

|  |
| --- |
|  |
| Compte rendu –  Projet L3  Outils maths |
| Transformée de Fourier |

|  |
| --- |
| Tom BARBIER – Brandon PERE  Année 2022 - 2023 |

Table des matières

[I. Introduction 3](#_Toc120724106)

[II. Transformée de Fourier Directe 3](#_Toc120724107)

[III. Optimisation 4](#_Toc120724108)

[1. Transformée 1D 4](#_Toc120724109)

[2. Inverse 6](#_Toc120724110)

[3. Transformée 2D 7](#_Toc120724111)

[IV. Complexité 8](#_Toc120724112)

[1. Transformée de Fourier directe 8](#_Toc120724113)

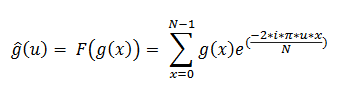
[2. Transformée de Fourier Rapide 8](#_Toc120724114)

[V. performance 9](#_Toc120724115)

# 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(𝑁²), c’est pourquoi il existe, la transformé de Fourier rapide, qui a une complexité simplifiée. En effet, sa complexité est de 𝑂(𝑁 ∗) pour des tableaux de tailles de taille . 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.

# Transformée de Fourier Directe



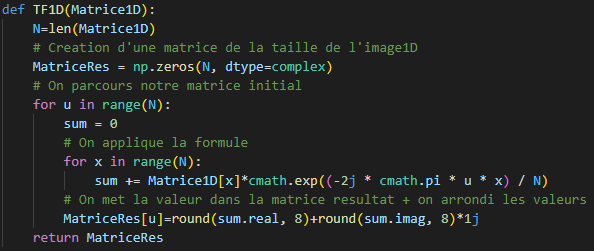
u index du tableau

N taille du tableau

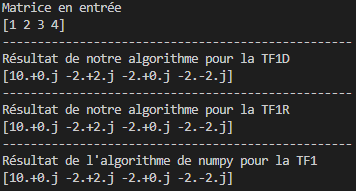
g tableau source

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(N²). En l’implémentant en python cela nous donne :



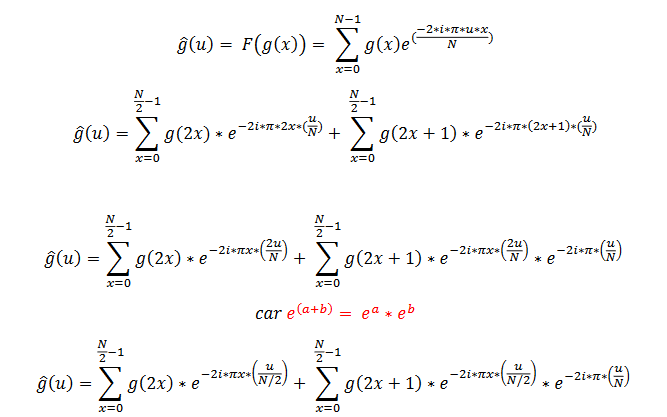
On peut comparer les résultats avec la fft de numpy. (bibliothèque que nous utilisons pour simplifier le code)

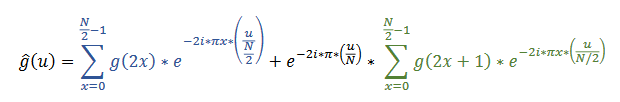


# Optimisation

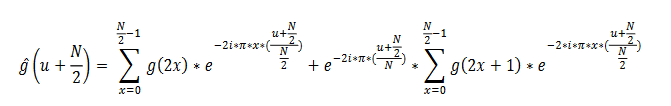
## Transformée 1D

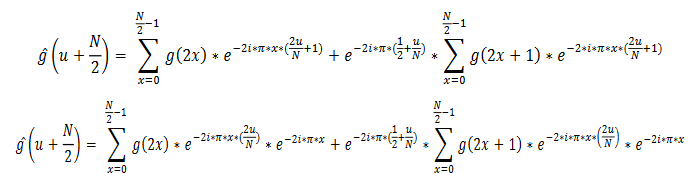
Nous pouvons séparer la somme en deux sommes afin d’avoir les éléments pairs d’un coté et impairs de l’autre côté ce qui donne :

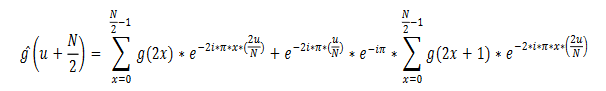
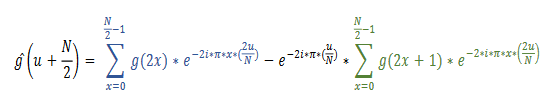


Le terme ne dépend pas de la variable de la somme il est donc possible de la sortir de la somme :

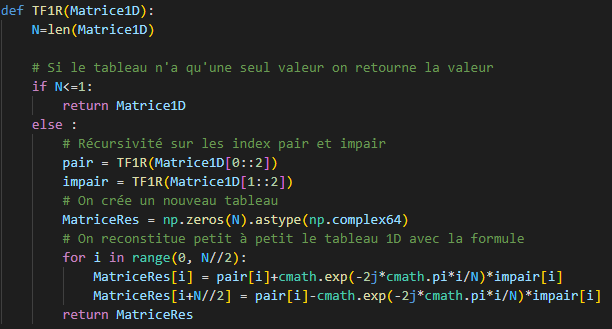
Nous avons donc en bleu à gauche les éléments pairs et en vert à droite les éléments impairs. On a donc notre tableau, mais celui-ci n’est pas complet car le tableau est de taille N/2.

Nous devons donc calculer la seconde moitié soit :

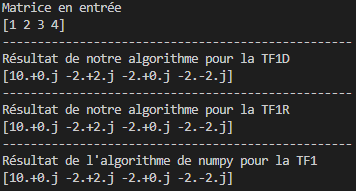
Simplifions cette expression :

avec , pour trouver ce résultat on se réfère au cercle trigonométrique et à la formule d’Euler. 

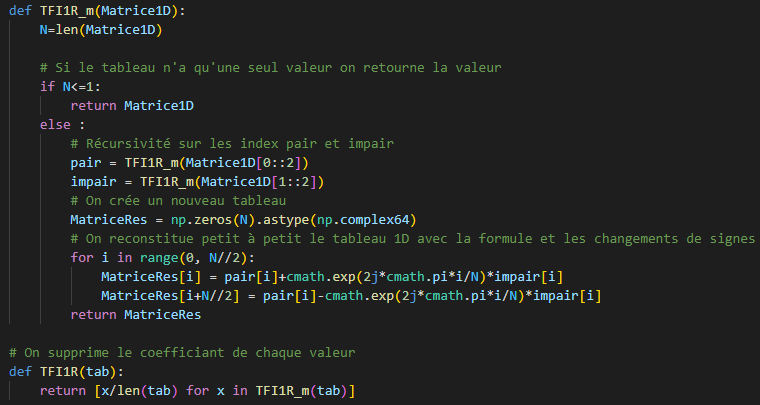
Maintenant que nous avons simplifier le problème nous pouvons passer à l’implémentation. L’avantage de la représentation mathématique que nous avons est qu’elle laisse bien voir un algorithme récursif.



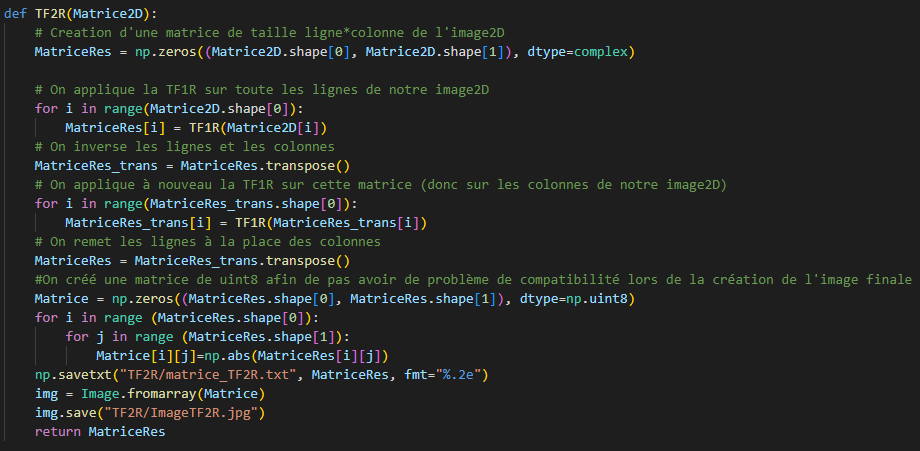
On peut comparer les résultats avec la fft de numpy.



## Inverse

Pour ce qui est de l’inverse nous allons juste faire des changements de signe comme décrit par la formule et supprimer le coefficient pour chaque valeur retournée.

## Transformée 2D

Nous avons donc notre transformée de Fourier rapide 1D. Pour passer en 2D il suffit d’appliquer la même formule sur toutes les lignes et sur toutes les colonnes. Pour cela on fait appel à la transposé car cela revient à appliquer la transformée de Fourier rapide sur chaque ligne puis à refaire la même opération sur la transposé du résultat.

Pour calculer son inverse on fait exactement la même implémentation sauf que l’on utilise la transformée de Fourier rapide Inverse 1D sur chaque ligne et sur chaque colonne.

# Complexité

## Transformée de Fourier directe

On fait deux boucles, une qui parcourt le tableau et se place sur la valeur à calculer, et l’autre qui une fois que l’on est sur cette valeurs doit de nouveau parcourir toute la liste pour faire la somme.

Soit sur un tableau de taille N on a : f(N) = N \* N

Donc une complexité de O(N²)

## Transformée de Fourier Rapide

On a vu que l’on avait 2 formules similaires pour une taille divisé par 2.

Soit un tableau de taille N on a : f(N) = 2 \* f( )

Or nous devons reconstituer le tableau donc on a une boucle qui parcourt ce tableau.

f(N) = 2 \* f( )+ N

Or cette formule est utilisée récursivement c’est-à-dire :

n lignes

avec N =

On remarque donc qu’a chaque fois les éléments s’annule dans les implications qui en découle :

La récursivité se fera n fois jusqu’à ce que le dénominateur soit égale à N.

Donc on a :

# performance

Maintenant que nous avons une implémentation fonctionnelle de la transformée de Fourier rapide nous pouvons comparer avec la transformée de Fourier directe le temps que l’algorithme va mettre pour s’exécuter sur une image. Pour cela nous utilisons la bibliothèque « time ».

Nous avons testé les deux algorithmes sur une image de taille 128 \* 128. On voit donc bien que la transformée de Fourier rapide est nettement plus rapide que la directe. Cela montre donc que la complexité de la FFT est bien inférieure à la directe.