«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Факультет программной инженерии и компьютерной техники  
Направление подготовки 09.03.04 Программная инженерия  
Дисциплина «Вычислительная математика»

Отчет

По лабораторной работе №3

Вариант 4

Студент:

*Ильин Н. С.*

*Р3210*

Преподаватель:

*Наумова Н. А.*

Санкт-Петербург, 2025 г.

**Оглавление**

[**Цель работы**: 3](#_Toc193643297)

[**2. Программная реализация** 7](#_Toc193643298)

[**Блок схемы** 7](#_Toc193643299)

[**Листинг программы** 12](#_Toc193643300)

[**Примеры и результаты работы программы** 14](#_Toc193643301)

[**Выводы**: 16](#_Toc193643302)

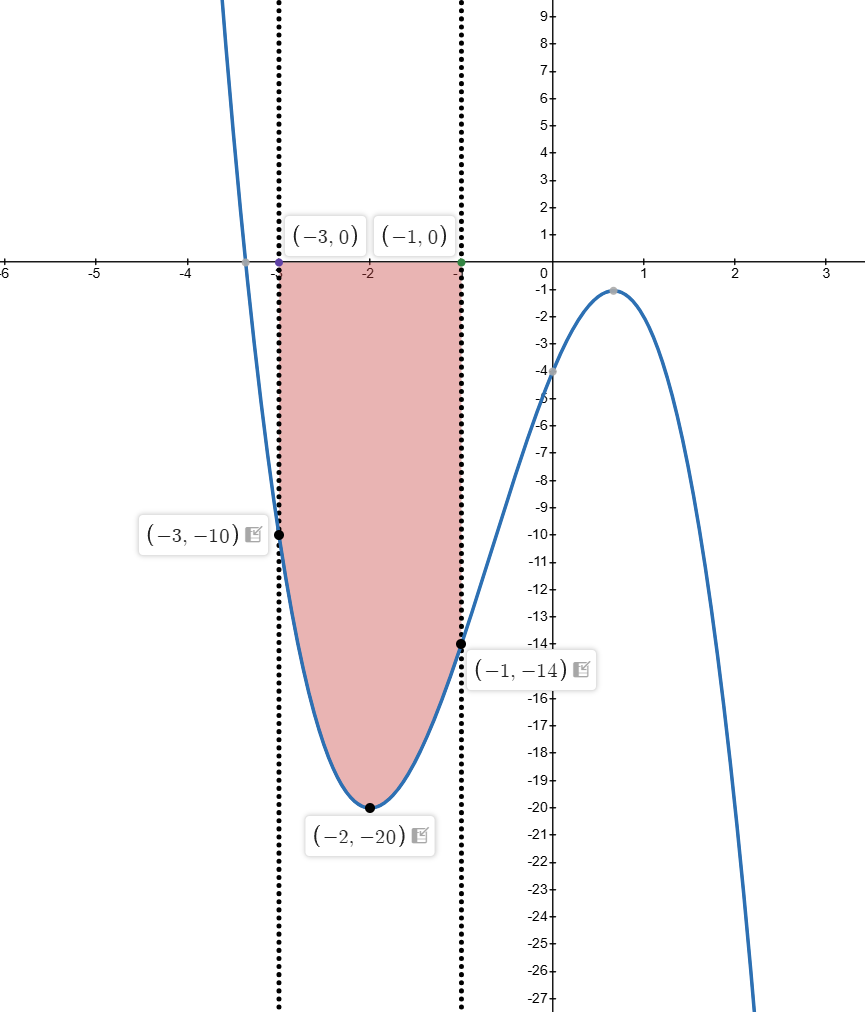
# **Цель работы**:

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

**1 Вычислительная реализация задачи:**

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

|  |
| --- |
|  |



1. **Вычислить интеграл по формуле Ньютона–Котеса** при :

**3.**

1. **Вычислить интеграл по формулам средних прямоугольников, трапеций и Симпсона** при :

* **Метод средних прямоугольников**:

* **Метод трапеций**:
* **Метод Симпсона**:

1. **Сравнить результаты с точным значением интеграла:**

Точное значение интеграла на интервале вычислено как

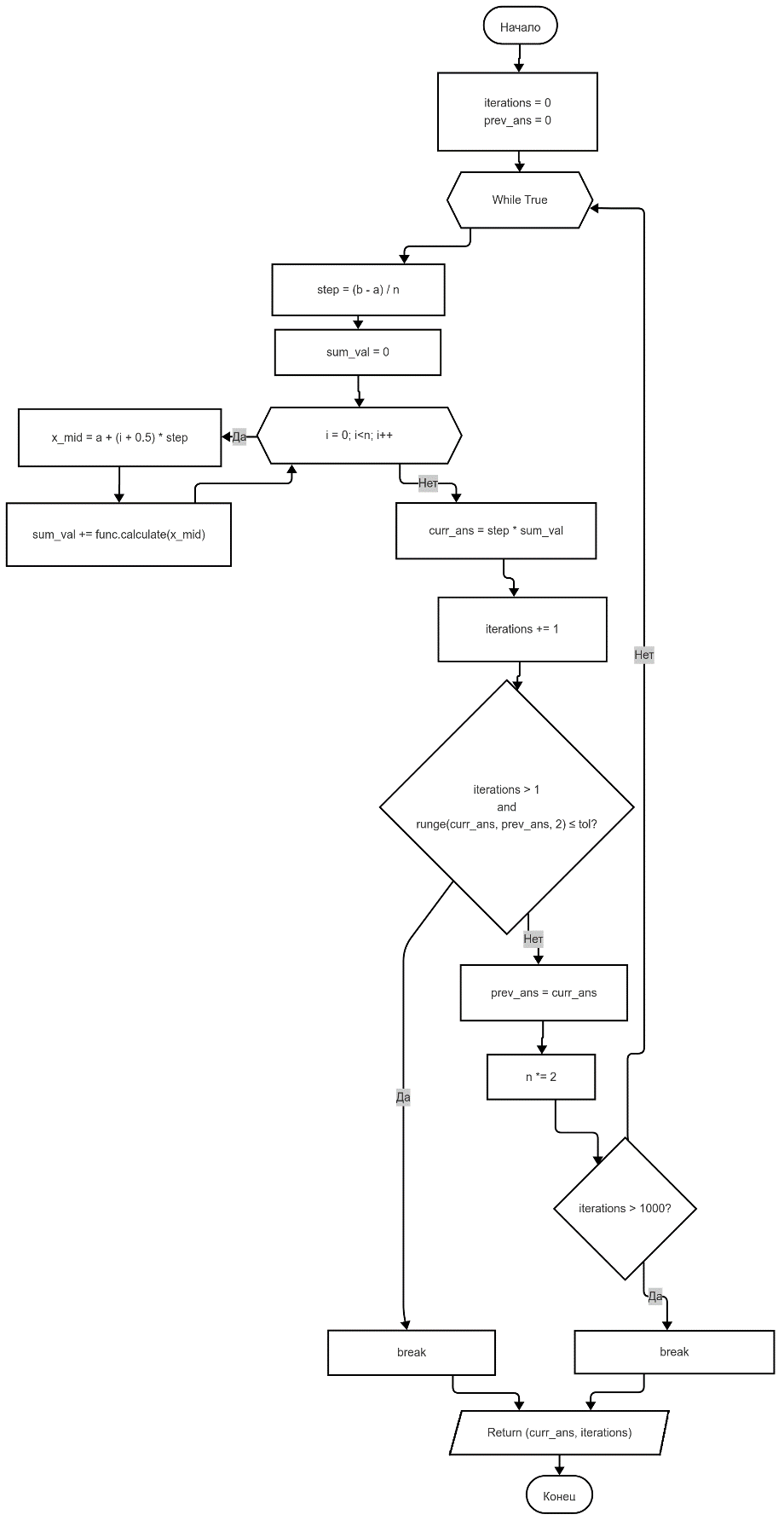
1. Для метода **Ньютона–Котеса** при : , **значения совпадают**.
2. Для метода **средних прямоугольников** при : .
3. Для метода **трапеций** при :
4. Для метода **Симпсона** при : , **значения совпадают**.
5. **Определить относительную погрешность вычислений для каждого метода.**
6. Для метода **Ньютона–Котеса**: **погрешности нет.**
7. Для метода **средних прямоугольников**:
8. Для метода **трапеций**:
9. Для метода **Симпсона**: **погрешности нет.**

Как видно из результатов, все методы дали относительно малую погрешность, особенно при использовании формулы Ньютона–Котеса и Симпсона. Наилучший результат был получен при использовании формулы Ньютона–Котеса с и формулы Симпсона с , при которых значения интеграла полностью совпали.

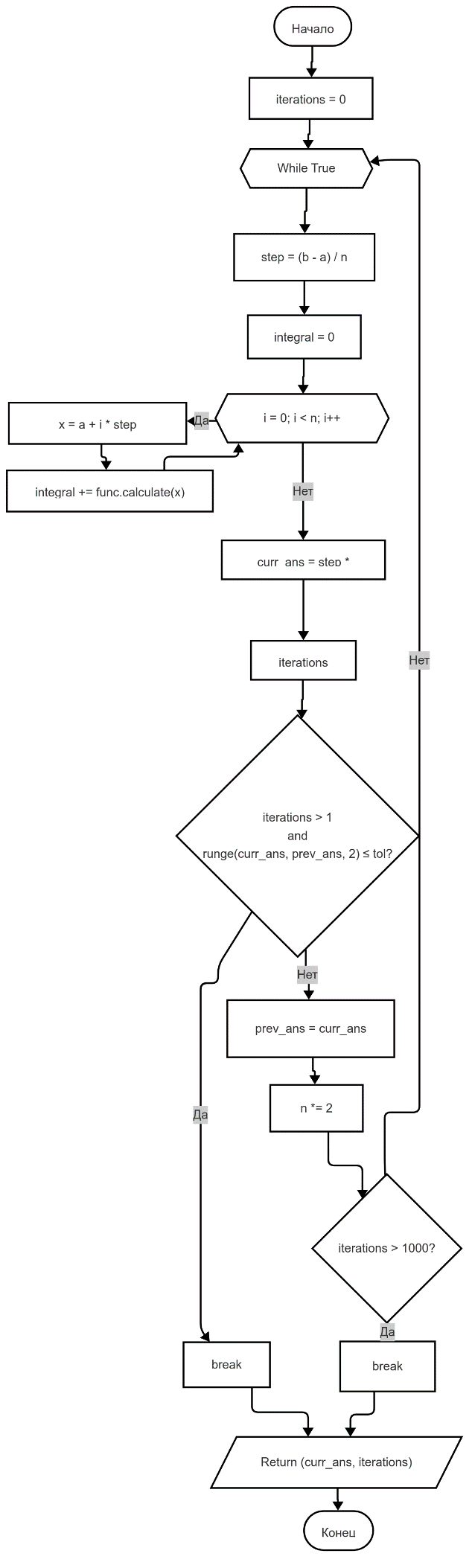
# **2. Программная реализация**

# **Блок схемы**

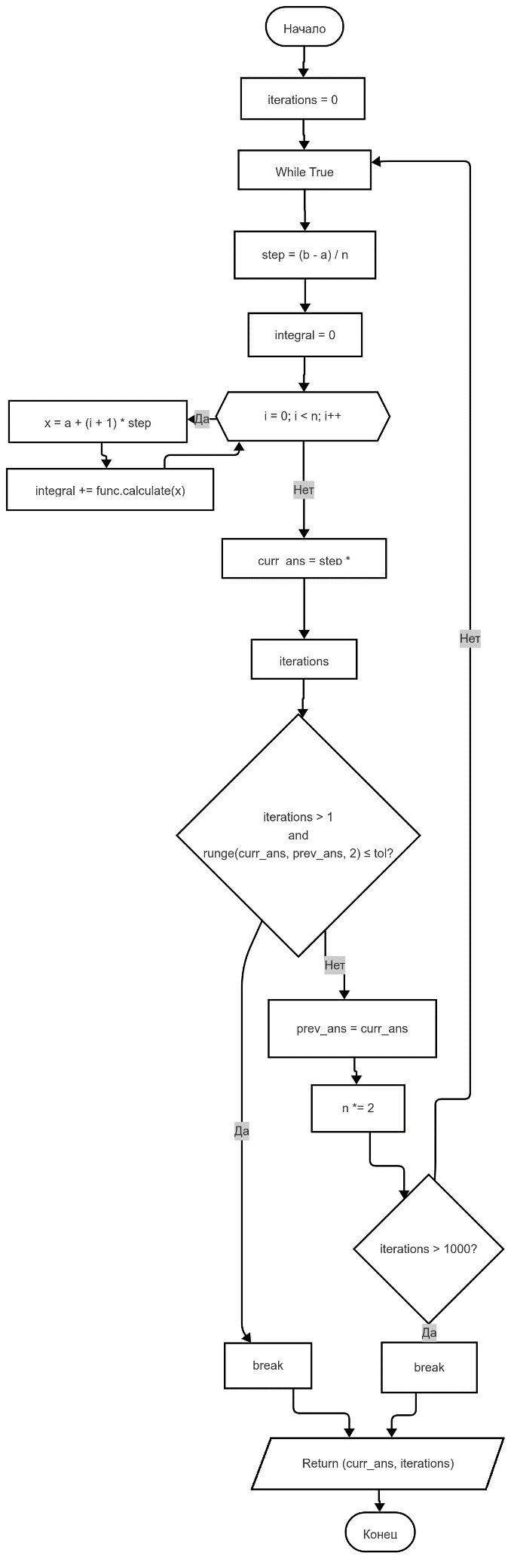
Метод средних прямоугольников



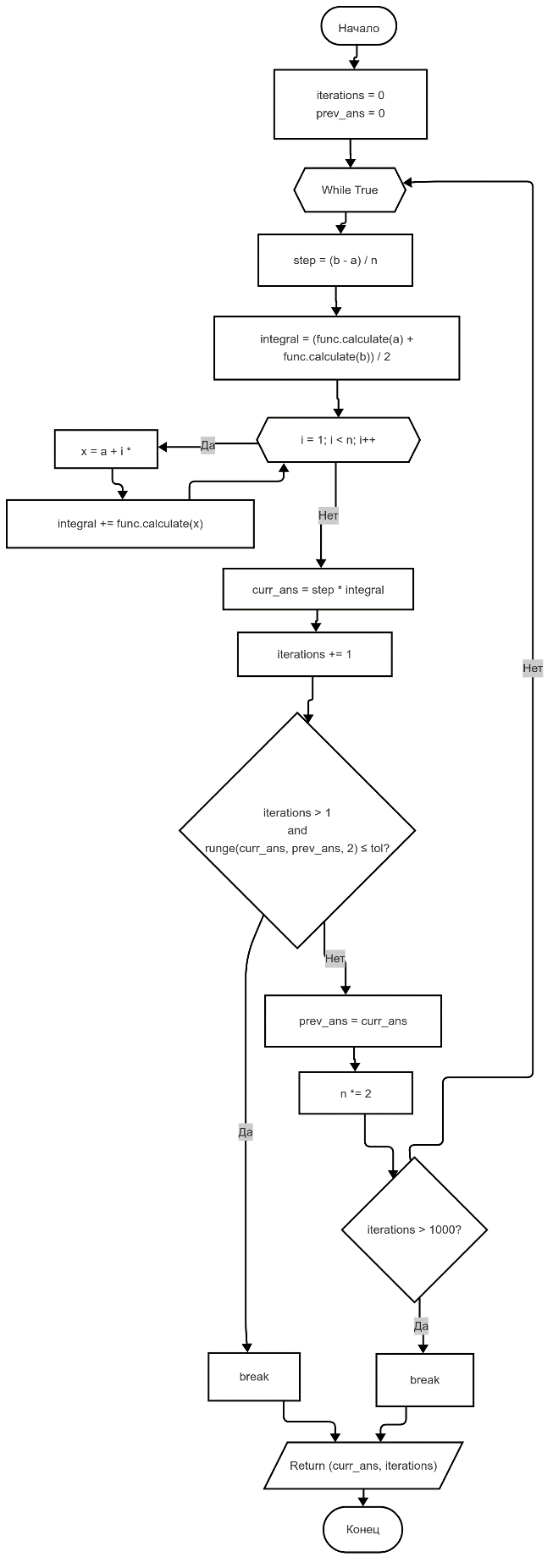
Метод левых прямоугольников



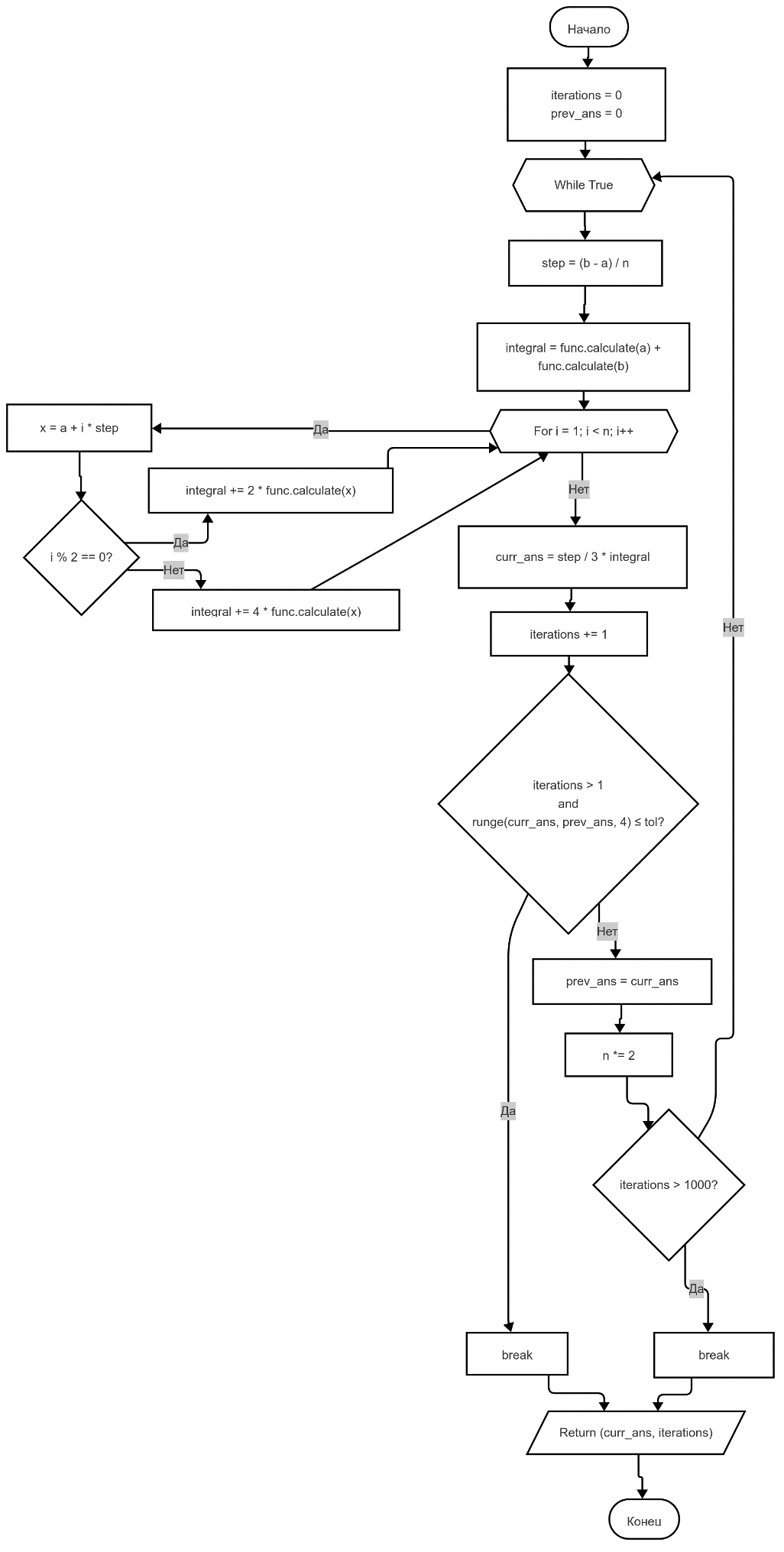
Метод правых прямоугольников



Метод трапеций



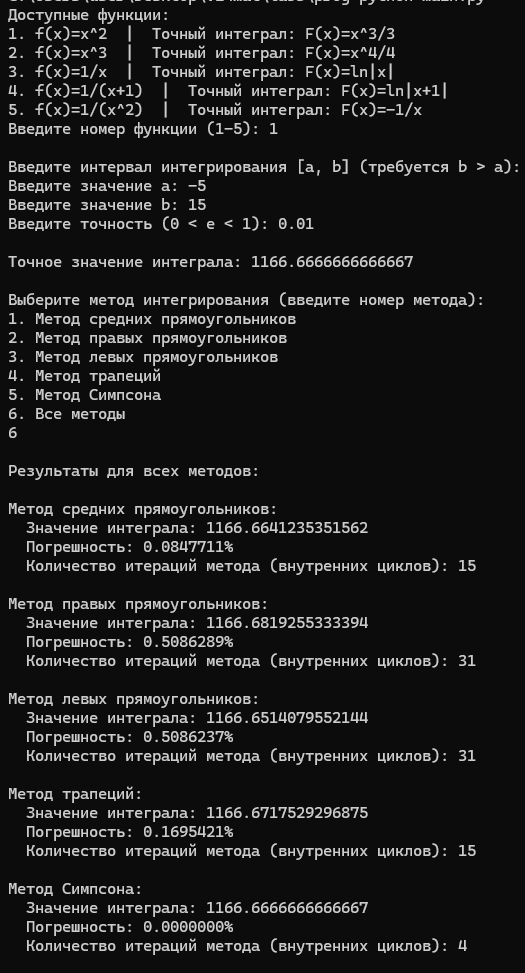
Метод Симпсона

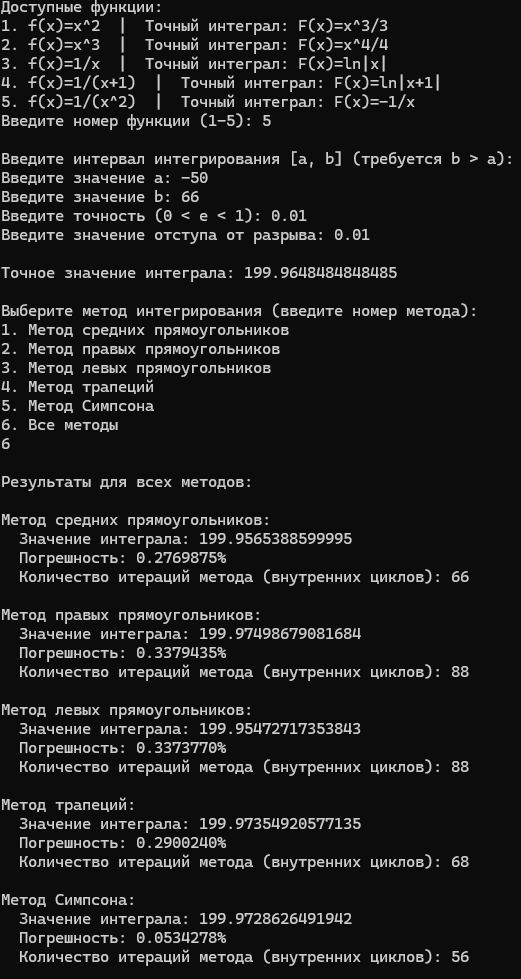


# **Листинг программы**

|  |
| --- |
| def runge(curr\_ans, prev\_ans, order):  return abs(curr\_ans - prev\_ans) / (order \*\* 2 - 1)  def mid\_rectangle\_method(func, a, b, n, tol):  iterations = 0  prev\_ans = 0  while True:  step = (b - a) / n  sum\_val = 0  for i in range(n):  x\_mid = a + (i + 0.5) \* step  sum\_val += func.calculate(x\_mid)  curr\_ans = step \* sum\_val  iterations += 1  if iterations > 1 and runge(curr\_ans, prev\_ans, 2) <= tol:  break  prev\_ans = curr\_ans  n \*= 2  if iterations > 1000:  break  return curr\_ans, iterations  def right\_rectangle\_method(func, a, b, n, tol):  iterations = 0  prev\_ans = 0  while True:  step = (b - a) / n  integral = sum(func.calculate(a + (i + 1) \* step) for i in range(n))  curr\_ans = step \* integral  iterations += 1  if iterations > 1 and runge(curr\_ans, prev\_ans, 2) <= tol:  break  prev\_ans = curr\_ans  n \*= 2  if iterations > 1000:  break  return curr\_ans, iterations  def left\_rectangle\_method(func, a, b, n, tol):  iterations = 0  prev\_ans = 0  while True:  step = (b - a) / n  integral = sum(func.calculate(a + i \* step) for i in range(n))  curr\_ans = step \* integral  iterations += 1  if iterations > 1 and runge(curr\_ans, prev\_ans, 2) <= tol:  break  prev\_ans = curr\_ans  n \*= 2  if iterations > 1000:  break  return curr\_ans, iterations  def trapezoidal\_method(func, a, b, n, tol):  iterations = 0  prev\_ans = 0  while True:  step = (b - a) / n  integral = (func.calculate(a) + func.calculate(b)) / 2  for i in range(1, n):  x = a + i \* step  integral += func.calculate(x)  curr\_ans = step \* integral  iterations += 1  if iterations > 1 and runge(curr\_ans, prev\_ans, 2) <= tol:  break  prev\_ans = curr\_ans  n \*= 2  if iterations > 1000:  break  return curr\_ans, iterations  def simpson\_method(func, a, b, n, tol):  iterations = 0  prev\_ans = 0  while True:  step = (b - a) / n  integral = func.calculate(a) + func.calculate(b)  for i in range(1, n):  x = a + i \* step  if i % 2 == 0:  integral += 2 \* func.calculate(x)  else:  integral += 4 \* func.calculate(x)  curr\_ans = step / 3 \* integral  iterations += 1  if iterations > 1 and runge(curr\_ans, prev\_ans, 4) <= tol:  break  prev\_ans = curr\_ans  n \*= 2  if iterations > 1000:  break  return curr\_ans, iterations  def calculate\_exact\_integral(integ, a, b):  return integ.calculate(b) - integ.calculate(a)  def calculate\_integral(method, func, a, b, tol, n):  total\_method\_iterations = 0  prev\_ans = 0  while True:  curr\_ans, inner\_iters = method(func, a, b, n, tol)  total\_method\_iterations += inner\_iters  if n > 4 and abs(curr\_ans - prev\_ans) / 3 <= tol:  break  prev\_ans = curr\_ans  n \*= 2  print(n)  if n > 1000:  break  return curr\_ans, total\_method\_iterations |

## **Примеры и результаты работы программы**





Github: <https://github.com/MrTheFall/computational_math/tree/main/lab3>

# **Выводы**:

В рамках лабораторной работы были исследованы численные методы интегрирования с использованием языка Python. В процессе работы были изучены различные подходы к вычислению определенных интегралов, включая методы прямоугольников (левых, правых и средних), метод трапеций, метод Ньютона-Котеса и метод Симпсона.

Была разработана программа, которая позволяет выбрать одну из предложенных функций, задать пределы интегрирования, точность и начальное значение числа разбиений интервала. После реализации всех рассмотренных методов вычисления интегралов было установлено, что метод Симпсона является наиболее точным и быстрым.

В ходе выполнения работы были проведены расчеты интегралов различными методами, а также выполнено сравнение полученных результатов с точными значениями интегралов.

Кроме того, была решена дополнительная задача, связанная с исследованием сходимости несобственных интегралов второго рода и их вычислением с использованием рассмотренных численных методов в случаях, когда подынтегральная функция имеет бесконечный разрыв в точке a, точке b или на отрезке интегрирования.