

TP6 – Méthode des moindres carrées

Réalisé par :

- Mathéo BEAUDOUIN

Encadré par :

- Mr. Cyrille Bertelle

I) Introduction

Dans le cadre du TP6, nous explorons l'implémentation de la méthode des moindres carrés en Java au sein du package AlgLin. Ce TP a pour objectif de déterminer un modèle polynomial permettant d'approximer un ensemble de points de support.

Pour cela, nous avons développé la classe ModPoly, qui représente ce modèle et ajuste ses coefficients à l'aide de la méthode identifiée, basée sur le critère des moindres carrés.

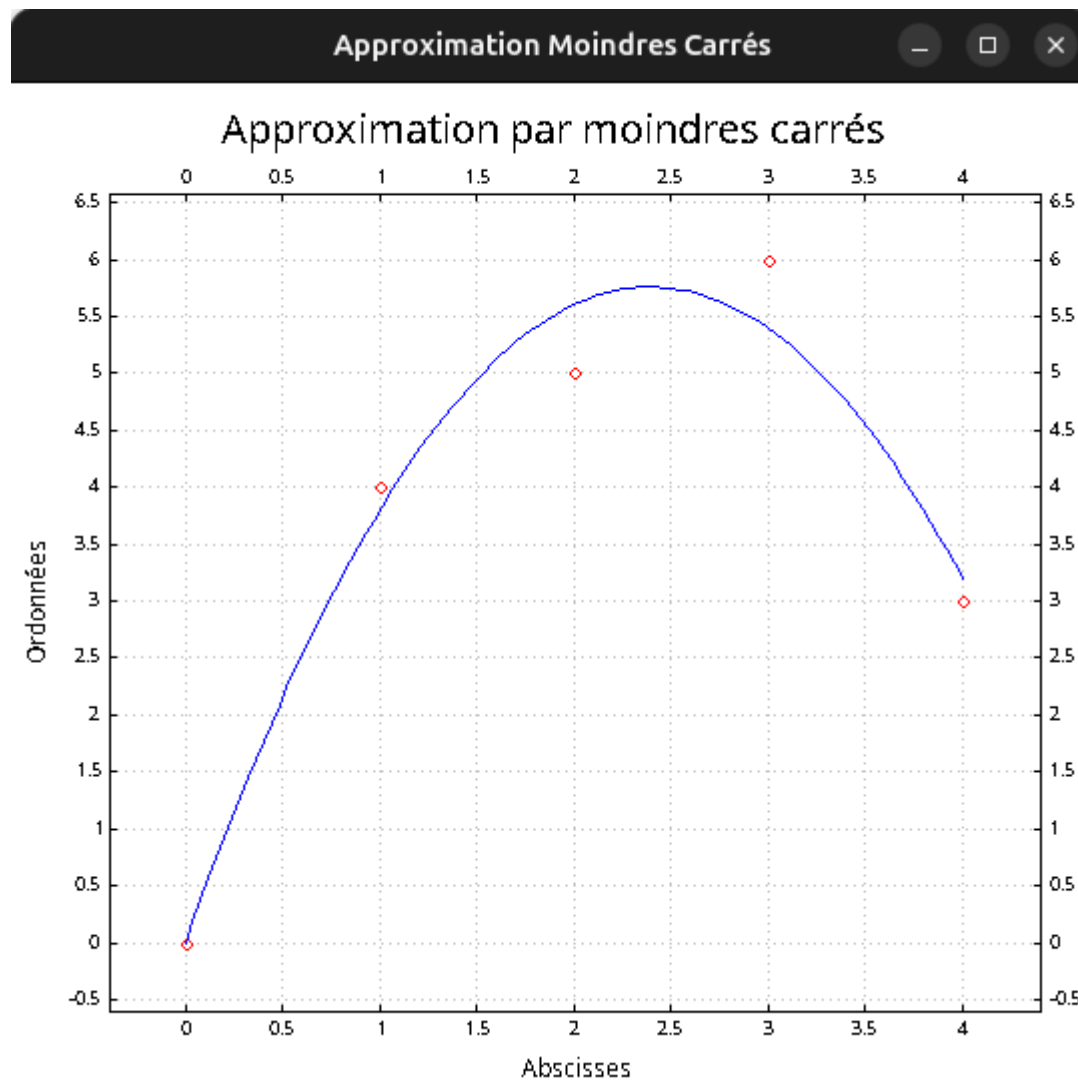
Dans ce rapport, nous présenterons les visualisations générées avec la bibliothèque UP des approximations obtenues pour des polynômes de degrés différents. Ces représentations nous permettront de comparer la courbe obtenue avec les points de données initiaux.

II) Résultats et tests

a) Première visualisation, polynôme de degré 2

Pour tester l'implémentation de notre méthode d'approximation par moindres carrés, nous avons utilisé un ensemble de points issus de l'exercice 15 du TD5. Le fichier est "ptsTp6.txt". Nous avons choisi d'approximer ces points à l'aide d'un polynôme de degré 2

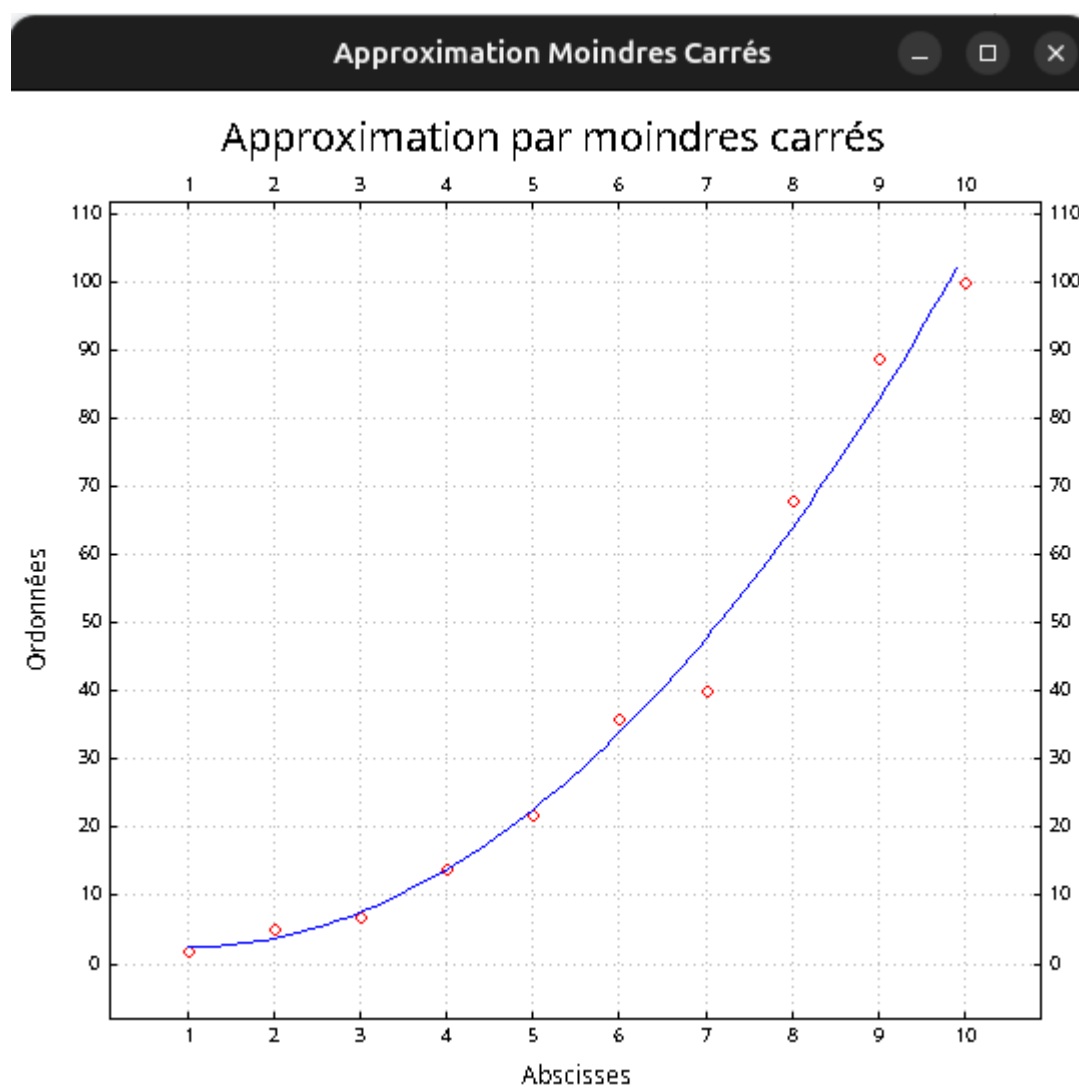
Voici la courbe obtenue :



En observant la figure obtenue, on remarque que les points de support sont représentés en rouge, tandis que la courbe ajustée est tracée en bleu. On observe aussi qu'elle épouse au mieux leur trajectoire tout en respectant la contrainte du degré du polynôme. Comme prévu pour une approximation par un polynôme de degré 2, la forme générale de la courbe est une parabole.

Il est intéressant de noter que la courbe ne passe pas nécessairement par tous les points, ce qui est une caractéristique normale de l'approximation par moindres carrés. Cette méthode ne cherche pas à interpoler exactement chaque point, mais plutôt à minimiser l'écart global entre la courbe et l'ensemble des points de support.

Un autre test, avec le fichier "pts2Tp6.txt" a été réalisé, toujours avec un polynôme de degré 2, cette fois en utilisant d'autres points de support. Voici la courbe obtenue :

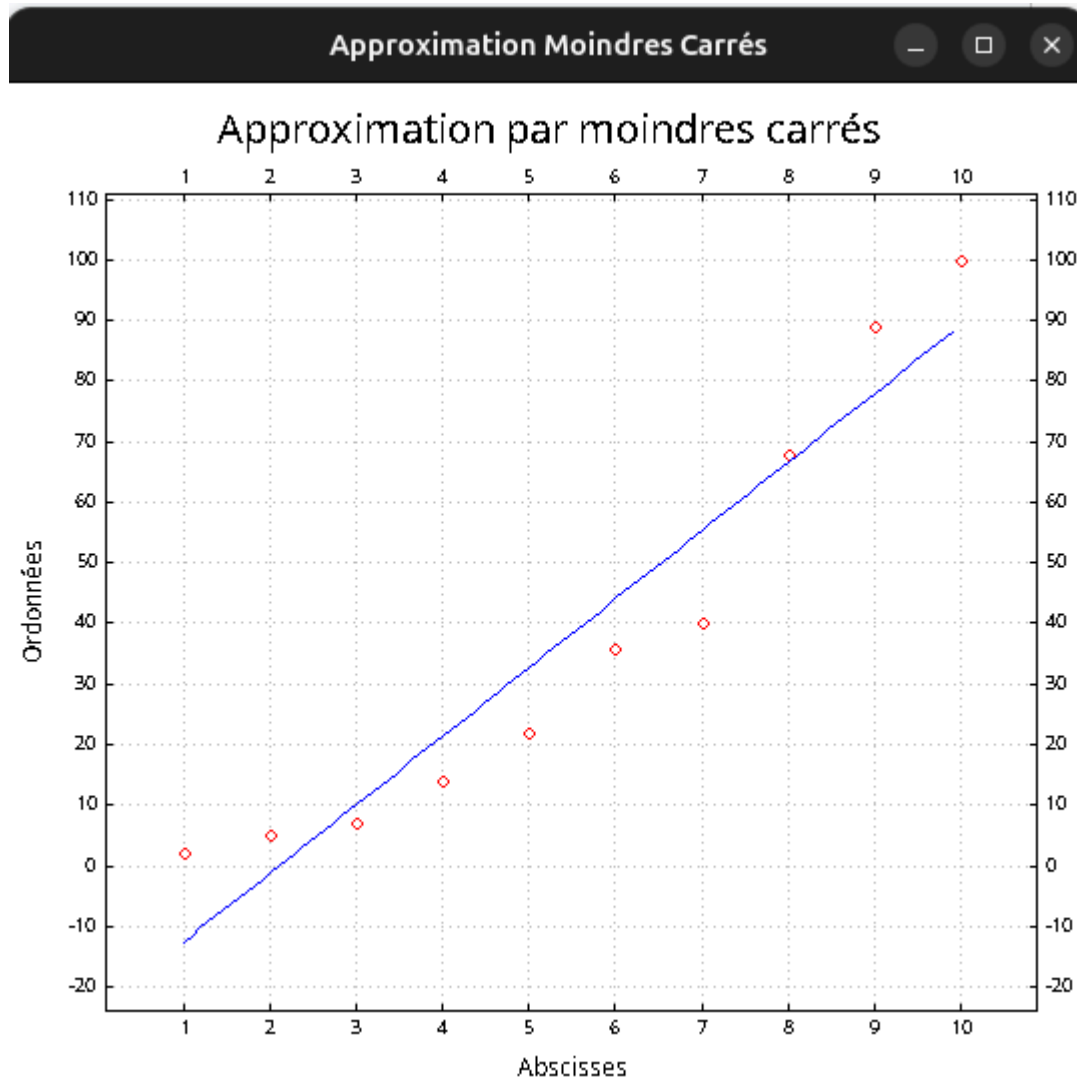


Ce second test confirme encore une fois nos propos précédents, cela montre la cohérence de notre méthode d'approximation.

b) Deuxième visualisation, polynôme de degré 1

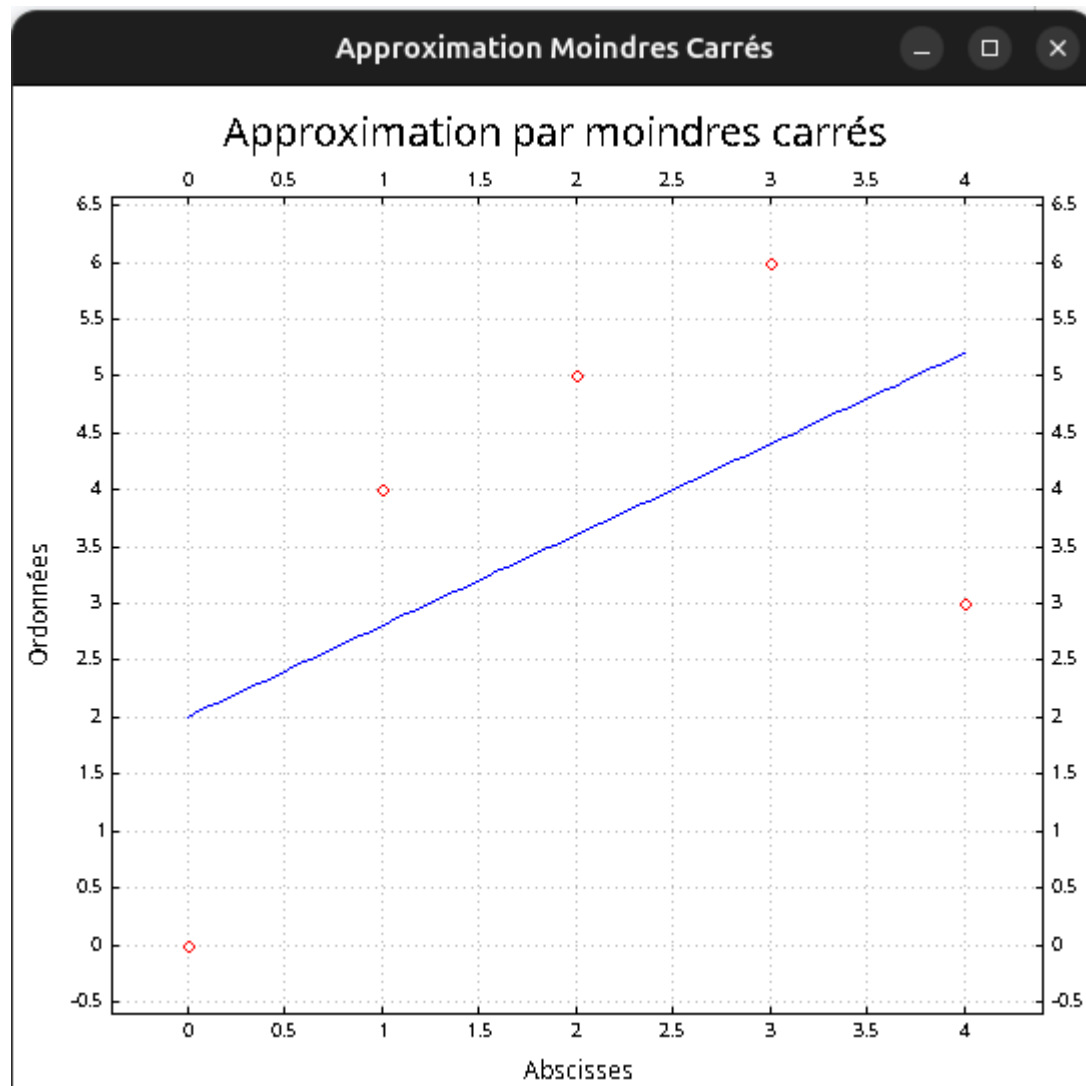
Nous avons maintenant utilisé le fichier pts2Tp6.txt afin de réaliser une approximation par un polynôme de degré 1.

Voici la courbe obtenue :



En observant la figure obtenue, on remarque que les points de support, sont bien dispersés autour de la courbe ajustée. Cette courbe suit une trajectoire linéaire qui minimise l'écart global entre les points et la droite obtenue. Contrairement à la première approximation avec un polynôme de degré 2, la forme de la courbe est ici une simple droite, ce qui est cohérent avec le degré choisi.

Un autre test, avec le premier fichier "ptsTp6.txt" a été réalisé, cette fois en utilisant d'autres points de support. Voici la courbe obtenue :



Sur cette courbe, on observe que la droite ajustée ne passe par aucun des points de support.

c) Comparaison de l'efficacité de la méthode des moindres carrés selon le degré du polynôme

L'analyse des deux visualisations met en évidence l'impact du choix du degré du polynôme sur l'approximation des points. Une approximation de degré 1 fonctionne bien pour des points de support suivant une tendance linéaire, mais peut être insuffisante si la répartition des points présente une courbure. En revanche, un polynôme de degré 2 s'adapte mieux aux variations non linéaires.

III) Conclusion

Ce TP6 nous a permis d'approfondir l'approximation par moindres carrés en développant la classe ModPoly du package AlgLin. Les tests réalisés ont mis en évidence l'impact du degré du polynôme sur la précision de l'approximation. Un polynôme de degré 2 s'adapte mieux aux variations des points de support qu'un polynôme de degré 1.

Grâce aux visualisations, nous avons pu identifier et corriger nos erreurs d'implémentation. Pour finir, ce travail nous a offert une application des notions vues en cours et en TD.

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.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.Scanner;

import javax.swing.JFrame;

import org.lucci.up.SwingPlotter;
import org.lucci.up.data.Figure;
import org.lucci.up.data.Point;
import org.lucci.up.data.rendering.DataElementRenderer;
import org.lucci.up.data.rendering.figure.ConnectedLineFigureRenderer;
import org.lucci.up.data.rendering.point.PointAsDotRenderer;
import org.lucci.up.system.Space;

public class ModPoly {

    private Vecteur coefficients; // vecteur pour les coefficients
    private Matrice fonctionBase; // matrice pour les fonctions de base

    public ModPoly(int deg, int nbPoints) {
        if (deg < 0 || nbPoints < 0) {
            throw new IllegalArgumentException("le degre du polynome et le nombre de points
de support doivent etre positifs ou nuls");
        }
        // Initialisation du vecteur des coefficients du polynome
        coefficients = new Vecteur(deg + 1);
        // Initialisation de la matrice des fonctions de base
        fonctionBase = new Matrice(nbPoints, deg + 1);
    }

    //Calcul des coefficients du polynome d'approximation
    public void identifie(double[] xSupport, double[] ySupport) throws
IrregularSysLinException {
        // on recupere le degre m du polynome
```



```

        int m = coefficients.taille() - 1;
        // on verifie que les tailles des tableaux sont coherentes
        if (xSupport.length != ySupport.length) {
            throw new IllegalArgumentException("les tableaux xSupport et ySupport doivent etre
de mm longueur");
        }
        if (m < 0) {
            throw new IllegalArgumentException("le degre du polynome doit être un entier positif
ou nul");
        }
        remplirMatriceFonctionBase(xSupport, m);
        coefficients = calculerCoefficients(ySupport);
    }

    // remplissage de la matrice des fonctions de base F
    private void remplirMatriceFonctionBase(double[] xSupport, int m) {
        for (int i = 0; i < xSupport.length; i++) {
            for (int j = 0; j <= m; j++) {
                fonctionBase.remplacecoef(i, j, Math.pow(xSupport[i], j));
            }
        }
    }

    // methode pour resoudre le systeme
    private Vecteur calculerCoefficients(double[] ySupport) throws
IrregularSysLinException {
        // transposé de la fonction de base
        Matrice transpose = fonctionBase.transpose();
        // (F^(T)*F) produit de la transpose de F avec F
        Matrice FTF = Matrice.produit(transpose, fonctionBase);
        Vecteur y = new Vecteur(ySupport);
        // (F^(T)*Y) produit de la transposee de F et du vecteur y
        Vecteur FTY = Vecteur.produitMatriceVecteur(transpose, y);
        Helder syslin = new Helder(FTF, FTY);
        // on resoud le systeme et on obtient les coefficients du polynome
        return syslin.resolution();
    }

    //evalue le polynôme en un point
    private static double evaluerPolynome(Vecteur coefficients, double x) {
        double resultat = 0;
        // parcours des coefficients
        for (int i = 0; i < coefficients.taille(); i++) {
            resultat += coefficients.getCoef(i) * Math.pow(x,i);
        }
        return resultat;
    }

```

```
// accesseurs pour les coefficients et la matrice des fonctions de base
public Vecteur getCoefficients() {
    return coefficients;
}

public Matrice getFonctionBase() {
    return fonctionBase;
}

// les méthodes pour tracer la courbe du polynome d approximation

// methode qui creer la figure avec les points de support en rouge
private Figure creerPointsFigure(double[] xSupport, double[] ySupport) {
    Figure pointsFigure = new Figure();
    for (int i = 0; i < xSupport.length; i++) {
        pointsFigure.addPoint(new Point(xSupport[i], ySupport[i]));
    }
    DataElementRenderer pointsRenderer = new PointAsDotRenderer();
    pointsRenderer.setColor(Color.RED);
    pointsFigure.addRenderer(pointsRenderer);
    return pointsFigure;
}

// tracé de la courbe polynomiale bleu
private Figure creerCourbeFigure(double[] xSupport) {
    Figure courbeFigure = new Figure();
    double minX = xSupport[0];
    double maxX = xSupport[xSupport.length - 1];
    int nbPoints = 100;
    double pas = (maxX - minX) / (nbPoints - 1);
    for (double x = minX; x <= maxX; x += pas) {
        courbeFigure.addPoint(new Point(x, evaluatePolynome(coefficients, x)));
    }
    DataElementRenderer courbeRenderer = new ConnectedLineFigureRenderer();
    courbeRenderer.setColor(Color.BLUE);
    courbeFigure.addRenderer(courbeRenderer);
    return courbeFigure;
}

private void afficherGraphique(Figure pointsFigure, Figure courbeFigure) {
    //ajout des figures au graphe
    Figure figureGlobale = new Figure();
    figureGlobale.addFigure(pointsFigure);
    figureGlobale.addFigure(courbeFigure);

    // configuration de l'affichage
```

```

        SwingPlotter plotter = new SwingPlotter();
        plotter.getGraphics2DPlotter().setFigure(figureGlobale);
        Space space = plotter.getGraphics2DPlotter().getSpace();
        space.setMode(Space.PHYSICS);
        space.setBackgroundColor(Color.WHITE);
        space.setColor(Color.BLACK);
        space.getLegend().setText("Approximation par moindres carrés");
        space.getXDimension().getLegend().setText("Abscisses");
        space.getYDimension().getLegend().setText("Ordonnées");

        // creation de la fenetre
        JFrame frame = new JFrame("Approximation Moindres Carrés");
        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));
        contentPane.add(plotter);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

    //methode qui appelle les 3 methodes précédentes pour bien tracer la courbe
    public void tracerCourbe(double[] xSupport, double[] ySupport) {
        Figure pointsFigure = creerPointsFigure(xSupport, ySupport);
        Figure courbeFigure = creerCourbeFigure(xSupport);
        afficherGraphique(pointsFigure, courbeFigure);
    }

    // methode pour lire le fichier et recuperer les points de support
    public static double[][] lireFichier(String nomFichier) {
        // 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 separe les valeurs par des espaces
                    if (parts.length == 2) { //on verifie que la ligne contient deux valeurs, un x et
un y
                        try {
                            xList.add(Double.parseDouble(parts[0]));

```

```
        yList.add(Double.parseDouble(parts[1]));
    } catch (NumberFormatException e) {
        System.err.println("Erreur de format, ligne ignorée : " + line);
    }
    } 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 null; // on retourne nul si le fichier est manquant
}

//on vérifie qu'il y a au moins deux points
if (xList.size() < 2) {
    System.err.println("Erreur : Il faut au moins 2 points pour l'interpolation.");
    return null;
}

// on converti en tableaux les listes qui contiendra les points de supports
double[] xSupport = xList.stream().mapToDouble(Double::doubleValue).toArray();
double[] ySupport = yList.stream().mapToDouble(Double::doubleValue).toArray();

// on retourne le tableaux de points
return new double[][]{xSupport, ySupport};
}

//une fois qu'on a nos points on va s'assurer qu'ils sont bien dans l'ordre suivant les x
// si jamais l'utilisateur entre des x non croissant
public static double[][] trierPoints(double[] x, double[] y) {
    int n = x.length;
    Double[][] points = new Double[n][2];

    for (int i = 0; i < n; i++) {
        points[i][0] = x[i];
        points[i][1] = y[i];
    }

    // Tri des indices par ordre croissant de x
    Arrays.sort(points, Comparator.comparingDouble(a -> a[0]));

    // Création des nouveaux tableaux triés
    double[] xTrie = new double[n];
    double[] yTrie = new double[n];
    for (int i = 0; i < n; i++) {
```

```
xTrie[i] = points[i][0];
yTrie[i] = points[i][1];
}

return new double[][]{xTrie, yTrie};
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    try {
        // Lecture du fichier
        System.out.print("Entrez le nom du fichier contenant les points de support : ");
        String nomFichier = scanner.nextLine();
        double[][] points = lireFichier(nomFichier);
        if (points == null) return; // on arrete le programme s'il ny a pas de points

        // Tri des points
        double[][] pointsTries = trierPoints(points[0], points[1]);
        double[] xSupport = pointsTries[0];
        double[] ySupport = pointsTries[1];

        // Demande du degré du polynome
        System.out.print("Entrez le degré du polynôme : ");
        int degre = scanner.nextInt();

        // Création et affichage du polynôme
        try {
            ModPoly modPoly = new ModPoly(degre, xSupport.length);
            modPoly.identifie(xSupport, ySupport);
            modPoly.tracerCourbe(xSupport, ySupport);
        } catch (IrregularSysLinException e) {
            System.err.println("Erreur lors du calcul des moindres carrés : " +
e.getMessage());
        }
        } catch (Exception e) {
            System.err.println("Une erreur inattendue s'est produite : " + e.getMessage());
        } finally {
            scanner.close();
        }
    }
}
```