Университет ИТМО, факультет ПИиКТ

Лабораторная работа №2

Дисциплина: Вычислительная математика

Вариант 17

Выполнил: Чайка Алексей

Группа: Р3214

Преподаватель: Малышева Т. А.

Санкт-Петербург, 2020 год

**Текст задания:**

**Лабораторная работа 2. «Численное интегрирование»**

**Исходные данные:**

1. Пользователь выбирает функцию, интеграл которой требуется вычислить (3-5 функций), из тех, которые предлагает программа.

2. Пределы интегрирования задаются пользователем.

3. Точность вычисления задаются пользователем.

4. Начальное значение числа разбиения интервала интегрирования: n=4.

5. Ввод исходных данных осуществляется с клавиатуры.

**Программная реализация задачи:**

1. Реализовать в программе методы по выбору пользователя:

· Метод прямоугольников (3 модификации: левые, правые, средние)

· Метод трапеций

· Метод Симпсона

2. Методы должны быть оформлены в виде отдельной(ого) функции/класса.

3. Вычисление значений функции оформить в виде отдельной(ого) функ-ции/класса.

4. Предусмотреть вывод результатов: значение интеграла, число разбиения интер-вала интегрирования для достижения требуемой точности.

**Вычислительная реализация задачи:**

1. Вычислить интеграл, приведенный в таблице (столбец 3), точно.

2. Вычислить интеграл методом, указанным в таблице (столбец 4) при n=10.

3. Определить абсолютную погрешность вычислений.

4. В отчете отразить последовательные вычисления.

**Цель работы:** найти приближенное значение определенного интеграла с требуемой точностью различными численными методами.

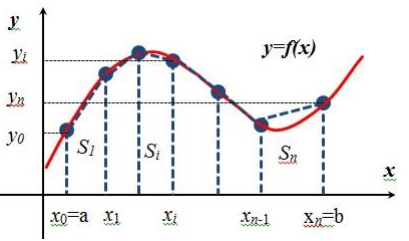
**Ход работы:**

**Метод трапеций:**

Подынтегральную функцию на каждом отрезке заменяют интерполяционным многочленом первой степени:

Используют линейную интерполяцию, т.е. график функции представляется в виде ломаной, соединяющий точки . Площадь всей фигуры (криволинейной трапеции):

Складывая все эти равенства, получаем формулу трапеций для численного интегрирования:

При формула трапеций:

Или

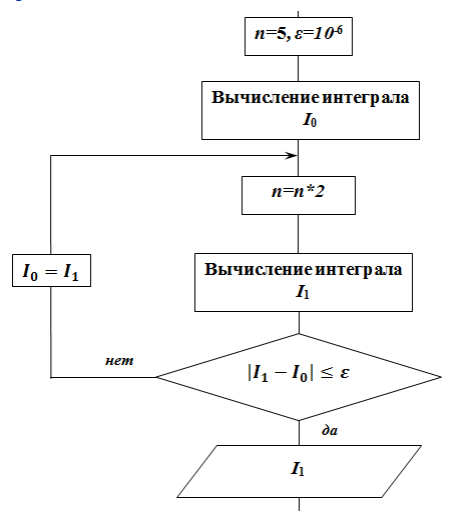
**Листинг:**

**import** lombok.Data;  
**import** org.apache.commons.math3.analysis.UnivariateFunction;  
**import** org.apache.commons.math3.analysis.integration.TrapezoidIntegrator;  
  
@Data  
**public class** TrapezoidMethod {  
 **private** UnivariateFunction **univariateFunction**;  
 **private** TrapezoidIntegrator **trapezoidIntegrator**;  
 **private double lowerBound**;  
 **private double upperBound**;  
 **private double precision**;  
 **private double answer**;  
 **private int iterations**;  
 **private double absoluteAccuracy**;  
 **private boolean swapped**;  
  
  
 **public** TrapezoidMethod(**double** lowerBound, **double** upperBound, **double** precision, **boolean** swapped) {  
 **this**.**lowerBound** = lowerBound;  
 **this**.**upperBound** = upperBound;  
 **this**.**precision** = precision;  
 **this**.**swapped** = swapped;  
 **trapezoidIntegrator** = **new** TrapezoidIntegrator(precision, precision, 4, 64);  
 }  
  
  
 **public void** solve() {  
 **answer** = **trapezoidIntegrator**.integrate(100, **univariateFunction**, **lowerBound**, **upperBound**);  
 **if** (**swapped**) **answer** = -**answer**;  
 **iterations** = **trapezoidIntegrator**.getIterations();  
 **absoluteAccuracy** = **trapezoidIntegrator**.getAbsoluteAccuracy();  
 }  
}

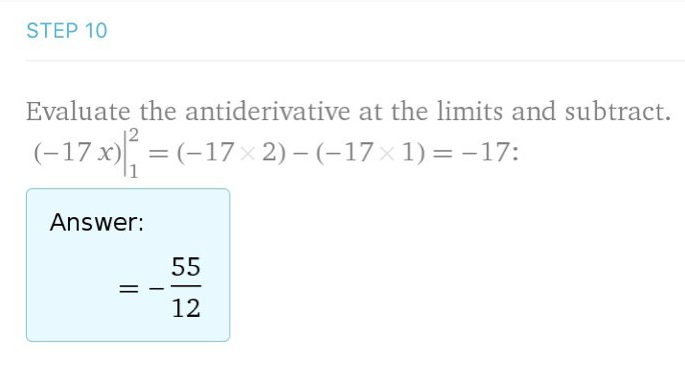
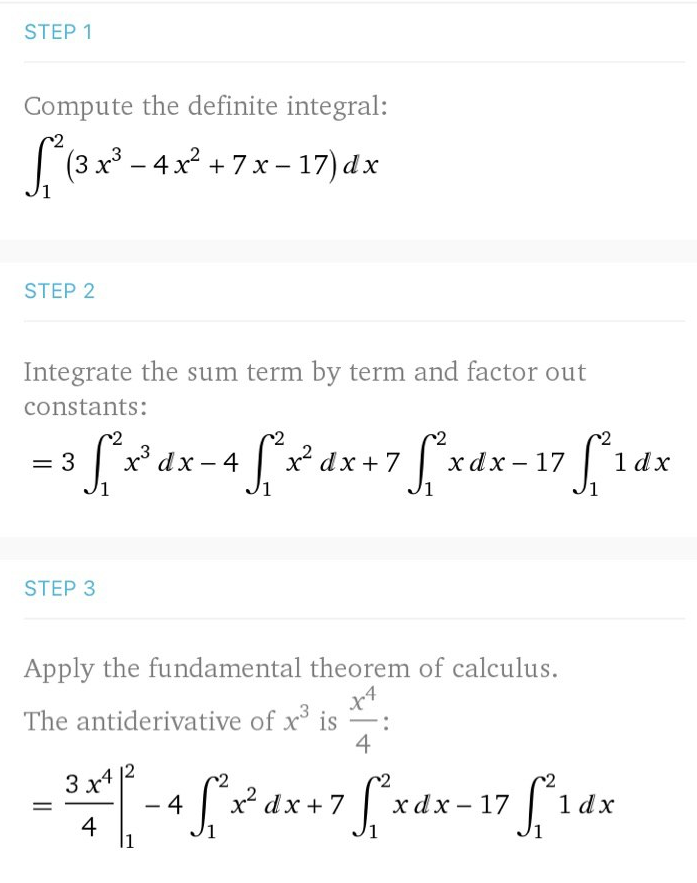
**import** org.apache.commons.math3.analysis.UnivariateFunction;  
**import** pepe.lmao.trapezoid.TrapezoidMethod;  
**import** java.util.Scanner;  
  
**public class** Main {  
 **public static void** main(String[] args) {  
 Scanner scanner = **new** Scanner(System.***in***);  
  
 **int** functionNumber;  
 **double** lower;  
 **double** upper;  
 **double** eps;  
 **double** answer = 0;  
 **int** iterations = 0;  
 **double** absoluteAccuracy = 0;  
 **boolean** swapped = **false**;  
 System.***out***.println(**"Pick a function:\n"** +  
 **"1. (x^2-1)/6\n"** +  
 **"2. x^3+x^2-1\n"** +  
 **"3. 3x^3-4x^2+6x\n"** +  
 **"4. -x^3-x^2+x+3\n"**);  
 functionNumber = scanner.nextInt();  
 System.***out***.println(**"Set upper bound:"**);  
 upper = scanner.nextDouble();  
 System.***out***.println(**"Set lower bound:"**);  
 lower = scanner.nextDouble();  
 System.***out***.println(**"Set precision:"**);  
 eps = scanner.nextFloat();  
 **if** (upper < lower) {  
 upper = *swap*(lower, lower=upper);  
 System.***out***.println(**"Bounds were swapped!"**);  
 System.***out***.println(**"Upper bound: "** + upper + **"\nLower bound: "** + lower);  
 swapped = **true**;  
 }  
 **if** (upper - lower < eps) {  
 System.***out***.println(answer);  
 } **else** {  
  
 TrapezoidMethod trapezoidMethod = **new** TrapezoidMethod(lower, upper, eps, swapped);  
 UnivariateFunction univariateFunction;  
 **switch** (functionNumber) {  
 **case** 1:  
 univariateFunction = x -> (Math.*pow*(x, 2) - 1) / 6;  
 trapezoidMethod.setUnivariateFunction(univariateFunction);  
 trapezoidMethod.solve();  
 answer = trapezoidMethod.getAnswer();  
 iterations = trapezoidMethod.getIterations();  
 absoluteAccuracy = trapezoidMethod.getAbsoluteAccuracy();  
 **break**;  
 **case** 2:  
 univariateFunction = x -> Math.*pow*(x, 3) + Math.*pow*(x, 2) - 1;  
 trapezoidMethod.setUnivariateFunction(univariateFunction);  
 trapezoidMethod.solve();  
 answer = trapezoidMethod.getAnswer();  
 iterations = trapezoidMethod.getIterations();  
 absoluteAccuracy = trapezoidMethod.getAbsoluteAccuracy();  
  
 **break**;  
 **case** 3:  
 univariateFunction = x -> 3 \* Math.*pow*(x, 3) - 4 \* Math.*pow*(x, 2) + 6 \* x;  
 trapezoidMethod.setUnivariateFunction(univariateFunction);  
 trapezoidMethod.solve();  
 answer = trapezoidMethod.getAnswer();  
 iterations = trapezoidMethod.getIterations();  
 absoluteAccuracy = trapezoidMethod.getAbsoluteAccuracy();  
  
 **break**;  
 **case** 4:  
 univariateFunction = x -> -Math.*pow*(x, 3) - Math.*pow*(x, 2) + x + 3;  
 trapezoidMethod.setUnivariateFunction(univariateFunction);  
 trapezoidMethod.solve();  
 answer = trapezoidMethod.getAnswer();  
 iterations = trapezoidMethod.getIterations();  
 absoluteAccuracy = trapezoidMethod.getAbsoluteAccuracy();  
  
 **break**;  
 **default**:  
 System.***out***.println(**"Illegal behavior"**);  
 **break**;  
 }  
  
  
 }  
  
 System.***out***.println(**"Answer: "** + answer + **"\n"** +  
 **"Number of iterations: "** + iterations + **"\n"** +  
 **"Absolute accuracy: "** + absoluteAccuracy);  
  
 }  
  
 **public static double** swap(**double**... args) {  
 **return** args[0];  
 }  
}

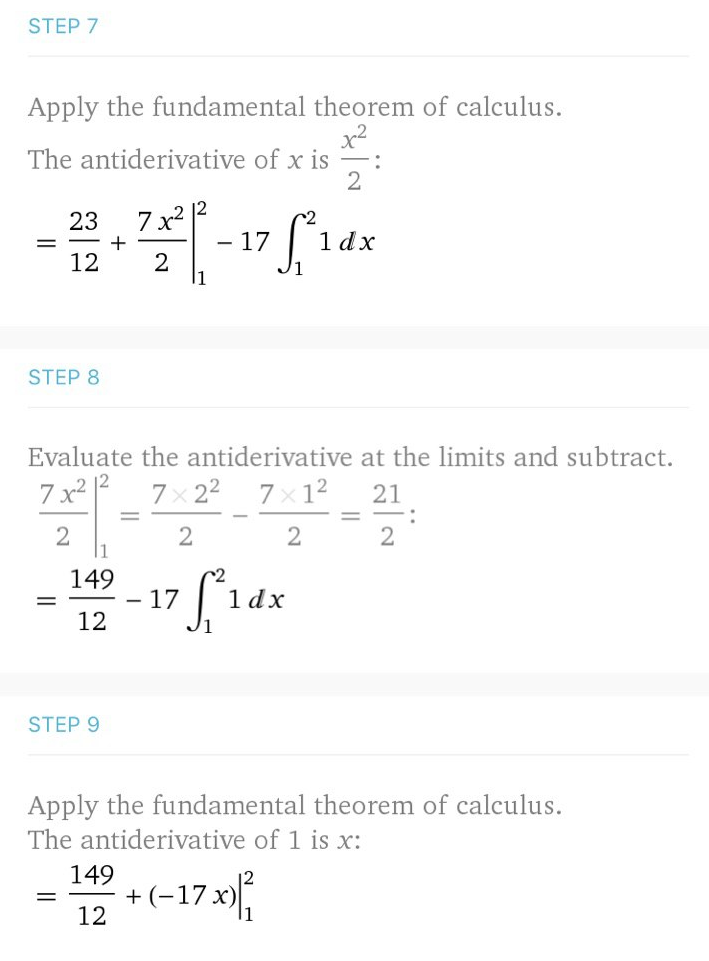
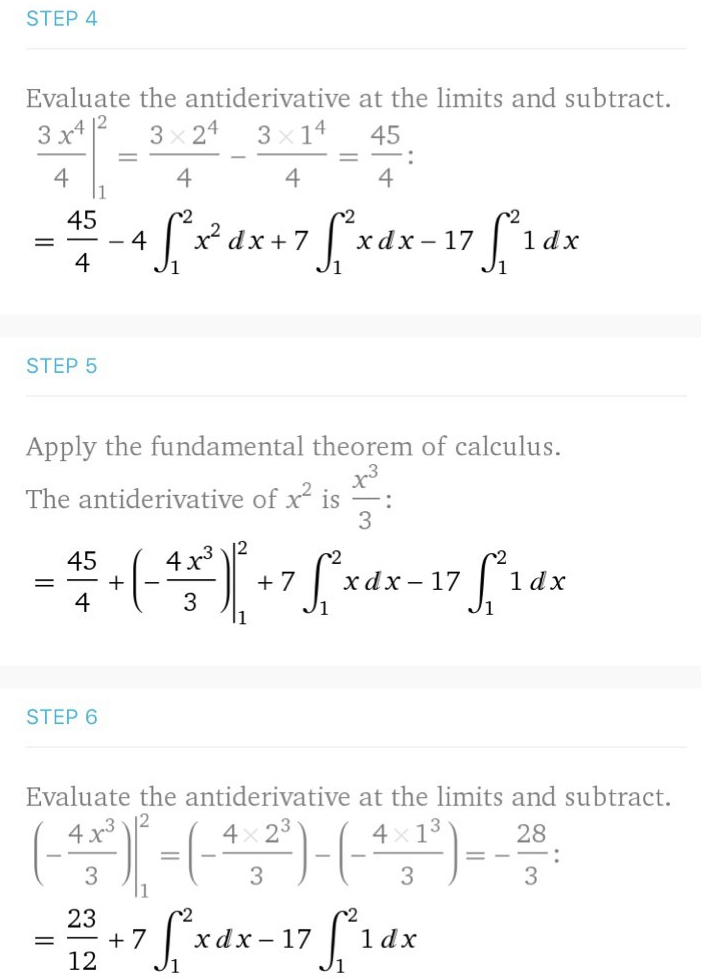
**Примеры работы программы:**

|  |  |
| --- | --- |
| Pick a function:  1. (x^2-1)/6  2. x^3+x^2-1  3. 3x^3-4x^2+6x  4. -x^3-x^2+x+3  2  Set upper bound:  1  Set lower bound:  2  Set precision:  0.001  Bounds were swapped!  Upper bound: 2.0  Lower bound: 1.0  Answer: -5.084228515625  Number of iterations: 5  Absolute accuracy: 0.0010000000474974513 | Pick a function:  1. (x^2-1)/6  2. x^3+x^2-1  3. 3x^3-4x^2+6x  4. -x^3-x^2+x+3  1  Set upper bound:  4  Set lower bound:  2  Set precision:  0.0001  Answer: 2.77783203125  Number of iterations: 6  Absolute accuracy: 9.999999747378752E-5 |
|  |  |

**Блок-схема:**

**Вычисления:**

*****Точно:*

****

*Метод Симпсона:*

*При*



*Погрешность вычислений составляет:*

**Вывод:** в результате выполнения работы я познакомился с методами решения интегралов путем реализации Java-программы. Освоил метод трапеций и на практике понял. Научился пользоваться методом Симпсона на листике, и с уверенностью могу сказать, что мне это пригодится в будущем.

**Вывод:** в результате выполнения лабораторной работы я познакомился с методом простых итераций для решения систем линейных уравнений, а также сильно постарался, для того, чтобы реализовать данный алгоритм на языке программирования JAVA.