

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

кандидат техн. наук, доцент

должность, уч. степень, звание

подпись, дата

Марковская Н.В.

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4

ИССЛЕДОВАНИЕ КОЭФФИЦИЕНТА ГОТОВНОСТИ РЕЗЕРВИРУЕМОЙ
ВОССТАНАВЛИВАЕМОЙ СИСТЕМЫ

по курсу: Надежность инфокоммуникационных систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 5912

подпись, дата

Нам Д.О.

инициалы, фамилия

Санкт-Петербург 2022

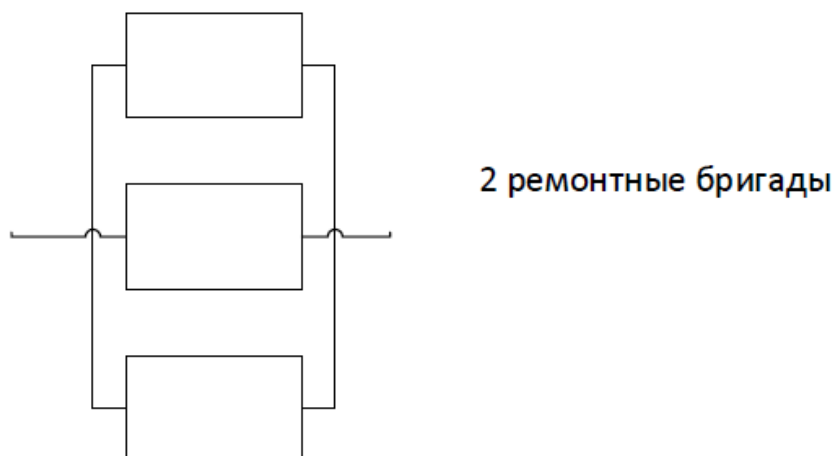
1. Цель работы

Исследовать коэффициент готовности для восстанавливаемых систем и провести имитационное моделирование функционирования системы со сложной схемой резервирования

2. Ход работы

Задание по варианту 1:

Вариант 1



$\lambda = 1.2$ - коэффициент интенсивности отказов

$\mu = 1.1$ - коэффициент интенсивности восстановления

$K_{\Gamma}^{+} = 1 - (1 - K_{\Gamma_{1,1}})^3 = 1 - \left(1 - \frac{\mu}{\mu + \lambda}\right)^3 = 0.857976$ - верхняя оценка коэффициента готовности;

$K_{\Gamma_1}^{-} = K_{\Gamma_{1,1}} + K_{\Gamma_{2,1}} - K_{\Gamma_{1,1}} \cdot K_{\Gamma_{2,1}} = \frac{\mu}{\mu + \lambda} + \frac{2\mu\lambda + \mu^2}{2\lambda + 2\mu\lambda + \mu^2} - \frac{\mu}{\mu + \lambda} * \frac{2\mu\lambda + \mu^2}{2\lambda + 2\mu\lambda + \mu^2} = 0.776729$ - нижняя оценка коэффициента готовности первым способом (распределение бригады);

$K_{\Gamma_2}^{-} = 1 - (1 - K_{\Gamma_{1,1}})^2 = 1 - \left(1 - \frac{\mu}{\mu + \lambda}\right)^2 = 0.727788$ - нижняя оценка коэффициента готовности вторым способом (исключение дублирования системы).

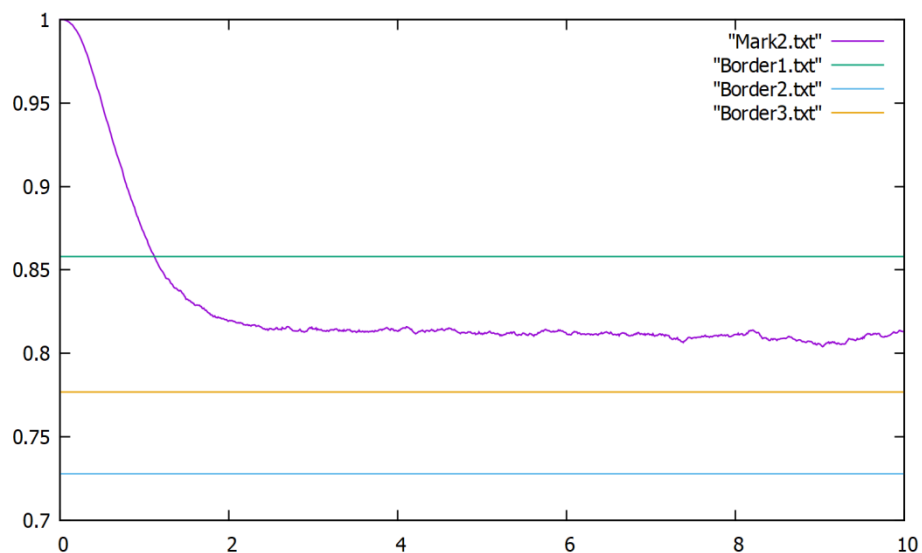


Рисунок 1 - график зависимости коэффициента готовности от времени

3. Вывод

В ходе данной лабораторной работы построен график зависимости коэффициента готовности от времени и его верхняя и нижние границы. По графикам видно, что коэффициент готовности, полученный эмпирическим путем, находится между нижней и верхней границами коэффициента готовности. Исходя из этого можно убедиться, что система реализована верно.

Листинг программы

```

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

public class Second {
    private ArrayList<Integer> model3on2(ArrayList<Double> t, double lambda,
double myu) {
        ArrayList<Integer> res = new ArrayList<>();
        int numberOfBlocks = 3;
        int numOfWorkers = 2;
        int freeWorkers = numOfWorkers;

        ArrayList<Boolean> workingBlocks = new ArrayList<>();
        ArrayList<Boolean> repairingNow = new ArrayList<>();
        ArrayList<Integer> queue = new ArrayList<>();

        for (int i = 0; i < numberOfBlocks; i++) {
            workingBlocks.add(true);
            repairingNow.add(false);
        }

        ArrayList<Double> T = new ArrayList<>();
        for (int i = 0; i < numberOfBlocks; i++) {
            T.add(-Math.log(Math.random()) / lambda);
        }

        for (int step = 0; step < t.size(); step++) {
            ArrayList<Integer> removeFromQueue = new ArrayList<>();
            for (int i = 0; i < T.size(); i++) {
                if (T.get(i) < t.get(step)) {
                    workingBlocks.set(i, false);
                    if (!queue.contains(i)) {
                        if (!repairingNow.get(i)) {
                            queue.add(i);
                        }
                    }
                    else {
                        freeWorkers += 1;
                        double tmp = -Math.log(Math.random()) / lambda;
                        T.set(i, T.get(i) + tmp);
                        repairingNow.set(i, false);
                        workingBlocks.set(i, true);
                    }
                }
            }
            if (queue.contains(i)) {
                if (queue.indexOf(i) < freeWorkers) {
                    double tmp = -Math.log(Math.random()) / myu;
                    T.set(i, T.get(i) + tmp);
                    repairingNow.set(i, true);
                    removeFromQueue.add(i);
                }
            }
        }

        freeWorkers -= removeFromQueue.size();
        for (int r : removeFromQueue) {
            queue.remove((Integer) r);
        }
        if (workingBlocks.get(0) || workingBlocks.get(1) ||

```

```

workingBlocks.get(2)) {
    res.add(1);
}
else {
    res.add(0);
}
}
return res;
}

public Second(double lambda, double myu) throws IOException {
    int N = 50000;
    double dt = 0.01;
    ArrayList<Double> t = fillP(10, dt);

    double min1 = (1 - pow2(1 - (myu / (myu + lambda)))));
    System.out.println(min1);

    double min2 = (myu / (myu + lambda)) +
        (2 * myu * lambda + pow2(myu)) / (2 * pow2(lambda) + 2 * myu
* lambda + pow2(myu)) -
        (myu / (myu + lambda)) * (2 * myu * lambda + pow2(myu)) / (2
* pow2(lambda) + 2 * myu * lambda + pow2(myu));
    System.out.println(min2);

    double max = (1 - Math.pow(1 - (myu / (myu + lambda)), 3));
    System.out.println(max);

    ArrayList<Double> experiment = fillNulls(t.size());
    ArrayList<Integer> exp;
    for (int n = 0; n < N; n++) {
        exp = model3on2(t, lambda, myu);
        for (int i = 0; i < experiment.size(); i++) {
            experiment.set(i, experiment.get(i) + exp.get(i));
        }
    }

    for (int i = 0; i < experiment.size(); i++) {
        experiment.set(i, experiment.get(i) / N);
    }
    Files files = new Files();
    files.graphFiles(2, experiment, dt);
    files.graphFileForNum(1, experiment.size(), max, dt);
    files.graphFileForNum(2, experiment.size(), min1, dt);
    files.graphFileForNum(3, experiment.size(), min2, dt);
}

public static double pow2(double num) {
    return Math.pow(num, 2);
}

private ArrayList<Double> fillNulls(int num) {
    ArrayList<Double> res = new ArrayList<>();
    for (int i = 0; i < num; i++) {
        res.add(0.0);
    }
    return res;
}

private ArrayList<Integer> fillNullsInt(int num) {
    ArrayList<Integer> res = new ArrayList<>();
    for (int i = 0; i < num; i++) {
        res.add(0);
    }
}

```

```

    }
    return res;
}

private ArrayList<Double> fillP(int end, double dt) {
    ArrayList<Double> res = new ArrayList<>();
    int size = (int) (end / dt);
    for (int i = 0; i < size; i++) {
        res.add(dt * i);
    }
    return res;
}
}

```