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

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа **№4**
«Численное интегрирование»

по дисциплине «Вычислительная математика**»**

Вариант: **13**

Преподаватель:
Наумова Надежда Александровна

Выполнил:
Саранча Павел Александрович
Группа: P3209

Санкт-Петербург, 2025 г.

Цель работы: найти приближенное значение определенного интеграла с требуемой точностью различными численными методами.

Линейная аппроксимация:

y = $\frac{31x}{x^4+13}$

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	0.952	1.849	2.468	2.537	2.138	1.611	1.166	0.842	0.617	0.461

φ(x) = a + bx

Вычисляем суммы: sx = 22, sxx = 61.6, sy = 14.64 sxy = 27.048

$$\begin{cases} n*a+sx*b=sy \\ sx*a+sxx*b=sxy \end{cases} \quad \begin{cases} 11*a+22*b=14.64 \\ 22*a+61.6*b=27.048 \end{cases} \quad \begin{cases} a=1.585 \\ b=-0.127 \end{cases}$$

φ(x) = $1.585 - 0.127 * x$

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	0.952	1.849	2.468	2.537	2.138	1.611	1.166	0.842	0.617	0.461
φ(xi)	1.585	1.534	1.483	1.433	1.382	1.331	1.280	1.229	1.179	1.128	1.077
(φ (xi)-yi)^2	2.512	0.339	0.134	1.072	1.334	0.651	0.109	0.004	0.113	0.261	0.379

$$\sigma = \sqrt{\frac{\sum(\varphi(xi)-yi)^2}{n}} = \mathbf{0.79257}$$

Квадратичная аппроксимация:

y = $\frac{31x}{x^4+13}$
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	0.952	1.849	2.468	2.537	2.138	1.611	1.166	0.842	0.617	0.461

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

Вычисляем суммы:
sx = 22, sxx = 61.6, sxxx = 193.6, sxxxx = 648.52, sy = 14.64, sxy = 27.0476, sxxy = 62.35152

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

$$\begin{cases} 11 * a + 22 * b + 61.6 * c = 14.64 \\ 22 * a + 61.6 * b + 193.6 * c = 27.0476 \\ 61.6 * a + 193.6 * b + 648.52 * c = 62.35152 \end{cases}$$

По методу Крамера:
$\Delta = 4252.385$
$\Delta_1 = 1767.773, \Delta_2 = 7746.592, \Delta_3 = -2071.613$

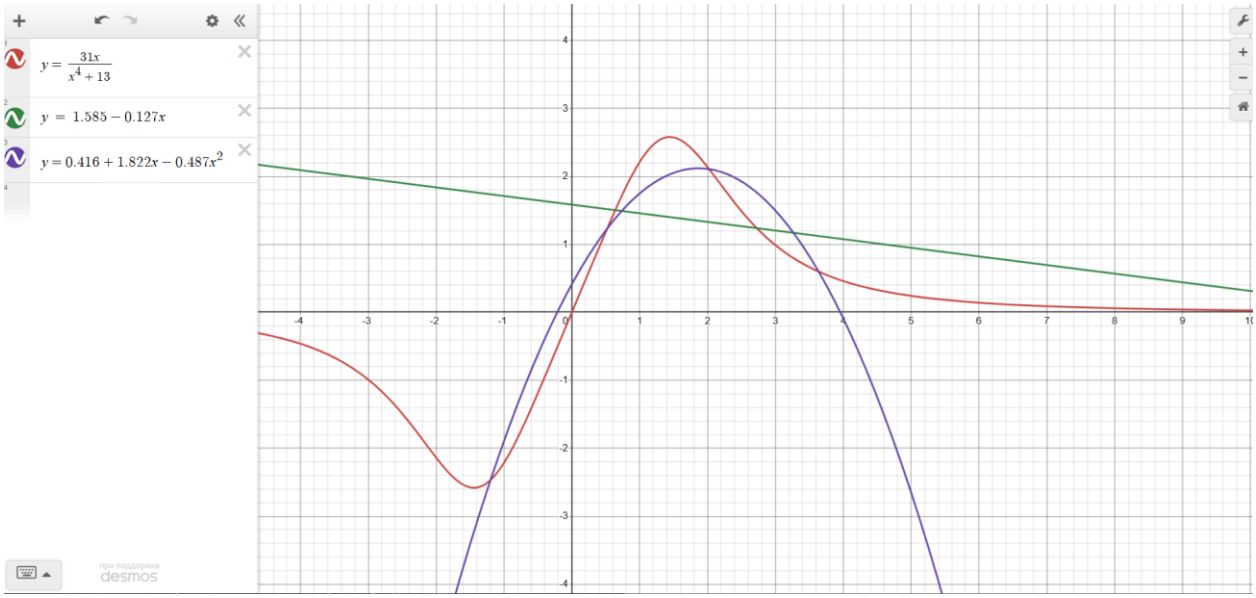
$$\begin{cases} a = \frac{\Delta_1}{\Delta} \approx 0.416 \\ b = \frac{\Delta_2}{\Delta} \approx 1.822 \\ c = \frac{\Delta_3}{\Delta} \approx -0.487 \end{cases}$$

$\varphi(x) = 0.416 + 1.822x - 0.487x^2$

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	0.952	1.849	2.468	2.537	2.138	1.611	1.166	0.842	0.617	0.461
φ(xi)	0.416	1.066	1.561	1.900	2.083	2.110	1.982	1.697	1.257	0.660	-0.092
(φ (xi)-yi)^2	0.173	0.013	0.083	0.322	0.206	0.001	0.137	0.282	0.172	0.002	0.306

$$\sigma = \sqrt{\frac{\sum(\varphi(xi) - yi)^2}{n}} = \mathbf{0.39276}$$

0.39276 < 0.79257, у квадратичной аппроксимации среднеквадратичное отклонение меньше, поэтому это приближение лучше.



2. Программная реализация задачи

https://github.com/PaulLocust/comp_math_lab4

Листинг методов:

```
def linear_approximation(x, y):
    n = len(x)
    sx, sy = sum(x), sum(y)
    sxx = sum(i ** 2 for i in x)
    sxy = sum(x[i] * y[i] for i in range(n))
    denominator = n * sxx - sx ** 2
    if denominator == 0:
        return None
    b = (n * sxy - sx * sy) / denominator
    a = (sy - b * sx) / n
    return (a, b), lambda t: a + b * t
```

```
def quadratic_approximation(x, y):
    n = len(x)
    sx = sum(x)
    sy = sum(y)
    sxx = sum(i ** 2 for i in x)
    sxxx = sum(i ** 3 for i in x)
    sxxxx = sum(i ** 4 for i in x)
    sxy = sum(x[i] * y[i] for i in range(n))
    sxxy = sum((x[i] ** 2) * y[i] for i in range(n))

    # Создаем матрицу коэффициентов и вектор правой части
    A = [
        [n, sx, sxx],
        [sx, sxx, sxxx],
        [sxx, sxxx, sxxxx]
    ]
    B = [sy, sxy, sxxy]

    try:
        a, b, c = solve_sle(A, B, 3)
        return (a, b, c), lambda t: a + b * t + c * t ** 2
    except:
        return None
```

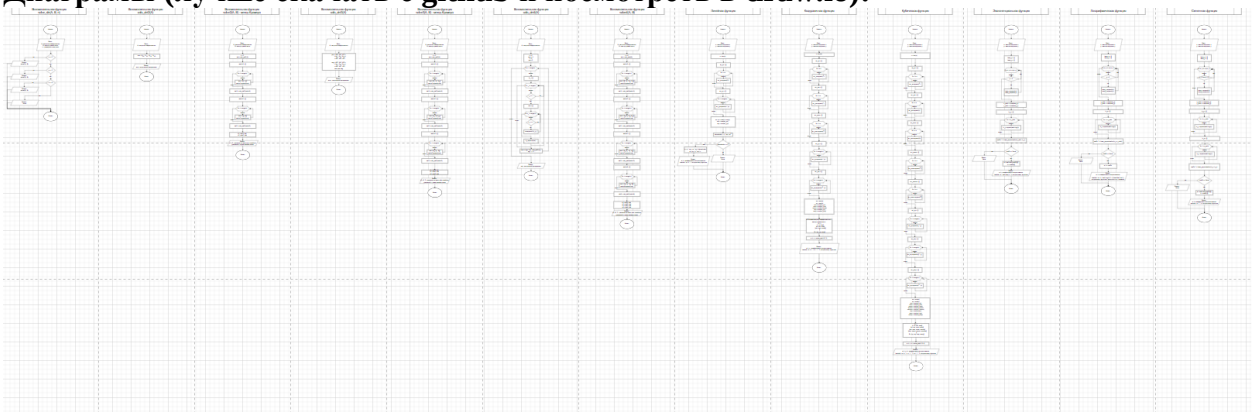
```
def cubic_approximation(x, y):
    n = len(x)
    sx = sum(x)
    sy = sum(y)
    sxx = sum(i ** 2 for i in x)
    sxxx = sum(i ** 3 for i in x)
    sxxxx = sum(i ** 4 for i in x)
    sxxxxx = sum(i ** 5 for i in x)
    sxxxxxx = sum(i ** 6 for i in x)
    sxy = sum(x[i] * y[i] for i in range(n))
    sxxy = sum((x[i] ** 2) * y[i] for i in range(n))
    sxxxy = sum((x[i] ** 3) * y[i] for i in range(n))

    A = [
        [n, sx, sxx, sxxx],
        [sx, sxx, sxxx, sxxxx],
        [sxx, sxxx, sxxxx, sxxxxx],
        [sxxx, sxxxx, sxxxxx, sxxxxxx]
    ]
    B = [sy, sxy, sxxy, sxxxy]

    try:
        a, b, c, d = solve_sle(A, B, 4)
        return (a, b, c, d), lambda t: a + b * t + c * t ** 2 + d * t ** 3
    except:
        return None
```

```python
def exponential_approximation(x, y):
    try:
        temp_x = []
        temp_y = []

        for xi, yi in zip(x, y):  # параллельный обход x и y
            if yi > 0:
                temp_x.append(xi)
                temp_y.append(yi)

        x_valid = tuple(temp_x)
        y_valid = tuple(temp_y)
        ln_y = [math.log(i) for i in y_valid]
        coeffs, f = linear_approximation(x_valid, ln_y)
        if coeffs is None:
            return None
        a = math.exp(coeffs[0])
        b = coeffs[1]
        return (a, b), lambda t: a * math.exp(b * t)
    except:
        return None
```

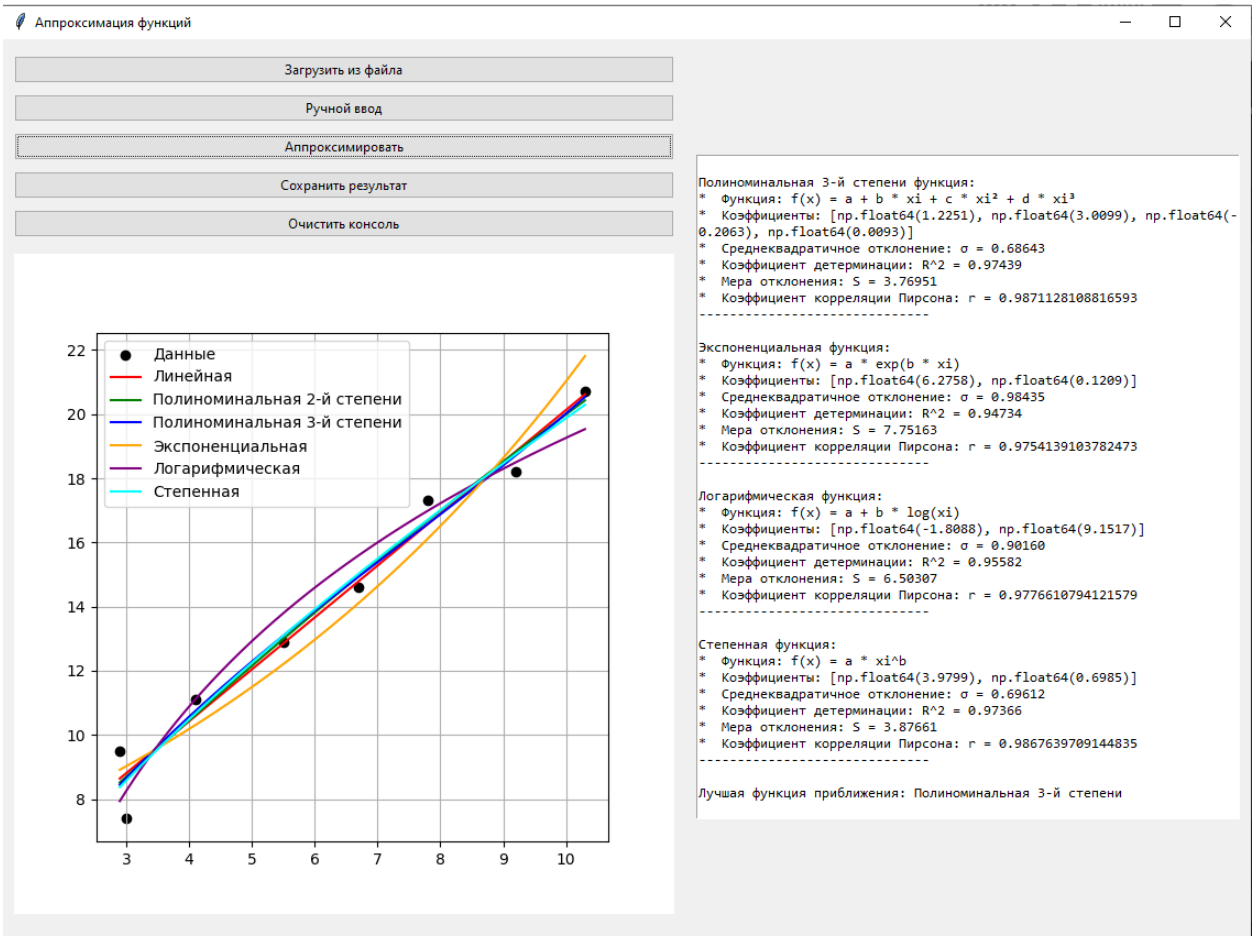
```python
def logarithmic_approximation(x, y):
    try:
        # Исключаем значения x <= 0, так как логарифм не определён для таких значений
        temp_x = []
        temp_y = []

        for xi, yi in zip(x, y):  # параллельный обход x и y
            if yi > 0:
                temp_x.append(xi)
                temp_y.append(yi)

        x_valid = tuple(temp_x)
        y_valid = tuple(temp_y)

        # Применяем логарифм только к положительным значениям x
        ln_x = [math.log(i) for i in x_valid]

        coeffs, f = linear_approximation(ln_x, y_valid)
        if coeffs is None:
            return None

        a, b = coeffs
        return (a, b), lambda t: a + b * math.log(t) if t > 0 else float('nan')  # Защита от отрицательных значений t
    except Exception as e:
        print(f"Ошибка при аппроксимации логарифмической функцией: {e}")
        return None
```

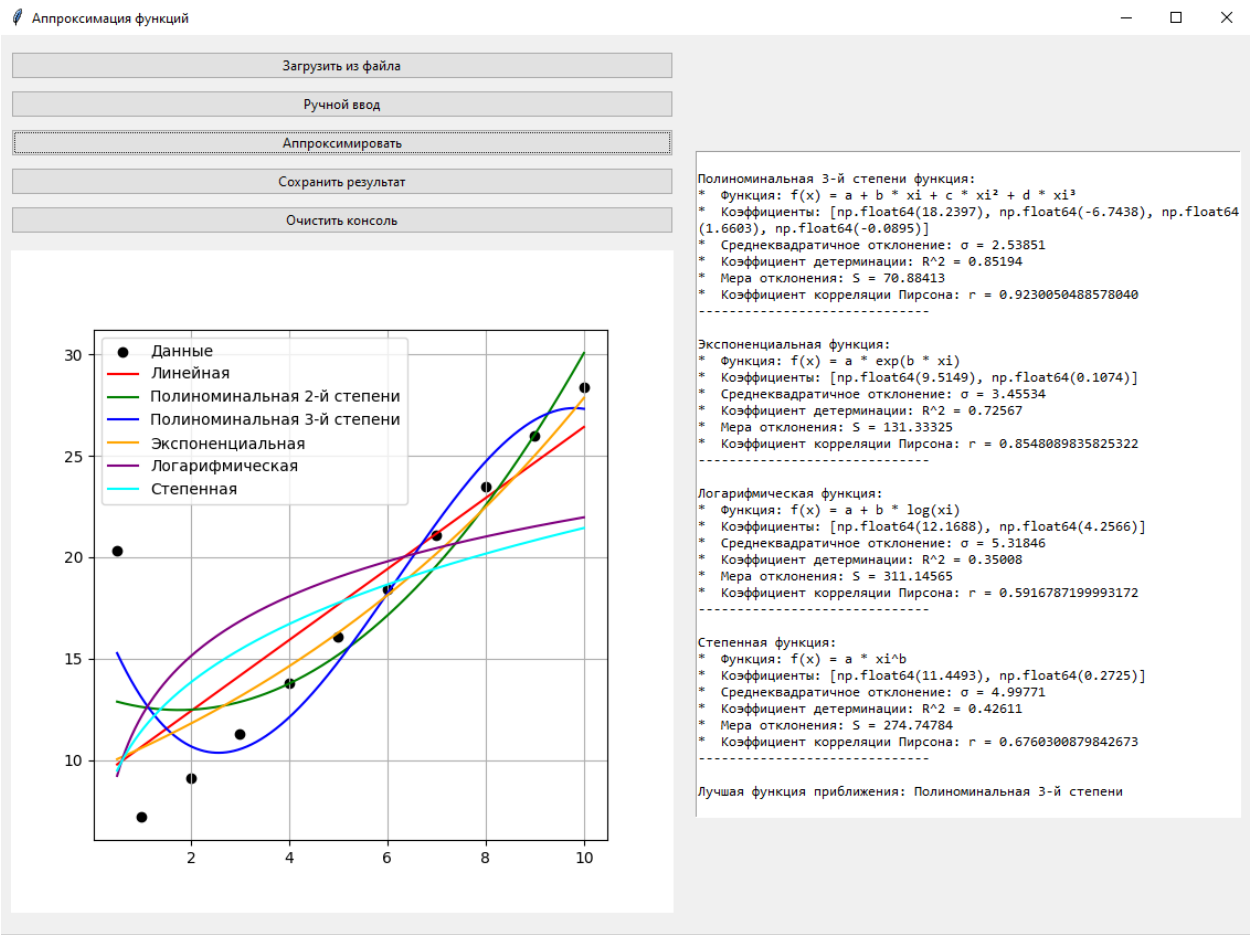
```python
def power_approximation(x, y):
    try:
        temp_x = []
        temp_y = []

        for xi, yi in zip(x, y):  # параллельный обход x и y
            if xi > 0 and yi > 0:
                temp_x.append(xi)
                temp_y.append(yi)

        x_valid = tuple(temp_x)
        y_valid = tuple(temp_y)

        ln_x = [math.log(i) for i in x_valid]
        ln_y = [math.log(i) for i in y_valid]
        coeffs, f = linear_approximation(ln_x, ln_y)
        if coeffs is None:
            return None
        a = math.exp(coeffs[0])
        b = coeffs[1]
        return (a, b), lambda t: a * t ** b
    except:
        return None
```

Диаграмы (лучше скачать с github и посмотреть в draw.io):



Результаты выполнения программы при различных исходных данных:





Вывод

В ходе данной работы была выполнена аппроксимация функций с использованием линейного, квадратичного, кубического, экспоненциального и логарифмического приближений. Также на основе этих методов был реализован Python скрипт, который 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.