Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Факультет программной инженерии и компьютерной техники

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ 4

ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

ВАРИАНТ 12

­

Студент: Пышкин Никита Сергеевич, P3213

Преподаватель:

Санкт Петербург 2025

Содержание

[**Цель лабораторной работы** 3](#_Toc196865546)

[**Порядок выполнения лабораторной работы** 3](#_Toc196865547)

[**Рабочие формулы** 3](#_Toc196865548)

[**Вычислительная часть задания** 4](#_Toc196865549)

[**Программная часть задания** 6](#_Toc196865550)

[**Заключение** 10](#_Toc196865551)

# **Цель лабораторной работы**

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

# **Порядок выполнения лабораторной работы**

**Вычислительная реализация задачи:**

1. Сформировать таблицу табулирования заданной функции на указанном интервале (см. табл. 1)

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.

**Программная реализация задачи:**

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

4. Для линейной зависимости вычислить коэффициент корреляции Пирсона.

5. Вычислить коэффициент детерминации, программа должна выводить соответствующее сообщение в зависимости от полученного значения .

6. Программа должна отображать наилучшую аппроксимирующую функцию.

7. Организовать вывод графиков функций, графики должны полностью отображать весь исследуемый интервал (с запасом).

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

# **Рабочие формулы**

**Линейная аппроксимация:**

**Квадратичная аппроксимация:**

# **Вычислительная часть задания**

Исследуемая функция:

Исследуемый интервал:

График функции:

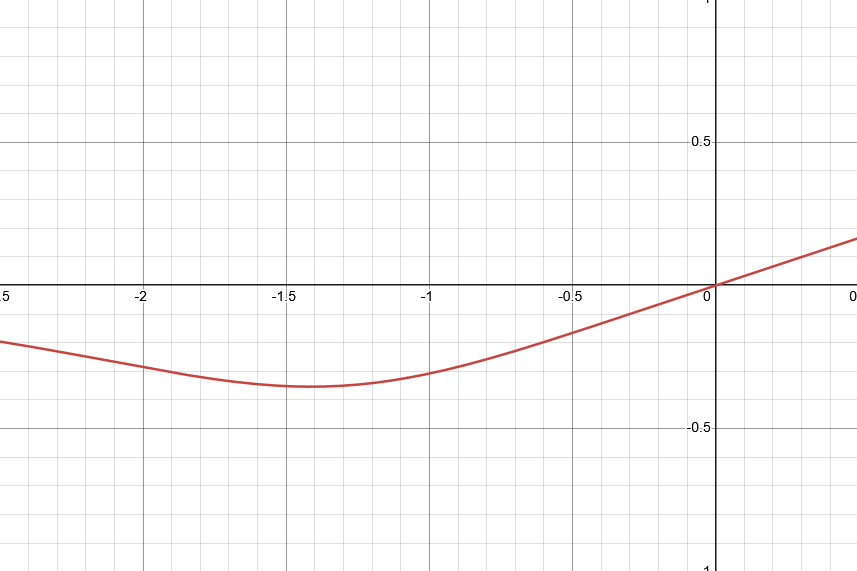


Таблица:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| x | -2 | -1.8 | -1.6 | -1.4 | -1.2 | -1 | -0.8 | -0.6 | -0.4 | -0.2 | 0 |
| y | -0.2857 | -0.32 | -0.3449 | -0.3535 | -0.3411 | -0.3077 | -0.2579 | -0.1979 | -0.133 | -0.0667 | 0 |

**Линейная аппроксимация:**

Суммы:

Рассчитаем

**Квадратичная аппроксимация:**

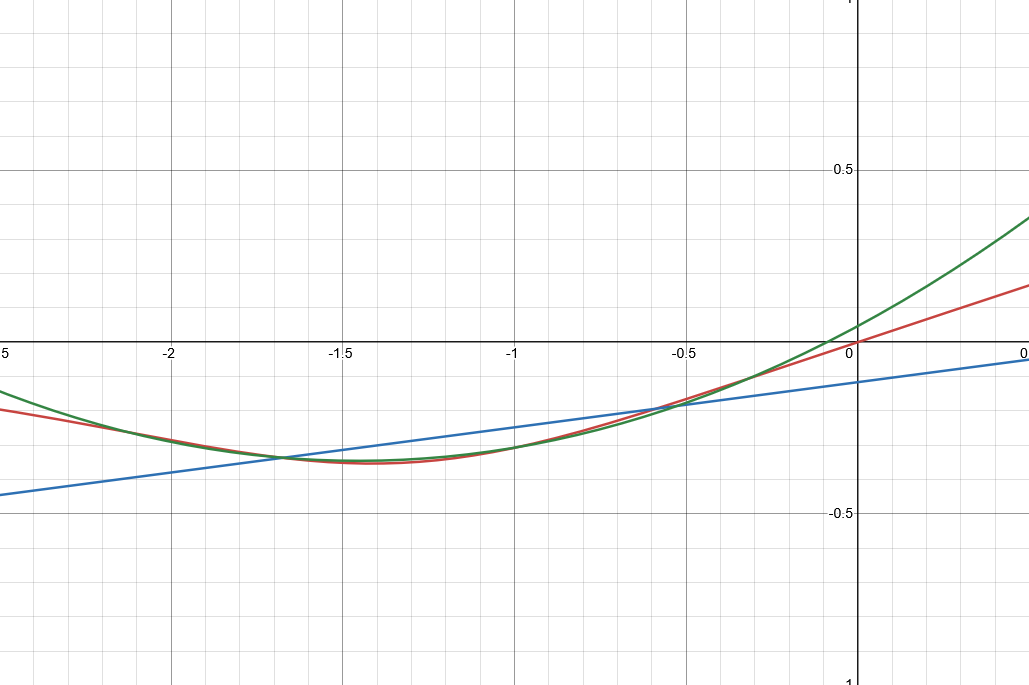
Суммы:

Рассчитаем

**Итоги:**

Наилучшее приближение – квадратичное (т.к. )

Графики:



Красная линия – исходная функция

Синяя линия – линейная аппроксимация

Красная линия – квадратичная аппроксимация

# **Программная часть задания**

**linear.py:**

from typing import Callable, Any, List

import sympy as sp

class LinearApproximation:

    def \_\_init\_\_(self, table: bool = False):

        self.table = table

    def solve(self, \*args, \*\*kwargs) -> Any:

        if self.table:

            return self.\_table\_solve(\*args, \*\*kwargs)

        return self.\_func\_solve(\*args, \*\*kwargs)

    def \_func\_solve(self, f: Callable, left: float, right: float, n: int, lambdify: bool = True) -> Any:

        sx = sxx = sy = sxy = 0

        h = (right - left) / n

        for i in range(n + 1):

            x = left + h \* i

            y = f(x)

            sx += x

            sxx += x\*\*2

            sy += y

            sxy += x\*y

        a, b = sp.symbols("a b")

        root = sp.solve([a \* sxx + b \* sx - sxy, a \* sx + b \* n - sy], [a, b], dict=True)[0]

        x = sp.Symbol("x")

        function = root[a] \* x + root[b]

        if lambdify:

            return sp.lambdify(x, function)

        return x, function

    def \_table\_solve(self, table: List[List[float]], lambdify: bool = True) -> Any:

        sx = sxx = sy = sxy = 0

        n = len(table[0])

        for x, y in zip(table[0], table[1]):

            sx += x

            sxx += x\*\*2

            sy += y

            sxy += x\*y

        a, b = sp.symbols("a b")

        root = sp.solve([a \* sxx + b \* sx - sxy, a \* sx + b \* n - sy], [a, b], dict=True)[0]

        x = sp.Symbol("x")

        function = root[a] \* x + root[b]

        if lambdify:

            return sp.lambdify(x, function)

        return x, function

**quadratic.py:**

from typing import Callable, Any, List

import sympy as sp

class QuadraticApproximation:

    def \_\_init\_\_(self, table: bool = False):

        self.table = table

    def solve(self, \*args, \*\*kwargs) -> Any:

        if self.table:

            return self.\_table\_solve(\*args, \*\*kwargs)

        return self.\_func\_solve(\*args, \*\*kwargs)

    def \_func\_solve(self, f: Callable, left: float, right: float, n: int, lambdify: bool = True) -> Any:

        sx = sx2 = sx3 = sx4 = sy = sxy = sx2y = 0

        h = (right - left) / n

        for i in range(n + 1):

            x = left + h \* i

            y = f(x)

            sx += x

            sx2 += x\*\*2

            sx3 += x\*\*3

            sx4 += x\*\*4

            sy += y

            sxy += x\*y

            sx2y += x\*\*2\*y

        a0, a1, a2 = sp.symbols("a0 a1 a2")

        root = sp.solve(

            [

                n \* a0 + sx \* a1 + sx2 \* a2 - sy,

                sx \* a0 + sx2 \* a1 + sx3 \* a2 - sxy,

                sx2 \* a0 + sx3 \* a1 + sx4 \* a2 - sx2y

            ],

            [a0, a1, a2],

            dict=True

        )[0]

        x = sp.Symbol("x")

        function = root[a0] + root[a1] \* x + root[a2] \* x \*\* 2

        if lambdify:

            return sp.lambdify(x, function)

        return x, function

    def \_table\_solve(self, table: List[List[float]], lambdify: bool = True) -> Any:

        sx = sx2 = sx3 = sx4 = sy = sxy = sx2y = 0

        n = len(table[0])

        for x, y in zip(table[0], table[1]):

            sx += x

            sx2 += x\*\*2

            sx3 += x\*\*3

            sx4 += x\*\*4

            sy += y

            sxy += x\*y

            sx2y += x\*\*2\*y

        a0, a1, a2 = sp.symbols("a0 a1 a2")

        root = sp.solve(

            [

                n \* a0 + sx \* a1 + sx2 \* a2 - sy,

                sx \* a0 + sx2 \* a1 + sx3 \* a2 - sxy,

                sx2 \* a0 + sx3 \* a1 + sx4 \* a2 - sx2y

            ],

            [a0, a1, a2],

            dict=True

        )[0]

        x = sp.Symbol("x")

        function = root[a0] + root[a1] \* x + root[a2] \* x \*\* 2

        if lambdify:

            return sp.lambdify(x, function)

        return x, function

# **Заключение**

В ходе лабораторной работы я изучил разные численные интегрирования и реализовал их на языке Python.