Physique

Capacité numérique 2 – Action d'un filtre sur un signal périodique

Pour le lundi 29 janvier 10:00

Objectifs

- Simuler, à l'aide d'un langage de programmation, l'action d'un filtre sur un signal périodique dont le spectre est fourni. Mettre en évidence l'influence des caractéristiques du filtre sur le filtrage.

✎ *Afin d'utiliser les modules* `numpy` *et* `matplotlib.pyplot`*, importez-les dès le début de votre code avec :*

```
4 import numpy as np
5 import matplotlib.pyplot as plt
```

1 Réponse en régime sinusoïdal d'un passe-bas d'ordre 1

Soit un filtre qui à un signal d'entrée $e(t)$ donne un signal de sortie $s(t)$. Ce filtre est décrit par une fonction de transfert $\underline{H}(\mathrm{j}\omega)$ telle que, en notation complexe,

$$\underline{S} = \underline{H}\,\underline{E}$$

avec $\underline{S}$ et $\underline{E}$ les amplitudes complexes respectives de s et e.

1. Dans le cas où $e(t) = E\cos(\omega t + \varphi_e)$, comment s'exprime $\underline{E}$ en fonction de E et φ_e ?

Il s'agit de passer en notation complexe,

$$e(t) = E\cos(\omega t + \varphi_e) \rightsquigarrow \underline{e}(t) = E\exp(\mathrm{j}(\omega t + \varphi_e)) = \underline{E}\exp(\mathrm{j}\omega t) \Rightarrow \boxed{\underline{E} = E\exp(\mathrm{j}\varphi_e)}.$$

2. Écrire le code définissant la fonction `sinusoide`, prenant comme paramètres `A` l'amplitude, `omega` la pulsation, `phi` la phase à l'origine et `t` un `np.array` correspondant au temps en seconde et qui rend un `np.array` contenant les valeurs prises par $A\cos(\omega t + \varphi)$ aux instants t compris dans `t`.

Le code est le suivant. Les indications sur les types des objets donnés en entrée à la fonction et de celui ou ceux rendus par la fonction sont données à titre informatif pour l'utilisateur (cela pourrait être fait avec des commentaires).

```
7 def sinusoide(A : float, omega : float, phi : float, t : np.ndarray) -> np.ndarray:
8     return A * np.cos(omega * t + phi)
```

3. Rappeler la forme canonique de la fonction de transfert $\underline{H}(\mathrm{j}\omega)$ d'un passe-bas d'ordre 1. Faire apparaître, en particulier, le gain en amplitude statique H_0 et la pulsation de coupure ω_0. Dans la suite, H_0 sera fixé à 1.

Voir le cours si vous ne le savez plus (pas pour l'apprendre, mais pour savoir la retrouver) ! La forme canonique de la fonction de transfert $\underline{H}(\mathrm{j}\omega)$ d'un passe-bas d'ordre 1 est

$$\boxed{\underline{H}(\mathrm{j}\omega) = \frac{H_0}{1 + \mathrm{j}\frac{\omega}{\omega_0}}}.$$

4. Écrire le code définissant la fonction de transfert générique du passe-bas d'ordre 1 `H_PasseBas1`, prenant comme paramètres `omega` la pulsation à laquelle évaluer $\underline{H}(\mathrm{j}\omega)$ et `omega0` la pulsation de coupure et qui rend un nombre complexe étant la valeur de $\underline{H}(\mathrm{j}\omega)$. Pour rappel, `1j` correspond en python à l'imaginaire pur j tel que $\mathrm{j}^2 = -1$.

Le code est le suivant.

```
10 def H_PasseBas1(omega : float, omega0 : float) -> complex:
11     return 1 / (1 + 1j * omega/omega0)
```

5. Écrire le code définissant la fonction de transfert du filtre passe-bas d'ordre 1 ayant comme *fréquence* de coupure $f_{c1} = 500\,\mathrm{Hz}$, `Filtre_1`, prenant comme paramètres `omega` la pulsation à laquelle évaluer cette fonction de transfert un nombre complexe étant la valeur de cette fonction de transfert.

Le code est le suivant.

```
13 def Filtre_1(omega : float) -> complex:
14     fc = 500 # fréquence de coupure
15     omega0 = 2*np.pi * fc # pulsation de coupure
16     return H_PasseBas1(omega, omega0) # valeur pour un passe-bas d'ordre 1 à omega
       ↪  pour cette valeur de omega0
```

6. Dans le cas d'un filtre linéaire, le signal de sortie est de la forme $s(t) = S\cos(\omega t + \varphi_s)$. Comment s'exprime $\underline{S}$ en fonction de S et φ_s ?

Il s'agit à nouveau de passer en notation complexe,

$$s(t) = S\cos(\omega t + \varphi_s) \rightsquigarrow \underline{s}(t) = S\exp(\mathrm{j}(\omega t + \varphi_s)) = \underline{S}\exp(\mathrm{j}\omega t) \Rightarrow \boxed{\underline{S} = S\exp(\mathrm{j}\varphi_s)}.$$

7. Sachant que $\underline{S} = \underline{H}\,\underline{E}$, comment déterminer S et φ_s à l'aide de $\underline{H}$ et $\underline{E}$?

Il suffit de prendre le module et l'argument de $\underline{S}$. Ainsi,

$$S = |\underline{S}| = |\underline{H}\,\underline{E}| = |\underline{H}||\underline{E}| \Rightarrow \boxed{S = HE}, \quad \varphi_s = \arg(\underline{H}\,\underline{E}) = \arg(\underline{H}) + \arg(\underline{E}) \Rightarrow \boxed{\varphi_s = \arg(\underline{H}) + \varphi_e}.$$

8. Écrire alors $s(t)$ à partir de E, ω, φ_e, $|\underline{H}|$ et $\arg(\underline{H})$.

Compte-tenu des questions précédentes,

$$\boxed{s(t) = |\underline{H}|E\cos(\omega t + \arg(\underline{H}) + \varphi_e)}.$$

9. Écrire le code définissant la fonction `sinusoide_filtree`, prenant comme paramètres `A` l'amplitude de la sinusoïde *non filtrée*, `omega` la pulsation, `phi` la phase à l'origine de la sinusoïde *non filtrée*, `t` un `np.array` correspondant au temps en seconde et `H` une fonction de transfert *elle-même fonction de* ω et qui rend un `np.array` contenant les valeurs prises par la sinusoïde en sortie du filtre. Utiliser dans `sinusoide_filtree` la fonction `sinusoide` précédemment définie.

Pour rappel, le module d'un nombre complexe `z` s'obtient avec `np.abs(z)`, son argument avec `np.angle(z)`.

Vous pourrez vérifier votre code avec les deux cas suivants :

- pour une fonction de transfert valant toujours 1, la sinusoïde filtrée est la même que la sinusoïde non filtrée ($s = e$) ;
- pour une fonction de transfert valant toujours $\frac{1}{2}\mathrm{j}$, la sinusoïde filtrée correspond à celle en entrée mais d'amplitude divisée par 2 et en avance de $\pi/2$.

Le code définissant la fonction `sinusoide_filtree` est le suivant.

```python
def sinusoide_filtree(A : float, omega : float, phi : float, t : np.ndarray, H :
    ↪ callable) -> np.ndarray:
    S = np.abs( H(omega) ) * A
    phiS = np.angle( H(omega) ) + phi
    return sinusoide(S, omega, phiS, t)
```

2 Série de Fourier pour un signal périodique

Un signal périodique n'est pas sinusoïdal. En revanche, le théorème de Fourier permet, en général, de décomposer un signal périodique en une somme de sinusoïdes.

10. Rappeler l'énoncé du théorème de Fourier pour un signal T-périodique.

Théorème de Fourier Soit un signal physique de période $T = 1/\nu$. Son caractère physique implique qu'il soit continu par morceaux et de carré intégrable. Il peut alors être décomposé en la somme :
- d'un terme constant égal à sa valeur moyenne ;
- d'une composante sinusoïdale de fréquence ν, le *fondamental* ;
- de composantes sinusoïdales de fréquences multiples de ν, les *harmoniques*.

✎ *La série de Fourier ainsi obtenue converge en norme L^2 vers le signal considéré car ce dernier est de carré intégrable.*

Pour un signal C créneau d'amplitude E, c'est-à-dire

$$C(t) = \begin{cases} +E \text{ si } nT < t < \left(n+\frac{1}{2}\right)T \\ -E \text{ si } \left(n+\frac{1}{2}\right)T < t < (n+1)T \\ 0 \text{ sinon} \end{cases}, \quad n \in \mathbb{Z},$$

il est possible de montrer que

$$C_n(t) = \sum_{k=0}^{n} c_k(t) = \sum_{k=0}^{n} \underbrace{\frac{4E}{(2k+1)\pi} \sin\left(2\pi(2k+1)\frac{t}{T}\right)}_{c_k(t)} \underset{n\to\infty}{\longrightarrow} C(t)\,. \tag{1}$$

Dans la suite, nous fixerons la *fréquence* de ce créneau à 440 Hz et $E = 1$ SI.

✎ *La fonction C_n ainsi définie correspond aux $2n+1$ premiers harmoniques de la décomposition en série de Fourier du créneau (impair et de moyenne nulle).*

11. Écrire le code de la fonction `Creneau`, prenant comme paramètres `E` l'amplitude du créneau $C(t)$, `T` la période du créneau $C(t)$, `t` un `np.array` correspondant au temps en seconde et qui rend un `np.array` contenant les valeurs prises par $C(t)$ en utilisant une boucle `for`. Il est possible de créer un `np.array` de longueur `L` et contenant uniquement des valeurs nulles grâce à `np.zeros(L)`.

Le code définissant la fonction `Creneau` est le suivant. Il existe d'autre manière d'y parvenir, l'essentiel est que vous puissiez tracer ce créneau exact.

```python
def Creneau(E : float, T : float, t : np.ndarray) -> np.ndarray:
    C = np.zeros(len(t)) # initialiser le créneau à 0
    # boucle for pour déterminer les valeurs du créneau à chaque instant
    for t_value in t :
        # si t_value n'est pas entre 0 et T,
        # se ramener à ce cas grâce à la périodicité du créneau
        # et alors lire l'énoncé dans le cas n = 0
        t_eval = t_value - t_value//T * T
        # il est aussi possible d'utiliser des boucles while
```

```
        if 0 < t_eval and t_eval < T/2:
            C[t == t_value] = E
        elif T/2 < t_eval and t_eval < T:
            C[t == t_value] = -E
        # sinon, la valeur est déjà nulle, mais il est possible de mettre
        # else:
        #     C[t_value] = 0
    return C
```

12. Exprimer $c_k(t)$ à l'aide d'un cosinus et non d'un sinus. En déduire la fonction `C_n`, prenant comme paramètres `n` un entier, `E` l'amplitude du créneau $C(t)$, `T` la période du créneau $C(t)$, `t` un `np.array` correspondant au temps en seconde et qui rend un `np.array` contenant les valeurs prises par $C_n(t)$ en utilisant la fonction `sinusoide`.

D'après l'énoncé,

$$c_k(t) = \frac{4E}{(2k+1)\pi} \sin\left(2\pi(2k+1)\frac{t}{T}\right).$$

Or, $\cos\left(\theta - \frac{\pi}{2}\right) = \sin(\theta)$. Ainsi,

$$\boxed{c_k(t) = \frac{4E}{(2k+1)\pi} \cos\left(2\pi(2k+1)\frac{t}{T} - \frac{\pi}{2}\right)}.$$

La fonction `C_n` est ainsi définie par le code suivant.

```
def C_n(n : int, E : float, T : float, t : np.ndarray) -> np.ndarray:
    Cn = np.zeros(len(t)) # initialiser le résultat
    # attention à ne pas le nommer comme la fonction elle-même !

    # boucle for pour faire la somme
    for k in range(0, n+1):
        A = 4*E / ( (2*k+1) * np.pi ) # amplitude de c_k
        omega = 2*np.pi/T * (2*k+1) # déterminer omega pour utiliser sinusoide
        phi = - np.pi/2 # pour le passage de sin à cos

        Cn += sinusoide(A, omega, phi, t)

    return Cn
```

13. Écrire un code permettant de tracer sur une même figure, sur 2 périodes, $C(t)$ le créneau exact et $C_n(t)$ pour $n = 0$ et de sauvegarder ce tracé dans une image « `fig1.png` ».

Le code est le suivant.

```
T = 1/440 # période pour une fréquence de 440 Hz
E = 1 # E vaut 1 SI

# Pour tracer sur deux périodes en prenant 1000 points
time_s = np.linspace(0, 2*T, 1000)

C_fct_t = Creneau(E, T, time_s)
C_0_fct_t = C_n(0, E, T, time_s)

plt.figure()
plt.title("$C$ et $C_0$ sur deux périodes")
plt.plot(time_s*1e3, C_fct_t, label = "Créneau exact")
plt.plot(time_s*1e3, C_0_fct_t, label = "$C_0(t)$")
plt.xlim([time_s.min()*1e3, time_s.max()*1e3]) # ajustement des bornes de l'axe des
↪ abscisses
```

```
plt.xlabel("temps (ms)")
plt.ylim([-1.75, 1.75]) # ajustement des bornes de l'axe des ordonnées
plt.ylabel("Valeurs en SI")
plt.grid() # pour afficher le quadrillage
plt.legend() # afficher les labels des courbes
plt.savefig("fig1.png")
```

Le tracé obtenu est donné en figure 1.

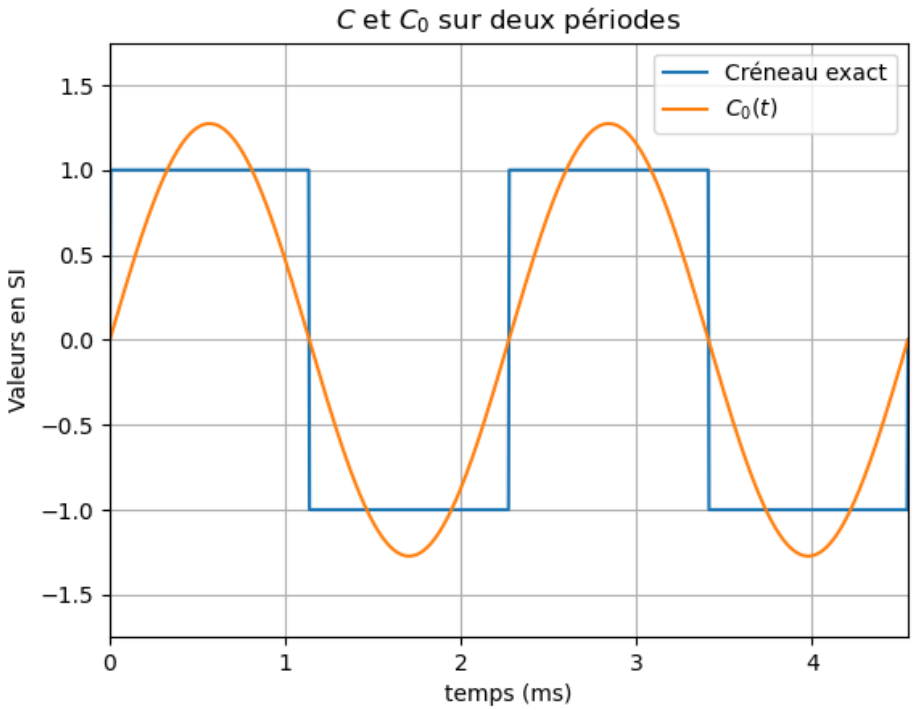


Figure 1 – *Le tracé « `fig1.png` ».*

14. Faire de même pour $n = 1, 2, 500$ dans des images « `fig2.png` », « `fig3.png` », « `fig4.png` ».
Le code est le suivant. Une boucle est utilisée, il est mieux d'y penser !

```
fig_num = 1
for n in [1, 2, 500]:
    fig_num += 1

    C_n_fct_t = C_n(n, E, T, time_s)

    plt.figure()
    plt.title(f"$C$ et $C_{{ {n} }}$ sur deux périodes")
    plt.plot(time_s*1e3, C_fct_t, label = "Créneau exact")
    plt.plot(time_s*1e3, C_n_fct_t, label = f"$C_{{ {n} }}(t)$")
    plt.xlim([time_s.min()*1e3, time_s.max()*1e3]) # ajustement des bornes de l'axe
    ↪ des abscisses
    plt.xlabel("temps (ms)")
    plt.ylim([-1.75, 1.75]) # ajustement des bornes de l'axe des ordonnées
    plt.ylabel("Valeurs en SI")
    plt.grid() # pour afficher le quadrillage
    plt.legend() # afficher les labels des courbes
    plt.savefig(f"fig{fig_num}.png")
```

Le tracés obtenus sont donnés en figures 2, 3 et 4.

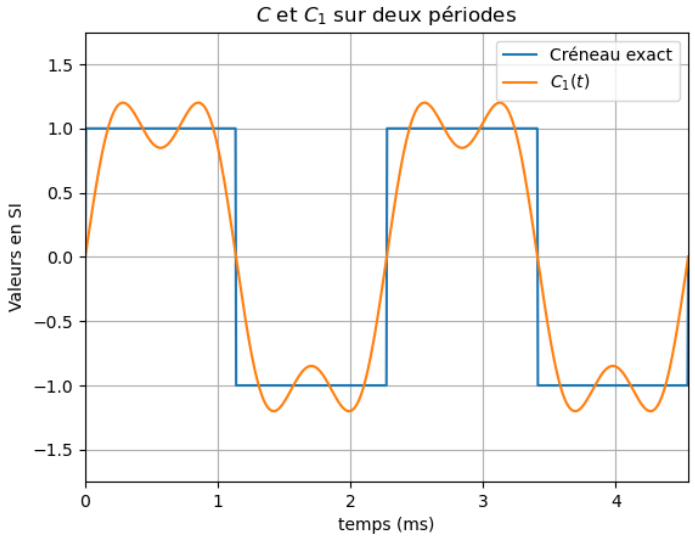


Figure 2 – *Le tracé « `fig2.png` ».*

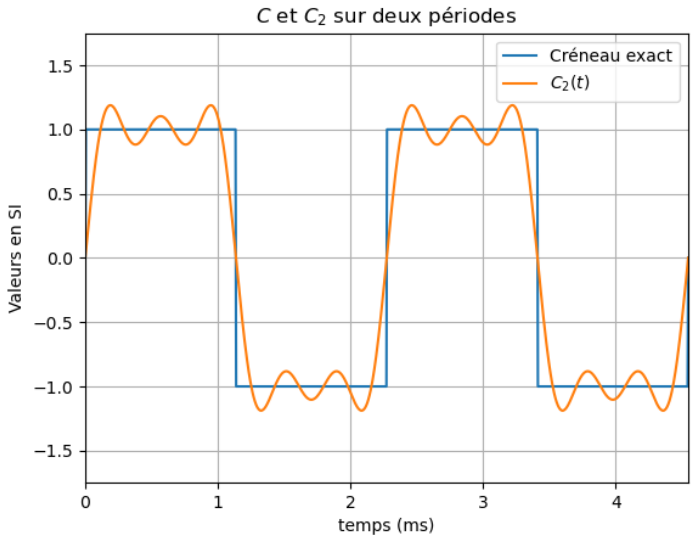


Figure 3 – *Le tracé « `fig3.png` ».*

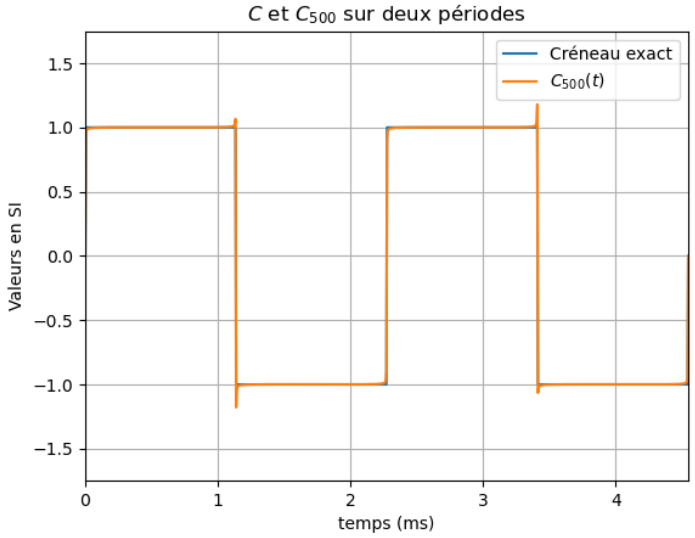


Figure 4 – *Le tracé « `fig4.png` ».*

3 Réponse du passe-bas d'ordre 1 à un signal périodique

Le filtre étant linéaire, « la sortie de la somme est la somme des sorties ».

15. Si s_k est la sortie du filtre pour la sinusoïde c_k (définie par (1)), exprimer la sortie du filtre S_n pour C_n en fonction de s_k.

Il suffit de sa baser sur la phrase de l'énoncé, « la sortie de la somme est la somme des sorties ». Ainsi,

$$S_n(t) = \text{sortie}\left(C_n(t)\right) = \text{sortie}\left(\sum_{k=0}^{n} c_k(t)\right) = \sum_{k=0}^{n} \underbrace{\text{sortie}\left(c_k(t)\right)}_{s_k(t)} \Rightarrow \boxed{S_n(t) = \sum_{k=0}^{n} s_k(t)}.$$

16. En déduire le code de la fonction `S_n`, prenant comme paramètres `n` un entier, `E` l'amplitude du créneau $C(t)$, `T` la période du créneau $C(t)$, `t` un `np.array` correspondant au temps en seconde et `H` une fonction de transfert *elle-même fonction de* ω et qui rend un `np.array` contenant les valeurs prises par $S_n(t)$, la sortie du filtre de fonction de transfert $\underline{H}$ lorsque son entrée est $C_n(t)$. en utilisant la fonction `sinusoide_filtree`.

Le code est le suivant. Il est possible de se baser sur la définition de `C_n` pour y parvenir.

```
94 def S_n(n : int, E : float, T : float, t : np.ndarray, H : callable) -> np.ndarray:
95     Sn = np.zeros(len(t)) # initialiser le résultat
96     # attention à ne pas le nommer comme la fonction elle-même !
97
98     # boucle for pour faire la somme
99     for k in range(0, n+1):
100        A = 4*E / ( (2*k+1) * np.pi ) # amplitude de c_k
101        omega = 2*np.pi/T * (2*k+1) # déterminer omega pour utiliser sinusoide
102        phi = - np.pi/2 # phi pour c_k (pour le passage de sin à cos)
103
104        Sn += sinusoide_filtree(A, omega, phi, t, H)
105
106    return Sn
```

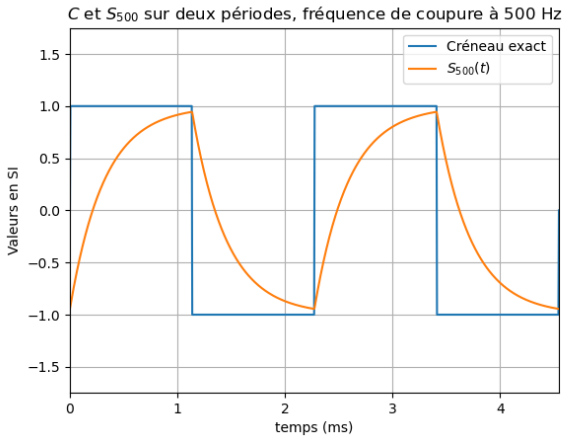
17.a. Écrire un code permettant de tracer sur une même figure la réponse temporelle, sur 2 périodes, $C(t)$ le créneau exact et la réponse du filtre `Filtre_1` à $C_{500}(t)$ et de sauvegarder ce tracé dans une image « `fig5-rep_temporelle.png` ».

Le code est le suivant.

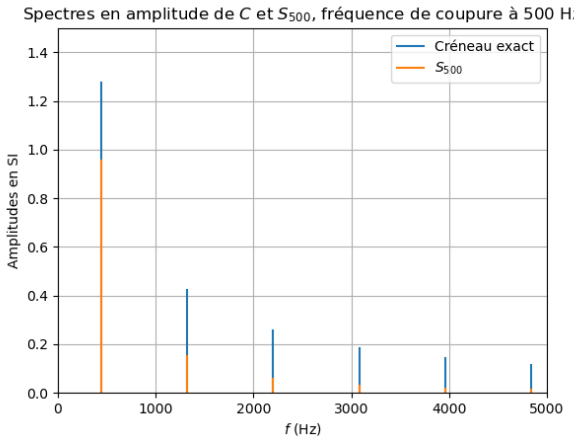
```
108 S_500_fct_t = S_n(500, E, T, time_s, Filtre_1)
109
110 plt.figure()
111 plt.title("$C$ et $S_{500}$ sur deux périodes, fréquence de coupure à 500 Hz")
112 plt.plot(time_s*1e3, C_fct_t, label = "Créneau exact")
113 plt.plot(time_s*1e3, S_500_fct_t, label = "$S_{500}(t)$")
114 plt.xlim([time_s.min()*1e3, time_s.max()*1e3]) # ajustement des bornes de l'axe des
    ↪ abscisses
115 plt.xlabel("temps (ms)")
116 plt.ylim([-1.75, 1.75]) # ajustement des bornes de l'axe des ordonnées
117 plt.ylabel("Valeurs en SI")
118 plt.grid() # pour afficher le quadrillage
119 plt.legend() # afficher les labels des courbes
120 plt.savefig("fig5-rep_temporelle.png")
```

Le tracé obtenu est donné en figure 5a.

b. Écrire un code permettant de tracer le spectre en amplitude de $C_{500}(t)$ et de la réponse du filtre à $C_{500}(t)$ entre 0 et 5 kHz et de sauvegarder ce tracé dans une image « `fig5-spectre.png` ». Pour cela, tracer pour chaque fréquence un trait allant de A à B avec $x_A = x_B = f$ la fréquence en Hz, $y_A = 0$ et



(a) *Le tracé « `fig5-rep_temporelle.png` ».*



(b) *Le tracé « `fig5-spectre.png` ».*

Figure 5

y_B ayant la valeur de l'amplitude de la sinusoïde de fréquence f, pour chaque fréquence contenue dans la décomposition en série de Fourier du signal.

Le code est le suivant.

```
# initialiser les listes pour tracer les spectres
Spectre_C_500 = []
Spectre_S_500 = []
Frequences = []

for k in range(500+1):
    frequence = (2*k+1)/T
    c_k = 4*E / ( (2*k+1) * np.pi ) # amplitude de c_k

    omega = 2*np.pi*frequence
    s_k = c_k * np.abs( Filtre_1(omega) )

    Spectre_C_500.append(c_k)
    Spectre_S_500.append(s_k)
    Frequences.append(frequence)

plt.figure()
plt.title("Spectres en amplitude de $C$ et $S_{500}$, fréquence de coupure à 500 Hz")
for k in range(len(Frequences)):
    frequence = Frequences[k]
    c_k = Spectre_C_500[k]
    s_k = Spectre_S_500[k]

    plt.plot(
        [frequence, frequence],
        [0, c_k],
        color = "C0", # pour avoir tout le spectre de C500 en bleu
        label = "Créneau exact" if k == 0 else None, # pour n'avoir qu'une seule fois
        ↪  le label
    )

    plt.plot(
        [frequence, frequence],
        [0, s_k],
        color = "C1", # pour avoir tout le spectre de S500 en orange
```

```
        label = "$S_{500}$" if k == 0 else None, # pour n'avoir qu'une seule fois le label
    )

plt.xlim([0, 5e3]) # ajustement des bornes de l'axe des abscisses
plt.xlabel("$f$ (Hz)")
plt.ylim([0, 1.5]) # ajustement des bornes de l'axe des ordonnées
plt.ylabel("Amplitudes en SI")
plt.grid() # pour afficher le quadrillage
plt.legend() # afficher les labels des courbes
plt.savefig("fig5-spectre.png")
```

Le tracé obtenu est donné en figure 5b.

18. Faire de même avec une fréquence de coupure $f_{c2} = 5\,\text{kHz}$ dans les images « `fig6-rep_temporelle.png` » et « `fig6-spectre.png` ».

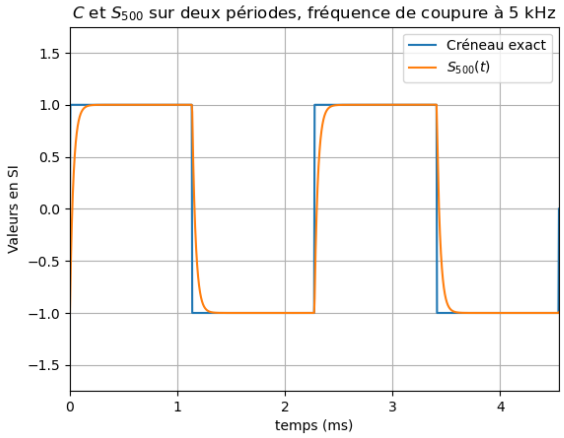
Le code est le suivant.

✎ *Il s'agit essentiellement du même qu'avec la fréquence de coupure à 500 Hz. Pour répondre « d'un coup » à tout le sujet, il aurait été possible de faire une boucle* `for` *sur les différentes valeurs de fréquence de coupure.*

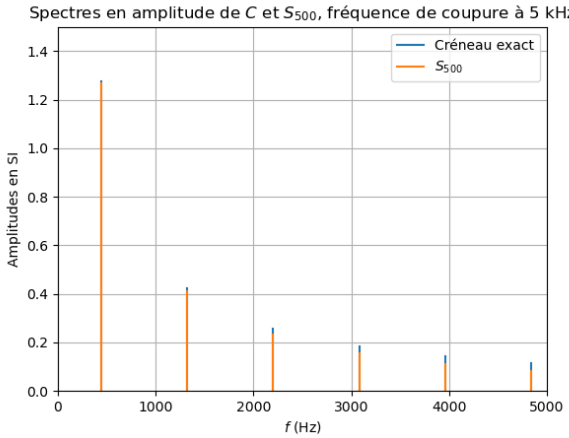
```
def Filtre_2(omega : float) -> complex:
    fc = 5e3 # fréquence de coupure
    omega0 = 2*np.pi * fc # pulsation de coupure
    return H_PasseBas1(omega, omega0) # valeur pour un passe-bas d'ordre 1 à omega pour cette valeur de omega0

S_500_fct_t = S_n(500, E, T, time_s, Filtre_2) # attention à bien changer de filtre !

plt.figure()
plt.title("$C$ et $S_{500}$ sur deux périodes, fréquence de coupure à 5 kHz")
plt.plot(time_s*1e3, C_fct_t, label = "Créneau exact")
plt.plot(time_s*1e3, S_500_fct_t, label = "$S_{500}(t)$")
plt.xlim([time_s.min()*1e3, time_s.max()*1e3]) # ajustement des bornes de l'axe des abscisses
plt.xlabel("temps (ms)")
plt.ylim([-1.75, 1.75]) # ajustement des bornes de l'axe des ordonnées
plt.ylabel("Valeurs en SI")
plt.grid() # pour afficher le quadrillage
plt.legend() # afficher les labels des courbes
plt.savefig("fig6-rep_temporelle.png")

# initialiser les listes pour tracer les spectres
Spectre_C_500 = []
Spectre_S_500 = []
Frequences = []

for k in range(500+1):
    frequence = (2*k+1)/T
    c_k = 4*E / ( (2*k+1) * np.pi ) # amplitude de c_k

    omega = 2*np.pi*frequence
    s_k = c_k * np.abs( Filtre_2(omega) ) # attention à bien changer de filtre !

    Spectre_C_500.append(c_k)
    Spectre_S_500.append(s_k)
    Frequences.append(frequence)

plt.figure()
plt.title("Spectres en amplitude de $C$ et $S_{500}$, fréquence de coupure à 5 kHz")
```

```python
for k in range(len(Frequences)):
    frequence = Frequences[k]
    c_k = Spectre_C_500[k]
    s_k = Spectre_S_500[k]

    plt.plot(
        [frequence, frequence],
        [0, c_k],
        color = "C0", # pour avoir tout le spectre de C500 en bleu
        label = "Créneau exact" if k == 0 else None, # pour n'avoir qu'une seule fois le label
    )

    plt.plot(
        [frequence, frequence],
        [0, s_k],
        color = "C1", # pour avoir tout le spectre de S500 en orange
        label = "$S_{500}$" if k == 0 else None, # pour n'avoir qu'une seule fois le label
    )

plt.xlim([0, 5e3]) # ajustement des bornes de l'axe des abscisses
plt.xlabel("$f$ (Hz)")
plt.ylim([0, 1.5]) # ajustement des bornes de l'axe des ordonnées
plt.ylabel("Amplitudes en SI")
plt.grid() # pour afficher le quadrillage
plt.legend() # afficher les labels des courbes
plt.savefig("fig6-spectre.png")
```

Les tracés obtenus sont donnés en figures 6a pour la réponse temporelle et 6b pour le spectre.



(a) *Le tracé « `fig6-rep_temporelle.png` ».*



(b) *Le tracé « `fig6-spectre.png` ».*

Figure 6 – *Le spectre est effectivement beaucoup moins modifié que dans le cas d'une fréquence de coupure à* 500 Hz, *figure 5b, mais l'action du filtre n'est pas nulle pour autant ! Cela se voit bien sur la réponse temporelle.*

19. Faire de même avec une fréquence de coupure $f_{c3} = 50\,\text{Hz}$ dans les images « `fig7-rep_temporelle.png` » et « `fig7-spectre.png` ».

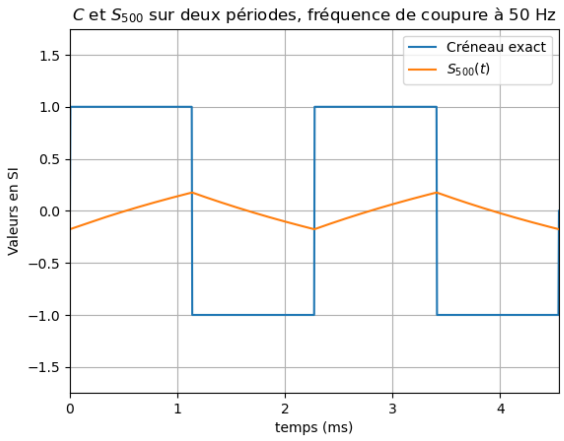
Le code est le suivant. Même démarche qu'à la question précédente !

```python
def Filtre_2(omega : float) -> complex:
    fc = 5e3 # fréquence de coupure
    omega0 = 2*np.pi * fc # pulsation de coupure
```

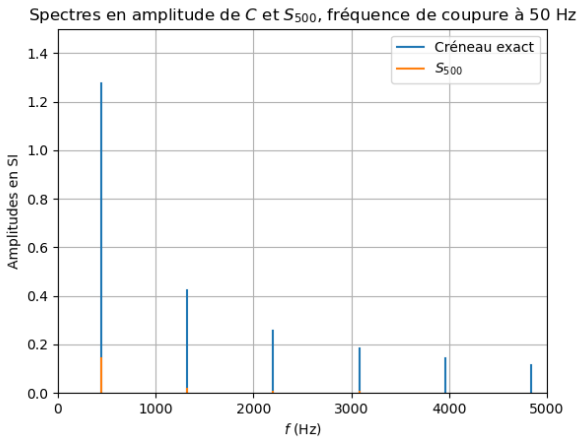
```
    return H_PasseBas1(omega, omega0) # valeur pour un passe-bas d'ordre 1 à omega pour cette valeur de omega0

S_500_fct_t = S_n(500, E, T, time_s, Filtre_2) # attention à bien changer de filtre !

plt.figure()
plt.title("$C$ et $S_{500}$ sur deux périodes, fréquence de coupure à 5 kHz")
plt.plot(time_s*1e3, C_fct_t, label = "Créneau exact")
plt.plot(time_s*1e3, S_500_fct_t, label = "$S_{500}(t)$")
plt.xlim([time_s.min()*1e3, time_s.max()*1e3]) # ajustement des bornes de l'axe des abscisses
plt.xlabel("temps (ms)")
plt.ylim([-1.75, 1.75]) # ajustement des bornes de l'axe des ordonnées
plt.ylabel("Valeurs en SI")
plt.grid() # pour afficher le quadrillage
plt.legend() # afficher les labels des courbes
plt.savefig("fig6-rep_temporelle.png")

# initialiser les listes pour tracer les spectres
Spectre_C_500 = []
Spectre_S_500 = []
Frequences = []

for k in range(500+1):
    frequence = (2*k+1)/T
    c_k = 4*E / ( (2*k+1) * np.pi ) # amplitude de c_k

    omega = 2*np.pi*frequence
    s_k = c_k * np.abs( Filtre_2(omega) ) # attention à bien changer de filtre !

    Spectre_C_500.append(c_k)
    Spectre_S_500.append(s_k)
    Frequences.append(frequence)

plt.figure()
plt.title("Spectres en amplitude de $C$ et $S_{500}$, fréquence de coupure à 5 kHz")
for k in range(len(Frequences)):
    frequence = Frequences[k]
    c_k = Spectre_C_500[k]
    s_k = Spectre_S_500[k]

    plt.plot(
        [frequence, frequence],
        [0, c_k],
        color = "C0", # pour avoir tout le spectre de C500 en bleu
        label = "Créneau exact" if k == 0 else None, # pour n'avoir qu'une seule fois le label
    )

    plt.plot(
        [frequence, frequence],
        [0, s_k],
        color = "C1", # pour avoir tout le spectre de S500 en orange
        label = "$S_{500}$" if k == 0 else None, # pour n'avoir qu'une seule fois le label
    )

plt.xlim([0, 5e3]) # ajustement des bornes de l'axe des abscisses
plt.xlabel("$f$ (Hz)")
```

```
225 plt.ylim([0, 1.5]) # ajustement des bornes de l'axe des ordonnées
226 plt.ylabel("Amplitudes en SI")
227 plt.grid() # pour afficher le quadrillage
228 plt.legend() # afficher les labels des courbes
229 plt.savefig("fig6-spectre.png")
```

Les tracés obtenus sont donnés en figures 7a pour la réponse temporelle et 7b pour le spectre.



(a) *Le tracé « `fig7-rep_temporelle.png` ».*



(b) *Le tracé « `fig7-spectre.png` ».*

Figure 7

20. Commentez vos résultats, comparez les effets des différents filtres sur les réponses temporelles et fréquentielles.

Il y a beaucoup de choses à dire ici :

Réponses temporelles : allure générale Les allures caractéristiques de charges (et éventuellement décharges ou charges « opposées » selon les valeurs prises par le créneau) d'un condensateur sont obtenues. Ce n'est pas anodin : un filtre passe-bas d'ordre 1 peut être obtenu en prenant en sortie la tension aux bornes du condensateur d'un circuit RC série avec en entrée la tension aux bornes de cet ensemble. Une autre possibilité est de prendre la tension aux bornes d'une résistance d'un circuit RL série.

Réponses temporelles : comparaison La fréquence de coupure vérifie

$$f_c = \frac{1}{2\pi}\omega_0 = \frac{1}{2\pi}\frac{1}{\tau}$$

avec τ le temps caractéristique du système du premier ordre servant de filtre. Ainsi, en augmentant f_c, τ diminue. Il n'est donc pas étonnant, **pour le premier filtre**, d'observer *presque* tout le régime transitoire en figure 5a, car dans ce cas

$$T = 2{,}3\,\text{ms}\,, \quad \tau_1 = \frac{1}{2\pi f_{c1}} = 0{,}32\,\text{ms} \sim 0{,}28\,\frac{T}{2} \Leftrightarrow \frac{T}{2} \sim 3{,}6\,\tau_1\,.$$

Il faut bien comparer τ à $T/2$, car le créneau n'est de valeur constante que sur $T/2$!

Dans le cas du **deuxième filtre**, figure 6a, $T/2 \sim 36\,\tau_2$. Ainsi, $T/2 \gg \tau_2$, le système servant de filtre *a le temps* de répondre, le régime transitoire est rapidement terminé face à l'échelle de temps de l'entrée (le créneau). Un régime transitoire est toutefois encore présent à la bascule du créneau, et en comparant la figure 6a à la figure 4, l'effet de *lissage* de la tension par un condensateur (ou du courant par une bobine) est observable.

Dans le cas du **troisième filtre**, figure 7a, $T/2 \sim 0{,}36\,\tau_3$. Ainsi, $T/2 \ll \tau_2$, le système servant de filtre *n'a pas le temps* de répondre, le régime transitoire est à peine débuté que le créneau rebascule. Alors, la réponse au créneau prend l'allure d'un signal triangulaire. Ici, c'est l'aspect *intégrateur* du passe-bas d'ordre 1 qui est observé : pente montante quasi constante lorsque le créneau est positif, descendante lorsque le créneau est négatif. Notez bien que la pente est *quasi* constante, il s'agit en fait de l'évolution en « $\exp(-t/\tau)$ » du système d'ordre 1 pour $t \ll \tau$, ce qui est en bonne approximation la droite tangente à l'origine (par rapport à l'instant de bascule du créneau).

Réponse fréquentielle - spectres Sans surprise, plus la réponse est proche de l'entrée dans leurs aspects *temporels* (figures 5a, 6a et 7a), plus la réponse est proche de l'entrée dans leurs aspects *fréquentiels* (figures 5b, 6b et 7b).
Si la bande passante correspond, pour un passe-bas d'ordre 1, à la zone $f < f_c$, l'action d'un tel filtre n'est pas nulle pour autant dans la bande passante, ce qui est visible en figure 5b sur le fondamental, seule fréquence inférieure à celle de coupure (500 Hz), et aussi, particulièrement, en figure 6b où toutes les fréquences tracées sont inférieures à celle de coupure (5 kHz). L'effet est de plus en plus marqué lorsque la fréquence augmente, ce qui est effectivement prévisible vu la fonction de transfert.
Ces spectres permettent également de justifier *a posteriori* l'approche de cet exercice, où le créneau filtré est étudié en se basant uniquement sur les $2 \times 500 + 1 = 1001$ premiers harmoniques (dont seuls 500 sont non nuls, l'harmonique 1 étant le fondamental lui-même). En effet, même avec $f_c = f_{c2} = 5\,\text{kHz}$, en termes d'amplitudes,

$$s_{500} = |\underline{H}| c_{500} = \frac{1}{\sqrt{1 + \left(\frac{f_{500}}{f_{c2}}\right)^2}} \frac{4E}{(2 \times 500 + 1)\pi} \sim 10^{-5}\, E \ll E\,,$$

les termes (harmoniques) suivants, au-delà de $k = 500$ dans S_n, sont donc effectivement négligeables.

> ✎ *Il aurait même été possible, éventuellement, de se limiter à moins d'harmoniques mais à partir du moment où la boucle* `for` *est mise en place, l'effort supplémentaire est uniquement à la charge de l'ordinateur, ce qui n'est pas gênant tant que l'exécution du code reste raisonnable en termes de ressources (temps utilisateur, temps CPU, énergie électrique consommée).*

4 Action d'un passe-bande d'ordre 2

21. Rappeler la forme canonique de la fonction de transfert $\underline{H}(\mathrm{j}\omega)$ d'un passe-bande d'ordre 2. Faire apparaître, en particulier, le facteur de qualité Q et la pulsation propre ω_0.

Voir le cours si vous ne le savez plus (pas pour l'apprendre, mais pour savoir la retrouver) ! La forme canonique de la fonction de transfert $\underline{H}(\mathrm{j}\omega)$ d'un passe-bande d'ordre 2 est

$$\boxed{\underline{H}(\mathrm{j}\omega) = \frac{1}{1 + \mathrm{j}Q\left(\frac{\omega}{\omega_0} - \frac{\omega_0}{\omega}\right)}}\,.$$

22. Écrire le code définissant la fonction de transfert du passe-bande d'ordre 2 `H_PasseBande2`, prenant comme paramètres `omega` la pulsation à laquelle évaluer $\underline{H}(\mathrm{j}\omega)$, `Q` le facteur de qualité et `omega0` la pulsation propre et qui rend un nombre complexe étant la valeur de $\underline{H}(\mathrm{j}\omega)$.

Le code est le suivant.

```
295 def H_PasseBande2(omega : float, Q : float, omega0 : float) -> complex:
296     return 1 / (1 + 1j * Q * (omega/omega0 - omega0/omega))
```

Les paramètres du filtre passe-bande seront, dans la suite, $Q = 1$ et $\omega_0 = 5 \times 2\pi/T$ avec T la période du créneau précédemment définie.

23. Déduire du travail effectué le code de la fonction `S_passe_bande_n`, prenant comme paramètres `n` un entier, `E` l'amplitude du créneau $C(t)$, `T` la période du créneau $C(t)$, `t` un `np.array` correspondant au temps en seconde et qui rend un `np.array` contenant les valeurs prises par la sortie du filtre passe-bande lorsqu'il est soumis à $C_n(t)$ en entrée.

Le code est le suivant. Voir les questions précédentes pour les détails.

```
298 def Filtre_4(omega : float) -> complex:
299     omega0 = 5 * 2*np.pi / T
300     Q = 1
301     return H_PasseBande2(omega, Q, omega0)
302
```

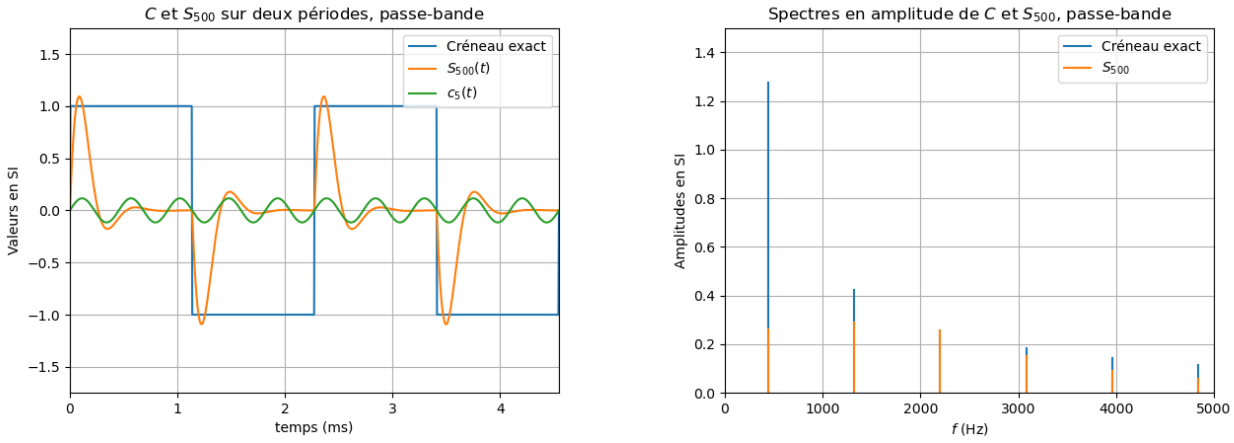
```python
def S_passe_bande_n(n : int, E : float, T : float, t : np.ndarray) -> np.ndarray:
    return S_n(n, E, T, t, Filtre_4)
```

24.a. Écrire un code permettant de tracer sur une même figure la réponse temporelle, sur 2 périodes, $C(t)$ le créneau exact et la réponse du filtre à $C_{500}(t)$ et de sauvegarder ce tracé dans une image « `fig8-rep_temporelle.png` ».

Le code est le suivant. Encore une fois, même démarche !

```python
S_500_fct_t = S_passe_bande_n(500, E, T, time_s)

# Bonus pour comparer, l'harmonique 5 qui est pile sur le centre de la bande passante
f5 = 5/T
A = 4*E/((2*5+1)*np.pi)
Harmo_rang_5 = sinusoide(A, 2*np.pi*f5, -np.pi/2, time_s)

plt.figure()
plt.title("$C$ et $S_{500}$ sur deux périodes, passe-bande")
plt.plot(time_s*1e3, C_fct_t, label = "Créneau exact")
plt.plot(time_s*1e3, S_500_fct_t, label = "$S_{500}(t)$")
plt.plot(time_s*1e3, Harmo_rang_5, label = "$c_5(t)$")
plt.xlim([time_s.min()*1e3, time_s.max()*1e3]) # ajustement des bornes de l'axe des
↪ abscisses
plt.xlabel("temps (ms)")
plt.ylim([-1.75, 1.75]) # ajustement des bornes de l'axe des ordonnées
plt.ylabel("Valeurs en SI")
plt.grid() # pour afficher le quadrillage
plt.legend() # afficher les labels des courbes
plt.savefig("fig8-rep_temporelle.png")
```

Le tracé obtenu est donné en figure 8a.



(a) *Le tracé « `fig8-rep_temporelle.png` ».*

(b) *Le tracé « `fig8-spectre.png` ».*

Figure 8

b. Écrire un code permettant de tracer le spectre en amplitude de $C_{500}(t)$ et de la réponse du filtre à $C_{500}(t)$ et de sauvegarder ce tracé dans une image « `fig8-spectre.png` ».

Le code est le suivant. Encore une fois, même démarche !

```python
# initialiser les listes pour tracer les spectres
Spectre_C_500 = []
Spectre_S_500 = []
```

```
Frequences = []

for k in range(500+1):
    frequence = (2*k+1)/T
    c_k = 4*E / ( (2*k+1) * np.pi ) # amplitude de c_k

    omega = 2*np.pi*frequence
    s_k = c_k * np.abs( Filtre_4(omega) ) # attention à bien changer de filtre !

    Spectre_C_500.append(c_k)
    Spectre_S_500.append(s_k)
    Frequences.append(frequence)

plt.figure()
plt.title("Spectres en amplitude de $C$ et $S_{500}$, passe-bande")
for k in range(len(Frequences)):
    frequence = Frequences[k]
    c_k = Spectre_C_500[k]
    s_k = Spectre_S_500[k]

    plt.plot(
        [frequence, frequence],
        [0, c_k],
        color = "C0", # pour avoir tout le spectre de C500 en bleu
        label = "Créneau exact" if k == 0 else None, # pour n'avoir qu'une seule fois
        ↪ le label
    )

    plt.plot(
        [frequence, frequence],
        [0, s_k],
        color = "C1", # pour avoir tout le spectre de S500 en orange
        label = "$S_{500}$" if k == 0 else None, # pour n'avoir qu'une seule fois le
        ↪ label
    )

plt.xlim([0, 5e3]) # ajustement des bornes de l'axe des abscisses
plt.xlabel("$f$ (Hz)")
plt.ylim([0, 1.5]) # ajustement des bornes de l'axe des ordonnées
plt.ylabel("Amplitudes en SI")
plt.grid() # pour afficher le quadrillage
plt.legend() # afficher les labels des courbes
plt.savefig("fig8-spectre.png")
```

Le tracé obtenu est donné en figure 8b.

25. Commentez vos résultats.

Cette fois, l'allure de la réponse du filtre au créneau ressemble beaucoup à celle de la tension aux bornes de la résistance d'un circuit RLC série qui serait alimenté par l'entrée. Ce n'est pas anodin : un filtre passe-bande d'ordre 2 peut justement être obtenu de cette manière !

Il est de plus intéressant de remarquer que, la bande passante du filtre étant centrée sur la fréquence de l'harmonique de rang 5 (à $f_5 = 2{,}20\,\text{kHz}$), et Q étant égal à 1, cette composante de la série de Fourier *n'est pas modifiée*, ce qui est particulièrement visible sur le spectre en figure 5b. En effet, $\underline{H}(\text{j}\omega_5) = 1$ ici.
En revanche, les amplitudes des autres fréquences sont réduites, et ce d'autant plus qu'elles sont éloignées de f_5 qui correspond à la fréquence propre de ce filtre, c'est bien ce qui est attendu avec un passe-bande.

> ✎ *Attention, il s'agit d'une diminution* relative *: comme $c_3 > c_5$, même si l'harmonique 3 est plus diminuée que le 5, il y a tout de même $s_3 > s_5$.*

Autre effet intéressant et notable, les oscillations amorties de la sortie ont une fréquence proche de celle de l'harmonique 5, ce qui apparaît en traçant $c_5(t) = s_5(t)$.