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:** Р3209

Санкт-Петербург, 2025 г.

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

Линейная аппроксимация:

y =

n = 11

x [0; 4]

h = 0.4

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 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

φ(x) =

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 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 |

σ = =**0.79257**

Квадратичная аппроксимация:

y =

n = 11

x [0; 4]

h = 0.4

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 0.952 | 1.849 | 2.468 | 2.537 | 2.138 | 1.611 | 1.166 | 0.842 | 0.617 | 0.461 |

φ(x) = a + bx + cx2

Вычисляем суммы:

sx = 22, sxx = 61.6, sxxx = 193.6, sxxxx = 648.52, sy = 14.64, sxy = 27.0476,

sxxy = 62.35152

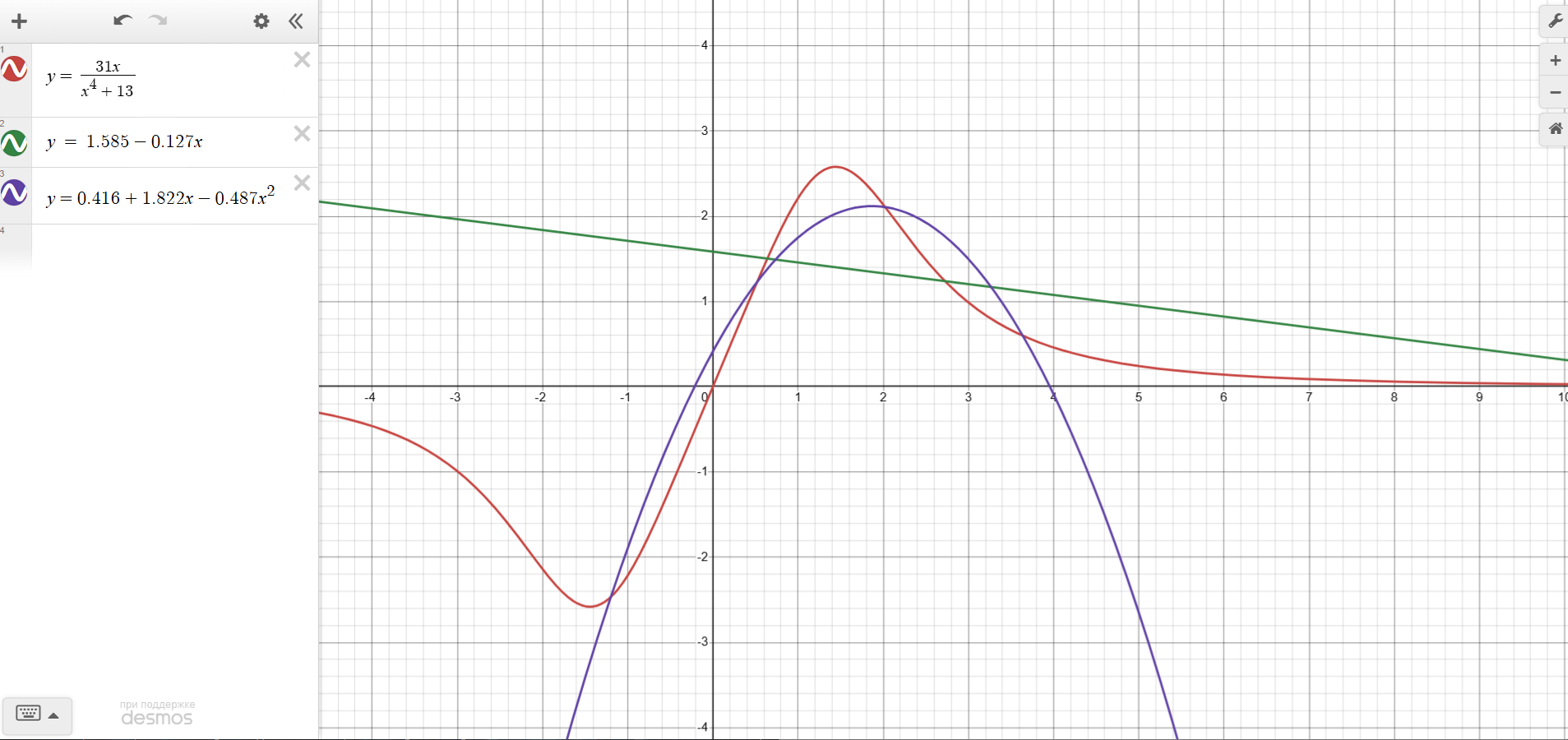
По методу Крамера:

φ(x)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 | 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 |

σ = = **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:**

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 |     sxxy = 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, sxxy]

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|     sxxy = sum((x[i] \*\* 2) \* y[i] **for** i **in** range(n))

12|     sxxxy = 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, sxxy, sxxxy]

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**

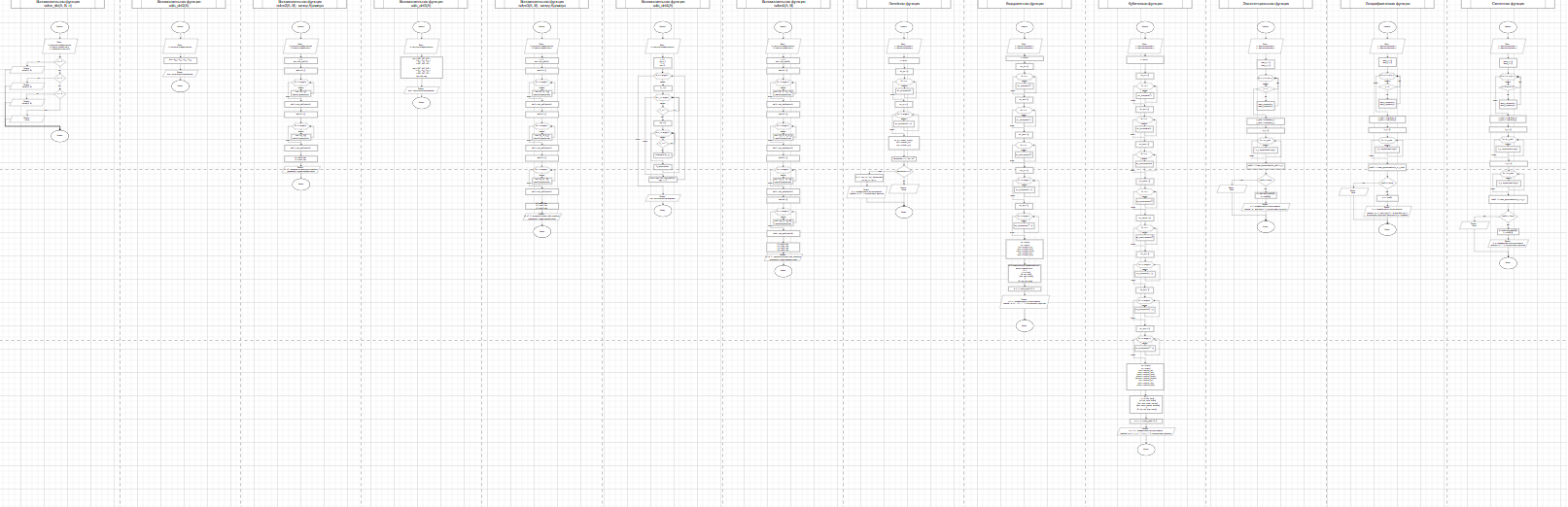
19|         a = math.exp(coeffs[0])

20|         b = coeffs[1]

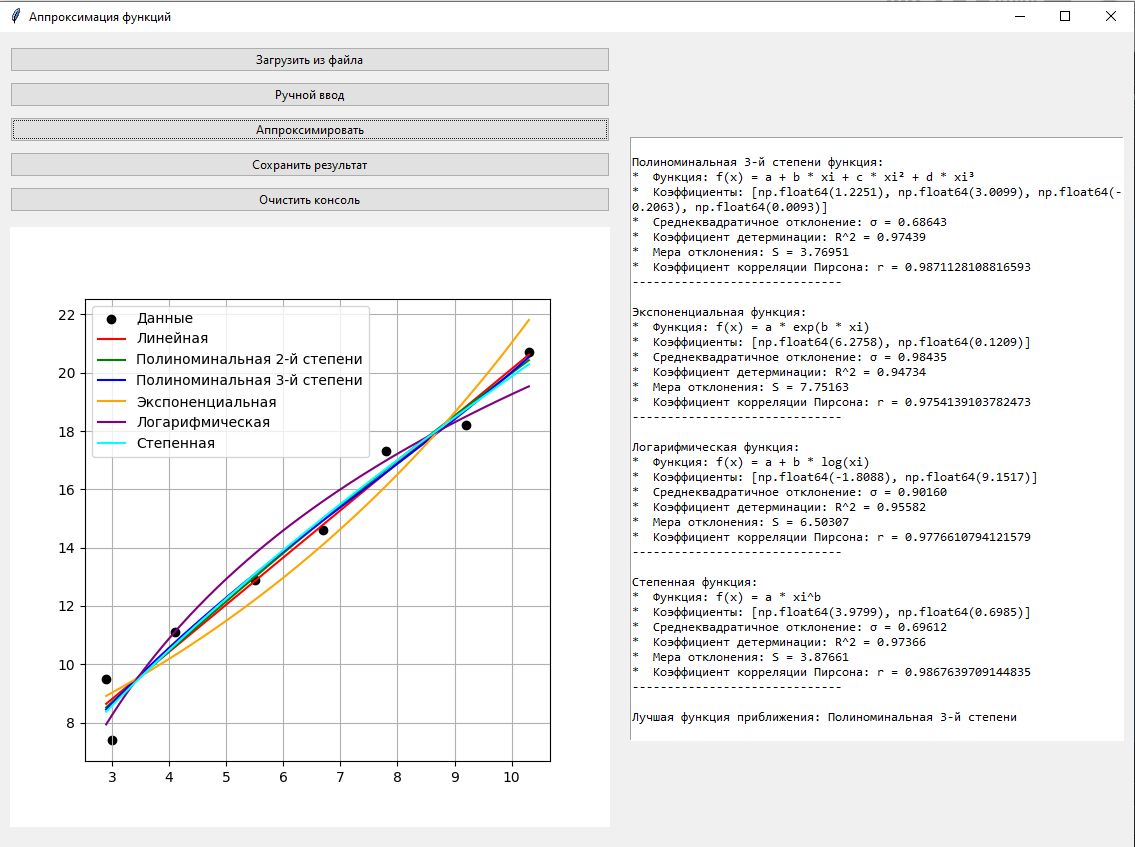
21|         **return** (a, b), **lambda** t: a \* t \*\* b

22|     **except**:

23|         **return** **None**

**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.