Compte rendu – Projet L3 Outils maths

Transformée de Fourier

Tom BARBIER – Brandon PERE
Année 2022 - 2023

Table des matières

I. INTRODUCTION

Nous allons voir comment et ce à quoi sert la transformé de Fourier, ainsi que la transformé de Fourier rapide. Nous allons expliquer pourquoi nous évoquons les deux. Il faut savoir que la transformé de Fourier est très utile, cependant, elle est difficilement utilisable lorsque nous l'implémentons, pourquoi ? En raison de sa complexité. En effet, la complexité de la transformé de Fourier est de $O(N^2)$, c'est pourquoi il existe, la transformé de Fourier rapide, qui a une complexité simplifiée. En effet, sa complexité est de $O(N * \log_2 N)$ pour des tableaux de tailles de taille 2^N. Dans un premier temps nous construirons la fonction pour des tableaux à 1 dimension. Puis nous généraliserons à deux dimensions en ce reposant sur la 1D.

II. TRANSFORMEE DE FOURIER DIRECT

$$\hat{g}(u) = F\big(g(x)\big) = \sum_{x=0}^{N-1} g(x) e^{\left(\frac{-2*i*\pi*u*x}{N}\right)}$$

u index du tableau

N taille du tableau

g tableau source

ĝ tableau de destination

On remarque donc que pour chaque élément nous devons parcourir l'entièreté du tableau source ce qui explique sa complexité de $O(N^2)$. En l'implémentant en python cela nous donne :

```
def TF1D(Matrice1D):
    N=len(Matrice1D)
    # Creation d'une matrice de la taille de l'image1D
    MatriceRes = np.zeros(N, dtype=complex)
    # On parcours notre matrice initial
    for u in range(N):
        sum = 0
        # On applique la formule
        for x in range(N):
            sum += Matrice1D[x]*cmath.exp((-2j * cmath.pi * u * x) / N)
        # On met la valeur dans la matrice resultat + on arrondi les valeurs
        MatriceRes[u]=round(sum.real, 8)+round(sum.imag, 8)*1j
    return MatriceRes
```

On peut comparer les résultats avec la fft de numpy. (bibliothèque que nous utilisons pour simplifier le code)

```
Matrice en entrée
[1 2 3 4]
--------------------------------------------------
Résultat de notre algorithme pour la TF1D
[10.+0.j -2.+2.j -2.+0.j -2.-2.j]
--------------------------------------------------



--------------------------------------------------
Résultat de l'algorithme de numpy pour la TF1
[10.+0.j -2.+2.j -2.+0.j -2.-2.j]
```

III. Optimisation

Nous pouvons séparer la somme en deux sommes afin d'avoir les éléments pairs d'un coté et impairs de l'autre côté ce qui donne :

$$\hat{g}(u) = F\big(g(x)\big) = \sum_{x=0}^{N-1} g(x)e^{\left(\frac{-2*i*\pi*u*x}{N}\right)}$$

$$\hat{g}(u) = \sum_{x=0}^{\frac{N}{2}-1} g(2x) * e^{-2i*\pi*2x*\left(\frac{u}{N}\right)} + \sum_{x=0}^{\frac{N}{2}-1} g(2x+1) * e^{-2i*\pi*(2x+1)*\left(\frac{u}{N}\right)}$$

$$\hat{g}(u) = \sum_{x=0}^{\frac{N}{2}-1} g(2x) * e^{-2i*\pi x*\left(\frac{2u}{N}\right)} + \sum_{x=0}^{\frac{N}{2}-1} g(2x+1) * e^{-2i*\pi x*\left(\frac{2u}{N}\right)} * e^{-2i*\pi*\left(\frac{u}{N}\right)}$$

$$car\ e^{(a+b)} = e^{a} * e^{b}$$

$$\hat{g}(u) = \sum_{x=0}^{\frac{N}{2}-1} g(2x) * e^{-2i*\pi x*\left(\frac{u}{N/2}\right)} + \sum_{x=0}^{\frac{N}{2}-1} g(2x+1) * e^{-2i*\pi x*\left(\frac{u}{N/2}\right)} * e^{-2i*\pi*\left(\frac{u}{N}\right)}$$

Le terme $e^{-2i*\pi*(\frac{u}{N})}$ ne dépend pas de la variable x de la somme il est donc possible de la sortir de la somme :

$$\hat{g}(u) = \sum_{x=0}^{\frac{N}{2}-1} g(2x) * e^{-2i*\pi x*\left(\frac{u}{\frac{N}{2}}\right)} + e^{-2i*\pi*\left(\frac{u}{N}\right)} * \sum_{x=0}^{\frac{N}{2}-1} g(2x+1) * e^{-2i*\pi x*\left(\frac{u}{N/2}\right)}$$

Nous avons donc en bleu à gauche les éléments pairs et en vert à droite les éléments impairs. On a donc notre tableau, mais celui-ci n'est pas complet car le tableau est de taille $\mathrm{N}/2$.

Nous devons donc calculer la seconde moitié soit :

$$\hat{g}\left(u+\frac{N}{2}\right) = \sum_{x=0}^{\frac{N}{2}-1} g(2x) * e^{-2i*\pi*x*(\frac{u+\frac{N}{2}}{\frac{N}{2}})} + e^{-2i*\pi*(\frac{u+\frac{N}{2}}{N})} * \sum_{x=0}^{\frac{N}{2}-1} g(2x+1) * e^{-2*i*\pi*x*(\frac{u+\frac{N}{2}}{\frac{N}{2}})}$$

Simplifions cette expression :

$$\hat{g}\left(u+\frac{N}{2}\right) = \sum_{x=0}^{\frac{N}{2}-1} g(2x) * e^{-2i*\pi*x*(\frac{2u}{N}+1)} + e^{-2i*\pi*(\frac{1}{2}+\frac{u}{N})} * \sum_{x=0}^{\frac{N}{2}-1} g(2x+1) * e^{-2*i*\pi*x*(\frac{2u}{N}+1)}$$

$$\hat{g}\left(u+\frac{N}{2}\right) = \sum_{x=0}^{\frac{N}{2}-1} g(2x) * e^{-2i*\pi*x*(\frac{2u}{N})} * e^{-2i*\pi*x} + e^{-2i*\pi*(\frac{1}{2}+\frac{u}{N})} * \sum_{x=0}^{\frac{N}{2}-1} g(2x+1) * e^{-2*i*\pi*x*\left(\frac{2u}{N}\right)} * e^{-2i*\pi*x}$$

$$e^{-2i*\pi*x} = e^{0} = 1 \text{ avec } x \in N$$

$$\hat{g}\left(u+\frac{N}{2}\right) = \sum_{x=0}^{\frac{N}{2}-1} g(2x) * e^{-2i*\pi*x*(\frac{2u}{N})} + e^{-2i*\pi*(\frac{u}{N})} * e^{-i\pi} * \sum_{x=0}^{\frac{N}{2}-1} g(2x+1) * e^{-2*i*\pi*x*\left(\frac{2u}{N}\right)}$$

$$e^{-i\pi} = -1$$

$$\hat{g}\left(u+\frac{N}{2}\right) = \sum_{x=0}^{\frac{N}{2}-1} g(2x) * e^{-2i*\pi*x*(\frac{2u}{N})} - e^{-2i*\pi*(\frac{u}{N})} * \sum_{x=0}^{\frac{N}{2}-1} g(2x+1) * e^{-2*i*\pi*x*\left(\frac{2u}{N}\right)}$$

Maintenant que nous avons simplifier le problème nous pouvons passer à l'implémentation. L'avantage de la représentation mathématique que nous avons est qu'elle laisse bien voir un algorithme récursif.

```python
def TF1R(Matrice1D):
    N=len(Matrice1D)

    # Si le tableau n'a qu'une seul valeur on retourne la valeur
    if N<=1:
        return Matrice1D
    else :
        # Récursivité sur les index pair et impair
        pair = TF1R(Matrice1D[0::2])
        impair = TF1R(Matrice1D[1::2])
        # On crée un nouveau tableau
        MatriceRes = np.zeros(N).astype(np.complex64)
        # On reconstitue petit à petit le tableau 1D avec la formule
        for i in range(0, N//2):
            MatriceRes[i] = pair[i]+cmath.exp(-2j*cmath.pi*i/N)*impair[i]
            MatriceRes[i+N//2] = pair[i]-cmath.exp(-2j*cmath.pi*i/N)*impair[i]
        return MatriceRes
```

On peut comparer les résultats avec la fft de numpy.

```
Matrice en entrée
[1 2 3 4]
---------------------------------------------------



---------------------------------------------------
Résultat de notre algorithme pour la TF1R
[10.+0.j -2.+2.j -2.+0.j -2.-2.j]
---------------------------------------------------
Résultat de l'algorithme de numpy pour la TF1
[10.+0.j -2.+2.j -2.+0.j -2.-2.j]
```