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.

1 Задание

Провести имитационное моделирование для схемы, изображенной на рис. 1.

Рисунок 1. Сложная схема резервирования

Входные данные:

N	λ	μ
35000	1.1	0.9

2 Выполнение задания

2.1 Нахождение верхней оценки:

Для нахождения верхней оценки необходимо увеличить количество бригад до количества элементов системы. В данном случае количество бригад будет равно 5. Таким образом, каждая бригада чинит один элемент, т.е. коэффициент готовности одного элемента $K_r = K_{1,1}$.

$$K_{1,1} = \frac{\mu}{\mu + \lambda} = 0.45$$

Чтобы заданная система работала в момент времени, необходимо, чтобы работал элемент 1 или 2 и 3 и 4 или 5, т.е. коэффициент готовности будет равен:

$$K^{+} = \left(1 - \left(1 - K_{1,1}\right)^{2}\right)^{2} * K_{1,1} = (1 - (1 - 0.45)^{2})^{2} * 0.45 = 0.2189$$

2.2 Нахождение нижней оценки:

2.2.1 1 способ. Распределение бригад:

Для нахождения нижней оценки распределим бригады:

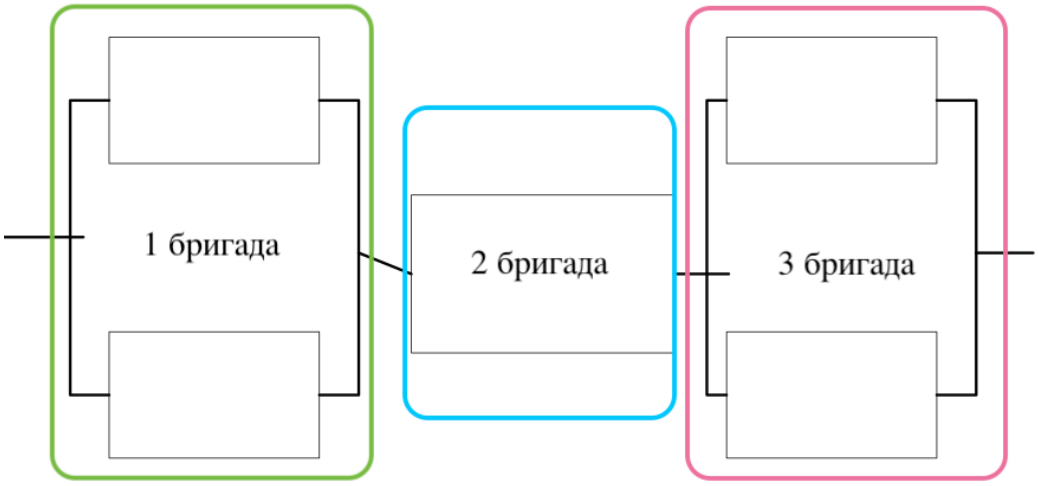

Рисунок 2. Распределение бригад

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

$$K_{2,1} = \frac{2\lambda\mu + \mu^{2}}{2\lambda^{2} + 2\lambda\mu + \mu^{2}} = \frac{1.98 + 0.81}{2.42 + 1.98 + 0.81} = 0.5355$$

Чтобы заданная система работала в момент времени, необходимо, чтобы одновременно работала и 1 группа элементов, и 2 группа элементов, и 3 группа элементов, т.е. коэффициент готовности системы будет равен:

$$K^{-} = K_{2,1}^{2} * K_{1,1} = 0.129$$

2.2.2 2 способ. Исключение систем:

Для нахождения нижней оценки исключим по одному элементу из параллельного соединения:

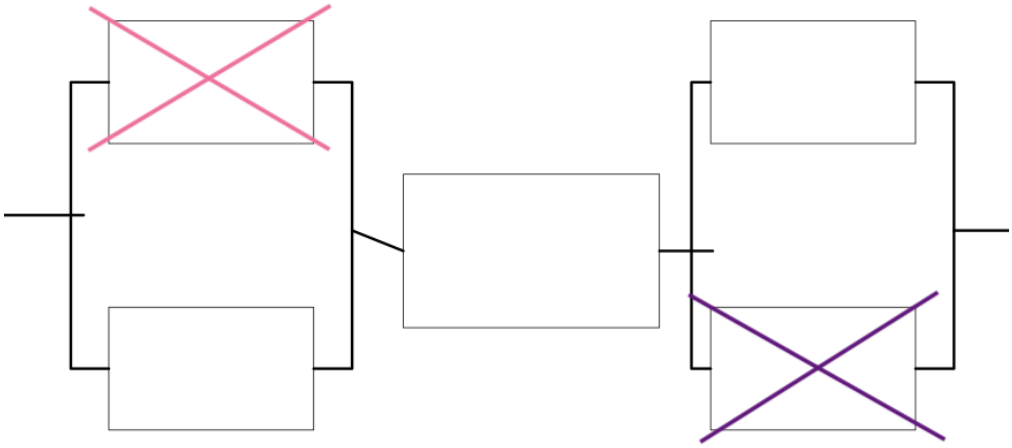

Рисунок 3. Исключение систем

Коэффициент готовности будет равен:

$$K^{-} = K_{1,1}^{3} = 0.0911$$

2.3 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 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 $E(t) = \{0,1\}$, где 0 означает, что система не работает.

Таким образом моделируется $N = 35000$ экспериментов. Коэффициент готовности определяется как:

$$K_r(t) = \sum_{j=1}^{N} \frac{E_j(i * \Delta t)}{N}$$

где i = 1, 2, ..., k.

В результате был получен график зависимости коэффициента готовности от времени:

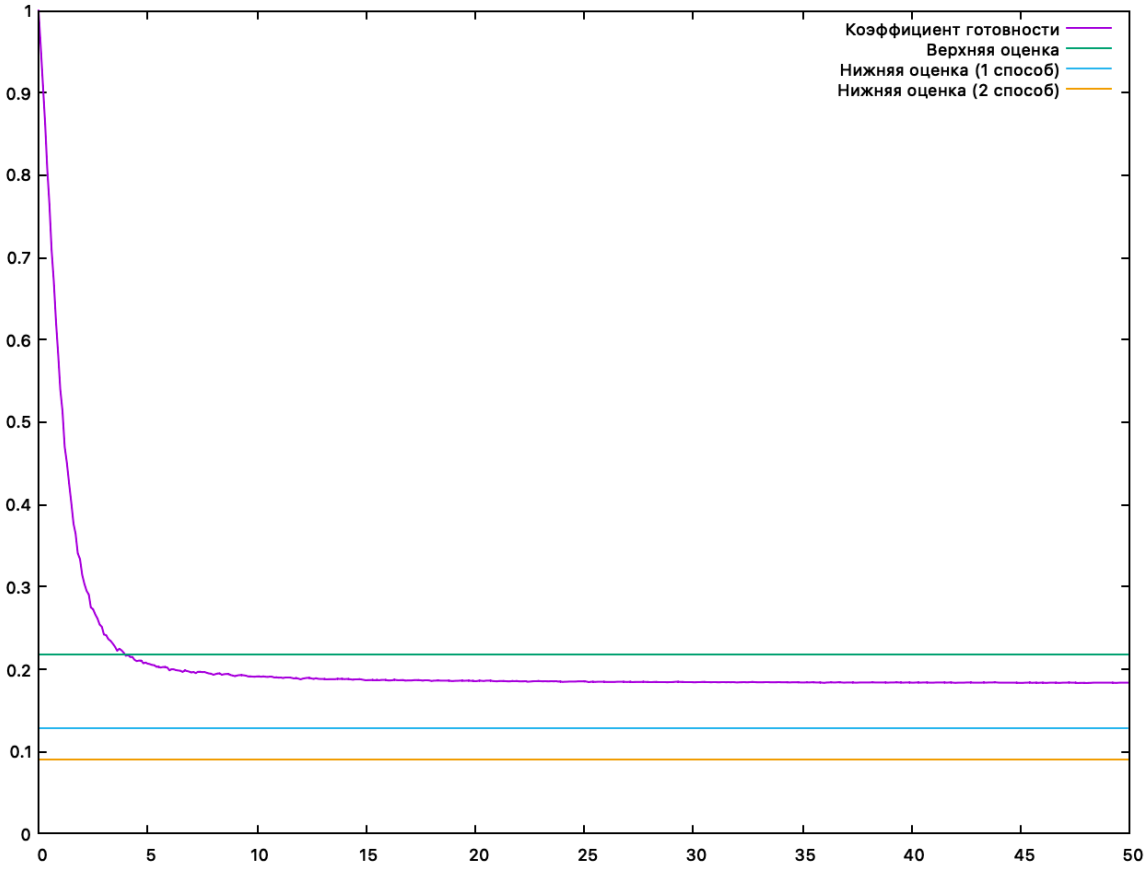

График 1. График зависимости коэффициента готовности от времени

Вывод

В ходе лабораторной работы было выполнено имитационное моделирование функционирования системы со сложной схемой резервирования, были получены верхняя и нижняя оценки коэффициента готовности системы и построен график зависимости коэффициента готовности от времени. По графику видно, что коэффициент готовности, полученный программно, находится между верхней и нижней оценкой коэффициента готовности. Следовательно, программа реализована верно.

Листинг программы

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

import static java.lang.Math.log;
import static java.lang.Math.random;

public class SystemsAndTeams {
    private double N = 35000;
    private int systems = 5;
    private int teams = 3;
    private double m = 0.9;
    private double l = 1.1;
    private int T = 50;
    private double step = 0.1;
    private double K11;
    private double K21;
    private boolean[] System;
    private boolean[] Teams;
    protected boolean[] isRec;

    SystemsAndTeams() {
        System = new boolean[systems];
        Teams = new boolean[teams];
        isRec = new boolean[systems];
        K11 = m / (m + l);
        K21 = (2 * l * m + Math.pow(m, 2)) / (2 * Math.pow(l, 2)
+ 2 * l * m + Math.pow(m, 2));
        Border();
    }

    private void Border() {
        try {
            FileWriter writer = new FileWriter("border.txt",
false);
            double K = Math.pow(1 - Math.pow(1 - K11, 2), 2) *
K11;
            double K1 = Math.pow(K21, 2) * K11;
            double K2 = Math.pow(K11, 3);
            writer.write("Верхняя оценка: " + K + "\nНижняя
оценка (1 способ): " + K1 + "\nНижняя оценка (2 способ): " +
K2);
            writer.flush();

        }
        catch (IOException er) {
            er.printStackTrace();
        }
```

```
    }

    private void saveToFile(double[] Kg) {
        try {
            FileWriter writer = new FileWriter("kg.txt", false);
            for (int i = 0; i < Kg.length; i++) {
                Kg[i] /= N;
            }
            double t = 0.0;
            for (int i = 0; i < Kg.length - 1; i++) {
                String str = t + " " + Kg[i] + "\n";
                writer.write(str);
                writer.flush();
                t += step;
            }
        }
        catch (IOException er) {
            er.printStackTrace();
        }
    }

    public void modulation() {
        double[] Kg = new double[(int) (T / step) + 1];
        for (int i = 0; i < N; i++) {
            Arrays.fill(System, true);
            double[] Tr = getTr();
            double[] Tw = getTw();
            double[] Twait = new double[5];
            double[] Ts = new double[5];
            int ind = 0;

            for (double t = 0.0; t < T; t += step) {
                for(int j = 0; j < systems; j++) {
                    if (t >= Tw[j] + Ts[j] && !isRec[j]) {
                        System[j] = false;
                        if (!Teams[0] || !Teams[1] || !Teams[2]
&& isMax(Twait, j)) {
                            Twait[j] = 0;
                            Ts[j] = startRecovery(t, j);
                        } else {
                            Twait[j] += step;
                        }
                    }
                    if (isRec[j] && t >= Ts[j] + Tr[j] &&
isMax(Twait, j)) {
                        Ts[j] = startWorking(t, j);
                    }
                }
                Kg[ind] += (isWorking()) ? 1 : 0;
                ind++;
            }
        }
        saveToFile(Kg);
```

```
}

public boolean isWorking() {
    if (!System[0] && !System[1])
        return false;

    if (!System[2])
        return false;

    if (!System[3] && !System[4])
        return false;

    return true;
}

public double startWorking(double T, int j) {
    System[j] = true;
    isRec[j] = false;
    if (Teams[0])
        Teams[0] = false;
    else if (Teams[1])
        Teams[1] = false;
    else if (Teams[2])
        Teams[2] = false;
    return T;
}

public double startRecovery(double T, int j) {
    if (!Teams[0]) {
        isRec[j] = true;
        Teams[0] = true;
    } else if (!Teams[1]) {
        isRec[j] = true;
        Teams[1] = true;
    } else if (!Teams[2]) {
        isRec[j] = true;
        Teams[2] = true;
    }
    return T;
}

private double[] getTr() {
    double[] Tr = new double[5];
    for(int i = 0; i < 5; i++) {
        Tr[i] = -log(random()) / m;
    }
    return Tr;
}

public double[] getTw() {
    double[] Tw = new double[5];
    for(int i = 0; i < 5; i++) {
        Tw[i] = -log(random()) / l;
```

```
        }
        return Tw;
    }

    public boolean isMax(double[] Tw, int j) {
        for (int i = 0; i < Tw.length; i++) {
            if (Tw[j] < Tw[i])
                return false;
        }
        return true;
    }

    public static void main(String[] args) {
        SystemsAndTeams systemsAndTeams = new SystemsAndTeams();
        systemsAndTeams.modulation();
    }
}
```