Федеральное государственное автономное образовательное учреждение высшего образования **«Национальный исследовательский университет ИТМО»**

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа **№4**
«Аппроксимация функции методом наименьших квадратов»

по дисциплине «Вычислительная математика»

Вариант: **2**

Преподаватель:
Машина Е. А.

Выполнил:
Вальц Мартин Эдуардович
Группа: P3210

Санкт-Петербург, 2024 г.

Цель работы: найти функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.

1. Вычислительная реализация задачи

Линейная аппроксимация:

$y = \frac{15x}{x^4+2}$

n = 11
x ∈ [0; 4]
h = 0.4

i	1	2	3	4	5	6	7	8	9	10	11
x_i	0	0.4	0.8	1.2	1.6	2.0	2.4	2.8	3.2	3.6	4.0
y_i	0.0	2.962	4.98	4.419	2.806	1.667	1.023	0.662	0.449	0.318	0.233

φ(x) = a + bx

Вычисляем суммы: sx = 22, sxx = 61.6, sy = 19.52 sxy = 26.116

$$\begin{cases} n*a+sx*b=sy \\ sx*a+sxx*b=sxy \end{cases} \begin{cases} 11*a+22*b=19.52 \\ 22*a+61.6*b=26.116 \end{cases} \begin{cases} 11*a+22*b=19.52 \\ 17.6*b=-12.924 \end{cases}$$

$$\begin{cases} b=-12.924/17.6=-0.7343 \\ 11a=19.52-22*(-0.7343)=35.6746 \end{cases} \begin{cases} b=-0.7343 \\ a=3.2431 \end{cases}$$

φ(x) = $3.2431-0.7343*x$

i	1	2	3	4	5	6	7	8	9	10	11
x_i	0	0.4	0.8	1.2	1.6	2.0	2.4	2.8	3.2	3.6	4.0
y_i	0.0	2.962	4.98	4.419	2.806	1.667	1.023	0.662	0.449	0.318	0.233
φ(xi)	3.243	2.949	2.656	2.362	2.068	1.775	1.481	1.187	0.893	0.6	0.306
(φ (xi)-yi)^2	10.518	0.0	5.403	4.231	0.544	0.012	0.21	0.276	0.197	0.079	0.00

σ = $\sqrt{\frac{\sum(\varphi(xi)-yi)^2}{n}}$ = **1.3972**

Квадратичная аппроксимация:

$y = \frac{15x}{x^4+2}$

n = 11

$x \in [0; 4]$

h = 0.4

i	1	2	3	4	5	6	7	8	9	10	11
x_i	0	0.4	0.8	1.2	1.6	2.0	2.4	2.8	3.2	3.6	4.0
y_i	0.0	2.962	4.98	4.419	2.806	1.667	1.023	0.662	0.449	0.318	0.233

φ(x) = a + bx + cx^2

Вычисляем суммы:

sx = 22, sxx = 61.6, sxxx = 193.6, sxxxx = 648.52, sy = 19.52, sxy = 26.116, sxxy = 47.405

$$\begin{cases} n*a+sx*b+sxx*c=sy \\ sx*a+sxx*b+sxxx*c=sxy \\ sxx*a+sxxx*b+sxxxx*c=sxxy \end{cases}$$

$$\begin{cases} 11*a+22*b+61.6*c=19.52 \\ 22*a+61.6*b+193.6*c=26.116 \\ 61.6*a+193.6*b+648.52*c=47.405 \end{cases}$$

По методу Крамера:

$\Delta=4251.456$

$\Delta_1=9043.80576, \Delta_2=4785.47696, \Delta_3=-1976.8496$

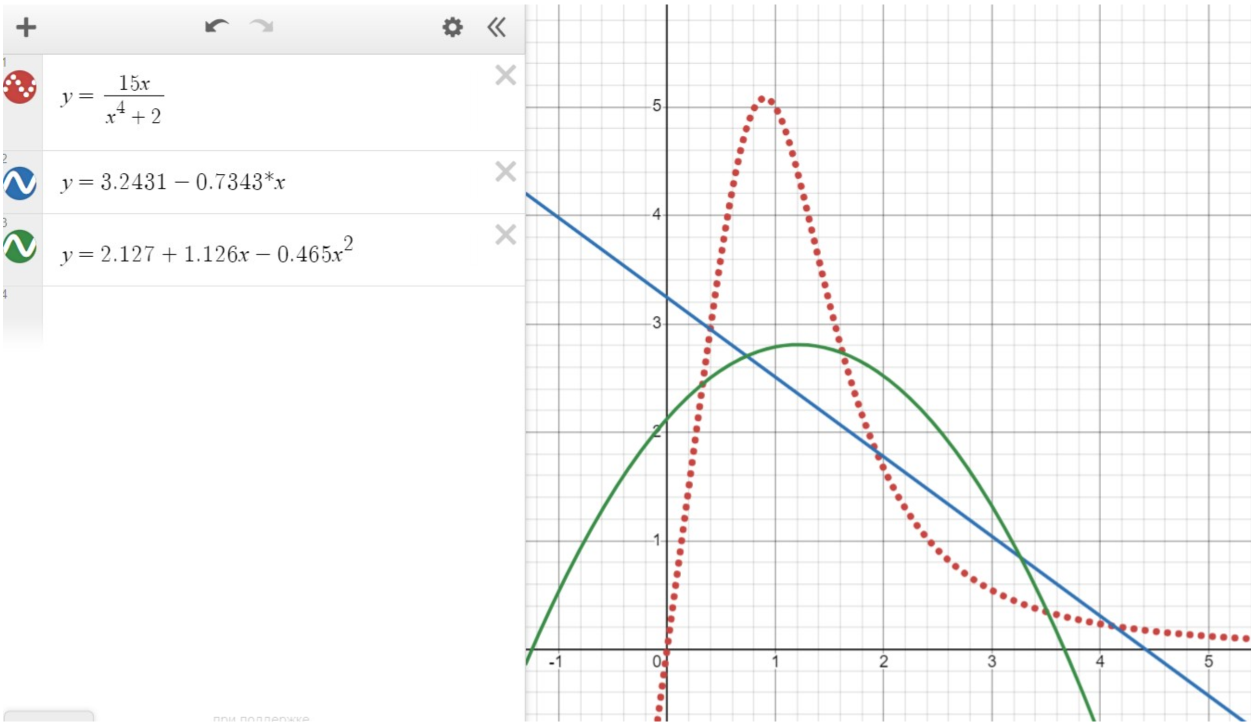
$$\begin{cases} a=\frac{\Delta_1}{\Delta}=\frac{9043.80576}{4251.456}\approx 2.127 \\ b=\frac{\Delta_2}{\Delta}=\frac{4785.47696}{4251.456}\approx 1.126 \\ c=\frac{\Delta_3}{\Delta}=\frac{-1976.8496}{4251.456}\approx -0.465 \end{cases}$$

φ(x) $2.127+1.126x-0.465x^2$

i	1	2	3	4	5	6	7	8	9	10	11
x_i	0	0.4	0.8	1.2	1.6	2.0	2.4	2.8	3.2	3.6	4.0
y_i	0.0	2.962	4.98	4.419	2.806	1.667	1.023	0.662	0.449	0.318	0.233
φ(xi)	2.127	2.503	2.73	2.809	2.738	2.519	2.151	1.634	0.969	0.154	-0.809
(φ (xi)-yi)^2	4.524	0.211	5.062	2.593	0.005	0.726	1.272	0.945	0.27	0.027	1.086

σ = $\sqrt{\frac{\sum(\varphi(xi)-yi)^2}{n}}$ = **1.23292**

1.23292 < 1.3972, у квадратичной аппроксимации среднеквадратичное отклонение меньше, поэтому это приближение лучше.



2. Программная реализация задачи

```
package lab4.work;

import javafx.scene.chart.XYChart;
import lab4.entity.Dot;
import lab4.entity.DotCollection;
import lab4.graph.Graphic;
import lab4.util.Printer;

import java.util.ArrayList;
import java.util.NoSuchElementException;

import static java.lang.Math.abs;

public class Approximation {

    private static int numberApprox = 1;

    private static int finalNumberApprox;

    private static double[] finalCoefficients;

    private static double minDeviation = Double.MAX_VALUE;

    public static XYChart.Series<Double, Double> s1;
    public static XYChart.Series<Double, Double> s2;
    public static XYChart.Series<Double, Double> s3;
    public static XYChart.Series<Double, Double> s4;
    public static XYChart.Series<Double, Double> s5;
    public static XYChart.Series<Double, Double> s6;



    public static int getNumberApprox() {
        return numberApprox;
    }

    public static int getFinalNumberApprox() {
        return finalNumberApprox;
    }

    public static double[] getFinalCoefficients() {
        return finalCoefficients;
    }

    public static void run() {
        linearApprox();
        quadraticApprox();
        cubicApprox();
        powerApprox();
        exponentialApprox();
```

```java
        logApprox();
    }

    private static void save(double[] coefficients, double deviation) {
        if (deviation < minDeviation) {
            minDeviation = deviation;
            finalCoefficients = coefficients;
            finalNumberApprox = numberApprox;
        }
    }

    private static double linearApprox() {
        Dot[] dots = DotCollection.getDots();
        double[] coefficients = getLinearCoefficients(dots);
        double deviation = getDeviation(coefficients);
        save(coefficients, deviation);
        s1 = Graphic.addGraph1(coefficients);
        Printer.printP(coefficients, deviation, getR2(coefficients));
        Printer.printR(getR(dots));
        return deviation;
    }

    private static double quadraticApprox() {
        numberApprox++;
        Dot[] dots = DotCollection.getDots();
        double[] coefficients = getQuadraticCoefficients(dots);
        double deviation = getDeviation(coefficients);
        save(coefficients, deviation);
        s2 = Graphic.addGraph2(coefficients);
        Printer.printP(coefficients, deviation, getR2(coefficients));
        return deviation;
    }

    private static double cubicApprox() {
        numberApprox++;
        Dot[] dots = DotCollection.getDots();
        double[] coefficients = getCubicCoefficients(dots);
        double deviation = getDeviation(coefficients);
        save(coefficients, deviation);
        s3 = Graphic.addGraph3(coefficients);
        Printer.printP(coefficients, deviation, getR2(coefficients));
        return deviation;
    }

    private static double powerApprox() {
        numberApprox++;
        Dot[] dots = DotCollection.getDots();
        Dot[] cloneDots = cloneDots(dots);

        for (Dot dot : cloneDots) {
            double x = dot.getX();
            double y = dot.getY();
            dot.setX(Math.log(x));
```

```java
        dot.setY(Math.log(y));
    }

    double[] coefficients = getLinearCoefficients(cloneDots);
    double[] ab = new double[2];
    ab[0] = Math.pow(Math.E, coefficients[1]);
    ab[1] = coefficients[0];
    double deviation = getDeviation(ab);
    save(ab, deviation);
    s4 = Graphic.addGraph4(ab);
    Printer.printP(ab, deviation, getR2(ab));
    return deviation;
}

private static double exponentialApprox() {
    numberApprox++;
    Dot[] dots = DotCollection.getDots();
    Dot[] cloneDots = cloneDots(dots);

    for (Dot dot : cloneDots) {
        double y = dot.getY();
        dot.setY(Math.log(y));
    }

    double[] coefficients = getLinearCoefficients(cloneDots);
    double[] ab = new double[2];
    ab[0] = Math.pow(Math.E, coefficients[1]);
    ab[1] = coefficients[0];
    double deviation = getDeviation(ab);
    save(ab, deviation);
    s5 = Graphic.addGraph5(ab);
    Printer.printP(ab, deviation, getR2(ab));
    return deviation;
}

private static double logApprox() {
    numberApprox++;
    Dot[] dots = DotCollection.getDots();
    Dot[] cloneDots = cloneDots(dots);
    for (Dot dot : cloneDots) {
        double x = dot.getX();
        dot.setX(Math.log(x));
    }
    double[] coefficients = getLinearCoefficients(cloneDots);
    double deviation = getDeviation(coefficients);
    save(coefficients, deviation);
    s6 = Graphic.addGraph6(coefficients);
    Printer.printP(coefficients, deviation, getR2(coefficients));
    return deviation;
}

private static double getR(Dot[] dots) {
    double averageX = 0;
```

```
    double averageY = 0;
    for (Dot dot : dots) {
        averageX += dot.getX();
        averageY += dot.getY();
    }
    averageX /= dots.length;
    averageY /= dots.length;

    double differenceX;
    double differenceY;
    double sum1 = 0;
    double sum2 = 0;
    double sum3 = 0;
    for (Dot dot : dots) {
        differenceX = dot.getX() - averageX;
        differenceY = dot.getY() - averageY;
        sum1 += differenceX * differenceY;
        sum2 += Math.pow(differenceX, 2);
        sum3 += Math.pow(differenceY, 2);
    }
    return sum1 / Math.sqrt(sum2 * sum3);

}

private static Dot[] cloneDots(Dot[] dots) {
    Dot[] cloneDots = new Dot[dots.length];
    for (int i = 0; i < cloneDots.length; i++) {
        cloneDots[i] = new Dot(dots[i].getX(), dots[i].getY());
    }
    return cloneDots;
}

private static double[] getLinearCoefficients(Dot[] dots) {
    double sx = 0, sxx = 0, sy = 0, sxy = 0;
    for (Dot dot : dots) {
        double x = dot.getX();
        double y = dot.getY();
        sx += x;
        sy += y;
        sxx += Math.pow(x, 2);
        sxy += x * y;
    }

    double[][] matrix = new double[2][3];
    matrix[0][0] = sxx;
    matrix[0][1] = sx;
    matrix[0][2] = sxy;

    matrix[1][0] = sx;
    matrix[1][1] = dots.length;
    matrix[1][2] = sy;

    return MatrixGaussMethod.calculateSolutions(matrix);
```

```java
}

private static double[] getQuadraticCoefficients(Dot[] dots) {
    double xi = 0, xi2 = 0, xi3 = 0, xi4 = 0, yi = 0, xiyi = 0, xi2yi = 0;
    for (Dot dot : dots) {
        double x = dot.getX();
        double y = dot.getY();
        xi += x;
        xi2 += Math.pow(x, 2);
        xi3 += Math.pow(x, 3);
        xi4 += Math.pow(x, 4);
        yi += y;
        xiyi += x * y;
        xi2yi += Math.pow(x, 2) * y;
    }

    double[][] matrix = new double[3][4];
    matrix[0][0] = dots.length;
    matrix[0][1] = xi;
    matrix[0][2] = xi2;
    matrix[0][3] = yi;

    matrix[1][0] = xi;
    matrix[1][1] = xi2;
    matrix[1][2] = xi3;
    matrix[1][3] = xiyi;

    matrix[2][0] = xi2;
    matrix[2][1] = xi3;
    matrix[2][2] = xi4;
    matrix[2][3] = xi2yi;
    return MatrixGaussMethod.calculateSolutions(matrix);
}

private static double[] getCubicCoefficients(Dot[] dots) {
    double xi = 0, xi2 = 0, xi3 = 0, xi4 = 0, xi5 = 0, xi6 = 0, yi = 0, xiyi = 0, xi2yi = 0, xi3yi = 0;
    for (Dot dot : dots) {
        double x = dot.getX(), y = dot.getY();
        xi += x;
        xi2 += Math.pow(x, 2);
        xi3 += Math.pow(x, 3);
        xi4 += Math.pow(x, 4);
        xi5 += Math.pow(x, 5);
        xi6 += Math.pow(x, 6);
        yi += y;
        xiyi += x * y;
        xi2yi += Math.pow(x, 2) * y;
        xi3yi += Math.pow(x, 3) * y;
    }

    double[][] matrix = new double[4][5];
    matrix[0][0] = dots.length;
    matrix[0][1] = xi;
```

```
        matrix[0][2] = xi2;
        matrix[0][3] = xi3;
        matrix[0][4] = yi;

        matrix[1][0] = xi;
        matrix[1][1] = xi2;
        matrix[1][2] = xi3;
        matrix[1][3] = xi4;
        matrix[1][4] = xiyi;

        matrix[2][0] = xi2;
        matrix[2][1] = xi3;
        matrix[2][2] = xi4;
        matrix[2][3] = xi5;
        matrix[2][4] = xi2yi;

        matrix[3][0] = xi3;
        matrix[3][1] = xi4;
        matrix[3][2] = xi5;
        matrix[3][3] = xi6;
        matrix[3][4] = xi3yi;
        return MatrixGaussMethod.calculateSolutions(matrix);
    }

    private static double getR2(double[] coefficients) {
        Dot[] dots = DotCollection.getDots();
        double S = 0;
        double avg = 0;
        double L = 0;
        for (Dot dot : dots) {
            double x = dot.getX();
            double phi_x;
            switch (numberApprox) {
                case 1 -> phi_x = getValueLinearApprox(coefficients[0], coefficients[1], x);
                case 2 -> phi_x = getValueQuadraticApprox(coefficients[0], coefficients[1], coefficients[2],
x);
                case 3 ->
                        phi_x = getValueCubicApprox(coefficients[0], coefficients[1], coefficients[2],
coefficients[3], x);
                case 4 -> phi_x = getValuePowerApprox(coefficients[0], coefficients[1], x);
                case 5 -> phi_x = getValueExponentialApprox(coefficients[0], coefficients[1], x);
                default -> phi_x = getValueLogApprox(coefficients[0], coefficients[1], x);
            }
            avg += phi_x;
        }
        avg = avg / dots.length;
        for (Dot dot : dots) {
            double x = dot.getX();
            double y = dot.getY();
            double phi_x;
            switch (numberApprox) {
                case 1 -> phi_x = getValueLinearApprox(coefficients[0], coefficients[1], x);
                case 2 -> phi_x = getValueQuadraticApprox(coefficients[0], coefficients[1], coefficients[2],
```

```
x);
            case 3 ->
                    phi_x = getValueCubicApprox(coefficients[0], coefficients[1], coefficients[2],
coefficients[3], x);
            case 4 -> phi_x = getValuePowerApprox(coefficients[0], coefficients[1], x);
            case 5 -> phi_x = getValueExponentialApprox(coefficients[0], coefficients[1], x);
            default -> phi_x = getValueLogApprox(coefficients[0], coefficients[1], x);
        }
        double e = phi_x - y;
        double f = avg - y;
        S += Math.pow(e, 2);
        L += Math.pow(f, 2);
    }
    return 1 - (S / L);
}

private static double getDeviation(double[] coefficients) {
    Dot[] dots = DotCollection.getDots();
    double S = 0;
    ArrayList<double[]> table = new ArrayList<>();
    Printer.printLabel();
    for (Dot dot : dots) {
        double x = dot.getX();
        double y = dot.getY();
        double phi_x;
        switch (numberApprox) {
            case 1 -> phi_x = getValueLinearApprox(coefficients[0], coefficients[1], x);
            case 2 -> phi_x = getValueQuadraticApprox(coefficients[0], coefficients[1], coefficients[2],
x);
            case 3 ->
                    phi_x = getValueCubicApprox(coefficients[0], coefficients[1], coefficients[2],
coefficients[3], x);
            case 4 -> phi_x = getValuePowerApprox(coefficients[0], coefficients[1], x);
            case 5 -> phi_x = getValueExponentialApprox(coefficients[0], coefficients[1], x);
            default -> phi_x = getValueLogApprox(coefficients[0], coefficients[1], x);
        }
        double e = phi_x - y;
        S += Math.pow(e, 2);
        table.add(new double[]{x, y, phi_x, e});
    }
    Printer.printTable(table);
    return Math.sqrt(S / dots.length);
}

public static double getValueLinearApprox(double a, double b, double x) {
    return a * x + b;
}

public static double getValueQuadraticApprox(double a0, double a1, double a2, double x) {
    return a0 + x * a1 + Math.pow(x, 2) * a2;
}

public static double getValueCubicApprox(double a0, double a1, double a2, double a3, double x) {
```

```java
        return a0 + x * a1 + Math.pow(x, 2) * a2 + Math.pow(x, 3) * a3;
    }

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

    public static double getValueExponentialApprox(double a, double b, double x) {
        return a * Math.pow(Math.E, b * x);
    }

    public static double getValueLogApprox(double a, double b, double x) {
        return a * Math.log(x) + b;
    }

    private static class MatrixGaussMethod {
        static Integer findMaxColumnElement(double[][] matrix, int i) throws NoSuchElementException
{
            int n = matrix.length;
            int point = 0;
            double max = 0;
            for (int j = i; j < n; j++) {
                if (max < abs(matrix[j][i])) {
                    max = abs(matrix[j][i]);
                    point = j;
                }
            }
            if (max == 0) throw new NoSuchElementException();
            return point;
        }

        static double[][] calculateTriangleMatrix(double[][] matrix) throws NoSuchElementException {
            int n = matrix.length;
            for (int i = 0; i < n; i++) {

                Integer point = findMaxColumnElement(matrix, i);

                for (int j = i; j <= n; j++) {
                    double temp = matrix[i][j];
                    matrix[i][j] = matrix[point][j];
                    matrix[point][j] = temp;
                }

                for (int k = n; k >= i; k--)
                    matrix[i][k] = matrix[i][k] / matrix[i][i];

                for (int k = i + 1; k < n; k++)
                    for (int j = n; j >= i; j--)
                        matrix[k][j] = matrix[k][j] - matrix[k][i] * matrix[i][j];

            }
            return matrix;
        }
```

```java
    static double[] calculateSolutions(double[][] matrix) throws NoSuchElementException {
        double[][] triangleMatrix = calculateTriangleMatrix(matrix);
        int size = triangleMatrix.length;
        double[] solutions = new double[size];
        for (int i = size - 1; i >= 0; i--) {
            double root = triangleMatrix[i][size];
            for (int j = i + 1; j < size; j++) {
                root -= solutions[j] * triangleMatrix[i][j];
            }
            solutions[i] = root / triangleMatrix[i][i];
        }
        return solutions;
    }
  }
}
```

Вывод

В ходе данной работы была выполнена аппроксимация функций с использованием линейного, квадратичного, кубического, экспоненциального и логарифмического приближений. Также на основе этих методов была реализована программа на Java, которая реализует метод наименьших квадратов и строит графики исходной функции и аппроксимаций.
Исследование позволило определить наилучшее приближение, вычислить среднеквадратические отклонения и коэффициент корреляции Пирсона для линейной зависимости.