Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №4
Интегрирование

по курсу «Вычислительной математики»

Выполнил:
Студент группы P3230
Пономаренко Алиса Валерьевна

Преподаватель:
Перл Ольга Вячеславовна

Санкт-Петербург
2024

Описание численного метода

Метод Симпсона

Метод Симпсона — это численный метод интегрирования, который использует квадратурные формулы для приближенного вычисления определенных интегралов. Основная идея метода заключается в аппроксимации подынтегральной функции кусочно-квадратичной функцией на каждом интервале интегрирования.

Описание метода Симпсона

Предположим, что нам нужно вычислить определенный интеграл $\int_a^b f(x)\,dx$.

1. **Разбиение интервала:** Интервал $[a, b]$ разбивается на четное количество подинтервалов с одинаковым шагом h. Таким образом, точки разбиения будут $x_i = a + i \cdot h$, где $h = \frac{b-a}{n}$, а n — количество частей (должно быть четным числом).

2. **Формула Симпсона:** Для каждых трех последовательных точек x_{i-1}, x_i, x_{i+1} используется квадратурная формула Симпсона:

$$\int_{x_{i-1}}^{x_{i+1}} f(x)\,dx \approx \frac{h}{3}[f(x_{i-1}) + 4f(x_i) + f(x_{i+1})]$$

3. **Суммирование:** Для вычисления интеграла на всем интервале $[a, b]$, суммируются значения квадратурных формул по всем частям интервала:

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + \ldots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right]$$

   Здесь $h = \frac{b-a}{n}$.

Алгоритм вычисления интеграла методом Симпсона

1. **Выбор числа разбиений** n: Выбирается четное число n, которое определяет количество подинтервалов, на которые будет разбит интервал интегрирования $[a, b]$.

2. **Вычисление шага** h: Вычисляется шаг разбиения как $h = \frac{b-a}{n}$.

3. **Вычисление узлов** x_i: Вычисляются узлы разбиения $x_i = a + i \cdot h$ для $i = 0, 1, \ldots, n$.

4. **Применение формулы Симпсона**: Для каждой тройки последовательных узлов x_{i-1}, x_i, x_{i+1} вычисляется приближенное значение интеграла по формуле Симпсона:

$$\int_{x_{i-1}}^{x_{i+1}} f(x)\,dx \approx \frac{h}{3}[f(x_{i-1}) + 4f(x_i) + f(x_{i+1})]$$

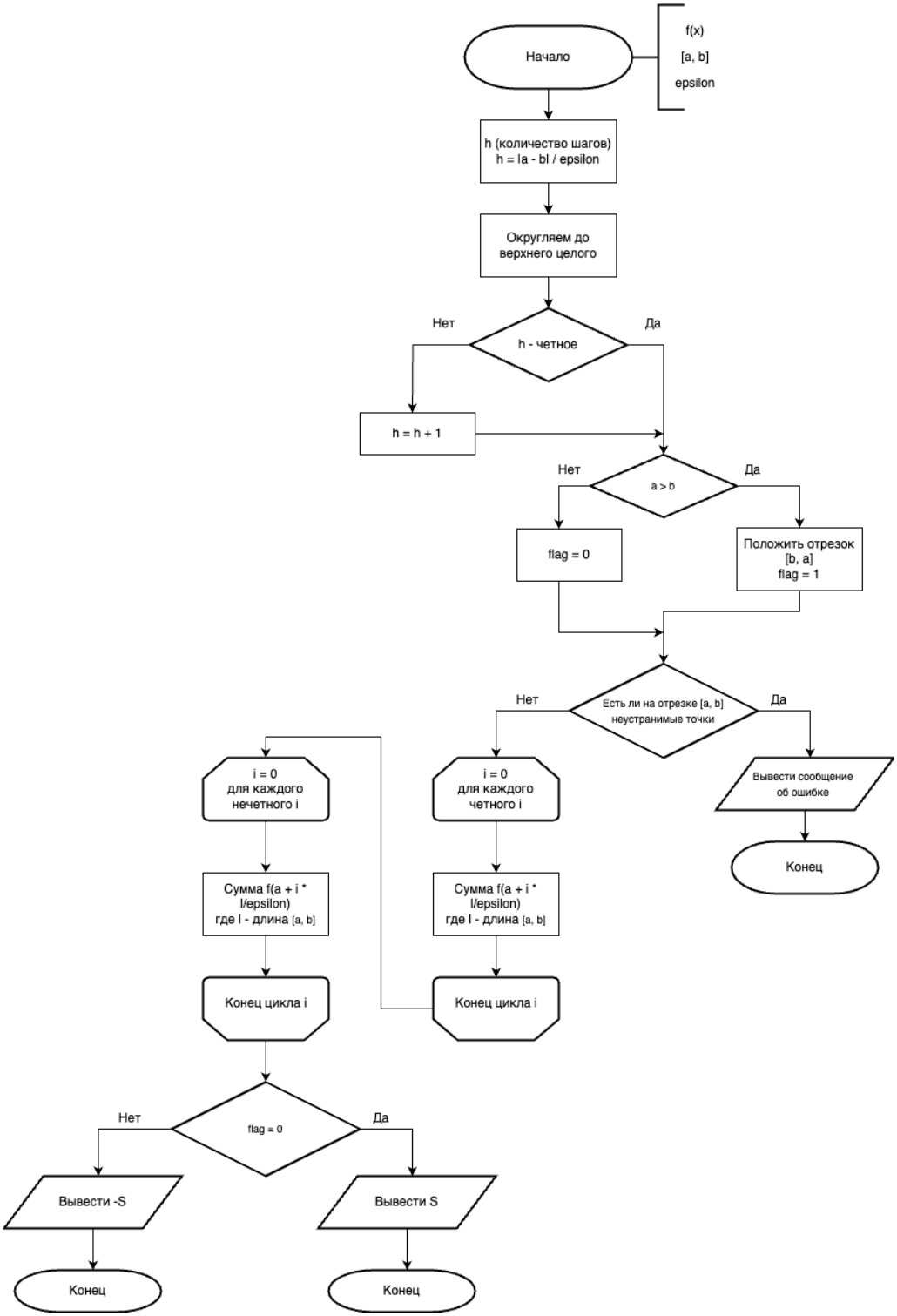
5. **Суммирование**: Суммируются значения, полученные из предыдущего шага, чтобы получить приближенное значение интеграла на всем интервале $[a, b]$:

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right]$$

   Здесь $h = \frac{b-a}{n}$.

6. **Вывод результата**: Полученная сумма является приближенным значением интеграла $\int_a^b f(x)\,dx$.

Блок-схема по составленному описанию метода



Код численного метода

```
1 usage
public static double calculate_integral(double a, double b, int f, double epsilon) {
    Function<Double, Double> func = get_function(f);
    boolean flag = !(a < b);
    double len = flag ? a - b : b - a;
    int n = (int) (len / epsilon);
    n = n % 2 == 0 ? n + 2 : n + 1;
    double h = len / n;

    // Проверка на то, имеется ли разрыв второго рода.
    for (double t = a; t < b; t += h) {
        if (func.apply(t).isInfinite() || func.apply(t).isNaN()) {
            has_discontinuity = true;
            error_message = "Integrated function has discontinuity or does not defined in current interval";
            return -1;
        }
    }

    //если a < b то полагаем отрезок [b, a]
    if (flag) {
        double c = a;
        a = b;
        b = c;
    }
    double integral = 0;
    double x0 = a;
    double x2 = a + 2 * h;

    while (x2 < b) {
        double x1 = x0 + h;
        double sum;
        try {
            sum = h / 3 * (func.apply(x0) + 4 * func.apply(x1) + func.apply(x2));
        } catch (ArithmeticException e) {
            //устранимая точка развыва, берем число рядом
            sum = h / 3 * (func.apply( t: x0 + h / 10) + 4 * func.apply( t: x1 + h / 10)
                    + func.apply( t: x2 + h / 10));
        }
        integral += sum;
        x0 = x2;
        x2 += 2 * h;

    }
```

```
        }
        // Последний шаг, который меньше 2
        if (x0 < b) {
            try {
                double x1 = (x0 + b) / 2;
                integral += (b - x0) / 6 * (func.apply(x0) + 4 * func.apply(x1) + func.apply(b));
            } catch (ArithmeticException e) {
                double x1 = (x0 + b) / 2;
                integral += (b - x0) / 6 * (func.apply( t: x0 - h / 10) +
                        4 * func.apply( t: x1 - h / 10) + func.apply( t: b - h / 10));
            }
        }
        return flag ? -integral : integral;
    }
```

Примеры работы программы

1. Входные данные: (Первая функция - без точек разрыва второго рода, a < b)

1
2
1
0.001

Выходные данные:
0.6931471805599575

2. Входные данные: (Первая функция - без точек разрыва второго рода, a > b)

2
1
1
0.001

Выходные данные:
-0.6931471805599575

3. Входные данные: (Первая функция - с точкой разрыва второго рода, a < b)

0
2
1
0.001

Выходные данные:
Integrated function has discontinuity or does not defined in current interval

4. Входные данные: (Пятая функция - с точками, которые выходят за область определения функции)

-1
1
5
0.001

Выходные данные:
Integrated function has discontinuity or does not defined in current interval

5. Входные данные: (Вторая функция - с устранимой точкой разрыва первого рода, a < b)
0
2
2
0.001

Выходные данные:
1.6054119768027186

6. Входные данные: (Третья функция - a < b, квадратичная функция, особых точек нет)
-1
1
3
0.001

Выходные данные:
4.6666666666666154

7. Входные данные: (Четвертая функция - a < b, линейная функция, особых точек нет)
-1
1
4
0.001

Выходные данные:
3.999999999999965

Выводы

1. Метод правильно работает как с целыми числами, так и с числами с дробной частью, потому что в коде вычисления производятся в типе `double`. Преобразование типов из `double` в `int` происходит автоматически в Java.

2. Метод обеспечивает корректную проверку на наличие точек разрыва первого и второго рода, устраняет первый из них. В случае наличия разрыва второго рода - выводит сообщение об ошибке.

3. Метод обеспечивает корректную проверку на вхождение отрезка в область определения функции. В случае выхода за ОДЗ - выводит сообщение об ошибке.

Сравнение с другими методами интегрирования

Метод Симпсона

Принцип: Численный метод интегрирования, использующий квадратичную аппроксимацию для приближенного нахождения интеграла функции. Делит интервал на подынтервалы и применяет параболические аппроксимации.

Преимущества: Высокая точность для гладких функций, быстрая сходимость (обычно лучше, чем у метода трапеций).

Недостатки: Требует четного числа подынтервалов, сложнее вычислительно по сравнению с методом трапеций.

Метод прямоугольников

Принцип: Деление интервала интегрирования на подынтервалы и замена функции на значение в одной из точек подынтервала (обычно в середине).
Преимущества: Простота реализации, быстрая вычисляемость.
Недостатки: Низкая точность, особенно для функций с большой кривизной.

Метод трапеций

Принцип: Деление интервала интегрирования на подынтервалы и аппроксимация функции линейными отрезками (трапециями).
Преимущества: Простота реализации, лучшая точность по сравнению с методом прямоугольников.
Недостатки: Меньшая точность по сравнению с методом Симпсона, особенно для функций с сильной кривизной.

Метод Гаусса-Кронрода

Принцип: Численный метод, использующий точки и веса Гауссовой квадратуры для точного интегрирования полиномов.
Преимущества: Высокая точность для полиномов и гладких функций, эффективное использование вычислений.
Недостатки: Сложность реализации, требует вычисления специфических точек и весов.

Преимущества метода Симпсона:

- Лучшая точность для гладких функций благодаря квадратичной аппроксимации.
- Меньшее количество подынтервалов для достижения той же точности.

Недостатки метода Симпсона:

- Требует четного числа подынтервалов, что может быть неудобно в некоторых случаях.
- Более сложный в реализации и вычислениях по сравнению с методом трапеций.

Анализ применимости метода Симпсона

Метод Симпсона идеально подходит в следующих случаях:

- Гладкие функции: Для функций, которые можно хорошо аппроксимировать параболами, метод Симпсона даёт высокую точность.
- Высокие требования к точности: Если требуется высокая точность интегрирования, метод Симпсона обеспечивает лучшие результаты.
- Малое количество подынтервалов: Когда важно минимизировать количество вычислений, метод Симпсона достигает нужной точности с меньшим числом подынтервалов.

Метод Симпсона может быть менее подходящим в следующих случаях:

- Разрывные функции: Для функций с разрывами или острыми пиками метод может давать большие погрешности.
- Неудобное число подынтервалов: Если требуется использовать нечетное количество подынтервалов, метод становится неудобным.
- Простота и быстрота: В случаях, когда простота и быстрота важнее точности, метод прямоугольников или трапеций может быть предпочтительнее.
- Сложность реализации: Для задач, где важна простота кода и легкость реализации, метод трапеций может быть более удобным выбором.

Алгоритмическая сложность метода: $O(n), n = |b - a|/\epsilon$

Анализ численных ошибок метода Симпсона

Ошибки округления

Ошибки округления возникают из-за конечной точности представления чисел в компьютере. Эти ошибки могут накапливаться при выполнении большого числа арифметических операций, особенно при многократном применении метода Симпсона на мелких подынтервалах. Для минимизации ошибок округления важно использовать числовые типы с достаточной точностью (например, `double` вместо `float`) и, при необходимости, алгоритмы с контролем ошибок округления.

Первоначальное приближение

Первоначальное приближение, точнее, количество подынтервалов n, существенно влияет на точность метода Симпсона. Недостаточное количество подынтервалов может привести к значительным ошибкам аппроксимации, особенно для функций с высокой кривизной или разрывами. Оптимальное количество подынтервалов определяется исходя из требуемой точности и свойств интегрируемой функции.

Несходимость функции

Метод Симпсона предполагает, что интегрируемая функция непрерывна и хорошо аппроксимируется параболами на заданных подынтервалах. В случае, если функция имеет разрывы или точки разрыва второго рода, метод может давать большие ошибки или вовсе не сходиться. Необходимо предварительно анализировать функцию на наличие таких особенностей и при необходимости выбирать другие методы интегрирования.

Вариации по уменьшению численных ошибок

Выбор начального приближения

Хорошее начальное приближение может существенно улучшить сходимость и уменьшить накопление ошибок. Оптимальное количество подынтервалов n должно быть выбрано таким образом, чтобы обеспечить достаточную точность аппроксимации при минимальном числе итераций.

Проверка на сходимость функций

При интегрировании функций с возможными разрывами или особыми точками необходимо провести предварительный анализ на сходимость. Если функция имеет разрывы или не определена на некоторых участках, следует постараться выбрать другую функцию или преобразовать исходную задачу. Если это невозможно, метод Симпсона не применим, и необходимо использовать другие численные методы.