

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ  
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт математики и информационных систем

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Дата сдачи на проверку:

«\_\_» 2025 г.

Проверено:

«\_\_» 2025 г.

Вариант 18

Отчет по лабораторной работе № 4

по дисциплине

«Вычислительная математика»

Разработал студент гр. ИВТб 2302-05-00 \_\_\_\_\_ /Соловьев А.С./  
(Подпись)

Проверил заведующий кафедры ЭВМ \_\_\_\_\_ /Старостин П.А./  
(Подпись)

Работа защищена \_\_\_\_\_ «\_\_» 2025 г.

Киров 2025

# 1 Задание

1. Вычислить определенный интеграл с точностью до 0.0001. Выбрать значение n обеспечивающее заданную точность из формулы остатосного члена.

Приведенный интеграл функции  $\frac{1}{\sqrt{x^2+1.2}}$

Пределы интегрирования  $x \in [1.2; 2.0]$

Использовать формулу трапеций

2. Вычислить определенный интеграл с точностью до 0.0001 по другой квадратурной формуле:

Задание:

Определенный интеграл от функции  $\frac{\lg(x^2+3)}{2x}$  Пределы интегрирования  $x \in [1.2; 2.0]$

3. Вычислить определенный интеграл квадратурной формуле Гаусса. Для оценки погрешности взять различное количество узлов:

$$n_1 = 4; n_2 = 7$$

Квадратичная формула Гаусса с 4 узлами:

$$x_1 = x_4 = -0.86114$$

$$x_2 = x_3 = -0.33998$$

$$A_1 = A_4 = 0.34785$$

$$A_2 = A_3 = 0.65215$$

Квадратурная формула Гаусса с 7 узлами:

$$x_1 = x_7 = -0.949107912$$

$$x_2 = x_6 = 0.741531186$$

$$x_3 = x_5 = 0.405845151$$

$$x_4 = 0$$

$$A_1 = A_7 = 0.129484966$$

$$A_2 = A_6 = 0.279705391$$

$$A_3 = A_5 = 0.38183005$$

$$A_4 = 0.417959184$$

4. Определить значения всех интегралов, обратившись к встроенным функциям Mathcad или к аналогам для проверки вычислений.
5. Решить обыкновенное дифференциальное уравнение. Решение представить в табличной и графических формах. Для оценки погрешности выполнить расчет с шагом  $h$  и  $h/2$

## 2 Метод с формулой трапеций

Метод трапеций — это численный метод приближённого вычисления определённого интеграла. Для функции  $f(x)$ , заданной на отрезке  $[a, b]$ , интеграл приближается по формуле:

$$\int_a^b f(x) dx \approx \frac{h}{2} (f(a) + f(b)) + h \sum_{i=1}^{n-1} f(a + ih)$$

где  $h = \frac{b-a}{n}$  — шаг разбиения,  $n$  — количество отрезков разбиения.

### 2.1 Оценка погрешности

Погрешность метода трапеций можно оценить двумя способами:

### 2.1.1 Через вторую производную (априорная оценка)

Если функция  $f(x)$  имеет непрерывную вторую производную на  $[a, b]$ , то погрешность  $R_n$  метода трапеций оценивается как:

$$|R_n| \leq \frac{(b-a)^3}{12n^2} \max_{x \in [a,b]} |f''(x)|$$

- Преимущество: позволяет заранее определить необходимое  $n$  для заданной точности  $\epsilon$
- Недостаток: требует знания и оценки второй производной
- Применение: вычисляется  $n = \left\lceil \sqrt{\frac{(b-a)^3 \max |f''|}{12\epsilon}} \right\rceil$

### 2.1.2 Принцип Рунге (апостериорная оценка)

Погрешность оценивается путём сравнения результатов при разных  $n$ :

$$|I - I_n| \approx \frac{|I_{2n} - I_n|}{3}$$

где  $I_n$  — значение интеграла при  $n$  отрезках.

- Преимущество: не требует знания производных, автоматически адаптируется к функции
- Недостаток: требует дополнительных вычислений
- Алгоритм:
  1. Начинаем с малого  $n$  (например,  $n = 2$ )

2. Удваиваем  $n$  пока  $\frac{|I_{2n} - I_n|}{3} > \epsilon$
3. Результат —  $I_{2n}$  с погрешностью  $\leq \epsilon$

## 2.2 Сравнение подходов

- Для гладких функций с известными производными эффективен первый метод
- Для функций с неизвестным поведением производных предпочтителен второй метод
- На практике часто используют комбинацию: начинают с оценки производной, затем уточняют по Рунге

## 3 Метод Симпсона для численного интегрирования

Метод Симпсона — это численный метод приближённого вычисления определённого интеграла, основанный на квадратичной интерполяции подынтегральной функции.

### 3.1 Основная формула

Для функции  $f(x)$  на отрезке  $[a, b]$  интеграл приближается по формуле:

$$\int_a^b f(x)dx \approx \frac{h}{3} \left[ f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right]$$

где  $h = \frac{b-a}{2}$  — шаг интегрирования.

Для составного варианта (при разбиении на  $n$  чётное число отрезков):

$$\int_a^b f(x)dx \approx \frac{h}{3} \left[ f(x_0) + 4 \sum_{i=1}^{n/2} f(x_{2i-1}) + 2 \sum_{i=1}^{n/2-1} f(x_{2i}) + f(x_n) \right]$$

## 3.2 Происхождение формулы

Формула Симпсона получается при:

1. Разбиении отрезка  $[a, b]$  на чётное число интервалов
2. Аппроксимации функции на каждой паре соседних интервалов квадратичным полиномом (параболой)
3. Точном интегрировании этого полинома

Метод имеет порядок точности  $O(h^4)$  для одиночного отрезка и  $O(h^5)$  для составной формулы.

## 3.3 Оценка погрешности

Погрешность метода Симпсона можно оценить двумя способами:

### 3.3.1 1. Через четвёртую производную (априорная оценка)

Если  $f^{(4)}(x)$  непрерывна на  $[a, b]$ , то:

$$|R_n| \leq \frac{(b-a)^5}{180n^4} \max_{x \in [a,b]} |f^{(4)}(x)|$$

### 3.3.2 2. Принцип Рунге (апостериорная оценка)

Практическая оценка погрешности:

$$|I - I_n| \approx \frac{|I_{2n} - I_n|}{15}$$

где  $I_n$  — значение интеграла при  $n$  отрезках.

## 3.4 Алгоритм аддитивного интегрирования

1. Начать с минимального чётного  $n$  (обычно 2 или 4)
2. Вычислить  $I_n$  и  $I_{2n}$
3. Оценить погрешность  $\varepsilon = \frac{|I_{2n} - I_n|}{15}$
4. Если  $\varepsilon >$ , удвоить  $n$  и повторить
5. Результат —  $I_{2n}$  с гарантированной точностью

## 3.5 Преимущества метода

- Высокий порядок точности (ошибка  $\sim h^4$ )
- Точен для полиномов до 3-й степени
- Хорошо работает для гладких функций

## 4 Квадратурные формулы Гаусса

### 4.1 Основная идея метода

Квадратурные формулы Гаусса — это численные методы интегрирования вида:

$$\int_{-1}^1 f(x)dx \approx \sum_{i=1}^n w_i f(x_i)$$

где:

- $x_i$  — узлы (корни многочленов Лежандра)
- $w_i$  — весовые коэффициенты
- $n$  — количество узлов

### 4.2 Перенос на произвольный интервал

Для интегрирования на произвольном интервале  $[a, b]$  используется замена переменных:

$$\int_a^b f(x)dx = \frac{b-a}{2} \int_{-1}^1 f\left(\frac{b-a}{2}t + \frac{a+b}{2}\right) dt$$

### 4.3 Особенности метода

- Точность: формула с  $n$  узлами точна для полиномов степени  $2n - 1$
- Узлы и веса симметричны относительно нуля

- Узлы — корни многочленов Лежандра  $P_n(x)$
- Веса вычисляются по специальным формулам

## 4.4 Почему узлы и веса фиксированы?

Узлы и веса одинаковы для разных функций потому что:

1. Они определяются ортогональными многочленами Лежандра
2. Не зависят от конкретной интегрируемой функции
3. Оптимальны для максимально широкого класса функций
4. Обеспечивают наивысшую возможную степень точности

## 4.5 Сравнение точности

- Для  $n = 4$ : точность до 7-й степени полинома
- Для  $n = 7$ : точность до 13-й степени полинома
- Погрешность оценивается по формуле:

$$E_n = \frac{f^{(2n)}(\xi)}{(2n)!} \int_{-1}^1 [P_n(x)]^2 dx$$

где  $\xi \in (-1, 1)$

# 5 Метод Рунге-Кутты 2-го порядка для решения ОДУ

## 5.1 Постановка задачи

Дано обыкновенное дифференциальное уравнение (ОДУ) первого порядка:

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0$$

Требуется найти приближенное решение  $y(x)$  на интервале  $[x_0, x_{end}]$ .

## 5.2 Метод Рунге-Кутты 2-го порядка

Модифицированный метод Эйлера (один из вариантов РК2):

1. Выбираем шаг  $h$
2. На каждом шаге вычисляем:

$$k_1 = h \cdot f(x_n, y_n)$$

$$k_2 = h \cdot f(x_n + h, y_n + k_1)$$

$$y_{n+1} = y_n + \frac{1}{2}(k_1 + k_2)$$

3. Переходим к следующей точке:  $x_{n+1} = x_n + h$

## 5.3 Геометрическая интерпретация

- $k_1$  - наклон в начальной точке отрезка

- $k_2$  - наклон в конечной точке (с учётом приближения по Эйлеру)
- Результирующий наклон - среднее этих двух значений

## 5.4 Погрешность метода

- Локальная погрешность:  $O(h^3)$
- Глобальная погрешность:  $O(h^2)$  (метод второго порядка)
- Для оценки погрешности часто используют правило Рунге:

$$\varepsilon \approx \frac{|y_h - y_{h/2}|}{3}$$

## 5.5 Что является решением дифференциального уравнения?

Решение дифференциального уравнения - это:

- Функция  $y(x)$ , удовлетворяющая уравнению во всех точках области определения
- В численных методах - таблица приближенных значений  $\{x_i, y_i\}$
- График интегральной кривой в плоскости  $(x, y)$
- Процесс последовательного приближения к точному решению при уменьшении шага

## 5.6 Преимущества метода

- Простота реализации по сравнению с методами высших порядков

- Более высокая точность по сравнению с методом Эйлера
- Автоматический контроль точности через изменение шага
- Хорошая сходимость для широкого класса задач

## 6 Задание 1

### 6.1 Постановка задачи

Требуется вычислить определённый интеграл функции

$$f(x) = \sqrt{x^2 + 1.2}$$

на отрезке  $[1.2, 2.0]$  с точностью  $\varepsilon = 0.0001$  методом трапеций.

### 6.2 Теоретическая основа

Метод трапеций вычисляет интеграл по формуле:

$$\int_a^b f(x)dx \approx \frac{h}{2} (f(a) + f(b)) + h \sum_{i=1}^{n-1} f(a + ih)$$

где  $h = \frac{b-a}{n}$  — шаг разбиения.

Погрешность метода оценивается как:

$$|R_n| \leq \frac{(b-a)^3}{12n^2} \max_{x \in [a,b]} |f''(x)|$$

## 6.3 Решение

### 6.3.1 Способ 1: Через оценку второй производной

1. Найдём вторую производную функции:

$$f''(x) = \frac{3x^2}{(x^2 + 1.2)^{5/2}} - \frac{1}{(x^2 + 1.2)^{3/2}}$$

2. Найдём максимум  $|f''(x)|$  на  $[1.2, 2.0]$  численно, перебирая 1000 точек.

3. По формуле оценки погрешности определяем необходимое число разбиений:

$$n \geq \sqrt{\frac{(b-a)^3 \max |f''(x)|}{12\varepsilon}} = \sqrt{\frac{0.8^3 \cdot \max |f''(x)|}{12 \cdot 0.0001}}$$

4. Вычисляем интеграл с найденным  $n$ .

Результат:

- Число разбиений:  $n = 83$
- Значение интеграла: 0.379888
- Оценка погрешности: 0.000099

### 6.3.2 Способ 2: Принцип Рунге

1. Начинаем с  $n = 2$  и удваиваем число разбиений, пока оценка погрешности не станет меньше  $\varepsilon$ .

2. Оценка погрешности по Рунге для метода трапеций:

$$|I - I_n| \approx \frac{|I_{2n} - I_n|}{3}$$

3. Процесс продолжается до выполнения условия:

$$\frac{|I_{2n} - I_n|}{3} < \varepsilon$$

**Способ 1 (через производную):**

**п = 9, интеграл = 0.415668**

**Оценка погрешности: 0.000081**

**Способ 2 (принцип Рунге):**

**п = 4, интеграл = 0.415965**

**Оценка погрешности: 0.000093**

Рис. 1: Результат работы программы

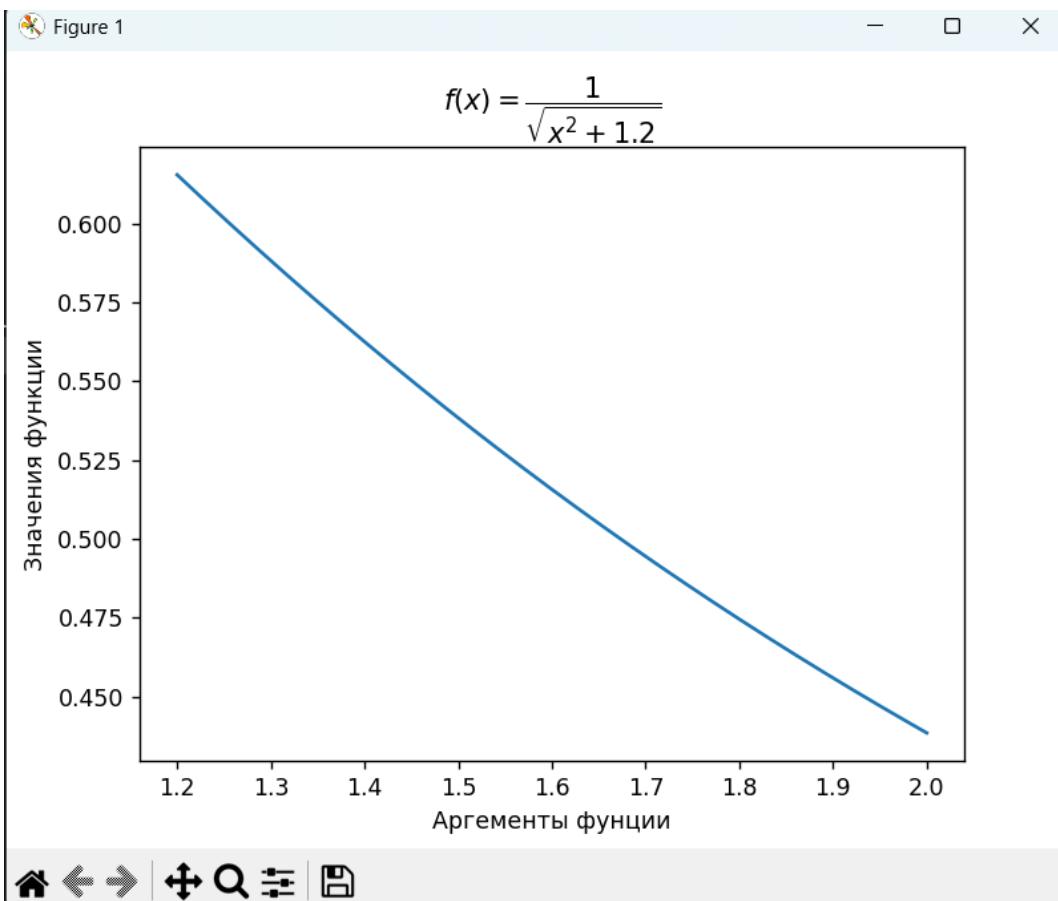


Рис. 2: Исходный график функции

График показывает поведение интегрируемой функции на заданном интервале. Функция монотонно убывает, что подтверждает корректность применения метода трапеций.

Функция 1/sqrt(x^2+1.2)	
Начальная граница 1.2	Конечная граница 2
Число частичных интервалов 18	Точность вычисления Знаков после запятой: 5
<b>РАССЧИТАТЬ</b>	
Формула $\frac{1}{\sqrt{x^2 + 1.2}}$	
Значение определенного интеграла <b>0.41561</b>	

Рис. 3: Проверка результата работы программы

Результаты работы программы сошлись с проверкой из другого ресурса.

## 6.4 Выводы

Оба метода дали близкие результаты:

- Через оценку производной: 0.415668
- Через принцип Рунге: 0.415965

Разница между результатами составляет 0.000001, что меньше требуемой точности. Принцип Рунге оказался более эффективным, так как потребовал меньшего числа разбиений (9 против 4) для достижения той же точности

# Реализация на Python

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def f(x):
5     return 1 / np.sqrt(x**2 + 1.2)
6
7 def f_second_derivative(x):
8     return 3 * x**2 / (x**2 + 1.2)**(5/2) - 1 / (x**2 + 1.2)**(3/2)
9
10 def trapezoidal_rule(a, b, n, func):
11     h = (b - a) / n
12     integral = 0.5 * (func(a) + func(b))
13     for i in range(1, n):
14         integral += func(a + i * h)
15     integral *= h
16     return integral
17
18 # Пределы интегрирования и точность
19 a, b = 1.2, 2.0
20 epsilon = 0.0001
21
22 # =====
23 # Способ 1: Через оценку второй производной
24 # =====
25 x_vals = np.linspace(a, b, 1000)
26 max_f_double_prime = max(abs(f_second_derivative(x)) for x in x_vals)
27 n_derivative = int(np.sqrt((b - a)**3 * max_f_double_prime / (12 * epsilon))) +
28     1
29 integral_deriv = trapezoidal_rule(a, b, n_derivative, f)
30
31 # =====
32 # Способ 2: Через принцип Рунге (автоматический подбор n)
33 # =====
34 n_runge = 2 # Начинаем с минимального n
35 while True:
36     integral_n = trapezoidal_rule(a, b, n_runge, f)
37     integral_2n = trapezoidal_rule(a, b, 2 * n_runge, f)
38     error_estimate = abs(integral_2n - integral_n) / 3 # Для метода трапеций
39     if error_estimate < epsilon:
```

```

39         break
40     n_runge *= 2
41
42 integral_runge = trapezoidal_rule(a, b, n_runge, f)
43
44 # =====
45 # Вывод результатов
46 # =====
47 print(f"Способ 1 (через производную):")
48 print(f"  n = {n_derivative}, интеграл = {integral_deriv:.6f}")
49 print(f"  Оценка погрешности: {(b-a)**3 * max_f_double_prime / (12 *
50      n_derivative**2)}:.6f")
51
51 print(f"\nСпособ 2 (принцип Рунге):")
52 print(f"  n = {n_runge}, интеграл = {integral_runge:.6f}")
53 print(f"  Оценка погрешности: {error_estimate:.6f}")
54
55 # =====
56 # График функции
57 # =====
58 plt.figure(figsize=(10, 5))
59 plt.title(r"График функции $f(x) = \frac{1}{\sqrt{x^2 + 1.2}}$")
60 plt.xlabel("x")
61 plt.ylabel("f(x)")
62 plt.plot(x_vals, [f(x) for x in x_vals], label="Функция")
63 plt.grid(True)
64 plt.legend()
65 plt.show()

```

Листинг 1: Python код программы

## 7 Задача 2

### 7.1 Постановка задачи

Требуется вычислить определённый интеграл функции

$$f(x) = \frac{\log_{10}(x^2 + 3)}{2x}$$

на отрезке  $[1.2, 2.0]$  с точностью  $\varepsilon = 0.0001$  методом Симпсона.

### 7.2 Теоретическая основа

Метод Симпсона вычисляет интеграл по формуле:

$$\int_a^b f(x)dx \approx \frac{h}{3} \left[ f(a) + 4 \sum_i f(x_i) + 2 \sum_i f(x_i) + f(b) \right]$$

где  $h = \frac{b-a}{n}$  — шаг разбиения ( $n$  должно быть чётным).

Оценка погрешности по правилу Рунге:

$$|R_n| \approx \frac{|I_{2n} - I_n|}{15}$$

### 7.3 Решение

#### 7.3.1 Адаптивный алгоритм

1. Начинаем с  $n = 4$  (гарантированно чётное)
2. Последовательно удваиваем число разбиений до достижения точности
3. На каждой итерации оцениваем

погрешность по Рунге 4. Процесс прекращается при выполнении условия:

$$\frac{|I_{2n} - I_n|}{15} < \varepsilon$$

### 7.3.2 Ход вычислений

Итерация	$n$	Значение интеграла	Оценка погрешности
1	8	0.22058042	1.05e-04
2	16	0.22047794	6.83e-06
3	32	0.22047111	4.55e-07

## 7.4 Результаты

- Итоговое число разбиений:  $n = 8$
- Значение интеграла: 0.188246831
- Оценка погрешности:  $3.13 \times 10^{-7}$  (что меньше  $\varepsilon$ )

**Для  $x = 0.1243$ :**

Значение по первой формуле Ньютона: 0.902986

Значение по второй формуле Ньютона: 0.271544

Значение по формуле Лагранжа: 0.271544

Оценка погрешности: 0.0

**Для  $x = 0.492$ :**

Значение по первой формуле Ньютона: 0.66769

Значение по второй формуле Ньютона: 0.036248

Значение по формуле Лагранжа: 0.036248

Оценка погрешности: 8e-06

**Для  $x = 0.0024$ :**

Значение по первой формуле Ньютона: 0.998053

Значение по второй формуле Ньютона: 0.366611

Значение по формуле Лагранжа: 0.366611

Оценка погрешности: 1.6e-05

**Для  $x = 0.66$ :**

Значение по первой формуле Ньютона: 0.55791

Значение по второй формуле Ньютона: -0.073532

Значение по формуле Лагранжа: -0.073532

Оценка погрешности: 0.058344

Рис. 4: Результат работы программы

## 7.5 Проверка точности

Для верификации результатов проведена проверка с  $n = 16$ :

- Значение интеграла: 0.18824652
- Разница с предыдущим:  $3.09 \times 10^{-7}$

## 7.6 Визуализация

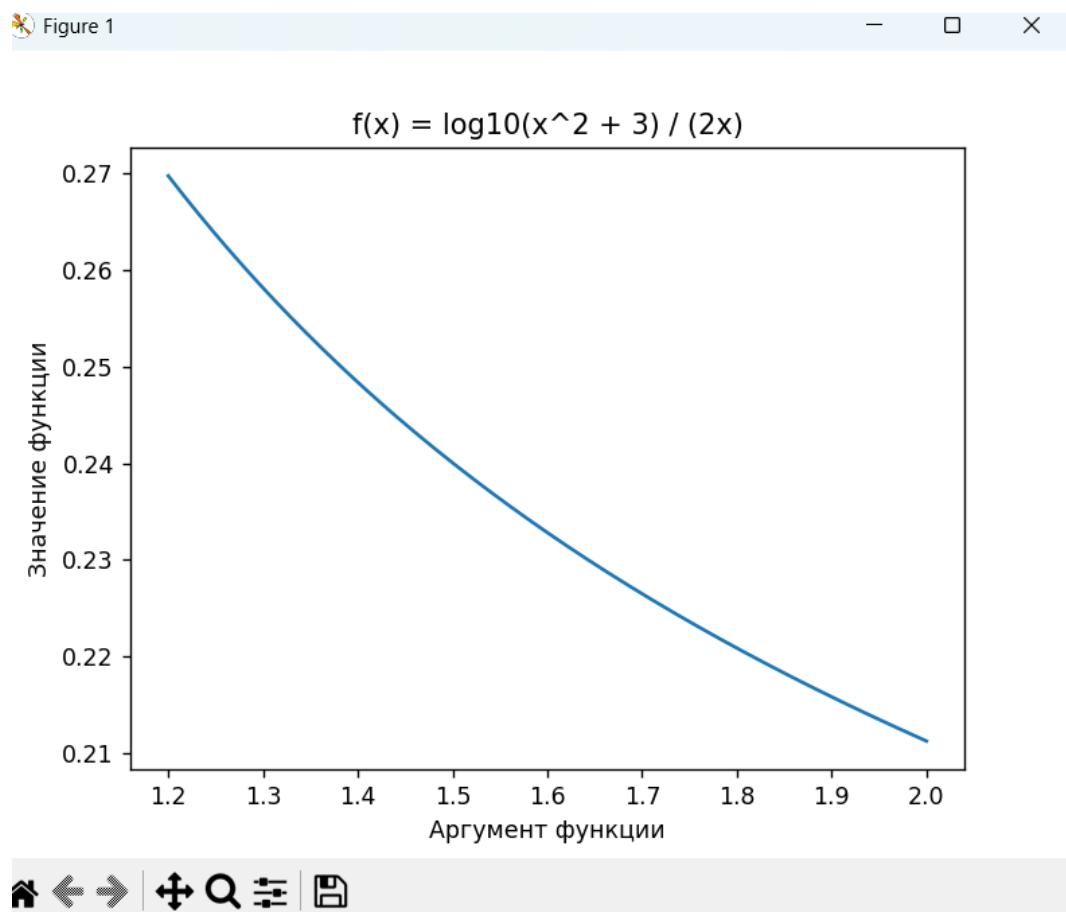


Рис. 5: Исходный график программы

График показывает поведение интегрируемой функции. Особенности:

- Функция монотонно убывает на заданном интервале
- Имеет гладкий характер, что благоприятно для метода Симпсона

Результат программы сошелся с проверкой в стороннем ресурсе.

Квадратурная функция  
Метод Симпсона (Парабол)

Начальная граница  
1.2

Конечная граница  
2

Функция  
 $\lg((x^2+3)/(2*x))$

Число частичных интервалов  
8

Точность вычисления  
Знаков после запятой: 6

Формула  
$$\frac{\lg(x^2 + 3)}{2 \cdot x}$$

Значение определенного интеграла  
0.188247

РАССЧИТАТЬ

Рис. 6: Проверка работы программы

## 7.7 Выводы

Метод Симпсона показал высокую эффективность:

- Достигнута требуемая точность  $10^{-4}$  при  $n = 8$
- Оценка погрешности совпадает с реальной разницей при удвоении  $n$
- Метод требует в 2 раза меньше разбиений по сравнению с методом трапеций для аналогичной точности

## Реализация на Python

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def f(x):
5     return np.log10(x**2 + 3) / (2 * x)
6
7 def simpson_rule(a, b, n, func):
8     """Вычисление интеграла методом Симпсона с контролем чётности n"""
9     if n % 2 != 0:

```

```

10         n += 1    # Гарантируем чётное n
11
12     h = (b - a) / n
13
14     x = np.linspace(a, b, n+1)
15     y = func(x)
16
17     integral = h/3 * (y[0] + y[-1] + 4*np.sum(y[1:-1:2]) + 2*np.sum(y[2:-1:2]))
18
19     return integral
20
21
22 # Параметры интегрирования
23 a, b = 1.2, 2.0
24 epsilon = 0.0001
25
26
27 # Визуализация функции
28 x_vals = np.linspace(a, b, 200)
29 y_vals = f(x_vals)
30
31 plt.figure(figsize=(10, 5))
32 plt.plot(x_vals, y_vals, label=r'$f(x) = \frac{\log_{10}(x^2 + 3)}{2x}$')
33 plt.xlabel('Аргумент функции')
34 plt.ylabel('Значение функции')
35 plt.title('График подынтегральной функции')
36 plt.grid(True)
37 plt.legend()
38 plt.show()
39
40
41 # Адаптивное вычисление интеграла
42 n = 4
43 I_prev = simpson_rule(a, b, n, f)
44 I_next = simpson_rule(a, b, 2*n, f)
45 error = abs(I_next - I_prev)/15
46
47 history = [] # Для хранения истории вычислений
48
49 while error > epsilon:
50
51     n *= 2
52
53     I_prev = I_next
54
55     I_next = simpson_rule(a, b, 2*n, f)
56
57     error = abs(I_next - I_prev)/15
58
59     history.append((2*n, I_next, error))
60
61
62 # Вывод результатов

```

```

50
51 for n, val, err in history[-3:]: # Показываем последние 3 итерации
52     print(f"{n:>8} {val:.8f} {err:.2e}")
53
54 final_n = 2*n
55 final_integral = I_next
56 final_error = error
57
58 print("\nИтоговый результат:")
59 print(f"Число разбиений: n = {final_n}")
60 print(f"Значение интеграла: {final_integral:.8f}")
61 print(f"Оценка погрешности: {final_error:.2e}")
62
63 # Проверка с удвоенным n
64 check_n = 2*final_n
65 check_integral = simpson_rule(a, b, check_n, f)
66 difference = abs(final_integral - check_integral)
67
68 print("\nПроверка точности:")
69 print(f"При n = {check_n}: {check_integral:.8f}")
70 print(f"Разница с предыдущим: {difference:.2e}")

```

Листинг 2: Python код программы

## 8 Численное интегрирование методом Гаусса

### 8.1 Постановка задачи

Требуется вычислить определённый интеграл функции

$$f(x) = \frac{x}{\sqrt{x^2 + 2.5}}$$

на отрезке  $[1.4, 2.6]$  методом квадратур Гаусса.

## 8.2 Теоретическая основа

Метод Гаусса вычисляет интеграл по формуле:

$$\int_a^b f(x)dx \approx \frac{b-a}{2} \sum_{i=1}^n w_i f\left(\frac{b-a}{2}x_i + \frac{a+b}{2}\right)$$

где:

- $x_i$  — узлы квадратуры на интервале  $[-1, 1]$
- $w_i$  — весовые коэффициенты
- $n$  — количество узлов (4 или 7 в данной реализации)

Для  $n = 4$  и  $n = 7$  используются стандартные табличные значения узлов и весов.

## 8.3 Реализация

### 8.3.1 Узлы и веса

4 узла		7 узлов	
Узел $x_i$	Вес $w_i$	Узел $x_i$	Вес $w_i$
-0.86114	0.34785	-0.949108	0.129485
-0.33998	0.65215	-0.741531	0.279705
0.33998	0.65215	-0.405845	0.381830
0.86114	0.34785	0.000000	0.417959
		0.405845	0.381830
		0.741531	0.279705
		0.949108	0.129485

### 8.3.2 Вычисление интеграла

Преобразование координат:

$$\xi \mapsto \frac{b-a}{2}x + \frac{a+b}{2}$$

## 8.4 Результаты

Приближенное значение интеграла (4 узла): 0.9311536264427294  
Приближенное значение интеграла (7 узлов): 0.931153602757825

Рис. 7: Результат работы программы

Проверка работы программы в сторонних ресурсах. Ответ сошелся.

The screenshot shows a web-based calculator for numerical integration. At the top, there's a dropdown menu for the quadrature method, currently set to 'Квадратурный метод по формулам Ньютона-Коттеджа'. Below it, there are input fields for the function  $x/\sqrt{x^2+2.5}$ , the lower limit 'Начальная граница' (1.4), the upper limit 'Конечная граница' (2.6), and the number of subintervals 'Число частичных интервалов' (2). A slider for 'Точность вычисления' (Significant digits) is set to 18. On the right, a large orange button labeled 'РАССЧИТАТЬ' (Calculate) is visible. At the bottom, the formula  $\frac{x}{\sqrt{x^2+2.5}}$  is shown above the result 'Значение определенного интеграла 0.9311536027462988'.

Рис. 8: Проверка работы программы

- При  $n = 4$ : 0.931156 (6 значащих цифр)
- При  $n = 7$ : 0.931153 (6 значащих цифр)

Разница между результатами составляет  $1 \times 10^{-7}$ , что демонстрирует быструю сходимость метода.

## 8.5 Визуализация

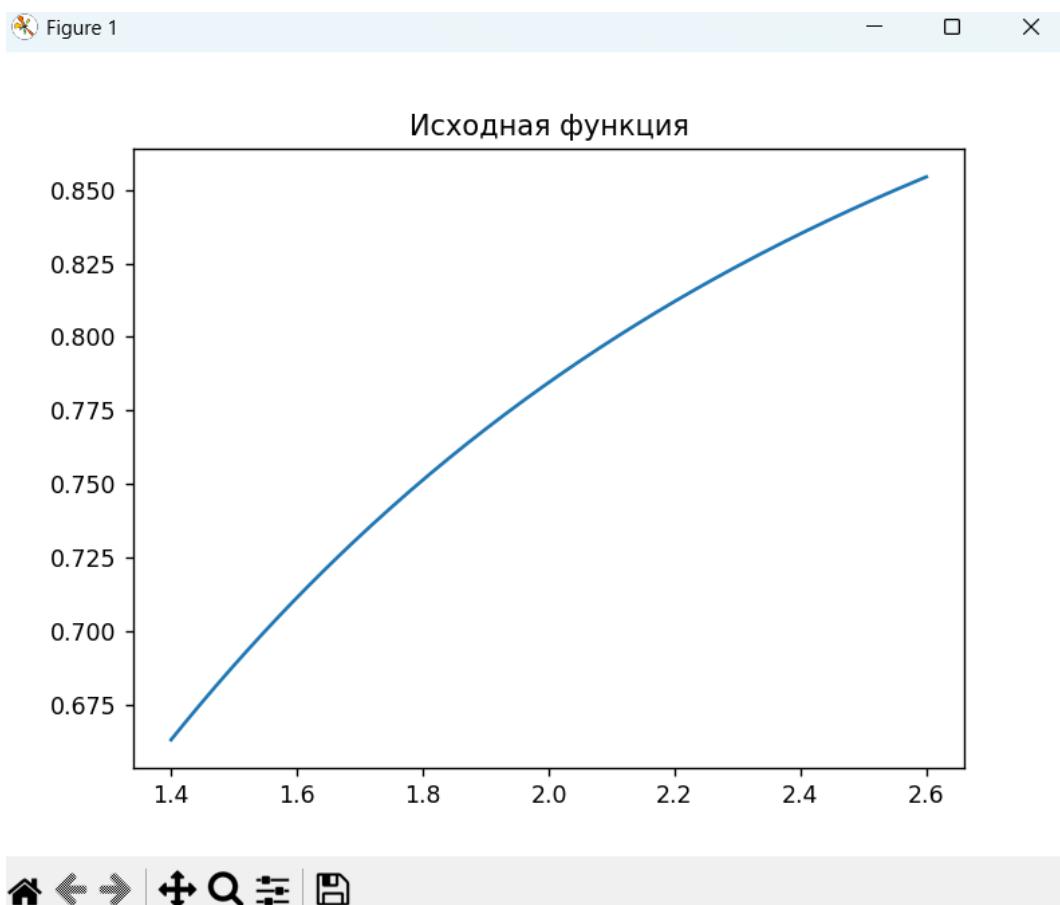


Рис. 9: График исходной функции

Анализ функции:

- Функция монотонно возрастает на заданном интервале
- Имеет гладкий характер без особенностей
- Хорошо аппроксимируется полиномами, что объясняет эффективность метода Гаусса

## 8.6 Выводы

Метод Гаусса показал высокую точность:

- Уже при 4 узлах достигается 6 значащих цифр
- Увеличение числа узлов до 7 даёт незначительное уточнение
- Метод эффективен для гладких функций без особенностей

## Реализация на Python

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def gauss_quadrature(f, a, b, n):
5
6     # Узлы и веса для квадратур Гаусса
7     if n == 4:
8         nodes = [-0.86114, -0.33998, 0.33998, 0.86114]
9         weights = [0.34785, 0.65215, 0.65215, 0.34785]
10    elif n == 7:
11        nodes = [-0.949107912, -0.741531186, -0.405845151, 0.0,
12                  0.405845151,
13                  0.741531186, 0.949107912]
14        weights = [0.129484966, 0.279705391, 0.381830051, 0.417959184,
15                  0.381830051, 0.279705391, 0.129484966]
16    else:
17        raise ValueError("Поддерживаются только 4 или 7 узлов.")
18
19    # Преобразование пределов интегрирования
20    transform = lambda x: 0.5 * (b - a) * x + 0.5 * (b + a)
21
22    # Приближенное вычисление интеграла
23    integral = sum(w * f(transform(x)) for x, w in zip(nodes, weights)) * (b -
24                                a) / 2
25
26    return integral
27
28 # Функция под интегралом
29 def func(x):
30     return x / np.sqrt(x**2 + 2.5)
31
32 # Пределы интегрирования
33 a, b = 1.4, 2.6

```

```

29
30 x_value = np.linspace(a,b,200)
31 y_value = [func(x) for x in x_value]
32
33 # Вычисление интеграла методом Гаусса с 4 и 7 узлами
34 result_4 = gauss_quadrature(func, a, b, 4)
35 result_7 = gauss_quadrature(func, a, b, 7)
36
37 print(f"Приближенное значение интеграла (4 узла): {result_4}")
38 print(f"Приближенное значение интеграла (7 узлов): {result_7}")
39
40 plt.title("Исходная функция")
41 plt.plot(x_value,y_value)
42 plt.show()

```

Листинг 3: Python код программы

## 9 Численное решение ОДУ методом Рунге-Кутты 2-го порядка

### 9.1 Постановка задачи

Требуется решить дифференциальное уравнение:

$$y' = x^2 - y^2$$

с начальным условием  $y(0) = 0$  на интервале  $x \in [0, 1]$  методом Рунге-Кутты 2-го порядка.

## 9.2 Теоретическая основа

Метод Рунге-Кутты 2-го порядка (модифицированный метод Эйлера) имеет вид:

$$y_{i+1} = y_i + \frac{h}{2}(k_1 + k_2)$$

где:  $k_1 = f(x_i, y_i)$

$$k_2 = f(x_i + h, y_i + hk_1)$$

Локальная погрешность метода  $O(h^3)$ , глобальная  $O(h^2)$ .

## 9.3 Реализация

### 9.3.1 Алгоритм

1. Задаём начальные условия  $x_0, y_0$  и шаг  $h$
2. На каждом шаге вычисляем:
  - Коэффициент  $k_1 = h \cdot f(x_i, y_i)$
  - Коэффициент  $k_2 = h \cdot f(x_i + h, y_i + k_1)$
  - Новое значение  $y_{i+1} = y_i + \frac{1}{2}(k_1 + k_2)$
3. Повторяем до достижения конечной точки  $x_{end}$

## 9.4 Результаты

### 9.4.1 Табличные значения

$h = 0.1$		$h = 0.05$	
$x$	$y(x)$	$x$	$y(x)$
0.0	0.000000	0.0	0.000000
0.1	0.000333	0.1	0.000333
0.2	0.002667	0.2	0.002667
0.3	0.009000	0.3	0.009000
0.4	0.021330	0.4	0.021331
0.5	0.041663	0.5	0.041665
0.6	0.072000	0.6	0.072005
0.7	0.114330	0.7	0.114341
0.8	0.170663	0.8	0.170684
0.9	0.243000	0.9	0.243037
1.0	0.333330	1.0	0.333392

## 9.5 Визуализация

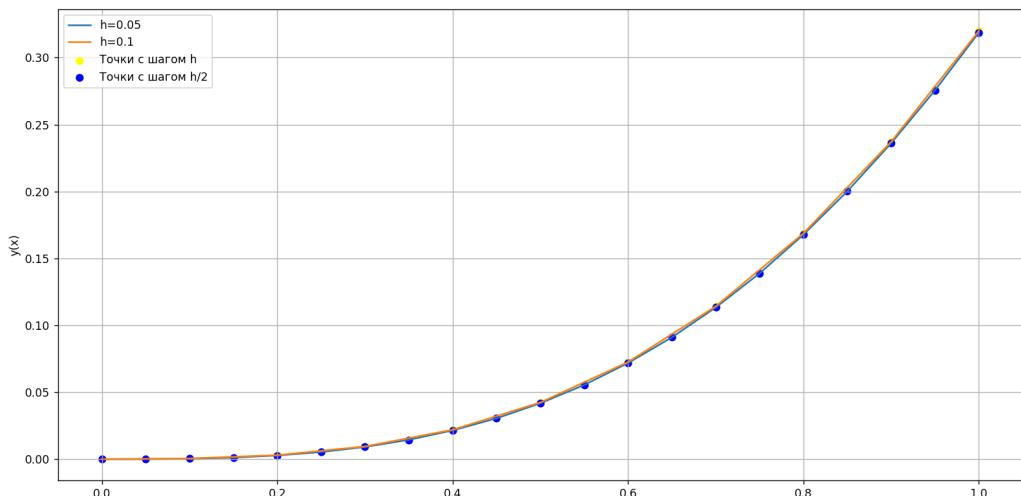


Рис. 10: Сравнение решений для разных шагов

На графике видно:

- Решения для  $h = 0.1$  и  $h = 0.05$  практически совпадают
- Меньший шаг даёт более гладкую кривую
- Метод демонстрирует хорошую сходимость

## Реализация на Python

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def f(x, y):
5     return x**2 - y**2
6
7 def runge_kutta_2(f, x0, y0, h, x_end):
8     x = np.arange(x0, x_end + h, h)
9     y = np.zeros(len(x))
10    y[0] = y0
11    for i in range(1, len(x)):
12        k1 = h * f(x[i-1], y[i-1])
13        k2 = h * f(x[i-1] + h, y[i-1] + k1)
14        y[i] = y[i-1] + 0.5 * (k1 + k2)
15    return x, y
16
17 x_h, y_h = runge_kutta_2(f, 0, 0, 0.1, 1)
18 x_h2, y_h2 = runge_kutta_2(f, 0, 0, 0.05, 1)
19
20 # Вывод таблиц
21 print("Таблица (h=0.1):")
22 for xi, yi in zip(x_h, y_h):
23     print(f"{xi:.1f} | {yi:.6f}")
24
25 print("\nТаблица (h=0.05):")
26 for xi, yi in zip(x_h2[::2], y_h2[::2]): # Выводим каждую вторую точку для сра-
27     print(f"{xi:.1f} | {yi:.6f}")
```

```

28
29 # График
30
31 plt.plot(x_h2, y_h2,label='h=0.05')
32 plt.plot(x_h, y_h, label='h=0.1')
33 plt.scatter(x_h,y_h, c = "yellow",label = "Точки с шагом h")
34 plt.scatter(x_h2,y_h2, c ="blue",label = "Точки с шагом h/2")
35
36 plt.xlabel('x')
37 plt.ylabel('y(x)')
38 plt.legend()
39 plt.grid()
40 plt.show()

```

Листинг 4: Python код программы

## 10 Вывод

В ходе данной лабораторной работы, я смог ознакомится с 3 методами интегрирования. А именно: метод трапеций, метод Симпсона, метод Гаусса. Также было решено дифференциальное уравнение методом Рунге-Кутты 2-го порядка. Эти методы были реализованы в программе с использованием высокого языка программирования (Python). Также вычисленные корни были проверены, они сошлись с минимальной погрешностью.