Университет ИТМО

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.04 Программная инженерия
Дисциплина «Вычислительная математика»

Отчет
По лабораторной работе №3
Вариант 7

Выполнил:
Зинатулин А.В.
P32121

Преподаватель:
Наумова Н.А.

Санкт-Петербург, 2023 г.

**Цель работы**
найти приближенное значение определенного интеграла с требуемой точностью различными численными методами.

**Порядок выполнения лабораторной работы**
1. Пользователь выбирает функцию, интеграл которой требуется вычислить (3-5 функций), из тех, которые предлагает программа.
2. Пределы интегрирования задаются пользователем.
3. Точность вычисления задается пользователем.
4. Начальное значение числа разбиения интервала интегрирования: n=4.
5. Ввод исходных данных осуществляется с клавиатуры.
Программная реализация задачи:
1. Реализовать в программе методы по выбору пользователя:
• Метод прямоугольников (3 модификации: левые, правые, средние)
• Метод трапеций
• Метод Симпсона
2. Методы должны быть оформлены в виде отдельной(ого) функции/класса.
3. Вычисление значений функции оформить в виде отдельной(ого) функции/класса.
4. Для оценки погрешности и завершения вычислительного процесса использовать правило Рунге.
5. Предусмотреть вывод результатов: значение интеграла, число разбиения интервала интегрирования для достижения требуемой точности.
Вычислительная реализация задачи:
1. Вычислить интеграл, приведенный в таблице 1, точно.
2. Вычислить интеграл по формуле Ньютона – Котеса при .
3. Вычислить интеграл по формулам средних прямоугольников, трапеций и Симпсона при .
4. Сравнить результаты с точным значением интеграла.
5. Определить относительную погрешность вычислений для каждого метода.
6. В отчете отразить последовательные вычисления.

**Рабочие формулы методов**

Формула Ньютона-Котеса

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n} f(x_i) * c_n^i\text{, } где\ c_n^i - коэффициенты\ Котеса$$

**Код программы**
https://github.com/uvuv-643/Computational_Mathematics

**Листинг кода:**

```
//
// Created by R1300-W-8-Stud on 16.03.2023.
//

#include "SimpsonMethod.h"

SimpsonMethodResult SimpsonMethod::performIteration(CFunctionSV* f, float a, float b, size_t
number_of_intervals) {
    float square = (f->f(a) + f->f(b));
    float dx = (b - a) / (float) number_of_intervals;
    for (size_t i = 1; i < number_of_intervals; i++) {
        square += f->f(a + dx * i) * ((i % 2 == 0) ? 2.0 : 4.0);
    }
    return { dx * square / 3 };
```

```
}

SimpsonMethodResult SimpsonMethod::perform(CFunctionSV* f, float a, float b, float eps, size_t
number_of_intervals) {
    SingleFunctionIntegralMethodData method_data(f, a, b);
    SimpsonMethodResult prev_iteration_result = performIteration(f, a, b, number_of_intervals);
    SimpsonMethodResult curr_iteration_result;
    CVector<CFloat> squares;
    CVector<CSize> intervals;
    squares.push_back(prev_iteration_result.getSquare());
    intervals.push_back(number_of_intervals);
    for (size_t i = 0; i < LIMIT_OF_ITERATIONS; i++) {
        curr_iteration_result = performIteration(f, a, b, 2 * number_of_intervals);
        squares.push_back(curr_iteration_result.getSquare());
        intervals.push_back(number_of_intervals * 2);
        number_of_intervals *= 2;
        if (abs(curr_iteration_result.getSquare() - prev_iteration_result.getSquare()) <= eps)
{
            return SimpsonMethodResult(squares, intervals, method_data);
        }
        prev_iteration_result = curr_iteration_result;
    }
    return SimpsonMethodResult(squares, intervals, method_data);
}
```

```
//
// Created by R1300-W-8-Stud on 16.03.2023.
//

#include "RectMethod.h"

RectMethodResult RectMethod::performIteration(CFunctionSV* f, enum RectMethodType type, float
a, float b, size_t number_of_intervals) {
    float square = 0;
    float dx = (b - a) / (float) number_of_intervals;
    for (size_t i = 0; i < number_of_intervals; i++) {
        float target_x;
        switch (type) {
            case LEFT_RECTANGULAR: {
                target_x = a + dx * i;
                break;
            }
            case MIDDLE_RECTANGULAR: {
                target_x = a + dx * (i + 0.5);
                break;
            }
            case RIGHT_RECTANGULAR: {
                target_x = a + dx * (i + 1);
                break;
            }
            default: {
                return {};
            }
        }
        square += f->f(target_x);
    }
    return {square * dx};
}

RectMethodResult RectMethod::perform(CFunctionSV* f, enum RectMethodType type, float a, float
b, float eps, size_t number_of_intervals) {
    SingleFunctionIntegralMethodData method_data(f, a, b);
    RectMethodResult prev_iteration_result = performIteration(f, type, a, b,
number_of_intervals);
    RectMethodResult curr_iteration_result;
```

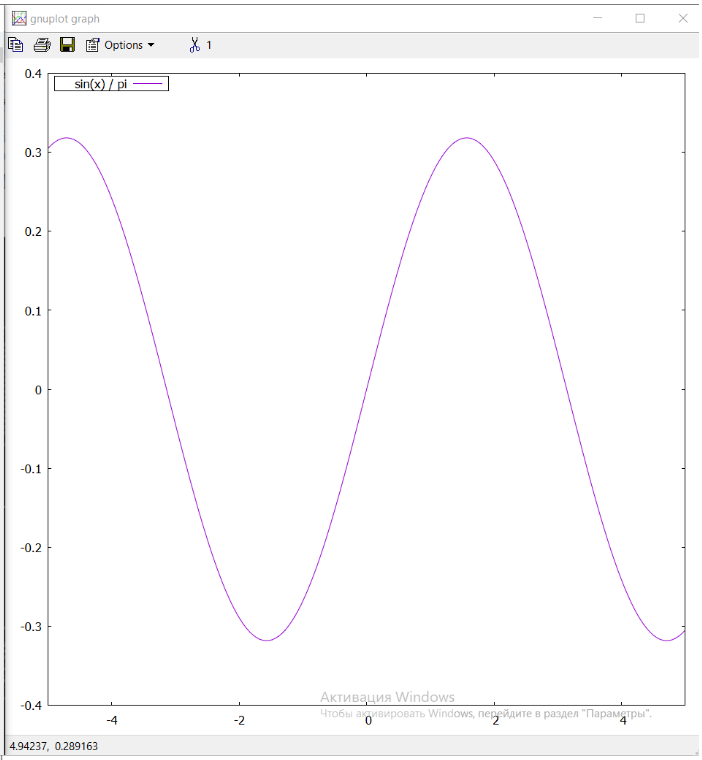
```
    CVector<CFloat> squares;
    CVector<CSize> intervals;
    squares.push_back(prev_iteration_result.getSquare());
    intervals.push_back(number_of_intervals);
    for (size_t i = 0; i < LIMIT_OF_ITERATIONS; i++) {
        prev_iteration_result = curr_iteration_result;
        curr_iteration_result = performIteration(f, type, a, b, 2 * number_of_intervals);
        squares.push_back(curr_iteration_result.getSquare());
        intervals.push_back(number_of_intervals * 2);
        number_of_intervals *= 2;
        if (abs(curr_iteration_result.getSquare() - prev_iteration_result.getSquare()) <= eps)
{
            return RectMethodResult(squares, intervals, method_data);
        }
    }
    return RectMethodResult(squares, intervals, method_data);
}
```

```
//
// Created by R1300-W-8-Stud on 16.03.2023.
//

#include "TrapezeMethod.h"

TrapezeMethodResult TrapezeMethod::performIteration(CFunctionSV* f, float a, float b, size_t
number_of_intervals) {
    float square = (f->f(a) + f->f(b)) / 2;
    float dx = (b - a) / (float) number_of_intervals;
    for (size_t i = 1; i < number_of_intervals; i++) {
        square += f->f(a + dx * i);
    }
    return {dx * square};
}

TrapezeMethodResult TrapezeMethod::perform(CFunctionSV* f, float a, float b, float eps, size_t
number_of_intervals) {
    SingleFunctionIntegralMethodData method_data(f, a, b);
    TrapezeMethodResult prev_iteration_result = performIteration(f, a, b, number_of_intervals);
    TrapezeMethodResult curr_iteration_result;
    CVector<CFloat> squares;
    CVector<CSize> intervals;
    squares.push_back(prev_iteration_result.getSquare());
    intervals.push_back(number_of_intervals);
    for (size_t i = 0; i < LIMIT_OF_ITERATIONS; i++) {
        prev_iteration_result = curr_iteration_result;
        curr_iteration_result = performIteration(f, a, b, 2 * number_of_intervals);
        squares.push_back(curr_iteration_result.getSquare());
        intervals.push_back(number_of_intervals * 2);
        number_of_intervals *= 2;
        if (abs(curr_iteration_result.getSquare() - prev_iteration_result.getSquare()) <= eps)
{
            return TrapezeMethodResult(squares, intervals, method_data);
        }
    }
    return TrapezeMethodResult(squares, intervals, method_data);
}
```

**Результат выполнения программы при различных исходных данных**

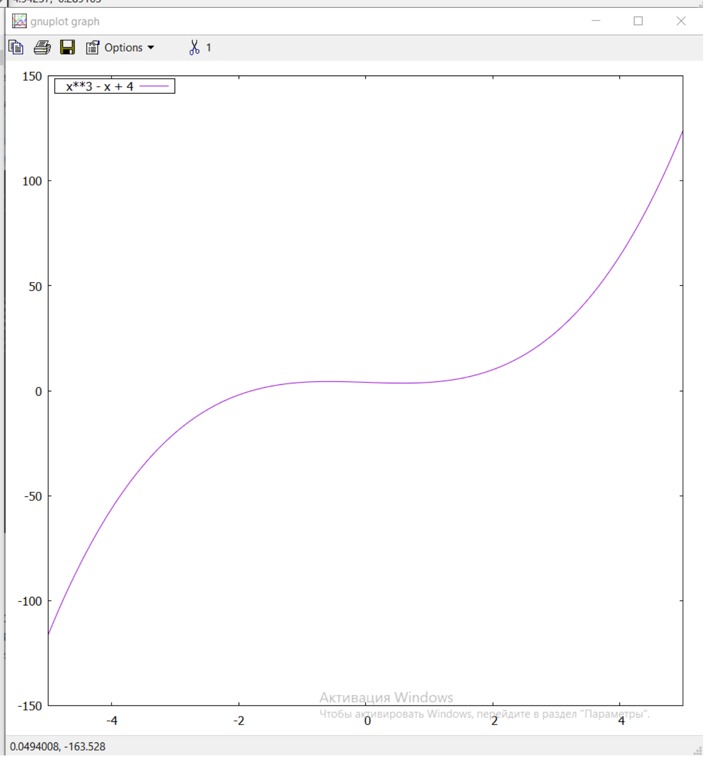
C:\Users\artem\CLionProjects\vm\cmake-build-debug\vm1.exe

```
Right-rectangle method
Choose function:
[1] - x^3 - x + 4
[2] - x^2 * exp(-x^2) - 0.2
[3] - sin(x) / pi
[4] - x^3 + 2.28 * x^2 - 1.934 * x - 3.907
Input a (left border):
Input float:
0
Input b (right border):
Input float:
3.14
Input epsilon:
Input float:
0.00001
| n_{i} | s_{i}      | eps          |
| 4     | 0.60378581 | 0            |
| 8     | 0.62852484 | 0.024739027  |
| 16    | 0.63462454 | 0.0060997009 |
| 32    | 0.63613337 | 0.001508832  |
| 64    | 0.63650417 | 0.0003708005 |
| 128   | 0.63659358 | 8.9406967e-05 |
| 256   | 0.63661474 | 2.1159649e-05 |
| 512   | 0.63661903 | 4.2915344e-06 |
Completed. Press any key to continue...
```




C:\Users\artem\CLionProjects\vm\cmake-build-debug\vm1.exe

```
Trapeze method
Choose function:
[1] - x^3 - x + 4
[2] - x^2 * exp(-x^2) - 0.2
[3] - sin(x) / pi
[4] - x^3 + 2.28 * x^2 - 1.934 * x - 3.907
Input a (left border):
Input float:
0
Input b (right border):
Input float:
1
Input epsilon:
Input float:
0.01
| n_{i} | s_{i}     | eps          |
| 4     | 3.765625  | 0            |
| 8     | 3.7539062 | 0.01171875   |
| 16    | 3.7509766 | 0.0029296875 |
Completed. Press any key to continue...
```




## Вычислительная часть лабораторной работы

$$a)\ \int_0^2 4x^3 - 5x^2 + 6x - 7\,dx = x^4 - \frac{5}{3}x^3 + 3x^2 - 7x \,\Big|\, \frac{2}{0} = 0.(6)$$

$$б)\ \int_0^2 4x^3 - 5x^2 + 6x - 7\,dx = \sum_{i=0}^{5} f(x_i)c_n^i = \sum_{i=0}^{5} f(x_i)c_n^i = \sum_{i=0}^{5} f\left(\frac{b-a}{5}i\right)\, c_n^i$$

$$\int_0^2 4x^3 - 5x^2 + 6x - 7\,dx$$
$$= \frac{19(b-a)}{288}\left(f(0)+f(2)\right) + \frac{75(b-a)}{288}\left(f\left(\frac{2}{5}\right)+f\left(2-\frac{2}{5}\right)\right) + \frac{50(b-a)}{288}\left(f\left(\frac{4}{5}\right)+f\left(2-\frac{4}{5}\right)\right) = 0.(6)$$

1

Метод средних прямоугольников:

2

$$\frac{(b-a)}{n}\sum_{i=1}^{n} f\left(\frac{(b-a)}{n}\cdot\left(i-\frac{1}{2}\right)\right)$$

$= 0.62$

3

Метод трапеций:

4

$$\left(\frac{(b-a)}{n}\left(\frac{\left(f(a)+f(b)\right)}{2}+\sum_{i=1}^{n-1} f\left(\frac{(b-a)}{n}\cdot(i)\right)\right)\right)$$

$= 0.76$

5

Метод Симпсона:

6

$$\left(\frac{(b-a)}{3n}\left(f(a)+f(b)+4\sum_{i=1}^{\frac{(n-1)}{2}} f\left(\frac{(b-a)}{n}\cdot(2i-1)\right)+2\sum_{i=1}^{\frac{(n-1)}{2}-1} f\left(\frac{(b-a)}{n}\cdot(2i)\right)\right)\right)$$

$= 0.666666666667$

$\delta_{\text{Ньютона}-\text{Котеса}} = 0$

$\delta_{\text{Симпсона}} = 0$

$\delta_{\text{средних прямоугольников}} = 0.07$

$\delta_{\text{трапеций}} = 0.14$

**Вывод**

В ходе выполнения лабораторной работы я научился использовать формулу Ньютона-Котеса, изучил частные случаи её применения, повторил, как брать неопределенные интегралы от полинома.