ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

Факультет Программной Инженерии и Компьютерной Техники

Дисциплина:
«*Вычислительная математика*»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4
«Аппроксимация функции методом наименьших квадратов»

Вариант 3

Выполнил:
Студент гр. P32151
Горинов Даниил Андреевич

Проверил:
Машина Екатерина Алексеевна

Санкт-Петербург
2023г.

Цель лабораторной работы:

Найти функцию, являющуюся наилучшим приближением заданной табличной функции по методу наименьших квадратов.

Порядок выполнения лабораторной работы:

Программная реализация задачи:

Для исследования использовать:

- линейную функцию,
- полиномиальную функцию 2-й степени,
- полиномиальную функцию 3-й степени,
- экспоненциальную функцию,
- логарифмическую функцию,
- степенную функцию.

Методика проведения исследования:

1. Вычислить меру отклонения для всех исследуемых функций;
2. Уточнить значения коэффициентов эмпирических функций, минимизируя функцию S;
3. Сформировать массивы предполагаемых эмпирических зависимостей;
4. Определить среднеквадратичное отклонение для каждой аппроксимирующей функции. Выбрать наименьшее значение n, следовательно, наилучшее приближение;
5. Построить графики полученных эмпирических функций.

Задание:

1. Предусмотреть ввод исходных данных из файла/консоли (таблица должна содержать от 8 до 12 точек);
2. Реализовать метод наименьших квадратов, исследуя все указанные функции;
3. Предусмотреть вывод результатов в файл/консоль: коэффициенты аппроксимирующих функций, среднеквадратичное отклонение, массивы значений $x_i, y_i, \varphi(x_i), \varepsilon_i$;
4. Для линейной зависимости вычислить коэффициент корреляции Пирсона;
5. Программа должна отображать наилучшую аппроксимирующую функцию;
6. Организовать вывод графиков функций, графики должны полностью отображать весь исследуемый интервал (с запасом);

7. Программа должна быть протестирована при различных наборах данных, в том числе и некорректных;

Вычислительная реализация задачи:

1. Сформировать таблицу табулирования заданной функции на указанном интервале;
2. Построить линейное и квадратичное приближения по 11 точкам заданного интервала;
3. Найти среднеквадратические отклонения для каждой аппроксимирующей функции. Ответы дать с тремя знаками после запятой;
4. Выбрать наилучшее приближение;
5. Построить графики заданной функции, а также полученные линейное и квадратичное приближения;
6. Привести в отчете подробные вычисления.

<u>Рабочие формулы методов:</u>

Параметры $a_0, a_1, a_2, \dots a_m$ эмпирической формулы находятся из условия минимума функции $S = S(a_0, a_1, a_2, \dots a_m)$. Так как здесь параметры выступают в роли независимых переменных функции S, то ее минимум найдем, приравнивая к нулю частные производные по этим переменным.

$$\frac{\partial S}{\partial a_0} = 2\sum_{i=1}^{n} a_0 + a_1x_i + \cdots + a_{m-1}x_i^{m-1} + a_mx_i^m - y_i = 0$$

$$\frac{\partial S}{\partial a_1} = 2\sum_{i=1}^{n} (a_0 + a_1x_i + \cdots + a_{m-1}x_i^{m-1} + a_mx_i^m - y_i)x_i = 0$$

$$\dots\dots\dots$$

$$\frac{\partial S}{\partial a_m} = 2\sum_{i=1}^{n} (a_0 + a_1x_i + \cdots + a_{m-1}x_i^{m-1} + a_mx_i^m - y_i)x_i^m = 0$$

Преобразуем полученную линейную систему уравнений: раскроем скобки и перенесем свободные слагаемые в правую часть выражения:

$$\begin{cases} a_0n + a_1\sum_{i=1}^{n} x_i + \cdots + a_{m-1}\sum_{i=1}^{n} x_i^{m-1} + a_m\sum_{i=1}^{n} x_i^m = \sum_{i=1}^{n} y_i \\ a_0\sum_{i=1}^{n} x_i + a_1\sum_{i=1}^{n} x_i^2 + \cdots + a_{m-1}\sum_{i=1}^{n} x_i^m + a_m\sum_{i=1}^{n} x_i^{m+1} = \sum_{i=1}^{n} x_iy_i \\ \dots\dots\dots \\ a_0\sum_{i=1}^{n} x_i^m + a_1\sum_{i=1}^{n} x_i^{m+1} + \cdots + a_{m-1}\sum_{i=1}^{n} x_i^{2m-1} + a_m\sum_{i=1}^{n} x_i^{2m} = \sum_{i=1}^{n} x_i^my_i \end{cases}$$

В матричном виде:

$$\begin{vmatrix} n & \sum_{i=1}^{n} x_i & \dots & \sum_{i=1}^{n} x_i^m \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & \dots & \sum_{i=1}^{n} x_i^{m+1} \\ & \dots & \dots & \dots \\ \sum_{i=1}^{n} x_i^m & \sum_{i=1}^{n} x_i^{m+1} & \dots & \sum_{i=1}^{n} x_i^{2m} \end{vmatrix} \cdot \begin{vmatrix} a_0 \\ a_1 \\ \dots \\ \dots \\ \dots \\ \dots \\ a_m \end{vmatrix} = \begin{vmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ \dots \\ \sum_{i=1}^{n} x_i^m y_i \end{vmatrix}$$

<u>Вычислительная часть:</u>

Функция: $y = \frac{4x}{x^4+3}$, исследуемый интервал: $x \in [-2, 0], h = 0{,}2$

Составим таблицу с точками и значениями функции в этих точках на промежутке x∈[-2,0] с шагом 0.2:

x_i	-2	-1.8	-1.6	-1.4	-1.2	-1	-0.8	-0.6	-0.4	-0.2	0
$f(x_i)$	-0.421	-0.533	-0.670	-0.819	-0.946	-1.000	-0.939	-0.767	-0.529	-0.267	0

$SX = -2 - 1.8 - 1.6 - 1.4 - 1.2 - 1 - 0.2 - 0.6 - 0.4 - 0.2 = -11$

$SXX = 4 + 3.24 + 2.25 + 1.96 + 1.44 + 1 + 0.64 + 0.36 + 0.16 + 0.04 = 15.4$

$SXXX = -8 - 5.832 - 4.096 - 2.744 - 1.728 - 1 - 0.512 - 0.216 - 0.064 - 0.008 = -24.2$

$SXXXX = 16 + 10.498 + 6.554 + 3.842 + 2.074 + 1 + 0.41 + 0.13 + 0.026 + 0.0016 = 40.553$

$SY = -0.421 - 0.533 - 0.670 - 0.819 - 0.946 - 1 - 0.939 - 0.767 - 0.529 - 0.267 = -6.890$

$SXY = 0.842 + 0.960 + 1.072 + 1.146 + 1.135 + 1 + 0.751 + 0.460 + 0.212 + 0.053 = 7.631$

$SXXY = -1.684 - 1.728 - 1.715 - 1.604 - 1.362 - 1 - 0.601 - 0.276 - 0.085 - 0.011 = -10.066$

Линейная аппроксимация:

$$\begin{cases} 15.4a - 11b = 7.631 \\ -11a + 11b = -6.89 \end{cases}$$

$$\Delta = 15.4 \cdot 11 - 11 \cdot 11 = 48.4$$

$$\Delta_1 = 7.631 \cdot 11 - 11 \cdot 6.89 = 8.151$$

$$\Delta_2 = -15.4 \cdot 6.89 + 11 \cdot 7.631 = -22.165$$

$$a = \frac{\Delta_1}{\Delta} = \frac{8.151}{48.4} = 0.168$$

$$b = \frac{\Delta_2}{\Delta} = -\frac{22.165}{48.4} = -0.458$$

$$\varphi(x) = 0.168x - 0.458$$

x_i	-2	-1.8	-1.6	-1.4	-1.2	-1	-0.8	-0.6	-0.4	-0.2	0
$f(x_i)$	-0.421	-0.533	-0.670	-0.819	-0.946	-1.000	-0.939	-0.767	-0.529	-0.267	0
$\varphi(x)$	-0.794	-0.760	-0.727	-0.693	-0.660	-0.626	-0.592	-0.559	-0.525	-0.492	-0.458
ε_i	-0.373	-0.227	-0.057	0.125	0.286	0.374	0.346	0.208	0.004	-0.225	-0.458
ε_{i^2}	0.139	0.052	0.003	0.016	0.082	0.140	0.120	0.043	0.000	0.051	0.210

$S = \sum \varepsilon_{i^2} = 0.855$

$\delta = \sqrt{\frac{S}{n}} = 0.279$

Квадратичная аппроксимация: 5.723 = 7.631, -5.167 = -6.89, -7.55=-10.066

$$\begin{cases} 11c - 11b + 15.4a = -6.89 \\ -11c + 15.4b - 24.2a = 7.631 \\ 15.4c - 24.2b + 40.553a = -10.066 \end{cases}$$

$$\Delta = \begin{vmatrix} 11 & -11 & 15.4 \\ -11 & 15.4 & -24.2 \\ 15.4 & -24.2 & 40.533 \end{vmatrix} = 66.453$$

$$\Delta_1 = \begin{vmatrix} -6.89 & -11 & 15.4 \\ 7.631 & 15.4 & -24.2 \\ -10.066 & -24.2 & 40.533 \end{vmatrix} = 0.409$$

$$\Delta_2 = \begin{vmatrix} 11 & -6.89 & 15.4 \\ -11 & 7.631 & -24.2 \\ 15.4 & -10.066 & 40.533 \end{vmatrix} = 113.993$$

$$\Delta_3 = \begin{vmatrix} 11 & -11 & -6.89 \\ -11 & 15.4 & 7.631 \\ 15.4 & -24.2 & -10.066 \end{vmatrix} = 51.401$$

$$c = \frac{\Delta_1}{\Delta} = \frac{0.318}{66.44} = 0.006$$

$$b = \frac{\Delta_2}{\Delta} = \frac{85.5}{66.44} = 1.715$$

$$a = \frac{\Delta_3}{\Delta} = \frac{38.556}{66.44} = 0.773$$

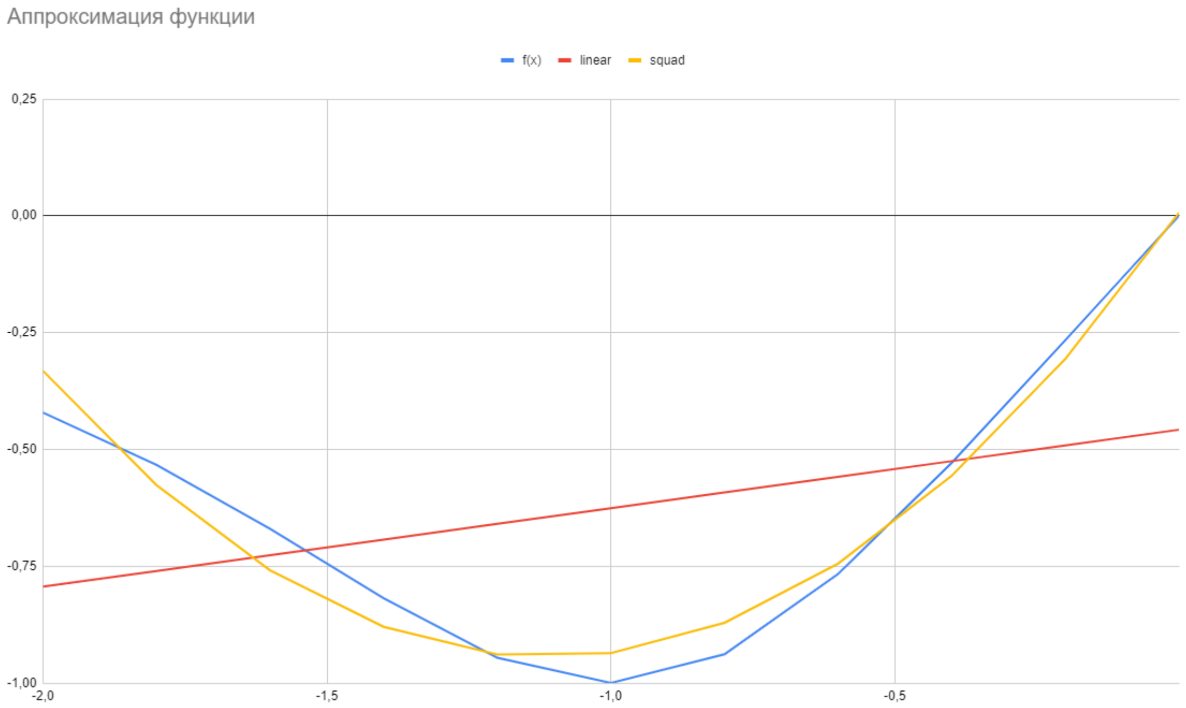
$$\varphi(x) = 0.773x^2 + 1.715x + 0.006$$

x_i	-2	-1.8	-1.6	-1.4	-1.2	-1	-0.8	-0.6	-0.4	-0.2	0
$f(x_i)$	-0.421	-0.533	-0.670	-0.819	-0.946	-1.000	-0.939	-0.767	-0.529	-0.267	0
$\varphi(x)$	-0.332	-0.576	-0.759	-0.880	-0.939	-0.936	-0.871	-0.745	-0.556	-0.306	0.006
ε_i	0.089	-0.043	-0.089	-0.061	0.007	0.064	0.067	0.022	-0.027	-0.040	0.006
ε_{i^2}	0.008	0.002	0.008	0.004	0.000	0.004	0.005	0.000	0.001	0.002	0.000

$$S = \sum \varepsilon_{i^2} = 0.033$$

$$\delta = \sqrt{\frac{S}{n}} = 0.055$$

График полученных функций:



Листинг программы:

```
public ApproximationResult cubicApproximation(double[][] functionTable) {
        double x_sum = 0, x2_sum = 0, x3_sum = 0, x4_sum = 0, x5_sum = 0, x6_sum = 0,
                y_sum = 0, xy_sum = 0, x2y_sum = 0, x3y_sum = 0;
        for (double[] doubles : functionTable) {
            x_sum += doubles[0];
            x2_sum += Math.pow(doubles[0], 2);
            x3_sum += Math.pow(doubles[0], 3);
            x4_sum += Math.pow(doubles[0], 4);
            x5_sum += Math.pow(doubles[0], 5);
            x6_sum += Math.pow(doubles[0], 6);
            y_sum += doubles[1];
            xy_sum += doubles[0] * doubles[1];
            x2y_sum += Math.pow(doubles[0], 2) * doubles[1];
            x3y_sum += Math.pow(doubles[0], 3) * doubles[1];
        }

        double[][] matrix = new double[][] {
                {functionTable.length, x_sum, x2_sum, x3_sum},
                {x_sum, x2_sum, x3_sum, x4_sum},
                {x2_sum, x3_sum, x4_sum, x5_sum},
                {x3_sum, x4_sum, x5_sum, x6_sum}
        };

        double[] constants = new double[] {y_sum, xy_sum, x2y_sum, x3y_sum };
        double[] solution = solveLinearSystem(matrix, constants);
        reverseArray(solution);
```

```java
        Function<Double, Double> function = coefficientsToCubicFunction(solution);
        double deviation = deviationMeasure(functionTable, function);
        return new ApproximationResult(ApproximationType.CUBIC, solution, function, deviation);
    }

    public ApproximationResult squareApproximation(double[][] functionTable) {
        double x_sum = 0, x2_sum = 0, x3_sum = 0, x4_sum = 0,
                y_sum = 0, xy_sum = 0, x2y_sum = 0;

        for (double[] doubles : functionTable) {
            x_sum += doubles[0];
            x2_sum += Math.pow(doubles[0], 2);
            x3_sum += Math.pow(doubles[0], 3);
            x4_sum += Math.pow(doubles[0], 4);
            y_sum += doubles[1];
            xy_sum += doubles[0] * doubles[1];
            x2y_sum += Math.pow(doubles[0], 2) * doubles[1];
        }

        double[][] matrix = new double[][] {
                {functionTable.length, x_sum, x2_sum},
                {x_sum, x2_sum, x3_sum},
                {x2_sum, x3_sum, x4_sum}
        };

        double[] constants = new double[] {y_sum, xy_sum, x2y_sum};
        double[] solution = solveLinearSystem(matrix, constants);
        reverseArray(solution);
        Function<Double, Double> function = coefficientsToSquareFunction(solution);
        double deviation = deviationMeasure(functionTable, function);
        return new ApproximationResult(ApproximationType.QUADRATIC, solution, function, deviation);
    }

    public ApproximationResult linearApproximation(double[][] functionTable) {
        double x_sum = 0, x2_sum = 0, y_sum = 0, xy_sum = 0;

        for (double[] doubles : functionTable) {
            x_sum += doubles[0];
            x2_sum += Math.pow(doubles[0], 2);
            y_sum += doubles[1];
            xy_sum += doubles[0] * doubles[1];
        }

        double[][] matrix = {
                {x2_sum, x_sum},
                {x_sum, functionTable.length}
        };
```

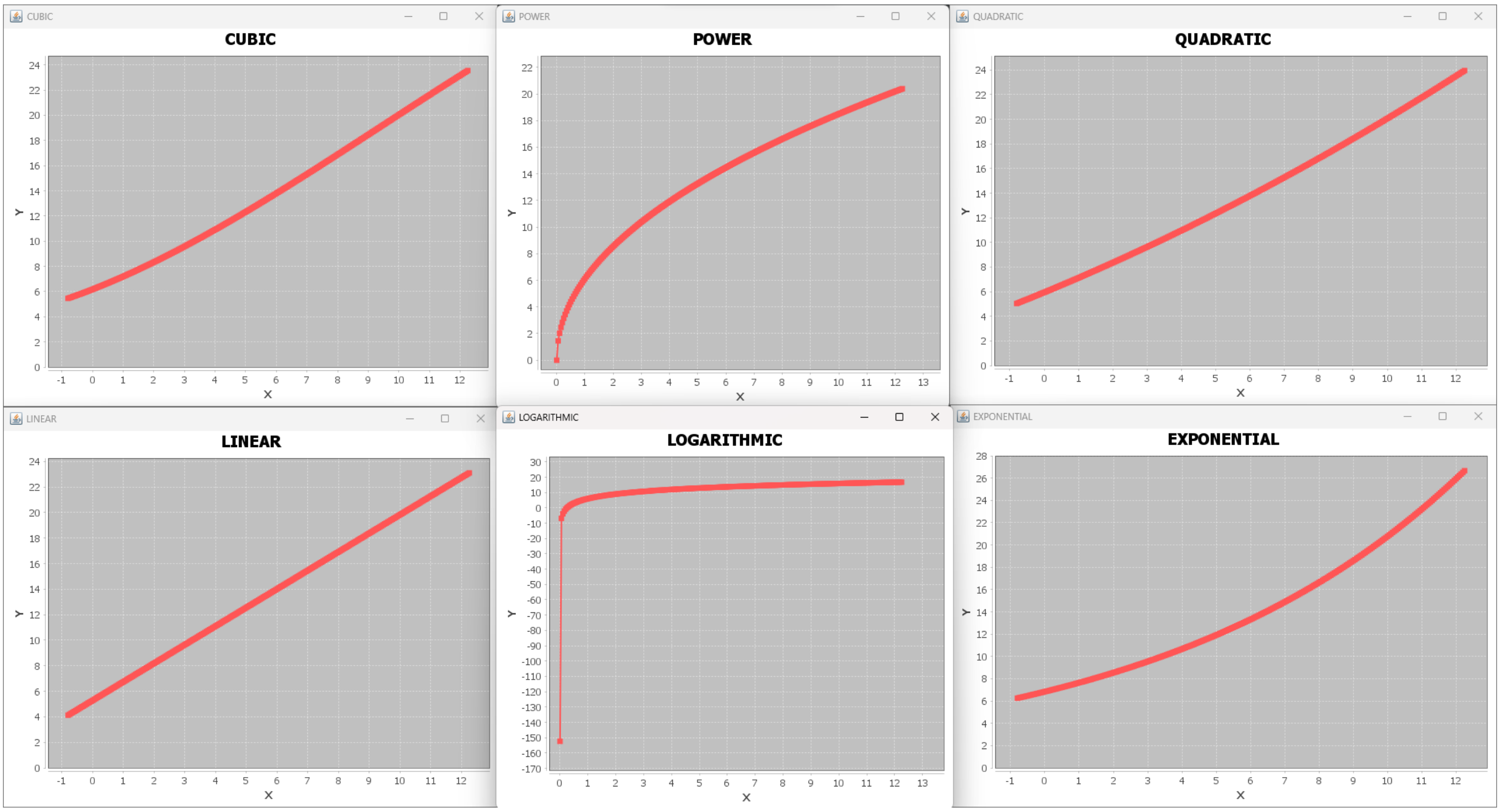
```java

        double[] constants = {
                xy_sum, y_sum
        };
        double[] solution = solveLinearSystem(matrix, constants);
        Function<Double, Double> function = coefficientsToLinearFunction(solution);
        double deviation = deviationMeasure(functionTable, function);
        return new ApproximationResult(ApproximationType.LINEAR, solution, function, deviation, linearCorrelation(functionTable));
    }

    public ApproximationResult exponentialApproximation(double[][] functionTable) {
        double[][] modifiedFunctionTable = Arrays.stream(functionTable).map(double[]::clone).toArray(double[][]::new);
        for (double[] xy: modifiedFunctionTable) {
            if (xy[1] <= 0) continue;
            xy[1] = Math.log(xy[1]);
        }
        ApproximationResult linear = linearApproximation(modifiedFunctionTable);
        double[] coefficients = linear.getCoefficients();
        coefficients[1] = Math.exp(coefficients[1]);
        Function<Double, Double> f = coefficientsToExpFunction(coefficients);
        return new ApproximationResult(ApproximationType.EXPONENTIAL, coefficients, f, deviationMeasure(functionTable, f));
    }

    public ApproximationResult logarithmicApproximation(double[][] functionTable) {
        double[][] modifiedFunctionTable = Arrays.stream(functionTable).map(double[]::clone).toArray(double[][]::new);
        for (double[] xy: modifiedFunctionTable) {
            xy[0] = Math.log(xy[0]);
        }
        ApproximationResult linear = linearApproximation(modifiedFunctionTable);
        double[] coefficients = linear.getCoefficients();
        Function<Double, Double> f = coefficientsToLogFunction(coefficients);
        return new ApproximationResult(ApproximationType.LOGARITHMIC, coefficients, f, deviationMeasure(functionTable, f));
    }

    public ApproximationResult powerApproximation(double[][] functionTable) {
        double[][] modifiedFunctionTable = Arrays.stream(functionTable).map(double[]::clone).toArray(double[][]::new);
        for (double[] xy: modifiedFunctionTable) {
            xy[0] = Math.log(xy[0]);
            xy[1] = Math.log(xy[1]);
```

```
            }
            ApproximationResult linear = linearApproximation(modifiedFunctionTable);
            double[] coefficients = linear.getCoefficients();
            coefficients[1] = Math.exp(coefficients[1]);
            Function<Double, Double> f = coefficientsToPowerFunction(coefficients);
            return new ApproximationResult(ApproximationType.POWER, coefficients, f, deviationMeasure(functionTable, f));
        }
```

Результаты выполнения программы:

```
Введите название файла или 0 для ввода с клавиатуры:
0
Введите название файла или 0 для вывода в консоль:
0
Введите количество пар (x, y) (не менее 8)
8
1.2 7.4
2.9 9.5
4.1 11.1
5.5 12.9
6.7 14.6
7.8 17.3
9.2 18.2
10.3 20.7
Approximation result.
Type: LINEAR
Function: 1,454330x + 5,291060
Deviation: 1,345854
Correlation: 0.9954179478701582

Approximation result.
Type: QUADRATIC
Function: 0,025974x^2 + 1,152563x + 5,943053
Deviation: 1,015890

Approximation result.
Type: EXPONENTIAL
Function: 6,839635e^(0,111077x)
Deviation: 2,718870

Approximation result.
Type: LOGARITHMIC
Function: 6,008624lnx + 4,295866
Deviation: 37,832367

Approximation result.
Type: POWER
Function: 6,128671x^(0,479894)
Deviation: 8,046184

Approximation result.
```

```
Type: CUBIC
Function: -0,002371x^3 + 0,066875x^2 + 0,954754x + 6,177879
Deviation: 0,999587
```



Вывод:

В результате проведения данной лабораторной работы, я ознакомился с методом наименьших квадратов и успешно реализовал его на языке Python. Метод наименьших квадратов обладает несколькими преимуществами, такими как: простота расчетов - требуется только найти коэффициенты, простота функции и широкий выбор возможных аппроксимирующих функций. Однако, основным недостатком метода наименьших квадратов является его чувствительность к резким выбросам, которые могут присутствовать в исходных данных.