Capacité numérique : réaliser, à l'aide d'un langage de programmation, un filtrage numérique d'un signal issu d'une acquisition, et mettre en évidence la limitation introduite par l'échantillonnage.

Ouvrir un fichier txt ou csv

Les logiciels, comme LatisPro, proposent l'exportation de données numériques au format txt (texte) ou csv, « coma separated values » (tableur). Les données peuvent être séparées par des virgules ou des points-virgules. Nous supposerons qu'on a choisi de les séparer par des points-virgules. La lecture d'un fichier `essai.txt` révèle alors la structure suivante :

```
# temps ;signal
0;-0.000499061308801174
1E-6;-0.000499061308801174
2E-6;-0.000499061308801174
3E-6;-0.000499061308801174
```

Il est constitué d'une première ligne de commentaires, puis de *N* lignes de données. Le symbole dièse (#) n'est pas toujours généré pour une ligne de commentaire. On peut alors l'ajouter dans le fichier txt ou csv afin que cette ligne soit ignorée grâce au code d'extraction que nous allons créer.

On cherche à obtenir un tableau numpy de deux colonnes contenant respectivement les valeurs du temps et du signal, et des *N* lignes de données.
Il faut commencer par ouvrir le fichier :

```
>>> with open('C:/Users/kalou/Desktop/tp_4/essai.txt','r') as F:
...     for ligne in F:
...         print(ligne)
...
...
#temps;signal

0;-0.000499061308801174

1E-6;-0.000499061308801174

2E-6;-0.000499061308801174

3E-6;-0.000499061308801174
```

La commande `with open('C:/Users/kalou/Desktop/tp_4/essai.txt','r')` ouvre le fichier puis le ferme après les différentes actions spécifiées. L'argument est le nom du fichier, avec son chemin d'accès (les dossiers sont séparés par `/` et pas `\`). Le chemin d'accès peut ne pas être spécifié si le fichier txt est placé dans le même dossier que le fichier Python qui l'ouvre).

L'objet obtenu, nommé `F`, est un fichier ouvert qui peut être parcouru ligne par ligne.

L'option `'r'` permet de l'ouvrir en lecture seule et donc de le protéger en écriture.
On peut au contraire l'option `'w'` pour écrire dans un fichier. Si ce fichier n'existe pas encore, il sera créé, mais s'il existe déjà, ses données seront écrasées :

```
>>> L = ['#temps;signal','0;-0.0004990613088011741','1E-6;-
0.0004990613088011741']
>>> with open('C:/Users/kalou/Desktop/tp_4/essai_1.txt','w') as F:
...     for elt in L:
...         F.write(elt)
...
...
13
23
26
```

Les valeurs affichées (13, 23 et 26) sont les nombre d'octets (nombre de caractères) ajoutés successivement au fichier.

Une surprise nous attent cependant à la lecture du fichier `essai_1.txt` :

```
>>> with open('C:/Users/kalou/Desktop/tp_4/essai_1.txt','r') as F:
...     for ligne in F:
...          print(ligne)
...
...
#temps;signal0;-0.000499061308801174E-6;-0.0004990613088011741
```

Il n'y a qu'une seule ligne ! La donnée de la fin d'une ligne n'est plus séparée de celle du début de la ligne suivante. C'est normal : il faut rajouter le caractère `'\n'` de fin de ligne :

```
>>> with open('C:/Users/kalou/Desktop/tp_4/essai_1.txt','w') as F:
...     for elt in L:
...         F.write(elt+'\n')
...
...
14
24
27
```

Cette fois-ci, tout va bien :

```
>>> with open('C:/Users/kalou/Desktop/tp_4/essai_1.txt','r') as F:
...     for ligne in F:
...         print(ligne)
...
...
#temps;signal

0;-0.0004990613088011741

1E-6;-0.0004990613088011741
```

Revenons à l'extraction des données. On peut commencer par écrire une fonction qui renvoie le nombre de lignes de données (en ne prenant pas en compte les lignes de commentaires) :

```
def taille(nom_du_fichier):
    with open(nom_du_fichier,'r') as F:
        l = 0
        for elt in F:
            if '#' not in elt:
                l +=1
    return l
```

On passe alors à l'extraction proprement dite :

```
def extraction_donnees(nom_du_fichier):
    N = taille(nom_du_fichier)
    A = []
    with open(nom_du_fichier,'r') as F:
        for elt in F:
            if '#' not in elt:
                A += [elt.replace('\n','')] # On enlève le caractère
de fin de ligne. On obtient une liste de chaines de caractères comme
par exemple ['0;-0.0004499061308801174', '1E-6;-0.0004499061308801174',
'2E-6;-0.0004499061308801174', '3E-6;-0.0004499061308801174']
    donnees = np.zeros((N,2)) # A adapter s'il y a plus de 2 colonnes
    for i in range(N):
        a,b = A[i].split(';')
        donnees[i,0],donnees[i,1] = float(a.replace(',','.')),float
(b.replace(',','.')) # On transforme les chaines de caractères en
flottants après avoir remplacé les éventuelles virgules par des points
    return donnees
```

La méthode `split` prend pour argument une chaine de caractères et la découpe en plusieurs chaines. Elle coupe aux endroits où se trouve le délimiteur spécifié (par défaut un espace) et renvoie une liste des chaines de caractères obtenues :

```
>>> '0;-0.0004499061308801174'.split(';')
['0', '-0.0004499061308801174']
```

La conversion en flottants est en fait inutile car `donnees` a été créée comme un tableau numpy de flottants. De même, les décimales des données numériques étaient ici déjà séparées de la partie entière par un point et pas une virgule.

L'extraction des données d'un fichier csv se fait de façon identique.

Nous allons maintenant procéder au filtrage numérique passe-bas du premier ordre d'un signal d'entrée sinusoïdal $e_1(t) = 5\sin(2\pi f_0 t)$ (en V) de fréquence $f_0 = 10$ kHz, *issu d'une acquisition*, et comparer avec le signal de sortie filtré analogiquement à l'aide d'une cellule RC : $s_{an}(t)$, et dont on a également fait l'acquisition. Les fichiers correspondants sont fournis et s'appellent : `entree_sinus_10kHz.txt` et `sortie_sinus_10kHz.txt`.

On a prélevé $N = 20000$ échantillons avec une période d'échantillonnage $T_e = 5$ µs.

Les valeurs utilisées sont $R = 10$ kΩ, $C = 10$ nF.

Q.1) Sous Python, entrer ces valeurs et calculer la durée d'acquisition T_a, la fréquence de coupure du filtre f_c, et le temps de réponse $\tau = RC$.

Extraire les données et créer deux tableaux numpy à 1D : le tableau `T` des valeurs des instants d'échantillonnage (en s), celui `E1` des valeurs du signal d'entrée (en V), et celui `S1` des valeurs du signal de sortie (en V)

Q.2) Une première méthode de filtrage numérique consiste à résoudre numériquement l'équation différentielle associée au filtre, soit ici $H(jf)=\dfrac{\underline{s}}{\underline{e}}=\dfrac{H_0}{1+jf/f_c}\leftrightarrow\dfrac{\mathrm{d}s}{\mathrm{d}t}+\tau s=H_0\tau e$, par la méthode d'Euler.

La solution cherchée est notée s_{euler}.

Exprimer $s_{\text{euler}}(t_k)$, valeur à la date $t_k=kT_\text{e}$, en fonction de $s_{\text{euler}}(t_{k-1})$ et $e(t_{k-1})$.

On prend $s_{\text{euler}}(0)=1\ \text{V}$. Expliquer pourquoi cette valeur n'influe pas sur la réponse cherchée.

Remplir le tableau numpy $\left[s_{\text{euler}}(0),s_{\text{euler}}(T_\text{e}),s_{\text{euler}}(2T_\text{e}),...,s_{\text{euler}}(kT_\text{e}),...,s_{\text{euler}}(t_{N-1})\right]$.

Q.3) Une deuxième méthode de filtrage consiste à travailler dans le domaine fréquentiel plutôt que temporel.

On calcule le spectre de l'entrée et on effectue le produit de chaque composante complexe de fréquence $\dfrac{n}{T_\text{a}}$ par la fonction de transfert à cette fréquence : la réponse exacte à un signal périodique $\underline{e}(t)=\sum_{n=-\infty}^{+\infty}C_n e^{\frac{2i\pi nt}{T_\text{a}}}$ est $\underline{s}(t)=\sum_{n=-\infty}^{+\infty}C_n\cdot H\left(j\dfrac{n}{T_\text{a}}\right)\cdot e^{\frac{2i\pi nt}{T_\text{a}}}$.

Transformée de Fourier directe et inverse sous numpy

Le module `fft` de numpy permet d'effectuer des transformées de Fourier discrètes directes et inverse.

`tfe = np.fft.fft(E)` calcule les coefficients $\left[C_0,C_1,C_2,...,C_{\frac{N}{2}-1},C_{\frac{N}{2}},...,C_{N-1}\right]$ de la transformée de Fourier discrète directe du signal $E=\left[e(0),e(T_\text{e}),e(2T_\text{e}),...,e(kT_\text{e}),...,e(t_{N-1})\right]$.

Ces coefficients correspondent aux fréquences $\left[0,\dfrac{1}{T_\text{a}},\dfrac{2}{T_\text{a}},...,\left(\dfrac{N}{2}-1\right)\dfrac{1}{T_\text{a}},\dfrac{N}{2}\dfrac{1}{T_\text{a}},...,\dfrac{N-1}{T_\text{a}}\right]$.

On rappelle que N est pair et que le spectre obtenu est f_e – périodique du fait de l'échantillonnage, avec $f_\text{e}=1/T_\text{e}=N/T_\text{a}$. Les coefficients $\left[C_{\frac{N}{2}},...,C_{N-1}\right]$ correspondent donc aux fréquences négatives $\left[-\dfrac{N}{2}\dfrac{1}{T_\text{a}},...,-\dfrac{1}{T_\text{a}}\right]$.

Rappelons aussi que dans le cas où la fonction étudiée est réelle et périodique, on doit multiplier C_0 par $\dfrac{1}{N}$ et les C_n par $\dfrac{2}{N}$, pour $n\in\left[\!\left[1,\dfrac{N}{2}-1\right]\!\right]$, afin d'obtenir les bonnes amplitudes spectrales.

`tfi = np.fft.ifft(tfe)` calcule $\left[e(0),e(T_\text{e}),e(2T_\text{e}),...,e(kT_\text{e}),...,e(t_{N-1})\right]$ à partir de $\left[C_0,C_1,C_2,...,C_{N-1}\right]$: transformée de Fourier discrète inverse.

Par exemple :

```
>>> A = [1.0,-1.0,2.0,5.0]
>>> B = np.fft.fft(A)
>>> B
array([ 7.+0.j, -1.+6.j, -1.+0.j, -1.-6.j])
>>> C = np.fft.ifft(B).real
>>> C
array([ 1., -1., 2., 5.])
```

Les calculs étant approchés, les parties imaginaires de la transformée de Fourier inverse peuvent être non nulle alors que le signal est réel. On peut alors extraire les parties réelles.

On obtient la liste des fréquences du spectre obtenus à partir de N échantillons d'un signal pris tous les T_e avec la commande :

`np.fft.fftfreq(N,Te)`

Par exemple :

```
>>> np.fft.fftfreq(10,0.1)
array([ 0., 1., 2., 3., 4., -5., -4., -3., -2., -1.])
```

On a exploité la périodicité du spectre pour ne faire intervenir que les fréquences telles que $|f| \leq \frac{N}{2}\frac{1}{T_a} = \frac{f_e}{2} = f_N$, fréquence de Nyquist.

Calculer la réponse $[s_{fourier}(0), s_{fourier}(T_e), s_{fourier}(2T_e), ..., s_{fourier}(kT_e), ..., s_{fourier}(t_{N-1})]$ par cette méthode.

Q.4) Extraire les valeurs $[s_{an}(0), s_{an}(T_e), s_{an}(2T_e), ..., s_{an}(kT_e), ..., s_{an}(t_{N-1})]$ de la sortie analogique. Tracer sur le même graphe les courbes de $t \mapsto e(t)$, $t \mapsto s_{euler}(t)$, $t \mapsto s_{fourier}(t)$ et de $t \mapsto s_{an}(t)$. Commenter.

Q.5) Pour mieux pouvoir faire varier les paramètres, on simule l'entrée et la sortie analogique :

```
def entree(t):
    return 5*np.sin(2*np.pi*f0*t)

def sortie(t): # sortie "analogique"
    G = H0/(1+(f0/fc)**2)**0.5
    phi = - np.arctan(f0/fc)
    return G*5*np.sin(2*np.pi*f0*t+phi)
```

Tracer les courbes de $t \mapsto e(t)$, $t \mapsto s_{euler}(t)$, $t \mapsto s_{fourier}(t)$ et de $t \mapsto s_{an}(t)$.
Faire varier la fréquence du signal ainsi que la période d'échantillonnage.
Quelles sont les limites du filtrage numérique ?