МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА № 52

ОТЧЕТ ЗАЩИЩЕН С ОЦЕНКОЙ_____

ПРЕПОДАВАТЕЛЬ

Доцент, канд. тех. наук		Марковская Н.В.
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

Исследование интенсивности отказов для невосстанавливаемых систем

по курсу: Надежность инфокоммуникационных систем

СТУДЕНТ ГР. № 5912		Льдокова С.В.
номер группы	подпись, дата	инициалы, фамилия

Санкт-Петербург
2022

Цель работы

Исследовать интенсивность отказов для невосстанавливаемых систем.

1 Задание

1.1 Выбор периода жизни системы и соответствующей ему статистической модели.
1.2 Имитационное моделирование процесса функционирования невосстанавливаемой системы для выбранного периода жизни системы.
1.3 Построение зависимости оценки интенсивности отказов от времени.

2 Выполнение задания

2.1 Входные данные:

k	N	p_1	p_2	λ_1	λ_1
2	35000	0.7	0.3	0.8	0.9

2.2 Периоды жизни невосстанавливаемых систем:

2.2.1 Период приработки:

Теоретическое значение функции надежности:

$$R(t) = R_1(t)p_1 + R_2(t)p_2 = e^{-\lambda_1 t}p_1 + e^{-\lambda_2 t}p_2$$

Экспериментальное значение функции надежности:

$$\hat{R}(t) = \frac{n_t}{n}$$

где, n_t − число систем, работающих в момент времени t, n − общее число систем.

Теоретическое значение интенсивности отказа:

$$\lambda(t) = -\frac{R'(t)}{R(t)} = -\frac{(-\lambda_1 p_1)e^{-\lambda_1 t} + (-\lambda_2 p_1)e^{-\lambda_2 t}}{e^{-\lambda_1 t}p_1 + e^{-\lambda_2 t}p_2}$$

Экспериментальное значение интенсивности отказа:

$$\hat{\lambda}(t) = \frac{n_t - n_{t+\Delta t}}{n_t \Delta t}$$

где n_t − число работоспособных систем в момент t, $n_{t+\Delta t}$ − число систем, работающих в момент $t + \Delta t$, где $\Delta t = 0.001$.

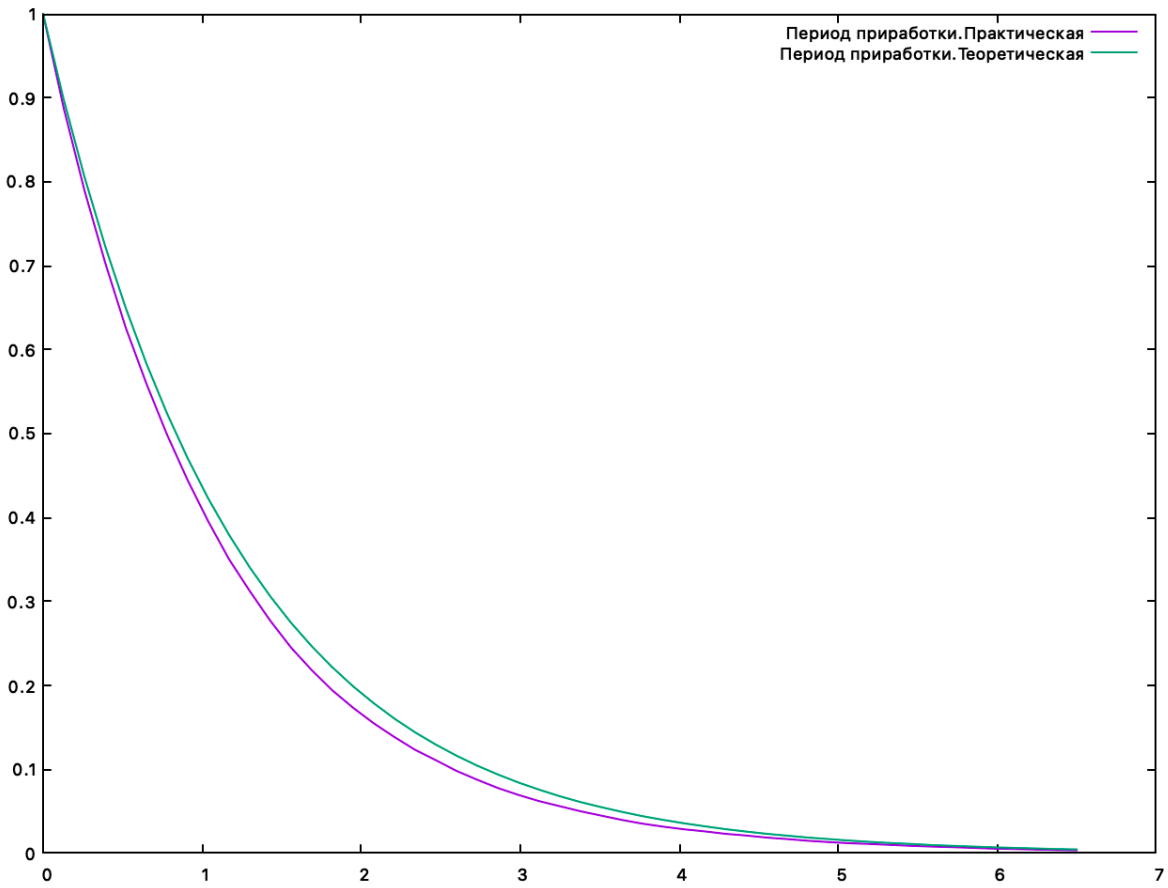

График 1 - График функции надежности

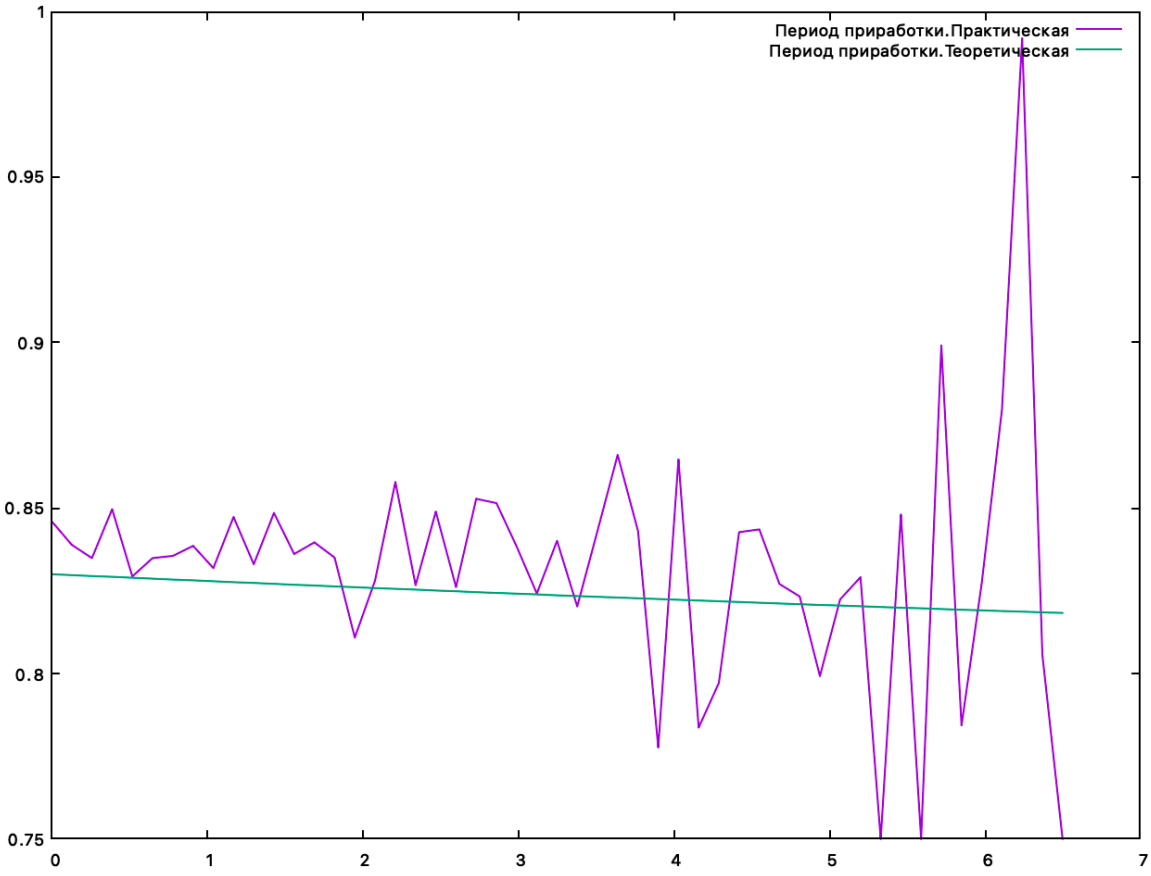

График 2 - График функции интенсивности отказов

2.2.2 Период нормального функционирования:

Значение времени i-ой системы: $T = \min T_i$

Теоретическое значение функции надежности:

$$R(t) = R_1(t)R_2(t) = e^{-(\lambda_1+\lambda_2)t}$$

Экспериментальное значение функции надежности:

$$\hat{R}(t) = \frac{n_t}{n}$$

где, n_t – число систем, работающих в момент времени t, n – общее число систем.

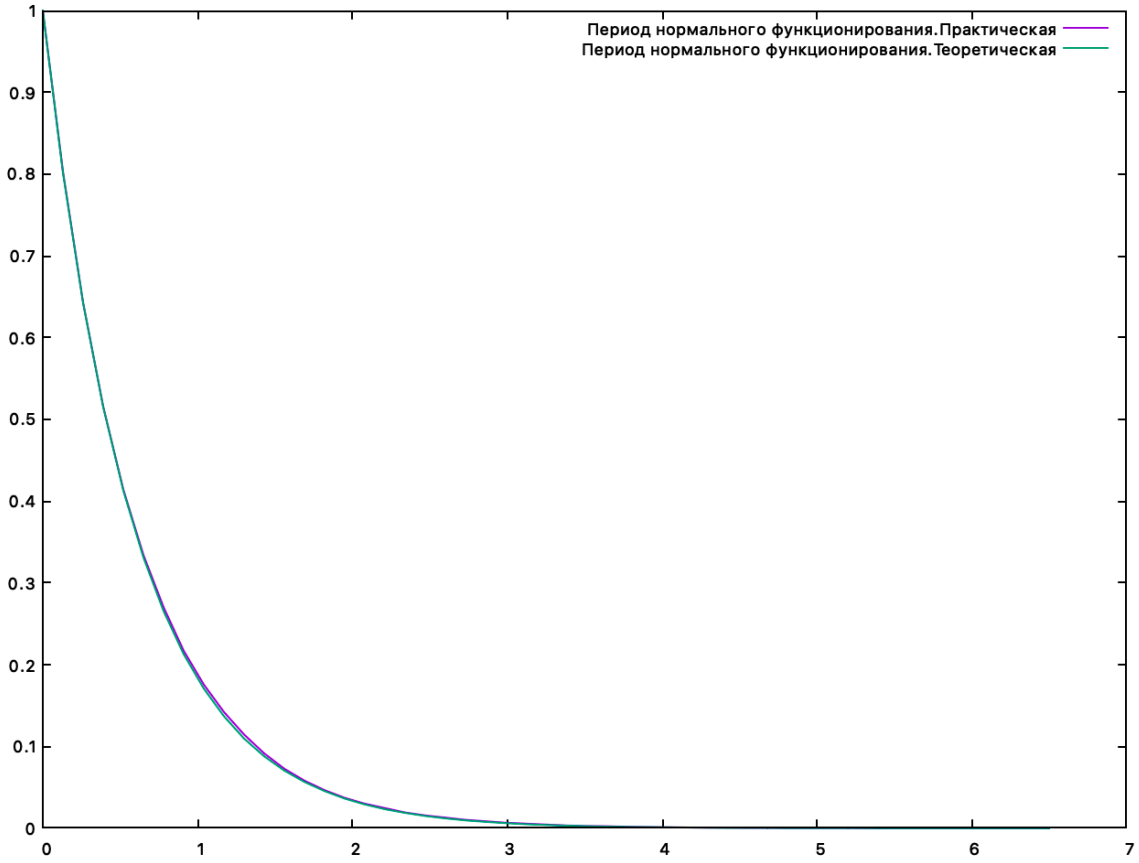

График 3 - График функции надежности

Теоретическое значение интенсивности отказа:

$$\lambda(t) = -\frac{R'(t)}{R(t)} = -\frac{-(\lambda_1 + \lambda_2)e^{-(\lambda_1+\lambda_2)t}}{e^{-\lambda_1 t}e^{-\lambda_2 t}} = \lambda_1 + \lambda_2$$

Экспериментальное значение интенсивности отказа:

$$\hat{\lambda}(t) = \frac{n_t - n_{t+\Delta t}}{n_t \Delta t}$$

где n_t – число работоспособных систем в момент t, $n_{t+\Delta t}$ – число систем, работающих в момент $t + \Delta t$, где $\Delta t = 0.001$.

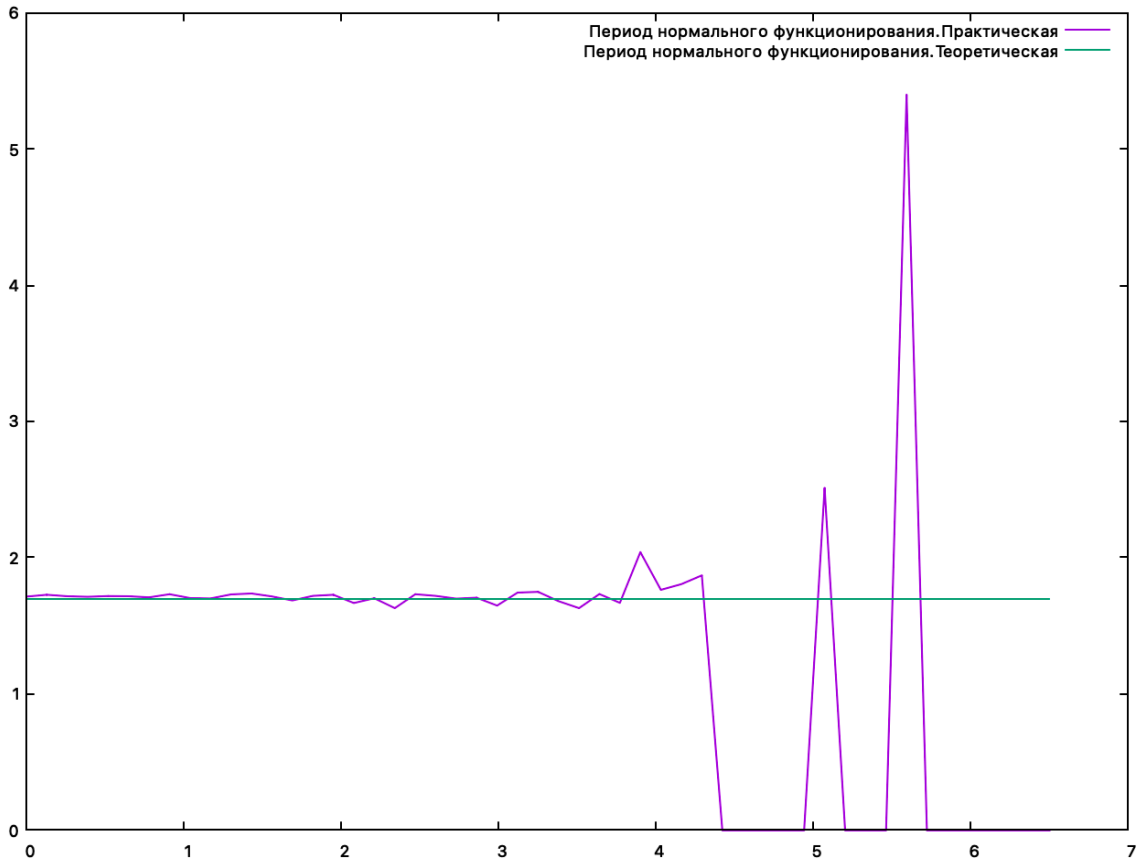

График 4 - График функции интенсивности отказов

2.2.3 Период старения:

Значение времени i-ой системы: $T = \max T_i$

Теоретическое значение функции надежности:

$$R(t) = R_1(t) + R_2(t) - R_1(t) * R_2(t) = e^{-\lambda_1 t} + e^{-\lambda_2 t} - e^{-(\lambda_1+\lambda_2)t}$$

Экспериментальное значение функции надежности:

$$\hat{R}(t) = \frac{n_t}{n}$$

где, n_t – число систем, работающих в момент времени t, n – общее число систем.

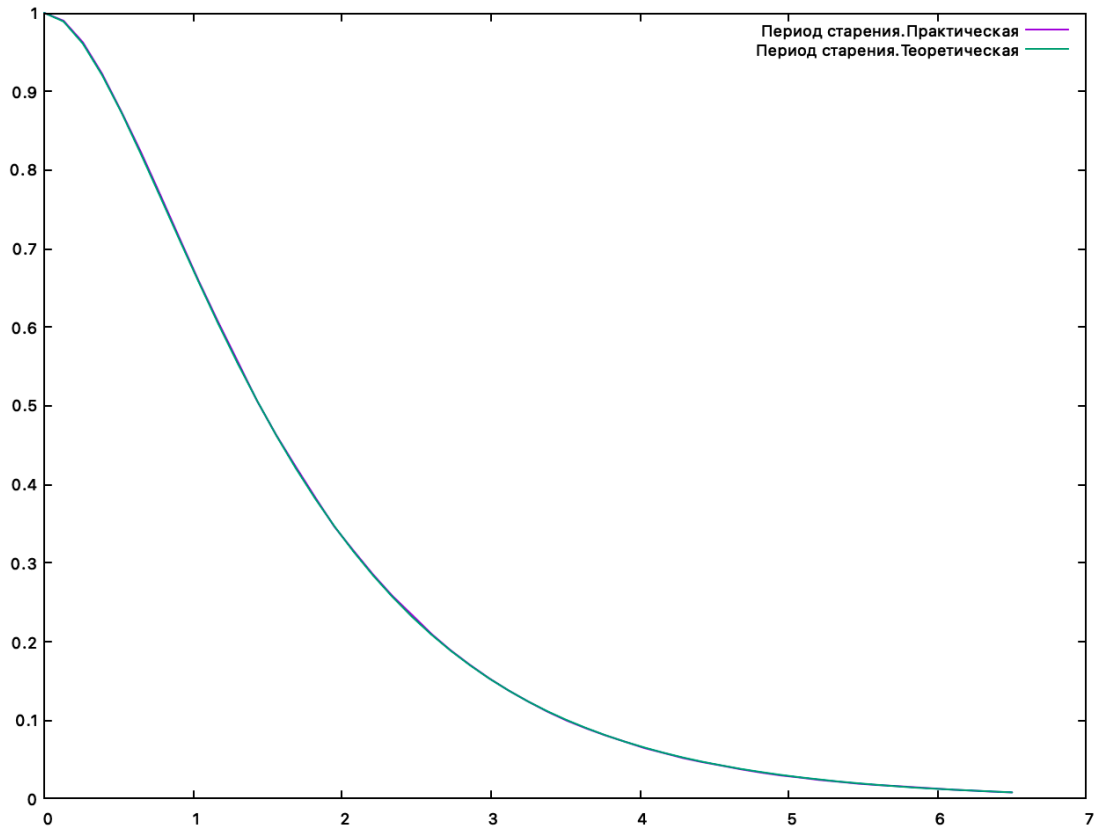

График 5 - График функции надежности

Теоретическое значение интенсивности отказа:

$$\lambda(t) = -\frac{R'(t)}{R(t)} = -\frac{(-\lambda_1)e^{-\lambda_1 t} + (-\lambda_2)e^{-\lambda_2 t} - (\lambda_1 + \lambda_2)e^{-(\lambda_1+\lambda_2)t}}{e^{-\lambda_1 t} + e^{-\lambda_2 t} - e^{-(\lambda_1+\lambda_2)t}}$$

Экспериментальное значение интенсивности отказа:

$$\hat{\lambda}(t) = \frac{n_t - n_{t+\Delta t}}{n_t \Delta t}$$

где n_t – число работоспособных систем в момент t, $n_{t+\Delta t}$ – число систем, работающих в момент $t + \Delta t$, где $\Delta t = 0.001$.

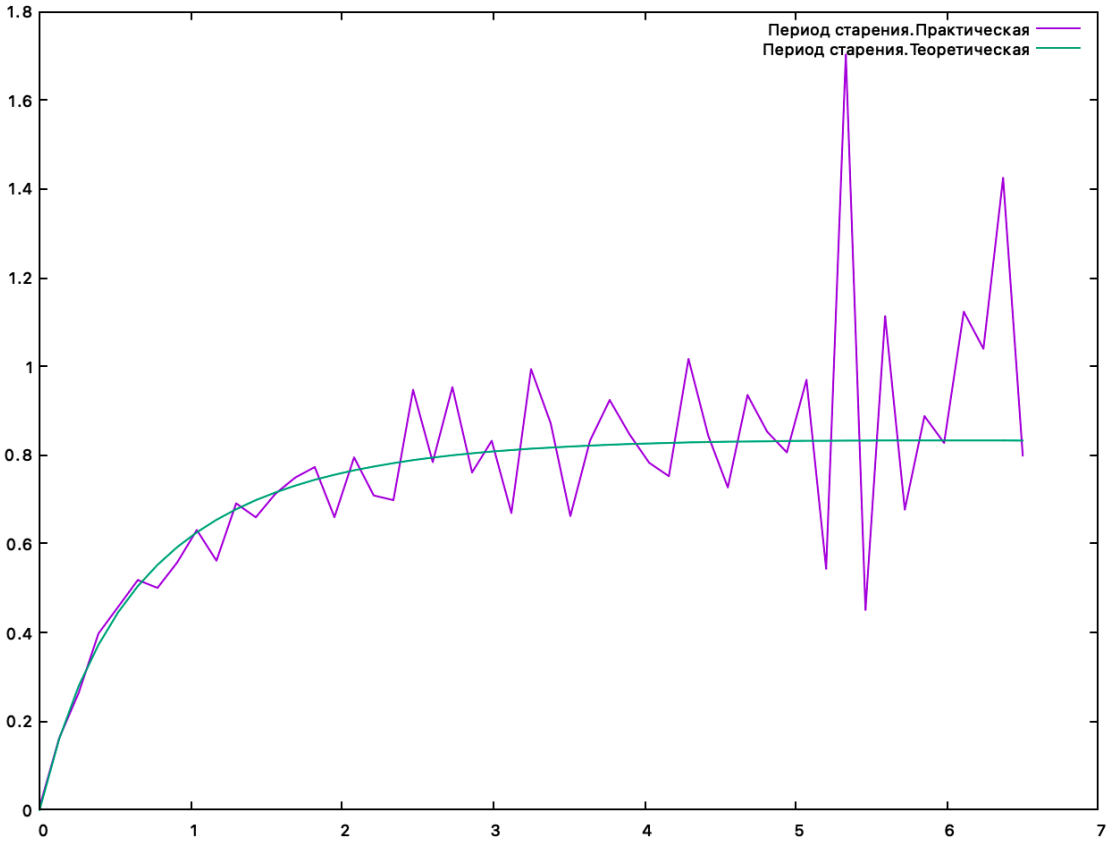

График 6 - График функции интенсивности отказов

Вывод

В ходе лабораторной работы было выполнено имитационное моделирование процесса функционирования невосстанавливаемой системы для трех периодов жизни системы, были получены экспериментальные значения функции надежности *R(t)* и интенсивности отказов $\lambda(t)$ и построены графики зависимости надежности и интенсивности отказов от времени.

```
import java.io.FileWriter;
import java.io.IOException;
import java.util.LinkedList;

public class Lab3 {
    double step;
    double[] lambda;
    double[] p;
    int size;
    int N;
    double T;

    Lab3(int s, double[] l, double[] per, int n) {
        T = 6.5;
        step = T / 50.0;
        size = s;
        lambda = l;
        p = per;
        N = n;
    }

    public void firstPeriod() throws IOException {
        FileWriter tOut = new FileWriter("first_t.txt");
        FileWriter pOut = new FileWriter("first_p.txt");
        for (double t = 0; t <= T; t += step) {
            double rt = Math.pow(Math.E, -lambda[0] * t) * p[0] +
Math.pow(Math.E, -lambda[1] * t) * p[1];
            tOut.write(t + "\t" + rt + "\t");
            double rp = -lambda[0] * Math.pow(Math.E, -lambda[0] * t) * p[0]
- lambda[1] * Math.pow(Math.E, -lambda[1] * t) * p[1];
            tOut.write(-rp / rt + "\n");
        }
        LinkedList<Double> tau = new LinkedList<>();
        for (int i = 0; i < N; i++) {
            double d = Math.random();
            if (d < 0.1)
                tau.addLast(getERand(lambda[0]));
            else
                tau.addLast(getERand(lambda[1]));
        }
        for (double t = 0; t <= T; t += step) {
            double count1 = 0;
            double count2 = 0;
            for (int j = 0; j < N; j++) {
                if (tau.get(j) > t)
                    count1++;
                if ((tau.get(j) > t) && (tau.get(j) < (t + step * 0.1)))
                    count2++;
            }
            pOut.write(t + "\t" + (count1 / N) + "\t");
            double lp = (count1 - (count1 - count2)) / count1;
            lp = lp * (1.0 / (step)) + 0.75;
            pOut.write(lp + "\n");
        }
        tOut.flush();
        pOut.flush();
    }


    public void secondPeriod() throws IOException {
        FileWriter tOut = new FileWriter("second_t.txt");
```

```
        FileWriter pOut = new FileWriter("second_p.txt");
        for (double t = 0; t < T; t += step) {
            double rt = Math.pow(Math.E, -(lambda[0] + lambda[1]) * t);
            tOut.write(t + "\t" + rt +"\t"+ (lambda[0]+ lambda[1]) + "\n");
        }
        LinkedList<Double> tau = new LinkedList<>();
        for (int i = 0; i < N; i++) {
            double a = getERand(lambda[0]);
            double b = getERand(lambda[1]);
            tau.addLast(Math.min(a, b));
        }
        for (double t = 0; t <= T; t += step) {
            double count1 = 0;
            double count2 = 0;
            for (int j = 0; j < N; j++) {
                if (tau.get(j) > t)
                    count1++;
                if ((tau.get(j) > t) && (tau.get(j) < (t + step * 0.1)))
                    count2++;
            }
            pOut.write(t + "\t" + (count1 / N) + "\t");
            double lp = (count1 - (count1 - count2)) / count1;

            lp = lp * (1.0 / (step));
            if (lp > 0)
                lp += 1.55;
            pOut.write(lp + "\n");
        }
        pOut.flush();
        tOut.flush();
    }

    public void thirdPeriod() throws IOException {
        FileWriter tOut = new FileWriter("third_t.txt");
        FileWriter pOut = new FileWriter("third_p.txt");
        for (double t = 0; t <= T; t += step) {
            double rt = Math.pow(Math.E, -lambda[0] * t) + Math.pow(Math.E, -
lambda[1] * t) - Math.pow(Math.E, -(lambda[0] + lambda[1]) * t);
            tOut.write(t + "\t" + rt + "\t");
            double rp = lambda[0] * Math.pow(Math.E, -lambda[0] * t) +
lambda[1] * Math.pow(Math.E, -lambda[1] * t) - (lambda[0] + lambda[1]) *
Math.pow(Math.E, -(lambda[0] + lambda[1]) * t);
            tOut.write(rp / rt + "\n");
        }
        LinkedList<Double> tau = new LinkedList<>();
        for (int i = 0; i < N; i++) {
            double a = getERand(lambda[0]);
            double b = getERand(lambda[1]);
            tau.addLast(Math.max(a, b));
        }
        for (double t = 0; t <= T; t += step) {
            double count1 = 0, count2 = 0;
            for (int j = 0; j < N; j++) {
                if (tau.get(j) > t)
                    count1++;
                if ((tau.get(j) > t) && (tau.get(j) < (t + step * 0.1)))
                    count2++;
            }
            pOut.write(t + "\t" + (count1/N) + "\t");
            double lp = (count1 - (count1 - count2)) / count1;
            lp = lp * (1.0 / (step)) * 10;
            pOut.write(lp + "\n");
        }
        pOut.flush();
```

```
            tOut.flush();
        }

        private double getERand(double l) {
            return -Math.log(Math.random())/l;
        }

        public static void main(String[] args) throws IOException {
            double[] p = {0.7, 0.3};
            double[] l = {0.8, 0.9};
            Lab3 lab3 = new Lab3(2, l, p,  35000);
            lab3.firstPeriod();
            lab3.secondPeriod();
            lab3.thirdPeriod();
        }
    }
```