НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет ПИиКТ

Отчёт по лабораторной работе на тему:

Интегрирование

Метод Симпсона

Выполнил студент

Агнистова Алина Юрьевна

Группа № P3225

Преподаватель: Перл Ольга Вячеславовна

г. Санкт-Петербург

2024

**Описание метода и расчётные формулы**

Метод Симпсона – метод интегрирования, заключающийся в приближении подынтегральной функции на отрезке интерполяционным многочленом второй степени :

Для точности вычислений интеграла интервал разбивают на чётное количество отрезков одинаковой длины и получают формулу Симпсона на отрезках. Значение исходного интеграла – сумма результатов интегрирования на отрезках:

**Блок-схема реализованного метода**

Изображение выглядит как снимок экрана, черно-белый, черный, дизайн

Автоматически созданное описание

**Код метода**

string error\_message = "";

**bool** has\_discontinuity = **false**;

**double** first\_function(**double** x) {

**return** 1 / x;

}

**double** second\_function(**double** x) {

**if** (x == 0)

**return** 1.0;

**return** sin(x) / x;

}

**double** third\_function(**double** x) {

**return** x \* x + 2;

}

**double** fourth\_function(**double** x) {

**return** 2 \* x + 2;

}

**double** five\_function(**double** x) {

**return** log(x);

}

**double** default\_function(**double** x){

**return** 0;

}

/\*

\* How to use this function:

\* fn\_t& f = get\_function(4);

\* f(0.0001);

\*/

fn\_t &get\_function(**int** n) {

**switch** (n) {

**case** 1:

**return** first\_function;

**case** 2:

**return** second\_function;

**case** 3:

**return** third\_function;

**case** 4:

**return** fourth\_function;

**case** 5:

**return** five\_function;

**default**:

**return** default\_function;

}

}

/\*

\* 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

\*/

**double** calculate\_integral(**double** a, **double** b, **int** f, **double** epsilon) {

**bool** neg = **false**;

**if** (a > b){

swap(a,b);

neg = **true**;

}

fn\_t &func = get\_function(f);

**int** n = 1;

**double** h = (b - a);

**double** x1 = 0, x2 = 0;

**double** s0, s1;

s1 = (h / 6) \* (func(a) + 4 \* func((a + b) / 2) + func(b));

**if**(isinf(func(a)) || isnan(func(a)) || isinf(func(b)) || isnan(func(b))){

error\_message = "Integrated function has discontinuity or does not defined in current interval";

has\_discontinuity = **true**;

**return** 0;

}

**do** {

s0 = s1;

s1 = 0;

n = 2 \* n;

h = (b - a) / n;

**double** k1 = 0, k2 = 0;

**for** (**int** i = 1; i < n; i += 2) {

x1 = func(a + i \* h);

x2 = func(a + (i + 1) \* h);

**if**(isinf(x1) || isnan(x1) || isinf(x2) || isnan(x2)){

error\_message = "Integrated function has discontinuity or does not defined in current interval";

has\_discontinuity = **true**;

**return** 0;

}

k1 += x1;

k2 += x2;

}

s1 = h / 3 \* (func(a) + 4 \* k1 + 2 \* k2);

} **while** (fabs(s1-s0) > epsilon);

**if** (neg){

s1 = - s1;

}

**return** s1;

}

**Примеры и результаты работы программы**

Пример 1

Выбранный случай: подынтегральная функция с неустранимым разрывом.

Ожидаемые выходные данные: сообщение об ошибке.

Входные данные:

|  |
| --- |
| -1 |
| 1 |
| 1 |
| 0.001 |

Выходные данные: сообщение об ошибке.

Изображение выглядит как снимок экрана, текст

Автоматически созданное описание

Пример 2

Выбранный случай: подынтегральная функция с устранимым разрывом.

Ожидаемые выходные данные:

Входные данные:

|  |
| --- |
| -1 |
| 1 |
| 2 |
| 0.001 |

Выходные данные:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Пример 3

Выбранный случай: стандартная подынтегральная функция.

Ожидаемые выходные данные:

Входные данные:

|  |
| --- |
| 1 |
| 2 |
| 3 |
| 0.001 |

Выходные данные:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Пример 4

Выбранный случай: Интервал .

Ожидаемые выходные данные:

Входные данные:

|  |
| --- |
| 2 |
| 1 |
| 4 |
| 0.001 |

Выходные данные:

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Пример 5

Выбранный случай: подынтегральная функция с устранимым разрывом.

Ожидаемые выходные данные:

Входные данные:

|  |
| --- |
| 0 |
| 1 |
| 5 |
| 0.001 |

Выходные данные: сообщение об ошибке.

Изображение выглядит как снимок экрана, текст, Шрифт

Автоматически созданное описание

Проанализировав результаты запуска реализованного метода, можно сделать вывод, что код справляется с основными крайними ситуациями, однако я не учла в коде устранимый разрыв 5 функции (пример 5).

Для оценки метода приведена таблица сравнения метода Симпсона с методом прямоугольников и методом трапеций:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Метод Симпсона | Метод прямоугольников | Метод трапеций |
| Описание | Метод, использующий параболическую аппроксимацию между тремя точками на каждом интервале. | Метод, основанный на аппроксимации подграфика функции прямоугольниками. | Метод, использующий линейную аппроксимацию между двумя точками на каждом интервале. |
| Вычислительная сложность | Высокая (необходимость вычислять большего числа значений функции) | Низкая (требует большего числа интервалов для достижения высокой точности) | Средняя (требует больше вычислений, чем метод прямоугольников, но меньше, чем метод Симпсона) |
| Применимость | Функции с небольшой кривизной, где требуется высокая точность. | Функции, где значения меняются медленно. | Может быть менее эффективным для функций с высокой кривизной. |
| Численная стабильность | Высокая | Может давать неточные результаты для функций с большой кривизной | Высокая |

Таким образом, метод Симпсона будет оптимальным выбором в случаях, когда требуется высокая точность и допускается высокое число вычислений. В случаях, когда допускается точность ниже можно отдать предпочтение методу трапеций. Метод прямоугольников отличается меньшей точностью вычислений, но и вычислительная сложность метода ниже всего.

Временная алгоритмическая сложность метода составляет , так как для вычисления интеграла используется цикл do-while, внутри которого находится цикл, в котором считается сумма.

Метод отличается высокой численной стабильностью, ошибки могут быть вызваны недостаточной точностью программной реализации метода, но в моём случае таких проблем нет.

**Вывод**

В результате выполнения лабораторной работы была реализовано вычисление интегралов методом Симпсона на языке программирования C++. Программа разделяется на 2 основных аспекта: вычисление сумм и нахождение решения интеграла. Алгоритмическая сложность метода составила .

Метод Гаусса – метод вычисления интегралов, отличающийся высокой эффективностью и точностью, однако может вызвать излишнюю вычислительную сложность в случаях, где допускаются погрешности результата.