Filtrage numérique

Capacités exigibles

- Réaliser, à l'aide d'un langage de programmation, un filtrage numérique passe-bas d'un signal issu d'une acquisition et mettre en évidence la limitation introduite par l'échantillonnage.

I Énoncé

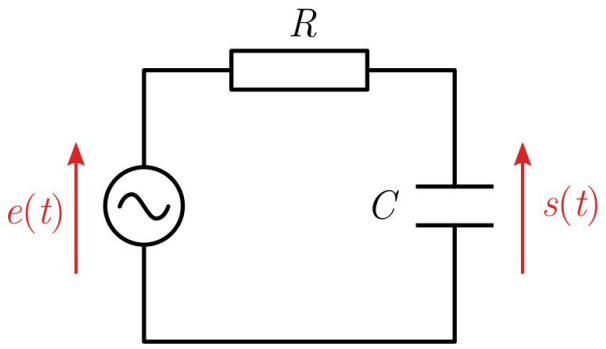

Figure 1 – Circuit RC

Théorie :

1. Rappelez (ou re-démontrez sinon) l'équation différentielle suivie par la sortie $s(t)$.
2. Rappelez la fonction d'un tel montage et donnez en son diagramme de Bode

Programmation :

Téléchargez le fichier python disponible sur l'ENT (visible en Figure 2 à la fin du sujet) puis complétez-le en suivant les étapes suivantes :

3. Compléter la 26 en utilisant les variables du code
4. Complétez la ligne 31 pour que que le signal d'entrée soit un sinus de fréquence égale à 1 Hz **bruité**.
   Vous pourrez utiliser pour cela la fonction `np.random.random(n)` qui créer un tableau de `n` valeurs aléatoires uniformément choisies entre 0 et 1.
5. Avant l'instant $t = 0$, le circuit est fermé. En déduire la condition initiale à intégrer ligne 33.
6. Complétez la ligne 36 donnant la valeur de $s(t + \mathrm{d}t)$ à partir de $s(t)$.
7. Remplissez alors les lignes 41 et 42 pour obtenir le gain en décibel $G_{\mathrm{dB}}(f)$.
8. La ligne 52 permet d'afficher l'évolution temporelle du signal d'entrée. Ajoutez une ligne permettant de visualiser $s(t)$.

Vérification :

9. Jouez avec le signal d'entrée (modifiez le bruit, ajoutez des composantes harmoniques...) et vérifiez que votre modélisation agit bien comme un passe-bas.

Pour aller plus loin :
Un bruit blanc est un signal complètement aléatoire. Son spectre contient toutes les fréquences à la même amplitude.

10. En utilisant cette notion de bruit blanc, trouvez un moyen de faire apparaître sur `ax2` le diagramme de Bode de votre filtre.

II Annexe

```python
## Import des modules

import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft, fftshift


## Définition des fonctions

def abs(array):
    '''Calcule la valeur absolue d'un tableau de complexes'''
    return np.sqrt( array.real ** 2 + array.imag ** 2 )

def get_tf(signal):
    '''Renvoie la transformée de Fourier d'un signal'''
    return abs(fftshift(fft(signal))[N//2:])


## Calculs

N = 10000           # Nombre de points de l'échantillonnage temporel
t_max = 6 * np.pi   # Durée de "l'acquisition"

RC = 1      # Produit RC
dt = 0.01   # Incrément temporel
fe = ?      # Fréquence d'échantillonage

t = np.linspace(0, t_max, N)        # Tableau des instants (abscisse du signal
    temporel)
f = np.linspace(0, .5 * fe, N/2)    # Tableau des fréquences (abscisse du spectre)

e = ?                   # Signal d'entrée
s = np.zeros(N)         # Signal de sortie (pour l'instant nul)
s[0] = ?                # Condition initiale pour la sortie

for k in range(0, N - 1):
    s[k+1] = ?          # Propagation par méthode d'Euler

tf_e = get_tf(e)        # Amplitude du spectre de l'entrée
tf_s = get_tf(s)        # Amplitude du spectre de sortie

H = ?   # Valeur absolue de la fonction de transfert
G = ?   # Gain en décibel


## Tracé de la figure

fig = plt.figure()
ax1 = fig.add_subplot(121)      # ax1 servira à afficher les signaux temporels
ax2 = fig.add_subplot(122)      # ax2 servira à afficher les spectres des signaux
ax2.set_xscale('log')

ax1.plot(t, e)
# ...

plt.show()
```

Figure 2 – Code principal