МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

**ЛАБОРАТОРНАЯ РАБОТА №3**

по дисциплине

«ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА»

Вариант № 13

***Выполнил:***

Студент группы P3218

Рамеев Тимур

Ильгизович

***Преподаватель:***

Бострикова Дарья

Константиновна

# Цель

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

# Используемые методы

## Метод прямоугольников

Наиболее простой метод поиска приближенного значения интеграла, метод прямоугольников, состоит в приближении площади под графиком интегрируемой функции серией прямоугольников. При этом выделяют 3 разновидности этого метода: метод левых, правых и средних прямоугольников, различающихся выбором точки, для которой считается значение функции – высота прямоугольника.

Метод левых прямоугольников:

Метод правых прямоугольников:

Метод средних прямоугольников:

## Метод трапеций

В методе трапеций на каждом участке разбиения кусочек функции интерполируется прямой. Интеграл считается для каждого интерполированного отрезка и суммируется (аналогично можно рассуждать и о приближении площади под графиком).

Рабочая формула для равномерного разбиения на отрезки длиной h:

## Метод Симпсона

Аналогично методу трапеций разобьём промежуток интегрирования на n-четное число отрезков с шагом h и интерполируем функцию на каждом таком промежутке многочленом 2-го порядка. Будем искать интегралы для таких интерполированных кусочков функции.

## Метод Ньютона-Котеса

Обобщением упомянутых методов является метод Ньютона-Котеса, основанный на разделении предела интегрирования на равные интервалы (далее n – число таких интервалов) и интерполяции интегрируемой функции на этих интервалах многочленом Лагранжа.

При этом удобно ввести коэффициент:

Для этих коэффициентов существуют таблицы.

# Вычислительная часть

Найдем точное решение интеграла ниже и сравним его с приближенными значениями, полученными по формуле ньютона-Котеса (n = 6), формуле средних прямоугольников, трапеций и Симпсона (n = 10):

## Точное решение

Найдем точное решение интеграла:

## Формула Ньютона-Котеса

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|  | 1 | 1.333 | 1.666 | 2 | 2.333 | 2.666 | 3 |
| f() | -13 | -17.291 | -24.490 | -35 | -49.280 | -67.836 | -91 |
|  |  |  |  |  |  |  |  |
|  | 0.0976 | 0.514 | 0.0642 | 0.6476 | 0.0642 | 0.514 | 0.0976 |
|  | -1.269 | -8.892 | -1.573 | -22.667 | -3.168 | -34.855 | -8.883 |

Видно, что результат вычислений метода Ньютона-Котеса ниже истинного значения (по модулю), рассчитанного по формуле Ньютона-Лейбница.

Найдем относительную погрешность вычислений для метода:

## Метод средних прямоугольников

Найдем длину элементарного промежутка:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 1 | 1.2 | 1.4 | 1.6 | 1.8 | 2 | 2.2 | 2.4 | 2.6 | 2.8 | 3 |
|  | - | 1.1 | 1.3 | 1.5 | 1.7 | 1.9 | 2.1 | 2.3 | 2.5 | 2.7 | 2.9 |
|  | - | -14.012 | -16.744 | -20.5 | -25.376 | -31.468 | -38.872 | -47.684 | -58 | -69.916 | -83.528 |
|  | - | -2.802 | -3.349 | -4.1 | -5.075 | -6.293 | -7.774 | -9.537 | -11.6 | -13.983 | -16.706 |

Видно, что результат вычислений метода средних прямоугольников ниже истинного значения (по модулю), рассчитанного по формуле Ньютона-Лейбница.

Найдем относительную погрешность вычислений для метода:

## Метод трапеций

Найдем длину элементарного промежутка:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 1 | 1.2 | 1.4 | 1.6 | 1.8 | 2 | 2.2 | 2.4 | 2.6 | 2.8 | 3 |
|  | -13 | -15.256 | -18.488 | -22.792 | -28.264 | -35 | -43.096 | -52.648 | -63.752 | -76.504 | -91 |

Видно, что результат вычислений метода трапеций выше истинного значения (по модулю), рассчитанного по формуле Ньютона-Лейбница.

Найдем относительную погрешность вычислений для метода:

## Метод Симпсона

Найдем длину элементарного промежутка:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 1 | 1.2 | 1.4 | 1.6 | 1.8 | 2 | 2.2 | 2.4 | 2.6 | 2.8 | 3 |
|  | -13 | -15.256 | -18.488 | -22.792 | -28.264 | -35 | -43.096 | -52.648 | -63.752 | -76.504 | -91 |

Видно, что результат вычислений метода Симпсона равен истинному значению, рассчитанному по формуле Ньютона-Лейбница.

Найдем относительную погрешность вычислений для метода:

# Реализация и пример работы

Программа, реализующая «метод Прямоугольников», «метод Трапеций» и «метод Симпсона» для численного интегрирования, была написана на языке Python

def rectangle\_left(equation, left, right, accuracy):

    n = 4

    h = (right - left) / n

    method = define\_method(equation)

    current\_value = 0

    current\_pointer = left

    while current\_pointer <= right:

        current\_value += method(current\_pointer) \* h

        current\_pointer += h

    prev\_value = current\_value + 10000

    counter = 1

    while abs(current\_value - prev\_value) > accuracy :

        counter += 1

        if counter == 25:

            return {"status" : 2, "value" : current\_value, "number" : n, "warning" : "Количество итераций превисило допустимое значение"}

        prev\_value = current\_value

        n \*= 2

        h = (right - left) / n

        current\_value = 0

        current\_pointer = left

        while current\_pointer <= right:

            current\_value += method(current\_pointer) \* h

            current\_pointer += h

    return {"status" : 1, "value" : current\_value, "number" : n}

def rectangle\_center(equation, left, right, accuracy):

    n = 4

    h = (right - left) / n

    method = define\_method(equation)

    current\_value = 0

    current\_pointer = left

    while current\_pointer + h / 2 <= right:

        current\_value += method(current\_pointer + h / 2) \* h

        current\_pointer += h

    prev\_value = current\_value + 10000

    counter = 1

    while abs(current\_value - prev\_value) > accuracy :

        counter += 1

        if counter == 25:

            return {"status" : 2, "value" : current\_value, "number" : n, "warning" : "Количество итераций превисило допустимое значение"}

        prev\_value = current\_value

        n \*= 2

        h = (right - left) / n

        current\_value = 0

        current\_pointer = left

        while current\_pointer + h / 2 <= right:

            current\_value += method(current\_pointer + h / 2) \* h

            current\_pointer += h

    return {"status" : 1, "value" : current\_value, "number" : n}

def rectangle\_right(equation, left, right, accuracy):

    n = 4

    h = (right - left) / n

    method = define\_method(equation)

    current\_value = 0

    current\_pointer = left

    while current\_pointer + h <= right:

        current\_value += method(current\_pointer + h) \* h

        current\_pointer += h

    prev\_value = current\_value + 10000

    counter = 1

    while abs(current\_value - prev\_value) > accuracy :

        counter += 1

        if counter == 25:

            return {"status" : 2, "value" : current\_value, "number" : n, "warning" : "Количество итераций превисило допустимое значение"}

        prev\_value = current\_value

        n \*= 2

        h = (right - left) / n

        current\_value = 0

        current\_pointer = left

        while current\_pointer + h <= right:

            current\_value += method(current\_pointer + h) \* h

            current\_pointer += h

    return {"status" : 1, "value" : current\_value, "number" : n}

def trapezoid(equation, left, right, accuracy):

    n = 4

    h = (right - left) / n

    method = define\_method(equation)

    current\_value = 0

    current\_pointer = left

    y\_values = []

    while current\_pointer <= right:

        y\_values.append(method(current\_pointer))

        current\_pointer += h

    for i in range(len(y\_values)):

        if  i == 0 or i == len(y\_values) - 1:

            current\_value += y\_values[i] / 2

        else:

            current\_value += y\_values[i]

    current\_value \*= h

    prev\_value = current\_value + 10000

    counter = 1

    while abs(current\_value - prev\_value) > accuracy :

        counter += 1

        if counter == 24:

            return {"status" : 2, "value" : current\_value, "number" : n, "warning" : "Количество итераций превисило допустимое значение"}

        prev\_value = current\_value

        n \*= 2

        h = (right - left) / n

        current\_value = 0

        current\_pointer = left

        y\_values = []

        while current\_pointer <= right:

            y\_values.append(method(current\_pointer))

            current\_pointer += h

        for i in range(len(y\_values)):

            if  i == 0 or i == len(y\_values) - 1:

                current\_value += y\_values[i] / 2

            else:

                current\_value += y\_values[i]

        current\_value \*= h

    return {"status" : 1, "value" : current\_value, "number" : n}

def simpson(equation, left, right, accuracy):

    n = 4

    h = (right - left) / n

    method = define\_method(equation)

    current\_value = 0

    current\_pointer = left

    y\_values = []

    while current\_pointer <= right:

        y\_values.append(method(current\_pointer))

        current\_pointer += h

    for i in range(len(y\_values)):

        if  i == 0 or i == len(y\_values) - 1:

            current\_value += y\_values[i]

        elif i % 2 == 1:

            current\_value += y\_values[i] \* 4

        else:

            current\_value += y\_values[i] \* 2

    current\_value \*= h / 3

    prev\_value = current\_value + 10000

    counter = 1

    while abs(current\_value - prev\_value) > accuracy :

        counter += 1

        if counter == 24:

            return {"status" : 2, "value" : current\_value, "number" : n, "warning" : "Количество итераций превисило допустимое значение"}

        prev\_value = current\_value

        n \*= 2

        h = (right - left) / n

        current\_value = 0

        current\_pointer = left

        y\_values = []

        while current\_pointer <= right:

            y\_values.append(method(current\_pointer))

            current\_pointer += h

        for i in range(len(y\_values)):

            if  i == 0 or i == len(y\_values) - 1:

                current\_value += y\_values[i]

            elif i % 2 == 1:

                current\_value += y\_values[i] \* 4

            else:

                current\_value += y\_values[i] \* 2

        current\_value \*= h / 3

    return {"status" : 1, "value" : current\_value, "number" : n}

# Вывод

В ходе лабораторной работе я численно искал значения определенных и собственных интегралов двумя способами: вручную и с помощью программы. Сравнение этих двух результатов дало понимание правильности подхода.

1. Метод левых прямоугольников:

Преимущества:

Прост в реализации

Высокая скорость вычислений

Недостатки:

Может давить недостаточно точный результат для некоторых функций, особенно если функция имеет большие значения на концах отрезка.

1. Метод правых прямоугольников:

Преимущества:

Прост в реализации

Высокая скорость вычислений

Недостатки:

Может давить недостаточно точный результат для некоторых функций, особенно если функция имеет большие значения на концах отрезка.

1. Метод средних прямоугольников:

Преимущества:

Прост в реализации

Дает более точный результат, чем методы левых и правых прямоугольников

Недостатки:

Требует больше вычислительных ресурсов

1. Метод трапеций:

Преимущества:

Прост в реализации

Дает более точный результат, чем методы прямоугольника

Недостатки:

Требует больше вычислительных ресурсов, чем методы прямоугольников.

1. Метод Симпсона:

Преимущества:

Обеспечивает еще более высокую точность по сравнению с методом трапеций

Недостатки:

Требует больше вычислительных ресурсов, чем методы прямоугольников и трапеций.

Сложнее в реализации