

# **TP 5 - Approximation par des méthodes de Splines Cubique**

Réalisé par :

- Mathéo BEAUDOUIN

Encadré par :

- Mr. Cyrille Bertelle

# I) Introduction

Dans le cadre du TP5, nous abordons l'interpolation par splines cubiques. Il s'agit d'une technique utilisée pour approximer une fonction inconnue à partir d'un ensemble de points de support. L'objectif principal de ce TP est de construire une classe Spline, capable de manipuler cette méthode.

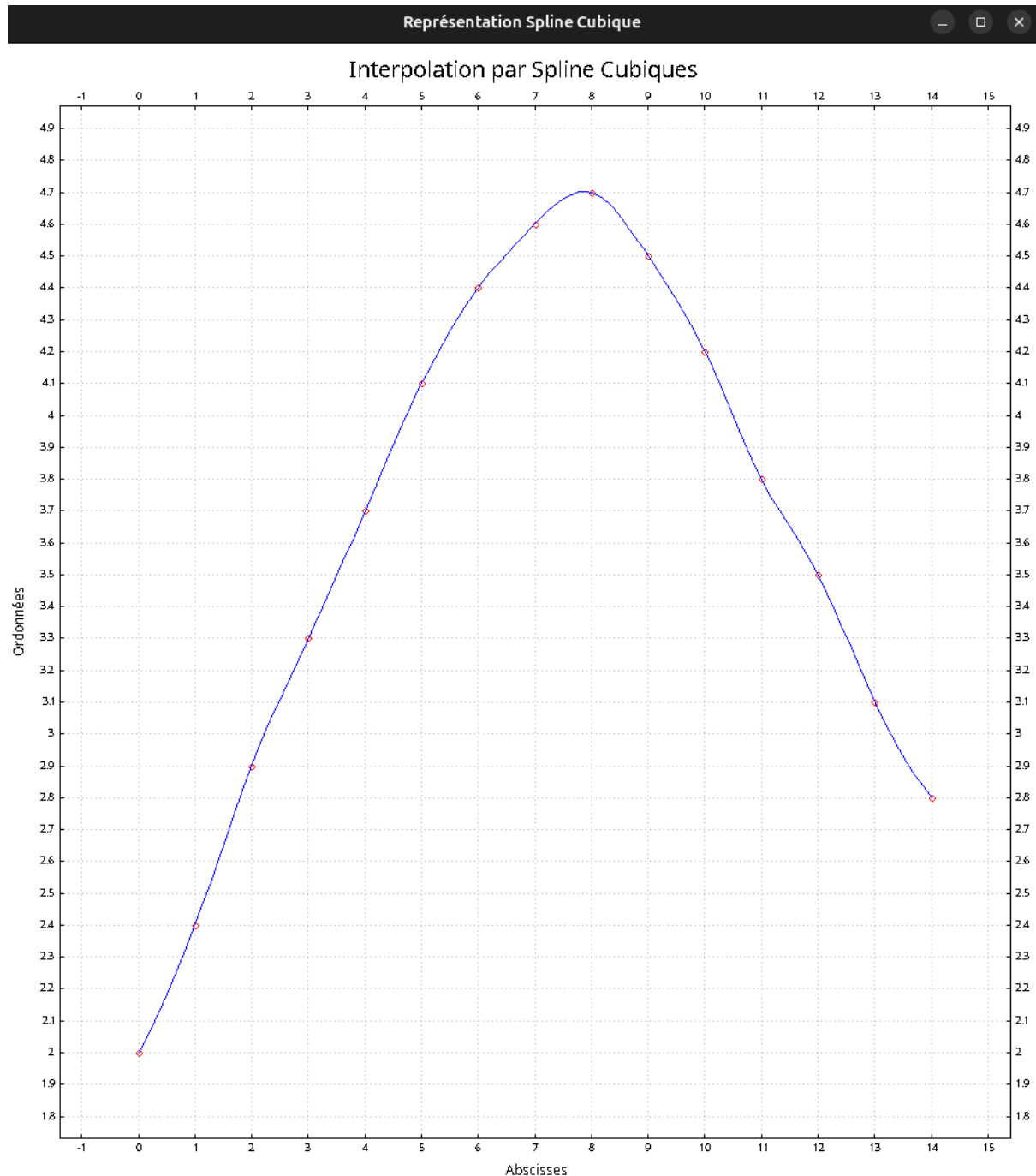
Chaque instance de cette classe est définie à partir d'un tableau de points de support, et une méthode interne permet de calculer les dérivées secondes nécessaires à l'interpolation. Une autre méthode publique permet ensuite d'évaluer la fonction interpolée en un point donné, tout en s'assurant que la valeur est bien comprise dans l'intervalle des abscisses fournies.

Pour tester et visualiser cette interpolation, nous avons développé un programme principal capable de lire les points de support à partir d'un fichier fourni par l'utilisateur, puis de déterminer l'intervalle des abscisses concerné. Il génère ensuite 100 points répartis uniformément sur cet intervalle et évalue leur image à l'aide de la fonction interpolée. Enfin, l'affichage graphique des résultats est réalisé grâce à la bibliothèque UP, qui permet de tracer la courbe interpolée tout en représentant distinctement les points de support.

Ce rapport présente les résultats obtenus ainsi que la visualisation graphique de l'interpolation.

## II ) Résultats et tests

Après avoir implémenté la classe Spline et son programme principal, nous avons effectué plusieurs tests afin de vérifier la justesse de notre interpolation. En entrant un fichier contenant un ensemble de 15 points de support, nous avons obtenu la représentation graphique suivante :



La courbe obtenue montre clairement que l'interpolation par splines cubiques passe au plus près des points de support. Contrairement à l'approximation polynomiale, cette méthode évite l'effet d'oscillation excessive aux extrémités, ce qui permet une approximation plus stable et plus précise des données.

Grâce à l'utilisation de la bibliothèque UP, la représentation visuelle permet de constater que les points de support sont bien respectés.

Ce test confirme ainsi le bon fonctionnement de notre implémentation et valide la pertinence de l'interpolation par splines cubiques.

## III) Conclusion

Ce travail nous a permis d'explorer l'interpolation par spline cubique et de comprendre ses mécanismes. L'affichage graphique a été un atout pour visualiser les résultats et identifier d'éventuelles erreurs pendant notre implémentation. Une fois ces erreurs corrigées, nous avons pu constater la précision de l'interpolation obtenue, confirmant ainsi l'efficacité de notre approche.

## ANNEXE

```
package AlgLin;
import java.awt.Color;
import java.awt.Container;
import java.awt.GridLayout;
import java.awt.Toolkit;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import javax.swing.*;
import org.lucci.up.*;
import org.lucci.up.data.*;
import org.lucci.up.data.math.*;
import org.lucci.up.data.rendering.*;
import org.lucci.up.data.rendering.figure.ConnectedLineFigureRenderer;
import org.lucci.up.data.rendering.figure.FigureRenderer;
import org.lucci.up.data.rendering.point.HistogramPointRenderer;
import org.lucci.up.data.rendering.point.OriginPointConnectedPointRenderer;
import org.lucci.up.data.rendering.point.PointRenderer;
import org.lucci.up.data.rendering.point.TextPointRenderer;
import org.lucci.up.system.*;
import org.lucci.up.data.rendering.point.PointAsDotRenderer;
public class Spline {

    // les abscisses et ordonnées des points de support
    private double[] x;
    private double[] y;

    //pour la derive seconde
    private Vecteur derive;
    // la constructeur et son tableau de point de support
    public Spline(double[] x, double[] y) throws IrregularSysLinException {
        if (x.length != y.length) {
            throw new IllegalArgumentException("les deux tableaux doivent etre
de meme taille");
        }
        if (x.length < 2) {
            throw new IllegalArgumentException("on a besoin d au moins
2 points de supports");
        }
        this.x = x;
        this.y = y;
        calculerDerriveeSeconde();
    }

    //méthode pour calculer les dérivées secondes aux points de support
    private void calculerDerriveeSeconde() throws IrregularSysLinException {
        int n = x.length;
        double[] a = new double[n]; // coeff sous-diagonaux ai
```

```

double[] b = new double[n]; // coeff de la diagonale principale bj
double[] c = new double[n]; // coeff sur-diagonaux cj
double[] d = new double[n]; // second membre dj
//debut de l'algo du cours
b[0] = 2 * (x[2] - x[0]);
c[0] = x[2] - x[1];
d[0] = 6 * ((y[2] - y[1]) / (x[2] - x[1]) - (y[1] - y[0]) / (x[1] -
x[0]));
for (int j = 1; j <= n - 2; j++) {
    a[j] = c[j - 1];
    b[j] = 2 * (x[j + 1] - x[j - 1]);
    c[j] = x[j + 1] - x[j];
    d[j] = 6 * ((y[j + 1] - y[j]) / c[j] - (y[j] - y[j - 1]) / a[j]);
}
a[n - 1] = c[n - 2];
b[n - 1] = 2 * (x[n - 1] - x[n - 3]);
d[n - 1] = 6 * ((y[n - 1] - y[n - 2]) / (x[n - 1] - x[n - 2]) - (y[n -
2] - y[n - 3]) / a[n-2]);
//on construit le systeme tridiagonal
double[][] coeffDiag = new double[][] {a, b, c};
Mat3Diag matTriDiag = new Mat3Diag(coeffDiag);
Vecteur v = new Vecteur(d);
Thomas thomas = new Thomas(matTriDiag, v);

//on le resoud par la methode de Thomas
this.derive = thomas.resolution();
}

//méthode pour évaluer la spline en une valeur donnée
public double evaluer(double valeur) throws DataOutOfRangeException{
    // on s assure que la valeur est dans le bon intervalle
    if (valeur < x[0] || valeur > x[x.length - 1]) {
        throw new DataOutOfRangeException("la valeur n'est pas dans le
bon intervalle");
    }
    // Recherche de l'intervalle [xj; xj+1] contenant la valeur
    int j = 0;
    while (j < x.length - 1 && valeur > x[j + 1]) {
        j++;
    }
    //calcul des paramètres utilisés dans la formule de l'interpolation
    double alpha = x[j + 1] - valeur;
    double beta = valeur - x[j];
    double gamma = x[j + 1] - x[j];
    // calcul des termes de la formule d'interpolation (suivant la
decomposition du cours)
    double t1 = derive.getCoef(j) / 6.0 * ((Math.pow(alpha, 3) / gamma) -
gamma * alpha);
    double t2 = derive.getCoef(j + 1) / 6.0 * ((Math.pow(beta, 3) / gamma)
- gamma * beta);
    double t3 = y[j] * (alpha / gamma) + y[j + 1] * (beta / gamma);
    return t1 + t2 + t3;
}

```

```

// Affichage du graphe
private static void afficheGraphe(double[] x, double[] y, double[]
xInterpolé, double[] yInterpolé) {

    // creation et configuration de la fenêtre au centre de l'écran
    JFrame frame = new JFrame("Représentation Spline Cubique");
    java.awt.Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
    int side = (int) (screenSize.getHeight() * 0.5);
    frame.setSize(side, side);
    frame.setLocation((int) (screenSize.getWidth() - side) / 2, (int)
(screenSize.getHeight() - side) / 2);
    Container contentPane = frame.getContentPane();
    contentPane.setLayout(new GridLayout(1, 1));

    // création des figures pour les points et la courbe interpolée
    Figure pointsSupport = new Figure();
    for (int i = 0; i < x.length; i++) {
        pointsSupport.addPoint(new Point(x[i], y[i]));
    }
    DataElementRenderer pointRenderer = new PointAsDotRenderer();
    pointRenderer.setColor(Color.RED);
    pointsSupport.addRenderer(pointRenderer);

    Figure courbeInterpolée = new Figure();
    for (int i = 0; i < xInterpolé.length; i++) {
        courbeInterpolée.addPoint(new Point(xInterpolé[i],
yInterpolé[i]));
    }
    DataElementRenderer courbeRenderer = new
ConnectedLineFigureRenderer();
    courbeRenderer.setColor(Color.BLUE);
    courbeInterpolée.addRenderer(courbeRenderer);
    // ajout des figures à une figure globale
    Figure figureList = new Figure();
    figureList.addFigure(pointsSupport);
    figureList.addFigure(courbeInterpolée);
    // création du plotter
    SwingPlotter plotter = new SwingPlotter();
    plotter.getGraphics2DPlotter().setFigure(figureList);
    Space space = plotter.getGraphics2DPlotter().getSpace();
    space.setMode(Space.PHYSICS);
    space.setBackground(Color.WHITE);
    space.setColor(Color.BLACK);
    space.getLegend().setText("Interpolation par Spline Cubiques");
    space.getXDimension().getLegend().setText("Abscisses");
    space.getYDimension().getLegend().setText("Ordonnées");
    // ajout du plotter à la fenêtre
    contentPane.add(plotter);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
}

```

```

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Entrez le nom du fichier contenant les points de
support :");
        String nomFichier = scanner.nextLine();
        scanner.close();
        // les listes qui contiendra les points de supports
        List<Double> xList = new ArrayList<>();
        List<Double> yList = new ArrayList<>();
        // lecture du fichier contenant les points
        try (Scanner fileScanner = new Scanner(new File(nomFichier))) {
            while (fileScanner.hasNextLine()) {
                String line = fileScanner.nextLine().trim();
                if (!line.isEmpty()) {
                    String[] parts = line.split("\\s+"); // on sépare les
valeurs par espaces
                    if (parts.length == 2) { // on vérifie que la ligne
contient deux valeurs, un x et un y
                        double xVal = Double.parseDouble(parts[0]);
                        double yVal = Double.parseDouble(parts[1]);
                        xList.add(xVal);
                        yList.add(yVal);
                        System.out.println("Point lu : (" + xVal + ", " + yVal
+ ")");
                    } else {
                        System.err.println("Ligne ignorée : " + line);
                    } // on s'assure que tous les points sont lu, on affiche
ainsi chaque points
                }
            }
        } catch (FileNotFoundException e) {
            System.err.println("Fichier introuvable : " + nomFichier);
            return; // on arrête le programme si le fichier est manquant
        }
        //on vérifie qu'il y a au moins deux points pour l'interpolation
        if (xList.size() < 2) {
            System.err.println("Erreur : il faut au moins 2 points");
            return;
        }
        // conversion en des listes en tableaux
        double[] x =
xList.stream().mapToDouble(Double::doubleValue).toArray();
        double[] y =
yList.stream().mapToDouble(Double::doubleValue).toArray();
        try {

            Spline spline = new Spline(x, y);

            // Génération des points interpolés
            int nbPoints = 100;
            double[] xInterpolé = new double[nbPoints];
            double[] yInterpolé = new double[nbPoints];

```



```
// on calcul l'intervalle des abscisses
double minX = x[0];
double maxX = x[x.length - 1];
double pas = (maxX - minX) / (nbPoints - 1);
// calcul des valeurs interpolées
for (int i = 0; i < nbPoints; i++) {
    xInterpolé[i] = minX + i * pas;
    yInterpolé[i] = spline.evaluer(xInterpolé[i]);
}
// Affichage du graphe
afficheGraphe(x, y, xInterpolé, yInterpolé);
} catch (IrregularSysLinException e) {
    System.err.println("Erreur lors du calcul de la spline : " +
e.getMessage());
} catch (DataOutOfRangeException e) {
    System.err.println("Erreur : valeur en dehors de l'intervalle.");
}
}
```