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

Факультет программной инженерии и компьютерной техники

Отчет по лабораторной работе №4

«Аппроксимация функции методом наименьших квадратов»

По дисциплине «Вычислительная математика»

Вариант 8

Выполнила: Иванова Мария Максимовна

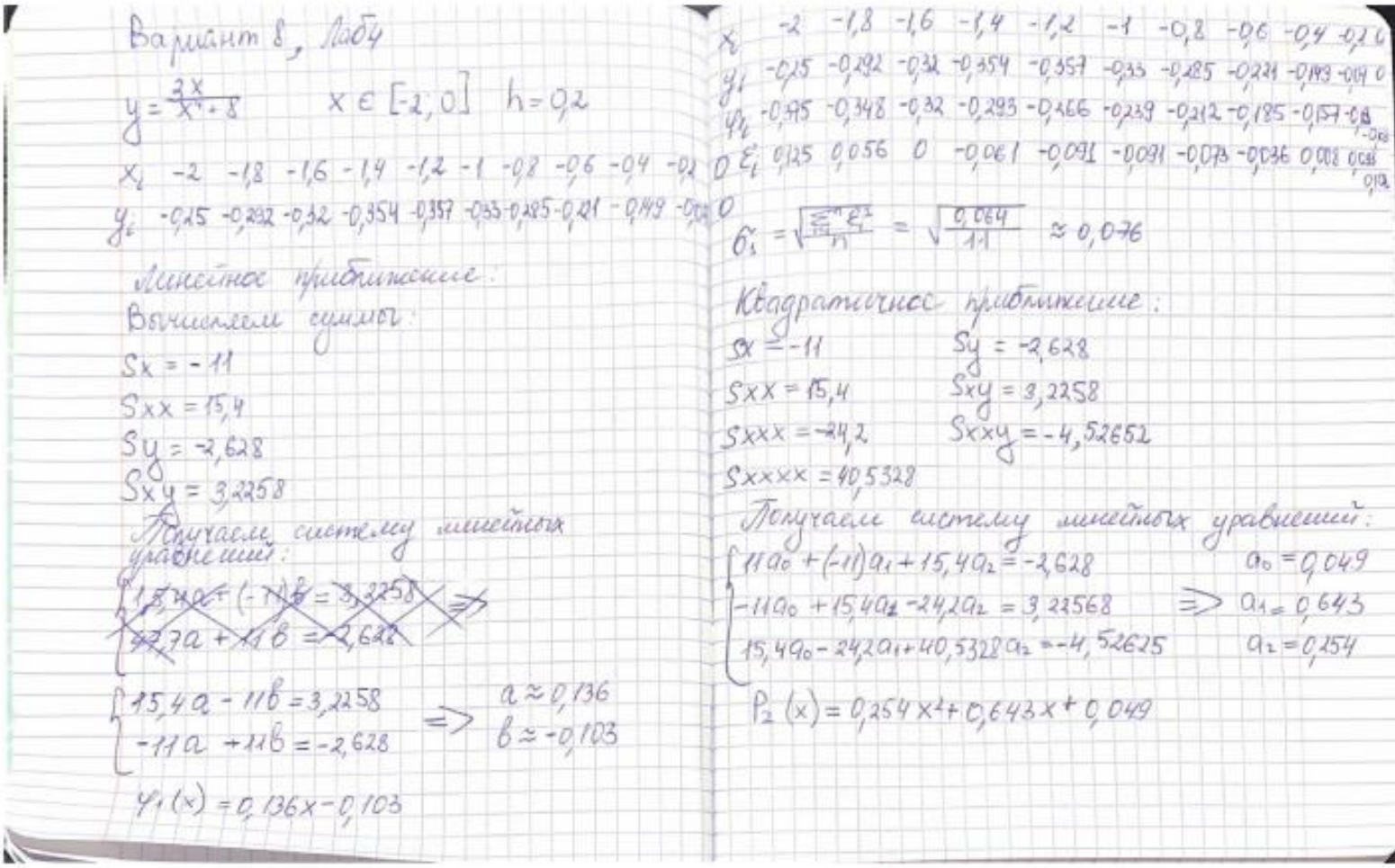
Группа: P3208

Преподаватель: Машина Екатерина Алексеевна

Санкт-Петербург

~ 2024 ~

Вычислительная часть:



Вариант 8, Лаб 4

$y = \frac{3x}{x^3 + 8}$ $x \in [-2, 0]$ $h = 0,2$

x_i	-2	-1,8	-1,6	-1,4	-1,2	-1	-0,8	-0,6	-0,4	-0,2	0
y_i	-0,25	-0,292	-0,32	-0,354	-0,357	-0,33	-0,285	-0,21	-0,149	-0,08	0
φ_i	-0,25	-0,348	-0,32	-0,293	-0,266	-0,239	-0,212	-0,185	-0,157	-0,08	0
ε_i	0,125	0,056	0	-0,061	-0,091	-0,091	-0,073	-0,036	0,008	0,08	0

Линейное приближение:

Вычисляем суммы:

$Sx = -11$

$Sxx = 15,4$

$Sy = -2,628$

$Sxy = 3,2258$

Получаем систему линейных уравнений:

$$\begin{cases} 15,4a - 11b = 3,2258 \\ -11a + 11b = -2,628 \end{cases} \Rightarrow \begin{matrix} a \approx 0,136 \\ b \approx -0,103 \end{matrix}$$

$y_1(x) = 0,136x - 0,103$

$$\sigma_1 = \sqrt{\frac{\sum_{i=1}^{11} \varepsilon_i^2}{11}} = \sqrt{\frac{0,064}{11}} \approx 0,076$$

Квадратичное приближение:

$Sx = -11$ $Sy = -2,628$

$Sxx = 15,4$ $Sxy = 3,2258$

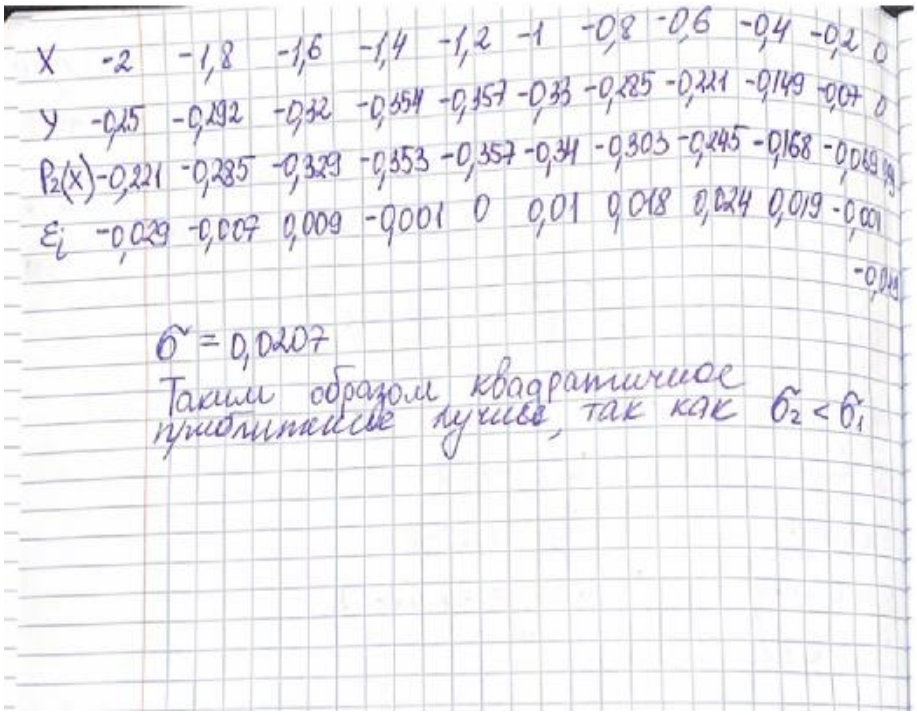
$Sxxx = -24,2$ $Sxxy = -4,52652$

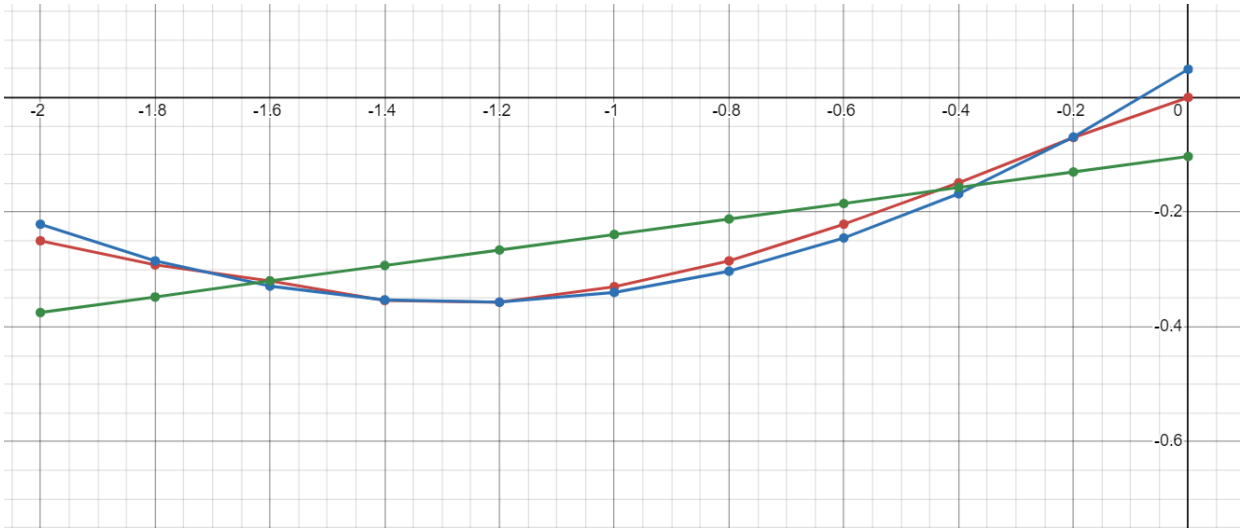
$Sxxxx = 40,5328$

Получаем систему линейных уравнений:

$$\begin{cases} 11a_0 + (-11)a_1 + 15,4a_2 = -2,628 \\ -11a_0 + 15,4a_1 - 24,2a_2 = 3,22568 \\ 15,4a_0 - 24,2a_1 + 40,5328a_2 = -4,52625 \end{cases} \Rightarrow \begin{matrix} a_0 = 0,049 \\ a_1 = 0,643 \\ a_2 = 0,254 \end{matrix}$$

$P_2(x) = 0,254x^2 + 0,643x + 0,049$





Листинг программы:

```
package org.example;

import javax.swing.*;

public class LinearApproximation {

    private final int number = 1;

    private final String NAME = "ЛИНЕЙНАЯ";
```

```
    private double a = 0;
    private double b = 0;

    private double sko = 0;

    private double[] epsilon;

    public double[] solve(double[] x, double[] y, int amount) {
        double sx = 0;
        double sxx = 0;
        double sy = 0;
        double sxy = 0;
        for (int i = 0; i < amount; i++) {
            sx += x[i];
        }
        for (int i = 0; i < amount; i++) {
            sxx += x[i] * x[i];
        }
        for (int i = 0; i < amount; i++) {
            sy += y[i];
        }
        for (int i = 0; i < amount; i++) {
            sxy += x[i] * y[i];
        }
        a = (sxy * amount - sx * sy) / (sxx * amount - sx * sx);
        b = (sxx * sy - sx * sxy) / (sxx * amount - sx * sx);
        double[] result = new double[amount];
        for (int i = 0; i < amount; i++) {
            result[i] = a * x[i] + b;
        }
        epsilon = new double[amount];
        for (int i = 0; i < amount; i++) {
            epsilon[i] = result[i] - y[i];
        }
        for (int i = 0; i < amount; i++) {
            sko += epsilon[i] * epsilon[i];
        }
        sko = Math.sqrt(sko / amount);

        return result;
    }

    public void draw(double[] x, double[] y, double[] result, int amount) {
        String title = "Linear Approximation";
        FunctionDrawer functionDrawer = new FunctionDrawer(title, amount, x,
y, result);
        functionDrawer.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        functionDrawer.pack();
        functionDrawer.setVisible(true);
    }

    public double getPearsonCoefficient(double[] x, double[] y, int amount) {
        double midX = 0;
        double midY = 0;
        double r = 0;
        for (int i = 0; i < amount; i++) {
            midX += x[i];
            midY += y[i];
        }
        midX = midX / amount;
        midY = midY / amount;
        double chisl = 0;
        double znam = 0;
        for (int i = 0; i < amount; i++) {
```

```
            chisl += (x[i] - midX) * (y[i] - midY);
            znam += (x[i] - midX) * (x[i] - midX) * (y[i] - midY) * (y[i] -
midY);
        }
        r = chisl / znam;
        return r;
    }

    public double getDeterminationCoefficient(double[] result, double y[],
int amount) {
        double midPhi = 0;
        double r2 = 0;
        for (int i = 0; i < amount; i++) {
            midPhi += result[i];
        }
        midPhi = midPhi / amount;
        double chisl = 0;
        double znam = 0;
        for (int i = 0; i < amount; i++) {
            chisl += (y[i] - result[i]) * (y[i] - result[i]);
            znam += (y[i] - midPhi) * (y[i] - midPhi);
        }
        r2 = 1 - chisl / znam;
        return r2;
    }

    public double getA() {
        return a;
    }

    public double getB() {
        return b;
    }

    public double[] getEpsilon() {
        return epsilon;
    }

    public double getSko() {
        return sko;
    }

}
```

```
package org.example;

import javax.swing.*;

public class QuadraticApproximation {

    private final int number = 2;
    private final String NAME = "КВАДРАТИЧНАЯ";
    private double [] epsilon;
    double a0 = 0;
    double a1 = 0;
    double a2 = 0;

    double sko = 0;

    public double [] solve(double [] x, double[] y, int amount){
        double sx = 0;
        double sxx = 0;
        double sxxx = 0;
```

```
        double sxxxx = 0;
        double sy = 0;
        double sxy = 0;
        double sxxy = 0;
        for (int i = 0; i < amount; i++){
            sx+=x[i];
        }
        for (int i = 0; i < amount; i++){
            sxx+=x[i]*x[i];
        }
        for (int i = 0; i < amount; i++){
            sxxx+=x[i]*x[i]*x[i];
        }
        for (int i = 0; i < amount; i++){
            sxxxx+=x[i]*x[i]*x[i]*x[i];
        }
        for (int i = 0; i < amount; i++){
            sy+=y[i];
        }
        for (int i = 0; i < amount; i++){
            sxy+=x[i]*y[i];
        }
        for (int i = 0; i < amount; i++){
            sxxy+=x[i]*x[i]*y[i];
        }

        a0 = (sy * sxx * sxxxx + sxxy*sx*sxxx + sxy*sxx*sxxx - sxxy * sxx *
sxx - sx*sxy*sxxxx - sy*sxxx*sxxx)/(amount*sxx*sxxxx+sx*sxx*sxxx+sxx*sx*sxxx
- sxx*sxx*sxx - sx * sx * sxxxx- amount*sxxx*sxxx);
        a1 = (amount*sxy*sxxxx+sx*sxx*sxxy+sxx*sy*sxxx-sxx*sxy*sxx -
sx*sy*sxxxx - amount*sxxy*sxxx)/(amount*sxx*sxxxx+sx*sxx*sxxx+sxx*sx*sxxx -
sxx*sxx*sxx - sx * sx * sxxxx- amount*sxxx*sxxx);
        a2 = (amount*sxx*sxxy+sx*sy*sxxx+sxx*sx*sxy - sxx*sxx*sy - sx*sx*sxxy
- amount*sxxx*sxy)/(amount*sxx*sxxxx+sx*sxx*sxxx+sxx*sx*sxxx - sxx*sxx*sxx -
sx * sx * sxxxx- amount*sxxx*sxxx);

        double [] result = new double[amount];
        for (int i = 0; i<amount; i++){
            result[i] = a0 + a1*x[i] + a2*x[i]*x[i];
        }

        epsilon = new double[amount];
        for (int i = 0; i < amount; i++) {
            epsilon[i] = result[i] - y[i];
        }

        for (int i = 0; i < amount; i++) {
            sko+= epsilon[i]*epsilon[i];
        }
        sko = Math.sqrt(sko/amount);

        return result;
    }

    public void draw(double [] x, double [] y,double [] result, int amount){
        String title = "Quadratic Approximation";
        FunctionDrawer functionDrawer = new FunctionDrawer(title, amount, x,
y, result);
        functionDrawer.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        functionDrawer.pack();
        functionDrawer.setVisible(true);
    }

    public double getDeterminationCoefficient(double[] result, double y[],
```

```
int amount) {
        double midPhi = 0;
        double r2 = 0;
        for (int i = 0; i < amount; i++) {
            midPhi += result[i];
        }
        midPhi = midPhi / amount;
        double chisl = 0;
        double znam = 0;
        for (int i = 0; i < amount; i++) {
            chisl += (y[i] - result[i]) * (y[i] - result[i]);
            znam += (y[i] - midPhi) * (y[i] - midPhi);
        }
        r2 = 1 - chisl / znam;
        return r2;
    }

    public double getA0(){
        return a0;
    }

    public double getA1() {
        return a1;
    }

    public double getA2() {
        return a2;
    }

    public double [] getEpsilon(){
        return epsilon;
    }

    public double getSko(){
        return sko;
    }
}
```

```
package org.example;

import javax.swing.*;

public class QubicApproximation {
    private final int number = 3;
    private final String NAME = "КВАДРАТИЧНАЯ";
    private double[] epsilon;
    double a0 = 0;
    double a1 = 0;
    double a2 = 0;
    double a3 = 0;

    double sko = 0;

    public static double findDeterminant(double[][] matrix) {
        double determinant = 0;

        determinant = matrix[0][0] * (
                matrix[1][1] * (matrix[2][2] * matrix[3][3] - matrix[2][3] *
matrix[3][2]) -
                        matrix[1][2] * (matrix[2][1] * matrix[3][3] -
matrix[2][3] * matrix[3][1]) +
                        matrix[1][3] * (matrix[2][1] * matrix[3][2] -
matrix[2][2] * matrix[3][1])
```

```
        ) - matrix[0][1] * (
                matrix[1][0] * (matrix[2][2] * matrix[3][3] - matrix[2][3] *
matrix[3][2]) -
                        matrix[1][2] * (matrix[2][0] * matrix[3][3] -
matrix[2][3] * matrix[3][0]) +
                        matrix[1][3] * (matrix[2][0] * matrix[3][2] -
matrix[2][2] * matrix[3][0])
        ) + matrix[0][2] * (
                matrix[1][0] * (matrix[2][1] * matrix[3][3] - matrix[2][3] *
matrix[3][1]) -
                        matrix[1][1] * (matrix[2][0] * matrix[3][3] -
matrix[2][3] * matrix[3][0]) +
                        matrix[1][3] * (matrix[2][0] * matrix[3][1] -
matrix[2][1] * matrix[3][0])
        ) - matrix[0][3] * (
                matrix[1][0] * (matrix[2][1] * matrix[3][2] - matrix[2][2] *
matrix[3][1]) -
                        matrix[1][1] * (matrix[2][0] * matrix[3][2] -
matrix[2][2] * matrix[3][0]) +
                        matrix[1][2] * (matrix[2][0] * matrix[3][1] -
matrix[2][1] * matrix[3][0])
        );

        return determinant;
    }
    public double[] solve(double[] x, double[] y, int amount) {

        double sx = 0;
        double sxx = 0;
        double sxxx = 0;
        double sxxxx = 0;
        double sy = 0;
        double sxy = 0;
        double sxxy = 0;
        double sxxxxx = 0;
        double sxxxy = 0;
        double sxxxxxx = 0;

        for (int i = 0; i < amount; i++) {
            sx += x[i];
            sxx += x[i] * x[i];
            sxxx += x[i] * x[i] * x[i];
            sxxxx += x[i] * x[i] * x[i] * x[i];
            sxxxxx += x[i] * x[i] * x[i] * x[i] * x[i];
            sxxxxxx += x[i] * x[i] * x[i] * x[i] * x[i] * x[i];
            sy += y[i];
            sxy += x[i] * y[i];
            sxxy += x[i] * x[i] * y[i];
            sxxxy += x[i] * x[i] * x[i] * y[i];
        }



        double[][] matrix = {
                {amount, sx, sxx, sxxx},
                {sx, sxx, sxxx, sxxxx},
                {sxx, sxxx, sxxxx, sxxxxx},
                {sxxx, sxxxx, sxxxxx, sxxxxxx}
        };
        double[][] matrixA0 = {
                {sy, sx, sxx, sxxx},
                {sxy, sxx, sxxx, sxxxx},
                {sxxy, sxxx, sxxxx, sxxxxx},
                {sxxxy, sxxxx, sxxxxx, sxxxxxx}
```

```
        };
        double[][] matrixA1 = {
                {amount, sy, sxx, sxxx},
                {sx, sxy, sxxx, sxxxx},
                {sxx, sxxy, sxxxx, sxxxxx},
                {sxxx, sxxxy, sxxxxx, sxxxxxx}
        };
        double[][] matrixA2 = {
                {amount, sx, sy, sxxx},
                {sx, sxx, sxy, sxxxx},
                {sxx, sxxx, sxxy, sxxxxx},
                {sxxx, sxxxx, sxxxy, sxxxxxx}
        };
        double[][] matrixA3 = {
                {amount, sx, sxx, sy},
                {sx, sxx, sxxx, sxy},
                {sxx, sxxx, sxxxx, sxxy},
                {sxxx, sxxxx, sxxxxx, sxxxy}
        };
        double determinantMatrix = findDeterminant(matrix);

        if (determinantMatrix != 0) {

            a0 = findDeterminant(matrixA0) / determinantMatrix;
            a1 = findDeterminant(matrixA1) / determinantMatrix;
            a2 = findDeterminant(matrixA2) / determinantMatrix;
            a3 = findDeterminant(matrixA3) / determinantMatrix;

        } else {
            System.out.println("Определитель равен нулю, невозможно вычислить
коэффициенты.");
            System.exit(0);
        }

        double[] result = new double[amount];
        for (int i = 0; i < amount; i++) {
            result[i] = a0 + a1 * x[i] + a2 * x[i] * x[i] + a3 * x[i] * x[i]
* x[i];
        }

        epsilon = new double[amount];
        for (int i = 0; i < amount; i++) {
            epsilon[i] = result[i] - y[i];
        }

        for (int i = 0; i < amount; i++) {
            sko+= epsilon[i]*epsilon[i];
        }
        sko = Math.sqrt(sko/amount);

        return result;
    }
    public void draw(double [] x, double [] y, double [] result, int amount){
        String title = "Qubic Approximation";
        FunctionDrawer functionDrawer = new FunctionDrawer(title, amount, x,
y, result);
        functionDrawer.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        functionDrawer.pack();
        functionDrawer.setVisible(true);
    }

    public double getDeterminationCoefficient(double[] result, double y[],
int amount) {
        double midPhi = 0;
```

```
        double r2 = 0;
        for (int i = 0; i < amount; i++) {
            midPhi += result[i];
        }
        midPhi = midPhi / amount;
        double chisl = 0;
        double znam = 0;
        for (int i = 0; i < amount; i++) {
            chisl += (y[i] - result[i]) * (y[i] - result[i]);
            znam += (y[i] - midPhi) * (y[i] - midPhi);
        }
        r2 = 1 - chisl / znam;
        return r2;
    }
    public double[] getEpsilon() {
        return epsilon;
    }

    public double getA0() {
        return a0;
    }

    public double getA1() {
        return a1;
    }

    public double getA2() {
        return a2;
    }

    public double getA3() {
        return a3;
    }

    public double getSko(){
        return sko;
    }
}
```

```
package org.example;

import javax.swing.*;

public class LogarithmicApproximation {
    private final int number = 5;
    private final String NAME = "ЛОГАРИФМИЧЕСКАЯ";
    private double[] epsilon;

    double a;
    double b;

    double sko = 0;

    public double[] solve(double[] x, double[] y, int amount) {
        epsilon = new double[amount];

        double sx = 0;
        double sxx = 0;
        double sy = 0;
        double sxy = 0;

        for (int i = 0; i < amount; i++) {
            sx += Math.log(x[i]);
```

```
            sxx += Math.log(x[i]) * Math.log(x[i]);
            sy += y[i];
            sxy += Math.log(x[i]) * y[i];
        }

        a = (sxy * amount - sx * sy) / (sxx * amount - sx * sx);
        b = (sxx * sy - sx * sxy) / (sxx * amount - sx * sx);

        double[] result = new double[amount];
        for (int i = 0; i < amount; i++) {
            result[i] = a * Math.log(x[i]) + b;
        }

        for (int i = 0; i < amount; i++) {
            epsilon[i] = result[i] - y[i];
            sko += epsilon[i] * epsilon[i];
        }
        sko = Math.sqrt(sko / amount);

        return result;
    }

    public void draw(double[] x, double[] y, double[] result, int amount) {
        String title = "Logarithmic Approximation";
        FunctionDrawer functionDrawer = new FunctionDrawer(title, amount, x,
y, result);
        functionDrawer.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        functionDrawer.pack();
        functionDrawer.setVisible(true);
    }

    public double getDeterminationCoefficient(double[] result, double y[],
int amount) {
        double midPhi = 0;
        double r2 = 0;
        for (int i = 0; i < amount; i++) {
            midPhi += result[i];
        }
        midPhi = midPhi / amount;
        double chisl = 0;
        double znam = 0;
        for (int i = 0; i < amount; i++) {
            chisl += (y[i] - result[i]) * (y[i] - result[i]);
            znam += (y[i] - midPhi) * (y[i] - midPhi);
        }
        r2 = 1 - chisl / znam;
        return r2;
    }

    public double[] getEpsilon() {
        return epsilon;
    }

    public double getA() {
        return a;
    }

    public double getB() {
        return b;
    }

    public double getSko() {
        return sko;
```

```
    }
}
```

```
package org.example;

import javax.swing.*;


public class ExponentialApproximation {
    private final String NAME = "ЭКСПОНЕНЦИАЛЬНАЯ";
    private double[] epsilon;
    private double a;
    private double b;
    private double sko;

    public double[] solve(double[] x, double[] y, int amount) {
        epsilon = new double[amount];

        double sx = 0;
        double sxx = 0;
        double sy = 0;
        double sxy = 0;

        for (int i = 0; i < amount; i++) {
            sx += x[i];
            sxx += x[i] * x[i];
            sy += Math.log(y[i]);
            sxy += x[i] * Math.log(y[i]);
        }

        b = (sxy * amount - sx * sy) / (sxx * amount - sx * sx);
        a = Math.exp((sy - b * sx) / amount);

        double[] result = new double[amount];
        for (int i = 0; i < amount; i++) {
            result[i] = a * Math.exp(x[i] * b);
        }

        sko = 0;
        for (int i = 0; i < amount; i++) {
            epsilon[i] = result[i] - y[i];
            sko += epsilon[i] * epsilon[i];
        }
        sko = Math.sqrt(sko / amount);

        return result;
    }

    public void draw(double[] x, double[] y, double[] result, int amount) {
        String title = "Exponential Approximation";
        FunctionDrawer functionDrawer = new FunctionDrawer(title, amount, x,
y, result);
        functionDrawer.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        functionDrawer.pack();
        functionDrawer.setVisible(true);
    }

    public double getDeterminationCoefficient(double[] result, double y[],
int amount) {
        double midPhi = 0;
        double r2 = 0;
        for (int i = 0; i < amount; i++) {
            midPhi += result[i];
```

```
        }
        midPhi = midPhi / amount;
        double chisl = 0;
        double znam = 0;
        for (int i = 0; i < amount; i++) {
            chisl += (y[i] - result[i]) * (y[i] - result[i]);
            znam += (y[i] - midPhi) * (y[i] - midPhi);
        }
        r2 = 1 - chisl / znam;
        return r2;
    }

    public double[] getEpsilon() {
        return epsilon;
    }

    public double getA() {
        return a;
    }

    public double getB() {
        return b;
    }

    public double getSko() {
        return sko;
    }
}
```

```
package org.example;

import javax.swing.*;

public class PowerApproximation {

    private final int number = 6;
    private final String NAME = "СТЕПЕННАЯ";
    private double[] epsilon;

    private double a;
    private double b;

    double sko = 0;

    public double[] solve(double[] x, double[] y, int amount) {
        epsilon = new double[amount];

        double sx = 0;
        double sxx = 0;
        double sy = 0;
        double sxy = 0;

        for (int i = 0; i < amount; i++) {
            sx += Math.log(x[i]);
            sxx += Math.log(x[i]) * Math.log(x[i]);
            sy += Math.log(y[i]);
            sxy += Math.log(x[i]) * Math.log(y[i]);
        }

        a = (sxy * amount - sx * sy) / (sxx * amount - sx * sx);
        b = (sxx * sy - sx * sxy) / (sxx * amount - sx * sx);

        a = Math.exp(a);
```

```
        double[] result = new double[amount];
        for (int i = 0; i < amount; i++) {
            result[i] = a * Math.pow(x[i], b);
        }

        for (int i = 0; i < amount; i++) {
            epsilon[i] = result[i] - y[i];
            sko += epsilon[i] * epsilon[i];
        }
        sko = Math.sqrt(sko / amount);

        return result;
    }
    public void draw(double [] x, double [] y, double[] result, int amount){
        String title = "Power Approximation";
        FunctionDrawer functionDrawer = new FunctionDrawer(title, amount, x,
y, result);
        functionDrawer.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        functionDrawer.pack();
        functionDrawer.setVisible(true);
    }

    public double getDeterminationCoefficient(double[] result, double y[],
int amount) {
        double midPhi = 0;
        double r2 = 0;
        for (int i = 0; i < amount; i++) {
            midPhi += result[i];
        }
        midPhi = midPhi / amount;
        double chisl = 0;
        double znam = 0;
        for (int i = 0; i < amount; i++) {
            chisl += (y[i] - result[i]) * (y[i] - result[i]);
            znam += (y[i] - midPhi) * (y[i] - midPhi);
        }
        r2 = 1 - chisl / znam;
        return r2;
    }
    public double[] getEpsilon() {
        return epsilon;
    }

    public double getA() {
        return a;
    }

    public double getB() {
        return b;
    }

    public double getSko() {
        return sko;
    }

    public int getNumber() {
        return number;
    }
}
```

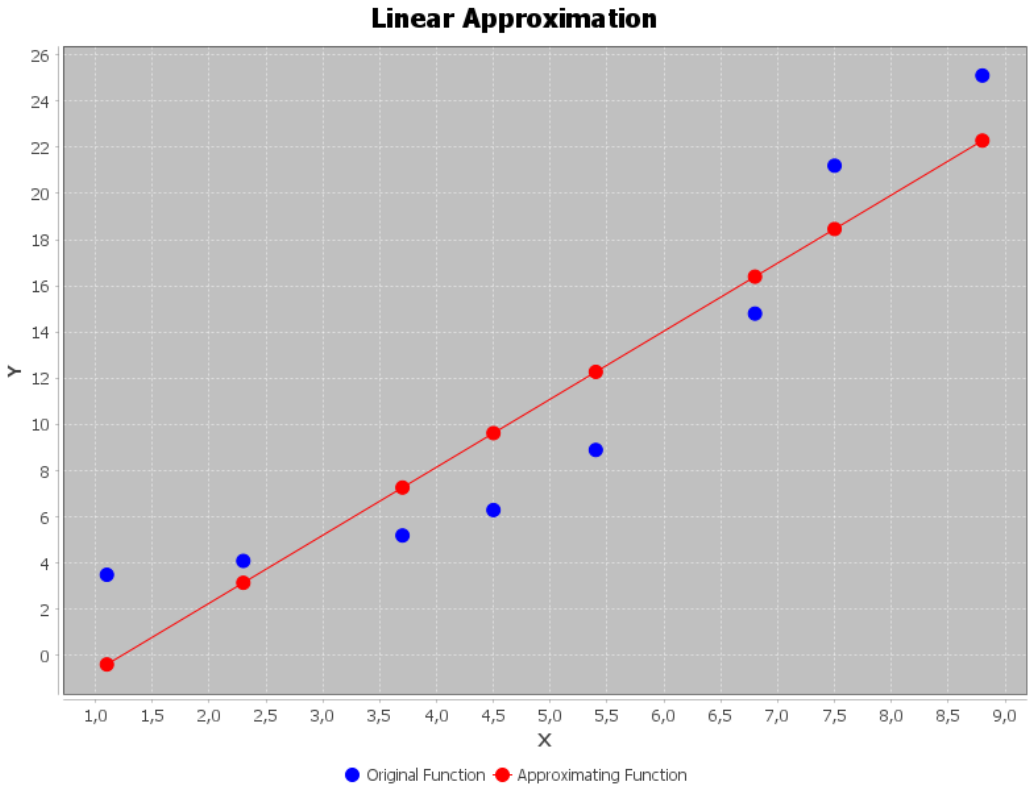
Выберете способ ввода: 1 - через консоль, 2 - через файл.

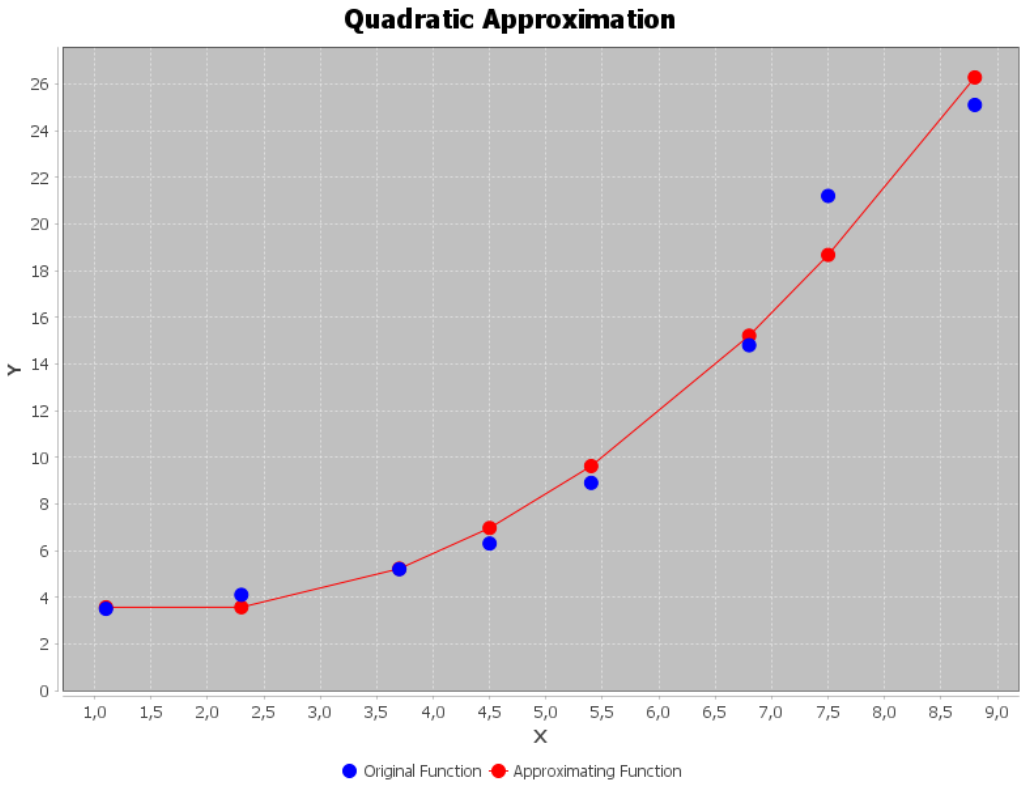
2
Количество точек: 8
1.1 3.5
2.3 4.1
3.7 5.2
4.5 6.3
5.4 8.9
6.8 14.8
7.5 21.2
8.8 25.1
Вывести ответ в консоль (1) или в файл (2) ?
2
Ответ записан в файл output.txt

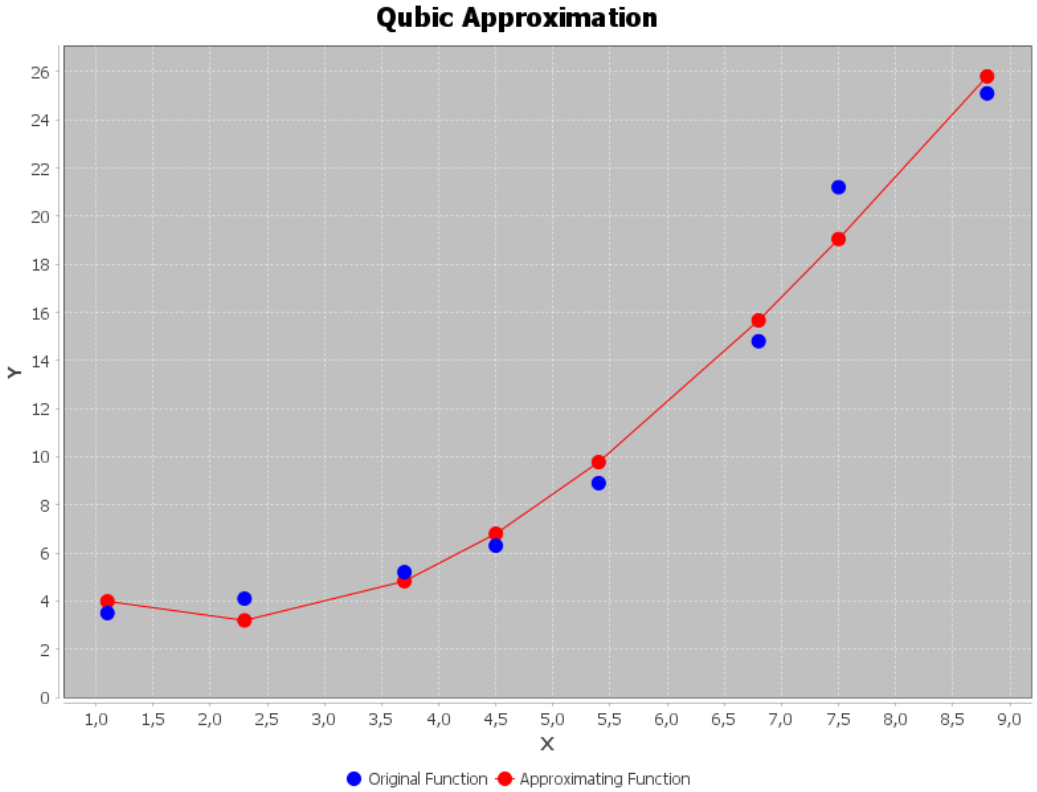
Содержимое файла output.txt:

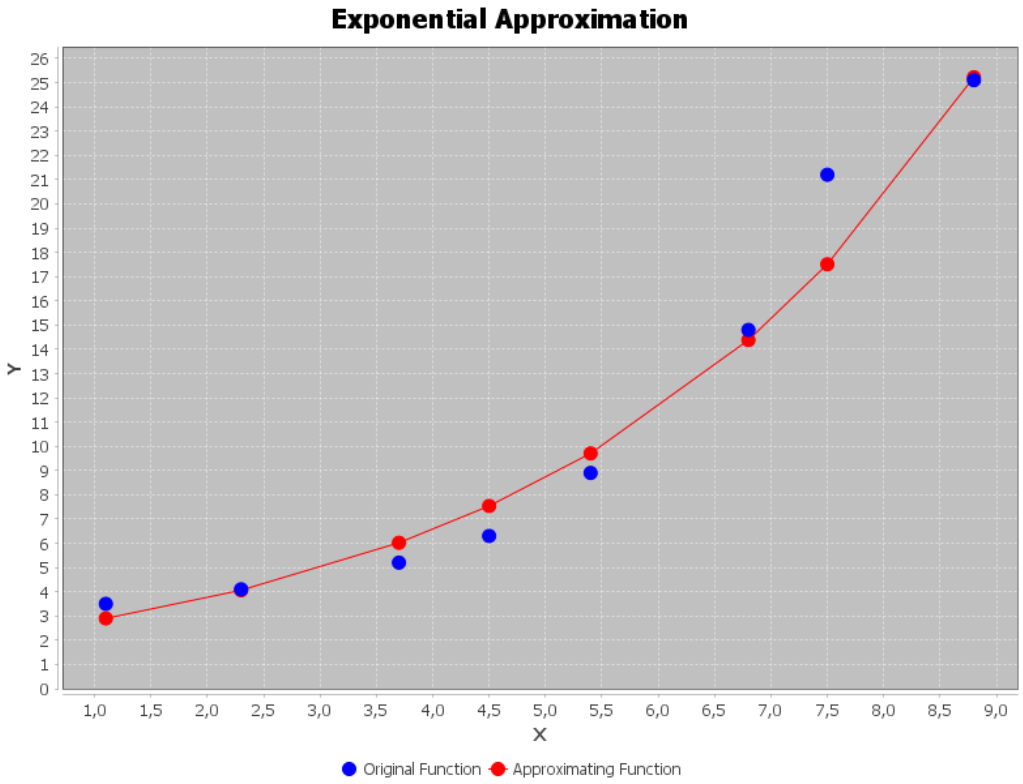
```
Коэффициенты аппроксимирующих функций:
Линейная аппроксимация: a = 2.943332560595528, b = -3.6159544599850837
x = 1.1, y = 3.5, phi(x) = -0.37828864333000256, eps = -3.8782886433300026
x = 2.3, y = 4.1, phi(x) = 3.1537104293846308, eps = -0.9462895706153689
x = 3.7, y = 5.2, phi(x) = 7.274376014218371, eps = 2.0743760142183705
x = 4.5, y = 6.3, phi(x) = 9.629042062694793, eps = 3.3290420626947936
x = 5.4, y = 8.9, phi(x) = 12.278041367230768, eps = 3.378041367230768
x = 6.8, y = 14.8, phi(x) = 16.398706952064508, eps = 1.5987069520645072
x = 7.5, y = 21.2, phi(x) = 18.459039744481377, eps = -2.7409602555186225
x = 8.8, y = 25.1, phi(x) = 22.285372073255566, eps = -2.8146279267444356
Коэффициент корреляции Пирсона: 0.029813848737087454
Коэффициент детерминации: 0.8737907421889473
Удовлетворительная аппроксимация
Квадратичная аппроксимация: a0 = 4.711910722550093, a1 = 0.45354363854168156,
a2 = 0.45354363854168156
x = 1.1, y = 3.5, phi(x) = 3.5663136649292912, eps = 0.06631366492929125
x = 2.3, y = 4.1, phi(x) = 3.568351862627093, eps = -0.5316481373729065
x = 3.7, y = 5.2, phi(x) = 5.221628604232918, eps = 0.02162860423291768
x = 4.5, y = 6.3, phi(x) = 6.964594974698176, eps = 0.6645949746981765
x = 5.4, y = 8.9, phi(x) = 9.619353908440367, eps = 0.7193539084403664
x = 6.8, y = 14.8, phi(x) = 15.209389432587983, eps = 0.4093894325879823
x = 7.5, y = 21.2, phi(x) = 18.671116343318065, eps = -2.528883656681934
x = 8.8, y = 25.1, phi(x) = 26.279251209168027, eps = 1.1792512091680258
Коэффициент детерминации: 0.9808783698309397
Высокая точность аппроксимации
Кубическая аппроксимация: a0 = 6.793068553969159, a1 = -3.536405553716257, a2
= 0.931412990295945, a3 = -0.0322736526997238
x = 1.1, y = 3.5, phi(x) = 3.9870759313960376, eps = 0.48707593139603755
x = 2.3, y = 4.1, phi(x) = 3.1938369666897786, eps = -0.9061630333102211
x = 3.7, y = 5.2, phi(x) = 4.824654512171385, eps = -0.3753454878286151
x = 4.5, y = 6.3, phi(x) = 6.799420013476562, eps = 0.49942001347656184
x = 5.4, y = 8.9, phi(x) = 9.774542912221822, eps = 0.8745429122218216
x = 6.8, y = 14.8, phi(x) = 15.666178294303558, eps = 0.8661782943035572
x = 7.5, y = 21.2, phi(x) = 19.046560372548164, eps = -2.153439627451835
x = 8.8, y = 25.1, phi(x) = 25.80773099719792, eps = 0.7077309971979169
Коэффициент детерминации: 0.9831608771330677
Высокая точность аппроксимации
Экспоненциальная аппроксимация: a = 2.129412697210215, b =
0.28089117169525846
x = 1.1, y = 3.5, phi(x) = 2.9003357366689193, eps = -0.5996642633310807
x = 2.3, y = 4.1, phi(x) = 4.062895543990902, eps = -0.0371044560090974
x = 3.7, y = 5.2, phi(x) = 6.020338867171206, eps = 0.8203388671712055
x = 4.5, y = 6.3, phi(x) = 7.537243151002621, eps = 1.2372431510026214
```

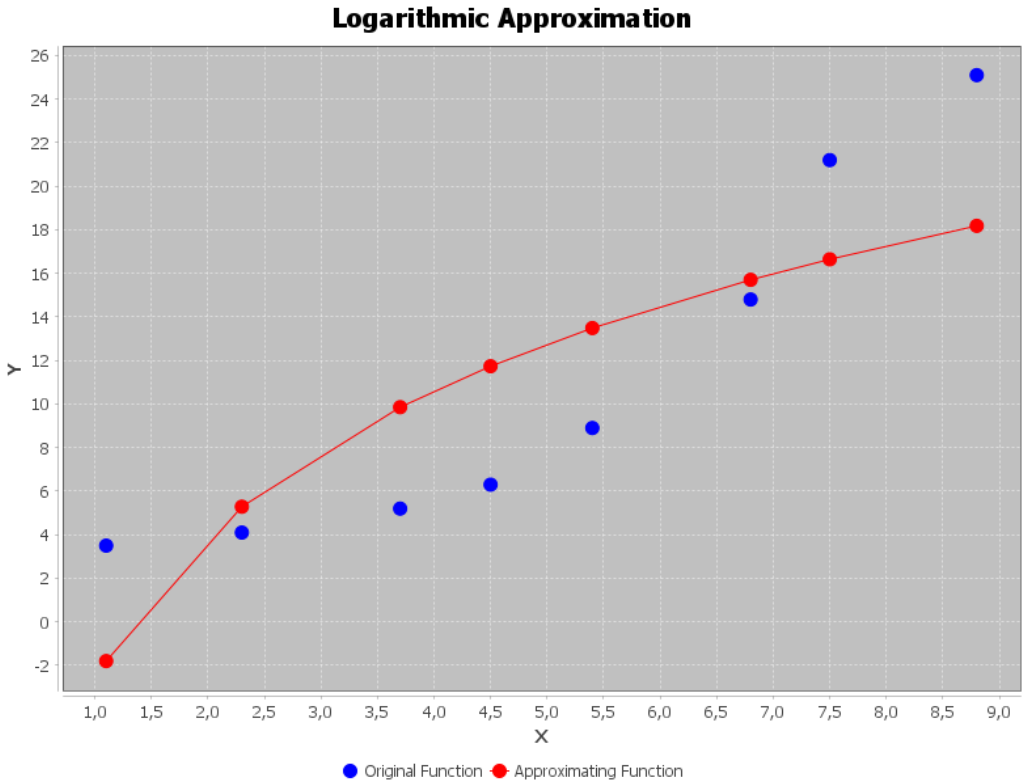
```
x = 5.4, y = 8.9, phi(x) = 9.70516812335712, eps = 0.8051681233571202
x = 6.8, y = 14.8, phi(x) = 14.38097539866477, eps = -0.4190246013352308
x = 7.5, y = 21.2, phi(x) = 17.50576053874445, eps = -3.694239461255549
x = 8.8, y = 25.1, phi(x) = 25.22129112235696, eps = 0.12129112235695771
Коэффициент детерминации: 0.9645904836695585
Высокая точность аппроксимации
Логарифмическая аппроксимация: a = 9.60869764363095, b = -2.7153663914913446
x = 1.1, y = 3.5, phi(x) = -1.7995596913914853, eps = -5.299559691391485
x = 2.3, y = 4.1, phi(x) = 5.28780553541391, eps = 1.1878055354139105
x = 3.7, y = 5.2, phi(x) = 9.856008089766366, eps = 4.656008089766366
x = 4.5, y = 6.3, phi(x) = 11.736858546751414, eps = 5.436858546751414
x = 5.4, y = 8.9, phi(x) = 13.488731259900614, eps = 4.588731259900614
x = 6.8, y = 14.8, phi(x) = 15.703763395205314, eps = 0.9037633952053135
x = 7.5, y = 21.2, phi(x) = 16.645227514138, eps = -4.554772485861999
x = 8.8, y = 25.1, phi(x) = 18.181165350215867, eps = -6.918834649784134
Коэффициент детерминации: 0.644128590222558
Слабая аппроксимация
Степенная аппроксимация: a = 2.129412697210215, b = 0.28089117169525846
x = 1.1, y = 3.5, phi(x) = 2.8534861122389255, eps = -0.6465138877610745
x = 2.3, y = 4.1, phi(x) = 4.98371640596542, eps = 0.8837164059654201
x = 3.7, y = 5.2, phi(x) = 7.1392119797937275, eps = 1.9392119797937273
x = 4.5, y = 6.3, phi(x) = 8.277887973920237, eps = 1.9778879739202369
x = 5.4, y = 8.9, phi(x) = 9.501268713852795, eps = 0.6012687138527948
x = 6.8, y = 14.8, phi(x) = 11.310191651931994, eps = -3.4898083480680064
x = 7.5, y = 21.2, phi(x) = 12.179798319314084, eps = -9.020201680685915
x = 8.8, y = 25.1, phi(x) = 13.744330468529979, eps = -11.355669531470022
Коэффициент детерминации: 0.5601134597008319Слабая аппроксимация
Среднеквадратичное отклонение:
Линейная аппроксимация: 2.8182987621930433
Квадратичная аппроксимация: 1.1367094324143496
Кубическая аппроксимация: 1.0707909133337803
Экспоненциальная аппроксимация: 1.4599098644232136
Логарифмическая аппроксимация: 4.68925338737997
Степенная аппроксимация: 5.444859195486482
Минимальное среднеквадратичное отклонение: 1.0707909133337803
Наилучшая аппроксимирующая функция: кубическая
```

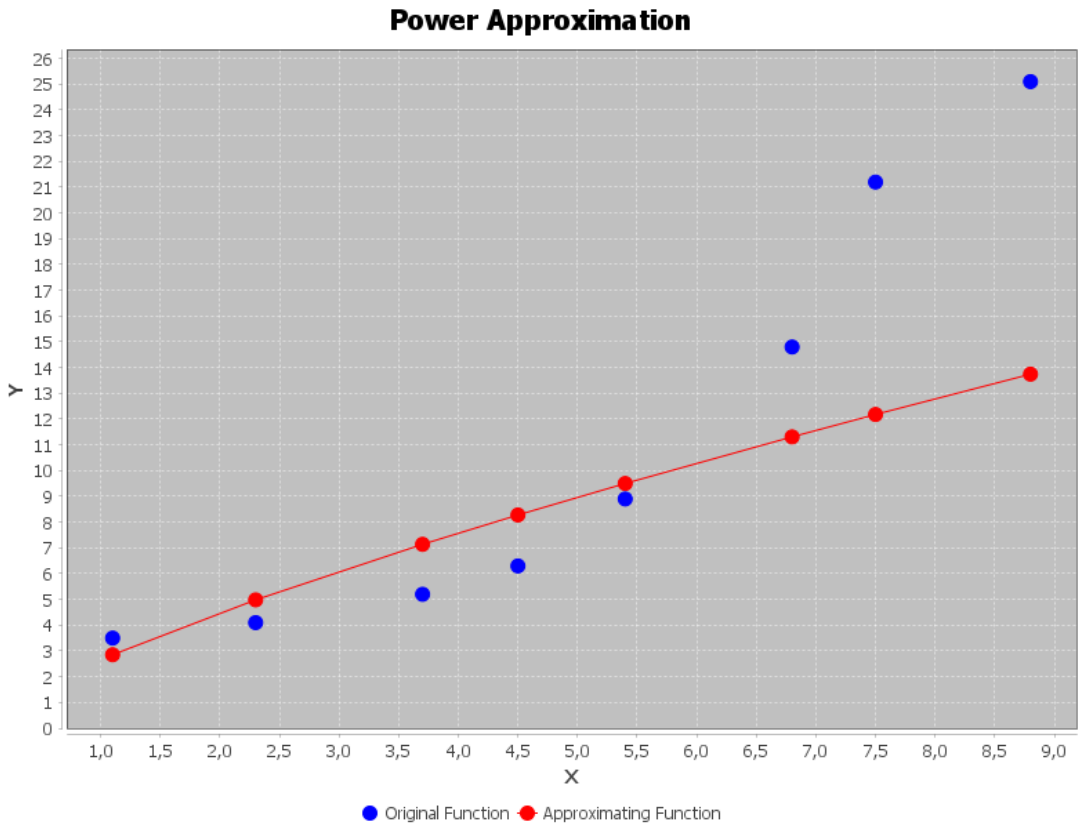



Quadratic Approximation
Y
X
Original Function
Approximating Function


Qubic Approximation
Y
X
Original Function
Approximating Function


Exponential Approximation
Y
X
Original Function
Approximating Function


Logarithmic Approximation
Y
X
Original Function
Approximating Function



Вывод:

Во время выполнения работы мне удалось изучить различные виды аппроксимации: линейную, квадратичную, кубическую, логарифмическую, экспоненциальную и степенную.

Нельзя однозначно сказать, какая аппроксимирующая функция лучше, так как это зависит от самих экспериментальных данных.