Transformée de Fourier rapide

1. Introduction

La *transformée de Fourier discrète* (TFD) permet de calculer le spectre en fréquence d'un signal numérique comportant N échantillons. La TFD de N nombres u_k (réels ou complexes) est constituée des N nombres complexes définis par :

$$F_{N,n} = \sum_{k=0}^{N-1} u_k \exp\left(-j2\pi n\frac{k}{N}\right) \quad (0 \leq n \leq N-1) \tag{1}$$

La TFD à N termes est en fait un signal discret de période N car :

$$F_{N,n+N} = F_{N,n} \tag{2}$$

Le calcul direct des N termes comporte le calcul de N sommes à N termes, soit un nombre d'opérations proportionnel à N^2.

L'algorithme de transformée de Fourier rapide (Fast Fourier Transform ou FFT) repose sur la décomposition suivante, consistant à séparer les termes de rang pair et les termes de rang impair de la somme (on suppose que $N = 2^p$) :

$$F_{N,n} = \sum_{k=0}^{\frac{N}{2}-1} u_{2k} \exp\left(-j2\pi n\frac{2k}{N}\right) + \exp\left(-j2\pi\frac{n}{N}\right) \sum_{k=0}^{\frac{N}{2}-1} u_{2k+1} \exp\left(-j2\pi n\frac{2k}{N}\right) \tag{3}$$

Les deux sommes sont des TFD à $N/2$ termes, qui sont calculées récursivement de la même manière.

2. Regroupement récursif pair-impair

Soit une liste de N nombres. On suppose que $N = 2^p$ où p est un entier.

Par exemple pour $p = 3$, on a la liste :

$$[L[0], L[1], L[2], L[3], L[4], L[5], L[6], L[7]] \tag{4}$$

On regroupe les termes d'indice pair et ceux d'indice impair :

$$[L[0], L[2], L[4], L[6]]\ [L[1], L[3], L[5], L[7]] \tag{5}$$

Les deux sous-listes comportant chacune $N/2 = 4$ éléments subissent aussi un regroupement des termes d'indice pair et des termes d'indice impair, ce qui conduit à :

$$[L[0], L[4]]\ [L[2], L[6]]\ [L[1], L[5]]\ [L[3], L[7]] \tag{6}$$

On obtient ainsi une liste comportant les mêmes nombres que la liste initiale, mais dans un ordre différent.

Écrire une fonction `separation(liste,liste_finale)` qui effectue cette séparation pair-impair de manière récursive et remplit `liste_finale` avec les éléments dans l'ordre final.
Définir une liste comportant les $N = 2^p$ nombres $0, 1, 2...N-1$ puis tester la fonction avec cette liste.
Exprimer le temps d'exécution en fonction de N.

3. Représentation binaire et inversion des bits

La représentation binaire (en base 2) d'un nombre entier x codé sur p bits est constituée de p bits $b_0, b_1, ...b_{p-1}$ (valant chacun 0 ou 1) tels que :

$$x = b_0 2^0 + b_1 2^1 + \dots b_{p-1} 2^{p-1} \tag{7}$$

b_0 est le bits de poids faible, b_{p-1} est le bit de poids fort.

On convient de stocker les bits dans une liste, le premier étant le bit de poids faible. Pour obtenir la représentation binaire à p bits d'un nombre entier, on peut procéder de manière récursive. Le bit de poids faible d'un nombre est le reste de sa division entière par 2. Pour enlever le bit de poids faible, il suffit de diviser le nombre par 2. Le reste de la division par 2 du nombre résultant est le deuxième bit b_1. On procède ainsi récursivement jusqu'au bit b_{p-1}.

Écrire une fonction `bits(nombre,liste_bits,p)` qui calcule récursivement les p bits d'un nombre entier strictement inférieur à 2^p.
Écrire une fonction `nombre(liste_bits)` qui calcule un nombre entier à partir de sa représentation binaire.

Il peut être utile d'inverser l'ordre des bits d'un nombre. Par exemple, le nombre à 4 bits (b_0, b_1, b_2, b_3) devient après inversion (b_3, b_2, b_1, b_0).

Écrire une fonction `inversion_bits(nombre,p)` qui effectue l'inversion des bits d'un nombre entier et renvoie le nombre en décimal.

Soit une liste comportant $N = 2^p$ éléments. Effectuons l'inversion des bits des indices de cette liste et rangeons les éléments par ordre croissant de ces indices à bits inversés.

Écrire une fonction `ranger_ordre_bits_inverse(liste,p)` qui range les éléments d'une liste en inversant les bits des indices. La nouvelle liste est renvoyée. Tester cette fonction sur la liste $0, 1, 2, ...2^p - 1$ et comparer au résultat de la fonction `separation(liste,liste_finale)`. Expliquer pourquoi ces deux fonctions conduisent au même résultat.

4. Transformée de Fourier rapide

L'algorithme de transformée de Fourier rapide est un algorithme de type *diviser pour régner* reposant sur la décomposition suivante

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

$$= P_{N/2,n} + \exp\left(-j2\pi \frac{n}{N}\right) I_{N/2,n} \tag{9}$$

$P_{N/2,n}$ regroupe les termes de rang pair et $I_{N/2,n}$ regroupe les termes de rang impair. Pour n variant de 0 à $N/2-1$, $P_{N/2,n}$ est en fait une TFD à N/2 termes. Il en est de même pour $I_{N/2,n}$.

La relation (9) doit être appliquée pour n variant de 0 à $N-1$. Afin d'accéder aux valeurs des TFD à $N/2$ termes pour $n > N/2-1$, on fait appel à la propriété de périodicité :

$$P_{N/2,n+N/2} = P_{N/2,n} \tag{10}$$

$$I_{N/2,n+N/2} = I_{N/2,n} \tag{11}$$

Voyons par exemple comment la TFD à $N = 8$ termes est calculée à partir des deux TFD à 4 termes :

$$F_{8,0} = P_{4,0} + \exp\left(-j2\pi \frac{0}{8}\right) I_{4,0}$$

$$F_{8,1} = P_{4,1} + \exp\left(-j2\pi \frac{1}{8}\right) I_{4,1}$$

$$F_{8,2} = P_{4,2} + \exp\left(-j2\pi \frac{2}{8}\right) I_{4,2}$$

$$F_{8,3} = P_{4,3} + \exp\left(-j2\pi \frac{3}{8}\right) I_{4,3}$$

$$F_{8,4} = P_{4,0} + \exp\left(-j2\pi \frac{4}{8}\right) I_{4,0}$$

$$F_{8,5} = P_{4,1} + \exp\left(-j2\pi \frac{5}{8}\right) I_{4,1}$$

$$F_{8,6} = P_{4,2} + \exp\left(-j2\pi \frac{6}{8}\right) I_{4,2}$$

$$F_{8,7} = P_{4,3} + \exp\left(-j2\pi \frac{7}{8}\right) I_{4,3}$$

Les N nombres $F_{N,n}$ sont donc calculés à partir des deux TFD à $N/2$ termes. Si N est une puissance de 2, soit $N = 2^p$, la procédure est répétée récursivement jusqu'aux TFD à $N = 1$ terme. La TFD à un terme est simplement l'identité :

$$F_{1,0} = u_0 \tag{12}$$

Autrement dit, la TFD d'un seul nombre est ce nombre lui même.

> Écrire une fonction `fft(liste)` qui fait ce calcul par des appels récursifs et renvoie la liste contenant la TFD.

Pour tester la fonction `fft`, utiliser le script suivant, qui consiste à calculer la TFD d'une fonction périodique échantillonnée exactement sur une période :

```
import numpy
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.p
tfd = fft(u)

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

show()
```

On doit retrouver dans le spectre les amplitudes des harmoniques définies dans le signal.

> Exprimer le temps d'exécution en fonction de N. Comparer à un calcul direct des sommes.

5. Solution

Voici la fonction de regroupement récursif pair-impair :

```
def separation(liste,liste_finale):
    n = len(liste)
    pair = []
    impair = []
    for k in range(0,n,2):
        pair.append(liste[k])
    for k in range(1,n,2):
        impair.append(liste[k])
    if len(pair)>2:
        separation(pair,liste_finale)
    else:
```

```
        liste_finale.append(pair[0])
    if len(impair)>2:
        separation(impair,liste_finale)
    else:
        liste_finale.append(impair[0])


p=4
N=2**p
liste=[]
for i in range(N):
    liste.append(i)
liste_finale = []
separation(liste,liste_finale)


print(liste_finale)
--> [0, 4, 2, 6, 1, 5, 3, 7]
```

Voici la fonction fournissant les bits d'un nombre entier :

```
def bits(nombre,liste_bits,p):
    liste_bits.append(nombre%2)
    if p >1:
        bits(int(nombre/2),liste_bits,p-1)


liste_bits=[]
bits(10,liste_bits,p)


print(liste_bits)
--> [0, 1, 0, 1]
```

Voici la fonction calculant un nombre en décimal à partir de la liste de ses bits :

```
def nombre(liste_bits):
    x = 0
    u=1
    for i in range(len(liste_bits)):
        x += liste_bits[i]*u
        u *= 2
    return x


print(nombre(liste_bits))
--> 10
```

Voici la fonction d'inversion des bits d'un nombre :

```
def inversion_bits(nombre,p):
    liste_bits=[]
    bits(nombre,liste_bits,p)
    x=0
    n=len(liste_bits)
    u = 1
    for i in range(n):
        x += liste_bits[n-1-i]*u
        u *= 2
    return x


print(inversion_bits(6,p))
--> 6
```

Voici la fonction permettant de ranger les éléments d'une liste par ordre croissant des indices à bits inversés :

```
def ranger_ordre_bits_inverse(liste,p):
    liste_inverse = liste.copy()
    for i in range(len(liste)):
        liste_inverse[inversion_bits(i,p)] = liste[i]
    return liste_inverse


print(ranger_ordre_bits_inverse(liste,p))
--> [0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15]
```

Voici l'implémentation récursif de l'algorithme de transformée de Fourier rapide :

```
def fft(liste):
    N = len(liste)
    if N==1:
        return liste
    pair = []
    impair = []
    for k in range(0,N,2):
        pair.append(liste[k])
    for k in range(1,N,2):
        impair.append(liste[k])
    tfd_pair = fft(pair)
    tfd_impair = fft(impair)
    tfd = [0]*N
    W = numpy.exp(-1j*2*numpy.pi/N)
    N2 = int(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 avec un polynôme trigonométrique :

```
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.p
tfd = fft(u)

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