**Санкт-Петербургский Национальный Исследовательский Университет ИТМО**

**Лабораторная работа №4**

по дисциплине

*«Вычислительная математика»*

Выполнил: Анисимов М. Д.

Группа: Р3233

Преподаватель: Перл О. В.

Санкт-Петербург

2024 г.

**Цель работы**

Разработать программу, которая производит интегрирование методом Симпсона. В качестве входных данных программа принимает границы a и b интеграла, номер интегрального уравнения и приближение, до которого мы должны вычислять уравнение. Выводом является вещественное число

**Описание метода**

Метод основан на разбиении графика подинтегральной функции на трапеции, в которой верхнее основание каждой является параболой, накладываемой на сам график. Функция делится на отрезки. От количества отрезков зависит точность вычисления нашего интеграла. Общая формула Симпсона выглядит так: ,

где:  
 – длина каждого из маленьких отрезков,

*,*

*–* сумма членов с чётными индексами умножается на 2,

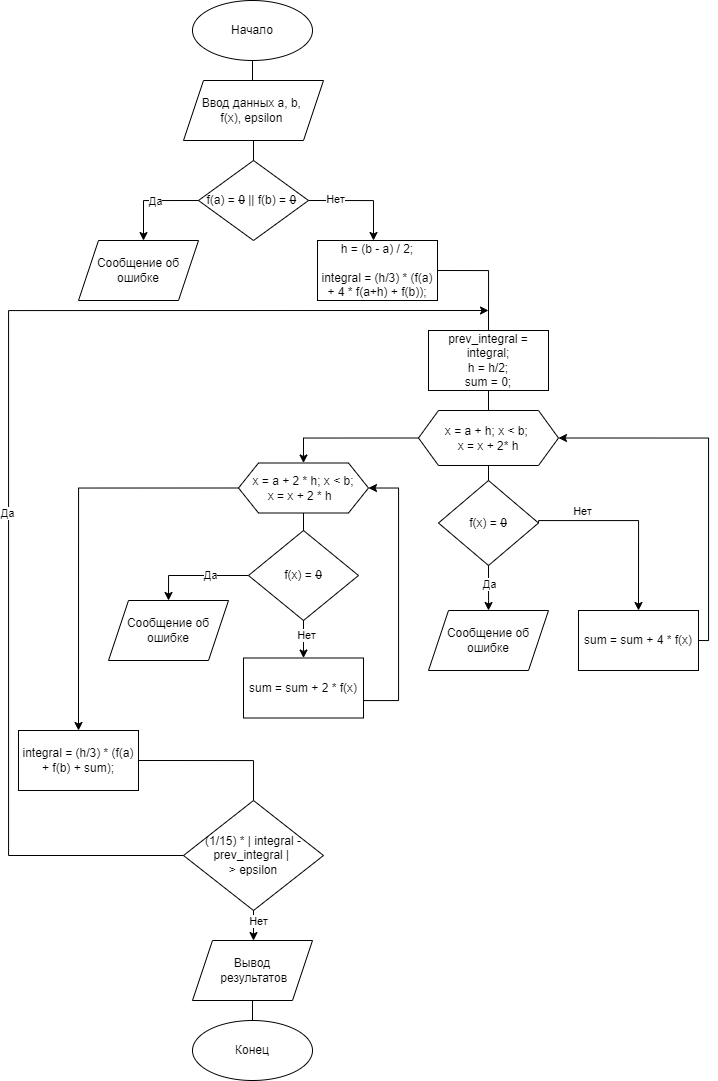
*–* сумма членов с нечётными индексами умножается на 4

**Код программы**

class Result {  
 public static String *error\_message* = "";  
 public static boolean *has\_discontinuity* = false;  
  
 private static double first\_function(double x) {  
 return 1 / x;  
 }  
  
 private static double second\_function(double x) {  
 return *sin*(x) / x;  
 }  
  
 private static double third\_function(double x) {  
 return x \* x + 2;  
 }  
  
 private static double fourth\_function(double x) {  
 return 2 \* x + 2;  
 }  
  
 private static double five\_function(double x) {  
 return *log*(x);  
 }  
  
 private static Function<Double, Double> get\_function(int n) {  
 switch (n) {  
 case (1):  
 return Result::*first\_function*;  
 case (2):  
 return Result::*second\_function*;  
 case (3):  
 return Result::*third\_function*;  
 case (4):  
 return Result::*fourth\_function*;  
 case (5):  
 return Result::*five\_function*;  
 default:  
 throw new UnsupportedOperationException("Function " + n + " not defined.");  
 }  
 }  
  
 public static double calculate\_integral(double a, double b, int f, double epsilon) {  
 Result.*has\_discontinuity* = false;  
  
 Function<Double, Double> func = *get\_function*(f);  
  
 if (Double.*isNaN*(func.apply(a)) || Double.*isNaN*(func.apply(b))|| Double.*isInfinite*(func.apply(a)) || Double.*isInfinite*(func.apply(b))) {  
 Result.*has\_discontinuity* = true;  
 Result.*error\_message* = "Integrated function has discontinuity or does not defined in current interval";  
 return 0.0;  
 }  
  
 double h = (b - a) / 2;  
  
 double integral = (h / 3) \* (func.apply(a) + 4 \* func.apply(a + h) + func.apply(b));  
  
 double prev\_integral = integral;  
 do {  
 prev\_integral = integral;  
 h /= 2;  
 double sum = 0;  
 for (double x = a + h; x < b; x += 2 \* h) {  
 if (Double.*isNaN*(func.apply(x)) || Double.*isInfinite*(func.apply(x))) {  
 Result.*has\_discontinuity* = true;  
 Result.*error\_message* = "Integrated function has discontinuity or does not defined in current interval";  
 return 0.0;  
 }  
 sum += 4 \* func.apply(x);  
 }  
 for (double x = a + 2 \* h; x < b; x += 2 \* h) {  
 if (Double.*isNaN*(func.apply(x)) || Double.*isInfinite*(func.apply(x))) {  
 Result.*has\_discontinuity* = true;  
 Result.*error\_message* = "Integrated function has discontinuity or does not defined in current interval";  
 return 0.0;  
 }  
 sum += 2 \* func.apply(x);  
 }  
 integral = (h / 3) \* (func.apply(a) + func.apply(b) + sum);  
 } while (((double) 1 /15) \*Math.*abs*(integral - prev\_integral) > epsilon);  
 if(integral <= 0 && a >b){  
 return integral;  
 }  
 else if((integral > 0 || integral < 0) && a <b){  
 return integral;  
 }  
 else if(integral > 0 && a > b){  
 return -integral;  
 }  
 else return integral;  
 }  
  
}

**Пример работы программы**

|  |  |
| --- | --- |
| Пример №1 |  |
| Пример №2 |  |
| Пример №3 |  |
| Пример №4 |  |
| Пример №5 |  |
| Пример №6 |  |
| Пример №7 |  |
| Примечания: | В приведённых выше примерах показана работа программы со всеми функциями, данными по условию (В случае, если пользователь ввёл номер несуществующей функции – программа выведет ошибку как в примере 6). Также показана работа программы в случаях разрыва второго рода (Вывод сообщения о невозможности подсчёта в примере 1). К сожалению, я поздно заметил, что в функции sin (x) / x присутствует разрыв первого рода, а не второго. По этой причине в примере 4 во время разрыва первого рода программа пишет о том, что произошёл разрыв второго рода. В примере 2 показана работа программы в случае, если a > b (Происходит вывод отрицательного результата). В примерах 3, 5, 7 показана стандартная работа программы с функциями |

**Блок-схема программы**

**Вывод**

Программа работает только с теми уравнениями, что даны по условию. Если пользователь вводит номер неизвестного уравнения, программа выдаст ошибку. Программа интегрирует в том случае, если в уравнении отсутствует разрыв первого и второго родов. К сожалению, в программе нет обработки случае разрыва первого рода, именно поэтому они воспринимаются как разрыв второго рода. В результате, вместо численного ответа выводится сообщение о невозможности подсчёта.

Метод Симпсона (Как и метод прямоугольников и метод трапеций) применяется в том случае, когда мы имеем дело с определённым интегралом, который невозможно посчитать через формулу Ньютона-Лейбница. В таких случаях и применяются приближённые вычисления.

Касательно разницы между методом прямоугольников, трапеций и Симпсона – все три метода отличаются приближением к нашей первоначальной функцией. Метод Симпсона даёт наилучшее приближение, так как график подынтегральной функции приближается маленькими параболами, что улучшает точность вычисления.

Алгоритмическая сложность метода Симпсона: O(1/epsilon) или O(log(1/epsilon)) (Зависит от выбора точности параметра epsilon)

Численная ошибка самого метода Симпсона зависит от количества отрезков, на которые мы делим подынтегральную функцию. Как правило: чем больше существует отрезков, тем выше точность вычисления