

Introduction au signal et bruit

Cours

Gabriel Dauphin

November 23, 2025

Contents

1 Relations entrées-sorties sans effet mémoire	4
1.1 Grandeurs étudiées	4
1.2 Première utilisation de Python	5
1.3 Notion de filtres	6
1.4 Association de filtres	9
2 Signaux temps continu, fonction affine par morceaux	11
2.1 Des exemples de signaux	11
2.2 Utilisation du crochet d'Iverson	13
2.3 Présentation de quatre fonctions de base	14
2.4 Effet d'un retard/avance sur un signal	15
2.5 Effet d'une dilatation/concentration de l'échelle des abscisses	17
2.6 Illustration sur les fonctions de base	18
3 Définition et utilisation de la transformée de Fourier	21
3.1 Somme, énergie, moyenne et puissance	21
3.2 Transformée de Fourier : définition	23
3.3 Simuler numériquement des intégrales de fonctions non-sommables	25
4 Propriété de la transformée de Fourier	27
4.1 Propriété de la transformée de Fourier de la dilatation de l'échelle des temps	27
4.2 Propriété de la transformée de Fourier d'un retard ou d'une avance	28
4.3 Symétrie et transformée de Fourier	28
5 Diracs	30
5.1 Introduction à la notion de distribution de Dirac	30
5.2 Règles de calcul pour le Dirac	30
5.3 Règles de calcul pour dériver un signal discontinu	31
5.4 Condition initiale et Dirac	33
5.5 Dérivées d'un Dirac	34
6 Transformées de Fourier et dérivation	35
6.1 Introduction	35
6.2 Propriété de la transformée de Fourier vis-à-vis de la dérivation	36
6.2.1 Calculs d'une primitive	37
6.3 Transformer la définition d'un spectre en une équation différentielle définissant un signal	37
7 Équations différentielles	39
7.1 Bref rappel pour résoudre analytiquement des équations différentielles à coefficients constant	39
7.2 Simulation numérique des solutions des équations différentielles	40

8 Filtres et effet mémoire	42
8.1 Notion de filtre et de relation entrée-sortie	42
8.2 Présence d'un effet mémoire dans la relation entrée-sortie	43
8.3 Filtre temps invariant	43
8.4 Produit de convolution	44
8.4.1 Réponse impulsionnelle	45
9 Description fréquentielle des filtres	47
9.1 Transformée de Fourier et produit de convolution	47
9.2 Réponse fréquentielle	47
9.3 Réponse impulsionnelle	48
9.4 Filtres passe-bas, passe-haut et passe-bande	48
9.5 Fréquence de coupure et bande passante d'un filtre	49
9.6 Sens de variation de la réponse fréquentielle	50
10 Signaux périodiques	52
10.1 Définition d'un signal périodique	52
10.2 Somme, moyenne, énergie et puissance	53
10.3 Transformée de Fourier	54
10.4 Périodisation	57
10.5 Calcul numérique des coefficients de la série de Fourier	58
10.6 Relation entre transformée de Fourier et calcul des coefficients de la série de Fourier	59
11 Filtres agissant sur des signaux périodiques	61
11.1 Réponse forcée, réponse transitoire	61
11.2 Utilisation de la réponse fréquentielle pour calculer la réponse forcée	62
12 Échantillonnage d'un signal non-périodique	64
12.1 Signal temps discret	64
12.2 Échantillonnage	65
12.3 Signaux temps discret non périodiques : transformées de Fourier à temps discret	66
12.4 Périodisation du spectre	67
12.5 Représentation du spectre	68
12.6 Interpolation	68
12.7 Produit de convolution à temps discret	70
12.8 Simulation de l'erreur quadratique moyenne en fonction de f_e	70
12.8.1 Reconstruction du signal par interpolation similaire à la formule de Shannon-Nyquist	72
12.8.2 Simulation de l'erreur quadratique moyenne en fonction de f_e après reconstruction	72
13 Modélisation stochastique du bruit	73
13.1 Qu'est-ce que le bruit ?	73
13.2 Bruit modélisé par une variable aléatoire	74
13.3 Covariance, corrélation et indépendance entre variables aléatoires	76
13.4 Processus aléatoire	77
13.4.1 Généralités	77
13.4.2 Processus aléatoire blanc	78
14 Autocorrélation et densité spectrale	80
14.1 Présentation physique du bruit thermique	80
14.2 Définition caractéristiques stochastiques	81
14.2.1 Distinction énergie finie ou infinie	81
14.2.2 Cas du bruit blanc tel que défini dans la section 13.4	82
14.2.3 Cas d'un processus aléatoire centré	82
14.2.4 Cas d'un processus déterministe d'énergie finie	83
14.2.5 Cas d'un processus quasi-déterministe	84

14.2.6	Définition d'un processus aléatoire approchant un bruit blanc au moyen de motifs	85
14.3	Propriété de l'autocorrélation et de la densité spectrale	85
14.3.1	Processus aléatoire à temps continu	85
14.3.2	Processus aléatoire à temps discret	85
14.4	Simulation de processus aléatoires	86
14.4.1	Bruit blanc échantillonné	86
15	Filtrage des processus aléatoires et application au bruit en $1/f$	88
15.1	Présentation physique du bruit en $1/f$	88
15.2	Filtrage d'un processus aléatoire	89
15.3	Nouvelle définition du produit de convolution	89
15.4	Filtre appliqué à une entrée qui est un bruit blanc échantillonné	89
15.5	Filtre appliqué à une entrée qui est un processus aléatoire quelconque	90
15.6	Simuler un bruit en $1/f$	90
15.6.1	Densité spectrale	90
15.6.2	Autocorrélation	90
15.6.3	Trajectoires	91
16	Densité de probabilité et application au bruit de grenaille	93
16.1	Présentation physique du bruit de grenaille	93
16.2	Filtre appliqué à une entrée qui est un bruit blanc avec utilisation d'un motif	93
16.2.1	Bruit blanc avec motifs	93
16.3	Cas où l'entrée est un processus aléatoire gaussien	95
16.3.1	Écart à la moyenne pour une densité de probabilité gaussienne	95
16.4	Cas où l'entrée est un processus aléatoire non-gaussien	96
16.4.1	Écart à la moyenne pour une densité de probabilité ayant un variance finie	96
A	Code pour simuler les différentes figures	97
A.1	Code commun	97
A.1.1	Spécificités de Python	97
A.2	Codes utilisant les fonctions de bases	99
A.3	Codes utilisant la transformée de Fourier et la transformée de Fourier inverse	101
A.4	Codes simulant la dérivée	102
A.5	Codes pour simuler une réponse impulsionale	103
A.6	Simulation d'un filtrage	103
A.7	Codes pour simuler la périodisation et la série de Fourier	105
A.8	Code pour simuler la solution d'une équation différentielle	108
A.9	Codes pour simuler des variables aléatoires	112
A.10	Codes pour simuler des processus aléatoires	115
B	Quelques outils pour calculer l'intensité ou la tension	118
B.1	Association de résistances	118
B.2	Diviseur de tension	118
B.3	Théorème de Millman	118
B.4	Quadripôle	118
C	Notations utilisées	120
C.1	Notations utilisées dans le cours signal et bruit	120
C.2	Notations utilisées dans en ce qui concerne l'électronique	120

Chapter 1

Relations entrées-sorties sans effet mémoire

1.1 Grandeurs étudiées

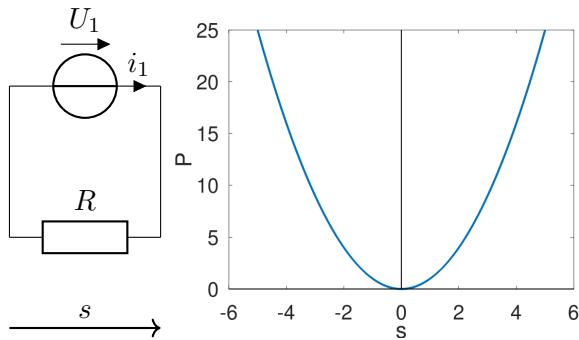


Figure 1.1: À gauche : Circuit générant une tension continue $s = U_1$. À droite : relation entre la puissance d'un signal et la valeur du signal.

Cours signal et bruit :

Dans ce chapitre, on ne s'intéresse pas à la dépendance vis-à-vis du temps. La grandeur étudiée pourrait être l'intensité ou la tension, mais ici on va considérer que c'est la tension et en traitement du signal, on n'indique pas l'unité du signal. On le note s pour [signal]. Il est à valeurs réelles dans le cadre de ce cours (sauf pour des astuces de calcul).

D'un point de vue électronique :

La figure 1.1 montre un circuit générant une tension $s = U_1$, celle-ci ne dépend pas de R . L'implémentation est dans ??.

En électronique, la puissance consommée ou fournie est le produit tension aux bornes d'un composant par intensité traversant le composant, elle est notée P_s , son unité est le W. Pour ce montage, $P_s = s \times \frac{s}{R} = \frac{s^2}{R}$, (s est la tension et $\frac{s}{R}$ est l'intensité. Et en général la puissance instantanée dépend de $u(t)i(t)$ et non uniquement d'un seul signal).

Plus généralement en physique, la notion d'énergie est différente, ce n'est pas une notion qui se déduit de la formule du signal. Le même signal peut représenter suivant le contexte à une énergie finie ou infinie et du coup être décrit plutôt par la notion d'énergie ou de puissance, celle-ci étant aussi la variation de l'énergie au cours du temps. Ainsi quand on considère un système fermé (i.e. sans interaction avec l'extérieur), l'énergie est conservé au cours du temps. Un tel phénomène pourrait être décrit par un signal constant et du point de vue du traitement du signal, on dit qu'on a une énergie infinie, mais pas en physique. Quand on subdivise ce système fermé en deux systèmes A et B qui interagissent et qui donc ont des transfert d'énergie, lorsque A transfert une quantité pendant un laps de temps T vers B, on a bien un transfert d'énergie au cours du temps T qui est le résultat d'une intégration au cours du temps d'une puissance instantanée transférée. Et si on imagine qu'en même temps B effectue vers A le même transfert, on pourrait avoir une puissance instantanée constante au cours du temps qui cumulée sur un temps infini correspond à une énergie infinie. On aurait ainsi un transfert d'énergie infinie entre deux systèmes, chacun ayant une énergie finie et même constante. Un circuit RC (i.e. formé d'une résistance et d'un condensateur) a un comportement voisin : la source de tension parfois alimente la résistance aidée du condensateur qui se décharge, et parfois la source de tension

alimente à la fois la résistance et le condensateur qui se charge. L'énergie dans le condensateur est finie et vaut $\frac{1}{2}Cu^2(t)$ alors qu'en traitement du signal $u(t)$ en tant que signal périodique a une énergie infinie.

Cours signal et bruit :

En traitement du signal, la puissance est le carré du signal :

$$P_s = s^2 \quad (1.1)$$

Et on observe qu'il y a ici un lien avec $P_s = \frac{P_s}{R}$. Dans la suite du cours on désignera ces deux notions par puissance instantanée, car dans la suite on considère des signaux dépendant du temps avec une puissance commune à l'ensemble du signal.

On appelle ici pseudo-code la liste des tâches à accomplir pour calculer les vecteurs utilisés pour représenter une figure.

1.2 Première utilisation de Python

Python :

Pour démarrer l'utilisation de Python, je propose de choisir deux répertoires, le premier que j'appelle `rep_prg` contient les programmes, notamment ceux disponibles sur mon site, et un autre répertoire que j'appelle `rep_tra` où vous mettez le travail effectué notamment les données et les figures.

Je propose ensuite d'effectuer les lignes suivantes qui utilisent un module `seb` que j'ai écrit pour ce cours. `plt` et `np` sont les modules `matplotlib` et `numpy` qui servent à tracer des graphes et à manipuler des vecteurs.

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

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.

```
s=np.linspace(-5,5,100)
```

Cette instruction génère 100 valeurs entre -5 et 5. Cette notion est utile pour générer un graphique, elle permet d'obtenir le graphe à droite 1.1 représentant la puissance instantanée en fonction de la valeur du signal. L'implémentation de $P_s = s^2$ se fait avec

```
P=s**2
```

On obtient alors un vecteur ligne de même taille que `s` et contenant successivement toutes les valeurs de P_s pour chacune des valeurs du vecteur `s`. Pour faire le graphe, on commence par

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

Ensuite pour la courbe on rajoute ¹ Le troisième argument permet de rajouter une légende.

```
ax.plot(s,P_s,label='puissance')
```

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

```
ax.set_xlabel('x')
```

¹Et si on avait plusieurs courbes, il suffit de rajouter une ligne par courbe.

La légende sur l'axe des ordonnées se fait de la même façon avec `ax.set_ylabel('y')`.

L'affichage de la légende est déclenchée par

```
ax.legend()
```

La gestion de la taille de la figure est faite avec

```
plt.tight_layout()
```

Je propose de sauvegarde la figure

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

La figure apparaît lorsqu'on exécute cette commande

```
fig.show()
```

On peut restreindre (ou dit autrement zoomer) le graphe à une plus petite plage de valeurs. Les commandes suivantes permettent par exemple d'afficher uniquement la partie du graphe où l'axe des abscisses varie entre 5 et 7 et où l'axe des ordonnées varie entre -1 et 2

```
ax.set_xlim(5,7)
ax.set_ylim(-1,2)
```

Il est souhaitable de ne pas avoir trop de figures ouvertes en même temps, (il est toujours possible de les sauvegarder). La commande suivante permet de fermer toutes les figures actuellement ouvertes.

```
plt.close('all')
```

1.3 Notion de filtres

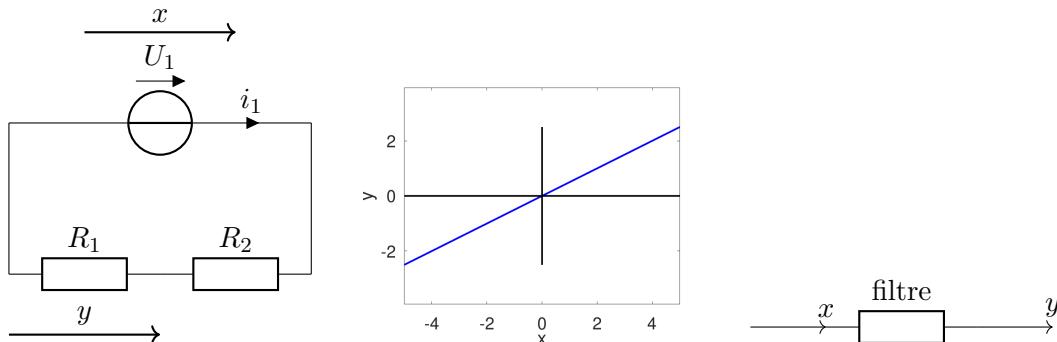


Figure 1.2: À gauche : diviseur de tension. Au centre : graphe de y en fonction de x . À droite : notation pour le filtre transformant x en y .

Cours signal et bruit :

Du point de vue de traitement du signal, on cherche à décrire la relation mais aussi la façon dont on peut utiliser cette relation.

- Une **entrée** est ce que l'on peut assigner.
- Une **sortie** est ce que l'on peut mesurer.

La sortie dépend de l'entrée et on utilise dans ce cours la notation \mathcal{H} pour indiquer cette relation

$$y = \mathcal{H}(x) \quad (1.2)$$

Ce filtre est représenté sur la figure 1.2.

Parmi les filtres, on distingue, les relations **linéaires**. Dans ce chapitre on dit qu'un filtre est linéaire, s'il existe un coefficient de proportionnalité appelé **gain statique** G

$$y = Gx \quad (1.3)$$

Lorsque $G > 1$, on dit qu'il y a une **amplification**. Si ce coefficient vaut $G < 1$ alors on dit qu'il y a une **atténuation**.

D'un point de vue électronique :

C'est un point de vue différent, les outils semblent être d'abord définis pour simuler un signal au sein d'un montage. La figure 1.2 présente à gauche un diviseur de tension et à droite la relation linéaire.

$$y = x \frac{R_1}{R_1 + R_2} \quad (1.4)$$

Une particularité technologique est que y est nécessairement du même signe que x et est inférieur à x .

Le fait que pour chaque valeur de l'entrée, il y ait une unique valeur possible en sortie, c'est justement un effet sans mémoire.

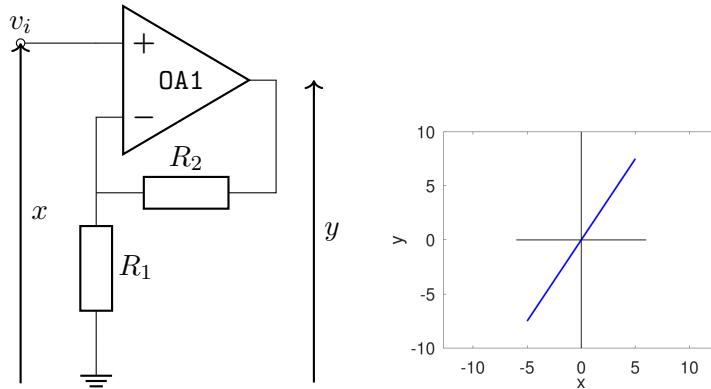


Figure 1.3: À gauche : utilisation d'un amplificateur en boucle fermé. À droite : graphe de y en fonction de x .

D'un point de vue électronique :

La figure 1.3 présente un amplificateur non-inversé en boucle fermé et à droite la relation linéaire correspondante.

$$y = x \left(\frac{R_1 + R_2}{R_1} \right) \quad (1.5)$$

Cette équation vient du théorème de Millman (voir section B.3 p. 118).

Une particularité technologique est que y est nécessairement du même signe que x et est supérieur à x .

D'un point de vue électronique :

On remarque que ici quand le gain statique est supérieur à 1, le montage ne peut être obtenu avec des composants passifs et une source extérieure d'alimentation est nécessaire. En réalité ce n'est pas toujours vrai quand on considère un signal dépendant du temps (par exemple un transformateur ou une alimentation à découpage). Les considérations énergétiques du traitement du signal ne peuvent avoir un vrai sens physique dans la mesure où en traitement du signal un filtre ne concerne pas à la fois $i(t)$ et $u(t)$.

Cours signal et bruit :

Le rapport entre la puissance en sortie d'une relation entrée-sortie par rapport à la puissance en entrée est

$$\frac{P_y}{P_x} = G^2 \quad (1.6)$$

En terme de visualisation :

Le graphe d'une relation linéaire est une droite passant par l'origine $(0,0)$. Un graphe ayant les mêmes graduations pour les deux axes est dit **orthonormé**. Et dans ce cas, il s'agit d'une amplification si l'angle avec l'horizontal est plus grand que $\frac{\pi}{4}$, et il s'agit d'une atténuation si l'angle avec l'horizontal est plus petit que $\frac{\pi}{4}$,

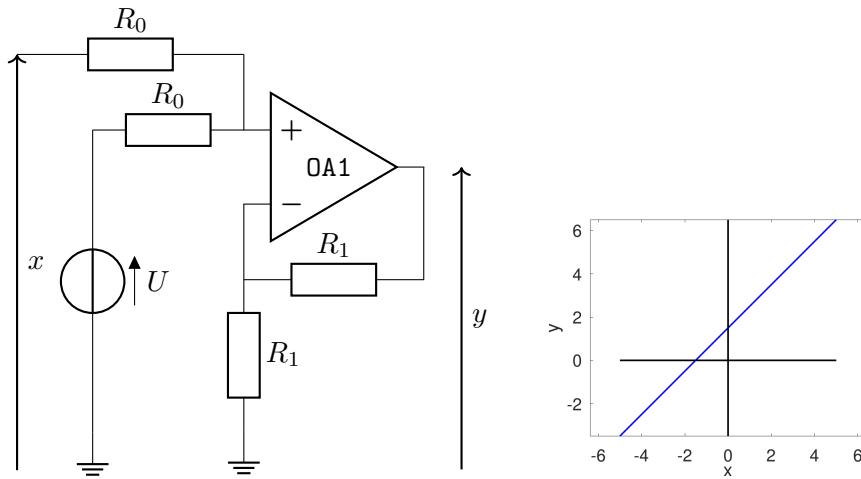


Figure 1.4: À gauche : utilisation d'un amplificateur en boucle fermé. À droite : graphe de y en fonction de x .

Cours signal et bruit :

La figure 1.4 montre à gauche un additionneur de tension et à droite la relation induite qui est l'ajout d'une constante.

$$y = x + U \quad (1.7)$$

Plus généralement lorsqu'une relation s'écrit $y = ax + b$, on dit qu'il s'agit d'une relation affine.

En terme de visualisation :

Le graphe d'une relation affine est une droite.

D'un point de vue électronique :

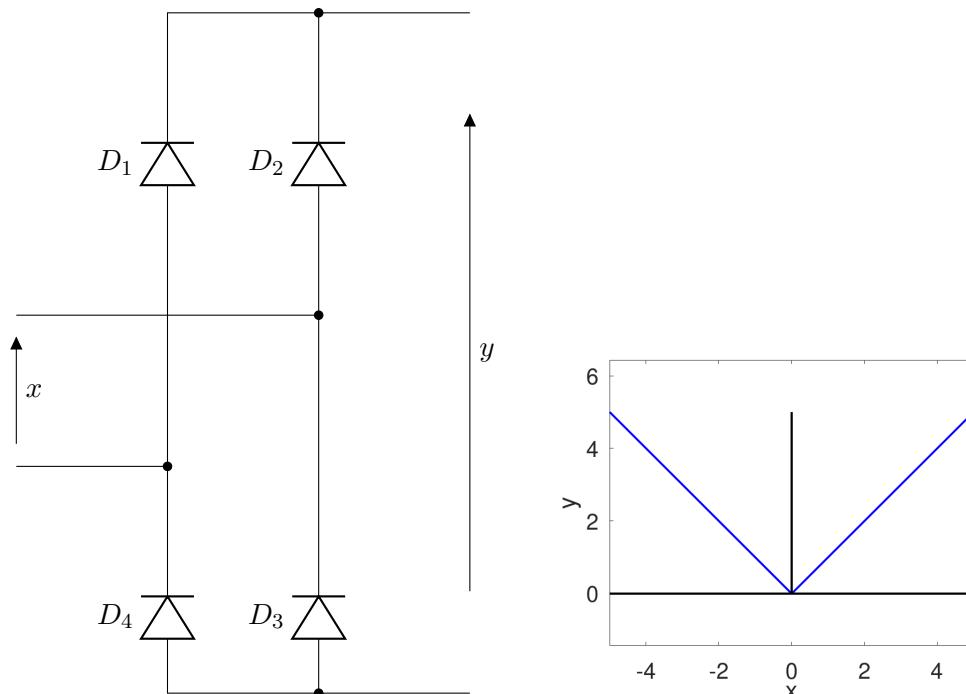


Figure 1.5

La figure 1.5 montre à gauche un pont de diode et à droite la relation entrée-sortie.

D'un point de vue mathématique :

Valeur absolue

$$|t| = \begin{cases} t & \text{si } t \geq 0 \\ -t & \text{si } t < 0 \end{cases}$$

$$y = \begin{cases} x & \text{si } x \geq 0 \\ -x & \text{si } x \leq 0 \end{cases} = |x| \quad (1.8)$$

Deux autres relations non-linéaires sont données par les opérateurs `min` et `max`.

$$\min(a, x) = \begin{cases} x & \text{si } x \leq a \\ a & \text{si } x \geq a \end{cases} \quad \text{et} \quad \max(a, x) = \begin{cases} a & \text{si } x \leq a \\ x & \text{si } x \geq a \end{cases} \quad (1.9)$$

Python :

La valeur absolue est ainsi implémentée

```
print(np.abs(-3), np.max((-3,3)), np.min((-3,3)), min(-3,3), max(-3,3))
```

L'opération consistant à limiter les valeurs d'une fonction à un certain intervalle au moyen de `max` et `min` sont ainsi implémentées

D'un point de vue mathématique :

Les fonctions définies avec des valeurs absolues et plus généralement les fonctions utilisant des définitions différentes suivant les intervalles considérés, elles ont des propriétés mathématiques délicates à utiliser. Par exemple la primitive de $|t|$ est en fait $\frac{t|t|}{2}$. Aussi plutôt que de chercher à étudier ces propriétés, la façon usuelle de procéder consiste à découper le problème en autant d'intervalles nécessaires et sur chacun de ces intervalles procéder à des calculs classiques.

Les relations affines et non-linéaires ne sont pas *linéaires*, c'est-à-dire qu'elles ne vérifient pas

$$\begin{cases} x = x_1 + x_2 \Rightarrow y = y_1 + y_2 \\ x = \alpha x_1 \Rightarrow y = \alpha y_1 \end{cases} \quad (1.10)$$

lorsque y_1, y_2, y sont les sorties de x_1, x_2, x .

En terme de visualisation :

L'équation (1.10) signifie que la relation entre x et y n'est pas une droite passant par l'origine.

1.4 Association de filtres

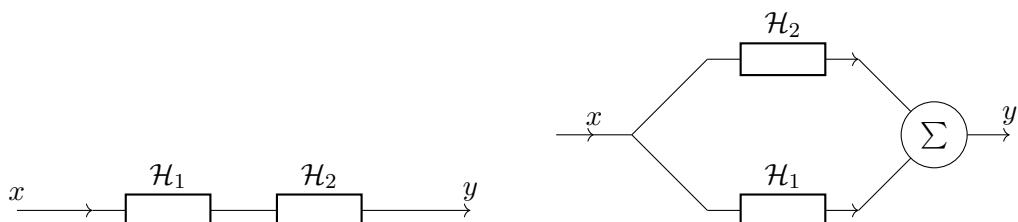


Figure 1.6: À gauche : deux filtres en série.

Cours signal et bruit :

On considère une première façon d'associer deux filtres, en les mettant l'un à la suite de l'autre, on dit aussi **[en série]**. Elle est représentée à gauche de la figure 1.6. L'ensemble de ces deux filtres constitue un nouveau filtre noté ici \mathcal{H} .

D'un point de vue mathématique :

il s'agit d'une composition de fonctions

$$\mathcal{H} = \mathcal{H}_2 \circ \mathcal{H}_1 \quad y = \mathcal{H}(x) = \mathcal{H}_2[\mathcal{H}_1(x)] \quad (1.11)$$

Remarquez que ce qui est dessiné en allant de la gauche vers la droite est représenté dans une équation en allant de la droite vers la gauche.

Cours signal et bruit :

On considère une deuxième façon d'associer deux filtres, en faisant suivre chacun d'eux du même opérateur de somme noté avec \sum , on dit qu'on **[ajoute les sorties]** des filtres. Elle est représentée à droite de la figure 1.6. L'ensemble de ces deux filtres constitue un nouveau filtre noté ici \mathcal{H} .

D'un point de vue mathématique :

il s'agit de la somme de deux fonctions

$$\mathcal{H} = \mathcal{H}_1 + \mathcal{H}_2 \quad y = \mathcal{H}(x) = \mathcal{H}_1(x) + \mathcal{H}_2(x) \quad (1.12)$$

D'un point de vue électronique :

On observe que la signification des associations est très différente de ce qui se fait en électronique. Avec l'utilisation des quadripôles présentés en annexe B.4, on peut retrouver des idées similaires.

Chapter 2

Signaux temps continu, fonction affine par morceaux

2.1 Des exemples de signaux

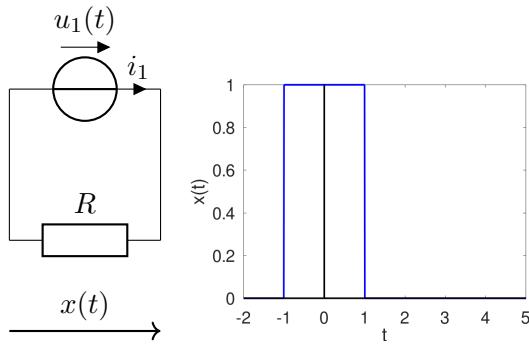


Figure 2.1: Illustration l'équation (2.6) pour $u_1(t) = 1$ pour $t \in [-1, 1]$.

Cours signal et bruit :

Un signal dépendant du temps et ayant une valeur définie pour chaque instant est dit **analogique**. Il est noté $x(t), y(t), s(t)\dots$ en fonction de la lettre, x,y ou z associée au signal.

Le signal décrit à droite de la figure 2.2 est défini par

$$x(t) = \begin{cases} 0 & \text{si } t < -1 \\ 1 & \text{si } t \in [-1, 1[\\ 0 & \text{si } t \geq 1 \end{cases} \quad (2.1)$$

- \mathbb{R} désigne l'ensemble des nombres et n'inclut pas i ,
- \mathbb{R}_+ sont les nombres positifs ou nuls,
- \mathbb{R}_- sont les nombres négatifs ou nuls.

Cours signal et bruit :

Ce qu'on appelle un signal est dans ce chapitre une fonction dépendant d'une variable notée t et qui correspond au temps. Une fonction dépendant d'une variable est notée en mathématiques f , sa valeur calculée pour une valeur particulière de t est notée $f(t)$, ces notations montrent qu'il est d'usage en mathématique de faire la différence entre une fonction et une valeur particulière pour une valeur de t . Parce que cette distinction n'est pas essentielle et qu'en revanche il existe des signaux en théorie du signal qui sont des suites, on préfère généralement noter le signal par $x(t)$ plutôt que x . Pour l'instant les signaux considérés sont à valeurs réelles mais on verra lors de la section ?? qu'un signal peut être à valeurs complexes. Un signal peut ne pas être une fonction.

Pour aider la compréhension des équations, il est important dans le cadre de ce cours que lorsqu'on écrit qu'un signal est égal à un autre signal, la variable t figure à gauche et à droite du signe $=$. Par exemple, on ne doit pas écrire $x + 1 = \frac{1}{t^2+1}$ mais $x(t) + 1 = \frac{1}{t^2+1}$.

Un signal défini sur un intervalle donné par exemple pour $t \geq 0$ est vu ici comme un signal défini sur \mathbb{R} , nul pour toutes les valeurs où ce signal n'est pas défini.

D'un point de vue mathématique :

On appelle **discontinuité** les sauts que l'on observe en $t = -1$ et en $t = 1$ sur la droite de la figure 2.1. Dans le cadre de ce cours, on ne s'intéresse pas à la valeur effective du signal sur le saut, en l'occurrence, le signal vaut-il 0, 1 ou une valeur intermédiaire au moment où exactement $t = -1$. En revanche si au cours d'un calcul on utilise des fonctions avec des discontinuités et qu'à la fin du calcul il n'y a plus de discontinuités, la valeur en ces points nous intéresse.

D'un point de vue électronique :

Une discontinuité pour $u(t)$ (ou pour $i(t)$) est effectivement quelque chose qui se produit physiquement : la tension est discontinue lorsqu'on connecte et l'intensité est discontinue lorsqu'on déconnecte. Lorsqu'on met sous tension entre $t = -1$ et $t = 1$, on mesure avec le voltmètre un signal dans le circuit décrit à gauche de la figure 2.1. L'explication est que quand la source de tension est nulle, il n'y a pas de courant et la tension est nulle. Quand il y a une source de tension, cela crée un courant qui est tel que $x(t) = U_1$.

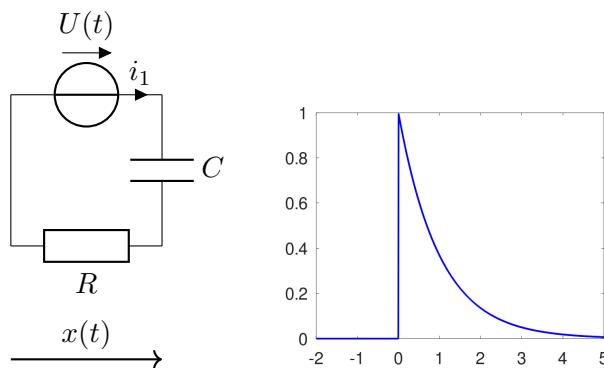


Figure 2.2: Illustration l'équation (2.8) avec $U_1 = 1V$ et $RC = 1s$

Cours signal et bruit :

Le signal décrit à droite de la figure 2.2

$$x(t) = \begin{cases} 0 & \text{si } t \leq 0 \\ e^{-t} & \text{si } t > 0 \end{cases} \quad (2.2)$$

Python :

Pour réaliser la droite de la figure 2.2, la première étape est de générer un grand nombre de valeurs entre -2 et 5 qui soient également réparties (ici c'est 10^3).

```
t=np.linspace(-2,5,10**3)
```

Pour chacune de ces valeurs, on calcule la valeur du signal $x(t)$. Cette première instruction n'est pas correcte car elle est non-nulle pour $t < 0$.

```
x=np.exp(-t)
```

Pour la rendre nulle pour $t < 0$, on multiplie par $(t>=0)$ qui vaut 1 quand $t \geq 0$ et 0 quand $t < 0$.

```
x=(t>=0)*np.exp(-t)
```

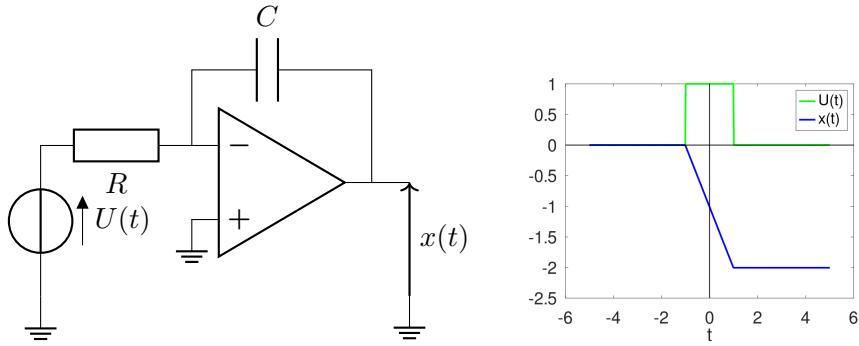


Figure 2.3: À gauche : montage intégrateur avec $RC = 1$. À droite : graphe de $x(t)$ en fonction de t .

Cours signal et bruit :

Le signal décrit à droite de la figure 2.3 est

$$x(t) = \begin{cases} 0 & \text{si } t \leq -1 \\ -(t+1) & \text{si } t \in [-1, 1[\\ -1 & \text{si } t \geq 1 \end{cases} \quad (2.3)$$

D'un point de vue mathématique :

La formule $-(t+1)$ est l'équation d'une droite, une technique générale pour trouver cette équation de droite est d'utiliser deux points, ici $(-1, 0)$ et $(1, -2)$. En notant (t_1, x_1) et (t_2, x_2) , l'équation de la droite est définie par

$$\frac{x - x_1}{t - t_1} = \frac{x_2 - x_1}{t_2 - t_1} \quad (2.4)$$

En l'occurrence,

$$\text{à partir de } \frac{x - 0}{t + 1} = \frac{-2 - 0}{1 - (-1)}, \text{ on a } x = (-2)/2 \times (t + 1) = -(t + 1) \quad (2.5)$$

2.2 Utilisation du crochet d'Iverson

Le crochet d'Iverson est une façon pratique de coder les équations (2.1), (2.2), (2.3).

$$\llbracket \mathcal{P} \rrbracket = \begin{cases} 1 & \text{si } \mathcal{P} \text{ est vraie} \\ 0 & \text{si } \mathcal{P} \text{ est fausse} \end{cases}$$

$$x(t) = \llbracket |t| \leq 1 \rrbracket = \llbracket -1 \leq t \leq 1 \rrbracket \quad (2.6)$$

$\llbracket \dots \rrbracket$ permet de définir l'échelon d'Heaviside défini par

$$x(t) = \llbracket t \geq 0 \rrbracket \quad (2.7)$$

Il permet aussi de définir un signal exponentiellement décroissant pour $t \geq 0$ et nul pour $t < 0$.

$$x(t) = e^{-t} \llbracket t \geq 0 \rrbracket \quad (2.8)$$

$\llbracket \dots \rrbracket$ permet aussi de représenter des fonctions affines par morceaux (i.e. des segments de droites joints). Le signal $x(t)$ est une portion de rampe suivie d'un échelon.

$$x(t) = -(t+1) \llbracket -1 \leq t < 1 \rrbracket - \llbracket t \geq 1 \rrbracket \quad (2.9)$$

D'un point de vue mathématique :

La notation utilisant le crochet d'Iverson devient ambigu si on y met deux variables à l'intérieur, parce qu'alors il n'est plus clair à laquelle de ces deux variables dans le crochet, la valeur entre parenthèse signifie. Par exemple

$$\llbracket \tau < t \rrbracket(0) \text{ signifie-t-il } \begin{cases} \llbracket 0 < t \rrbracket = \begin{cases} 0 \text{ si } t \leq 0 \\ 1 \text{ si } t > 0 \end{cases} \\ \text{ou} \\ \llbracket \tau < 0 \rrbracket = \begin{cases} 0 \text{ si } \tau \geq 0 \\ 1 \text{ si } \tau < 0 \end{cases} \end{cases} \quad (2.10)$$

Dans le cadre de ce cours, on n'utilisera pas cette notation avec deux variables à l'intérieur.

2.3 Présentation de quatre fonctions de base

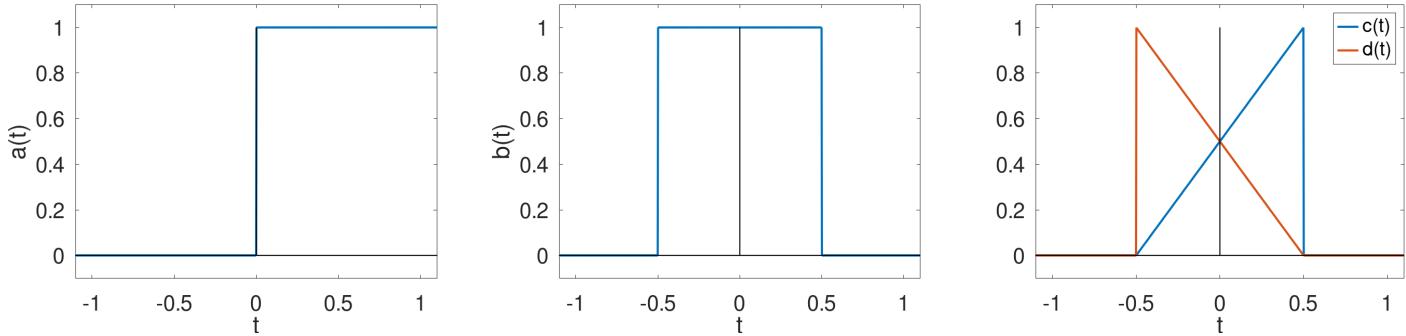


Figure 2.4: Représentation de $\mathbb{H}(t)$, $\Pi(t)$, $\mathbb{C}(t)$, $\mathbb{D}(t)$.

Cours signal et bruit :

L'échelon d'Heaviside est représenté à gauche de la figure 2.4 est noté ici $\mathbb{H}(t)$, il est défini par

$$\mathbb{H}(t) = \begin{cases} 0 \text{ si } t < 0 \\ 1 \text{ si } t \geq 0 \end{cases} \quad (2.11)$$

La porte centrée et de largeur 1 est représentée en haut au milieu de la figure 2.4, elle est notée ici $\Pi(t)$, elle est définie par

$$\Pi(t) = \begin{cases} 0 \text{ si } t < -0.5 \\ 1 \text{ si } t \in [-0.5, 0.5[\\ 0 \text{ si } t \geq 0.5 \end{cases} \quad (2.12)$$

Le demi-triangle croissant de largeur 1 est représenté en haut à droite de la figure 2.4, il est notée ici $\mathbb{C}(t)$, il est définie par

$$\mathbb{C}(t) = \begin{cases} 0 \text{ si } t < -0.5 \\ t + 0.5 \text{ si } t \in [-0.5, 0.5[\\ 0 \text{ si } t \geq 0.5 \end{cases} \quad (2.13)$$

Le demi-triangle décroissant de largeur 1 est représenté en haut à droite de la figure 2.4, il est notée ici $\mathbb{D}(t)$, il est définie par

$$\mathbb{D}(t) = \begin{cases} 0 \text{ si } t < -0.5 \\ 0.5 - t \text{ si } t \in [-0.5, 0.5[\\ 0 \text{ si } t \geq 0.5 \end{cases} \quad (2.14)$$

Le triangle de largeur 2 est représenté en haut à droite de la figure 2.4, il est notée ici $\mathbb{T}(t)$, il est définie par

$$\mathbb{T}(t) = \begin{cases} 0 & \text{si } t < -1 \\ 1 - |t| & \text{si } t \in [-1, 1[\\ 0 & \text{si } t \geq 1 \end{cases} \quad (2.15)$$

Le crochet d'Iverson se prête à une implémentation très simple, parce que une expression conditionnelle utilisant des vecteurs de mêmes tailles vaut 1 si cette expression est correcte et 0 sinon.

Python :

- L'équation (2.11) devient
 $x=(t>0)$ | $x=\text{seb.fonction_H}(t)$
- L'équation (2.12) devient
 $x=((t>=-1/2)&(t<=1/2))$ | $x=\text{seb.fonction_P}(t);$
- L'équation (2.13) devient
 $x=(0.5+t)*((t>=-1/2)&(t<=1/2))$ | $x=\text{seb.fonction_C}(t);$
- L'équation (2.14) devient
 $x=(0.5-t)*((t>=-1/2)&(t<=1/2))$ | $x=\text{seb.fonction_D}(t);$
- L'équation (2.15) devient
 $x=(1-\text{np.abs}(t))*(\text{np.abs}(t)<=1)$ | $x=\text{seb.fonction_T}(t);$

2.4 Effet d'un retard/avance sur un signal

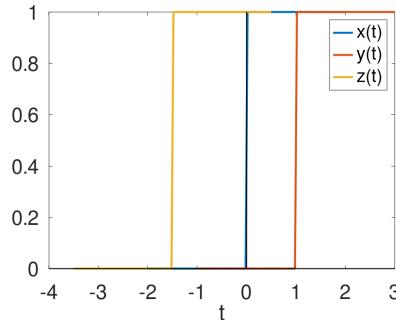


Figure 2.5: À gauche : visualisation en fonction du temps, de trois signaux $x(t) = \mathbb{H}(t)$ au centre, $y(t) = x(t - 1)$ à droite et $z(t) = x(t + 1.5)$ à gauche.

Cours signal et bruit :

On appelle avancer un signal le fait de transformer $x(t)$ en $x(t + t_0)$. On appelle retarder un signal le fait de transformer $x(t)$ en $x(t - t_0)$.

D'un point de vue électronique :

Introduire un retard ou une avance peut se voir comme retarder ou avancer le moment précis où l'expérience commence. Mais en général, ce qu'on appelle un retard est un décalage dans le temps entre deux signaux. On nomme les composants qui réalisent cette modification une ligne à retard. Il n'est pas physiquement possible de réaliser un composant qui ferait une avance, parce que cela amènerait à prédire l'avenir.

En terme de visualisation :

Lorsqu'on modifie un signal $y(t) = x(t - t_0)$, t_0 étant une valeur fixe, le graphe $t \mapsto x(t)$ est translaté vers la **droite** d'une valeur t_0 alors qu'il y a un signe moins. La gauche de la figure 2.5 illustre où la courbe du milieu est $x(t)$, celle de droite est $x(t - 1)$ et celle de gauche est $x(t + 1.5)$.

Il existe une façon simple de visualiser un signal retardé ou avancé, cela consiste à modifier l'échelle de temps en lui ajoutant ou en retranchant une valeur au vecteur.

Je considère un premier signal $x(t) = \mathbb{H}(t) = \llbracket t \geq 0 \rrbracket$. On obtient la gauche de la figure 2.5 avec l'algorithme 1. Celui-ci est implémentée dans ???. Remarquez que cette fois-ci le retard se code avec un signe plus.

Algorithm 1 générant la figure 2.5.

Créer une échelle de temps \mathbf{tx} entre -2 et 2 avec 100 points

Calculer \mathbf{x} un échelon associé à \mathbf{tx}

Calculer $\mathbf{ty}=\mathbf{tx}+1$

Calculer $\mathbf{tz}=\mathbf{tx}-1$

Visualiser $(\mathbf{tx}, \mathbf{x}, \mathbf{ty}, \mathbf{x}, \mathbf{tz}, \mathbf{x})$.

Python :

```
tx=np.linspace(-2,2,10**2)
x=seb.fonction_echelon(tx)
```

Je considère deux signaux $y(t)$ et $z(t)$, le premier en retardant $x(t)$ et le deuxième en l'avançant.

$$y(t) = x(t - 1) \text{ et } z(t) = x(t + 1.5) \quad (2.16)$$

Les trois signaux peuvent alors être ainsi représentés

```
ty=tx+1
tz=tx-1.5
fig,ax = plt.subplots()
ax.plot(tx,x,label='x(t)')
ax.plot(ty,y,label='y(t)')
ax.plot(tz,z,label='z(t)')
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('nom_figure.png')
fig.show()
```

On peut aussi simuler la gauche de la figure 2.5 en utilisant la même échelle de temps mais cette fois faisant une opération dessus avant de calculer les signaux correspondant.

Algorithm 2 générant aussi la figure 2.5.

Créer une échelle de temps \mathbf{t} entre -2 et 2 avec 100 points

Calculer \mathbf{x} un échelon associé à \mathbf{t}

Calculer \mathbf{y} un échelon associé à $\mathbf{t}-1$

Calculer \mathbf{z} un échelon associé à $\mathbf{t}+1.5$

Visualiser $(\mathbf{t}, \mathbf{x}, \mathbf{t}, \mathbf{y}, \mathbf{t}, \mathbf{z})$.

D'un point de vue mathématique :

$$x(t) = \llbracket t_1 \leq t \leq t_2 \rrbracket \Rightarrow x(t - t_0) = \llbracket t_0 + t_1 \leq t \leq t_0 + t_2 \rrbracket \quad (2.17)$$

En terme de visualisation :

On définit une fonction triangle notée $\mathbb{T}(t)$ en avançant $\mathbb{C}(t)$ et en retardant $\mathbb{D}(t)$

$$\begin{aligned}\mathbb{T}(t) &= \mathbb{C}(t + 0.5) + \mathbb{D}(t - 0.5) = ((t + 0.5) + 0.5) \llbracket |t + 0.5| \leq 0.5 \rrbracket(t) + (0.5 - (t - 0.5)) \llbracket |t - 0.5| \leq 0.5 \rrbracket(t) \\ &= (1 - |t|) \llbracket |t| \leq 1 \rrbracket(t)\end{aligned}\quad (2.18)$$

$\mathbb{T}(t)$ est représentée à gauche de la figure 2.6. Cette figure est implémentée dans A.1.

2.5 Effet d'une dilatation/concentration de l'échelle des abscisses

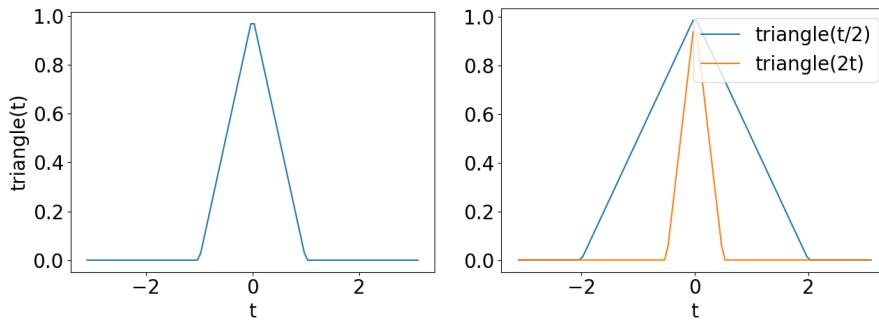


Figure 2.6: À gauche : fonction triangle $\mathbb{T}(t)$. À droite : superposition de $\mathbb{T}(t/2)$ à l'extérieur et de $\mathbb{T}(2t)$ à l'intérieur.

Cours signal et bruit :

On appelle dilater un signal $x(t)$ le fait de le transformer en $x\left(\frac{t}{a}\right)$ lorsque $a > 1$. On appelle concentrer un signal $x(t)$ le fait de le transformer en $x(at)$ lorsque $a > 1$.

En terme de visualisation :

Lorsqu'on modifie un signal $y(t) = x\left(\frac{t}{a}\right)$, $a > 0$ étant une valeur fixe, le graphe $t \mapsto x(t)$ est dilatée si $a > 1$ et concentrée si $a < 1$. La droite de la figure 2.6 montre à l'extérieur la fonction triangle dilatée $\mathbb{T}(t/2)$ et à l'intérieur la fonction triangle concentrée $\mathbb{T}(2t)$.

En termes numériques :

De la même façon que pour les algorithmes 1 et 2, on peut utiliser deux algorithmes différent pour générer le graphe à droite de la figure 2.6. Il importe de remarquer que la multiplication dans le premier algorithme devient une division dans le deuxième et vice-versa.

Algorithm 3 générant la figure 2.6 en modifiant les échelles de temps.

Créer une échelle de temps \mathbf{tx} entre -3.1 et 3.1 avec 100 points

Calculer \mathbf{x} un échelon associé à \mathbf{tx}

Calculer $\mathbf{ty}=2*\mathbf{tx}$

Calculer $\mathbf{tz}=\mathbf{tx}/2$

Visualiser à gauche (\mathbf{tx}, \mathbf{x})

Visualiser à droite ($\mathbf{ty}, \mathbf{x}, \mathbf{tz}, \mathbf{x}$)

D'un point de vue mathématique :

$$x(t) = \llbracket t_1 \leq t \leq t_2 \rrbracket \Rightarrow x\left(\frac{t}{a}\right) = \llbracket at_1 \leq t \leq at_2 \rrbracket \quad (2.19)$$

Algorithm 4 générant aussi la figure 2.6 en modifiant la façon de calculer les fonctions triangle.

Créer une échelle de temps t entre -3.1 et 3.1 avec 100 points

Calculer x avec fonction _T associé à t

Calculer y avec fonction _T associé à $t/2$

Calculer z avec fonction _T associé à $2t$

Visualiser à gauche (t, x)

Visualiser à droite (t, y, t, z)

2.6 Illustration sur les fonctions de base

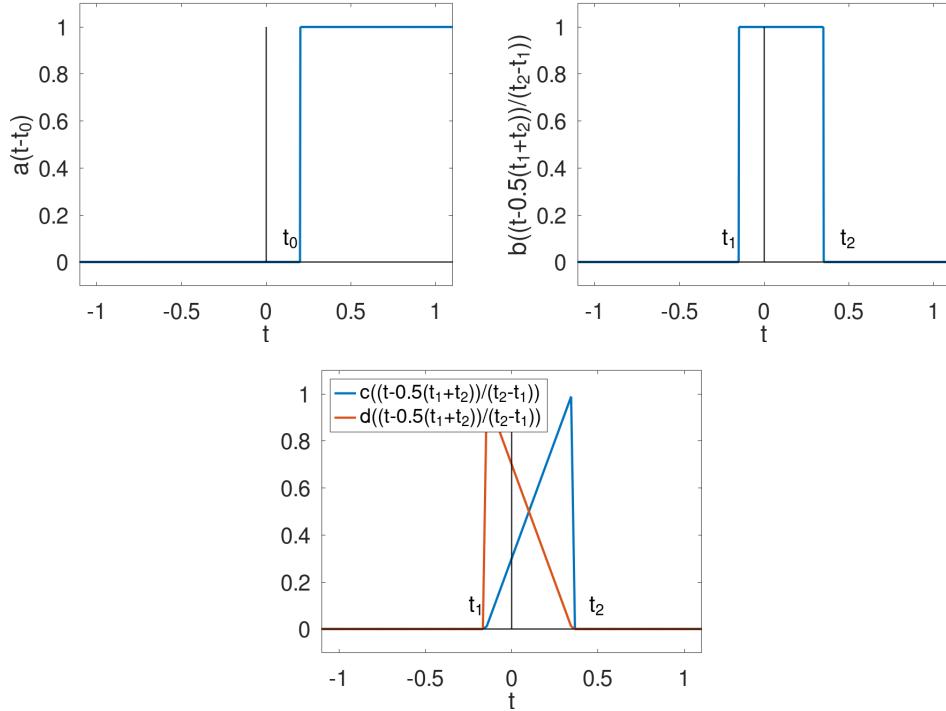


Figure 2.7: visualisation de $\mathbb{H}(t - t_0)$, $\Pi\left(\frac{t - (t_1 + t_2)/2}{t_2 - t_1}\right)$, $\mathbb{C}\left(\frac{t - (t_1 + t_2)/2}{t_2 - t_1}\right)$, $\mathbb{D}\left(\frac{t - (t_1 + t_2)/2}{t_2 - t_1}\right)$.

D'un point de vue mathématique :

Un intervalle $[t_1, t_2]$ décrit par son début t_1 et sa fin t_2 peut aussi être décrit par `centre` = $\frac{t_1+t_2}{2}$ et `longueur` = $t_2 - t_1$

$$t \in [t_1, t_2] \Leftrightarrow \frac{|t - \text{centre}|}{\text{longueur}} \leq 0.5 \quad (2.20)$$

Cours signal et bruit :

On peut adapter une fonction définie sur $[-0.5, 0.5]$ en une fonction définie sur $[t_1, t_2]$ en lui appliquant lui appliquant d'abord un retard de centre = $\frac{t_1+t_2}{2}$ puis une dilatation de longueur = $t_2 - t_1$. La figure 2.7 illustre les fonctions $\mathbb{H}(t)$, $\mathbb{C}(t)$ et $\mathbb{D}(t)$ adaptées à des nouveaux intervalles de temps.

En termes numériques :

De la même façon que pour les algorithmes 1 et 2, on peut utiliser deux algorithmes différent pour générer la droite de figure 2.7. Il importe de remarquer que l'addition, la multiplication et l'ordre sont échangés avec la soustraction, la division et l'ordre inverse.

D'un point de vue mathématique :

Pour trouver l'équation d'un signal à partir de sa courbe représentative définie comme des droites par morceaux, on découpe l'intervalle en une succession d'intervalles, de façon à ce que sur chacun de ces intervalles, la courbe soit un

Algorithm 5 générant la figure 2.7 en modifiant les échelles de temps.

Require: t0, t1,t2

Créer une échelle de temps t entre -1.1 et 1.1 avec 100 points

Calculer x un échelon associé à t

Calculer tx=t+t0

Visualiser à gauche (tx,x)

Calculer centre=(t1+t2)/2

Calculer longueur=t2-t1

Calculer y une porte associée à t

Calculer ty=tx*longueur+centre

Visualiser au milieu (ty,y)

Calculer z1 avec fonction_C associée à t

Calculer z2 avec fonction_D associée à t

Visualiser à droite (ty,z1,ty,z2)

Algorithm 6 générant aussi la figure 2.7 en modifiant la façon de calculer les fonctions.

Require: t0, t1,t2

Créer une échelle de temps t entre -1.1 et 1.1 avec 100 points

Calculer x un échelon associé à t-t0

Visualiser à gauche (t,x)

Calculer centre=(t1+t2)/2

Calculer longueur=t2-t1

Calculer y une porte associé à (t-centre)/longueur

Visualiser au milieu (t,y)

Calculer z1 avec fonction_C associé à (t-centre)/longueur

Calculer z2 avec fonction_D associé à (t-centre)/longueur

Visualiser à gauche (t,z1,t,z2)

portion de droite. Par ailleurs on sait que

$$x(t) = \frac{t - t_a}{t_b - t_a} (x_b - x_a) + x_a \quad (2.21)$$

est l'équation d'une droite vérifiant $x(t_a) = x_a$ et $x(t_b) = x_b$.

Par exemple un signal qui vaut 0 en $t = -1$, 1 en $t = 0$ et 0 en $t = 1$ et qui est obtenu en joignant ces différents segments a pour équation

- $x(t) = 0$ pour $t \leq -1$.
- $x(t) = t - 1$ pour $t \in]-1, 0]$
- $x(t) = 1 - t$ pour $t \in]0, 1]$
- $x(t) = 0$ pour $t > 1$.

Finalement

$$x(t) = (t - 1)\mathbf{1}_{]-1,0]}(t) + (1 - t)\mathbf{1}_{]0,1]}(t)$$

Chapter 3

Définition et utilisation de la transformée de Fourier

3.1 Somme, énergie, moyenne et puissance

Cours signal et bruit :

Les signaux déterministes non-périodiques à temps continu que l'on étudie dans ce cours, font en fait partie d'une catégorie appelée **[signaux à énergie finie et sommable]** que l'on résume en disant qu'ils sont non-périodiques.

D'un point de vue mathématique :

Ces signaux vérifient que $\int_{-\infty}^{+\infty} x^2(t) dt$ et $\int_{-\infty}^{+\infty} |x(t)| dt$. Ce n'est pas une condition que l'on ne va pas chercher à tester. Mais il y a des difficultés qui apparaissent quand cette condition n'est pas vérifiée et par exemple $\frac{1}{\sqrt{t}} \llbracket 0 < t \leq 1 \rrbracket(t)$ et $\frac{1}{\sqrt{t^2+1}} \llbracket t \geq 0 \rrbracket(t)$ ne la vérifient pas. En fait il est rare (mais possible) qu'à la fois le signal en temps $x(t)$ et sa transformée de Fourier vérifient ces conditions. Du coup le sens que l'on donne à la transformée de Fourier d'un signal est soit la transformée du signal si celui vérifie la condition soit d'être un signal égal à la transformée de Fourier inverse d'une autre fonction qui est d'énergie finie et sommable.

Somme

$$A_x = \int_{-\infty}^{+\infty} x(t) dt \quad (3.1)$$

C'est la cummulation de $x(t)$ au cours du temps, ici il y a une interprétation avec la transformée de Fourier quand le signal est non-périodique, $A_x = \hat{X}(0)$.

En particulier

$$x(t) = a \llbracket t \in [t_1, t_2] \rrbracket \Rightarrow A_x = a(t_2 - t_1) \quad (3.2)$$

Le retard ne modifie pas la somme.

$$y(t) = x(t - t_0) \Rightarrow A_y = A_x \quad (3.3)$$

La dilatation de l'échelle des abscisses modifie la somme.

$$y(t) = x\left(\frac{t}{a}\right) \Rightarrow A_y = aA_x \quad (3.4)$$

La **[moyenne temporelle]** est dans le cadre de ce cours défini comme nulle.

$$M_x = 0 \quad (3.5)$$

D'un point de vue mathématique :

On peut donner un sens à la moyenne temporelle pour des signaux non-périodiques, qui sort du cadre de ce cours

$$M_x = \lim_{T \rightarrow +\infty} \int_{-T/2}^{T/2} x(t) dt \quad (3.6)$$

et on peut démontrer que pour les signaux considérés cela vaut 0.

D'un point de vue électronique :

L'énergie et la puissance en traitement du signal sont définies avec $x^2(t)$ au lieu de pour chaque composant le produit de $u(t)$ par $i(t)$, le premier étant la tension aux bornes et le deuxième est l'intensité traversant.

Énergie

$$E_x = \int_{-\infty}^{+\infty} x^2(t) dt \quad (3.7)$$

En particulier

$$x(t) = a[t \in [t_1, t_2]] \Rightarrow E_x = a^2(t_2 - t_1) \quad (3.8)$$

Le retard ne modifie pas l'énergie

$$y(t) = x(t - t_0) \Rightarrow E_y = E_x \quad (3.9)$$

La dilatation de l'échelle des abscisses modifie l'énergie.

$$y(t) = x\left(\frac{t}{a}\right) \Rightarrow E_y = aE_x \quad (3.10)$$

La Puissance des signaux considérés est définie comme nulle.

$$P_x = 0 \quad (3.11)$$

D'un point de vue mathématique :

Ce sont les mêmes considérations que pour la somme qui permettent de définir

$$P_x = \lim_{T \rightarrow +\infty} \frac{1}{T} \int_{-T/2}^{T/2} x^2(t) dt \quad (3.12)$$

qui vaut 0 pour les signaux considérés.

En terme de visualisation :

Pour calculer la somme d'un signal d'énergie finie, il est possible d'utiliser les formules de géométrie classiques. Ainsi la somme de $x(t) = T(t)$ est la surface d'un triangle de base 2 et de hauteur 1, c'est-à-dire $A_x = \frac{2 \times 1}{2} = 1$.

Cours signal et bruit :

La transformée de Fourier en la fréquence nulle est la somme

$$\text{TF}[x(t)](0) = A_x \quad (3.13)$$

En appliquant cette formule aux signaux $x^2(t)$ on trouve que cela vaut l'énergie.

$$\text{TF}[x^2(t)](0) = E_x \quad (3.14)$$

En termes numériques :

Il est possible numériquement de trouver une approximation de A_x et E_x en utilisant la fonction proposée `TF` qui lorsqu'appliquée à la fréquence nulle fournit une simple intégrale.

Algorithm 7: calculant la somme et l'énergie de $x(t) = T(t)$.

Créer une échelle de temps `t` entre -3 et 3 avec 10000 points

Calculer `x` avec fonction `_T` associé à `t`

Afficher "la somme est" suivi de `TF` utilisant `t,x` pour `f=0`.

Afficher "l'énergie est" suivi de `TF` utilisant `t,x*x` pour `f=0`.

Cette expérience donne ici le résultat exact pour A_x et une approximation à 10^{-8} pour E_x .

Python :

On crée une échelle de temps avec

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

On génère la fonction triangle à l'aide de `fonction_T` défini dans `seb.py`

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

Le calcul de la A_x et E_x peut se faire avec les équations (3.13) et (3.14) en utilisant TF de `seb.py`. On rajoute la partie réelle parce qu'on sait que le résultat est réel. Cette partie réelle est implémentée avec `np.real`. On fait l'affichage avec `print` en mettant [entre accolades] l'expression à évaluer lors de l'affichage et en faisant précéder la chaîne de caractère à afficher de [la lettre f]. Les caractères `:.2f` signifient que l'affichage utilise 2 chiffres après la virgule. On peut utiliser aussi `:.2e` pour avoir un affichage avec un exposant.

```
print(f"Ax={np.real(seb.TF(t,x,0)):.2f} Ex={np.real(seb.TF(t,x**2,0)):.2f}")
```

3.2 Transformée de Fourier : définition

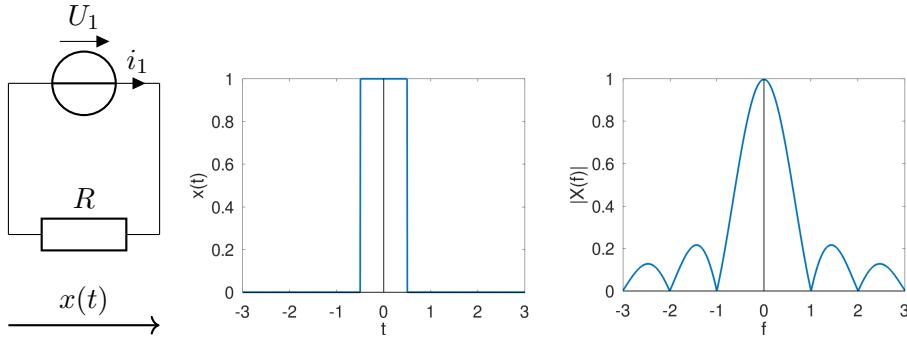


Figure 3.1: Au milieu fonction porte et à droite module de la transformée de Fourier de la fonction porte.

Cours signal et bruit :

Transformée de Fourier pour un signal temps continu non-périodique

$$\hat{X}(f) = \int_{-\infty}^{+\infty} x(t) e^{-j2\pi ft} dt \quad (3.15)$$

f étant une fréquence en Hertz (Hz).

D'un point de vue mathématique :

Cette définition a un sens lorsque $\int_{-\infty}^{+\infty} |x(t)| dt < +\infty$ Par exemple, elle n'a pas de sens si $x(t) = \llbracket t \geq 0 \rrbracket(t)$.

Cours signal et bruit :

Transformée de Fourier inverse d'un spectre défini sur toutes les fréquences et non-périodique

$$x(t) = \int_{-\infty}^{+\infty} \hat{X}(f) e^{+j2\pi ft} dt \quad (3.16)$$

Cette définition suppose que cette intégrale a un sens $\int_{-\infty}^{+\infty} |\hat{X}(t)| dt < +\infty$

Une deuxième définition de la transformée de Fourier inverse est que si on a pu calculer $\hat{X}(f)$ à partir de $x(t)$ alors par définition $\text{TF}^{-1}[\hat{X}(f)](t) = x(t)$.

De même si

En terme de visualisation :

La transformée de Fourier est un complexe, $f \mapsto \hat{X}(f)$ ne peut donc être représentée sur un graphique 2D car pour chaque valeur de fréquence il y a une partie réelle et une partie imaginaire. On représente généralement le module $|\hat{X}(f)|$ et parfois aussi la phase $\arg(\hat{X}(f))$.

Python :

Pour tracer la droite de la figure 3.1, on crée d'abord une échelle de temps

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

On utilise la fonction porte dans `seb.py` pour générer le signal $x(t)$

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

On crée une échelle de fréquence f

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

On calcule la transformée de Fourier notée $\hat{X}(f)$

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

On en déduit le module `np.abs(X)` que l'on trace en utilisant aussi `f`.

La transformée de Fourier inverse s'utilise de la même façon

```
x2=seb.TFI(f,X,t)
```

Attention il n'est pas possible de retrouver $x(t)$ en utilisant seulement le module de $\hat{X}(f)$

D'un point de vue mathématique :

Pour un complexe $z = a + jb$,

- $a = \operatorname{Re}(z)$ s'appelle la partie réelle.
- $b == \operatorname{Im}(z)$ s'appelle la partie imaginaire.
- $\sqrt{a^2 + b^2} = |z|$ s'appelle le module.
- $\arg(z)$ s'appelle l'argument de z , on $z = |z|e^{j\phi}$ et $a = |z|\cos(\arg(z))$ et $b = |z|\sin(\arg(z))$. La phase d'un nombre complexe nul n'est pas défini.
- $\bar{z} = a - jb$ s'appelle le conjugué de z .
- Si $b = 0$, alors z est un réel. Et si $a = 0$, alors z est un imaginaire pur.

Python :

On peut aussi en déduire l'argument `np.angle(X)` à partir du complexe X et donc la phase de la transformée de Fourier quand X est la transformée de Fourier de t, x . On obtient le conjugué d'un complexe `z` avec `z.conjugate()`

Cours signal et bruit :

La figure 3.1 montre à gauche un montage très simple avec une tension continue imposée par un générateur de tension pendant les instants $t \in [-0.5, 0.5]$. Au milieu c'est la visualisation du signal $x(t)$ et à droite c'est le module de la transformée de Fourier $|\hat{X}(f)|$. L'implémentation est en ??.

$$\hat{X}(f) = \int_{-\infty}^{+\infty} x(t)e^{-j2\pi ft} dt = \frac{\sin(\pi f)}{\pi f} = \operatorname{sinc}(f) \quad (3.17)$$

On définit ici $\operatorname{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$, mais attention, cette définition est celle plutôt utilisée dans la communauté des physiciens, celle des mathématiciens utilisent plutôt la définition $\operatorname{sinc}(x) = \frac{\sin(x)}{x}$. Ces fonctions sont appelées sinus cardinal.

Python :

On peut vérifier que Python utilise la même définition

```
assert np.abs(np.sinc(1))<1e-12
```

L'équation (3.1) permet de confirmer le résultat : $A_x = 1 = \hat{X}(0)$.

Cours signal et bruit :

On obtient le tableau suivant :

• $\text{TF}[e^{-t}[\![t \geq 0]\!]](f) = \frac{1}{1+j2\pi f}$	• $\text{TF}^{-1}\left[\frac{1}{1+j2\pi f}\right](t) = e^{-t}[\![t \geq 0]\!] = e^{-t}\mathbb{H}(t)$
• $\text{TF}[\![t \in [-0.5, 0.5]]\!](f) = \frac{\sin(\pi f)}{\pi f}$	• $\text{TF}^{-1}\left[\frac{\sin(\pi f)}{\pi f}\right](t) = [\![t \in [-0.5, 0.5]]\!] = e^{-t}\Pi(t)$

Table 3.1: de transformées de Fourier

Dans ces deux exemples, l'écriture intégrale de TF^{-1} n'a pas de sens.

3.3 Simuler numériquement des intégrales de fonctions non-sommables

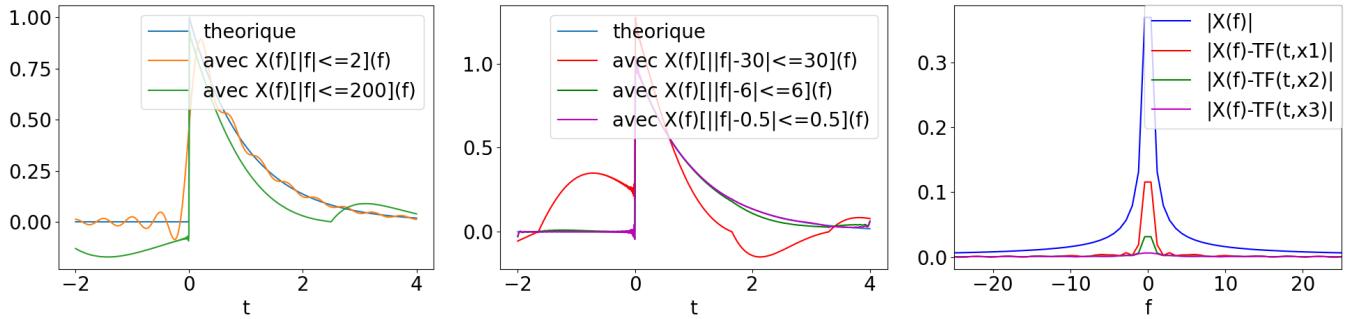


Figure 3.2: À gauche et au milieu approximation de e^{-t} . À droite visualisation en terme de spectre.

L'estimation de $\text{TF}^{-1}\left[\frac{1}{1+j2\pi f}\right](t)$ ou de $\text{TF}^{-1}\left[\frac{\sin(\pi f)}{\pi f}\right](t)$ pose le même genre de problème. Pour avoir une bonne précision, il faut à la fois décrire finement le spectre et considérer aussi des grandes fréquences parce que leur contribution n'est pas négligeable.

La gauche de la figure 3.2 montre en une courbe orange relativement oscillante estimant $x(t)$ en utilisant les valeurs de $\hat{X}(f) = \frac{1}{1+j2\pi f}$ en utilisant 1000 points entre -2 et 2Hz. La courbe verte avec des ondulations plus amples estime $x(t)$ en utilisant $\hat{X}(f)$ avec aussi 1000 points entre -200 et 200Hz. La courbe bleue donne le résultat théorique de $x(t) = [\![t \geq 0]\!]e^{-t}$. La première courbe n'est pas correcte car elle ne décrit pas correctement le comportement haute fréquence. La deuxième n'est pas correcte parce qu'elle n'utilise pas assez de points.

La droite de la figure 3.2 utilise des approximations successives du spectre. La courbe orange $x_3(t)$ est obtenue avec 100 points régulièrement répartis dans l'intervalle $[-42, 42]$. Cette approximation est moins bonne que les deux approximations montrées à gauche. C'est la différence entre $\hat{X} - \text{TF}[x_3(t)](f)$ qui est approchée encore avec 100 points sur l'intervalle $[-3.1, 3.1]$ et qui ajoutée à $x_3(t)$ donne une bonne approximation de $x(t)$.

L'implémentation de la figure 3.2 est dans A.3.

Et comme la formule de la transformée de Fourier inverse ressemble à celle de la transformée de Fourier on a de manière similaire à l'équation (3.13)

$$x(0) = \int_{-\infty}^{+\infty} \hat{X}(f) df \quad (3.18)$$

Algorithm 8 Algorithme de la figure 3.2

Créer l'échelle de temps t souhaitée

Créer une échelle de fréquence f très large

Initialiser

Calculer $\widehat{U}(f) = \text{TF}[u(t)](f)$

Calculer $\widehat{X}(f) = \frac{1}{j2\pi f RC} \widehat{U}(f)$

Calculer $x(t) = \text{TF}^{-1} [\widehat{X}(f)](f)$

Retrancher la première valeur de $x(t)$ au vecteur x

Chapter 4

Propriété de la transformée de Fourier

4.1 Propriété de la transformée de Fourier de la dilatation de l'échelle des temps

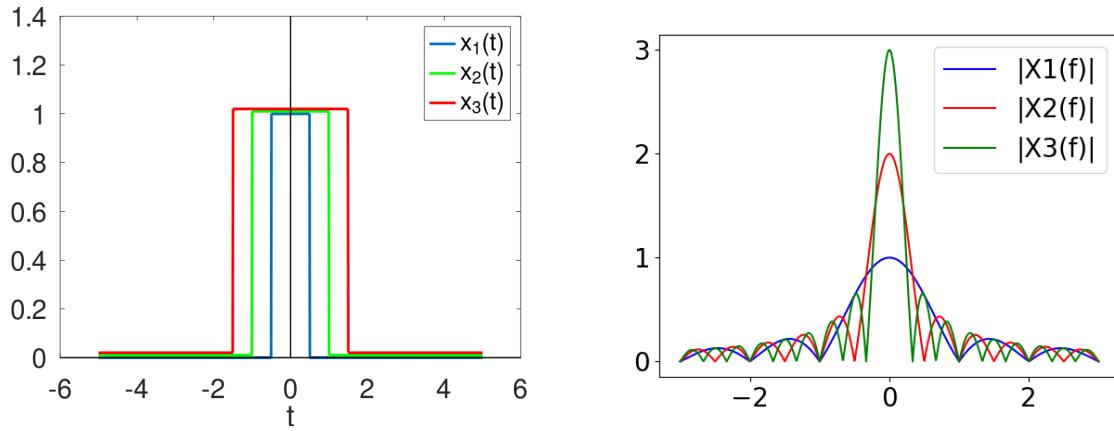


Figure 4.1: À gauche : fonction porte plus ou moins dilatée. À droite : module des transformées de Fourier correspondantes.

Cours signal et bruit :

Vis-à-vis de la dilatation de l'échelle des temps

$$y(t) = x\left(\frac{t}{a}\right) \Rightarrow \hat{Y}(f) = a\hat{X}(af) \quad (4.1)$$

Python :

Je n'ai pas réalisé de fonction qui réalisent une dilatation (ou une contraction) de l'échelle des temps. Par contre une façon d'implémenter la transformation du signal du fait de cette modification de l'échelle des temps peut se faire en faisant appel à une fonction qui calcule le signal x en fonction d'une échelle de temps.

J'illustre cette idée avec la fonction $\llbracket 0 \leq t \leq 1 \rrbracket(t)$ que j'appelle $x(t)$ et dont je cherche la dilatation par un facteur 3 : $y(t) = x(t/3)$. L'implémentation proposée est la suivante.

```
t = np.linspace(-1,4,500)
def x(t):
    return (0<=t)*(t<=1)+0
y = x(t/3)
```

Le signal y utilise l'échelle de temps t .

D'un point de vue mathématique :

L'équation (4.1) montre par exemple que s'il est possible de donner un sens à $\text{TF}[1](f)$, remarquant que la fonction constante $x(t) = 1$ reste inchangée lorsqu'on dilate l'échelle des temps $x(\frac{t}{a}) = x(t)$, la transformée de Fourier de $\text{TF}[1](f)$ doit rester identique à elle-même lorsqu'on la multiplie par n'importe quelle coefficient a .

En terme de visualisation :

La figure 4.1 est obtenue à gauche en introduisant une dilatation de l'échelle des temps à partir de la figure 3.1. D'une façon générale, une dilatation d'un signal amplifie le spectre et le concentre.

Cours signal et bruit :

$$x(t) = \Pi\left(\frac{t}{a}\right) \Rightarrow \hat{X}(f) = a \frac{\sin(\pi a f)}{\pi a f} = \frac{\sin(\pi a f)}{\pi f} = a \text{sinc}(af) \quad (4.2)$$

4.2 Propriété de la transformée de Fourier d'un retard ou d'une avance

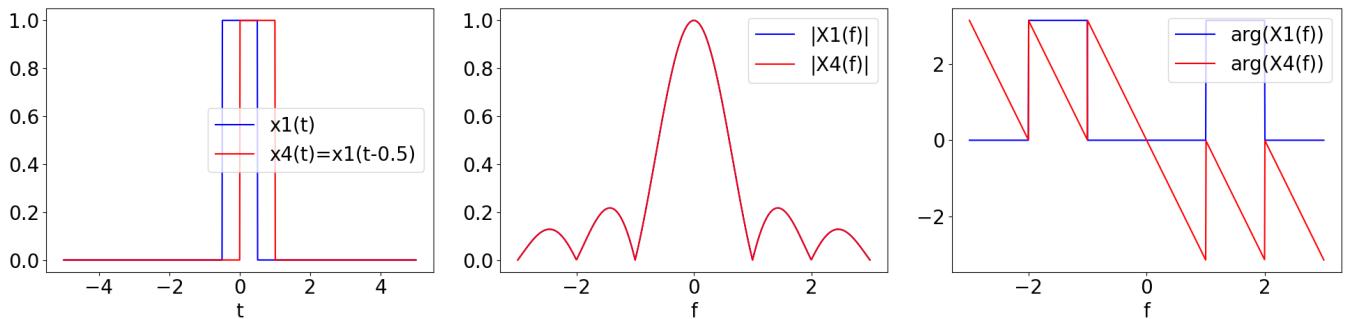


Figure 4.2: À gauche : Fonction porte sans retard et avec retard. Au milieu : module des transformées de Fourier correspondantes. À droite : argument des transformées de Fourier correspondantes (phase).

Cours signal et bruit :

La transformée de Fourier d'un signal retardé de t_0 a le même module et sa phase est diminuée par $-2\pi f t_0$.

$$y(t) = x(t - t_0) \Rightarrow \hat{Y}(f) = \hat{X}(f)e^{-j2\pi f t_0} \quad (4.3)$$

En terme de visualisation :

La figure 4.2 est obtenue à partir de la figure 3.1 en introduisant un retard. Les implémentations se trouvent dans A.2. Un retard ne modifie pas le module de la transformée de Fourier, en revanche il introduit un déphasage. Le graphe 3 de la figure 4.2 indique la phase associée à $\text{TF}[\Pi(t)](f)$. Il se trouve que $\text{TF}[\Pi(t)](f)$ est réelle soit positive dans ce cas la phase vaut 0 soit négative et dans ce cas sa phase vaut π . Ceci explique le signal carré bleu variant entre 0 et π . Du fait du retard, $\text{TF}[\Pi(t - 0.5)](f) = \text{TF}[\Pi(t)](f)e^{-j\pi f}$ a une phase égale à $0 - \pi f$ ou à $\pi - \pi f$. Dans les deux cas elle est décroissante avec une pente de $-\pi$, et des discontinuités permettant aussi de rester entre $-\pi$ et π .

D'un point de vue électronique :

Un signal retardé est aussi appelé un signal avec un retard de phase.

4.3 Symétrie et transformée de Fourier

En terme de visualisation :

Quand on représente le module de la transformée de Fourier d'un signal, le graphe est symétrique par rapport à l'axe des abscisses. C'est ce que l'on observe sur les figures 4.1 et 4.2.

D'un point de vue mathématique :

- Un signal pair est défini par

$$x(t) = x(-t) \quad (4.4)$$

Visuellement il est symétrique par rapport à l'axe des ordonnées.

- Un signal impair est défini par

$$x(t) = -x(-t) \quad (4.5)$$

Visuellement il a une symétrie centrale par rapport au point $(0, 0)$.

Cours signal et bruit :

Quand $x(t)$ est réel (ce qui est toujours le cas dans ce cours), on a

- $|\text{TF}[x(t)](f)|$ est pair
- $\arg(\text{TF}[x(t)](f))$ est impair

On peut rajouter que si $x(t)$ est pair, alors sa transformée de Fourier est réelle. Et si $x(t)$ est impair alors sa transformée de Fourier est imaginaire pur.

Chapter 5

Diracs

5.1 Introduction à la notion de distribution de Dirac

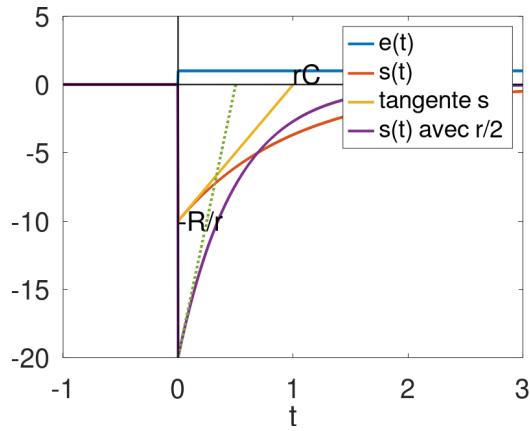
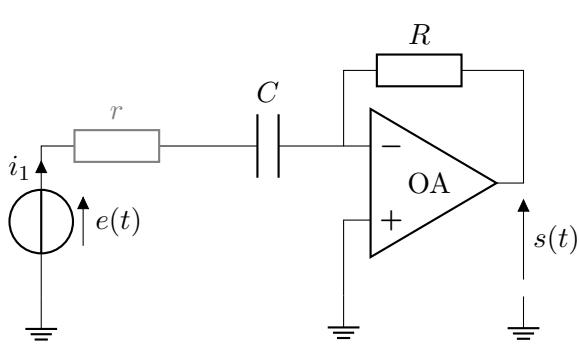


Figure 5.1: À gauche : montage dérivateur avec une résistance r faible et inconnue. À droite réponse dépendant de la résistance r et illustrée avec deux valeurs de r .

D'un point de vue électronique :

Le montage décrit à gauche de la figure 5.1 est un montage dérivateur alimenté par un générateur de tension continue. Ce montage décrit une saute de tension. À droite de la figure 5.1, $e(t)$ en rouge est la tension d'entrée, c'est un échelon. $s(t)$ en bleu est la tension de sortie, c'est un signal négatif en exponentiel et croissant. $\frac{ds}{dt}$ est la dérivée représentée par la courbe en verte. On peut noter que la surface dans le triangle est $\frac{1}{2} \frac{R}{r} rC = \frac{1}{2} RC$ et la surface sous l'exponentielle est RC . Les deux ne dépendent pas de r . On a ainsi une description de cette saute de tension qui ne dépend pas de r .

Ce qu'on appelle un Dirac est la limite théorique de cette saute de tension quand r tend vers 0, plus précisément on écrit que $s(t) = -RC\delta(t)$.

5.2 Règles de calcul pour le Dirac

Cours signal et bruit :

Intégration :

$$\int_{-\infty}^{+\infty} s(t)\delta(t) dt = s(0) \quad (5.1)$$

Dérivation d'une discontinuité en $t = 0$:

$$\frac{d}{dt} [t \geq 0] = \delta(t) \quad (5.2)$$

Multiplication par une fonction :

$$s(t)\delta(t) = s(0)\delta(t) \quad (5.3)$$

D'un point de vue mathématique :

En général la multiplication d'une fonction avec une distribution n'est pas définie.

D'un point de vue électronique :

Dans ce qui suit, je souhaite montrer comment grâce aux chapitres qui suivent, on peut faire différemment la description d'un montage électrique.

Le théorème de Millman appliqué au montage 5.1 en utilisant les impédances et les transformées de Fourier des signaux donne

$$\frac{\widehat{E}(f)}{\frac{1}{Cj2\pi f}} + \frac{\widehat{S}(f)}{R} = 0 \Rightarrow \widehat{S}(f) = -jRC2\pi f \widehat{E}(f) \quad (5.4)$$

En remplaçant la multiplication $j2\pi f$ par la dérivation

$$s(t) = -RC \frac{d}{dt} e(t) = -RC \frac{d}{dt} [\mathbb{I}[t \geq 0]] = -RC \delta(t) \quad (5.5)$$

À l'inverse on peut interpréter l'équation (5.4) avec une intégration

$$\widehat{E}(f) = -\frac{1}{jRC2\pi f} \widehat{S}(f) \Rightarrow e(t) = -\frac{1}{RC} \int_{-\infty}^t s(\tau) d\tau \quad (5.6)$$

En remplaçant $s(t)$ par $\delta(t)$ on obtient l'échelon pour $e(t)$.

$$e(t) = \int_{-\infty}^{+\infty} [\mathbb{I}[\tau \leq t]](\tau) \delta(\tau) d\tau = \begin{cases} 0 & \text{si } t < 0 \\ 1 & \text{si } t > 0 \end{cases} = [\mathbb{I}[t \geq 0]](t) \quad (5.7)$$

Cours signal et bruit :

La transformée de Fourier d'un Dirac est 1

$$\text{TF}[\delta(t)](f) = 1 \quad \text{TF}^{-1}[1](t) = \delta(t)$$

On rajoute cela à la table des transformations de Fourier 3.1.

Cours signal et bruit :

On définit aussi Dirac retardé ou avancé dont l'intégrale vaut :

$$\int_{-\infty}^{+\infty} s(t) \delta(t - t_0) dt = s(t_0) \text{ et } s(t) \delta(t - t_0) = s(t_0) \delta(t - t_0) \quad (5.8)$$

5.3 Règles de calcul pour dériver un signal discontinu

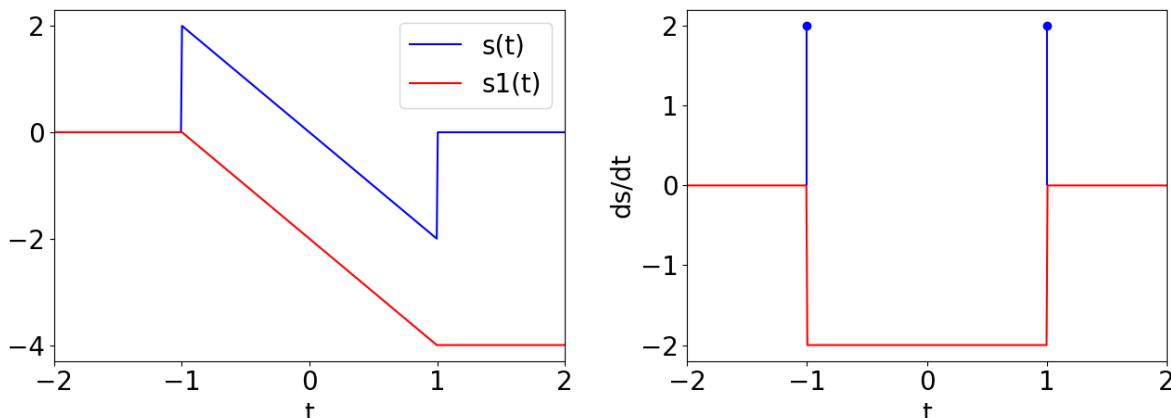


Figure 5.2: Exemple de signal $s(t)$ et de sa dérivée. La courbe rouge sur le premier graphe est $s(t)$ sans ses discontinuités. Celle sur le deuxième graphe est la dérivée de $s_1(t)$.

On définit la discontinuité d'un signal en un instant particulier t_0 par

$$s(t_0^+) - s(t_0^-) = \lim_{\substack{t \rightarrow t_0 \\ t > t_0}} s(t) - \lim_{\substack{t \rightarrow t_0 \\ t < t_0}} s(t) \quad (5.9)$$

La gauche de la figure 5.2 montre une discontinuité positive en $t = -1$ où la courbe passe de la valeur 0 à la valeur 2 et une discontinuité positive en $t = 1$ où la courbe passe de la valeur -2 à la valeur 0. On a les valeurs suivantes

$$s(-1^-) = 0, \quad s(-1^+) = 2, \quad s(1^-) = -2, \quad s(1^+) = 0 \quad (5.10)$$

D'un point de vue mathématique :

Dérivation d'une courbe discontinue en t_0, t_1, \dots

$$\frac{d}{dt}s(t) = \sum_n \delta(t - t_n)(s(t_n^+) - s(t_n^-)) + \frac{d}{dt}\tilde{s}(t) \quad (5.11)$$

où $\tilde{s}(t)$ est la courbe sans les discontinuités et $s(t_n^+) - s(t_n^-)$ est la discontinuité en t_n .

Cette partie du cours est différente du cours sur les équations différentielles n'utilisant pas le Dirac parce qu'un cours résolvant l'équation différentielle pour $t \geq 0$, n'a pas absolument besoin de dériver une discontinuité. Mais dans ce cours, les signaux sont supposés avoir un sens pour $t < 0$ et il y a donc forcément des discontinuités.

En terme de visualisation :

Graphiquement on représente un Dirac avec une flèche verticale orientée vers le haut si le signe est positif et vers le bas sinon. La longueur de la flèche dépend du coefficient devant le Dirac.

Le signal à gauche de la figure 5.2 est

$$s(t) = -2t[\mathbb{I}[t \in [-1, 1]]](t) = \begin{cases} 0 & \text{si } t < -1 \\ -2t & \text{si } t \in [-1, 1] \\ 0 & \text{si } t > 1 \end{cases} \quad (5.12)$$

Le signal a deux discontinuités une en $t = -1$ vers le haut d'une amplitude de 2 et une vers le haut en $t = 1$ d'une amplitude de 2. Sans considérer ces discontinuités la dérivée est de -2 pour $t \in [-1, 1]$ et 0 sinon. On obtient ainsi le signal représenté à droite de la figure 5.2.

$$\frac{ds}{dt} = -2\llbracket t \in [-1, 1] \rrbracket + 2\delta(t+1) + 2\delta(t-1) \quad (5.13)$$

À l'inverse si on considère le signal à droite de la figure 5.2, $x(t) = -2[t \in [-1, 1]] + 2\delta(t+1) + 2\delta(t-1)$, son intégrale vaut

$$y(t) = \int_{-\infty}^t x(\tau) d\tau = \begin{cases} 0 & +0 \quad +0 \quad \text{si } t < -1 \\ \int_{-1}^t -2 d\tau & +2 \quad +0 \quad \text{si } t \in]-1, 1[\\ -2 \times 2 & +2 \quad +2 \quad \text{si } t > 1 \end{cases} \quad (5.14)$$

$$= \begin{cases} 0 & \text{si } t < -1 \\ (-2t - 2) + 2 = -2t & \text{si } t \in]-1, 1[\\ 0 & \text{si } t > 1 \end{cases} = -2t \llbracket t \in [-1, 1] \rrbracket(t) = s(t)$$

qui est le signal à gauche de la figure 5.2.

Python :

L'évaluation du signal représenté à gauche sur la figure 5.2 s'implémente de façon similaire au crochet d'Iverson, une fois l'échelle de temps créée.

```
s=(t>=-1)*(t<=1)*(-2*t)
```

En ce qui concerne la droite de la figure 5.2, je note `dsdt` la dérivée du signal au sens classique, `t_` les instants associés aux Dirac et `dsdt_`, les pondérations des Diracs. `array` est une façon de définir un vecteur, elle est différente de `[-1,1]` qui est une liste de nombre et ne relève pas du module `numpy` notée ici `np`.

```
dsdt=(t>=-1)*(t<=1)*(-2);
t_,dsdt_=np.array([-1,1]),np.array([2,2])
```

Pour représenter les Diracs visible à droite de la figure 5.2, je propose d'utiliser `stem`. L'option `basefmt` dans `stem`, sert ici à supprimer l'axe horizontal.

```
fig,ax = plt.subplots()
ax.plot(t,dsdt,'b-')
ax.stem(t_,dsdt_,'b-',basefmt=" ")
```

`seb.py` donne un outil pour approcher la dérivée, mais cet outil suppose qu'il n'y a pas de discontinuité. Tout d'abord on détecte les discontinuités avec la fonction `diff` de `numpy` qui calcule la différence termes à termes d'un signal on cherche les indices pour la valeur absolue de cette différence dépasse un seuil. Ici le seuil est choisi à 0.1.

$$n \text{ est un indice de discontinuité si } |s_{n+1} - s_n| \geq \text{seuil} \quad (5.15)$$

```
ind = np.argwhere(abs(np.diff(s))>0.1)
```

On trouve ensuite les instants associés et les valeurs de cette différence en considérant d'une part `t` l'échelle en temps et d'autre part le signal `s`. En appelant \mathcal{I} l'ensemble des indices associés à l'équation (5.15) et t_n et $s(t_n)$ les instants et les valeurs du signal à ces instants. t_n est l'instant associé à la discontinuité et $s(t_{n+1}) - s(t_n)$ est la valeur de cette discontinuité.

$$\text{Pour } n \in \mathcal{I}, \quad t_n \text{ et } s(t_{n+1}) - s(t_n) \quad (5.16)$$

```
t_app_=(t[ind]+t[ind+1])/2
dsdt_app_=np.diff(s)[ind]
```

Pour chaque Dirac, on retranche au signal un échelon ayant une discontinuité au même instant et de la même longueur. Le signal sans discontinuités est ici noté `s1`.

$$s_{\text{sans discontinuités}}(t) = s(t) - \sum_{n \text{ discontinue}} (s(t_{n+1}) - s(t_n)) \mathbb{H}(t - t_n) \quad (5.17)$$

```
s_ech1 = dsdt_app_[0][0]*seb.retarder(t,seb.fonction_H(t),t_app_[0][0])
s_ech2 = dsdt_app_[1][0]*seb.retarder(t,seb.fonction_H(t),t_app_[1][0])
s1 = s-s_ech1-s_ech2
```

Pour obtenir la dérivée classique du signal $s(t)$, on applique la fonction `seb.deriver` à `s1`.

```
dsdt_app=seb.deriver(t,s1)
```

Le résultat obtenu contient un pic là où il y a une discontinuité.

L'implémentation de la figure 5.2 est dans A.4.

5.4 Condition initiale et Dirac

D'un point de vue électronique :

La notion de condition initiale est liée à la possibilité de décrire complètement l'état d'un système à un instant donné, c'est ce qu'on fait en automatique. Plus précisément choisir une condition initiale c'est considérer que la sortie du système ainsi qu'éventuellement certaines de ses dérivées décrivent complètement le système.

Cependant lorsqu'on ne connaît un système que par sa relation entrée-sortie, la description de l'état du système à un instant donné revient à imaginer une entrée passé. C'est ce point de vue qui est utilisé dans ce cours.

Cours signal et bruit :

Il n'y a pas de notion de condition initiale dans ce cours, cependant il est parfois possible rajouter un Dirac dans une relation entrée-sortie pour modéliser un comportement similaire au fait d'utiliser une condition initiale.

Exemple pour illustrer :

Ainsi dans l'équation différentielle

$$\frac{d}{dt}y(t) + y(t) = x(t) + y(0)\delta(t) \quad (5.18)$$

est équivalente à l'équation différentielle $\frac{d}{dt}y(t) + y(t) = x(t)$ pour $t \geq 0$ avec $y(0)$ non-nul.
En effet dans le cadre de ce cours, on vérifie que

$$y(t) = e^{-t} \int_{-\infty}^t e^{\tau} x(\tau) d\tau + y(0)[t \geq 0](t)e^{-t} \quad (5.19)$$

est bien solution de l'équation (5.18) :

$$\frac{d}{dt} \left[e^{-t} \int_{-\infty}^t e^{\tau} x(\tau) d\tau + y(0)[t \geq 0](t)e^{-t} \right] = -e^{-t} \int_{-\infty}^t e^{\tau} x(\tau) d\tau + e^{-t} e^t x(t) + y(0)\delta(t) - y(0)e^{-t}[t \geq 0](t) \quad (5.20)$$

Et dans le cadre du cours sur la résolution des équations différentielles, on a d'une part une solution générale $y(t) = Ae^{-t}$ pour $t \geq 0$ qui pour coïncider avec la valeur en $t = 0$ amène à $A = y(0)$. On a aussi une solution particulière obtenue en remplaçant A par une fonction $A(t)$. Constatant que

$$\frac{d}{dt}y(t) + y(t) = \left(\frac{d}{dt}A(t) \right) e^{-t} - A(t)e^{-t} + A(t)e^{-t} = x(t) \quad (5.21)$$

on définit

$$A(t) = \int_0^t e^{\tau} x(\tau) d\tau \quad (5.22)$$

On obtient ainsi une expression identique pour $t \geq 0$

$$y(t) = e^{-t} \int_0^t e^{\tau} x(\tau) d\tau + y(0)e^{-t} \quad (5.23)$$

5.5 Dérivées d'un Dirac

Cours signal et bruit :

On définit $\delta'(t)$ et par suite les dérivées successives de $\delta(t)$. Mais dans le cadre de ce cours, on n'utilise que leur définition en tant que dérivée successive de $\delta(t)$.

D'un point de vue mathématique :

Les distributions dérivées de $\delta(t)$ ont des propriétés surprenantes, ainsi

$$\int_{-\infty}^{+\infty} s(t)\delta'(t-t_0) dt \neq s'(t_0) \text{ et } s(t)\delta'(t-t_0) \neq s'(t_0)\delta(t-t_0)$$

Chapter 6

Transformées de Fourier et dérivation

6.1 Introduction

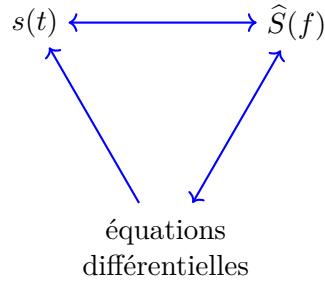


Figure 6.1: Représentation schématique des transformations qu'il est possible d'effectuer parfois avec un calcul parfois avec une simulation

Cours signal et bruit :

La figure 6.1 donne trois façons de définir un signal, $s(t)$ indique la relation en fonction du temps (voir chapitre 2). $\hat{S}(f)$ indique la dépendance vis-à-vis de la fréquence (voir chapitre 3). Vous avez déjà eu un cours présentant les équations différentielles. Ce cours montrait aussi la façon analytiquement de trouver $s(t)$ à partir de ces équations différentielles rappelée très brièvement en section 7.1 pour le type d'équations différentielles qui nous intéresse ici, c'est la flèche diagonale contre-oblique¹. Numériquement il est aussi possible de simuler les solutions des équations différentielles, ceci est présenté en section 7.2. Enfin pour une certaine catégorie de spectres, il est très simple d'identifier équations différentielles et spectres, ceci est présenté en section 6.3, ceci correspond à la flèche diagonale oblique²

D'un point de vue mathématique :

Calculs de dérivées

La dérivée d'une sinusoïde est

$$\frac{d}{dt} \cos(2\pi f_0 t) = -2\pi f_0 \sin(2\pi f_0 t) \text{ et } \frac{d}{dt} \sin(2\pi f_0 t) = 2\pi f_0 \cos(2\pi f_0 t) \quad (6.1)$$

La dérivée d'une exponentielle et d'une gaussienne est

$$\frac{d}{dt} e^{\alpha t} = \alpha e^{\alpha t} \text{ et } \frac{d}{dt} e^{\alpha t^2} = 2\alpha t e^{\alpha t^2} \quad (6.2)$$

La dérivée d'un produit de fonctions est donnée par

$$\frac{d}{dt} (x(t)y(t)) = x(t) \frac{d}{dt} y(t) + y(t) \frac{d}{dt} x(t) \quad (6.3)$$

¹direction correspondant à \

²direction correspondant à /.

6.2 Propriété de la transformée de Fourier vis-à-vis de la dérivation

Cours signal et bruit :

Une propriété importante des transformées de Fourier est de transformer une dérivation en produit par $j2\pi f$.

$$y(t) = \frac{d}{dt}x(t) \Leftrightarrow \widehat{Y}(f) = \widehat{X}(f) j2\pi f \quad (6.4)$$

De même $\frac{1}{j2\pi f}$ correspond à un intégration.

Dans la mesure où la dérivée d'un échelon est un Dirac, on aboutit à une propriété de la transformée de Fourier de l'échelon :

$$j2\pi f \text{TF} [\mathbb{H}(t)](f) = \text{TF} \left[\frac{d}{dt} \mathbb{H}(t) \right] (f) = \text{TF} [\delta(t)](f) = 1 \quad (6.5)$$

La transformée de Fourier de l'échelon n'est pour autant pas définie.

On rajoute à la table 3.1,

$j2\pi f \Leftrightarrow \frac{d}{dt}$	$\frac{1}{j2\pi f} \Leftrightarrow \int_{-\infty}^t d\tau$
$j2\pi f \text{TF} [\mathbb{H}(t)](f) = 1$	

Table 6.1: additionnelle de transformées de Fourier

D'un point de vue mathématique :

Formellement identifier un $j2\pi f \text{TF}[x(t)](f)$ avec $\text{TF} \left[\frac{d}{dt} x(t) \right] (f)$ pose de nombreux problèmes théoriques, le cadre généralement utilisé est la transformée de Laplace qui est définie seulement pour $t \geq 0$. Un autre cadre théorique est celui des distributions tempérées. La présentation qui est faite ici est donc une simplification qui sans surprise pose des problèmes numériques.

D'un point de vue électronique :

Les transformées de Fourier de $v(t)$ et $i(t)$ sont notées ici $\widehat{V}(f)$ et $\widehat{I}(f)$. L'impédance généralise la relation entre tension et résistance pour un condensateur et une bobine.

$$\widehat{V}(f) = Z(f) \widehat{I}(f) \quad (6.6)$$

Ainsi pour un condensateur,

$$\widehat{V}(f) = \frac{1}{C j2\pi f} \widehat{I}(f) \Leftrightarrow \frac{d}{dt} v(t) = \frac{1}{C} i(t) \quad (6.7)$$

et pour une bobine,

$$\widehat{V}(f) = L j2\pi f \widehat{I}(f) \Leftrightarrow v(t) = L \frac{d}{dt} i(t) \quad (6.8)$$

On peut utiliser les impédances et la notion de transformée de Fourier pour trouver la transformée de Fourier de signaux à partir de certains montages électroniques.

Exemple pour illustrer :

L'utilisation de la transformée de Fourier permet de retrouver l'équation (2.8) à partir du circuit 2.2 en considérant $u_1(t) = \llbracket t \geq 0 \rrbracket$.

- En utilisant la notion de diviseur de tension, on trouve

$$\widehat{X}(f) = \frac{\text{R}}{\text{R} + \frac{1}{j\text{C}2\pi f}} \widehat{U}_1(f) = \frac{j\text{RC}2\pi f}{j\text{RC}2\pi f + 1} \widehat{U}_1(f) \quad (6.9)$$

- D'un point de vue théorique, l'entrée vérifie $j2\pi f \widehat{U}_1(f) = 1$ d'après (6.5). Aussi

$$\widehat{X}(f) = \frac{\text{RC}}{j\text{RC}2\pi f + 1} = \frac{1}{j2\pi f + 1} \text{ en choisissant } \text{RC} = 1 \quad (6.10)$$

En utilisant la table 3.1, on obtient finalement

$$x(t) = \llbracket t \geq 0 \rrbracket e^{-t} \quad (6.11)$$

- D'un point de vue numérique la figure 3.2 montre qu'il est difficile, (mais apparemment possible) de trouver numériquement $x(t)$ à partir de $\widehat{X}(f)$. Remarquez que cette figure ressemble effectivement à la figure 2.2 obtenue avec l'équation (6.11).

6.2.1 Calculs d'une primitive

$F(t)$ est une primitive de $f(t)$ si

$$\frac{d}{dt} F(t) = f(t) \quad (6.12)$$

Pour certains signaux, on ne connaît pas la primitive. Pour d'autres signaux, il est simple de les calculer.

$$\begin{aligned} \frac{t^{n+1}}{n+1} &\text{ est la primitive de } t^n \\ \frac{1}{\alpha} e^{\alpha t} &\text{ est la primitive de } e^{\alpha t} \end{aligned} \quad (6.13)$$

6.3 Transformer la définition d'un spectre en une équation différentielle définissant un signal

D'un point de vue mathématique :

Tout inverse d'un polynôme de variable $j2\pi f$ est la transformée de Fourier d'une équation différentielle.
Considérons

$$\widehat{Y}(f) = \frac{1}{a_0 (j2\pi f)^N + a_1 (j2\pi f)^{N-1} + \dots + a_N} \quad (6.14)$$

$\widehat{Y}(f)$ vérifie aussi

$$a_0 (j2\pi f)^N \widehat{Y}(f) + a_1 (j2\pi f)^{N-1} \widehat{Y}(f) + \dots + a_N \widehat{Y}(f) = 1 \quad (6.15)$$

On remplace chaque occurrence de $j2\pi f$ par $\frac{d}{dt}$ et 1 par $\delta(t)$ pour obtenir

$$a_0 \frac{d^N}{dt^N} y(t) + a_1 \frac{d^{N-1}}{dt^{N-1}} y(t) + \dots + a_N y(t) = \delta(t) \quad (6.16)$$

Plus généralement, quand on a, non pas juste l'inverse d'un polynômes en $j2\pi f$ mais un quotient de polynômes, il suffit d'abord d'interpréter l'inverse du dénominateur avec une équation différentielle et ensuite considérer le numérateur pour en déduire une deuxième relation. Considérant

$$\widehat{Y}(f) = \frac{b_0 (j2\pi f)^M + \dots + b_M}{a_0 (j2\pi f)^N + a_1 (j2\pi f)^{N-1} + \dots + a_N} \quad (6.17)$$

On pose $\widehat{Y}_1(f) = \frac{1}{a_0 (j2\pi f)^N + a_1 (j2\pi f)^{N-1} + \dots + a_N}$ de sorte que $\widehat{Y}(f) = (b_0 (j2\pi f)^M + \dots + b_M) \widehat{Y}_1(f)$. Cela conduit à

$$\begin{cases} a_0 \frac{d^N}{dt^N} y_1(t) + a_1 \frac{d^{N-1}}{dt^{N-1}} y_1(t) + \dots + a_N y_1(t) = \delta(t) \\ y(t) = b_0 \frac{d^M}{dt^M} y_1(t) + \dots + b_M y_1(t) \end{cases} \quad (6.18)$$

Exemple pour illustrer :

L'équation (6.9) permet d'en déduire une équation différentielle en posant d'abord $y_1(t)$ tel que $y(t) = RC \frac{d}{dt} y_1(t)$ ce qui amène à

$$\begin{cases} \widehat{Y}(f) = RC j2\pi f \widehat{Y}_1(f) \\ \widehat{Y}_1(f) = \frac{1}{j2\pi f RC + 1} \end{cases} \quad (6.19)$$

Et finalement on obtient

$$\begin{cases} y(t) = RC \frac{d}{dt} y_1(t) \\ RC \frac{d}{dt} y_1(t) + y_1(t) = \delta(t) \end{cases} \quad (6.20)$$

Chapter 7

Équations différentielles

7.1 Bref rappel pour résoudre analytiquement des équations différentielles à coefficients constant

Cours signal et bruit :

Considérant une équation différentielle définie par

$$a_0 \frac{d^N}{dt^N} y(t) + a_1 \frac{d^{N-1}}{dt^{N-1}} y(t) + \dots + a_N y(t) = \delta(t) \quad (7.1)$$

On définit le polynôme de degré N

$$Q(p) = a_0 p^N + a_1 p^{N-1} + \dots + a_0 \quad (7.2)$$

On cherche les N racines de ce polynôme. On appelle racine les complexes $z_0 \dots z_{N-1}$ qui vérifient $Q(z_n) = 0$. On appelle degré d'un polynôme, l'exposant le plus élevé utilisé pour définir le polynôme.

La solution de l'équation différentielle est alors de la forme

$$y(t) = \sum_{n=0}^{N-1} c_n e^{jz_n t} \llbracket t \geq 0 \rrbracket(t) \quad (7.3)$$

où c_0, c_1, \dots, c_{N-1} sont N complexes à déterminer de façon que l'équation différentielle soit juste.

En général $y(t)$ ou certaines de ses dérivées présente des discontinuités et il convient d'appliquer le cours de la section 5.3. Parfois à force de dériver on se retrouve à dériver un Dirac, mais il convient de ne pas faire de calculs avec ces dérivées de Dirac, il suffit de les constater et d'en déduire que les coefficients devant sont nuls.

D'un point de vue mathématique :

Cette présentation simplifiée masque quelques difficultés qui ne font pas l'objet du cours.

- Il n'y a pas de formules générales pour trouver les racines quand Q est un polynôme de degré élevé.
- Comme le polynôme est en fait à valeurs réelles, les racines quand elles ne sont pas réelles, elles vont par paires, l'une étant le conjugué de l'autre.
- Quand il y a moins de racines que le degré du polynôme Q , certaines racines ont une multiplicité plus grande que 1. La multiplicité est un indice entier strictement positif associé à chaque racine z_m notée ν_m . Elle est définie par le fait qu'on peut alors écrire $Q(p)$ sous la forme

$$Q(p) = C(p - z_0)^{\nu_0}(p - z_1)^{\nu_1} \dots (p - z_M)^{\nu_M} \text{ avec } \sum_{m=0}^{M-1} \nu_m = N \text{ et } C \text{ est une constante} \quad (7.4)$$

Dans ce cas l'équation (7.3) doit alors être modifiée, au lieu de c_n on met $C_m(t)$ un polynôme de degré $\nu_m - 1$ à déterminer.

Exemple pour illustrer :

Considérons l'équation différentielle

$$\frac{d^2}{dt^2}y(t) + y(t) = \delta(t) \quad (7.5)$$

Le polynôme associé est $Q(p) = p^2 + 1$ dont les racines sont $p = j$ et $p = -j$, ce sont des racines simples ($Q(p) = (p+j)(p-j)$). Donc il existe A et B tel que

$$y(t) = (Ae^{jt} + Be^{-jt})\llbracket t \geq 0 \rrbracket(t) \quad (7.6)$$

Cette courbe a une discontinuité en $t = 0$, aussi

$$\frac{d}{dt}y(t) = (A + B)\delta(t) + (jAe^{jt} - jBe^{-jt})\llbracket t \geq 0 \rrbracket(t) \quad (7.7)$$

$\frac{d}{dt}y(t)$ a une discontinuité en $t = 0$ qui vaut $jA - jB$.

$$\frac{d^2}{dt^2}y(t) = (A + B)\delta'(t) + (jA - jB)\delta(t) + (j^2 Ae^{jt} + j^2 Be^{-jt})\llbracket t \geq 0 \rrbracket(t) \quad (7.8)$$

J'annule $A + B$ parce que regroupant les termes devant $\delta'(t)$. En regroupant les autres termes, je trouve

$$\delta(t) = \frac{d^2}{dt^2}y(t) + y(t) = (jA - jB)\delta(t) \quad (7.9)$$

De sorte que A et B vérifie

$$\begin{cases} A + B = 0 \\ jA - jB = 1 \end{cases} \quad (7.10)$$

La solution du système est $A = -j/2 = \frac{1}{2j}$ et $B = -\frac{1}{2j}$ et donc finalement

$$y(t) = \sin(t)\llbracket t \geq 0 \rrbracket(t) \quad (7.11)$$

7.2 Simulation numérique des solutions des équations différentielles

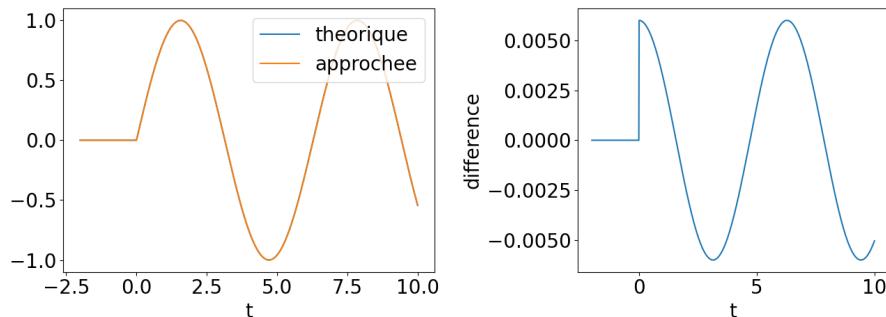


Figure 7.1: À gauche : courbe théorique et expérimentales. À droite : différences entre les deux courbes

Python :

Le module `seb.py` contient un programme `sol_eq_diff` qui permet de simuler numériquement la solution de l'équation différentielle. La première étape est de créer un vecteur contenant les différents instants où l'on veut calculer la solution.

```
t=np.linspace(t_debut,t_final,nombre_de_points)
```

Puis on définit `coef`, un tuple contenant les coefficients du polynôme

$$\text{coef} = (a_0, a_1, \dots, a_N)$$

Enfin on calcule le vecteur `y` contenant les valeurs prises par $y(t)$ aux instants indiqués par `t`.

```
y=seb.sol_eq_diff(coef,t)
```

Exemple pour illustrer :

L'implémentation se fait en utilisant `sol_eq_diff` fourni dans `seb.py`. Les coefficients de l'équations différentielles (7.5) sont 1, 0, 1.

```
t=np.linspace(-2,10,10**3)
y=seb.sol_eq_diff((1,0,1),t)
```

La figure 7.1 présente à gauche la solution théorique et approchée numériquement. À droite, il s'agit de la différence entre ces deux courbes. L'implémentation est faite dans A.12.

Chapter 8

Filtres et effet mémoire

8.1 Notion de filtre et de relation entrée-sortie

Cours signal et bruit :

Généralement l'entrée est notée $x(t)$ (ou $e(t)$) et la sortie est $y(t)$ (ou $s(t)$). On note les sorties associées à un signal

$$y(t) = \mathcal{H}[x(t)](t) \quad (8.1)$$

D'un point de vue électronique :

Un filtre généralise la relation entrée-sortie à des signaux dépendant du temps. Les figures 2.1, 2.2 et 2.3 peuvent se voir comme des filtres où l'entrée est $U(t)$ et la sortie est $x(t)$. Mais on pourrait aussi considérer que l'entrée est une intensité et la sortie pourrait aussi être une sortie. Les formules en électronique pour déterminer la tension ou l'intensité n'utilisent pas cette notion d'entrée-sortie par exemple pour les quadripôles, cela peut être l'entrée qui s'exprime en fonction de la sortie (cf : annexe B.4, page 118), parce que l'intention est de faire du calcul. En revanche il est fréquent que pour un circuit donné, on ne puisse intervertir entrée et sortie, par exemple on ne peut imposer la tension de sortie d'un amplificateur opérationnel et lire la tension à son entrée.

Cours signal et bruit :

La définition d'un filtre linéaire est plus complexe que l'équation (1.3) pour une relation entre deux grandeurs. Un filtre est linéaire si et seulement si étant donnée α , β , $x_1(t)$ et $x_2(t)$ on a

$$\boxed{x(t) = \alpha x_1(t) + \beta x_2(t) \Rightarrow y(t) = \alpha y_1(t) + \beta y_2(t)} \quad (8.2)$$

où $y_1(t)$, $y_2(t)$ et $y(t)$ sont les sorties associées à $x_1(t)$, $x_2(t)$ et $x(t)$.

On peut écrire l'équation (8.2) ainsi

$$\mathcal{H}[\alpha x_1(t) + \beta x_2](t) = \alpha \mathcal{H}[x_1(t)](t) + \beta \mathcal{H}[x_2(t)](t) \quad (8.3)$$

D'un point de vue électronique :

La question de la linéarité se pose peu dans ce cours parce que les exemples considérés sont le plus souvent linéaires. Ce cours ne s'applique que pour des systèmes linéaires. Et dans la pratique il y a de nombreux systèmes pour lesquels le fonctionnement linéaire est en fait une approximation.

Python :

On peut utiliser les simulations numériques pour vérifier si un système est linéaire. Considérons un filtre défini par $\mathcal{H}_1[x(t)](t) = \int_{-\infty}^t x(\tau) d\tau$ et un deuxième filtre défini par $\mathcal{H}_2[x(t)](t) = \int_{-\infty}^t x^2(\tau) d\tau$ qui sont implémentés par ces deux fonctions

```
def H1(t,x):
    return seb.integrer(t,x)
```

```
def H2(t,x):
    return H1(t,x**2)
```

On génère aléatoirement α et β suivant une loi gaussienne de moyenne nulle et d'écart-type 1.

```
alpha, beta = np.random.normal(0,1,2)
```

On considère une échelle de temps t , une fonction porte et une fonction triangle en entrée et on vérifie que l'équation (8.2).

```
t = np.linspace(-3,3,10**3)
x1, x2 = seb.fonction_P(t), seb.fonction_T(t)
assert all(np.abs(H1(t,alpha*x1+beta*x2)-alpha*H1(t,x1)-beta*H1(t,x2))<1e-8)
```

Quand on applique le même test à $H2$, cela ne fonctionne plus.

```
assert all(np.abs(H2(t,alpha*x1+beta*x2)-alpha*H2(t,x1)-beta*H2(t,x2))<1e-8)
```

8.2 Présence d'un effet mémoire dans la relation entrée-sortie

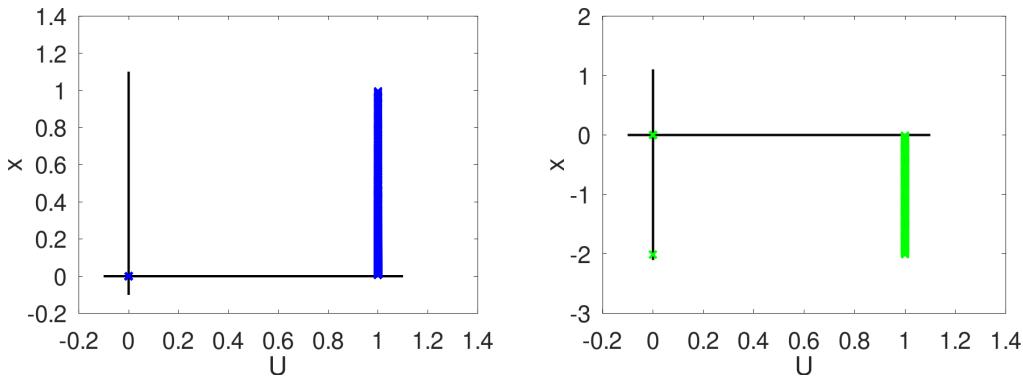


Figure 8.1: Graphe visualisant la sortie $x(t)$ en fonction de l'entrée $u(t)$ pour les montages 2.2 et 2.3.

La figure 8.1 représente les graphes des valeurs des sorties en fonction des valeurs des entrées pour les montages décrits par les figures 2.3 et 2.2. Ce qui montre qu'il y a un effet mémoire, c'est que ce n'est pas une courbe au sens où pour chaque position sur l'axe des abscisses, il n'y a pas qu'une unique position sur l'axe des ordonnées.

8.3 Filtre temps invariant

Cours signal et bruit :

On dit qu'un filtre est **temps invariant** lorsqu'une entrée retardée produit une sortie retardée.

$$x_2(t) = x_1(t - t_0) \Rightarrow y_2(t) = y_1(t - t_0) \quad (8.4)$$

Lorsqu'un filtre est **linéaire** et **temps invariant** alors on peut le décrire par son comportement fréquentiel et donc parler d'une réponse impulsionnelle et d'une réponse fréquentielle.

D'un point de vue électronique :

La propriété de temps-invariance est généralement vérifiée dans les montages électriques. Ce n'est pas le cas si par exemple des composants dépendent fortement de la température et que cette température est influencée par un phénomène extérieur.

Python :

On peut aussi tester par simulation pour chercher à savoir si un filtre est temps invariant. On considère ici deux filtres $\mathcal{H}_1[x(t)](t) = \int_{-\infty}^t x(\tau) d\tau$ et $\mathcal{H}_2[x(t)](t) = \int_{-\infty}^t \tau x(\tau) d\tau$ qui sont implémentées avec deux fonctions

```
def H1(t,x):
    return seb.integrer(t,x)
def H2(t,x):
    return H1(t,t*x)
```

On génère une échelle de temps et un signal, ici la fonction porte.

```
t = np.linspace(-3,3,10**3)
x = seb.fonction_P(t)
```

On retarde de τ où celui-ci est tiré aléatoirement suivant une loi uniforme

```
tau = np.random.uniform(-1,1)
```

Et on teste l'équation (8.4) pour \mathcal{H}_1 .

```
assert all(np.abs(H1(t,seb.retarder(t,x,tau))-seb.retarder(t,H1(t,x),tau))<1e-8)
```

En revanche cela ne fonctionne pas pour \mathcal{H}_2 .

```
assert all(np.abs(H2(t,seb.retarder(t,x,tau))-seb.retarder(t,H2(t,x),tau))<1e-8)
```

8.4 Produit de convolution

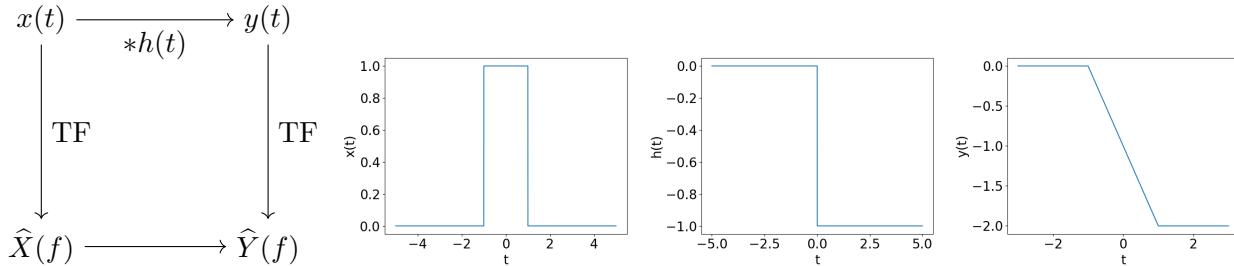


Figure 8.2: À gauche : diagramme filtre et réponse fréquentielle. À droite : simulation d'un filtre déjà illustré sur la figure 2.3 en utilisant la flèche du milieu dans le sens inverse l'algorithme 9.

Une vidéo illustre ce qu'est le produit de convolution est sur <https://www.youtube.com/watch?v=9Hk-RAIz0aw>

Cours signal et bruit :

On définit un produit de convolution entre deux signaux par

$$z(t) = x(t) * y(t) = \int_{-\infty}^{+\infty} x(\tau)y(t - \tau) d\tau = \int_{-\infty}^{+\infty} x(t - \tau)y(\tau) d\tau \quad (8.5)$$

Cela donne aussi une deuxième façon d'écrire la relation entrée-sortie en utilisant la définition du produit de convolution (utilisation de l'équation (9.3)).

$$y(t) = h(t) * x(t) = \int_{-\infty}^{+\infty} h(t - \tau)x(\tau) d\tau \quad (8.6)$$

En termes numériques :

Algorithm 9: pour la figure 8.2

Créer une échelle de temps tx pour $x(t) = \llbracket t \in [-1, 1] \rrbracket(t)$

Créer une échelle de temps ty pour $y(t)$

Déterminer les valeurs de x avec tx

Déterminer les valeurs de h avec ty, sachant que $h(t) = \llbracket t \geq 0 \rrbracket$

Calculer le produit de convolution $y(t) = x(t) * h(t)$

En terme de visualisation :

La droite de la figure 8.2 montre le signal d'entrée et de sortie. L'implémentation est faite dans A.6.

Le produit de convolution est implémenté avec la fonction `seb.convolution`. Cette fonction utilise le premier signal avec son échelle de temps, le deuxième signal et son échelle de temps ainsi que l'échelle de temps du signal résultant. La particularité de cette fonction est qu'elle requiert que les échelles de temps aient toutes la même résolution, c'est-à-dire la même période d'échantillonnage. Ainsi si on dispose d'une certaine échelle de temps `tx` pour un signal `x`, qu'on a une fonction `h(t)` qui transforme une échelle de temps en la réponse impulsionnelle, et que cette réponse impulsionnelle est décrite correctement sur l'intervalle $[0, 10]$, on peut implémenter $y(t) = x(t) * h(t)$ avec les étapes suivantes.

- Récupération de la période d'échantillonnage de `tx`

```
Te = tx[1]-tx[0]
```

- Fabrication de l'échelle de temps pour $h(t)$

```
th = np.arange(0,10,Te)
```

- Calcul du produit de convolution

```
y=seb.convolution(tx,x,th,h(th),tx)
```

Python :

Quand on utilise la fonction `seb.convolution` il est important que toutes les échelles de temps partagent la même période d'échantillonnage T_e . On peut utiliser la fonction `seb.convolution` pour calculer une seule composante dans ce cas on met une unique valeur pour le dernier argument. Il est aussi souhaitable qu'elles soient synchronisées ou plus exactement que les composantes de toutes ces échelles de temps soient des multiples de T_e .

- La fonction `seb.convolution` modifie les échelles de temps pour qu'elles soient synchronisées. Mais ce faisant cela peut produire des petits problèmes.
- La fonction `seb.synchroniser` permet de transformer une échelle de temps de façon à ce qu'elle vérifie la propriété d'être synchronisée.
- Les fonctions `seb.arange` et `seb.linspace` sont des fonctions qui utilisent les mêmes paramètres que `np.arange` et `np.linspace` et produisent des échelles de temps synchronisées.

8.4.1 Réponse impulsionnelle

On a vu qu'un filtre linéaire et temps invariant s'écrit

$$y(t) = h(t) * x(t) \quad (8.7)$$

où $h(t)$ est la réponse impulsionnelle, $x(t)$ une entrée quelconque et $y(t)$ la sortie associée.

Observant que $h(t) * \delta(t) = h(t)$, on peut affirmer que si $x(t) = \delta(t)$ alors $y(t) = h(t)$. Ceci donne une façon de calculer la réponse impulsionnelle à partir d'une relation entrée sortie.

Exemple pour illustrer :

La relation entrée-sortie suivante

$$y(t) = \int_{t-1}^t x(\tau) d\tau \quad (8.8)$$

a une réponse impulsionnelle $h(t) = \llbracket t \in [0, 1] \rrbracket(t)$

Exemple pour illustrer :

La relation entrée-sortie suivante

$$\frac{d}{dt}y(t) + y(t) = x(t) \quad (8.9)$$

a une réponse impulsionale qui vérifie

$$\frac{d}{dt}h(t) + h(t) = \delta(t) \quad (8.10)$$

Une fois $h(t)$ calculée, ceci permet de réécrire l'équation différentielle sous une forme intégrale avec $y(t) = h(t) * x(t)$.

Chapter 9

Description fréquentielle des filtres

9.1 Transformée de Fourier et produit de convolution

Une propriété importante est ce qui est un produit de convolution dans le domaine temporel devient un simple produit dans le domaine fréquentiel

$$\hat{Z}(f) = \hat{X}(f)\hat{Y}(f) \quad (9.1)$$

Cela donne une relation entre la réponse impulsionnelle et la réponse fréquentielle

$$\hat{H}(f) = \text{TF}[h(t)](f) \quad (9.2)$$

9.2 Réponse fréquentielle

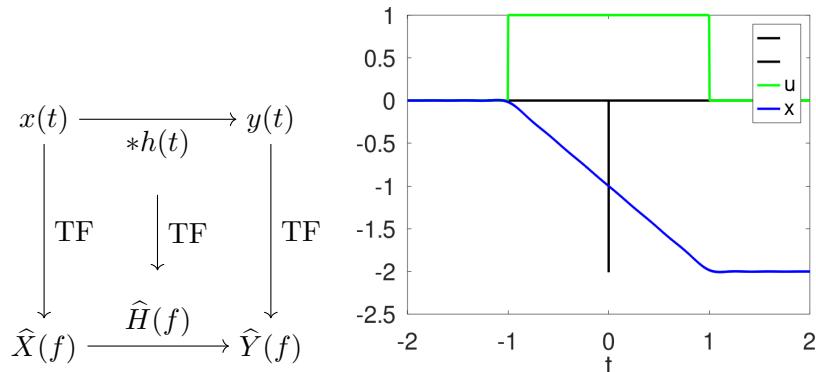


Figure 9.1: À gauche : diagramme filtre et réponse fréquentielle. À droite : simulation d'un filtre déjà illustré sur la figure 2.3 en utilisant l'algorithme 10 (p. 48).

Cours signal et bruit :

Comme illustré sur la figure 9.1, un filtre qui est linéaire et temps invariant peut se mettre sous la forme

$$\hat{Y}(f) = \hat{H}(f)\hat{X}(f) \quad (9.3)$$

$\hat{H}(f)$ est la réponse fréquentielle du filtre.

D'un point de vue électronique :

On peut voir la relation entre l'intensité qui le traverse et la tension autour du dipôle comme la relation entrée-sortie d'un filtre.

- Pour une résistance, $\hat{U}(f) = R\hat{I}(f)$ et $\hat{Z}(f) = R$ est la réponse fréquentielle.
- Pour un condensateur, $\hat{U}(f) = \frac{1}{jC2\pi f}\hat{I}(f)$ et $\hat{Z}(f) = \frac{1}{jC2\pi f}$ est la réponse fréquentielle.
- Pour un bobine, $\hat{U}(f) = jL2\pi f\hat{I}(f)$ et $\hat{Z}(f) = jL2\pi f$ est la réponse fréquentielle.

En termes numériques :

Je considère le montage présenté sur la figure 2.3 avec l'objectif de retrouver numériquement les courbes à droite en utilisant la réponse fréquentielle. Le calcul indiqué par le diagramme à gauche de la figure 9.1 est

$$x(t) = \text{TF}^{-1} \left[\frac{1}{j2\pi f RC} \text{TF}[u(t)](f) \right] (t) \quad (9.4)$$

La simulation utilise l'algorithme 10. Son implémentation est dans la figure ??.

Algorithm 10 Algorithme de la figure 9.1

Créer une échelle de temps t

Créer une échelle de fréquence f

Retirer la valeur 0 de l'échelle de fréquence.

Calculer $\hat{U}(f) = \text{TF}[u(t)](f)$

Calculer $\hat{X}(f) = \frac{1}{j2\pi f RC} \hat{U}(f)$

Calculer $x(t) = \text{TF}^{-1} [\hat{X}(f)](f)$

Retrancher la première valeur de $x(t)$ au vecteur x

9.3 Réponse impulsionale

Cours signal et bruit :

On peut définir la réponse impulsionale d'un filtre à partir de la réponse fréquentielle

$$h(t) = \text{TF}^{-1}[\hat{H}(f)](t) \quad (9.5)$$

Python :

On peut utiliser l'équation (9.5) pour justement simuler cette réponse impulsionale, (dans la mesure où intégrer ces réponse fréquentielle ne pose pas trop de problèmes).

Considérons l'exemple de $\hat{H}(f) = \frac{1}{1+j2\pi f}$.

On se donne tout d'abord une échelle en fréquence décrite finement

```
f = np.linspace(-30,30,10**4)
```

puis on calcule la réponse fréquentielle

```
H = 1/(1+1j*2*np.pi*f)
```

On se donne une échelle en temps pour faire de l'affichage

```
t = np.linspace(-1,3,10**2)
```

Et on calcule la réponse impulsionale en prenant la partie réelle de l'équation (9.5).

```
h = np.real(seb.TFI(f,H,t))
```

On mesure la précision de la simulation qui ici est de 0.04.

```
err = np.max(np.abs(h-np.exp(-t)*(t>=0)))
print(f"err max = {err}")
```

9.4 Filtres passe-bas, passe-haut et passe-bande

Cours signal et bruit :

À partir de la visualisation du graphe du module de la réponse fréquentielle, on définit trois types de filtres.

- Un **filtre passe-bas** a une réponse fréquentielle plus élevée sur les basses fréquences.
- Un **filtre passe-bande** a une réponse fréquentielle plus élevée sur des fréquences intermédiaires.
- Un **filtre passe-haut** a une réponse fréquentielle plus élevée sur des fréquences élevées.

Les filtres peuvent tout à fait n'être ni passe-bas, ni passe-bande, ni passe-haut.

D'un point de vue mathématique :

Le calcul des limites du module de la réponse fréquentielle en 0 ou en $+\infty$ peut servir d'indice pour dire si un filtre est susceptible d'être d'un de ces trois types.

9.5 Fréquence de coupure et bande passante d'un filtre

- Un filtre passe-bas vérifie a priori $\lim_{f \rightarrow 0} |\hat{H}(f)| \neq 0$ et $\lim_{f \rightarrow +\infty} |\hat{H}(f)| \approx 0$
- Un filtre passe-bande vérifie a priori $\lim_{f \rightarrow 0} |\hat{H}(f)| \approx 0$ et $\lim_{f \rightarrow +\infty} |\hat{H}(f)| \approx 0$
- Un filtre passe-haut vérifie a priori $\lim_{f \rightarrow 0} |\hat{H}(f)| \approx 0$ et $\lim_{f \rightarrow +\infty} |\hat{H}(f)| \neq 0$

Généralement une réponse fréquentielle se présente comme le quotient de deux polynômes en f à coefficients complexes, donc de cette forme-là

$$\hat{H}(f) = \frac{b_0 f^M + b_1 f^{M-1} + \dots + b_M}{a_0 f^N + a_1 f^{N-1} + \dots + a_N} \quad (9.6)$$

Pour trouver la limite lorsque f tend vers 0, il suffit de remplacer f par 0 si cela conduit à une division par zéro. Mais dans ce cas on met en facteur les termes en f de façon à pouvoir conclure.

Pour trouver la limite lorsque f tend vers $+\infty$, on ne peut pas procéder de la même façon parce que dans le domaine complexe il y a plein de façon de tendre vers l'infini, suivant l'argument du complexe. Par contre on met des termes en f en facteur de façon à ce que les autres termes tendent vers une constante.

Exemple pour illustrer :

Je considère la réponse fréquentielle $\hat{H}(f) = \frac{jf}{1-f^2+jf}$. La limite en $f = 0$ est

$$\lim_{f \rightarrow 0} |\hat{H}(f)| = \lim_{f \rightarrow 0} |f| \left| \frac{j}{1 - f^2 + jf} \right| = 0 \quad (9.7)$$

La limite en $f \rightarrow +\infty$ est

$$\lim_{f \rightarrow +\infty} |\hat{H}(f)| = \lim_{f \rightarrow +\infty} \left| \frac{1}{f} \right| \left| \frac{j}{\frac{1}{f^2} - 1 + \frac{j}{f}} \right| = 0 \quad (9.8)$$

Cours signal et bruit :

La fréquence de coupure est définie par rapport à la valeur maximale du module de la réponse fréquentielle. Dans ce cours, on note f_{\max} , la fréquence à laquelle le module est maximale.

$$\hat{H}(f_{\max}) = \max_f |\hat{H}(f)| \quad (9.9)$$

- Pour un filtre passe-bas, $f_{\max} = 0$ et il y a une fréquence de coupure f_1

$$|\hat{H}(f_1)| = \frac{\sqrt{2}}{2} |\hat{H}(0)| \quad (9.10)$$

- Pour un filtre passe-bande, $f_{\max} = 0$ et il y a deux fréquences de coupure f_1 et f_2

$$|\hat{H}(f_1)| = |\hat{H}(f_2)| = \frac{\sqrt{2}}{2} |\hat{H}(0)| \text{ et } f_1 \leq f_{\max} \leq f_2 \quad (9.11)$$

- Pour un filtre passe-haut, $f_{\max} = 0$ et il y a une fréquence de coupure f_1

$$|\hat{H}(f_1)| = \frac{\sqrt{2}}{2} \lim_{f \rightarrow +\infty} |\hat{H}(f)| \quad (9.12)$$

La **bande passante** d'un filtre est définie comme un intervalle $[f_1, f_2]$. Parfois on utilise le terme de bande passante pour désigner la longueur de cette intervalle : $f_2 - f_1$.

Le **Facteur de qualité**, Q est définie à partir de la bande passante et de fréquence maximale.

$$Q = \frac{f_{\max}}{f_2 - f_1} \quad (9.13)$$

Python :

Numériquement il est possible d'évaluer cette bande fréquence.

Considérons l'exemple de réponse fréquentielle

$$\hat{H}(f) = \frac{j2\pi f}{1 - 4\pi^2 f^2 + \frac{2}{10} j\pi f} \quad (9.14)$$

On cherche tout d'abord la fréquence donnant le module le plus élevé en créant une échelle de fréquence et en récupérant la valeur la plus élevée de ce module.

```
f = np.linspace(0,10,10**5)
H = 1j*2*np.pi*f/(1-4*(np.pi**2)*f*f+1j*2*np.pi*f/10)
ind_max = np.argmax(np.abs(H))
f_max, H_max = f[ind_max], H[ind_max]
print(f"f_max = {f_max} H_max = {H_max}")
```

Pour cet exemple il se trouve que $|\hat{H}(f)|$ est croissant entre 0 et f_{\max} et décroissant après

```
assert all(np.diff(np.abs(H[:ind_max]))>=0)
assert all(np.diff(np.abs(H[ind_max:])))<=0)
```

Pour trouver les fréquences f_1 et f_2 qui sont respectivement dans les intervalles $[0, f_{\max}]$ et $[f_{\max}, +\infty[$ et qui vérifient approximativement $|\hat{H}(f_1)| = |\hat{H}(f_2)| = \frac{\sqrt{2}}{2} |\hat{H}(f_{\max})|$, on peut utiliser la fonction `seb.where_nearest` qui trouve l'indice dans un vecteur dont la composante est la plus proche d'une valeur cible.

```
ind1 = seb.where_nearest(np.abs(H[:ind_max]), np.abs(H_max)/np.sqrt(2))
ind2 = ind_max + seb.where_nearest(np.abs(H[ind_max:]), np.abs(H_max)/np.sqrt(2))
```

Finalement on en déduit la bande de fréquence et le facteur de qualité.

```
B,Q = f[ind2]-f[ind1], f_max/(f[ind2]-f[ind1])
print(f"B={B}, Q={Q}")
```

9.6 Sens de variation de la réponse fréquentielle

D'un point de vue mathématique :

On appelle un **tableau de variation**, un tableau qui indique pour quelles valeurs le signal est croissant ou décroissant, en outre ce tableau indique les valeurs atteintes lorsque le signal change de sens de croissance et en moins l'infini et plus l'infini.

Le fait qu'un signal soit croissant ou décroissant peut être déterminé en calculant la dérivée du signal.

$$\begin{cases} \frac{d}{dt}x(t) \geq 0 & \Rightarrow x(t) \text{ est croissante} \\ \frac{d}{dt}x(t) \leq 0 & \Rightarrow x(t) \text{ est décroissante} \end{cases}$$

Une autre façon de déterminer un tableau de variation

Plutôt que de déterminer le signe de la dérivée d'un signal, on peut déduire le sens de variation d'un signal au moyen des règles suivantes ¹ :

- f et g sont des fonctions croissantes alors $f \circ g$ est croissante.
- f et g sont des fonctions décroissantes alors $f \circ g$ est croissante.
- f est croissante et g est décroissante alors $f \circ g$ est décroissante.
- f est décroissante et g est croissante alors $f \circ g$ est décroissante.

On peut y rajouter ces deux règles

- La somme de deux fonctions croissantes est croissante.
- La somme de deux fonctions décroissantes est décroissante.

Python :

Étant donné un signal défini par son échelle de temps t et les valeurs du signal x , on peut numériquement tester s'il est croissant : l'évaluation de l'instruction suivante est **True** si le signal est croissant.

```
all(np.diff(x)>=0)
```

Supposons qu'on sache que le signal est d'abord croissant puis décroissant, on peut trouver l'instant à partir duquel il est décroissant. On utilise pour cela `seb.vect(np.where(np.diff(x)<=0))` qui renvoie un vecteur contenant les indices pour lesquelles la condition `np.diff(x)<=0` est vraie. `seb.vect` permet de vérifier que ce qui est renvoyé est de type `numpy.array` ou de type `numpy.ndarray`, et si ce n'est pas le cas déréférence la première valeur (i.e. cherche la première composante de l'objet) et vérifie que ce dernier est bien du type souhaité.

```
t=np.linspace(0,3,10**3)
x=2-(t-1)**2
ind=seb.vect(np.where(np.diff(x)<=0))[0]
assert x[ind-1]<=x[ind]
assert x[ind]>=x[ind+1]
print(f" la fonction est croissante pour t<={t[ind]} et decroissante apres")
```

¹ $f \circ g$ est une notation signifiant $f \circ g](x) = f(g(x))$

Chapter 10

Signaux périodiques

10.1 Définition d'un signal périodique

Cours signal et bruit :

Un signal périodique est un signal qui se répète dans le temps

$$x(t) = x(t + T_x) \quad (10.1)$$

T_x est la période du signal. Sur un graphique du signal en fonction du temps, ce signal déplacé vers la gauche est identique à lui-même et on appelle la période la longueur du déplacement horizontal effectué.

D'un point de vue électronique :

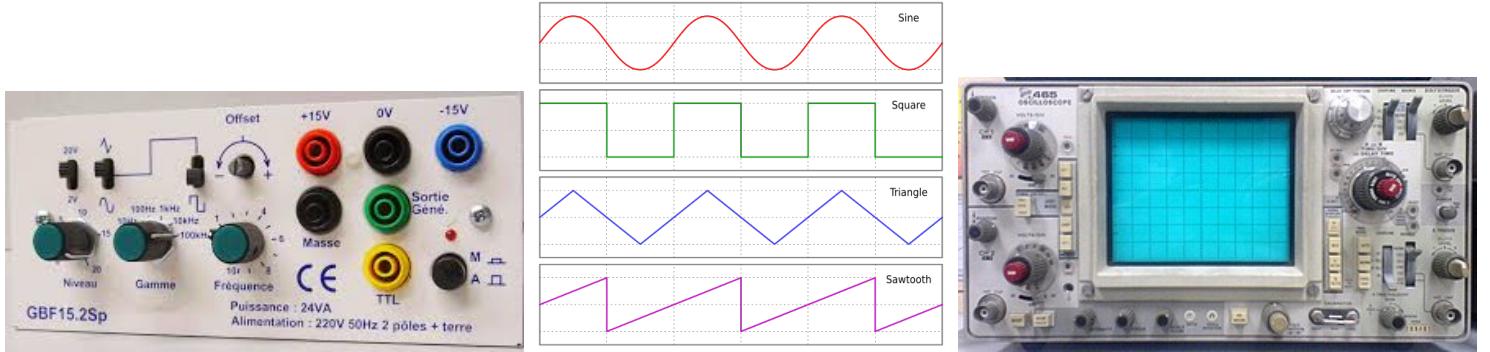


Figure 10.1: Générateur de basse fréquences, signaux et oscilloscope

La figure 10.1 montre à gauche un générateur de basses fréquences qui peut générer des signaux avec une tension qui varie de façon sinusoïdale, triangulaire ou carrée visualisés au milieu. Ces signaux sont périodiques. Il est en général possible de leur ajouter un signal constant et de modifier l'amplitude ou la période. Sur le GBF la fréquence désigne en fait l'inverse de cette période. L'oscilloscope permet de visualiser le graphe de la tension en fonction du temps. C'est adapté aux signaux périodiques, la valeur 0 du temps indiqué correspond non pas à $t = 0$ mais au dépassement d'une valeur seuil par la tension. Il n'est donc pas possible en général de lire φ dans $v(t) = \sin(2\pi f_0 t + \varphi)$ en visualisant un signal avec un oscilloscope.

Cours signal et bruit :

$$x(t) = \sin(2\pi f_0 t + \varphi) \text{ et } x(t) = \cos(2\pi f_0 t + \varphi) \text{ sont périodiques de période } T_x = \frac{1}{f_0} \quad (10.2)$$

On décrit un signal périodique en indiquant qu'il est périodique de période T et en décrivant ce signal sur l'intervalle $[0, T]$.

10.2 Somme, moyenne, énergie et puissance

Cours signal et bruit :

La somme est infinie, la moyenne se calcule sur une période.

$$S_x = +\infty \text{ et } M_x = \frac{1}{T_x} \int_0^{T_x} x(t) dt \quad (10.3)$$

L'énergie est infinie et la puissance se calcule sur une période

$$E_x = +\infty \text{ et } P_x = \frac{1}{T_x} \int_0^{T_x} x^2(t) dt \quad (10.4)$$

D'un point de vue électronique :

Pour un signal périodique de période T , correspondant à une tension,

- l'amplitude crête à crête est définie par $\max_{t \in [0, T]} v(t) - \min_{t \in [0, T]} v(t)$.
- La tension efficace ou la tension en moyenne quadratique est définie par $\sqrt{\frac{1}{T} \int_0^T v^2(t) dt}$ ce qui correspond à $\sqrt{P_v}$.

En termes numériques :

On observe que la définition de la moyenne et de la puissance coïncident avec la définition utilisant une limite

$$M_x = \lim_{T \rightarrow +\infty} \frac{1}{T} \int_{-T/2}^{T/2} x(t) dt \text{ et } P_x = \lim_{T \rightarrow +\infty} \frac{1}{T} \int_{-T/2}^{T/2} x^2(t) dt \quad (10.5)$$

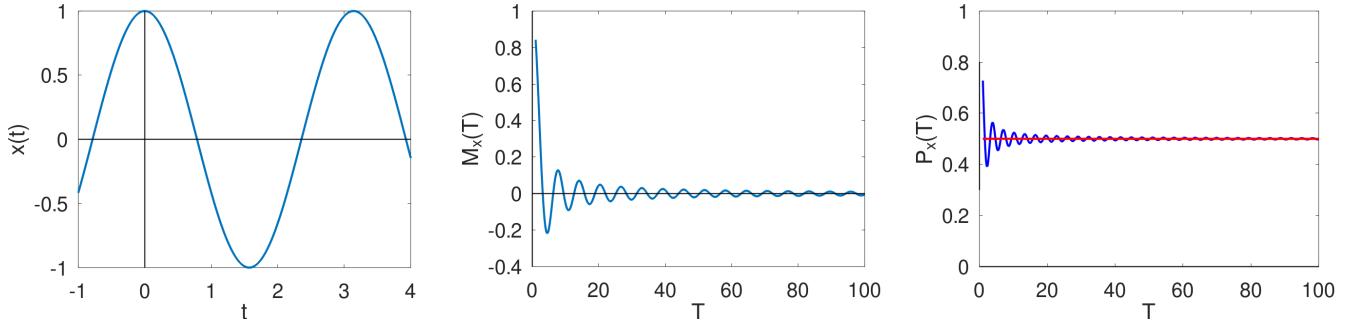


Figure 10.2: Signal $x(t) = \cos(2t)$ et moyenne et puissance de $x(t)$ évaluées entre $-T/2$ et $T/2$ en fonction de T .

Python :

On vérifie tout d'abord que le signal construit t, x commence bien en $t = 0$.

```
assert x[seb.where_nearest(t,0)]==x[0]
```

`where_nearest` est une fonction que j'ai programmé et placé dans `seb` qui donne l'indice dont la valeur du vecteur ici `t` est la plus proche de la valeur ici 0.

On vérifie ce signal est périodique de période T (du moins que $x(T) = x(0)$).

```
assert x[seb.where_nearest(t,T)]==x[0]
```

On calcule le nombre de composantes correspondant à une période :

```
N=seb.where_nearest(t,T)
```

On calcule la moyenne

```
print(f"moyenne={np.mean(x[:N]):.2f}")
```

cette instruction utilise les notions suivantes :

- `x[:N]` est un vecteur de N composantes, 0 jusqu'à $N - 1$.
- `np.mean(x[:N])` effectue le calcul de la moyenne.
- `f"moyenne=np.mean(x[:N]):.2f"` fabrique une chaîne de caractères dont l'expressoin entre accolades est évaluées et affichées avec deux chiffres significatifs. Attention au `f` précédent l'accordade.
- `print` est l'instruction affichant cette chaîne de caractères à l'écran.

L'implémentation est faite dans ??.

Python :

On peut simuler numériquement l'équation (10.5). Dans l'exemple de la figure ?? où $x(t) = \cos(2\pi t)$, on peut estimer la moyenne et la puissance. On crée tout d'abord une échelle de temps entre $-T/2$ et $T/2$ pour une valeur de T grande. `numpy.arange` permet de fabriquer un vecteur avec des valeurs également réparties.

```
T=10**4  
t=np.arange(-T/2,T/2,1e-2)
```

On détermine ensuite les valeurs du signal.

```
x=np.cos(2*np.pi*t)
```

Puis on estime la moyenne et la puissance

```
M=np.mean(x)  
P=np.mean(x**2)  
print(f"M={M:.3f} P={P:.3f}")
```

10.3 Transformée de Fourier

D'un point de vue électronique :



Figure 10.3: Analyseur de spectre

La figure 10.3 montre un analyseur de spectre. L'échelle des abscisses varie entre $\text{fréquence centrale}-\frac{1}{2}\text{span}$ et $\text{fréquence centrale}+\frac{1}{2}\text{span}$.

Pour des raisons technologiques, il est nécessaire de préciser la largeur de bande du filtre de résolution. Plus cette largeur est faible, plus le `sweep time` est long pour faire la mesure et tracer la figure. Ce filtre de résolution transforme un pic en un signal avec une certaine largeur égale à la largeur de bande du filtre. Ceci n'est donc pas un filtre au sens du cours.

L'axe vertical est soit la tension en mV ou en dBmV , ou la puissance en mW ou en dBm . En ce qui concerne la puissance, elle est en réalité déduite de la tension en tenant compte de la résistance équivalente de l'analyseur de spectre vue de l'entrée, celle-ci est donc proportionnelle au carré de la tension. Les unités dBm et dBmV correspondent à des échelles logarithmiques.

$$U_{\text{dBmV}} = 10 \log_{10} (U_{\text{mV}}) \text{ et } P_{\text{dBm}} = 10 \log_{10} (P_{\text{mW}}) \quad (10.6)$$

Cours signal et bruit :

La transformée de Fourier d'un signal temps continu et périodique est toujours un ensemble de raies espacées de $\frac{1}{T}$ (ou un

Les fréquences considérées sont $f_k = \frac{k}{T}$, leur unité est le Hertz (Hz), k est un entier positif ou négatif.

Transformée de Fourier pour un signal temps continu périodique

$$\hat{X}_k = \frac{1}{T} \int_{-\frac{T}{2}}^{+\frac{T}{2}} x(t) e^{-j2\pi k \frac{t}{T}} dt \quad (10.7)$$

Transformée de Fourier inverse d'un spectre défini sur toutes les fréquences et non-périodique

$$x(t) = \sum_{k=-\infty}^{+\infty} \hat{X}_k e^{\frac{j2\pi k t}{T}} \quad (10.8)$$

Python :

La fonction `seb.py` permet d'implémenter le calcul des coefficients de la série de Fourier en utilisant `seb.coef_serie_Fourier`. Pour cela il faut une échelle de temps `t`, des valeurs du signal `x(t)` à ces instants mémoisées dans un vecteur `x`, une indication `T` de l'intervalle sur lequel le signal est défini et est périodique et un ensemble de valeur `k` pour lesquelles les coefficients vont être calculés. Le signal `x(t)` est périodique de période 1, on peut par exemple choisir `T=(-0.5,0.5)`

```
t=np.arange(-0.5,0.5,1e-3)
x=np.cos(2*np.pi*t)
T=(-0.5,0.5)
k=np.arange(-10,10,1)
Xk=seb.coef_serie_Fourier(t,x,T,k)
```

On pourrait aussi choisir que le signal périodique est défini par ses valeurs entre 0 et 1.

```
t=np.arange(0,1,1e-3)
x=np.cos(2*np.pi*t)
T=1
k=np.arange(-10,10,1)
Xk=seb.coef_serie_Fourier(t,x,T,k)
```

Il n'y a pas de fonctions prévues dans `seb.py` pour retrouver le signal à partir des coefficients de la série de Fourier. Mais on peut quand même programmer ce calcul assez simplement. L'exemple qui suit montre le fait d'utiliser `Xk` calculé précédemment pour trouver une approximation de `x` notée `x_app`. L'échelle de temps est maintenant plus large qu'une période, elle est choisie en fonction de l'affichage souhaité. Les commandes qui suivent créent un vecteur contenant le résultat avec des zéros et pour chaque entier `k`, y rajoutent $\hat{X}_k e^{j2\pi k t}$.

```
t=np.linspace(-3,3,10**2*2)
x_app=np.zeros(len(t))
for k_ in range(len(k)):
    x_app += Xk[k_]*np.exp(1j*k_*2*np.pi*t)
```

Cours signal et bruit :

On peut noter la transformée de Fourier d'un signal périodique au moyen de la distribution $\delta(f)$.

$$\widehat{X}(f) = \sum_{k=-\infty}^{+\infty} \widehat{X}_k \delta\left(f - \frac{k}{T}\right) \quad (10.9)$$

Cette écriture n'apporte aucune information supplémentaire autre que \widehat{X}_k et $f_k = \frac{k}{T}$.

D'un point de vue mathématique :

On appelle \widehat{X}_k les coefficients de la série de Fourier. La façon dont avec l'équation (10.8), les coefficients \widehat{X}_k permettent de retrouver $x(t)$ est appelé la série de Fourier.

Cours signal et bruit :

La définition de la transformée de Fourier adaptée aux signaux périodiques permet de compléter la table 3.1.

$\bullet \text{ TF} [e^{j2\pi f_0 t}] (f) = \delta(f - f_0)$	$\bullet \text{ TF}^{-1} [\delta(f - f_0)] (t) = e^{j2\pi f_0 t}$
--	---

D'un point de vue mathématique :

Attention quand on écrit $\text{TF} [e^{j2\pi f_0 t}] (f)$ en remplaçant TF par une intégrale de $-\infty$ à $+\infty$, on a une expression qui n'a aucun sens (du moins pas au sens des théories classiques pour l'intégration). Dans ce cours le sens donné à cette expression se limite au fait que si on connaît une distribution (en l'occurrence $\delta(f - f_0)$), et qu'on luit applique la transformée de Fourier inverse avec son expression intégrale on aboutit bien à $e^{j2\pi f_0 t}$.

Cours signal et bruit :

Tout signal périodique est la somme de sinusoïdes pondérées et déphasées.

Le calcul de ces sinusoïdes se fait à partir de l'équation (10.8).

$$x(t) = \widehat{X}_0 + 2 \sum_{k=1}^{+\infty} |\widehat{X}_k| \cos\left(\frac{2\pi k t}{T} + \varphi_k\right) \text{ où } \varphi_k = \arg(\widehat{X}_k) \quad (10.10)$$

En effet en posant $\varphi_k = \arg(\widehat{X}_k)$ on a

$$\widehat{X}_k = |\widehat{X}_k| e^{j\varphi_k} \text{ et } \widehat{X}_{-k} = \overline{\widehat{X}_k} = |\widehat{X}_k| e^{-j\varphi_k} \quad (10.11)$$

L'équation (10.8) s'écrit alors

$$x(t) = \widehat{X}_0 + \sum_{k=1}^{+\infty} |\widehat{X}_k| \left(e^{\frac{j2\pi k t}{T} + j\varphi_k} + e^{-\frac{j2\pi k t}{T} - j\varphi_k} \right) \quad (10.12)$$

et finalement on trouve l'équation (10.10).

En termes numériques :

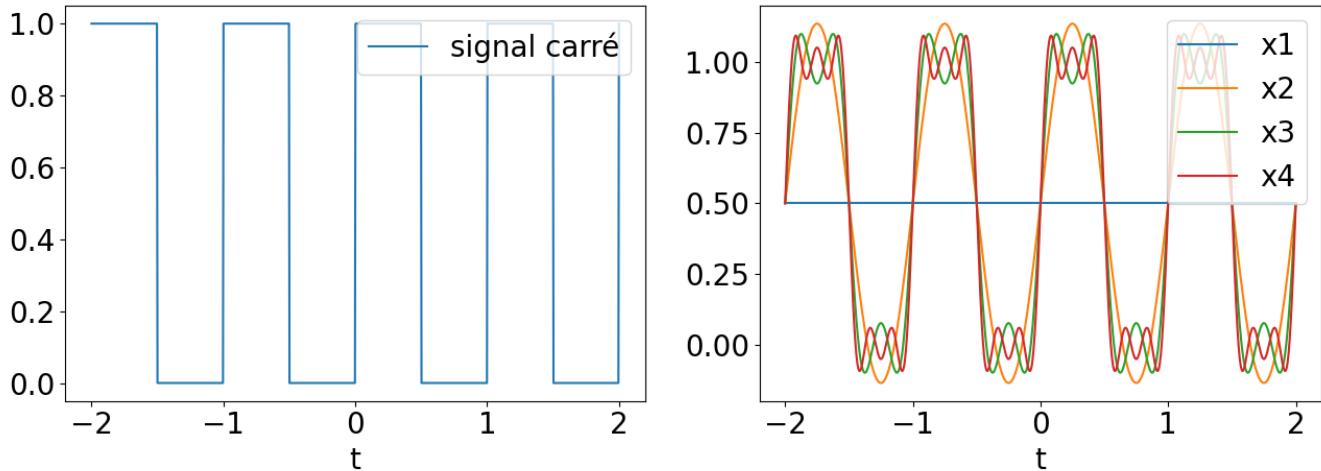


Figure 10.4: À gauche le signal carré périodique d'origine $x(t)$ et à droite sa reconstruction au moyen de sinusoïdes. L'implémentation est dans A.8.

La gauche de la figure 10.4 représente un signal périodique de période 1 qui vaut $\Pi(2t - 0.5)$ sur l'intervalle $[0, 1]$. Les coefficients de la série de Fourier sont

$$\hat{X}_k = \int_0^1 x(t)e^{-j2\pi kt} dt = \int_0^{0.5} e^{-j2\pi kt} dt \quad (10.13)$$

- Si $k = 0$, $\hat{X}_0 = 0.5$.
- Si $k \neq 0$

$$\hat{X}_k = \int_0^1 x(t)e^{-j2\pi kt} dt = \left[\frac{-e^{-j2\pi kt}}{j2\pi k} \right]_0^{0.5} = \frac{1 - (-1)^k}{j2\pi k} \quad (10.14)$$

- Si $k \neq 0$ et k pair, $\hat{X}_k = 0$
- Si k est impair, $\hat{X}_k = \frac{1}{k\pi} e^{-j\pi/2}$

Finalement

$$x(t) = 0.5 + 2 \sum_{k \text{ impair et positif}} \cos \left(2\pi kt - \frac{\pi}{2} \right) \quad (10.15)$$

10.4 Périodisation

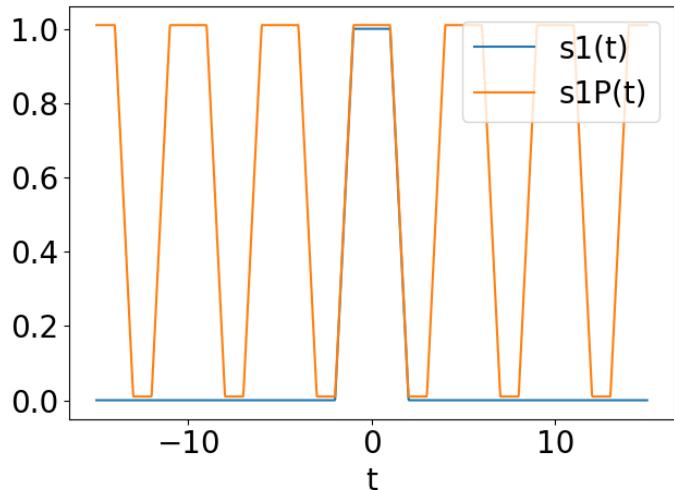


Figure 10.5: Signal $s_1(t)$ en bleu périodisé en $s_1^P(t)$ en répétant le motif défini sur $\left[-\frac{5}{2}, \frac{5}{2}\right]$.

Cours signal et bruit :

À partir de $x(t)$ un signal non-nul seulement dans l'intervalle $[0, T]$, on peut définir $x_P(t)$ un signal périodique de période T appelé **périodisation** de $x(t)$ en lui ajoutant des copies retardées de T , $2T$, etc... À partir de ce signal périodisé $x_P(t)$, on peut retrouver $x(t)$ par **apériodisation** en le multipliant par une porte sur $[0, T]$.

$$\begin{array}{ccc} x(t) & \xrightarrow{\text{périodisation}} & x_P(t) = \sum_{k=-\infty}^{+\infty} x(t - kT) \\ x(t) = x_P(t)[t \in [0, T]](t) & \xleftarrow{\text{apériodisation}} & x_P(t) \end{array} \quad (10.16)$$

Dans le cadre de ce cours, on note aussi la périodisation

$$x_P(t) = \text{Périodiser}_{T_e}(x_t) \quad (10.17)$$

La figure 10.5 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 A.13.

Python :

Pour représenter un signal périodique sur une échelle de temps t , au moyen d'une définition de ce signal périodique qui n'est valable que sur une période de temps entre t_0 et t_1 , on utilise la fonction `periodiser_ech_t` du module `seb.py` qui transforme l'échelle de temps t en une échelle de temps tp dont les valeurs sont entre t_0 et t_1 mais qui associé à t donne un signal périodique. Par exemple pour la figure 10.5, on crée d'abord l'échelle de temps souhaité.

```
t=np.linspace(-15,15,10**3)
```

Puis on périodise cette échelle de temps en lui donnant des valeurs entre -2.5 et 2.5 .

```
tp=seb.periodiser_ech_t(t,(-2.5,2.5))
```

Il se trouve que le signal non-périodique considéré est $2\mathbb{T}\left(\frac{t}{2}\right) - \mathbb{T}(t)$, aussi le signal périodisé est

```
sp = seb.fonction_T(tp/2)*2-seb.fonction_T(tp)
```

10.5 Calcul numérique des coefficients de la série de Fourier

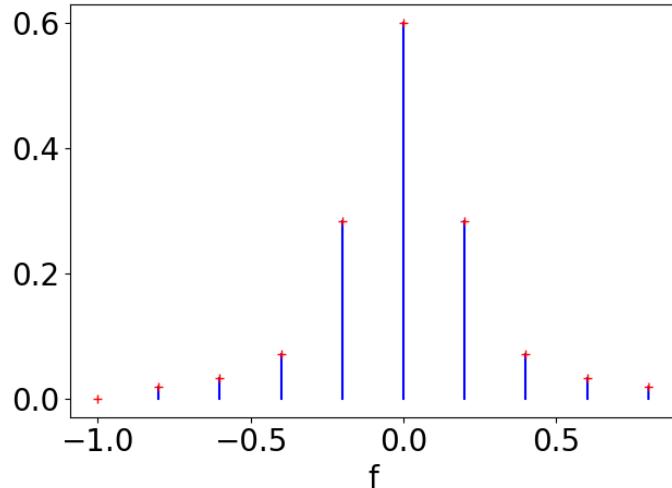


Figure 10.6: Signal $s_1(t)$ en bleu périodisé en $s_1^P(t)$ en répétant le motif $\left[-\frac{5}{2}, \frac{5}{2}\right]$.

La figure 10.6 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 A.15.

Le module `seb.py` contient la fonction `coef_serie_Fourier` qui approxime le calcul des coefficients de la série de Fourier noté \widehat{S}_k . Les arguments sont l'échelle de temps et les valeurs du signal temps continu, l'intervalle sur lequel ce signal est évalué ($-T/2$ et $T/2$ ou 0 et T) et les coefficients k . Cette fonction fournit aussi les fréquences associées aux coefficients k transmis, ce qui forme l'échelle en fréquence.

En l'occurrence pour la figure 10.6, on crée une échelle des indices k entre -5 et 5

```
k = np.arange(-5,5,1)
```

On définit une échelle pour le signal temps continu entre $-T/2$ et $T/2$

```
t=np.linspace(-T/2,T/2,10**3)
```

On évalue le signal temps continu sur cet échelle en utilisant sa définition $s(t) = 2\mathbb{T}(t/2) - \mathbb{T}(t)$.

```
s=2*seb.fonction_T(t/2)-seb.fonction_T(t)
```

Et on utilise la fonction `coef_serie_Fourier`

```
f,SP = seb.coef_serie_Fourier(t,s,(-T/2,T/2),k)
```

10.6 Relation entre transformée de Fourier et calcul des coefficients de la série de Fourier

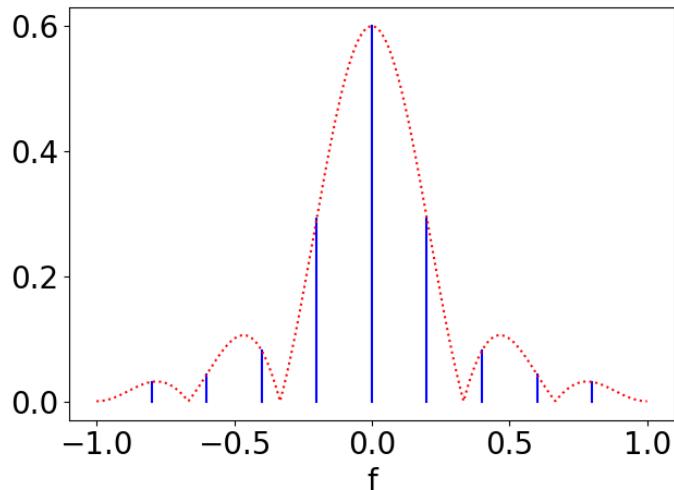


Figure 10.7: L'enveloppe est la transformée de Fourier du signal temps continu divisé par $T = 5$, et les raies sont les coefficients de la série de Fourier.

La ressemblance entre le calcul des coefficients de la série de Fourier défini par (10.7) et le calcul de la transformée de Fourier (3.15) amène à penser qu'il pourrait exister une relation entre les coefficients de la série de Fourier et le résultat de la transformée de Fourier.

Cours signal et bruit :

Les coefficients de la série de Fourier de $x^P(t) = \text{Périodiser}_T(x(t))$ sont des valeurs particulières de la transformée de Fourier, ce sont les valeurs associées à $f_k = \frac{k}{T}$.

$$x^P(t) = \text{Périodiser}_T \Rightarrow \widehat{X}_k^P = \frac{1}{T} \widehat{X} \left(\frac{k}{T} \right) \quad (10.18)$$

Pour représenter la figure 10.7, on représente d'abord la courbe rouge qui est la transformée de Fourier du signal temps continu en considérant d'abord une échelle de fréquence pour afficher,

```
f=np.linspace(-1,1,10**2)
une échelle de temps pour la simulation,
tx=np.linspace(-2.5,2.5,10**3)
une évaluation du signal temps continu,
x=2*fonction_T(tx/2)-fonction_T(tx)
et finalement le calcul lui-même
X=syb.TF(tx,x,f)
```

Pour en extraire les coefficients de la Série de Fourier, on modifie la dernière instruction en utilisant une période de $T = 5$.

```
T=5
k=np.arange(-5,5,1)
XP_k=syb.TF(tx,x,k/T)/T
```

La figure 10.7 est implémentée dans A.14

Chapter 11

Filtres agissant sur des signaux périodiques

11.1 Réponse forcée, réponse transitoire

Cours signal et bruit :

La sortie associée à une entrée périodique de période T est périodique de période T .

D'un point de vue électronique :

Expérimentalement, il n'y a pas de vrais signaux périodiques qui seraient définis depuis un temps infini. La sortie observée n'est donc qu'approximativement périodique, elle est la somme d'un signal non-périodique appelé **réponse transitoire** et d'un signal périodique appelé **réponse forcée**. Ce dernier signal correspond à la sortie du filtre associée à l'entrée périodique.

Cours signal et bruit :

Pour illustrer la notion de réponse transitoire et réponse forcée, je considère le signal d'entrée $x_1(t) = \cos(\pi t)$ et le filtre de réponse impulsionnelle $h(t) = \llbracket t \in [0, 1] \rrbracket$ défini par

$$y(t) = \int_{t-1}^t x(\tau) d\tau \quad (11.1)$$

La sortie associée à $x_1(t)$ est

$$y_1(t) = \int_{t-1}^t \cos(\pi\tau) d\tau = \left[\frac{\sin(\pi\tau)}{\pi} \right]_{t-1}^t = \frac{2}{\pi} \sin(\pi t) \quad (11.2)$$

Pour voir ce qui se passe quand on commence l'expérience à $t = 0$, je définit $x_2(t) = x_1(t) \llbracket t \geq 0 \rrbracket$ et je note $y_2(t)$ la sortie correspondante.

- Si $t < 0$, $y_2(t) = \int_{t-1}^t x_2(\tau) d\tau = 0$.
- Si $t \in [0, 1[$, $y_2(t) = \int_{t-1}^t x_2(\tau) d\tau = \int_0^t x_2(\tau) d\tau = \frac{\sin(\pi t)}{\pi}$
- Si $t \geq 1$, $y_2(t) = \int_{t-1}^t x_2(\tau) d\tau = y_1(t)$.

La réponse transitoire $y_3(t)$ est définie par

$$y_2(t) = \begin{cases} y_1(t) + y_3(t) & \text{si } t \geq 0 \\ 0 & \text{si } t < 0 \end{cases} \quad (11.3)$$

On en déduit que la réponse transitoire est

$$\begin{aligned} y_3(t) &= (y_2(t) - y_1(t)) \llbracket t \geq 0 \rrbracket = \frac{\sin(\pi t)}{\pi} \llbracket t \in [0, 1] \rrbracket + \frac{2}{\pi} \sin(\pi t) \llbracket t \geq 1 \rrbracket - \frac{2}{\pi} \sin(\pi t) \llbracket t \geq 0 \rrbracket \\ &= -\frac{1}{\pi} \sin(\pi t) \llbracket t \in [0, 1] \rrbracket \end{aligned} \quad (11.4)$$

En terme de visualisation :

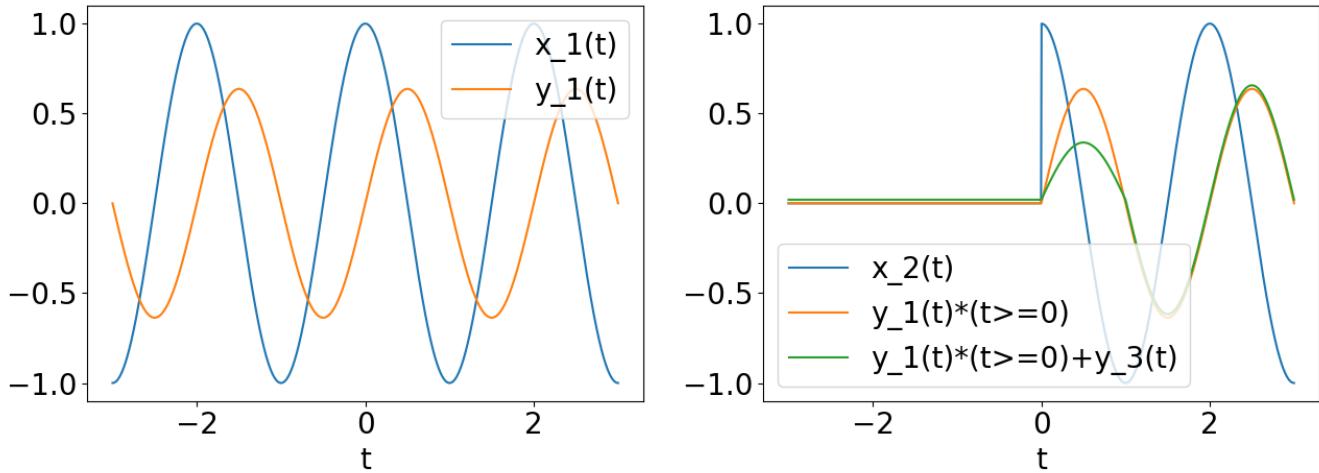


Figure 11.1: À gauche : entrée et sortie périodiques. À droite : entrée et sortie commençant à $t = 0$.

La figure 11.1 montre à gauche un signal sinusoïdal placé en entrée qui est retardé et atténue en sortie. Le retard correspond à un quart de la période, c'est donc un déphasage de $-\frac{\pi}{2}$. Cette figure montre à droite une entrée qui n'est plus périodique puisqu'elle commence en $t = 0$. La sortie est la somme d'une réponse forcée, une sinusoïdale commençant à $t = 0$ (courbe orange) et d'une réponse transitoire, c'est la différence entre la courbe orange et la courbe verte, elle est nulle pour $t \in [0, 1]$. L'implémentation est dans A.7.

11.2 Utilisation de la réponse fréquentielle pour calculer la réponse forcée

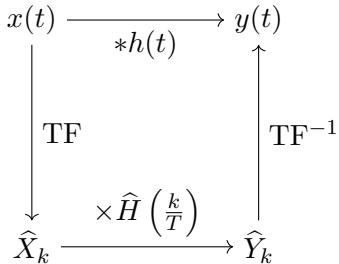


Figure 11.2: Diagramme illustrant la façon de calculer la sortie d'un filtre dont l'entrée est périodique

D'un point de vue mathématique :

Il est tentant d'utiliser la définition du produit de convolution pour calculer la sortie d'un filtre dont l'entrée est périodique. Mais pour une réponse impulsionnelle générale, ceci s'avère en général compliqué.

Cours signal et bruit :

La méthode classique pour calculer la sortie d'un filtre dont l'entrée est périodique est illustré sur la figure 11.2.

- Calcul des \hat{X}_k de la série de Fourier en fonction de $x(t)$.
- Calcul de \hat{Y}_k en fonction de \hat{X}_k et de la réponse fréquentielle évaluée en les fréquences $f_k = \frac{k}{T}$.
- Calcul de $y(t)$ en utilisant la série de Fourier.

Python :

Pour illustrer cette façon de procéder, supposons que le signal périodique considéré est le signal périodique de période 2 défini sur $[0, 2[$ par $x(t) = e^{-t} \llbracket 0 \leq t < 2 \rrbracket(t)$. Les coefficients de la série de Fourier associés sont

```

t=np.arange(0,2,1e-3)
x=np.exp(-t)
k=np.arange(-10,10)
Xk=seb.coef_serie_Fourier(t,x,2,k)

```

Dans l'exemple décrit plus haut, la réponse impulsionnelle est $h(t) = \llbracket 0 \leq t \leq 1 \rrbracket(t)$ a une transformée de Fourier calculée pour les fréquences $\frac{k}{2}$, 2 étant la période de $x(t)$.

```

f=k/2
t=np.arange(0,2,1e-3)
h=np.ones(len(t))
H=seb.TF(t,h,f)

```

On en déduit les coefficients de la série de Fourier de la sortie du filtre $\hat{Y}_k = \hat{H}\left(\frac{k}{2}\right)\hat{X}_k$

```
Yk=H*Xk
```

On en déduit finalement le signal $y(t)$

```

ty=np.linspace(-3,3,10**2)
y=np.zeros(len(ty))
for k_ in range(len(k)):
    y += Yk*np.exp(1j*2*np.pi*t)

```

Chapter 12

Échantillonnage d'un signal non-périodique

12.1 Signal temps discret

Cours signal et bruit :

Un **signal temps discret** est décrit par

- des instants régulièrement espacées de T_e appelé **période d'échantillonnage** ;
- ces instants régulièrement espacées doivent commencer à $t = 0$, aussi ces instants s'écrivent nT_e ;
- à chacun de ces instants nT_e , le signal prend une valeur notée $s_n^{\#}$.

La notation avec $\#$ est utilisée dans ce cours pour pouvoir à la fois utiliser la même lettre, ici s pour le signal temps continu et temps discret tout en pouvant les distinguer puisque le signal temps discret a le $\#$. Cette notation n'est pas du tout universelle, elle n'est d'ailleurs utilisée dans ce cours que lorsqu'il y a une ambiguïté. J'utilise aussi parfois l'indice e pour noter un signal échantillonné par exemple s_e et dans ce cas on ne peut plus mettre les indices en bas et du coup la notation est $s_e[n]$.

La période d'échantillonnage est seconde (s).

D'un point de vue mathématique :

Le signal commençant à $n = 0$ et étant composé de N instants, le dernier instant ne peut être NT_e . En effet,

- soit on a $N + 1$ échantillons qui couvrent $0, T_e, \dots, NT_e$ et $t_{N+1} = NT_e$;
- soit on a N échantillons qui couvrent $0, T_e, \dots, (N - 1)T_e$ et $t_N = (N - 1)T_e$.

C'est la deuxième convention qu'on utilise dans ce cours.

On appelle **fréquence d'échantillonnage**

$$f_e = \frac{1}{T_e} \quad (12.1)$$

La fréquence d'échantillonnage est en Hertz (Hz).

Python :

Une fois définies les valeurs de `N, Te, t0`, les commandes suivantes permettent de définir la durée, la fréquence d'échantillonnage et l'échelle de temps.

```
T=N*Te  
fe=1/Te  
t=np.arange(t0,t0+T,Te)
```

En déclenchant une erreur quand une affirmation est fausse, la commande `assert` est très pratique pour quelqu'un qui relit un code, parce qu'elle lui permet d'être sûr d'une affirmation.

```

assert t[N-1]==t[-1]
assert t[-1]==t0+(N-1)*Te
assert Te==t[1]-t[0]

```

Cours signal et bruit :

Ainsi un signal est décrit par $t = nT_e$ et $s_n^\#$. On peut écrire aussi un signal en utilisant des Diracs.

$$s^\#(t) = \sum_{n=-\infty}^{+\infty} s_n^\# \delta(t - nT_e) \quad (12.2)$$

Cette autre formulation n'apporte rien de plus que $t = nT_e$ et $s_n^\#$. Formellement $s^\#(t)$ et $s_n^\#$ sont différents, le premier est une somme de distributions, le deuxième est une suite. Mais les deux décrivent le même signal au moyen de la même modélisation. C'est la raison pour laquelle je propose d'utiliser la même notation.

D'un point de vue électronique :

Un signal numérique traité par un processeur électronique est généralement modélisé par signal temps discret, avec une période d'échantillonnage qui est égale ou est un multiple de l'inverse de la fréquence de l'horloge. Et la valeur du signal est codé en binaire.

Un signal temps discret peut aussi être utilisé pour rendre compte de mesures physiques faites à des instants régulièrement répartis dans le temps, par exemple la prise de la température toutes les 30 secondes, ou la mesure de la crue du Nil tous les ans depuis 2000 ans.

12.2 Échantillonnage

Cours signal et bruit :

On appelle [échantillonnage] le fait de transformer un signal temps continu (i.e. défini à chaque instant) en un signal temps discret. Cela suppose nécessairement le choix d'une période d'échantillonnage.

$$s_n^\# = s(nT_e) \quad (12.3)$$

On peut à nouveau utiliser la notation avec des Diracs

$$s^\#(t) = \sum_{n=-\infty}^{+\infty} \delta(t - nT_e) s(nT_e) \quad (12.4)$$

$s(t)$ est différent de $s^\#(t)$. Le premier modélise un signal qui est temps continu et le deuxième modélise un autre signal qui est temps discret et dont le lien avec $s(t)$ est que les valeurs coïncident en $t = nT_e$.

D'un point de vue mathématique :

Le produit d'une fonction avec une distribution n'est pas nécessairement défini pour toutes les fonctions et toutes les distributions. Ici il y a un sens $s(t)\delta(t - nT_e) = s(nT_e)\delta(t - nT_e)$, en admettant que $s(t)$ tend vers une valeur notée $s(nT_e)$ quand t tend vers nT_e . Dans le cas où cette limite existe à gauche et à droite, on remplace la valeur par la demi-somme de la limite à gauche et à droite.

On appelle [peigne de Diracs] la succession infinie de Diracs

$$\text{III}(t) = \sum_{n=-\infty}^{+\infty} \delta(t - nT_e) \quad \text{et} \quad \text{III}\left(\frac{t}{T_e}\right) = \sum_{n=-\infty}^{+\infty} \delta\left(\frac{t}{T_e} - n\right) \quad (12.5)$$

Cette notation permet d'écrire l'échantillonnage comme

$$s^\#(t) = \text{III}\left(\frac{t}{T_e}\right) s(t) \quad (12.6)$$

D'un point de vue électronique :

La notion d'échantillonnage ainsi définie est une situation idéalisée. En réalité la conversion d'un signal temps continu en un signal temps discret correspond plutôt à une moyenne des valeurs du signal pendant une durée donnée. Du coup un échantillonnage réaliste peut être modélisé par un filtre analogique appliqué au signal temps continu dont la sortie est, elle, échantillonnée en un signal temps discret, au sens de prélever une valeur sur une durée infiniment petite.

En termes numériques :

Numériquement les signaux sur ordinateur sont de fait échantillonnés parce qu'ils sont gérés par un tableau de valeurs. Par contre il y a une différence entre simuler un signal échantillonné et simuler un signal temps continu au moyen d'un signal échantillonné. Dans le premier cas, la valeur de la période d'échantillonnage compte beaucoup, dans le second cas, ce qui nous importe est que les calculs soient suffisamment précis on cherche donc une période d'échantillonnage très petite. Cette période d'échantillonnage ne doit pas être trop petite pour éviter que la simulation ne prenne trop de temps. Par exemple j'évite en général de définir un variable contenant plus de 10^7 composantes à stocker en mémoire.

12.3 Signaux temps discret non périodiques : transformées de Fourier à temps discret

Cours signal et bruit :

La transformée de Fourier d'un signal temps discret est un signal périodique de période f_e . Et à l'inverse tout fonction périodique est la transformée d'un signal temps discret dont la période d'échantillonnage est l'inverse de la période de cette fonction périodique.

D'un point de vue mathématique :

Cette idée de périodicité peut se retrouver avec le peigne de Diracs. La transformée de Fourier d'un peigne de Dirac est donnée par

$$\text{TF}[\text{III}(t)](f) = \text{III}(f) \text{ et } \text{TF}\left[\text{III}\left(\frac{t}{T_e}\right)\right](f) = T_e \text{III}\left(\frac{f}{f_e}\right) \quad (12.7)$$

Le coefficient T_e vient de l'application de la formule (4.1) (p. 27) adaptées aux signaux temps continu.

La transformée de Fourier d'un produit est un produit de convolution et le produit de convolution par un peigne de Dirac est une périodisation aussi

$$\widehat{S}^\#(f) = \text{TF}[s^\#(t)](f) = \frac{1}{f_e} \text{III}\left(\frac{f}{f_e}\right) * \widehat{S}(f) = \frac{1}{f_e} \sum_{k=-\infty}^{+\infty} \widehat{S}(f - kf_e) = \frac{1}{f_e} \text{Périodisation}_{f_e} [\widehat{S}(f)] \quad (12.8)$$

On retrouve le coefficient T_e maintenant écrit sous la forme $\frac{1}{f_e}$.

Cours signal et bruit :

On définit une nouvelle formulation de la transformée de Fourier adaptée aux signaux temps discret non périodiques et appellés Transformée de Fourier à Temps Discret, que l'on note ici TFTD.

$$\text{TFTD}\left[s_n^\#\right](f) = \sum_{n=-\infty}^{+\infty} s_n^\# e^{-j2\pi fnT_e} \quad (12.9)$$

On peut noter que

$$\text{TF}[s^\#(t)](f) = \text{TFTD}[s_n^\#](f) \quad (12.10)$$

En effet $\text{TF}[s^\#(t)](f) = \int_{-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} s_n^\# \delta(t - nT_e) e^{-j\pi f n T_e} df = \text{TFTD}[s_n^\#](f)$

Lors de l'application de la transformée de Fourier, je fais une différence entre $s^\#(t)$ et $s_n^\#$. Le premier étant une somme de Dirac, je peux utiliser l'écriture intégrale de la transformée de Fourier, écriture que je note ici TF. En revanche pour $s_n^\#$, comme il s'agit d'une suite, j'utilise la transformée de Fourier à temps discret (TFTD) qui est une expression avec des sommes.

En termes numériques :

La transformée de Fourier à temps discret permet d'approcher la transformée de Fourier à temps continu à condition de prendre en compte T_e qui est la largeur des rectangles utilisés pour faire l'intégration.

$$\int_{-\infty}^{+\infty} x(t) e^{-j2\pi ft} dt = \text{TF}[x(t)](f) \approx T_e \text{TFTD}[x(nT_e)](f) = \sum_{n=-\infty}^{n=+\infty} x(nT_e) e^{-j2\pi fnT_e} \quad (12.11)$$

Python :

Il y a deux outils distincts pour calculer la transformée de Fourier des signaux à temps continu `seb.TF` et à temps discret `seb.TFTD`, parce que `seb.TF` utilise des correctifs par rapport à cette approximation pour améliorer l'estimation de la transformée de Fourier des signaux à temps continu.

D'un point de vue mathématique :

Les signaux échantillonnés sont parfois (en particulier ceux qui ne sont pas trop compliqués) des suites géométriques. On dispose de deux relations

$$1 + z + z^2 + \dots + z^N = \frac{1 - z^{N+1}}{1 - z} \text{ et quand } |z| < 1, \quad \sum_{n=0}^{+\infty} z^n = \frac{1}{1 - z} \quad (12.12)$$

La première se démontre en faisant le calcul explicite de $(1 - z)(1 + z + z^2 + \dots + z^N)$ et la deuxième se déduit de la première.

12.4 Périodisation du spectre

Cours signal et bruit :

La transformée de Fourier du signal échantillonné est aussi la périodisation de la transformée de Fourier du signal temps continu.

$$\widehat{S}^\#(f) = \text{TF} \left[s^\#(t) \right] (f) = f_e \sum_{k=-\infty}^{+\infty} \widehat{S}(f - kf_e) \quad (12.13)$$

Python :

Pour illustrer l'équation 12.13, je considère un signal temps continu, $s(t) = e^{-t} \llbracket t \geq 0 \rrbracket(t)$. J'échantillonner ce signal en $x^\#[n] = x(nT_e)$ avec $f_e = 5\text{Hz}$. La discontinuité en $t = 0$, fait que pour que les calculs fonctionnent bien, je prend

$$s_0^\# = 0.5 \lim_{t \rightarrow 0^+} s(t) + 0.5 \lim_{t \rightarrow 0^-} s(t) = 0.5 \quad (12.14)$$

```
fe=5
n=np.arange(0,10**2)
Te=1/fe
se=np.exp(-n*Te); se[0]=0.5
```

Je calcule la TFTD de ce signal sur l'intervalle $[-2.5, 2.5]$.

```
f=np.linspace(-2.5,2.5,10**2)
Se=seb.TFTD(n*Te,se,f)
```

Je compare ces valeurs avec la transformée de Fourier de $x(t)$ périodisé. Le signal $s(t)$ est d'abord défini

```
ts=np.linspace(-1,100,10**5)
s=np.exp(-ts)*(ts>=0)
```

Je calcule sa transformée de Fourier

```
f=np.linspace(-2.5,2.5,10**2)
Sf=seb.TF(ts,s,f)
```

Je calcule maintenant une approximation de $\widehat{S}(f)$ périodisé pour les valeurs de f :

$$\widehat{S}_p(f) = \widehat{S}(f) + \widehat{S}(f - f_e) + \widehat{S}(f + f_e) + \widehat{S}(f - 2f_e) + \widehat{S}(f + 2f_e)$$

```
Sf += seb.TF(ts,s,f-fe)
```

```
Sf += seb.TF(ts,s,f+fe)
```

```
Sf += seb.TF(ts,s,f-2*fe)
```

```
Sf += seb.TF(ts,s,f+2*fe)
```

Au final je vérifie que les calculs coïncident avec

```
print(np.max(np.abs(Sf-Se/fe)))
```

12.5 Représentation du spectre

Cours signal et bruit :

Comme le spectre est périodique de période f_e , on représente généralement celui-ci sur un intervalle de longueur f_e .

- On appelle représentation centrée le fait de représenter le spectre sur l'intervalle $[-\frac{f_e}{2}, \frac{f_e}{2}]$.
- On appelle représentation non-centrée le fait de représenter le spectre sur l'intervalle $[0, f_e]$.

Python :

Ici La simulation d'une représentation centrée ou d'une représentation non-centrée ne diffèrent que par le choix de l'échelle en fréquence. Pour illustrer ceci, je considère un signal temps discret t, x . Je calcule tout d'abord la fréquence d'échantillonnage qui a été utilisée pour ce signal temps discret.

```
fe=1/(t[1]-t[0])
```

Je vérifie que ce signal a bien été échantillonné avec cette fréquence d'échantillonnage.

```
assert all(np.abs(np.diff(t)-1/fe)<1e-8)
```

Pour la représentation centrée en utilisant $\text{len}(x)$ points,

```
f_ce=seb.arange(-fe/2,fe/2,fe/len(x))
```

```
X_ce=seb.TFTD(t,x,f_ce)
```

et pour la représentation non-centrée en utilisant 100 points

```
f_nce=seb.arange(0,fe,fe/len(x))
```

```
X_nce=seb.TFTD(t,x,f_nce)
```

12.6 Interpolation

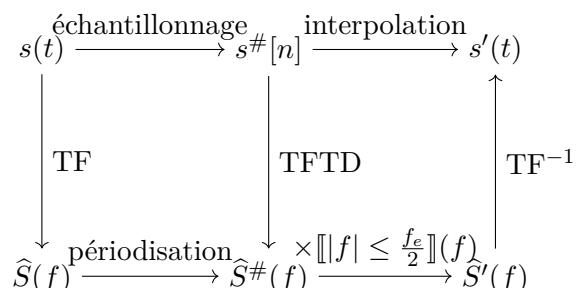


Figure 12.1: Diagramme décrivant l'interpolation

Cours signal et bruit :

L'interpolation signifie estimer la valeur du signal à un instant t connaissant les valeurs des échantillons du signal aux instants nT_e . Dans la figure 12.1, $s'(t)$ est le signal temps continu obtenu en interpolant $s_n^{\#}$ qui est le signal $s(t)$ échantilloné. Cette interpolation donne $s(t)$ qui est alors égale à $s'(t)$ lorsque le critère de Shannon-Nyquist est respecté. Ce critère est respecté quand

$$f_e > 2f_{\max} \text{ où } f_{\max} \text{ vérifie que pour tout } f > f_{\max} \text{ on a } \hat{S}(f) = 0 \quad (12.15)$$

En terme de visualisation :

On peut visualiser le problème de l'interpolation en t , en considérant que pour une période d'échantillonnage T_e , il existe un entier n tel que $nT_e \leq t < (n+1)T_e$, cet entier est donné par $n = \lfloor \frac{t}{T_e} \rfloor$, où $\lfloor \dots \rfloor$ est appelée la partie entière. Du coup l'interpolation peut se voir comment utiliser la valeur du signal en nT_e et en $(n+1)T_e$ pour en déduire celle en t . Le point de vue en traitement du signal est différent parce qu'on s'autorise à utiliser toutes les valeurs du signal en nT_e , quelque soit la valeur de n .

D'un point de vue mathématique :

Il existe de nombreuses techniques pour interpoler un signal, ces techniques reposent sur ce qu'on sait du signal. Par exemple si on sait que le signal est une succession de traits horizontaux, on peut approcher le signal en prolongeant la valeur en nT_e :

$$s^0(t) = \begin{cases} \vdots \\ s_0^{\#} \text{ si } t \in [0, T_e[\\ s_1^{\#} \text{ si } t \in [T_e, 2T_e[\\ \vdots \end{cases} = \sum_{n=-\infty}^{+\infty} s_n^{\#} \llbracket nT_e \leq t < (n+1)T_e \rrbracket(t) \quad (12.16)$$

C'est ce qu'on appelle le bloqueur d'ordre 0, mais pas ce qu'on appelle l'interpolation dans le cadre de ce cours de signal et bruit.

Cours signal et bruit :

L'interpolation du signal est décrite sur la figure 12.1.

- On calcule $\hat{S}^{\#}(f)$ la transformée de Fourier à temps discret du signal échantilloné $s_n^{\#}$.
- Le spectre périodique de période f_e , $\hat{S}_e(f)$, est rendu non-périodique en le multipliant par la porte qui vaut 1 pour $f \in [-\frac{f_e}{2}, \frac{f_e}{2}[$ et 0 ailleurs, le résultat est noté $\hat{S}'(f)$.
- On calcule la transformée de Fourier inverse de $\hat{S}'(f)$ pour trouver $s'(t)$.

Python :

Considérons un signal temps discret décrit par \mathbf{t}, \mathbf{x} . On cherche à interpoler ce signal en des instants décrits par le vecteur \mathbf{ti} , qui eux ne sont pas nécessairement régulièrement répartis.

On calcule la fréquence d'échantillonnage de \mathbf{t}, \mathbf{x}

```
fe=1/(t[1]-t[0])
```

On définit une échelle en fréquence suffisamment précise.

```
f=np.linspace(-fe/2,fe/2,10**6)
```

On calcule le spectre \mathbf{X} sur cette échelle de fréquence.

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

On lui applique la porte sur l'intervalle $[-f_e/2, f_e/2]$.

```
Xpt=X*(f>=-fe/2)*(f<fe/2)
```

On obtient alors le signal interpolé avec

```
x=seb.TFTDI(f,Xpt,ti)
```

12.7 Produit de convolution à temps discret

Cours signal et bruit :

Pour un signal temps discret, on définit un produit de convolution dont la définition doit être adapté, il n'y a pas d'intégrales mais des sommes.

$$x_n^{\#} * y_n^{\#} = \sum_{k=0}^{+\infty} x_k y_{n-k} \quad (12.17)$$

Pour le distinguer de la convolution entre signaux temps continu, je rajoute un d au dessus du signe étoile.

En termes numériques :

Le produit de convolution à temps discret permet d'approcher le produit de convolution à temps continu à condition de multiplier par T_e qui est la largeur des rectangles utilisées pour approcher l'intégrale.

$$\int_{-\infty}^{+\infty} x(nT_e - \tau) y(\tau) d\tau = [x(t) * y(t)](nT_e) \approx T_e \times [x(nT_e) * y(nT_e)](n) = T_e \sum_{k=-\infty}^{+\infty} x(nT_e - kT_e) y(kT_e) \quad (12.18)$$

Python :

La fonction `seb.convolution` utilise cette technique pour approcher le produit de convolution à temps continu. Aussi pour estimer $z(t) = x(t) * y(t)$, on utilise

```
z=seb.convolution(tx,x,ty,y,tz)
```

Et pour calculer $z_n = x_n * y_n$, on utilise

```
z=seb.convolution(tx,x,ty,y,tz)/Te
```

12.8 Simulation de l'erreur quadratique moyenne en fonction de f_e

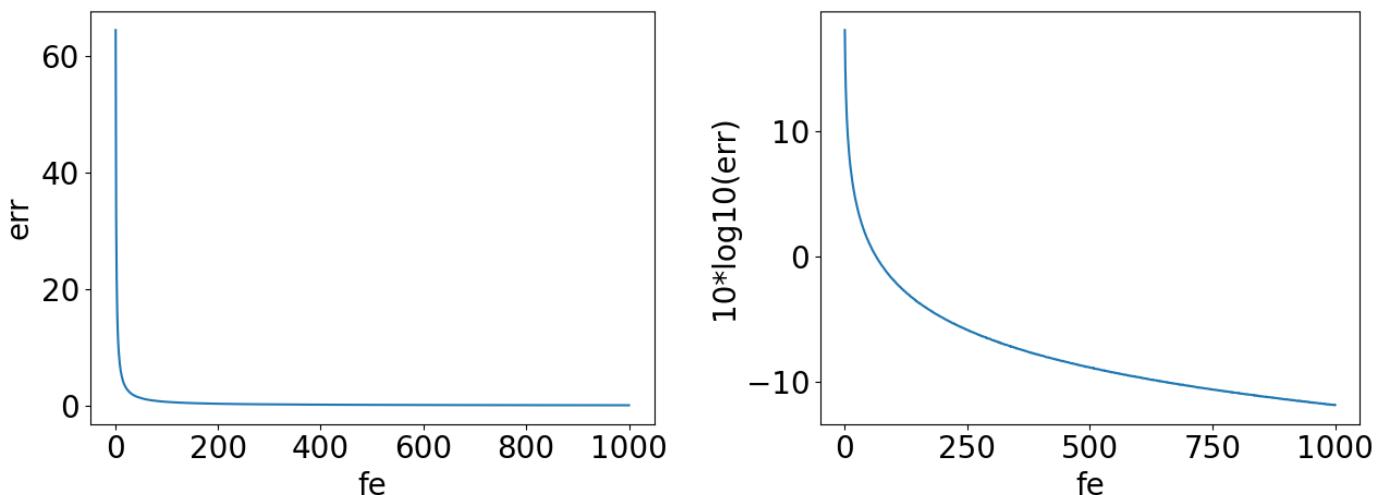


Figure 12.2: Signal étudié $s_1(t)$ et deux échantillonnages

Cours signal et bruit :

On cherche ici à mesurer l'erreur faite en interpolant un signal. Pour cela on considère un signal temps continu, que l'on échantillonne et qu'ensuite on interpole, on mesure alors la différence entre le signal d'origine et à partir de cette différence, on calcule une erreur appelée erreur moyenne quadratique, c'est une intégrale sur le temps du carré de la différence.

- Signal d'origine $s(t)$
- Signal échantillonné $s_n^{\#}$ à la fréquence f_e
- Signal interpolé $s'(t)$
- Erreur calculé sur un intervalle ici $[t_a, t_b]$.

$$\text{err} = \sqrt{\int_{t_a}^{t_b} (s'(t) - s(t))^2 dt} \quad (12.19)$$

En termes numériques :

Pour que l'évaluation numérique soit moins sensible à des valeurs particulières utilisées pour approcher l'intégrale, je considère une variable aléatoire notée \hat{T} qui suit une loi uniforme sur le support du signal (i.e. l'intervalle ou les intervalles sur lesquels le signal est non-nul).

$$\text{err}' = \sqrt{\mathbb{E} \left[(s'(\hat{T}) - s(\hat{T}))^2 \right]} \quad (12.20)$$

Python :

Pour simuler ce nouveau calcul, on tire aléatoirement la valeur de \hat{T} suivant la loi uniforme sur $[t_a, t_b]$ décrite par `ta, tb`.

```
import numpy.random as nr
T=nr.uniform(low=ta,high=tb)
```

L'estimation de l'espérance ($\mathbb{E} []$) se fait avec une boucle qui moyenne les erreurs obtenues à chaque itération.

```
K=1000
err=0
for k in range(K):
    T=nr.uniform(low=ta,high=tb)
    ...
    err += (s-si)**2
err = np.sqrt(err/K)
```

Les trois points `...` remplacent les instructions qui calculent `s` le signal d'origine en `T` et `si` le signal échantillonné puis interpolé en `T`. `K` est le nombre d'itérations dans la boucle.

En terme de visualisation :

La figure 12.2 montre la façon dont l'erreur quadratique dépend de la fréquence d'échantillonnage. L'implémentation est réalisée dans la section A.16.

12.8.1 Reconstruction du signal par interpolation similaire à la formule de Shannon-Nyquist

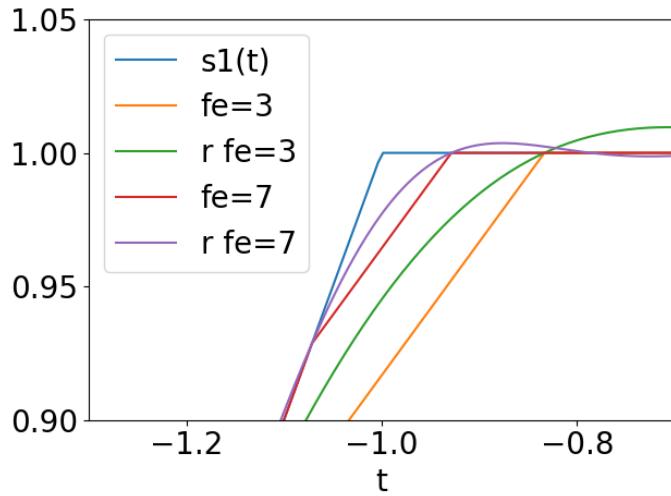


Figure 12.3: Signal étudié $s_1(t)$ et deux échantillonnages

La figure 12.3 un zoom sur la reconstruction du signal. L'implémentation est réalisée dans la section A.17.

12.8.2 Simulation de l'erreur quadratique moyenne en fonction de f_e après reconstruction

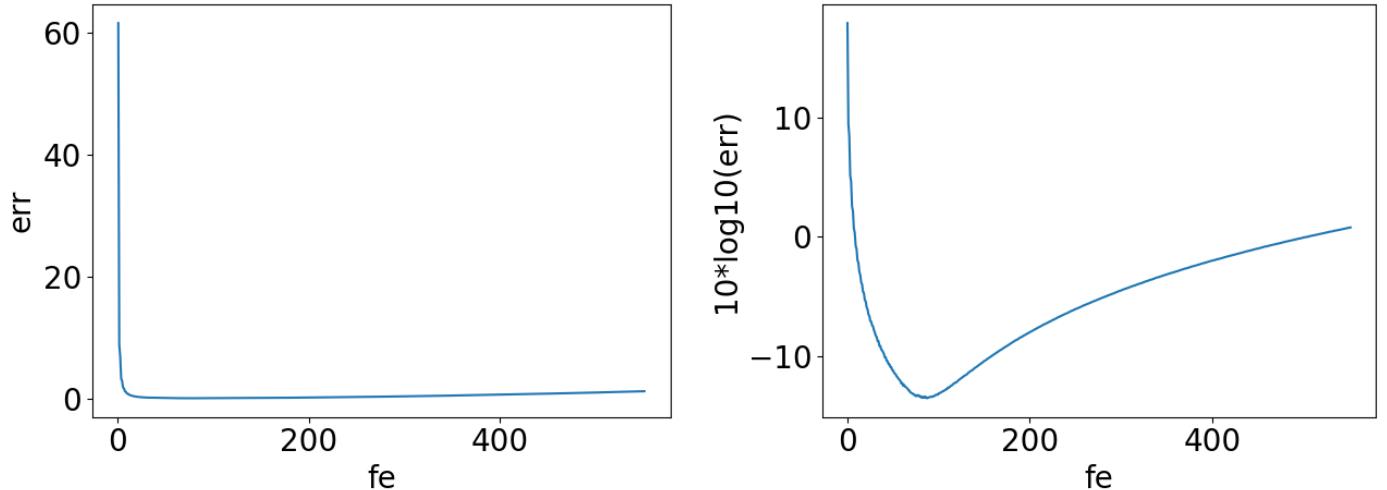


Figure 12.4: Signal étudié $s_1(t)$ et deux échantillonnages

La figure 12.4 un zoom sur la reconstruction du signal. L'implémentation est réalisée dans la section A.18.

Chapter 13

Modélisation stochastique du bruit

13.1 Qu'est-ce que le bruit ?

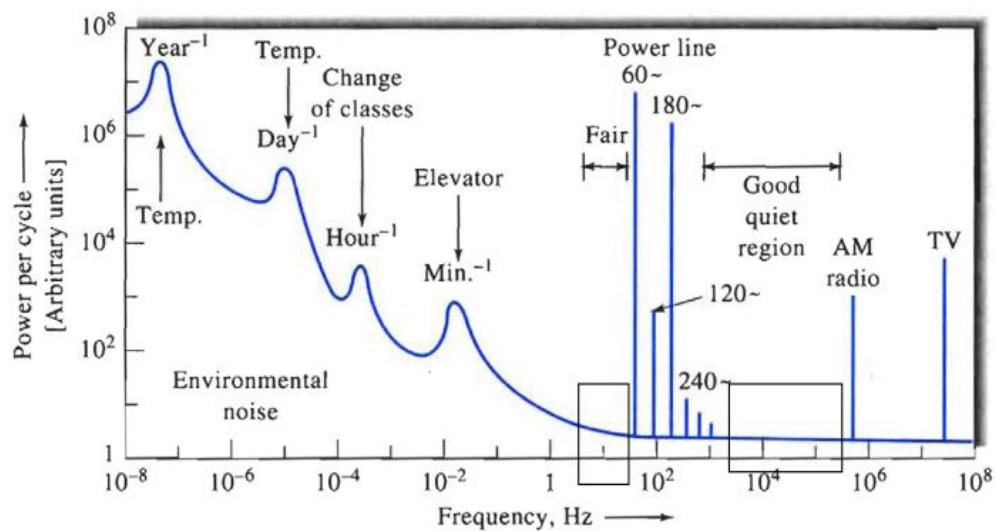


FIGURE 5-3 Some sources of environmental noise in a university laboratory. Note the frequency dependence and regions where various types of interference occur. (From T. Coor, *J. Chem. Educ.*, 1968, 45, A540. With permission.)

D'un point de vue électronique :

En électronique, on définit le bruit comme une conséquence de phénomènes physiques identifiés. Ces phénomènes produisent un écart à la valeur théorique, ces écarts sont généralement de moyennes nulles mais le carré de ces écarts n'est pas de moyenne nulle. Les sources du bruit sont :

- bruit thermique
- bruit de grenaille
- bruit en $1/f$
- bruit lié à la quantification d'un signal
- perturbation liée à l'alimentation
- perturbation électromagnétique

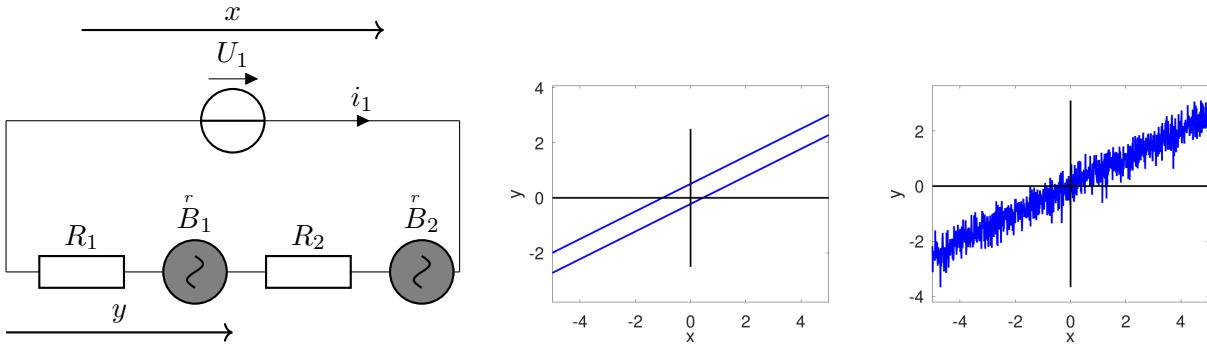


Figure 13.1: À gauche : diviseur de tension. Au milieu : graphe de y en fonction de x pour deux réalisations différentes. À droite : graphe de y en fonction de x lorsqu'on teste successivement pour chaque valeur de x la valeur de y .

Cours signal et bruit :

En théorie du signal, on définit le bruit en fonction de l'application envisagée, on la définit comme tout écart au modèle. Ce bruit est ensuite modélisé par un processus stochastique, c'est-à-dire un signal dont la valeur à chaque instant est lui-même aléatoire. Ainsi ce qui est un bruit pour une application donnée peut s'avérer le signal dans une autre application. De ce point de vue on distingue le bruit modélisé par un processus aléatoire et le bruit modélisé par un signal déterministe qui peut dépendre de certains signaux.

13.2 Bruit modélisé par une variable aléatoire

D'un point de vue électronique :

Un composant ne se comporte pas exactement tel qu'il est modélisé. La modélisation plus précise utilise de multiples composants supplémentaires, elle utilise aussi une source de tension ou d'intensité dont la valeur est modélisée par une variable aléatoire.

Exemple pour illustrer :

La figure 13.1 montre un diviseur de tension où chaque résistance est modélisée par une résistance sans bruit et une source de tension qui génère un bruit gaussien.

$$y = R \frac{x}{2} + \frac{1}{2} \overset{r}{B}_1 - \frac{1}{2} \overset{r}{B}_2 \quad (13.1)$$

D'un point de vue mathématique :

On appelle une **réalisation** le fait de procéder à un tirage aléatoire.

Une **variable aléatoire gaussienne** est définie par $\overset{r}{B} \sim \mathcal{N}(\mu, \sigma^2)$

$$E\left[\overset{r}{B}\right] = \mu \quad \text{et} \quad \text{Var}\left[\overset{r}{B}\right] = \sigma^2 \quad (13.2)$$

En général pour un bruit $\mu = 0$. On dit que la variable aléatoire est **centrée**.

Les signaux dont on a parlé dans les chapitres précédents ne dépendent pas d'un tirage aléatoire, on les appelle **signaux déterministes**.

La notation $\overset{r}{B}$ est spécifique à ce cours, elle est pratique parce qu'elle me permet de distinguer la notation d'un signal aléatoire de celle d'un signal déterministe.

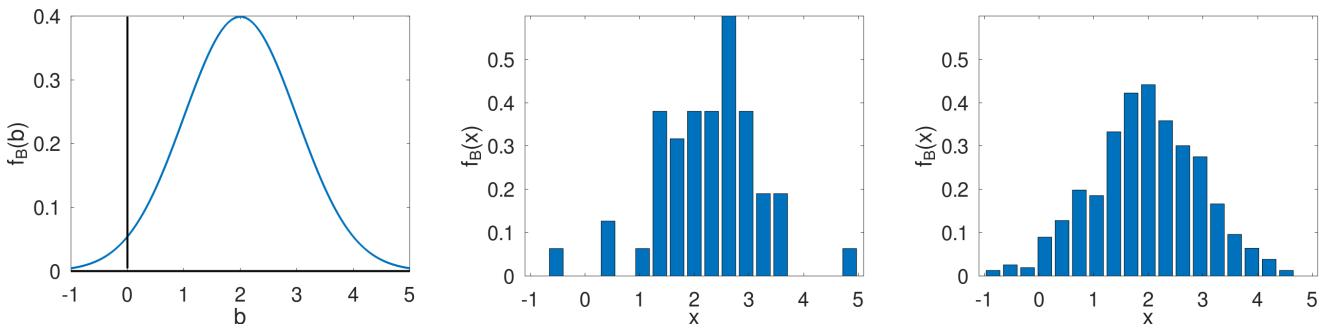


Figure 13.2: À gauche fonction Gaussienne pour $\mu = 2$ et $\sigma = 1$. Au milieu et à droite histogrammes obtenues avec 50 et 500 tirages aléatoires de \vec{B} suivant la loi $\mathcal{N}(2, 1)$.

D'un point de vue mathématique :

La densité de probabilité d'une gaussienne $\vec{B} \sim \mathcal{N}(\mu, \sigma)$ est

$$f_B(b) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\frac{(b-\mu)^2}{\sigma^2}} \quad (13.3)$$

Cette gaussienne est représentée à gauche de la figure 13.2.

$$E[g(\vec{B})] = \int_{-\infty}^{+\infty} f_B(b)g(b) db \text{ et } \text{Var}[\vec{B}] = E[\vec{B}^2] - E[\vec{B}]^2 = E\left[\left(\vec{B} - E[\vec{B}]\right)^2\right] \quad (13.4)$$

Pour un tirage d'un certain nombre de valeurs.

- 68 % des échantillons sont entre $\mu - \sigma$ et $\mu + \sigma$.
- 95 % des échantillons sont entre $\mu - 2\sigma$ et $\mu + 2\sigma$.
- 99.7 % des échantillons sont entre $\mu - 3\sigma$ et $\mu + 3\sigma$.

Il est important de distinguer la fonction gaussienne qui en tant que fonction est déterministe d'un tirage aléatoire dont la densité de probabilité est décrit par cette fonction gaussienne.

Les différentes sources de bruit sont généralement supposées indépendantes. Et dans ce cas

$$E[g_1(\vec{B}_1)g_2(\vec{B}_2)] = E[g_1(\vec{B}_1)]E[g_2(\vec{B}_2)] \quad (13.5)$$

où g_1 et g_2 sont deux fonctions quelconques.

Ajouter une constante à \vec{B} c'est modifier sa moyenne statistique.

$$E[\vec{B} + \mu'] = \mu + \mu' \quad (13.6)$$

Amplifier (ou atténuer) un bruit c'est modifier sa moyenne et son écart-type.

$$E[\lambda\vec{B}] = \lambda\mu \quad \text{et} \quad \text{Var}[\lambda\vec{B}] = \lambda^2\sigma^2 \quad (13.7)$$

Python :

La gauche de la figure 13.2 est une fonction déterministe. Pour la simuler il convient d'abord de faire une échelle des abscisses :

```
b = np.linspace(-5,5,10**3)
```

Puis on calcule la densité de probabilité pour chacune de ces valeurs de b en utilisant le fait que cette gaussienne est centrée en 2 et a un écart-type de 1 :

```
fb = 1/np.sqrt(2*np.pi)*np.exp(-(b-2)**2/2)
```

En fait on peut simplement utiliser la commande toute faite.

```
fb = seb.gaussian(b,2,1)
```

On vérifie qu'approximativement ceci est bien une densité de probabilité :

```
assert all(fb>=0)
assert np.abs(1-seb.TF(b,fb,0))<1e-3
```

Le milieu et la droite de la figure 13.2 sont obtenues avec des tirages aléatoires. On utilise la fonction `normal` du module `numpy.random`, ses paramètres sont la moyenne, l'écart-type et le nombre de termes à tirer aléatoirement. La fonction renvoie un vecteur de type `numpy.ndarray`

```
import numpy.random as rn
B1 = rn.normal(2,1,50)
B2 = rn.normal(2,1,500)
```

À partir des valeurs tirées aléatoirement, on réalise deux histogrammes.

```
fb1,b1=np.histogram(B1, bins=int(np.sqrt(len(B1))),density=True)
fb2,b2=np.histogram(B2, bins=int(np.sqrt(len(B2))),density=True)
```

Ici `b1` et `b2` sont les 8 et 23 extrémités des intervalles qui subdivisent les différentes valeurs de `B1` et `B2`. C'est-à-dire que la première valeur de `b1` est la valeur minimale de `B1`.

```
assert np.min(B1)==b1[0]
assert np.max(B1)==b1[-1]
```

Les intervalles sont de même tailles et cette taille est `max B1-min B1` divisé par le nombre de ces intervalles qui est le nombre de composantes de `fb1`.

```
assert all(np.abs(np.diff(b1)-(np.max(B1)-np.min(B1))/len(fb1))<=1e-10)
```

`fb1` est une estimation de la densité de probabilité associée à chacun de ces intervalles, il y a donc une composante en moins par rapport au nombre de composantes de `b1`.

```
assert len(b1)==1+len(fb1)
```

Le fait de choisir dans la commande `numpy.histogram`, `density=True` fait que la densité de probabilité est normalisée de façon à ce qu'approximativement l'intégrale vaille 1.

```
assert np.abs(1-np.sum(fb1)*(b1[1]-b1[0]))<1e-10
```

Pour afficher le graphe il est pratique d'utiliser la fonction `stairs` de `numpy` qui est très similaire à `plot` sauf que l'ordre des arguments est inversé. L'échelle des abscisses est le deuxième argument.

```
ax.stairs(fb1,b1)
```

La différence est qu'on met ici la densité de probabilité avant les extrémités des intervalles.

13.3 Covariance, corrélation et indépendance entre variables aléatoires

En termes probabilité :

Le terme covariance entre deux variables aléatoires signifie

$$\text{Cov}(\overset{r}{X}, \overset{r}{Y}) = E \left[\left(\overset{r}{X} - E[\overset{r}{X}] \right) \left(\overset{r}{Y} - E[\overset{r}{Y}] \right) \right] \quad (13.8)$$

Le terme **corrélation** entre deux variables aléatoires signifie

$$\text{Cor}(\overset{r}{X}, \overset{r}{Y}) = \frac{\mathbb{E}\left[\left(\overset{r}{X} - \mathbb{E}[\overset{r}{X}]\right)\left(\overset{r}{Y} - \mathbb{E}[\overset{r}{Y}]\right)\right]}{\sqrt{\text{Var}[\overset{r}{X}]}\sqrt{\text{Var}[\overset{r}{Y}]}} \quad (13.9)$$

Deux variables aléatoires sont dites **indépendantes au second ordre** lorsque $\text{Cov}(\overset{r}{X}, \overset{r}{Y}) = 0$. Deux variables aléatoires sont **indépendantes** lorsque

$$\text{Pour tout } x, y, \quad \mathbb{P}(\overset{r}{X} \leq x \text{ et } \overset{r}{Y} \leq y) = \mathbb{P}(\overset{r}{X} \leq x)\mathbb{P}(\overset{r}{Y} \leq y) \quad (13.10)$$

On a aussi une inégalité

$$|\text{Cov}(\overset{r}{X}, \overset{r}{Y})| \leq \sqrt{\text{Var}[\overset{r}{X}]}\sqrt{\text{Var}[\overset{r}{Y}]} \text{ et } |\text{Cor}(\overset{r}{X}, \overset{r}{Y})| \leq 1 \quad (13.11)$$

Cours signal et bruit :

On utilise le terme **corrélation** entre deux variables aléatoires pour désigner $\mathbb{E}[\overset{r}{X}\overset{r}{Y}]$ mais on ne l'utilise que quand on sait que $\overset{r}{X}$ et $\overset{r}{Y}$ sont de moyennes statistiques nulles.

On utilise aussi le terme **indépendance** dans l'expression deux variables aléatoires sont indépendantes au sens de l'équation (13.10). Mais souvent ce qui nous intéresse dans le domaine du signal et du bruit ce sont les propriétés au premier et au deuxième ordre. Aussi l'**indépendance** désigne parfois simplement le fait qu'elles ne sont pas corrélées, c'est-à-dire au sens probabiliste du terme que leur covariance est nulle comme décrit par l'équation (13.9).

Dans le cadre de ce cours, je propose de toujours préciser s'il s'agit de l'**indépendance au sens strict** ou s'il s'agit de l'**indépendance au second ordre**.

13.4 Processus aléatoire

13.4.1 Généralités

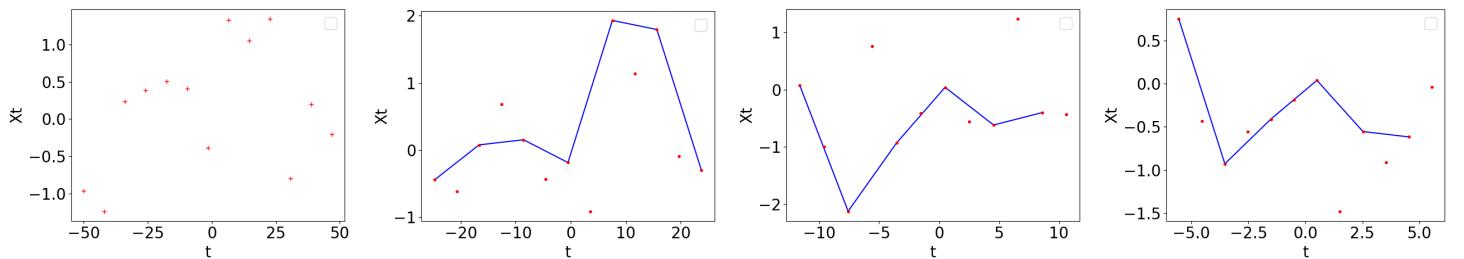


Figure 13.3: Extrait d'une réalisation d'un bruit blanc et zoom successif permettant de compléter les points avec des voisins.

Cours signal et bruit :

Un **processus aléatoire** peut se voir comme une variable aléatoire dépendant du temps.

- $\overset{r}{X}(t)$ est un processus aléatoire
- $x(t) = \overset{r}{X}(t, \omega)$ est une réalisation particulière de $\overset{r}{X}(t)$.
- $\overset{r}{X}(t_0)$ est une variable aléatoire

La difficulté vient de ce que $\overset{r}{X}(t)$ pourrait dépendre des autres instants.

13.4.2 Processus aléatoire blanc

On définit un processus aléatoire où il n'y a aucune dépendance.

$\overset{r}{X}(t)$ est un **bruit blanc** signifie que

$$t_0 \neq t_1, \Rightarrow \overset{r}{X}(t_0) \text{ est indépendant de } \overset{r}{X}(t_1) \quad (13.12)$$

En termes numériques :

La simulation d'une partie d'une réalisation d'un processus aléatoire dépend d'un vecteur \mathbf{t} correspondant à une échelle de temps et caractérisée par un début t_0 , une fin $t_0 + T$ et un nombre de points N associé à une période d'échantillonnage T_e et une fréquence d'échantillonnage f_e . Le vecteur de temps est la liste de N valeurs espacées de T_e notées t_n

$$T = NT_e \quad f_e = \frac{1}{T_e} \quad t_n = nT_e + t_0 \quad t_{N-1} = T - T_e + t_0 \quad T_e = t_1 - t_0 \quad (13.13)$$

Il se trouve que pour que la théorie du traitement du signal fonctionne il faut que tous les instants t_n soient des multiples de T_e .

En terme de visualisation :

La figure 13.3 montre dans ses quatre graphiques un même signal qui est une réalisation d'un bruit blanc à bande limitée sur $[-10^6, 10^6]$ en Hz. Le premier graphique montre son évolution entre -50 et 50 s, le deuxième est un zoom entre -25 et 25 s, le troisième est encore un zoom entre -12.5 et 12.5 s, le quatrième est encore un zoom entre -6.25 et 6.25 s. Chaque graphique ne montre que 12 points. Pour les graphiques 2,3,4, les traits bleus joints par des pointillés sont les 6 points déjà montrés respectivement dans les graphiques 1,2,3. Les 12 points en bleus sont pour 6 déjà présents dans le graphe à gauche et 6 sont nouveaux. Ceci est implémentée dans A.9.

Python :

Pour trouver les valeurs du bruit blanc associé à une échelle de temps \mathbf{t} , on utilise $\mathbf{t}.shape[0]$ pour avoir le nombre de composantes du vecteur \mathbf{t} .

```
import numpy.random as nr
B=nr.normal(2,1,len(t))
```

D'un point de vue mathématique :

Mathématiquement, un bruit blanc temps continu n'est pas défini. Si on tient à la notion de temps continu, on utilise un processus de Wiener aussi appelé *Brownian Motion*. Il est continu par rapport au temps et sert de brique de base pour définir d'autres processus. En autorisant un nombre infini de sauts non-infiniment petit, l'utilisation de la notion de filtrage fait perdre la notion de densité de probabilité.

Cours signal et bruit :

Dans ce cours, on considère un bruit blanc à temps continu comme étant défini à partir d'un signal temps discret échantillonné à une fréquence d'échantillonnage f_e et dont les composantes sont des variables aléatoires notées $\overset{r}{B}_n$ indépendantes de moyenne nulle et d'écart-type 1. À partir de ce signal on peut construire $\overset{r}{X}_n = \sigma \overset{r}{B}_n + \mu$, μ et σ^2 étant la moyenne statistique et la variance de $\overset{r}{X}_n$. Je met à gauche les notations classiques pour les signaux à temps discret et à droite les notations correspondantes quand on souhaite utiliser les distributions de Dirac.

- Le signal

$$\overset{r}{B}_n$$

$$\overset{r}{B}(t) = \sum_{n=-\infty}^{+\infty} \overset{r}{B}_n \delta(t - nT_e)$$

- La moyenne statistique

$$\mathbb{E}[\overset{r}{X}_n] = \mu$$

$$\mathbb{E}[\overset{r}{X}(t)] = \sum_{n=-\infty}^{+\infty} \mu \delta(t - nT_e)$$

- La variance

$$\text{Var}[\overset{r}{X}_n] = \sigma^2$$

$$\text{Var}[\overset{r}{X}(t)] = \sum_{n=-\infty}^{+\infty} \sigma^2 \delta(t - nT_e)$$

- La corrélation

$$\text{E}[\overset{r}{B}_n \overset{r}{B}_{n-k}] = \delta_k = [\![k=0]\!][\![k]\!]$$

$$\text{E}[\overset{r}{B}(t) \overset{r}{B}(t-\tau)] = \sum_{n=-\infty}^{+\infty} \delta(t - nT_e) \delta(\tau)$$

- L'énergie est infinie

$$E_B = \sum_{n=-\infty}^{+\infty} \text{E}[\overset{r}{B}_n^2] = +\infty$$

$$E_B = +\infty$$

- La puissance est finie

$$P_B = \lim_{N \rightarrow +\infty} \frac{1}{2N+1} \sum_{n=-N}^{+N} \text{E}[\overset{r}{B}_n^2] = \mu^2 + \sigma^2 \quad P_B = \mu^2 + \sigma^2$$

Attention le produit des distributions n'est pas défini en général. Ce qui est souvent défini, c'est le produit d'une distribution dépendant d'une variable par une autre distribution dépendant d'une autre variable, ce qu'on appelle un produit tensoriel et qui permet par exemple de définir le produit de convolution et dans ce cadre le produit apparaît comme la restriction à une valeur d'une distribution, ce qui pose bien des problèmes. En fait le plus simple est de faire les produits de suites sans utiliser de Diracs et de réécrire le résultat à droite.

Chapter 14

Autocorrélation et densité spectrale

14.1 Présentation physique du bruit thermique

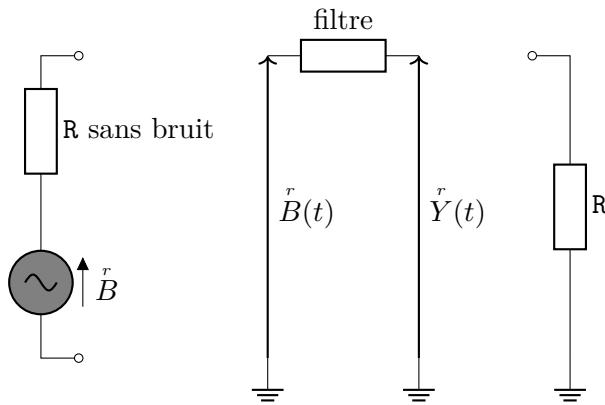


Figure 14.1: Modélisation du bruit thermique par un bruit en tension $\overset{r}{B}$ en série avec une résistance sans bruit amplifié d'un facteur a par un filtre à bande limitée ΔB .

D'un point de vue électronique :

Une résistance R , avec ou sans courant, provoque un bruit thermique dont la **puissance électronique par unité de fréquence** appelé **[densité spectrale de puissance]**, son unité est WHz^{-1} qui est homogène à un Joule J mais ne correspond pas du tout à cette notion.

$$P_{W/\text{Hz}} = 4k_B T \quad (14.1)$$

- k_B est la constante de Boltzmann au sens de la physique.
- T est la température en degré Kelvin au sens de la physique.

Ce bruit est aussi appelé *thermal noise* ou *Johnson noise*. Le bruit est modélisé par une nouvelle source de bruit en tension comme le montre la gauche de la figure 14.1. À partir de $P_{W/\text{Hz}}$ et la résistance on récupère la densité spectrale de la puissance qui fournit une telle puissance électronique par unité de fréquence

$$S_B(f) = P_{W/\text{Hz}} R \quad (14.2)$$

Ce bruit pourrait laisser penser qu'il est infini. En réalité la mécanique quantique amène à donner une autre formulation plus précise utilisant la constante de Planck $\hbar \approx 6 \times 10^{-34}$

$$P_{W/\text{Hz}} = 4 \frac{\hbar f}{e^{\frac{\hbar f}{k_B T}} - 1} \quad (14.3)$$

En utilisant un changement de variable $u = \frac{\hbar f}{k_B T}$ et en utilisant que $\int_0^{+\infty} \frac{u}{e^u - 1} du = \frac{\pi^2}{6}$, on trouve qu'au total la puissance disponible avec la résistance est de l'ordre de 80nW et celle qu'on pourrait idéalement récupérer serait de

$\frac{80}{2} = 40\text{nW}$. L'approximation de l'équation (14.2) est valable pour des fréquences inférieures à 10THz. Pour observer ce bruit thermique (et par exemple mesurer la constante de Boltzmann k_B), le signal aléatoire $\overset{r}{B}(t)$ est amplifié et filtré en $\overset{r}{Y}(t)$ comme le montre le milieu de la figure 14.1.

$$\overset{r}{Y}(t) = h(t) * \overset{r}{B}(t) \quad (14.4)$$

Ce filtre est caractérisé par une amplification a et une bande de fréquence \mathcal{B} aussi

$$S_Y(f) = a^2 P_{W/Hz} R [f \in \mathcal{B}] \quad (14.5)$$

À partir de $S_Y(f)$ et en notant ΔB la largeur de l'intervalle \mathcal{B} , on récupère la puissance de $\overset{r}{Y}(t)$, noté P_y

$$P_y = a^2 P_{W/Hz} R \Delta B \quad (14.6)$$

et on récupère la variance du bruit

$$\text{Var} \left(\overset{r}{Y}(t) \right) = P_y = a^2 P_{W/Hz} R \Delta B \quad (14.7)$$

Une résistance est mise en sortie du filtre, elle dissipe une puissance électronique donnée par

$$P_W = \frac{P_y}{R} = a^2 P_{W/Hz} \Delta B \quad (14.8)$$

14.2 Définition caractéristiques stochastiques

14.2.1 Distinction énergie finie ou infinie

D'un point de vue mathématique :

La distinction entre énergie finie et infinie est importante sur le plan théorique parce qu'elle pose des problèmes mathématiques, comment définir la transformée de Fourier pour un signal d'énergie infinie et comment définir le carré d'un signal si un signal est défini au moyen de distributions pour lesquels le produit n'a pas de sens.

En termes numériques :

En pratique, un signal est une modélisation d'un phénomène réel et un même phénomène peut souvent être approché à la fois par un signal d'énergie finie et d'énergie infinie. Mais les problèmes théoriques ont des conséquences dans la pertinence de l'approximation.

Cours signal et bruit :

Souvent on précise les termes autocorrélation et densité spectrale en ajoutant l'expression *de puissance ou d'énergie* pour dire si on utilise la formule adaptée aux signaux d'énergie infinie ou d'énergie finie.

Cas des signaux déterministes

Pour les signaux déterministes, on a distingué le cas des signaux périodiques et non-périodiques. En fait implicitement cette distinction recouvre une autre distinction, celles des signaux d'énergie infinie et des signaux d'énergies finies.

- Un signal périodique non-nul est d'énergie infinie.
- Un signal d'énergie finie est non-périodique sauf s'il est nul.
- Dans le cadre de ce cours les signaux d'énergies infinies considérées sont périodiques (mais naturellement ce n'est pas du tout vrai en général).
- Pour les signaux d'énergie infinie qui dans ce cours sont périodiques, on a une formule très simple pour la puissance.

Cette observation est vrai pour les signaux à temps continu et les signaux à temps discret où l'énergie est définie par $E_s = \sum_{n=-\infty}^{+\infty} s_n^2$.

Cas des processus aléatoires

Pour les processus aléatoires, l'énergie est définie quand il est à temps continu par

$$E_x = \int_{-\infty}^{+\infty} E[\overset{r}{X}(t)] dt \quad (14.9)$$

et à temps discret par

$$E_x = \sum_{n=-\infty}^{+\infty} E[\overset{r}{X}_n^2] \quad (14.10)$$

Dans le cadre de ce cours, les processus aléatoires sont tous d'énergie infinie et ces expressions sont délicates à utiliser. Un processus aléatoire d'énergie infinie ne peut pas être périodique, mais la façon de traiter un tel processus ressemble au cas périodique. On utilise la puissance au lieu de l'énergie et cette puissance P_x est définie dans le cas temps continu.

$$P_x = \lim_{T \rightarrow +\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{+\frac{T}{2}} E[\overset{r}{X}(t)] dt \quad (14.11)$$

et dans le cas temps discret

$$P_x = \lim_{N \rightarrow +\infty} \frac{1}{2N+1} \sum_{n=-N}^{+N} E[\overset{r}{X}_n^2] \quad (14.12)$$

14.2.2 Cas du bruit blanc tel que défini dans la section 13.4

Cours signal et bruit :

Un bruit blanc est dans ce cours un signal d'énergie infinie. On utilise les notations du signal temps discret.

- L'autocorrélation

$$R_B[k] = \delta_k, \quad E[\overset{r}{B}_n \overset{r}{B}_{n-k}] = \delta_k \text{ et } \text{Var}[\overset{r}{B}_n] = 1 \quad (14.13)$$

- La densité spectrale

$$S_B(f) = 1 \quad (14.14)$$

14.2.3 Cas d'un processus aléatoire centré

On considère un processus aléatoire centré (et d'énergie infinie). On utilise pour simplifier les notations d'un signal temps discret.

- La moyenne statistique et la variance

$$E[\overset{r}{X}_n] = 0 \text{ et } \text{Var}[\overset{r}{X}_n] \quad (14.15)$$

- L'autocorrélation est définie

$$R_X[k] = \lim_{N \rightarrow +\infty} \frac{1}{2N+1} \sum_{n=-N}^{+N} E[\overset{r}{X}_n \overset{r}{X}_{n-k}] \quad (14.16)$$

- La densité spectrale est définie à partir de l'autocorrélation

$$S_X(f) = \text{TFTD}[R_X[k]](f) \quad (14.17)$$

Elle est aussi définie à partir du processus aléatoire lui-même

$$S_X(f) = \lim_{N \rightarrow +\infty} \frac{1}{2N+1} E \left[\left| \sum_{n=-N}^{+N} \overset{r}{X}_n e^{-j2\pi f n T_e} \right|^2 \right] \quad (14.18)$$

On dit qu'un processus aléatoire est **stationnaire** quand ces caractéristiques ne dépendent pas de n . Cela ne signifie pas que le processus aléatoire ne dépende pas du temps, seulement que ces caractéristiques stochastiques ne dépendent pas du temps.

- $E[\overset{r}{X}_n] = 0$ et $\text{Var}[\overset{r}{X}_n]$ ne dépendent pas de n
- $E[\overset{r}{X}_n \overset{r}{X}_{n-k}]$ ne dépend pas de n et dans ce cas [il vaut aussi $R_X[k]$ et s'appelle alors l'autocorrélation].

Un processus aléatoire est **ergodique** quand on peut retrouver toutes les caractéristiques à partir d'une seule réalisation d'un processus aléatoire.

Quand le processus n'est pas centré, on le remplace l'utilisation des définitions par le processus moins sa moyenne statistique.

Exemple pour illustrer :

Le processus aléatoire proposé est défini par $E[\overset{r}{X}(t)] = 0$ et $R_x(t) = e^{-|t|}$. Sa densité spectrale est $S(f) = \frac{1}{1+4\pi^2f^2}$. Son énergie est alors $E_x = R_x(0) = E[\overset{r}{X}^2(t)] = \text{Var}(\overset{r}{X}(t))$.

On peut faire sur Python les simulations transformant $R_x(t)$ en $S_x(f)$ et vice-versa. On verra dans le chapitre suivant comment faire des simulations du processus lui-même.

14.2.4 Cas d'un processus déterministe d'énergie finie

On peut donner un sens à toutes ces notions dans le cas où le processus aléatoire est un signal déterministe à énergie finie, mais les formules sont adaptées du fait que l'énergie est finie.

- Un tel signal est par définition ergodique puisque toutes les réalisations sont identiques.
- Un tel signal n'est pas stationnaire sinon il serait constant.
- Il n'est pas centré sinon il serait nul.
- La variance est nulle car il est déterministe.
- L'autocorrélation est ainsi définie

$$R_x(\tau) = \int_{-\infty}^{+\infty} x(t)x(t-\tau) dt \quad (14.19)$$

- La densité spectrale est définie de la même façon

$$S(f) = \text{TF}[R_x(t)](f) \quad (14.20)$$

- Pour l'énergie, on a bien

$$R_x(0) = E_x = \int_{-\infty}^{+\infty} x^2(t) dt \quad (14.21)$$

- On a aussi les deux propriétés suivantes

$$S(f) = |\widehat{X}(f)|^2 \text{ et } E_x = \int_{-\infty}^{+\infty} S(f) df \quad (14.22)$$

Les propriétés sur l'autocorrélation et la densité spectrale sont aussi vraies.

Exemple pour illustrer :

Quand on considère $x(t) = e^{-t} \mathbb{I}[t \geq 0](t)$, l'autocorrélation est $R_x(t) = e^{-|t|}$, cet exemple vérifie bien les propriétés précédentes (symétrie $R_x(t)$, majoration de $R_x(t)$ par 1 et positivité de $S(f)$). Du processus aléatoire, on sait que

$$E[\overset{r}{X}(t)] = x(t), \quad E[\overset{r}{X}^2(t)] = x^2(t), \quad E[\overset{r}{X}(t)\overset{r}{X}(t-\tau)] \neq e^{-|\tau|}, \text{ et } E\left[\left|\text{TF}[\overset{r}{X}(t)](f)\right|^2\right] = |\widehat{X}(f)|^2 = S_x(f) \quad (14.23)$$

14.2.5 Cas d'un processus quasi-déterministe

L'enjeu de cette partie est de raccorder le cours à ce que est le plus souvent étudié en traitement du signal aléatoire, mais cette partie à la limite du cours.

On appelle un processus aléatoire quasi-déterministe un processus aléatoire défini par une fonction déterministe ayant des paramètres qui sont en fait des variables aléatoires. L'exemple considéré ici est d'énergie infinie, on note $\overset{r}{A} \sim \mathcal{N}(0, 1)$ une variable aléatoire gaussienne et $\overset{r}{\varphi} \sim \mathcal{U}(0, 2\pi)$ une variable aléatoire uniforme, on définit

$$\overset{r}{X}(t) = \overset{r}{A} + \cos(2\pi t + \overset{r}{\varphi}) \quad (14.24)$$

- Ce processus n'est pas ergodique. Par exemple si on considère la réalisation ou $\overset{r}{A}(\omega) = 0$, on a aucune information comme quoi il pourrait y avoir une composante continue.

- Ce processus est centré

$$E[\overset{r}{X}(t)] = E[\overset{r}{A}] + \frac{1}{2\pi} \int_0^{2\pi} \cos(2\pi t + \varphi) d\varphi = 0 + 0 \quad (14.25)$$

car la primitive de $\varphi \mapsto \cos(2\pi t + \varphi)$ est périodique de période 2π .

- Ce processus est centré

$$\text{Var}[\overset{r}{X}(t)] = \text{Var}[\overset{r}{A}] + \frac{1}{2\pi} \int_0^{2\pi} \cos^2(2\pi t + \varphi) d\varphi = 1 + \frac{1}{2} = \frac{3}{2} \quad (14.26)$$

car $\cos^2(2\pi t + \varphi) = \frac{1}{2} + \frac{1}{2} \cos(4\pi t + \varphi)$ et la primitive de $\varphi \mapsto \cos(4\pi t + \varphi)$ est périodique de période 2π .

- L'autocorrélation est ainsi définie

$$R_x(\tau) = \lim_{T \rightarrow +\infty} \frac{1}{T} \int_{-T/2}^{+T/2} E[\overset{r}{X}(t) \overset{r}{X}(t - \tau)] dt \quad (14.27)$$

Par un calcul trigonométrique, on a

$$\cos(2\pi t + \varphi) \cos(2\pi(t - \tau) + \varphi) = \frac{1}{2} \cos(4\pi t - 2\pi\tau + 2\varphi) + \frac{1}{2} \cos(2\pi\tau) \quad (14.28)$$

Donc

$$\begin{aligned} E[\overset{r}{X}(t) \overset{r}{X}(t - \tau)] &= E\left[\left(\overset{r}{A} + \cos(2\pi t + \varphi)\right) \left(\overset{r}{A} + \cos(2\pi(t - \tau) + \varphi)\right)\right] \\ &= E[\overset{r}{A}^2] + E[\overset{r}{A} (\cos(2\pi t + \varphi)) + \cos(2\pi(t - \tau) + \varphi)] + \cos(2\pi t + \varphi) \cos(2\pi(t - \tau) + \varphi) \\ &= 1 + 0 + \frac{1}{2} \cos(2\pi\tau) \end{aligned} \quad (14.29)$$

Ce calcul montre que le signal considéré est stationnaire car $E[\overset{r}{X}(t) \overset{r}{X}(t - \tau)]$ ne dépend que de τ et non de t .

Finalement on a

$$R_x(\tau) = \lim_{T \rightarrow +\infty} \frac{1}{T} \int_{-T/2}^{+T/2} 1 + \frac{1}{2} \cos(2\pi\tau) dt = 1 + \frac{1}{2} \cos(2\pi\tau) \quad (14.30)$$

- La densité spectrale est définie de la même façon $S(f) = \text{TF}[R_x(t)](f)$

$$S(f) = \delta(f) + \frac{1}{4}\delta(f - 1) + \frac{1}{4}\delta(f + 1) \quad (14.31)$$

- Pour la puissance, on a bien

$$P_x = \lim_{T \rightarrow +\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{+\frac{T}{2}} E[\overset{r}{X}^2(t)] dt = R_x(0) = \frac{3}{2} \quad (14.32)$$

- Les propriétés de l'autocorrélation sont bien vérifiées.

14.2.6 Définition d'un processus aléatoire approchant un bruit blanc au moyen de motifs

On considère ici une autre façon de définir un bruit blanc. On considère toujours une fréquence d'échantillonnage f_e et un ensemble de variable aléatoire centrée, d'écart-type 1 et indépendantes. On considère aussi un motif décrit par un signal temps continu déterministe noté $m(t)$ et non-nul en dehors d'un intervalle pas plus grand que $T_e = \frac{1}{f_e}$. Cette condition assure que les motifs ne se chevauchent pas. Ce motif $m(t)$ est d'énergie finie, mais le bruit blanc associé est d'énergie infinie.

$$\overset{r}{B}_m(t) = \sum_{n=-\infty}^{+\infty} \overset{r}{B}_n m(t - nT_e) \quad (14.33)$$

En tant que fonction déterministe $m(t)$ a une autocorrélation notée $R_m(t)$ et une densité spectrale notée $S_m(f)$ en utilisant les définitions associées à l'énergie finie et données par les équations 14.19 et 14.20.

L'autocorrélation de $\overset{r}{B}_m(t)$ définie par l'équation (14.16) est

$$R_B(t) = \frac{1}{T_e} R_m(t) \quad (14.34)$$

Et sa densité spectrale est

$$S_B(f) = \frac{1}{T_e} S_m(f) \quad (14.35)$$

14.3 Propriété de l'autocorrélation et de la densité spectrale

14.3.1 Processus aléatoire à temps continu

Les propriétés de l'autocorrélation et de la densité spectrale sont communes aux signaux d'énergie finie et infinie.

- L'autocorrélation en $t = 0$ est relié à la puissance ou l'énergie

$$P_x = R_x(0) = \int_{-\infty}^{+\infty} S_x(f) df \text{ ou } E_x = R_x(0) = \int_{-\infty}^{+\infty} S_x(f) df \quad (14.36)$$

- L'autocorrélation est maximale en $t = 0$

$$|R_x(t)| \leq R_x(0) \quad (14.37)$$

- La densité spectrale est positive

$$S_x(f) \geq 0 \quad (14.38)$$

14.3.2 Processus aléatoire à temps discret

On a les propriétés suivantes :

- Pour un processus aléatoire centré, ce qu'on appelle la puissance est $P_x = R_x[0]$.
- L'autocorrélation est symétrique et majorée par la puissance

$$R_x[-k] = R_x[k] \text{ et } |R_x[k]| \leq P_x = R_x[0] \quad (14.39)$$

- La densité spectrale est périodique de période f_e et est toujours positive $S_x(f) \geq 0$.

- On a aussi

$$P_x = \frac{1}{f_e} \int_{-f_e/2}^{+f_e/2} S_x(f) df \quad (14.40)$$

14.4 Simulation de processus aléatoires

Les simulations qui suivent illustrent de façon très imprécise que les caractéristiques stochastiques ne dépendent pas de la fréquence d'échantillonnage en ce sens que les simulations sont un peu proches des valeurs théoriques elles-mêmes ne dépendant pas de f_e . Pour obtenir ce résultat qui est nécessaire pour que les simulations aient un sens, il faut :

- Diviser par la durée du signal quand on intègre sur la durée du signal (cas de l'autocorrélation et de la transformée de Fourier)
- Utiliser la TFTD pour passer de l'autocorrélation à la densité spectrale.

14.4.1 Bruit blanc échantillonné

En terme de visualisation :

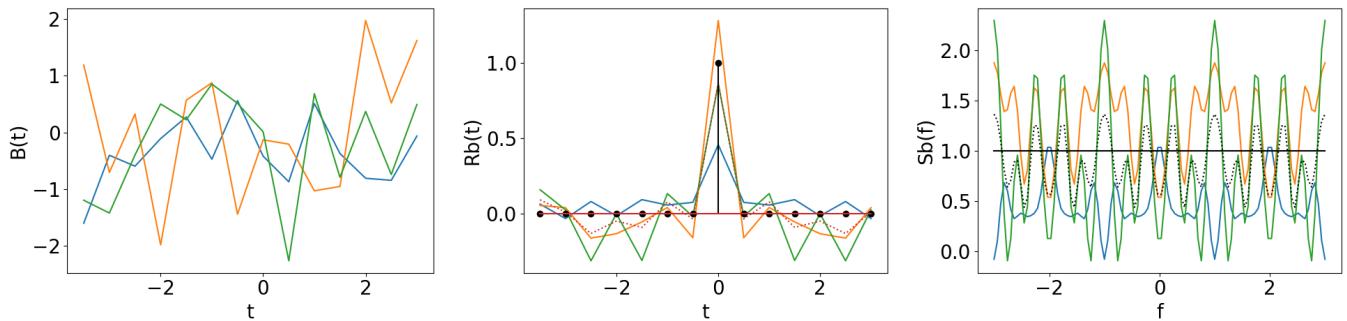


Figure 14.2: Bruit blanc échantillonné : réalisation de trois trajectoires, trois autocorrélations, trois densités spectrales.

La figure 14.2 montre un bruit blanc échantillonné. L'implémentation est dans A.21.

Simulation d'une trajectoire

J'illustre la façon de procéder avec la gauche de la figure 14.2.

On se donne un fréquence d'échantillonnage $f_e = 2$ (il y a deux points par unité de temps, en apparence, il y en a plus mais ce sont les bornes absentes qui donnent cette impression). On créé une échelle de temps qui va de $-T/2$ à $T/2$. (seb.arange évite les problèmes de synchronisation).

```
T      = 7
fe    = 2
Te    = 1/fe
t     = seb.arange(-T/2,T/2,1/fe)
```

On génère trois réalisations de bruits blanc.

```
b1,b2,b3 = nr.normal(0,1,len(t)), nr.normal(0,1,len(t)), nr.normal(0,1,len(t))
```

Simulation de l'autocorrélation

On utilise la même échelle de temps, d'autant qu'elle est déjà symétrique par rapport à $t = 0$. Théoriquement l'autocorrélation est donnée par δ_k

```
R_th = (t==0)+0
```

On peut regarder les valeurs de $\hat{B}_n \hat{B}_{n-k}$ pour une certaine valeur de n et quelques valeurs de k , ici $\{0, 1, 2, 3\}$.

```
ind = seb.where_nearest(t,0)
k   = np.arange(4)
print(np.round(b1[ind]*b1[ind-k]*100)/100)
print(np.round(b2[ind]*b2[ind-k]*100)/100)
print(np.round(b3[ind]*b3[ind-k]*100)/100)
```

L'affichage donne

$$\begin{array}{cccc} 0.41 & 0.27 & -1.38 & 0.01 \\ 3.81 & 0.6 & 0.61 & -0.92 \\ 0.25 & -0.45 & -0.54 & 0.83 \end{array} \quad (14.41)$$

Il est difficile de prétendre que ces trois lignes valeurs tournent autour de la suite 1, 0, 0, 0. Par contre si on moyenne sur un très grand nombre de tirages aléatoire c'est ce qu'on devrait trouver.

Une autre façon de faire est de moyenner sur les valeurs de n , c'est ce que simule l'autocorrélation qui estimée ici par

$$R_b[k] \approx \frac{1}{T} \sum_{n=-N/2}^{N/2} B_n^\omega B_{n-k}^\omega T_e \quad (14.42)$$

avec $N = 2\lfloor \frac{T}{2T_e} \rfloor$. Les indices $n - k$ qui débordent les entiers $-N/2 \dots N/2$ sont exclues donnent une valeur nulle à B_{n-k} et B_n^ω est la composante n d'une réalisation du bruit blanc. La multiplication par T_e est ici considérée parce qu'elle permet de rendre la somme une approximation d'une intégrale. Du coup il faut diviser par $NT_e = T$ au lieu de N .

En utilisant `seb.correlation`, on a finalement

```
Rb1 = seb.correlation(t,b1,t,b1,t)/T
```

et de même pour `Rb2` et `Rb3`. T_e est pris en compte dans le programme `seb.correlation`.

Sur la partie centrale de la figure 14.2, la courbe noire est l'autocorrélation théorique, les courbes colorées sont les simulation `Rb1`, `Rb2`, `Rb3` et la courbe en pointillé est la moyenne de ces trois courbes, on observe que cette moyenne est plus proche de la valeur théorique.

Densité spectrale

La densité spectrale peut s'obtenir soit à partir du signal lui-même, soit à partir de l'autocorrélation, c'est cette deuxième technique qui est utilisée ici. Je créé une échelle de fréquence, dont les caractéristiques dépendent d'abord de ce que l'on souhaite afficher.

```
f = np.linspace(-3,3,10**2)
```

En choisissant une valeur plus grande que $f_e/2$, je montre sur la figure qu'il s'agit d'une densité spectrale périodique de période f_e .

Cette densité spectrale est réelle.

```
Gb1 = np.real(seb.TFTD(t,Rb1,f))
```

La partie droite de la figure 14.2 montrent différentes densités spectrales, on voit qu'elles varient de façon importante. Celles qui est en pointillé est la moyenne et est un tout petit peu proche de la droite ligne.

Dans ces simulations, j'ai privilégié la simplicité, mais il faudrait modifier ces simulations pour vraiment se convaincre que les simulations convergent vers les valeurs théoriques.

Chapter 15

Filtrage des processus aléatoires et application au bruit en $1/f$

15.1 Présentation physique du bruit en $1/f$

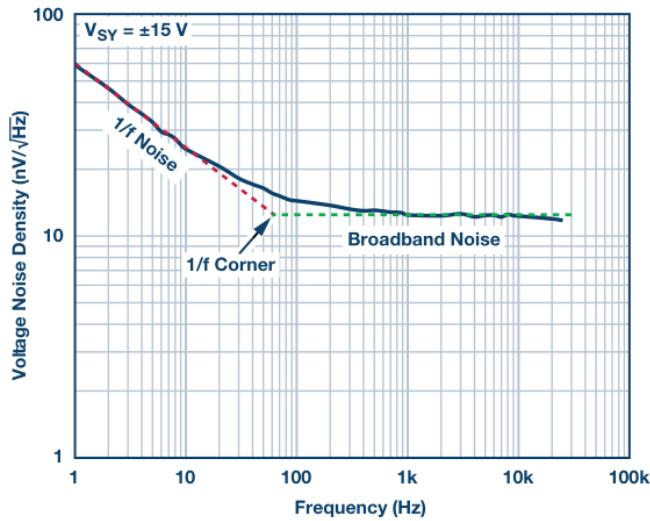


Figure 15.1: Racine carré de la densité spectrale du signal de tension.

La figure 15.1 est extraite d'un document <https://www.analog.com/en/resources/analog-digital/articles/understanding-and-eliminating-1-f-noise.html> concernant l'amplificateur opérationnel ADA4622-2. L'expérience à gauche peut être comprise dans ce cours, comme le fait de considérer la tension que l'on mettrait en entrée du composant qui expliquerait les observations avec un composant parfait. Cette tension, notée ici $\tilde{X}(t)$ est un processus aléatoire. Le graphe montre $\sqrt{S_x(f)}$ en fonction de f avec une échelle logarithmique en ordonnée et en abscisse. V_{SY} indiquée est la tension d'alimentation du composant. L'unité est en $\text{nV}/\sqrt{\text{Hz}}$, où $1\text{nV} = 10^{-9}\text{V}$. La partie gauche correspond au bruit en $1/f$, la partie droite correspond au bruit thermique. Le terme de *corner frequency* signifie ici l'intersection entre ces deux courbes qui sont des droites du fait de l'échelle log-log, et vaut ici environ 60Hz.

D'une façon plus générale, les caractéristiques du bruit en $1/f$ sont variables d'un composant à un autre : la décroissance peut être en $1/f^\alpha$ avec $\alpha \approx 1$ et la fréquence à laquelle ce bruit devient plus faible que le bruit thermique est aussi variable. Expérimentalement, mesurer le bruit à très basse fréquence amène à surveiller le composant sur une durée très longue où l'usure du composant contribue aussi à ce bruit, et il n'a pas été observé que la pente diminue lorsqu'on cherche de très basses fréquences. L'interprétation de ce bruit est d'être une aggrégation de multiples sources de bruit liées à la présence d'impureté dans le composant.

15.2 Filtrage d'un processus aléatoire

Cours signal et bruit :

Calculer les caractéristiques d'un processus aléatoire est beaucoup plus simple quand on interprète celui-ci comme étant la sortie d'un filtre dont l'entrée est un bruit blanc.

Suivant l'application, parfois on connaît une modélisation pertinente du filtre, parfois on se donne un filtre dont l'impact sur un bruit blanc produit un signal qui a les mêmes caractéristiques stochastiques que le processus aléatoire étudié.

Dans le cadre de ce cours, on a trois situations.

- L'entrée du filtre est un bruit blanc échantillonné tel que défini en sections 13.4.2 et 14.2.2.
- L'entrée du filtre est un processus aléatoire temps continu à énergie infinie.
- L'entrée du filtre est un bruit blanc avec utilisation d'un motif tel que défini en section 14.2.6.

Les notions introduites ici ne sont pas tout à fait normalisées et les équations peuvent être assez différentes dans les ouvrages de théorie du signal aléatoire.

15.3 Nouvelle définition du produit de convolution

On a défini le produit de convolution à temps continu et à temps discret. En fait dans le cadre particulier de ce cours, il est intéressant de généraliser le produit de convolution à temps discret au cas où l'un des termes est temps discret et l'autre est temps continu.

On considère ici un signal temps discret x_n et un signal temps continu $y(t)$, et on définit un signal temps continu $z(t)$

$$z(t) = x_n \stackrel{d}{*} y(t) = \sum_{k=-\infty}^{+\infty} x_k y(t - kT_e) \quad (15.1)$$

15.4 Filtre appliqué à une entrée qui est un bruit blanc échantillonné

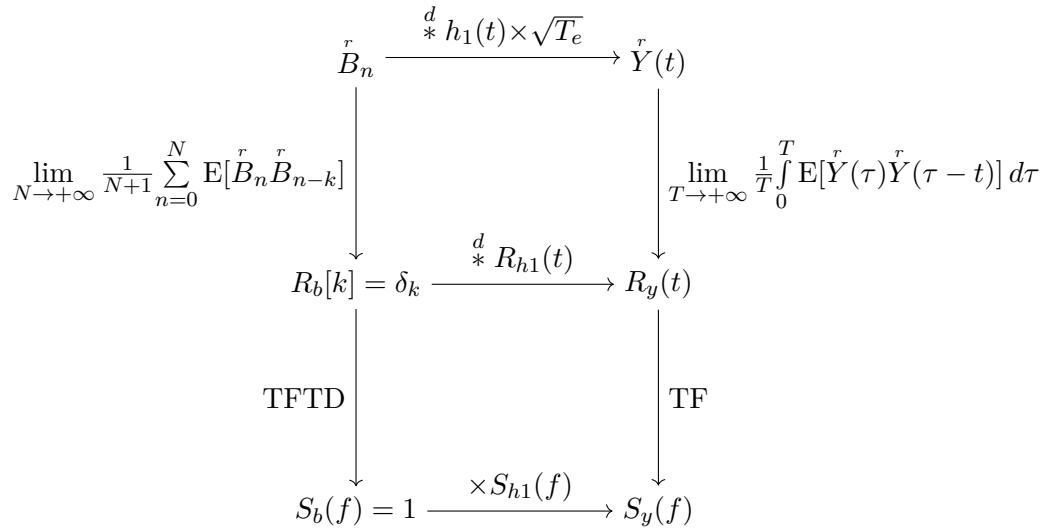


Figure 15.2: Diagramme décrivant les calculs

- $rB_n \rightarrow R_b[k]$ s'implémente avec `Rb = seb.correlation(t,B,t,B,t)/T`
- $R_B[k] \rightarrow S_b(f)$ s'implémente avec `Sb = seb.TFTD(t,Rb,f)`
- $rB_n \rightarrow S_b(f)$ est donné par

$$\lim_{N \rightarrow +\infty} \frac{1}{2N+1} E \left[\left| TFTD[rB_n] \right|^2 \right] \quad (15.2)$$

et s'implémente avec $S_b = Te*np.abs(seb.TFTD(t,B,f))**2/T$

15.5 Filtre appliqué à une entrée qui est un processus aléatoire quelconque

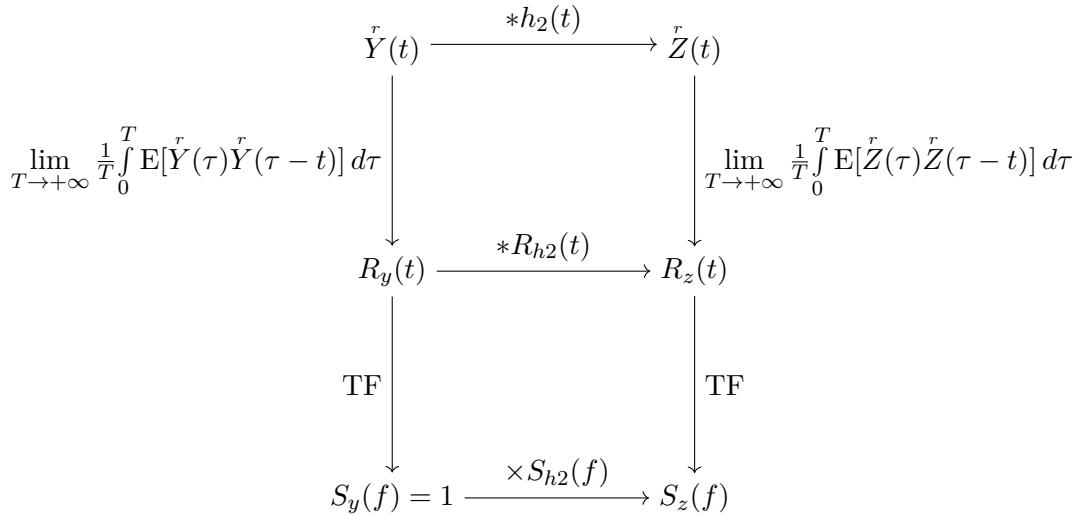


Figure 15.3: Diagramme décrivant les calculs

15.6 Simuler un bruit en $1/f$

Du point de vue expérimental, la modélisation d'un bruit en $1/f$ est en réalité un bruit en $1/f^\beta$ ce qui amène à choisir un exposant $\beta \approx 1$, fixé à 1 quand non-précisé, une fréquence minimale f_0 et une fréquence maximale f_1 . La première est généralement fixée comme l'inverse de la durée de l'expérimentation et la deuxième avec la fréquence de coupure du filtre associé à l'amplificateur utilisé pour observer ce bruit.

Pour simuler des valeurs, on utilise un filtre ici non-causal dont l'effet est de produire un bruit en $1/f$.

15.6.1 Densité spectrale

$$S_x(f) = \frac{K}{|f|^\beta} \llbracket f_0 \leq |f| \leq f_1 \rrbracket(f) \quad (15.3)$$

La constante K est fixée de telle façon que $\int_{-\infty}^{+\infty} S_x(f) df = 1$.

$$K = \frac{1}{2 \int_{f_0}^{f_1} \frac{K}{f^\beta} df} = \begin{cases} \frac{1}{2(\ln(f_1) - \ln(f_0))} & \text{si } \beta = 1 \\ \frac{1}{2 \left(\frac{f_1^{1-\beta}}{1-\beta} - \frac{f_0^{1-\beta}}{1-\beta} \right)} & \text{si } \beta \neq 1 \end{cases} \quad (15.4)$$

La puissance du processus aléatoire vaut ici 1 du fait que $\int_{-\infty}^{+\infty} S_x(f) df = 1$. Pour implémenter ces calculs d'intégrale on peut utiliser `seb.TF(f, Sx, 0)`.

Cette densité spectrale est aussi la densité spectrale du filtre $h(t)$ qui permet de générer un bruit en $1/f$ à partir d'un bruit blanc.

$$S_x(f) = S_h(f) = S_h(f)S_b(f) \quad (15.5)$$

15.6.2 Autocorrélation

Il n'y a pas de formules analytiques simples pour calculer cette autocorrélation ¹.

¹L'article "On the Autocorrelation Function of $1/f$ Noises" de Pedro Carpén et Ana V. Coronado donne une approximation analytique.

Dans ce cours on simule l'autocorrélation en se donnant une fréquence d'échantillonnage $f_e > 2f_1$, une échelle en fréquence entre $-f_e/2$ et $f_e/2$ et une échelle de temps entre $-\frac{1}{f_e}$ et $\frac{1}{f_e}$ par pas de $T_e = \frac{1}{f_e}$. Il est important que l'échelle de temps couvre aussi la partie négative de l'axe des temps parce que une autocorrélation est un signal pair.

```
f = seb.linspace(-fe/2,fe/2,400)
tr = seb.arange(-t[-1],t[-1]+Te,Te)
```

En l'occurrence $t[-1]$ signifie la dernière composante du vecteur t . Le $+Te$ est juste là pour obtenir un vecteur symétrique par rapport à $t = 0$.

Et on calcule la transformée de Fourier inverse avec **seb.TFI**.

```
Rx = seb.TFI(f,Sx,tr)
```

Cette autocorrélation est aussi l'autocorrélation de la réponse impulsionnelle du filtre $h(t)$ et elle peut aussi être déduite à partir de l'autocorrélation du bruit blanc.

$$R_x(t) = R_h(t) = R_b[n] * R_h(t) \quad (15.6)$$

La puissance est donnée par $R_x(0) = R_h(0)$

15.6.3 Trajectoires

La simulation d'une trajectoire se fait en simulant un bruit blanc échantillonné et en lui appliquant le filtre $h(t)$ dont on connaît $S_h(f)$ et $R_h(t)$. Pour obtenir une approximation de $h(t)$ non-causale, on utilise

$$h(t) = \text{TF}^{-1}[S_h(f)](f) \quad (15.7)$$

Le processus aléatoire est alors donné par

$$\overset{r}{X}(t) = \overset{r}{B}_n * h(t) \times \sqrt{T_e} = \sum_{n=-\infty}^{+\infty} \overset{r}{B}_n h(t - nT_e) \sqrt{T_e} \quad (15.8)$$

Pour faire correctement la simulation il faudrait rajouter une nouvelle échelle de temps pour décrire comment le processus aléatoire évolue pendant la durée qui s'écoule entre deux pas de temps successifs, donc une échelle de temps avec une résolution plus fine que T_e .

À défaut de faire cette simulation plus précise, on peut utiliser

```
t = seb.linspace(0,T,1000)
import numpy.random
B = nr.normal(0,1,len(t))
h = np.real(seb.TFI(f,Sh(f),tr))
X = seb.convolution(t,B,tr,h,t)/np.sqrt(Te)
```

La dernière ligne contient une division et non une multiplication par `np.sqrt(Te)`, parce que le programme **seb.convolution** normalement fait pour approximation la convolution entre signaux temps continus contient une multiplication par `Te`.

On peut retrouver les caractéristiques attendues du bruit.

- La puissance est donnée par

$$P_x = \lim_{T \rightarrow +\infty} \frac{1}{T} \int_0^T \mathbb{E}[\overset{r}{X}(t)^2] dt \quad (15.9)$$

qui s'implémente avec `np.real(seb.TF(t,X**2,0))` et en fait en faisant une moyenne sur plusieurs réalisations de $\overset{r}{X}(t)$.

- L'autocorrélation s'obtient approximativement avec

$$R_x(\tau) = \lim_{T \rightarrow +\infty} \frac{1}{T} \int_0^T \mathbb{E}[\overset{r}{X}(t)\overset{r}{X}(t - \tau)] dt \quad (15.10)$$

qui s'implémente avec `seb.correlation(t,X,t,X,tr)/T` et en fait en faisant une moyenne sur plusieurs réalisations de $\overset{r}{X}(t)$.

- La densité spectrale s'obtient approximativement avec

$$S_x(f) = \lim_{T \rightarrow +\infty} \frac{1}{T} E \left[\left| \text{TF}[X^r(t)](f) \right|^2 \right] \quad (15.11)$$

avec $\hat{X}(t)$ défini sur $[0, T]$. L'implémentation se fait avec `(np.abs(seb.TF(t,X,f)))**2/T` et en fait en faisant une moyenne sur plusieurs réalisations de $\hat{X}(t)$.

Remplacer les deux fréquences f_0 et f_1 par des filtres

À la place d'utiliser deux fréquences, on peut considérer un filtre passe-bande obtenu en associant un filtre passe-bas en série avec un filtre passe-haut. Le filtre passe-bas correspond au fait de retirer la moyenne du signal calculée sur la durée de l'expérience. Le filtre passe-haut est lié au filtre de l'amplificateur mais pour simplifier on peut considérer qu'il s'agit d'un passe-haut d'ordre 1 déterminé par une certaine fréquence de coupure.

Chapter 16

Densité de probabilité et application au bruit de grenaille

16.1 Présentation physique du bruit de grenaille

16.2 Filtre appliqué à une entrée qui est un bruit blanc avec utilisation d'un motif

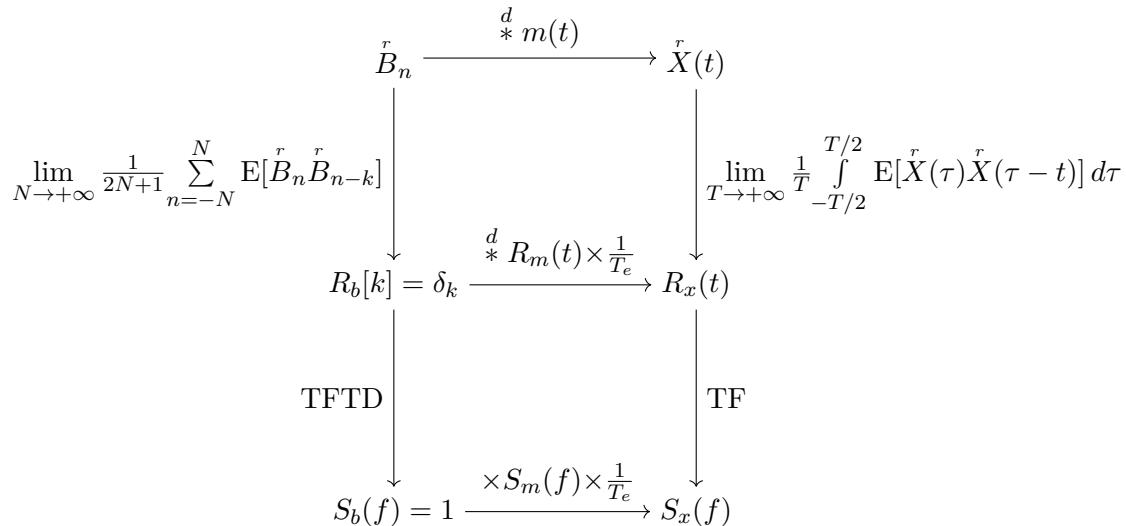


Figure 16.1: Diagramme décrivant les calculs

16.2.1 Bruit blanc avec motifs

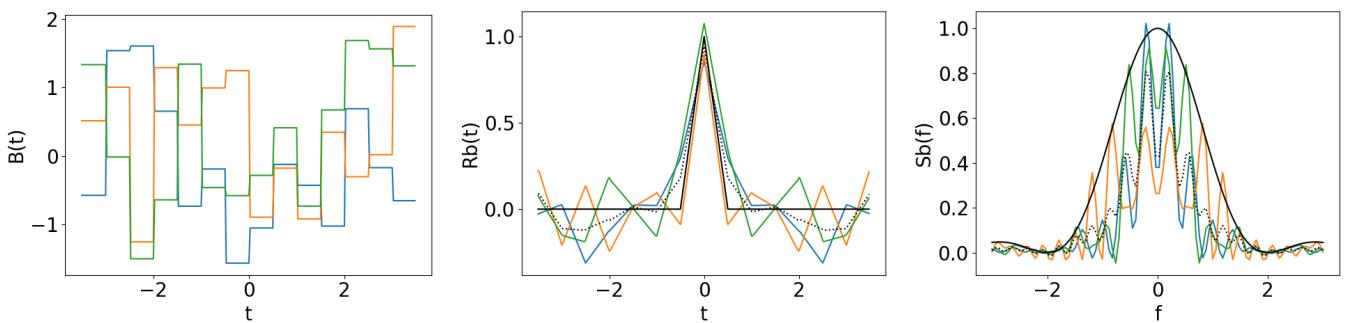


Figure 16.2: Bruit blanc avec motifs : réalisation de trois trajectoires, trois autocorrélations, trois densités spectrales.

La figure 16.2 montre un bruit blanc échantillonné. L'implémentation est dans A.22.

Simulation d'une trajectoire

J'illustre la façon de procéder avec la gauche de la figure 16.2.

Cette fois-ci, on a besoin de deux fréquences d'échantillonnages et deux échelles de temps notées $f_{e,t}$ et $f_{e,rv}$. La première série décrit la dépendance vis-à-vis du bruit blanc. La deuxième décrit le motif point par point.

```
T    = 7
fe   = 2
Te   = 1/fe
t    = seb.arange(-T/2,T/2,1/fe) #echelle de temps pour l'aleatoire
fev = 20*fe
tv   = seb.arange(-T/2,T/2,1/fev)#echelle de temps pour la visualisation
```

On utilise la première pour générer les coefficients du bruit blanc.

```
b1 = nr.normal(0,1,len(t))
```

Je considère ici un motif $m(t) = \llbracket -\frac{T_e}{2} \leq t < \frac{T_e}{2} \rrbracket(t)$ et j'utilise une petite astuce pour implémenter consistant à faire correspondre chaque indice de $bv1$ avec un indice de $b1$.

```
ind = (np.floor(tv/Te-np.min(tv)/Te)).astype(int)
bv1,bv2,bv3 = b1[ind], b2[ind], b3[ind]
```

C'est ainsi que je réalise la partie gauche de la figure 16.2.

Simulation d'une autocorrélation

La simulation de l'autocorrélation est assez similaire au cas du bruit blanc échantillonné, cette fois j'utilise l'échelle tv pour avoir plus de précision sur l'autocorrélation.

```
Rb1 = seb.correlation(tv,bv1,tv,bv1,tv)/T
```

Comme précédemment il faut diviser par la durée du signal ici T . C'est ainsi que j'obtiens les courbes colorées d'autocorrélation au milieu de la figure 16.2.

D'un point de vue théorique, je peux calculer $R_m(t)$ sachant que $m(t) = \llbracket -\frac{T_e}{2} \leq t < \frac{T_e}{2} \rrbracket(t) = \Pi\left(\frac{t}{T_e}\right)$. Parce que $m(t)$ est pair,

$$R_m(t) = \Pi\left(\frac{t}{T_e}\right) * \Pi\left(-\frac{t}{T_e}\right) = \Pi\left(\frac{t}{T_e}\right) * \Pi\left(\frac{t}{T_e}\right) = \text{TF}^{-1} \left[\left(T_e \frac{\sin(\pi f T_e)}{\pi f T_e} \right)^2 \right] (t) = T_e \mathbb{T}\left(\frac{t}{T_e}\right) \quad (16.1)$$

Pour faire ce calcul, j'ai en fait utilisé le fait que je sais déjà que $\Pi(t) * \Pi(t) = \mathbb{T}(t)$ qui fait l'objet d'un exercice. La courbe en noir en trait plein est justement $\mathbb{T}\left(\frac{t}{T_e}\right)$. La courbe en pointillé est la moyenne des trois autocorrélations trouvées, elle s'approche un peu de cette courbe théorique.

Simulation d'une densité spectrale

On crée une échelle de fréquence pour la visualisation

```
f    = np.linspace(-3,3,10**2)
```

Et avec cette échelle en fréquence, on calcule la densité spectrale au moyen de la transformée de Fourier (ici ce n'est plus la TFTD, c'est bien la transformée de Fourier en utilisant l'échelle de temps tv plus précise).

```
Sb1 = np.real(seb.TF(tv,Rb1,f))
```

J'ai considéré la partie réelle, parce que je sais que la transformée de Fourier doit être réelle.

C'est ainsi que j'obtiens la partie droite de la figure 16.2. La courbe en noire est la transformée de Fourier de l'autocorrélation théorique et ici

$$Sm(f) = \left(T_e \frac{\sin(\pi f T_e)}{\pi f T_e} \right)^2 \quad (16.2)$$

16.3 Cas où l'entrée est un processus aléatoire gaussien

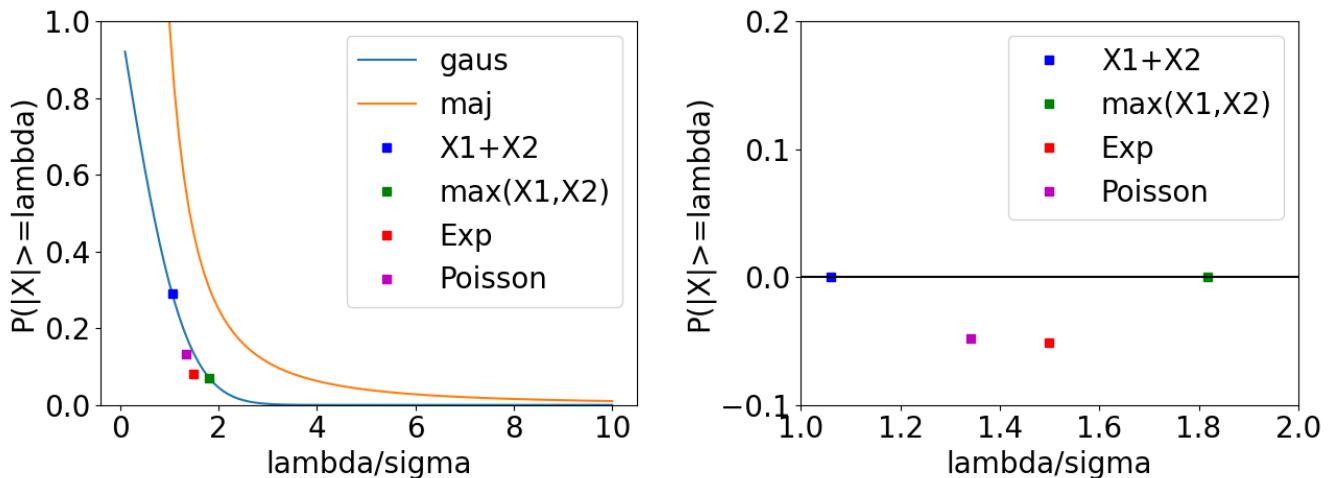


Figure 16.3: Probabilité d'un écart à la moyenne en fonction de l'écart lui-même. À gauche : cas gaussien et cas général avec des exemples pour différentes distributions de probabilité. À droite : différence entre ces probabilités et celles pour le cas gaussien.

Cours signal et bruit :

Lorsque l'entrée est un processus aléatoire gaussien alors la sortie du filtre est aussi un processus aléatoire gaussien. D'une façon générale une combinaison linéaire de variable aléatoire gaussienne est gaussienne, mais une transformation non-linéaire n'est pas gaussienne.

16.3.1 Écart à la moyenne pour une densité de probabilité gaussienne

En termes probabilité :

Une variable aléatoire gaussienne vérifie cette inégalité.

$$P(|X - E[X]| \geq \lambda) = 1 - \text{erf}\left(\frac{\lambda}{\sqrt{2}\sqrt{\text{Var}[X]}}\right) \quad (16.3)$$

Cette équation est représentée par la courbe basse de la gauche de la figure 16.3. Cette courbe basse est la ligne noire horizontale sur la droite de la figure 16.3.

Cette égalité se démontre à partir de

$$P\left(\overset{r'}{X} \leq \lambda\right) = 0.5 + 0.5\text{erf}\left(\frac{\lambda}{\sqrt{2}}\right)$$

$\overset{r'}{X}$ est une variable aléatoire gaussienne centrée et normalisée : $E[\overset{r'}{X}] = 0$ et $\text{Var}[\overset{r'}{X}] = 1$.

En effet

D'un point de vue mathématique :

La fonction erf est appelée la fonction erreur. Elle est définie par

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-\tau^2} d\tau \quad (16.4)$$

Python :

Il n'est pas du tout utile de connaître la définition mathématique de erf, elle est implémentée dans de nombreux logiciels. Sur Python, j'ai inclus un lien vers cette fonction avec `scipy.erf` qui transforme une valeur ou un vecteur de valeur en une valeur ou un vecteur de valeurs.

On cherche ici à vérifier l'équation (16.3). On choisit aléatoirement une moyenne μ , un écart-type σ , un nombre de points N et une valeur positive λ

```
import numpy.random as nr
mu,sigma,N = nr.uniform(0,2), nr.uniform(0,2), 10**7
Lambda = nr.uniform(mu+sigma)
```

On génère aléatoirement un processus aléatoire blanc gaussien sur ces N termes. Et on compte le nombre de fois où l'évènement $|X - E[X]| \geq \lambda$ est effectivement vérifié. La proportion de fois où ceci est vérifié est une approximation de la probabilité de $P(|X - E[X]| \geq \lambda)$

```
x      = nr.normal(mu,sigma,N)
p_app = np.sum(np.abs(x-mu)>=Lambda)/N
p_th  = 1-seb.erf(Lambda/sigma/np.sqrt(2))
print(f"err = {np.abs(p_app-p_th):.3g}")
```

16.4 Cas où l'entrée est un processus aléatoire non-gaussien

Cours signal et bruit :

Lorsque l'entrée n'est pas un processus aléatoire gaussien alors la sortie du filtre n'est pas un processus aléatoire gaussien. Parfois la sortie est correctement approximée par un processus aléatoire gaussien.

16.4.1 Écart à la moyenne pour une densité de probabilité ayant un variance finie

$$P(|X - E[X]| \geq \lambda) \leq \frac{\text{Var}(X)}{\lambda^2} \quad (16.5)$$

D'un point de vue mathématique :

Cette inégalité porte le nom de Bienaymé-Tchebycheff, elle fait partie de ce qu'on appelle plus généralement les inégalités de concentration. Le terme de droite de l'équation est une majoration de la probabilité, ce n'est donc pas une probabilité et encore moins une densité de probabilité.

Appendix A

Code pour simuler les différentes figures

A.1 Code commun

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('c:/a/simu/seb/prg')
import os
os.chdir('c:/a/simu/seb/')
import seb
plt,np=seb.debut()
```

A.1.1 Spécificités de Python

Types

Dans le cadre de ces TP, on utilise essentiellement deux structures de données, les scalaires, les et les vecteurs et matrices. L'instruction `type` permet de tester si une variable est de ce type.

- `int` illustré par `a=2` ou `numpy.int64` illustré par `a=np.zeros(2,dtype=int)[0]`
- `float` illustré par `a=2.0` ou `numpy.float64` illustré par `a=np.zeros(2)[0]`
- `complex` illustré par `a=1j` ou `numpy.complex128` illustré par `a=np.zeros(2,dtype=complex)[0]`
- `str` illustré par `a='e'` ou `a="abc"` ou `a="""bcd"""` ou `a='13'`
- `bool` illustré par `a=True`
- `ndarray` illustré par
 - `a=np.array([1.2])` est un vecteur à une composante
 - `a=np.array([1.2,1.4])` est un vecteur colonne à deux composantes pouvant être utilisé comme vecteur ligne à deux composantes.
 - `a=np.array([[1.2],[1.4]])` est une matrice à une ligne et deux colonnes ne pouvant pas être utilisé comme un vecteur ligne.
 - `a=np.array([[1.2,0],[0,1.4]])` est une matrice diagonale.

Il y a deux types que l'on est amené à utiliser de fait, mais sans que les données que l'on utilise en fasse partie.

- `tuple` illustré par `a=(1,)` ou `a=(1,2)` ou `a=1,2`
- `list` illustré par `a=[1,]` ou `a=(1,2)` ou `a=1,2` que l'on n'utilisera pas parce que par exemple

```
a=[1,2]
b=a+[1]
b
```

```
[1,2,1]
```

Conversions

Certaines conversions sont implicites.

- Les types `int` ou `float` sont convertis en `float` ou en `complex`. Ainsi

```
2/3 == 2/3.0 == 2.0/3.0
```

- Les types `np.array` ne sont pas convertis en scalaires lorsqu'ils n'ont qu'une seule composante. C'est l'inverse ce sont les scalaires qui sont convertis en `np.array` lorsqu'ils sont en relation avec un type `np.array`. Ainsi les instructions suivantes sont vraies

```
1 == np.array([1])
(1 + np.array([0]))[0]==1
```

- Le type `bool` est converti en `int`, `float` ou `complex` quand il est en scalaire, **mais** pas quand il est regroupé avec un `bool`. Ainsi les instructions suivantes sont vraies.

```
True+True == 2
np.array((True))+np.array((True))+0==1
(np.array((True))+0)+(np.array((True))+0)==2
```

Pour faire des conversions explicites, on distingue les scalaires.

•

Copie par valeurs ou par références

La copie par valeurs consiste à recopier les valeurs d'une structure. La copie par référence consiste à donner à une structure de données un nouveau nom, les données ne sont pas dupliquées et leur modification affecte les données référencées par les deux noms. Voici deux exemples de copies, à gauche par références et à droite par valeurs.

```
a=[1,2] ou a=np.array([1,2])
b=a
b[1]=3
a=[1,3]

a=1
b=a
b=b+1
a
1
```

`a=[1,2] ou a=np.array([1,2])`
`b=a.copy() ou b=a[:]`
`b[1]=3`
`a`
`[1,2]`

Fonctions

L'écriture d'une fonction se fait au moyen de `def`. Par exemple la fonction suivante calcule la somme de deux nombres.

```
def somme(a,b):
    return a+b
```

Il convient quand on écrit une fonction de mettre deux espaces (ou 4 espaces) au début de chaque ligne qui suit la ligne contenant `def`. Il faut aussi s'assurer quand on fait un copier coller d'une fonction de la faire suivre par une ligne sans aucun caractère ni espaces.

Cette fonction a un comportement particulier qui fait qu'on peut y référencer des variables que l'on n'a pas initialisé dans la fonction. Lors de l'exécution de cette fonction, l'interpréteur va chercher si cette variable existe et utiliser sa valeur. C'est ainsi que l'on peut comprendre les instructions suivantes.

```
def x():
    return a

a=3
x()

3

a=4
x()

4
```

On peut utiliser le système de valeurs par défaut et aussi changer l'ordre des valeurs transmises lorsqu'on rappelle les noms des variables.

```
def ex1(a,b=4):
    return a+2*b

ex1(2,3)=8 ex1(2)=10 ex1(1,b=5)=11 ex1(b=5,a=2)=12
```

En poussant à l'extrême, on peut même fabriquer une fonction très curieuse qui modifie une valeur non-indiquée dans l'appel à la fonction.

```
def modifie():
    z = y
    z[0] = z[0]+1

y=[2] ou y=np.array([2])
modifie()
y

3 ou array([3])
```

A.2 Codes utilisant les fonctions de bases

Simulation de la figure 2.6

Table A.1

```
t=np.linspace(-3.1,3.1,10**2);
x=seb.fonction_T(t)
fig, ax = plt.subplots()
ax.plot(t,x)
ax.set_xlabel('t')
ax.set_ylabel('triangle(t)')
plt.tight_layout()
fig.savefig('./figures/fig_C2_5_2a.png');
fig.show()
fig, ax = plt.subplots()
y=seb.fonction_T(t/2)
ax.plot(t,y,label=r'triangle(t/2)')
```

```

z=seb.fonction_T(2*t)
ax.plot(t,z,label=r'triangle(2t)')
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C2_5_2b.png')
fig.show()

```

Table A.2: Simulation des figures 4.1.

```

plt,np=seb.debut()
fig, ax = plt.subplots()
t=np.linspace(-5,5,10**3)
x1=seb.fonction_P(t)
ax.plot(t,x1,'b-',label='x1(t)=P(t)')
x2=seb.fonction_P(t/2)
ax.plot(t,x2,'r-',label='x2(t)=P(t/2)')
x3=seb.fonction_P(t/3)
ax.plot(t,x3,'g-',label='x3(t)=P(t/3)')
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C2_2_fig2a.png');
fig.show()

f=np.linspace(-3,3,10**3)
f=np.delete(f,np.where(abs(f)<10**(-10)))
X1=np.sinc(f); X2=2*np.sinc(2*f); X3=3*np.sinc(3*f);
fig, ax = plt.subplots()
ax.plot(f,np.abs(X1),'b-',label='|X1(f)|')
ax.plot(f,np.abs(X2),'r-',label='|X2(f)|')
ax.plot(f,np.abs(X3),'g-',label='|X3(f)|')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/C2_2_fig2b.png')
fig.show()

fig, ax = plt.subplots()
x4=seb.fonction_P(t-0.5)
ax.plot(t,x1,'b-',label='x1(t)')
ax.plot(t,x4,'r-',label='x4(t)=x1(t-0.5)')
ax.set_xlabel('t')
ax.legend(loc='center right')
plt.tight_layout()
fig.savefig('./figures/C2_2_fig2c.png')
fig.show()

fig, ax = plt.subplots()
X4=np.sinc(f)*np.exp(-1j*2*np.pi*f*0.5)
ax.set_xlabel('f')
ax.plot(f,abs(X1),'b-',label='|X1(f)|')
ax.plot(f,abs(X4),'r-',label='|X4(f)|')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/C2_2_fig2d.png')

```

```

fig.show()

fig, ax = plt.subplots()
ax.set_xlabel('f')
ax.plot(f,np.angle(X1),'b-',label='arg(X1(f))')
ax.plot(f,np.angle(X4),'r-',label='arg(X4(f))')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/C2_2_fig2e.png')
fig.show()

```

A.3 Codes utilisant la transformée de Fourier et la transformée de Fourier inverse

Simulation de la figure 3.2

Table A.3

```

t=np.linspace(-2,4,10**3)
x_ex=np.exp(-t)*(t>=0)
f1=np.linspace(-2,2,10**3)
X1=1/(1+1j*2*np.pi*f1)
f2=np.linspace(-200,200,10**3)
X2=1/(1+1j*2*np.pi*f2)
x1=np.real(seb.TFI(f1,X1,t))
x2=np.real(seb.TFI(f2,X2,t))
fig,ax = plt.subplots()
ax.plot(t,x_ex,label='theorique')
ax.plot(t,x1,label='avec X(f)[|f|<=2](f)')
ax.plot(t,x2,label='avec X(f)[|f|<=200](f)')
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C2_2_4a.png')
fig.show()

X = lambda f : 1/(1+1j*2*np.pi*f)
dX = lambda f,t,x : X(f)-seb.TF(t,x,f)
ech1_f = lambda fa,fb : np.linspace(max(0,fa-fb),fa+fb,10**2)
ech_f = lambda fa,fb : np.concatenate((np.flip(-ech1_f(fa,fb)),ech1_f(fa,fb)))
t=np.linspace(-2,4,10**3)
x_ex=np.exp(-t)*(t>=0)
f0 = np.linspace(-2*10**3,2*10**3,5000)
f0_a = 30; f0_b=30
f1 = ech_f(f0_a,f0_b)
x1 = np.real(seb.TFI(f1,X(f1),t))
f1_a = 6; f1_b=6
f2 = ech_f(f1_a,f1_b)
x2=np.real(seb.TFI(f2,dX(f2,t,x1),t))+x1
f2_a = 0.5; f2_b=0.5
f3 = ech_f(f2_a,f2_b)
x3=np.real(seb.TFI(f3,dX(f3,t,x2),t))+x2

```

```

fig,ax = plt.subplots()
ax.plot(t,x_ex,label='theorique')
str1=f"avec X(f)[||f|-{round(f0_a,1)}|<={round(f0_b,1)}](f)"
ax.plot(t,x1,'r-',label=str1)
str2=f"avec X(f)[||f|-{round(f1_a,1)}|<={round(f1_b,1)}](f)"
ax.plot(t,x2,'g-',label=str2)
str3=f"avec X(f)[||f|-{round(f2_a,1)}|<={round(f2_b,1)}](f)"
ax.plot(t,x3,'m-',label=str3)
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C2_2_4b.png')
fig.show()

```

```

fig,ax = plt.subplots()
ax.plot(f0,abs(X(f0)),'b-',label='|X(f)|')
ax.plot(f0,abs(dX(f0,t,x1)),'r-',label='|X(f)-TF(t,x1)|')
ax.plot(f0,abs(dX(f0,t,x2)),'g-',label='|X(f)-TF(t,x2)|')
ax.plot(f0,abs(dX(f0,t,x3)),'m-',label='|X(f)-TF(t,x3)|')
plt.xlim(-25,25)
fig.legend()
ax.set_xlabel('f')
plt.tight_layout()
fig.savefig('./figures/fig_C2_2_4c.png')
fig.show()

```

A.4 Codes simulant la dérivée

Python :

Table A.4: Simulation de la figure 5.2

```

plt.close('all')
t=np.linspace(-2,2,10**3)
s=(t>=-1)*(t<=1)*(-2*t)
fig,ax = plt.subplots()
ax.plot(t,s)
ax.set_xlabel('t')
ax.set_ylabel('s(t)')
plt.tight_layout()
fig.savefig('./figures/C3_1_fig2a.png')
fig.show()

t=np.linspace(-4,4,10**3)
dsdt=(t>=-1)*(t<=1)*(-2);
t_,dsdt_=np.array([-1,1]),np.array([2,2])
fig,ax = plt.subplots()
ax.plot(t,dsdt,'r-')
ax.stem(t_,dsdt_,'b-',basefmt=" ")
ax.set_xlabel('t')
ax.set_xlim(-2,2)

```

```

ax.set_ylabel('ds/dt')
plt.tight_layout()
fig.savefig('./figures/C3_1_fig2b.png')
fig.show()

t=np.linspace(-4,4,10**3)
s=(t>=-1)*(t<=1)*(-2*t)
ind = np.argwhere(abs(np.diff(s))>0.1)
t_app_=(t[ind]+t[ind+1])/2
dsdt_app_=np.diff(s)[ind]
s_ech1 = dsdt_app_[0][0]*seb.retarder(t,seb.fonction_H(t),t_app_[0][0])
s_ech2 = dsdt_app_[1][0]*seb.retarder(t,seb.fonction_H(t),t_app_[1][0])
s1 = s-s_ech1-s_ech2
fig,ax = plt.subplots()
ax.plot(t,s,'b-',label='s(t)')
ax.plot(t,s1,'r-',label='s1(t)')
ax.set_xlim(-2,2)
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/C3_1_fig2c.png')
fig.show()

```

A.5 Codes pour simuler une réponse impulsionale

Table A.5: Simulation de la figure ??

```

t=np.linspace(-3,3,10**3)
f1=np.linspace(-300,300,10**2)+10**(-3)
f2=np.linspace(-30,30,10**2)+10**(-4)
f3=np.linspace(-3,3,10**2)+10**(-5)
h_th=(t>=0)
H1=1/(1j*2*np.pi*f1)
H2=1/(1j*2*np.pi*f2)
H3=1/(1j*2*np.pi*f3)
h1=np.real(seb.TFI(f1,H1,t)); h1=h1-h1[0]
h2=np.real(seb.TFI(f2,H2-seb.TF(t,h1,f2),t))+h1; h2=h2-h2[0]
h3=np.real(seb.TFI(f3,H3-seb.TF(t,h2,f3),t))+h2; h3=h3-h3[0]
fig,ax = plt.subplots()
ax.plot(t,h_th,label='theorique')
ax.plot(t,h1,label='h1')
ax.plot(t,h2,label='h2')
ax.plot(t,h3,label='h3')
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C5_2_fig1a.png')
fig.show()

```

A.6 Simulation d'un filtrage

Table A.6: Simulation de la figure 8.2

```

tx=np.linspace(-5,5,10**4)
Te=tx[1]-tx[0]
x=(tx>=-1)&(tx<=1)
th=np.arange(-5, 5, Te)
h=-1*(th>=0)
ty=np.arange(-3, 3, Te)
y=seb.convolution(tx,x,th,h,ty)

fig,ax = plt.subplots()
ax.plot(tx,x)
ax.set_xlabel('t')
ax.set_ylabel('x(t)')
plt.tight_layout()
fig.savefig('./figures/C5_3_fig1a.png')
fig.show()

fig,ax = plt.subplots()
ax.plot(th,h)
ax.set_xlabel('t')
ax.set_ylabel('h(t)')
plt.tight_layout()
fig.savefig('./figures/C5_3_fig1b.png')
fig.show()

fig,ax = plt.subplots()
ax.plot(ty,y)
ax.set_xlabel('t')
ax.set_ylabel('y(t)')
plt.tight_layout()
fig.savefig('./figures/C5_3_fig1c.png')
fig.show()

```

Table A.7: Simulation de la figure 11.1

```

t=np.linspace(-3,3,10**3)
x=np.cos(np.pi*t)
y=2/np.pi*np.sin(np.pi*t)
fig,ax = plt.subplots()
ax.plot(t,x,label='x_1(t)')
ax.plot(t,y,label='y_1(t)')
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C11_1_1a.png')
fig.show()

t=np.linspace(-3,3,10**3)
x2=np.cos(np.pi*t)*(t>=0)
y2=2/np.pi*np.sin(np.pi*t)*(t>=0)
y3=2/np.pi*np.sin(np.pi*t)*(t>=0)-1/np.pi*np.sin(np.pi*t)*(t>=0)*(t<1)
fig,ax = plt.subplots()
ax.plot(t,x2,label='x_2(t)')
ax.plot(t,y2,label='y_1(t)*(t>=0)')
ax.plot(t,y3+0.02,label='y_1(t)*(t>=0)+y_3(t)')
ax.set_xlabel('t')

```

```

ax.legend(loc='lower left')
plt.tight_layout()
fig.savefig('./figures/fig_C11_1_1b.png')
fig.show()

```

A.7 Codes pour simuler la périodisation et la série de Fourier

Table A.8: Simulation de la figure 10.4

```

t=np.linspace(-2,2,10**3)
t2=seb.periodiser_ech_t(t,1)
x=seb.fonction_P(2*t2-0.5)
fig,ax = plt.subplots()
ax.plot(t,x,label='signal carré')
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C9_2_2a.png')
fig.show()

x1=0.5*np.ones_like(t)
x2=2/np.pi*np.cos(2*np.pi*t-np.pi/2)
x3=2/np.pi/3*np.cos(6*np.pi*t-np.pi/2)
x4=2/np.pi/5*np.cos(10*np.pi*t-np.pi/2)
fig,ax = plt.subplots()
ax.plot(t,x1,label='x1')
ax.plot(t,x1+x2,label='x2')
ax.plot(t,x1+x2+x3,label='x3')
ax.plot(t,x1+x2+x3+x4,label='x4')
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C9_2_2b.png')
fig.show()

```

Table A.9: Simulation de la figure 13.3

```

t=np.linspace(-50,50,100)
x=np.random.normal(0,1,len(t))
t1=t[::8]; x1=x[::8]
fig,ax = plt.subplots()
ax.plot(t1,x1,'r+')
ax.set_xlabel('t')
ax.set_ylabel('Xt')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C8_4_1a.png')
fig.show()

deb2=round(len(t)/4); fin2=round(len(t)*3/4)
t1=t[deb2:fin2:8]; x1=x[deb2:fin2:8]
t2=t[deb2:fin2:4]; x2=x[deb2:fin2:4]
fig,ax = plt.subplots()
ax.plot(t1,x1,'b-')
ax.plot(t2,x2,'r.')

```

```

ax.set_xlabel('t')
ax.set_ylabel('Xt')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C8_4_1b.png')
fig.show()

deb3=round(3*len(t)/8); fin3=round(len(t)*5/8)
t2=t[deb3:fin3:4]; x2=x[deb3:fin3:4]
t3=t[deb3:fin3:2]; x3=x[deb3:fin3:2]
fig,ax = plt.subplots()
ax.plot(t2,x2,'b-')
ax.plot(t3,x3,'r.')
ax.set_xlabel('t')
ax.set_ylabel('Xt')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C8_4_1c.png')
fig.show()

deb4=round(7*len(t)/16); fin4=round(len(t)*9/16)
t3=t[deb4:fin4:2]; x3=x[deb4:fin4:2]
t4=t[deb4:fin4]; x4=x[deb4:fin4]
fig,ax = plt.subplots()
ax.plot(t3,x3,'b-')
ax.plot(t4,x4,'r.')
ax.set_xlabel('t')
ax.set_ylabel('Xt')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C8_4_1d.png')
fig.show()

```

Table A.10: Simulation de la figure ??

```

fe=10
tx=np.arange(-2,2,1/fe)
x1=np.random.normal(0,1,len(tx))
x2=np.random.normal(0,1,len(tx))
th=np.arange(0,1,1/fe)
h=np.ones(len(th))/fe
y1=seb.convolution(tx,x1,th,h,tx)*fe
y2=seb.convolution(tx,x2,th,h,tx)*fe
fig,ax=plt.subplots()
ax.plot(tx,y1)
ax.plot(tx,y2)
ax.set_xlabel('t')
plt.tight_layout()
fig.savefig('./figures/fig_C8_4_3a.png')
fig.show()
print(np.sum(y1**2)/len(y1),np.sum(y2**2)/len(y2))

fe=100
tx=np.arange(-2,2,1/fe)
x1=np.random.normal(0,1,len(tx))

```

```

x2=np.random.normal(0,1,len(tx))
th=np.arange(0,1,1/fe)
h=np.ones(len(th))/fe
y1=seb.convolution(tx,x1,th,h,tx)*fe
y2=seb.convolution(tx,x2,th,h,tx)*fe
fig,ax=plt.subplots()
ax.plot(tx,y1)
ax.plot(tx,y2)
ax.set_xlabel('t')
plt.tight_layout()
fig.savefig('./figures/fig_C8_4_3b.png')
fig.show()
print(np.sum(y1**2)/len(y1),np.sum(y2**2)/len(y2))

```

Table A.11: Simulation de la figure ??

```

import seb; plt,np=seb.debut();
plt.close('all')
fe=10
tx=np.arange(-2,2,1/fe)
x1=np.random.normal(0,1,len(tx))*np.sqrt(fe)
x2=np.random.normal(0,1,len(tx))*np.sqrt(fe)
th=np.arange(0,1,1/fe)
h=np.ones(len(th))/fe
y1=seb.convolution(tx,x1,th,h,tx)*fe
y2=seb.convolution(tx,x2,th,h,tx)*fe
fig,ax=plt.subplots()
ax.plot(tx,y1)
ax.plot(tx,y2)
ax.set_xlabel('t')
plt.tight_layout()
fig.savefig('./figures/fig_C8_4_3a.png')
fig.show()
print(np.sum(y1**2)/len(y1),np.sum(y2**2)/len(y2))

fe=100
tx=np.arange(-2,2,1/fe)
x1=np.random.normal(0,1,len(tx))*np.sqrt(fe)
x2=np.random.normal(0,1,len(tx))*np.sqrt(fe)
th=np.arange(0,1,1/fe)
h=np.ones(len(th))/fe
y1=seb.convolution(tx,x1,th,h,tx)*fe
y2=seb.convolution(tx,x2,th,h,tx)*fe
fig,ax=plt.subplots()
ax.plot(tx,y1)
ax.plot(tx,y2)
ax.set_xlabel('t')
plt.tight_layout()
fig.savefig('./figures/fig_C8_4_3b.png')
fig.show()
print(np.sum(y1**2)/len(y1),np.sum(y2**2)/len(y2))

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))
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
K=10**2
fe_l=np.linspace(5,300,K)
std_l=np.zeros(K)
for k in range(K):
    t,s=s_traj(fe_l[k])
    std_l[k]=np.std(s)
fig,ax = plt.subplots()
ax.plot(fe_l,std_l**2)
ax.plot(fe_l,1/fe_l)
ax.set_xlabel('fe')
plt.tight_layout()
fig.savefig('./figures/fig_C8_4_3c.png')
fig.show()

```

A.8 Code pour simuler la solution d'une équation différentielle

Table A.12: Simulation de la figure 7.1

```

t=np.linspace(-2,10,10**3)
y_th=(t>=0)*np.sin(t)
y_num=seb.sol_eq_diff((1,0,1),t)
fig,ax = plt.subplots()
ax.plot(t,y_th,label='theorique')
ax.plot(t,y_num,label='approchée')
ax.set_xlabel('t')
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C13_3_fig1a.png')
fig.show()
fig,ax = plt.subplots()
ax.plot(t,y_th-y_num)
ax.set_xlabel('t')
ax.set_ylabel('difference')
plt.tight_layout()
fig.savefig('./figures/fig_C13_3_fig1b.png')
fig.show()

```

Python :

Table A.13: Implémentation de la figure 10.5.

```

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
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()

```

Python :

Table A.14: Implémentation de la figure 10.7.

```

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 premiere valeur',max(t),T/2)
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(s1(t2)), ' S1P[0]=' ,
      np.sum(s1(seb.periodiser_ech_t(t1,(-T/2,T/2))))/len(t1), ' S1DP[0]=' ,np.sum(s1(t3))/len(t3))
)
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()

```

Python :

Table A.15: Implémentation de la figure 10.6.

```

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()

```

Python :

Table A.16: Implémentation de la figure 12.2.

```

fe_l=np.linspace(1,10**3,10**3)
err_l=np.zeros_like(fe_l)
for k in range(len(err_l)):
    Te=1/fe_l[k]
    fun = lambda t: s1(t)-s1(Te*np.round(t/Te))
    err_l[k]=seb.erreur_quad(fun,(-2,2))

fig,ax = plt.subplots()
ax.plot(fe_l,err_l)
ax.set_xlabel('fe')
ax.set_ylabel('err')
plt.tight_layout()
fig.savefig('./figures/fig_TP2_fig3a.png')
fig.show()
fig,ax = plt.subplots()
ax.plot(fe_l,10*np.log10(err_l))
ax.set_xlabel('fe')
ax.set_ylabel('10*log10(err)')
plt.tight_layout()
fig.savefig('./figures/fig_TP2_fig3b.png')
fig.show()

```

Python :

Table A.17: Implémentation de la figure 12.3.

```

def s_reconstruit(t,s,tr):
    Te=t[1]-t[0]; fe=1/Te

```

```

f=np.linspace(-fe/2,fe/2,10**4)
S=seb.TFTD(t,s,f)/fe
sr=np.real(seb.TFI(f,S,tr))
return sr

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)
s1ar=s_reconstruit(ta,s1(ta),t)
s1br=s_reconstruit(tb,s1(tb),t)
fig,ax = plt.subplots()
ax.plot(t,s1a(t),label='s1(t)')
ax.plot(ta,s1(ta),label='fe=3')
ax.plot(t,s1ar,label='r fe=3')
ax.plot(tb,s1(tb),label='fe=7')
ax.plot(t,s1br,label='r fe=7')
ax.set_xlabel('t')
ax.legend(loc='upper left')
plt.axis([-1.3, -0.7, 0.9, 1.05])
plt.tight_layout()
fig.savefig('./figures/fig_TP2_fig4.png')
fig.show()

```

Python :

Table A.18: Implémentation de la figure 12.4.

```

K=1000
fe_l_fun = lambda k: k+0.5
fe_l=np.array([])

nom_fichier='fig_TP2_fig5.pkl'
if seb.isfile(nom_fichier):
    dc = seb.load(nom_fichier)
    a_consulter = True
    fe_l_=dc['fe_l']
    err_l_=dc['err_l']
else:
    a_consulter = False
err_l_=np.array([])
for k in range(K):
    if a_consulter:
        if k<len(fe_l_):
            assert np.abs(fe_l_fun(k)-fe_l_[k])<1e-8
            err_l_=np.append(err_l_,err_l_[k])
            fe_l_=np.append(fe_l_,fe_l_fun(k))
            continue
        else:
            assert fe_l_fun(k)>max(fe_l_)
    Te=1/fe_l_fun(k)
    print('k=',k)
    te=np.arange(-2,2,Te)
    fun = lambda t: s1(t)-s_reconstruit(te,s1(te),t)

```

```

err_1=np.append(err_1,seb.erreur_quad(fun,(-2,2)))
fe_1 =np.append(fe_1,fe_1_fun(k))
seb.save(nom_fichier,['fe_1','err_1'],[fe_1,err_1])

fig,ax = plt.subplots()
ax.plot(fe_1,err_1)
ax.set_xlabel('fe')
ax.set_ylabel('err')
plt.tight_layout()
fig.savefig('./figures/fig_TP2_fig5a.png')
fig.show()
fig,ax = plt.subplots()
ax.plot(fe_1,10*np.log10(err_1))
ax.set_xlabel('fe')
ax.set_ylabel('10*log10(err)')
plt.tight_layout()
fig.savefig('./figures/fig_TP2_fig5b.png')
fig.show()

```

A.9 Codes pour simuler des variables aléatoires

Python :

Table A.19: Implémentation de la figure 16.3.

```

import numpy.random as nr
Lambda_tilde = np.linspace(1e-1,1e1,100)
%p_gauss      = 1-seb.erf(Lambda_tilde/np.sqrt(2))
def p_gauss(L):
    return 1-seb.erf(L/np.sqrt(2))

p_maj        = 1/Lambda_tilde**2
L            = 1.5
N            = 10**7
x1           = nr.normal(0,1,N)+nr.normal(0,1,N)
p1_app       = np.mean(np.abs(x1)>=L)
def max_(x,y):
    return (x+y+np.abs(x-y))/2

x2a,x2b     = nr.normal(0,1,N),nr.normal(0,1,N)
x2           = max_(x2a,x2b)
assert all(x2>=x2a)
assert all(x2<=np.abs(x2a)+np.abs(x2b))

mu2          = np.mean(x2)
sigma2        = np.std(x2)
p2_app       = np.mean(np.abs(x2-mu2)>=L)

x3           = nr.exponential(1,N)
mu3          = np.mean(x3)
sigma3        = np.std(x3)
p3_app       = np.mean(np.abs(x3-mu3)>=L)

x4           = nr.poisson(1.25,N)

```

```

mu4          = np.mean(x4)
sigma4       = np.std(x4)
p4_app       = np.mean(np.abs(x4-mu4)>=L)

fig,ax = plt.subplots()
ax.plot(Lambda_tilde,p_gauss(Lambda_tilde),label='gaus')
ax.plot(Lambda_tilde,p_maj,label='maj')
ax.plot(L/np.sqrt(2),p1_app,'sb',label='X1+X2')
ax.plot(L/sigma2,p2_app,'sg',label='max(X1,X2)')
ax.plot(L/sigma3,p3_app,'sr',label='Exp')
ax.plot(L/sigma4,p4_app,'sm',label='Poisson')
ax.set_xlabel('lambda/sigma')
ax.set_ylabel('P(|X|>=lambda)')
ax.set_xlim(0,1)
ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C15_1_fig1a.png')
fig.show()

fig,ax = plt.subplots()
ax.plot(L/np.sqrt(2),p1_app-p_gauss(L/np.sqrt(2)),'sb',label='X1+X2')
ax.plot(L/sigma2,    p2_app-p_gauss(L/sigma2),    'sg',label='max(X1,X2)')
ax.plot(L/sigma3,    p3_app-p_gauss(L/sigma3),    'sr',label='Exp')
ax.plot(L/sigma4,    p4_app-p_gauss(L/sigma4),    'sm',label='Poisson')
ax.set_xlabel('lambda/sigma')
ax.set_ylabel('P(|X|>=lambda)')
ax.set_xlim(1,2)
ax.set_ylim(-0.1,0.2)
ax.legend()
ax.plot(Lambda_tilde,np.zeros(Lambda_tilde.shape[0]),'k-')
plt.tight_layout()
fig.savefig('./figures/fig_C15_1_fig1b.png')
fig.show()

```

Python :

Table A.20: Implémentation de la figure ??.

```

def Hinv(f,H,N):
    fe = 2*np.max(f)
    Ni = np.floor((N-1)/2)
    t = np.arange(-Ni/fe,Ni/fe+1/fe,1/fe)
    assert t[0] == -Ni/fe
    assert t.shape[0] == 2*Ni+1, (t.shape[0],2*Ni+1)
    h = seb.TFI(f,H,t)
    return t+Ni/fe,h

def noise(t,H_fun):
    Te = t[1]-t[0]
    t = np.append(np.arange(t[0]-100*Te,t[0],Te),t)
    fe = 1/Te
    f = np.linspace(-fe/2,fe/2,10**3)
    H = H_fun(f)

```

```

th,h = Hinv(f,H,100)
import numpy.random as nr
x      = nr.uniform(0,1,len(t))
y      = seb.convolution(t,x,th,h,t)
y      = y[100:]
y      = y - np.mean(y)
f_P    = np.linspace(-5,5,10**3)
P      = np.real(seb.TFI(f_P,np.abs(H_fun(f_P))**2,0))
y      = y/np.std(y)*np.sqrt(P)
return y

H_fun   = lambda f: 1/np.sqrt(1+2*np.pi**2*f**2)
t       = np.linspace(-10,10,10**4)
b       = noise(t,H_fun)
fig,ax  = plt.subplots()
ax.plot(t,b,label='y(t)')
fig.show()

f       = np.linspace(-5,5,500)
G_app   = np.abs(seb.TF(t,b,f))**2
print(f"moy={np.mean(y):.3g} std**2={np.std(y)**2:.3g} E={seb.TFI(f,np.abs(H_fun(f))**2,0):.3g}")
fig,ax  = plt.subplots()
ax.plot(f,np.abs(H_fun(f))*2,label='G(f)')
ax.plot(f,G_app,label='G_app(f)')
fig.show()

f       = np.linspace(-10,10,10**3)
H       = 1/np.sqrt(1+2*np.pi**2*f**2)
th,h   = Hinv(f,H,50)
H_app  = seb.TF(t,h,f)
plt.close('all')
fig,ax = plt.subplots()
ax.plot(f,np.abs(H),label='H(f)')
ax.plot(f,np.abs(H_app),label='H1(f)')
ax.legend()
plt.tight_layout()
fig.show()

import numpy.random as nr
Te     = th[1]-th[0]
tx     = np.arange(-10,10,Te)
x      = nr.uniform(0,1,len(tx))
y      = seb.convolution(tx,x,th,h,tx)

fig,ax = plt.subplots()
%ax.plot(tx,x,label='x(t)')
ax.plot(tx,y,label='y(t)')
fig.show()

```

```

Te      = tx[1]-tx[0]
%th    = np.arange(-5,5,Te)
f      = np.linspace(-10,10,10**3)
H      = 1/np.sqrt(1+2*np.pi**2*f**2)
%h    = seb.TFI(f,1/np.sqrt(1+2*np.pi**2*f**2),th)
x      = nr.uniform(0,1,len(tx))
X      = seb.TF(tx,x,f)
Y      = H*X
y      = seb.TFI(f,np.abs(Y),tx)
fig,ax = plt.subplots()
%ax.plot(tx,x,label='x(t)')
ax.plot(tx,y,label='y(t)')
fig.show()
%ty    = np.arange(0,10,Te)
%y    = seb.convolution(tx,x,th-5,h,ty)
%tz    = np.arange(-5,5,Te)
%Ry_app = seb.correlation(ty,y,ty,y,tz)
%Gy_app = seb.TF(tz,Ry_app,f)

```

A.10 Codes pour simuler des processus aléatoires

Table A.21: Implémentation de la figure 14.2.

```

import seb
import numpy.random as nr
plt,np = seb.debut()
T     = 7
fe   = 2
Te   = 1/fe
t    = seb.arange(-T/2,T/2,1/fe)
assert len(t)>1
b1,b2,b3 = nr.normal(0,1,len(t)), nr.normal(0,1,len(t)), nr.normal(0,1,len(t))
plt.close('all')
fig,ax = plt.subplots()
ax.plot(t,b1)
ax.plot(t,b2)
ax.plot(t,b3)
ax.set_xlabel('t')
ax.set_ylabel('B(t)')
plt.tight_layout()
fig.savefig('./figures/fig_C8_9_fig1a.png')
fig.show()

ind = seb.where_nearest(t,0)
k    = np.arange(4)
print(np.round(b1[ind]*b1[ind-k]*100)/100)
print(np.round(b2[ind]*b2[ind-k]*100)/100)
print(np.round(b3[ind]*b3[ind-k]*100)/100)

Rb1,Rb2,Rb3 = seb.correlation(t,b1,t,b1,t)/T, seb.correlation(t,b2,t,b2,t)/T, seb.correlation(t,b3,t,
Rb   = (Rb1+Rb2+Rb3)/3
fig,ax = plt.subplots()
ax.plot(t,Rb1)
ax.plot(t,Rb2)

```

```

ax.plot(t,Rb3)
ax.plot(t,Rb,:')
ax.stem(t,(t==0).astype(float),'k-')
ax.set_xlabel('t')
ax.set_ylabel('Rb(t)')
plt.tight_layout()
fig.savefig('./figures/fig_C8_9_fig1b.png')
fig.show()

f = np.linspace(-3,3,10**2)
Gb1,Gb2,Gb3 = np.real(seb.TFTD(t,Rb1,f)), np.real(seb.TFTD(t,Rb2,f)), np.real(seb.TFTD(t,Rb3,f))
Gb = (Gb1+Gb2+Gb3)/3
fig,ax = plt.subplots()
ax.plot(f,Gb1)
ax.plot(f,Gb2)
ax.plot(f,Gb3)
ax.plot(f,Gb,'k:')
ax.plot(f,np.ones(len(f)),'k-')
ax.set_xlabel('f')
ax.set_ylabel('Sb(f)')
plt.tight_layout()
fig.savefig('./figures/fig_C8_9_fig1c.png')
fig.show()

```

Table A.22: Implémentation de la figure 16.2.

```

import seb
import numpy.random as nr
plt,np = seb.debut()
T = 7
fe = 2
Te = 1/fe
t = np.arange(-T/2,T/2,1/fe) #echelle de temps pour l'aleatoire
fev = 20*fe
tv = np.arange(-T/2,T/2,1/fev)#echelle de temps pour la visualisation
assert len(t)>1
t = seb.synchroniser(t)
b1,b2,b3 = nr.normal(0,1,len(t)), nr.normal(0,1,len(t)), nr.normal(0,1,len(t))
ind = (np.floor(tv/Te-np.min(tv)/Te)).astype(int)
bv1,bv2,bv3 = b1[ind], b2[ind], b3[ind]
assert len(bv1)==len(tv)
plt.close('all')
fig,ax = plt.subplots()
ax.plot(tv,bv1)
ax.plot(tv,bv2)
ax.plot(tv,bv3)
ax.set_xlabel('t')
ax.set_ylabel('B(t)')
# ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C8_9_fig2a.png')
fig.show()

Rb1,Rb2,Rb3 = Te*seb.correlation(tv,bv1,tv,bv1,tv), Te*seb.correlation(tv,bv2,tv,bv2,tv), Te*seb.correlation(tv,bv3,tv,bv3,tv)
Rb = (Rb1+Rb2+Rb3)/3

```

```

fig,ax = plt.subplots()
ax.plot(tv,Rb1)
ax.plot(tv,Rb2)
ax.plot(tv,Rb3)
ax.stem(tv,T/fe*(tv==0),'k-')
ax.set_xlabel('t')
ax.set_ylabel('Rb(t)')
# ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C8_9_fig2b.png')
fig.show()

f    = np.linspace(-3,3,10**2)
Gb1,Gb2,Gb3 = fe*np.real(seb.TF(t,Rb1,f)), fe*np.real(seb.TF(t,Rb2,f)), fe*np.real(seb.TF(t,Rb3,f))
Gb  = (Gb1+Gb2+Gb3)/3
fig,ax = plt.subplots()
ax.plot(f,Gb1)
ax.plot(f,Gb2)
ax.plot(f,Gb3)
ax.plot(f,Gb,'k: ')
ax.plot(f,T/fe*np.ones(len(f)),'k-')
ax.set_xlabel('f')
ax.set_ylabel('Gb(f)')
# ax.legend()
plt.tight_layout()
fig.savefig('./figures/fig_C8_9_fig2c.png')
fig.show()

```

Appendix B

Quelques outils pour calculer l'intensité ou la tension

B.1 Association de résistances

Les résistances et impédances en série sont équivalentes à une résistance ou une impédance, celle-ci étant la somme. Les résistances et impédances en parallèles sont équivalentes à une résistance ou une impédance, celle-ci étant l'inverse de la somme des inverses. Ceci est illustré sur la figure B.1. On remarque à ce propos que

$$\frac{Z_1 Z_2}{Z_1 + Z_2} = \frac{1}{\frac{1}{Z_1} + \frac{1}{Z_2}} \quad (\text{B.1})$$

B.2 Diviseur de tension

La figure B.2 illustre le diviseur de tension.

B.3 Théorème de Millman

B.4 Quadripôle

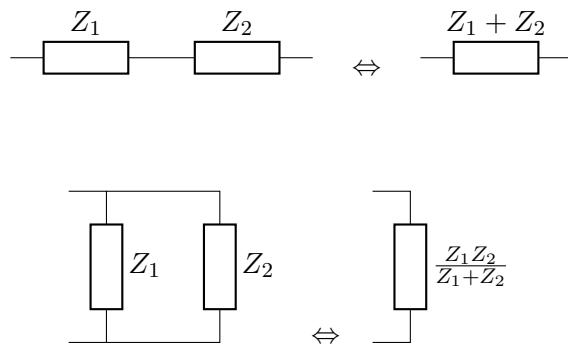


Figure B.1: Illustration des calculs d'impédances en série et en parallèles

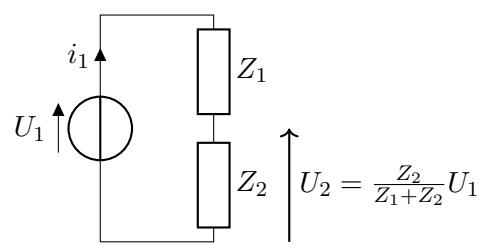


Figure B.2: Diviseur de tension

Appendix C

Notations utilisées

C.1 Notations utilisées dans le cours signal et bruit

- T_x période du signal $x(t)$
- P_x puissance du signal $x(t)$
- E_x énergie du signal $x(t)$
- M_x moyenne du signal $x(t)$
- A_x somme du signal $x(t)$
- $\Pi(t) = \llbracket t \in [-0.5, 0.5] \rrbracket(t)$ est la porte centrée de largeur 1.

C.2 Notations utilisées dans en ce qui concerne l'électronique

-

Un dipôle est ici un composant électronique ayant deux bornes.