НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет ПИиКТ

Отчёт по лабораторной работе на тему:

Интегрирование

Метод Симпсона

Выполнил
студент

Агнистова Алина Юрьевна

Группа № Р3225

Преподаватель: Перл Ольга Вячеславовна

г. Санкт-Петербург

2024

Описание метода и расчётные формулы

Метод Симпсона – метод интегрирования, заключающийся в приближении подынтегральной функции на отрезке $[a, b]$ интерполяционным многочленом второй степени $p_2(x)$:

$$\int_a^b f(x)dx = \int_a^b p_2(x)dx = \frac{h}{6}\left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right), \text{где}$$

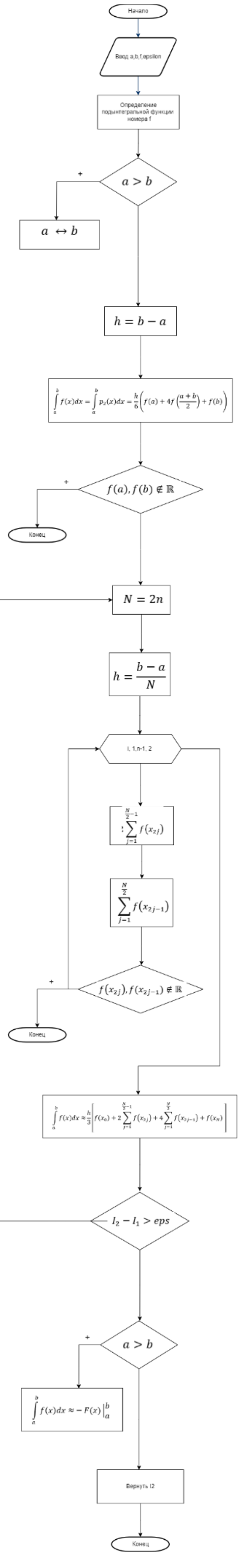
$$h = b - a - \text{ширина интервала}$$

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:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{\frac{N}{2}-1} f(x_{2j}) + 4\sum_{j=1}^{\frac{N}{2}} f(x_{2j-1}) + f(x_N)\right], \text{где}$$

$$N = 2n - \text{количество отрезков}, h = \frac{b-a}{N} - \text{величина шага}$$

Блок-схема реализованного метода



Код метода

```cpp
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.

```
-1
1
1
0.001
Integrated function has discontinuity or does not defined in current interval

Process finished with exit code 0
```

Пример 2

Выбранный случай: подынтегральная функция с устранимым разрывом.

Ожидаемые выходные данные: ~1.89

Входные данные:

-1
1
2
0.001

Выходные данные: 1.89271

```
-1
1
2
0.001
1.89271

Process finished with exit code 0
```

Пример 3

Выбранный случай: стандартная подынтегральная функция.

Ожидаемые выходные данные: ~4.3

Входные данные:

1
2
3
0.001

Выходные данные: 4.33431

```
1
2
3
0.001
4.33431

Process finished with exit code 0
```

Пример 4

Выбранный случай: Интервал $[a, b]$, где $a > b$.

Ожидаемые выходные данные: ~ -5

Входные данные:

2
1
4
0.001

Выходные данные: -5.00098

```
2
1
4
0.001
-5.00098

Process finished with exit code 0
```

Пример 5

Выбранный случай: подынтегральная функция с устранимым разрывом.

Ожидаемые выходные данные: -1

Входные данные:

0
1
5
0.001

Выходные данные: сообщение об ошибке.

```
0
1
5
0.001
Integrated function has discontinuity or does not defined in current interval

Process finished with exit code 0
```

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	Метод, основанный на аппроксимации подграфика функции прямоугольниками.	Метод, использующий линейную аппроксимацию между двумя точками на каждом

	интервале.		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 $O(n)$, 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 С++. Программа разделяется на 2 основных аспекта: вычисление сумм и нахождение решения интеграла. Алгоритмическая сложность метода составила $O(n)$.

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