MANZANO Lilian

Groupe de TP1

Rapport

Transformée de Fourier discrète

Table des matières

GÉNÉRALITÉS

1) Code :

J'ai décidé de réaliser mon projet en Python, il utilise 4 bibliothèques qui doivent être installés pour que mon code fonctionne, celles-ci sont :

- **'PIL'** dont j'utilise le module '**Image'** qui me permet de charger des images, la transformer en niveau de gris, en mettre les valeurs dans un tableau, et à la fin sauvegarder le nouveau tableau en tant qu'image.
- **'numpy'** qui donne la possibilité d'utiliser des tableaux à deux dimensions qui représenteront la matrice de valeurs (niveaux de gris des pixels de l'image chargée ou simple matrice créée de toute mains).
- **'cmath'** qui permet d'utiliser les nombres complexes ainsi que plusieurs fonctions mathématiques comme l'exponentiel.
- **'time**' que j'ai utilisé pour récupérer le temps d'exécution de mes fonctions de transformée de Fourier, et ainsi pouvoir comparer celui des transformée de Fourier direct et rapide.

2) Utilisation :

Mon code s'utilise uniquement dans une console donc sans interface graphique. Lorsque l'on exécute le programme, il nous est demandé de choisir entre trois choix :

- Importer une image, celle-ci peut être de différents formats comme '.jpeg', '.jpg', ou encore '.png'. L'image peut être en couleur car elle sera transformée en niveau de gris par mon programme.

- Importer un fichier '.npy' (fichier numpy), sachant qu'après l'utilisation des fonctions de transformée de Fourier, mon code sauvegarde dans un fichier avec cette extension le résultat.
  Pouvoir l'importer permet par exemple de tester la transformée de Fourier sur une image, puis de tester la transformée de Fourier inverse sur le fichier '.npy' résultant qui contiendra un tableau avec toutes les valeurs, ceci en ayant gardé les parties complexes des nombres.

- Créer sa propre matrice, ici toutes les informations nécessaires seront indiquées dans la console. Lorsqu'il faut renseigner des valeurs les nombres complexes sont acceptés, à savoir qu'en python ce n'est pas le ***i*** mais le ***j*** qui représente la partie complexe et celui-ci doit coller le chiffre par exemple ***2+3j*** est un nombre complexe accepté.

Remarque : Si l'intention est d'utiliser les fonctions de transformée Fourier rapide, il faudra importer/créer une matrice de hauteur et longueur de taille puissance de 2.

Une fois que la matrice sur laquelle on souhaite faire des traitements a été choisi, un deuxième choix est nécessaire, il faudra choisir entre faire la transformée de Fourier direct, la transformé de Fourier direct inverse, la transformée de Fourier rapide, et la transformée de Fourier rapide inverse. Il n'est pas nécessaire de préciser si l'on veut utiliser la 1D ou 2D, mon programme détecte si la matrice est d'une seule ligne ou d'une seule colonne et applique lui-même la bonne fonction.

Une fois le choix de la transformée de Fourier fait, celle-ci est calculée, et mon programme affiche le temps en secondes que cela a pris, il affiche également la matrice résultante, cependant si celle-ci est trop grande cela peut être dérangeant d'observer les résultats dans la console, un fichier '.txt' est donc créé pour faciliter ceci. De plus 3 autres fichiers sont produits, un second fichier '.txt' pour stocker le module de la matrice résultante, le fichier '.npy' précédemment expliqué, et enfin une image résultat.

TRANSFORMÉE DIRECT 1D

Pour calculer la transformée de Fourier direct une dimension, on nous donne la formule suivante :

$$F\big(g(x)\big) = \sum_{x=0}^{N-1} g(x) \times \exp\left(\frac{-2i\pi ux}{N}\right)$$

Ma fonction **'transformeeDirect1D'** permettant de la calculer utilise donc deux boucles imbriquées, la première permettant de se placer sur la valeur dont on veut le résultat, et la seconde qui permet de calculer la somme.

Cette fonction possède 1 paramètre, qui doit être un tableau de dimension 1, le tableau résultat qui est retourné est de même type.

TRANSFORMÉE DIRECT 2D

1) Formule :

On nous donne la formule :

$$F\big(g(x,y)\big) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} g(x,y) \times \exp\left(-2i\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)\right)$$

Or dans mon programme je n'applique pas juste cette formule, je calcule déjà la transformée de Fourier 1D à l'aide de la formule vu précédemment en traitant chaque ligne indépendamment, puis je calcule sur les résultats obtenus la transformée de Fourier 1D en traitant chaque colonne indépendamment, voyons ci cela reviens au même :

> On commence par reprendre la formule vue au point 2, ceci en renommant ***N*** par ***M*** :
>
> $$F\big(g(x)\big) = \sum_{x=0}^{M-1} g(x) \times \exp\left(\frac{-2i\pi ux}{N}\right)$$
>
> Ici nous avons donc la formule pour calculer la transformée de Fourier sur une ligne. La formule pour calculer la transformée de Fourier sur une colonne est similaire mais avec des noms de variables qui changent pour les différencier ; ***x*** devient ***y***, ***M*** devient ***N***, et ***u*** devient ***v*** :
>
> $$F\big(g(y)\big) = \sum_{y=0}^{N-1} g(y) \times \exp\left(\frac{-2i\pi vy}{N}\right)$$

Ici ***g(y)*** correspond à la valeur de l'élément en position y de la colonne, or si l'on suit mon programme, sa valeur n'est plus celle initial mais celle calculé par la transformée de Fourier 1D sur la ligne, ainsi on peut remplacer ***g(y)*** par la valeur de la transformée de Fourier sur la ligne (avec x qui varie), cependant l'on garde ***g(y)*** car l'on continue de varier en hauteur dans la matrice, et ceci on le regroupe avec ***g(x)*** :

$$F\big(g(x,y)\big) = \sum_{y=0}^{N-1}\sum_{x=0}^{M-1} g(x,y) \times \exp\left(\frac{-2i\pi ux}{M}\right) \times \exp\left(\frac{-2i\pi vy}{N}\right)$$

Par relation fonctionnelle on a ***exp(a) x exp(b) =exp(a+b)***, donc on peut regrouper les 2 exponentielles :

$$F\big(g(x,y)\big) = \sum_{y=0}^{N-1}\sum_{x=0}^{M-1} g(x,y) \times \exp\left(\frac{-2i\pi ux}{M} + \frac{-2i\pi vy}{N}\right)$$

$$F\big(g(x,y)\big) = \sum_{y=0}^{N-1}\sum_{x=0}^{M-1} g(x,y) \times \exp\left(-2i\pi \times \left(\frac{ux}{M} + \frac{vy}{N}\right)\right)$$

Or on peut inverser les sommes car les 2 variables variant en fonctions de celles-ci sont imbriquées dans les deux, et on obtient donc la même formule que donnée dans le cours :

$$F\big(g(x,y)\big) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} g(x,y) \times \exp\left(-2i\pi \times \left(\frac{ux}{M} + \frac{vx}{N}\right)\right)$$

2) Implémentation :

La méthode est donc validée puisque cela revient à la même formule, voyons maintenant comment je l'implémente dans ma fonction **'transformeeDirect2D'**.

Cette fonction prend en paramètre un tableau 2D représentant la matrice dont l'on veut calculer la transformée de Fourier, et retourne la matrice résultat.

Tout d'abord cette fonction calcule la transformée de Fourier 1D de chaque ligne grâce à une boucle et la fonction précédemment expliqué **'transformeeDirect1D'**.

Ensuite il me suffit de refaire la même chose mais en calculant la transformée sur la colonne. Pour cela j'utilise la méthode **'transpose'** de la bibliothèque *numpy*, cette méthode permet de transformée une liste de valeur d'une dimension à l'autre dans la matrice, plus clairement elle me permet de transformer la colonne en ligne pour pouvoir utiliser ma fonction, puis de la remettre en colonne dans ma matrice résultat.

TRANSFORMÉE DIRECT 1D INVERSE

En ce qui concerne la formule de la transformée de Fourier direct inverse une dimension, elle est donnée ainsi dans le cours :

$$F'\big(g(x)\big) = \sum_{x\prime=0}^{N-1} g(x') \times \exp\left(\frac{2i\pi ux}{N}\right)$$

Dans mon programme c'est la fonction **'transformeeDirect1Dinverse'** qui permet de la calculer, elle est structurée comme ma fonction **'transformeeDirect1D'** vue précédemment avec comme seul changement la disparition du signe moins.

Remarque : Cette formule ne permet pas de retomber exactement sur les valeurs initiales, pour cela il faudrait rajouter un coefficient devant le signe sigma, or dans mon programme je recadre après chaque opération entre 0 et 255, cela n'a donc pas d'importance.

TRANSFORMÉE DIRECT 2D INVERSE

La fonction permettant de calculer la transformée de Fourier direct inverse 2 dimension est nommé **'transformeeDirect2Dinverse'.**

Dans cette dernière, pour les mêmes raisons que pour la transformée direct 2D, je fais d'abord le calcule en utilisant **'transformeeDirect1Dinverse'** sur toutes les lignes, puis sur toutes les colonnes, ceci toujours en utilisant la méthode de *numpy* permettant la transposé de ligne à colonne et inversement.

TRANSFORMÉE RAPIDE 1D

1) Formule :

Pour calculer la transformée de Fourier rapide, nous allons séparer les valeurs en position pairs et impaires, pour cela repartons de la formule donner :

$$F\big(g(x)\big) = \sum_{x=0}^{N-1} g(x) \times \exp\left(\frac{-2i\pi ux}{N}\right)$$

Commençons par traiter les valeurs en position paire, soit en position 2x', la somme va donc de x' à N/2 puisque que l'on parcourt que la moitié des valeurs :

$$pair\big(g(x)\big) = \sum_{x\prime=0}^{N/2-1} g(2x') \times \exp\left(\frac{-2i\pi u2x'}{N}\right) = \sum_{x\prime=0}^{N/2-1} g(2x') \times \exp\left(\frac{-2i\pi ux'}{N/2}\right)$$

Ensuite traitons les valeurs en positions impair, soit en position 2x'+1 :

$$impair\big(g(x)\big) = \sum_{x\prime=0}^{N/2-1} g(2x'+1) \times \exp\left(\frac{-2i\pi u(2x'+1)}{N}\right)$$

Par relation on a ***exp(a) x exp(b) =exp(a+b)***, donc on peut séparer les 2 exponentielles :

$$impair\big(g(x)\big) = \sum_{x\prime=0}^{N/2-1} g(2x'+1) \times \exp\left(\frac{-2i\pi u2x'}{N}\right) \times \exp\left(\frac{-2i\pi u}{N}\right)$$

Ensuite n'ayant pas de variable dans le second exponentiel on peut le mettre en coefficient de la somme :

$$\begin{aligned} impair\big(g(x)\big) &= \exp\left(\frac{-2i\pi u}{N}\right) \times \sum_{x'=0}^{N/2-1} g(2x'+1) \times \exp\left(\frac{-2i\pi u2x'}{N}\right) \\ &= \exp\left(\frac{-2i\pi u}{N}\right) \times \sum_{x'=0}^{N/2-1} g(2x'+1) \times \exp\left(\frac{-2i\pi ux'}{N/2}\right) \end{aligned}$$

Ainsi nous avons d'un côté la somme des valeurs en position pair et de l'autre la somme des valeurs en position impairs, on les additionne et renomme x' par x pour simplifier, ce qui donne :

$$F\big(g(x)\big) = \sum_{x=0}^{N/2-1} g(2x) \times \exp\left(\frac{-2i\pi ux}{N/2}\right) + \exp\left(\frac{-2i\pi u}{N}\right) \times \sum_{x=0}^{N/2-1} g(2x+1) \times \exp\left(\frac{-2i\pi ux}{N/2}\right)$$

En excluant le coefficient avant la seconde somme, on voit que les deux sommes sont réciproquement le calcul de la transformée de Fourier pour les listes de nombres pairs et impaires, listes seulement de longueur N/2, ainsi on peut obtenir la première moitié de la transformée de Fourier, mais il nous faut aussi la seconde moitié.

Sachant que F(g(x)) est enfaite égal au calcul de la transformée de Fourier pour l'élément positionné à l'index u, c'est-à-dire $\hat{g}(u)$= F(g(x)), pour pouvoir calculer la seconde moitié il ne faut plus utiliser u comme variable mais u+N/2 :

$$F\big(g(x)\big) = \sum_{x=0}^{N/2-1} g(2x) \times \exp\left(\frac{-2i\pi(u+\frac{N}{2})x}{N/2}\right) + \exp\left(\frac{-2i\pi(u+\frac{N}{2})}{N}\right) \times \sum_{x=0}^{N/2-1} g(2x+1) \times \exp\left(\frac{-2i\pi(u+\frac{N}{2})x}{N/2}\right)$$

On peut ensuite manipuler les contenus des exponentiel pour simplifier les fractions, et pouvoir réutiliser ***exp(a) x exp(b) =exp(a+b)*** dans chacun des exponentielles :

$$F\big(g(x)\big) = \sum_{x=0}^{N/2-1} g(2x) \times \exp\left(-2i\pi x\frac{(u+\frac{N}{2})}{N/2}\right) + \exp\left(-2i\pi\frac{(u+\frac{N}{2})}{N}\right) \times \sum_{x=0}^{N/2-1} g(2x+1) \times \exp\left(-2i\pi x\frac{(u+\frac{N}{2})}{N/2}\right)$$

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

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

$$F(g(x)) = \sum_{x=0}^{\frac{N}{2}-1} g(2x) \times \exp\left(-2i\pi x\left(\frac{u}{\frac{N}{2}}\right)\right) \times \exp(-2i\pi x) + \exp\left(-2i\pi\left(\frac{u}{N}\right)\right) \times \exp(-i\pi) \times \sum_{x=0}^{\frac{N}{2}-1} g(2x+1) \times \exp\left(-2i\pi x\left(\frac{u}{\frac{N}{2}}\right)\right) \times \exp(-2i\pi x)$$

Certains des exponentielles sont simples à calculer :

$$\exp(-i\pi) = \cos(\pi) - i \times \sin(\pi) = -1 - i \times 0 = -1$$

$$\exp(-2i\pi x) = \cos(2\pi x) - i \times \sin(2\pi x)$$

x est un entier donc :

$$\cos(2\pi x) - i \times \sin(2\pi x) = 1 - i \times 0 = 1$$

On peut ainsi les remplacer dans la formule et obtenir la formule finale :

$$F(g(x)) = \sum_{x=0}^{N/2-1} g(2x) \times \exp\left(\frac{-2i\pi ux}{N/2}\right) - \exp\left(\frac{-2i\pi u}{N}\right) \times \sum_{x=0}^{N/2-1} g(2x+1) \times \exp\left(\frac{-2i\pi ux}{N/2}\right)$$

2) Implémentation :

Nous avons donc 2 formules pour calculer chacune des moitiés, ce que l'on peut remarquer dans ces formules c'est que malgré les manipulations nous retrouvons le schéma de la formule initiale dans chaque somme avec comme seule différence que ce n'est plus de taille N mais de taille N/2, ainsi nous devinons que si nous recommençons on aura encore une fois le même schéma mais de taille N/4, et même principe si on sépare de multiples fois.

Ainsi ici appliquer une fonction récursive est possible. Il est logique que dans la longueur exprimer sous la forme N/d, il ne faudra jamais que d>N. Dans le code il faudra donc avant d'appeler la fonction récursivement vérifier que la longueur de la liste soit supérieure à 1.

Voici comment est implémenter ma fonction **'transformeeRapide1D'** :

```
def transformeeRapide1D(m):
    N=len(m)
    if N<=1 :
        return m
    else:
        pair=transformeeRapide1D(m[0::2])
        impair=transformeeRapide1D(m[1::2])
        matriceResultat=np.zeros(N).astype(np.complex64)
        for i in range(0, N//2):
            matriceResultat[i] = pair[i]+cmath.exp(-2j*cmath.pi*i/N)*impair[i]
            matriceResultat[i+N//2] = pair[i]-cmath.exp(-2j*cmath.pi*i/N)*impair[i]
        return matriceResultat
```

Le paramètre m correspond à un tableau 1 dimension. Comme dit précédemment, il faut vérifier que N (la longueur du tableau) soit supérieur à 1, si ce n'est pas le cas on renvoie l'unique valeur du tableau. En revanche si c'est le cas, on peut appeler récursivement la fonction, ceci deux fois, la première avec un tableau contenant uniquement les indices pairs, et la seconde avec un tableau contenant uniquement les indices impairs.

La suite correspond au calcul et à la reconstitution du tableau, pour cela nous parcourons le tableau à l'aide d'une boucle *'for'*, l'indice de cette boucle va de 0 à la moitié de la longueur du tableau, cependant grâce à nos 2 formules trouvées précédemment nous pouvons remplir les 2 moitiés du tableau. Une fois les calculs réalisés, nous pouvons retourner le tableau.

3) Complexité :

Nous avons donc 2 méthodes différentes pour calculer la transformée de Fourier, mais il n'est pas encore avéré que la seconde est plus rapide, pour en être sûr voyons les complexités de ces 2 algorithmes.

Pour calculer ces complexités nous n'allons pas compter chaque opérateur car ils n'ont pas beaucoup d'impact, ce sont les itérations de boucles qui vont nous intéresser :

- Pour la première formule nous avions vu qu'il nous fallait une boucle qui parcourt la liste pour se placer sur la valeur à calculer, et une seconde qui parcourt à chaque fois de nouveau toute la liste pour réaliser la somme. Ceci sur une liste de taille N donc une complexité de N^2 soit $O(N^2)$

- Toujours sur un tableau de taille N, nous avons vu que la seconde formule nous fait faire 2 formule similaire sur un nombre de valeurs divisé par 2, pour noter ça on va noter *f* la formule, ainsi cela donne :
  $f(N) = 2 * f(N/2)$

  Cependant, il faut aussi parcourir le tableau une fois de plus pour le reconstruire, on ajoute donc N :
  $f(N) = 2 * f(N/2) + N$

  Il faut bien comprendre que cette formule est utilisée récursivement, et donc que ce $2 * f(N/2)$ aura lui-même son égalité, et de même pour ce qui va en résulter, ceci jusqu'à ce que N soit égal à son dénominateur :
  $2 * f(N/2) = 4 * f(N/4)+N/2$ -> $4 * f(N/4) = 8 * f(N/8)+N/4$ -> $8 * f(N/8) = 16 * f(N/16)+N/8$ -> …

  On peut remarquer que des éléments s'annule comme $f(N/2)$ dans l'implication suivante (en bleu) :
  $f(N) = 2 * f(N/2) + N$ -> $2 * f(N/2) = 4 * f(N/4)+N/2$

  Ainsi cet élément s'annule aussi dans les cas suivants, sachant que la récursivité se fera uniquement jusqu'à ce que N soit égal à son dénominateur, le facteur (en rouge ci-dessus) atteindra la valeur de N, et donc la complexité sera $O(2^n+n*2^n)$ que l'on peut aussi noter $O(2^n *(1+n))$ avec n le nombre de division par 2 qu'il aura été nécessaire pour que N ai atteint son dénominateur (plus clairement que la taille des tableaux que l'on traite soit 1).

Il reste à comparer ces 2 complexités. On a vu que n correspond au nombre de division par 2 pour que N atteigne 1, soit $N/2^n=1$ ce qui équivaut à $N=2^n$. Donc comparé les 2 complexité $O(N^2)=O(N*N)$ et $O(2^n *(1+n))$ revient à comparer O(N) et O(1+n).

Or puisque que N est le résultat de 2^n c'est logique que dans $2^n=N$, n soit inférieur à N, le seul doute reste pour les plus petits cas c'est-à-dire n=0, et n=1 :

- N=1+n avec N=1 et n=0 car $2^0=1$
- N=1+n avec N=2 et n=1 car $2^1=2$
- N>1+n avec N=4 et n=2 car $2^2=4$
- N>1+n avec N=8 et n=3 car $2^3=4$

On remarque qu'effectivement pour des cas où N est petit il n'y a pas de différence, en revanche on remarque aussi que l'écart va continuer à grandir dès que le tableau contient plus de 2 éléments, ainsi la complexité de la transformée direct est bien supérieur à la complexité de la transformée rapide :
$O(N^2) > O(2^n *(1+n))$ avec n>1

TRANSFORMÉE RAPIDE 2D -- TRANSFORMÉE RAPIDE INVERSE

Pour les mêmes raisons que la transformée de Fourier direct, je peux réaliser la transformée de Fourier rapide, en appliquant ma fonction précédente d'abord sur toutes les lignes, puis sur toutes les colonnes, ainsi ma fonction **'transformeeRapide2D'** est exactement structuré comme **'transformeeDirect2D'.**

Concernant la formule pour la transformée rapide inverse il n'y a là aussi pas de grande différence puisque le seul changement est le signe moins dans l'exponentielle en facteur de la seconde somme qui disparait, ceci est la conséquence de l'avoir enlever dès le départ avant la séparation des pairs et impairs.
Mes 2 fonctions de la transformée de Fourier rapide inverse sont nommées **'transformeeRapide1Dinverse'** et **'transformeeRapide2Dinverse'**.