**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**

**по практической работе №2**

**по дисциплине «Вычислительная математика»**

**Тема:**

**Исследование обусловленности задачи нахождения корня уравнения.**

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

Санкт-Петербург

2025

**Цель работы.**

Исследование обусловленности задачи нахождения корня уравнения.

**Теоретические положения.**

Под обусловленностью вычислительной задачи понимают чувствительность ее решения к малым погрешностям входных данных. Задачу называют хорошо обусловленной, если малым погрешностям входных данных отвечают малые погрешности решения, и плохо обусловленной, если возможны сильные изменения решения. Количественной мерой степени обусловленности вычислительной задачи является число обусловленности, которое можно интерпретировать как коэффициент возможного возрастания погрешностей в решении по отношению к вызвавшим их погрешностям входных данных. Пусть между абсолютными погрешностями входных данных x и решения y установлено неравенство:

,

где x\* и y\* - приближённые входные данные и приближённое решение соответственно. Тогда величина называется абсолютным числом обусловленности. Если же установлено неравенство

между относительными ошибками данных и решения, то величину называют относительным числом обусловленности. Для плохо обусловленной задачи 𝜈 ≫ 1. Грубо говоря, если , где 𝜈 − относительное число обусловленности, то порядок N показывает число верных цифр, которое может быть утеряно в результате по сравнению с числом верных цифр входных данных. Ответ на вопрос о том, при каком значении 𝜈 задачу следует признать плохо обусловленной, зависит, с одной стороны, от предъявляемых требований 3 к точности решения и, с другой, – от уровня обеспечиваемой точности исходных данных. Например, если требуется найти решение с точностью 0.1%, а входная информация задается с точностью 0.02%, то уже значение 𝜈 = 10 сигнализирует о плохой обусловленности. Однако, при тех же требованиях к точности результата, гарантия, что исходные данные задаются с точностью не ниже 0.0001%, означает, что при задача хорошо обусловлена. Если рассматривать задачу вычисления корня уравнения 𝑦 = 𝑓(𝑥), то роль числа обусловленности будет играть величина

,

где – корень уравнения. Из уравнения можно получить  *-* теоретическое количество итераций.

**Задание. Вариант 6.**

1) Нахождение области определения функции, локализация корня уравнения 𝑓(𝑥) = 0, взятие производной вручную.

2) Написание программы, включающей метод нахождения функции, производной на Java, вычисляющей простой корень уравнения по методу Бисекции с логическим условием .

3) Получение теоретического значения (используя логарифм по основанию 2 и ) и сравнение его с количеством итераций , полученной программой.

4) Вычисление , сравнение его с получение вывода об обусловленности задачи.

5) Проведение вычислений по программе, варьируя значения параметров.

6) Поменять условие окончания итераций на N (большое значение) и поймать интервал неопределенности по правилу Гарвика (, при нарушении начинается разболтка).

После каждого этапа сформировать таблицу и сделать анализ результатов:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *eps* | *delta* |  |  |  |  |  | Обусловленность |
|  |  |  |  |  |  |  |  |

План варьирования входных параметров:

1. eps постоянен и равен 0.01, delta варьируется от 0.00001 до 0.1.

3. delta постоянна и равна 0.01, eps варьируется от 0.000001 до 10.

4. delta и eps одновременно варьируются от 0.000001 до 1. Построить график зависимости eps от количества итераций. – если уже сделано, оставляем.

Упрощённый вариант: eps, который является условием остановки итераций меняем в диапазоне от 0.00001 и 0.1 и заполняем таблицу

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *eps* |  |  |  |  |  | Обусловленность |
|  |  |  |  |  |  |  |

**Выполнение работы.**

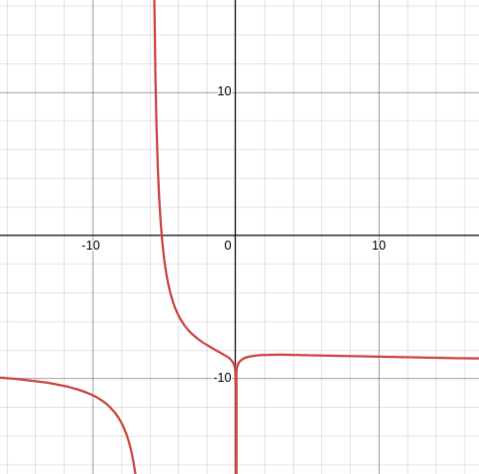
**Вариант 6:**

|  |  |
| --- | --- |
| 6 | [ln(x^(2)+4)/(x+6)]-9 |

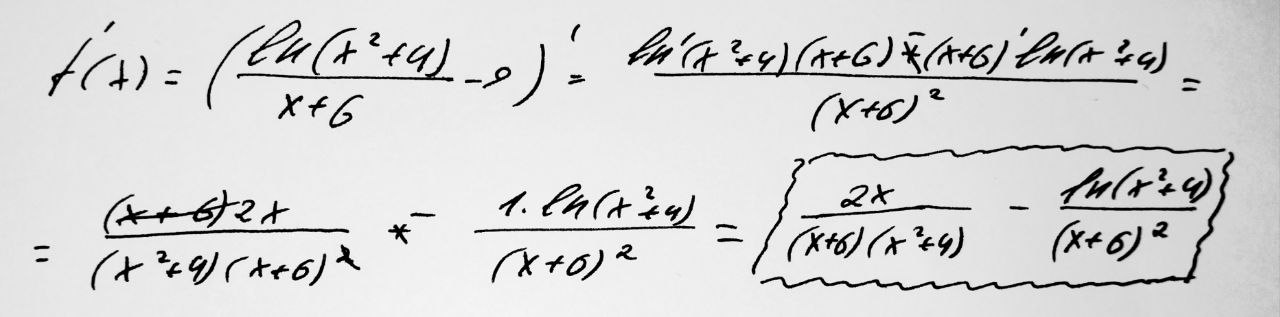
**1) Нахождение области определения функции, локализация корня уравнения 𝑓(𝑥) = 0, взятие производной вручную.**

**Область определения функции:**

**x != 6** - ввиду невозможности деления на ноль

 Аргумент логарифма всегда положителен, поэтому не накладывает ограничений на область опредления функции.

**Взятие производной функции вручную:**

 **Производная функции:**

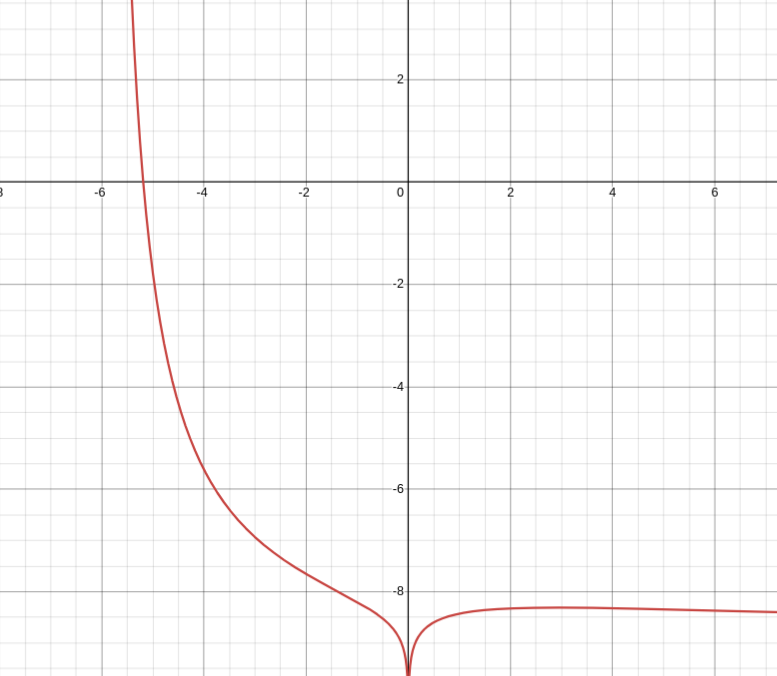
**Локализация корня:**

Производная терпитт разрыв в точке x = -6, после чего принимает отрицательное значение.

При x = -5.99 значение функции положительно.

При x = 0 значение функции отрицательно

Рассмотрим отрезок, ограниченный данными точками, в качестве отрезка локлизации.



**2) Написание программы, включающей метод нахождения функции, производной на Java, вычисляющей простой корень уравнения по методу Бисекции с логическим условием .**

Реализованы функции по нахождению значения функции при заданном х, производной при заданном х, корня при заданном отрезке локализации и eps:

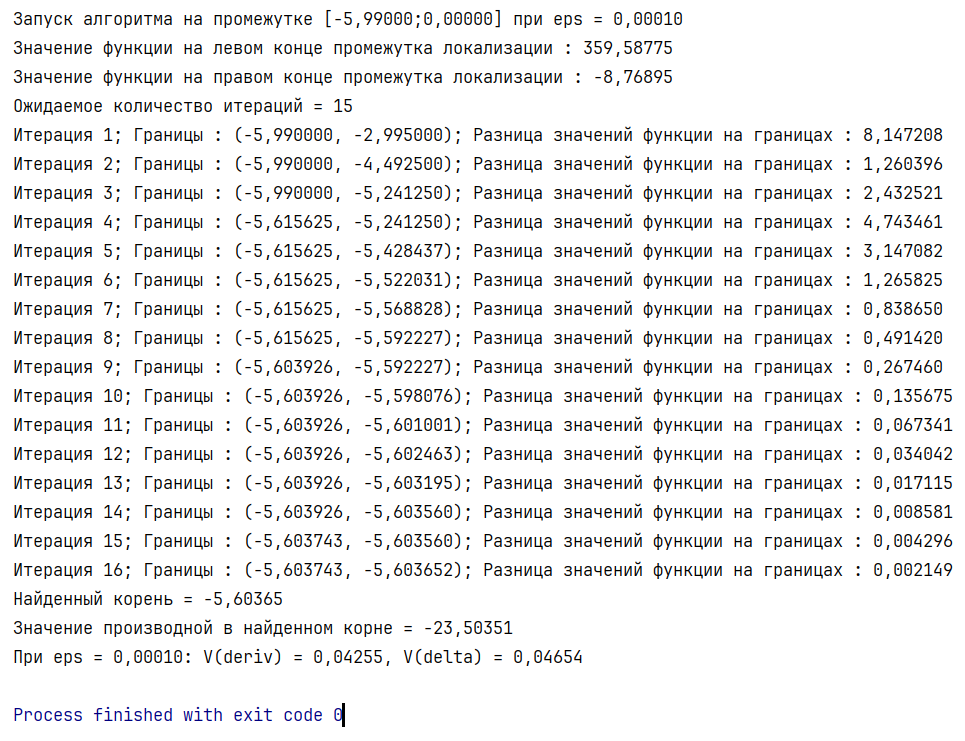
**static double function(double x){...}**

**static double derivativeFunction(double x){...}**

**static double findRootUsingBisection(double leftBorder, double rightBorder, double maxDifference){...}**

Для удобства дополнительно реализована функция по нахождению ожидаемого количества итераций:

**static int calculateTheoreticalNumberOfIterations(double a, double b, double eps){...}**

******

**3) Получение теоретического значения (используя логарифм по основанию 2 и ) и сравнение его с количеством итераций , полученной программой.**

Теоретическое значение N можно найти при помощи следующей формулы:

То есть, количество итераций должно быть не менее 15.

В случае реализованной функции количество итераций равно 16.

Теоретическое значение немного меньше полученного на практике.

Значение, вычисленное вручную, совпало со значением, вычисленным при помощи реализованной функции.

**4) Вычисление , сравнение его с получение вывода об обусловленности задачи.**

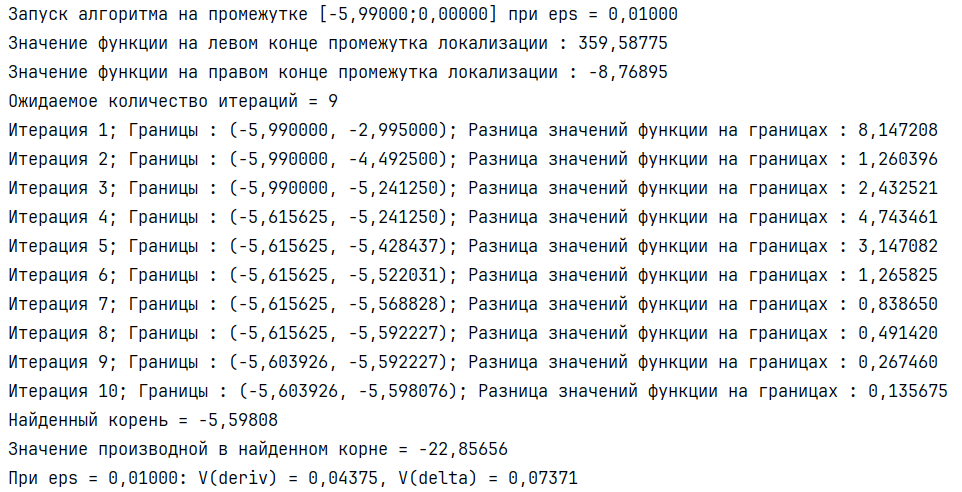
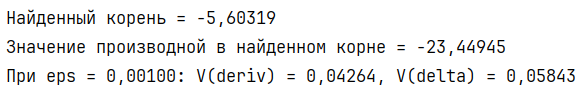
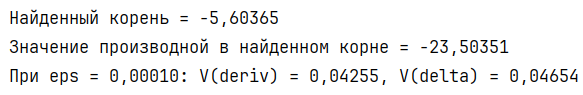
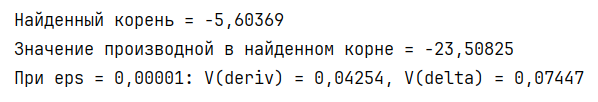
Значение производной в точке, соответствующей вычисленному выше корню, равно -23.5085679514835, следовательно абсолютное число обусловленности, вычисляемое по формуле , равно

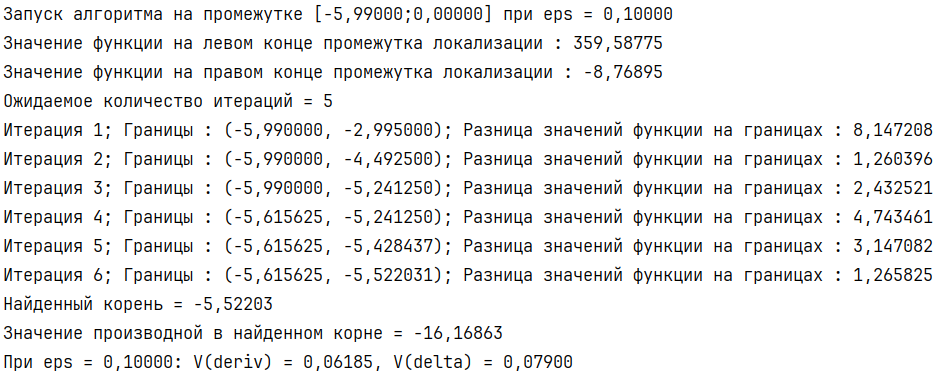
Задача является плохо обусловленной, если <<.

В нашем случае разница между вычисленными значениями не превышает и одного порядка (0.04654 / 0.04255 = 1.09377), из-за чего можно сделать вывод о хорошей обусловленности задаче при используемых данных.

**5) Проведение вычислений по программе, варьируя значения параметров.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *eps* |  |  |  |  |  | Обусловленность |
| 0.00001 | -5,60369 | 20 | 19 | 0,04254 | 0,07447 | Хорошая |
| 0.0001 | -5,60365 | 16 | 15 | 0,04255 | 0,04654 | Хорошая |
| 0.001 | -5,60319 | 13 | 12 | 0,04264 | 0,05843 | Хорошая |
| 0.01 | -5,59808 | 10 | 9 | 0,04375 | 0,07371 | Хорошая |
| 0.1 | -5,52203 | 6 | 5 | 0,06185 | 0,07900 | Хорошая |





6) Поменять условие окончания итераций на N (большое значение) и поймать интервал неопределенности по правилу Гарвика (, при нарушении начинается разболтка).



Разболтка началась после 53 операций, когда порядок разницы между значениями х стал равен порядку машинного эпсилон для double, найденного в первой работе.

**Интервал неопределенности равен 8.881784197001252E-16.**

**Вывод.**

В ходе выполнения данной практической работы была изучена проблема обусловленности задачи нахождения корня методом бисекции при различных начальных исходныхданных.

**ПРИЛОЖЕНИЕ А**

**Исходный код программы**

**Функция по вычислению значения исходной функции.**

static double function(double x){ return (Math.*log*(x\*x+4) / (x+6) - 9);}

**Функция по вычислению значения производной функции.**

static double derivativeFunction(double x){ return ((2 \* x) / ((x + 6) \* (x \* x + 4))) - ((Math.*log*(x \* x + 4)) / ((x + 6) \* (x + 6)));}

**Функция по вычислению предполагаемого количества итераций до нахождения корня.**

static int calculateTheoreticalNumberOfIterations(double a, double b, double eps){ double absDiff = Math.*abs*(b - a); *// Вычисляем |b - a|*int n = 0; *// Находим минимальное n, при котором |b - a| / (2^(n+1)) < eps*while (absDiff / Math.*pow*(2, n + 1) >= eps) { n++; } return n;}

**Функция по нахождению корня уравнения методом бисекции с вычислением и данных, необходимых в работе.**

static double findRootUsingBisection(double leftBorder, double rightBorder, double eps){ double middleValue = (leftBorder + rightBorder) / 2; double differenceBetweenFunctionValues = 0; double previousFunctionValue = 0; double currentFunctionValue = 0; System.*out*.printf("Запуск алгоритма на промежутке [%.5f;%.5f] при eps = %.5f\n", leftBorder, rightBorder, eps); System.*out*.printf("Значение функции на левом конце промежутка локализации : %.5f\n", *function*(leftBorder)); System.*out*.printf("Значение функции на правом конце промежутка локализации : %.5f\n", *function*(rightBorder)); System.*out*.println("Ожидаемое количество итераций = " + *calculateTheoreticalNumberOfIterations*(leftBorder, rightBorder, eps)); while(true) { *iterationCounter* += 1; if(Math.*abs*(rightBorder - leftBorder) >= eps){ middleValue = (leftBorder + rightBorder) / 2; if (*function*(leftBorder) \* *function*(middleValue) < 0){ rightBorder = middleValue; } else { if (*function*(middleValue) \* *function*(rightBorder) < 0) { leftBorder = middleValue; } } previousFunctionValue = currentFunctionValue; currentFunctionValue = *function*(middleValue); differenceBetweenFunctionValues = Math.*abs*(currentFunctionValue - previousFunctionValue); System.*out*.printf("Итерация %d; Границы : (%f, %f); Разница значений функции на границах : %f\n", *iterationCounter*, leftBorder, rightBorder, differenceBetweenFunctionValues); } else { break; } } System.*out*.printf("Найденный корень = %.5f\n", middleValue); System.*out*.printf("Значение производной в найденном корне = %.5f\n", *derivativeFunction*(middleValue)); System.*out*.printf("При eps = %.5f: V(deriv) = %.5f, V(delta) = %.5f\n", eps, 1 / Math.*abs*(*derivativeFunction*(middleValue)), eps / differenceBetweenFunctionValues); *iterationCounter* = 0; return middleValue;}

**Функция по нахождению интервала разболтки по Гарвикую**

static double findGarwickInterval(double leftBorder, double rightBorder, double eps){ double middleValue = (leftBorder + rightBorder) / 2; *iterationCounter* = 0; List<Double> midValues = new ArrayList<>(); double garWickInterval = 1; while(true) { middleValue = (leftBorder + rightBorder) / 2; midValues.add(middleValue); *iterationCounter* += 1; if (*function*(leftBorder) \* *function*(middleValue) < 0){ rightBorder = middleValue; } else { if (*function*(middleValue) \* *function*(rightBorder) < 0) { leftBorder = middleValue; } } System.*out*.println("Итерация " + *iterationCounter* + ". Интервал между последними значениями x = " + garWickInterval); if (midValues.size() >= 3){ double prevValue = midValues.getLast(); double prevPrevValue = midValues.get(midValues.size() - 2); double prevPrevPrevValue = midValues.get(midValues.size() - 3); garWickInterval = Math.*abs*(prevValue - prevPrevValue); if (Math.*abs*(prevValue - prevPrevValue) / Math.*abs*(prevPrevValue - prevPrevPrevValue) >= 1) { break; } } } System.*out*.printf("Найден интервал разболтки %f после %d итераций\n", garWickInterval, *iterationCounter*); *iterationCounter* = 0; return middleValue;}