Университет ИТМО
МФ КТиУ, Ф ПИиКТ

Лабораторная работа №2
Дисциплина «Вычислительная математика»

Интегрирование
Метод трапеций

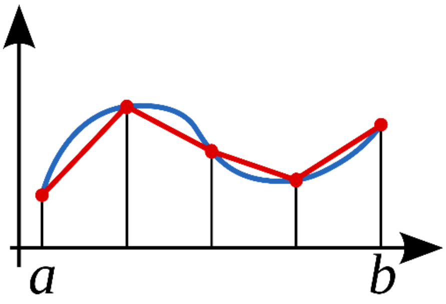
Выполнил:
Студент группы P3212
Анищенко Анатолий Алексеевич

Преподаватель:
Перл Ольга Вячеславовна

г. Санкт-Петербург
2020 г.

Описание метода

Метод трапеций – метод численного интегрирования функции от одной переменной. Суть данного метода заключается в замене на каждом элементарном отрезке подынтегральной функции на линейную функцию. Площадь под графиком функции аппроксимируется прямоугольными трапециями.



Если отрезок $[x_i, x_{i+1}]$ является элементарным и не подвергается дальнейшему разбиению, значение интеграла можно найти по формуле:

$$\int_{x_i}^{x_{i+1}} f(x)dx = \frac{f(x_i) + f(x_{i+1})}{2}h + E(f), \qquad E(f) - \text{остаточный член}$$

$$|E(f)| \leq \frac{h^3}{12} \max_{x \in [x_i, x_{i+1}]} |f''(x)|$$

Это простое применение формулы для площади трапеции – произведение полусуммы оснований, которыми в данном случае являются значения функции в крайних точках отрезка, на высоту (длину отрезка интегрирования). Погрешность аппроксимации можно оценить через максимум второй производной

Тогда формула для интегрирования на всё отрезке $[a, b]$:

$$\int_a^b f(x)dx = \frac{1}{2}\sum_{i=1}^{n}(y_{i-1} + y_i)h + R(f), \quad R(f) - \text{остаточный член}$$

$$|R(f)| \leq |E(f)| * n \leq \frac{nh^3}{12} \max_{x \in [a,b]} |f''(x)| = \frac{(b-a)h^2}{12} \max_{x \in [a,b]} |f''(x)|$$

При этом мы вычисляем значение интеграла по формуле для разбиения на n и $2n$ отрезков, после чего вычисляем погрешность по формуле Рунге.

$$\Delta_{2n} \approx \theta |I_{2n} - I_n|,$$

$$\theta = \frac{1}{2^k - 1}, \text{где } k - \text{порядок точности квадратурной формулы}$$

$$\Delta_{2n} = \frac{|I_{2n} - I_n|}{3} \text{ для метода трапеций}$$

При этом если найденная погрешность оказывается больше заданной точности, то увеличиваем n в два раза и снова вычисляем интегралы. Затем повторяем увеличение n до тех пор погрешность не окажется меньше нужной точности.

Листинг программы (Java, только численный метод)

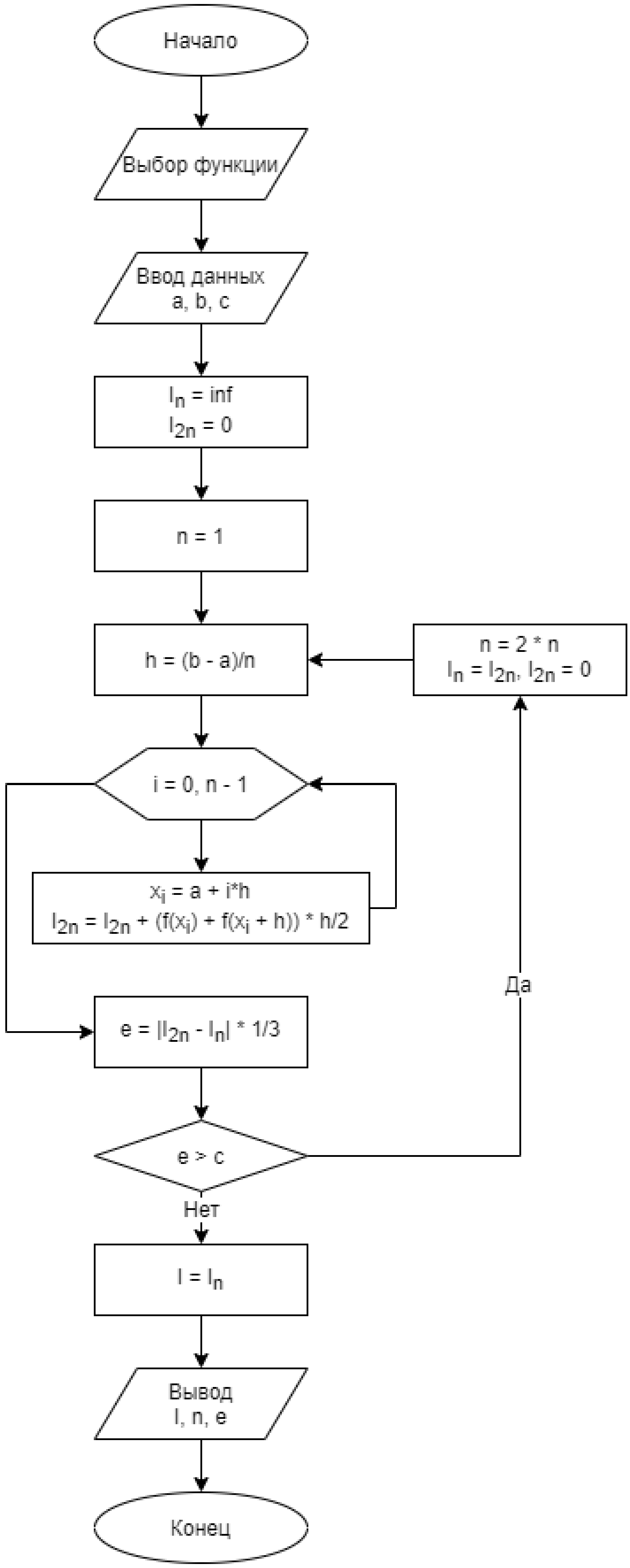
```java
import exception.*;

public class ReimannSum {
    static private final long N_MAX_VALUE = 100_000_000L;
    static private final double DOUBLE_MAX_VALUE = 1e30d;

    public ReimannSumAnswer getReimannSum(
            Function function,
            Bounds bounds,
            double accuracy,
            ReimannSumRule rule,
            TypeOfSolution solutionType
    ) throws
            NotImplementedMethodException,
            UnknownReimannSumRuleException,
            NotAllowedScopeException,
            NotSolvableIntegralException {
        checkAllowedScope(function, bounds);

        switch (solutionType) {
            case SOLUTION_BY_FORMULAS:
                return getSumByFormulasSolution(function, bounds, accuracy, rule);
            case SOLUTION_BY_RUNGE:
                return getSumByRungeSolution(function, bounds, accuracy, rule);
            default:
                throw new NotImplementedSolutionException();
        }
    }

    public ReimannSumAnswer getReimannSum(
            Function function,
            Bounds bounds,
            double accuracy
    ) throws
            NotImplementedMethodException,
            UnknownReimannSumRuleException,
            NotAllowedScopeException,
            NotSolvableIntegralException {
        return getReimannSum(
                function,
                bounds,
                accuracy,
                ReimannSumRule.TRAPEZOIDAL_RULE,
                TypeOfSolution.SOLUTION_BY_RUNGE
        );
    }

    private void checkAllowedScope(Function function, Bounds bounds)
            throws NotAllowedScopeException {
        for (Interval notAllowedScope : function.getNotAllowedScope()) {
            if (notAllowedScope.isPoint()) continue;
            if (notAllowedScope.isIntersect(bounds)) {
                throw new NotAllowedScopeException();
            }
        }
    }

    private ReimannSumAnswer getSumByFormulasSolution(
            Function function,
            Bounds bounds,
            double accuracy,
```

```
            ReimannSumRule rule
    ) throws
            NotImplementedMethodException,
            UnknownReimannSumRuleException,
            NotAllowedScopeException {
        int n = getCountOfSections(function, bounds, accuracy, rule);

        return new ReimannSumAnswer(
                getSumByRuleByN(function, bounds, rule, n),
                Double.NaN,
                n
        );
    }

    private ReimannSumAnswer getSumByRungeSolution(
            Function function,
            Bounds bounds,
            double accuracy,
            ReimannSumRule rule
    ) throws
            NotImplementedSolutionException,
            UnknownReimannSumRuleException,
            NotSolvableIntegralException,
            NotAllowedScopeException {
        int n = 15;
        double curValue = getSumByRuleByN(function, bounds, rule, n);
        double prevValue;

        do {
            n <<= 1;

            prevValue = curValue;
            curValue = getSumByRuleByN(function, bounds, rule, n);

            if (n > N_MAX_VALUE || !isAvailableValue(curValue)) {
                throw new NotSolvableIntegralException();
            }
        } while (!(getMeasurementError(prevValue, curValue, rule) < accuracy));

        return new ReimannSumAnswer(
                curValue,
                getMeasurementError(prevValue, curValue, rule),
                n
        );
    }

    private boolean isAvailableValue(double curValue) {
        return Math.abs(curValue) < DOUBLE_MAX_VALUE;
    }

    private double getMeasurementError(double value1, double value2, ReimannSumRule rule)
            throws UnknownReimannSumRuleException {
        double res = Math.abs(value1 - value2);

        switch (rule) {
            case LEFT_RULE:
            case MIDPOINT_RULE:
            case RIGHT_RULE:
            case TRAPEZOIDAL_RULE:
                return res / 3;
            case SIMPSONS_RULE:
```

```
                return res / 15;
            default:
                throw new UnknownReimannSumRuleException();
        }
    }

    private double getSumByRuleByN(Function function, Bounds bounds,
ReimannSumRule rule, int n)
            throws NotImplementedSolutionException, NotAllowedScopeException {
        double sum = 0d;
        double step = bounds.getLength() / n;
        double curLeftBound = bounds.getLeftBound();

        for (int i = 0; i < n; i++) {
            switch (rule) {
                case LEFT_RULE:
                    sum += function.getValue(curLeftBound) * step;
                    break;
                case RIGHT_RULE:
                    sum += function.getValue(curLeftBound + step) * step;
                    break;
                case MIDPOINT_RULE:
                    sum += function.getValue(
                            (curLeftBound * 2 + step) / 2
                    ) * step;
                    break;
                case TRAPEZOIDAL_RULE:
                    sum += (
                            function.getValue(curLeftBound) +
                            function.getValue(curLeftBound + step)
                    ) * step / 2;
                    break;
                case SIMPSONS_RULE:
                    //TODO: solve
                default:
                    throw new NotImplementedSolutionException();
            }

            curLeftBound += step;
        }

        return sum;
    }

    private int getCountOfSections(
            Function function,
            Bounds bounds,
            double accuracy,
            ReimannSumRule rule
    ) throws
            UnknownReimannSumRuleException,
            NotImplementedMethodException,
            NotAllowedScopeException {
        double res;

        switch (rule) {
            case LEFT_RULE:
            case RIGHT_RULE:
            case MIDPOINT_RULE:
                res = Math.sqrt(
                        Math.pow(bounds.getLength(), 3) *
```

```
                        function.get2Derivative().getMaxValue(bounds) / 24 / accuracy
                );
                return (int) (res + 1.0d);
            case SIMPSONS_RULE:
                res = Math.sqrt(
                        Math.sqrt(
                                Math.pow(bounds.getLength(), 5) *
                                function.get4Derivative().getMaxValue(bounds) / 180 / accuracy
                        )
                );
                return (int) (res + 1.0d);
            case TRAPEZOIDAL_RULE:
                res = Math.sqrt(
                        Math.pow(bounds.getLength(), 3) *
                        function.get2Derivative().getMaxValue(bounds) / 12 / accuracy
                );
                return (int) (res + 1.0d);
            default:
                throw new UnknownReimannSumRuleException();
        }
    }
}
```

Блок-схема численного метода



Тестовые данные

```
Тест 1:
Hello!
This program calculates the integrals by the trapezoid method.
Supported Commands:
Use 'choose <number of function>' to select function
Use 'exit' to quit
Use 'help' to see this text
Supported Functions:
1)y = x
2)y = sqrt(x)
3)y = 0.1*x^4 + 0.2*x^2 - 7
4)y = 0.01 / x
5)y = sin(x) / x
choose 5
You choose function y = sin(x)/x
Enter the integration limits '<start_bound> <end_bound>' ('start_bound' can be more than 'end_bound'):
-2 2
Enter the accuracy. It should be more than 0.000001:
0.001
Value of the integral is 3.21054248146843
count of steps: 64
measurement error: 2.835123225447174E-4

Тест 2:
You choose function y = 0.01/x
Enter the integration limits '<start_bound> <end_bound>' ('start_bound' can be more than 'end_bound'):
-2 2
Enter the accuracy. It should be more than 0.000001:
0.01
Value of the integral is 0.0
count of steps: 4
measurement error: 0.0

Тест 3:
You choose function y = 0.01/x
Enter the integration limits '<start_bound> <end_bound>' ('start_bound' can be more than 'end_bound'):
3 -2
Enter the accuracy. It should be more than 0.000001:
0.1
Value of the integral is -0.004061868686868687
count of steps: 4
measurement error: 7.154882154882375E-6

Тест 4:
You choose function y = x
Enter the integration limits '<start_bound> <end_bound>' ('start_bound' can be more than 'end_bound'):
2 4
Enter the accuracy. It should be more than 0.000001:
0.0001
Value of the integral is 6.0
count of steps: 4
measurement error: 0.0
```

Вывод

- Метод прямоугольников:
  Метод заключается в том, что мы разбиваем фигуру под графиком на прямоугольники и считаем интеграл как сумму площадей этих прямоугольников с учетом значения функции в левой верхней точке (метод левых прямоугольников) в средней точке (метод средних прямоугольников) и в правой верхней точке (метод правых прямоугольников). В силу того, что в методе средних прямоугольников симметрия не нарушается, то погрешность у данного метода будет меньше, чем у левых и правых прямоугольников.
- Метод трапеций:
  В данном методе мы аппроксимируем функцию к прямой и считаем интеграл как площадь многоугольника образованного получившимися трапециями. Данный метод менее точный чем метод средних прямоугольников, так как значение в средней точке точнее, чем полусумма значений на концах.
- Метод Симпсона:
  Метод Симпсона является самым точным из всех представленных методов, так как мы аппроксимируем функцию параболой (которая зачастую находится гораздо ближе к графику (в сравнении с прямой)), а значит интерполируется многочлен второй степени.

  Методы трапеции и прямоугольников плохо работают на функциях, большими скачками, например, функция $\frac{1}{x}$ в районе 0. Также численный метод плохо обрабатывает точки разрыва, так как вычислительная мощность мала, а на небольших изменениях происходят огромные скачки функции.