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 №4

Исследование коэффициента готовности резервируемой восстанавливаемой системы

по курсу: Надежность инфокоммуникационных систем

СТУДЕНТ ГР. № 5912		Исаева В.И.
номер группы	подпись, дата	инициалы, фамилия

Санкт-Петербург
2022

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.

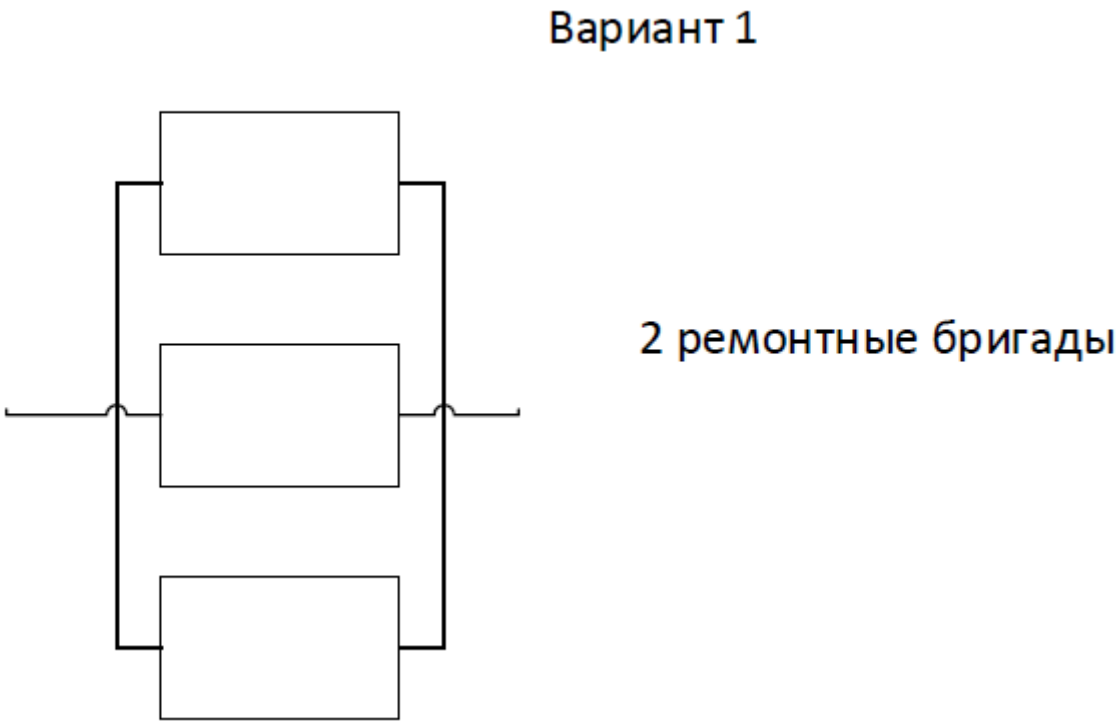

Рисунок 1. Сложная схема резервирования

Входные данные:
N = 30000
ƛ = 0.8
μ = 0.5

2 Ход выполнения работы

2.1 Нахождение верхней оценки:

Для нахождения верхней оценки необходимо увеличить количество бригад до количества элементов системы. В данном случае количество бригад будет

равно 3. Таким образом, каждая бригада чинит один элемент, т.е. коэффициент готовности одного элемента $K_r = K_{1,1}$.

$$K_{1,1} = \frac{\mu}{\mu + \lambda} = 0.3846$$

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

$$K^+ = 1 - \left(1 - K_{1,1}\right)^3 = 1 - (1 - 0.3846)^3 = 0.766937$$

2.2 Нахождение нижней оценки:

2.2.1 Путем распределения бригад:

Для нахождения нижней оценки распределим бригады:

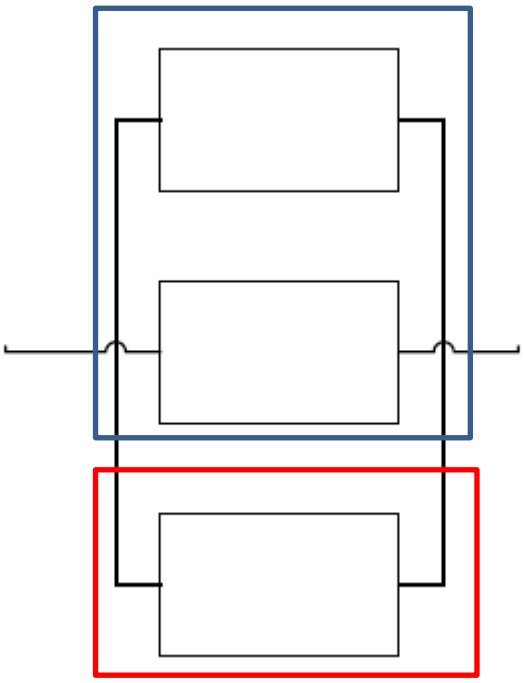

Рисунок 2. Распределение бригад

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

$$K_{2,1} = \frac{2\lambda\mu + \mu^2}{2\lambda^2 + 2\lambda\mu + \mu^2} = \frac{0.4 + 0.25}{1.28 + 0.4 + 0.25} = 0.45$$

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

$$K^- = K_{2,1} + K_{1,1} - K_{2,1} * K_{1,1} = 0.66$$

2.2.2 Путем исключения систем:

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

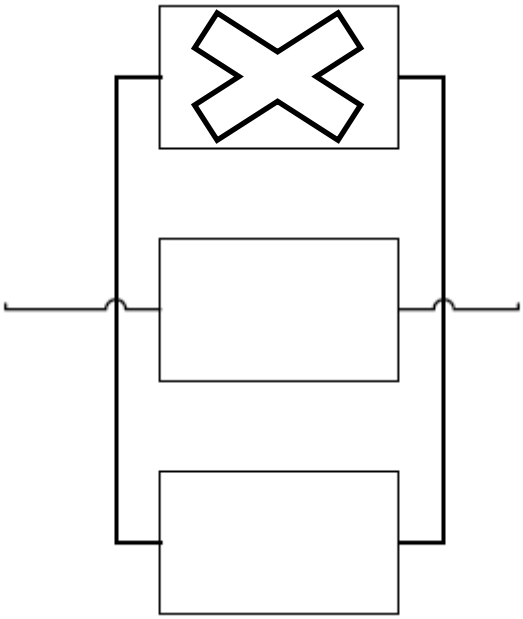

Рисунок 3. Исключение систем

Коэффициент готовности будет равен:

$$K^{-} = 2 * K_{1,1} - K_{1,1}^{2} = 0.6213$$

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 $N = 30000$ 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:

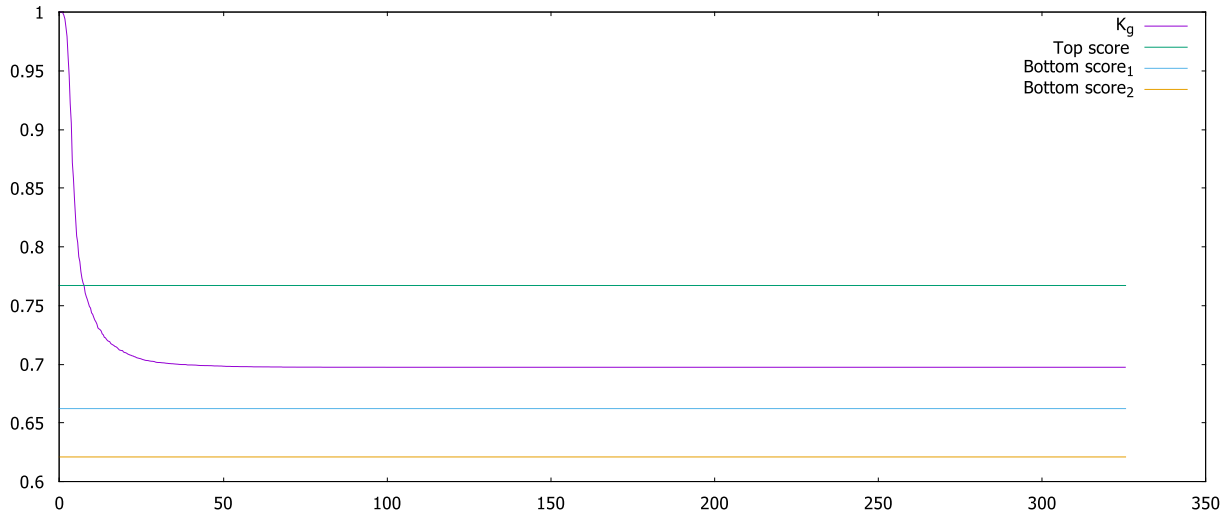

График 1. График зависимости коэффициента готовности от времени

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, в установившемся режиме находится между верхней и нижней оценкой коэффициента готовности. Следовательно, программа реализована верно.

```
public class Model {
    double k = 1000;
    double N = 30000;
    boolean[] S = new boolean[3];
    boolean[] recBr = new boolean[2];
    boolean[] isRec = new boolean[3];
    double[] t_Kg_plus = new double[1];
    double[] t_Kg1_minus = new double[1];
    double[] t_Kg2_minus = new double[1];
    double Tmax;
    double T;
    double Tr;
    double L = 0.8;
    double M = 0.5;

    public void modulation() throws IOException {
        double[] Kg = new double[(int) k + 1];
        double Kg1_1 = 0;
        double Kg2_1 = 0;
        for(int i = 0; i < N; i++) {
            Arrays.fill(S, true);
            double[] Tr = Tr(M);
            double[] Tw = Tw(L);
            T += Tmid(Tw);
            this.Tr += Tmid(Tr);
            double[] Twait = new double[3];
            double[] Ts = new double[3];
            double Tmax = (Tmid(Tr) + Tmid(Tw)) * 100;
            this.Tmax += Tmax/N;
            double dt = Tmax/k;
            int ind = 0;

            for(double t = 0; t < Tmax; t+=dt) {
                for(int j = 0; j < 3; j++) {
                    if (t >= Tw[j] + Ts[j] && !isRec[j]) {
                        S[j] = false;
                        if (!recBr[0] || !recBr[1] && isMax(Twait, j)) {
                            Twait[j] = 0;
                            Ts[j] = startRecovery(t, j);
                        } else {
                            Twait[j] += dt;
                        }
                    }
                    if (isRec[j] && t >= Ts[j] + Tr[j] && isMax(Twait, j)) {
                        Ts[j] = startWorking(t, j);
                    }
                }
                Kg[ind] += (isWorking(S)) ? 1 : 0;
                ind++;
            }
            Kg1_1 += Kg1_1(L, M);
            Kg2_1 += Kg2_1(L, M);
        }

        for(int i = 0; i < Kg.length; i++) {
            Kg[i] /= N;
        }
        Kg1_1 /= N;
        Kg2_1 /= N;

        System.out.println(Kg1_1);
        System.out.println(Kg2_1);
        t_Kg_plus[0] = (1-pow((1-Kg1_1), 3));
        System.out.println(t_Kg_plus[0]);
        t_Kg1_minus[0] = Kg1_1 + Kg2_1 - (Kg1_1 * Kg2_1);
        System.out.println(t_Kg1_minus[0]);
        t_Kg2_minus[0] = 2*Kg1_1 - (Math.pow(Kg1_1, 2));
        System.out.println(t_Kg2_minus[0]);
```

```
    writeIntoFile("expKg.txt", Kg);
    writeIntoFile("tKg_plus.txt", t_Kg_plus);
    writeIntoFile("tKg1_minus.txt", t_Kg1_minus);
    writeIntoFile("tKg2_minus.txt", t_Kg2_minus);
}

public double[] Tr(double M) {
    double[] Tr = new double[4];
    for(int i = 0; i < 4; i++) {
        Tr[i] = -log(random())/M;
    }
    return Tr;
}

public double[] Tw(double L) {
    double[] Tw = new double[3];
    for(int i = 0; i < 3; i++) {
        Tw[i] = -log(random())/L;
    }
    return Tw;
}

public double Tmid(double[] T) {
    double res = 0;
    double size = T.length;
    for(int i = 0; i < size; i++) {
        res+=T[i];
    }
    return res/size;
}

public boolean isWorking(boolean[] S) {
    if (!S[0] && !S[1] && !S[2]) return false;
    return true;
}

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

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

public double Kg1_1(double L, double M) {
    return M/(L+M);
}

public double Kg2_1(double L, double M) {
    return (2*M*L + M*M)/(2*L*L + 2*L*M + M*M);
}

public double[] Kg(double T, double Tw) {
    double[] K = new double[1];
    K[0] = T/(T+Tw);
    return K;
}
public void writeIntoFile(String filename, double[] K) throws IOException {
    File file = new File(filename);
```

```
        if (!file.exists()) file.createNewFile();
        FileWriter writer = new FileWriter(file);
        double t = 0;
        double dt = Tmax /k;
        for(int i = 0; i < k; i++) {
            writer.append(Double.toString(t)).append("
").append(Double.toString(K[i%K.length])).append("\n");
            writer.flush();
            t += dt;
        }
        writer.close();
    }

    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){
        Model model = new Model();
        try {
            model.modulation();
            FileWriter fileWriter  = new FileWriter("Gr1.plt");
            fileWriter.write("set terminal win \n" +
                    "plot \"expKg.txt\" u 1:2 w l t \"K_g\",\\\n" +
                    "\"tKg_plus.txt\" u 1:2 w l t \"Top score\",\\\n" +
                    "\"tKg1_minus.txt\" u 1:2 w l t \"Bottom score_1\",\\\n" +
                    "\"tKg2_minus.txt\" u 1:2 w l t \"Bottom score_2\"");
            fileWriter.flush();
            fileWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```