



LICENCE 3 – OUTILS MATHÉMATIQUES

Rapport de Projet
*Transformée de Fourier – Naïve et
Rapide (FFT)*

Auteurs :
Kramer Axel
Cléry Arsène

Année universitaire 2025–2026

Introduction

Ce projet a pour but d'implémenter les différentes versions de la transformée de Fourier discrète (TFD), en une et deux dimensions, ainsi que leurs versions inverses et rapides (FFT). Nous avons choisi pour ce faire de l'implémenter en langage **Java**, un choix personnel motivé par le fait qu'il s'agit du langage que nous estimons maîtriser le mieux. Ce projet aurait toutefois pu être réalisé dans d'autres langages tels que **Python**, **Scilab** ou **Matlab**, qui offrent également des bibliothèques adaptées au traitement du signal et à la manipulation de nombres complexes.

Transformée de Fourier discrète

La transformée de Fourier discrète (TFD) est un outil mathématique permettant de représenter un signal selon ses fréquences. Elle transforme un signal discret $g(x)$ en un signal $\hat{g}(u)$, appelé spectre de fréquences. La TFD décompose le signal en une somme de sinusoides de différentes fréquences, ce qui permet d'analyser son contenu fréquentiel.

La formule directe de la TFD à une dimension est donnée par :

$$\hat{g}(u) = \sum_{x=0}^{N-1} g(x) e^{-2i\pi ux/N} \quad \text{avec } u = 0..N-1$$

et la formule inverse par :

$$g(x) = \sum_{u=0}^{N-1} \hat{g}(u) e^{2i\pi ux/N} \quad \text{avec } x = 0..N-1$$

La classe Complexe

La transformée de Fourier utilise des nombres complexes pour représenter les composantes sinusoïdales du signal. Nous avons donc implémenté une classe **Complexe** permettant de manipuler ces nombres. Elle contient deux attributs : la partie réelle et la partie imaginaire, et fournit les opérations de base nécessaires :

```
public class Complexe {
    private double reel;
    private double imag;

    public Complexe(double reel, double imag) { ... }
    public Complexe ajouter(Complexe autre) { ... }
    public Complexe soustraire(Complexe autre) { ... }
    public Complexe multiplier(Complexe autre) { ... }
}
```

Cette classe simplifie la manipulation des valeurs complexes dans les calculs et rend le code plus lisible.

Transformée de Fourier 1D (version naïve)

Nous avons ensuite implémenté la version **naïve** de la transformée de Fourier 1D. Elle applique directement la formule mathématique précédente à l'aide de deux boucles imbriquées.

```
// ----- Version naïve -----
public static Complexe[] TransformeeFourier1DN(Complexe[] g) {
    int N = g.length;
    Complexe[] gtransfo = new Complexe[N];

    for (int u = 0; u < N; u++) {
        Complexe somme = new Complexe(0, 0);

        for (int x = 0; x < N; x++) {
            double angle = -2 * Math.PI * u * x / N;
            Complexe facteur = new Complexe(Math.cos(angle), Math.sin(angle));
            somme = somme.ajouter(g[x].multiplier(facteur));
        }

        gtransfo[u] = somme;
    }

    return gtransfo;
}
```

Explication du fonctionnement

La première boucle `for (u)` parcourt les fréquences de sortie du signal transformé. La seconde boucle `for (x)` parcourt les valeurs du signal d'entrée et calcule leur contribution à chaque fréquence.

La fonction applique donc la formule de la TFD directement, sans optimisation.

Analyse de la complexité

La version naïve de la transformée de Fourier 1D repose sur deux boucles imbriquées.

Pour chaque fréquence u , la fonction doit donc effectuer N multiplications et additions. Comme il y a N fréquences à calculer, le programme réalise au total environ $N \times N$ opérations.

Ainsi, la complexité de la version naïve est dite en $O(N^2)$. Cela signifie que le temps de calcul augmente très rapidement lorsque la taille du signal croît. Par exemple, si on double le nombre d'échantillons du signal, le temps de calcul est multiplié par quatre.

Cette méthode est donc peu efficace pour les grands signaux. C'est pour cela que, dans la suite du projet, nous avons implémenté une version dite « rapide », qui possède une complexité plus faible.

Transformée de Fourier 1D Inverse (version naïve)

L'implémentation de la version naïve de la transformée de Fourier 1D inverse suit exactement la même logique que la transformée 1D directe. Il suffit simplement de reprendre la même structure de code, en changeant le signe de l'angle dans l'exponentielle (qui devient positif).

La complexité de cette version est la même que celle de la transformée directe, soit $O(N^2)$.

Transformée de Fourier 2D (version naïve)

La transformée de Fourier 2D est une extension directe de la version 1D. Elle est utilisée principalement pour le traitement d'images, car elle permet d'analyser les fréquences spatiales dans deux directions : horizontale et verticale. L'idée est la même que pour la 1D, mais on travaille ici sur une matrice bidimensionnelle représentant une image.

La formule mathématique de la TFD 2D est donnée par :

$$\hat{g}(u, v) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} g(x, y) e^{-2i\pi \left(\frac{ux}{N} + \frac{vy}{M} \right)}$$

Elle transforme l'image $g(x, y)$ en une image $\hat{g}(u, v)$ dans le domaine fréquentiel.

// ----- Version naïve -----

```
public static Complexe[][] TransformeeFourier2DN(Complexe[][] g) {
    int N = g.length;
    int M = g[0].length;
    Complexe[][] gtransfo = new Complexe[N][M];

    for (int u = 0; u < N; u++) {
        for (int v = 0; v < M; v++) {
            Complexe somme = new Complexe(0, 0);

            for (int x = 0; x < N; x++) {
                for (int y = 0; y < M; y++) {
                    double angle = -2 * Math.PI * ((double) u * x / N +
                    (double) v * y / M);
                    Complexe facteur = new Complexe(Math.cos(angle),
                    Math.sin(angle));
                    somme = somme.ajouter(g[x][y].multiplier(facteur));
                }
            }

            gtransfo[u][v] = somme;
        }
    }
}
```

```

        return gtransfo;
    }

```

Explication du fonctionnement

La première boucle `for (u)` parcourt les fréquences horizontales, et la seconde `for (v)` parcourt les fréquences verticales.

Pour chaque couple (u, v) , les boucles internes parcourent tous les pixels de l'image et additionnent leurs valeurs. On applique simplement la formule de la transformée pour obtenir la fréquence correspondante.

Le résultat est stocké dans la matrice `gtransfo`, qui contient l'image transformée dans le domaine fréquentiel.

Analyse de la complexité

La version 2D naïve utilise quatre boucles imbriquées (deux boucles parcourant les N éléments et deux boucles parcourant les M éléments), ce qui entraîne une complexité de $O(N^2M^2)$. Cette méthode est donc coûteuse pour les grandes images, mais elle applique la formule de la TFD directement, sans aucune optimisation.

Transformée de Fourier 2D Inverse (version naïve)

La version 2D inverse suit exactement la même logique que la version directe. On reprend la même structure de code, mais avec un signe positif dans l'exponentielle. Elle permet de reconstruire l'image d'origine à partir de son spectre fréquentiel.

La complexité de cette version inverse est identique à celle de la version directe, soit $O(N^2M^2)$.

Transformée de Fourier 1D – Version rapide (FFT)

La transformée de Fourier rapide, appelée **FFT (Fast Fourier Transform)**, est une version optimisée de la transformée de Fourier discrète. Alors que la version naïve nécessite environ $O(N^2)$ opérations pour un signal de taille N , la FFT réduit ce nombre à seulement $O(N \log_2 N)$. Cette amélioration rend possible le traitement rapide de signaux ou d'images de grande taille.

Principe mathématique

L'idée de base de la FFT est de tirer parti des **symétries** et **périodicités** de la fonction exponentielle complexe utilisée dans la transformée de Fourier :

$$e^{-2i\pi ux/N}$$

Ces symétries permettent d'éviter de recalculer plusieurs fois les mêmes valeurs.

On part de la formule classique de la TFD :

$$\hat{g}(u) = \sum_{x=0}^{N-1} g(x) e^{-2i\pi ux/N}$$

Pour simplifier le calcul, on suppose que la taille du signal N est une **puissance de 2**, c'est-à-dire qu'il existe un entier n tel que $N = 2^n$. Cette condition permet de diviser le signal en deux sous-signaux de taille $N/2$ à chaque étape, jusqu'à obtenir des signaux de taille 1.

On sépare alors la somme en deux parties :

- une partie contenant les indices pairs $x = 2k$,
- et une autre contenant les indices impairs $x = 2k + 1$.

On obtient :

$$\hat{g}(u) = \sum_{k=0}^{N/2-1} g(2k)e^{-2i\pi u(2k)/N} + \sum_{k=0}^{N/2-1} g(2k+1)e^{-2i\pi u(2k+1)/N}$$

En factorisant, on a :

$$\hat{g}(u) = \underbrace{\sum_{k=0}^{N/2-1} g(2k)e^{-2i\pi uk/(N/2)}}_{\text{partie paire}} + e^{-2i\pi u/N} \underbrace{\sum_{k=0}^{N/2-1} g(2k+1)e^{-2i\pi uk/(N/2)}}_{\text{partie impaire}}$$

On remarque ici que chaque somme est une **transformée de Fourier discrète de taille $N/2$** . Cela signifie que, pour calculer la transformée d'un signal de taille N , on peut réutiliser les résultats de deux transformées de taille $N/2$.

Implémentation de la méthode rapide

L'algorithme de la FFT repose sur une idée simple : au lieu de calculer directement la somme complète de la transformée, on découpe le signal en deux moitiés, puis on combine les résultats.

On commence par séparer le signal d'entrée en deux parties :

- une partie contenant les valeurs d'indice pair $g(2k)$,
- une partie contenant les valeurs d'indice impair $g(2k + 1)$.

On calcule ensuite la transformée de Fourier pour ces deux moitiés séparément (chacune de taille $N/2$). Puis on combine les deux résultats pour retrouver la transformée complète du signal.

Combinaison des deux sous-résultats

Une fois la transformée calculée séparément sur les parties paires et impaires, il faut recombinaison les deux moitiés pour obtenir la transformée complète du signal. Cette étape repose sur une propriété essentielle de la transformée de Fourier : son spectre est **périodique**.

En effet, lorsque l'on passe de la première moitié du spectre à la seconde, le terme exponentiel de la transformée s'écrit :

$$e^{-2i\pi(u+N/2)x/N} = e^{-2i\pi ux/N} \cdot e^{-i\pi x} = e^{-2i\pi ux/N} \cdot (-1)^x$$

Ce facteur $(-1)^x$ fait apparaître un changement de signe pour les indices impairs. Ainsi, pour la seconde moitié du spectre (les hautes fréquences), la contribution des termes impairs est inversée.

Lors de la combinaison des deux moitiés, on distingue les basses fréquences ($u = 0$ à $N/2 - 1$) et les hautes fréquences ($u = N/2$ à $N - 1$). Le passage à $u + N/2$ revient

à se placer au début des hautes fréquences du spectre. Ce décalage modifie la phase des valeurs issues de la partie impaire, ce qui entraîne une inversion de signe dans la formule de combinaison. C'est pour cette raison que la première moitié du spectre se calcule avec une addition, tandis que la seconde se calcule avec une soustraction.

C'est cette propriété qui justifie la manière dont les deux moitiés sont combinées :

$$\begin{cases} \hat{g}(u) = \hat{g}_{pairs}(u) + e^{-2i\pi u/N} \hat{g}_{impairs}(u), \\ \hat{g}(u + N/2) = \hat{g}_{pairs}(u) - e^{-2i\pi u/N} \hat{g}_{impairs}(u) \end{cases}$$

Autrement dit :

- pour la première moitié des fréquences, on **additionne** les deux moitiés (elles sont en phase) ;
- pour la seconde moitié, on **soustraie** les contributions

Cette étape de combinaison permet donc de reconstruire les N fréquences du signal original à partir des deux transformées de taille $N/2$, tout en respectant les relations de phase entre les composantes.

Cette méthode permet de réduire fortement le temps de calcul : la complexité passe de $O(N^2)$ pour la version naïve à $O(N \log N)$ pour la version rapide. Toutes ces étapes nous donnent l'implémentation suivante :

```
// ----- Version rapide (FFT) -----
public static Complexe[] TransformeeFourier1DR(Complexe[] g) {
    int N = g.length;
    if (N == 1) return new Complexe[]{g[0]};

    if ((N & (N - 1)) != 0) {
        throw new IllegalArgumentException(" signale  n'est pas une puissance de 2");
    }

    Complexe[] pairs = new Complexe[N / 2];
    Complexe[] impairs = new Complexe[N / 2];

    for (int i = 0; i < N / 2; i++) {
        pairs[i] = g[2 * i];
        impairs[i] = g[2 * i + 1];
    }

    Complexe[] transfoPairs = TransformeeFourier1DR(pairs);
    Complexe[] transfoImpairs = TransformeeFourier1DR(impairs);
    Complexe[] gtransfo = new Complexe[N];

    for (int u = 0; u < N / 2; u++) {
        double angle = -2 * Math.PI * u / N;
        Complexe facteur = new Complexe(Math.cos(angle), Math.sin(angle));
        Complexe temp = facteur.multiplier(transfoImpairs[u]);
        gtransfo[u] = transfoPairs[u].ajouter(temp);
        gtransfo[u + N / 2] = transfoPairs[u].soustraire(temp);
    }
}
```

```

    return gtransfo;
}

```

Explication de l'implémentation

Ce code traduit directement le principe de la FFT expliqué précédemment. Il commence par vérifier que la taille du signal N est bien une puissance de 2, condition nécessaire à la division récursive du tableau. Ensuite, le signal est séparé en deux parties :

- la partie des indices pairs, notée $g(2k)$;
- la partie des indices impairs, notée $g(2k + 1)$.

L'algorithme appelle ensuite la fonction récursivement sur ces deux sous-tableaux jusqu'à obtenir des tableaux de taille 1 (cas de base). Enfin, les deux résultats intermédiaires sont combinés selon les formules vues précédemment : les fréquences basses sont obtenues par addition, et les fréquences hautes par soustraction.

Ainsi, cette implémentation applique fidèlement le principe mathématique de la FFT, tout en divisant le temps de calcul par rapport à la version naïve

Analyse de la complexité

Le calcul détaillé de la complexité a été réalisé en cours. Nous avons montré que la version rapide (FFT) possède une complexité en $O(N \log N)$.

Transformée de Fourier 1D Inverse – Version rapide (IFFT)

L'algorithme inverse de la IFFT (IFFT) repose sur le même principe que la version directe. Il s'agit simplement d'inverser le signe dans le terme exponentiel, sans ajouter de facteur de normalisation, conformément à la convention utilisée dans le cours.

Comme la structure de l'algorithme reste la même (division du signal en parties paires et impaires, puis combinaison des résultats), nous ne détaillons pas davantage cette partie.

La complexité reste identique à celle de la FFT directe, soit $O(N \log N)$.

Transformée de Fourier 2D – Version rapide (FFT 2D)

La transformée de Fourier rapide en deux dimensions, appelée **FFT 2D**, est une extension directe de la FFT 1D. Elle permet d'analyser le contenu fréquentiel d'une image ou d'un signal en 2 dimension. L'idée principale est que la transformée 2D peut être calculée à partir de plusieurs transformées 1D, ce qui permet de réduire considérablement le temps de calcul.

Principe mathématique

La transformée de Fourier discrète en deux dimensions est définie par :

$$\hat{g}(u, v) = \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} g(x, y) e^{-2i\pi(ux/N + vy/M)}$$

On remarque que cette transformée est **séparable**, c'est-à-dire qu'elle peut être décomposée en deux transformées 1D :

$$\hat{g}(u, v) = FFT_x(FFT_y(g(x, y)))$$

Ce qui signifie qu'on peut d'abord appliquer une FFT 1D sur chaque ligne de l'image, puis sur chaque colonne du résultat obtenu.

Principe d'implémentation

Mathématiquement, on peut réécrire la formule sous la forme :

$$\hat{g}(u, v) = \sum_{x=0}^{N-1} e^{-2i\pi ux/N} \left[\sum_{y=0}^{M-1} g(x, y) e^{-2i\pi vy/M} \right]$$

Cette réécriture montre qu'il suffit d'appliquer deux transformées successives :

1. D'abord une FFT 1D sur chaque **ligne** de l'image (direction horizontale),
2. Puis une FFT 1D sur chaque **colonne** du résultat (direction verticale).

Ainsi, la FFT 2D peut être réalisée efficacement en combinant plusieurs FFT 1D.

Explication du fonctionnement

Concrètement, l'algorithme procède comme suit :

- On applique la **FFT 1D** à chaque ligne de l'image $g(x, y)$, ce qui transforme la dimension horizontale.
- On applique ensuite la **FFT 1D** à chaque colonne du résultat, ce qui transforme la dimension verticale.

Cette méthode exploite la propriété de séparabilité pour éviter de recalculer la double somme complète. La complexité totale devient alors :

$$O(NM \log N \log M)$$

au lieu de $O(N^2M^2)$ pour la version naïve, ce qui représente un gain considérable pour le traitement d'images de grande taille.

Toutes ces étapes nous donnent l'implémentation suivante :

```
// ----- Version rapide (FFT 2D) -----
public static Complexe[] [] TransformeeFourier2DR(Complexe[] [] image) {
    int N = image.length;          // nombre de lignes
    int M = image[0].length;       // nombre de colonnes

    // Étape 1 : FFT sur chaque ligne
    Complexe[] [] temp = new Complexe[N][M];
    for (int x = 0; x < N; x++) {
        temp[x] = TransformeeFourier1DR(image[x]);
    }

    // Étape 2 : FFT sur chaque colonne
    Complexe[] [] imageTransfo = new Complexe[N][M];
```

```

    for (int v = 0; v < M; v++) {
        Complexe[] colonne = new Complexe[N];
        for (int x = 0; x < N; x++) {
            colonne[x] = temp[x][v];
        }

        Complexe[] colonneTransfo = TransformeeFourier1DR(colonne);

        for (int x = 0; x < N; x++) {
            imageTransfo[x][v] = colonneTransfo[x];
        }
    }

    return imageTransfo;
}

```

Explication du fonctionnement

La première boucle applique la FFT 1D sur chaque ligne de l'image, ce qui transforme les valeurs selon la direction horizontale. Le résultat est stocké dans une matrice temporaire `temp`.

La seconde partie du code parcourt ensuite les colonnes de cette matrice, extrait chaque colonne sous forme de tableau, et lui applique à nouveau la FFT 1D. Les résultats sont replacés dans la matrice finale `imageTransfo`.

Ainsi, l'algorithme réalise d'abord une transformation selon l'axe horizontal, puis selon l'axe vertical, permettant de calculer efficacement la transformée de Fourier.

Analyse de la complexité

Comme nous l'avons vu, la version rapide 2D consiste à appliquer deux transformées de Fourier rapides 1D : une sur les lignes et une sur les colonnes.

Chaque transformée 1D a une complexité en $O(N \log N)$. Ainsi, pour l'ensemble des lignes et des colonnes, on multiplie les deux complexités, ce qui donne :

$$O(N \log N) \times O(M \log M) = O(NM \log N \log M)$$

où N représente le nombre de lignes et M le nombre de colonnes de l'image.

Cette complexité reste bien plus faible que celle de la version naïve, qui est en $O(N^2 M^2)$, et permet de traiter efficacement des images de grande taille.

Transformée de Fourier 2D – Version rapide inverse (IFFT 2D)

La transformée de Fourier rapide inverse (IFFT 2D) permet de retrouver l'image d'origine à partir de son spectre de fréquences. Le principe mathématique et algorithmique est exactement le même que pour la FFT 2D directe, à la différence près du **signe de l'exponentielle complexe**.

Principe général

La formule de la transformée de Fourier inverse s'écrit :

$$g(x, y) = \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} \hat{g}(u, v) e^{2i\pi(\frac{ux}{N} + \frac{vy}{M})}$$

On remarque que la seule différence avec la transformée directe réside dans le **signe positif** de l'exponentielle. Cela signifie que, dans l'implémentation, il suffit d'inverser le signe de l'angle calculé à chaque étape.

Principe d'implémentation

L'implémentation de l'IFFT 2D reprend donc la même logique que la FFT 2D :

- on applique d'abord la **FFT inverse 1D** à chaque ligne de l'image fréquentielle,
- puis la **FFT inverse 1D** à chaque colonne du résultat obtenu.

Le code est donc très similaire à celui de la FFT 2D directe, à l'exception du signe des angles utilisés dans les calculs internes. Pour cette raison, l'implémentation détaillée n'est pas redonnée ici.

Analyse de la complexité

La complexité de la version rapide inverse reste identique à celle de la version directe, car les mêmes opérations sont effectuées, seules les phases changent de signe. Ainsi, la complexité est :

$$O(NM \log N \log M)$$

Cette méthode permet donc, tout comme la FFT 2D directe, de reconstruire efficacement une image à partir de son spectre fréquentiel, même pour des dimensions importantes.