Introduction au signal et bruit
Travaux pratiques

Gabriel Dauphin

November 12, 2025

Contents

1 Séance 1 de travaux pratiques **5**

1.1 Préparation à faire avant la séance 5

1.1.1 Montage étudié 7

1.2 Travail à effectuer pendant la séance 13

1.2.1 Préparation à l'utilisation de Python 13

1.2.2 Visualisation de la réponse fréquentielle 15

1.2.3 Détermination numérique de la réponse impulsionnelle 19

1.2.4 Détermination de la réponse du système à $x_a(t) = \mathbb{T}(t)$ 24

1.3 Travail à rendre une semaine après la séance 28

1.4 Codes permettant de réaliser le TP1 29

1.4.1 Figure 1.2 29

1.4.2 Figure 1.3 30

1.4.3 Simulation de la figure 1.4 33

2 Séance 2 de travaux pratiques **36**
2.1 Préparation à faire avant la séance 36
2.2 Travail à effectuer pendant la séance 37
2.2.1 Signal étudié 37
2.2.2 Calculs théoriques sur le signal $s_1(t)$ 38
2.2.3 Échantillonnage du signal 38
2.2.4 Simulation de la transformée de Fourier 40
2.2.5 Simulation de la périodisation du spectre 41
2.2.6 Périodisation de $s_1(t)$ 42
2.2.7 Simulation de la transformée de Fourier 43
2.2.8 Simulation de la périodisation de la transformée de Fourier 44
2.3 Travail à rendre une semaine après la séance 45
2.4 Codes Python permettant de réaliser le TP2 45
2.4.1 Code pour générer la gauche de la figure 2.1 45
2.4.2 Droite de la figure 2.1. 46
2.4.3 Figure 2.2 47
2.4.4 figure 2.3 49
2.4.5 Figure 2.4 51
2.4.6 Figure 2.6 52
2.4.7 Figure 2.5 54

3 Séance 3 de travaux pratiques **55**

3.1 Préparation à faire avant la séance 55
3.2 Travail à effectuer pendant la séance 56
3.2.1 Estimation de la réponse impulsionnelle 62
3.2.2 Estimation de l'autocorrélation 63
3.2.3 Tirage aléatoire d'une valeur particulière de $\overset{r}{Y}(t_0)$ 65
3.2.4 Représentation de la densité de probabilité du signal en sortie en $t = t_0$ pour une fréquence d'échantillonnage f_e 66
3.2.5 Détermination théorique de la moyenne statistique et de la variance de de $\overset{r}{Y}(t)$ à partir de celles de $\overset{r}{B}(t)$ 69
3.2.6 Différentes façons de simuler l'autocorrélation 70
3.2.7 Différentes façons de simuler la densité spectrale 74
3.2.8 Simulation dans le cas où le bruit placé en entrée du filtre est obtenu avec un motif 77
3.3 Codes permettant de réaliser le TP3 78
3.3.1 Figure 3.1 78
3.3.2 Figure 3.3 78
3.3.3 Figure 3.2 80
3.3.4 Figure 3.4 81
3.3.5 Figure 3.5 82
3.3.6 Simulation de la section 3.2.6 85

A Supplément **87**

A.1 Outils 87
A.1.1 Commandes générales 88

B Codes pour simuler les différentes figures **97**
B.1 Installation de Python sous Windows 97
B.2 Codes pour simulation une réponse fréquentielle 102
B.3 Simulation d'une sortie d'un filtre 102
B.4 Simulation d'un signal défini par des fonctions de base 102
B.5 Simulation d'un signal échantillonné 102
B.6 Simulation d'un calcul d'erreur quadratique 102
B.7 Simulation d'une reconstruction par interpolation 102
B.8 Code pour simuler une réponse fréquentielle 102

Chapter 1

Séance 1 de travaux pratiques

1.1 Préparation à faire avant la séance

Ceci constitue une partie théorique à faire avant la séance et à inclure dans le document `pdf` à rendre après la séance.

1. Choisissez un montage électronique simple pour lequel une source de tension ou d'intensité est considérée comme l'entrée d'un filtre et la tension aux bornes d'un composant ou l'intensité traversant un composant est considéré comme la sortie. Ce montage devra fonctionner ici en régime linéaire (par exemple l'amplificateur opérationnel devra être utilisé en rétro-action stable et non comme un comparateur). Vous

préciserez les valeurs numériques de chaque composant. Vous pouvez vous inspirer du montage présenté en section 1.1.1. Il est souhaitable que les caractéristiques du filtre soient numériquement simples à simuler, vous pouvez modifier les valeurs des composants pour que ce soit le cas.

2. En vous inspirant dé la section 1.1.1 et à partir d'une analyse physique du montage proposé, justifiez que le filtre proposé peut être modélisé avec une relation linéaire et temps invariante.

3. En vous inspirant dé la section 1.1.1, déterminez la réponse statique du filtre et le comportement limite à haute fréquence.

4. Calculez la réponse fréquentielle du filtre en utilisant vos connaissances d'électronique.

5. Déduisez une relation entrée-sortie décrite avec une équation différentielle.

6. Déduisez la réponse impulsionnelle théorique (il n'est pas nécessaire de conduire les calculs jusqu'au bout, il suffit qu'on puisse programmer le calcul de la réponse impulsionnelle).

Dans un premier temps, on considère dans ce TP ce montage et cette étude théorique.

1.1.1 Montage étudié

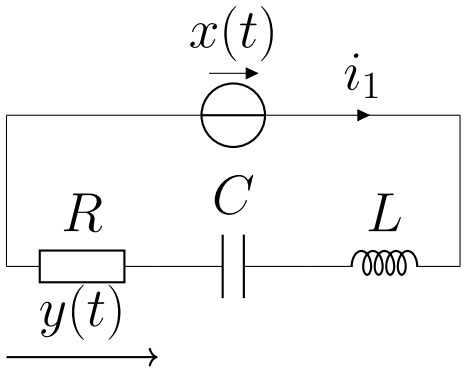

Figure 1.1: Montage correspondant au filtre étudié d'entrée la tension $x(t)$ et de sortie la tension $y(t)$. $R = 3\Omega$, $C = 0.5\text{F}$, $L = 1\text{H}$.

Analyse physique rapide du montage

Tous les composants sont linéaires entre l'intensité et la tension aussi la relation entre $x(t)$ et $y(t)$ est linéaire. On suppose ici que les composants ont des caractéristiques fixes, donc le système est temps invariant. À très basse fréquence, le condensateur se comporte comme un circuit ouvert et donc $\widehat{Y}(f)$ est nul à très basse fréquence. À très haute fréquence, la bobine se comporte comme un circuit ouvert et donc $\widehat{Y}(f)$ est nul à très haute fréquence.

Calcul théorique de la réponse fréquentielle

En utilisant les impédances de chaque composant et en observant qu'il y a un diviseur de tension, on a

$$\widehat{H}(f) = \frac{\widehat{Y}(f)}{\widehat{X}(f)} = \frac{R}{R + j2\pi fL + \frac{1}{j2\pi fC}} = \frac{j2\pi fRC}{1 + j2\pi fRC - 4\pi^2 f^2 LC} \tag{1.1}$$

On peut remarquer que les affirmation de la section 1.1.1 sont confirmées par l'équation (1.1)

$$\begin{cases} \lim\limits_{f\to 0} \widehat{H}(f) = \lim\limits_{f\to 0} \frac{j2\pi fRC}{1+j2\pi fRC-4\pi^2 f^2 LC} = 0 \\ \text{et } \lim\limits_{f\to +\infty} \widehat{H}(f) = \lim\limits_{f\to +\infty} \frac{R}{R+j2\pi fL+\frac{1}{j2\pi fC}} = 0 \end{cases} \tag{1.2}$$

Écriture de la relation entrée-sortie avec une équation différentielle

On remplace chaque terme $j2\pi f$ par $\frac{d}{dt}$

$$LC\frac{d^2}{dt^2}y(t) + RC\frac{d}{dt}y(t) + y(t) = RC\frac{d}{dt}x(t) \tag{1.3}$$

Calcul théorique de la réponse impulsionnelle : solution 1

Les calculs qui suivent peuvent dans une certaine mesure être obtenus en utilisant la toolbox [1] `sympy` à installer sur `Python` https://https://docs.sympy.org et en util-

[1] Remarquez que certaines commandes de `sympy` commencent par une majuscule.

isant Wolfram https://www.wolframalpha.com/
Wolfram `Fourier transform of exp(-p*t)*heaviside(t)` *Il reste à multiplier par* $\sqrt{2\pi}$ *et à remplacer* ω *par* $2\pi f$.

En posant $x(t) = \delta(t)$, on sait que $y(t)$ est la réponse impulsionnelle $h(t)$ qui vérifie donc

$$LC\frac{d^2}{dt^2}h(t) + RC\frac{d}{dt}h(t) + h(t) = RC\frac{d}{dt}\delta(t) \tag{1.4}$$

Cette équation peut se mettre sous une forme plus classique

$$LC\frac{d^2}{dt^2}\tilde{h}(t) + RC\frac{d}{dt}\tilde{h}(t) + \tilde{h}(t) = \delta(t) \text{ et } h(t) = RC\frac{d}{dt}\tilde{h}(t) \tag{1.5}$$

Le polynôme $Q(p) = LCp^2 + RCp + 1$ a deux racines réelles ($R^2C^2 - 4LC > 0$) : $p_1 = \frac{-RC-\sqrt{R^2C^2-4LC}}{2LC}$ et $p_2 = \frac{-RC+\sqrt{R^2C^2-4LC}}{2LC}$.

Python :

Le calcul symbolique permet de faire faire la résolution.

```
import sympy
R=sympy.Symbol('R')
L=sympy.Symbol('L')
```

```
C=sympy.Symbol('C')
p=sympy.Symbol('p')
p1,p2=sympy.solve('L*C*p**2+R*C*p+1','p'); print(f"p1={p1}, p2={p2}")
sympy.simplify(p1+p2)
sympy.simplify(p1*p2)
```

La lettre `f` précédent l'expression entre guillement contenu dans `print` permet un affichage particuler au sens où les expressions dans les accolades sont évaluées et leurs valeurs sont affichées. Les valeurs de $p1$ et 2 sont négatives et distinctes. Aussi $\tilde{h}(t)$ se met sous la forme

$$\tilde{h}(t) = (c_1 e^{p_1 t} + c_2 e^{p_2 t}) [\![t \geq 0]\!] \tag{1.6}$$

Pour trouver c_1 et c_2, on pourrait calculer sa transformée de Fourier et identifier avec $\widehat{H}(f)$. On peut aussi calculer les dérivées successives pour vérifier avec l'équation (1.5).

$$\begin{aligned} RC\tfrac{d}{dt}\tilde{h}(t) &= RC(c_1 p_1 e^{p_1 t} + c_2 p_2 e^{p_2 t}) [\![t \geq 0]\!] + RC(c_1 + c_2)\delta(t) \\ LC\tfrac{d^2}{dt^2}\tilde{h}(t) &= LC(c_1 p_1^2 e^{p_1 t} + c_2 p_2^2 e^{p_2 t}) [\![t \geq 0]\!] + (c_1 + c_2)\delta'(t) + LC(c_1 p_1 + c_2 p_2)\delta(t) \end{aligned} \tag{1.7}$$

Ces dérivées successives peuvent ensuite être réintroduites dans l'équation (1.5), on a alors

l'égalité quand ceci est vérifié.

$$\begin{cases} c_1(1+RCp_1+LCp_1^2)=0 & \text{termes en facteur de } e^{p_1 t}[\![t\geq 0]\!] \\ c_2(1+RCp_2+LCp_2^2)=0 & \text{termes en facteur de } e^{p_2 t}[\![t\geq 0]\!] \\ RC(c_1+c_2)+LC(c_1p_1+c_2p_2)=1 & \text{termes en facteur de } \delta(t) \\ c_1+c_2=0 & \text{termes en facteur de } \delta'(t) \end{cases} \Rightarrow \begin{cases} c_1+c_2=0 \\ LC(c_1p_1+ \end{cases}$$

La résolution de ce système donne

$$c_1=\frac{1}{LC(p_1-p_2)} \text{ et } c_2=-\frac{1}{LC(p_1-p_2)} \tag{1.9}$$

Et en utilisant $c_1+c_2=0$, on a finalement

$$h(t)=RC\frac{d}{dt}\tilde{h}(t)=RC\frac{d}{dt}\tilde{h}(t)=RC(c_1p_1e^{p_1t}+c_2p_2e^{p_2t})=\frac{R}{L}\frac{p_1}{p_1-p_2}e^{p_1t}[\![t\geq 0]\!]-\frac{R}{L}\frac{p_2}{p_1-}$$

Python :

Pour continuer avec le calcul symbolique.

```
from sympy.matrices import *
M=Matrix([[1,1],[p2,p1]])
B=Matrix([[R/L],[0]])
X=M.solve(B); c1=X[0]; c2=X[1]
```

```
print(f"c1={c1} c2={c2}")
#Pour vérifier
sympy.simplify(c1+c2)
sympy.simplify(p2*c1+p1*c2)
```

On peut observer que les matrices ici sont remplies ligne par ligne (i.e. `[1,1]` est la première ligne).

Calcul théorique de la réponse impulsionnelle : solution 2

Une seconde solution consiste à utiliser p_1 et p_2 calculé à partir de l'équation différentielle vérifiée par $\tilde{h}(t)$ et à supposer qu'ils sont valable pour $h(t)$ et que donc $h(t)$ serait de cette forme-là. Et n'étant pas tout à fait sûr, on rajoute un possible Dirac.

$$h(t) = d_1 e^{p_1 t}[\![t \geq 0]\!] + d_2 e^{p_2 t}[\![t \geq 0]\!] + d_3\delta(t) \tag{1.11}$$

La transformée de Fourier de cette réponse impulsionnelle est

$$\widehat{H}(f) = d_1 \frac{1}{j2\pi f - p_1} + d_2 \frac{1}{j2\pi f - p_2} + d_3 \tag{1.12}$$

On identifie avec $\widehat{H}(f)$ pour trois fréquences quelconques, par exemple $f_0 = 0$, $f_1 = 1$ et

$f_2 = 2$.

$$\begin{cases} d_1 \frac{1}{j2\pi f_0 - p_1} + d_2 \frac{1}{j2\pi f_0 - p_2} + d_3 = \widehat{H}(f_0) \\ d_1 \frac{1}{j2\pi f_1 - p_1} + d_2 \frac{1}{j2\pi f_1 - p_2} + d_3 = \widehat{H}(f_1) \\ d_1 \frac{1}{j2\pi f_2 - p_1} + d_2 \frac{1}{j2\pi f_2 - p_2} + d_3 = \widehat{H}(f_2) \end{cases} \tag{1.13}$$

Et on demande à l'ordinateur de résoudre ce système en utilisant des valeurs numériques de R, L, C.

1.2 Travail à effectuer pendant la séance

1.2.1 Préparation à l'utilisation de Python

Python :

L'annexe B.1 montre comment installer Python sous Windows.

Pour démarrer l'utilisation de Python, je propose de créer deux répertoires placés au sein d'un répertoire, le premier que j'appelle `rep_prg` contient les programmes, notamment ceux disponibles sur mon site internet, et un autre répertoire que j'appelle `rep_tra` où vous mettez le travail effectué notamment les données et les figures. Je propose d'écrire un module placé dans `rep_prg` par séances de TP.

Je propose ensuite d'effectuer les lignes suivantes qui utilisent un module `seb` que j'ai écrit pour ce cours et qui est disponible sur https://htmlpreview.github.io/

?https://github.com/Gabriel0010/l2ti/blob/main/Signal_Noise.htm ou sur https://www-l2ti.univ-paris13.fr/~dauphin/Signal_Noise.htm `plt` et `np` sont les modules `matplotlib` et `numpy` qui servent à tracer des graphes et à manipuler des vecteurs.

Pour le nom des répertoires, que ce soit sous Windows ou sous Linux, il convient sous Python d'utiliser des slash `/` et non des anti-slash \. La deuxième ligne avec `sys.path.append` signifie ajouter `rep_prg` parmi les répertoires que Python va scruter pour trouver un module ou un fichier. La quatrième ligne avec `os.chdir` signifie un changement de répertoire, c'est l'équivalent de la commande Windows/Linux `cd`.

```
import sys
sys.path.append('rep_prg')
import os
os.chdir('rep_tra')
import seb
plt,np=seb.debut()
import importlib
```

Sous Windows, `Notepad++` est un éditeur de texte que j'utilise pour Python. Mais certains préfèrent `Thonny` installé dans les salles F100...

1.2.2 Visualisation de la réponse fréquentielle

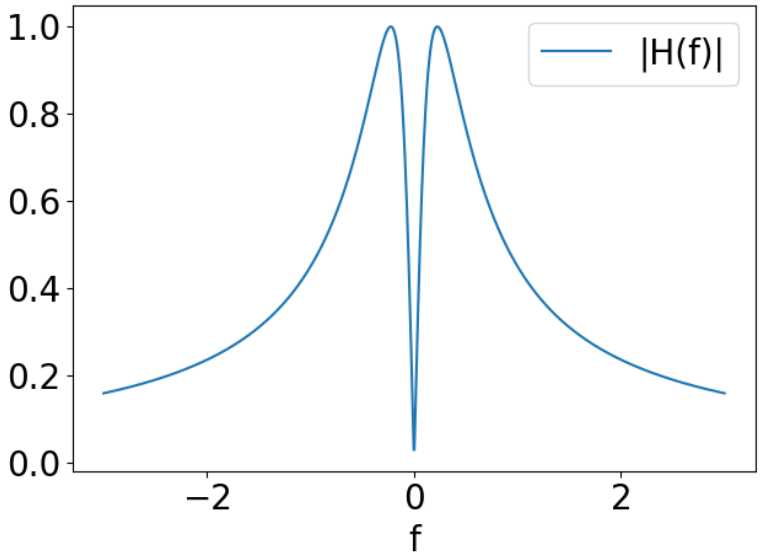

Figure 1.2: Module de la réponse fréquentielle

Ici l'implémentation de la figure 1.2 est indiquée dans la section 1.4.1.

- Créez une échelle de fréquences avec le vecteur `f` par exemple entre -3 et 3 et représentez graphiquement la réponse fréquentielle

On appelle vecteur un tableau de valeurs composé d'une ligne (on parle alors de vecteur ligne) ou d'une seule colonne (on parle de vecteur colonne). La fonction `linspace` de `numpy` permet de générer un ensemble de valeurs en indiquant la première, la dernière et le nombre de ces valeurs. Ici ceci permet de construire l'échelle en fréquence.

```
f=np.linspace(-3,3,10**3)
```

Cette instruction génère 1000 valeurs entre -3 et 3. Cette notion est utile pour générer un graphique, elle permet d'obtenir la figure 1.2 représentant le module de la réponse fréquentielle.

On définit les valeurs des variables. En Python il est possible d'allouer plusieurs variables en même temps.

```
R,C,L = 3,0.5,1
```

On obtient la réponse fréquentielle, (π, j et le carré sont implémentés avec respectivement `np.pi` et `1j` et `**2`).

```
H=1j*2*np.pi*f*R*C/(1+1j*2*np.pi*f*R*C-4*np.pi**2*f**2*L*C)
```

On obtient alors un vecteur ligne de même taille que `f` et contenant successivement toutes les valeurs complexes de H pour chacune des valeurs du vecteur `f`.

Pour faire le graphe, on commence par

```
fig,ax = plt.subplots()
```

Ensuite pour la courbe on rajoute [2]

```
ax.plot(f,np.abs(H),label='|H(f)|')
```

[2]Et si on avait plusieurs courbes, il suffit de rajouter une ligne par courbe.

On implémente le module avec `np.abs`. Le troisième argument permet de rajouter une légende.

Sur le graphe on peut préciser ce que signifie l'axe des abscisses [3]

```
ax.set_xlabel('f')
```

L'affichage de la légende est déclenchée par [4]

```
ax.legend()
```

La gestion de la taille de la figure est faite avec

```
plt.tight_layout()
```

Je propose de sauvegarde la figure dans `rep_tra`.

```
fig.savefig('nom_figure.png')
```

La figure apparaît lorsqu'on exécute cette commande

```
fig.show()
```

[3] Si on voulait préciser l'axe des ordonnées on pourrait rajouter `ax.set_ylabel()`

[4] Cette commande ne fonctionne que si précédemment `label` a été précisé.

Et comme on va utiliser à nouveau ces calculs il est intéressant de construire une fonction qui calcule $\widehat{H}(f)$. Notez qu'il est très important ici de laisser deux espaces au début de chaque ligne après la première ligne pour indiquer que les instructions font parti de la fonction appelée `H1`. Cet ensemble d'instructions sont à mettre dans le module `nom_TP1.py`

```
def H1(R,L,C,f):
  """fonction de la reponse frequentielle obtenue à partir du montage da
  import numpy as np
  H=1j*2*np.pi*f*R*C/(1+1j*2*np.pi*f*R*C-4*(np.pi**2)*(f**2)*L*C)
  return H
```

On peut faire appel à cette fonction avec

```
ax.plot(f,np.abs(H1(f)),label='|H(f)|')
```

Et cette commande peut elle-même être dans une fonction appelée `Q1_2_2`. Pour lancer cette nouvelle fonction

```
import nom_TP1
nom_TP1.Q1_2_2()
```

Lorsque le fichier `nom_TP1.py` est modifié, ces modifications ne sont prises en compte que si on fait la commande

```
importlib.reload(nom_TP1)
```

1.2.3 Détermination numérique de la réponse impulsionnelle

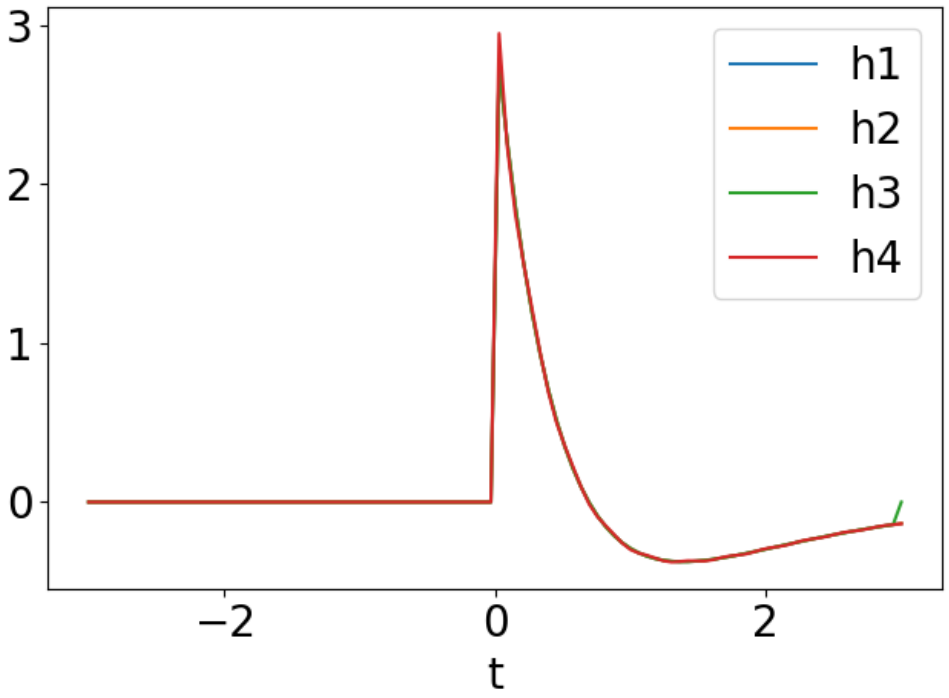

Figure 1.3: Réponse impulsionnelle $h(t)$.

- Créez une échelle de temps avec le vecteur `t` par exemple entre -3 et 3 et représentez graphiquement cette réponse impulsionnelle soit avec `h1` notée $h_1(t)$ soit avec `h2` notée $h_2(t)$.
- Déterminez $h_3(t)$ la réponse impulsionnelle à partir de l'équation différentielle (1.5).
- Déterminez $h_4(t)$ la réponse impulsionnelle cette fois-ci à partir de la réponse fréquentielle (1.1).

La figure 1.3 montre $h_1(t)$, $h_2(t)$, $h_3(t)$, $h_4(t)$. Ici l'implémentation est indiquée dans la section 1.4.2.

Implémentation de la première solution

La partie théorique permet de calculer la réponse impulsionnelle liée à la première solution

Python :

On calcule d'abord p_1 et p_2.

```
p1=(-C*R - np.sqrt(C*(C*R**2 - 4*L)))/(2*C*L)
p2=(-C*R + np.sqrt(C*(C*R**2 - 4*L)))/(2*C*L)
```

On peut ensuite vérifier si les formules sont justes.

```
assert np.abs(L*C*p1**2+R*C*p1+1)<1e-8
assert np.abs(L*C*p2**2+R*C*p2+1)<1e-8
```

On calcule ensuite c_1 et c_2

```
c1,c2 = R/L*p1/(p1-p2), -R/L*p2/(p1-p2)
```

Puis on en déduit la réponse impulsionnelle.

```
y=c1*np.exp(t*p1)*(t>=0)+c2*np.exp(t*p2)*(t>=0)
y[t<0]=0
```

La dernière ligne a l'objectif de donner une valeur nulle lorsque $t < 0$ y compris quand les expressions de $y(t)$ pour $t \geq 0$ n'ont pas de sens.

Je propose de mettre toutes ces lignes de code dans une fonction notée `h1` dépendant de `R,L,C,t`.

Implémentation de la deuxième solution

La deuxième solution utilise le calcul déjà effectué de p_1 et p_2, que l'on peut retrouver avec

```
p1,p2=np.roots(np.array([L*C,R*C,1]))
```

Pour simplifier le code, je propose de définir une fonction notée `Hp` calculant $\frac{1}{j2\pi f - p}$

```
def Hp(f,p):
  return 1/(1j*2*np.pi*f-p)
```

Je considère trois fréquences

```
f=np.array([0,0.1,0.2])
```

La matrice contenant les paramètres à gauche de l'équation matricielle (1.13)

```
M = np.array([[Hp(f[0],p1),Hp(f[0],p2),1],[Hp(f[1],p1),Hp(f[1],p2),1],[H
```

Le vecteur colonne contenant les valeurs complexes souhaitées de la réponse fréquentielle sont les valeurs à droite de l'équation matricielle (1.13)

```
B = np.array([[H1(R,L,C,f[0])], [H1(R,L,C,f[1])], [H1(R,L,C,f[2])]])
```

L'ordinateur résout le système d'équations linéaires

```
d1,d2,d3=np.linalg.solve(M,B); d1,d2,d3=d1[0],d2[0],d3[0]
```

On vérifie qu'il était inutile de considérer un Dirac supplémentaire

```
assert np.abs(d3)<1e-10, (f"d3={d3:.2e}")
```

La réponse impulsionnelle trouvée est alors

```
y=d1*np.exp(p1*t)*(t>=0)+d2*np.exp(p2*t)*(t>=0)
y[t<0]=0
```

Je propose de la même façon de mettre ces lignes de code dans une deuxième fonction `h2`.

Détermination de $h(t)$ à partir de l'équation différentielle

On peut trouver $\tilde{h}$ en utilisant directement la fonction `sol_eq_diff` du module `seb` à partir de l'équation (1.4).

```
y1=seb.sol_eq_diff((L*C,R*C,1),t)
```

Notez que le premier argument est un `tuple` composé de trois arguments qui sont respectivement les coefficients devant $\frac{d^2}{dt^2}y(t)$, $\frac{d}{dt}y(t)$, $y(t)$.

On trouve alors $h(t)$ avec l'équation (1.5)

```
y2=R*C*seb.deriver(t,y1)
```

Je propose de mettre ces lignes de code dans une fonction `h3`.

Détermination de la réponse impulsionnelle à partir de la réponse fréquentielle

On utilise ici la fonction `H1` donnant la réponse fréquentielle du filtre étudié en utilisant (1.1). On évalue cette réponse fréquentielle en utilisant un très grand nombre de valeurs de fréquences. Puis on lui applique la transformée de Fourier inverse `TFI` définie dans `seb.py`.

```
f1=np.linspace(-20,20,10**5)
y1=seb.TFI(f1,H1(R,L,C,f1),t)
```

Comme on sait qu'avant $t = 0$, cette réponse impulsionnelle est nulle et que cette réponse impulsionnelle est réelle, on rajoute

```
y1[t<0]=0
y2=np.real(y1)
```

Je propose de mettre ces instructions dans une fonction `h4` et d'afficher sur un même graphique ces trois ou quatre fonctions.

1.2.4 Détermination de la réponse du système à $x_a(t) = \mathbb{T}(t)$

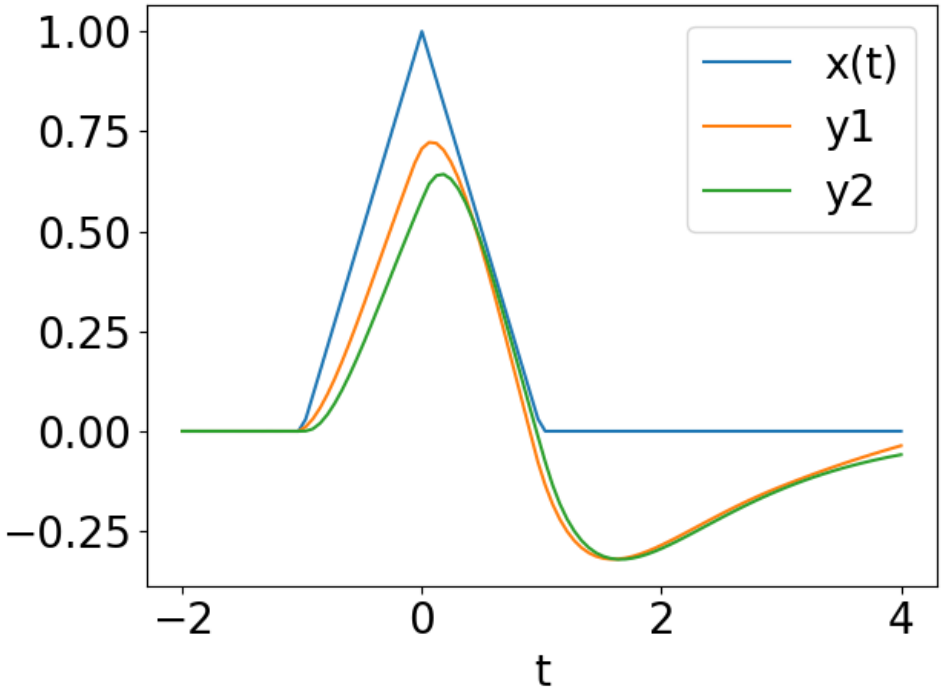

Figure 1.4: Courbe bleue en triangle : signal d'entrée. Les deux autres courbes sont le signal de sortie calculée de deux façons différentes et notéees `y1` et `y2`.

- Déterminez $y_{a1}(t)$ en utilisant la réponse impulsionnelle $h_1(t)$ ou $h_2(t)$ et la définition de $x_a(t)$ en utilisant la notion de produit de convolution.
- Calculez la réponse fréquentielle de la sortie $y_{a2}(t)$ en utilisant la réponse fréquentielle calculée en section 1.2.2 et la transformée Fourier de la fonction triangle pour en déduire $y_{a3}(t)$ avec

$$y_{a2}(t) = \mathrm{TF}^{-1}\left[\widehat{H}(f)\mathrm{TF}\left[\mathbb{T}(t)\right](f)\right](t) \tag{1.14}$$

  où $\widehat{H}(f)$ est la réponse fréquentielle du filtre considéré. Comme on sait que la sortie est réelle, il est intéressant de prendre la partie réelle de la sortie.

La figure 1.4 montre $y_{a1}(t)$, et y_{a2}. Ici l'implémentation est indiquée dans la section 1.4.3.

Utilisation du produit de convolution

Une fonction simulant le produit de convolution entre deux signaux est disponible sur `seb.py`. Pour calculer le produit de convolution de deux signaux, elle requière ces deux signaux et leurs échelles de temps ainsi que l'échelle de temps du nouveau signal généré.

On créé l'échelle de temps du premier signal avec la fonction **arange** de **numpy** qui permet d'utiliser une période d'échantillonnage déjà fixée $T_e = \frac{1}{fe}$. Ici **-1** est l'instant initial pour $x(t)$ et 1 est l'instant final pour $x(t)$.

```
tx = np.arange(-1,1,1/fe)
\begin{verbatim}
Pour choisir le début et la fin, il suffit de donner des valeurs qui per
de rallonger le signal sur des période où le signal est nul).
{\tt tx} est ensuite utilisée pour créer le signal d'entrée avec {\tt fo
\begin{verbatim}
x = seb.fonction_T(tx)
```

On créé une échelle de temps pour la réponse impulsionnelle. Il est important que cette deuxième échelle de temps ait la même période d'échantillonnage.

```
th = np.arange(0,4,1/fe)
```

De même que pour **tx**, le choix du début et de la fin de **th** est fait de façon à donner suffisamment d'informations. La valeurs de la réponse impulsionnelle sont récupérées avec **h1(R,L,C,th)** qui utilise la fonction **h1**.

On créé l'échelle de temps du signal à reconstruire

```
t=np.arange(-2,4,1/fe)
```

On obtient $y_{a1}(t)$ avec la fonction **convolution**

```
y1=seb.convolution(tx,x,th,h1(R,L,C,th),t)
```

Utilisation de la transformée de Fourier et de la transformée de Fourier inverse

Cette fois-ci on a besoin d'une échelle de temps pour calculer la réponse fréquentielle de $x(t)$. Celle-ci doit être suffisemment détaillée pour donner une bonne approximation à $\mathrm{TF}[x(t)](f)$.

```
tx=np.linspace(-1,1,10**4)
```

On calcule $x(t)$ sur cette autre échelle de temps

```
x=seb.fonction_T(t)
```

On a besoin d'une échelle de fréquence assez détaillée pour en calculer la transformée de Fourier inverse.

```
f=np.linspace(-3,3,10**4)
```

On calcule $\mathrm{TF}[x(t)](f)$ avec

```
X=seb.TF(t,x,f)
```

On obtient alors $\widehat{Y}(f)$ la transformée de Fourier de la sortie

```
Y=H1(f)*X
```

On choisit une échelle de temps pour `y2`, qui ne sert que pour l'affichage.

```
ty=np.linspace(-2,4,10**2)
```

On calcule la sortie en utilisant la transformée de Fourier inverse et en considérant la partie réelle.

```
y=np.real(seb.TFI(f,Y,ty)
```

1.3 Travail à rendre une semaine après la séance

En vous inspirant du travail effectué précédemment, répondez aux questions suivantes. Cette partie est à mettre après la partie correspondant à la section 1.1.

7. Créez une échelle de fréquences avec le vecteur `f` par exemple entre -3 et 3 et représentez graphiquement cette réponse fréquentielle.

8. Créez une échelle de temps avec le vecteur `t` par exemple entre -3 et 3 et représentez graphiquement cette réponse impulsionnelle soit avec `h1` notée $h_1(t)$ soit avec `h2` notée $h_2(t)$.

9. Déterminez $h_3(t)$ la réponse impulsionnelle à partir de la réponse fréquentielle (1.1)

10. Déterminez $h_4(t)$ la réponse impulsionnelle à partir de l'équation différentielle (1.5).

11. Choisissez un signal d'entrée noté $x(t)$.

12. Déterminez $y_1(t)$ la réponse du filtre à $x(t)$ en utilisant la réponse impulsionnelle $h_1(t)$ ou $h_2(t)$ et $x(t)$.

13. Calculez la réponse fréquentielle de la sortie $y_2(t)$ en utilisant la réponse fréquentielle et la simulation de la transformée de Fourier du signal d'entrée choisi.

$$y_2(t) = \mathrm{TF}^{-1}\left[\widehat{H}(f)\mathrm{TF}\left[\mathbb{T}(t)\right](f)\right](t) \tag{1.15}$$

Le document à rendre à un `pdf` dont la première page doit lister toutes les figures, toutes les formules et toutes les réponses aux questions. La suite du document expliquant la justification des réponses et/ou les programmes utilisés.

1.4 Codes permettant de réaliser le TP1

1.4.1 Figure 1.2

```
def H1(R,L,C,f):
  """fonction de la reponse frequentielle obtenue à partir du montage da
  import numpy as np
  H=1j*2*np.pi*f*R*C/(1+1j*2*np.pi*f*R*C-4*(np.pi**2)*(f**2)*L*C)
  return H
```

```
R=3; C=0.5; L=1;
f=np.linspace(-3,3,10**3)
fig,ax = plt.subplots()
ax.plot(f,np.abs(H1(R,L,C,f)),label='|H(f)|')
ax.set_xlabel('f')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_TP1_fig1a.png')
fig.show()
```

1.4.2 Figure 1.3

```
def h1(R,L,C,t):
  """fonction obtenue avec les calculs theoriques solution 1"""
  p1=(-C*R - np.sqrt(C*(C*R**2 - 4*L)))/(2*C*L)
  p2=(-C*R + np.sqrt(C*(C*R**2 - 4*L)))/(2*C*L)
  assert np.abs(L*C*p1**2+R*C*p1+1)<1e-8, 'p1 erronnee'
  assert np.abs(L*C*p2**2+R*C*p2+1)<1e-8, 'p2 erronnee"'
  c1,c2 = R/L*p1/(p1-p2), -R/L*p2/(p1-p2)
  y=c1*np.exp(t*p1)*(t>=0)+c2*np.exp(t*p2)*(t>=0)
  y[t<0]=0
  return y
```

```
def h2(R,L,C,t):
  """ calcule d1,d2,d3,p1,p2 en fonction de R,L,C pour ensuite calculer
  p1,p2=np.roots(np.array([L*C,R*C,1]))
  f=np.array([0,0.1,0.2])
  def Hp(f,p):
    return 1/(1j*2*np.pi*f-p)

  M = np.array([[Hp(f[0],p1),Hp(f[0],p2),1],[Hp(f[1],p1),Hp(f[1],p2),1],
  B = np.array([[H1(R,L,C,f[0])], [H1(R,L,C,f[1])], [H1(R,L,C,f[2])]])
  d1,d2,d3=np.linalg.solve(M,B); d1,d2,d3=d1[0],d2[0],d3[0]
  assert np.abs(d3)<1e-10, (f"d3={d3:.2e}")
  y=d1*np.exp(p1*t)*(t>=0)+d2*np.exp(p2*t)*(t>=0)
  y[t<0]=0
  return y


def h3(R,L,C,t):
  """ utilise l'équation differentielle en fonction de R,L,C pourcalcule
  y1=seb.sol_eq_diff((L*C,R*C,1),t)
```

```
  y2=R*C*seb.deriver(t,y1)
  y2[t<0]=0
  return y2

def h4(R,L,C,t):
  """ utilise la reponse frequentielle en fonction de R,L,C pourcalculer
  import seb
  plt,np=seb.debut()
  f1=np.linspace(-20,20,10**5)
  y1=seb.TFI(f1,H1(R,L,C,f1),t)
  Te=t[1]-t[0]
  y1[t<0]=0
  return y1

R=3; C=0.5; L=1;
t=np.linspace(-3,3,10**2)
plt.close('all')
fig,ax = plt.subplots()
ax.plot(t,h1(R,L,C,t),label='h1')
ax.plot(t,h2(R,L,C,t),label='h2')
ax.plot(t,h3(R,L,C,t),label='h3')
ax.plot(t,h4(R,L,C,t),label='h4')
```

```
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/TP1_fig2a.png')
fig.show()
```

1.4.3 Simulation de la figure 1.4

On utilise `h1` défini dans 1.4.2 et `H1` défini dans 1.4.1.

```
def y1(R,L,C,t):
  """réponse à une fonction triangle utilisant h1(t)"""
  import seb
  plt,np=seb.debut()
  Te=t[1]-t[0]
  th=np.arange(0,4,Te)
  tx=np.arange(-1,1,Te)
  x=seb.fonction_T(tx)
  y=seb.convolution(tx,x,th,h1(R,L,C,th),t)
  return y
```

```
def y2(R,L,C,t):
  """réponse à une fonction triangle en utilisant la réponse fréquentiel
  import seb
  plt,np=seb.debut()
  f=np.linspace(-10,10,10**4)
  Y=H1(R,L,C,f)*seb.TF(t,seb.fonction_T(t),f)
  y=np.real(seb.TFI(f,Y,t))
  return y



R=3; C=0.5; L=1;
t=np.linspace(-2,4,10**2)
plt.close('all')
fig,ax = plt.subplots()
ax.plot(t,seb.fonction_T(t),label='x(t)')
ax.plot(t,y1(R,L,C,t),label='y1')
ax.plot(t,y2(R,L,C,t),label='y2')
ax.set_xlabel('t')
ax.legend()
```

```
plt.tight_layout()
fig.savefig('./figures/fig_TP1_fig3a.png')
fig.show()
```

Chapter 2

Séance 2 de travaux pratiques

2.1 Préparation à faire avant la séance

1. Choisissez un signal ayant une représentation graphique simple. C'est ce signal noté $s_2(t)$ que vous étudierez à la place de $s_1(t)$ représenté sur la figure 2.1. Ce signal doit être non-nul seulement sur un intervalle petit (il n'est donc pas périodique).

2. Explicitez $s_2(t)$ au moyen des fonctions de base présentées dans le cours de façon similaire à l'équation (2.1).

3. Calculez l'intégrale de ce signal $\int_{-\infty}^{+\infty} s_2(t)\,dt$.

4. Si possible calculez la transformée de Fourier $\widehat{S}_2(f)$, ou donnez un algorithme permettant de le faire.

2.2 Travail à effectuer pendant la séance

2.2.1 Signal étudié

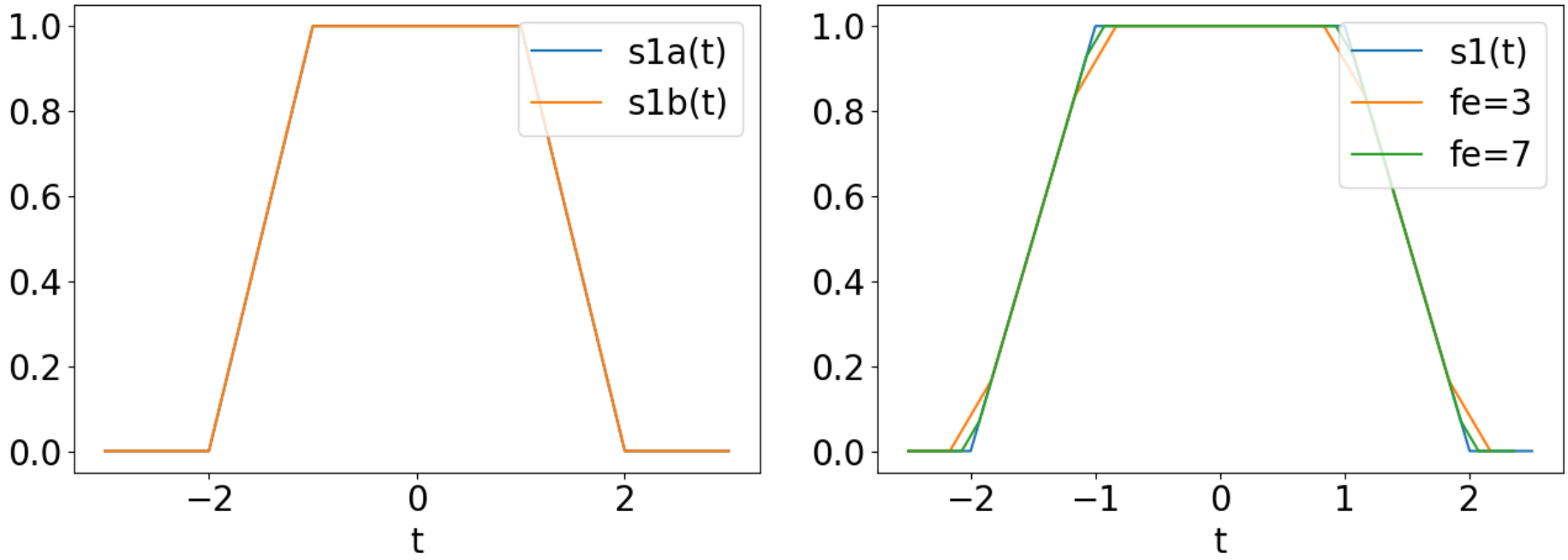

Figure 2.1: Signal étudié $s_1(t)$ et deux échantillonnages

Graphiquement on voit que le signal peut se mettre sous deux formes, la première

notée $s_{1a}(t)$ et la deuxième notée $s_{1b}(t)$.

$$s_1(t) = 2\mathbb{T}(t/2) - \mathbb{T}(t) = \mathbb{C}(t+1.5) + \Pi(t/2) + \mathbb{D}(t-1.5) \tag{2.1}$$

La gauche de la figure 2.1 montre le signal étudié $s_1(t)$ avec ces deux formes. La droite montre l'échantillonnage avec deux fréquences d'échantillonnages. L'implémentation est réalisée dans la section 2.4.1.

2.2.2 Calculs théoriques sur le signal $s_1(t)$

Graphiquement on peut voir la surface du signal qui est composé de deux triangles et d'un rectangle, il a donc une surface de $2(1\times1/2) + (1\times2) = 3$.

$$\int_{-\infty}^{+\infty} s_1(t)\,dt = 3 \tag{2.2}$$

Je propose d'utiliser l'expression de $s_1(t)$ en fonction de $\mathbb{T}(t)$ dont la transformée de Fourier vaut $\text{sinc}^2(f)$:

$$\widehat{S}_1(f) = 2\text{TF}\left[\mathbb{T}(t/2)\right](f) - \text{TF}\left[\mathbb{T}(t)\right](f) = 4\text{TF}\left[\mathbb{T}(t)\right](2f) - \text{TF}\left[\mathbb{T}(t)\right](f) = 4\,\text{sinc}^2(2f) -$$

On remarque que ces deux équations sont cohérentes : $3 = \widehat{S}_1(0) = 4 - 1$

2.2.3 Échantillonnage du signal

La droite de la figure 2.1 montre le signal étudié $s_1(t)$ et deux échantillonnages faits à deux fréquences différentes, ici $fe_a = 3\text{Hz}$ et $fe_b = 7\text{Hz}$.

Python :

L'équation (2.1) permet de définir une fonction

```
def s1a(t):
  return seb.fonction_T(t/2)*2-seb.fonction_T(t)
```

Pour simuler le ignal **s1** en tant que signal temps continu, on créé une échelle de temps très précise.

```
t=np.linspace(-2,2,10**4)
s1=s1a(t)
```

La fréquence d'échantillonnage plus élevée permettra de faire des calculs un peu plus précis. Pour simuler ce même signal en tant que signal temps discret avec une fréquence d'échantillonnage de $f_e = 3$Hz, on utilise la fonction **np.arange** qui permet de préciser exactement la fréquence d'échantillonnage choisie.

```
fe1=3
t=np.arange(-2,2,1/fe1)
s1=s1a(t)
```

L'implémentation de la droite de la figure 2.1 est réalisée dans la section 2.4.2.

2.2.4 Simulation de la transformée de Fourier

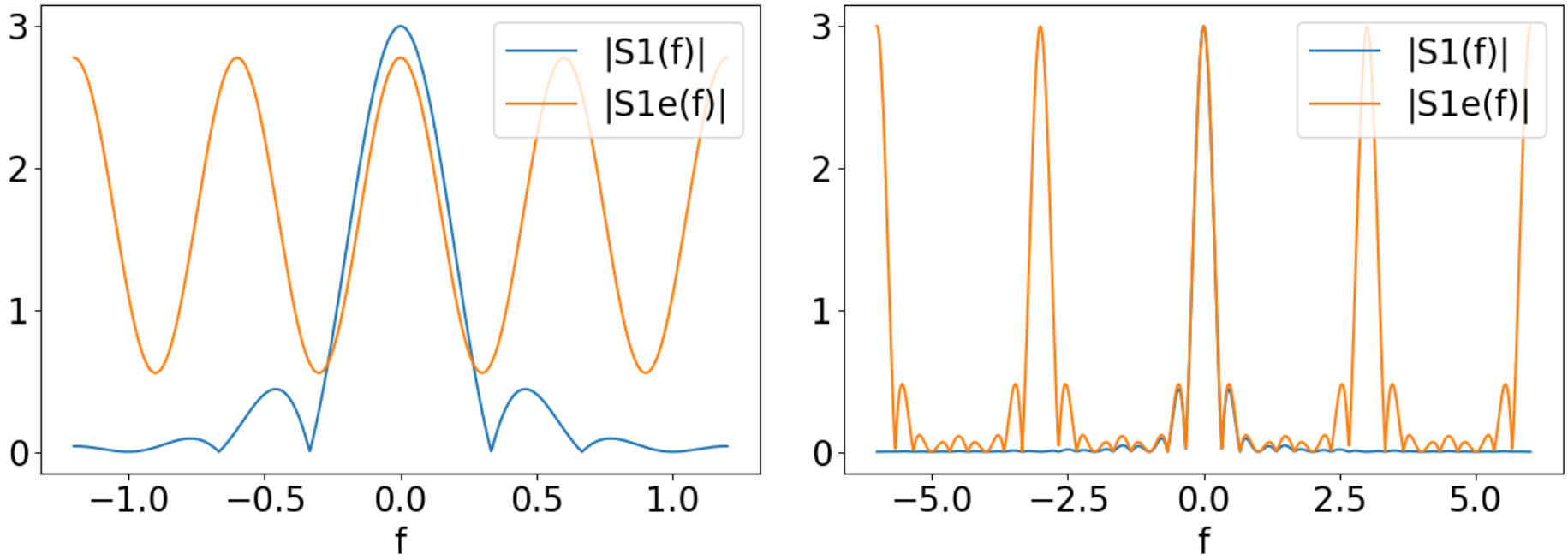

Figure 2.2: Spectre de $s_1(t)$ et à gauche de $s_1(t)$ échantillonné à 0.6Hz et à droite à 3Hz.

La figure 2.2 montre à gauche et à droite deux spectres périodique, ce sont les transformées de Fourier des signaux échantillonnés. C'est le même spectre en bleu et non-périodique qui est représenté à gauche et à droite.

Python :

La fonction `seb.TFTD` implémente la transformée de Fourier à temps discret, son utilisation est similare à `seb.TF`, on précise une échelle de temps et les valeurs du signal

ainsi qu'une échelle en fréquence souhaitée. Pour bien montrer la périodicité de période f_e, je considère une échelle de fréquence entre $-3f_e$ et $3f_e$.

```
fe1=3
t1=np.arange(-2,2,1/fe1)
f=np.linspace(-3*fe1,3*fe1,300)
S1e_a=seb.TFTD(ta,s1a(t1),f)
```

L'implémentation est réalisée dans la section 2.4.3.

2.2.5 Simulation de la périodisation du spectre

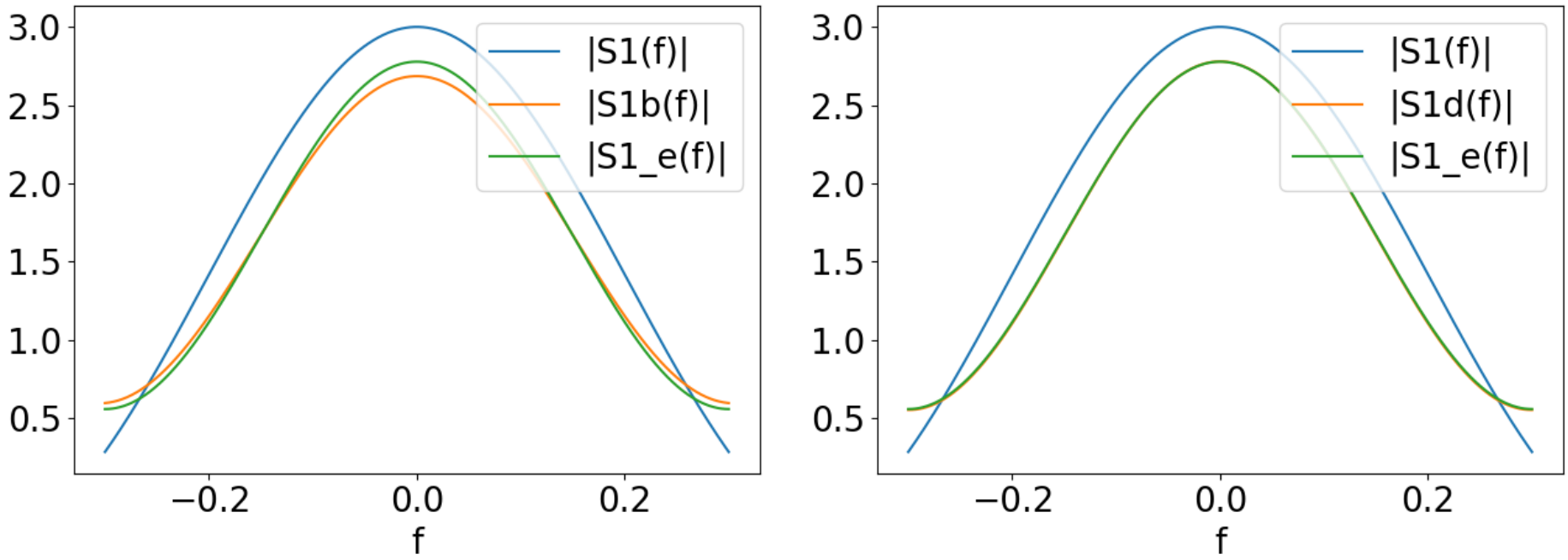

Figure 2.3: Spectre de $s_1(t)$ et à gauche de $s_1(t)$ échantillonné à 0.6Hz et à droite à 3Hz.

- Représentez $\left|\widehat{S}_1(f)\right|, \left|\widehat{S}_1(f) + \widehat{S}_1(f+f_e) + \widehat{S}_1(f-f_e)\right|, \left|\widehat{S}_1(f) + \widehat{S}_1(f+f_e) + \widehat{S}_1(f-f\right.$ et $\frac{1}{f_e}\left|\widehat{S}e_1(f)\right|$ avec une représentation centrée. Vous pouvez utiliser une version approchée ou exacte de $\widehat{S}_1(f)$.

La figure 2.3 montre dans les deux figures en haut $S_1(f)$ et avec une forme plus en cloche $\widehat{S}_1^{\#}(f) = \frac{1}{f_e}\text{TFTD}\left[s_1[n]\right](f)$. À gauche, $\widehat{S}_1^{\#}(f)$ est approchée avec $\left|\widehat{S}_1(f) + \widehat{S}_1(f+f_e) + \widehat{S}_1(f\right.$ on voit la différence entre les deux courbes. Cette différence avec $\widehat{S}_1^{\#}(f)$ disparaît à droite en visualisant $\left|\widehat{S}_1(f) + \widehat{S}_1(f+f_e) + \widehat{S}_1(f-f_e) + \widehat{S}_1(f+2f_e) + \widehat{S}_1(f-2f_e)\right|$. L'implémenta est réalisée dans la section 2.4.4.

2.2.6 Périodisation de $s_1(t)$

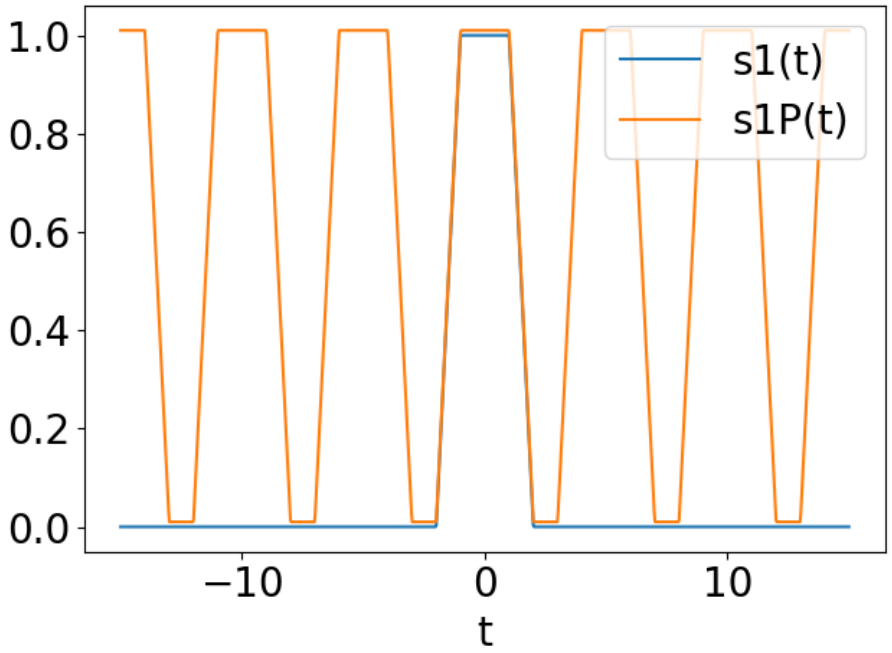

Figure 2.4: Signal $s_1(t)$ en bleu périodisé en $s_1^P(t)$ en répétant le motif défini sur $[-\frac{5}{2}, \frac{5}{2}]$.

La figure 2.4 montre $s_1(t)$ et $s_1^P(t)$ périodisés. La courbe de $s_1^P(t)$ est légèrement surélevée pour montrer la différence avec $s_1(t)$. L'implémentation est dans 2.4.5.

2.2.7 Simulation de la transformée de Fourier

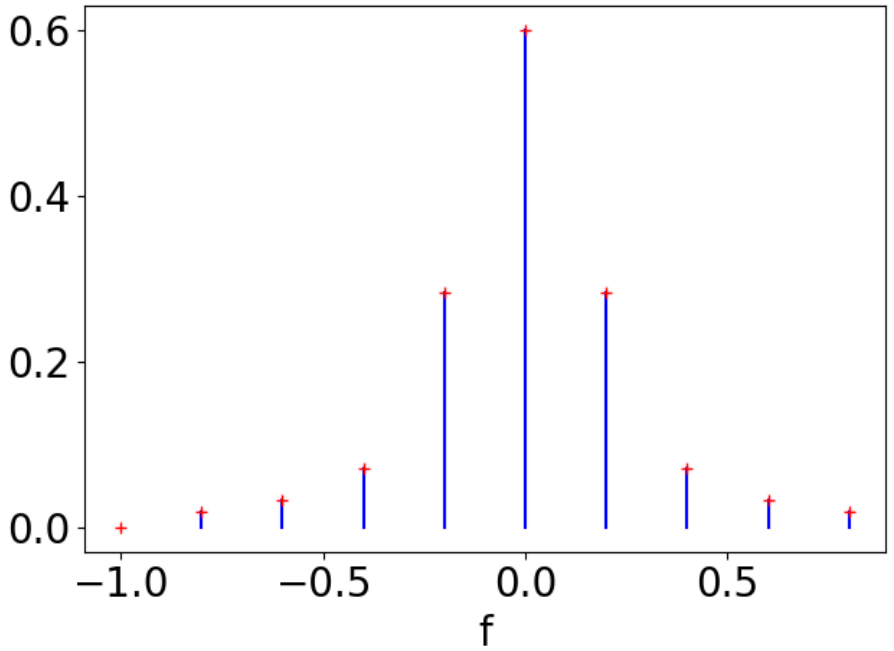

Figure 2.5: Signal $s_1(t)$ en bleu périodisé en $s_1^P(t)$ en répétant le motif $[-\frac{5}{2}, \frac{5}{2}]$.

La figure 2.5 montre des raies représentant les coefficients de la série de Fourier de $s_1^P(t)$. Comme la restriction de $s_1^P(t)$ sur $[-\frac{5}{2}, \frac{5}{2}]$ coïncide avec $s_1(t)$, ces raies peuvent être obtenues avec $\widehat{S}_1(f)$ pour $f_k = \frac{k}{T}$, ces points sont les plus indiqués en rouge.

L'implémentation est dans 2.4.7.

2.2.8 Simulation de la périodisation de la transformée de Fourier

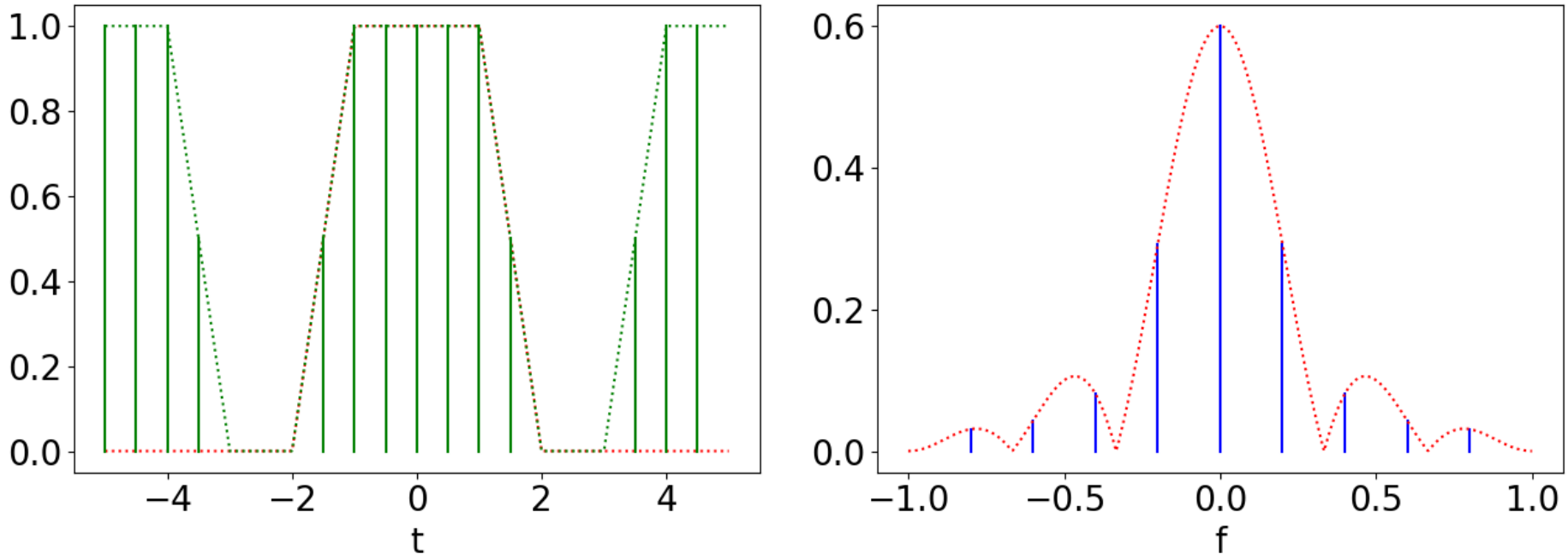

Figure 2.6

- Choisissez une fréquence d'échantillonnage f_{ec} tel que la période T soit un multiple de $\frac{1}{fe_c}$. On note $N = Tfe_c$.
- On note $s_1^{\#P}[n]$ le signal $s_1^P(t)$ échantillonné à fe_c. Représentez $s_1^{\#P}[n], s_1^{\#}[n], s_1^P(t)$ sur le même graphique.
- Représentez sur la même figure $\text{TFD}[s_1^{\#P}[n]][k]$ et $\frac{1}{N}\text{TFTD}[s_1^{\#P}[n]]\left(k\frac{fe_c}{N}\right)$

La figure 2.6 est implémentée dans 2.4.6

2.3 Travail à rendre une semaine après la séance

5. Choisissez deux fréquence d'échantillonnage fe_a et fe_b, montrez que les transformées de Fourier des signaux échantillonnés ne sont pas les mêmes que celle du signal non-échantillonné et qu'elles sont périodiques de période f_e.

2.4 Codes Python permettant de réaliser le TP2

2.4.1 Code pour générer la gauche de la figure 2.1

```
def s1a(t):
  return seb.fonction_T(t/2)*2-seb.fonction_T(t)

def s1b(t):
  x1 = seb.retarder(t,seb.fonction_C(t),-1.5)
  x2 = seb.fonction_P(t/2)
  x3 = seb.retarder(t,seb.fonction_D(t),1.5)
  return x1+x2+x3

t=np.linspace(-3,3,10**3)
fig,ax = plt.subplots()
ax.plot(t,s1a(t),label='s1a(t)')
```

```
ax.plot(t,s1b(t),label='s1b(t)')
#ax.plot(t,y1(R,L,C,t),label='y1')
#ax.plot(t,y2(R,L,C,t),label='y2')
#ax.plot(t,seb.retarder(t,y1(R,L,C,t),1),label='y2')
#ax.plot(t,y3(R,L,C,t),label='y3')
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_TP2_fig1.png')
fig.show()
```

2.4.2 Droite de la figure 2.1.

```
def s1(t):
  return seb.fonction_T(t/2)*2-seb.fonction_T(t)

fe_a=3; fe_b=7
t=np.linspace(-2.5,2.5,10**3)
ta=np.arange(-2.5,2.5,1/fe_a)
tb=np.arange(-2.5,2.5,1/fe_b)
fig,ax = plt.subplots()
ax.plot(t,s1a(t),label='s1(t)')
```

```
ax.plot(ta,s1(ta),label='fe=3')
ax.plot(tb,s1(tb),label='fe=7')
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_TP2_fig2.png')
fig.show()
```

2.4.3 Figure 2.2

On utilise la fonction `s1` définie dans 2.4.2

```
t=np.linspace(-2,2,10**3)
fe_a=0.6
f=np.linspace(-2*fe_a,2*fe_a,10**3)
S1=seb.TF(t,s1(t),f)
ta=np.arange(-2,2,1/fe_a)
t0=seb.find_nearest(ta,0)
ta=ta-t0
S1e_a=seb.TFTD(ta,s1(ta),f)
fig,ax = plt.subplots()
ax.plot(f,np.abs(S1),label='|S1(f)|')
ax.plot(f,np.abs(S1e_a)/fe_a,label='|S1e(f)|')
```

```
ax.set_xlabel('f')
ax.legend()
plt.tight_layout()

fig.savefig('./figures/fig_TP2_fig6a.png')
fig.show()
t=np.linspace(-2,2,10**3)
fe_a=3
f=np.linspace(-2*fe_a,2*fe_a,10**3)
S1=seb.TF(t,s1(t),f)
ta=np.arange(-2,2,1/fe_a)
S1e_a=seb.TFTD(ta,s1(ta),f)
fig,ax = plt.subplots()
ax.plot(f,np.abs(S1),label='|S1(f)|')
ax.plot(f,np.abs(S1e_a)/fe_a,label='|S1e(f)|')
ax.set_xlabel('f')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_TP2_fig6b.png')
fig.show()
```

2.4.4 figure 2.3

On utilise la fonction `s1` définie dans 2.4.4

```
import seb; plt,np=seb.debut();
def s1(t):
  return seb.fonction_T(t/2)*2-seb.fonction_T(t)
def S1_th(f):
  return 4*(np.sinc(2*f)**2)-(np.sinc(f)**2)

plt.close('all')
t=np.linspace(-2,2,10**3)
fe=0.6
f=np.linspace(-fe/2,fe/2,10**3)
S1=seb.TF(t,s1(t),f)
S1a=seb.TF(t,s1(t),f+fe)+S1
S1b=seb.TF(t,s1(t),f-fe)+S1a
S1c=seb.TF(t,s1(t),f-2*fe)+seb.TF(t,s1(t),f+2*fe)+S1b
ta=seb.synchroniser(np.arange(-2,2,1/fe))
S1e_a=seb.TFTD(ta,s1(ta),f)
fig,ax = plt.subplots()
ax.plot(f,np.abs(S1),label='|S1(f)|')
ax.plot(f,np.abs(S1b),label='|S1b(f)|')
```

```
ax.plot(f,np.abs(S1e_a)/fe,label='|S1_e(f)|')
ax.set_xlabel('f')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_TP2_fig7a.png')
fig.show()

t=np.linspace(-2,2,10**3)
fe=0.6
f=np.linspace(-fe/2,fe/2,10**3)
ta=seb.synchroniser(np.arange(-2,2,1/fe))
S1e_a=seb.TFTD(ta,s1(ta),f)
fig,ax = plt.subplots()
ax.plot(f,np.abs(S1_th(f)),label='|S1(f)|')
ax.plot(f,np.abs((S1_th(f)+S1_th(f+fe)+S1_th(f-fe)+S1_th(f+2*fe)+S1_th(f
  +S1_th(f+3*fe)+S1_th(f-3*fe)+S1_th(f+4*fe)+S1_th(f-4*fe)
)),label='|S1d(f)|')
ax.plot(f,np.abs(S1e_a)/fe,label='|S1_e(f)|')
ax.set_xlabel('f')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_TP2_fig7b.png')
```

```
fig.show()
```

2.4.5 Figure 2.4

```
import seb; plt,np=seb.debut();
def s1(t):
  return seb.fonction_T(t/2)*2-seb.fonction_T(t)
def S1_th(f):
  return 4*(np.sinc(2*f)**2)-(np.sinc(f)**2)
T=5
t1=np.linspace(-3*T,3*T,10**3)
t2=seb.periodiser_ech_t(t1,(-2.5,2.5))
plt.close('all')
fig,ax=plt.subplots()
ax.plot(t1,s1(t1),label='s1(t)')
ax.plot(t1,s1(t2)+0.01,label='s1P(t)')
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_TP2_fig8a.png')
fig.show()
```

2.4.6 Figure 2.6

```
import seb; plt,np=seb.debut();
def s1(t):
  return seb.fonction_T(t/2)*2-seb.fonction_T(t)
T=5; N=10; fe=N/T
t=seb.synchroniser(np.arange(-T/2,T/2,1/fe))
assert len(t)==N
assert max(t)<T/2, ('il faut eviter que le signal contienne deux fois la
s1DP=s1(t)
f,S1DP=seb.TFD(t,s1DP,(-T/2,T/2),True)
ta=seb.synchroniser(np.arange(-T/2,T/2,1/fe))
S1e=lambda f:seb.TFTD(ta,s1(ta),f)
f2=np.linspace(-fe/2,fe/2,10**3)

plt.close('all')
t1=np.linspace(-T,T,10**3)
t2=np.arange(-T,T,1/fe)
t3=seb.periodiser_ech_t(t2,(-T/2,T/2))
print('fe=',fe,' S1[0]=',np.sum(s1(t1))*(t1[1]-t1[0]),' S1D[0]=',np.sum(
  np.sum(s1(seb.periodiser_ech_t(t1,(-T/2,T/2))))/len(t1),' S1DP[0]=',np
)
```

```
fig,ax=plt.subplots()
ax.plot(t1,s1(t1),'r:')
ax.plot(t1,s1(seb.periodiser_ech_t(t1,(-T/2,T/2))),'g:')
for t_ in range(len(t2)):
  ax.plot([t2[t_],t2[t_]],[0,s1(t1[t_])],'b-')
  ax.plot([t2[t_],t2[t_]],[0,s1(t3[t_])],'g-')
ax.set_xlabel('t')
# ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_TP2_fig10a.png')
fig.show()

fig,ax=plt.subplots()
for f_ in range(len(f)):
  ax.plot([f[f_],f[f_]],[0,np.abs(S1DP[f_])],'b-')

ax.plot(f2,np.abs(S1e(f2))/N,'r:')
ax.set_xlabel('f')
plt.tight_layout()
fig.savefig('./figures/fig_TP2_fig10b.png')
fig.show()
```

2.4.7 Figure 2.5

```
import seb; plt,np=seb.debut();
def s1(t):
  return seb.fonction_T(t/2)*2-seb.fonction_T(t)
def S1_th(f):
  return 4*(np.sinc(2*f)**2)-(np.sinc(f)**2)
T=5
t=np.linspace(-T/2,T/2,10**3)
k=np.arange(-5,5,1)
f,S1P=seb.coef_serie_Fourier(t,s1(t),(-T/2,T/2),k)
plt.close('all')
fig,ax=plt.subplots()
for f_ in range(len(f)):
  ax.plot([f[f_],f[f_]],[0,np.abs(S1P[f_])],'b-')
ax.plot(f,np.abs(S1_th(f))/T,'r+')
ax.set_xlabel('f')
# ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_TP2_fig9a.png')
fig.show()
```

Chapter 3

Séance 3 de travaux pratiques

3.1 Préparation à faire avant la séance

1. Choisissez la distribution de probabilité du bruit blanc considéré en déduire la moyenne et la variance associée notée $\overset{r}{X}(t)$.
2. Choisissez éventuellement une forme particulière entre deux instants successifs.
3. Choisissez un filtre défini par une équation différentielle, on note ici $\mathcal{H}$ sa relation entrée sortie. L'entrée est $\overset{r}{X}(t)$ et la sortie est $\overset{r}{Y}(t)$.
4. Calculez la réponse impulsionnelle $h(t)$, (ou donnez un algorithme).

5. Calculez l'autocorrélation théorique associée à $\overset{r}{Y}(t)$ notée $R_y(t)$.

6. Calculez la réponse fréquentielle $\widehat{H}(f)$, (ou donnez un algorithme).

7. Calculez la densité spectrale théorique $\widehat{S}_y(f)$.

8. Choisissez une fréquence d'échantillonnage f_e pour simuler le bruit considéré.

9. Choisissez deux instants d'observation t_0 et t_1.

3.2 Travail à effectuer pendant la séance

Choix du sujet d'étude

Dans un premier temps, on considère dans ce TP, une loi binômiale

$$\mathrm{P}(\overset{r}{B}(t) = 1) = 0.5 \text{ et } \mathrm{P}(\overset{r}{B}(t) = -1) = 0.5 \tag{3.1}$$

On ne considère pas de forme particulière, c'est-à-dire que lors de la simulation ce sont des points également répartis qui sont tirés aléatoirement.

On considère le filtre $\mathcal{H}$ défini par cette équation différentielle.

$$\frac{d}{dt}y(t) + y(t) = x(t-1) \tag{3.2}$$

Calcul de la moyenne statistique et de la variance

La moyenne est donnée par

$$\mathrm{E}[\overset{r}{B}(t)] = 1\times\mathrm{P}(\overset{r}{B}(t)=1) + (-1)\times\mathrm{P}(\overset{r}{B}(t)=-1) = 1\times 0.5 - 1\times 0.5 = 0 \tag{3.3}$$

La variance est donnée par

$$\mathrm{Var}[\overset{r}{B}(t)] = (1-0)^2\times\mathrm{P}(\overset{r}{B}(t)=1) + (-1-0)^2\times\mathrm{P}(\overset{r}{B}(t)=-1) = 1\times 0.5 + 1\times 0.5 = 1 \tag{3.4}$$

Calcul de la réponse impulsionnelle

Je pose $\tilde{h}(t)$ la solution de l'équation différentielle

$$\frac{d}{dt}y(t) + y(t) = \delta(t) \tag{3.5}$$

Le polynôme associé est $p+1$, il a une racine $p=-1$, donc les solutions sont de type

$$\tilde{h}(t) = Ae^{-t}[\![t \geq 0]\!](t) \tag{3.6}$$

Pour trouver A, je remplace $\tilde{h}(t)$ dans l'équation (3.5) et je trouve

$$\delta(t) = \frac{d}{dt}\tilde{h}(t) + \tilde{h}(t) = \left(-Ae^{-t}[\![t \geq 0]\!](t) + A\delta(t)\right) + Ae^{-t}[\![t \geq 0]\!](t) = A\delta(t) \tag{3.7}$$

J'en déduis que $A = 1$ et $\tilde{h}(t) = e^{-t}[\![t \geq 0]\!](t)$

La réponse impulsionnelle de $\mathcal{H}$ est obtenue en mettant en entrée $x(t) = \delta(t)$ qui devient $x(t-1) = \delta(t-1)$ qui modifie la relation entrée-sortie de (3.5) en retardant l'entrée de $t = 1$ et par suite en retardant la sortie de $t = 1$. Donc $h(t) = \tilde{h}(t-1)$.

$$h(t) = e^{-(t-1)}[\![t \geq 1]\!](t) \tag{3.8}$$

Calcul de l'autocorrélation théorique

J'utilise pour $h(t)$ l'équation (3.8). L'autocorrélation de $h(t)$ est

$$R_h(t) = \int_{-\infty}^{+\infty} h(\tau)h(\tau - t)\, d\tau \tag{3.9}$$

Je sais que $R_h(t)$ est pair et je considère ici $t \geq 0$.

$$R_h(t) = \int_{t+1}^{+\infty} e^{-(\tau-1)}e^{-(\tau-t-1)}\, d\tau = e^{t+2}\int_{t+1}^{+\infty} e^{-2\tau}\, d\tau = \frac{e^{-t}}{2} \tag{3.10}$$

Par symétrie on obtient $R_h(t) = \frac{e^{-|t|}}{2}$

Comme l'entrée $\overset{r}{X}(t)$ est un bruit blanc centré d'écart-type 1, $R_x(t) = \delta(t)$, la sortie est aussi

$$R_y(t) = \delta(t) * R_h(t) = \frac{e^{-|t|}}{2} \tag{3.11}$$

Calcul de la réponse fréquentielle

À partir de l'équation différentielle (3.2) et en remarquant qu'un retard devient un déphasage proportionnel à la fréquence, on a

$$\widehat{H}(f) = \frac{e^{-j2\pi f}}{1 + j2\pi f} \tag{3.12}$$

Modification de la densité spectrale par le filtre

Je note $S_x(f), S_y(f)$ les densités spectrales de puissance de $\overset{r}{X}(t)$ et $\overset{r}{Y}(t)$.
Du coup

$$\frac{S_y(f)}{S_x(f)} = S_h(f) = |\widehat{H}(f)|^2 = \frac{1}{1 + 4\pi^2 f^2} \tag{3.13}$$

Modification de l'énergie par le filtre

Je note l'énergie E_x et E_y les énergies en entrée et en sortie du filtre en supposant ici que ces énergies existent.

$$E_y = \mathrm{R}_y(0) = \int_{-\infty}^{+\infty} S_y(f)\, df = \int_{-\infty}^{+\infty} \frac{1}{1 + 4\pi^2 f^2} S_x(f)\, df \tag{3.14}$$

Au moyen de deux changements de variable $u = 2\pi f$ et $\theta = \arctan(u)$, on calcule l'intégrale

$$\int_{-\infty}^{+\infty} \frac{df}{1+4\pi^2 f^2} = \frac{1}{2\pi}\int_{-\infty}^{+\infty} \frac{du}{1+u^2} = \frac{1}{2\pi}\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} d\theta = \frac{1}{2} \tag{3.15}$$

Finalement on obtient

$$\frac{E_y}{E_x} = \frac{1}{2} \tag{3.16}$$

Modification de l'autocorrélation du signal d'entrée par le filtre.

Je note $\mathrm{R}_x(t), \mathrm{R}_h(t), \mathrm{R}_y(t)$ les autocorrélations de l'entrée, du filtre et de la sortie.

$$\mathrm{R}_h(t) = \mathrm{TF}^{-1}\left[S_h(f)\right](t) \tag{3.17}$$

Pour trouver cette transformée de Fourier inverse, on suppose qu'elle s'écrit sous la forme $\mathrm{R}_h(t) = be^{-a|t|}$ La valeur absolue fait qu'on découpe en deux l'intégrale à effectuer.

$$S_h(f) = b\int_{-\infty}^{0} b^{at}e^{-j2\pi ft}\,dt + b\int_{0}^{+\infty} b^{-at}e^{-j2\pi ft}\,dt \tag{3.18}$$

La deuxième intégrale vaut

$$\int_{0}^{+\infty} e^{-at}e^{-j2\pi ft}\,dt = \left[\frac{e^{-t(a+j2\pi f)}}{a+j2\pi f}\right]_0^{+\infty} = \frac{1}{a+j2\pi f} \tag{3.19}$$

Après un changement de variable $t' = -t$, la première intégrale vaut

$$\int_{-\infty}^{0} e^{at} e^{-j2\pi ft}\, dt = \int_{0}^{+\infty} e^{-at} e^{j2\pi ft}\, dt = \int_{0}^{+\infty} e^{-at} e^{-j2\pi(-f)t}\, dt = \frac{1}{a + j2\pi(-f)} \quad (3.20)$$

Du coup avec les deux termes on trouve que

$$S_h(f) = b\left(\frac{1}{a + j2\pi f} + \frac{1}{a - j2\pi f}\right) = \frac{2ab}{a^2 + 4\pi^2 f^2} \quad (3.21)$$

Par identification avec la vraie valeur de $S_y(f)$, on trouve que $a = 1$ et $b = 0.5$.

$$\mathrm{R}_h(t) = 0.5e^{-|t|} \quad (3.22)$$

Finalement on a

$$\mathrm{R}_y(t) = \mathrm{R}_h(t) * \mathrm{R}_x(t) \quad (3.23)$$

3.2.1 Estimation de la réponse impulsionnelle

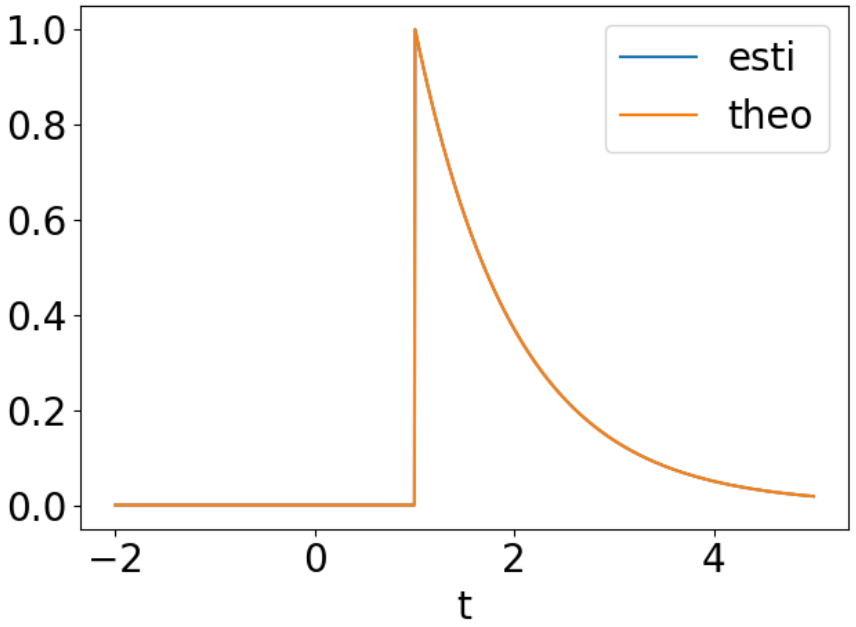

Figure 3.1: Réponse impulsionnelle estimée avec l'équation différentielle et avec le résultat théorique.

Ici la réponse impulsionnelle est définie à partir d'une équation différentielle.

$$h(t) = \tilde{h}(t-1) \text{ et } \frac{d}{dt}\tilde{h}(t) + \tilde{h}(t) = \delta(t) \tag{3.24}$$

Du coup la réponse impulsionnelle se calcule ainsi

```
t=np.linspace(-2,5,10**3)
h_tilde=seb.sol_eq_diff((1,1),t)
h=seb.retarder(t,h_tilde,1)
```

On peut alors vérifier que cette estimation coïncide avec l'équation (3.8). C'est ce que montre la figure 3.1. L'implémentation est en section 3.3.1.

3.2.2 Estimation de l'autocorrélation

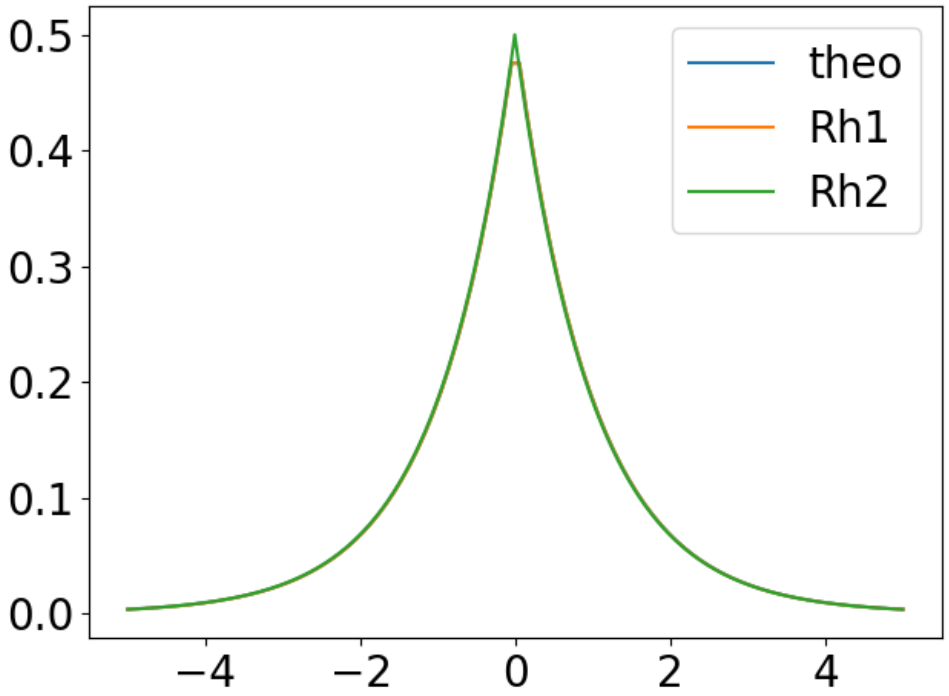

Figure 3.2: Autocorrélation de la réponse impulsionnelle du filtre calculé et estimé par deux techniques

La première technique consiste à passer par la densité spectrale.
On utilise une échelle en fréquence

```
f    = np.linspace(-50,50,10**4)
```

On récupère la réponse fréquentielle avec laquelle on calcule la densité spectrale.

```
H   = np.exp(-1j*2*np.pi*f)/(1+1j*2*np.pi*f)
G_h = np.abs(H)**2
```

Et on en déduit l'autocorrélation.

```
th  = np.linspace(-5,5,10**2)
R_h = seb.TFI(f,G_h,th)
```

La deuxième technique utilise la réponse impulsionnelle déjà calculée et évaluée ici le long d'une échelle en temps.

```
t   = np.linspace(-50,50,10**4)
h   = np.exp(-(t-1))*(t>=1)
```

On se donne ici une deuxième échelle de temps moins précise pour le graphique et la fonction `correlation` de `seb.py` permet d'en déduire l'autocorrélation.

```
th2 = np.arange(-5,5,t[1]-t[0])
Rh2 = seb.correlation(t,h,t,h,th2)
```

L'implémentation est en section 3.3.3.

3.2.3 Tirage aléatoire d'une valeur particulière de $\overset{r}{Y}(t_0)$

Le tirage aléatoire se fait avec un tirage d'un bruit blanc à une fréquence d'échantillonnage f_e sur un intervalle de temps $[-T/2, T/2]$ avec T tendant vers l'infini.

```
import numpy.random as nr
fe,t0,T = 10**3, 1, 100
t=np.arange(-T,T,1/fe)
B=nr.randint(0,2,len(t))*2-1
```

En utilisant une implémentation `h1` de la définition de la réponse impulsionnelle notée ici $h_1(t)$, on en déduit un tirage aléatoire de $\overset{r}{Y}(t_1)$.

```
Yt0=seb.convolution(t,B,t,h1(t),t0)*np.sqrt(fe)
```

3.2.4 Représentation de la densité de probabilité du signal en sortie en $t = t_0$ pour une fréquence d'échantillonnage f_e

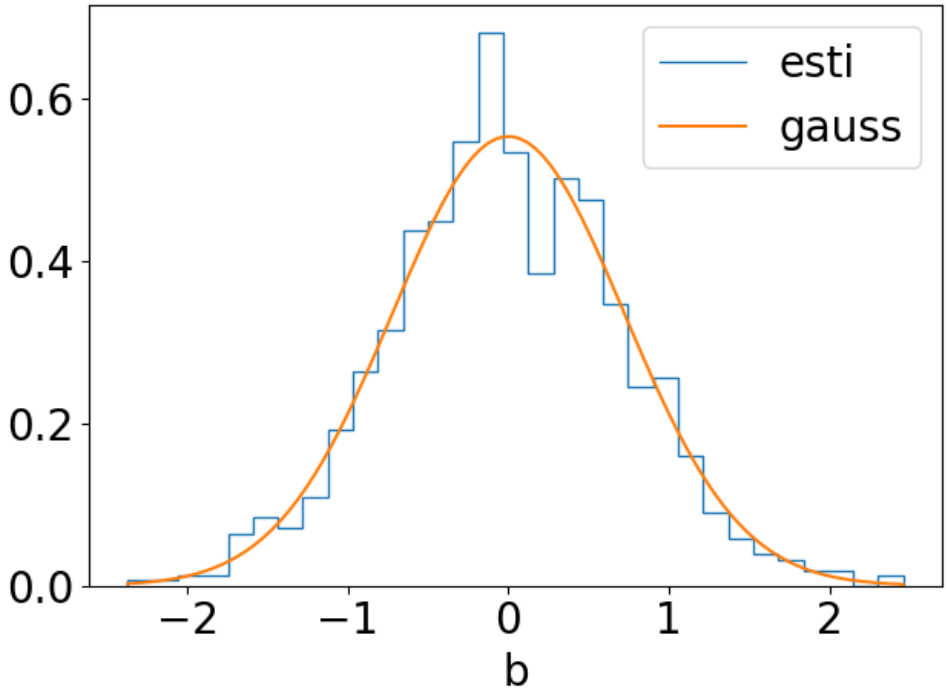

Figure 3.3: Distribution de probabilité estimée et fonction gaussienne estimée

La figure 3.3 représente la distribution de probabilité de $\overset{r}{Y}(t_0)$ estimée avec 1000 tirages aléatoires. L'implémentation est faite en section 3.3.2.

Pour représenter une distribution de probabilité, on stocke dans un tableau `tab` des réalisations de $\overset{r}{Y}(t_0)$. Ce tableau est d'abord initialisé

```
K = 1000; tab=np.zeros(K)
```

On met dans chaque case du tableau la quantité `Yt0` calculée précédemment, et on place le tout dans une boucle `for` pour que cette tâche soit réalisée K fois.

```
for k in range(K):
  B=nr.normal(0,1,len(t))
  tab[k]=seb.convolution(t,B,t,h1(t),t0)*np.sqrt(fe)
```

Pour estimer la densité de probabilité, on utilise la fonction `numpy.histogram` qui donne les estimations des densités de probabilité pour des intervalles contigus. Le nombre de ces intervalles doit être spécifié et je propose d'utiliser la valeur arrondi obtenu avec la racine carré du nombre de tirages aléatoires `int(np.sqrt(K))`. Cette information est transmise avec l'option `bins` avec cette valeur. Je propose d'utiliser la normalisation qui est ajustée de façon qu'approximativement l'intégrale de la densité de probabilité vale 1. Ceci se fait avec l'option `density` avec la valeur `True`. La fonction `numpy.histogram` renvoie d'abord la densité de probabilité et après les extrémités des différents intervalles.

```
proba,b_val = np.histogram(tab,bins=int(np.sqrt(K)),density=True)
```

Pour afficher la densité de probabilité, le plus simple est d'utiliser `ax.stairs` qui s'utilise comme `ax.plot` sauf que les deux premiers arguments sont ceux que donnent `np.histogram`, (l'inverse de `ax.plot`). La fonction `ax.stairs` requière que le deuxième argument ait une composante en plus que le premier, ce que justement `np.histogram` fait.

```
ax.stairs(proba,b_val)
```

On peut ensuite comparer ces valeurs avec la distribution de probabilité gaussienne obtenue en utilisant la moyenne et l'écart-type estimés à partir des valeurs tirées.

```
mu,sigma = np.mean(tab), np.std(tab)
print(f"{mu=:.3g} {sigma=:.3g}")
```

Dans l'expérimentation proposée, on trouve $\mu = -0.009$ et $\sigma = 0.722$.

La densité de probabilité d'une gaussienne est alors

$$f_B(b) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{b^2}{2\sigma^2}} \tag{3.25}$$

Il reste à calculer l'échelle de valeurs de B à partir de `b_val` qui sont les extrémités d'intervalles contigus de mêmes tailles.

```
b=seb.milieux(b_val)
```

On peut ensuite évaluer la densité de probabilité gaussienne ajustée avec μ et σ.

```
fb=seb.gaussian(b,mu,sigma)
```

On peut alors compléter l'affichage

```
ax.plot(b,fb,label='gauss')
```

3.2.5 Détermination théorique de la moyenne statistique et de la variance de de $\overset{r}{Y}(t)$ à partir de celles de $\overset{r}{B}(t)$

Le gain statique $\widehat{H}(0) = 1$ d'après l'équation (3.12). Aussi la moyenne statistique de la sortie est la même que la moyenne statistique de l'entrée qui vaut 0.

$$\mathrm{E}\left[\overset{r}{Y}(t)\right] = \widehat{H}(0)\mathrm{E}\left[\overset{r}{X}(t)\right] = 0 \tag{3.26}$$

Ici $\int_{-\infty}^{+\infty} \left|\widehat{H}(f)\right|^2 df = \mathrm{R}_h(0) = 0.5$ aussi $\mathrm{Var}[\overset{r}{Y}(t)] = 0.5 T_e \mathrm{Var}[\overset{r}{X}(t)] = 0.5$ et $\sigma_Y = \frac{1}{\sqrt{2}}$.

On observe que la valeur théorique coïncide approximativement avec l'expérimentation pour la figure 3.3 : $\mu = -0.009 \approx 0$ et $\sigma = 0.722 \approx 0.707 = \frac{\sqrt{2}}{2}$.

3.2.6 Différentes façons de simuler l'autocorrélation

On cherche à générer la courbe de l'autocorrélation

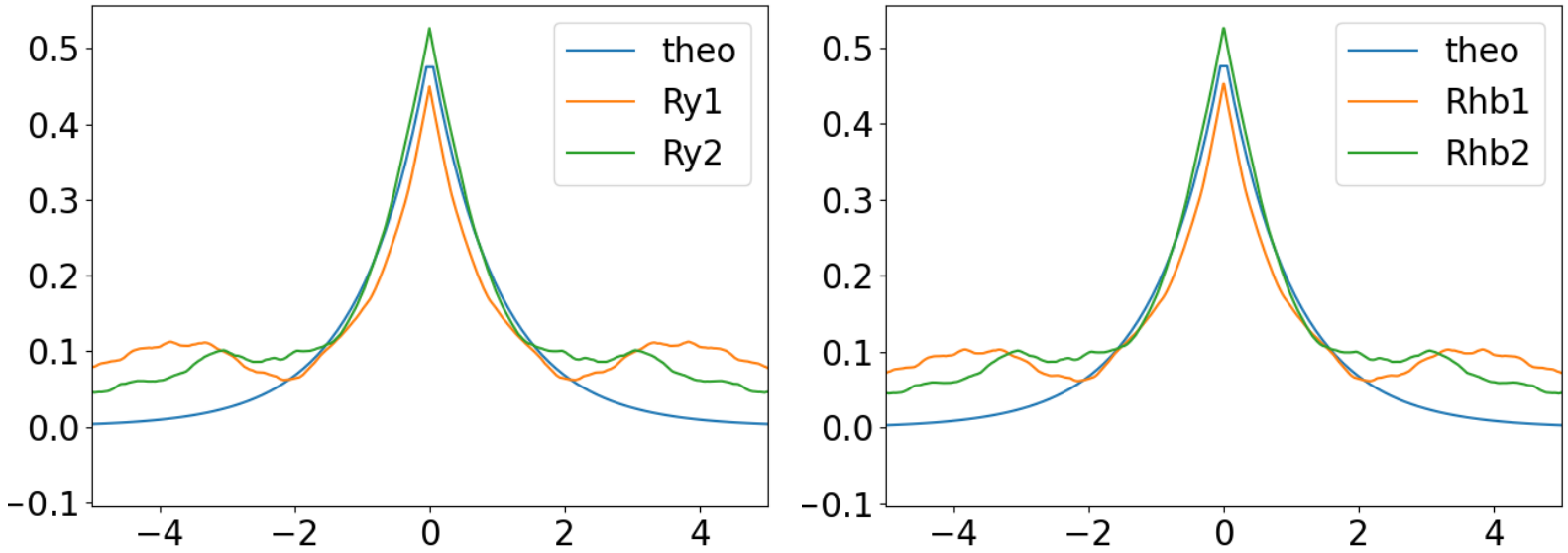

Figure 3.4: Autocorrélation de la sortie du filtre estimée à partir de deux sorties à gauche et à partir de l'autocorrélation de deux réalisations du bruit blanc à droite. Ces courbes sont comparées à la figure 3.2

Dans cette sous-partie, on cherche à approcher la figure 3.2 en utilisant tout les valeurs du signal, ce qui est une façon d'utiliser la propriété de l'ergodicité.

Je considère une échelle de temps `t` avec sa durée `T` et sa fréquence d'échantillonnage `fe`.

```
t    = np.linspace(-50,50,10**4)
fe   = 1/(t[1]-t[0])
```

```
T    = t[-1]-t[0]
```

Je calcule dessus deux réalisations du même bruit blanc.

```
B1   = nr.randint(0,2,len(t))*2-1
B2   = nr.randint(0,2,len(t))*2-1
```

Je calcule la sortie du filtre `Y1` et `Y2` pour chaque réalisation. Le produit par `np.sqrt(fe)` est la conséquence du fait que j'ai remplacé le bruit blanc théorique par un bruit blanc échantillonné à la fréquence `fe`. Cette multiplication est nécessaire pour que l'autocorrélation de $\overset{r}{Y}$ soit correcte.

```
h    = np.exp(-(t-1))*(t>=1)
Y1   = seb.convolution(t,B1,t,h,t)*np.sqrt(fe)
Y2   = seb.convolution(t,B2,t,h,t)*np.sqrt(fe)
```

Je calcule l'autocorrélation des deux réalisations. Dans ces calculs je divise par la durée du signal parce ce qui est pertinent dans le bruit blanc utilisé ce sont les quantités stochastiques moyennées sur la durée.

```
Ry1  = seb.correlation(t,Y1,t,Y1,t)/T
Ry2  = seb.correlation(t,Y2,t,Y2,t)/T
```

L'implémentation est en section 3.3.4.

On cherche à générer l'autocorrélation à partir de deux points `t0` et `t1`

Dans cette façon de procéder, on n'utilise pas vraiment l'ergodicité, on simule un grand nombre d'expériences et on moyenne les résultats concernant `t0` et `t1`. Numériquement c'est une expérience beaucoup plus lourde. Cette experience est implémentée dans 3.3.6

Je considère deux instants ici $t_0 = 1$ et $t_1 = 2$.

```
t0,t1 = 1,2
```

Je créé une échelle de temps sur laquelle j'évalue la réponse impulsionnelle. Je calcule aussi la fréquence d'échantillonnage et la durée du signal.

```
t    = seb.arange(-25,25,1e-3)
h    = np.exp(-(t-1))*(t>=1)
fe   = 1/(t[1]-t[0])
T    = t[-1]-t[0]
```

L'expérience à réaliser de nombreuses fois consiste à générer un bruit blanc puis à calculer la sortie du filtre correspondante. Je la répète `K=1000` fois. À chaque calcul, je retiens dans un tableau à deux colonnes notés `Y`, la sortie du filtre à l'instant `t0` et à l'instant `t1`. Les deux dernières lignes sont juste là pour indiquer la progression du calcul qui prend plusieurs heures.

```
K    = 10**3
Y    = np.zeros((K,2))
```

```
for k in range(K):
  B  = nr.randint(0,2,len(t))*2-1
  Y[k,0] = seb.convolution(t,B,t,h,t0)*np.sqrt(fe)
  Y[k,1] = seb.convolution(t,B,t,h,t1)*np.sqrt(fe)
  if 0==k%10:
    print(f"{k=}")
```

À partir des valeurs calculées, j'estime un coefficient de corrélation

$$\rho = \frac{\mathrm{E}\left[\left(\overset{r}{Y}(t_0) - \mathrm{E}[\overset{r}{Y}(t_0)]\right)\left(\overset{r}{Y}(t_1) - \mathrm{E}[\overset{r}{Y}(t_1)]\right)\right]}{\sqrt{\mathrm{Var}[\overset{r}{Y}(t_0)]}\sqrt{\mathrm{Var}[\overset{r}{Y}(t_1)]}} \tag{3.27}$$

L'estimation de ρ est

$$\begin{aligned}
\mu_0 &= \tfrac{1}{K}\Sigma_k\, Y_k(t_0) \\
\mu_1 &= \tfrac{1}{K}\Sigma_k\, Y_k(t_1) \\
\sigma_0 &= \sqrt{\tfrac{1}{K}\Sigma_k (Y_k(t_0) - \mu_0)^2} \\
\sigma_1 &= \sqrt{\tfrac{1}{K}\Sigma_k (Y_k(t_1) - \mu_1)^2} \\
\rho &= \tfrac{1}{K}\Sigma_k (Y_k(t_0) - \mu_0)(Y_k(t_1) - \mu_1)\sigma_0\sigma_1
\end{aligned} \tag{3.28}$$

J'implémente ceci avec

```
rho  = np.mean((Y[:,0]-np.mean(Y[:,0]))*(Y[:,1]-np.mean(Y[:,1])))/np.std
```

Théoriquement ce coefficient de corrélation est

$$\rho = \frac{R_h(t1 - t_0)}{\sqrt{R_h(0)}\sqrt{R_h(0)}} = \frac{0.5e^{-|t_1 - t_0|}}{0.5} = e^{-|t_1 - t_0|} \tag{3.29}$$

```
rho_th = np.exp(-np.abs(t1-t0))
```

Le résultat expérimental obtenu est $\rho = 0.34$, alors que la valeur théorique est $e^{-1} = 0.37$.

3.2.7 Différentes façons de simuler la densité spectrale

On cherche à générer la courbe de la densité spectrale

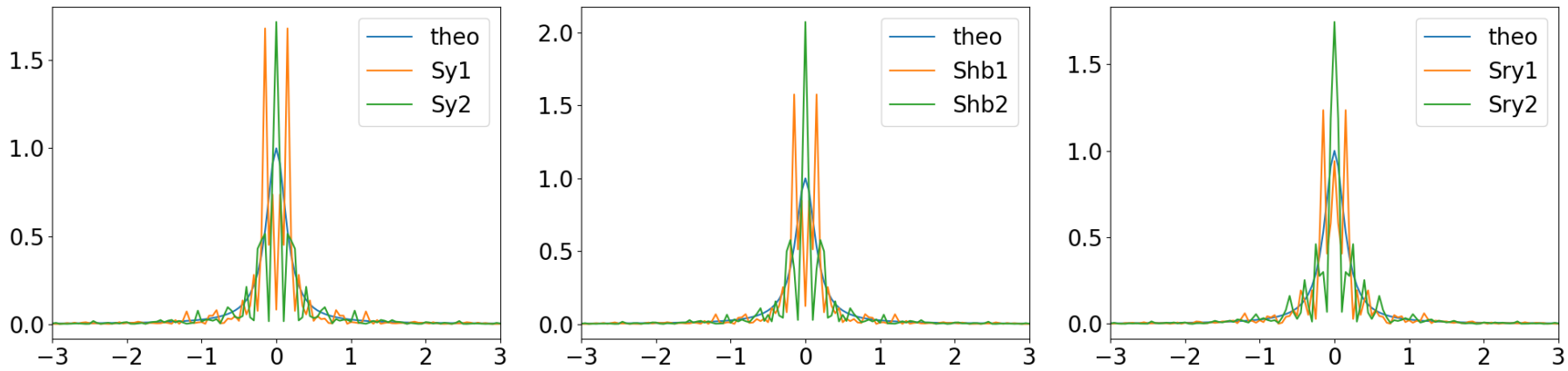

Figure 3.5: Densité spectrale de la sortie du filtre estimée à partir de deux sorties à gauche et à partir de la densité spectrale de deux réalisations du bruit blanc à droite. Ces courbes sont comparées à la figure 3.2

Dans cette sous-partie, on cherche à approcher la densité spectrale théorique $\left|\widehat{H}(f)\right|^2 = \frac{1}{1+4\pi^2 f^2}$ en utilisant tout les valeurs du signal, ce qui est une façon d'utiliser la propriété de l'ergodicité.

Je considère une échelle de fréquences `f` qui va servir pour le graphique à la fin et pour cette échelle je calcule la valeur théorique de la densité spectrale à partir de l'équation (3.13).

```
f    = seb.linspace(-5,5,2*10**2)
Sh   = 1/(1+4*np.pi**2*f**2)
```

Je génère deux réalisations du même bruit blanc à partir d'une échelle de temps.

```
t    = np.linspace(-50,50,10**4)
B1   = nr.randint(0,2,len(t))*2-1
B2   = nr.randint(0,2,len(t))*2-1
```

Je simule la sortie du filtre pour chaque réalisation à partir de l'équation de la réponse impulsionnelle calculé dans l'équation (3.8). J'ai aussi besoin de la fréquence d'échantillonnage `fe`.

```
h    = np.exp(-(t-1))*(t>=1)
fe   = 1/(t[1]-t[0])
Y1   = seb.convolution(t,B1,t,h,t)*np.sqrt(fe)
Y2   = seb.convolution(t,B2,t,h,t)*np.sqrt(fe)
```

Je calcule la densité spectrale associée à chacune des réalisations des sorties du filtre. Pour cela j'ai besoin de normaliser par la durée du signal car le bruit blanc créé a des propriétés stochastiques qui se calculent avec une moyenne. Cette durée du signal `T` se calcule avec l'échelle de temps, `t[-1]` est la valeur de la dernière composante de `t`.

```
T    = t[-1]-t[0]
Sy1  = np.abs(seb.TF(t,Y1,f))**2/T
Sy2  = np.abs(seb.TF(t,Y2,f))**2/T
```

Ceci permet d'obtenir la gauche de la figure 3.5, `Sy1` et `Sy2` sont des courbes qui varient fortement autour la courbe lisse qui est la courbe théorique de la densité spectrale $S_y(f)$.

Une deuxième façon d'obtenir la densité spectrale est de calculée celle du bruit blanc notée `Sb1` et `Sb2` pour en déduire ensuit deux estimations de $S_y(f)$.

```
SB1  = np.abs(seb.TF(t,B1,f))**2/T*fe
SB2  = np.abs(seb.TF(t,B2,f))**2/T*fe
```

J'obtiens les deux estimations de $S_y(f)$ en multipliant par $S_h(f) = |\widehat{H}(f)|^2$

```
Shb1 = SB1*Sh
Shb2 = SB2*Sh
```

Ceci me donne le graphe du milieu de la figure 3.5.

Une troisième façon d'estimer $S_y(f)$ est d'estimer l'autocorrélation avec $\overset{r}{Y}(t)$ et d'appliquer la transformée de Fourier. Je normalise par `T` parce que l'autocorrélation est calculée sur

un signal qui est à la base induit par un bruit blanc de durée théoriquement infinie et dont les caractéristiques stochastiques théoriques sont constantes.

```
Ry1  = seb.correlation(t,Y1,t,Y1,t)/T
Ry2  = seb.correlation(t,Y2,t,Y2,t)/T
```

J'obtiens finalement deux estimations des densités spectrales avec une transformée de Fourier.

```
Sry1 = seb.TF(t,Ry1,f)
Sry2 = seb.TF(t,Ry2,f)
```

Ceci me donne le graphe de droite de la figure 3.5.

L'implémentation est en section 3.3.5.

```
Sexp=0.0923151198926404 2 Sth=0.0919996683503752 4
```

3.2.8 Simulation dans le cas où le bruit placé en entrée du filtre est obtenu avec un motif

Je considère ici un motif $M(t) = \mathbb{T}(t)$ et une suite de variables aléatoire $\overset{r}{B}_n$ qui ont une probabilité 0.5 de valoir 1 ou -1. Je considère une fréquence d'échantillonnage f_e et une période d'échantillonnage $T_e = \frac{1}{f_e}$. Le bruit considéré est

$$\overset{r}{B}'(t) = \sum_{n=-\infty}^{+\infty} \overset{r}{B}_n M\left(\frac{t - nT_e}{T_e}\right) \tag{3.30}$$

3.3 Codes permettant de réaliser le TP3

3.3.1 Figure 3.1

```
plt,np = seb.debut()
plt.close('all')
t=np.linspace(-2,5,10**3)
h_tilde=seb.sol_eq_diff((1,1),t)
h=seb.retarder(t,h_tilde,1)
h_th=np.exp(t-1)*(t>=1)
fig,ax=plt.subplots()
ax.plot(t,h,label='esti')
ax.plot(t,h,label='theo')
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_TP3_fig1a.png')
fig.show()
```

3.3.2 Figure 3.3

```
plt,np = seb.debut()
plt.close('all')
```

```
fe,t0,T = 10**3,5,100
import numpy.random as nr
def h1(t):
  return np.exp(-(t-1))*(t>=1)

def c_B(t0,fe):
  t,th=np.arange(-T/2,T/2,1/fe),np.arange(1,10,1/fe)
  B=nr.randint(0,2,len(t))*2-1
  return seb.convolution(t,B,th,h1(th),t0)*np.sqrt(fe)

K = 1000; tab=np.zeros(K)
for k in range(K):
  tab[k]=c_B(t0,fe)
  print(f"{k=:.3g}")

proba,b_val = np.histogram(tab,bins=int(np.sqrt(K)),density=True)
b=seb.milieux(b_val)
fig,ax=plt.subplots()
ax.stairs(proba,b_val,label='esti')
mu,sigma = np.mean(tab),np.std(tab)
print(f"{mu=:.3g} {sigma=:.3g}")
b_l=np.linspace(min(b_val),max(b_val),10**2)
```

```
gauss=1/np.sqrt(2*np.pi)/sigma*np.exp(-b_l**2/2/sigma**2)
ax.plot(b_l,gauss,label='gauss')
ax.set_xlabel('b')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_TP3_fig2a.png')
fig.show()
```

3.3.3 Figure 3.2

```
plt,np = seb.debut()
plt.close('all')
f   = np.linspace(-50,50,10**4)
H   = np.exp(-1j*2*np.pi*f)/(1+1j*2*np.pi*f)
G_h = np.abs(H)**2
th  = np.linspace(-5,5,10**2)
Rh1 = seb.TFI(f,G_h,th)
Rh_th = 0.5*np.exp(-np.abs(th))
t   = np.linspace(-50,50,10**4)
h   = np.exp(-(t-1))*(t>=1)
th2 = np.arange(-5,5,t[1]-t[0])
Rh2 = seb.correlation(t,h,t,h,th2)
```

```
fig,ax = plt.subplots()
ax.plot(th,Rh_th,label='theo')
ax.plot(th,Rh1,label='Rh1')
ax.plot(th2,Rh2,label='Rh2')
plt.tight_layout()
ax.legend()
fig.savefig('./figures/fig_TP3_fig3a.png')
fig.show()
```

3.3.4 Figure 3.4

```
plt,np = seb.debut()
import numpy.random as nr
plt.close('all')

th   = np.linspace(-5,5,10**2)
Rh_th= 0.5*np.exp(-np.abs(th))

t    = np.linspace(-50,50,10**4)
h    = np.exp(-(t-1))*(t>=1)
fe   = 1/(t[1]-t[0])
T    = t[-1]-t[0]
```

```
Rh   = 0.5*np.exp(-np.abs(t))
B1   = nr.randint(0,2,len(t))*2-1
Rb1  = seb.correlation(t,B1,t,B1,t)/T
Rhb1 = seb.convolution(t,Rb1,t,Rh,t)*fe
Y1   = seb.convolution(t,B1,t,h,t)*np.sqrt(fe)
Ry1  = seb.correlation(t,Y1,t,Y1,t)/T
B2   = nr.randint(0,2,len(t))*2-1
Rb2  = seb.correlation(t,B2,t,B2,t)/T
Rhb2 = seb.convolution(t,Rb2,t,Rh,t)*fe
Y2   = seb.convolution(t,B2,t,h,t)*np.sqrt(fe)
Ry2  = seb.correlation(t,Y2,t,Y2,t)/T
```

3.3.5 Figure 3.5

```
plt,np = seb.debut()
import numpy.random as nr
plt.close('all')

f    = seb.linspace(-5,5,2*10**2)
Sh   = 1/(1+4*np.pi**2*f**2)
```

```
t    = np.linspace(-50,50,10**4)
h    = np.exp(-(t-1))*(t>=1)
fe   = 1/(t[1]-t[0])
T    = t[-1]-t[0]

B1   = nr.randint(0,2,len(t))*2-1
Y1   = seb.convolution(t,B1,t,h,t)*np.sqrt(fe)
Sy1  = np.abs(seb.TF(t,Y1,f))**2/T
B2   = nr.randint(0,2,len(t))*2-1
Y2   = seb.convolution(t,B2,t,h,t)*np.sqrt(fe)
Sy2  = np.abs(seb.TF(t,Y2,f))**2/T

SB1  = np.abs(seb.TF(t,B1,f))**2/T*fe
Shb1 = SB1*Sh
SB2  = np.abs(seb.TF(t,B2,f))**2/T*fe
Shb2 = SB2*Sh

Ry1  = seb.correlation(t,Y1,t,Y1,t)/T
Sry1 = seb.TF(t,Ry1,f)
Ry2  = seb.correlation(t,Y2,t,Y2,t)/T
Sry2 = seb.TF(t,Ry2,f)
```

```
fig,ax = plt.subplots()
ax.plot(f,Sh,label='theo')
ax.plot(f,Sy1,label='Sy1')
ax.plot(f,Sy2,label='Sy2')
ax.set_xlim(-3,3)
plt.tight_layout()
ax.legend()
fig.savefig('./figures/fig_TP3_fig5a.png')
fig.show()
```

```
fig,ax = plt.subplots()
ax.plot(f,Sh,label='theo')
ax.plot(f,Shb1,label='Shb1')
ax.plot(f,Shb2,label='Shb2')
ax.set_xlim(-3,3)
plt.tight_layout()
ax.legend()
fig.savefig('./figures/fig_TP3_fig5b.png')
fig.show()
```

```
fig,ax = plt.subplots()
ax.plot(f,Sh,label='theo')
ax.plot(f,Sry1,label='Sry1')
ax.plot(f,Sry2,label='Sry2')
ax.set_xlim(-3,3)
plt.tight_layout()
ax.legend()
fig.savefig('./figures/fig_TP3_fig5c.png')
fig.show()
```

3.3.6 Simulation de la section 3.2.6

```
t0,t1 = 1,2
t    = seb.arange(-25,25,1e-3)
h    = np.exp(-(t-1))*(t>=1)
fe   = 1/(t[1]-t[0])
T    = t[-1]-t[0]
K    = 10**3
Y    = np.zeros((K,2))
for k in range(K):
  B  = nr.randint(0,2,len(t))*2-1
  Y[k,0] = seb.convolution(t,B,t,h,t0)*np.sqrt(fe)
```

```
  Y[k,1] = seb.convolution(t,B,t,h,t1)*np.sqrt(fe)
  if 0==k%10:
    print(f"{k=}")

def R_th(t):
  return 0.5*np.exp(-np.abs(t))

rho  = np.mean((Y[:,0]-np.mean(Y[:,0]))*(Y[:,1]-np.mean(Y[:,1])))/np.std
rho_th = R_th(t0-t1)/np.sqrt(R_th(0))/np.sqrt(R_th(0))
print(rho,rho_th)
```

Appendix A

Supplément

A.1 Outils

Python :

Lignes à mettre en début de séance

Les deux premières servent à faire en sorte que les programmes dans `rep_prg` soient utilisables lorsqu'on est dans le répertoire `rep_tra`. `rep_tra` et `rep_prg` sont deux noms de répertoires et de leurs chemins absolues. On peut aussi utiliser des chemins relatifs mais dans ce cas il faut veiller à l'endroit où l'on se trouve.

```
import sys
sys.path.append('rep_prg')
```

```
import os
os.chdir('rep_tra')
import seb
plt,np=seb.debut()
```

Cas d'un fichier `Python` avec extension `.py.txt`

`seb.debut`

La fonction **`debut`** renvoie un tuple **`(plt,np)`**.

- **`plt`** sert pour faire des graphes et ses paramètres sont modifiés par **`debut`** de façon à rendre les graphes plus visibles.
- **`np`** correspond à **`numpy`**, c'est utile pour faire des calculs.

A.1.1 Commandes générales

- **`help`** suivi de parenthèses avec le nom de la fonction. Le contenu affiché est celui entre trois guillemets dans une fonction.
- **`assert`** provoque une erreur si ce qui suit n'est pas vrai.
- **`len`** suivi de parenthèses et le nom d'un vecteur ligne ou colonne, cela donne sa longueur.

- `1j` est le complexe imaginaire j.
- **`round`** qui fournit [1] un entier à partir d'un nombre réel.
- **`def`** et **`return`** pour définir une fonction.
- `#` pour mettre une ligne en commentaire.
- **`type`** pour connaître le type d'une valeur ou d'une variable.
- **`min`** et **`max`**

Contenu susceptible d'être utilisé dans `np` pour `numpy`

- **`linspace`** permet de définir un ensemble de valeur régulièrement réparties en précisant la valeur initiale, la valeur finale et le nombre de ces valeurs. La dernière valeur est atteinte. Cette fonction n'entraîne pas en général que les valeurs soient des multiples d'un pas donné ce qui est contradictoire avec ce qui est contradictoire avec ce qu'on attend d'une échelle en temps ou en fréquence en traitement du signal. **`seb.linspace`** assure cette propriété.
- **`arange`** permet de définir un ensemble de valeur régulièrement réparties en précisant la valeur initiale, la valeur finale et l'espacement entre ces valeurs. La dernière valeur

[1] La fonction **`round`** de **`numpy`** fournit un entier de type **`float`**.

n'est jamais atteinte. Comme `linspace`, cette fonction n'est pas non plus conforme avec ce qu'on attend d'une échelle en temps ou en fréquence en traitement du signal. `seb.arange` assure cette propriété.

- `sqrt` pour racine carré
- `exp` pour exponentielle
- `array` suivi d'une liste entre crochets de valeurs espacées de virgules, pour définir un vecteur de
  type `numpy.ndarray`. On peut l'utiliser pour définir une matrice en utilisant deux séries de crochets.
- `real` suivi d'un complexe entre parenthèses pour prendre la partie réelle.
- `abs` suivi d'un complexe ou d'un réel entre parenthèses pour prendre le module ou la valeur absolue.
- `sinc` est la fonction sinus cardinal définie par $\frac{\sin(\pi t)}{\pi t}$
- `concatenate`
- `zeros`, `ones`, `zeros_like` et `ones_like`. Pour les deux premières le premier paramètre est un entier ou un `tuple` indiquant soit la taille du vecteur soit le nombre de lignes et de colonnes de la matrice à définir. Les deux dernières permettent de

créer un vecteur ou une matrice ayant les mêmes dimension qu'un certain objet qu'on transmet en paramètre. Les valeurs de l'objet créés sont nulles si on utilise `zeros` ou `zeros_like` et 1 si on utilise `ones` ou `ones_like`.

Contenu susceptible d'être utilisé dans `plt` récupéré dans `debut`

- `subplots`
- `plot`
- `set_xlabel` et `set_ylabel`
- `set_legend`
- `tight_layout`
- `savefig`
- `show`

Contenu susceptible d'être utilisé dans `seb`

- Les fonctions pour définir un signal

- `fonction_H` est la fonction échelon $[\![t \geq 0]\!](t)$.
- `fonction_P` est la fonction porte sur $[-0.5, 0.5]$.
- `fonction_T` est la fonction triangle sur $[-1, 1]$.
- `fonction_C` est la demi-fonction triangle croissante sur $[-0.5, 0.5]$.
- `fonction_D` est la demi-fonction triangle décroissante sur $[-0.5, 0.5]$.
- `gaussian(x,mu,sigma)` renvoie la fonction associe a densité de probabilité de la Gaussienne de moyenne `mu` et d'écart-type `sigma`.

- Les fonctions pour calculer une transformée de Fourier
  - `TF` et `TFI` pour transformée de Fourier et transformée de Fourier inverse
  - `TFTD(t,x,f)` calcule la TFTD du signal temps discret défini par `t,x` en les fréquences `f`.
  - `TFTDI(f,X,t)` calcule la TFTD inverse du spectre complexe `X` définis pour les fréquences `f`. Le résultat est un signal temps discret pour les instants contenus dans `t`.
  - `TFD(t,s,T,bool)` calcule la TFD, `T` indique soit la période soit l'intervalle utilise pour décrire le signal periodique `t,s` défini l échelle de temps et le signal. `bool` vaut `True` si on veut une représentation centrée et `False` si on n'en veut pas . `f` est l'échelle de fréquence retournée. `S` est l'ensemble des coefficients associés `f,S=seb.TFD(t,s,T,bool)`

- `coef_serie_Fourier(t,x,T,k)` calcule les coefficients de la série de Fourier X_k `T` est soit la periode du signal soit un tuple indiquant un intervalle sur lequel est defini $x(t)$. Si `T` est une valeur alors l'intervalle considere est $[0, T]$ `k` est la liste des indices des fréquences calculées. Le programme retourne un tuple avec d'abord les fréquences et d'autre part les coefficients associés.

- Les fonctions relatives aux échelles

  - `synchroniser(t)` change l'échelle de temps de façon que le vecteur soit un multiple de la periode d'échantillonnage.
  - `linspace(start,stop,num,dtype)` et `arange(start,stop,step,dtype)` similaires à celles définies dans `numpy` mais qui assurent que les échelles sont bien constituées de multiples d'un certain pas et ainsi conforme au traitement du signal.

- Les fonctions pour transformer un signal

  - `retarder(t,x,tau)` retarde le signal $x(t)$ défini par `t` et `x` de `tau` lorsque `tau` est positif et avance de `-tau` si `tau` est négatif.
  - `periodiser_ech_t(t,T)` produit un vecteur de meme taille que `t` mais dont les valeurs sont entre `0` et `T` de facon a définir un signal periodique de période T. Si `T` est un intervalle alors c'est le motif entre `T[0]` et `T[1]` qui est répété.

- Les fonctions associées au produit de convolution

- `convolution(tx,x,th,h,ty)` Le programme fournit le produit de convolution de x par h aux instants demandes par ty tx, th et ty sont les échelles de temps de x h et y x et tx doivent être de même taille th et h doivent être de même taille tx doit contenir au moins deux composantes x et h sont supposés d'énergie finie.
- `correlation(tx,x,ty,y,tz)` calcule l'intercorrelation entre tx,x et ty,y en tz

- Les fonctions associées aux équations différentielles et au fait de dériver et d'intégrer.
  - `sol_eq_diff(coef,t)` `coef` sont les coefficients devant les termes de l'équation différentielle définis comme un `tuple t` est l'ensemble des instants dont a cherche a calculer `y(t) t` est un vecteur avec des points régulièrement espacés
  - `deriver(t,x)` dérive le signal défini par `t,x` en retournant la dérivée en tous les instants de `t`.
  - `integrer(t,x)` intègre le signal défini par `t,x` en retournant l'intégrale $\int_{-\infty}^{t} x(t)\,dt$ en tous les instants de `t`.

- Des fonctions pour trouver une valeur particulière dans un vecteur.
  - `find_nearest(array, value)` retourne la valeur de `array` la plus proche de `value`.

  - `where_nearest(array, value)` retourne l'indice dans `array` correspondant à la valeur la plus proche de value.

- `erreur_quad(fun,intervalle)` est une fonction pour calculer l'erreur quadratique. Cette fonction utilise une variable aléatoire sur un support uniforme pour calculer une erreur quadratique. `intervalle` doit indique avec un tuple contenant deux valeurs. `fun` est une fonction à transmettre qui estime l'erreur pour une valeur particulière.

- Des fonctions pour aider à gérer les types en Python. Python distingue une liste contenant une unique valeur de la valeur elle-même et d'un tableau de type `numpy` contenant cette valeur.

  - `val(x)` vérifie si x est un `numpy array` contenant une seule valeur, si c'est une seule valeur ou autre chose. Si c'est autre chose, cette fonction déclenche une erreur. Sinon elle renvoie cette unique valeur.
  - `vect(x)` vérifie si x est un `numpy array` contenant une ou plusieurs valeurs, une seule valeur ou autre chose Si c'est autre chose, elle prend le premier élément et vérifie que celui-ci est bien un `numpy array` et elle renvoie ce vecteur. Sinon une erreur est déclenchée.
  - `milieux(b_val)` renvoie un vecteur ayant une composante en moins que `b_val` et correspondant aux milieux des termes consécutifs de `b_val`. Cette fonction suppose que `b_val` est régulièrement réparti.

- Les fonctions pour enregistrer des données et pour les récupérer à nouveau.
  - `save(nom_fichier,list_nom_var,list_var)` sauvegarde sous format binaire la liste des variables indiquées dans list_var le fichier s'appelle nom_fichier les noms des variables doivent etre mises avec des apostrophes autour.
  - `load(nom_fichier)` lit le fichier binaire et renvoie un dictionnaire dont les clés sont les noms des variables enregistrés.

Contenu susceptible d'être utilisé dans `sympy`

- `Symbol`
- `solve`
- `simplify`
- `matrices.Matrix`

Appendix B

Codes pour simuler les différentes figures

B.1 Installation de Python sous Windows

Je présente ici une façon d'installer `Python`. Celle que j'utilise à titre personnel. A priori cela fonctionne pour Windows 10 et 11 en octobre 2025.

- Il me semble que d'autres logiciels qui présentent une interface commune pour écrire un programme et exécuter un code Python gère aussi l'installation de Python.
- Une installation de `Python` doit pour ces TP inclure l'installation de `numpy`, `matplotlib` et `scipy`.

1. Paramétrer Windows de façon à ce qu'il affiche les extensions de noms de fichiers. L'objectif est qu'un fichier `seb.py` n'apparaisse pas à l'écran comme `seb` et que le fichier indiqué comme `seb.py` ne corresponde pas en réalité à `seb.py.txt`.

   - La façon de modifier ce paramétrage dépend fortement de la version de Windows. Une façon de faire est :
     - Lancer l'explorateur (touche `Win+E`)
     - Cliquer sur `...`
     - Cliquer sur `option`
     - Décocher la ligne `Masquer les extensions des fichiers dont le type est connu`

2. Vérifier la présence de Python en ouvrant une fenêtre `dos`. Cette fenêtre s'ouvre par exemple en appuyant sur la touche `Windows` et la touche `R` puis en écrivant `cmd`. Dans cette fenêtre écrire `python` ou `python3`.

   - Il est possible que `Python` soit déjà installé mais pas accessible dans une fenêtre `dos` quelconque. Pour trouver où il est installé, il semble qu'il ne suffise pas d'ouvrir un explorateur de fichier et d'écrire dans recherche `python`. Pour la recherche manuelle, je propose de regarder les répertoires suivants :
     - `C:\\Program Files\\`
     - `C:\\Program Files(x86)\\`

  - `C:\\Users\\` nom utilisateur `\\AppData\\`

  Les deux premiers répertoires correspondent à une installation commune à tous les utilisateurs de l'ordinateur, tandis que la dernière correspond à une installation spécifique à un utilisateur.

- Si Python est effectivement installé quelque part, pour le rendre accessible, il suffit d'ajouter son chemin dans une variable d'environnement de Windows. Je propose ici de suivre les instructions du forum **Malekal**.
  - Appuyer **Win+R** puis **Sysdm.cpl**
  - Cliquer sur l'onglet **Paramètres système avancé**
  - Cliquer sur l'onglet **Variable d'environnement**
  - Sélection la variable utilisateur **Path** et sur l'onglet **Modifier**
  - Cliquer sur l'onglet **Nouveau**
  - Écrire le chemin
- Si **Python** n'est pas installé pour l'installer, aller sur le site https://www.python.org/downloads/ Il est alors possible soit de télécharger un fichier **.msix** pour une installation organisée avec les autres logiciels ou **.exe** qui installe tout ce qui est nécessaire éventuellement en doublons avec d'autres logiciels. Une fois ce fichier téléchargé, il suffit de l'exécuter pour avoir l'installation.

  Au moment de l'exécution, je crois qu'avoir une installation commune à tous les utilisateurs réduit les problèmes d'installations. Et je crois qu'on a cela par

défaut si on exécute avec les droits administrateurs, ce qui peut se faire en cliquant avec le boutton droit de la souris et en cliquant sur l'onglet **exécuter en tant qu'administrateur**.

- Tester si l'installation a fonctionné et éventuellement modifier la variable **Path** comme indiqué plus haut.

3. Vérifier si **pip** est installé sur l'ordinateur en ouvrant une fenêtre dos et en tapant **pip** et en observant des explications sur son utilisation. Si ce n'est pas le cas :

   - Télécharger le fichier **get-pip.py** présent dans https://bootstrap.pypa.io/get-pip.py Ceci affiche dans le logiciel de navigation le contenu de ce fichier. Pour l'enregistrer, cela peut se faire sur **Firefox** en tapant **Ctr+S** (sans le shift).
   - Dans une fenêtre **Dos**, taper **python get-pip.py**.

     pip s'installe aussi différemment et agit différemment suivant qu'il concerne tous les utilisateurs ou un utilisateur. Pour avoir par défaut l'installation à tous les utilisateurs, il me semble qu'il faut ouvrir une fenêtre **Dos** en tant qu'administrateur, ce qui se fait d'abord avec les touches **Win** et **R**, puis en écrivant **cmd** et au lieu de cliquer sur l'onglet **Ok**, taper sur les touches **Ctrl+Maj** et **Entrée**.
   - Lors de l'installation, il peut y avoir un Warning qui indique que le logiciel **pip** est installé mais pas ajouté à la variable **Path**. Ce Warning indique où

se trouve le logiciel à installer. Il suffit de suivre les indications de l'étape 2 pour ajouter ce chemin et ainsi permettre l'utilisation de `pip` depuis n'importe quelle fenêtre `dos`.

4. Vérifier si `Python` tel qu'il est installé contient `numpy`, `matplotlib` et `scipy`.

   - Ouvrir une fenêtre `Dos`, taper `python` puis les lignes suivantes
     - `import numpy`
     - `import matplotlib`
     - `import scipy`

     Pour les lignes qui entraînent une erreur, l'installation des modules manquant se fait en fermant `Python` avec `exit()` puis avec l'instruction suivant dans la fenêtre `Dos` `pip install numpy`, `pip install matplotlib`, `pip install scipy`

     De même pour utiliser une installation commune à tous les utilisateurs, utiliser une fenêtre `Dos` en tant qu'administrateur comme indiqué plus haut.

Pour l'éditeur de texte, j'utilise à titre personnel `Notepad++`. Il faut surtout vérifier que l'indentation du texte se fasse avec des espaces et non des tabulations, (l'indentation signifie que le texte est décalé vers la droite, ce qui est requis par Python dans les boucles ou dans les fonctions).

B.2 Codes pour simulation une réponse fréquentielle

B.3 Simulation d'une sortie d'un filtre

B.4 Simulation d'un signal défini par des fonctions de base

B.5 Simulation d'un signal échantillonné

B.6 Simulation d'un calcul d'erreur quadratique

B.7 Simulation d'une reconstruction par interpolation

B.8 Code pour simuler une réponse fréquentielle

Python :

Table B.1: Implémentation de la figure **??**.

```
import seb; plt,np=seb.debut();
plt.close('all')
def s_traj(fe):
  """realise 1 trajectoires de frequences fe"""
  tx=np.arange(-2,2,1/fe)
  x=np.random.normal(0,1,len(tx))*np.sqrt(fe)
  th=np.arange(0,1,1/fe)
  h=np.ones(len(th))/fe
  y=seb.convolution(tx,x,th,h,tx)*fe
  return tx,y

N=100
fe=100
s=np.zeros(len(np.arange(-2,2,1/fe)))
for n in range(N):
  t,s1 = s_traj(fe)
  s += s1/N
fig,ax = plt.subplots()
ax.plot(t,s)
ax.set_xlabel('t')
plt.tight_layout()
fig.savefig('./figures/fig_C8_4_6a.png')
```

```
fig.show()

K=100
fe_l=np.linspace(5,300,K)
s=np.zeros(K)
for k in range(K):
  t,s1 = s_traj(fe)
  s[k] += np.mean(s1)
fig,ax = plt.subplots()
ax.plot(fe_l,s)
ax.set_xlabel('fe')
plt.tight_layout()
fig.savefig('./figures/fig_C8_4_6b.png')
fig.show()

N=100
fe=100
s=np.zeros(len(np.arange(-2,2,1/fe)))
for n in range(N):
  t,s1 = s_traj(fe)
  s += s1**2/N
fig,ax = plt.subplots()
```

```
ax.plot(t,s)
ax.set_xlabel('t')
plt.tight_layout()
fig.savefig('./figures/fig_C8_4_6c.png')
fig.show()
```