Pinon Alexandre
Perion Maxence

2021-2022

Projet d'outils mathématiques
Transformées de Fourier

Introduction : Dans le cadre de notre projet d'outils mathématiques, nous avons programmé différentes transformées de Fourier discrètes : normale ou inverse, directe ou rapide, 1D ou 2D. Les transformées donnent des nombres complexes, issus de l'usage de la fonction exponentielle complexe, notée exp. Nous allons expliquer dans ce rapport les différents algorithmes que nous avons implémentés en python.
Pour les transformées rapides, nous supposons que les tailles des données sont des puissances de 2.

I. Transformées normales

1. 1D

Nous allons commencer par évoquer la transformée de Fourier la plus simple : directe et en 1 dimension. Il s'agit donc de prendre un tableau I à une dimension et de taille N pour y appliquer simplement la formule pour chacun de ses éléments u, permettant de calculer sa transformée de Fourier discrète tf : $tf(u)=\sum_{x=0}^{N-1} I(x)*\exp(-2i\pi\frac{ux}{N})$.

Nous pouvons remarquer que la formule a recours à chacun des éléments du tableau et puisqu'elle est appliquée à chacun des éléments du tableau, la complexité asymptotique de l'algorithme est donc de $O(n^2)$, avec n le nombre d'éléments du tableau.

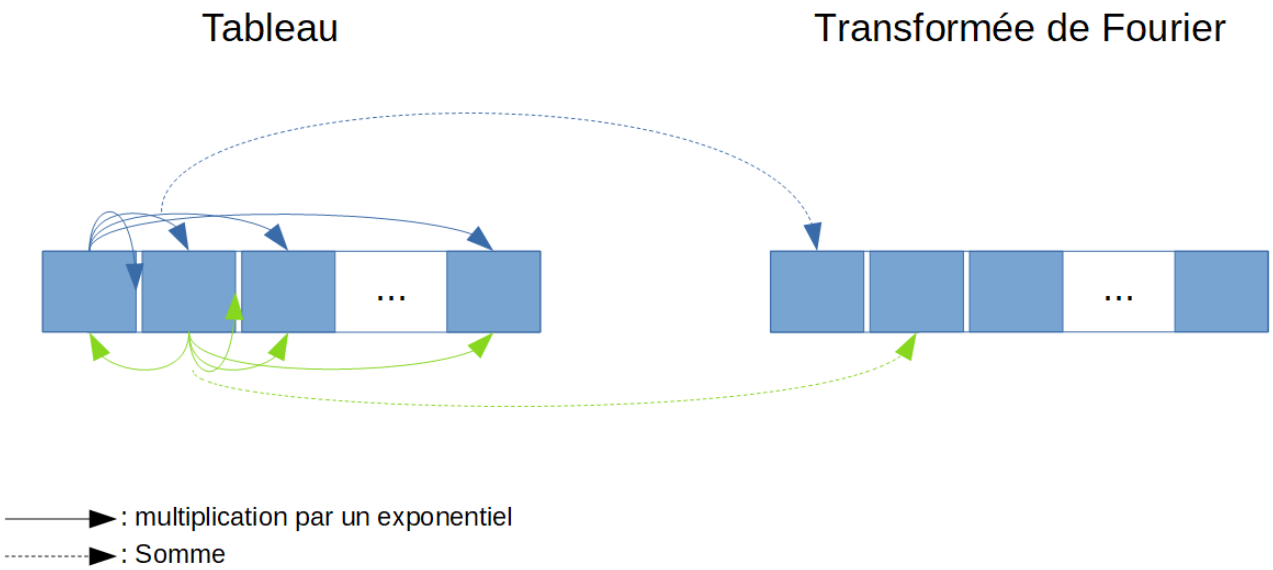

Schéma de la Transformée de Fourier 1D discrète directe

Nous avons obtenu une transformée, qui se situe dans le domaine fréquentiel. Elle peut être utilisée pour appliquer des filtres, une signature cachée, … mais il est alors nécessaire de pouvoir effectuer la transformée inverse.
La transformée de Fourier discrète inverse et directe 1D s'obtient grâce au même principe, donnant lieu au même algorithme à l'exception de la formule qui est à peine différente. En effet, elle devient : $itf(u)=\frac{1}{N}\sum_{x=0}^{N-1} tf(x)*\exp(2i\pi\frac{ux}{N})$, avec itf la transformée inverse et tf la transformée de Fourier obtenu auparavant.
La différence avec la formule précédente est que le signe dans l'exponentiel change. On peut aussi remarquer une division par la taille du tableau qui permet de retomber sur exactement les mêmes valeurs qu'avant les transformées (pour la partie réelle), sinon nous pouvons observer un phénomène de dilation entre les valeurs.

2. 2D

D'un point de vue informatique, la transformée discrète en 2 dimensions est plus intéressante car elle s'applique sur des images, nous permettant de les filtrer dans le domaine fréquentielle, de laisser un « tatouage » signature, … Dans ce cas, le tableau a 2 dimensions et chaque élément représente le niveau d'intensité de la couleur d'un pixel.

Pour l'algorithme, le principe est le même mais en 2 dimensions : on va appliquer pour chaque pixel une formule dépendant de tous les autre pixels (donc sur toutes les lignes et toutes les colonnes). Cette formule est, pour un pixel de coordonnées (u,v) dans le tableau I :
$tf(u,v)=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} I(x,y)*\exp(-2i\pi(\frac{xu}{M}+\frac{yv}{N}))$, avec M la hauteur du tableau et N sa largeur.

Nous allons donc parcourir toutes les lignes et parmi chaque ligne toutes les colonnes pour sélectionner chaque pixel, puis pour appliquer la formule nous devons encore parcourir tous les pixels du tableau (donc encore toutes les lignes et toutes les colonnes).
La complexité asymptotique de l'algorithme permettant de calculer cette transformée en 2 dimensions est donc en $O(M*N*M*N)=O(M^2*N^2)$.

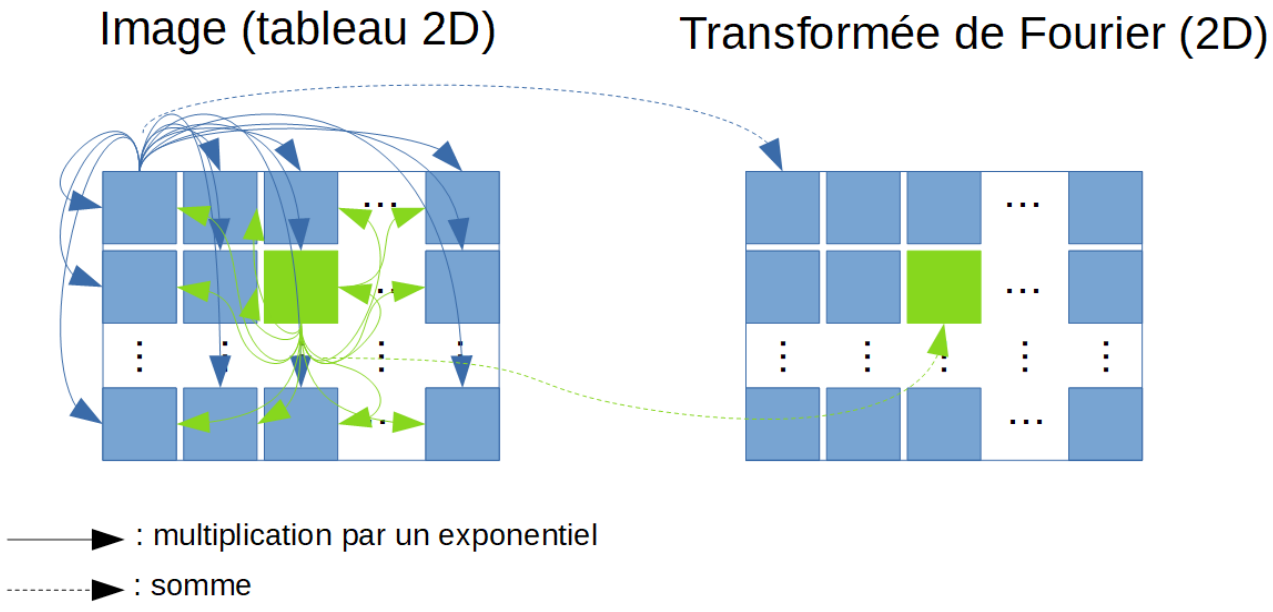

Schéma de la Transformée de Fourier 2D discrète directe

Tout comme pour le cas à une dimension, il peut être interressant d'effectuer l'opération inverse : passer de la transformée 2D à l'image. Encore une fois, cette opération est très similaire au cas précédent : seule la formule à appliquer à chaque pixel change et devient :

$$itf(u,v)=\frac{1}{M*N}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} tf(x,y)*\exp(2i\pi(\frac{xu}{M}+\frac{yv}{N}))$$.

Le signe négatif dans l'exponentiel a encore une fois disparu et nous divisons par les dimensions du tableau afin de supprimer la dilation occasionnée lors de l'opération.

Comme nous pouvons le voir sur le dernier schéma et avec les complexités asymptotiques des algorithmes, le temps de calcul nécessaire et le nombre d'opérations à effectuer devient très vite conséquent lors de l'augmentation des tailles des données. Il est donc judicieux de s'intéresser à d'autres algorithmes différents de ceux dits « naïfs » présentés jusqu'alors fournissants le même résultat tout en étant plus efficaces.

II. Transformées rapides

1. 1D

Nous allons commencer par nous concentrer sur l'amélioration de l'algorithme pour les transformées à une dimension. En effet, calculer la transformée en un point dépend de la valeur de tous les autres éléments du tableau mais il est possible de ne pas effectuer certains calculs superflus dans le but d'améliorer notre temps de calcul global. L'amélioration s'effectue grâce à deux remarques astucieuses combinées : une symétrie et une méthode de récursion se basant sur la séparation des données (méthode dite de « diviser pour mieux régner »).

Premièrement, nous avons supposé que nos données ont une taille en puissance de 2, ce qui nous permet de les découper en 2 parties égales : disons les éléments d'indice pairs et ceux d'indices impairs.

Nous allons voir ce que cela donne mathématiquement :

$$tf(u)=\sum_{x=0}^{N-1} I(x)*\exp(-2i\pi\frac{ux}{N})$$

$$tf(u)=\sum_{x=0,\,x\,pair}^{N-1} I(x)*\exp(-2i\pi\frac{ux}{N})+\sum_{x=0,\,x\,impair}^{N-1} I(x)*\exp(-2i\pi\frac{ux}{N})$$

on peut exprimer un nombre pair comme pair = 2*k et un impair comme impair = 2*k + 1, donc ici x pair = 2x' et x impair = 2x'+1:

$$tf(u)=\sum_{x'=0}^{N/2-1} I(2x')*\exp(-2i\pi\frac{u*2x'}{N})+\sum_{x'=0}^{N/2-1} I(2x'+1)*\exp(-2i\pi\frac{u*(2x'+1)}{N})$$

Puisque nous sommons toujours pour autant de x mais de deux en deux, x' ne va donc plus que jusqu'à la moitié par rapport à x : N/2-1.

Pour simplifier les notations, on effectue désormais un changement de variable pour repasser des x' aux x :

$$tf(u)=\sum_{x=0}^{N/2-1} I(2x)*\exp(-2i\pi\frac{u*2x}{N})+\sum_{x=0}^{N/2-1} I(2x+1)*\exp(-2i\pi\frac{u*(2x+1)}{N})$$

On va maintenant développer le « 2x+1 » :

$$tf(u)=\sum_{x=0}^{N/2-1} I(2x)*\exp(-2i\pi\frac{u*2x}{N})+\sum_{x=0}^{N/2-1} I(2x+1)*\exp(-2i\pi(\frac{u*(2x)}{N}+\frac{u}{N}))$$

$$tf(u)=\sum_{x=0}^{N/2-1} I(2x)*\exp(-2i\pi\frac{u*2x}{N})+\sum_{x=0}^{N/2-1} I(2x+1)*\exp(-2i\pi\frac{u*(2x)}{N}+-2i\pi\frac{u}{N})$$

Grâce aux propriétés de l'exponentiel : exp(a+b) = exp(a)*exp(b), nous obtenons :

$$tf(u)=\sum_{x=0}^{N/2-1} I(2x)*\exp(-2i\pi\frac{u*2x}{N})+\sum_{x=0}^{N/2-1} I(2x+1)*\exp(-2i\pi\frac{u*(2x)}{N})\exp(-2i\pi\frac{u}{N})$$

Puisque le facteur est commun à toute la somme, on peut la factoriser avec :

$$tf(u)=\sum_{x=0}^{N/2-1} I(2x)*\exp(-2i\pi\frac{u*2x}{N})+\exp(-2i\pi\frac{u}{N})\sum_{x=0}^{N/2-1} I(2x+1)*\exp(-2i\pi\frac{u*(2x)}{N})$$

On remarque qu'il est possible de diviser par 2 afin d'obtenir N/2 comme taille du tableau dans les sommes et dans les exponentiels (le 2x de l'exponentiel devient x et la multiplication du numérateur va diviser le dénumérateur) :

$$tf(u)=\sum_{x=0}^{N/2-1} I(2x)*\exp(-2i\pi\frac{u*x}{N/2})+\exp(-2i\pi\frac{u}{N})\sum_{x=0}^{N/2-1} I(2x+1)*\exp(-2i\pi\frac{u*x}{N/2})$$

On peut désormais reconnaître dans chaque somme une autre transformée de Fourier discrète : à gauche celle selon les indices pairs et à droite selon les indices impairs, mais multipliée par un coefficient.
En résumé : tf(u) = tf_pair(u) + coeff * tf_impair(u). Puisque ceci est vrai pour tout u du tableau, on peut calculer directement en une seule fois les transformées des indices pairs et impairs et sélectionner chaque élément en fonction de celui que l'on recherche.

On a donc trouvé un moyen de séparer en 2 les calculs. Cependant, il y en a toujours autant à faire ! Certes, nous n'avons plus que la moitié des calculs (N/2 dans les sommes), mais il faut le faire 2 fois ! Nous devons donc trouver un autre moyen de réduire les calculs et c'est la qu'intervient la symétrie.

Deuxièmement, il existe une symétrie que nous pouvons exploiter pour que, coupler avec la séparation, nous n'ayons plus qu'à faire que la moitié des calculs et ce à chaque étape !
Nous nous intéressons à la transformée de u + N, avec N la taille du tableau.

$$tf(u)=\sum_{x=0}^{N-1} I(x)*\exp(-2i\pi\frac{ux}{N}) \quad \text{d'où} \quad tf(u+N)=\sum_{x=0}^{N-1} I(x)*\exp(-2i\pi\frac{(u+N)x}{N})$$

On peut développer :

$$tf(u+N)=\sum_{x=0}^{N-1} I(x)*\exp(\frac{-2i\pi ux}{N}+\frac{-2i\pi Nx}{N})$$

On peut désormais simplifier par N au numérateur et au dénumérateur :

$$tf(u+N)=\sum_{x=0}^{N-1} I(x)*\exp(\frac{-2i\pi ux}{N}+(-2i\pi x))$$

Nous allons encore une fois utiliser cette propriété de l'exponentiel : exp(a+b) = exp(a)exp(b) :

$$tf(u+N)=\sum_{x=0}^{N-1} I(x)*\exp(\frac{-2i\pi ux}{N})\exp(-2i\pi x)$$

On peut utiliser une autre de ses propriétés : $\exp(an)=\exp(a)^n$ pour tout n entier.

Or $\exp(-2i\pi)=1$ et x est un entier variant de 0 à N-1 donc $\exp(-2i\pi x)=1$ pour tout x.

On tombe donc sur :

$$tf(u+N)=\sum_{x=0}^{N-1} I(x)*\exp(\frac{-2i\pi ux}{N})=tf(u)$$

Lorsqu'on sépare les indices pairs et les indices impairs, on peut donc calculer la seconde moitié avec les 2 mêmes transformées !

On avait résumé par :
tf(u) = tf_pair(u) + coeff * tf_impair(u)

mais on a aussi, grâce à la symétrie :
tf(u + N/2) = tf_pair(u) + coeff * tf_impair(u)

Pour que cela fonctionne d'un point de vu algorithmique, il ne reste plus qu'à trouver le « cas de base », c'est le cas sur lequel on donnera directement le résultat sans appel récursif, ce qui permet de ne pas faire des appels récursifs en boucle sans fin, ce qui ne nous donnerait aucun résultat. Pour le cas de base, nous avons choisis le cas où il n'y a qu'un seul élément dans le tableau (N=1). En effet, sa transformée est toute simple : c'est lui même.
Mathématiquement, nous avons :

$$tf(u)=\sum_{x=0}^{N-1} I(x)*\exp(-2i\pi\frac{ux}{N})$$

$$tf(u)=\sum_{x=0}^{0} I(x)*\exp(-2i\pi\frac{ux}{1})$$

d'où

$$tf(u)=I(0)*\exp(-2i\pi u*0)$$
$$tf(u)=I(0)*\exp(0)$$
$$tf(u)=I(0)$$, pour u variant de 0 à 0 (u = 0).

L'algorithme est désormais simple : on teste si nous sommes dans le cas de base (taille du tableau = 1) : on renvoie le seul élément du tableau et sinon on découpe selon les indices pairs et impairs pour effectuer des appels récursifs, puis pour chaque élément u allant jusqu'à la moitié, on applique les formules sur les résultats :
tf(u) = tf_pair(u) + coeff * tf_impair(u)
tf(u + N/2) = tf_pair(u) + coeff * tf_impair(u)
où coeff dépend de u ou u+N/2 : coeff(x) = $coeff(x)=\exp(-2i\pi\frac{x}{N})$, x pouvant donc être u ou u + N/2.

Nous arrivons au moment fatidique où il faut se demander si l'algorithme est bel et bien plus efficace, grâce à la complexité asymptotique. Lors de chaque appel récursif, on divise par 2 le nombre de calculs à effectuer par rapport aux données et ce pour la puissance de 2 de la taille de

l'image : log(N). La complexité est donc de Nlog(N) ce qui est nettement inférieur à celle de son homologue naïf qui est de N^2.

L'algorithme nous donnant la transformée inverse est exactement le même, à la différence près que le paramètre du coefficient est de signe positif.

2. 2D

Maintenant que nous sommes capables de calculer une transformée de Fourier discrète à une dimension avec une efficacité confortable, nous pouvons améliorer notre algorithme pour la 2D, en nous basant sur celui que nous venons d'améliorer. En effet, nous pouvons déduire la transformée 2D à partir de celles sur les lignes et les colonnes, qui sont donc des tableaux à une dimension !

Démontrons le mathématiquement :

$$tf(u,v)=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} I(x,y)*\exp\left(-2i\pi\left(\frac{xu}{M}+\frac{yv}{N}\right)\right)$$

On peut développer et utiliser les propriétés de l'exponentiel comme précédemment pour obtenir :

$$tf(u,v)=\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} I(x,y)*\exp\left(-2i\pi\frac{xu}{M}\right)\exp\left(-2i\pi\frac{yv}{N}\right)$$

Puisque le premier exponentiel ne dépend pas de y, nous pouvons le factoriser avec la deuxième somme (pour les y) :

$$tf(u,v)=\sum_{x=0}^{M-1}\exp\left(-2i\pi\frac{xu}{M}\right)\sum_{y=0}^{N-1} I(x,y)\exp\left(-2i\pi\frac{yv}{N}\right)$$

Dans la deuxième somme, nous pouvons reconnaître la formule des transformées de Fourier des lignes x en v.
On a donc :

$$tf(u,v)=\sum_{x=0}^{M-1}\exp\left(-2i\pi\frac{xu}{M}\right)tfLigneX(v)$$

Et maintenant on peut également reconnaître la transformée de Fourier en u selon les colonnes des transformées des lignes. Il s'agit donc d'une composition mathématique, de plusieurs transformées à une dimension, dont nous avons déjà une version améliorée !

L'algorithme qui découle de cette nouvelle remarque mathématique va simplement appeler l'algorithme 1D sur les lignes, puis l'appeler à nouveau selon les colonnes des transformées obtenues. Cette version effectue donc M transformées (selon les lignes) puis N selon les colonnes : elle effectue donc M+N transformées rapides, pour lesquelles on a une complexité de Nlog(N). On a donc une complexité totale de $O(Mlog(M)+Nlog(N))$ ce qui est bien inférieur à une complexité de $O(M^2*N^2)$ pour son homologue naïf.

La transformée inverse avec exactement le même algorithme, sauf qu'à la place d'appeler des transformées 1D directes nous appelons des transformées 1D inverses.

Conclusion : Nous avons pu voir qu'il a été possible d'améliorer le temps de calcul asymptotique des divers algorithmes grâce à des propriétés mathématiques, notamment de l'exponentiel. Nous avons aussi remarquer que les algorithmes sont exactement les mêmes pour les transformées normales et inverses, à l'exception de signes dans les formules.