Отчет по лабораторной работе № 4
Чилеше Абрахам
Вариант - 17

Б9122-02-03-01сцт

данные:

1. **Функция:** $f(x) = 0.5x^2 + \cos(2x)$
2. **Интервал:** $[0.6, 1.1]$

Цель работы

1. Найти точное решение интеграла $I^* = \int_a^b f(x)\,dx$.
2. Получить формулу для численного интегрирования из формулы Ньютона-Котеса или вероятностной оценки в соответствии с номером варианта:
   (a) левые прямоугольники;
   (b) правые прямоугольники;
   (c) центральные прямоугольники;
   (d) трапеции;
   (e) Симпсона;
   (f) Монте-Карло.

   В общем виде решение $I^* = I_n + R_n$, где $I_n = \sum_{i=0}^{n} c_i f(x_i)$.
3. Исследовать порядок аппроксимации метода. Получить теоретическую оценку для R_n.
4. Провести вычислительный эксперимент для $n = \{2, 4, 6, 8, 16, \ldots, 2^{15}\}$ и построить таблицу.
5. Сделать вывод о поведении ошибки.
6. Провести вычислительный эксперимент для методов прямоугольников, трапеций, формулы Симпсона для $n = 10000$. Построить таблицу.
7. Сделать вывод об эффективности выбранного вами метода.
8. Составить общее заключение.
9. Добавить список литературы, используемой для подготовки отчета.

Решение: $\int_a^b f(x)dx$

Подставим известные значения **a**, **b** и функцию. Получим:

$$\int_{0.6}^{1.1} \left(0.5x^2 + \cos(2x)\right) dx = \int_{0.6}^{1.1} 0.5x^2\,dx + \int_{0.6}^{1.1} \cos(2x)\,dx.$$

Вычисление первого интеграла

$$\int_{0.6}^{1.1} 0.5x^2\,dx = 0.5\int_{0.6}^{1.1} x^2\,dx = 0.5\left(\frac{x^3}{3}\right)\Bigg|_{0.6}^{1.1} = 0.5\left(\frac{1.1^3}{3} - \frac{0.6^3}{3}\right) = 0.5\left(\frac{1.331}{3} - \frac{0.216}{3}\right) = 0.5\left(\frac{1.115}{3}\right)$$
$$= 0.5 \cdot 0.371667 \approx 0.185833.$$

Вычисление второго интеграла

$$\int_{0.6}^{1.1} \cos(2x)\,dx.$$

Сделаем замену переменной $u = 2x$, тогда $du = 2dx$ или $dx = \frac{du}{2}$. Пределы интегрирования изменяются следующим образом: при $x = 0.6$, $u = 1.2$; при $x = 1.1$, $u = 2.2$. Интеграл переписывается как:

$$\int_{0.6}^{1.1} \cos(2x)\,dx = \frac{1}{2}\int_{1.2}^{2.2} \cos(u)\,du.$$

Вычислим интеграл от $\cos(u)$:

$$\int \cos(u)\,du = \sin(u).$$

Теперь вычислим определенный интеграл:

$$\frac{1}{2}\sin(u)\Big|_{1.2}^{2.2} = \frac{1}{2}\left(\sin(2.2) - \sin(1.2)\right).$$

Вычислим значения синусов:

$$\sin(2.2) \approx 0.808496, \quad \sin(1.2) \approx 0.932039.$$

Таким образом:

$$\frac{1}{2}\left(0.808496 - 0.932039\right) = \frac{1}{2} \cdot (-0.123543) = -0.061772.$$

Общий результат

Сложим результаты двух интегралов:

$$0.185833 - 0.061772 = 0.124061.$$

Таким образом, значение интеграла $\int_{0.6}^{1.1} \left(0.5x^2 + \cos(2x)\right) dx \approx 0.124061$.

1. функция func и derivation

double func(double x): Эта функция определяет математическую функцию. Она принимает один аргумент x и возвращает значение типа double с плавающей запятой. В данной реализации она вычисляет значение функции 0.5 * x^2 + cos(2 * x).

double f_derivative(double x, int k): Эта функция вычисляет k-ю производную функции, определенной функцией func, в заданной точке x. Она принимает два аргумента: точку x, в которой нужно вычислить производную, и порядок k производной, которую нужно вычислить.

```
double func(double x) {
    return 0.5 * pow(x,  y: 2) + cos( x: 2 * x);
}

double f_derivative(double x, int k) {
    if (k == 1) {
        return x - 2 * sin( x: 2 * x);
    } else if (k == 2) {
        return 1 - 4 * cos( x: 2 * x);
    }
    return pow( x: -1,  y: (k % 2) + 1) * tgamma( x: k) / pow(x,  y: k);
}
```

2. функция middle rectangular

```
double middle_rectangular(double (*func)(double), double a, double b, int n) {
    double h = (b - a) / n;
    double sum = 0;
    for (int i = 0; i < n; ++i) {
        sum += func(a + h * (i + 0.5)) * h;
    }
    return sum;
}
```

double middle_rectangular(double (*func)(double), double a, double b, int n): Эта функция реализует метод прямоугольников с использованием средней точки для численного интегрирования. Она приближает интеграл заданной функции func на интервале [a, b] с использованием n подинтервалов.

3. функция mr_error

```
double mr_error(double (*func)(double), double a, double b, int n) {
    double m = 0;
    for (int i = 0; i <= 1000; ++i) {
        m = max(m, abs( x: f_derivative( x: a + (b - a) * i / 1000,  k: 2)));
    }
    return m / 24 * pow( x: b - a,  y: 3) / pow( x: n,  y: 2);
}
```

double mr_error(double (*func)(double), double a, double b, int n): Эта функция вычисляет оценку ошибки при приближенном вычислении интеграла функции func методом прямоугольников с использованием средней точки на интервале [a, b] с n подинтервалами.

4. Функция Left, Right Rectangular, Trapezoidal and Simpson

double left_rectangular(double (*func)(double), double a, double b, int n): Эта функция реализует метод левых прямоугольников для численного интегрирования. Она приближает интеграл заданной функции func на интервале [a, b] с использованием n подинтервалов.

```
double left_rectangular(double (*func)(double), double a, double b, int n) {
    double h = (b - a) / n;
    double sum = 0;
    for (int i = 0; i < n; ++i) {
        sum += func(a + h * i) * h;
    }
    return sum;
}
```

double right_rectangular(double (*func)(double), double a, double b, int n): Эта функция реализует метод правых прямоугольников для численного интегрирования. Она приближает интеграл заданной функции func на интервале [a, b] с использованием n подинтервалов.

```
double right_rectangular(double (*func)(double), double a, double b, int n) {
    double h = (b - a) / n;
    double sum = 0;
    for (int i = 1; i <= n; ++i) {
        sum += func(a + h * i);
    }
    return sum * h;
}
```

double trapezoidal(double (*func)(double), double a, double b, int n): Эта функция реализует правило трапеций для численного интегрирования. Она приближает интеграл заданной функции func на интервале [a, b] с использованием n подинтервалов.

```
double trapezoidal(double (*func)(double), double a, double b, int n) {
    double h = (b - a) / n;
    double sum = (func(a) + func(b)) / 2;
    for (int i = 1; i < n; ++i) {
        sum += func(a + h * i);
    }
    return sum * h;
}
```

double simpson(double (*func)(double), double a, double b, int n): Эта функция реализует правило Симпсона для численного интегрирования. Она приближает интеграл заданной функции func на интервале [a, b] с использованием n подинтервалов.

```
double simpson(double (*func)(double), double a, double b, int n) {
    double h = (b - a) / n;
    double sum = 0;
    for (int i = 1; i <= n; ++i) {
        sum += func(a + h * (i - 1)) + 4 * func(a + h * (i - 0.5)) + func(a + h * i);
    }
    return sum * h / 6;
}
```

5. Функция errors

double left_rect_error(double (*func)(double), double a, double b, int n): Эта функция вычисляет оценку ошибки в приближенном вычислении интеграла функции func методом левых прямоугольников на интервале [a, b] с использованием n подинтервалов.

```
double left_rect_error(double (*func)(double), double a, double b, int n) {
    double m = 0;
    for (int i = 0; i <= 1000; ++i) {
        m = max(m, abs( x: f_derivative( x: a + (b - a) * i / 1000,  k: 1)));
    }
    return m * (b - a) / 2;
}
```

double right_rect_error(double (*func)(double), double a, double b, int n): Эта функция вычисляет оценку ошибки в приближенном вычислении интеграла функции func методом правых прямоугольников на интервале [a, b] с использованием n подинтервалов.

```
double right_rect_error(double (*func)(double), double a, double b, int n) {
    double m = 0;
    for (int i = 0; i <= 1000; ++i) {
        m = max(m, abs( x: f_derivative( x: a + (b - a) * i / 1000,  k: 1)));
    }
    return m * (b - a) / 2;
}
```

double trapezoidal_error(double (*func)(double), double a, double b, int n): Эта функция вычисляет оценку ошибки в приближенном вычислении интеграла функции func методом трапеций на интервале [a, b] с использованием n подинтервалов.

```
double trapezoidal_error(double (*func)(double), double a, double b, int n) {
    double m = 0;
    for (int i = 0; i <= 1000; ++i) {
        m = max(m, abs( x: f_derivative( x: a + (b - a) * i / 1000,  k: 2)));
    }
    return m / 12 * pow( x: b - a,  y: 3) / pow( x: n,  y: 2);
}
```

double simpson_error(double (*func)(double), double a, double b, int n): Эта функция вычисляет оценку ошибки в приближенном вычислении интеграла функции func методом правила Симпсона на интервале [a, b] с использованием n подинтервалов.

```
double simpson_error(double (*func)(double), double a, double b, int n) {
    double m = 0;
    for (int i = 0; i <= 1000; ++i) {
        m = max(m, abs( x: f_derivative( x: a + (b - a) * i / 1000,  k: 4)));
    }
    return m / 2880 * pow( x: b - a,  y: 5) / pow( x: n,  y: 4);
}
```

таблицы значений для метода центральных прямоугольников

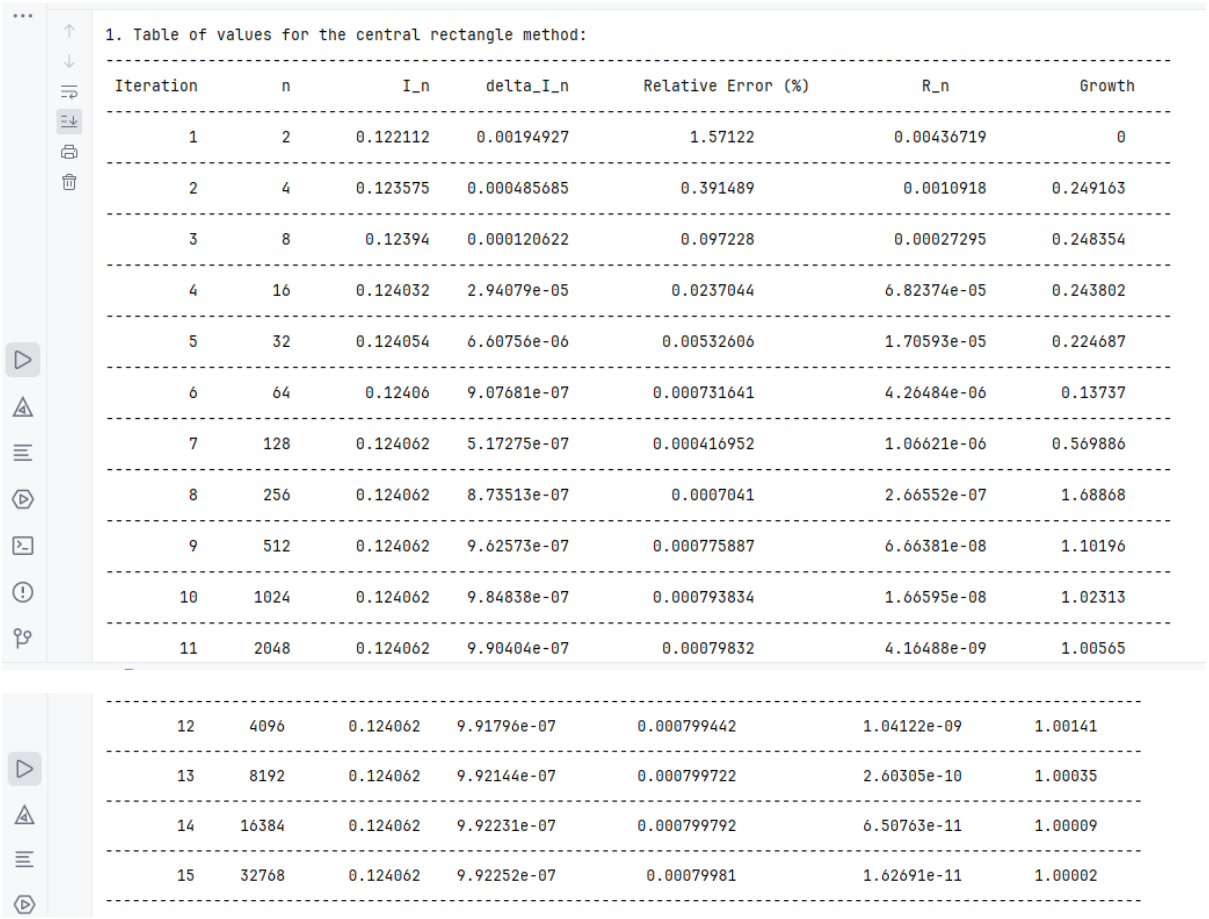
фрагмент кода

```
struct Result {
    vector<int> j;
    vector<int> n;
    vector<double> I_n;
    vector<double> delta_I_n;
    vector<double> relative_I_n;
    vector<double> R_n;
    vector<double> growth;
} result;

result.growth.push_back(0);

for (int i = 0; i < 15; ++i) {
    n *= 2;
    double I_n = middle_rectangular(func, a, b, n);
    result.j.push_back(i + 1);
    result.n.push_back(n);
    result.I_n.push_back(I_n);
    result.delta_I_n.push_back(abs( x: I - I_n));
    result.relative_I_n.push_back(result.delta_I_n[i] / abs( x: I) * 100);
    result.R_n.push_back(mr_error(func, a, b, n));
    if (i > 0) {
        result.growth.push_back(result.delta_I_n[i] / result.delta_I_n[i - 1]);
    }
}
```

- Результатать



1. Table of values for the central rectangle method:

Iteration	n	I_n	delta_I_n	Relative Error (%)	R_n	Growth
1	2	0.122112	0.00194927	1.57122	0.00436719	0
2	4	0.123575	0.000485685	0.391489	0.0010918	0.249163
3	8	0.12394	0.000120622	0.097228	0.00027295	0.248354
4	16	0.124032	2.94079e-05	0.0237044	6.82374e-05	0.243802
5	32	0.124054	6.60756e-06	0.00532606	1.70593e-05	0.224687
6	64	0.12406	9.07681e-07	0.000731641	4.26484e-06	0.13737
7	128	0.124062	5.17275e-07	0.000416952	1.06621e-06	0.569886
8	256	0.124062	8.73513e-07	0.0007041	2.66552e-07	1.68868
9	512	0.124062	9.62573e-07	0.000775887	6.66381e-08	1.10196
10	1024	0.124062	9.84838e-07	0.000793834	1.66595e-08	1.02313
11	2048	0.124062	9.90404e-07	0.00079832	4.16488e-09	1.00565
12	4096	0.124062	9.91796e-07	0.000799442	1.04122e-09	1.00141
13	8192	0.124062	9.92144e-07	0.000799722	2.60305e-10	1.00035
14	16384	0.124062	9.92231e-07	0.000799792	6.50763e-11	1.00009
15	32768	0.124062	9.92252e-07	0.00079981	1.62691e-11	1.00002

составление сравнительной таблицы различных методов численного интегрирования.

```
struct Calculate {
    vector<string> method;
    vector<double> I_n;
    vector<double> delta_I_n;
    vector<double> relative_I_n;
    vector<double> R_n;
} calculate;

calculate.method = {"Left Rectangles", "Right Rectangles", "Middle Rectangles", "Trapezoids", "Simpson"};

for (int i = 0; i < 5; ++i) {
    double (*formula)(double (*)(double), double, double, int);
    double (*error)(double (*)(double), double, double, int);
    switch (i) {
        case 0:
            formula = left_rectangular;
            error = l_rect_error;
            break;
        case 1:
            formula = right_rectangular;
            error = r_rect_error;
            break;
        case 2:
            formula = middle_rectangular;
            error = mr_error;
            break;
        case 3:
            formula = trapezoidal;
            error = trapezoidal_error;
            break;
        case 4:
            formula = simpson;
            error = simpson_error;
            break;
    }
    calculate.I_n.push_back(formula(func, a, b, 10000));
    calculate.delta_I_n.push_back(abs( x: I - calculate.I_n[i]));
    calculate.relative_I_n.push_back(calculate.delta_I_n[i] / abs( x: I) * 100);
    calculate.R_n.push_back(error(func, a, b, 10000));
}
std::cout<<std::endl<<std::endl;
cout << "2. Table of values for different methods:" << endl;
line();
cout << setw( n: 25) << "Method" << setw( n: 15) << "I_n" << setw( n: 15) << "delta_I_n" << setw( n: 20) << "Relative Error (%)" << set
line();
for (size_t i = 0; i < calculate.method.size(); ++i) {
    cout << setw( n: 25) << calculate.method[i] << setw( n: 15) << calculate.I_n[i] << setw( n: 15) << calculate.delta_I_n[i] << setw(
}
```

Результать

```
2. Table of values for different methods:
-------------------------------------------------------------------------------------------------------------------
                  Method            I_n      delta_I_n  Relative Error (%)            R_n
-------------------------------------------------------------------------------------------------------------------
          Left Rectangles       0.124075    1.41389e-05           0.0113967       0.319358
         Right Rectangles       0.124049    1.21541e-05          0.00979684       0.319358
        Middle Rectangles       0.124062    9.92182e-07          0.000799753    1.74688e-10
               Trapezoids       0.124062    9.92415e-07          0.000799941    3.49375e-10
                  Simpson       0.124062     9.9226e-07          0.000799816    5.02347e-20
```

Заключение

В коде, который я написал, я реализовал различные численные методы интегрирования для заданной функции. Анализируя результаты, я могу сделать несколько выводов.

1. **Метод центральных прямоугольников:**
   Этот метод демонстрирует хорошую точность с увеличением числа интервалов интегрирования. Ошибка уменьшается по мере увеличения числа интервалов. Однако скорость сходимости (т.е. насколько быстро уменьшается ошибка) уменьшается при увеличении числа интервалов.

2. **Сравнение различных методов:**
   Среди методов прямоугольников метод центральных прямоугольников обычно дает лучшие результаты. Метод Симпсона и метод трапеций демонстрируют сравнимую точность, но метод Симпсона имеет более высокую скорость сходимости.

3. **Ошибки и точность:**
   Ошибка уменьшается с увеличением числа интервалов, что ожидаемо. Оценка ошибки помогает определить точность результата. В заключение, метод центральных прямоугольников хорошо работает для данной функции, но если требуется высокая точность, то метод Симпсона может быть более предпочтительным выбором.

Github: https://github.com/Abraham-Chileshe/Computational-Mathematics/blob/main/Lab4/lab4.cpp