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

КАФЕДРА №52

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Исследование интенсивности отказов для невосстанавливаемых систем

по курсу: СЕТИ С СИСТЕМЫ ПЕРЕДАЧИ ИНФОРМАЦИИ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР.	5512		К.А.Абдулжамилов
		подпись, дата	инициалы, фамилия

Санкт-Петербург 2019

1. Цель работы:
   Смоделировать периоды жизни невосстанавливаемой системы.
2. Исходные данные:
   Система состоит из двух групп. Для этих групп определены интенсивности:
   $\lambda_1 = 0.9$
   $\lambda_2 = 1.3$
   Также определены вероятности того, что выбрана конкретная система для первого периода:
   $p_1 = 0.7$
   $p_2 = 0.3$
3. Имитационное моделирование первого периода:
   Всего моделируется N систем. Из них N*p_1 имеют время жизни $T_i = -\frac{\ln(\alpha)}{\lambda_1}$ и оставшиеся системы имеют время жизни $T_i = -\frac{\ln(\alpha)}{\lambda_2}$, где α – случайная равномерно распределенная величина. Затем для сгенерированных N систем считаются T_{max} и $\overline{T} = \frac{\sum T_i}{N}$.
   Далее выбирается некая промежуточная точка между T_{max} и $\overline{T}$, данный промежуток разбивается на k отрезков. В каждый момент времени считается, сколько систем находится в рабочем состоянии и высчитывается функция надежности $R(t) = \frac{n_{cur}}{N}$, где n_{cur} – количество работающих систем. Также для модели рассчитывается интенсивность отказов $\lambda(t) = \frac{(n_t - n_{t+\Delta})}{n_t \cdot \Delta}$. Полученные результаты сравниваются с теоритическими. $R(t) = e^{-\lambda_1 t} p_1 + e^{-\lambda_2 t} p_2$
   $\lambda(t) = -\frac{R\prime(t)}{R(t)}$.
   Для заданной модели $\lambda(t) = \frac{\lambda_1 e^{-\lambda_1 t} p_1 + \lambda_2 e^{-\lambda_2 t} p_2}{e^{-\lambda_1 t} p_1 + e^{-\lambda_2 t} p_2}$

   Результат моделирования и расчетов представлен на графиках

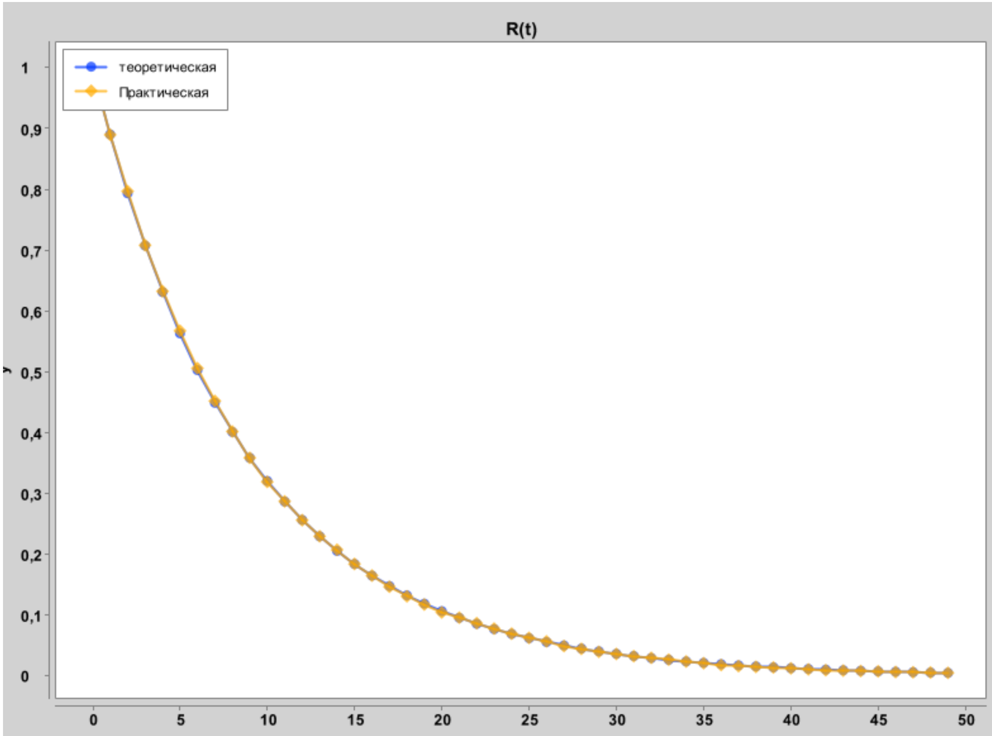

Рисунок 1 - R(t) для первой модели

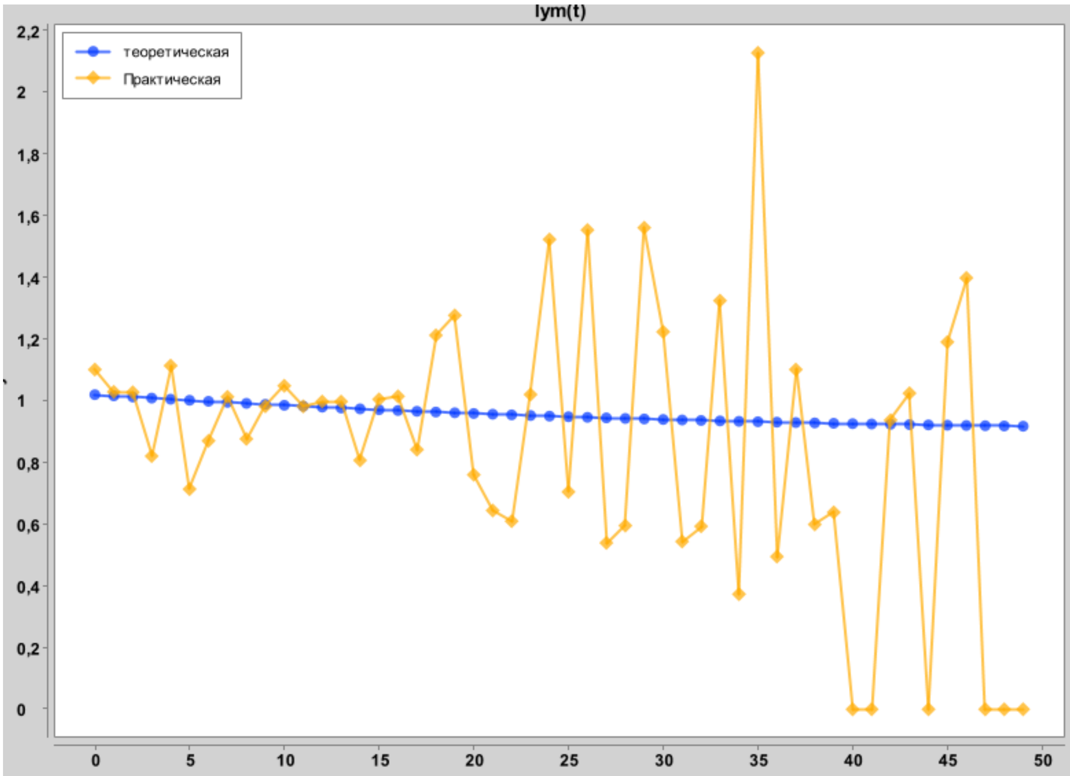

Рисунок 2 - λ(t) для первого периода

4. Имитационное моделирование второго периода:
Система представляет собой последовательное соединение элементов.
Всего моделируется N систем. $T_i = \min(-\frac{\ln(\alpha)}{\lambda_1}, -\frac{\ln(\beta)}{\lambda_2})$,где α и β – случайная равномерно распределенная величина. Затем для сгенерированных N систем считаются T_{max} и $\bar{T} = \frac{\sum T_i}{N}$. Далее выбирается некая промежуточная точка между T_{max} и $\bar{T}$, данный промежуток разбивается на k отрезков. В каждый момент времени считается, сколько систем находится в рабочем состоянии и высчитывается функция надежности $R(t) = \frac{n_{cur}}{N}$, где n_{cur} – количество работающих систем. Также для модели рассчитывается интенсивность отказов $\lambda(t) = \frac{(n_t - n_{t+\Delta})}{n_t \cdot \Delta}$. Полученные результаты сравниваются с теоритическими.
$R(t) = e^{-\lambda_1 t} + e^{-\lambda_2 t}$
$\lambda(t) = -\frac{R\prime(t)}{R(t)}$.
Для заданной модели $\lambda(t) = \frac{(\lambda_1 + \lambda_2)\, e^{-\lambda_1 t} e^{-\lambda_2 t}}{e^{-\lambda_1 t} e^{-\lambda_2 t}} = (\lambda_1 + \lambda_2)$
Результат моделирования и расчетов представлен на графиках

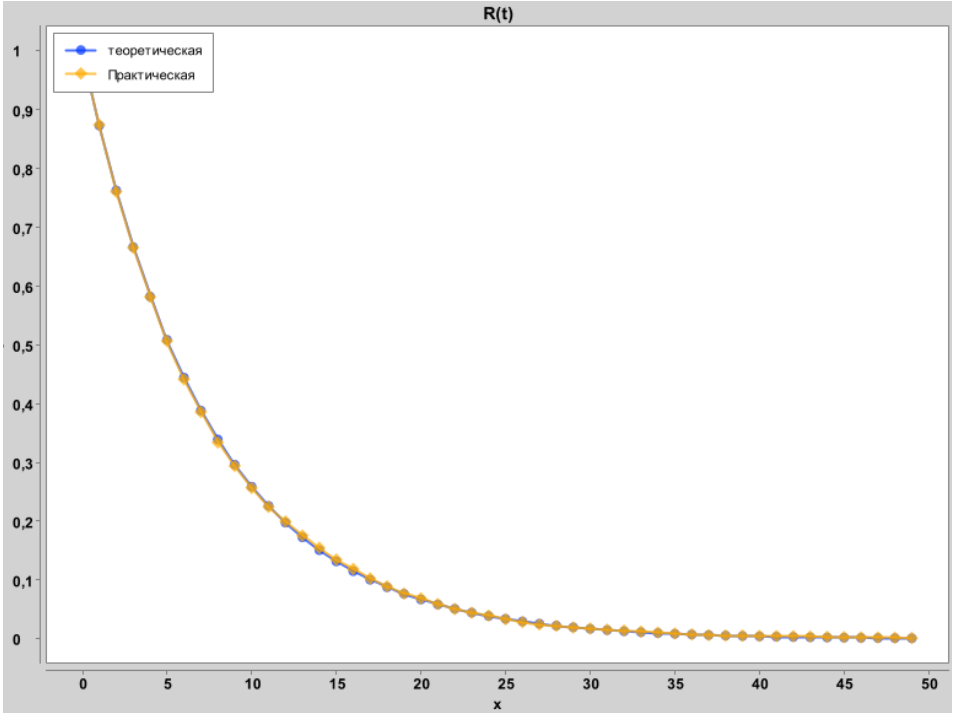

Рисунок 3 - R(t) для второй модели

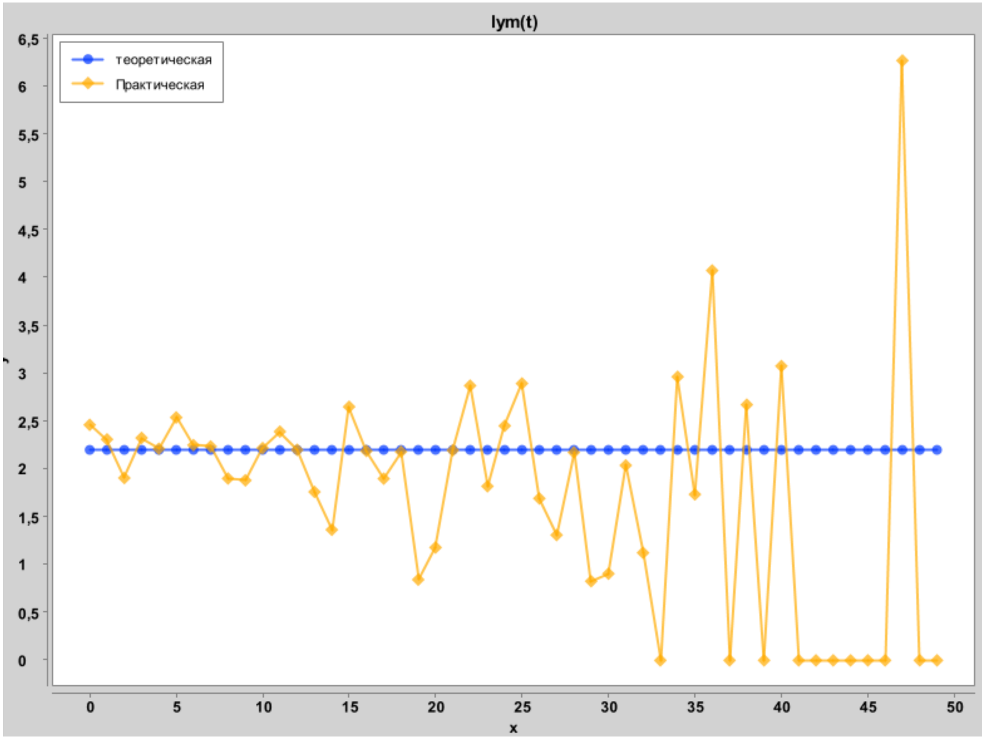

Рисунок 4 - λ(t) для второго периода

5. Имитационное моделирование третьего периода:
   Система представляет собой параллельное соединение элементов.
   Всего моделируется N систем. $T_i = \max\left(-\frac{\ln(\alpha)}{\lambda_1}, -\frac{\ln(\beta)}{\lambda_2}\right)$,где α и β — случайная равномерно распределенная величина. Затем для сгенерированных N систем считаются T_{max} и $\overline{T} = \frac{\sum T_i}{N}$. Далее выбирается некая промежуточная точка между T_{max} и $\overline{T}$, данный промежуток разбивается на k отрезков. В каждый момент времени считается, сколько

систем находится в рабочем состоянии и высчитывается функция надежности $R(t) = \frac{n_{cur}}{N}$, где n_{cur} – количество работающих систем. Также для модели рассчитывается интенсивность отказов $\lambda(t) = \frac{(n_t - n_{t+\Delta})}{n_t \cdot \Delta}$. Полученные результаты сравниваются с теоритическими.

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

$\lambda(t) = -\frac{R\prime(t)}{R(t)}$.

Для заданной модели $\lambda(t) = \frac{\lambda_1 e^{-\lambda_1 t} + \lambda_2 e^{-\lambda_2 t} - (\lambda_1 + \lambda_2)\, e^{-(\lambda_1+\lambda_2)\,t}}{e^{-\lambda_1 t} + e^{-\lambda_2 t} - e^{-(\lambda_1+\lambda_2)\,t}}$

Результат моделирования и расчетов представлен на графиках

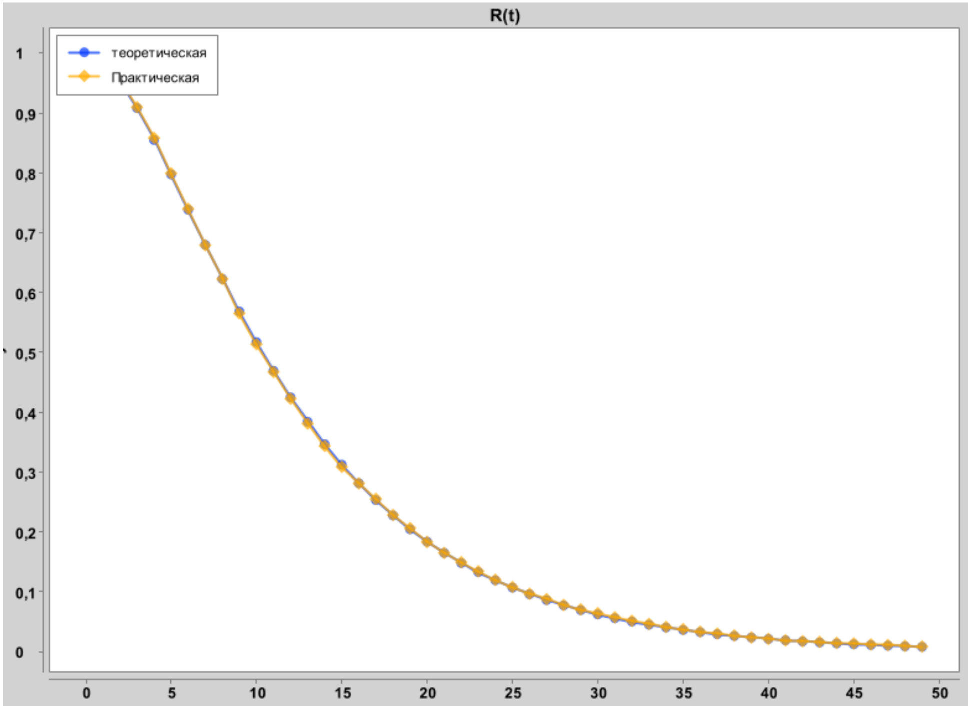

Рисунок 5 - R(t) для третьей модели

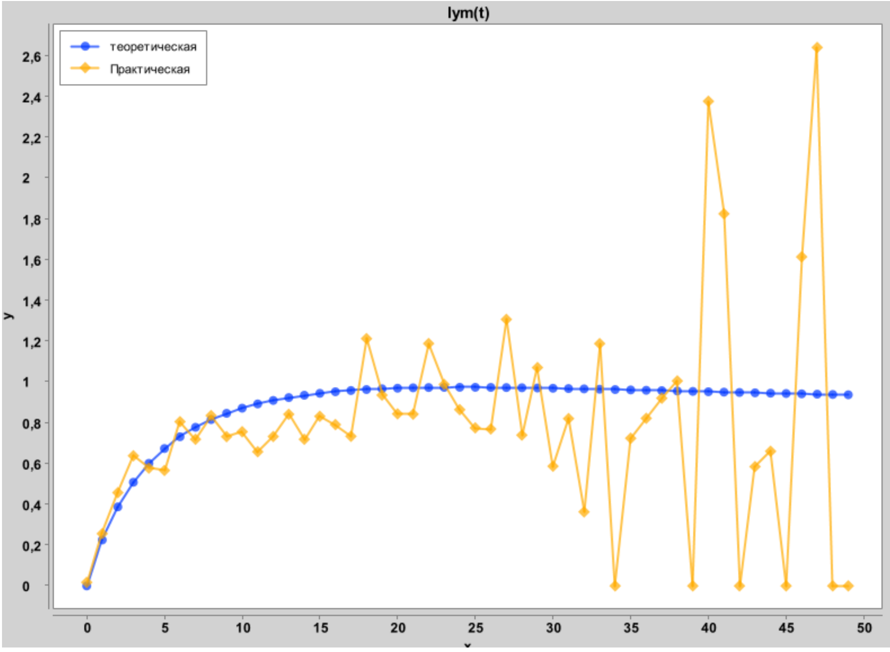

Рисунок 6- λ(t) для третьего периода

Вывод: в данной лабораторной работе было выполнено имитационное моделирование для моделей разного периода. Также для каждой модели были построены графики теоритических значений.

Листинг программы

```
import java.util.ArrayList;

public class Lab3 {
    private int N;
    private double lymda[];
    private double prob[];
    private double step;
    private double delta;

    public Lab3(int n,double[] l, double []pr, double s, double d) {
        N = n;
        lymda = l;
        prob = pr;
        step = s;
        delta = d;
    }
    public void firstStage()
    {
        ArrayList<Double>R = new ArrayList<>();
        ArrayList<Double>curs = new ArrayList<>();
        ArrayList<Double>R_teor = new ArrayList<>();
        ArrayList<Double>lym = new ArrayList<>();
        ArrayList<Double>lym_teor = new ArrayList<>();
        double T_average = 0;
        double T_max = 0;
        double T[] = new double[N];
        int index = 0;
        for(int j = 0; j < N; j++)
        {
            if(j ==(int)(N * prob[index]))
            {
                index++;
            }
            T[j] = getLymda(index);
            T_average += T[j];
            if(T[j] > T_max)
                T_max = T[j];
        }
        T_average = T_average / N;
        double Tmidle = (T_average + T_max) / 2;
        double st = Tmidle/step;
        double delta_S = st / delta;
        double cur = 0;
        for(int i = 0; i < step; i++)
        {
            int counter = 0;
            int counter_delta = 0 ;
            for(int j = 0; j < N; j++)
            {
                if(T[j] > cur)
                    counter++;
                if(T[j] > cur + delta_S)
                    counter_delta++;
            }
            R.add((double)counter/N);
            lym.add((counter-counter_delta)/(counter * delta_S));
            R_teor.add((Math.pow(Math.exp(1), -lymda[0]*cur) * prob[0] +
Math.pow(Math.exp(1), -lymda[1]*cur) * prob[1]));
            lym_teor.add(-(-lymda[0] * Math.pow(Math.exp(1), -lymda[0] * cur)
* prob[0] - lymda[1] * Math.pow(Math.exp(1), -lymda[1] * cur) * prob[1]) /
(Math.pow(Math.exp(1), -lymda[0]*cur) * prob[0] + Math.pow(Math.exp(1), -
lymda[1]*cur) * prob[1]));
```

```java
            curs.add(cur);
            cur += st;
        }
        Schedule.print(R_teor,R,curs,"R(t)");
        Schedule.print(lym_teor,lym,curs,"lym(t)");

    }
    public void secondAndThirdStage(boolean stage)
    {
        ArrayList<Double>R = new ArrayList<>();
        ArrayList<Double>R_teor = new ArrayList<>();
        ArrayList<Double>curs = new ArrayList<>();
        ArrayList<Double>lym = new ArrayList<>();
        ArrayList<Double>lym_teor = new ArrayList<>();
        double T_average = 0;
        double T_max = 0;
        double T[] = new double[N];
        double T1 = 0;
        double T2 = 0;
        int index = 0;
        for(int j = 0; j < N; j++)
        {
            T1 = getLymda(0);
            T2 = getLymda(1);
            if(!stage) {
                if (T1 < T2)
                    T[j] = T1;
                else
                    T[j] = T2;
            }
            else {
                if (T1 > T2)
                    T[j] = T1;
                else
                    T[j] = T2;
            }
            T_average += T[j];
            if(T[j] > T_max)
                T_max = T[j];
        }
        T_average = T_average / N;
        double Tmidle = (T_average + T_max) / 2;
        double st = Tmidle/step;
        double delta_S = st / delta;
        double cur = 0;
        for(int i = 0; i < step; i++)
        {
            int counter = 0;
            int counter_delta = 0 ;
            for(int j = 0; j < N; j++)
            {
                if(T[j] > cur)
                    counter++;
                if(T[j] > cur + delta_S)
                    counter_delta++;
            }
            R.add((double)counter/N);
            lym.add((counter-counter_delta)/(counter * delta_S));
            if(!stage) {
                R_teor.add((Math.pow(Math.exp(1), (-1) * lymda[0] * cur)) *
(Math.pow(Math.exp(1), (-1) * lymda[1] * cur)));
                lym_teor.add(lymda[1] + lymda[0]);
            }
            else {
```

```
                R_teor.add(Math.pow(Math.exp(1), (-1) * lymda[0] * cur) +
Math.pow(Math.exp(1), (-1) * lymda[1] * cur) -  Math.pow(Math.exp(1), (-1) *
(lymda[1] + lymda[0]) * cur));
                lym_teor.add(((lymda[0] * Math.pow(Math.exp(1),(-
1)*lymda[0]*cur)) + (lymda[1] * Math.pow(Math.exp(1),(-1)*lymda[1]*cur))  -
(lymda[1] + lymda[0])* Math.pow(Math.exp(1),(-1)*(lymda[1]+
lymda[0])*cur))/((Math.pow(Math.exp(1),(-1)*lymda[0]*cur)) +
(Math.pow(Math.exp(1),(-1)*lymda[1]*cur))  - Math.pow(Math.exp(1),(-
1)*(lymda[1]+ lymda[0])*cur)));
            }
            curs.add(cur);
            cur += st;
        }
        Schedule.print(R_teor,R,curs,"R(t)");
        Schedule.print(lym_teor,lym,curs,"lym(t)");

    }
    private double getLymda(int index)
    {
        return Math.log(Math.random())/lymda[index]*(-1);
    }
public class Main {
    public static void main(String[] arg)
    {
        Lab3 lab3 = new Lab3(10000,new double[]{0.9,1.3},new
double[]{0.7,0.3},50,10);
        lab3.firstStage();
        lab3.secondAndThirdStage(false);
        lab3.secondAndThirdStage(true);
    }

}
import java.util.ArrayList;
import java.util.List;
import org.knowm.xchart.*;
import org.knowm.xchart.style.Styler;

public class Schedule {
    public static void print (ArrayList<Double> array, ArrayList<Double>
array1,ArrayList<Double> list,String name) {

        ArrayList<Integer> list1 = new ArrayList<>();
        for (int i = 0; i < 50;i++)
        {
            list1.add(i);
        }

        XYChart chart = new
XYChartBuilder().width(800).height(600).title(name).xAxisTitle("x").yAxisTitl
e("y").build();

        chart.getStyler().setLegendPosition(Styler.LegendPosition.InsideNW);
        chart.getStyler().setHasAnnotations(false);
        chart.getStyler().setPlotGridLinesVisible(false);

        chart.addSeries("теоретическая", list1, array);
        chart.addSeries("Практическая", list1,array1);
        List<XYChart> charts = new ArrayList<XYChart>();
        charts.add(chart);
        new SwingWrapper<XYChart>(charts).displayChartMatrix();

    }
}
```