

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, КТН

Н.В. Марковская

---

должность, уч. степень,  
звание

---

подпись, дата

---

инициалы,  
фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4

по курсу: НАДЕЖНОСТЬ ИНФОКОММУНИКАЦИОННЫХ СЕТЕЙ

Студент гр. №

5912

И.К. Лобач

---

номер  
группы

---

подпись,  
дата

---

инициалы,  
фамилия

Санкт-Петербург 2022

## 1 Цель работы

Имитационное моделирование функционирования системы со сложной схемой резервирования. Построение временной зависимости, отражающей изменение коэффициента готовности восстанавливаемой  $K_r$  системы. Проверка того, что установившееся значение  $K_r$  находится в пределах границ.

## 2 Исходные данные

Имитационное моделирование необходимо провести для схемы, изображенной ниже:

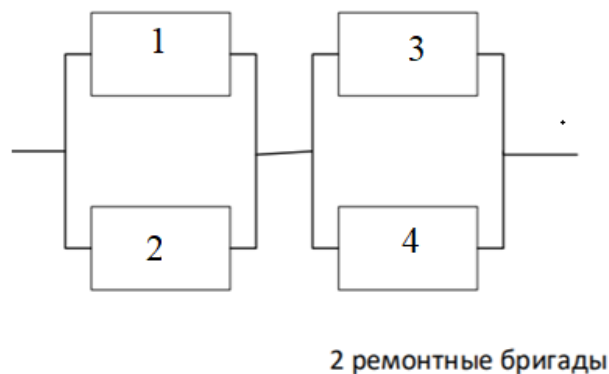


Схема 1 - Исходная схема

Для каждого элемента задается  $\lambda$  и  $\mu$ . Пусть  $\lambda = 1.1$  и  $\mu = 0.9$ . Пусть число экспериментов  $N = 30\,000$ . При моделировании будет рассматриваться время  $[0, T]$ , где  $T = 5$  с шагом  $\Delta t = 0.01$ .

## 3 Верхняя и нижняя граница

### 3.1 Верхняя граница

Для подсчета верхней границы необходимо увеличить количество бригад до количества элементов системы. В данном случае количество бригад будет равно 4. Таким образом, каждая бригада чинит один элемент, т.е. коэффициент готовности одного элемента  $K_r = K_{1,1}$ .

$$K_{1,1} = \frac{\mu}{\lambda + \mu}$$

Чтобы заданная система работала в момент времени, необходимо, чтобы работал элемент 1 или 2 и 3 или 4, т.е. коэффициент готовности будет равен:

$$K_r^+ = \left(1 - (1 - K_{1,1})^2\right)^2$$

Для заданных значений  $K_r^+ = (1 - (1 - 0,45)^2)^2 = 0,48$ .

### 3.2 Нижняя граница (распределение бригады)

Для подсчета нижней границы будем считать, что 1 бригада будет работать с двумя элементами:

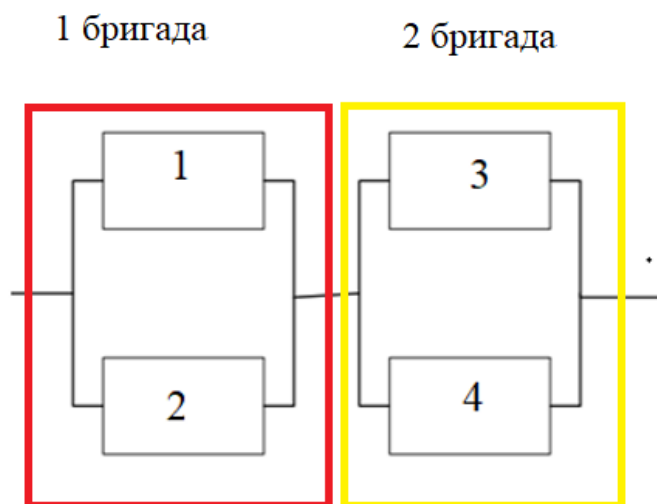


Схема 2 - Распределение бригады

Коэффициент готовности  $K_{2,1}$  будет равен:

$$K_{2,1} = \frac{2\lambda\mu + \mu^2}{2\lambda^2 + 2\lambda\mu + \mu^2}$$

Чтобы заданная система работала в момент времени, необходимо, чтобы одновременно работала и 1 группа элементов, и вторая группа элементов, т.е. коэффициент готовности системы будет равен:

$$K_r^- = K_{2,1} * K_{2,1}$$

Для заданных значений  $K_r^- = 0,53 * 0,53 = 0,28$ .

### 3.3 Нижняя граница (исключение систем)

Для подсчета нижней границы будем считать коэффициент готовности системы, исключив по одному элементу из параллельного соединения:

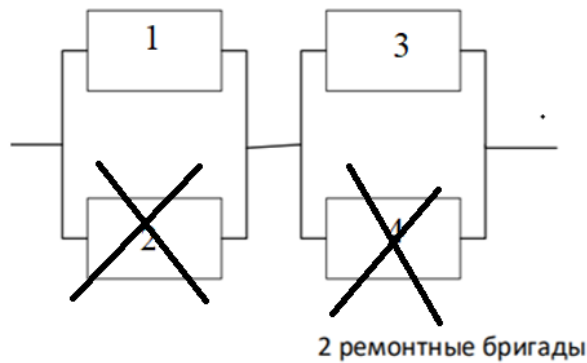


Схема 3 - Исключение систем

В таком случае каждая бригада чинит один элемент. Оставшиеся элементы соединены последовательно, значит, чтобы системы работала в момент времени, необходимо, чтобы одновременно работал и элемент 1, и элемент 3, тогда коэффициент готовности будет равен:

$$K_r^- = K_{1,1} * K_{1,1}$$

Для заданных значений  $K_r^- = 0,45 * 0,45 = 0,2$ .

### 4 Имитационное моделирование

В каждый момент времени происходит оценка работоспособности системы в целом. Если один из элементов ломается, то одна из свободных бригад приступает к ремонту. Если все бригады заняты, то время ремонта элемента увеличивается на  $t_{step}$  (элемент дожидается, пока бригада освободится).

Для подсчета коэффициента готовности выбирается произвольное время  $T$  рассматривается интервал времени  $[0, T]$  с шагом  $\Delta t$ . Было выбрано время  $T = 5$  и шаг  $\Delta t = 0,01$ . Затем осуществляется моделирование одного эксперимента.

Для каждого из элементов системы необходимо случайным образом сгенерировать время работы  $T_w$  и время ремонта  $T_r$  по следующим формулам:

$$T_w = \frac{-\ln [0,1]}{\lambda}$$

$$T_r = \frac{-\ln [0,1]}{\mu}$$

Затем в каждый момент времени функция проверки работоспособности системы возвращается  $\hat{E}(t) = \{0,1\}$ , где 0 означает, что система не работает, иначе 1.

Таким образом моделируется  $N = 30\,000$  экспериментов. Коэффициент готовности определяется как:

$$K_r(t) = (\sum_{j=1}^N E_j(i * \Delta t)) / N, \text{ где } i = 1, 2, \dots, k$$

Результаты имитационного моделирования сравниваются с результатами аналитического расчета границ коэффициента готовности.

В результате был получен график зависимости коэффициента готовности от времени:

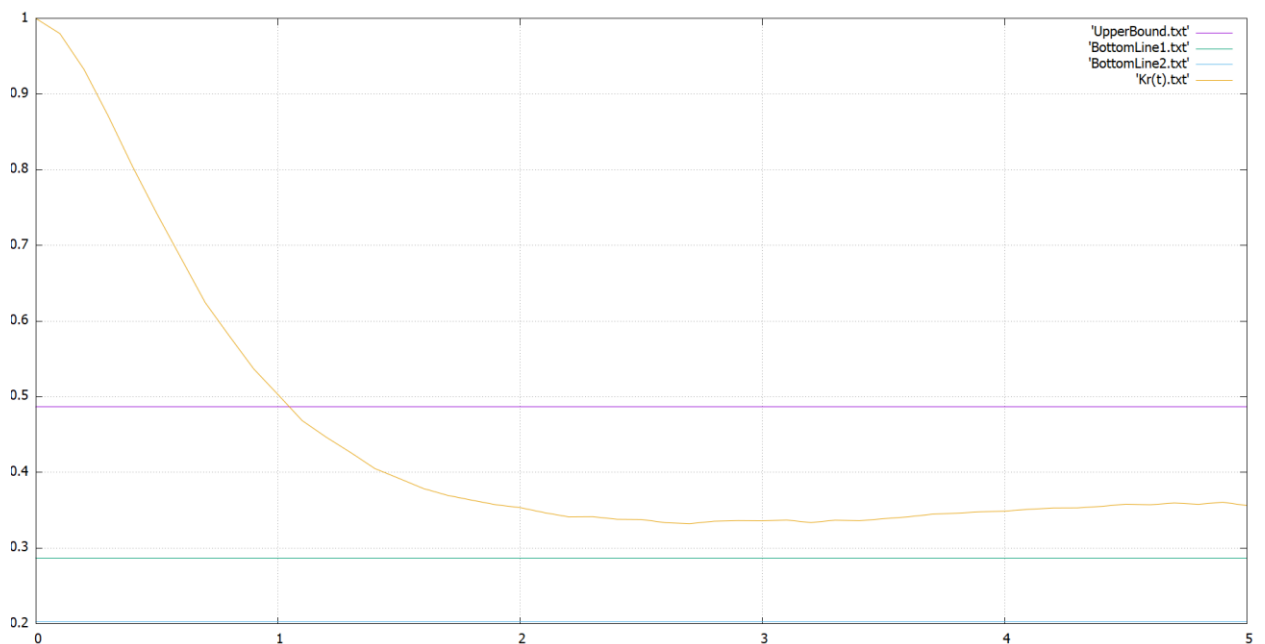


График 1 - Коэффициент готовности

## 5 Выводы

Таким образом в ходе моделирования были получены верхняя и нижняя оценка коэффициента готовности системы. Также были получены экспериментальные значения коэффициента готовности, которые, как видно из графика 1 в установившемся состоянии полностью находятся в пределах заданных границ, что говорит о корректности работы программы.

### Листинг программы

```
package com.suai;

import java.io.FileWriter;

public class Modeling4 {

    private class Repair {

        private boolean status = true; // 0 - занята 1 - свободна
        private double ending; // время от 0
        private byte repairNum = (byte)n;

        public void createRepair(boolean s, double e, byte num) {
            status = s;
            ending = e;
            repairNum = num;
        }

        public void free() {
            status = true;
            ending = 0;
            repairNum = (byte)n;
        }

        public String toString() {
            StringBuilder str = new StringBuilder();
            str.append("is free = ").append(status).append(" ").append(ending).append("
").append(repairNum);
            return str.toString();
        }
    }

    private class Element {
        private double Tw = 0;
        private double Tr = 0;
        private int waitNumber = 0;
        private boolean isWorking = false;
        private byte index = 0;

        public String toString() {
            StringBuilder str = new StringBuilder();
```

```

        str.append(index).append(": ");
        str.append("Tw = ").append(Tw).append(" Tr = ").append(Tr).append("
").append(isWorking);
        str.append("\n");
        return str.toString();
    }
}

```

```

private int N = 30000;
private int n = 4;
private double l = 1.1;
private double m = 0.9;
private int T = 5;
private double step = 0.01;
private Element[] elements = new Element[n];
private int[] E = new int[(int) ((double)T / step) + 1];

```

```

public void testing() {
    elements[0].Tw = 0.497;
    elements[1].Tw = 0.391;
    elements[2].Tw = 0.608;
    elements[3].Tw = 0.232;
    elements[0].Tr = 0.902;
    elements[1].Tr = 0.814;
    elements[2].Tr = 0.175;
    elements[3].Tr = 0.298;
}

```

```

public void bottomLine() throws Exception { // нижние границы
    double K21 = ((2 * m * l) + Math.pow(m, 2)) / (2 * Math.pow(l, 2) + 2 * m * l +
Math.pow(m, 2));
    double K1 = K21 * K21;
    double K11 = m / (m + l);
    double K2 = Math.pow(K11, 2);
    FileWriter file1 = new FileWriter("BottomLine1.txt");
    FileWriter file2 = new FileWriter("BottomLine2.txt");
    for (int t = 0; t <= T; t += 1) {
        StringBuilder str1 = new StringBuilder();
        StringBuilder str2 = new StringBuilder();
        str1.append(t).append(" ").append(K1).append("\n");
        file1.write(str1.toString());
        file1.flush();
        str2.append(t).append(" ").append(K2).append("\n");
        file2.write(str2.toString());
        file2.flush();
    }
}

```

```
}  
}
```

```
public void upperBound() throws Exception { // верхняя граница  
    double K11 = m / (m + l);  
    double K = Math.pow((1 - Math.pow(1 - K11, 2)), 2);  
    FileWriter file = new FileWriter("UpperBound.txt");  
    for (int t = 0; t <= T; t += 1) {  
        StringBuilder str = new StringBuilder();  
        str.append(t).append(" ").append(K).append("\n");  
        file.write(str.toString());  
        file.flush();  
    }  
}
```

```
public void timeModeling(Element e) {  
    double Tw = (-1) * ((Math.log(Math.random())) / l);  
    double Tr = (-1) * ((Math.log(Math.random())) / m);  
    e.Tw = Tw;  
    e.Tr = Tr;  
}
```

```
private void isWorking(Element e, double t, Repair r1, Repair r2) {  
    int tmp = (int) Math.floor(t / (e.Tw + e.Tr));  
    if ((t - (tmp * (e.Tw + e.Tr))) <= e.Tw) {  
        e.isWorking = true;  
    } else {  
        e.isWorking = false;  
        if (r1.repairNum == e.index) {  
            if (r1.ending <= (t + step)) {  
                e.Tr -= step * e.waitNumber;  
                e.waitNumber = 0;  
                r1.free();  
            }  
            return;  
        }  
        if (r2.repairNum == e.index) {  
            if (r2.ending <= (t + step)) {  
                e.Tr -= step * e.waitNumber;  
                e.waitNumber = 0;  
                r2.free();  
            }  
            return;  
        }  
    }  
}
```



```

    }
    if (r1.status || r2.status) { // если одна из бригад свободна
        if (r1.status) {
            r1.createRepair(false, (tmp + 1) * (e.Tw + e.Tr), (byte)e.index);
        } else {
            r2.createRepair(false, (tmp + 1) * (e.Tw + e.Tr), (byte)e.index);
        }
    } else {
        e.Tr += step;
        e.waitNumber++;
    }
}
}
}

```

```

private void simulationOneExperiment() { // возвращает E
    elements = new Element[n];
    for (int i = 0; i < n; i++) {
        elements[i] = new Element();
        elements[i].index = (byte)i;
    }
    for (int i = 0; i < n; i++) {
        timeModeling(elements[i]);
    }
    // testing();
    Repair repair1 = new Repair();
    Repair repair2 = new Repair();
    int index = 0;
    for (double t = 0; t < T; t += step) {
        for (int i = 0; i < n; i++) {
            isWorking(elements[i], t, repair1, repair2);
        }
        if ((elements[0].isWorking || elements[1].isWorking) && (elements[2].isWorking ||
elements[3].isWorking)) {
            E[index]++;
        }
        index++;
    }
    // System.out.println("t = " + t);
    // System.out.println(elements[0]);
    // System.out.println(elements[1]);
    // System.out.println(elements[2]);
    // System.out.println(elements[3]);
    // System.out.println(repair1);
    // System.out.println(repair2);
    // System.out.println();
    // System.out.println();
}

```

```

        clear();
    }

}

public void modeling() throws Exception {
    upperBound();
    bottomLine();
    for (int i = 0; i < N; i++) {
        simulationOneExperiment();
    }
    FileWriter file = new FileWriter("Kr(t).txt");
    int index = 0;
    for (double t = 0; t < T; t += step) {
        StringBuilder str = new StringBuilder();
        str.append(t).append(" ").append((((double)E[index]) / N).append("\n");
        file.write(str.toString());
        file.flush();
        index++;
    }
}

public void clear() {
    for (int i = 0; i < n; i++)
        elements[i].isWorking = false;
}

public void clearFile() {
    try {
        FileWriter file = new FileWriter("UpperBound.txt");
        file.close();
        FileWriter file1 = new FileWriter("BottomLine1.txt");
        file1.close();
        FileWriter file2 = new FileWriter("BottomLine2.txt");
        file2.close();
        FileWriter file3 = new FileWriter("Kr(t).txt");
        file3.close();
    } catch (Exception e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}

public static void main(String[] args) {
    Modeling4 m = new Modeling4();
}

```

```
try {  
    m.modeling();  
    //m.simulationOneExperiment();  
} catch (Exception e) {  
    System.out.println(e.getMessage());  
    e.printStackTrace();  
}  
}  
}
```