UFR sciences & techniques

UNIVERSITÉ DE BOURGOGNE

JavaFFT

LACHAUD Samuel / PAZOLA Loïs – Info S5 TP4

SOMMAIRE

Introduction

Afin de mener à bien le projet d'outils mathématiques, nous avons choisis d'utiliser le langage JAVA.

Cependant dans JAVA, il n'existe pas de bibliothèques pour la gestion des nombres complexes. Ainsi nous avons donc créé une classe **Complexe** dans notre projet.

La classe **Complexe** est composée de deux attributs :

```
1.  double real = 0; // désigne la partie Réelle du complexe.
2.  Double imag = 0; // désigne la partie Imaginaire du complexe.
```

Nous avons ensuite ajouté quelques opérations afin de pouvoir effectuer les calculs élémentaires :

```
1.  Complexe add(Complexe c)      // ajout d'un complexe c à celui que l'on a.
2.  Complexe minus(Complexe c)    // retrait d'un complexe c à celui que l'on a.
3.  Complexe multiply(Complexe c) // multiplie le complexe que l'on a par c.
4.  Complexe divide(Complexe c)   // divise le complexe que l'on a par c.
```

Transformée 1D

```java
public static ArrayList<Complexe> Transformee1D(ArrayList<Complexe> tableau1D, int sensTransformee) {
        // Récupération de la taille du tableau
        int N = tableau1D.size();

        // Création du tableau de résultat
        ArrayList<Complexe> resultat = new ArrayList<Complexe>();

        for(int i = 0; i < N; i++) {
            // Création de la variable de somme des éléments du tableau
            Complexe additionLocale = new Complexe(0, 0);
            for (int j = 0; j < N; j++) {
                double theta = (sensTransformee * 2.0 * Math.PI * i * j) / N; // Angle thêta

                final Complexe complexeOriginal = tableau1D.get(j);
                double realPart = complexeOriginal.getReal() * Math.cos(theta) + complexeOriginal.getImag() * Math.sin(theta);
                double imagPart = -complexeOriginal.getReal() * Math.sin(theta) + complexeOriginal.getImag() * Math.cos(theta);
                Complexe a = new Complexe(realPart, imagPart); // Création du nombre complexe

                additionLocale = additionLocale.add(a); // Addition avec les nombres complexes précédents
            }
            if(sensTransformee == 1){ // Si sens transformé inverse
                additionLocale = additionLocale.multiply(1.0/N);
            }
            resultat.add(additionLocale);
        }
        return resultat;
    }
```

Notre méthode de transformée 1D est composée d'un tableau en une dimension qui stocke des nombres complexes précédemment vus. Nous avons également un paramètre sensTransformee :

- Si sensTransformee vaux -1 alors nous sommes en train de faire une Transformée de Fourier 1D.
- Si sensTransformee vaux +1 alors nous sommes en train de faire la Transformée de Fourier 1D Inverse.

Voici la formule de la transformée 1D et de l'inverse :

$$\hat{g}(u) = F(g(x)) = \sum_{x=0}^{N-1} g(x) * \exp\left(-\frac{2i\pi ux}{N}\right) \text{ avec u = 0..N-1.}$$

$$F(\hat{g}(u)) = g(x) = \frac{1}{N} * \sum_{u=0}^{N-1} \hat{g}(u) * \exp\left(\frac{2i\pi ux}{N}\right) \text{ avec x = 0..N-1.}$$

Nous pouvons donc voir que la différence entre ces deux formules est le signe du -2 dans l'exp() et le 1/N qui multiplie la somme.

Nous avons donc décidé de mettre la transformée inverse directement dans la même méthode que la transformée 1D.

La somme est représentée par une boucle for avec j allant de 0 à N – 1, de la même manière, la deuxième boucle for représente le parcours du tableau de complexes. :

```
1.   for(int j = 0 ; j < N ; j++) {//code }
```

Puisque nous utilisons la classe Complexe, nous avons besoin de décomposer la formule suivante :

$$\exp\left(\frac{2i\pi ij}{N}\right) \rightarrow \exp(\theta) \text{ avec } \theta = \left(\frac{sensTransformee * \pi ij}{N}\right)$$

$$\exp(\theta) \rightarrow Complexe(\cos(\theta) * ReelleTableau,\ \sin(\theta) * ImaginaireTableau)$$

- La partie réelle du complexe est donc cos(θ) * ReelleTableau
- La partie imaginaire sin(θ) * ImaginaireTableau.

Enfin, pour chacune des cases du tableau, si SensTransformee est dans le sens inverse, on divise la somme à mettre dans la case par la taille du tableau N (pareil que de multiplier par 1/N).

Transformée Rapide 1D

```java
public static ArrayList<Complexe> TransformeeRapide1D(ArrayList<Complexe> tableauR1D, int SensTransformee) throws Exception{
        // sensTransformee vaut -1 si on fait une transformée rapide 1D
        // sensTransformee vaut +1 si on fait une transformée rapide inverse 1D

        // Tableau temporaire pour inverser les valeurs du tableau
        ArrayList<Complexe> tableauTemp = InversionTableau(tableauR1D);

        int N = tableauTemp.size();

        if((N & (N - 1)) != 0){
            throw new Exception("Le tableau doit avoir comme taille une puissance de 2 !");
        }
        else{
            // On calcule maintenant la transforméeRapide1D
            tableauTemp = CalculTransformeeRapide1D(tableauTemp, SensTransformee);

            // Il faut ré-inverser l'ordre du tableau pour obtenir le résultat du calcul
            //ArrayList<Complexe> resultatTransformee = InversionTableau(tableauTemp);
            ArrayList<Complexe> resultatTransformee = tableauTemp;

            // Si le sens de la transformée est inverse, on divise chaque élément du tableau par la taille du tableau
            if(SensTransformee == 1){
                for(int i =0; i < N; i++){
                    Complexe c = new Complexe(N, 0);
                    resultatTransformee.set(i, resultatTransformee.get(i).divide(c));
                }
            }
            return resultatTransformee;
        }
}

private static ArrayList<Complexe> CalculTransformeeRapide1D(ArrayList<Complexe> tableauR1D, int SensTransformee){
        // Taille du tableau de complexe
        int tailleTableau = tableauR1D.size();

        // Création du tableau de résultat
        ArrayList<Complexe> tableauTransforme = new ArrayList<Complexe>(tableauR1D);

        if(tailleTableau != 1){ // Si la taille est de 1, on n'a pas à traiter le tableau
            ArrayList<Complexe> part1 = new ArrayList<Complexe>();
            ArrayList<Complexe> part2 = new ArrayList<Complexe>();
            for(int x = 0; x < tailleTableau / 2; x++){ // Initialisation du tableau
                part1.add(null);
                part2.add(null);
            }
            for(int i = 0; i < tableauR1D.size(); i++) {
                if (i % 2 != 0) { // Si rang impair
                    part1.set(i / 2, tableauR1D.get(i));
                }
                else { // Si rang pair
                    part2.set((i + 1 ) / 2, tableauR1D.get(i));
                }
            }
            // Appel récursif pour calcul de la transformée
            ArrayList<Complexe> resPart1 = CalculTransformeeRapide1D(part1, SensTransformee);
            ArrayList<Complexe> resPart2 = CalculTransformeeRapide1D(part2, SensTransformee);

            // On applique alors la formule de la transformée
            for(int u = 0; u < tailleTableau / 2; u++){
                double teta = SensTransformee * 2 * Math.PI * u / tailleTableau;
                Complexe coef = new Complexe(Math.cos(teta), Math.sin(teta));

                tableauTransforme.set(u,resPart1.get(u).add(coef.multiply(resPart2.get(u))));
                tableauTransforme.set(u + (tailleTableau / 2),resPart1.get(u).minus(coef.multiply(resPart2.get(u))));
            }
        }
        return tableauTransforme;
}

private static ArrayList<Complexe> InversionTableau(ArrayList<Complexe> tableau){
        int tailleTableau = tableau.size();
        ArrayList<Complexe> nouveauTableau = new ArrayList<Complexe>();
        for(int x = 0; x < tailleTableau; x++){ // Initialisation du tableau
            nouveauTableau.add(null);
        }
        for(int i = 0; i < tailleTableau ; i++){
            nouveauTableau.set(i,tableau.get((tailleTableau - 1) - i));
        }
        return nouveauTableau;
}
```

Notre méthode de transformée 1D Rapide est composée d'un tableau en une dimension qui stocke des nombres complexes précédemment vus. Nous avons également un paramètre sensTransformée comme la transformée 1D classique. Voici la formule

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

- TransformeeRapide1D → met en place le tableau et les données pour appel du calcul.
- InversionTableau → Inverse le sens du tableau (on le lit du dernier index au premier).
- CalculTransformeeRapide1D → effectue récursivement les calculs de la transformée rapide.

TransformeeRapide1D :

La transformée rapide 1D ne peut s'effectuer (facilement) qu'avec des tailles de tableau de taille 2^n. Nous faisons alors une comparaison bit à bit entre N et N-1, si cette comparaison renvoie autre chose que 0, alors nous ne sommes pas sur une puissance de 2 et on lève une Exception :

```
1.  if( (N & (N - 1)) != 0)
2.  {
3.      throw new Exception("Le tableau doit avoir comme taille une puissance de 2 !");
4.  }
```

Nous pouvons découper la formule en deux parties, la première partie à gauche du « + » désigne la partie avec les rangs paire du tableau (en rouge), la seconde partie désigne les rangs impairs du tableau (en violet). On effectuera donc sur chaque branche (impaire et paire) les calculs. Ceux-ci sont récursifs, on appellera donc la méthode CalculTransformeeRapide1D dans elle-même.

```
1.  for(int i = 0; i < tableauR1D.size(); i++) {
2.     if (i % 2 != 0) { // Si rang impair
3.        part1.set(i / 2, tableauR1D.get(i));
4.     }
5.     else { // Si rang pair
6.        part2.set((i + 1 ) / 2, tableauR1D.get(i));
7.     }
8.  }
9.  ArrayList<Complexe> resPart1 = CalculTransformeeRapide1D(part1, SensTransformee);
10. ArrayList<Complexe> resPart2 = CalculTransformeeRapide1D(part2, SensTransformee);
```

CalculTransformeeRapide1D :

Pour voir le fonctionnement, nous allons faire l'algorithme à la main avec la formule suivante :

$$X(u) = \sum_{x=0}^{N=4} tf(x) * \exp\left(\frac{-2i\pi uZ}{N}\right)$$

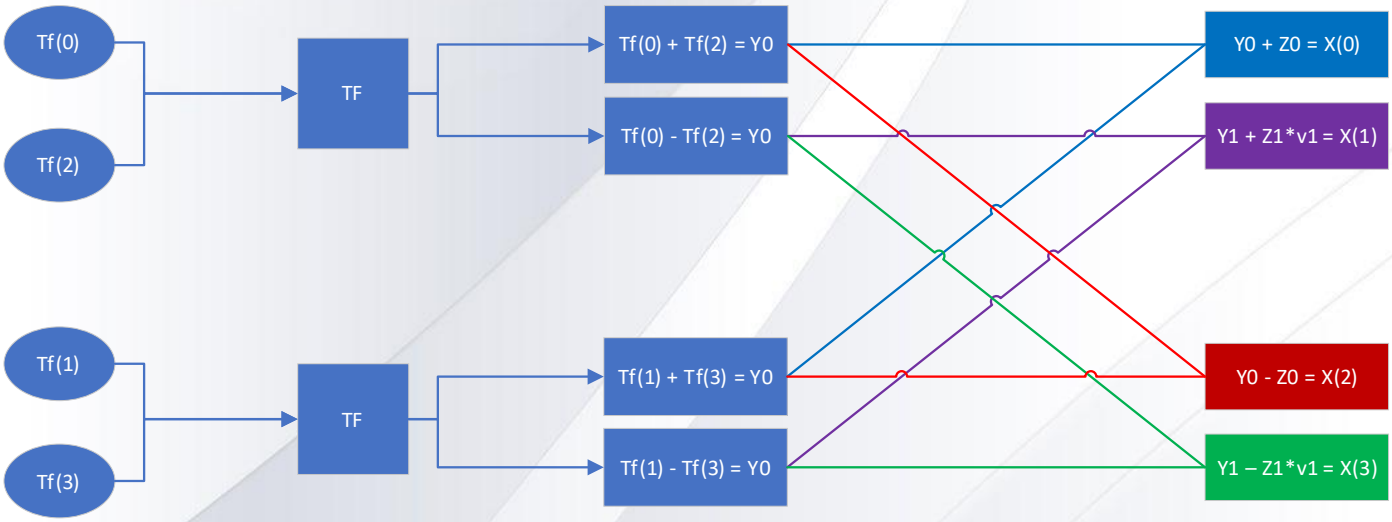
un tableau de taille **N = 4**. Prenons un tableau 1D avec pour valeurs **[1,3,2,0]**. **n** est la valeur qui parcours le tableau du rang 0 à N -1. On prend alors v comme variable pour la formule $\exp\left(\frac{-2i\pi uZ}{\frac{N}{2}}\right)$.

On peut donc déterminer les sommes à la main :

```
1.  X(n=0) = tf(0) + tf(1) + tf(2) + tf(3)
2.  X(1) = tf(0) + tf(1)*v1 + tf(2)*v2 + tf(3)*v3
3.  X(2) = tf(0) + tf(1)*v2 + tf(2)*v4 + tf(3)*v6
4.  X(3) = tf(0) + tf(1)*v3 + tf(2)*v6 + tf(3)*v9
```

Le pas entre les différents exposants de w correspond avec la valeur n de X(n).

On applique alors la Transformée de Fourier sur les rangs paires et impaires ce qui nous donne :



Transformée 2D

```java
public static ArrayList<ArrayList<Complexe>> Transformee2D(ArrayList<ArrayList<Complexe>> tableau2D, int sensTransformee) {
        // Récupération de la taille du tableau
        int X = tableau2D.size();
        int Y = tableau2D.get(0).size();

        // Création du tableau de résultat et initialisation de ses colonnes
        ArrayList<ArrayList<Complexe>> resultat = new ArrayList<ArrayList<Complexe>>();
        for (int i = 0; i < X; i++){
            resultat.add(new ArrayList<Complexe>());
        }

        for (int i = 0; i < X; i++) {  //i correspond au rang horizontal du prochain résultat
            for (int j = 0 ; j < Y; j++) {  //j correspond au rang vertical du prochain résultat
                // Création de la variable de somme des éléments du tableau
                Complexe additionLocale = new Complexe(0, 0);
                for (int k = 0; k < X; k++) {  //horizontal
                    for (int l = 0; l < Y; l++) {  //vertical
                        double theta = (sensTransformee * 2.0 * Math.PI * i * j * k * l) / (X*Y); // Angle thêta

                        final Complexe complexeOriginal = tableau2D.get(k).get(l);
                        double realPart = complexeOriginal.getReal() * Math.cos(theta) + complexeOriginal.getImag() * Math.sin(theta);
                        double imagPart =-complexeOriginal.getReal() * Math.sin(theta) + complexeOriginal.getImag() * Math.cos(theta);
                        Complexe a = new Complexe(realPart, imagPart); // Création du nombre complexe

                        additionLocale = additionLocale.add(a); // Addition avec les nombres complexes précédents
                    }
                }
                if(sensTransformee == 1){ // Si sens transformé inverse
                    additionLocale = additionLocale.multiply(1.0/(X*Y));
                }
                resultat.get(i).add(additionLocale);
            }
        }
        return resultat;
    }
```

La transformée 2D met en place la même formule que la 1D, mais avec deux boucles supplémentaires afin de couvrir la deuxième dimension du tableau et la deuxième dimension de calcul.

Sources utilisées

https://perso.esiee.fr/~bercherj/New/polys/poly_tfd.pdf → Cours de transformée d'une école d'ingénieur.
http://adrian.gaudebert.fr/downloads/licence3/maths/rapport-despret-gaudebert.pdf → Un élève ayant travaillé sur un projet similaire
http://jl.baril.u-bourgogne.fr/c2.pdf → Le cours de la faculté sur les transformée de Fourier