Transformée de Fourier rapide

1. Transformée de Fourier discrète

La transformée de Fourier discrète est une transformation mathématique permettant d'obtenir le spectre de fréquence d'un signal échantillonné.

On dispose de N échantillons d'un signal, que l'on note u_k avec $k = 0, ..., N-1$. Ces échantillons ont été prélevés avec une période d'échantillonnage T_e. La durée du signal échantillonné est donc $T = NT_e$.

Pour obtenir le spectre du signal analogique, on cherche à calculer les coefficients de Fourier complexes d'une fonction périodique de période T :

$$C_n = \frac{2}{T} \int_0^T u(t) \exp\left(-jn\frac{2\pi}{T}t\right) dt \tag{1}$$

La méthode des rectangles permet d'obtenir une approximation de cette intégrale :

$$C_n \approx \frac{2}{NT_e} \sum_{k=0}^{N-1} u_k \exp\left(-jn\frac{2\pi}{NT_e}kT_e\right) T_e = \frac{2}{N} \sum_{k=0}^{N-1} u_k \exp\left(-jn\frac{2\pi}{N}k\right) \tag{2}$$

On définit la transformée de Fourier discrète (TFD) par :

$$F_n = \sum_{k=0}^{N-1} u_k \exp\left(-j2\pi\frac{n}{N}k\right) = \sum_{k=0}^{N-1} u_k S_{nk} \tag{3}$$

Le nombre complexe F_n, multiplié par $2/N$, est une approximation du coefficient de Fourier complexe C_n de la fonction périodique de période T. La TFD définit le spectre du signal échantillonné. Le nombre complexe F_n, multiplié par $2/N$, est la composante de fréquence $n/(NT_e)$ du spectre du signal échantillonné.

La TFD transforme N nombres, que nous supposerons réels, en N nombres complexes. Introduisons une notation faisant apparaître le nombre N :

$$F_n^{(N)} = \sum_{k=0}^{N-1} u_k S_{nk}^{(N)} \tag{4}$$

Le nombre N est placé en exposant mais il ne s'agit pas d'un exposant.

La TFD est périodique, de période N :

$$F_{n+N}^{(N)} = F_n^{(N)} \tag{5}$$

En terme de fréquences, cela implique la périodicité du spectre avec une période $\frac{N}{NT_e} = \frac{1}{T_e}$, qui est la fréquence d'échantillonnage.

Les échantillons du signal étant des nombres réels, nous avons :

$$F_{N-n}^{(N)} = [F_n^{(N)}]^* \tag{6}$$

où $[z]^*$ désigne le complexe conjugé de z. La représentation du spectre consiste à tracer le module de $F_n^{(N)}$ en ordonnée et la fréquence $n/(NT_e)$ en abscisse. La propriété :

$$|F_n^{(N)}| = |F_{N-n}^{(N)}| \tag{7}$$

implique que la composante du spectre de fréquence $n/(NT_e)$ est égale à celle de fréquence $\frac{N-n}{NT_e} = \frac{1}{T_e} - \frac{n}{NT_e}$.

2. Transformée de Fourier rapide

La méthode basique de calcul de la TFD consiste à calculer les N nombres $F_n^{(N)}$ ($n = 0, ..., N-1$) tels qu'ils sont écrits dans la TFD. Chacun de ces nombres est la somme de N termes, chacun étant le produit de u_k par une valeur complexe dont l'évaluation nécessite le calcul d'un cosinus et d'un sinus. La complexité temporelle de cet algorithme est donc $\Theta(N^2)$. Le nombre d'échantillons est de l'ordre de plusieurs milliers à plusieurs millions et cette méthode conduit à des calculs extrêmement longs.

L'algorithme de transformée de Fourier rapide (FFT pour Fast Fourier Transform) a été inventé en 1965 par Cooley et Tukey. Il s'agit d'un algorithme de type *diviser pour régner*, qui fonctionne si le nombre d'échantillons est une puissance de 2, soit $N = 2^p$. En pratique, cette condition n'est pas contraignante car il suffit d'ajouter si nécessaire des zéros pour allonger la liste des échantillons (l'ajout de zéro permet d'ailleurs d'améliorer la précision du spectre).

L'algorithme FFT repose sur le regroupement des termes de rangs pairs d'une part, des termes de rangs impairs d'autre part :

$$F_n^{(N)} = \sum_{k=0}^{\frac{N}{2}-1} u_{2k} \exp\left(-j2\pi\frac{n}{N}2k\right) + \exp\left(-j2\pi\frac{n}{N}\right) \sum_{k=0}^{\frac{N}{2}-1} u_{2k+1} \exp\left(-j2\pi\frac{n}{N}2k\right) \tag{8}$$

$$= \sum_{k=0}^{\frac{N}{2}-1} u_{2k} S_{nk}^{(N/2)} + \exp\left(-j2\pi\frac{n}{N}\right) \sum_{k=0}^{\frac{N}{2}-1} u_{2k+1} S_{nk}^{(N/2)} \tag{9}$$

Le premier terme est la TFD des échantillons de rangs pairs (TFD de taille $N/2$). Le second terme est la TFD des échantillons de rangs impairs, multipliée par le coefficient suivant :

$$W_n^{(N)} = \exp\left(-j2\pi\frac{n}{N}\right) \tag{10}$$

Notons $P_n^{(N/2)}$ la TFD des termes de rangs pairs et $I_n^{(N/2)}$ la TFD des termes de rangs impairs. Chacune de ces TFD contient $N/2$ nombres :

$$P_n^{(N/2)} = \sum_{k=0}^{\frac{N}{2}-1} u_{2k} S_{nk}^{(N/2)} \text{ pour } n = 0, \ldots \frac{N}{2} - 1 \tag{11}$$

$$I_n^{(N/2)} = \sum_{k=0}^{\frac{N}{2}-1} u_{2k+1} S_{nk}^{(N/2)} \text{ pour } n = 0, \ldots \frac{N}{2} - 1 \tag{12}$$

Ces deux TFD sont en fait définis pour $n = 0, \ldots, N-1$ grace à la périodicité de la TFD :

$$P_{N/2+n}^{(N/2)} = P_n^{(N/2)} \tag{13}$$

$$I_{N/2+n}^{(N/2)} = I_n^{(N/2)} \tag{14}$$

qui fait que, pour $n = N/2, \ldots, N-1$ on a :

$$P_n^{(N/2)} = P_{n-N/2}^{(N/2)} \tag{15}$$

$$I_n^{(N/2)} = I_{n-N/2}^{(N/2)} \tag{16}$$

La TFD des échantillons de rangs pairs (respectivement de rangs impairs) est calculée récursivement de la même manière puisque $N/2$ est divisible par 2. Lorsque le calcul de ces deux TFD est terminé, les N valeurs de la TFD sont obtenues par :

$$F_n^{(N)} = P_n^{(N/2)} + W_n^{(N)} I_n^{(N/2)} \text{ pour } n = 0, \ldots, N-1 \tag{17}$$

La récursion s'arrête lorsqu'on arrive à une TFD de taille 1. La TFD d'un seul nombre est ce nombre.

Voyons la complexité temporelle de cet algorithme. Il y a $p = ln(N)/ln(2)$ niveaux de récursion. À chaque niveau, la relation (17) est appliquée N fois. L'ordre de grandeur de la complexité est donc $\Theta(Nln(N))$, ce qui en fait un algorithme beaucoup plus efficace que la méthode basique. L'algorithme FFT a permis le développement de l'analyse spectrale numérique, à une époque où les ordinateurs étaient beaucoup moins rapides qu'aujourd'hui.

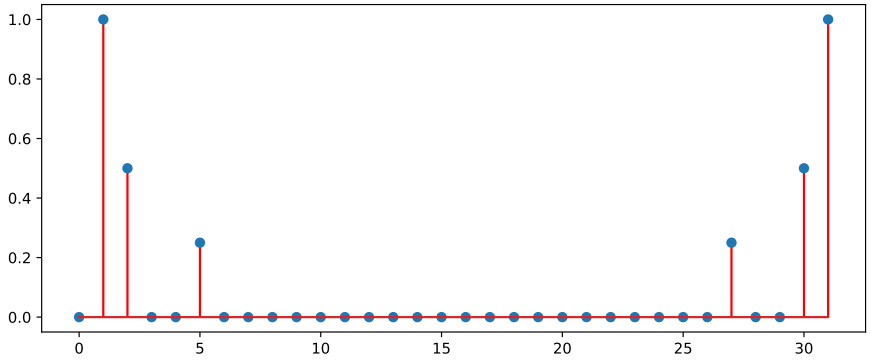
3. Implémentation récursive

L'implémentation récursive de l'algorithme FFT est immédiate :

```
def fft(liste):
    # liste : numpy.ndarray
    N = len(liste)
    if N==1:
        return liste
    pair = liste[0::2]
    impair = liste[1::2]
    tfd_pair = fft(pair)
    tfd_impair = fft(impair)
    tfd = numpy.zeros(N,dtype=numpy.complex64)
    W = numpy.exp(-1j*2*numpy.pi/N)
    N2 = N//2
    for n in range(N2):
        tfd[n] = tfd_pair[n]+tfd_impair[n]*W**n
    for n in range(N2,N):
        tfd[n] = tfd_pair[n-N2]+tfd_impair[n-N2]*W**n
    return tfd
```

On teste la fonction `fft` avec un signal comportant 3 harmoniques, échantillonné sur une période. Si la fréquence d'échantillonnage vérifie la condition de Nyquist-Shannon pour le dernier harmonique, la TFD donne exactement les coefficients de Fourier (aux erreurs d'arrondis près).

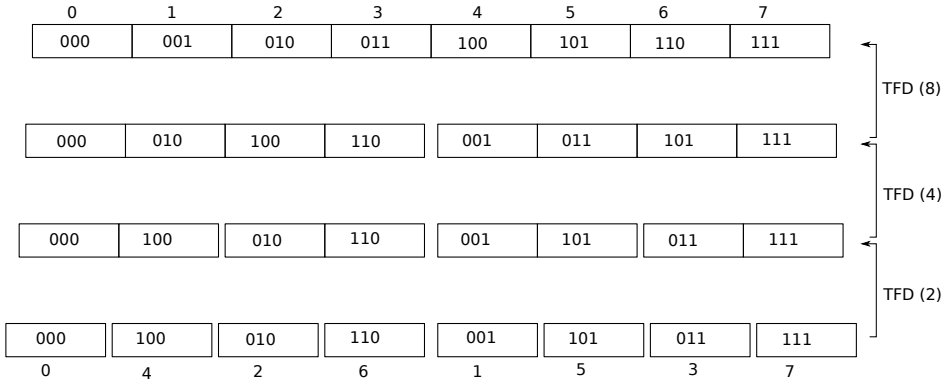
```
import numpy
from matplotlib.pyplot import *

p = 5
N = 2**p
u = numpy.zeros(N,dtype=numpy.complex64)
k = numpy.arange(N)
u = numpy.sin(2*numpy.pi*k/N)+\
    0.5*numpy.sin(4*numpy.pi*k/N)+0.25*numpy.cos(10*numpy.pi*k/N)
tfd = fft(u)

from matplotlib.pyplot import *
spectre = numpy.absolute(tfd)*2/N
figure(figsize=(10,4))
stem(k,spectre,'r')
```



4. Implémentation itérative

Le code précédent n'est pas optimal car il nécessite, à chaque appel de la fonction, la création et le remplissage de deux listes, puis la création d'une nouvelle liste pour contenir la TFD. On préfère en général une implémentation itérative qui évite ces opérations de recopie en mémoire.

Une implémentation itérative consiste à commencer par calculer toutes les TFD à 2 éléments, puis toutes les TFD à 4 éléments, et ainsi de suite jusqu'à la TFD à N éléments. Il faut pour cela déterminer dans quel ordre les échantillons (u_k) doivent être mis pour calculer les TFD à 2 éléments. Voyons le cas particulier $p = 3$. Sur la figure suivante, la liste initiale est représentée en haut, avec les indices en base 10 et en base 2.



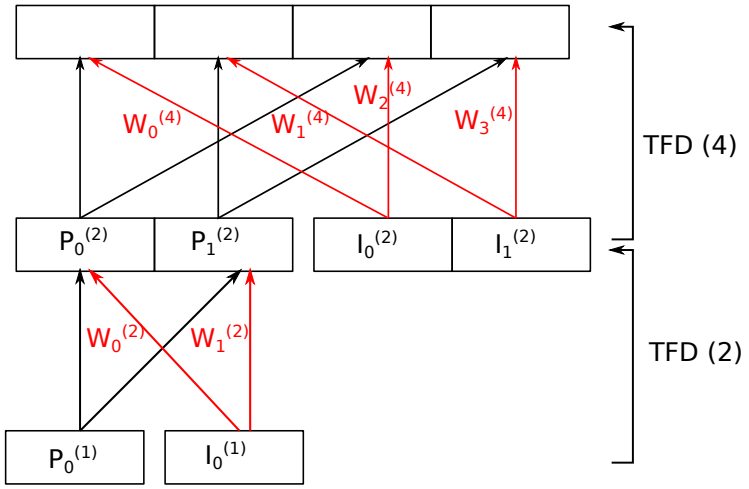
La première division en une liste d'éléments de rangs pairs et une liste d'éléments de rangs impairs se fait en fonction du bit de poids faible de l'indice initial (c.a.d. le bit situé à droite). La seconde division se fait en fonction du deuxième bit et la dernière division se fait en fonction du troisième bit (toujours de l'indice initial). Pour effectuer la FFT de manière itérative, il faut partir de la dernière ligne, appliquer tout d'abord les 4 TFD à 2 éléments, puis les 2 TFD à 4 éléments et enfin la TFD à 8 éléments. La liste de départ de l'algorithme (en bas sur la figure) contient bien sûr les N échantillons mais dans un ordre différent de l'ordre chronologique. Par exemple, l'échantillon $k = 4$ se trouve à l'indice $j = 1$. On constate que le rangement des échantillons dans la liste de départ se fait en inversant l'ordre des bits de l'indice. Pour savoir où placer l'échantillon d'indice k, il suffit d'inverser l'ordre des bits dans la représentation binaire de k, ce qui donne l'indice j où il faut le placer.

Une fois les échantillons placés dans le bon ordre, on peut commencer la boucle des p niveaux de TFD. Le premier niveau consiste à calculer les TFD à 2 éléments, le deuxième les TFD à 4 éléments et ainsi de suite jusqu'au p-ième niveau, qui calcule la TFD à N éléments. Les TFD de niveau q (allant de 1 à p) sont des TFD à 2^q éléments. Il y a 2^{p-q} TFD pour le niveau q.

Voyons comment se fait le premier niveau. Une TFD à 2 éléments est définie par la relation :

$$F_n^{(2)} = P_n^{(1)} + W_n^{(2)} I_n^{(1)} \text{ pour } n = 0, 1 \tag{18}$$

avec $P_1^{(1)} = P_0^{(1)}$ et $I_1^{(1)} = I_0^{(1)}$. Les opérations à effectuer pour une TFD à 2 éléments sont schématisées sur la figure suivante :

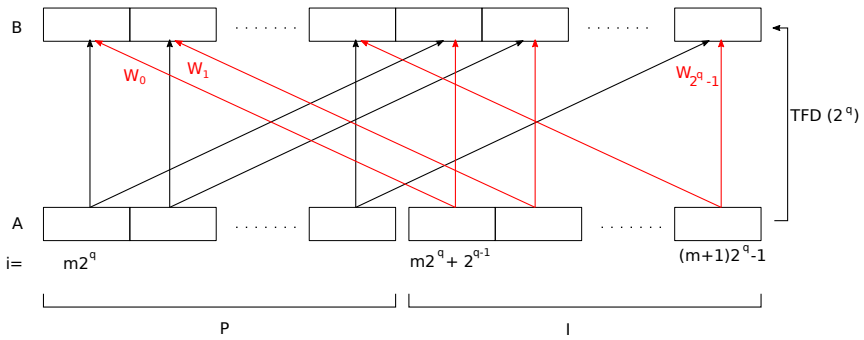


Une TFD à 4 éléments est définie par la relation :

$$F_n^{(4)} = P_n^{(2)} + W_n^{(4)} I_n^{(2)} \text{ pour } n = 0, 1, 2, 3 \tag{19}$$

Les opérations à effectuer sont schématisées ci-dessus.

Supposons que les TFD de niveau $q-1$ soient stockées dans un tableau à N éléments. Nous avons besoin d'un second tableau à N éléments pour générer les TFD de niveau q. Au total, l'algorithme peut donc fonctionner avec seulement deux tableaux à N éléments (on procédera à un échange des références à chaque niveau). Soit A le tableau contenant les TFD de niveau $q-1$ et B celui devant recevoir les TFD de niveau q. Il y a 2^{p-q} groupes de 2^q éléments à calculer. Considérons le groupe numéro m, dont les indices vont de $i = m2^q$ à $i = (m+1)2^q - 1$. Il se calcule avec les éléments de mêmes indices du tableau A. Pour ce calcul, il faut écrire une boucle sur la première moitié des indices, soit i variant de $m2^q$ à $m2^q + 2^{q-1} - 1$, et une boucle sur la seconde moitié des indices, soit i variant de $m2^q + 2^{q-1}$ à $(m+1)2^q - 1$. Les opérations à effectuer sont schématisées sur la figure suivante.

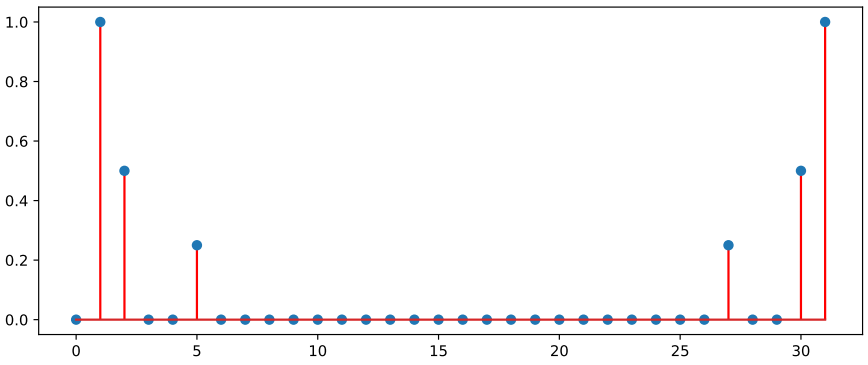


Pour effectuer l'inversion des bits des indices, nous avons besoin d'une fonction qui convertit un nombre en représentation binaire à p bits, effectue l'inversion des bits puis renvoie la valeur en base 10 du résultat. Voici cette fonction :

```
def inversion_bits(i,p): # p = nombre de bits
    c = "{0:0%db}"%p
    binaire = c.format(i)
    inv = binaire[::-1]
    return int(inv,2)
```

Voici la fonction `fft(u,p)`, qui effectue le calcul itératif de la TFD. Les tableaux `u`, `A` et `B` sont des tableaux de numpy (`ndarray`) d'éléments de type `numpy.complex64`.

```
import numpy

def fft(u,p):
    N=len(u)
    if N!=2**p:
        print("Erreur de taille")
        return
    A = numpy.zeros(N,dtype=numpy.complex64)
    B = numpy.zeros(N,dtype=numpy.complex64)
    for k in range(N):
        j = inversion_bits(k,p)
        A[j] = u[k]
    for q in range(1,p+1): # FFT à 2**q éléments
        taille = 2**q
        taille_precedente = 2**(q-1)
        nombre_tfd = 2**(p-q) # nombre de TFD à calculer
        for m in range(nombre_tfd):
            position = m*taille
            phi = -1j*2*numpy.pi/taille
            for i in range(taille_precedente):
                W = numpy.exp(phi*i)
                B[position+i] = A[position+i] + W*A[position+taille_precedente+i]
            for i in range(taille_precedente,taille):
                W = numpy.exp(phi*i)
                B[position+i] = A[position+i-taille_precedente] + W*A[position+i]
        (A,B)=(B,A) # échange des références des tableaux
    return A

import numpy
from matplotlib.pyplot import *

p = 5
N = 2**p
u = numpy.zeros(N,dtype=complex)
k = numpy.arange(N)
u = numpy.sin(2*numpy.pi*k/N)\
    +0.5*numpy.sin(4*numpy.pi*k/N)+0.25*numpy.cos(10*numpy.pi*k/N)
tfd = fft(u,p)


spectre = numpy.absolute(tfd)*2/N
figure(figsize=(10,4))
stem(k,spectre,'r')
```


1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30