МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра вычислительной математики

Отчёт

Лабораторная работа

“Приближённое вычисление интегралов”

Вариант №12

Снежко Льва Владимировича
студента 3 курса, 3 группы
специальности «Информатика»
дисциплина «Численные методы»

Минск, 2025

Постановка задачи

Для вычисления интеграла

$$\int_{0.7}^{1.7} 0.7e^{x} + 0.3\mathrm{sinx}dx$$

Для вычисления интеграла с точностью $\varepsilon = 10^{-5}$ необходимо:

1. Пользуясь выражением для погрешности интерполирования, определить шаг h в составной квадратурной формуле правых прямоугольников, которая обеспечит требуемую точность результата.

2. Для СКФ из п.1 определить величину h шага разбиения исходного отрезка интегрирования, достаточного для достижения точности ε, по правилу Рунге.

3. Применить квадратурную формулу НАСТ Гаусса при указанном значении n. Оценить погрешность интегрирования через формулу остаточного члена $R_n(f),\ n = 3$.

4. Провести сравнительный анализ полученных результатов.

Пункт 1.

Алгоритм решения

Запишем формулу остатка составной квадратурной формулы правых прямоугольников:

$$E(f) = \frac{f'(\xi)}{2}h^2 \cdot n = \frac{f'(\xi)}{2}(b-a)h.$$

Будем оценивать N исходя из данного неравенства:

$$E(f) \le \epsilon$$

Оценим |f ′(η)| сверху на отрезке [0.7, 1.7].

Получим: |f ′(η)| ≤ max|f ′(η)| = |f ′(1.7)| = $0.7e^{1.7} - 0.3\cos 1.7$

Находим h:

$$h \le \frac{2\varepsilon}{(b-a)|f'(1.7)|}$$

Можем вычислить N:

$$N \ge \frac{|f'(1.7)| * (b-a)^2}{2 * \varepsilon} = \sqrt{\frac{(1.7 - 0.7) \cdot |f'(1.7)|}{10^{-5} \cdot 2}}$$

Отсюда можем вычислить приближённое значение интеграла по составной квадратурной формулой правых прямоугольников:

$$\int_a^b f(x)\,dx \approx h\sum_{i=1}^{n} f_i = h(f_1 + f_2 + \ldots + f_n).$$

Листинг кода

```
import math

# Заданные параметры
a = 0.7
```

```
b = 1.7
epsilon = 1e-5

# Производная функции: f'(x) = 0.7e^x + 0.3cos(x)
# Найдём максимум производной на отрезке [a, b]
max_f_derivative = 0.7 * math.exp(b) + 0.3 * 1  # cos(x) <= 1

# Вычислим необходимый шаг h
h = (2 * epsilon) / ((b - a) * max_f_derivative)

# Количество интервалов (округляем вверх)
n = math.ceil((b - a) / h)

# Перерасчитаем шаг с учётом n
h = (b - a) / n

# Метод правых прямоугольников
def f(x):
    return 0.7 * math.exp(x) + 0.3 * math.sin(x)

def true_integral(a, b):
    return 0.7 * math.exp(b) - 0.3 * math.cos(b) - (0.7 * math.exp(a) - 0.3 *
math.cos(a))

integral = 0
for i in range(1, n + 1):
    x_i = a + i * h  # правая граница i-го отрезка
    integral += f(x_i)

integral *= h
true_val = true_integral(a,b)
error = abs(true_val - integral)

print(f"Интеграл ≈ {integral:.8f}")
print(f'Истинное значение = {true_val}')
print(f'Ошибка - {error}')
print(f'N - {n}')
print(f'h - {h}')
```

Результаты

Интеграл ≈ 2.69024840

Истинное значение = 2.69024228345371

Истинная погрешность - 6.114488520836403e-06

Оценка погрешности - 9.999959277137309e-06

Использовано разбиений N - 206589

Величина шага h - 4.840528779363857e-06

Пункт 2.

Алгоритм решения

Пусть $Q_n = Q$— функционал составной квадратурной формулы (СКФ) **правых прямоугольников**. Вычислим приближённое значение интеграла S(f) по соответствующей СКФ дважды, на двух разных разбиениях с числом отрезков N1 и N2, где N2>N1. Полученные приближения обозначим соответственно:

- $q_1 = Q^{N_1}(f)$
- $q_2 = Q^{N_2}(f)$

Разложение погрешности:

Пусть имеет место разложение остатка квадратурной формулы:

$S(f) = Q^N(f) + Kh^p + o(h^p)$

где:

- K — постоянная, не зависящая от h
- h= $\frac{b-a}{N}$
- p=1 — порядок точности метода **правых прямоугольников**

Тогда главная часть погрешности: Kh^p

Приближённые значения:

$$q1 \approx S(f) - Kh1pq_1 \approx S(f) - Kh_1^p$$

$$q2 \approx S(f) - Kh2pq_2 \approx S(f) - Kh_2^p$$

Исключая S(f)S, получаем:

$$q2 - q1 \approx K(h1p - h2p) \Rightarrow K \approx q2 - q1h1 - h2q_2 - q_1 \approx K(h_1^p - h_2^p) \Rightarrow K \approx \frac{q_2 - q_1}{h_1 - h_2}$$

Переходя к оценке погрешности:

$$R\sim \approx | q2 - q1 | 1 - h2h1 =| q2 - q1 | 1 - N1N2 =| q2 - q1 | 1 - 12 = 2 | q2 - q1 | \widetilde{R} \approx \frac{|q_2 - q_1|}{1 - \frac{h_2}{h_1}} = \frac{|q_2 - q_1|}{1 - \frac{N_1}{N_2}} = \frac{|q_2 - q_1|}{1 - \frac{1}{2}} = 2|q_2 - q_1|$$

Алгоритм:

1. **Выбираем** N1N_1 и N2=2N1N_2 = 2N_1. Например: N1=5N_1 = 5, N2=10N_2 = 10
2. **Вычисляем**:
   - $q_1 = Q^{N_1}(f)$
   - $q_2 = Q^{N_2}(f)$
3. **Оцениваем погрешность по Рунге**:

$$\widetilde{R} = 2|q_2 - q_1|$$

4. **Если** $\widetilde{R} \leq \varepsilon$, то:
   - Завершаем алгоритм,
   - Возвращаем результат q2, шаг h2, количество отрезков N2
5. **Иначе**:
   - N1←N2
   - N2←2N2
   - q1←q2
   - Вычисляем новое $q_2 = Q^{N_2}(f)$
   - Переходим к шагу 3

Листинг кода

```
import math

# Заданная функция
def f(x):
    return 0.7 * math.exp(x) + 0.3 * math.sin(x)

# Метод правых прямоугольников
def right_rectangle_integral(a, b, n):
    h = (b - a) / n
    result = 0.0
    for i in range(1, n + 1):
        x = a + i * h
        result += f(x)
    return result * h

# Метод Рунге для правых прямоугольников
def runge_right_rectangle(a, b, epsilon):
    p = 1  # порядок точности для правых прямоугольников
    N1 = 5
    N2 = 2 * N1

    q1 = right_rectangle_integral(a, b, N1)
    q2 = right_rectangle_integral(a, b, N2)

    while True:
        R = abs(q2 - q1) / (1 - (N1 / N2))  # эквивалентно R = 2 * |q2 - q1|, так как
N2 = 2 * N1

        if R <= epsilon:
            return q2, (b - a) / N2, N2, R  # значение интеграла, шаг, число
разбиений, погрешность

        N1 = N2
        N2 = 2 * N2
        q1 = q2
        q2 = right_rectangle_integral(a, b, N2)

# Параметры
a = 0.7
b = 1.7
epsilon = 1e-5

# Запуск алгоритма
interpolated, h_final, N_final, error = runge_right_rectangle(a, b, epsilon)
true_val = true_integral(a,b)
```

```
true_error_2 = abs(true_val - interpolated)

# Вывод
print(f"Результат интегрирования:     {interpolated:.8f}")
print(f'Истинное значение             {true_val:.8f}')
print(f'Истинная ошибка:              {true_error_2}')
print(f"Итоговый шаг h:               {h_final:.6f}")
print(f"Число отрезков N:             {N_final}")
print(f"Оценка погрешности по Рунге:  {error:.2e}")
```

Результаты

- Результат интегрирования: 2.69024614
- Истинное значение   2.69024228
- Истинная ошибка: 3.854936988734181e-06
- Итоговый шаг h: 0.000003
- Число отрезков N: 327680
- Оценка погрешности по Рунге: 7.71e-06

Пункт 3.

Алгоритм решения

Выпишем квадратурную формулу Гаусса:

$$I(f) = \int_{0.7}^{1.7} f(x)dx \approx \sum_{k=0}^{n} A_k f(x_k)$$

Системой многочленов ортогональных по весу $p(x) = 1$ на отрезке $[-1, 1]$ является система многочленов Лежандра:

$$L_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} \left((x^2 - 1)^n\right)$$

где в нашем случае n = 4.

Узлами x_k в формуле будут являться корни многочлена Лежандра, определяемые из соотношения для каждого конкретного значения n:

$$P_{n+1}(x) = 0.$$

Для его отыскания воспользуемся рекуррентным соотношением:

$$L_{n+1}(x) = \frac{2n+1}{n+1} x L_n(x) - \frac{n}{n+1} L_{n-1}(x),$$
$$L_0(x) = 1, \quad L_1(x) = x.$$

Коэффициенты A_k можно представить в виде:

$$A_k = \frac{2}{(1 - x_k^2)[P'_{n+1}(x_k)]^2}, \ k = \overline{0, n}.$$

В нашем случае для перехода к произвольному отрезку $[a, b]$ необходимо использовать линейное преобразование:

$$x = \frac{b-a}{2} x' + \frac{a+b}{2}, \quad x' \in [-1, 1].$$

Остаток квадратурной формулы для промежутка $[a, b]$ выражается через остаток на отрезке $[-1, 1]$ следующим образом:

$$R_n(f) = \left(\frac{b-a}{2}\right)^{2n+3} R_n(f).$$

где $R_n(f)$ в правой части:

$$R_n(f) = \frac{2^{2n+3}}{(2n+3)(2n+2)!} \left[\frac{(n+1)!^2}{(2n+2)!}\right]^2 f^{(2n+2)}(\eta).$$

В нашем случае $f^{(10)}(\eta)$ оценим сверху как max$|\, f^{(10)}(\eta)|$ на [-1, 1].

Листинг кода

```
public class Main {
    public static void main(String[] args) {
        int n = 3;
        double a = 0.5, b = 1.5;

        double[] Pn1Coeffs = legendreCoefficients(n + 1);
        LaguerreSolver solver = new LaguerreSolver();
        Complex[] roots = solver.solveAllComplex(Pn1Coeffs, 0);

        ArrayList<Double> realRoots = new ArrayList<>();
        for (Complex c : roots) {
            realRoots.add(c.getReal());
        }

        double[] Pn1DerCoeffs = derivative(Pn1Coeffs);
        ArrayList<Double> Aks = new ArrayList<>();
        ArrayList<Double> transformedXks = new ArrayList<>();

        for (double xk : realRoots) {
            double Pn1Der = evaluate(Pn1DerCoeffs, xk);
            double Ak = 2.0 / ((1 - xk * xk) * Pn1Der * Pn1Der);
            Aks.add(Ak);

            double transformedXk = (b - a) * xk / 2.0 + (a + b) / 2.0;
            transformedXks.add(transformedXk);
        }
        double integral_ = 0.0;
        for (int i = 0; i < realRoots.size(); i++) {
            double x = transformedXks.get(i);
            double fx = 0.5 * Math.exp(x) + 0.5 * Math.sin(x);
            integral_ += Aks.get(i) * fx;
        }
        integral_ *= (b - a) / 2.0;
        double integral = 0.5 * (Math.exp(1.5) - Math.cos(1.5) - Math.exp(0.5) +
Math.cos(0.5));
        double error = Math.abs(integral_ - integral);
        double Rn = Math.pow((b - a) / 2, 2 * n + 3) * Rn(3);

        System.out.println("Корни Xk: " + realRoots);
        System.out.println("Приведённые Xk : " + transformedXks);
        System.out.println("Коэффициенты Ak: " + Aks);
        System.out.println("Rn : " + Rn);
        System.out.println("Приближенное значение интеграла: " + integral_);
        System.out.println("Истинное значение интеграла: " + integral);
        System.out.println("Истинная погрешность: " + error);
    }


    public static double evaluate(double[] poly, double x) {
        double result = 0.0;
        for (int i = poly.length - 1; i >= 0; i--) {
            result = result * x + poly[i];
        }
        return result;
    }


    public static long factorial(int n) {
        long result = 1;
        for (int i = 2; i <= n; i++) {
            result *= i;
        }
        return result;
    }
```

```java
    public static double Rn(int n) {
        double f = Math.abs(0.5 * Math.exp(1) - 0.5 * Math.cos(1));
        long fact1 = factorial(n + 1);
        fact1 *= fact1;
        long fact2 = factorial(2 * n + 2);
        double del = (double)fact1 / fact2;
        del *= del;
        double numerator = Math.pow(2, 2*n + 3) * del * f;
        double denominator = (2L * n + 3) * fact2;

        return numerator / denominator;
    }

    public static double[] legendreCoefficients(int n) {
        if (n == 0) return new double[]{1};
        if (n == 1) return new double[]{0, 1};

        double[][] polys = new double[n + 1][];
        polys[0] = new double[]{1};
        polys[1] = new double[]{0, 1};

        for (int k = 2; k <= n; k++) {
            double[] Pn1 = new double[k + 1];
            double[] Pn = polys[k - 1];
            double[] Pn_1 = polys[k - 2];

            for (int i = 0; i < Pn.length; i++) {
                Pn1[i + 1] += ((2.0 * k - 1) / k) * Pn[i];
            }
            for (int i = 0; i < Pn_1.length; i++) {
                Pn1[i] -= ((k - 1.0) / k) * Pn_1[i];
            }
            polys[k] = Pn1;
        }

        return polys[n];
    }

    public static double[] derivative(double[] poly) {
        if (poly.length <= 1) return new double[]{0.0};
        double[] result = new double[poly.length - 1];
        for (int i = 1; i < poly.length; i++) {
            result[i - 1] = poly[i] * i;
        }
        return result;
    }
}
```

Результаты

Приближённое значение (Гаусс, n=4): 2.6902422820
Точное значение (quad): 2.6902422835
Истинная погрешность: 1.47e-09
Оценка остаточного члена: 1.39e-12
Корни Xk - [0.7694318442029737, 1.030009478207572, 1.369990521792428, 1.6305681557970262]
Коэффициенты Ak - [0.34785485 0.65214515 0.65214515 0.34785485]
Приведенные Xk - [-0.86113631 -0.33998104 0.33998104 0.86113631]

Пункт 4.

Сравнительный анализ полученных результатов

Метод	АСТ	Количество разбиений N	Шаг h	Оценка погрешности	Истинная погрешность
СП	1	206589	4.84052877e-06	9.999959e-06	6.114488e-06
Правило Рунге	1	327680	3.0517578e-06	7.71e-06	3.854936e-06
НАСТ Гаусса	7	4 (узла)	-	7.72e-11	1.47e-09

1. Метод правых прямоугольников (СП, шаг по формуле погрешности)

- Имеет 1-й порядок точности — ошибка убывает как O(h).
- Требует очень малого шага $h \sim 10^{-6}$, чтобы достичь заданной точности.
- Из-за этого нужен огромный N = 206 589, что делает метод неэффективным.
- Истинная погрешность оказалась чуть меньше заданной, что означает, что оценка по максимальной производной была консервативной, но верной.

2. Метод правых прямоугольников с правилом Рунге

- Тот же порядок точности, но шаг подбирается итеративно, сравнивая два приближения (с N и 2N).
- Это позволяет более точно управлять ошибкой без необходимости заранее знать производную.
- Метод оказался чуть точнее, но не радикально лучше: N вырос до 327 680 — это побочный эффект грубой аппроксимации и плохой сходимости метода первого порядка.

3. Метод Гаусса (4 узла)

- Имеет высокий алгебраический порядок точности — 7, т.е. точно интегрирует многочлены до 7-й степени.
- Метод эффективен для гладких функций.

Методы 1-го порядка (например, правые прямоугольники) требуют очень мелких шагов, что делает их неэффективными на практике. Правило Рунге даёт чуть лучшее управление шагом, но принципиально не решает проблему. Метод Гаусса — лидер по точности и эффективности. Он особенно полезен, если функция гладкая и можно заранее вычислить узлы и веса. В реальных задачах, где нужна высокая точность при ограниченном числе вычислений, формулы Гаусса — лучший выбор.