Introduction à l'analyse spectrale

1. Introduction

Ce document est une introduction à l'analyse spectrale des signaux périodiques. Après avoir expliqué la décomposition d'un signal périodique en somme de fonctions sinusoïdales, on verra comment effectuer l'analyse spectrale d'un signal échantillonné.

2. Série de Fourier et spectre d'un signal périodique

On considère un signal périodique, représenté par une fonction u d'une variable t réelle à valeurs réelles, de période T et de classe C^1 par morceaux.

La fréquence fondamentale du signal est :

$$f_1 = \frac{1}{T} \tag{1}$$

D'après le théorème de Fourier, cette fonction peut s'écrire comme une somme de sinusoïdes dont les fréquences sont multiples de la fréquence fondamentale. La somme obtenue est la série de Fourier :

$$u(t) = \frac{A_0}{2} + \sum_{n=1}^{P} A_n \cos\left(n\frac{2\pi}{T}t + \phi_n\right) \tag{2}$$

Dans certains cas, la somme peut être stoppée à un rang P fini. Dans d'autres cas, il faut en principe considérer la limite $P \to \infty$.

Le terme de rang n est appelé l'harmonique de rang n du signal : c'est une sinusoïde de fréquence

$$f_n = nf_1 = \frac{n}{T} \tag{3}$$

L'harmonique de rang n est défini par son amplitude A_n (positive) et son déphasage φ_n.

Le terme constant $A_0/2$, qui peut être vu comme le terme de fréquence nulle, est la valeur moyenne du signal :

$$\frac{A_0}{2} = \frac{1}{T}\int_0^T u(t)\,dt \tag{4}$$

Considérons comme exemple une fonction dont la série de Fourier s'arrête au rang $P = 3$. On dit dans ce cas que le signal comporte trois harmoniques. Par convention, la période est prise égale à 1 :

```
import numpy
import math

f1=1.0
def u(t):
    return 0.4+1.0*numpy.cos(2*numpy.pi*f1*t)\
           +0.5*numpy.cos(2*2*numpy.pi*f1*t-numpy.pi/3)\
```

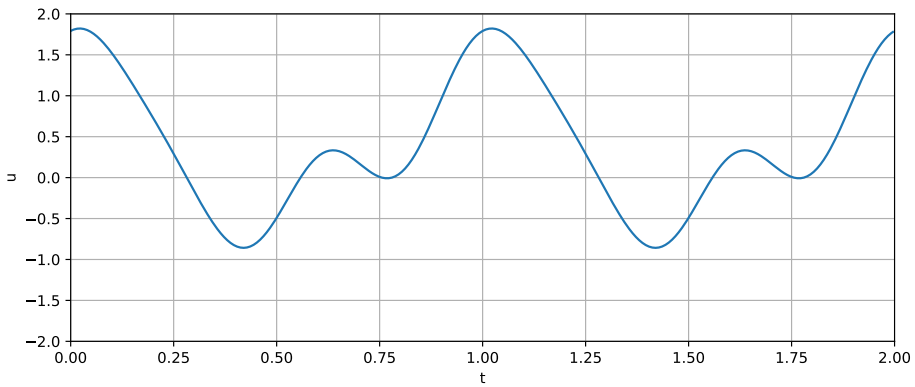
```
+0.2*numpy.cos(3*2*numpy.pi*f1*t+numpy.pi/4)
```

Pour tracer ce signal, il faut l'échantillonner, c'est-à-dire calculer les valeurs de $u(t)$ pour des instants régulièrement répartis sur un intervalle et les placer dans un tableau. Voici un échantillonnage sur deux périodes comportant 500 points :

```
import numpy
N = 500
Tmax = 2.0
Te = Tmax/N
t = numpy.arange(N)*Te
x = u(t)
```

La courbe du signal est obtenue avec la fonction `plot`, qui relie les points par des segments de droites. Si la période d'échantillonnage T_e est petite par rapport à la période du dernièr harmonique (ici celui de rang 3), on obtient ainsi une bonne représentation graphique du signal.

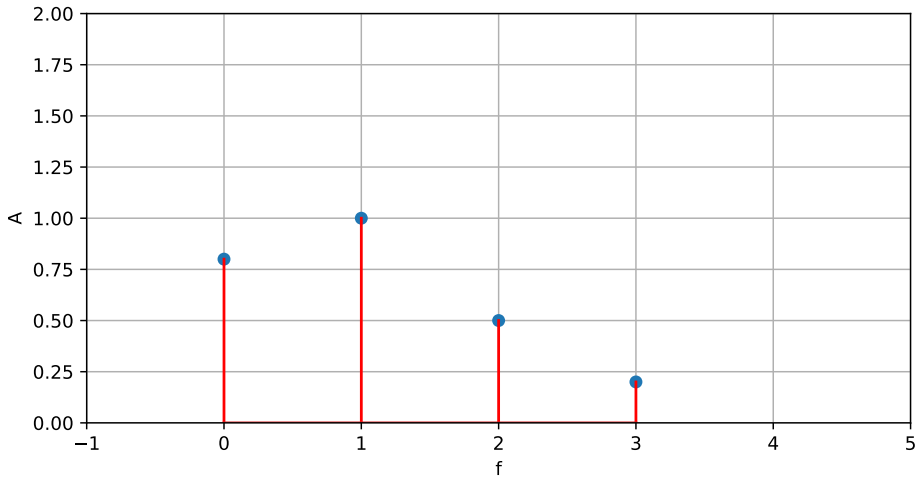
```
from matplotlib.pyplot import *
figure(figsize=(10,4))
plot(t,x)
xlabel('t')
ylabel('u')
axis([0,2,-2,2])
grid()
```



Le spectre du signal est la représentation graphique de l'amplitude A_n en fonction de la fréquence. Un signal périodique a en théorie un spectre discret formé de raies, chacune correspondant à un harmonique. Pour cet exemple, il y a 4 raies : une pour la valeur moyenne (fréquence nulle) et 3 raies pour les harmoniques de rang 1, 2 et 3 :

```
figure(figsize=(8,4))
stem([0,1,2,3],[0.8,1.0,0.5,0.2],'r')
xlabel('f')
```

```
ylabel('A')
axis([-1,5,0,2])
grid()
```



La raie de fréquence nulle a une amplitude égale au double de la valeur moyenne du signal (la composante dite *continue*).

La première courbe (u en fonction de t) est la représentation temporelle du signal. Le spectre est la représentation fréquentielle du signal. En principe, il faudrait aussi tracer la phase φ_n pour avoir une représentation complète.

Pour les calculs, il est commode d'introduire la série de Fourier sous forme complexe :

$$u(t) = \sum_{n=-P}^{P} c_n \exp\left(in\frac{2\pi}{T}t\right) \tag{5}$$

Le coefficient c_n est un nombre complexe appelé coefficient de Fourier. Il se calcule à partir de la fonction u avec l'intégrale suivante :

$$c_n = \frac{1}{T}\int_0^T u(t) \exp\left(-in\frac{2\pi}{T}t\right) \tag{6}$$

Puisque $u(t)$ est réel, le coefficient de Fourier vérifie la propriété suivante :

$$c_n = c_{-n}^* \tag{7}$$

où l'étoile désigne le complexe conjugué. La somme s'écrit donc :

$$u(t) = c_0 + \sum_{n=1}^{P} c_n \exp\left(in\frac{2\pi}{T}t\right) + c_{-n} \exp\left(-in\frac{2\pi}{T}t\right) \tag{8}$$

$$= c_0 + \sum_{n=1}^{P} 2\,Re\left[c_n \exp\left(in\frac{2\pi}{T}t\right)\right] \tag{9}$$

Par identification, on en déduit :

$$c_0 = \frac{A_0}{2} \tag{10}$$

$$c_n = \frac{A_n}{2} e^{i\phi_n} \ (n > 0) \tag{11}$$

3. Transformée de Fourier discrète

3.a. Définition

La transformée de Fourier discrète (TFD) est la transformation qui permet de calculer le spectre d'un signal discret, obtenu par échantillonnage d'un signal continu. Considérons un échantillonnage de la fonction u sur l'intervalle $[0, T]$, comportant N points et défini par :

$$t_k = k\frac{T}{N} \ (0 \leq k \leq N-1) \tag{12}$$

$$u_k = u(t_k) \tag{13}$$

On définit la période d'échantillonnage et la fréquence d'échantillonnage par :

$$T_e = \frac{T}{N} \tag{14}$$

$$f_e = \frac{1}{T_e} \tag{15}$$

Une valeur approchée de l'intégrale (6) définissant les coefficients de Fourier peut être obtenue par la méthode des rectangles :

$$c_n \simeq \frac{1}{T} \sum_{k=0}^{N-1} \frac{T}{N} u_k \exp\left(-i\frac{2\pi\, n\, k}{N}\right) \tag{16}$$

Par définition, la transformée de Fourier discrète ([1]) est l'application qui aux N nombres u_k associe les N nombres complexes suivants $(0 \leq n \leq N-1)$:

$$\tilde{U_n} = \frac{1}{N} \sum_{k=0}^{N-1} u_k \exp\left(-i\frac{2\pi nk}{N}\right) \tag{17}$$

La TFD vérifie la propriété suivante (si u_k est réel) :

$$\tilde{U}_n = \tilde{U}^*_{N-n} \tag{18}$$

La transformée de Fourier discrète est calculée numériquement avec l'algorithme dit de Transformée de Fourier rapide. La fonction qui effectue ce calcul (sur un oscilloscope ou dans un logiciel) est souvent désignée par FFT (Fast Fourier Transform).

3.b. Exemple : polynôme trigonométrique

Si u est un polynôme trigonométrique, il existe un rang P fini tel que $\forall n > P$ on a $c_n = 0$. Autrement dit, la série de Fourier s'arrête à un rang fini. Il existe donc dans le spectre du signal une fréquence maximale :

$$f_{max} = \frac{P}{T} \tag{19}$$

On suppose que la condition de Nyquist-Shannon est vérifiée : la fréquence d'échantillonnage est supérieure au double de la fréquence maximale du spectre

$$f_e > 2f_{max} \tag{20}$$

ou encore :

$$N > 2P \tag{21}$$

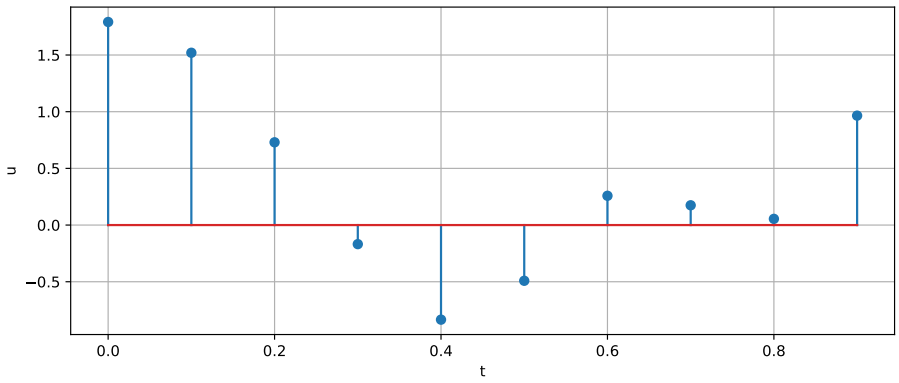
Si cette condition est vérifiée, alors les $P+1$ premières valeurs de la TFD sont exactement les coefficients de Fourier :

$$c_n = \tilde{U}_n \text{ pour } 0 \leq n \leq P \tag{22}$$

Cette propriété est étonnante puisque la TFD est initialement définie pour donner des valeurs approchées des coefficients de Fourier. Sa démonstration est donnée plus loin (paragraphe 3.d).

Pour voir comment la TFD permet d'obtenir le spectre du signal, reprenons l'exemple précédent. Comme la période est égale à 1, l'indice n correspond exactement à la fréquence. La plus grande fréquence du spectre est $P = 3$ (harmonique de rang 3). Pour respecter la condition, il faut donc échantillonner à au moins 6 points par période. Voyons le résultat pour $N = 10$. La période du signal étant connue a priori, on peut échantillonner exactement 10 points sur une période :

```
N = 10
Tmax = 1.0
Te = Tmax/N
t = numpy.arange(0,N)*Te
x=u(t)
figure(figsize=(10,4))
stem(t,x)
xlabel('t')
ylabel('u')
grid()
```



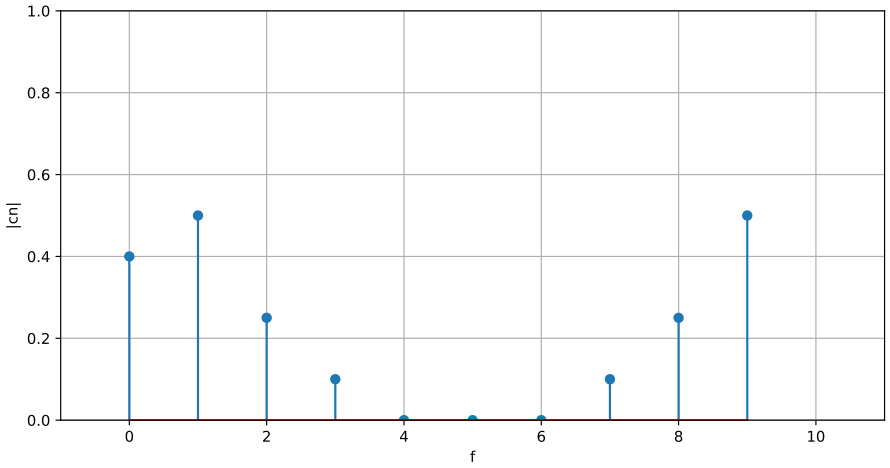
Voici le calcul de la transformée de Fourier discrète :

```
import numpy.fft
tfd = numpy.fft.fft(x)
```

```
print(tfd)
--> array([  4.00000000e+00 +0.00000000e+00j,
         5.00000000e+00 -2.10942375e-15j,
         1.25000000e+00 -2.16506351e+00j,
         7.07106781e-01 +7.07106781e-01j,
         7.49400542e-16 +4.44089210e-16j,
         1.77635684e-15 +2.22044605e-15j,
         7.49400542e-16 -4.44089210e-16j,
         7.07106781e-01 -7.07106781e-01j,
         1.25000000e+00 +2.16506351e+00j,   5.00000000e+00 -2.60902411e-15j])
```

On trace le module en fonction de la fréquence. On doit diviser par N car la fonction `numpy.fft.fft` calcule la TFD sans le facteur $1/N$. Ici la fréquence est égale à l'indice n :

```
figure(figsize=(10,5))
f=numpy.arange(0,N,1)
stem(f,numpy.absolute(tfd)/N)
xlabel("f")
ylabel("|cn|")
axis([-1,11,0,1])
grid()
```



Les 4 premières valeurs de la TFD donnent bien les coefficients de Fourier de rang 0 à 3. Les éléments d'indice 4,5 et 6 sont nulles, aux erreurs d'arrondis près (erreurs de l'ordre de $1e-16$). Les trois derniers éléments (indices 7 à 9) sont les conjugués des trois coefficients c_1, c_2 et c_3 ; ils ont donc le même module. Le terme d'indice 1 est conjugué du terme d'indice 9, celui d'indice 2 conjugué du terme d'indice 8, etc. La fréquence centrale du spectre est égale à la moitié de la fréquence d'échantillonnage ; c'est la fréquence de Nyquist. Ici, la fréquence de Nyquist est $f_n = 5$. Comme la condition de Nyquist-Shannon est respectée, cette fréquence est supérieure à $P = 3$. La partie du spectre située à gauche de cette fréquence est le spectre du signal continu $u(t)$. La totalité du spectre est celui du signal discret. La partie située à droite de la fréquence de Nyquist est l'image de la partie gauche (à l'exclusion de la composante de fréquence nulle). On voit que la condition de Nyquist-Shannon se traduit par le non chevauchement du spectre et de son image.

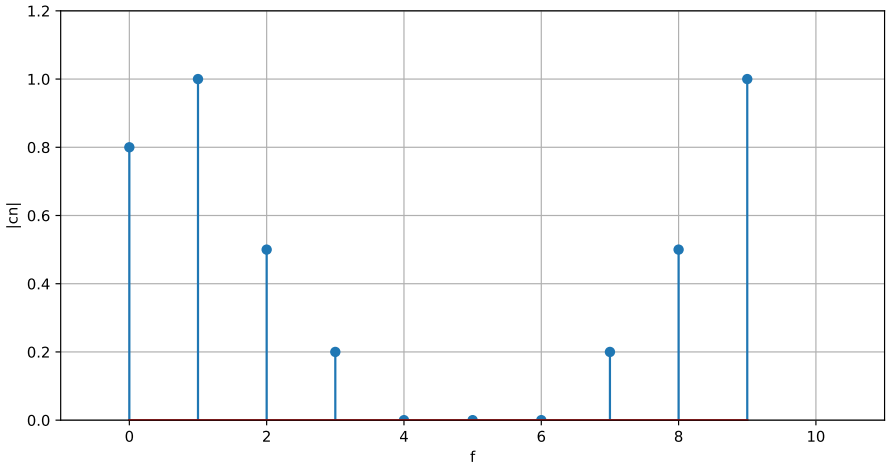
En fait, le spectre du signal discret est périodique, de période égale à la fréquence d'échantillonnage. La relation définissant la TFD vérifie en effet la relation :

$$\tilde{U}_{N+n} = \tilde{U}_n \tag{23}$$

La TFD que nous avons calculée donne donc les valeurs de ce spectre sur une période. Cependant, ce qui nous intéresse en pratique est plutôt le spectre du signal continu $u(t)$, qui est donné par la première moitié de la TFD.

En pratique, on s'intéresse aux amplitudes A_n des harmoniques plutôt qu'aux valeurs des coefficients de Fourier c_n. Il faut alors multiplier les coefficient par deux :

```
figure(figsize=(10,5))
f=numpy.arange(0,N,1)
stem(f,numpy.absolute(tfd)*2/N)
xlabel("f")
ylabel("|cn|")
axis([-1,11,0,1.2])
grid()
```



On remarque que la composante de fréquence nulle du spectre a une amplitude égale au double de la valeur moyenne. Dans le cas présent, on pourrait facilement diviser le terme d'indice 0 par deux pour obtenir directement la composante de fréquence nulle du signal, mais cette méthode n'est pas toujours applicable sur les spectres expérimentaux.

3.c. Transformée de Fourier discrète inverse

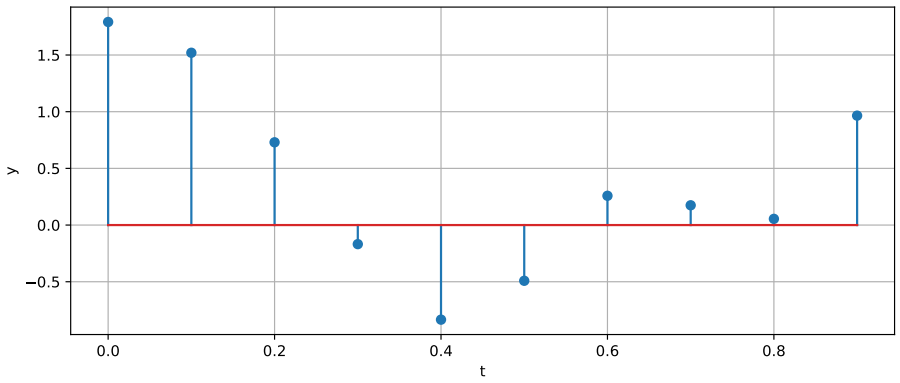
L'information présente dans le signal échantillonné est entièrement contenue dans sa TFD. On peut en effet calculer le signal à partir de sa TFD par la relation suivante (voir [1] pour la démonstration) :

$$u_k = \sum_{n=0}^{N-1} \tilde{U}_n \exp\left(i\frac{2\pi nk}{N}\right) \tag{24}$$

La transformation qui permet ainsi de retrouver le signal discret est la transformation de Fourier discrète inverse. Elle ressemble beaucoup à la TFD directe : on remarque le changement de signe dans l'exponentielle et l'absence du facteur $1/N$. En ce qui concerne ce facteur, il existe d'ailleurs différentes conventions pour la définition de la TFD.

Comme exemple, calculons la TFD inverse de la TFD obtenue précédemment :

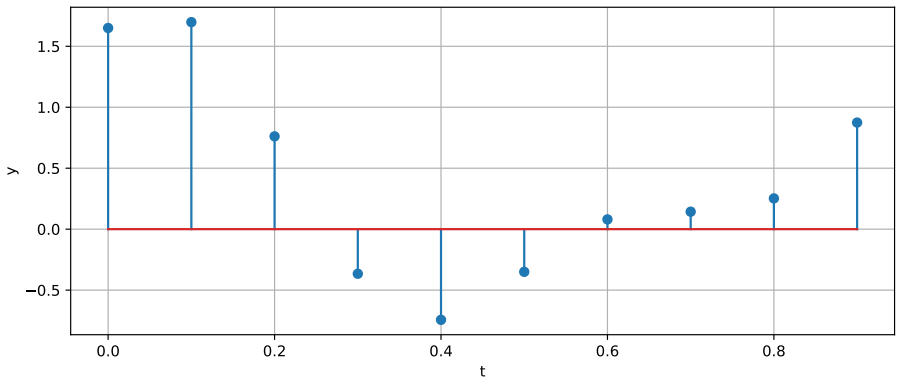
```
y = numpy.fft.ifft(tfd)
figure(figsize=(10,4))
stem(t,y)
xlabel('t')
ylabel('y')
grid()
```



On retrouve bien le signal discret de départ.

Il est intéressant de modifier la TFD avant de calculer la TFD inverse. On peut par exemple enlever l'harmonique de rang 3. Pour cela, il faut annuler le coefficient d'indice 3 et son image, d'indice $N - 3 = 7$:

```
i=3
tfd[i] = 0
tfd[N-i] = 0
y = numpy.fft.ifft(tfd)
figure(figsize=(10,4))
stem(t,y)
xlabel('t')
ylabel('y')
grid()
```



Le résultat est le signal discret qui serait obtenu par échantillonnage du signal $u(t)$ auquel on aurait enlevé l'harmonique de rang 3. On a donc là une méthode de filtrage très sélective, appelée filtrage dans le domaine fréquentiel, ou filtrage par transformée de Fourier. Elle peut être utilisée lorsqu'on doit filtrer un signal discret dont la taille N est fixée à l'avance (et pas trop grand). Cependant, cette méthode est plus difficile à mettre en œuvre pour faire un filtrage temps-réel, pour lequel il y a un flux continu de données à traiter. Dans ce cas, on effectue plutôt un filtrage dans le domaine temporel, appelé aussi filtrage par convolution. Voir à ce sujet le document Introduction aux filtres numériques.

3.d. Relation entre la TFD et les coefficients de Fourier

Ce paragraphe donne la démonstration de l'égalité entre les coefficients de Fourier et la TFD.

On considère la série de Fourier pour l'instant $t_k = kT/N$:

$$u_k = \sum_{m=-\infty}^{+\infty} c_m \exp\left(im\frac{2\pi}{T}\frac{kT}{N}\right) = \sum_{m=-\infty}^{+\infty} c_m \exp\left(i\frac{2\pi mk}{N}\right) \tag{25}$$

On se place pour l'instant dans le cas théorique général, où la somme doit être étendue à l'infini. Considérons la division entière de l'indice m par N :

$$m = qN + n \tag{26}$$

q est l'entier résultat de cette division et n est le reste. La somme sur l'indice m peut s'écrire comme une double somme, sur n et sur q. En regroupant tous les termes qui ont le même reste, on obtient ainsi :

$$u_k = \sum_{n=0}^{N-1}\left(\sum_{q=-\infty}^{+\infty} c_{qN+n}\right)\exp\left(i\frac{2\pi nk}{N}\right) \tag{27}$$

En identifiant cette expression avec celle de la TFD inverse, on montre que :

$$\tilde{U}_n = \sum_{q=-\infty}^{\infty} c_{qN+n} \tag{28}$$

Pour le coefficient de Fourier de rang n, l'erreur de l'approximation par $\tilde{U}_n$ est donc :

$$\tilde{U}_n - c_n = \sum_{q\neq 0} c_{qN+n} \tag{29}$$

On se place à présent dans le cas où il existe un rang P tel que pour $|k| > P$ on ait $c_k = 0$. On suppose de plus que $N > 2P$ (condition de Nyquist-Shannon).

Soit un entier n tel que $0 \leq n \leq P$. Pour $q > 0$, on a $qN+n > 2qP+n > 2P+n > P$ donc $c_{qN+n} = 0$. De même pour $q < 0$ on a $qN + n < 2qP + n < -2P + n < -P$ donc $c_{qN+n} = 0$. Tous les termes de la somme sont donc nuls :

$$\tilde{U}_n - c_n = 0 \text{ pour } 0 \leq n \leq P \tag{30}$$

Cela démontre l'égalité entre les $P + 1$ premiers termes de la TFD et les coefficients de Fourier, pourvu que la condition de Nyquist-Shannon soit respectée.

4. Analyse spectrale des signaux réels

4.a. Fenêtre d'analyse

Un signal périodique réel a une période qui, le plus souvent, n'est pas connue *a priori*. L'objectif de l'analyse spectrale est justement de déterminer les fréquences qu'il contient. Par ailleurs, sa périodicité peut être imparfaite, voire très grossière. On voit donc que l'approche précédente, qui consistait à échantillonner le signal sur sa période T, n'est pas applicable en pratique.

Au lieu de cherche à échantillonner sur une période, on considère une durée T grande devant la période supposée. Par exemple, pour un son de fréquence 400 Hz, on peut prendre $T = 1\,\text{s}$. On obtient ainsi un spectre dont les raies sont les coefficients de Fourier d'une fonction de période T. Les fréquences de ces raies sont donc multiples de $1/T$. La durée T est la largeur de la fenêtre d'analyse. La résolution fréquentielle du spectre obtenu est $1/T$.

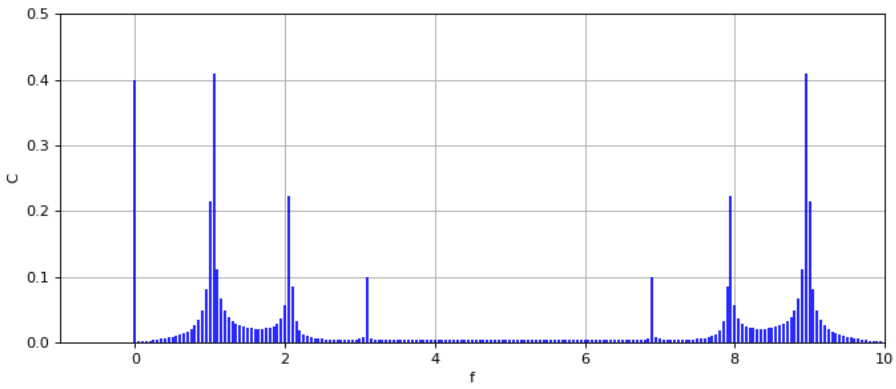
4.b. Exemple : fonction échantillonnée

On reprend le signal déjà utilisé en modifiant sa fréquence :

```
f1 = 1.0324
```

On choisit une durée d'analyse $T = 20$, qui donnera une résolution fréquentielle de 0,05. La fréquence d'échantillonnage doit respecter la condition de Nyquist-Shannon, c'est-à-dire $f_e > 7$.

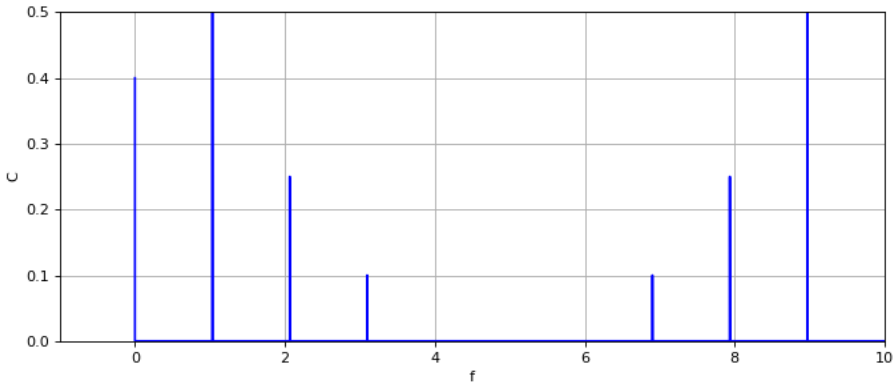
```
T=20.0
fe=10.0
N=int(T*fe)
t = numpy.arange(0,N)*Te
x=u(t)
tfd=numpy.fft.fft(x)
f=numpy.arange(0,N)*1.0/T
figure(figsize=(10,4))
vlines(f,[0],numpy.absolute(tfd)/N,'b')
xlabel('f')
ylabel('C')
axis([-1,10,0,0.5])
grid()
```



Le spectre obtenu est composé de raies très resserées, espacées de $1/T$. Le spectre de la fonction apparaît comme la courbe dessinée par ces raies. La démonstration de cette propriété sort du cadre de cette introduction, car elle fait appel à la notion de transformée de Fourier.

On voit sur cet exemple que le spectre obtenu est imparfait : la position du maximum des raies est définie au mieux avec une précision de $1/T$. Dans le cas présent, cela induit une erreur bien visible sur la hauteur des raies. Pour améliorer la précision, il faut augmenter la durée de la fenêtre d'analyse. Voyons le résultat avec $T = 10000$:

```
T=10000.0
fe=10.0
N=int(T*fe)
t = numpy.arange(0,N)*Te
x=u(t)
tfd=numpy.fft.fft(x)
f=numpy.arange(0,N)*1.0/T
figure(figsize=(10,4))
plot(f,numpy.absolute(tfd)/N,'b')
xlabel('f')
ylabel('C')
axis([-1,10,0,0.5])
grid()
```



La résolution fréquentielle est bien meilleure. Les amplitudes des harmoniques attendues (d'après la définition du signal) sont correctes : $c_0 = 0,4, c_1 = 0,5, c_2 = 0,25, c_3 = 0,1$. Remarque : on a ici utilisé la fontion `plot` et non `vlines` car cette dernière n'est pas optimisée pour le traitement d'un gros volume de données.

Dans certains cas, l'augmentation de la durée de la fenêtre d'analyse n'est pas possible, ou bien n'est pas suffisante pour obtenir précisément la hauteur des raies. On peut alors utiliser la technique du remplissage par des zéros associée à un fenêtrage progessif, expliquée dans Analyse spectrale d'un signal numérique.

4.c. Exemple : analyse d'un son

Voyons à présent un exemple plus réaliste, l'analyse du son émis par un instrument de musique. Il s'agit du son émis par une clarinette (note La_3) pendant une durée d'environ 3 secondes. Evidemment, la largeur de la fenêtre d'analyse est limitée par cette durée.

Le son est enregistré dans un fichier WAV. On commence par lire le fichier et on extrait la fréquence d'échantillonnage, le nombre d'échantillons et la durée du son :

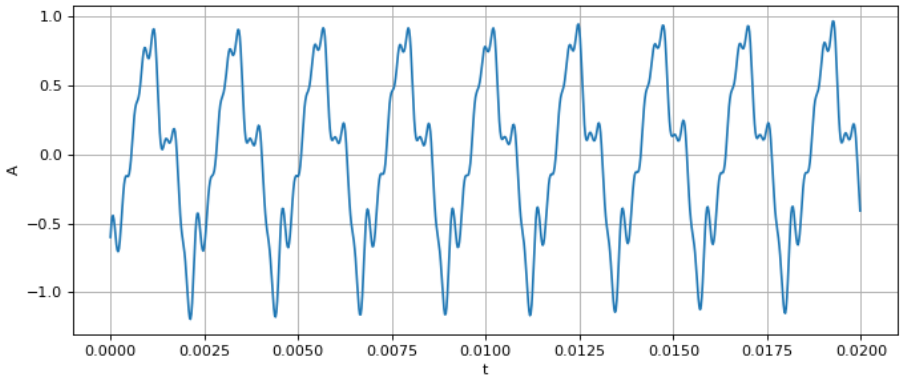
```
import scipy.io.wavfile as wave
fe,data = wave.read("clarinette_la3.wav")
N = data.size
T = N*1.0/fe
data =data/data.max()


print(fe)
--> 44100

print(T)
--> 2.8908843537414968
```

Pour obtenir une représentation temporelle du signal (la forme d'onde), il faut se limiter à une fenêtre comportant une dizaine d'oscillations, soit ici environ 0,02 s :

```
te = 1.0/fe
T = 0.02
N = int(T/te)
t = numpy.arange(N)*te
x = data[0:N]
figure(figsize=(10,4))
plot(t,x)
xlabel("t")
ylabel("A")
grid()
```

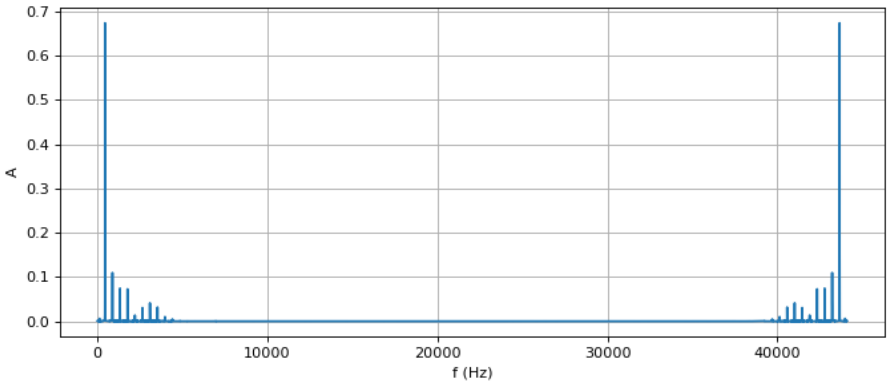


La forme d'onde laisse deviner un spectre très riche en harmoniques.

Pour faire l'analyse spectrale, on choisit une fenêtre de largeur $T = 1$ Hz, qui donnera une précision de 1 Hz sur la fréquence :

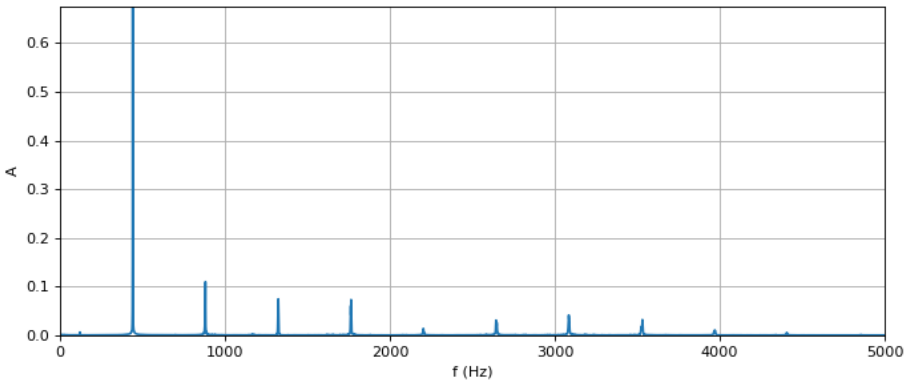
```
T=1.0
N=int(T/te)
x = data[0:N]
tfd = numpy.fft.fft(x)
f = numpy.arange(N)*1.0/T
```

```
a = numpy.absolute(tfd)*2/N
figure(figsize=(10,4))
plot(f,a)
xlabel("f (Hz)")
ylabel("A")
grid()
```



On s'intéresse au spectre du son en tant que signal continu, c'est-à-dire à la première moitié. Voyons le détail dans la bande $[0,5\ kHz]$:

```
figure(figsize=(10,4))
plot(f,a)
xlabel("f (Hz)")
ylabel("A")
grid()
axis([0,5000,0,a.max()])
```



On repère sur ce spectre le fondamental (à 440 Hz) et des harmoniques de rang 2 à 8. Les harmoniques de rang élevé (par exemple 7) ont une fréquence très élevée (pour l'oreille). Bien que leur amplitude soit relativement faible, elles contribuent au timbre du son.

Références

[1] Gasquet C., Witomski P., *Analyse de Fourier et applications*, (Masson, 1995)