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 3

ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ

ВАРИАНТ 12

­

Студент: Пышкин Никита Сергеевич, P3213

Преподаватель:

Санкт Петербург 2025

Содержание

[**Цель лабораторной работы** 3](#_Toc193300059)

[**Порядок выполнения лабораторной работы** 3](#_Toc193300060)

[**Рабочие формулы используемых методов** 3](#_Toc193300061)

[**Вычисление заданного интеграла** 4](#_Toc193300062)

[**Сравнение результатов** 6](#_Toc193300063)

[**Расчет относительной погрешности** 6](#_Toc193300064)

[**Листинг программы** 6](#_Toc193300065)

[**Результаты выполнения программы при различных исходных данных** 6](#_Toc193300066)

[**Заключение** 6](#_Toc193300067)

# **Цель лабораторной работы**

Найти приближенное значение определенного интеграла с требуемой точностью различными численными методами.

# **Порядок выполнения лабораторной работы**

**Исходные данные:**

1. Пользователь выбирает функцию, интеграл которой требуется вычислить (3-5 функций), из тех, которые предлагает программа.

2. Пределы интегрирования задаются пользователем.

3. Точность вычисления задается пользователем.

4. Начальное значение числа разбиения интервала интегрирования: n=4.

5. Ввод исходных данных осуществляется с клавиатуры.

**Программная реализация задачи:**

1. Реализовать в программе методы по выбору пользователя:

• Метод прямоугольников (3 модификации: левые, правые, средние)

• Метод трапеций

• Метод Симпсона

2. Методы должны быть оформлены в виде отдельной(ого) функции/класса.

3. Вычисление значений функции оформить в виде отдельной(ого) функции/класса.

4. Для оценки погрешности и завершения вычислительного процесса использовать правило Рунге.

5. Предусмотреть вывод результатов: значение интеграла, число разбиения интервала интегрирования для достижения требуемой точности.

**Вычислительная реализация задачи:**

1. Вычислить интеграл, приведенный в таблице 1, точно.

2. Вычислить интеграл по формуле Ньютона – Котеса при 𝑛 = 6.

3. Вычислить интеграл по формулам средних прямоугольников, трапеций и Симпсона при 𝑛 = 10.

4. Сравнить результаты с точным значением интеграла.

5. Определить относительную погрешность вычислений для каждого метода.

6. В отчете отразить последовательные вычисления

# **Рабочие формулы используемых методов**

**Ньютона-Котеса**:

Рабочая формула:

**Метод прямоугольников:**

Рабочая формула:

Вычисление дельты:

Рабочая формула для средних:

**Метод трапеций:**

Рабочая формула:

Рабочая формула при h = const:

**Метод Симпсона:**

Рабочая формула:

# **Вычисление заданного интеграла**

**1) Точное вычисление**

**2) По формуле Ньютона-Котеса при n=6**

Пусть

Подставим коэффициенты из таблицы:

Подставим наши значения a и b:

Сократим:

Посчитаем:

**3) По формуле среднего прямоугольника при n = 10**

Составим табличку значений:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 1 | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 1.7 | 1.8 | 1.9 | 1.10 |
|  | -12 | 11.549 | -10.992 | -10.323 | -9.536 | -8.625 | -7.584 | -6.407 | -5.088 | -3.621 | -2 |
|  |  | 1.05 | 1.15 | 1.25 | 1.35 | 1.45 | 1.55 | 1.65 | 1.75 | 1.85 | 1.95 |
|  |  | -11.787375 | -11.284125 | -10.671875 | -9.944625 | -9.096375 | -8.121125 | -7.012875 | -5.765625 | -4.373375 | -2.830125 |

Вычислим результат:

**4) По формуле трапеций при n = 10**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 1 | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 1.7 | 1.8 | 1.9 | 1.10 |
|  | -12 | -11.549 | -10.992 | -10.323 | -9.536 | -8.625 | -7.584 | -6.407 | -5.088 | -3.621 | -2 |

**5) По формуле Симпсона при n = 10**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 1 | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 1.7 | 1.8 | 1.9 | 1.10 |
|  | -12 | -11.549 | -10.992 | -10.323 | -9.536 | -8.625 | -7.584 | -6.407 | -5.088 | -3.621 | -2 |

# **Сравнение результатов**

Результат точных вычисление:

Метод Ньютона-Котеса: – не отличается от точных вычислений

Метод средних прямоугольников: – отличается от точных вычислений на

Метод трапеций: – отличается от точных вычислений на

Метод Симпсона: – не отличается от точных вычислений

# **Расчет относительной погрешности**

Метод Ньютона-Котеса: погрешности нет

Метод средних прямоугольников:

Метод трапеций:

Метод Симпсона: погрешности нет

# **Листинг программы**

abstract\_solve\_integral\_method.py:

from typing import Tuple, Callable

from abc import ABC, abstractmethod

class AbstractSolveIntegralMethod(ABC):

    @classmethod

    @abstractmethod

    def solve(cls, integral\_func: Callable[[float], float], a: float, b: float, e: float, max\_iter\_count: int = 10\*\*6) -> Tuple[int, int]:

        pass

    @classmethod

    def \_check\_calc\_is\_end(cls, first\_value: float, second\_value: float, k: int, e: float) -> bool:

        return abs(first\_value - second\_value) / (2\*\*k + 1) < e

rectangle\_method.py:

from enum import IntEnum

from typing import Tuple, Callable

from .core import AbstractSolveIntegralMethod

class RectangleMethodType(IntEnum):

    LEFT = 0

    MIDDLE = 1

    RIGHT = 2

class RectangleMethod(AbstractSolveIntegralMethod):

    @classmethod

    def solve(cls, integral\_func: Callable[[float], float], a: float, b: float, e: float, max\_iter\_count: int = 10\*\*6, m\_type: RectangleMethodType = RectangleMethodType.MIDDLE) -> Tuple[int, int]:

        results, n = [], 4

        while len(results) < 2 or not cls.\_check\_calc\_is\_end(results[-2], results[-1], 2, e):

            if n >= max\_iter\_count:

                raise ValueError(f"Не удалось найти решение с удовлетворяющей точностью за {max\_iter\_count} разбиений")

            h = (b - a) / n

            match m\_type:

                case RectangleMethodType.LEFT:

                    start = a

                case RectangleMethodType.MIDDLE:

                    start = a + h / 2

                case RectangleMethodType.RIGHT:

                    start = a + h

            s = 0

            for i in range(n):

                s += integral\_func(start + h \* i)

            results.append(s \* h)

            n += 1

        return results[-1], n

simpsons\_method.py:

from typing import Tuple, Callable

from .core import AbstractSolveIntegralMethod

class SimpsonsMethod(AbstractSolveIntegralMethod):

    @classmethod

    def solve(cls, integral\_func: Callable[[float], float], a: float, b: float, e: float, max\_iter\_count: int = 10\*\*6) -> Tuple[int, int]:

        results, n = [], 4

        while len(results) < 2 or not cls.\_check\_calc\_is\_end(results[-2], results[-1], 4, e):

            if n >= max\_iter\_count:

                raise ValueError(f"Не удалось найти решение с удовлетворяющей точностью за {max\_iter\_count} разбиений")

            h = (b - a) / n

            s = 0

            for i in range(n):

                result = integral\_func(a + h \* i)

                if i % 2 == 0 and i != 0 and i != n - 1:

                    result \*= 4

                elif i % 2 != 0 and i != 0 and i != n - 1:

                    result \*= 2

                s += result

            results.append(s \* (h / 3))

            n += 2

        return results[-1], n

trapezoid\_method.py:

from typing import Tuple, Callable

from .core import AbstractSolveIntegralMethod

class TrapezoidMethod(AbstractSolveIntegralMethod):

    @classmethod

    def solve(cls, integral\_func: Callable[[float], float], a: float, b: float, e: float, max\_iter\_count: int = 10\*\*6) -> Tuple[int, int]:

        results, n = [], 4

        while len(results) < 2 or not cls.\_check\_calc\_is\_end(results[-2], results[-1], 2, e):

            if n >= max\_iter\_count:

                raise ValueError(f"Не удалось найти решение с удовлетворяющей точностью за {max\_iter\_count} разбиений")

            h = (b - a) / n

            s = 0

            for i in range(n):

                result = integral\_func(a + h \* i)

                if i != 0 and i != n - 1:

                    result \*= 2

                s += result

            results.append(s \* (h / 2))

            n += 1

        return results[-1], n

# **Результаты выполнения программы при различных исходных данных**

**Пример 1:**

Выберите функцию:

1. x\*\*2 - 0.5

2. x\*\*3 - 4\*x + 1

3. 2\*\*x - 4

4. x\*\*3 + 2\*x\*\*2 - 3\*x - 12

Введите номер функции: 4

Выберите метод:

1. Метод прямоугольников

2. Метод трапеций

3. Метод Симпсонов

Введите номер метода: 3

Введите пределы интегрирования a b: 1 2

Введите точность вычисления: 0.0001

Результат: -8.014143886878387 (найден за 90 разбиений)

**Пример 2:**

Выберите функцию:

1. x\*\*2 - 0.5

2. x\*\*3 - 4\*x + 1

3. 2\*\*x - 4

4. x\*\*3 + 2\*x\*\*2 - 3\*x - 12

Введите номер функции: 2

Выберите метод:

1. Метод прямоугольников

2. Метод трапеций

3. Метод Симпсонов

Введите номер метода: 1

Выберите тип метода:

1. Метод левых прямоугольников

2. Метод средних прямоугольников

3. Метод правых прямоугольников

Введите номер метода: 3

Введите пределы интегрирования a b: 0.00001

Некорректный ввод, проверьте соответствие формату.

Введите пределы интегрирования a b: 1 5

Введите точность вычисления: 0.00001

Результат: 112.1038683431954 (найден за 2081 разбиений)

# **Заключение**

В ходе лабораторной работы я изучил разные численные методы, их погрешности и реализовал их на языке Python.