Национальная научно-образовательная корпорация ИТМО
Факультет программной инженерии и компьютерной техники

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине
«Вычислительная математика»

Вариант № 8

Выполнил:
Студент группы P3210
Мальков Павел Александрович
Преподаватель:
Машина Екатерина Алексеевна

Санкт-Петербург, 2024

Выполнение первой части

y = $\frac{3x}{x^4+8}$

n = 11
x = [-2;0]
h = 0.2

i	1	2	3	4	5	6	7	8	9	10	11
x_i	-2	-1.8	-1.6	-1.4	-1.2	-1	-0.8	-0.6	-0.4	-0.2	0
y_i	-1.412	1.879	-2.542	-3.47	-4.685	-6.0	-6.81	-6.374	-4.68	-2.396	0.0

φ(x) = a + b * x

Вычисляем суммы: sx = -11, sxx = 15.4, sy = -40.248, sxy = 38.377

$$\begin{cases} n*a+sx*b=sy \\ sx*a+sxx*b=sxy \end{cases} \quad \begin{cases} 11*a-11*b=-40.248 \\ -11*a+15.4*b=38.377 \end{cases}$$

$$\begin{cases} a=-4.084 \\ b=-0.425 \end{cases}$$

φ(x) = -4.084 – 0.425 * x
x_i -2.0 -1.8 -1.6 -1.4 -1.2 -1.0 -0.8 -0.6 -0.4 -0.2 0.0
y_i -1.412 -1.879 -2.542 -3.47 -4.685 -6.0 -6.81 -6.374 -4.68 -2.396 0.0
φ (x_i) 7.743 6.926 6.109 5.293 4.476 3.659 2.842 2.025 1.209 0.392 -0.425
delta 83.814 77.528 74.84 76.79 83.924 93.296 93.161 70.543 34.68 7.773 0.181

$$\sigma = \sqrt{\frac{\sum(\varphi(xi)-yi)^2}{n}} = 2.602$$

<u>Квадратичная аппроксимация</u>:

y = $\frac{12x}{x^4+1}$

n = 11
x in [0;2]
h = 0.2

i	1	2	3	4	5	6	7	8	9	10	11
x_i	-2	-1.8	-1.6	-1.4	-1.2	-1	-0.8	-0.6	-0.4	-0.2	0
y_i	-1.412	1.879	-2.542	-3.47	-4.685	-6.0	-6.81	-6.374	-4.68	-2.396	0.0

φ(x) = a + b * x + c * x^2
Вычисляем суммы: sx = -11, sxx = 15.4, sxxx = -24.2, sxxxx = 40.533,
sy = -40.248, sxy = 38.377, sxxy = -45.289

$$\begin{cases} n*a + sx*b + sxx*c = sy \\ sx*a + sxx*b + sxxx*c = sxy \\ sxx*a + sxxx*b + sxxxx*c = sxxy \end{cases} \begin{cases} 11*a - 11*b + 15.4*c = -40.248 \\ -11*a + 15.4*b - 24.2*c = 38.377 \\ 15.4*-24.2*b + 40.533*c = -45.289 \end{cases}$$

$$\begin{cases} 11*a = -6.9 + 11*b - 15.4*c \\ 4.4*b - 8.8*c = 0.73 \\ 15.4*a - 24.2*b + 40.53*c = -10.06 \end{cases} \begin{cases} a = -0.63 + b - 1.4*c \\ 4.4*b - 8.8*c = 0.73 \\ 15.4*a - 24.2*b + 40.53*c = -10.06 \end{cases}$$

$$\begin{cases} a = -0.63 + b - 1.4*c \\ 4.4*b - 8.8*c = 0.73 \\ -9.7 + 15.4*b - 21.56*c - 24.2*b + 40.53*c = -10.06 \end{cases}$$

$$\begin{cases} a = -0.63 + b - 1.4*c \\ 4.4*b - 8.8*c = 0.73 \\ -8.8*b + 18.97*c = -0.36 \end{cases} \begin{cases} a = -0.63 + b - 1.4*c \\ 4.4*b = 0.73 + 8.8*c \\ -1.46 - 17.6*c + 18.97*c = -0.36 \end{cases}$$

$$\begin{cases} a = -0.63 + b - 1.4*c \\ 4.4*b = 0.73 + 8.8*c \\ 1.37*c = 1.1 \end{cases} \begin{cases} a = -0.63 + b - 1.4*c \\ 4.4*b = 7.77 \\ 1.37*c = 1.1 \end{cases} \begin{cases} a = -0.63 + 1.77 - 1.4*0.8 \\ b = 1.77 \\ 1.37*c = 1.1 \end{cases}$$

$$\begin{cases} a = -0.887 \\ b = 10.232 \\ c = 5.329 \end{cases}$$

φ(x) = 5.329 + 10.232 * x - 0.887 * x ^ 2

x_i	-2.0	-1.8	-1.6	-1.4	-1.2	-1.0	-0.8	-0.6	-0.4	-0.2	0.0
y_i	-1.412	-1.879	-2.542	-3.47	-4.685	-6.0	-6.81	-6.374	-4.68	-2.396	0.0
phi_i	-0.035	-2.039	-3.616	-4.767	-5.492	-5.79	-5.662	-5.108	-4.127	-2.72	-0.887
delta	1.896	0.026	1.153	1.682	0.651	0.044	1.318	1.603	0.306	0.105	0.787

$$\sigma = \sqrt{\frac{\sum(\varphi(xi) - yi)^2}{n}} = \mathbf{0.933}$$

У квадратичной аппроксимации среднеквадратичное отклонение меньше, поэтому это приближение наилучшее.

Выполнение второй части

Кубическая аппроксимация:

```
package com.example.lab4.methods;

import lombok.Getter;
import org.apache.commons.math3.linear.*;

import java.util.ArrayList;
@Getter
public class CubApproximation extends Method{
    private Double a;
    private Double b;
    private Double c;
    private Double d;

    public double f(double x) {
        return a * Math.pow(x, 3) + b * Math.pow(x, 2) + c * x + d;
    }
    protected Double determ;

    @Override
    public void calculate(ArrayList<Double> arrayOfX, ArrayList<Double> arrayOfY, int
n) {
        double[][] matrixData = new double[4][4];
        double[] vector = new double[4];

        for (int i = 0; i < n; i++) {
            double x = arrayOfX.get(i);
            double y = arrayOfY.get(i);
            double x2 = Math.pow(x, 2);
            double x3 = Math.pow(x, 3);
            double x4 = Math.pow(x, 4);
            double x5 = Math.pow(x, 5);
            double x6 = Math.pow(x, 6);

            matrixData[0][0] += x6;
            matrixData[0][1] += x5;
            matrixData[0][2] += x4;
            matrixData[0][3] += x3;
            matrixData[1][0] += x5;
            matrixData[1][1] += x4;
            matrixData[1][2] += x3;
            matrixData[1][3] += x2;
            matrixData[2][0] += x4;
            matrixData[2][1] += x3;
            matrixData[2][2] += x2;
            matrixData[2][3] += x;
            matrixData[3][0] += x3;
            matrixData[3][1] += x2;
            matrixData[3][2] += x;
            matrixData[3][3] += 1;

            vector[0] += y * x3;
            vector[1] += y * x2;
            vector[2] += y * x;
            vector[3] += y;
        }

        RealMatrix coefficients = new Array2DRowRealMatrix(matrixData, false);
        DecompositionSolver solver = new LUDecomposition(coefficients).getSolver();

        RealVector constants = new ArrayRealVector(vector, false);
        RealVector solution = solver.solve(constants);
```

```
        a = solution.getEntry(0);
        b = solution.getEntry(1);
        c = solution.getEntry(2);
        d = solution.getEntry(3);
        double phiAvg = 0;

        for (int i = 0; i < n; i++){
            ArrayList<Double> tmp = new ArrayList<>();
            tmp.add(arrayOfX.get(i));
            tmp.add(arrayOfY.get(i));
            tmp.add(f(arrayOfX.get(i)));
            tmp.add(f(arrayOfX.get(i)) - arrayOfY.get(i));
            table.add(tmp);
            S += Math.pow(f(arrayOfX.get(i)) - arrayOfY.get(i), 2);
        }
        sko = Math.sqrt(S/n);

        phiAvg = phiAvg / n;
        double ssTot = 0;
        double ssRes = 0;

        for (int i = 0; i < n; i++) {
            double fi = f(arrayOfX.get(i));
            ssTot += Math.pow(arrayOfY.get(i) - phiAvg, 2);
            ssRes += Math.pow(arrayOfY.get(i) - fi, 2);
        }

        determ = 1 - (ssRes / ssTot);
    }

    @Override
    public String getNameMethod() {
        return "Кубическая аппроксимация";
    }
    @Override
    protected String getStringFun() {
        return "phi(x)="+ a +"* x ^ 3 + " + b + " * x ^ 2 + " + c +" * x + " + d;
    }

}
```

Экспоненциальная аппроксимация:

```
package com.example.lab4.methods;

import java.util.ArrayList;

public class ExponentialApproximation extends Method{
    private Double a;
    private Double b;

    public double f(double x) {
        return a * Math.exp(b * x);
    }

    @Override
    public void calculate(ArrayList<Double> arrayOfX, ArrayList<Double> arrayOfY, int
n) {
        double sumX = 0;
        double sumYln = 0;
        double sumX2 = 0;
        double sumXYln = 0;

        for (int i = 0; i < n; i++) {
            double x = arrayOfX.get(i);
            double y = Math.log(arrayOfY.get(i));
```

```
            sumX += x;
            sumYln += y;
            sumX2 += x * x;
            sumXYln += x * y;
        }


        double b = (n * sumXYln - sumX * sumYln) / (n * sumX2 - sumX * sumX);
        double aLn = (sumYln - b * sumX) / n;
        double phiAvg = 0;
        this.a = Math.exp(aLn);
        this.b = b;

        for (int i = 0; i < n; i++) {
            double fi = f(arrayOfX.get(i));
            double eps = fi - arrayOfY.get(i);
            ArrayList<Double> row = new ArrayList<>();
            row.add(arrayOfX.get(i));
            row.add(arrayOfY.get(i));
            phiAvg += f(arrayOfX.get(i));
            row.add(fi);
            row.add(eps);
            table.add(row);
            S += Math.pow(eps, 2);
        }
        sko = Math.sqrt(S/n);

        phiAvg = phiAvg / n;
        double down = 0;
        double up = 0;

        for (int i = 0; i< n; i++) {
            double fi = f(arrayOfX.get(i));
            down += Math.pow(arrayOfY.get(i) - phiAvg, 2);
            up += Math.pow(arrayOfY.get(i) - fi, 2);
        }

        determ = 1 - (up / down);

    }

    @Override
    public String getNameMethod() {
        return "Экспоненциальная аппроксимация";
    }
    @Override
    protected String getStringFun() {
        return "Phi(x)=" + a + " * e ^ " + b + " x";
    }
}
```

Линейная аппроксимация:

```
package com.example.lab4.methods;
import lombok.Getter;
import java.util.ArrayList;
@Getter
public class LineApproximation extends Method {
    private Double a;
    private Double b;
    protected Double S = 0.0;
    protected Double sko;
    protected Double korrelPirs;
    protected Double determ;

     public double f(double x){
```

```
        return a*x + b;
    }

    @Override
    public void calculate(ArrayList<Double> arrayOfX, ArrayList<Double> arrayOfY, int
n){
        double sumX = 0;
        double sumXX = 0;
        double sumY = 0;
        double sumXY = 0;
        double sumYY = 0;
        for (int i = 0; i < n; i++){
            sumX += arrayOfX.get(i);
            sumXX += Math.pow(arrayOfX.get(i), 2);
            sumY += arrayOfY.get(i);
            sumXY += arrayOfX.get(i) *arrayOfY.get(i);
            sumYY += arrayOfY.get(i) * arrayOfY.get(i);
        }
        a = (sumXY*n - sumX*sumY)/(sumXX*n - sumX*sumX);
        b = (sumXX*sumY - sumX*sumXY)/(sumXX*n - sumX*sumX);

        for (int i = 0; i < n; i++){
            ArrayList<Double> tmp = new ArrayList<>();
            tmp.add(arrayOfX.get(i));
            tmp.add(arrayOfY.get(i));
            tmp.add(f(arrayOfX.get(i)));
            tmp.add(f(arrayOfX.get(i)) - arrayOfY.get(i));
            table.add(tmp);
            S += Math.pow(f(arrayOfX.get(i)) - arrayOfY.get(i), 2);
        }

        sko = Math.sqrt(S/n);

        korrelPirs = (sumXY * n - sumX * sumY) / Math.sqrt((sumXX * n - sumX * sumX) *
(sumYY * n - sumY * sumY));
        determ = Math.pow(korrelPirs ,2);
    }

    @Override
    public String getNameMethod() {
        return "Линейная аппроксимация";
    }

    @Override
    protected String getStringFun() {
        return "phi(x)="+ a +" * x +" + b + "\nКоэффициент Пирсона " + korrelPirs;
    }
}
```

Логарифмическая аппроксимация:

```
package com.example.lab4.methods;
import lombok.Getter;
import java.util.ArrayList;
@Getter
public class LogarithmApproximation extends Method {
    private Double a;
    private Double b;
    public double f(double x) {
        return a * Math.log(x) + b;
    }

    @Override
    public void calculate(ArrayList<Double> arrayOfX, ArrayList<Double> arrayOfY, int
n) {
        double sumLnX = 0;
        double sumY = 0;
```

```
        double sumLnX2 = 0;
        double sumYLnX = 0;

        for (int i = 0; i < n; i++) {
            double x = arrayOfX.get(i);
            double y = arrayOfY.get(i);
            double lnX = Math.log(x);

            sumLnX += lnX;
            sumY += y;
            sumLnX2 += lnX * lnX;
            sumYLnX += y * lnX;
        }

        double b = (sumY * sumLnX2 - sumLnX * sumYLnX) / (n * sumLnX2 - sumLnX *
sumLnX);
        double a = (n * sumYLnX - sumLnX * sumY) / (n * sumLnX2 - sumLnX * sumLnX);

        this.a = a;
        this.b = b;
        double phiAvg = 0;

        for (int i = 0; i < n; i++) {
            ArrayList<Double> row = new ArrayList<>();
            row.add(arrayOfX.get(i));
            row.add(arrayOfY.get(i));
            phiAvg += f(arrayOfX.get(i));
            row.add(f(arrayOfX.get(i)));
            row.add(f(arrayOfX.get(i)) - arrayOfY.get(i));
            table.add(row);
            S += Math.pow(f(arrayOfX.get(i)) - arrayOfY.get(i), 2);
        }
        sko = Math.sqrt(S/n);

        phiAvg = phiAvg / n;
        double ssTot = 0;
        double ssRes = 0;

        for (int i = 0; i < n; i++) {
            double fi = f(arrayOfX.get(i));
            ssTot += Math.pow(arrayOfY.get(i) - phiAvg, 2);
            ssRes += Math.pow(arrayOfY.get(i) - fi, 2);
        }


        determ = 1 - (ssRes / ssTot);
    }

    @Override
    public String getNameMethod() {
        return "Логарифмическая аппроксимация";
    }
    @Override
    protected String getStringFun() {
        return "phi(x)="+a+" * log(x) + " + b ;
    }
}
```

Степенная аппроксимация:

```
package com.example.lab4.methods;

import java.util.ArrayList;

public class PowerApproximation extends Method{
    private Double a;
    private Double b;
```

```
    public double f(double x) {
        return a * Math.pow(x, b);
    }

    @Override
    public void calculate(ArrayList<Double> arrayOfX, ArrayList<Double> arrayOfY, int
n) {
        double sumLnX = 0;
        double sumLnY = 0;
        double sumLnX2 = 0;
        double sumLnXLnY = 0;

        for (int i = 0; i < n; i++) {
            double lnX = Math.log(arrayOfX.get(i));
            double lnY = Math.log(arrayOfY.get(i));

            sumLnX += lnX;
            sumLnY += lnY;
            sumLnX2 += lnX * lnX;
            sumLnXLnY += lnX * lnY;
        }

        b = (n * sumLnXLnY - sumLnX * sumLnY) / (n * sumLnX2 - sumLnX * sumLnX);
        double lnA = (sumLnY - b * sumLnX) / n;
        a = Math.exp(lnA);
        double phiAvg = 0;

        for (int i = 0; i < n; i++) {
            double fi = f(arrayOfX.get(i));
            double eps = fi - arrayOfY.get(i);
            ArrayList<Double> row = new ArrayList<>();
            row.add(arrayOfX.get(i));
            row.add(arrayOfY.get(i));
            phiAvg += f(arrayOfX.get(i));
            row.add(fi);
            row.add(eps);
            table.add(row);
            S += Math.pow(eps, 2);
        }
        sko = Math.sqrt(S/n);

        phiAvg = phiAvg / n;
        double ssTot = 0;
        double ssRes = 0;

        for (int i = 0; i < n; i++) {
            double fi = f(arrayOfX.get(i));
            ssTot += Math.pow(arrayOfY.get(i) - phiAvg, 2);
            ssRes += Math.pow(arrayOfY.get(i) - fi, 2);
        }
        determ = 1 - (ssRes / ssTot);
    }

    @Override
    public String getNameMethod() {
        return "Степенная аппроксимация";
    }

    @Override
    protected String getStringFun() {
        return "Phi(x)= " + a  + " * x" + "^ b";
    }
}
```

Квадратичная аппроксимация:

```java
package com.example.lab4.methods;

import java.util.ArrayList;
import lombok.Getter;
import org.apache.commons.math3.linear.*;
@Getter
public class QuadApproximation extends Method {
    private Double a;
    private Double b;
    private Double c;

    public double f(double x) {
        return a * Math.pow(x, 2) + b * x + c;
    }
    protected Double determ;

    @Override
    public void calculate(ArrayList<Double> arrayOfX, ArrayList<Double> arrayOfY, int
n) {
        double[] terms = new double[3];
        double[][] matrixData = new double[3][3];
        double[] vector = new double[3];

        for (int i = 0;i<n;i++) {
            double x = arrayOfX.get(i);
            double y = arrayOfY.get(i);
            terms[0] += Math.pow(x, 4);
            terms[1] += Math.pow(x, 3);
            terms[2] += Math.pow(x, 2);

            matrixData[0][0] += Math.pow(x, 4);
            matrixData[0][1] += Math.pow(x, 3);
            matrixData[0][2] += Math.pow(x, 2);
            matrixData[1][0] += Math.pow(x, 3);
            matrixData[1][1] += Math.pow(x, 2);
            matrixData[1][2] += x;
            matrixData[2][0] += Math.pow(x, 2);
            matrixData[2][1] += x;
            matrixData[2][2] += 1;

            vector[0] += x * x * y;
            vector[1] += x * y;
            vector[2] += y;
        }

        RealMatrix coefficients = new Array2DRowRealMatrix(matrixData, false);
        DecompositionSolver solver = new LUDecomposition(coefficients).getSolver();

        RealVector constants = new ArrayRealVector(vector, false);
        RealVector solution = solver.solve(constants);

        a = solution.getEntry(0);
        b = solution.getEntry(1);
        c = solution.getEntry(2);
        double phiAvg = 0;

        for (int i = 0; i < n; i++){
            ArrayList<Double> tmp = new ArrayList<>();
            tmp.add(arrayOfX.get(i));
            tmp.add(arrayOfY.get(i));
            phiAvg += f(arrayOfX.get(i));
            tmp.add(f(arrayOfX.get(i)));
            tmp.add(f(arrayOfX.get(i)) - arrayOfY.get(i));
            table.add(tmp);
            S += Math.pow(f(arrayOfX.get(i)) - arrayOfY.get(i), 2);
        }
        sko = Math.sqrt(S/n);
```

```
        phiAvg = phiAvg / n;
        double ssTot = 0;
        double ssRes = 0;

        for (int i = 0; i < n; i++) {
            double fi = f(arrayOfX.get(i));
            ssTot += Math.pow(arrayOfY.get(i) - phiAvg, 2);
            ssRes += Math.pow(arrayOfY.get(i) - fi, 2);
        }

        determ = 1 - (ssRes / ssTot);
    }

    @Override
    public String getNameMethod() {
        return "Квадратичная аппроксимация";
    }
    @Override
    protected String getStringFun() {
        return "phi(x)="+ a +" * x^2 + " + b +" * x +" + c;
    }
}
```

Вывод

В ходе лабораторной работы я познакомился с разными аппроксимациями, реализовал их на языке JAVA.