Analyse spectrale d'un signal numérique

1. Introduction

Le spectre d'un signal analogique est obtenu avec sa transformée de Fourier. Le spectre d'un signal échantillonné est obtenu par sa transformée de Fourier discrète (TFD). Pour la définition et l'utilisation de la TFD, voir le document Introduction à l'analyse spectrale. Ce document décrit l'analyse spectrale d'un point de vue pratique.

2. TFD d'un signal échantillonné

Considérons comme exemple un signal périodique (de période 1) :

```
import math
import numpy as np
from matplotlib.pyplot import *
from numpy.fft import fft

def signal(t):
    return 0.1+np.cos(2*np.pi*t)+0.5*np.cos(2*2*np.pi*t) \
    +0.25*np.sin(3*2*np.pi*t)+0.01*np.sin(4*2*np.pi*t)
```

Le signal est échantillonné sur une durée T qui doit être beaucoup plus grande que sa période. On fixe cette durée à une valeur qui ne coïncide pas avec une période du signal, comme c'est le cas dans la numérisation des signaux physiques. Dans le cas présent, le choix de l'origine du temps est sans importance ; on choisit l'intervalle $[0, T]$. La fréquence d'échantillonnage soit être supérieure au double de la plus grande fréquence du signal, ici 4.

```
T=100.345821
fe = 10
t = np.arange(start=0.0,stop=T,step=1.0/fe)
echantillons = signal(t)
```

On calcule la transformée de Fourier discrète puis sa norme (spectre) :

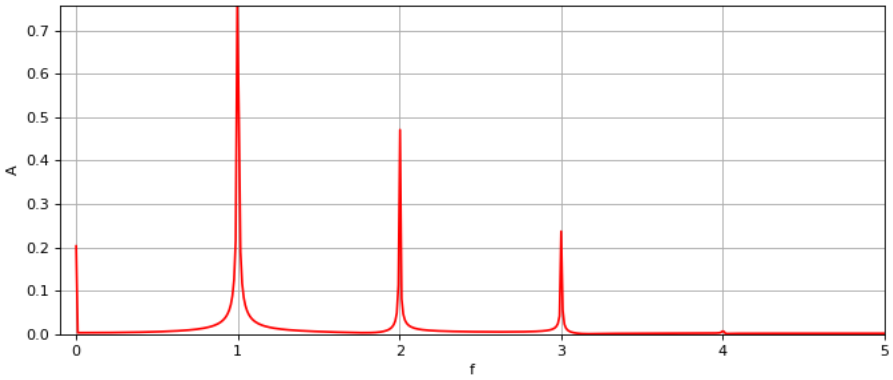
```
tfd = fft(echantillons)
N=len(echantillons)
spectre = np.absolute(tfd)*2/N
```

L'échelle des fréquences est construite sachant que l'espacement fréquentiel de deux points de la TFD est l'inverse de la durée T :

```
freq=np.arange(N)*1.0/T
```

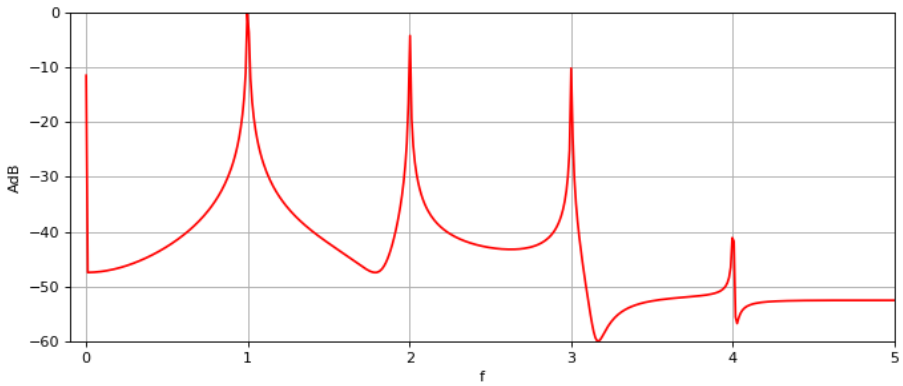
On trace le spectre pour des fréquences allant de zéro à la moitié de la fréquence d'échantillonnage :

```
figure(figsize=(10,4))
plot(freq,spectre,'r')
xlabel('f')
ylabel('A')
axis([-0.1,fe/2,0,spectre.max()])
grid()
```



L'harmonique de rang 4 est quasiment invisible. Une représentation en décibel permet de voir ces composantes faibles :

```
spectre_db = 20*np.log10(spectre/spectre.max())
figure(figsize=(10,4))
plot(freq,spectre_db,'r')
xlabel('f')
ylabel('AdB')
axis([-0.1,fe/2,spectre_db.min(),spectre_db.max()])
grid()
```

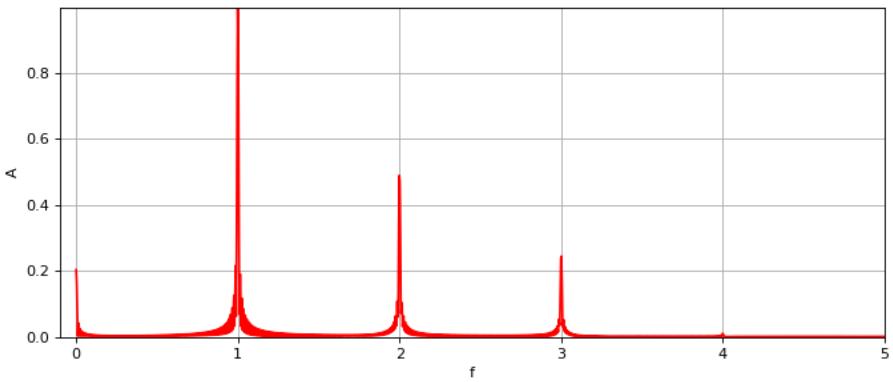


Si le signal était échantillonné sur une durée infinie, les raies seraient de largeur nulle (puisque le signal est périodique). L'élargissement de la base des raies est un effet de

la durée finie de l'échantillon, c'est-à-dire de l'application d'une fenêtre de troncature au signal. S'il n'est pas gênant dans le cas d'un spectre discret, cet élargissement peut réduire la résolution dans le cas d'un spectre continu. On constate également des erreurs sur les hauteurs des raies, qui viennent du fait que la résolution fréquentielle (inverse de la durée de l'échantillon) est insuffisante pour saisir le maximum des raies.

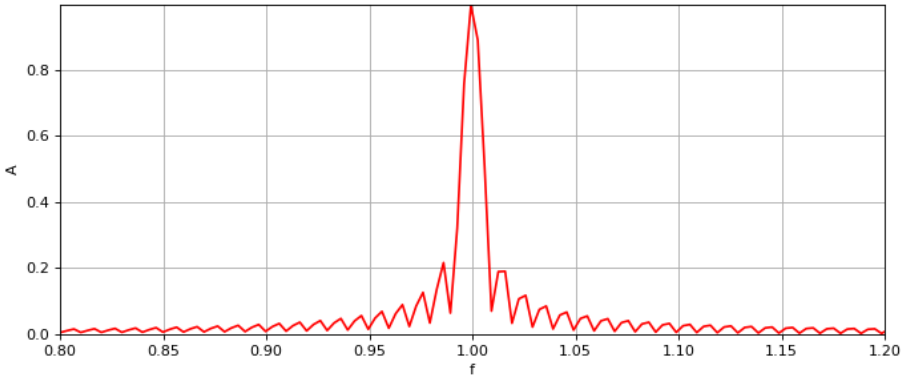
Pour obtenir des hauteurs de raie correctes, il faudrait faire une interpolation dans le domaine fréquentiel. Cette opération se réalise très simplement dans le domaine temporel, en complétant le signal échantillonné par des zéros avant et après. Voici un exemple, avec une nombre de zéros égal au nombres d'échantillons :

```
zeros=np.zeros(N)
echantillons_z = np.concatenate((zeros,echantillons,zeros))
spectre_z = np.absolute(fft(echantillons_z))*2/N
N_z=len(echantillons_z)
freq_z=np.arange(N_z)*fe/N_z
figure(figsize=(10,4))
plot(freq_z,spectre_z,'r')
xlabel('f')
ylabel('A')
axis([-0.1,fe/2,0,spectre_z.max()])
grid()
```




```
axis([0.8,1.2,0,spectre_z.max()])
```

Voici un détail montrant la raie du fondamental :



La hauteur des raies est correcte. L'inconvénient de cette méthode est l'apparition de lobes secondaires autour du maximum principal. Ces lobes sont causés par la fenêtre de troncature rectangulaire. Ils ne sont pas gênants pour un signal périodique car les raies sont bien séparées. Il suffit de savoir que ces lobes sont dûs à la fenêtre et pas au signal lui-même. Pour un signal non périodique pouvant présenter des raies très voisines, ou même un spectre continu, ces lobes sont gênants. Il convient donc de les réduire en modifiant la fenêtre de troncature.

3. Fenêtre de troncature

Soit $u(t)$ le signal analysé sur l'intervalle $[0, T]$. La fenêtre de troncature $w(t)$ est une fonction nulle en dehors de cette intervalle. L'analyse spectrale consiste alors à calculer la transformée de Fourier suivante :

$$S(f) = \int_0^T u(t)w(t)\exp(-i2\pi\, f\, t)\, dt \tag{1}$$

Dans l'exemple précédent, la fenêtre de troncature est rectangulaire ($w(t) = 1$).

Si l'on note $U(f)$ la transformée de Fourier de $u(t)$ et $W(f)$ celle de la fenêtre $w(t)$, la transformée de Fourier calculée est le produit de convolution :

$$S(f) = \int_{-\infty}^{+\infty} U(f - x)W(x)\, dx \tag{2}$$

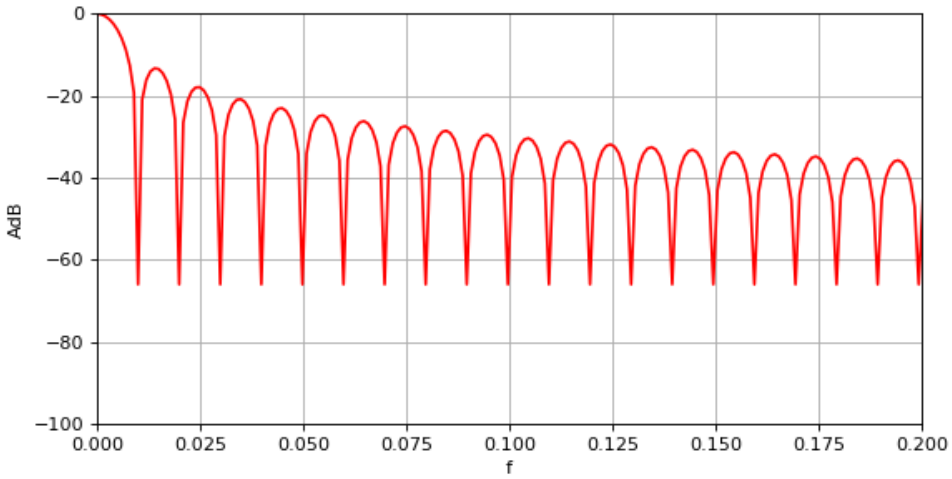
La transformée de Fourier de la fenêtre rectangulaire est :

$$W(f) = T\exp(-i\pi\, f\, T)sinc(\pi f\, T) \tag{3}$$

Pour caractériser l'effet de la fenêtre, on trace le module de sa transformée de Fourier en décibel. Bien que la TF de la fenêtre rectangulaire soit connue, nous allons calculer son spectre au moyen de la TFD. Pour cela, il faut échantillonner la fenêtre sur une durée beaucoup plus grande que sa largeur.

```
def fenetre(t):
    if t < T:
        return 1.0
    else:
        return 0.0
Tf = T*10
t = np.arange(start=0.0,stop=Tf,step=1.0/fe)
Nf = t.size
echantillons_fenetre = np.zeros(Nf)
for k in range(Nf):
    echantillons_fenetre[k] = fenetre(t[k])
spectre_fenetre = np.absolute(fft(echantillons_fenetre))
spectre_fenetre_db = 20*np.log10(spectre_fenetre/spectre_fenetre.max())
Nf = spectre_fenetre_db.size
freq_fenetre = np.zeros(Nf)
for k in range(Nf):
    freq_fenetre[k] = 1.0/Tf*k
```

```
figure(figsize=(8,4))
plot(freq_fenetre,spectre_fenetre_db,'r')
xlabel('f')
ylabel('AdB')
axis([0,0.2,-100,spectre_fenetre_db.max()])
grid()
```

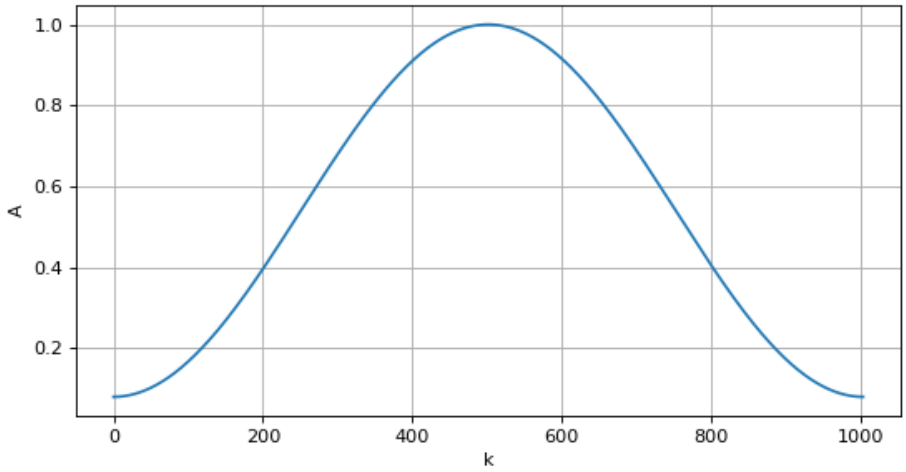


Pour réduire les lobes constatés plus haut, on utilise une fonction de troncature dont la valeur décroit vers les bords. Par exemple, la fenêtre de Hamming définie sur l'intervalle $[0, T]$ par :

$$w(t) = 0,54 - 0,4 \cos\left(\frac{2\pi}{T} t\right) \tag{4}$$

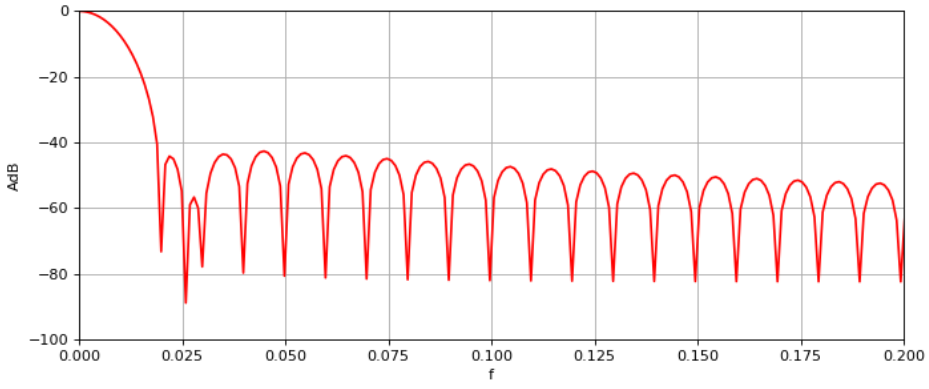
Le module `scipy.signal` fournit des tableaux d'échantillons pour différentes fenêtres de troncature :

```
from scipy import signal
hamming = signal.hamming(N)
figure(figsize=(8,4))
plot(hamming)
xlabel('k')
ylabel('A')
grid()
```



Pour obtenir son spectre, il faut calculer la TFD en complétant le tableau d'échantillons avec des zéros. La fonction `fft` permet de faire cela :

```
spectre_fenetre = np.absolute(fft(hamming,Nf))
spectre_fenetre_db = 20*np.log10(spectre_fenetre/spectre_fenetre.max())
figure(figsize=(10,4))
plot(freq_fenetre,spectre_fenetre_db,'r')
xlabel('f')
ylabel('AdB')
axis([0,0.2,-100,spectre_fenetre_db.max()])
grid()
```

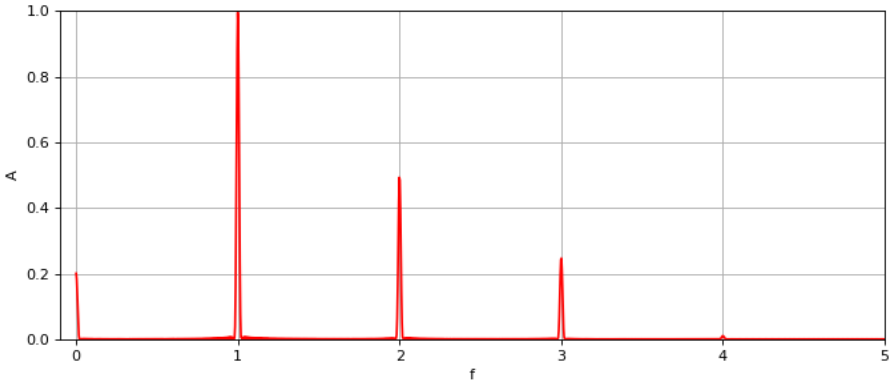


En comparant au spectre de la fenêtre rectangulaire, on voit que le maximum principal est plus large et que les maxima secondaires sont beaucoup plus faibles.

L'utilisation de la fenêtre de Hamming pour calculer un spectre nécessite une division par 0,54 pour obtenir des hauteurs de raie correctes. Voyons le spectre du signal de départ multiplié par la fenêtre de Hamming et complété par des zéros :

```
echantillons = echantillons*hamming
echantillons_z = np.concatenate((zeros,echantillons,zeros))
spectre_z = np.absolute(fft(echantillons_z))*2/N/0.54
```

```
N_z=len(echantillons_z)
freq_z=np.arange(N_z)*fe/N_z
figure(figsize=(10,4))
plot(freq_z,spectre_z,'r')
xlabel('f')
ylabel('A')
axis([-0.1,fe/2,0,1])
grid()
```



Avec cette méthode, les raies ont la bonne hauteur et les lobes secondaires sont très faibles. Le seul inconvénient est une largeur de raie plus grande qu'avec la fenêtre rectangulaire. La raie de fréquence nulle a une hauteur double de la valeur moyenne du signal.

4. Fonction de calcul du spectre

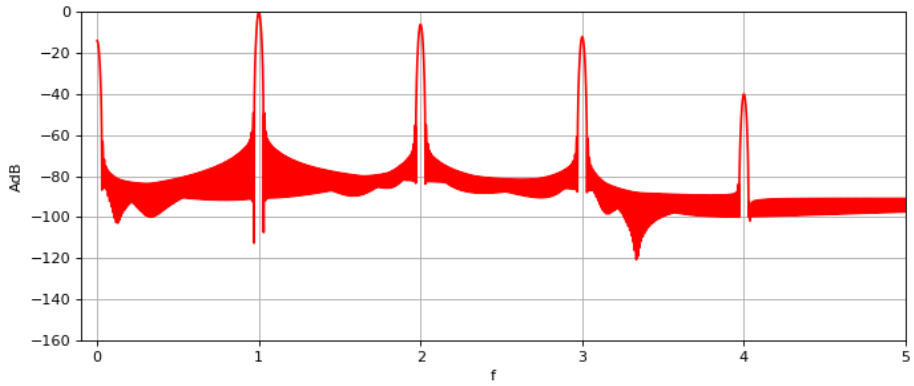
La fonction suivante calcule le spectre d'un signal échantillonné en lui appliquant une fenêtre de troncature et en le complétant par des zéros :

```
def calculerSpectre(echantillons,fe,fenetre,nz=2,db=False):
    N = echantillons.size
    zeros=np.zeros(nz*N)
    echantillons = np.concatenate((zeros,echantillons*signal.get_window(fenetre,N),ze
    spectre = np.absolute(fft(echantillons))
    spectre = spectre / spectre.max()
    NN = len(echantillons)
    if db:
        spectre = 20*np.log10(spectre)
    freq = np.arange(NN)*fe/NN
    return [freq,spectre]
```

Exemple d'utilisation :

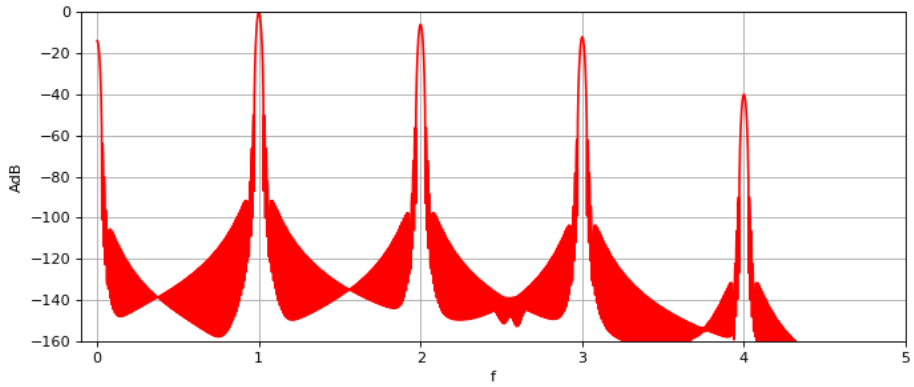
```
[freq,spectre] = calculerSpectre(echantillons,fe,"hamming",nz=2,db=True)
```

```
figure(figsize=(10,4))
plot(freq,spectre,'r')
xlabel('f')
ylabel('AdB')
axis([-0.1,fe/2,-160,0])
grid()
```



Voici un calcul de spectre avec une fenêtre de Hann :

```
[freq,spectre] = calculerSpectre(echantillons,fe,"hann",nz=2,db=True)
figure(figsize=(10,4))
plot(freq,spectre,'r')
xlabel('f')
ylabel('AdB')
axis([-0.1,fe/2,-160,0])
grid()
```



Les lobes secondaires ont une amplitude plus rapidement décroissante que ceux de la fenêtre de Hamming. Avec des signaux réels, il seront probablement complètement masqués par le bruit.