«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Вычислительная математика»

Отчет

По лабораторной работе №4

Вариант 4

Студент:

Ильин Н. С.

P3210

Преподаватель:

Наумова Н. А.

Санкт-Петербург, 2025 г.

Оглавление

Цель работы:

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

1 Вычислительная реализация задачи:

Линейная аппроксимация:

$y = \frac{15x}{x^4+4}$; n = 11 ; x ∈ [-4; 0] ; 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
yi	0	1.490	2.721	2.964	2.274	1.500	0.968	0.642	0.441	0.314	0.231

φ(x) = a + bx

Вычисляем суммы: sx = 22, sxx = 61.6, sy = 13,55 sxy = 20,55

$$\begin{cases} n * a + sx * b = sy \\ sx * a + sxx * b = sxy \end{cases} \quad \begin{cases} 11 * a + 22 * b = 13{,}55 \\ 22 * a + 61.6 * b = 20{,}55 \end{cases}$$

$$\begin{cases} a = 1.976 \\ b = -0.372 \end{cases}$$

$$\varphi(x) = 1.976 - 0.372 * x$$

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
yi	0	1,49	2,721	2,964	2,274	1,5	0,968	0,642	0,441	0,314	0,231
φ(xi)	1,976	1,827	1,678	1,530	1,381	1,232	1,083	0,934	0,786	0,637	0,488
(φ(xi)-yi)^2	3,905	0,114	1,087	2,058	0,798	0,072	0,013	0,085	0,119	0,104	0,066

$\boldsymbol{\sigma = \sqrt{\frac{\sum(\varphi(xi) - yi)^2}{n}} = 0{,}875}$

Квадратичная аппроксимация:

$y = \frac{15x}{x^4+4}$; n = 11 ; x ∈ [-4; 0] ; 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
yi	0	1.490	2.721	2.964	2.274	1.500	0.968	0.642	0.441	0.314	0.231

$\varphi(x) = a + bx + cx^2$

Вычисляем суммы:

sx = 22, sxx = 61.6, sxxx = 193,6; sxxxx = 648,5248; sy = 13,55; sxy = 20,55; sxxy = 40,96

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

$$\begin{cases} 11 * a + 22 * b + 61.6 * c = 13{,}55 \\ 22 * a + 61.6 * b + 193{,}6 * c = 20{,}55 \\ 61.6 * a + 193{,}6 * b + 648.52 * c = 40{,}96 \end{cases}$$

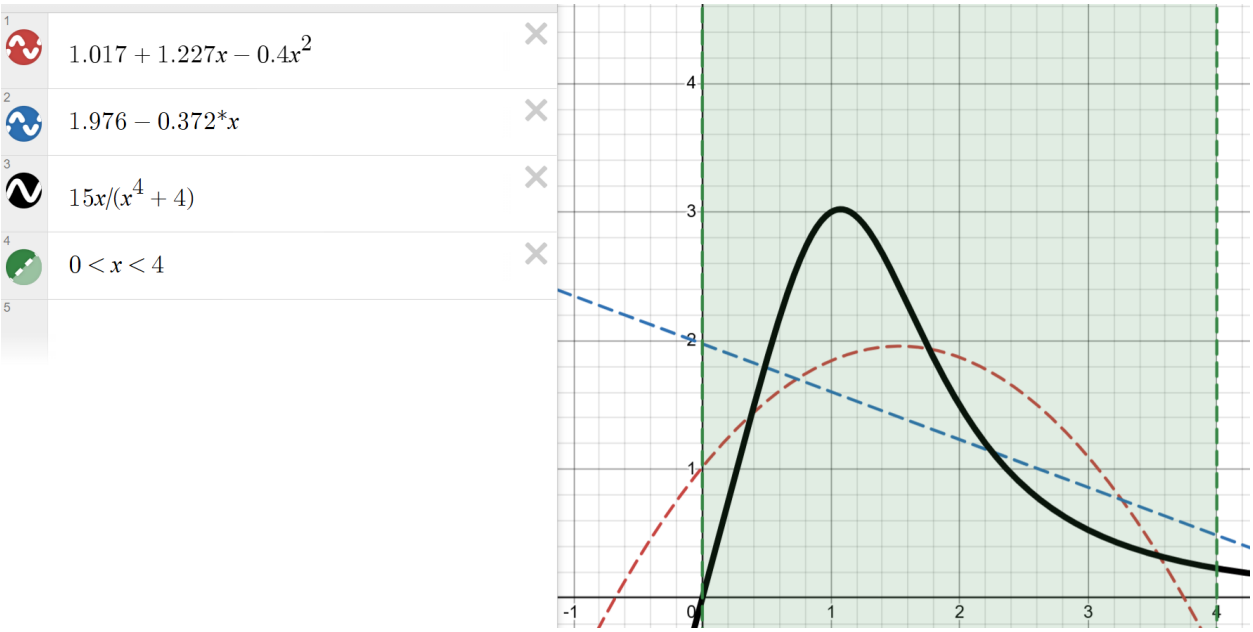
$$\begin{cases} a = 1.017 \\ b = 1.227 \\ c = -0.4 \end{cases}$$

$$\varphi(x) = 1.017 + 1.227x - 0.4x^2$$

i	1	2	3	4	5	6	7	8	9	10	11
xi	0	0.4	0.8	1.2	1.6	2.0	2.4	2.8	3.2	3.6	4.0
yi	0	1.490	2.721	2.964	2.274	1.500	0.968	0.642	0.441	0.314	0.231
φ(xi)	1,017	1,444	1,743	1,913	1,956	1,871	1,658	1,317	0,847	0,250	-0,475
(φ(xi)-yi)^2	1,034	0,002	0,957	1,104	0,101	0,138	0,476	0,455	0,165	0,004	0,498

$$\sigma = \sqrt{\frac{\sum(\varphi(xi) - yi)^2}{n}} = \underline{\mathbf{0{,}67}}$$

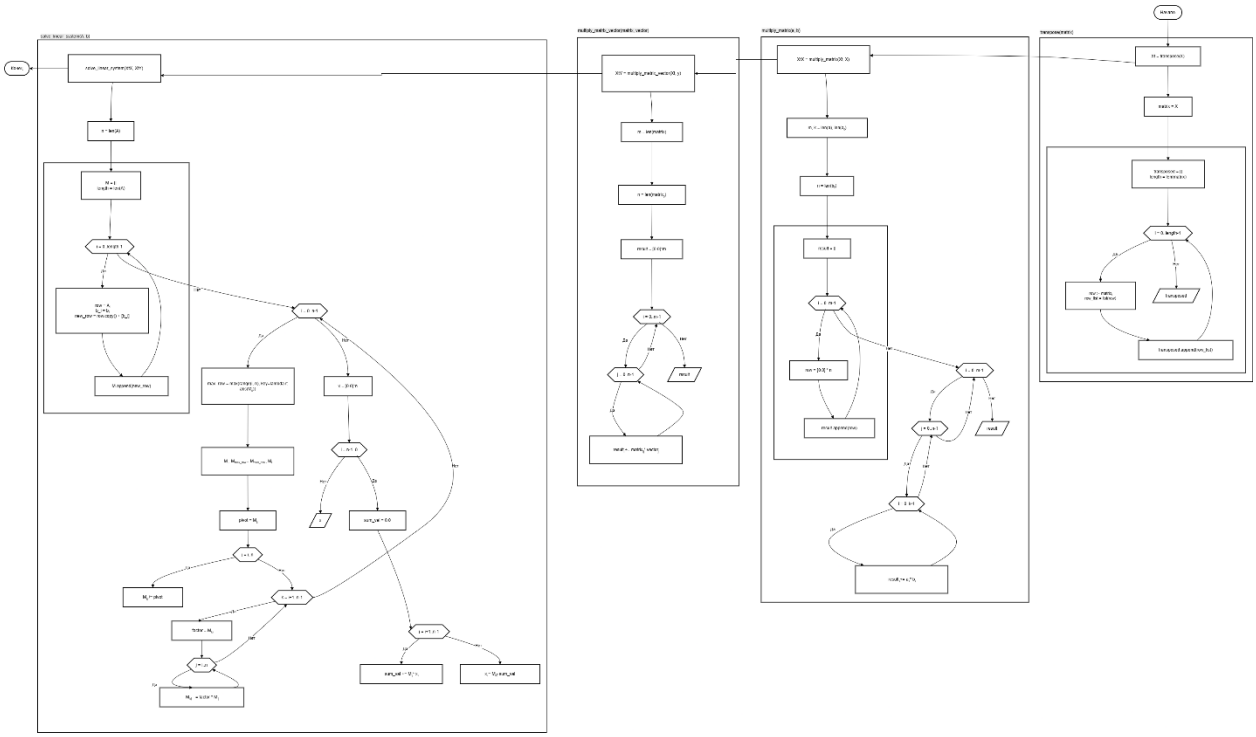
<u>0,67</u> < 0,875 у квадратичной аппроксимации среднеквадратичное отклонение меньше, поэтому это приближение лучше.



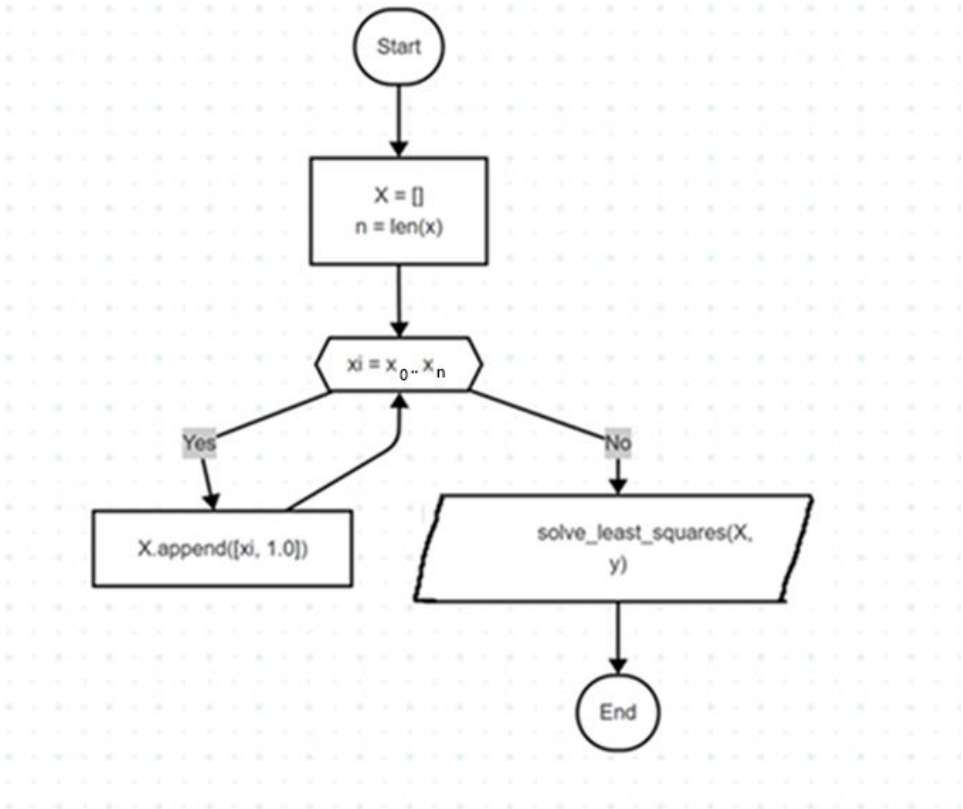
2. Программная реализация

Блок схемы

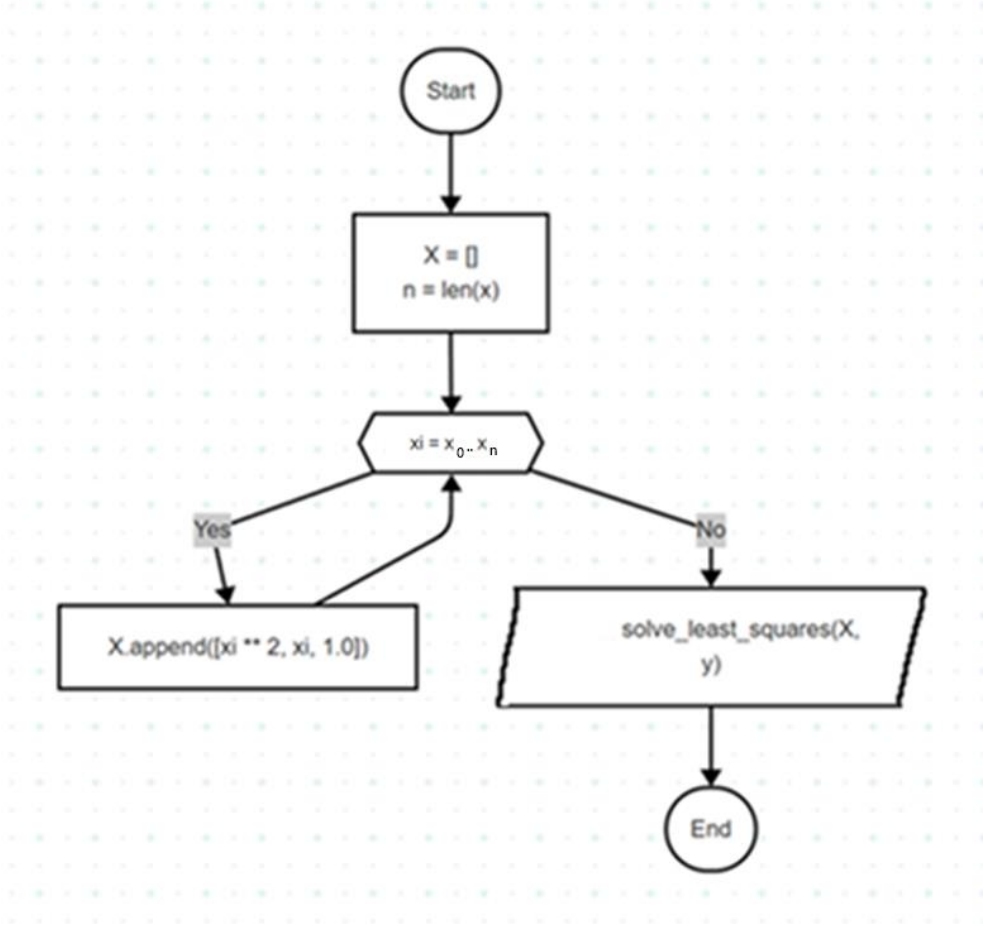
МНК



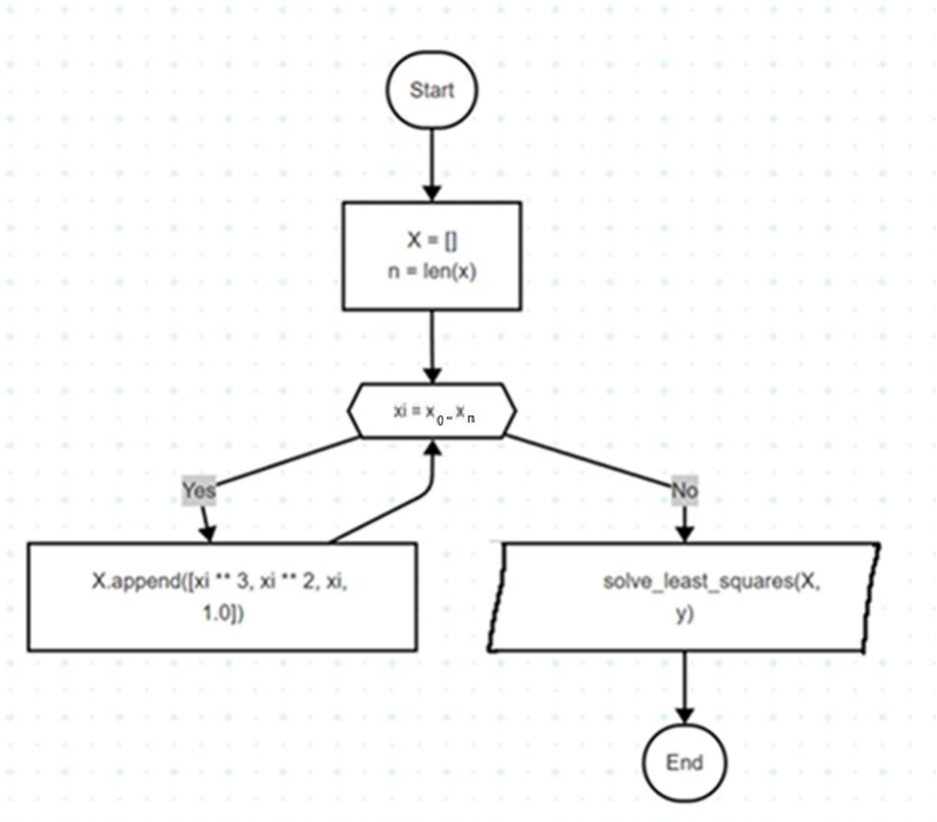
Линейная фукнция



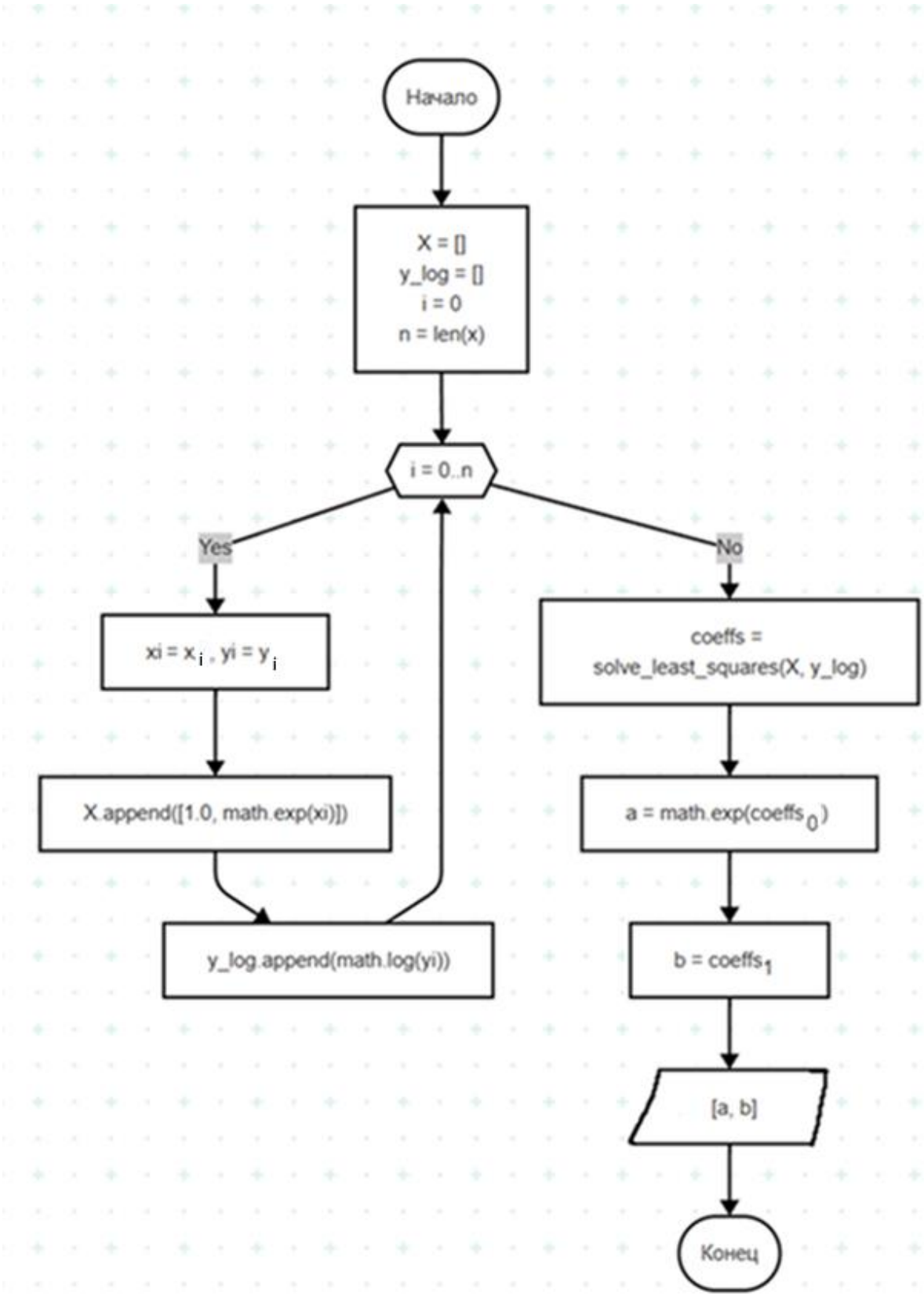
Полиномиальная функция 2-й степени



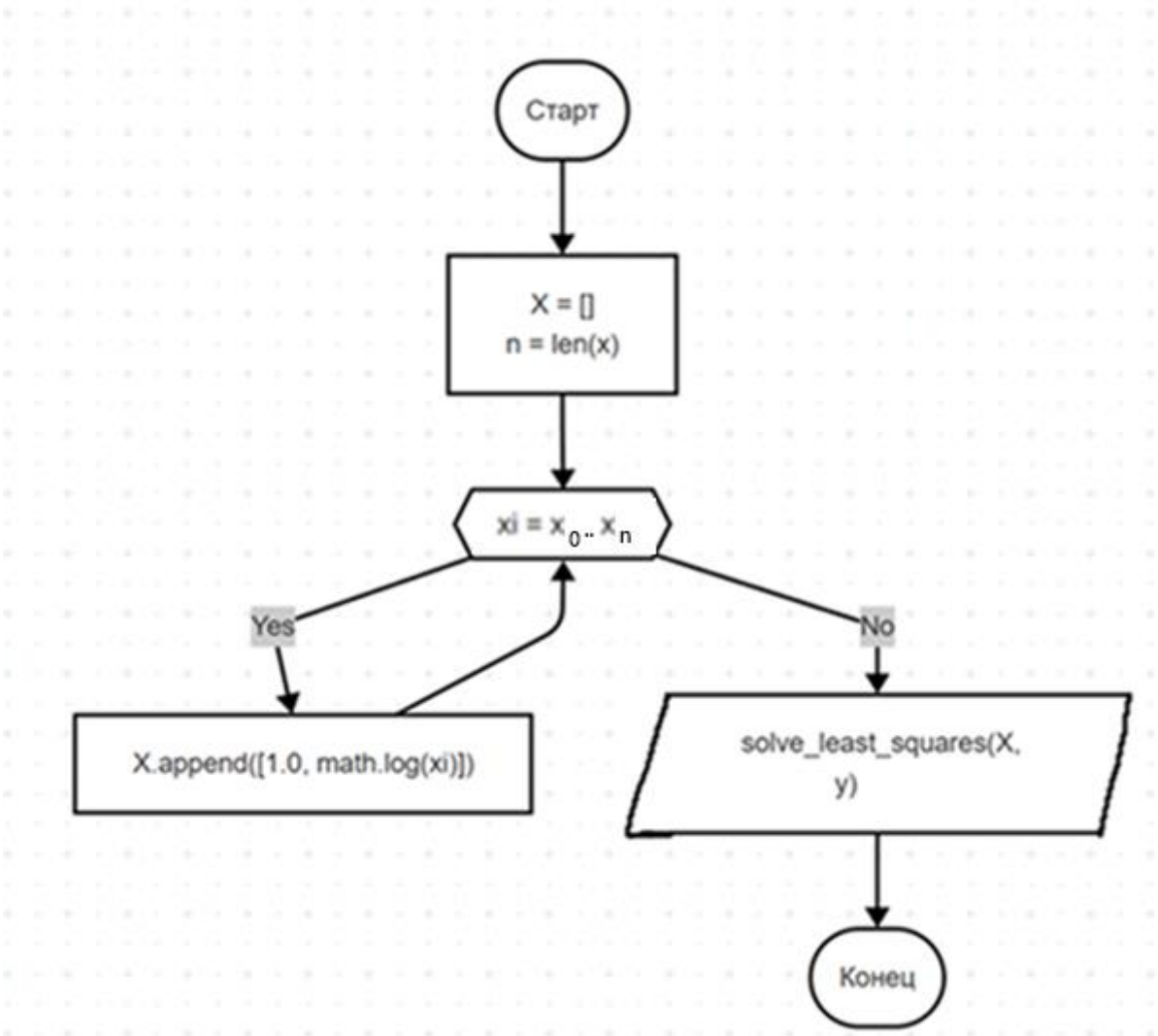
Полиномиальная функция 3-й степени



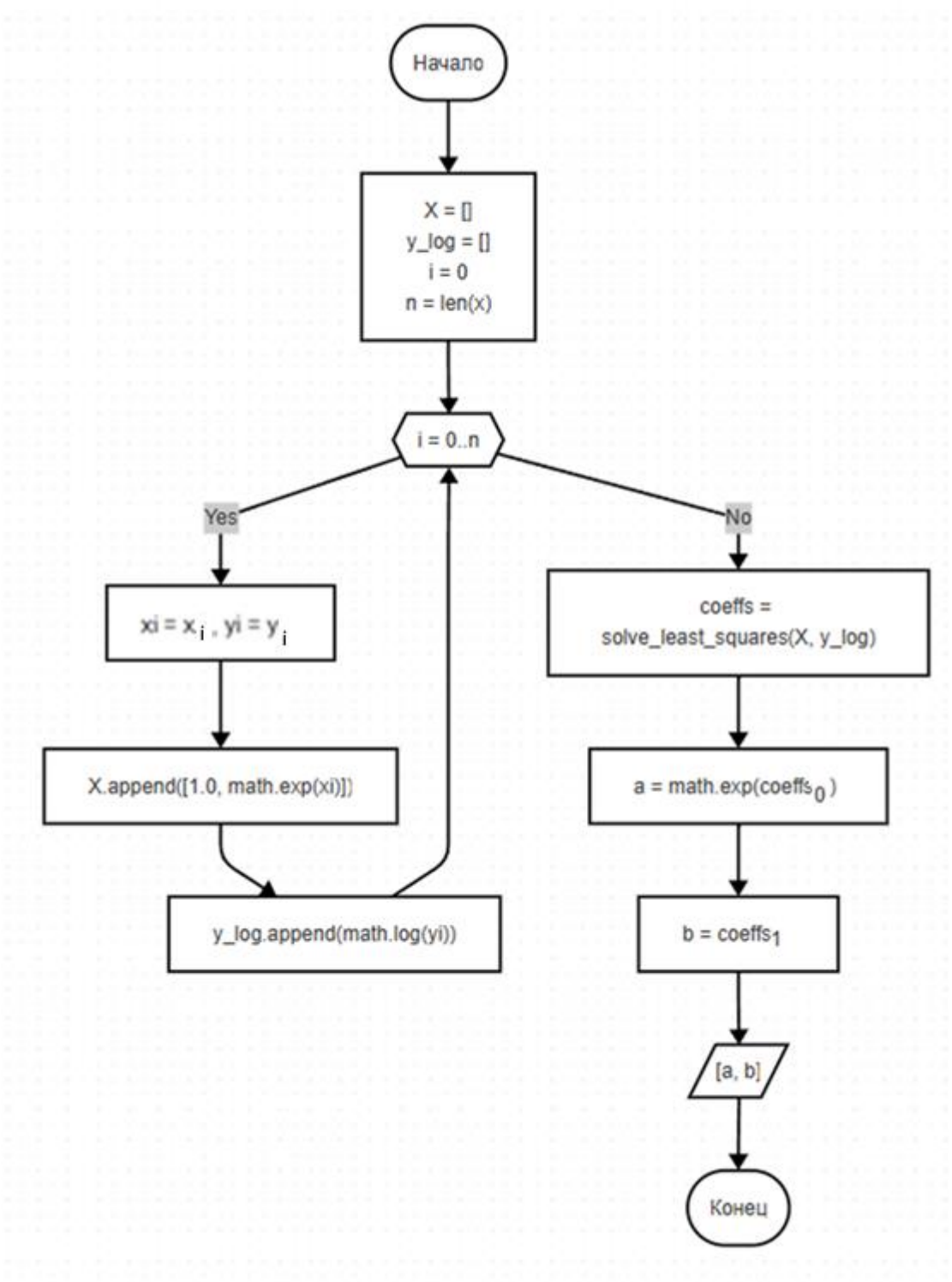
Экспоненциальная функция: y = a * exp(b*x)



Логарифмическая функция



Степенная функция



Листинг программы

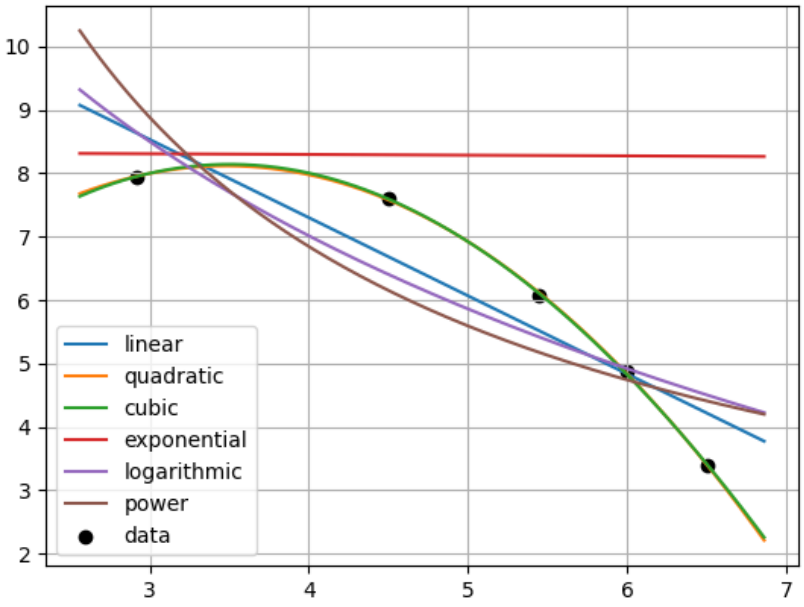
```
import math


def calculate_mse(errors):
    """Расчет среднеквадратичного отклонения"""
    return sum(e * e for e in errors) / len(errors)


def calculate_pearson_correlation(x, y, coeffs=None):
    """Расчет коэффициента корреляции Пирсона"""
```

```
    mean_x = sum(x) / len(x)
    mean_y = sum(y) / len(y)
    numerator = 0.0
    denominator_x = 0.0
    denominator_y = 0.0
    for xi, yi in zip(x, y):
        numerator += (xi - mean_x) * (yi - mean_y)
        denominator_x += (xi - mean_x) ** 2
        denominator_y += (yi - mean_y) ** 2
    return numerator / math.sqrt(denominator_x * denominator_y)


def fit_linear(x, y):
    """Линейная функция: y = a*x + b"""
    X = [[xi, 1.0] for xi in x]
    return solve_least_squares(X, y)


def fit_quadratic(x, y):
    """Полиномиальная функция 2-й степени: y = a*x^2 + b*x + c"""
    X = [[xi ** 2, xi, 1.0] for xi in x]
    return solve_least_squares(X, y)


def fit_cubic(x, y):
    """Полиномиальная функция 3-й степени: y = a*x^3 + b*x^2 + c*x + d"""
    X = [[xi ** 3, xi ** 2, xi, 1.0] for xi in x]
    return solve_least_squares(X, y)


def fit_exponential(x, y):
    """Экспоненциальная функция: y = a * exp(b*x)"""
    X = [[1.0, math.exp(xi)] for xi in x]
    y_log = [math.log(yi) for yi in y]
    coeffs = solve_least_squares(X, y_log)
    a = math.exp(coeffs[0])
    b = coeffs[1]
    return [a, b]


def fit_logarithmic(x, y):
    """Логарифмическая функция: y = a + b*ln(x)"""
    X = [[1.0, math.log(xi)] for xi in x]
    return solve_least_squares(X, y)


def fit_power(x, y):
    """Степенная функция: y = a * x^b"""
    X = [[1.0, math.log(xi)] for xi in x]
    y_log = [math.log(yi) for yi in y]
    coeffs = solve_least_squares(X, y_log)
    a = math.exp(coeffs[0])
    b = coeffs[1]
    return [a, b]


def solve_least_squares(X, y):
    """Решение системы уравнений методом наименьших квадратов"""
    Xt = transpose(X)
    XtX = multiply_matrix(Xt, X)
    XtY = multiply_matrix_vector(Xt, y)
    return solve_linear_system(XtX, XtY)


def transpose(matrix):
    """Транспонирование матрицы"""
```

```
    return [list(row) for row in zip(*matrix)]


def multiply_matrix(a, b):
    """Умножение двух матриц"""
    m, k = len(a), len(a[0])
    n = len(b[0])
    result = [[0.0] * n for _ in range(m)]
    for i in range(m):
        for j in range(n):
            for l in range(k):
                result[i][j] += a[i][l] * b[l][j]
    return result


def multiply_matrix_vector(matrix, vector):
    """Умножение матрицы на вектор"""
    m = len(matrix)
    n = len(matrix[0])
    result = [0.0] * m
    for i in range(m):
        for j in range(n):
            result[i] += matrix[i][j] * vector[j]
    return result


def solve_linear_system(A, b):
    """Решение системы линейных уравнений Ax = b методом Гаусса с выбором
главного элемента"""
    n = len(A)
    M = [row[:] + [b_i] for row, b_i in zip(A, b)]

    # Прямой ход
    for i in range(n):
        max_row = max(range(i, n), key=lambda r: abs(M[r][i]))
        M[i], M[max_row] = M[max_row], M[i]

        pivot = M[i][i]
        for j in range(i, n + 1):
            M[i][j] /= pivot
        for k in range(i + 1, n):
            factor = M[k][i]
            for j in range(i, n + 1):
                M[k][j] -= factor * M[i][j]

    # Обратный ход
    x = [0.0] * n
    for i in range(n - 1, -1, -1):
        x[i] = M[i][n] - sum(M[i][j] * x[j] for j in range(i + 1, n))
    return x
```

Примеры и результаты работы программы

Approximation Results



Linear Model

Coefficients: [-1.2343574745311559, 12.237401110822024]

MSE: 0.461343, R^2: 0.840771

Достоверность аппроксимации: Удовлетворительная аппроксимация (модель в целом адекватно описывает явление)

Pearson r: -0.916936, r^2: 0.840771

x	y	y_pred	error
2.9200	7.9488	8.6331	-0.6843
4.5000	7.6088	6.6828	0.9260
5.4400	6.0687	5.5225	0.5462
6.0000	4.8687	4.8313	0.0374
6.5000	3.3887	4.2141	-0.8254

Quadratic Model

Coefficients: [-0.5178247146614562, 3.6043876500755245, 1.8443283422265697]

MSE: 0.001218, R^2: 0.999580

Достоверность аппроксимации: Высокая точность аппроксимации (модель хорошо описывает явление)

x	y	y_pred	error

Github: https://github.com/MrTheFall/computational_math/tree/main/lab4

Выводы:

В рамках лабораторной работы были исследованы численные методы интегрирования с использованием языка Python. В процессе работы были изучены различные подходы к вычислению определенных интегралов, включая методы прямоугольников (левых, правых и средних), метод трапеций, метод Ньютона-Котеса и метод Симпсона.

Была разработана программа, которая позволяет выбрать одну из предложенных функций, задать пределы интегрирования, точность и начальное значение числа разбиений интервала. После реализации всех рассмотренных методов вычисления интегралов было установлено, что метод Симпсона является наиболее точным и быстрым.

В ходе выполнения работы были проведены расчеты интегралов различными методами, а также выполнено сравнение полученных результатов с точными значениями интегралов.

Кроме того, была решена дополнительная задача, связанная с исследованием сходимости несобственных интегралов второго рода и их вычислением с использованием рассмотренных численных методов в случаях, когда подынтегральная функция имеет бесконечный разрыв в точке a, точке b или на отрезке интегрирования.