1. **Цель работы:**

Исследовать коэффициент готовности резервируемой восстанавливаемой системы.

2. **Вариант лабораторной работы:**

Рисунок 1. Вариант лабораторной работы.

3. **Моделирование восстанавливаемой системы.**

Начальные параметры: N = 30000 – число экспериментов, μ=1.2 – интенсивность восстановления системы, λ=0.8 – интенсивность отказов системы.

Описание моделирования: для каждого элемента генерируются собственные интервалы нормального функционирования T_i и интервалы восстановления $T_{вi}$ (для каждого эксперимента генерируются собственные значения) по формулам:

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

$$T_{вi} = -\frac{\ln[0:1]}{\mu}.$$

Для построения графика коэффициента готовности используется промежуток времени $T_m = 50$, который поделен на 100 интервалов. Для каждого интервала определяется значение коэффициента готовности по следующей формуле:

$$К_г = \frac{N_t}{N},$$

где N_t – число систем, функционирующий в момент времени t. Система функционирует, когда булева функция condition[0] == 1 || condition[1] == 1) && condition[2] == 1 && (condition[3] == 1 || condition[4] == 1 равна true, где condition[i] – состояние i-ого элемента.

Так как число ремонтных бригад меньше, чем число элементов системы, то при вычислении времени восстановления отдельных элементов учитывается процесс занятости ремонтных бригад. Если на момент отказа элемента свободных бригад в наличии не имеется, то элемент начнет восстанавливать первая освободившаяся бригада. Отказавшие элементы системы обслуживаются в хронологическом порядке их выхода из строя.

4. Расчет оценок сложной системы:

1) Верхняя оценка:

Для расчета верхней оценки предположим, что число ремонтных бригад увеличится до числа элементов в системе, т.е. до 5. Тогда верхнюю оценку можно рассчитать по формуле:

$$К^{+} = К_{1,1} * К_{2,2} * К_{2,2} = К_{1,1} * (1 - (1 - К_{1,1})^{2})^{2},$$

где $К_{1,1} = \frac{\mu}{\mu + \lambda}$ – коэффициент готовности для случая одного элемента и одной ремонтной бригады.

Рисунок 2. Система для верхней оценки.

2) Первая нижняя оценка:

Для расчета первой нижней оценки распределим бригады по частям системы. Тогда первую нижнюю оценку можно рассчитать по формуле:

$$К_{1}^{-} = К_{2,1} * К_{1,1} * К_{2,1},$$

где $К_{2,1} = \frac{2\mu\lambda + \mu^{2}}{2\lambda^{2} + 2\mu\lambda + \mu^{2}}$ – коэффициент готовности для случая двух элементов и одной ремонтной бригады.

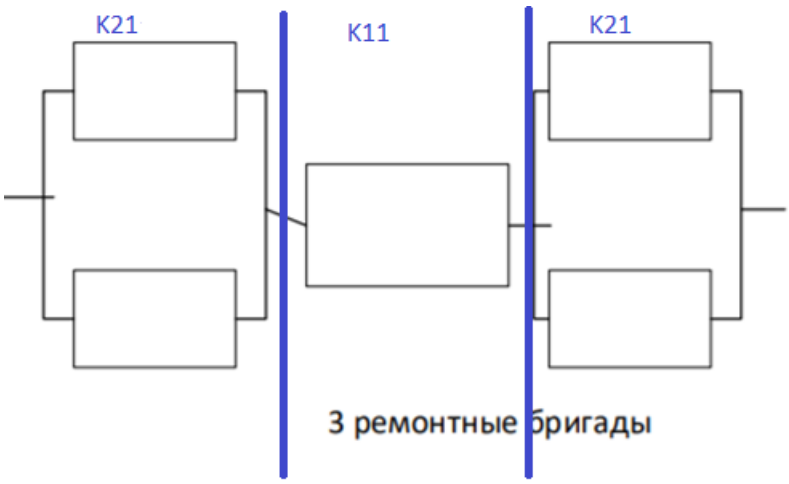

Рисунок 3. Система для первой нижней оценки.

3) Вторая нижняя оценка:

Для расчета второй нижней оценки исключим дублирование системы. Тогда вторую нижнюю оценку можно рассчитать по формуле:

$$К_2^- = К_{1,1} * К_{1,1} * К_{1,1}.$$

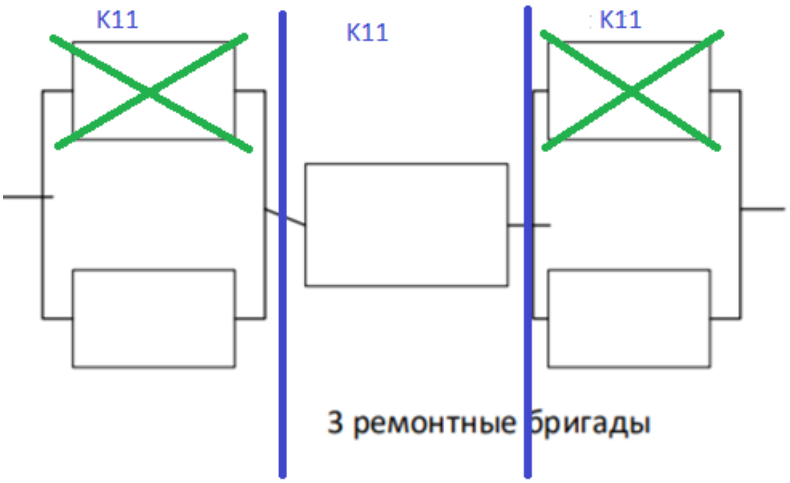

Рисунок 4. Система для второй нижней оценки.

5. Построение графиков:

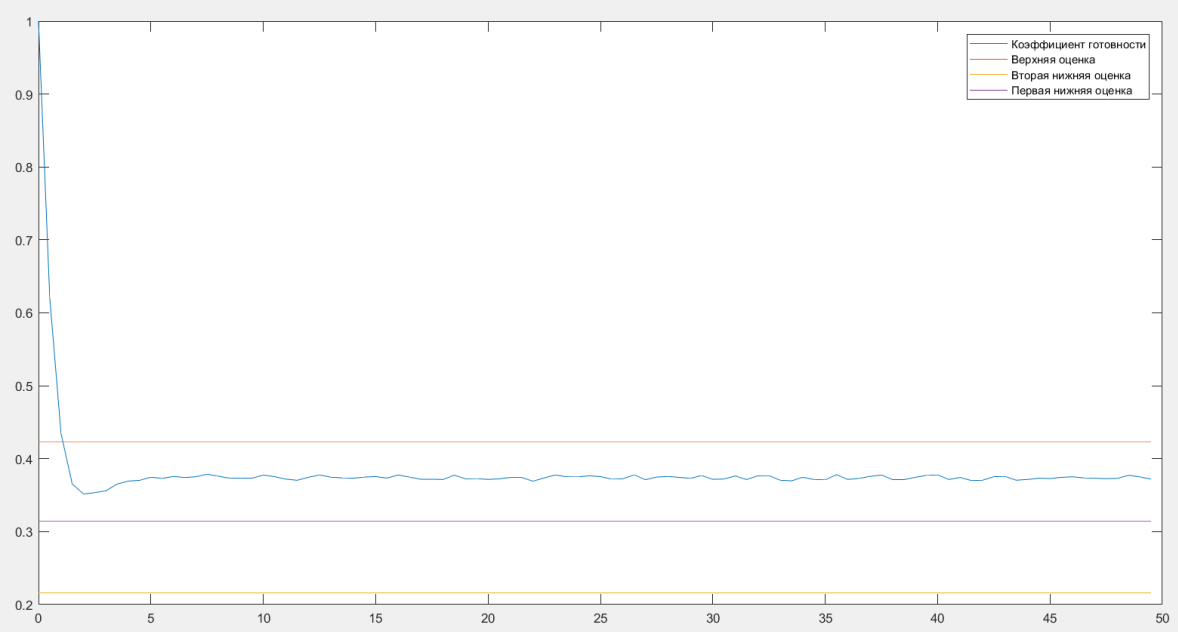

Рис. 5. Графики коэффициента готовности и его оценок.

Из графиков можно сделать вывод о том, что экспериментальное значение коэффициента готовности в установившемся режиме лежит ниже максимальной оценки и выше обеих минимальных оценок. Значит, моделирование заданной системы было произведено верно.

6. Вывод

В ходе выполнения работы, промоделировали работу восстанавливаемой резервируемой системы. Выяснили, что при стремлении t к бесконечности коэффициент готовности переходит в устойчивый режим и является константой, значение которой лежит между границами максимума и минимума.

7. Приложение с кодом:

```
package com.company;

import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import static java.lang.Math.*;

public class Main {
    public static final int elements = 5;

    public static void main(String[] args) {
        int n = 30000;
        double lambda = 0.8;
        double mu = 1.2;
        ArrayList<Double> TiLambda1 = new ArrayList<>();
        ArrayList<Double> TiMu1 = new ArrayList<>();
        ArrayList<Double> TiLambda2 = new ArrayList<>();
        ArrayList<Double> TiMu2 = new ArrayList<>();
        ArrayList<Double> TiLambda3 = new ArrayList<>();
        ArrayList<Double> TiMu3 = new ArrayList<>();
        ArrayList<Double> TiLambda4 = new ArrayList<>();
        ArrayList<Double> TiMu4 = new ArrayList<>();
        ArrayList<Double> TiLambda5 = new ArrayList<>();
        ArrayList<Double> TiMu5 = new ArrayList<>();
        ArrayList<Double> K = new ArrayList<>();
        ArrayList<Double> KEstMax = new ArrayList<>();
        ArrayList<Double> KEstMin1 = new ArrayList<>();
        ArrayList<Double> KEstMin2 = new ArrayList<>();
        double K11 = mu / (lambda + mu);
        double K21 = (2 * mu * lambda + pow(mu, 2)) / (2 *
pow(lambda, 2) + 2 * mu * lambda + pow(mu, 2));
        for (int i = 0; i < n; i++) {
            TiLambda1.add(-log(random()) / lambda);
            TiMu1.add(-log(random()) / mu);
            TiLambda2.add(-log(random()) / lambda);
            TiMu2.add(-log(random()) / mu);
            TiLambda3.add(-log(random()) / lambda);
            TiMu3.add(-log(random()) / mu);
            TiLambda4.add(-log(random()) / lambda);
            TiMu4.add(-log(random()) / mu);
            TiLambda5.add(-log(random()) / lambda);
            TiMu5.add(-log(random()) / mu);
```

```
        }
        double Tm = 50;
        double deltaT = Tm / 100;

        for (double i = 0; i < Tm; i += deltaT) {
            int nt = countNt(TiLambda1, TiMu1, TiLambda2,
TiMu2, TiLambda3, TiMu3, TiLambda4, TiMu4, TiLambda5,
TiMu5, i);
            K.add((double) nt / n);
            KEstMax.add((1 - pow(1 - K11, 2)) * (1 - pow(1
- K11, 2)) * K11);
            KEstMin1.add(K21 * K21 * K11);
            KEstMin2.add(K11 * K11 * K11);
        }
        save(K, "K.txt", deltaT);
        save(KEstMax, "KEstMax.txt", deltaT);
        save(KEstMin1, "KEstMin1.txt", deltaT);
        save(KEstMin2, "KEstMin2.txt", deltaT);
    }

    public static int countNt(ArrayList<Double> TiL1, Ar-
rayList<Double> TiM1, ArrayList<Double> TiL2, Ar-
rayList<Double> TiM2, ArrayList<Double> TiL3, Ar-
rayList<Double> TiM3, ArrayList<Double> TiL4, Ar-
rayList<Double> TiM4, ArrayList<Double> TiL5, Ar-
rayList<Double> TiM5, double t) {
        int count = 0;
        for (int i = 0; i < TiL1.size(); i++) {
            if (work(TiL1.get(i), TiM1.get(i), TiL2.get(i),
TiM2.get(i), TiL3.get(i), TiM3.get(i), TiL4.get(i),
TiM4.get(i), TiL5.get(i), TiM5.get(i), t))
                count++;
        }
        return count;
    }

    public static boolean work(double TiL1, double TiM1,
double TiL2, double TiM2, double TiL3, double TiM3, double
TiL4, double TiM4, double TiL5, double TiM5, double t) {
        double[] TiL = new double[]{TiL1, TiL2, TiL3, TiL4,
TiL5};
        double[] TiM = new double[]{TiM1, TiM2, TiM3, TiM4,
TiM5};
        double[] tmp = new double[elements];
        boolean[] check = new boolean[elements];
        int[] condition = new int[elements];
```

```
        do {
            for (int i = 0; i < elements; i++) {
                tmp[i] += TiL[i];
            }
            checkCondition(tmp, elements, t, check, condi-
tion, 1);
            int indexMax1 = findMax(tmp, elements, 10);
            int indexMax2 = findMax(tmp, elements, in-
dexMax1);
            for (int i = 0; i < elements; i++) {
                if (i == indexMax1 || i == indexMax2)
                    continue;
                tmp[i] += TiM[i];
            }
            int indexMin1 = findMin(tmp, elements, in-
dexMax1, indexMax2);
            if (tmp[indexMin1] > tmp[indexMax2])
                tmp[indexMax2] = tmp[indexMin1];
            tmp[indexMax2] += TiM[indexMax2];
            int indexMin2 = findMin(tmp, elements, in-
dexMax1, indexMin1);
            if (tmp[indexMin2] > tmp[indexMax1])
                tmp[indexMax1] = tmp[indexMin2];
            tmp[indexMax1] += TiM[indexMax1];
            checkCondition(tmp, elements, t, check, condi-
tion, 0);
        } while (!check[0] || !check[1] || !check[2] ||
!check[3] || !check[4]);
        return (condition[0] == 1 || condition[1] == 1) &&
condition[2] == 1 && (condition[3] == 1 || condition[4] ==
1);
    }

    public static void checkCondition(double[] tmp, int
size, double t, boolean[] check, int[] condition, int c) {
        for (int i = 0; i < size; i++) {
            if (tmp[i] >= t && !check[i]) {
                condition[i] = c;
                check[i] = true;
            }
        }
    }

    public static int findMax(double[] tmp, int size, int
indexMax) {
        double max = 0;
```

```
        int index = 100;
        for (int i = 0; i < size; i++) {
            if (i == indexMax)
                continue;
            if (tmp[i] > max) {
                max = tmp[i];
                index = i;
            }
        }
        return index;
    }

    public static int findMin(double[] tmp, int size, int
indexMax1, int indexMax2) {
        double min = 1000000;
        int index = 100;
        for (int i = 0; i < size; i++) {
            if (i == indexMax1 || i == indexMax2)
                continue;
            if (tmp[i] < min) {
                min = tmp[i];
                index = i;
            }
        }
        return index;
    }

    public static void save(List<Double> list, String
filepath, double step) {
        try {
            FileWriter fileWriter = new File-
Writer(filepath);
            for (int i = 0; i < list.size(); ++i) {
                fileWriter.write(i * step + "   " +
list.get(i) + "\n");
                fileWriter.flush();
            }
        } catch (IOException exception) {
            exception.printStackTrace();
        }
    }
}
```