Университет ИТМО  
Факультет ПИиКТ

Лабораторная работа № 2

по предмету Вычислительная Математика

Интегрирование методом Симпсона

Выполнил:

Тарасов А.С.

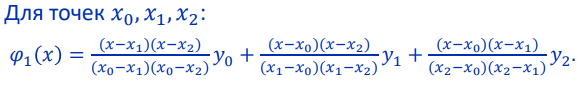
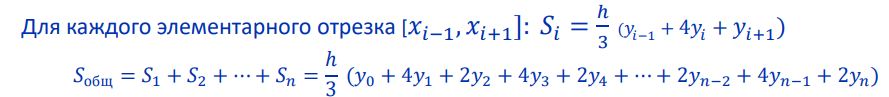
Преподаватель:

Перл О. В.

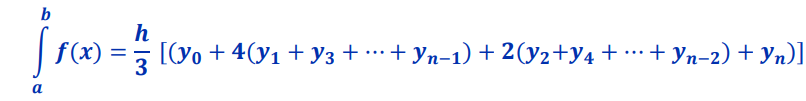
Санкт-Петербург – 2020

**Описание метода.**В данной работе будем производить численное интегрирование с помощью численного метода – метода Симпсона. Численное интегрирование применяется тогда, когда

- подынтегральная функция не задана аналитически  
 - первообразная функции не выражается аналитической функцией (пример: exp(-x2))

****Суть метода заключается в разбиении отрезка интегрирования [a;b] на четное число n равных отрезков и заменой подынтегральной функции на интерполяционный многочлен второй степени каждом таком из таких отрезков.   
В качестве 𝜑𝑖(𝑥) можно принять интерполяционный многочлен Лагранжа второй степени, проходящий через точки (𝑥𝑖−1, 𝑦𝑖−1), (𝑥𝑖 , 𝑦𝑖), (𝑥𝑖+1, 𝑦𝑖+1).

**Конечная формула – формула Симпсона:**

****

**Исходный код.**

UserInterface.java

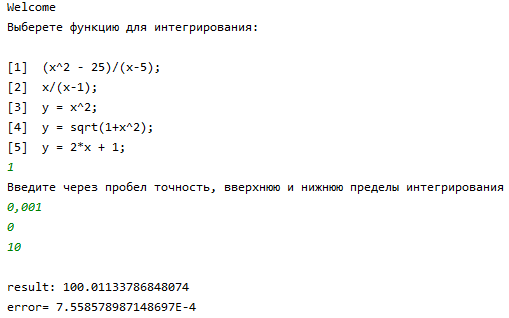
<…> **public static void** go(){  
  
 String[] functionsList = {**"(x^2 - 25)/(x-5)"**, **"x/(x-1)"**, **"y = x^2"**, **"y = sqrt(1+x^2)"**, **"y = 2\*x + 1"**};  
 System.***out***.println(**"Welcome\n"** +  
 **"Выберете функцию для интегрирования: \n"**);  
 **for** (**int** i = 0; i < 5; i++) {  
 System.***out***.println(**"["** + (i+1) +**"] "** + functionsList[i] + **";"**);  
 }  
 *in* = **new** Scanner(System.***in***);  
  
 Functions currentFunc = **null**;  
 **switch** (*in*.nextInt()){  
 **case** 1: {currentFunc = (n)-> (n\*n - 25)/(n-5); **break**;}  
 **case** 2: {currentFunc = (n)-> n/(n-1); **break**;}  
 **case** 3: {currentFunc = (n)-> n \* n; **break**;}  
 **case** 4: {currentFunc = (n)-> Math.*sqrt*(1 + Math.*pow*(n, 2)); **break**;}  
 **case** 5: {currentFunc = (n)-> 2\*n + 1; **break**;}  
 }  
  
 System.***out***.println(**"Введите через пробел точность, вверхнюю и нижнюю пределы интегрирования"**);  
 **while** (**true**){  
 *in* = **new** Scanner(System.***in***);  
 **try** {  
 *accuracy* = *in*.nextDouble();  
 *low* = *in*.nextInt();  
 *high* = *in*.nextInt();  
  
 **break**;  
 }**catch** (java.util.InputMismatchException ex){  
 System.***out***.println(**"Некорректный ввод данных.\n"** +  
 **"Введите через пробел точнсть, раздеяя целую и дробную частии запятой, \n"** +  
 **"вверхнюю и ннижнюю границы интегрирования"**);  
  
 }  
 }  
  
 SimpsonMethod method = **new** SimpsonMethod(*accuracy*, *high*, *low*, currentFunc);  
 System.***out***.println(method.solve());

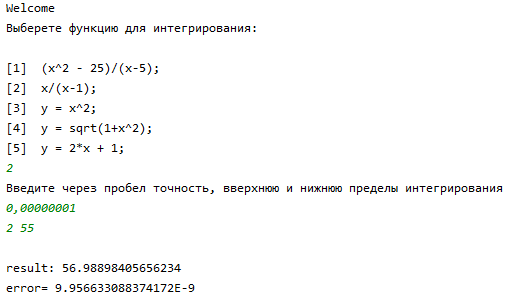
<…>

SimpsonMethod.java

**public class** SimpsonMethod {  
 **double high**, **low**;  
 **double accuracy**;  
 String **description** = **""**;  
 Functions **currentFunc** = **null**;  
  
 **public** SimpsonMethod(**double** accuracy, **int** high, **int** low, Functions currentFunc) {  
 **this**.**accuracy** = accuracy;  
 **this**.**currentFunc** = currentFunc;  
 **if** (low > high) {  
 **this**.**low** = high;  
 **this**.**high** = low;  
 }  
 **else**{  
 **this**.**high** = high;  
 **this**.**low** = low;  
 }  
 }  
 **public** String solve(){  
 **double** IntgrN, Intgr2N, h, result, error, stepsNumber;;  
  
 **if** (**high** != **low**)  
 {  
 **for** (**int** n = 4; n <= 10000; n += 2)  
 {  
 IntgrN = integrate(n);  
 Intgr2N = integrate(2 \* n);  
  
 **if** ((Math.*abs*(Intgr2N - IntgrN) / 15) < **accuracy**)  
 {  
 result = Intgr2N;  
 stepsNumber = n;  
 error = Math.*abs*(Intgr2N - IntgrN) / 15;  
 **description** += **"\nresult: "**+result+ **"\nerror= "** + error;  
 **break**;  
 }  
 **if** (n == 10000) { System.***out***.println(**"Заданная точность не достигнута. Интеграл не имеет решения."**); stepsNumber = 0; }  
 }  
 }  
 **else** { result = 0; stepsNumber = 0; System.***out***.println(**"Пределы интегрирования равны. Result = 0."**); }  
  
 **return description**;  
 }  
 **private double** integrate(**int** n)  
 {  
 **double** sum = 0;  
 **double** h = (**high** - **low**) / n; *//вычисление размера шага* **for** (**int** i = 1; i < n; i++)  
 {  
 sum += 4 \* **currentFunc**.solve(**low** + i \* h);  
 ++i;  
 sum += 2 \* **currentFunc**.solve(**low** + i \* h);  
 }  
 **return** (sum + **currentFunc**.solve(**low**) - **currentFunc**.solve(**high**)) \* h / 3;  
 }  
  
}

**Примеры.**

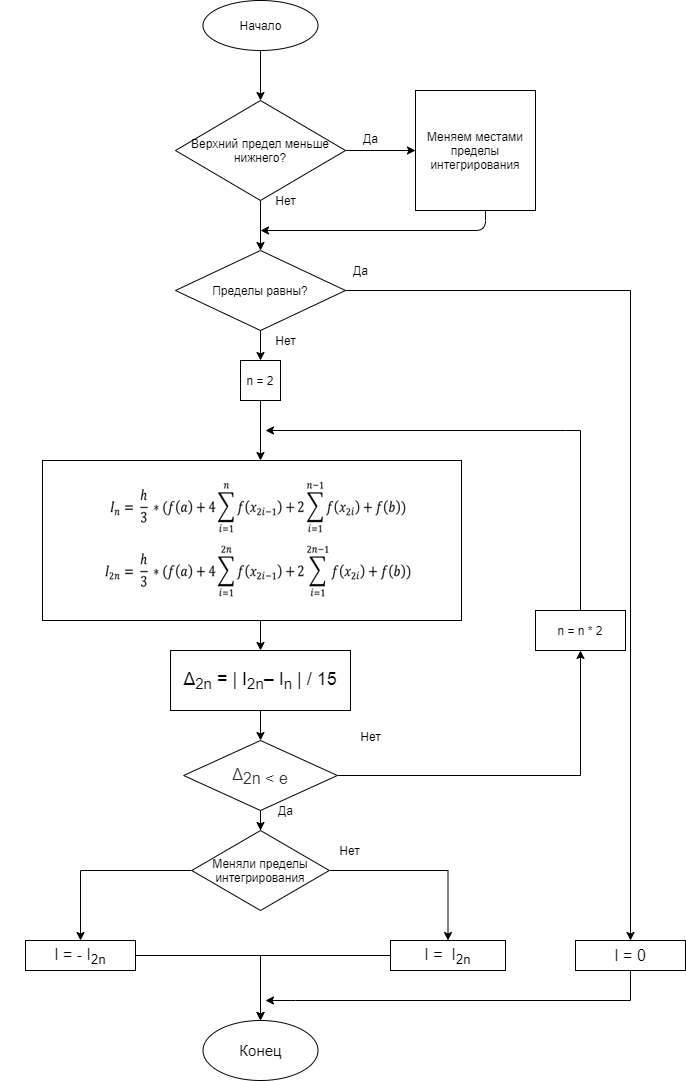
****

****

**Вывод.**

В данной работе реализован алгоритм численного интегрирования методом Симпсона, а также использовалось правило оценки Рунге этого метода.

Алгебраический порядок точности методов прямоугольников или трапеций (0 и 1, соответственно) меньше алгебраического порядка точности метода Симпсона (3). Также стоит отметить, что метод Симпсона дает более точный результат, чем методы прямоугольников или трапеций, так как в методе Симпсона используется квадратичная интерполяция, а не линейная.

**Блок-схема реализованного алгоритма**