Университет ИТМО

Факультет программной инженерии и компьютерной техники

Дисциплина «Вычислительная математика»

Отчет

По лабораторной работе №4

Метод средних прямоугольников.

Выполнил:

Терновский И.Е

Преподаватель:

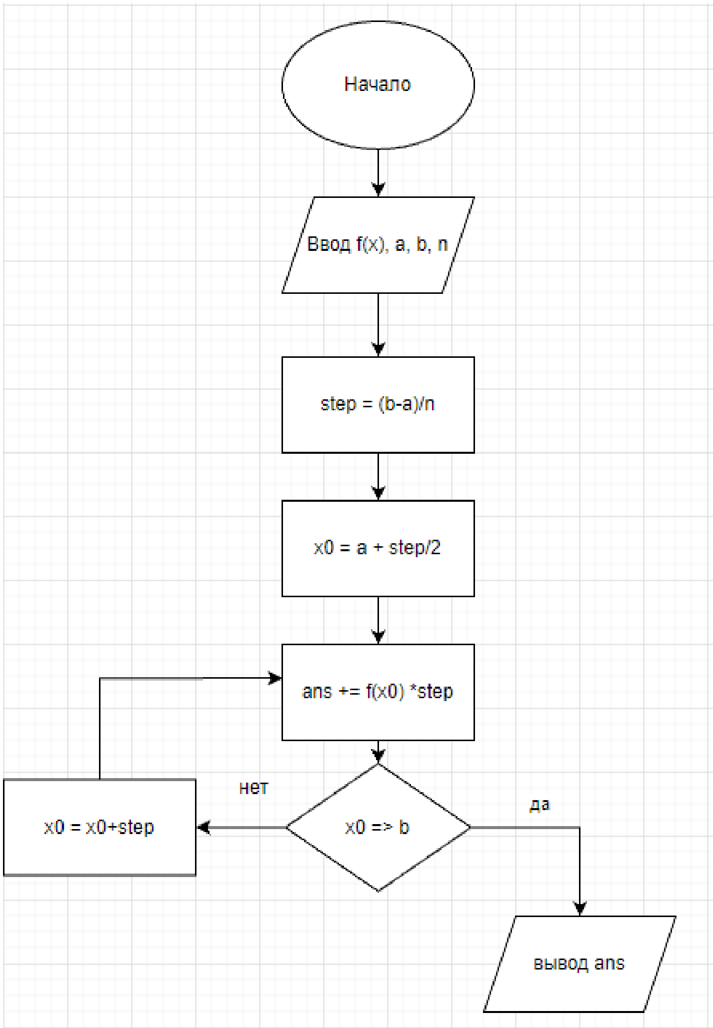
Перл О.В

Санкт-Петербург, 2024 г.

Описание метода

Суть метода заключается в том, что мы делим площадь под функцией (интеграл) на некоторое количество прямоугольников, затем складываем площади всех этих прямоугольников и таким образом у нас получается приближенное значения площади под графиком функции(интеграла). В частности, в методе средних прямоугольников мы высотой прямоугольника берем значение функции от центра этого прямоугольника.

Блок-схема метода



Код численного метода:

```
class Result:
    error_message = ""
    has_discontinuity = False

    def first_function(x: float):
        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):
        return math.log(x)

    # How to use this function:
    # func = Result.get_function(4)
    # func(0.01)
    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.")

    #
    # Complete the 'calculate_integral' function below.
    #
    # The function is expected to return a DOUBLE.
    # The function accepts following parameters:
    #  1. DOUBLE a
    #  2. DOUBLE b
    #  3. INTEGER f
    #  4. DOUBLE epsilon
    #

    def calculate_integral(a, b, f, epsilon):
        Result.error_message = ""
        Result.has_discontinuity = False

        # Get the function to integrate
        func = Result.get_function(f)

        def remove_discontinuity(f):
            def g(x):
                if f(x):
                    return f(x)
```

```
                else:
                    return 0.5 * (f(x + 1e-6) + f(x - 1e-6))

            return g

        # Check for discontinuity
        try:
            test_points = [a + epsilon * i for i in range(int((b - a) /
epsilon) + 1)]
            for point in test_points:
                func(point)
        except:
            Result.error_message = "Integrated function has discontinuity or
does not defined in current interval"
            Result.has_discontinuity = True
            return -69.420
        # Убираем разрывы первого рода принимая среднее значение
        func = remove_discontinuity(func)

        def integrate_middle_rect(f, a, b, n):
            """
            Вычисляет приближенное значение определенного интеграла функции f
на интервале [a, b]
            с использованием метода средних прямоугольников с n
прямоугольниками.
            """
            dx = (b - a) / n  # ширина прямоугольников
            x = a + dx / 2  # координата центра первого прямоугольника
            area = 0  # суммарная площадь всех прямоугольников

            for i in range(n):
                area += f(x) * dx  # вычисляем площадь текущего
прямоугольника
                x += dx  # перемещаемся к центру следующего прямоугольника
            return area

        n = 1
        while True:
            l = integrate_middle_rect(func, a, b, n)
            r = integrate_middle_rect(func, a, b, n * 2)
            if abs(l - r) < epsilon:
                return r
            n += 1

try:
    fun = int(input("Enter function number: "))
    a = float(input("Enter a: "))
    b = float(input("Enter b: "))
    eps = float(input("Enter epsilon: "))
except ValueError:
    print("Please enter valid values")
    exit()
Result.calculate_integral(a, b, fun, eps)
if Result.has_discontinuity:
    print(Result.error_message)
else:
    print(Result.calculate_integral(a, b, fun, eps))
```

p.s. стоит отметить, что в данной реализации метод модифицирован, так как в поставленной задаче нужно было находить ответ с определенной точностью. В оригинальном же методе, точность задается количеством прямоугольников (чем их больше, тем точнее будет ответ), а не эпсилоном.

Пример работы:

1) Для функции $x^2 + 2$

```
Enter function number: 3
Enter a: 0
Enter b: 10
Enter epsilon: 0.001
353.33299999999434
```

2) Для функции $log(x)$

```
Enter function number: 5
Enter a: -1
Enter b: 1
Enter epsilon: 0.001
Integrated function has discontinuity or does not defined in current interval
```

3) Для той же функции, но уже в определенной области

```
Enter function number: 5
Enter a: 1
Enter b: 122
Enter epsilon: 0.001
465.0909012594881
```

4) Для функции $2x + 2$

```
Enter function number: 4
Enter a: -10
Enter b: 0
Enter epsilon: 0.001
-80.0
```

5) Для той же функции, но уже в положительном поле

```
Enter function number: 4
Enter a: 10
Enter b: 500
Enter epsilon: 0.0001
250880.0
```

Вывод:

Метод вычисления интеграла методом средних прямоугольников является достаточно простым и в то же время достаточно эффективным методом для вычисления интегралов, его алгоритмическая сложность O(n) в случае классической реализации и $O(n^2)$ в моем случае, так как условие задачи предполагает использование некоторого эпсилона.
Из примеров работы, можно сделать вывод, что метод не применим только если функция не определена на нужном интервале, или если функция имеет разрывы на интервале. В сравнении с другими

методами прямоугольников мне кажется, что метод средних прямоугольников, является оптимальным, так как он берет среднее значение функции в этом прямоугольнике, тогда как правые и левые прямоугольники берут крайние значения, которые могут дать бОльшую ошибку. В сравнении с другими методами, а именно с методом трапеций, метод прямоугольников может не подходить для функций, которые резко меняются, в то же время метод трапеций может работать лучше, но при этом немного медленнее, так как для трапеции требуется дополнительные вычисления (хотя оба метода имеют сложность O(n)).

Численная ошибка может накапливаться, во-первых, от неточности вычислений в компьютере, во-вторых, в методе прямоугольников она может накапливаться от того, что мы не учитываем поведение функции и прямоугольники могут не очень хорошо аппроксимировать эту функцию.