**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Вычислительная математика»**

Тема: **Решение нелинейного уравнения методом бисекции и Ньютона**



|  |  |  |
| --- | --- | --- |
| Студент гр. 3343 |  | Коршков А.А. |
| Преподаватель |  | Попова Е.В. |

Санкт-Петербург

2025

**Цель работы.**

Изучение особенностей вычисления чисел с плавающей точкой.

**Задание.**

Вариант 10:

* Eps (начальное) = 10

1. С помощью языка программирования Java, онлайн-компилятора **Trinket** исследовать величину машинного эпсилон при заданных в варианте начальных значениях на определенных технических устройствах, указать эти технические устройства, привести скриншоты.
2. С помощью **метода границ** найти приближенное значение функции зависящей от двух аргументов, и величину абсолютной погрешности функции, если , . Предварительно определить область определения функции, отобразить эту область графически. (Если границы аргументов не попадают в область определения – замените их, аргументируя).
3. Для **нечетных** номеров: по неточному значению все цифры которого верны в строгом смысле, определить предельное значение относительной погрешности.

Для **четных** номеров: неточное число из предыдущего варианта (для 2-го из 1-го) имеет относительную погрешность . Определить количество верных знаков в широком смысле неточного числа.

1. Исследовать ошибки, связанные с числами с плавающей точкой.
2. В программе **TestSum** (в онлайн компиляторе замените имя класса) суммируется последовательность чисел, которая начинается с **0.01** и заканчивается **1.0**. Числа в последовательности увеличиваются на **0.01** следующим образом: **0.01 + 0.02 + 0.03** и так далее.

public class TestSum {

public static void main(String[] args) {

// Инициализировать sum

float sum, i;

sum = 0;

// Прибавить 0.01, 0.02, ..., 0.99, 1 к sum

for (i = 0.01f; i <= 1.0f; i = i + 0.01f)

sum += i;

// Отобразить sum

System.out.println("Сумма равна " + sum); }}

Сравните результат с точным значением.

1. В программе замените тип **float** на **double** для повышения точности. Объясните полученный результат.
2. Измените программу, используя целочисленный счетчик для прибавления всех чисел к **sum**.
3. Измените программу, прибавляя эти же числа от наибольшего к наименьшему. Выведите все результаты в таблицу и дайте им объяснения.

**Выполнение работы**

1. На языке Java была написана программа (прил. A), которая вычисляет значение машинного эпсилона для



Рисунок 1 – Машинный эпсилон

Также была получена информация о технических устройствах (версии операционной системы и java):

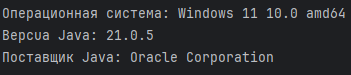


Рисунок 2 – Информация о техническом устройстве

2. С помощью метода границ найти приближенное значение функции u. Определим область определения функции при ограничениях , . На функцию накладываются ограничения из-за области определения логарифма , и y > 0, из-за области определения на рисунке показана область определения функции на пересечении трёх графиков:

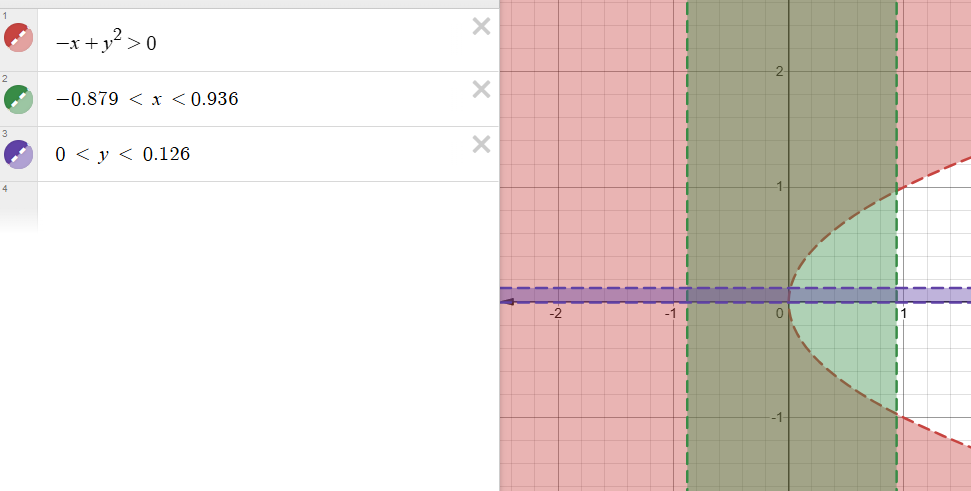


Рисунок 3 – График

Изменим границу

-0,879 < x < 0.015876

0 < y < 0.126

|  |  |  |
| --- | --- | --- |
| Выражение | Нижняя граница (НГ) | Верхняя граница (ВГ) |
| x | НГ(x) = -0,879 | ВГ(x) = 0.015876 |
| y | НГ(y) = 0 | ВГ(y) = 0.126 |
|  | НГ() = 0 | ВГ() = 0,354965 |
| -x | НГ(x^2) = 0.055696 | ВГ(x^2)=0.876096 |
| e^-x | НГ(e^-x) = 2,41 | ВГ(x^2)-НГ(y)=1.703096 |
| cos(x^2-y) | НГ(cos(x^2-y))=-0.131914 | ВГ(cos(x^2-y))=0.997529 |
| cos^3(x^2-y) | НГ(cos^3(x^2-y))=  -0.002295 | ВГ(cos^3(x^2-y))=  0.992607 |
| x-y^2 | НГ(x)-ВГ(y^2)=  0.220124 | ВГ(x)-НГ(y^2)=  0.252071 |
|  | НГ(ln((x-y^2))=  -1.513564 | НГ(ln(x-y^2))=  -1.378044 |
| u | НГ(cos^3(x^2-y))/  ВГ(ln(x-y^2)^2)=  -0.001002 | ВГ(cos^3(x^2-y))/  НГ(ln(x-y^2)^2)=  0.522698 |

3.

4. Исследование ошибок с плавающей точкой

А) Результат программы: 50.499985

Результат точных вычислений через арифметическую прогрессию: 50,5

Программа

Б) При изменении типа float на double программа вывела данный результат: 50.49999887123704

**Вывод:**

В процессе выполнения работы при помощи метода бисекции был вычислен локальный корень уравнения , а также обнаружено, что число итераций прямо пропорционально желаемой точности итогового результата – чем выше точность, тем большее число итераций потребуется. Также было выяснено, что методы чувствительны к изменению точности исходных данных, особенно при высокой ожидаемой точности результата.

ПрИЛОЖЕНИЕ A

Файл M**achineEpsilon**.java

public class MachineEpsilon {

public static void main(String[] args) {

// Вычисление для float

float epsFloat = 10.0f, tenFloat = 10.0f;

while (tenFloat + epsFloat / 2.0f != tenFloat) epsFloat /= 2.0f;

System.out.println("Машинный эпсилон (float, начиная с 10): " + epsFloat);

// Вычисление для double

double epsDouble = 10.0, tenDouble = 10.0;

while (tenDouble + epsDouble / 2.0 != tenDouble) epsDouble /= 2.0;

System.out.println("Машинный эпсилон (double, начиная с 10): " + epsDouble);

}

}