Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет Программной инженерии и компьютерной техники

Кафедра информатики и прикладной математики

**Дисциплина «Вычислительная математика»**

**Лабораторная работа №2**

**Метод прямоугольников**

Выполнил:

Съестов Дмитрий Вячеславович

Группа Р3217

Преподаватель:

Калёнова Ольга Вячеславовна

Санкт-Петербург, 2017 г.

**Описание метода**

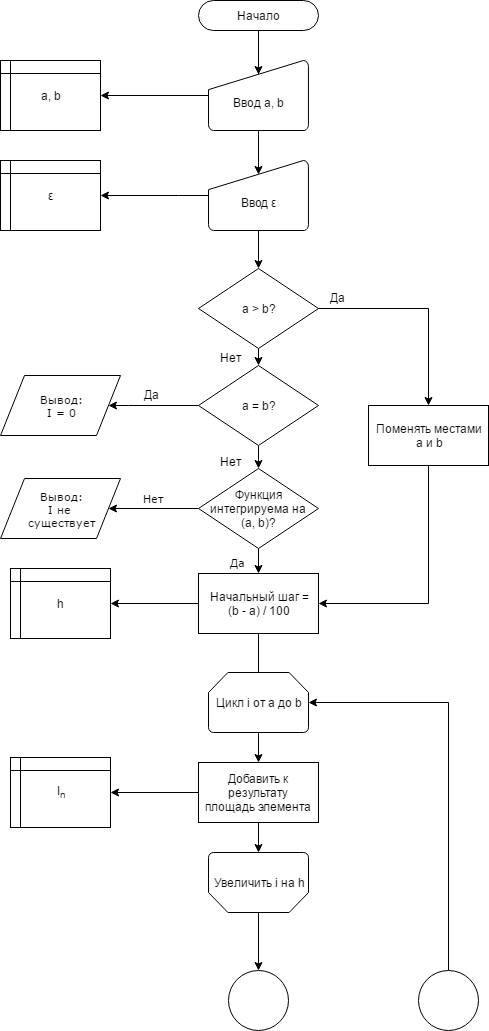
Метод прямоугольников — метод численного интегрирования функции одной переменной, заключающийся в замене подынтегральной функции на константу на каждом элементарном отрезке. Если рассмотреть график подынтегральной функции, то метод будет заключаться в приближённом вычислении площади под графиком суммированием площадей конечного числа прямоугольников, ширина которых будет определяться расстоянием между соответствующими соседними узлами интегрирования, а высота - значением подынтегральной функции в этих узлах.

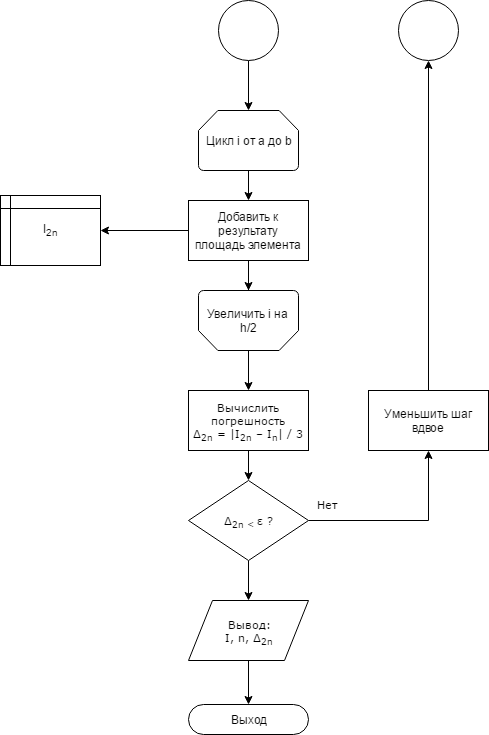
1. **Формула левых прямоугольников**:
2. **Формула правых прямоугольников**:
3. **Формула средних прямоугольников**:

После вычисления интеграла проводится повторное вычисление с удвоенным количеством разбиений и по правилу Рунге вычисляется текущая погрешность:

∆2n = θ |I2n – In|, где θ = 1/3.

Затем шаг уменьшается вдвое. Алгоритм повторяется до тех пор, пока погрешность не окажется меньше заданной пользователем точности.

****



**Листинг программы (только численный метод, C#)**

private const double INITIAL\_STEP\_AMOUNT = 100;

private static double approximateIntegral(TwoArgFunction formula, double a, double b, double step)

{

double x, result = 0;

for (x = a; x <= b; x += step)

result += formula(x, x + step);

if(double.IsInfinity(result) || double.IsNaN(result))

throw new UndefinedFunctionException("Функция не определена на заданном отрезке");

return result;

}

//Возвращает значение интеграла, количество разбиений и погрешность

public static Tuple<double, int, double> Calculate(Function f, double a, double b, double precision, FormulaType formulaType)

{

if (f == null) throw new ArgumentNullException("Функция для интегрирования не задана");

if (a == b) return Tuple.Create(0d, 0, 0d);

if (a > b) Utils.Swap(ref a, ref b);

//если в одном из пределов функция не определена, сдвигаем на эпсилон

if (double.IsInfinity(f(a)) || double.IsNaN(f(a)))

a += double.Epsilon;

if (double.IsInfinity(f(b)) || double.IsNaN(f(b)))

b -= double.Epsilon;

TwoArgFunction formula = null; //подберём нужную формулу для площади сегмента

switch (formulaType)

{

case FormulaType.Left:

formula = (a, b) => f(a) \* (b - a);

break;

case FormulaType.Right:

formula = (a, b) => f(b) \* (b - a);

break;

case FormulaType.Average:

formula = (a, b) => f((a + b) / 2) \* (b - a);

break;

}

const double THETA = 1f / 3;

double step = (b - a) / INITIAL\_STEP\_AMOUNT;

double error = precision;

double integral\_n;

do

{

//вычисляем интегралы с N и 2N шагов

integral\_n = approximateIntegral(formula, a, b, step);

double integral\_2n = approximateIntegral(formula, a, b, step / 2);

error = THETA \* Math.Abs(integral\_2n - integral\_n);

//если нужная точность не достигнута, уменьшаем шаг в два раза

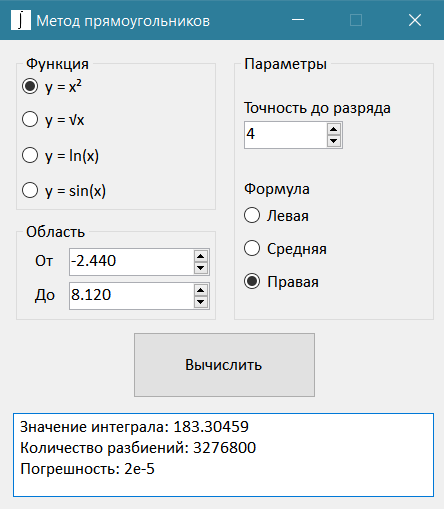
if (error >= precision) step /= 2;

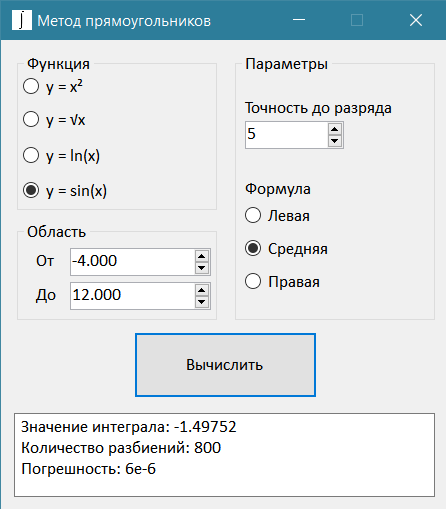
} while (error >= precision);

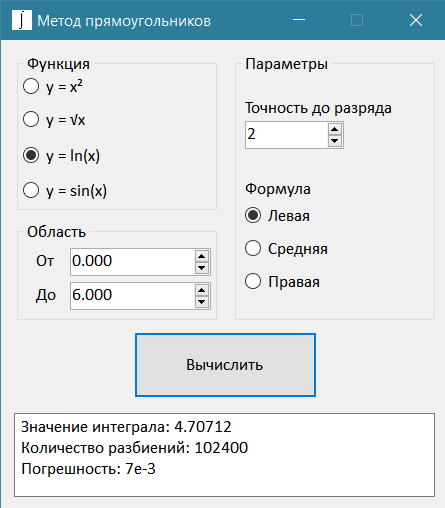
return Tuple.Create(integral\_n, (int)((b - a) / step), error);

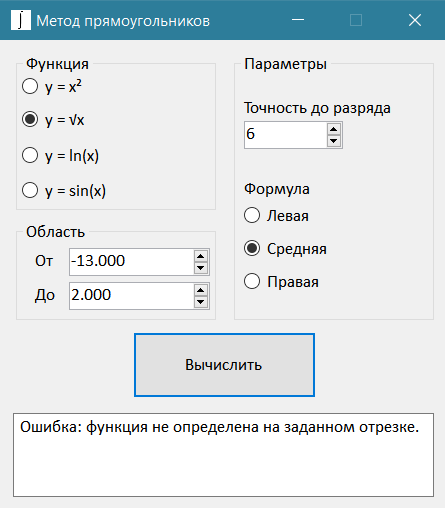
}

**Тестовые данные**









**Вывод**

Метод прямоугольника менее точен, чем метод Симпсона. Порядок точности для левых и правых прямоугольников равен 0, для средних прямоугольников и трапеций – 1, в то время как метод Симпсона имеет порядок точности 3.

Это означает, что методы прямоугольников и трапеций дают низкую точность и большое количество разбиений при интегрировании полиномов степени 2 и более.