МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего профессионального образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №52

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Доцент. |  |  |  | Марковская Н.В. |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ |
| Исследование коэффициента готовности резервируемой  восстанавливаемой системы |
| по курсу: СЕТИ И СИСТЕМЫ ПЕРЕДАЧИ ИНФОРМАЦИИ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. | 5512 |  |  |  | К.А.Абдулжамилов |
|  |  |  | подпись, дата |  | инициалы, фамилия |

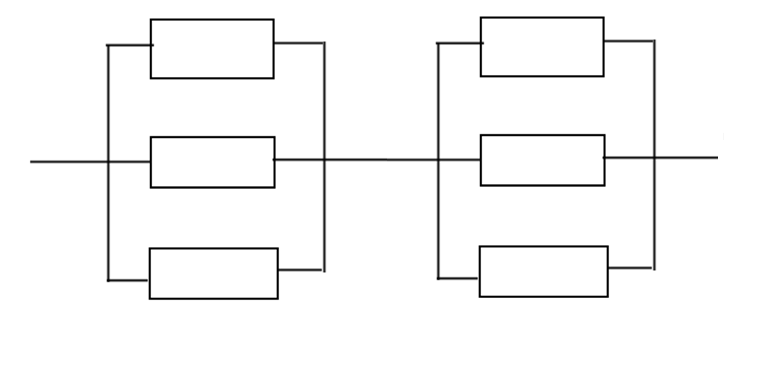
Санкт-Петербург 2019

1. Цель работы:

Смоделировать работу восстанавливаемой системы и расчёт коэффициента готовности.

1. Исходные данные:

Система состоит из 4-ёх подсистем



Каждая подсистема имеет интенсивность отказов λ = 0.8 и интенсивность восстановления μ = 1.2.

Для восстановления данных подсистем выделено 3 рабочих.

1. Имитационное моделирование восстанавливаемой систем без резервирования:

Такая система имеет всего 2 состояния: рабочее и нерабочее. Коэффициент готовности для i-го момента времени вычисляется по формуле

где состояние системы в момент времени.

Теоретический расчёт готовности коэффициента готовности производится по формуле

Результат моделирования и расчетов представлен на графиках

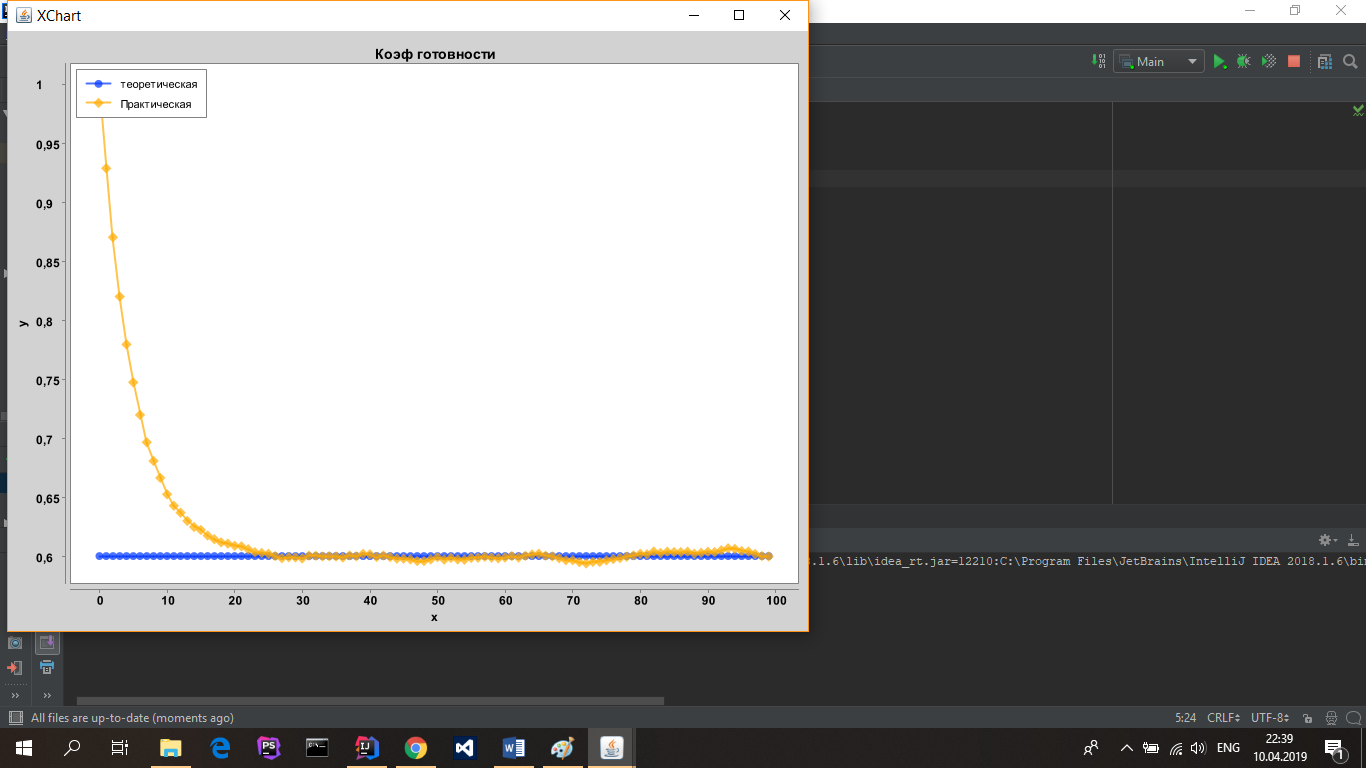


Рисунок 1 - Кг для восстанавливаемой системы без резервирования

1. Имитационное моделирование "стандартной резервируемой системы":

При моделировании данной системы подразумевается, что системы могут ожидать ремонтные бригады, так как их меньше чем подсистем. Расчёт коэффициента готовности осуществляется аналогично предыдущему моделированию. Однако точное значение не определяется, а определяются верхняя и нижняя граница. Для оценки верхней границы будем считать, что работников хватает на все системы. Тогда верхнюю границу считаем как

Нижнюю границу будем рассчитывать, исключая из системы одну из подсистем, находящихся в первом блоке.

Для вышеприведённой схемы коэффициент готовности будет считаться как

Ещё один метод расчёта нижней границы подразумевает объединение нескольких подсистем в одну и распределением рабочих бригад для получившихся подсистем.

В этом случае можем воспользоваться готовыми формулами расчёта ,

Для вышеприведённой схемы коэффициент готовности будет считаться как

Результат моделирования и расчетов представлен на графиках

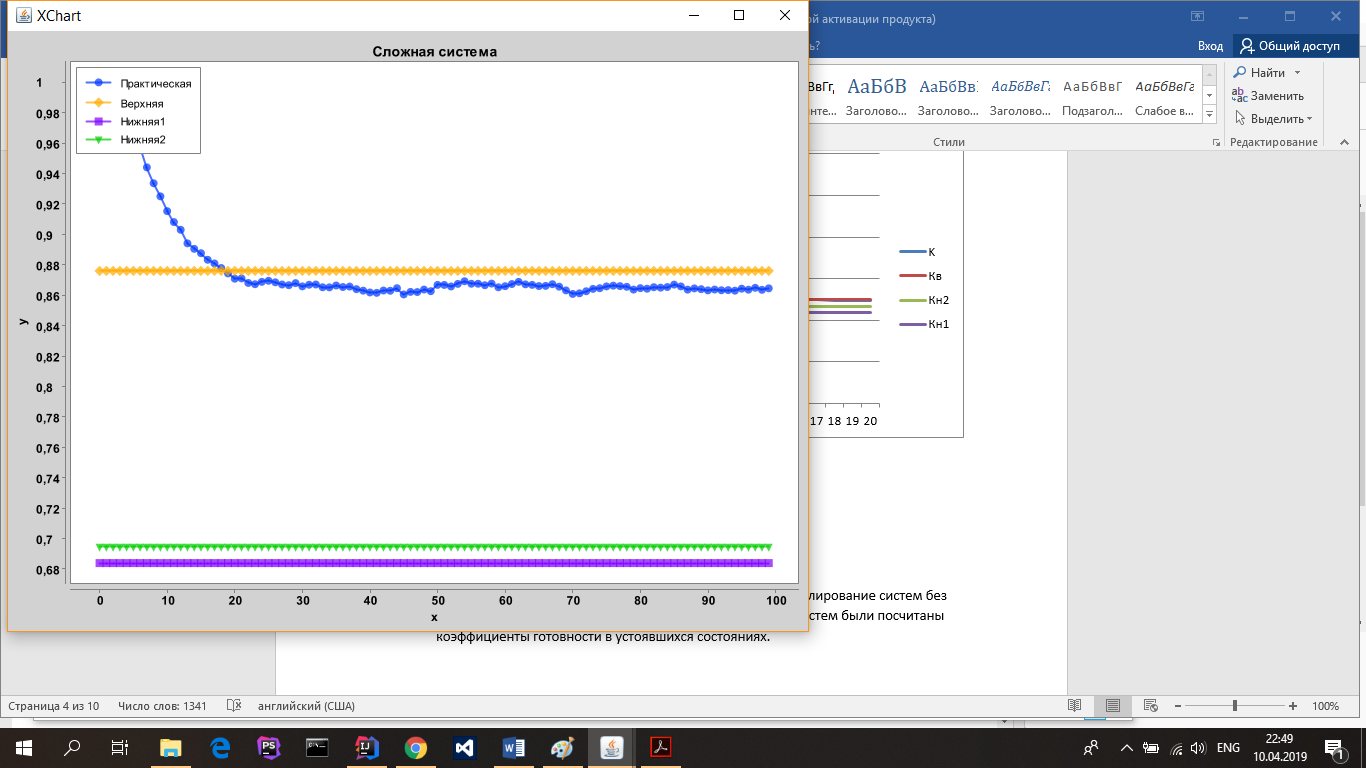


Рисунок 2- Кг для резервируемой системы

Вывод: в данной лабораторной работе было сделано моделирование систем без резервирования и резервируемые системы. Для данных систем были посчитаны коэффициенты готовности в устоявшихся состояниях.

Листинг программы

import javafx.util.Pair;  
  
import java.util.ArrayList;  
  
public class Model {  
 private double mu;  
 private double lymda;  
 private double delta;  
 private int worker;  
 private int N;  
 private int k;  
 private int blocks;  
 public Model(double mu,double lymda,double delta, int worker,int n,int k,int blocks)  
 {  
 this.mu = mu;  
 this.lymda = lymda;  
 this.delta = delta;  
 this.worker = worker;  
 this.N = n;  
 this.k = k;  
 this.blocks = blocks;  
 }  
 public void workFirst()  
 {  
 ArrayList<ArrayList<Pair<Double,Double>>> time = new ArrayList<>();  
 ArrayList<Double> K = new ArrayList<>();  
 ArrayList<Double> K\_teor = new ArrayList<>();  
 for(int i = 0; i < N; i++)  
 {  
 time.add(new ArrayList<Pair<Double,Double>>());  
 }  
 double fullTime = k \* delta;  
 double Tav = 0;  
 double TRecAv = 0;  
 int count = 0;  
 for(int i = 0; i < N; i++)  
 {  
 double tmp = 0;  
 while (tmp < fullTime)  
 {  
 double work = getLymda();  
 double recovery = getMu();  
 time.get(i).add(new Pair(tmp,tmp+work));  
 Tav += work;  
 TRecAv += recovery;  
 tmp = tmp + work + recovery;  
 count++;  
 }  
 }  
 Tav /= count;  
 TRecAv /= count;  
 for(int i = 0; i < k; i++)  
 {  
 int counter = 0;  
 for(int j = 0; j < N; j++)  
 {  
 counter+= Iswork(time.get(j), i \* delta);  
 }  
 K.add((double)counter/N);  
 K\_teor.add(Tav/(Tav + TRecAv));  
 }  
 Schedule.*print*(K\_teor,K,"Коэф готовности", k);  
 }  
 public void workSecond()  
 {  
 ArrayList<ArrayList<ArrayList<Pair<Double,Double>>>> list = new ArrayList<>();  
 ArrayList<Double> K = new ArrayList<>();  
 ArrayList<Double> K\_teorUp = new ArrayList<>();  
 ArrayList<Double> K\_teorDown1 = new ArrayList<>();  
 ArrayList<Double> K\_teorDown2 = new ArrayList<>();  
  
 for(int i = 0; i < N;i++)  
 {  
 list.add(Oneblock());  
 }  
 for(int i = 0; i < k; i++)  
 {  
 int counter = 0;  
 for(int j = 0; j < N; j++)  
 {  
 if(Workhard(list.get(j), i \* delta))  
 {  
 counter++;  
 }  
 }  
 K.add((double)counter/N);  
 K\_teorUp.add(Math.*pow*(1-Math.*pow*(1 - (mu/(mu+lymda)),3),2));  
 K\_teorDown1.add(Math.*pow*(1-(1 - ((2\*mu\*lymda + Math.*pow*(mu,2))/(Math.*pow*(2\*lymda,2)+ 2\*mu\*lymda + Math.*pow*(mu,2))))\*(1 - (mu/(mu+lymda))),2));  
 K\_teorDown2.add((1-Math.*pow*(1 - (mu/(mu+lymda)),2)) \* (1-(1 - ((2\*mu\*lymda + Math.*pow*(mu,2))/(Math.*pow*(2\*lymda,2)+ 2\*mu\*lymda + Math.*pow*(mu,2))))\*(1 - (mu/(mu+lymda)))));  
 }  
 Schedule.*print4*(K,K\_teorUp,K\_teorDown1,K\_teorDown2,"Сложная система",k);  
  
  
 }  
 private boolean Workhard(ArrayList<ArrayList<Pair<Double,Double>>> list , double t)  
 {  
 if(Iswork(list.get(0),t) == 0 && Iswork(list.get(1),t) == 0 && Iswork(list.get(2),t) == 0)  
 {  
 return false;  
 }  
 else if(Iswork(list.get(3),t) == 0 && Iswork(list.get(4),t) == 0 && Iswork(list.get(5),t) == 0)  
 {  
 return false;  
 }  
 return true;  
 }  
  
 private ArrayList<ArrayList<Pair<Double,Double>>> Oneblock()  
 {  
 ArrayList<ArrayList<Pair<Double,Double>>> timeWork = new ArrayList<>();  
 double workerB[] = new double[worker];  
 boolean times[] = new boolean[blocks];  
 double t[] = new double[blocks];  
 double recovery[] = new double[blocks];  
 double fullTime = k \* delta;  
 double time = 0;  
 for(int i = 0; i < worker; i++)  
 {  
 workerB[i] = 0.0;  
 }  
 for(int i = 0; i< blocks ; i++)  
 {  
 timeWork.add(new ArrayList<>());  
 times[i] = false;  
 t[i] = 0;  
 }  
 boolean allFlag = true;  
 double nowTime = 0;  
 for(int i = 0; i < blocks; i++)  
 {  
  
 double workT = getLymda();  
 recovery[i] = getMu();  
 double tmpsr = t[i];  
 timeWork.get(i).add(new Pair<Double, Double>(tmpsr,(t[i] + workT)));  
 t[i] = t[i] + workT;  
  
 }  
 while (nowTime <= fullTime)  
 {  
 for(int i = 0; i < blocks; i++)  
 {  
 if(times[i])  
 {  
 double workT = getLymda();  
 double tmps = t[i];  
 t[i] = t[i] + workT;  
 double tmpss = t[i];  
 timeWork.get(i).add(new Pair<Double, Double>(tmps,tmpss));  
 times[i] = false;  
 }  
 }  
 ArrayList<Integer> tmp = getMin(t);  
 ArrayList<Integer> tmp1;  
 nowTime += 0.1;  
 for(int i = 0; i < blocks; i++)  
 {  
 int index = tmp.get(i);  
 if(t[index] <= nowTime) {  
 tmp1 = getMin(workerB);  
 if (t[index] < workerB[tmp1.get(0)]) {  
 t[index] = workerB[tmp1.get(0)];  
 times[index] = true;  
 }  
 else {  
 times[index] = true;  
 }  
 t[index] += getMu();  
 workerB[tmp1.get(0)] = t[index];  
 }  
 }  
 }  
 return timeWork;  
 }  
 private ArrayList<Integer> getMin(double[] time)  
 {  
 ArrayList<Integer> minn = new ArrayList<>();  
 double [] mas = new double[time.length];  
 for(int i = 0; i < time.length; i++)  
 {  
 mas[i] = time[i];  
 }  
 boolean flag = true;  
 while (minn.size() != time.length)  
 {  
 int min = 0;  
 for(int i = 0; i < mas.length;i++)  
 {  
 if(mas[min] > mas[i])  
 {  
 min = i;  
 }  
 }  
 minn.add(min);  
 mas[min] = 100;  
 }  
 return minn;  
 }  
 private int Iswork(ArrayList<Pair<Double,Double>> time,double t)  
 {  
 for(int i = 0; i < time.size(); i++)  
 {  
 Pair<Double,Double> tmp = time.get(i);  
 if(tmp.getKey() <= t && tmp.getValue() >= t)  
 {  
 return 1;  
 }  
 }  
 return 0;  
 }  
  
 private double getLymda()  
 {  
 return Math.*log*(Math.*random*())/lymda\*(-1);  
 }  
 private double getMu()  
 {  
 return Math.*log*(Math.*random*())/mu\*(-1);  
 }  
  
  
}

}  
 private double getT()  
 {  
 return (-1 / lyambda) \* Math.*log*(r.nextDouble());  
 }  
 private double getTv()  
 {  
 return (-1 / mu) \* Math.*log*(r.nextDouble());  
 }  
  
 private static void wtf(String filename, double [][] value )  
 {  
 try  
 {  
 File f = new File(filename);  
 BufferedWriter bw = new BufferedWriter(new FileWriter(f));  
 for(int i = 0; i < value.length; i++)  
 {  
 for(int j = 0; j < value[i].length; j++)  
 {  
 bw.write(String.*valueOf*(value[i][j]).replace(',', '.') + "\r\n");  
 }  
 bw.write("\r\n");  
 }  
 bw.flush();  
 bw.close();  
 }  
 catch (IOException ioe)  
 {  
 System.*out*.println(ioe);  
 }  
 }  
}

public class Main {  
  
 public static void main(String[] args) {  
 double lyambda = 0.8;  
 double mu = 1.2;  
 double delta = 0.1;  
 int k = 100;  
 int N = 50000;  
 int workers = 4;  
  
 Model mod = new Model(mu, lyambda, delta,workers, N,k,6);  
 mod.workFirst();  
 mod.workSecond();  
 }  
}