CАНКТ-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 ![\left[ {a},{b} \right]](). Этот отрезок делится точками  на  равных отрезков длиной Обозначим через  значение функции  в точках  Далее составляем суммы  Каждая из сумм — интегральная сумма для  на ![\left[ {a},{b} \right]]() и поэтому приближённо выражает интеграл



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



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

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



где 

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

**Метод трапеций**

Если функцию на каждом из частичных отрезков [аппроксимировать](http://ru.wikipedia.org/wiki/%D0%90%D0%BF%D0%BF%D1%80%D0%BE%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D1%86%D0%B8%D1%8F) прямой, проходящей через конечные значения, то получим метод трапеций.

Площадь трапеции на каждом отрезке:



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

 где 

**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

.

Если разбить интервал интегрирования на  равных частей, то имеем



где .

**Погрешность определяется оценкой Рунге по формуле** $∆ = \frac{I\_{n}-I\_{2n}}{2^{k}-1}$**,** 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**