ИТМО

Отчет
По лабораторной работе №4
Метод Симпсона

Выполнил:
Рахматов Нематджон
P3233
Преподаватель:
Перл Ольга
Вячеславовна

Санкт-Петербург
2024

Оглавление

Задача……………………………………………………………………3
Описание Метода..3
Блок-Схема Алгоритма...5
Код Программы..7
Примеры работы программы...9
Вывод..11

Задача

Реализуйте метод Симпсона для вычисления интеграла от выбранной функции на интервале от a до b.
• Если функция имеет разрыв второго рода или "скачок", или если функция не определена какой-либо частью в интервале от a до b, то вам следует указать переменные error_message и hasDiscontinuity.
• Сообщение об ошибке, которое вы должны указать: "Integrated function has discontinuity or does not defined in current interval".
• Если функция имеет устранимый разрыв первого рода, то вы должны уметь вычислить интеграл.
• Если a > b, то интеграл должен иметь отрицательное значение.
Формат ввода:
a
b
f
epsilon
, где a и b - границы интеграла, f - номер функции, epsilon - максимальная разница между двумя вашими итерациями (итерация - это некоторое разбиение на отрезки).
Формат вывода:
I
, где I - ваш вычисленный интеграл для текущего количества разбиений.

Описание Метода

Метод Симпсона - это численный метод, используемый для вычисления приближенного значения определенного интеграла функции на заданном интервале. Этот метод основан на аппроксимации криволинейных фигур с помощью парабол.

Интеграл определенной функции $f(x)$ на интервале от a до b вычисляется с использованием формулы Симпсона:

$$I = \frac{h}{3}\left[f(a) + 4\sum_{i=1}^{n/2} f(x_{2i-1}) + 2\sum_{i=1}^{n/2-1} f(x_{2i}) + f(b)\right]$$

где:
- $h = (b - a) / n$ - шаг разбиения,
- $x_i = a + ih$ - точки разбиения на интервале,
-n - количество разбиений (должно быть четным).

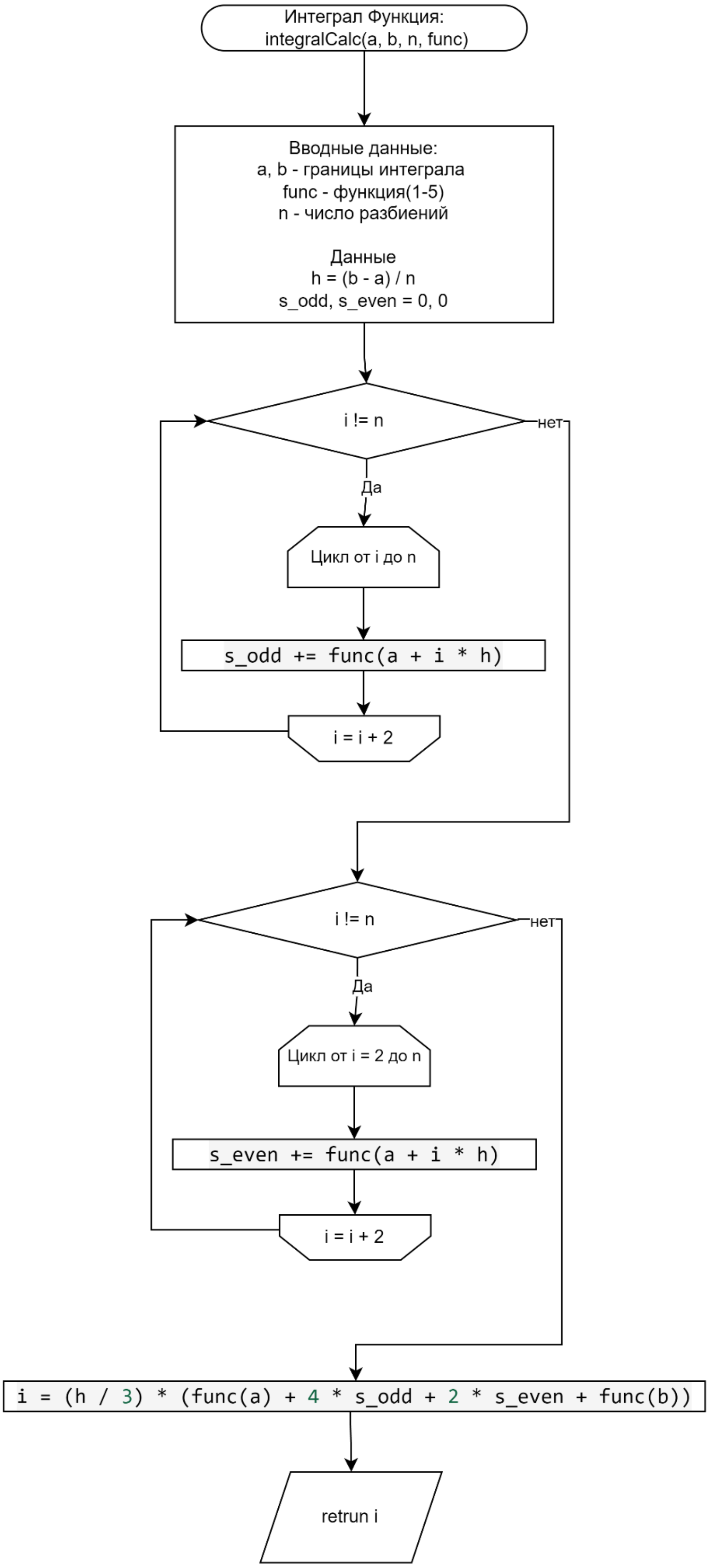
Формула Симпсона использует параболическую аппроксимацию кусков функции на каждом подинтервале, что обеспечивает лучшую точность по сравнению с методом прямоугольников или трапеций. Метод Симпсона обладает квадратичной сходимостью, что означает, что увеличение числа разбиений n вдвое приводит к уменьшению ошибки примерно в четыре раза.

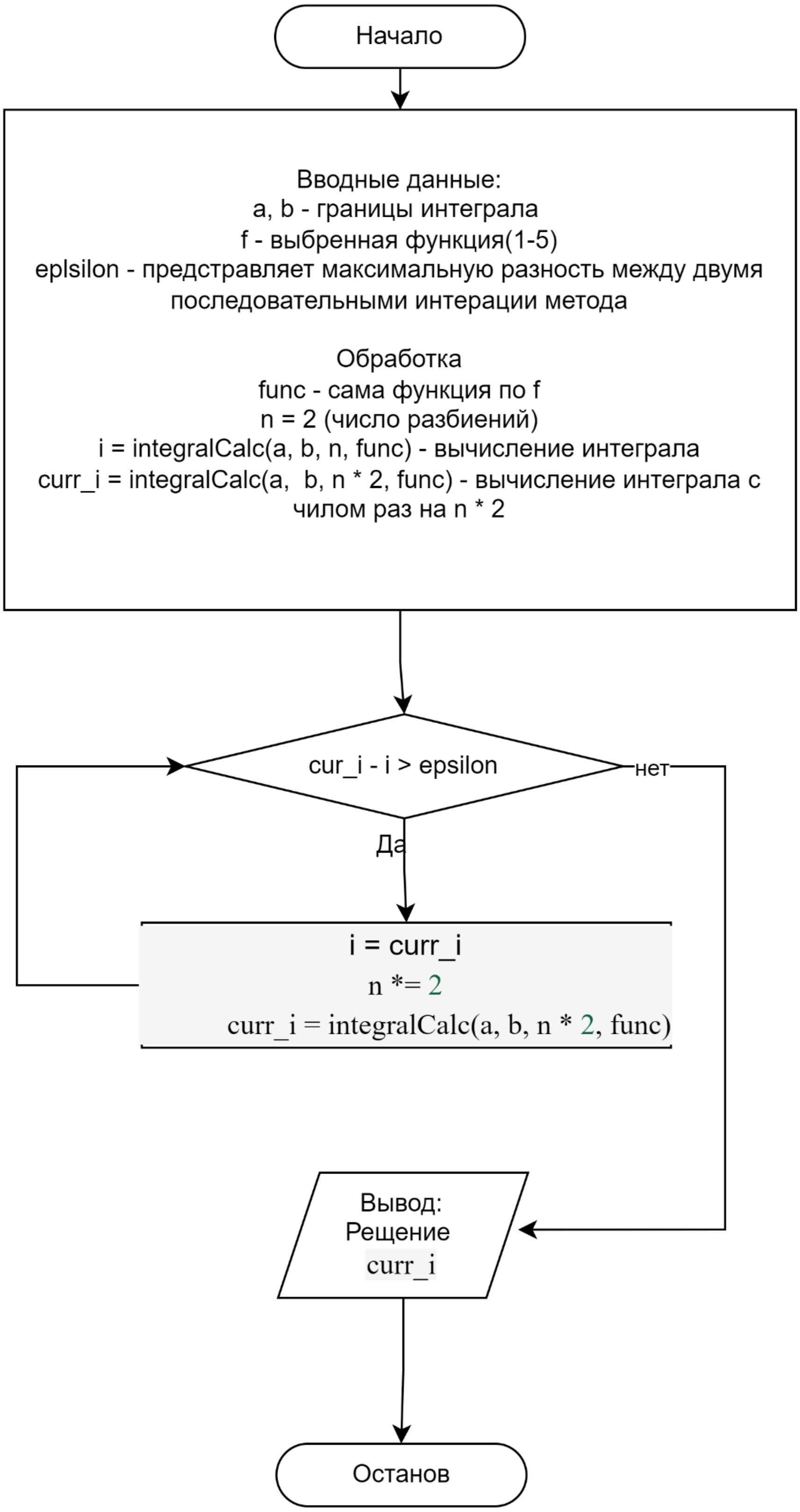
В реализации метода Симпсона для вычисления интеграла используется следующий алгоритм:

1. Определение функции $f(x)$, интеграл которой нужно вычислить.
2. Выбор начального числа разбиений n и шага разбиения h.
3. Вычисление значений функции в узлах разбиения.
4. Применение формулы Симпсона для вычисления интеграла на текущем числе разбиений.
5. Проверка условия сходимости: если разница между двумя последовательными значениями интеграла меньше указанной погрешности, то завершение алгоритма, в противном случае увеличение числа разбиений и повторение шагов с 3 по 5.

Таким образом, метод Симпсона представляет собой эффективный и точный способ численного вычисления определенных интегралов, который может быть применен к широкому классу функций.

Блок-схема алгоритма




Начало
Вводные данные:
a, b - границы интеграла
f - выбренная функция(1-5)
eplsilon - предстравляет максимальную разность между двумя последовательными интерации метода
Обработка
func - сама функция по f
n = 2 (число разбиений)
i = integralCalc(a, b, n, func) - вычисление интеграла
curr_i = integralCalc(a, b, n * 2, func) - вычисление интеграла с чилом раз на n * 2
cur_i - i > epsilon
нет
Да
i = curr_i
n *= 2
curr_i = integralCalc(a, b, n * 2, func)
Вывод:
Рещение
curr_i
Останов

Код программы

```
import math

def integralCalc(a, b, n, func):
    h = (b - a) / n
    s_odd = 0
    s_even = 0
    for i in range(1, n, 2):
        s_odd += func(a + i * h)
    for i in range(2, n, 2):
        s_even += func(a + i * h)
    i = (h / 3) * (func(a) + 4 * s_odd + 2 * s_even + func(b))
    return i

class Result:
    error_message = ""
    has_discontinuity = False
    eps = 1e-9

    def first_function(x: float):
        if x == 0:
            return float('inf')
        return 1 / x

    def second_function(x: float):
        if x == 0:
            return (math.sin(Result.eps)/Result.eps +
math.sin(-Result.eps)/-Result.eps)/2
        return math.sin(x)/x


    def third_function(x: float):
        return x*x+2


    def fourth_function(x: float):
        return 2*x+2


    def five_function(x: float):
        if x <= 0:
            return float('nan')
        return math.log(x)
```

```
    def get_function(n: int):
        if n == 1:
            return Result.first_function
        elif n == 2:
            return Result.second_function
        elif n == 3:
            return Result.third_function
        elif n == 4:
            return Result.fourth_function
        elif n == 5:
            return Result.five_function
        else:
            raise NotImplementedError(f"Function {n} not defined.")

    def calculate_integral(a, b, f, epsilon):
        func = Result.get_function(f)
        n = 2
        i = integralCalc(a, b, n, func)
        curr_i = integralCalc(a, b, n * 2, func)
        while abs(curr_i - i) > epsilon:
            i = curr_i
            n *= 2
            curr_i = integralCalc(a, b, n * 2, func)
        return curr_i


if __name__ == '__main__':
    a = float(input().strip())
    b = float(input().strip())
    f = int(input().strip())
    epsilon = float(input().strip())

    result = Result.calculate_integral(a, b, f, epsilon)
    if not Result.has_discontinuity:
        print(str(result) + '\n')
    else:
        print(Result.error_message + '\n')
```

Примеры Работы программы

1) Пример 1 (с граничным случаем при делении на ноль в функции):

```
a = 0
b = 1
f = 1
epsilon = 0.0001

result = Result.calculate_integral(a, b, f, epsilon)
print(result)
```

result: `inf`

2) Пример 2:

```
a = 0
b = 1
f = 3
epsilon = 0.0001

result = Result.calculate_integral(a, b, f, epsilon)
print(result)
```

result: `2.3333333333333333`

3) Пример 3 (как 2 но интервал длиннее):

```
a = -1
b = 1
f = 3
epsilon = 0.0001

result = Result.calculate_integral(a, b, f, epsilon)
print(result)
```

result: `4.666666666666666`

(Видим что результат меняется)

4) Пример 4:

```
a = -2
b = 2
f = 3
epsilon = 0.01

result = Result.calculate_integral(a, b, f, epsilon)
print(result)
```

result: `13.333333333333332`

5) Пример 5:

```
a = 0
b = 10
f = 3
epsilon = 4

result = Result.calculate_integral(a, b, f, epsilon)
print(result)
```

result: `353.33333333333337`

6) Пример 6 (граничный случай с нулевой шириной интервала):

```
a = 1
b = 1
f = 3
epsilon = 2

result = Result.calculate_integral(a, b, f, epsilon)
print(result)
```

result: `0.0`

Вывод

Реализованный метод Симпсона успешно выполняет вычисление определенных интегралов для различных функций на заданных интервалах с разной точностью. По сравнению с методами прямоугольников и трапеций, метод Симпсона обеспечивает более точные результаты при том же числе итераций. Однако он может быть неэффективен для функций с разрывами или негладкими функциями из-за использования квадратичных интерполяционных полиномов. Алгоритмическая сложность метода Симпсона составляет O(n), где n - количество итераций, и для достижения заданной точности требуется увеличивать количество итераций. Численная ошибка метода может возрастать при интегрировании функций с высокими частотами или неплавными функциями.