БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа №1

**Интегрирование функций**

Вариант 5

**Выполнила:**

Зуйкевич Лидия

3 курс 7 группа

**Преподаватель:**

Будник А.М.

Минск, 2022

Оглавление

[**1. Использование правила Рунге** 3](#_Toc118753263)

[Постановка задачи 3](#_Toc118753264)

[Алгоритм решения 3](#_Toc118753265)

[Листинг программы 3](#_Toc118753266)

[**2. Составные квадратурные формулы средних прямоугольников и Симпсона** 5](#_Toc118753267)

[Постановка задачи 5](#_Toc118753268)

[**Составная квадратурная формула средних прямоугольников** 5](#_Toc118753269)

[Алгоритм решения 5](#_Toc118753269)

[Листинг программы 6](#_Toc118753270)

[Вывод программы 7](#_Toc104902787)

[Выводы 7](#_Toc104902787)

[**Составная квадратурная формула Симпсона** 7](#_Toc118753269)

[Алгоритм решения 7](#_Toc118753271)

[Листинг программы 8](#_Toc118753272)

[Вывод программы 9](#_Toc104902787)

[Выводы 9](#_Toc104902787)

[**3. Формула НАСТ Гаусса** 9](#_Toc118753273)

[Постановка задачи 9](#_Toc118753274)

[Алгоритм решения 9](#_Toc118753275)

[Листинг программы 11](#_Toc118753276)

[Вывод программы 12](#_Toc104902787)

[Выводы 12](#_Toc104902787)

Общая постановка задачи

Вычислить интеграл с точностью , используя указанные методы, провести сравнительный анализ.

Вычисление точного значения интеграла

*,* где *F(x) –* первообразная.

- это значение возьмем в качестве точного значения интеграла.

# Использование правила Рунге

# Постановка задачи

Для вычисления интеграла с точностью применить правило Рунге, используя составную квадратурную формулу правых прямоугольников. Определить величину h шага разбиения исходного отрезка интегрирования, достаточного для достижения точности .

# Алгоритм решения

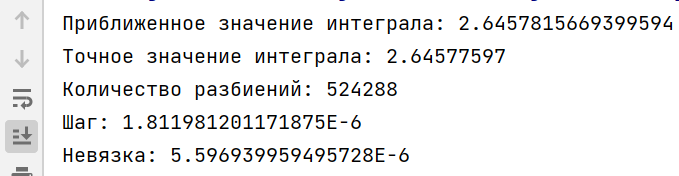
Составная КФ правых прямоугольников имеет вид: , где Ее остаток имеет вид: . Здесь h – шаг, который нужно определить с помощью правила Рунге. Для этого в цикле строим два приближенных значения интеграла с шагом h и соответственно: и . Оценка погрешности: , где m - константа, показатель степени, в которой h содержится в остатке КФ. В данном случае m = 1, поэтому погрешность имеет вид: . В качестве критерия остановки процесса используем . В качестве приближенного значения интеграла возьмем

# Листинг программы

public class Runge {  
 private double a;  
 private double b;  
 private int N;  
 private double h;  
 private double Q1;  
 private double Q2;  
 private double R;  
 private double r;  
 private final double I = 2.64577597;  
 private final double EPS = Math.*pow*(10, -5);  
  
 public Runge(double a, double b) {  
 this.a = a;  
 this.b = b;  
 this.N = 2;  
 this.h = (b - a) / N;  
 this.R = Double.*MAX\_VALUE*;  
 }  
  
 public double f(double x){  
 return Math.*sqrt*(Math.*exp*(x) - 1);  
 }  
  
 public void solve(){  
 for (int i = 0; i < N; i++){  
 Q1 += f(a + i \* h);  
 }  
  
 Q1 \*= h;  
 h /= 2;  
 N \*= 2;  
  
 while (Math.*abs*(R) > EPS){  
 for (int i = 0; i < N; i++){  
 Q2 += f(a + i \* h);  
 }  
 Q2 \*= h;  
  
 R = Q2 - Q1;  
 Q1 = Q2;  
 h /= 2;  
 N \*= 2;  
 }  
  
 r = Math.*abs*(I - Q2);  
 System.*out*.println("Приближенное значение интеграла: " + Q2);

System.*out*.println("Точное значение интеграла: " + I);  
 System.*out*.println("Количество разбиений: " + N/2);  
 System.*out*.println("Шаг: " + h);  
 System.*out*.println("Невязка: " + r);  
 }  
 public static void main(String[] args){  
 Runge obj = new Runge (0.1, 2);  
 obj.solve();  
 }  
}

Вывод программы



Выводы

Значение интеграла вычислено с заданной точностью, что видно по невязке, имеющей порядок Найденное приближенное значение интеграла совпадает с точным до 4-ех знаков после запятой, дальнейшее несовпадение можно объяснить в том числе накопленной погрешностью вычислений, т.к. количество разбиений велико.

# Составные квадратурные формулы средних прямоугольников и Симпсона

# Постановка задачи

Пользуясь выражением для погрешности интегрирования, определить шаги h в составной КФ средних прямоугольников и составной КФ Симпсона, которые обеспечат точность результата приближенного вычисления интеграла .

* 1. Составная квадратурная формула средних прямоугольников

# Алгоритм решения

Составная КФ средних прямоугольников имеет вид: , где .

Ее остаток:

Оценим вторую производную на отрезке интегрирования:

- функция возрастает и положительна на всем отрезке интегрирования, поэтому максимум достигается при x = 2.

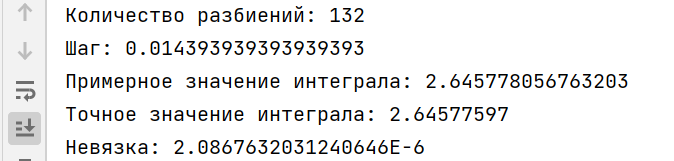
По условию, . Выразим из этого неравенства N (так результат получается более точным, чем при оценке h):

Возьмем наименьшее целое N, удовлетворяющее неравенству, рассчитаем шаг h и найдем приближенное значение интеграла по формуле средних прямоугольников.

# Листинг программы

public class MiddleRiemannSum {  
 private double a;  
 private double b;  
 private int N;  
 private double h;  
 private double Q;  
 private double f\_2;  
 private double r;  
 private final double I = 2.64577597;  
 private final double EPS = Math.*pow*(10, -5);  
  
 public MiddleRiemannSum(double a, double b) {  
 this.a = a;  
 this.b = b;  
 this.f\_2 = (Math.*exp*(2) \* (Math.*exp*(2) - 2)) / (4 \* Math.*pow*(Math.*exp*(2) - 1, 3./2));  
 }  
  
 public double f(double x){  
 return Math.*sqrt*(Math.*exp*(x) - 1);  
 }  
  
 public void findH(){  
 N = (int)Math.*sqrt*((Math.*pow*((b - a), 3) \* f\_2) / (24 \* EPS));  
 h = (b - a) / N;  
 System.*out*.println("Количество разбиений: " + N);  
 System.*out*.println("Шаг: " + h);  
 }  
  
 public void findSum(){  
 for (int i = 0; i < N; i++){  
 Q += f(a + i \* h + h / 2);  
 }  
 Q \*= h;  
 r = Math.*abs*(I - Q);  
 System.*out*.println("Примерное значение интеграла: " + Q);  
 System.*out*.println("Точное значение интеграла: " + I);  
 System.*out*.println("Невязка: " + r);  
 }  
  
 public static void main(String[] args){  
 MiddleRiemannSum obj = new MiddleRiemannSum(0.1, 2);  
 obj.findH();  
 obj.findSum();  
 }  
}

Вывод программы



Выводы

Требуемая точность достигнута, невязка имеет порядок , найденное значение интеграла совпадает с точным до 5 знаков после запятой, невязка так же лучше, чем у значения, найденного с использованием правила Рунге. Это можно объяснить тем, что для нахождения необходимого числа разбиений использовалась оценка сверху. Также количество разбиений гораздо меньше, чем при использовании правила Рунге, поэтому погрешность вычислений меньше. К тому же, сама КФ средних прямоугольников является более точной, чем КФ правых прямоугольников, т.к. их АСТ 1 и 0 соответственно.

* 1. Составная квадратурная формула Симпсона

# Алгоритм решения

Составная КФ Симпсона имеет вид: , где .

Ее остаток:

Оценим четвертую производную на отрезке интегрирования:

- функция возрастает, но меняет знак на отрезке интегрирования. Максимум модуля достигается при x = 0.1.

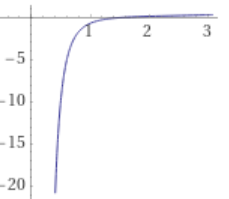
**

Рис. – график .

По условию, ,

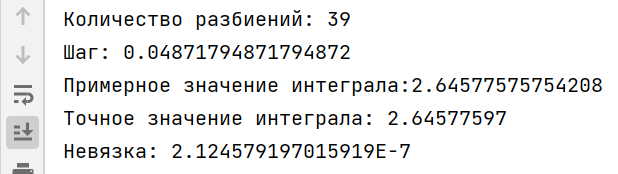
Выразим из этого неравенства N (так результат получается более точным, чем при оценке h):

Возьмем наименьшее целое N, удовлетворяющее неравенству, рассчитаем шаг h и найдем приближенное значение интеграла по формуле Симпсона.

# Листинг программы

public class SimpsonsRule {  
 private double a;  
 private double b;  
 private int N;  
 private double h;  
 private double Q;  
 private double f\_4;  
 private double r;  
 private final double I = 2.64577597;  
 private final double EPS = Math.*pow*(10, -5);  
  
 public SimpsonsRule(double a, double b) {  
 this.a = a;  
 this.b = b;  
 this.f\_4 = Math.*abs*(Math.*exp*(0.1) \* (-4 \* Math.*exp*(0.1) - 4 \* Math.*exp*(0.2) + Math.*exp*(0.3) - 8)  
 / (16 \* Math.*pow*(Math.*exp*(0.1) - 1, 7./2)));  
 }  
  
 public double f(double x){  
 return Math.*sqrt*(Math.*exp*(x) - 1);  
 }  
  
 public void findH(){  
 N = (int)(Math.*pow*((Math.*pow*(b - a, 5) \* f\_4) / (EPS \* 2880), 1. / 4));  
 h = (b - a) / N;  
  
 System.*out*.println("Количество разбиений: " + N);  
 System.*out*.println("Шаг: " + h);  
 }  
  
 public void findSum(){  
 for (int i = 0; i < N; i++){  
 Q += f(a + i \* h) + 4 \* f(a + i \* h + h / 2) + f(a + (i + 1) \* h);  
 }  
 Q \*= h / 6;  
  
 r = Math.*abs*(I - Q);  
 System.*out*.println("Примерное значение интеграла:" + Q);  
 System.*out*.println("Точное значение интеграла: " + I);  
 System.*out*.println("Невязка: " + r);  
 }  
  
 public static void main(String[] args){  
 SimpsonsRule obj = new SimpsonsRule(0.1, 2);  
 obj.findH();  
 obj.findSum();  
 }  
}

Вывод программы



Выводы

Требуемая точность достигнута, невязка имеет порядок , найденное значение интеграла совпадает с точным до 6 знаков после запятой, невязка так же лучше, чем у значения, найденного с использованием правила Рунге и с использованием КФ средних прямоугольников. Это можно объяснить тем, что для нахождения необходимого числа разбиений использовалась оценка сверху. Также количество разбиений гораздо меньше, чем при использовании правила Рунге и формулы средних прямоугольников, поэтому погрешность вычислений меньше. К тому же, сама КФ Симпсона является более точной, чем КФ средних и правых прямоугольников, т.к. их АСТ 2, 1 и 0 соответственно.

# Формула НАСТ Гаусса

# Постановка задачи

Для вычисления интеграла вычисления интеграла применить квадратурную формулу НАСТ Гаусса при n = 4. Оценить погрешность интегрирования через формулу остаточного члена

# Алгоритм решения

Формула НАСТ Гаусса имеет вид: , где и вычисляются определенным образом (в нашем случае, - корни многочленов Лежандра, а вычисляются по формулам и зависят от (значения полиномов в )).

Будем использовать узлы и коэффициенты из соответствующей таблицы. Приведенные в таблице узлы и коэффициенты используются для вычисления интегралов по отрезку [-1; 1], поэтому необходимо выполнить замену:

Тогда интеграл имеет вид:

Вычислять приближенное значение интеграла будем в виде:

Остаток формулы имеет вид:

С учетом замены для исходного отрезка получим:

Тогда окончательная формула:

Оценка погрешности интегрирования:

Оценим десятую производную на отрезке [0.1, 2]

Максимум модуля достигается при x = 0.1, |*1.059500297788033E14*

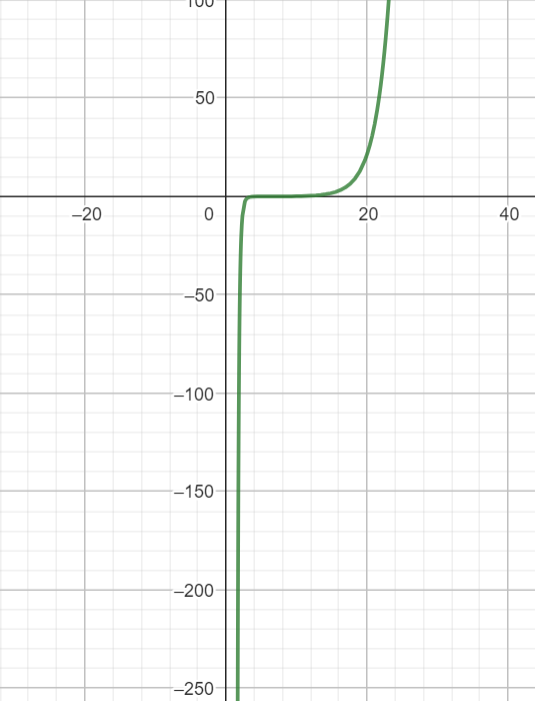
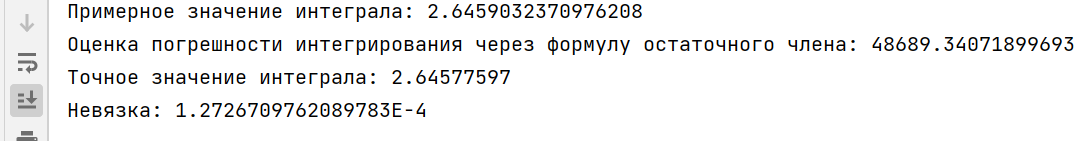


Рис. – график

# Листинг программы

public class GaussianQuadrature {  
 private double a;  
 private double b;  
 private int n;  
 private double Q;  
 private double[] x;  
 private double[] A;  
 private final double EPS = Math.*pow*(10, -5);  
 private double r;  
 private final double I = 2.64577597;  
 private double f\_10;  
 private double R;  
  
 public GaussianQuadrature(double a, double b, int n) {  
 this.a = a;  
 this.b = b;  
 this.n = n;  
 this.Q = 0.;  
 this.x = new double[]{0.906179845938664, 0.538469310105683, 0., -0.538469310105683, -0.906179845938664};  
 this.A = new double[]{0.236926885056189, 0.478628704993665, 0.568888888888889, 0.478628704993665, 0.236926885056189};  
 this.f\_10 = Math.*abs*((Math.*exp*(0.1) \* (-126208 \* Math.*exp*(0.1) - 2554624 \* Math.*exp*(0.2) - 11281600 \* Math.*exp*(0.3)  
 - 14415136 \* Math.*exp*(0.4) - 5533648 \* Math.*exp*(0.5) - 540352 \* Math.*exp*(0.6) - 7336 \* Math.*exp*(0.7)  
 - 10 \* Math.*exp*(0.8) + Math.*exp*(0.9) - 512))/(1024 \* Math.*pow*(-1 + Math.*exp*(0.1),19./2)));  
 }  
  
 public double f(double x){  
 return Math.*sqrt*(Math.*exp*(x) - 1);  
 }  
  
 public static int factorial(int num) {  
 if (num == 0){  
 return 1;  
 }  
 int result = 1;  
 for (int i = 2; i <= num; i++) {  
 result \*= i;  
 }  
 return result;  
 }  
  
 public void findSum(){  
 for(int i = 0; i <= n; i++){  
 Q += A[i] \* f(x[i] \* (b - a) / 2 + (b + a) / 2);  
 }  
 Q \*= (b - a) / 2;  
  
 R = (Math.*pow*(b - a, 2 \* n + 3) \* Math.*pow*(*factorial*(n + 1), 4) \* f\_10) /  
 ((2 \* n + 3) \* Math.*pow*(*factorial*(2 \* n + 2), 3));  
  
 r = Math.*abs*(I - Q);  
 System.*out*.println("Примерное значение интеграла: " + Q);  
 System.*out*.println("Оценка погрешности интегрирования через формулу остаточного члена: " + R);  
 System.*out*.println("Точное значение интеграла: " + I);  
 System.*out*.println("Невязка: " + r);  
 }  
  
 public static void main(String[] args){  
 GaussianQuadrature obj = new GaussianQuadrature(0.1, 2, 4);  
 obj.findSum();  
 }  
}

Вывод программы



Выводы

Невязка имеет порядок и найденное значение совпадает с точным до 3 знаков после запятой, что говорит о том, что значение, найденное с помощью КФ НАСТ, получилось наименее точным из всех. Формула является простой, в то время как все остальные – составными, использовалось всего 5 узлов, что значительно меньше, чем в остальных формулах. Преимуществом использования данной формулы является то, что узлы и коэффициенты были заранее известны (взяты из соответствующих таблиц), поэтому количество вычислений было наименьшим. Оценка погрешности интегрирования получилось грубой из-за оценки максимума модуля 10-той производной, т.к. функция быстро возрастает, что видно из графика (граничные значения на отрезке: *1.059500297788033E14* (модуль взят в качестве оценки) и ). Наиболее точное значение было получено с помощью составной формулы Симпсона.