

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:

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

$$n = 11$$

$$x \in [0; 4]$$

$$h = 0.4$$

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	0.952	1.849	2.468	2.537	2.138	1.611	1.166	0.842	0.617	0.461

$$\varphi(x) = a + bx$$

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

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

$$\varphi(x) = 1.585 - 0.127 * x$$

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(x _i)	1.585	1.534	1.483	1.433	1.382	1.331	1.280	1.229	1.179	1.128	1.077
(φ(x _i) - y _i) ²	2.512	0.339	0.134	1.072	1.334	0.651	0.109	0.004	0.113	0.261	0.379

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

Квадратичная аппроксимация:

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

$$n = 11$$

$$x \in [0; 4]$$

$$h = 0.4$$

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	0.952	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:

$$sx = 22, sxx = 61.6, sxxx = 193.6, sxxxx = 648.52, sy = 14.64, sxy = 27.0476, \\ sxxxy = 62.35152$$

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

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

По методу Крамера:

$$\Delta = 4252.385$$

$$\Delta_1 = 1767.773, \Delta_2 = 7746.592, \Delta_3 = -2071.613$$

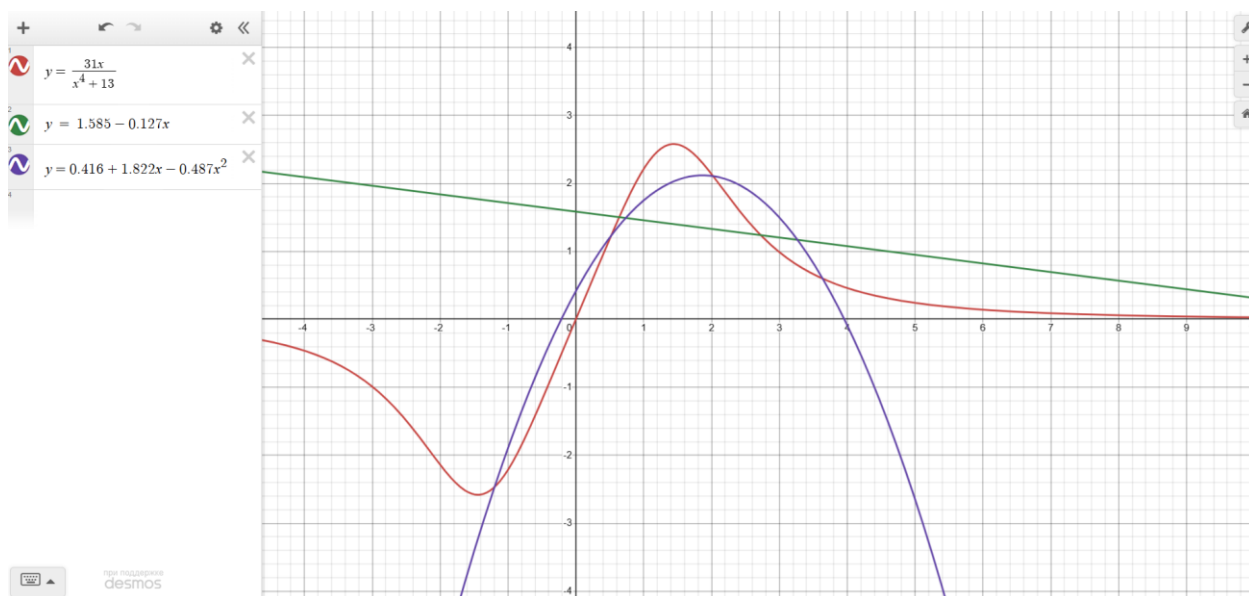
$$\begin{cases} a = \frac{\Delta_1}{\Delta} \approx 0.416 \\ b = \frac{\Delta_2}{\Delta} \approx 1.822 \\ c = \frac{\Delta_3}{\Delta} \approx -0.487 \end{cases}$$

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

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(x _i)	0.416	1.066	1.561	1.900	2.083	2.110	1.982	1.697	1.257	0.660	-0.092
(φ(x _i) - y _i) ²	0.173	0.013	0.083	0.322	0.206	0.001	0.137	0.282	0.172	0.002	0.306

$$\sigma = \sqrt{\frac{\sum(\varphi(x_i) - y_i)^2}{n}} = 0.39276$$

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



2. Программная реализация задачи

https://github.com/PaulLocust/comp_math_lab4

Листинг методов:

```
0 | def linear_approximation(x, y):
1 |     n = len(x)
2 |     sx, sy = sum(x), sum(y)
3 |     sxx = sum(i ** 2 for i in x)
4 |     sxy = sum(x[i] * y[i] for i in range(n))
5 |     denominator = n * sxx - sx ** 2
6 |     if denominator == 0:
7 |         return None
8 |     b = (n * sxy - sx * sy) / denominator
9 |     a = (sy - b * sx) / n
10 |    return (a, b), lambda t: a + b * t
```

```
1 | def quadratic_approximation(x, y):
2 |     n = len(x)
3 |     sx = sum(x)
4 |     sy = sum(y)
5 |     sxx = sum(i ** 2 for i in x)
6 |     sxxx = sum(i ** 3 for i in x)
7 |     sxxxx = sum(i ** 4 for i in x)
8 |     sxy = sum(x[i] * y[i] for i in range(n))
9 |     sxxxy = sum((x[i] ** 2) * y[i] for i in range(n))
10 |
11 |     # Создаем матрицу коэффициентов и вектор правой части
12 |     A = [
13 |         [n, sx, sxx],
14 |         [sx, sxx, sxxx],
15 |         [sxx, sxxx, sxxxx]
16 |     ]
17 |     B = [sy, sxy, sxxxy]
18 |
19 |     try:
20 |         a, b, c = solve_sle(A, B, 3)
21 |         return (a, b, c), lambda t: a + b * t + c * t ** 2
22 |     except:
23 |         return None
```

```
1 | def cubic_approximation(x, y):
2 |     n = len(x)
3 |     sx = sum(x)
4 |     sy = sum(y)
5 |     sxx = sum(i ** 2 for i in x)
6 |     sxxx = sum(i ** 3 for i in x)
7 |     sxxxx = sum(i ** 4 for i in x)
8 |     sxxxxx = sum(i ** 5 for i in x)
9 |     sxxxxxx = sum(i ** 6 for i in x)
10 |    sxy = sum(x[i] * y[i] for i in range(n))
11 |    sxxxy = sum((x[i] ** 2) * y[i] for i in range(n))
12 |    sxxxxy = sum((x[i] ** 3) * y[i] for i in range(n))
13 |
14 |    A = [
15 |        [n, sx, sxx, sxxx],
16 |        [sx, sxx, sxxx, sxxxx],
17 |        [sxx, sxxx, sxxxx, sxxxxx],
18 |        [sxxx, sxxxx, sxxxxx, sxxxxxx]
19 |    ]
20 |    B = [sy, sxy, sxxxy, sxxxxy]
21 |
22 |    try:
23 |        a, b, c, d = solve_sle(A, B, 4)
24 |        return (a, b, c, d), lambda t: a + b * t + c * t ** 2 + d * t ** 3
25 |    except:
26 |        return None
```

```

1 | def exponential_approximation(x, y):
2 |     try:
3 |         temp_x = []
4 |         temp_y = []
5 |
6 |         for xi, yi in zip(x, y): # параллельный обход x и y
7 |             if yi > 0:
8 |                 temp_x.append(xi)
9 |                 temp_y.append(yi)
10 |
11 |         x_valid = tuple(temp_x)
12 |         y_valid = tuple(temp_y)
13 |         ln_y = [math.log(i) for i in y_valid]
14 |         coeffs, f = linear_approximation(x_valid, ln_y)
15 |         if coeffs is None:
16 |             return None
17 |         a = math.exp(coeffs[0])
18 |         b = coeffs[1]
19 |         return (a, b), lambda t: a * math.exp(b * t)
20 |     except:
21 |         return None

```

```

1 | def logarithmic_approximation(x, y):
2 |     try:
3 |         # Исключаем значения x <= 0, так как логарифм не определён для таких значений
4 |         temp_x = []
5 |         temp_y = []
6 |
7 |         for xi, yi in zip(x, y): # параллельный обход x и y
8 |             if yi > 0:
9 |                 temp_x.append(xi)
10 |                 temp_y.append(yi)
11 |
12 |         x_valid = tuple(temp_x)
13 |         y_valid = tuple(temp_y)
14 |
15 |         # Применяем логарифм только к положительным значениям x
16 |         ln_x = [math.log(i) for i in x_valid]
17 |
18 |         coeffs, f = linear_approximation(ln_x, y_valid)
19 |         if coeffs is None:
20 |             return None
21 |
22 |         a, b = coeffs
23 |         return (a, b), lambda t: a + b * math.log(t) if t > 0 else float('nan') # Защита от отрицательных значений t
24 |     except Exception as e:
25 |         print(f"Ошибка при аппроксимации логарифмической функцией: {e}")
26 |         return None

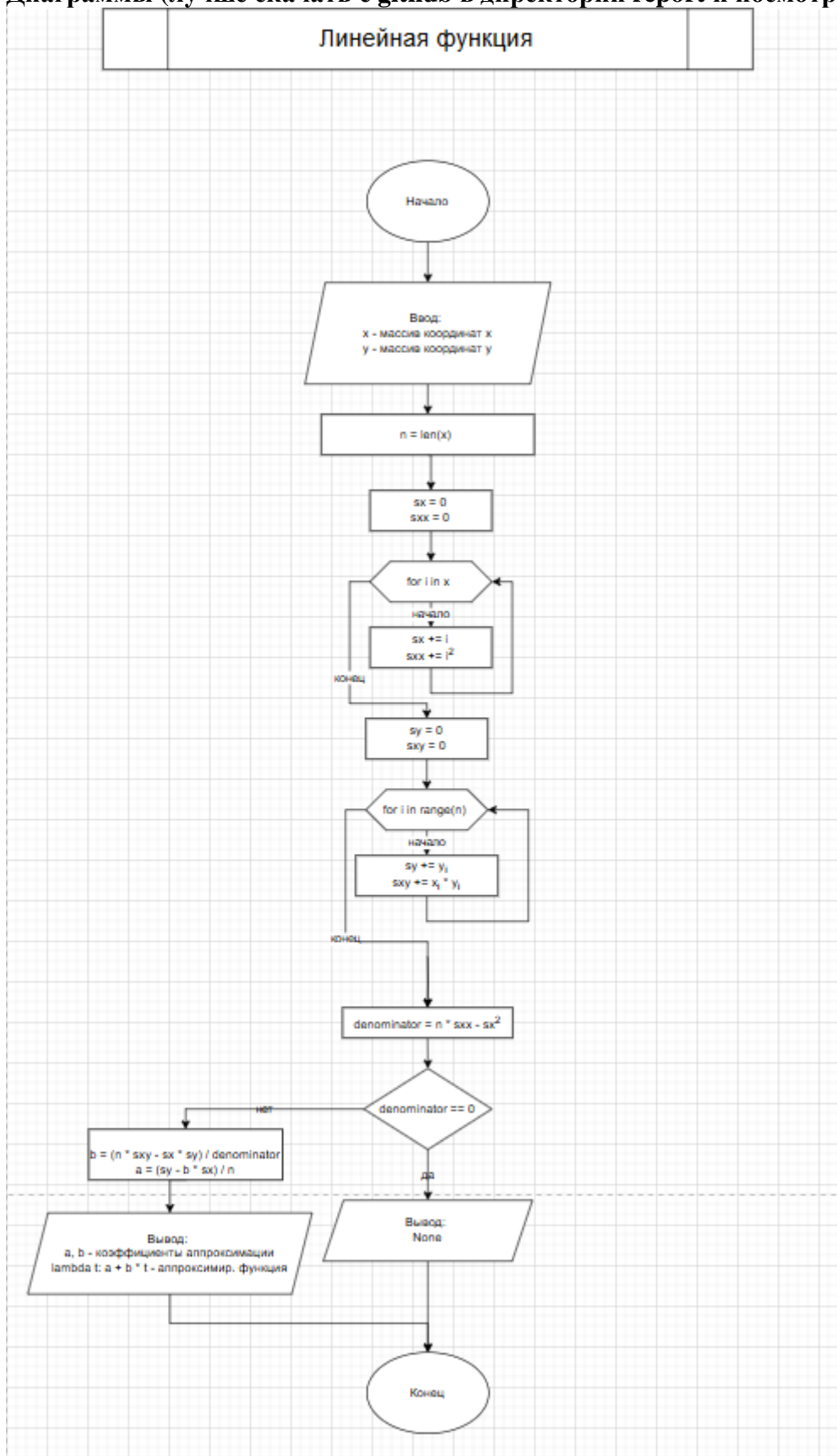
```

```

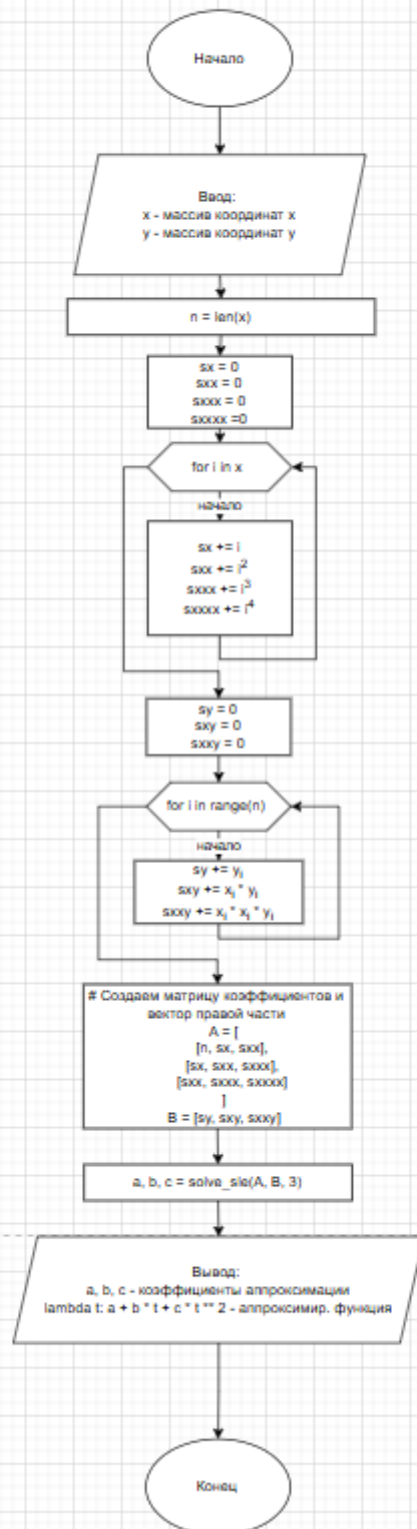
1 | def power_approximation(x, y):
2 |     try:
3 |         temp_x = []
4 |         temp_y = []
5 |
6 |         for xi, yi in zip(x, y): # параллельный обход x и y
7 |             if xi > 0 and yi > 0:
8 |                 temp_x.append(xi)
9 |                 temp_y.append(yi)
10 |
11 |         x_valid = tuple(temp_x)
12 |         y_valid = tuple(temp_y)
13 |
14 |         ln_x = [math.log(i) for i in x_valid]
15 |         ln_y = [math.log(i) for i in y_valid]
16 |         coeffs, f = linear_approximation(ln_x, ln_y)
17 |         if coeffs is None:
18 |             return None
19 |         a = math.exp(coeffs[0])
20 |         b = coeffs[1]
21 |         return (a, b), lambda t: a * t ** b
22 |     except:
23 |         return None

```

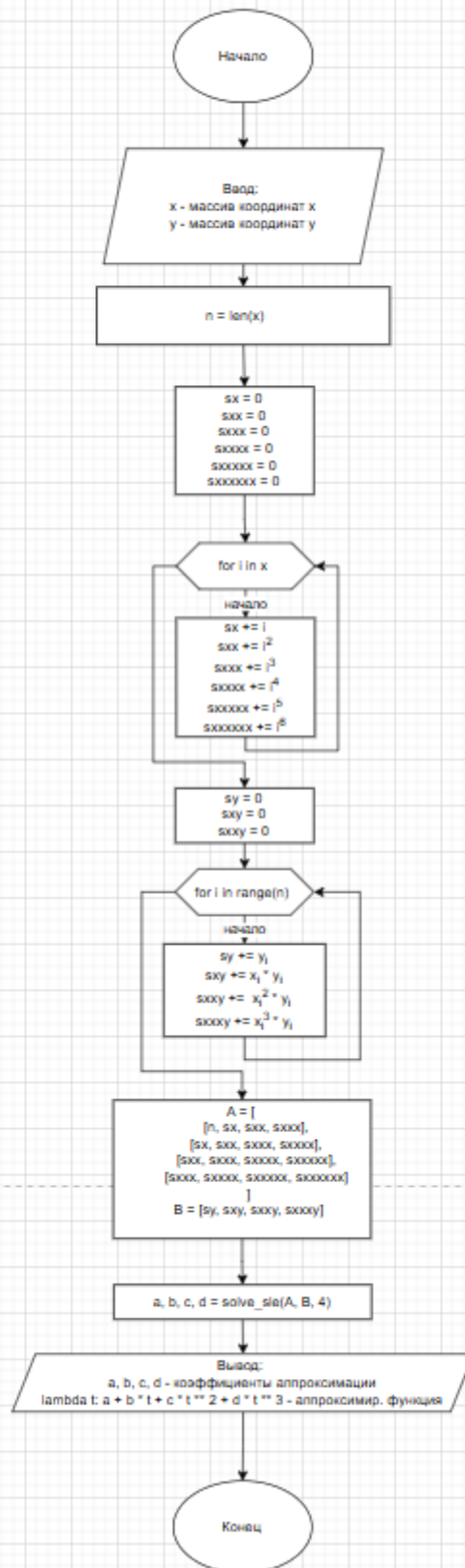
Диаграммы (лучше скачать с [github](#) в директории report и посмотреть в [draw.io](#)):



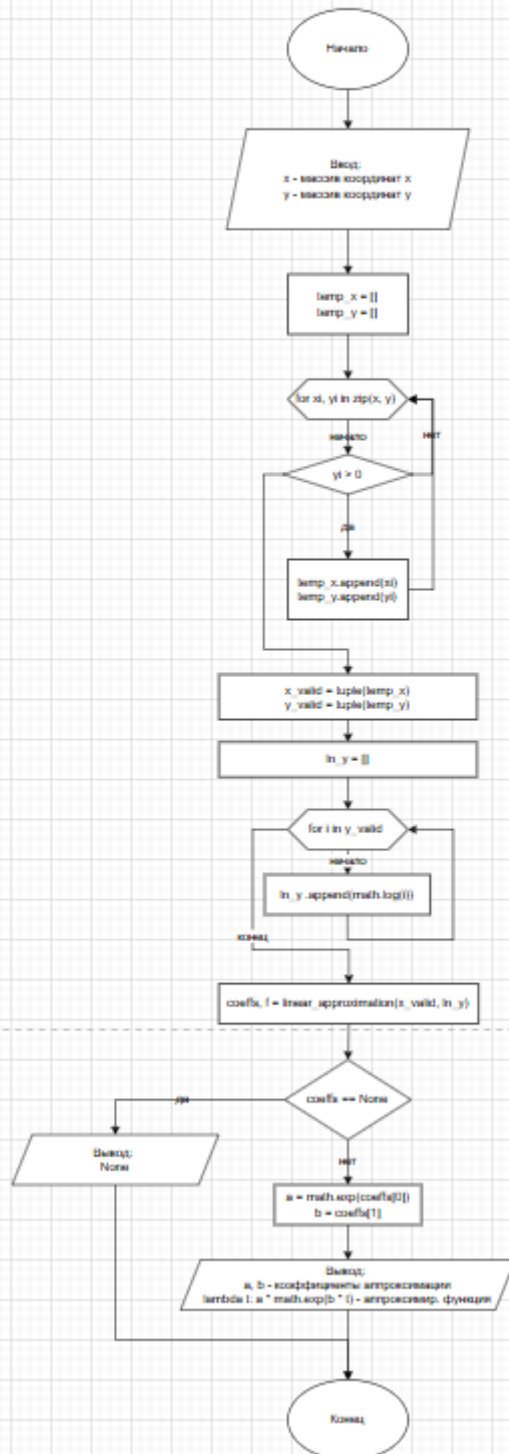
Квадратичная функция



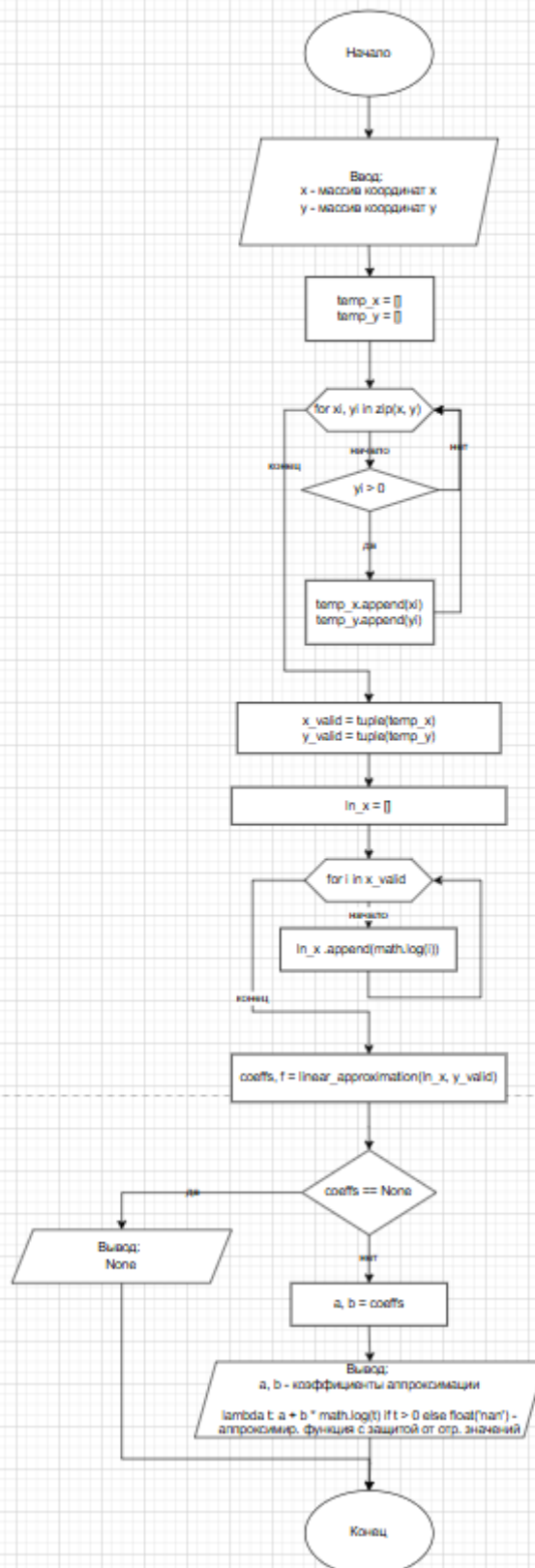
Кубическая функция

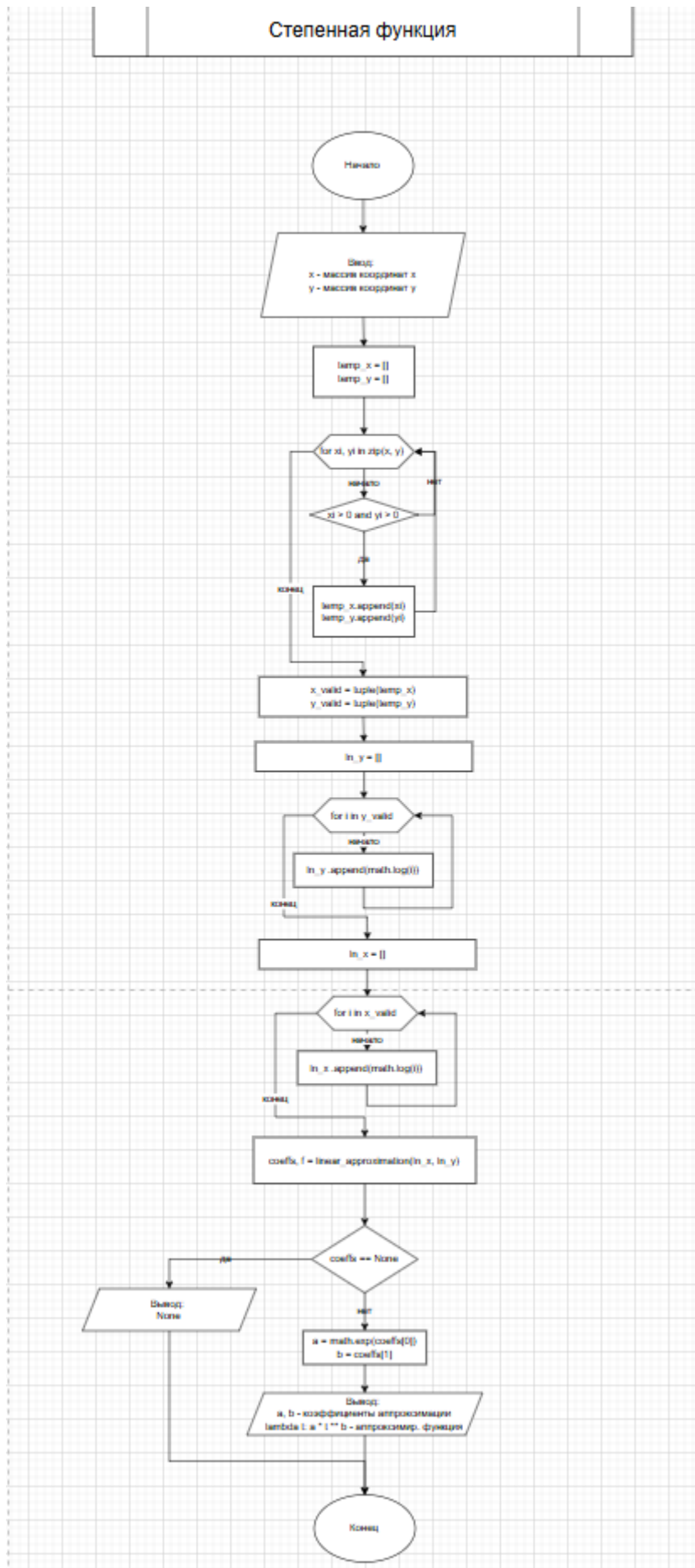


Экспоненциальная функция



Логарифмическая функция





Результаты выполнения программы при различных исходных данных:

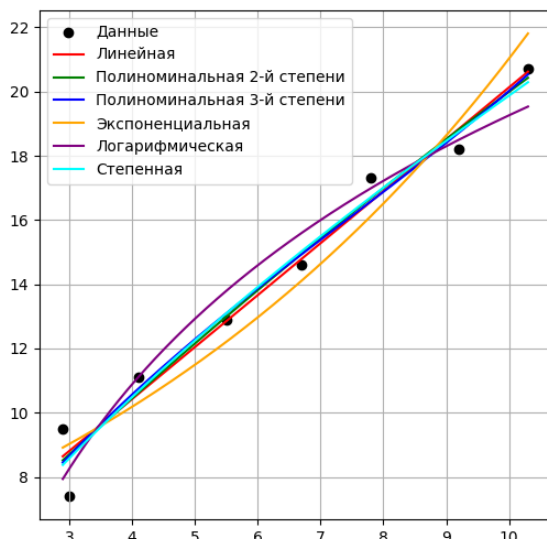
Загрузить из файла

Ручной ввод

Аппроксимировать

Сохранить результат

Очистить консоль



Полиномиальная 3-й степени функция:

* Функция: $f(x) = a + b \cdot x + c \cdot x^2 + d \cdot x^3$
 * Коэффициенты: [np.float64(1.2251), np.float64(3.0099), np.float64(-0.2063), np.float64(0.0093)]
 * Среднеквадратичное отклонение: $\sigma = 0.68643$
 * Коэффициент детерминации: $R^2 = 0.97439$
 * Мера отклонения: $S = 3.76951$
 * Коэффициент корреляции Пирсона: $r = 0.9871128108816593$

Экспоненциальная функция:

* Функция: $f(x) = a \cdot \exp(b \cdot x)$
 * Коэффициенты: [np.float64(6.2758), np.float64(0.1209)]
 * Среднеквадратичное отклонение: $\sigma = 0.98435$
 * Коэффициент детерминации: $R^2 = 0.94734$
 * Мера отклонения: $S = 7.75163$
 * Коэффициент корреляции Пирсона: $r = 0.9754139103782473$

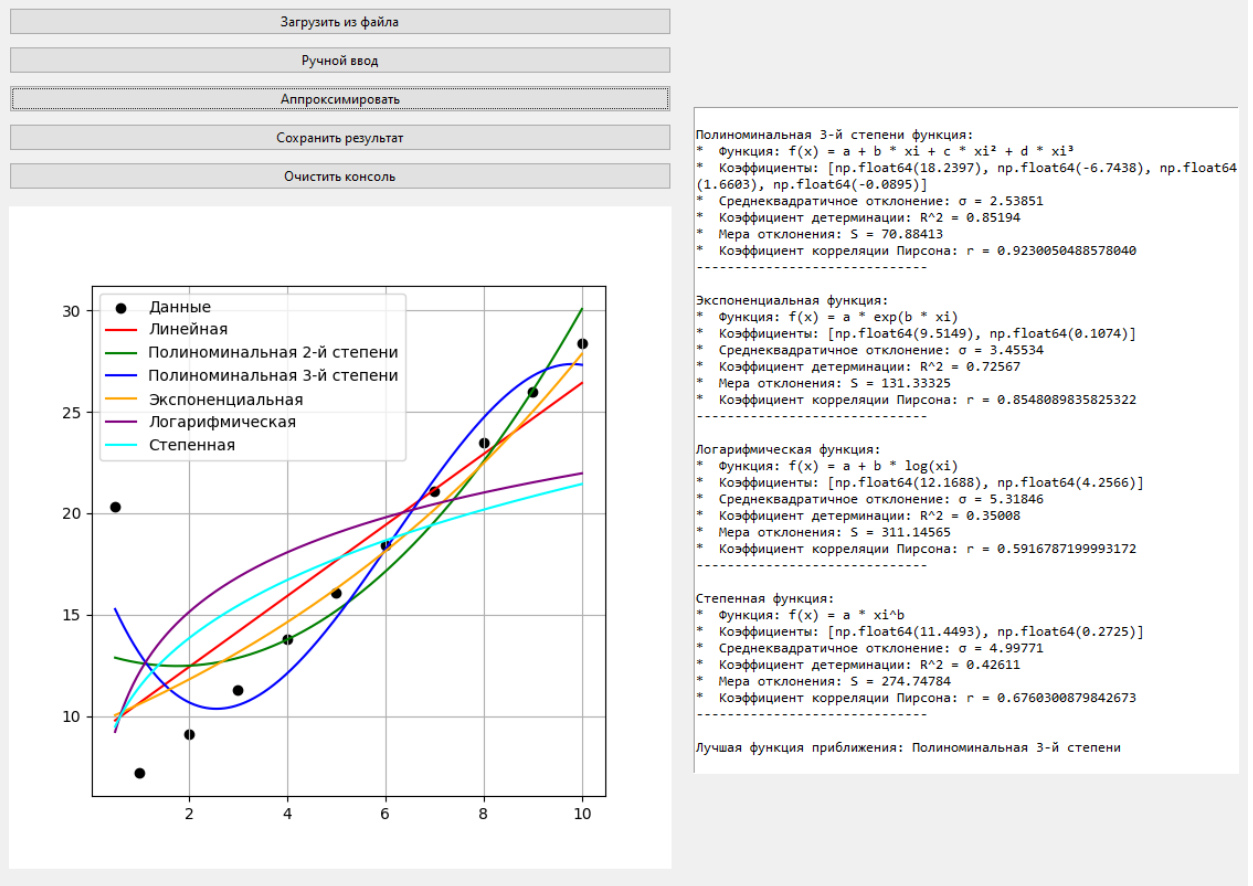
Логарифмическая функция:

* Функция: $f(x) = a + b \cdot \log(x)$
 * Коэффициенты: [np.float64(-1.8088), np.float64(9.1517)]
 * Среднеквадратичное отклонение: $\sigma = 0.90160$
 * Коэффициент детерминации: $R^2 = 0.95582$
 * Мера отклонения: $S = 6.50307$
 * Коэффициент корреляции Пирсона: $r = 0.9776610794121579$

Степенная функция:

* Функция: $f(x) = a \cdot x^b$
 * Коэффициенты: [np.float64(3.9799), np.float64(0.6985)]
 * Среднеквадратичное отклонение: $\sigma = 0.69612$
 * Коэффициент детерминации: $R^2 = 0.97366$
 * Мера отклонения: $S = 3.87661$
 * Коэффициент корреляции Пирсона: $r = 0.9867639709144835$

Лучшая функция приближения: Полиномиальная 3-й степени



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.