**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МОЭВМ**

**ОТЧЕТ**

**по практической работе №4**

**по дисциплине «Вычислительная математика»**

**Тема: Численное интегрирование**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3344 |  | Гайфутдинов А.Р. |
| Преподаватель |  | Попова Е.В. |

Санкт-Петербург

2025

## Цель работы.

Изучение и сравнение различных методов численного интегрирования на примере составных формул прямоугольников, трапеций, Симпсона.

**Основные теоретические положения.**

Пусть требуется найти определенный интеграл

,

где функция f(x) непрерывна на отрезке [a,b]. Для приближенного вычисления интегралов чаще всего подынтегральную функцию заменяют «близкой» ей вспомогательной функцией, интеграл от которой вычисляется аналитически. За приближенное значение интеграла принимают значение интеграла от вспомогательной функции.

Заменим функцию на отрезке [a,b] её значением в середине отрезка. Искомый интеграл, равный площади криволинейной фигуры, заменяется на площадь прямоугольника. Из геометрических соображений нетрудно записать формулу прямоугольников:

Приблизив f(x) линейной функцией и вычислив площадь соответствующей трапеции, получим формулу трапеций:

Если же приблизить подынтегральную функцию параболой, проходящей через точки (𝑎, 𝑓(𝑎)), ( 𝑎+𝑏/ 2 , 𝑓( (𝑎+𝑏) /2 )), (𝑏, 𝑓(𝑏)), то получим формулу Симпсона (парабол):

Для повышения точности интегрирования применяют составные формулы. Для этого разбивают отрезок [𝑎, 𝑏] на четное 𝑛 = 2𝑚 число отрезков длины ℎ = (𝑏−𝑎)/𝑛 и на каждом из отрезков длины 2h применяют соответствующую формулу. Таким образом получают составные формулы прямоугольников, трапеций и Симпсона.

На сетке 𝑥𝑖 = 𝑎 + 𝑖ℎ, 𝑦𝑖= f(𝑥𝑖 ), 𝑖 = 0̅̅,̅2̅̅𝑚̅̅ составные формулы имеют следующий вид:

Формула прямоугольников:

Формула трапеций:

Формула Симпсона:

где 𝑅1, 𝑅2, 𝑅3 – остаточные члены (априорные погрешности). Оценки остаточных членов получены в предположении, что соответствующие производные f(x) непрерывны на [a,b].

Для практической оценки погрешности квадратурной можно использовать правило Рунге. Для этого проводят вычисления на сетках с шагом h и h/2, получают приближенные значения интеграла 𝐼ℎ и 𝐼ℎ/2 за окончательные значения интеграла принимают величины:

▪ – для формулы прямоугольников;

▪ – для формулы трапеций;

▪ – для формулы Симпсона.

При этом за погрешность приближённого значения интеграла принимается величина | | (апостериорные формулы), причём для формул прямоугольников и трапеций k = 2, для формулы Симпсона k = 4.

**Такую оценку погрешностей применяют обычно для построения адаптивных алгоритмов, т.е. таких алгоритмов, которые автоматически так определяют величину шага h, что результат удовлетворяет требуемой точности.**

**Задание:** вычислить определенный интеграл с границами интегрирования от *а* = 0 до *в* = 1, разбив интервал интегрирования на 10 частей, с точностью до .

**Порядок выполнения работы.**

1. Найти область определения функции и сместить границы интегрирования при необходимости.

2. Составить подпрограмму-функцию *f* для вычисления подынтегральной функции.

3. Составить подпрограммы-функции RECT, TRAP, SIMPS для вычисления интегралов по формулам прямоугольников, трапеций и Симпсона соответственно на языке программирования Java.

4. Вычислить априорную погрешность и определить по ней верхнее значение шага *h* для достижения заданной точности.

5. Составить головную программу, содержащую оценку по Рунге погрешности каждой из перечисленных ранее квадратурных формул, удваивающих n до тех пор, пока погрешность не станет меньше *ε*, и осуществляющих печать результатов значения интеграла и значения *h* для каждой формулы.

6. Составить таблицу

|  |  |  |  |
| --- | --- | --- | --- |
|  | Метод прямоугольников | Метод трапеций | Метод Симпсона |
| eps |  |  |  |
| I |  |  |  |
| Априорная погрешность |  |  |  |
| Апостериорная погрешность |  |  |  |
| Верхняя оценка *h* |  |  |  |
| Значение *h*, полученное по методу Рунге |  |  |  |

7. Сделать выводы по полученным результатам.

## Выполнение работы.

**Вариант 6.**

|  |  |
| --- | --- |
| 6 |  |

1. Найти область определения функции и сместить границы интегрирования при необходимости.

Подкоренное выражение должно быть неотрицательно:

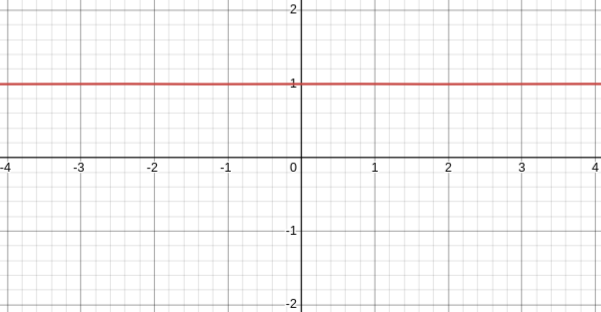
1 - (k\*sinx)^2 >=0

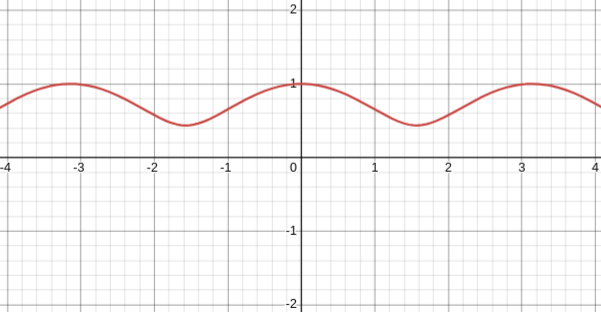
(sinx)^2 принимает значения в интервале [0;1].

При значении к, принадлежащем промежутку [-1;1] область определения функции - вся числовая прямая.

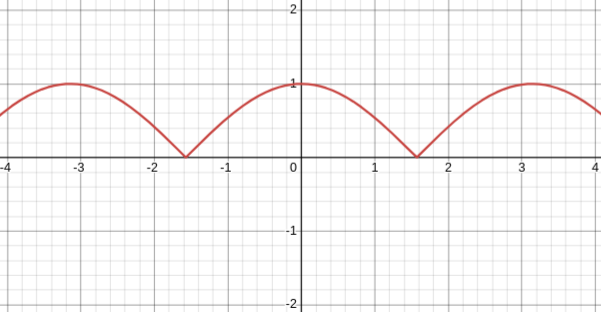
Поведение кривой при к, принадлежащем данному промежутку, варьируется от вида параллельной к ох прямой, проходящей через (0;1) при к = 0, к виду, указанному на третьем рисунке.

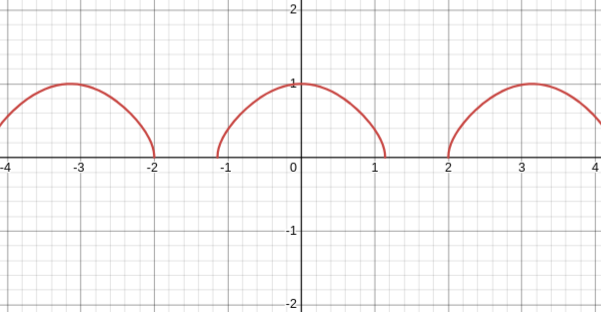
к = 0:

****

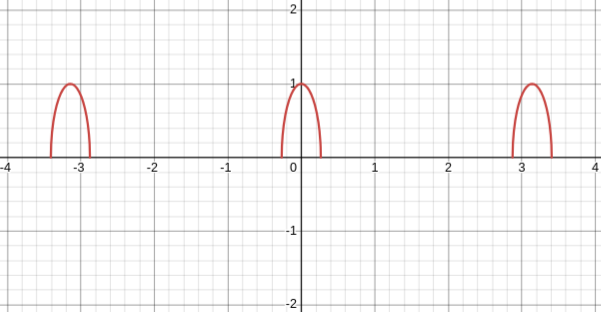
 **к = 0.7:**

**к = 1:**



**к** **= 1.1:**

**к = 3.8:**

Рассмотрим случай k = 0.7:

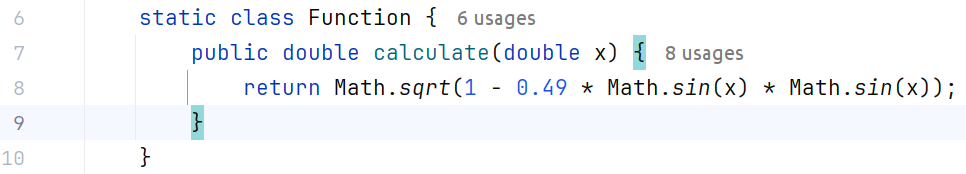
Функция f(x) определена на всей числовой прямой:

Область определения:

x∈(−∞,+∞)

Границы интегрирования a=0 и b=1 не требуют смещения, так как функция f(x) определена и непрерывна на всём интервале [0,1]**.**

2. Реализован класс под названием *Function* с методом *calculate(x)* для вычисления значения функции (см. Рис. 1)

*Рис. 1 - класс Function.*

3. Составлены классы *RECT*, *TRAP*, *SIMPS* с соответствующими методами для вычисления интегралов по формулам прямоугольников, трапеций и Симпсона соответственно на языке программирования Java (см. Рис. 2).

*Рис. 2 - реализованные классы с методами calculate по численному интегрированию разными способами.*

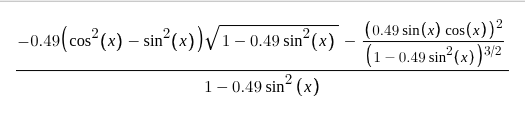
4. Вычисление априорной погрешности и определение по ней верхнего значения шага *h* для достижения заданной точности.

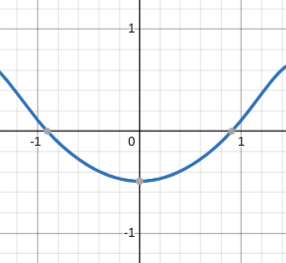
Сначала найдём шаг итегрирования *h* для вычисления интеграла с точностью .

1. Метод прямоугольников:

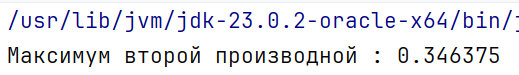
, найдём первую и втору производну (см. Рис. 3 и Рис. 4).

*Рисунок 3 - Первая производная функции.*

*Рис. 4 - Вторая производная функции.*

*Рис. 5 - График второй производной функции.*

Вторая функция монотонно возрастает на промежутке [0;1], следовательно наибольшее значение она принимает при х = 1.



Тогда .

, , , .

Найдём количество шагов, на которое нужно разделить отрезок с шагом для достижения точности .

, .

Возьмём количество частей отрезка *n = 4*. Следовательно, шаг интегрирования .

Вычислим априорную погрешность

1. Для формулы трапеции:

, , , .

Найдём количество шагов, на которое нужно разделить отрезок с шагом для достижения точности .

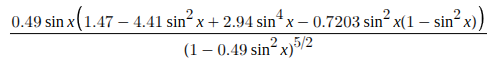
, .

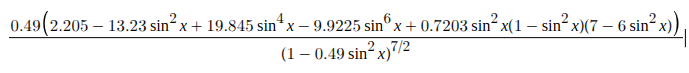
Возьмём количество частей отрезка кратное 4, т.е. *n = 8*. Следовательно, шаг интегрирования .

Посчитаем априорную погрешность:

1. Для формулы Симпсона:

, найдём четвёртую производную.

*Рис. 6 - Третья производная функции.*

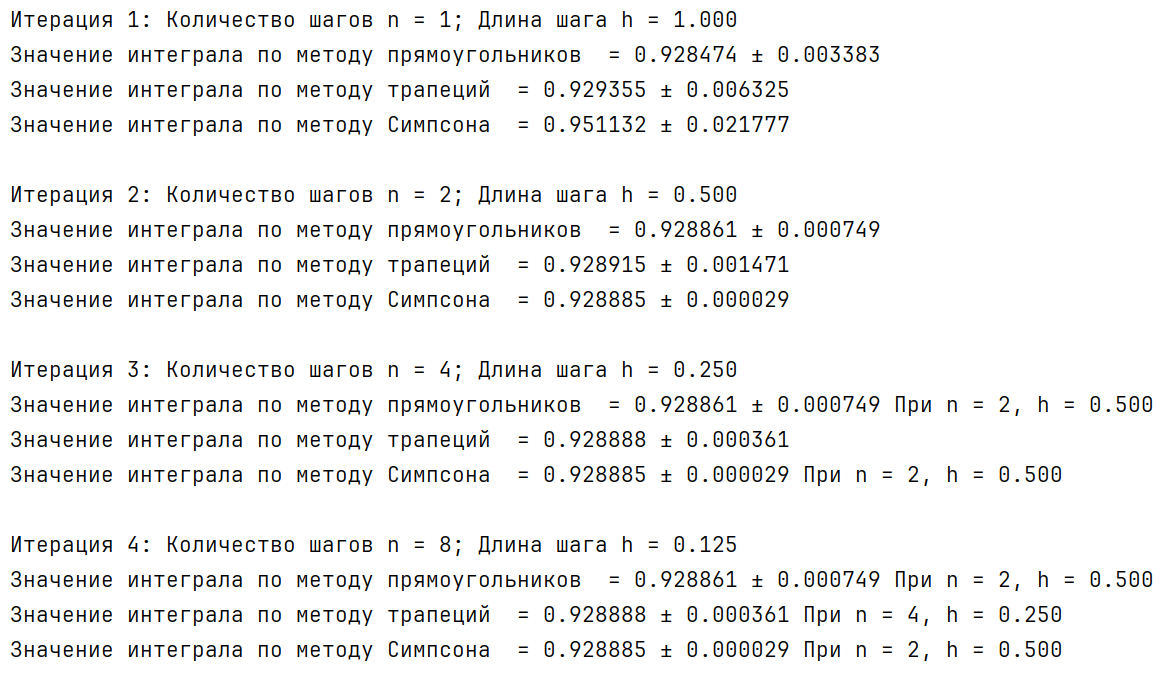
*Рис. 7 - Четверая производная функции.*

Тогда .

, , , .

В данном случае для для достижения точности достаточно одного отрезка длиной 1, шаг интегрирования = 1.

5. Составим головную программу, содержащую оценку по Рунге погрешности для следующих квадратурных формул, удваивающих *n* до тех пор, пока погрешность не станет меньше *ε*, и осуществляющих печать результатов значения интеграла и значения *h* для каждой формулы. Вывод работы программы см Рис. 8.

*Рис. 8 - результат работы программы.*

1. Формула прямоугыольников:
2. Формула трапеций:
3. Формула Симпсона:

Правило Рунге:

, для формул (1) и (2) порядок p = 2, для (3) — p = 3.

Результат можно видеть на том же рисунке 2.

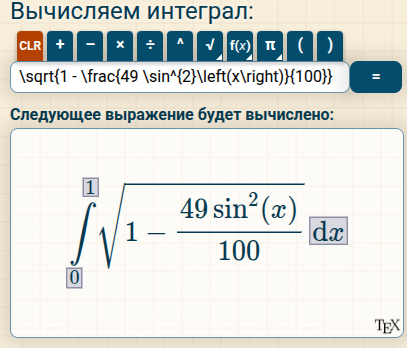
6. Составим таблицу

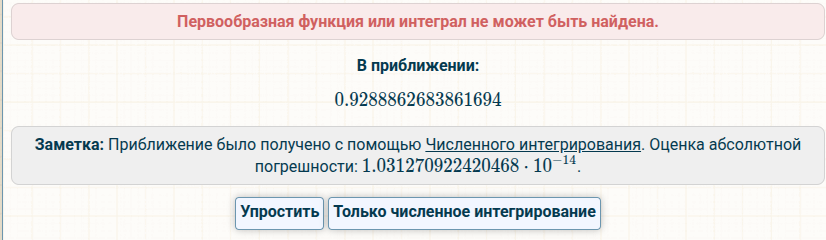
|  |  |  |  |
| --- | --- | --- | --- |
|  | Метод прямоугольников | Метод трапеций | Метод Симпсона |
| eps |  |  |  |
| I |  |  |  |
| Априорная погрешность |  |  |  |
| Апостериорная погрешность |  |  |  |
| Верхняя оценка *h* |  |  |  |
| Значение *h*, полученное по методу Рунге |  |  |  |

**Вывод.**

В ходе выполнения лабораторной работы были исследованы методы вычисления интеграла с помощью квадратурных формул: средних прямоугольников, трапеций, Симпсона.

Во всех случаях апостериорная погрешность меньше априорной, что свидетельствует о корректной реализации вычислений.

Кроме того, результат, полученный в ходе выполнения лабораторной работы совпадает с результатом, полученным на сайте https://www.integral-calculator.ru:



**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

package com.github.chelovekkrokant;  
  
import java.util.Locale;  
  
public class Main {  
 static class Function {  
 public double calculate(double x) {  
 return Math.*sqrt*(1 - 0.49 \* Math.*sin*(x) \* Math.*sin*(x));  
 }  
  
 public double calculateFirstDerivative(double x) {  
 double sinX = Math.*sin*(x);  
 double cosX = Math.*cos*(x);  
 double denominator = Math.*sqrt*(1 - 0.49 \* sinX \* sinX);  
 return (-0.49 \* sinX \* cosX) / denominator;  
 }  
  
 public double calculateSecondDerivative(double x) {  
 double sinX = Math.*sin*(x);  
 double cosX = Math.*cos*(x);  
 double term = 1 - 0.49 \* sinX \* sinX;  
 double sqrtTerm = Math.*sqrt*(term);  
  
 double numerator = -0.49 \* (cosX \* cosX - sinX \* sinX) \* sqrtTerm;  
 numerator += (0.49 \* 0.49 \* sinX \* sinX \* cosX \* cosX) / sqrtTerm;  
  
 return numerator / term;  
 }  
  
 public double calculateThirdDerivative(double x) {  
 double sin = Math.*sin*(x);  
 double cos = Math.*cos*(x);  
 double sin2 = sin \* sin;  
 double term = 1 - 0.49 \* sin2;  
  
 double numerator = 0.49 \* sin \* (1.47 - 4.41 \* sin2 + 2.94 \* Math.*pow*(sin, 4)  
 - 0.7203 \* sin2 \* (1 - sin2));  
 double denominator = Math.*pow*(term, 2.5);  
  
 return numerator / denominator;  
 }  
  
 *// Четвертая производная (аналитическая)*  
public double calculateFourthDerivative(double x) {  
 double sin = Math.*sin*(x);  
 double cos = Math.*cos*(x);  
 double sin2 = sin \* sin;  
 double term = 1 - 0.49 \* sin2;  
  
 double numerator = 0.49 \* (2.205 - 13.23 \* sin2 + 19.845 \* Math.*pow*(sin, 4)  
 - 9.9225 \* Math.*pow*(sin, 6) + 0.7203 \* sin2 \* (1 - sin2)  
 \* (7 - 6 \* sin2));  
 double denominator = Math.*pow*(term, 3.5);  
  
 return numerator / denominator;  
 }  
 }  
  
 *// Метод средних прямоугольников, основанный на суммировании площадей прямоугольников*  
 *// С высотой значений функции в средних точках интервалов*  
static class RECT {  
 public double calculate(double x, int n, double h) {  
 Function f = new Function();  
 double integralSum = 0;  
 for (int i = 0; i < n; ++i) {  
 integralSum += f.calculate(x + i \* h + h / 2);  
 }  
 return integralSum \* h;  
 }  
 }  
 *// Метод трапеций, осованный на суммировании площадей трапеций*  
 *// С перепадом высот, соответствующим значениям в крайних точкках*  
static class TRAP {  
 public double calculate(double x, int n, double h) {  
 Function f = new Function();  
 double integralSum = (f.calculate(x) + f.calculate(x + n \* h)) / 2;  
 for (int i = 1; i < n; ++i) {  
 integralSum += f.calculate(x + i \* h);  
 }  
 return integralSum \* h;  
 }  
 }  
 *// Метод Симпсона, основанный на суммировании площадей параболических трапеций*  
 *// Со "сглаженным" ребром, связывающим значения в крайних точках*  
static class SIMPS {  
 public double calculate(double x, int n, double h) {  
 Function f = new Function();  
 *// Вычисление четного количества интервалов*  
int m = n / 2;  
 double integralSum = f.calculate(x) + f.calculate(x + h \* n);  
 *// Умножение четных узлов на 2*  
for (int i = 1; i < m; ++i) {  
 integralSum += 2 \* f.calculate(x + 2 \* i \* h);  
 }  
 *// Умножение нечетных узлов на 4*  
for (int i = 0; i < m; ++i) {  
 integralSum += 4 \* f.calculate(x + (2 \* i + 1) \* h);  
 }  
 return integralSum \* h / 3; *// Формула Симпсона*  
}  
 }  
  
 public static void main(String[] args) {  
 *// Настройка локали для корректного вывода чисел с точкой*  
Locale.*setDefault*(Locale.*US*);  
  
 *// Инициализация переменных*  
int n = 0;  
 *// Текущее количество интервалов*  
int next\_n = 1;  
 *// Следующее количество интервалов*  
int iter = 1;  
 *// Счетчик итераций*  
double a = 0;  
 *// Нижний предел интегрирования*  
double b = 1;  
 *// Верхний предел интегрирования*  
double eps = 0.001;  
 *// Требуемая точность*  
double h;  
 *// Шаг интегрирования*  
  
 *// Переменные для хранения результатов и погрешностей*  
double INT\_VAL\_PR = 0, INT\_VAL\_TR = 0, INT\_VAL\_SI = 0;  
 double RES\_INT\_VAL\_PR = 0, RES\_INT\_VAL\_TR = 0, RES\_INT\_VAL\_SI = 0;  
 double H\_VAL\_PR = 0, H\_VALUE\_TR = 0, H\_VAL\_SI = 0;  
  
 *// Флаги для контроля точности методов*  
boolean flagP = true, flagTR = true, flagS = true;  
  
 *// Переменные для хранения оптимальных параметров*  
int N\_VAL\_PR = 0, N\_VALUE\_TR = 0, N\_VAL\_SI = 0;  
  
 *// Создание экземпляров классов методов интегрирования*  
RECT IntP = new RECT();  
 TRAP IntTR = new TRAP();  
 SIMPS IntS = new SIMPS();  
 Function ff = new Function();  
 *// Основной цикл*  
while (next\_n != n) {  
 n = next\_n;  
 h = (b - a) / n; *// Вычисление текущего шага*  
  
 *// Вывод информации о текущей итерации*  
System.*out*.printf("Итерация %d:\tКоличество шагов n = %d; Длина шага h = %.3f\n", iter, n, h);  
  
 *// Блок обработки метода прямоугольников*  
if (flagP) {  
 H\_VAL\_PR = h; *// Сохраняем текущий шаг*  
N\_VAL\_PR = n; *// Сохраняем текущее n*  
 *// Уточнение результата по Рунге*  
INT\_VAL\_PR = IntP.calculate(a, 2 \* n, h / 2) +  
 (IntP.calculate(a, 2 \* n, h / 2) - IntP.calculate(a, n, h)) / 3;  
 next\_n = 2 \* n; *// Удваиваем количество интервалов*  
 *// Вычисление текущей погрешности*  
System.*out*.printf("Значение интеграла по методу прямоугольников = %.6f ± %.6f\n",  
 INT\_VAL\_PR, Math.*abs*(IntP.calculate(a, n, h) - IntP.calculate(a, 2 \* n, h / 2)) / 3);  
 } else {  
 *// Вывод финального результата для метода*  
RES\_INT\_VAL\_PR = Math.*abs*(IntP.calculate(a, N\_VAL\_PR, H\_VAL\_PR) - IntP.calculate(a, 2 \* N\_VAL\_PR, H\_VAL\_PR / 2)) / 3;  
 System.*out*.printf("Значение интеграла по методу прямоугольников = %.6f ± %.6f При n = %d, h = %.3f\n",  
 INT\_VAL\_PR, RES\_INT\_VAL\_PR, N\_VAL\_PR, H\_VAL\_PR);  
 }  
 *// Проверка достижения требуемой точности*  
if (Math.*abs*(IntP.calculate(a, n, h) - IntP.calculate(a, 2 \* n, h / 2)) / 3 < eps)  
 flagP = false;  
  
 *// Блок для метода трапеций*  
if (flagTR) {  
 H\_VALUE\_TR = h;  
 N\_VALUE\_TR = n;  
 INT\_VAL\_TR = IntTR.calculate(a, 2 \* n, h / 2) +  
 (IntTR.calculate(a, 2 \* n, h / 2) - IntTR.calculate(a, n, h)) / 3;  
 next\_n = 2 \* n;  
 System.*out*.printf("Значение интеграла по методу трапеций = %.6f ± %.6f\n",  
 INT\_VAL\_TR, Math.*abs*(IntTR.calculate(a, n, h) - IntTR.calculate(a, 2 \* n, h / 2)) / 3);  
 } else {  
 RES\_INT\_VAL\_TR = Math.*abs*(IntTR.calculate(a, N\_VALUE\_TR, H\_VALUE\_TR) - IntTR.calculate(a, 2 \* N\_VALUE\_TR, H\_VALUE\_TR / 2)) / 3;  
 System.*out*.printf("Значение интеграла по методу трапеций = %.6f ± %.6f При n = %d, h = %.3f\n",  
 INT\_VAL\_TR, RES\_INT\_VAL\_TR, N\_VALUE\_TR, H\_VALUE\_TR);  
 }  
 if (Math.*abs*(IntTR.calculate(a, n, h) - IntTR.calculate(a, 2 \* n, h / 2)) / 3 < eps)  
 flagTR = false;  
  
 *// Блок для метода Симпсона*  
if (flagS) {  
 H\_VAL\_SI = h;  
 N\_VAL\_SI = n;  
 INT\_VAL\_SI = IntS.calculate(a, 2 \* n, h / 2) +  
 (IntS.calculate(a, 2 \* n, h / 2) - IntS.calculate(a, n, h)) / 15;  
 next\_n = 2 \* n;  
 System.*out*.printf("Значение интеграла по методу Симпсона = %.6f ± %.6f\n",  
 INT\_VAL\_SI, Math.*abs*(IntS.calculate(a, n, h) - IntS.calculate(a, 2 \* n, h / 2)) / 15);  
 } else {  
 *// Вывод финального результата для метода*  
RES\_INT\_VAL\_SI = Math.*abs*(IntS.calculate(a, N\_VAL\_SI, H\_VAL\_SI) - IntS.calculate(a, 2 \* N\_VAL\_SI, H\_VAL\_SI / 2)) / 15;  
 System.*out*.printf("Значение интеграла по методу Симпсона = %.6f ± %.6f При n = %d, h = %.3f\n",  
 INT\_VAL\_SI, RES\_INT\_VAL\_SI, N\_VAL\_SI, H\_VAL\_SI);  
 }  
 if (Math.*abs*(IntS.calculate(a, n, h) - IntS.calculate(a, 2 \* n, h / 2)) / 15 < eps)  
 flagS = false;  
  
 System.*out*.println();  
 iter++;  
 }  
 }  
}