Glody KUTUMBAKANA

Compte Rendu ModPoly

Année 2024 -2025

Glody KUTUMBAKANA

Introduction

Dans ce TP on nous demande d’implémente un modèle polynomial pour ajuster un ensemble de points à l'aide d'une méthode d'interpolation polynomial. L'objectif est de trouver un polynôme de degré spécifié qui approxime au mieux un ensemble de points de support donnés dans un fichier externe.

Code :

Class ModPoly

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.Arrays;
import java.util.Scanner;

import org.knowm.xchart.SwingWrapper;
import org.knowm.xchart.XYChart;
import org.knowm.xchart.XYChartBuilder;
import org.knowm.xchart.style.lines.SeriesLines;
import org.knowm.xchart.style.markers.SeriesMarkers;

public class ModPoly {
private double[] coef;
private int taille;
public ModPoly(int m) {
taille = m + 1;
coef = new double[taille];
}

public double fonction(double x , int i){
return Math.pow(x, i);
}

public double p(double x) {
double res = 0;
for (int i = 0; i < taille; i++) {
res += coef[i] * fonction(x, i);
}
return res;
}

public void identifie(double[] x, double[] y) throws Exception {
if(x.length != y.length) throw new Exception("Les tableaux x et y doivent avoir
la même taille");
```

```
int n = x.length;
int m = taille;

Matrice matrice = new Matrice(n, m);
for (int i = 0; i < n; i++) {
for (int j = 0; j < m; j++) {
matrice.remplacecoef(i, j, fonction(x[i], j));
}
}

Matrice matriceT = matrice.transpose();
Matrice matriceProduit = Matrice.produit(matriceT, matrice);
Vecteur yVecteur = new Vecteur(y);

Matrice vecteurProduit = Matrice.produit(matriceT, yVecteur);
Vecteur yProduit = new Vecteur(m);

for (int i = 0; i < m; i++) {
yProduit.remplacecoef(i, vecteurProduit.getCoef( i, 0));
}

Helder ldr = new Helder(matriceProduit, yProduit);
Vecteur solution = ldr.resolution();

for (int i = 0; i < m; i++) {
coef[i] = solution.getCoef(i);
}

}

public static void main(String[] args) {
try {
double[] x = null;
double[] y = null;
Scanner scanner = new Scanner(System.in);
System.out.print("Entrez le nom du fichier contenant les points de support : ");
String filename = scanner.nextLine();
System.out.print("Entrez le degré du polynôme : ");
int degre = scanner.nextInt();
scanner.close();

BufferedReader br = new BufferedReader(new FileReader(filename));
String line = br.readLine();
if (line != null) {
int size = Integer.parseInt(line);
x = new double[size];
y = new double[size];

line = br.readLine();
String[] xParts = line.split(" ");
for (int i = 0; i < size; i++) {
x[i] = Double.parseDouble(xParts[i]);
}

line = br.readLine();
String[] yParts = line.split(" ");
for (int i = 0; i < size; i++) {
y[i] = Double.parseDouble(yParts[i]);
}
}
br.close();
```

```
ModPoly modPoly = new ModPoly(degre);
modPoly.identifie(x, y);

XYChart chart = new XYChartBuilder().width(800)
.height(600).title("Modèle Polynomial")
.xAxisTitle("X").yAxisTitle("Y")
.build();
chart.addSeries("Points de support", x,
y).setMarker(SeriesMarkers.CIRCLE).setLineStyle(SeriesLines.NONE);

double[] interpolX = new double[100];
double[] interpolY = new double[100];
double minX = Arrays.stream(x).min().getAsDouble();
double maxX = Arrays.stream(x).max().getAsDouble();
for (int i = 0; i < 100; i++) {
interpolX[i] = minX + i * (maxX - minX) / 99;
interpolY[i] = modPoly.p(interpolX[i]);
}
chart.addSeries("Interpolation", interpolX,
interpolY).setLineStyle(SeriesLines.SOLID);

new SwingWrapper<>(chart).displayChart();
} catch (Exception e) {
e.printStackTrace();
}
}
}
```

et la classe Matrice qui a été modifier en ajoutant la methode transpose()

Classe Matrice

```
package AlgLin;

import java.io.*;
import java.util.*;

public class Matrice {

/**
* Définir ici les attributs de la classe *
*/
protected double coefficient[][];
public final static double EPSILON = 1.0E-06;

/**
* Définir ici les constructeur de la classe *
*/
Matrice(int nbligne, int nbcolonne) {
this.coefficient = new double[nbligne][nbcolonne];
}

Matrice(double[][] tableau) {
coefficient = tableau;
}
```

```
Matrice(String fichier) {
try {
Scanner sc = new Scanner(new File(fichier));
int ligne = sc.nextInt();
int colonne = sc.nextInt();
this.coefficient = new double[ligne][colonne];
for (int i = 0; i < ligne; i++) {
for (int j = 0; j < colonne; j++) {
this.coefficient[i][j] = sc.nextDouble();
}
}
sc.close();

} catch (FileNotFoundException e) {
System.out.println("Fichier absent");
}
}

/**
* Definir ici les autres methodes
*/

public void recopie(Matrice arecopier) {
int ligne, colonne;
ligne = arecopier.nbLigne();
colonne = arecopier.nbColonne();
this.coefficient = new double[ligne][colonne];
for (int i = 0; i < ligne; i++) {
for (int j = 0; j < colonne; j++) {
this.coefficient[i][j] = arecopier.coefficient[i][j];
}
}
}

public int nbLigne() {
return this.coefficient.length;
}

public int nbColonne() {
return this.coefficient[0].length;
}

public double getCoef(int ligne, int colonne) {
return this.coefficient[ligne][colonne];
}

public void remplacecoef(int ligne, int colonne, double value) {
this.coefficient[ligne][colonne] = value;
}

public String toString() {
int ligne = this.nbLigne();
int colonne = this.nbColonne();
String matr = "";
for (int i = 0; i < ligne; i++) {
for (int j = 0; j < colonne; j++) {
if (j == 0) {
matr += this.getCoef(i, j);
} else {
matr += " " + this.getCoef(i, j);
}
```

```
}
matr += "\n";
}
return matr;
}

public Matrice produit(double scalaire) {
int ligne = this.nbLigne();
int colonne = this.nbColonne();
for (int i = 0; i < ligne; i++) {
for (int j = 0; j < colonne; j++) {
this.coefficient[i][j] *= scalaire;
}
}
return this;
}

static Matrice addition(Matrice a, Matrice b) {
int ligne = a.nbLigne();
int colonne = a.nbColonne();
Matrice mat = new Matrice(ligne, colonne);
for (int i = 0; i < ligne; i++) {
for (int j = 0; j < colonne; j++) {
mat.coefficient[i][j] = a.coefficient[i][j] + b.coefficient[i][j];
}
}
return mat;
}

static Matrice verif_addition(Matrice a, Matrice b) throws Exception {
if ((a.nbLigne() == b.nbLigne()) && (a.nbColonne() == b.nbColonne())) {
int ligne = a.nbLigne();
int colonne = a.nbColonne();
Matrice mat = new Matrice(ligne, colonne);
for (int i = 0; i < ligne; i++) {
for (int j = 0; j < colonne; j++) {
mat.coefficient[i][j] = a.coefficient[i][j] + b.coefficient[i][j];
}
}
return mat;
} else {
throw new Exception("Les deux matrices n'ont pas les mêmes dimensions !!!");
}
}

static Matrice produit(Matrice a, Matrice b) {
int ligne, colonne;
ligne = a.nbLigne();
colonne = b.nbColonne();
Matrice mat = new Matrice(ligne, colonne);
for (int i = 0; i < ligne; i++) {
for (int j = 0; j < colonne; j++) {
mat.coefficient[i][j] = 0;
for (int k = 0; k < a.nbColonne(); k++) {
mat.coefficient[i][j] += a.coefficient[i][k] * b.coefficient[k][j];
}
}
}
return mat;
}
```

```
static Matrice verif_produit(Matrice a, Matrice b) throws Exception {
int ligne = 0;
int colonne = 0;
if (a.nbColonne() == b.nbLigne()) {
ligne = a.nbLigne();
colonne = b.nbColonne();
} else {
throw new Exception("Dimensions des matrices à multiplier incorrectes");
}

Matrice mat = new Matrice(ligne, colonne);
for (int i = 0; i < ligne; i++) {
for (int j = 0; j < colonne; j++) {
mat.coefficient[i][j] = 0;
for (int k = 0; k < a.nbColonne(); k++) {
mat.coefficient[i][j] += a.coefficient[i][k] * b.coefficient[k][j];
}
}
}
return mat;
}

/**
* La methode inverse permet de calculer l'inverse d'une matrice
*
* @return Matrice
* @throws Exception
*/
public Matrice inverse() throws Exception {
if (this.nbLigne() != this.nbColonne()) {
throw new Exception("La matrice n'est pas carrée");
}
Matrice mattrice = new Matrice(this.nbLigne(), this.nbColonne());
Matrice matrice = new Matrice(this.nbLigne(), this.nbColonne());
mattrice.recopie(this);
Vecteur vecteur = new Vecteur(this.nbLigne());

Helder ldr = new Helder(mattrice, vecteur);
ldr.factorisationLDR();

for (int i = 0; i < this.nbLigne(); i++) {
if (0 < i) {
vecteur.remplacecoef(i - 1, 0);
}
vecteur.remplacecoef(i, 1);
ldr.setSecondMembre(vecteur);
Vecteur res = ldr.resolutionPartielle();
for (int j = 0; j < this.nbLigne(); j++) {
matrice.remplacecoef(j, i, res.getCoef(j));
}
}
return matrice;
}

/**
* methode de calcul de la transposée de la matrice
*
* @return Matrice
*/

public Matrice transpose() {
```

```
Matrice mat = new Matrice(this.nbColonne(), this.nbLigne());
for (int i = 0; i < this.nbLigne(); i++) {
for (int j = 0; j < this.nbColonne(); j++) {
mat.coefficient[j][i] = this.coefficient[i][j];
}
}
return mat;
}

/**
* norme de la matrice
*
* @return double
*/
public double norme() {
double norme = 0;
for (int i = 0; i < this.nbLigne(); i++) {
double somme = 0;
for (int j = 0; j < this.nbColonne(); j++) {
somme += Math.abs(this.getCoef(i, j));
}
if (somme > norme) {
norme = somme;
}
}
return norme;
}

/**
* methode de calcul de la norme de la matrice infinie
*
* @return double
*/
public double normeInfinie() {
double norme = 0;
for (int i = 0; i < this.nbLigne(); i++) {
double somme = 0;
for (int j = 0; j < this.nbColonne(); j++) {
somme += Math.abs(this.getCoef(j, i));
}
if (somme > norme) {
norme = somme;
}
}
return norme;
}

/**
* methode de calcul de conditionnement de la matrice
*
* @return double
*/
public double conditionnement() throws Exception {
return this.norme() * this.inverse().norme();
}

/**
* methode de calcul de conditionnement de la matrice infinie
*
* @return double
*/
```

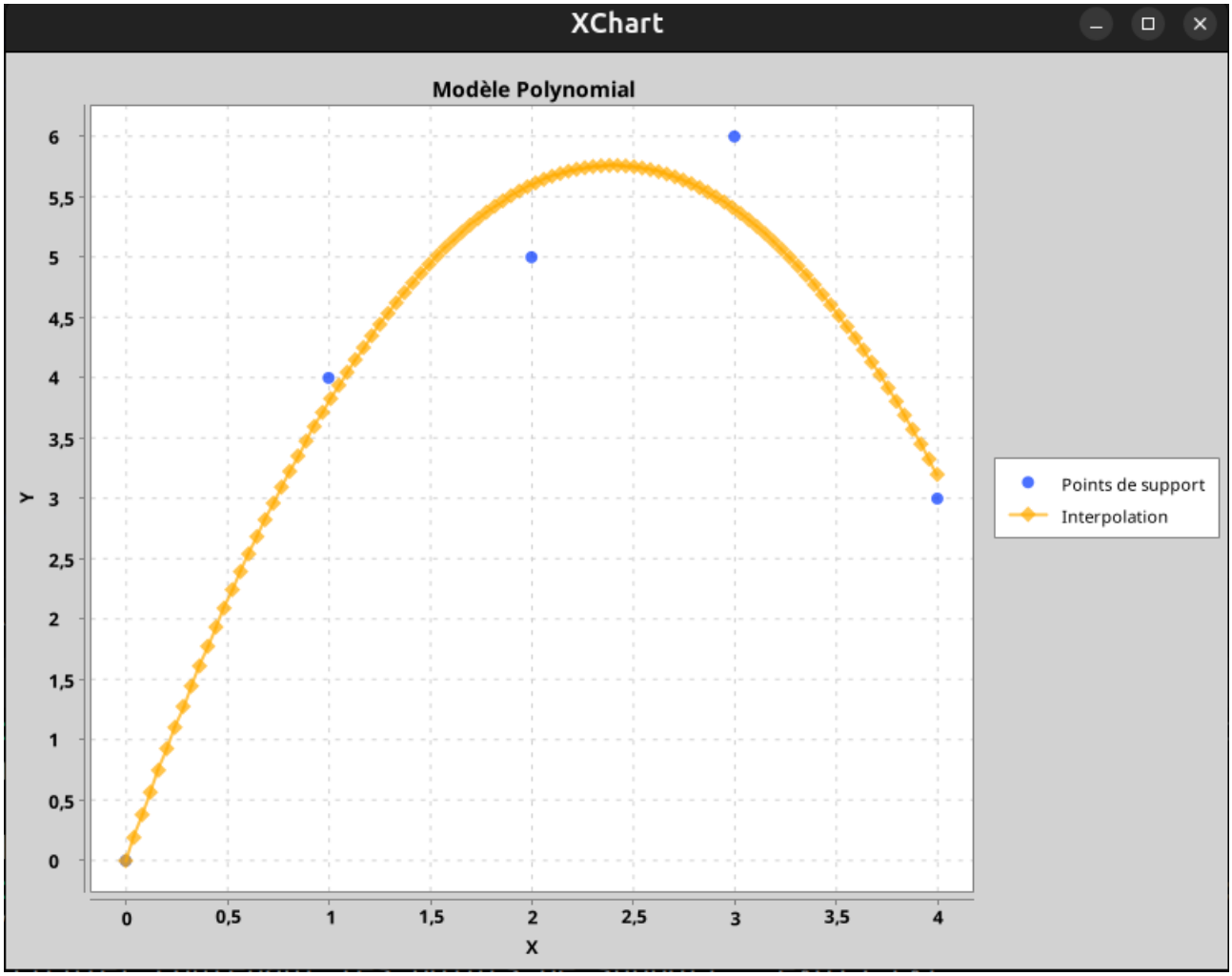
```
public double conditionnementInfinie() throws Exception {
return this.normeInfinie() * this.inverse().normeInfinie();
}

public static void main(String[] args) throws Exception {

double mat[][] = {{2, 1}, {0, 1}};
Matrice a = new Matrice(mat);
System.out.println("construction d'une matrice par affectation d'un tableau :\n"
+ a);
Matrice b = new Matrice("matrice.txt");
System.out.println("Construction d'une matrice par lecture d'un fichier :\n" +
b);
Matrice c = new Matrice(2, 2);
c.recopie(b);
System.out.println("Recopie de la matrice b :\n" + c);
System.out.println("Nombre de lignes et colonnes de la matrice c : " +
c.nbLigne()
+ ", " + c.nbColonne());
System.out.println("Coefficient (2,2) de la matrice b : " + b.getCoef(1, 1));
System.out.println("Nouvelle valeur de ce coefficient : 8");
b.remplacecoef(1, 1, 8);
System.out.println("Vérification de la modification du coefficient");
System.out.println("Coefficient (2,2) de la matrice b : " + b.getCoef(1, 1));
System.out.println("Addition de 2 matrices : affichage des 2 matrices "
+ "puis de leur addition");
System.out.println("matrice 1 :\n" + a + "matrice 2 :\n" + b + "somme :\n"
+ Matrice.addition(a, b));
System.out.println("Produit de 2 matrices : affichage des 2 matrices "
+ "puis de leur produit");
System.out.println("matrice 1 :\n" + a + "matrice 2 :\n" + b + "produit :\n"
+ produit(a, b));

//test de la methode inverse
double matrice[][] = {{-1, 1, -1}, {1, 1, 1}, {-1, 1, 1}};
double identite[][] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
Matrice matI = new Matrice(matrice);
Matrice id = new Matrice(identite);
Matrice inverse = matI.inverse();
Matrice produit = Matrice.produit(matI, inverse);
System.out.println("Matrice identité : \n" + id);
System.out.println("Matrice inverse : \n" + inverse);
System.out.println("Produit de la matrice et son inverse : \n" + produit);
System.out.println("Norme du produit entre inverse et la matrice : " +
(produit.norme() != id.norme() ? "Erreur de calcul" : "la norme est : " +
produit.norme()));
System.out.println("Norme Infinie de la matrice du prouit entre inverse et la
matrice : " + (produit.normeInfinie() != id.normeInfinie() ? "Erreur de
calcul" : "la norme est : " + produit.normeInfinie()));
System.out.println("Conditionnement de la matrice : " +
produit.conditionnement());
System.out.println("Conditionnement Infinie de la matrice : " +
produit.conditionnementInfinie());

}
}
```

Résultat Obtenu

Glody KUTUMBAKANA

L’implemation a ete tester avec des fichier (txt) de points de support contenant la taille et les point X et Y

le resultat obtenus est :



Conclusion

Cette classe est une implémentation efficace pour l'ajustement polynomial via la méthode des moindres carrés.