Transformée de Fourier discrète : série de Fourier

1. Coefficients de Fourier et Transformée de Fourier discrète

Ce document introduit la définition de la transformée de Fourier discrète (TFD) comme moyen de calculer les coefficients de Fourier d'une fonction périodique.

Soit $u(t)$ une fonction de période T développable en série de Fourier :

$$u(t) = \sum_{n=-\infty}^{\infty} c_n \exp\left(j2\pi\frac{n}{T}t\right)$$

Les coefficients de Fourier sont, pour $n \in Z$:

$$c_n = \frac{1}{T}\int_0^T u(t)\exp\left(-j2\pi\frac{n}{T}t\right)\,dt$$

On considère un échantillonnage de $u(t)$ sur une période, de N points, avec $0 \leq k \leq N-1$:

$$t_k = k\frac{T}{N}$$
$$u_k = u(t_k)$$

Une approximation des coefficients de Fourier peut être obtenue par la méthode des rectangles :

$$c_n \simeq \frac{1}{T}\sum_{k=0}^{N-1} u_k \exp\left(-j2\pi n\frac{k}{N}\right)\frac{T}{N}$$

La formule obtenue définit une suite de période N. Il suffit donc de calculer, pour $0 \leq n \leq N-1$:

$$S_n = \frac{1}{N}\sum_{k=0}^{N-1} u_k \exp\left(-j2\pi n\frac{k}{N}\right)$$

L'application qui associe aux N nombres u_k les N valeurs S_n est la transformée de Fourier discrète (TFD) ([1]). S_n est une approximation du coefficient de Fourier c_n, correspondant à l'harmonique de fréquence :

$$f_n = \frac{n}{T}$$

2. Applications

2.a. Polynôme trigonométrique

La transformée de Fourier discrète est calculée avec la méthode de transformée de Fourier rapide (Fast Fourier Transform, FFT). On utilise pour cela la fonction `fft` du module `numpy.fft`. La TFD calculée par cette fonction ne comporte par le facteur $1/N$.

```
from numpy.fft import fft
```

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

On définit un polynôme trigonométrique de période T :

```
T = 1.0
w0 = 2*math.pi/T
def signal(t):
    return 1.0+2.0*np.cos(w0*t)+0.5*np.cos(2*w0*t)\
    +1.0*np.sin(3*w0*t)+0.2*np.cos(10*w0*t)
```

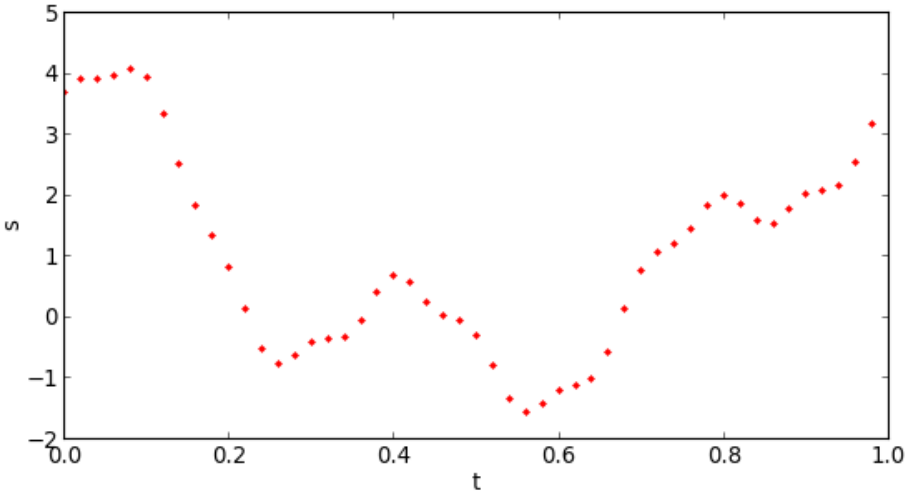
Remarquer que les fonctions trigonométriques sont celles de `numpy` et non de `math`. Cela permet de définir une `ufunc`, qui peut s'appliquer aux tableaux numpy.

On échantillonne le temps sur une période, puis la fonction :

```
fe = 50.0
te = 1/fe
temps = np.arange(start=0.0,stop=T,step=te)
echantillons = signal(temps)
```

Tracé des échantillons :

```
figure(figsize=(8,4))
plot(temps,echantillons,'r.')
xlabel('t')
ylabel('s')
```

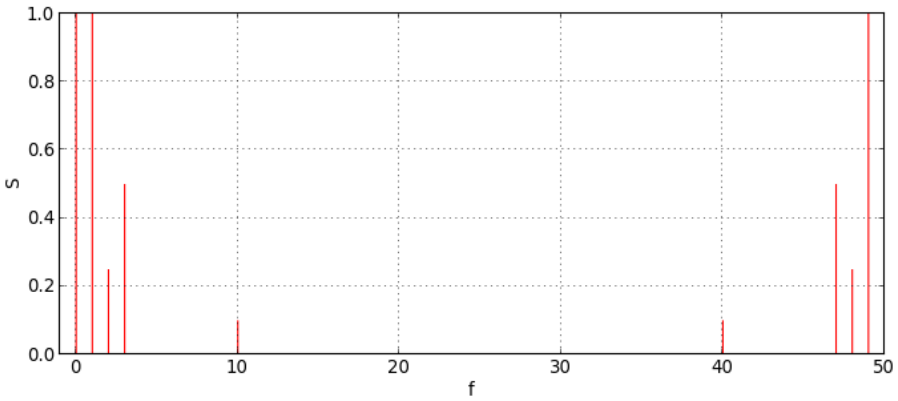


Calcul de la transformée de Fourier discrète et des fréquences

```
N = temps.size
tfd = fft(echantillons)/N
freq = np.zeros(N)
for k in range(N):
    freq[k] = k*1.0/T
```

La représentation graphique du module de la tfd en fonction de la fréquence :

```
spectre = np.absolute(tfd)
figure(figsize=(10,4))
vlines(freq,[0],spectre,'r')
xlabel('f')
ylabel('S')
axis([-1,fe,0,spectre.max()])
grid()
```



On reconnait sur la première moitié du spectre les coefficients de Fourier $c0 = 1$, $c1 = 2/2$, $c2 = 0.5/2$, $c3 = -j/2$, $c10 = 0.2/2$

On remarque la symétrie suivante :

$$|S_n| = |S_{N-n}|$$

En effet, si la fonction $u(t)$ est à valeurs réelles, on a :

$$S_{N-n} = S_n^*$$

La deuxième partie du spectre (appelée aussi l'image du spectre) correspond donc aux coefficients de Fourier d'indices négatifs :

$$c_{-n} \simeq S_{N-n}$$

Il y a par ailleurs la périodicité de la suite S_n :

$$S_n = S_{N+n}$$

On obtient donc un spectre dont la périodicité est la fréquence d'échantillonnage

$$f_e = \frac{N}{T}$$

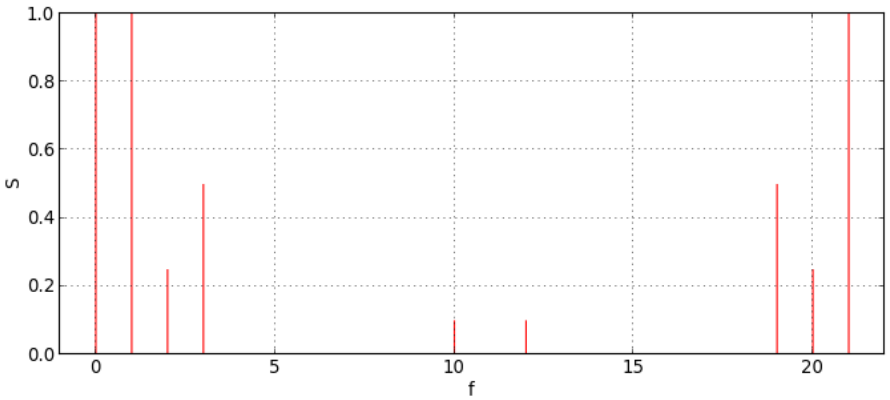
Dans le cas d'un polynome trigonométrique, il existe une plus grande fréquence dans le spectre (ici 10). Si la moitié de la fréquence d'échantillonnage est strictement supérieure à cette plus grande fréquence (critère de Shannon), la TFD donne des valeurs exactes des coefficients de Fourier (aux erreurs d'arrondis près).

Pour simplifier la suite, on encapsule l'ensemble du traitement dans une seule fonction qui prend en argument la fonction et la fréquence d'échantillonnage. Contrairement au cas précédent, la fonction fournie n'est pas nécessairement une `ufunc` ; il faut donc créer le tableaux des échantillons et le remplir dans une boucle.

```
def tracerSpectre(fonction,fe):
    t = np.arange(start=0.0,stop=1.0,step=1.0/fe)
    echantillons = t.copy()
    for k in range(t.size):
        echantillons[k] = fonction(t[k])
    N = echantillons.size
    tfd = fft(echantillons)/N
    spectre = np.absolute(tfd)
    freq = np.arange(N)
    figure(figsize=(10,4))
    vlines(freq,[0],spectre,'r')
    xlabel('f')
    ylabel('S')
    axis([-1,fe,0,spectre.max()])
    grid()
    return tfd
```

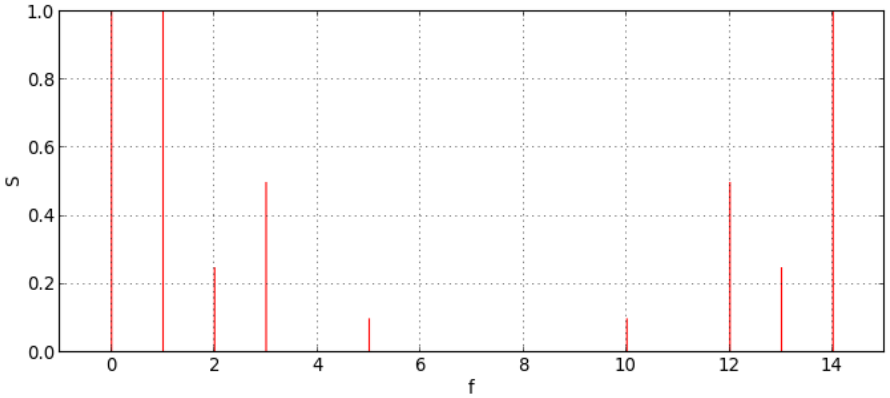
Voyons le spectre avec la plus petite fréquence d'échantillonnage qui respecte le critère de Shannon :

```
tracerSpectre(signal,22)
```



On obtient bien les coefficients de Fourier. Voyons à présent le cas où la moitié de la fréquence d'échantillonnage est inférieure à la plus grande fréquence du polynome :

```
tracerSpectre(signal,15)
```

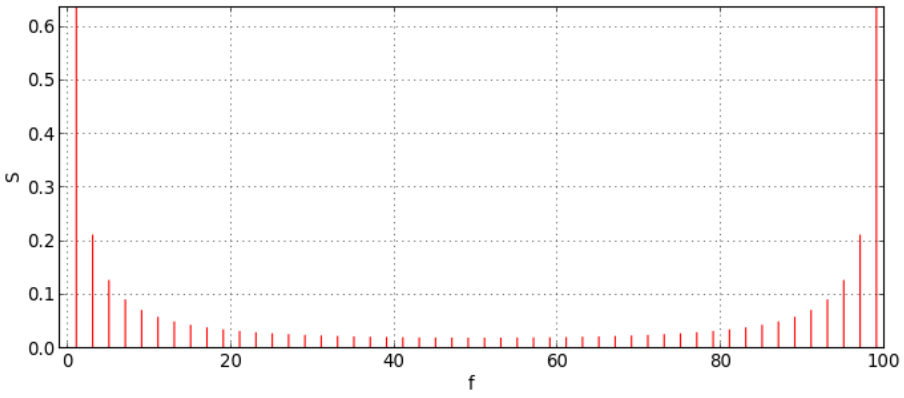


On remarque l'apparition d'une harmonique de rang $n = 5$ qui n'existe pas dans la fonction d'origine, position qui correspond en fait à l'image de l'harmonique de rang 10. Ce phénomène est appelé le repliement de bande. Il se produit lorsqu'une partie du spectre se superpose à son image.

2.b. Fonction créneau

La série de Fourier de la fonction créneau n'a pas de fréquence maximale. Le repliement de bande est donc inévitable, d'autant plus que la décroissance des coefficients de Fourier en 1/n est relativement lente.

```
def signal(t):
    if t < 0.5:
        return 1.0
    else:
        return -1.0
tfd = tracerSpectre(signal,100)
```



La valeur importante des harmoniques au voisinage de la moitié de la fréquence d'échantillonnage laisse deviner un repliement de bande non négligeable. En théorie, les coefficients de Fourier impairs de la fonction créneau vérifient :

$$\frac{|c_1|}{|c_n|} = n$$

Le tableau suivant permet de déterminer la précision du calcul par la TFD :

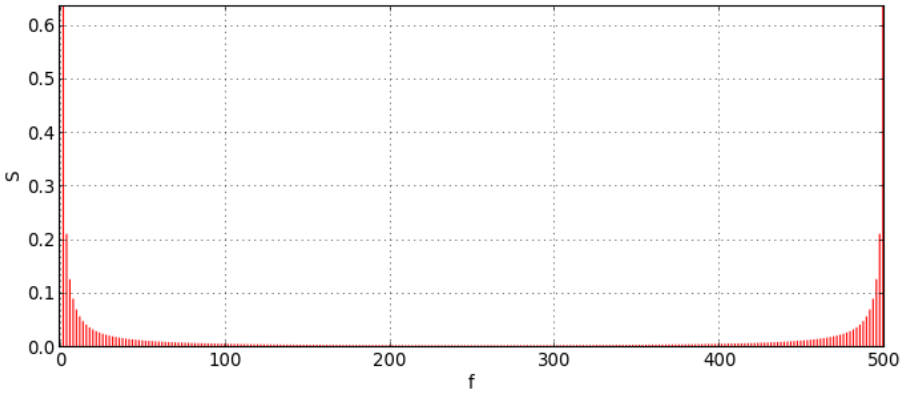
```
pmax = 20
rapport = np.zeros(pmax)
for p in range(pmax):
    rapport[p] = abs(tfd[1])/abs(tfd[2*p+1])/(2*p+1)


print(rapport)
--> array([ 1.        ,  0.99868449,  0.99605657,  0.99212248,  0.98689152,
        0.98037607,  0.97259151,  0.96355625,  0.9532916 ,  0.94182174,
        0.92917367,  0.91537711,  0.90046443,  0.88447055,  0.86743286,
        0.84939108,  0.83038718,  0.81046527,  0.78967143,  0.76805364])
```

Le repliement de bande a une effet notable même sur les premières harmoniques. L'erreur dépasse 5 pour cent à partir du rang 17.

Pour résoudre ce problème, on peut augmenter la fréquence d'échantillonnage :

```
tfd = tracerSpectre(signal,500)
pmax = 20
rapport = np.zeros(pmax)
for p in range(pmax):
    rapport[p] = abs(tfd[1])/abs(tfd[2*p+1])/(2*p+1)
```




```
print(rapport)
--> array([ 1.        ,  0.99994736,  0.99984209,  0.9996842 ,  0.9994737 ,
        0.99921062,  0.99889497,  0.99852679,  0.99810611,  0.99763297,
        0.99710742,  0.99652951,  0.99589929,  0.99521682,  0.99448216,
        0.9936954 ,  0.99285659,  0.99196582,  0.99102317,  0.99002874])
```

On arrive ainsi à maintenir une erreur inférieure à 1 pour cent sur les harmoniques jusqu'au rang 40, mais au prix d'un accroissement considérable du nombre de points. En pratique il n'est pas toujours possible d'augmenter la fréquence d'échantillonnage ainsi, c'est pourquoi les signaux réels sont traités par un filtre analogique passe-bas avant d'être échantillonnés.

Références

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