САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Лабораторная работа по выч.математике №2

«Методы численного интегрирования»

Выполнил: Припадчев Артём
группа 2125

Проверил: Шипилов П.А.

2013 г.

Задание: составить программу вычисляющую значение интеграла тремя методами: средних прямоугольников, трапеций, парабол (Симпсона). Проанализировать изменение их погрешности в зависимости от количества интервалов разбиения.

Описание методов

Метод прямоугольников

Пусть требуется определить значение интеграла функции на отрезке $[a, b]$. Этот отрезок делится точками $x_0, x_1, \ldots, x_{n-1}, x_n$ на n равных отрезков длиной $\Delta x = \frac{b-a}{n}$. Обозначим через $y_0, y_1, \ldots, y_{n-1}, y_n$ значение функции $f(x)$ в точках $x_0, x_1, \ldots, x_{n-1}, x_n$. Далее составляем суммы $y_0 \Delta x + y_1 \Delta x + \ldots + y_{n-1} \Delta x$. Каждая из сумм — интегральная сумма для $f(x)$ на $[a, b]$ и поэтому приближённо выражает интеграл

$$\int_a^b f(x)\,dx \approx \frac{b-a}{n}(y_0 + y_1 + \ldots + y_{n-1}).$$

Если заданная функция — положительная и возрастающая, то эта формула выражает площадь ступенчатой фигуры, составленной из «входящих» прямоугольников, также называемая формулой левых прямоугольников, а формула

$$\int_a^b f(x)\,dx \approx \frac{b-a}{n}(y_1 + y_2 + \ldots + y_n)$$

выражает площадь ступенчатой фигуры, состоящей из «выходящих» прямоугольников, также называемая формулой правых прямоугольников. Чем меньше длина отрезков, на которые делится отрезок $[a, b]$, тем точнее значение, вычисляемое по этой формуле, искомого интеграла.

Очевидно, стоит рассчитывать на бóльшую точность если брать в качестве опорной точки для нахождения высоты точку посередине промежутка. В результате получаем формулу средних прямоугольников:

$$\int_a^b f(x)\,dx \approx h\sum_{i=1}^{n} f(x_{i-1} + \frac{h}{2}) = h\sum_{i=1}^{n} f(x_i - \frac{h}{2}),$$

где $h = \frac{b-a}{n}$

Учитывая априорно бóльшую точность последней формулы при том же объёме и характере вычислений её называют формулой прямоугольников

Метод трапеций

Если функцию на каждом из частичных отрезков аппроксимировать прямой, проходящей через конечные значения, то получим метод трапеций.

Площадь трапеции на каждом отрезке:

$$I_i \approx \frac{f(x_{i-1}) + f(x_i)}{2}(x_i - x_{i-1})$$

Полная формула трапеций в случае деления всего промежутка интегрирования на отрезки одинаковой длины h:

$$I \approx h\left(\frac{f(x_0)+f(x_n)}{2}+\sum_{i=1}^{n-1} f(x_i)\right), \quad \text{где} \quad h=\frac{b-a}{n}$$

Метод парабол (метод Симпсона)

Использовав три точки отрезка интегрирования, можно заменить подынтегральную функцию параболой. Обычно в качестве таких точек используют концы отрезка и его среднюю точку. В этом случае формула имеет очень простой вид

$$I \approx \frac{b-a}{6}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right).$$

Если разбить интервал интегрирования на $2N$ равных частей, то имеем

$$I \approx \frac{b-a}{6N}\left(f_0+4\left(f_1+f_3+\ldots+f_{2N-1}\right)+2\left(f_2+f_4+\ldots+f_{2N-2}\right)+f_{2N}\right),$$

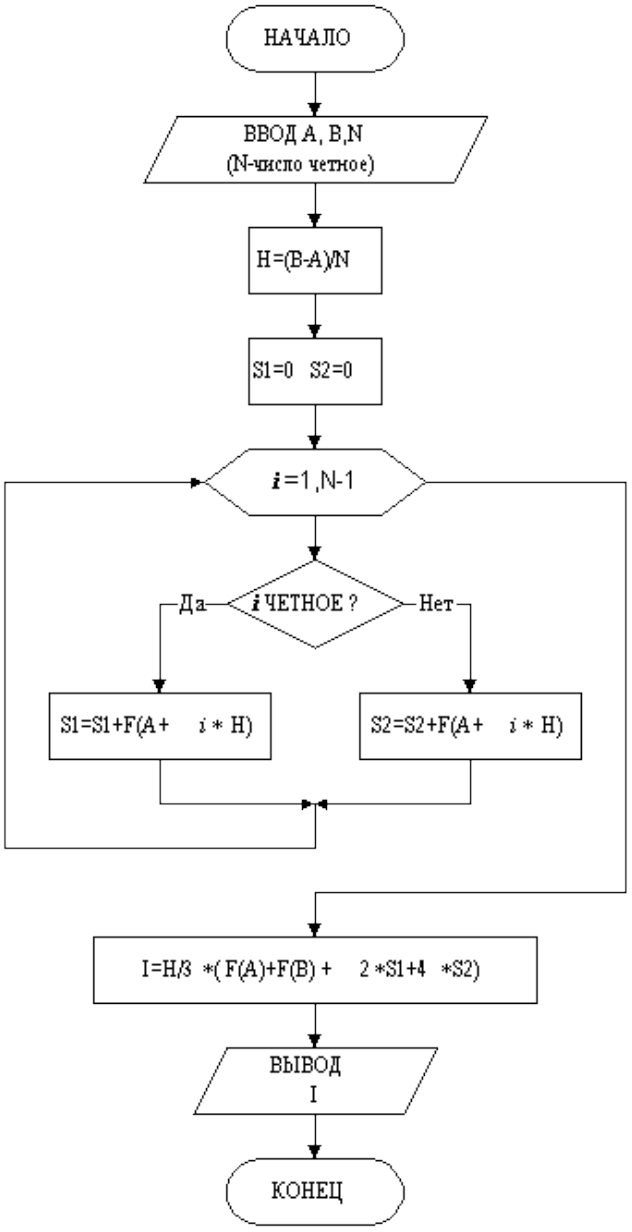
где $f_i=f\left(a+\frac{(b-a)i}{2N}\right)$.

Погрешность определяется оценкой Рунге по формуле $\Delta = \frac{I_n - I_{2n}}{2^k - 1}$, где k – порядок точности метода (для метода прямоугольников k = 1, для метода трапеций k = 2, для метода Симпсона k = 4).

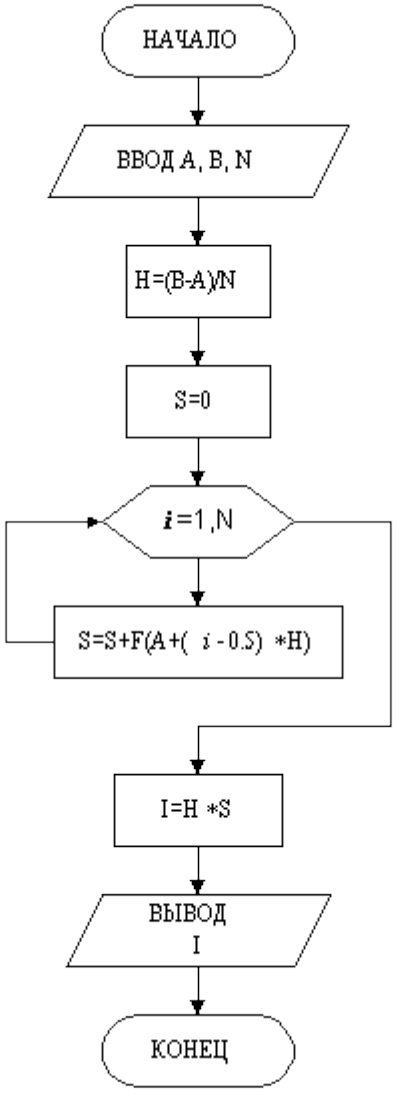
Вывод: анализируя полученные диаграммы зависимостей оценки Рунге от количества интервалов разбиения можно сделать следующие выводы:

- метод прямоугольников достигает хорошей точности только при достаточно большом количестве интервалов разбиения;
- метод трапеций дает средний результат, но также наилучшая точность достигается при бОльшем количестве разбиений, однако этот метод намного точнее метода прямоугольников при малом разбиении;
- метод Симпсона дает наилучший результат, имея небольшую погрешность даже при относительно небольшом разбиении, а при разбиении относительно большом (в рамках лабораторной работы n>1000) погрешность устремляется к нулю.

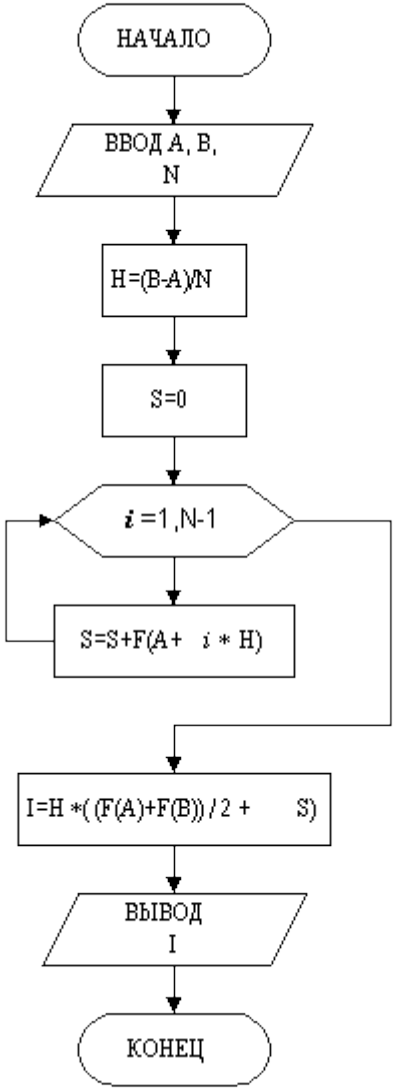
Блок-схема алгоритма метода Симпсона



Блок-схема алгоритма метода прямоугольников



Блок-схема алгоритма метода трапеций



Код программы

```
public class IntegrationMethods
    {
        public delegate double Function(double x);
        private List<RatingRunge> ratingRunge = new List<RatingRunge>();

        public void Calculate(Function function, int Step, double primaryX, double finalX)
        {
            double RatingRungeRectangle = Rectangle(function, Step, primaryX, finalX);
            double RatingRungeTrapeze = Trapeze(function, Step, primaryX, finalX);
            double RatingRungeSimpson = Simpson(function, Step, primaryX, finalX);
            ratingRunge.Add(new RatingRunge(Step, RatingRungeRectangle, RatingRungeTrapeze,
RatingRungeSimpson));
        }

        public List<RatingRunge> GetRating()
        {
            return ratingRunge;
        }

        public void ClearRating()
        {
            ratingRunge.Clear();
        }

        private double Rectangle(Function function, int step, double primaryX, double finalX)
        {
            double RecRatingRunge = 0;
            ResultOfIntegration.Rectangle = 0;
            double interval = (finalX - primaryX) / (double) step;
            double x = primaryX + interval / 2.0;
            for (int i = 0; i < step; i++)
            {
                ResultOfIntegration.Rectangle += function(x);
                x = x + interval;
            }
            ResultOfIntegration.Rectangle *= interval;
            step = step / 2;
            interval = (finalX - primaryX) / (double) step;
            x = primaryX + interval / 2.0;
            double tempResultOfIntegration = 0;
            for (int i = 0; i <= step; i++)
            {
                tempResultOfIntegration += function(x);
                x = x + interval;
            }
            tempResultOfIntegration *= interval;
            RecRatingRunge = Math.Abs(ResultOfIntegration.Rectangle - tempResultOfIntegration);
            return RecRatingRunge;
        }

        private double Trapeze(Function function, int step, double primaryX, double finalX)
        {
            ResultOfIntegration.Trapeze = 0;
            double TrapRatingRunge = 0;
            double interval = (finalX - primaryX) / (double) step;
            double s = (function(primaryX) + function(finalX)) / 2.0;
            double x = primaryX;
            for (int i = 1; i < step; i++)
            {
                x = x + interval;
                ResultOfIntegration.Trapeze += function(x);
            }
            ResultOfIntegration.Trapeze *= interval;

            step = step / 2;
            interval = (finalX - primaryX) / (double) step;
            s = (function(primaryX) + function(finalX)) / 2.0;
            x = primaryX;
            double tempResultOfIntegration = 0;
```

```
            for (int i = 1; i < step; i++)
            {
                x = x + interval;
                tempResultOfIntegration += function(x);
            }
            tempResultOfIntegration *= interval;
            TrapRatingRunge = (1.0 / 3.0) * Math.Abs(ResultOfIntegration.Trapeze -
tempResultOfIntegration);
            return TrapRatingRunge;
        }

        private double Simpson(Function function, int step, double primaryX, double finalX)
        {
            double SimRatingRunge = 0;
            double interval = (finalX - primaryX) / (double) step;
            ResultOfIntegration.Simpson = 0;
            double x = primaryX;
            for (int i = 1; i < step; i++)
            {
                x = x + interval;
                if (i % 2 == 0)
                {
                    ResultOfIntegration.Simpson += 2 * function(x);
                }
                else
                {
                    ResultOfIntegration.Simpson += 4 * function(x);
                }
            }
            ResultOfIntegration.Simpson = (ResultOfIntegration.Simpson + function(primaryX) +
function(finalX)) * interval / 3.0;

            step = step / 2;
            interval = (finalX - primaryX) / (double) step;
            x = primaryX;
            double tempResultOfIntegration = 0;
            for (int i = 1; i < step; i++)
            {
                x = x + interval;
                if (i % 2 == 0)
                {
                    tempResultOfIntegration +=  2 * function(x);
                }
                else
                {
                    tempResultOfIntegration += 4 * function(x);
                }
            }
            tempResultOfIntegration = (tempResultOfIntegration + function(primaryX) +
function(finalX)) * interval / 3.0;

            SimRatingRunge = (1.0 / 15.0) * Math.Abs(ResultOfIntegration.Simpson -
tempResultOfIntegration);
            return SimRatingRunge;
        }
    }
}
```

Диаграммы зависимостей оценки Рунге от количества шагов разбиения