**­­­­Министерство науки и высшего образования Российской Федерации**

Федеральное государственное автономное образовательное учреждение

высшего образования «Уральский федеральный университет

имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ

Центр ускоренного обучения

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**

по дисциплине «Конструирование программного обеспечения»

**Тема:** **Модель Джелинского-Моранды**

Студенты гр. РИВ-400027у Л.А.Кайгородова

|  |
| --- |
| О.В.Дрон  Д.И.Кудинов  В.И.Пинтак  Я.В.Козлов |

Преподаватель С.И.Тимошенко,

доц., к.т.н

**Екатеринбург 2023**

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc146126437)

[2 Результаты проделанной работы 5](#_Toc146126438)

[3 Анализ полученных результатов 7](#_Toc146126439)

[ПРИЛОЖЕНИЕ А (справочное) Исходный текст программы 8](#_Toc146126440)

# 1 Постановка задачи

Разработать программное обеспечение, выполняющее расчеты при любых заданных входных параметрах для модели Джелинского-Моранды. В качестве тестовых использовать входные параметры, приведенные ниже в описании модели. Необходимо рассчитать параметры, описанные ниже в п. “Порядок расчета”.

Модель Джелинского-Моранды – одна из первых и наиболее простых моделей классического типа. Модель использовалась при разработке ПО для весьма ответственных проектов, в частности для ряда модулей программы Apollo.

Входные параметры.

В таблице 1 приведены данные тестирования и отладки критической программной системы. Найти общее число ошибок в программной системе, время до появления следующей ошибки и время до окончания тестирования. Для расчетов использовать модель Джелинского- Моранды.

1. Хронология тестирования

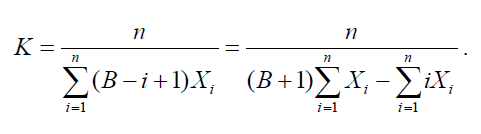
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Номер ошибки  (i) | Интервал  между ошибками  (Xi, часы) | Номер ошибки  (i) | Интервал  между ошибками  (Xi, часы) | Номер ошибки  (i) | Интервал  между ошибками  (Xi, часы) | Номер ошибки  (i) | Интервал  между ошибками  (Xi, часы) |
| 1 | 9 | 8 | 8 | 15 | 4 | 22 | 33 |
| 2 | 12 | 9 | 5 | 16 | 1 | 23 | 7 |
| 3 | 11 | 10 | 7 | 17 | 3 | 24 | 91 |
| 4 | 4 | 11 | 1 | 18 | 3 | 25 | 2 |
| 5 | 7 | 12 | 6 | 19 | 6 | 26 | 1 |
| 6 | 2 | 13 | 1 | 20 | 1 |  |  |
| 7 | 5 | 14 | 9 | 21 | 11 |  |  |

Порядок расчета.

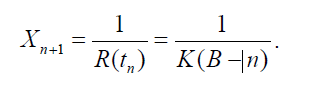
1. Решить численным методом следующее нелинейное уравнение для оценки максимального правдоподобия величины *B* (общее число ошибок в программе):



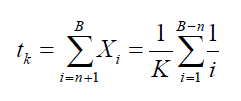
1. Найти коэффициент пропорциональности K (используется в функции риска R(t)=K[B-(i-1)], где t –произвольная точка времени между обнаружением (i-1)-й и i-й ошибок):



1. Найти среднее время Xn+1 до появления (n+1)-й ошибки как величину, обратную оценочной интенсивности для предыдущей ошибки:



1. Найти время до окончания тестирования:



# 2 Результаты проделанной работы

Для решения данной задачи был выбран язык программирования Java SE, версия языка jdk 20. Из стандартной библиотеки для нахождения модуля числа используется класс Math. Для чтения из файла используется класс FileReader для получения потока данных с файла и класс Scanner для парсинга чисел из потока данных. Полученные данные помещались в коллекцию ArrayList.

В качестве способа решения нелинейного уравнения был выбран метод итераций.

Метод итераций - это способ численного решения математических задач. Его суть – нахождение алгоритма поиска по известному приближению (приближенному значению) искомой величины следующего, более точного приближения. Применяется в случае, когда последовательность приближений по указанному алгоритму сходится.

Данный метод называют также методом последовательных приближений, методом повторных подстановок, методом простых итераций и т.п.

При запуске программы происходит загрузка данных из файла data.txt после чего они передаются в методы расчета параметров.

Максимальное количество итераций было выбрано значение 100000, оно позволяет обеспечить приближение числа *B* к точности равной 0.0001. Так же число 0.0001 выступает в роли шага итерации приближения.

В расчете оценки максимального правдоподобия величины *B* первым делом находится сумма интервалов между ошибками и сумма произведения интервалов между ошибками и их порядковым номером, далее эти переменные понадобятся в дальнейших расчетах. В качестве первого, приближенного значение *B* используем количество ошибок, в нашем случае это 26. Далее мы вычисляем левую и правую часть нелинейного уравнения, в случае если их разность меньше чем заданная точность, то считаем, что приближенное число *В* найдено и выходим из цикла, иначе увеличиваем число на шаг и повторяем пока не сработает условие выхода или приблизимся к максимальному числу итераций. За выполнения данного алгоритма отвечает метод CalcB() которая принимает в качестве параметра массив данных.

Для расчета коэффициент пропорциональности *K* используем был написан метод CalcK(). Для работы данного метода необходимо иметь расчетное значение B, поэтому для работы метода независимо внутри нее вызывается метод CalcB(). В качестве параметра принимается массив данных.

Для вычисления среднего время *Xn+1* до появления (n+1)-й ошибки как величину, обратную оценочной интенсивности для предыдущей ошибки написан метод CalcXinc(). В методе используется значение *В* и *К*, поэтому они вычисляются внутри метода что бы значения на выходе были получены независимо. В качестве параметров принимает массив данных.

Для нахождения времени до окончания тестирования был написан метод CalcTime(), в данном методе так же применяются значения, *B*, *K* которые так же вычисляются внутри функции. В качестве параметров принимает массив данных.

Результат работы программы можно увидеть на рисунке 1 и реализацию всех методом можно увидеть в приложении А.

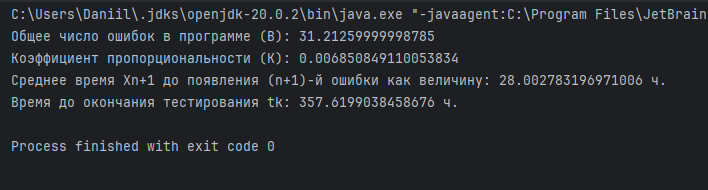


Рисунок 1 – Результат работы программы

# 3 Анализ полученных результатов

В процессе выполнения лабораторной работы № 2 было разработано программное обеспечение, выполняющее расчеты при любых заданных входных параметрах для модели Джелинского-Моранды. Оформлен отчет и приведен исходный код программы.

В качестве системы командной разработки мы выбрали GitHub, где фиксировали результаты выполненных задач.

Ссылка на командный репозиторий: https://github.com/dron-olya7/labpo.

ПРИЛОЖЕНИЕ А  
(справочное)  
Исходный текст программы

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

class Main

{

static double Step = 0.0001; //шаг поиска B

static long IT\_MAX = 100000; //Максимальное количество итераций

static double Sum = 0; // Сумма интервалов между ошибками.

static double SumI = 0; // Сумма произведений интервалов между ошибками на их порядковый номер.

static double CalcB(ArrayList<Integer> data)

{

double r; //переменные результата разности: r (result)

double d = data.size(); //в качестве начального значени В берем размер массива

Sum = 0; //обнуляем данные для подсчета новой суммы

SumI = 0; //обнуляем данные для подсчета новой суммы

//Подсчет сумма интервалов между ошибками

for (Integer x : data) {

Sum += x;

}

//Подсчет суммы произведений интервалов между ошибками на их порядковый номер.

for (int i = 0, j = 1; i < data.size(); i++, j++) {

SumI += j \* data.get(i);

}

//поиск B

for (int it = 1; it < IT\_MAX; it++)

{

double left = 0; //переменная для вычесления левой части уровнения.

double right = 0; //переменная для вычсления правой части уровнения.

//вычисления левой части уровнения.

for (int i = 0, j = 1; i < data.size(); i++, j++)

{

left += ((double) 1 / (d - j + 1));

}

//вычесления правой части уровнения

right = (data.size() \* Sum) / ((d + 1) \* Sum - SumI);

//вычесления абсолютного значения между частами уровнения(модуль числа)

r = Math.abs(left - right);

// Если текущая разница меньше точности, выходим

if (r < Step)

{

break;

}

//добавляем шаг и начинаем новый цикл

d = d + Step;

}

return d;

}

static double CalcK(ArrayList<Integer> data)

{

double resultK = 0;

double B = CalcB(data);

resultK = Math.abs(data.size() / (((B + 1) \* Sum) - SumI));

return resultK;

}

static double CalcXinc(ArrayList<Integer> data)

{

double result = 0;

double B = CalcB(data);

double K = CalcK(data);

result = 1 / (K \* (B - data.size()));

return result;

}

static double CalcTime(ArrayList<Integer> data)

{

double result = 0;

double B = CalcB(data);

double K = CalcK(data);

double sum\_buff = 0;

for(int i = 0; i < B - data.size(); i++)

{

sum\_buff += (double) 1 / (1 + i);

}

result = sum\_buff / K;

return result;

}

public static void main(String[] args)

{

//массив входных данных

ArrayList<Integer> data = new ArrayList<Integer>();

//чтение данных из файла

try

{

Scanner scanner = new Scanner(new FileReader("data.txt"));

while (scanner.hasNextBigInteger())

{

data.add(scanner.nextInt());

}

scanner.close();

}

catch (FileNotFoundException e)

{

throw new RuntimeException(e);

}

System.out.println("Общее число ошибок в программе (B): " + CalcB(data));

System.out.println("Коэффициент пропорциональности (K): " + CalcK(data));

System.out.println("Cреднее время Xn+1 до появления (n+1)-й ошибки как величину: " + CalcXinc(data) + " ч.");

System.out.println("Время до окончания тестирования tk: " + CalcTime(data) + " ч.");

}