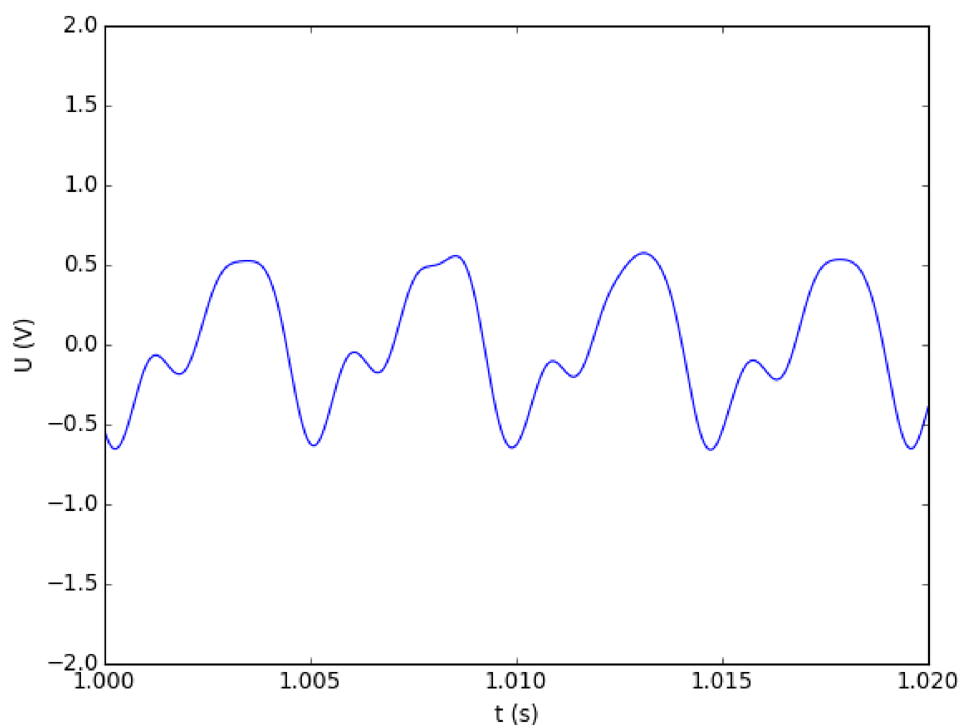
Cette technique consiste à augmenter la fréquence d'échantillonnage du signal numérique, avant de lui appliquer un lissage par interpolation. C'est la méthode utilisée dans les lecteurs de CD audio.

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 :

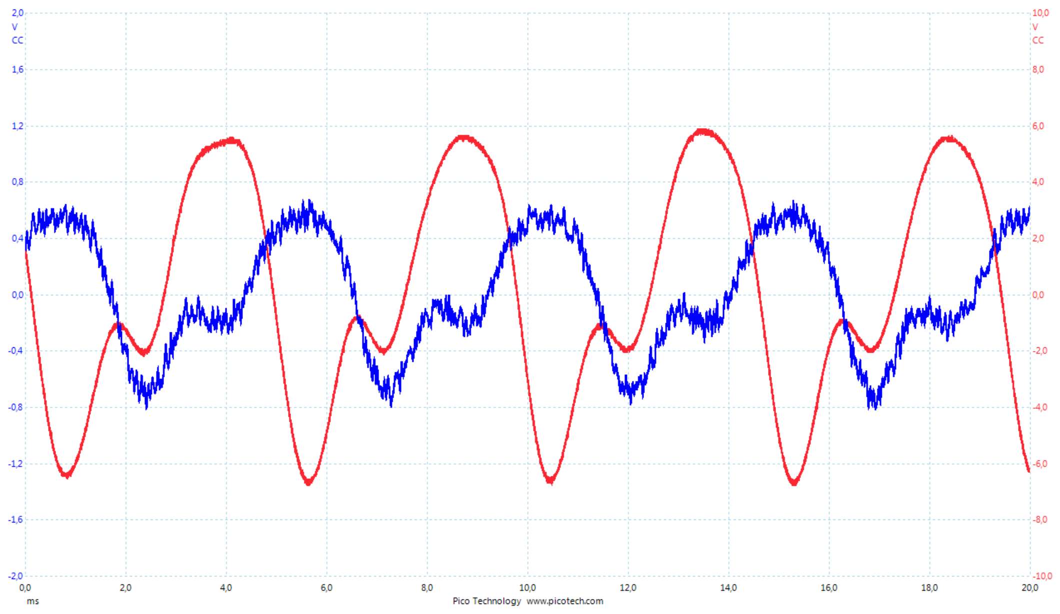
```
P = 10
h = scipy.signal.firwin(numtaps=2*P+1,cutoff=[0.5/n],nyq=0.5,window='hann')
y2 = scipy.signal.convolve(y1,h,mode='same')
figure()
plot(t1,y2)
xlabel('t (s)')
ylabel('U (V)')
axis([1,1.02,-2,2])
```



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

Chapitre 9

Acquisition et traitement parallèles

9.1. Introduction

Ce chapitre montre comment effectuer une acquisition de signaux en mode parallèle avec la carte SysamSP5. On verra comment faire un tracé des signaux simultanément à l'acquisition, avec éventuellement un filtrage des signaux.

9.2. Configuration des entrées

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

La configuration des entrées se fait comme pour une acquisition normale. On configure ici les voies 0 et 1 avec le calibre 10 volts. La période d'échantillonnage est de 10 ms. L'acquisition dure 100 s.

```
sys=pycan.Sysam("SP5")
sys.config_entrees([0,1],[10,10])
te=0.01
ne=10000
duree=te*ne
sys.config_echantillon(te*1e6,ne)
```

9.3. Tracé des signaux en mode parallèle

Pour lancer l'acquisition en mode parallèle, on exécute :

```
sys.lancer()
```

Lorsqu'on effectue un traitement des données parallèlement à leur acquisition, il faut définir un délai d'attente, qui correspond au temps approximatif qui doit s'écouler entre deux traitements consécutifs. Un traitement consiste à lire un paquet de données fourni par l'interface et à les traiter (filtrage, représentation graphique, etc). On définit le délai en secondes :

```
delai=0.1
```

Pour ce délai de 0.1 secondes, il y aura environ 10 nouveaux échantillons à chaque traitement.

On peut aussi avoir besoin du nombre de lectures à faire pendant l'acquisition complète (avec une marge de sécurité) :

```
n_lec=int(math.floor(duree*1.1/delai))
```

Voyons comment programmer une animation pour afficher les signaux. On définit une fonction qui sera appelée pour chaque image de l'animation. Cette fonction récupère les données déjà acquise par la carte SysamSP5 depuis le début et met à jour les deux courbes.

```
fig,ax = subplots()
x = [0]
y = [0]

line0, = ax.plot(x,y)
line1, = ax.plot(x,y)
ax.grid()
ax.axis([0,duree,-10,10])

def animate(i):
    global sys,line0,line1,t0,t1,u0,u1
    data=sys.paquet(0) # paquet des données déjà acquises (temps et tension)
    t0=data[0]
    t1=data[1]
    u0=data[2]
    u1=data[3]
    line0.set_xdata(t0)
    line0.set_ydata(u0)
    line1.set_xdata(t1)
    line1.set_ydata(u1)
```

Voici comment lancer l'animation. Le délai entre deux images est donné en millisecondes.

```
ani = animation.FuncAnimation(fig,animate,n_lec,interval=delai*1000)
show()
```

9.4. Filtrage en mode parallèle

Il est aisé d'ajouter un traitement des signaux pendant l'animation, par exemple un filtrage par convolution :

```
P=30
fc = 0.1
h = scipy.signal.firwin(numtaps=2*P+1,cutoff=[fc],nyq=0.5,window='hamming')

fig,ax = subplots()
```

```
x = [0]
y = [0]

line0, = ax.plot(x,y)
line1, = ax.plot(x,y)
ax.grid()
ax.axis([0,duree,-10,10])

def animate(i):
    global sys,h,line0,line1,t0,t1,u0,u1
    data=sys.paquet(0) # paquet des données déjà acquises (temps et tension)
    t0=data[0]
    t1=data[1]
    u0=data[2]
    u1=data[3]
    u1_filtre = scipy.signal.convolve(u1,h,mode='same')
    line0.set_data(t0,u0)
    line1.set_data(t1,u1_filtre)

ani = animation.FuncAnimation(fig,animate,n_lec,interval=delai*1000)
show()
```

Chapitre 10

Acquisition en mode permanent

10.1. 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 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).

10.2. Acquisition d'un grand nombre d'échantillons

10.2.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 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 s dans cet exemple), ou bien lancer l'acquisition sur une tâche parallèle (appel non bloquant) par :

```
sys.lancer_permanent()
```

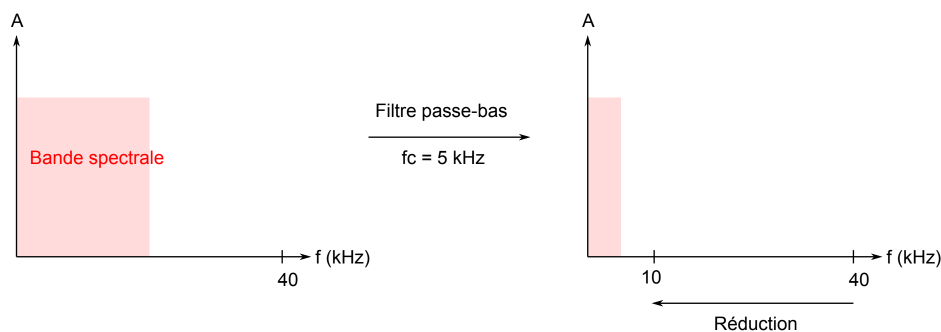
Cette fonction retourne dès que l'acquisition est lancée (voir exemple d'utilisation ci-dessous).

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]
```

10.2.b. Sur-échantillonnage et filtrage

Voici un exemple d'application du mode permanent. On fait l'acquisition d'un son à 40 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 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 :



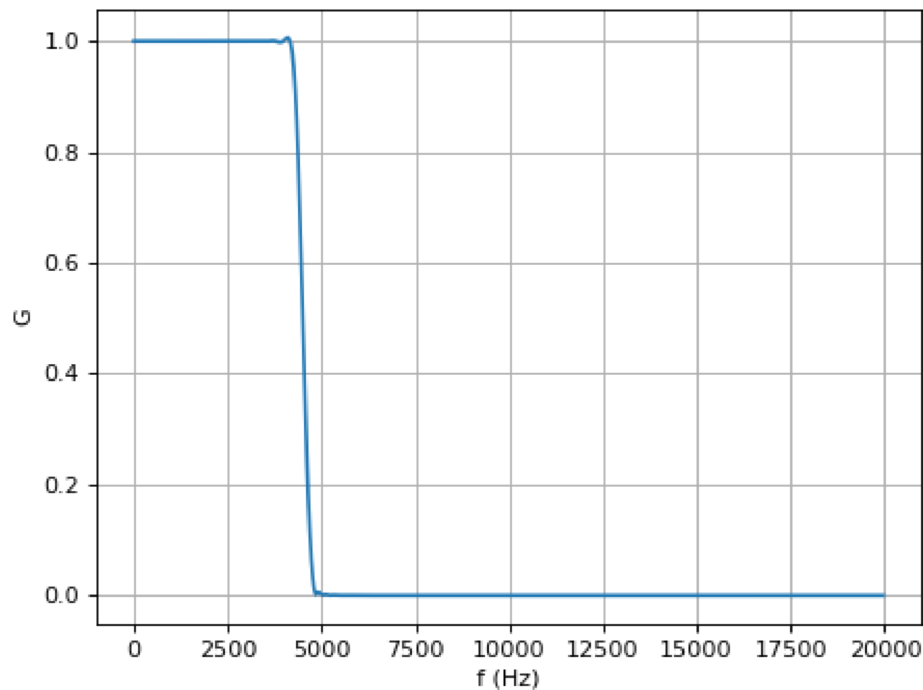
On définit un filtre RIF avec une fréquence de coupure inférieure à 5 kHz :

```
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)
```

10.2.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=F
show(block=True)
numpy.savetxt("data.txt",[t,u])
sys.fermer()
```

10.3. 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*1
ani1 = animation.FuncAnimation(fig1,animate1,frames=100,repeat=True,interval=duree*1
show(block=True)
sys.stopper_acquisition()
time.sleep(1)
sys.fermer()
```

Chapitre 11

Filtrage en temps réel d'un signal audio

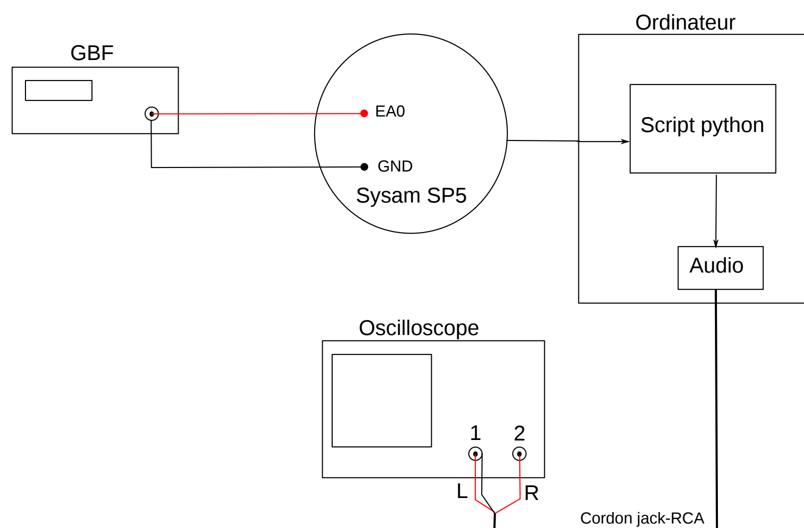
11.1. Introduction

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.

11.2. Montage expérimental

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



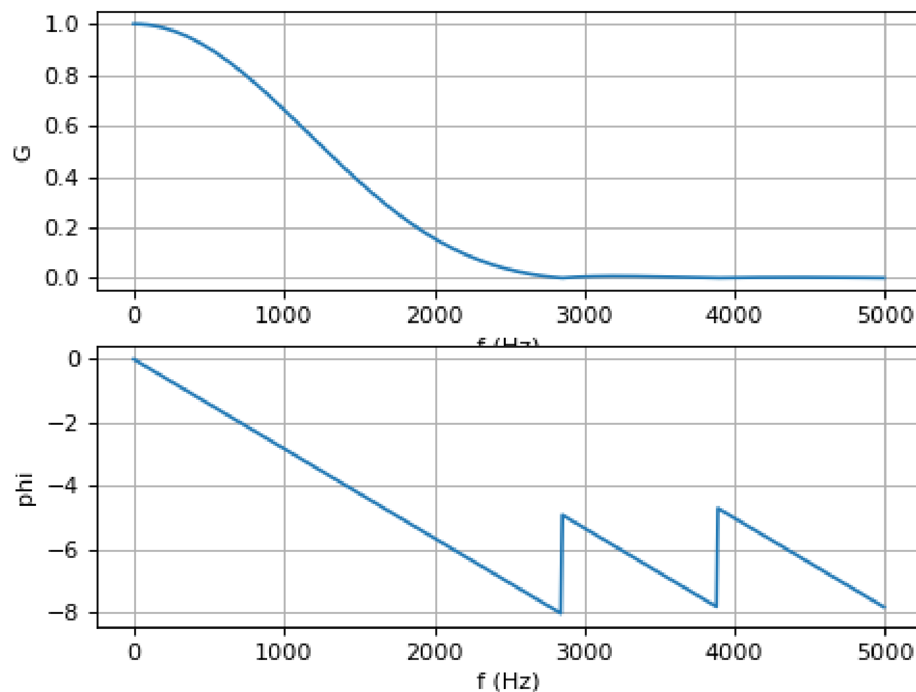
Un générateur de fonctions (GBF) délivre une sinusoïde dont l'acquisition est faite sur la voie EA0 de la carte SysamSP5. Le script python effectue le filtrage en temps réel. Le signal non filtré et le signal filtré sont envoyés sur les deux canaux (R et L) de la sortie audio, pour une observation à 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).

11.3. 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).

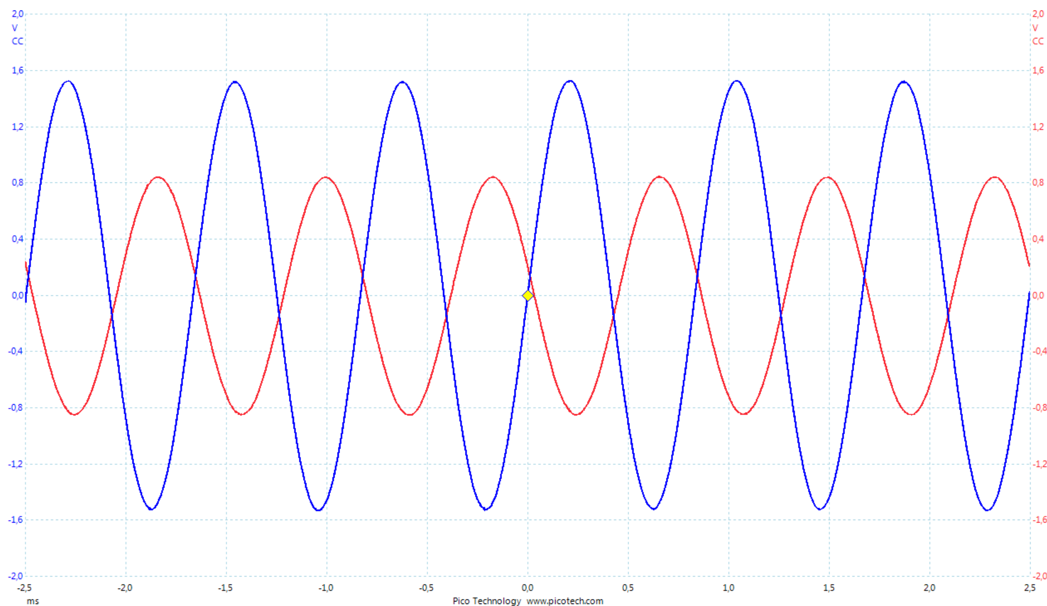
Voici le script complet, qui effectue le filtrage d'un nombre déterminé de paquets. Le signal non filtré et le signal filtrés sont redirigés vers les deux voies R et L de la sortie audio.

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 V. Ce gain dépend bien sûr du réglage du volume. Pour une amplitude en entrée supérieure à 2,1 V (crête à crête), il se produit un écrêtage.

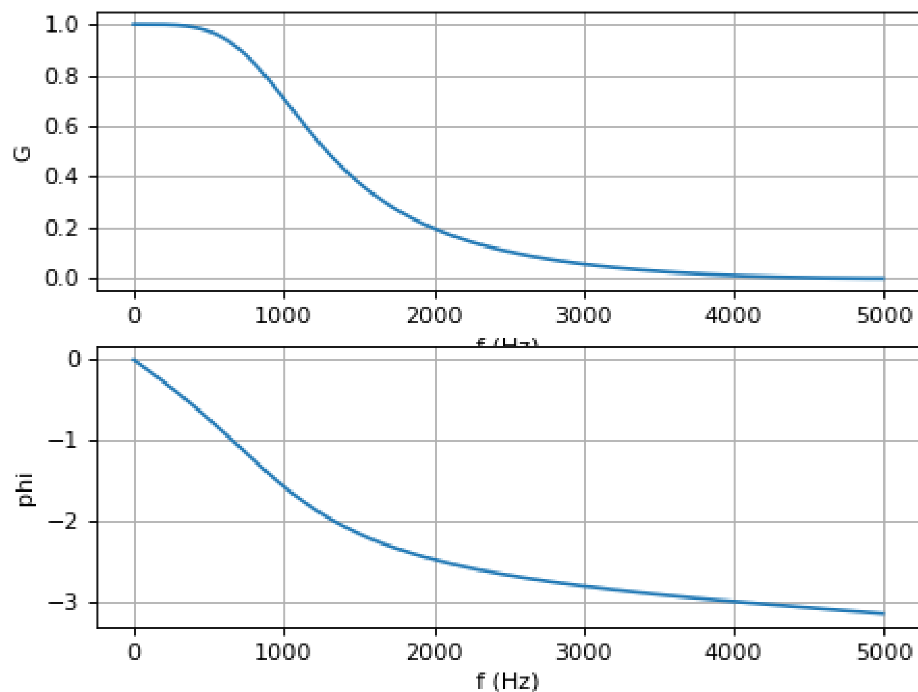
On remarque la qualité du lissage effectué par le circuit audio, alors que la fréquence d'échantillonnage est seulement 8 fois plus grande que la fréquence du signal.

11.4. 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.

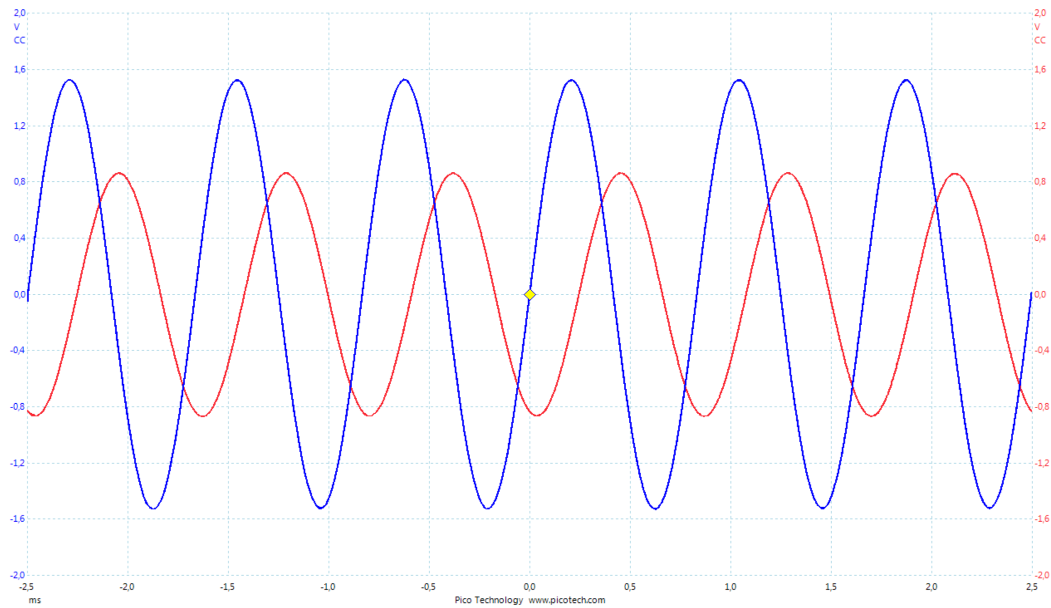
filtrageBIQUADSortieAudio.py

```
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 Hz avec une amplitude de crête à crête de 2 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 $40\,kHz$ avec $N = 1000$ fonctionne bien. Le délai est alors de $25\,ms$.