УНИВЕРСИТЕТ ИТМО

Факультет Программной Инженерии и Компьютерных Технологий - 2020

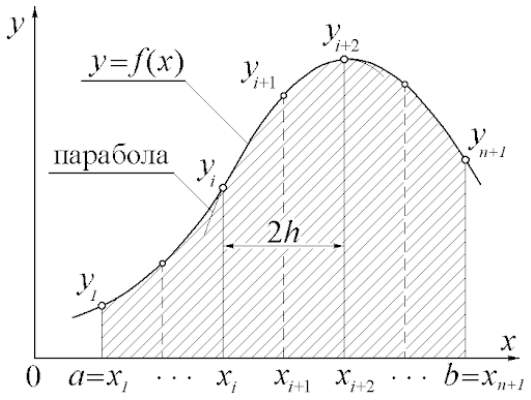
Вычислительная математика

ЛАБОРАТОРНАЯ РАБОТА №2:

Интегрирование

Метод симпсона

Преподаватель: Калёнова О.В.
Выполнил: Купирянов А.А, P3212



1 Теория

Метод симпсона - численный метод интегрирования, суть которого заключается в приближении графика функции на определенном отрезке параболой.

Основой метода симпсона является разбиение подынтегральной функции на несколько частей, а затем вычисление площади каждой из частей как криволинейной трапеции, ограниченной сверху параболой

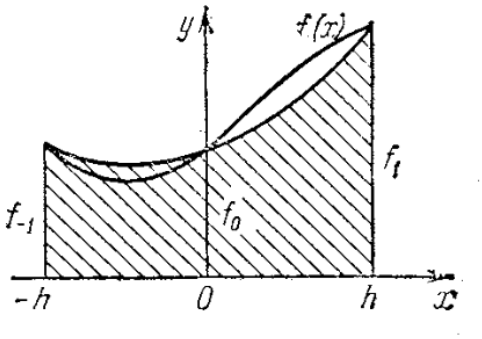


Рис. 1: Криволинейная трапеция

Очевидно, что чем больше количество частей на которые разбивается подынтегральная функция, тем больше точность, так как не смотря на то, что используется парабола - это все равно интерполяционный многочлен.

Таким образом, можно получить необходимую точность путем задания количества разбиений.

2 Формула Симпсона

Рассмотрим рисунок 1. В ней трапеция ограничена параболой проходящей через точки $(-h, f_{-1})$, $(0, f_0)$, (h, f_1), где $f_i = f(ih)$
Уравнение указанной параболы:

$$y = f_0 + \frac{f_1 - f_{-1}}{2h}x + \frac{f_{-1} - 2f_0 + f_1}{2h^2}x^2$$

Отсюда находим:

$$\int_{-h}^{h} ydx = \frac{h}{3}(f_{-1} + 4f_0 + f_1)$$

Из интересного можно заметить, что точка $(0, f_0)$ в 4 раза весомее каждой из двух своих соседей

Таким образом, формула Симпсона имеет вид:

$$\int_{-h}^{h} f(x)dx \approx \frac{h}{3}(f_{-1} + 4f_0 + f_1)$$

2.1 Приближенное вычисление интеграла

Для использования этой формулы для численного метода нам необходимо знать отрезок интегрирования и количество частей, на которые будет делиться подынтегральная функция.

С другой стороны, пользователь не заинтересован в количестве разбиений - его интересует точность вычислений. Поэтому разумно будет дать пользовтелю ввод точности и в зависимости от этих данных вычислить количество разбиений.

Для этого в лабораторной работе было предложено использование оценки погрешности с помощью правила Рунге.

Суть его заключается в том, чтобы сравнить найденное значение с разбиением в два раза меньшим, чем с текущим.

$$I - I_{h/2} \approx \frac{I_{h/2} - I_h}{2^k - 1}$$

Так как значение точности вычисляется в ходе выполнения программы, то мы можем сами задать начальное рабиение, и дальше увеличивать его в двое.

3 Программа

Технические данные: Java 11, OpenJDK 11, OS: Windows Linux Subsystem

Пусть, n - коилчество разбиений функции на части, при том, n - четное, такое что каждое разбиение - имеет длину на оси x: $h/2$

При каждой итерации (кроме первой) мы будет сравнивать предыдущее найденное значение с $n/2$, таким образом вычисляя текущую точность.

Как ранее уже говорилось, средняя часть имеет вес в 4 раза больший, чем соседние. Тем не менее, соседние учитываются дважды (при вычислении соседнего разбиения).

Таким образом, в зависимости от четности порядкового номера мы можем раздельно их суммировать, а затем в результате умножить эту сумму на соответствущий коэффициент

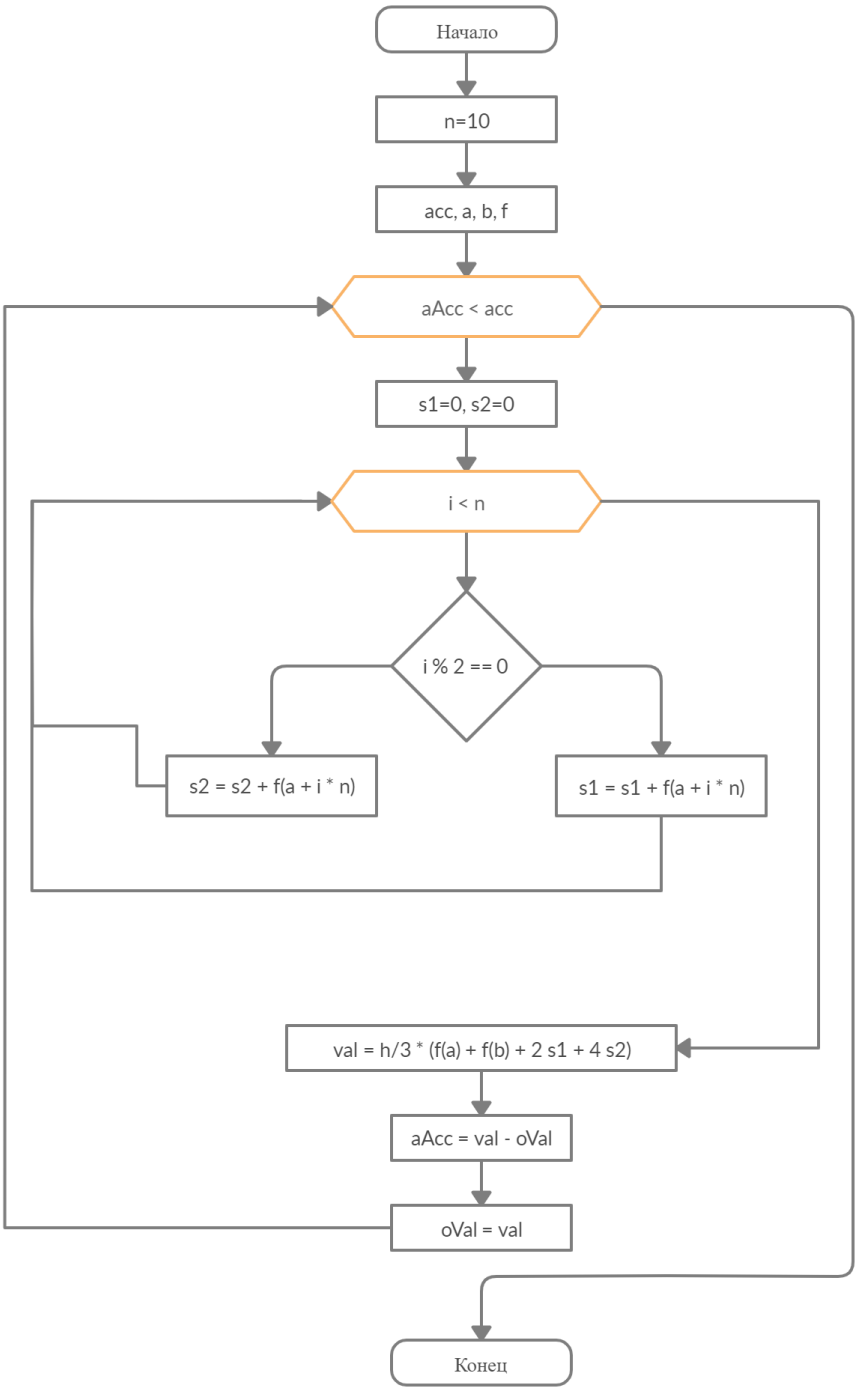


Рис. 2: Диаграмма

3.1 Реализация

```
public double solve(Function<Double, Double> function, int top, int bottom, int partition) {
    int [] boundaries = new int[2];
    addBoundariesTo(boundaries, top, bottom);

    double step = (boundaries[1] - boundaries[0] * 1d) / partition;

    double sum1 = 0;
    double sum2 = 0;

    for (int i = 0; i < partition; i++) {
        double functionValue = function.apply(boundaries[0] + i * step);
        if (i % 2 == 0) {
            sum1 = sum1 + functionValue;
        } else {
            sum2 = sum2 + functionValue;
        }
    }
    byte sign = (byte) (top > bottom ? 1 : -1);
    return sign * step / 3 * (function.apply((double) boundaries[0]) + function.apply((double)
        boundaries[1]) + 2 * sum1 + 4 * sum2);
}
```

```
public double solveWithAccuracy
(Function<Double, Double> function, int top, int bottom, double expectedAccuracy){
    int partition = 10;
    double actualAccuracy = Double.MAX_VALUE;
    double oldValue = solve(function, top, bottom, partition);
    while (actualAccuracy > expectedAccuracy) {

        partition = partition * 2;
        double actualValue = solve(function, top, bottom, partition);
        actualAccuracy = Math.abs(actualValue - oldValue) / 15;
        oldValue = actualValue;

    }
    lastAccuracy = actualAccuracy;
    lastPartition = partition;
    return oldValue;
}
```

4 Заключение

Зачастую метод Симпсона точнее остальных методов через квадратурные формулы прямоугольников или трацеций. Это вызвано тем, что для интерполяции используется многочлен второго порядка, который дает более точное приближение, чем остальные (в зависимости от подынтегральной функции). Но также основным плюсом данного метода является безусловно формула Симсона, которая упрощает вычисление интерполированного интеграла путем сведения его к одной формуле.

Из плюсов также можно отметить, то что мы имеем контроль точности, а следовательно можем экономить вычислительные мощности по мере возможности.

Хоть и формулы прямоугольников и трапеций по отдельности уступают формуле Симпсона при интегрировании "гладких"функций, но в паре они могут дать двустороннее приближение интеграла. Но эта тема уже касается усложненных квадратурных формул.

А одним из минусов самого метода Симпсона является излишняя нагрузка ресурсов для вычисления, допустим, линейных функций, так как она в методе также будет интерполироваться в многочлен второй степени.