**БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ**

Лабораторная №1

**Интегрирование функций**

Вариант 8

**Выполнил:**

Кендысь Алексей Максимович

студент 3 курса, 7 группы,

специальность

“прикладная математика”

**Преподаватель:**

Доцент кафедры вычислительной

математики ФПМИ,

А.М. Будник

Минск, 2022

**Содержание:**

Постановка задачи 2

Предварительные вычисления 2

Правило Рунге 3

Краткие теоретические сведения 3

Листинг программы 4

Результаты 5

Выводы 5

Составные формулы средних прямоугольников и Симпсона 6

Краткие теоретические сведения 6

Листинг программы 7

Результаты 8

Выводы 9

Формула НАСТ Гаусса 9

Краткие теоретические сведения 9

Листинг программы 11

Результаты 12

Выводы 12

Постановка задачи

Для вычисления интеграла с точностью = необходимо:

1. Применить правило Рунге, используя составную квадратурную формулу правых прямоугольников. Определить величину шага разбиения исходного отрезка интегрирования, достаточного для достижения точности .
2. Пользуясь выражением для погрешности интегрирования, определить шаги в двух нижеуказанных квадратурных формулах, которые обеспечат требуемую точность результата:
3. Составная квадратурная формула средних прямоугольников.
4. Составная квадратурная формула Симпсона.
5. Применить квадратурную формулу НАСТ Гаусса при . Оценить погрешность интегрирования через формулу остаточного члена
6. Провести сравнительный анализ полученных в п.п. 1-3 результатов.

Предварительные вычисления

Вычислим точное значение интеграла.

Вычислим некоторые производные подынтегральной функции.

Аналогично:

Правило Рунге

Краткие теоретические сведения

Составная квадратурная формула правых прямоугольников (ФПП):

– число разбиений.

– остаток формулы.

Представление для погрешности по правилу Рунге:

Выбираются два различных шага и строятся два приближения, таким образом постепенно уменьшаем шаг для достижения требуемой точности. Будем использовать . При этом – порядок точности, т.е. степень в выражении для остатка используемой формулы. В нашем случае , т.к. мы используем ФПП. В итоге получим:

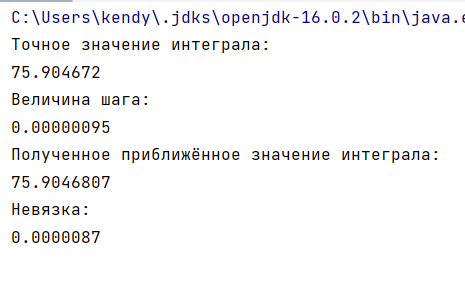
Для проверки окончания процесса используем . В качестве начального шага возьмём В качестве результата приближения интеграла берём , где была достигнута точность по остатку.

Листинг программы

import java.util.\*;  
  
class F {  
 public static double getValue(double x) {  
 return x \* Math.*pow*(1. + x, 1. / 3.);  
 }  
}  
  
class RRS {  
 public static double approxIntegral(double a, double b, double h) {  
 double res = 0.;  
 double x = a + h;  
 int n = (int)((b - a) / h);  
  
 for (int i = 0; i < n; i++) {  
 res += F.*getValue*(x);  
 x += h;  
 }  
  
 return res \* h;  
 }  
}  
  
class RungeLaw {  
 private final static int *STEP\_DECREASE* = 2;  
 private final double integralVal;  
 private double hRes;  
 private double qRes;  
  
 public RungeLaw(double integralVal) {  
 this.integralVal = integralVal;  
 }  
  
 public void approxIntegral(double a, double b, double h0, double e) {  
 double h1 = h0;  
  
 double h2 = h1 / *STEP\_DECREASE*;  
  
 double q1 = RRS.*approxIntegral*(a, b, h1);  
 double q2 = RRS.*approxIntegral*(a, b, h2);  
  
 while (Math.*abs*(this.getResidual(q1, q2)) > e) {  
 h1 = h2;  
 h2 = h1 / *STEP\_DECREASE*;  
 q1 = q2;  
 q2 = RRS.*approxIntegral*(a, b, h2);  
 }  
  
 this.hRes = h2;  
 this.qRes = q2;  
 }  
  
 public void outRes() {  
 Formatter fmt = new Formatter();  
 fmt.format("Величина шага:\n");  
 fmt.format("%.8f\n", this.hRes);  
 fmt.format("Полученное приближённое значение интеграла:\n");  
 fmt.format("%.7f\n", this.qRes);  
 fmt.format("Невязка:\n");  
 fmt.format("%.7f\n", Math.*abs*(this.qRes - this.integralVal));  
 System.*out*.println(fmt);  
 }  
  
 private double getResidual(double q1, double q2) {  
 return q2 - q1;  
 }  
}

public class Main {  
 private static final double *A* = 1.;  
 private static final double *B* = 9.;  
 private static final double *H0* = 1.;  
 private static final double *E* = 0.00001;  
 private static final double *I* = 75.90467203;   
  
 public static void main(String[] args) {  
 System.*out*.println("Точное значение интеграла:");  
 System.*out*.println(*I*);   
  
 RungeLaw rl = new RungeLaw(*I*);  
 rl.approxIntegral(*A*, *B*, *H0*, *E*);  
 rl.outRes();  
 }  
}

Результаты



Выводы

Правило Рунге позволяет вычислить приближённое значение остатка. Из результатов программы видно, что несмотря на то, что остаток приближённый, интеграл всё равно получается требуемой точности, что подтверждает справедливость метода. Также из результатов видно, что формула правых прямоугольников требует достаточно малого шага для хороших вычислений, это неудивительно, т.к. алгебраическая степень точности формулы равна нулю.

Составные формулы средних прямоугольников и Симпсона

Краткие теоретические сведения

Составная формула средних прямоугольников:

, где

Оценим остаток и найдём нужный шаг.

*.*

для

для убывает на

*.*

Тогда . Откуда получаем . Следовательно, . Для достижения нужной точности используем формулу с .

Составная формула Симпсона:

Формула выше используется для чётных (используем такую форму формулы Симпсона для удобства программирования, с ней меньше лишних вычислений).

Аналогично оценим остаток и найдём нужный шаг.

для . Аналогично для убывает на *.*

Тогда . Откуда получаем . Следовательно, . Для достижения нужной точности используем формулу с .

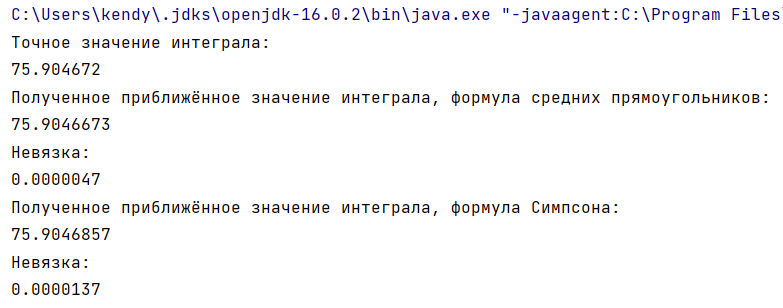
Все округления произведены в большую сторону для сохранения неравенств.

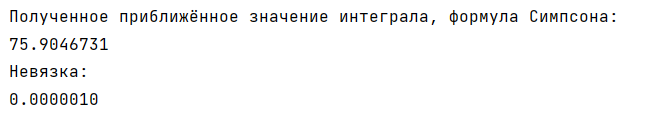
Листинг программы

import java.util.\*;  
  
class F {  
 public static double getValue(double x) {  
 return x \* Math.*pow*(1. + x, 1. / 3.);  
 }  
}

class MRS {  
 private final double integralVal;  
 private double qRes;  
  
 public MRS(double integralVal) {  
 this.integralVal = integralVal;  
 }  
  
 public void approxIntegral(double a, double b, int n) {  
 double res = 0.;  
 double h = (b - a) / n;  
  
 for(double i = 0.; i < n; i++) {  
 res += F.*getValue*(a + (0.5 \* h) + (i \* h));  
 }  
  
 this.qRes = res \* h;  
 }  
  
 public void outRes() {  
 Formatter fmt = new Formatter();  
 fmt.format("Полученное приближённое значение интеграла, формула средних прямоугольников:\n");  
 fmt.format("%.7f\n", this.qRes);  
 fmt.format("Невязка:\n");  
 fmt.format("%.7f\n", Math.*abs*(this.qRes - this.integralVal));  
 System.*out*.print(fmt);  
 }  
}  
  
class Simpson {  
 private final double integralVal;  
 private double qRes;  
  
 public Simpson(double integralVal) {  
 this.integralVal = integralVal;  
 }  
  
 public void approxIntegral(double a, double b, int n) {  
 double res = 0.;  
 double h = (b - a) / n;  
  
 res += F.*getValue*(a);  
 res += F.*getValue*(b);  
  
 double x = a + h;  
 for(int i = 1; i < n - 1; i++, x += h) {  
 res += 4. \* F.*getValue*(x);  
 x += h;  
 i++;  
 res += 2. \* F.*getValue*(x);  
 }  
 res += 4. \* F.*getValue*(x);  
  
 this.qRes = (res \* h) / 3.;  
 }  
  
 public void outRes() {  
 Formatter fmt = new Formatter();  
 fmt.format("Полученное приближённое значение интеграла, формула Симпсона:\n");  
 fmt.format("%.7f\n", this.qRes);  
 fmt.format("Невязка:\n");  
 fmt.format("%.7f\n", Math.*abs*(this.qRes - this.integralVal));  
 System.*out*.println(fmt);  
 }  
}  
  
public class Main {  
 private static final double *A* = 1.;  
 private static final double *B* = 9.;   
 private static final double *E* = 0.00001;  
 private static final double *I* = 75.90467203;  
 private static final int *N1* = 870;  
 private static final int *N2* = 42;  
  
 public static void main(String[] args) {  
 System.*out*.println("Точное значение интеграла:");  
 System.*out*.println(*I*);  
  
 MRS mrs = new MRS(*I*);  
 mrs.approxIntegral(*A*, *B*, *N1*);  
 mrs.outRes();  
  
 Simpson s = new Simpson(*I*);  
 s.approxIntegral(*A*, *B*, *N2*);  
 s.outRes();  
 }  
}

Результаты





Выводы

Что касается формулы средних прямоугольников, мы смогли получить нужную точность (даже чуть больше, чем нужно), это объясняется тем, что остаток мы оценивали, а не вычисляли точно. При этом для этого потребовался больший шаг (меньшее количество разбиений), чем для формулы правых прямоугольников из прошлого пункта, т.к. алгебраическая степень точности формулы больше, она равна 1.

Формула Симпсона предоставила нужную точность, при этом использовав значительно меньшее количество разбиений , чем в прошлых формулах. Это объясняется разностью в алгебраической степени точности. Алгебраическая степень точности формулы Симпсона равна 3.

Формула НАСТ Гаусса

Краткие теоретические сведения

Формула НАСТ Гаусса вычисляет интеграл вида , для которого можем записать:

Для него известны коэффициенты и узлы для различных . В нашем случае :

Для того, чтобы свести формулу к нашему интегралу, используем замену

.

Тогда:

, где:

Остаток для формулы НАСТ Гаусса:

Оценим производную.

для . Аналогично для убывает на *.*

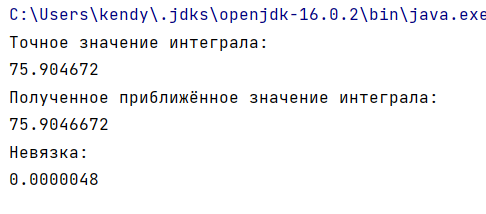
Откуда получаем

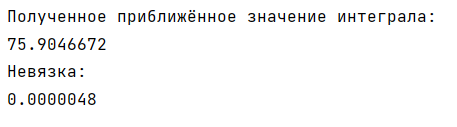
Листинг программы

import java.util.\*;  
  
class F {  
 public static double getValue(double x) {  
 return x \* Math.*pow*(1. + x, 1. / 3.);  
 }  
}

class Gauss {  
 private static final int *N* = 4;  
 private static final double[] *X* = { 8.624719, 7.153877, 5., 2.846123, 1.375281 };  
 private static final double[] *A* = { 0.947708, 1.914515, 2.275556, 1.914515, 0.947708 };  
 private final double integralVal;  
 private double qRes;  
  
 public Gauss(double integralVal) {  
 this.integralVal = integralVal;  
 }  
  
 public void approxIntegral() {  
 double res = 0.;  
  
 for(int i = 0; i <= *N*; i++) {  
 res += *A*[i] \* F.*getValue*(*X*[i]);  
 }  
  
 this.qRes = res;  
 }  
  
 public void outRes() {  
 Formatter fmt = new Formatter();  
 fmt.format("Полученное приближённое значение интеграла:\n");  
 fmt.format("%.7f\n", this.qRes);  
 fmt.format("Невязка:\n");  
 fmt.format("%.7f\n", Math.*abs*(this.qRes - this.integralVal));  
 System.*out*.println(fmt);  
 }  
}  
  
public class Main {  
 private static final double *A* = 1.;  
 private static final double *B* = 9.;  
 private static final double *E* = 0.00001;  
 private static final double *I* = 75.90467203;  
  
 public static void main(String[] args) {  
 System.*out*.println("Точное значение интеграла:");  
 System.*out*.println(*I*);  
  
 Gauss g = new Gauss(*I*);  
 g.approxIntegral();  
 g.outRes();  
 }  
}

Результаты





Выводы

Из результатов видно, что мы смогли достичь достаточно хорошей точности: такой же, что и в предыдущих формулах, при этом её не требуя. К тому же формула является простой, и в нашем случае использовалось только 5 узлов. Это позволяет сделать вывод, что формула Гаусса является наилучшим методом из представленных в плане точности. Сложность состоит в нахождении коэффициентов и узлов, мы использовали готовые. Также заметим, что в нашем случае оценка для остатка получилась довольно грубой, на практике интеграл вышел точнее. Это объясняется тем, что производная, фигурирующая в остатке, имеет большой разброс значений на отрезке. Как уже было показано выше, . При этом . Также производная имеет график, похожий на график квадратного корня, т.е. имеет большие значения на маленьком промежутке.

Формула НАСТ Гаусса при имеет алгебраическую степень точности 9.