Практическое занятие №5

Армбристер Никита Владиславович

Группа ПМИ-31

Вариант 1

Цель: сформировать практические навыки применения правила Рунге для оценки численного интегрирования и уточнения по Ричардсону для повышения точности решения прикладных задач

Задание:

1. Разработать класс, реализующий дюбые две схемы численного интегрирования семейства Ньютона-Котеса и Гаусса с числом узлов интегрирования не менее двух

2. Задайте на отрезке [a, b] непрерывную полиноминальную функцию φ(x)

3. Вычислить аналитически $I^{¿} = \int_a^b (x)\,dx$

4. Для отрезка [a, b] постройте три вложенные сетки с равномерным шагом h, h/2 и h/4. Для каждой из реализованных схем численного интегрирования выполните оценку порядка аппроксимации относительно шага равномерного сеточного разбиения.

5. Для каждой квадратурной формулы заполнить следующую таблицу (все значения приводить в экспоненциальной форме)

h	$I^{¿} - I^{h}$	$\frac{I^{¿} - I^{h}}{I^{¿} - I^{h/2}}$	$\frac{I^{h/2} - I^{h}}{2^{k} - 1}$	I^{R}	$I^{¿} - I^{R}$

Где I^{h} - вычисленное значение интеграла на равномерной сетке с шагом h, k – порядок малости остаточного члена (аналитический порядок аппроксимации квадратурной формулы)

6) С помощью AI-ассистента DeepSeek решите предыдущую задачу о выборе схемы численного интегрирования максимлаьной точности. Какие явные недостатки можно обнаружить?

Промт. Оптимальный метод численного интегрирования Int{exp(x)}dx x in [0,1] с максимальной точностью

Вариант: a=0.04; b=0.6

Ссылка на репозиторий с проектом: https://github.com/Gribnik24/NumericalMethods-PracticalTasks/tree/main/Task%20%232.5%20Simpson%20and%20Gauss%20Integrator

Решение:

1) Разработаем на языке C++ класс Integrator, который реализует метод парабол и квадратуру Гаусса-2:

```cpp
class Integrator {
private:
    double a, b;
    int N;

public:
    Integrator(double a_val, double b_val, int segments = 1)
        : a(a_val), b(b_val), N(segments) {
    }

    double simpson(double (*f)(double)) {
        double h = (b - a) / N;
        double sum = f(a) + f(b);

        for (int i = 1; i < N; i += 2) {
            sum += 4.0 * f(a + i * h);
        }
        for (int i = 2; i < N; i += 2) {
            sum += 2.0 * f(a + i * h);
        }

        return h * sum / 3.0;
    }

    double gauss(double (*f)(double), int n = 2) {
        double h_k = (b - a) / N;
        vector<double> xi, alpha_i;

        if (n == 2) {
            xi = { -1.0 / sqrt(3.0), 1.0 / sqrt(3.0) };
            alpha_i = { 1.0, 1.0 };
        }

        double sum = 0.0;
        for (int k = 0; k < N; k++) {
            double x_start = a + k * h_k;
            double inner_sum = 0.0;

            for (int i = 0; i < n; i++) {
                double x = h_k * (xi[i] + 1.0) / 2.0 + x_start;
                inner_sum += alpha_i[i] * f(x);
            }
            sum += h_k * inner_sum;
        }

        return sum / 2.0;
    }
};
```

2) Зададим функцию $(x)=x^4+x^3-2x^2+5x-3$
3) Аналитически вычислим интеграл

$$I^{¿}=\int_{0.04}^{0.6} x^4+x^3-2x^2+5x-3\,dx=\left(\frac{x^5}{5}+\frac{x^4}{4}-\frac{2x^3}{3}+\frac{5x^2}{2}-3x\right)\Bigg|_{0.04}^{0.6}=¿$$

$$¿\frac{0.6^5}{5}+\frac{0.6^4}{4}-\frac{2¿0.6^3}{3}+\frac{5¿0.6^2}{2}-3*0.6-\frac{0.04^5}{5}-\frac{0.04^4}{4}+\frac{2¿0.04^3}{3}-\frac{5¿0.04^2}{2}+3*0.04\approx$$

$$\approx-0.880006$$

Также для дальнейшего удобства подсчёта формула распишем данную операцию и в коде:

```
// Аналитическое значение интеграла
double analytical_integral(double a, double b) {
    auto F = [](double x) {
        return pow(x, 5) / 5.0 + pow(x, 4) / 4.0 - 2.0 * pow(x, 3) / 3.0 + 5.0 * pow(x, 2) / 2.0 - 3.0 * x;
        };
    return F(b) - F(a);
}
```

4) Для отрезка построим три вложенные сетки с равномерным шагом h, h/2, h/4

```
    // Три вложенные сетки
    vector<int> ns = { 2, 4, 8 };  // h, h/2, h/4
    vector<double> hs;
    for (int n : ns) {
        hs.push_back((b - a) / n);
    }
```

Оценку порядка аппроксимации относительно шага выполним по формуле:

$$k \approx \log\left(\left|1+\frac{I^{h/2}-I^{h}}{I^{h/4}-I^{h/2}}\right|\right)$$

```
// Оценка порядка
double estimate_order(double Ih, double Ih2, double I_s) {
    double numerator = Ih2 - Ih;
    double denominator = I_s - Ih2;

    if (abs(denominator) < 1e-15) {
        return 0.0;
    }

    double ratio = numerator / denominator;
    double value = 1.0 + ratio;

    if (value <= 0) {
        return 0.0;
    }

    return log2(abs(value));
}
```

Получаем k=4:

```
ОЦЕНКА ПОРЯДКА СИМПСОНА:
h и h/2: k = 4.00
h/2 и h/4: k = 4.00
```

```
ОЦЕНКА ПОРЯДКА ГАУССА-2:
h и h/2: k = 4.00
h/2 и h/4: k = 4.00
```

5) На основе результатов вывода программы заполним таблицу

```
АНАЛИТИЧЕСКОЕ ЗНАЧЕНИЕ ИНТЕГРАЛА: I* = -8.800060e-01
==========================================

МЕТОД СИМПСОНА:
==========================================
h = 0.28 | I* - I^h = -4.59e-04
h = 0.14 | I* - I^h = -2.87e-05
h = 0.07 | I* - I^h = -1.79e-06

ОЦЕНКА ПОРЯДКА СИМПСОНА:
h и h/2: k = 4.00
h/2 и h/4: k = 4.00

ДАННЫЕ ДЛЯ ТАБЛИЦЫ - СИМПСОН:
h=0.28: I*-I^h = -4.59e-04
h=0.14: I*-I^h = -2.87e-05
h=0.07: I*-I^h = -1.79e-06
(I*-I^h)/(I*-I^{h/2}) для h=0.28 h/2=0.14: 1.60e+01
(I*-I^h)/(I*-I^{h/2}) для h=0.14  h/2=0.07: 1.60e+01
(I^{h/2}-I^h)/(2^k-1) для h=0.28 h/2=0.14: -2.87e-05
(I^{h/2}-I^h)/(2^k-1) для h=0.14  h/2=0.07: -3.05e-05
I^R для h=0.28 h/2=0.14: -8.80e-01
I*-I^R для h=0.28 h/2=0.14: 1.11e-16
I^R для h=0.14 h/2=0.07: -8.80e-01
I*-I^R для h=0.14 h/2=0.07: -1.11e-16

МЕТОД ГАУССА-2:
==========================================
h = 0.28 | I* - I^h = 1.91e-05
h = 0.14 | I* - I^h = 1.20e-06
h = 0.07 | I* - I^h = 7.47e-08

ОЦЕНКА ПОРЯДКА ГАУССА-2:
h и h/2: k = 4.00
h/2 и h/4: k = 4.00

ДАННЫЕ ДЛЯ ТАБЛИЦЫ - ГАУСС-2:
h=0.28: I*-I^h = 1.91e-05
h=0.14: I*-I^h = 1.20e-06
h=0.07: I*-I^h = 7.47e-08
(I*-I^h)/(I*-I^{h/2}) для h=0.28 h/2=0.14: 1.60e+01
(I*-I^h)/(I*-I^{h/2}) для h=0.14  h/2=0.07: 1.60e+01
(I^{h/2}-I^h)/(2^k-1) для h=0.28 h/2=0.14: 1.20e-06
(I^{h/2}-I^h)/(2^k-1) для h=0.14  h/2=0.07: 1.27e-06
I^R для h=0.28 h/2=0.14: -8.80e-01
I*-I^R для h=0.28 h/2=0.14: 0.00e+00
I^R для h=0.14 h/2=0.07: -8.80e-01
I*-I^R для h=0.14 h/2=0.07: 0.00e+00
```

	h	$I^{¿} - I^{h}$	$\frac{I^{¿} - I^{h}}{I^{¿} - I^{h/2}}$	$\frac{I^{h/2} - I^{h}}{2^{k} - 1}$	I^{R}	$I^{¿} - I^{R}$
Симпсон	0.28	-4.59e-04	1.60e+01	-2.87e-05	-8.80e-01	1.11e-16
	0.14	-2.87e-05	1.60e+01	-3.05e-05	-8.80e-01	-1.11e-16
	0.07	-1.79e-06	-	-	-	-
Гаусс-2	0.28	1.91e-05	1.60e+01	1.20e-06	-8.80e-01	0.00e+00
	0.14	1.20e-06	1.60e+01	1.27e-06	-8.80e-01	0.00e+00
	0.07	7.47e-08	-	-	-	-

6) Решим предложенную задачу с помощью DeepSeek

Оптимальный метод численного интегрирования Int{x^4+x^3-2x^2+5x-3}dx x in [0.04,0.6] с максимальной точностью

Для вычисления интеграла $\int_{0.04}^{0.6}(x^4 + x^3 - 2x^2 + 5x - 3)dx$ с максимальной точностью я рекомендую использовать **адаптивные методы**, в частности **адаптивный метод Симпсона**.

Оптимальное решение:

1. Аналитическое решение (для сравнения)

Первообразная:

$$F(x) = \frac{x^5}{5} + \frac{x^4}{4} - \frac{2x^3}{3} + \frac{5x^2}{2} - 3x$$

Точное значение:

$$\int_{0.04}^{0.6} f(x)dx = F(0.6) - F(0.04) \approx -1.330752$$

2. Адаптивный метод Симпсона (рекомендуемый)

python

Копировать Скачать

```python
def adaptive_simpson(f, a, b, tol=1e-12, max_depth=50):
    def _simpson(f, a, b):
        c = (a + b) / 2
        h = (b - a) / 6
        return h * (f(a) + 4*f(c) + f(b))

    def _adaptive(f, a, b, tol, depth):
        c = (a + b) / 2
        whole = _simpson(f, a, b)
        left = _simpson(f, a, c)
        right = _simpson(f, c, b)

        if abs(whole - left - right) < 15*tol or depth >= max_depth:
            return left + right + (left + right - whole)/15
        else:
            return (_adaptive(f, a, c, tol/2, depth+1) +
                    _adaptive(f, c, b, tol/2, depth+1))

    return _adaptive(f, a, b, tol, 0)

# Применение
f = lambda x: x**4 + x**3 - 2*x**2 + 5*x - 3
result = adaptive_simpson(f, 0.04, 0.6)
# Результат: -1.330752000000000
```

3. Квадратурные формулы Гаусса

Для достижения максимальной точности можно использовать квадратуру Гаусса высокого порядка (например, 10-20 узлов):

python Копировать Скачать

```python
from scipy import integrate
result, error = integrate.quadrature(f, 0.04, 0.6, maxiter=100, tol=1e-14)
# Точность ~1e-15
```

Сравнение методов:

- **Адаптивный Симпсон:** точность ~1e-12, надежный
- **Квадратура Гаусса:** точность ~1e-15, максимальная точность
- **Метод Ромберга:** точность ~1e-14, быстрая сходимость

Рекомендация:

Для максимальной точности используйте **квадратурные формулы Гаусса высокого порядка** или **метод Ромберга** из библиотеки SciPy:

python Копировать Скачать

```python
from scipy import integrate
result = integrate.romberg(f, 0.04, 0.6, tol=1e-15)
```

Это обеспечит точность порядка 10^{-15} при минимальных вычислительных затратах.

DeepSeek сначала решил интеграл аналитически, но сделал это неправильно. Затем он предложил несколько реализаций на языке Python – ручной и методы из библиотеки Scipy. Если проверить данные методы, то они сработают верно.

```python
def adaptive_simpson(f, a, b, tol=1e-12, max_depth=50):
    def _simpson(f, a, b):
        c = (a + b) / 2
        h = (b - a) / 6
        return h * (f(a) + 4*f(c) + f(b))

    def _adaptive(f, a, b, tol, depth):
        c = (a + b) / 2
        whole = _simpson(f, a, b)
        left = _simpson(f, a, c)
        right = _simpson(f, c, b)

        if abs(whole - left - right) < 15*tol or depth >= max_depth:
            return left + right + (left + right - whole)/15
        else:
            return (_adaptive(f, a, c, tol/2, depth+1) +
                    _adaptive(f, c, b, tol/2, depth+1))

    return _adaptive(f, a, b, tol, 0)

# Применение
f = lambda x: x**4 + x**3 - 2*x**2 + 5*x - 3
result = adaptive_simpson(f, 0.04, 0.6)
# Результат: -1.330752000000000

result
```

```
-0.8800059938133332
```

```python
from scipy import integrate
result, error = integrate.quadrature(f, 0.04, 0.6, maxiter=100, tol=1e-14)
# Точность ~1e-15
print(result, error, sep='\n')
```

```
-0.8800059938133333
0.0
C:\Temp\ipykernel_22076\1572976635.py:2: DeprecationWarning: `scipy.integrate.quadrature` is deprecated as of SciPy 1.12.0and will be removed in SciPy 1.15.0. Please use`scipy.integrate.quad` instead.
  result, error = integrate.quadrature(f, 0.04, 0.6, maxiter=100, tol=1e-14)
```

```python
from scipy import integrate
result = integrate.romberg(f, 0.04, 0.6, tol=1e-15)
result
```

```
C:\Temp\ipykernel_22076\4077458029.py:2: DeprecationWarning: `scipy.integrate.romberg` is deprecated as of SciPy 1.12.0and will be removed in SciPy 1.15.0. Please use`scipy.integrate.quad` instead.
  result = integrate.romberg(f, 0.04, 0.6, tol=1e-15)
-0.8800059938133332
```

Далее он привел точности данных методов, пометив их как «сравнение», но вот только эти точности задаются пользователем в методах, поэтому я считаю такое сравнение неуместным

Вывод

В ходе данной лабораторной работы был создан класс на языке C++, который позволяет считать интегралы с использованием методов Симпсона и Гаусса-2. Оба эти метода обеспечивают 4 порядок аппроксимации, что было проверено внутри кода. Была создана и просчитана с использованием обоих методов полиноминальная функция 4 степени. Для нее были найдены прилиженне значения с разным шагом h и уточнения по Ричардсону

В свою очередь DeeepSeek подсказал несколько оптимальных решений на языке Python, но его ответ я считаю в корне структурно неверным, поскольку фактически он сначала привел неверное аналитическое решение, затем попытался подогнать программное решение под свой прошлый ответ. Сравнение погрешностей и принятие решения о выборе лучшего метода было также основано на пользовательском параметре методов из библиотеки, что является не сильно конструктивным