МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования "Рыбинский государственный авиационный технический университет имени П.А. Соловьева"

КАФЕДРА "ЭМиЭИС"

Контрольня работа №2

по дисциплине "Математическое и имитационное моделирование"

Вариант 2

Группа: ЗИП-14

Студент: Лебедев Е. В.

Преподаватель: Буров Н. Н.

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Рыбинск 2018

Содержание

[Введение 3](#_Toc503730724)

[1. Имитационное моделирование 4](#_Toc503730725)

[2. Имитационное исследование 10](#_Toc503730726)

[2.1. Испытание и исследование свойств имитационной модели 10](#_Toc503730727)

[2.2. Планирование и проведение направленного вычислительного эксперимента модели 12](#_Toc503730728)

[2.3. Интерпретация результатов эксперимента 13](#_Toc503730729)

[Заключение 15](#_Toc503730730)

[Список литературы 16](#_Toc503730731)

[Приложение. 17](#_Toc503730732)

# Введение

Целью контрольной работы является овладение технологией и приёмами практического решения задач моделирования процессов функционирования системы на ЭВМ, что необходимо для того, чтобы научиться моделированию и освоить широкий круг его возможностей.

В данной контрольной работе рассматривается моделирование процессов в Q-схемах – одном из важнейших, с точки зрения применения на практике, классов математических схем, разработанных для формализации процессов функционирования систем массового обслуживания (СМО) в теории массового обслуживания. Предметом изучения в теории массового обслуживания являются системы, в которых появление заявок (требований) на обслуживание и завершение обслуживания происходит в случайные моменты времени, т.е. характер их функционирования носит стохастический характер. Следует отметить, что СМО описывают различные по своей физической природе процессы функционирования экономических, производственных, технических и других систем, например потоки поставок продукции некоторому предприятию, потоки деталей и комплектующих изделий на сборочном конвейере цеха, заявки на обработку информации в ЭВМ от удаленных терминалов и т.д.

# Имитационное моделирование

Основная проблема исследования системы заключается в проверке предположений, выдвинутых относительно ее функционирования, на основе анализа результатов, полученных при выполнении программной реализации модели на ЭВМ.

Объектом моделирования служит система передачи цифровой информации. Основной целью имитационного исследования является построение достоверной модели, которая позволит изучить особенности моделируемой системы, в частности:

– поведение системы при длительном функционировании;

– определение частот уничтожения транзактов и частот подключения ресурсов.

Также имитация поможет определить возможность оптимизации исследуемой системы.

Из условия задания нам неизвестно как располагаются каналы передачи данных, поэтому предположим два варианта. Первый вариант - это пакеты передаются по двум каналам одновременно, т.е. каналы располагаются параллельно. При этом пакеты подаются поочередно в накопители каналов с интервалом 6±3 мс, таким образом, минимальный интервал поступления пакета в накопитель канала составляет 6 мс (3мс+3мс), что больше времени передачи по каналу (5 мс). Получаем, что частота поступления пакетов в очередь (накопитель) канала ниже, чем частота извлечения их из очереди, следовательно, пакеты не будут задерживаться в буфере и суммарное время передачи не превысит 10 мс.

Вторым вариантом предположим, что пакеты передаются последовательно – сначала по одному каналу, затем по второму. В таком случае, существует вероятность накопления пакетов в буфере первого канала и увеличения времени передачи (более 10 мс). Примем за основу вариант с последовательным расположением каналов. Анализируя исходные данные, можно выдвинуть следующие гипотезы:

– в накопителе первого канала ожидается один и более пакетов в конкретный момент времени;

– в накопителе второго канала ожидается не более одного пакета в конкретный момент времени;

– предполагается, что часть пакетов будет иметь время передачи более 10 мс, следовательно, ожидается периодическое подключение ресурса;

Исследуемая система массового обслуживания относится к классу систем, в которых требуется ожидание (очередь). В таких системах заявка, поступившая в момент, когда все каналы заняты, становится в очередь и ожидает, пока не освободится один из каналов. Когда канал освобождается, одна из заявок, стоящих в очереди, принимается к обслуживанию. В системе заявки обслуживаются в порядке поступления. Время ожидания неограниченное, т.е. заявка ждет, когда освободится канал, который примет ее к обслуживанию.

K2

H2

K1

H1

U

W

Рисунок 1 – Структурная схема модели в символике Q-схем

- W – источник, отражающий входящий поток пакетов;

- H1 – очередь перед каналом А;

- H2 – очередь перед каналом B;

- K1 – канал А;

- K2 – канал B;

- Y – выходящий поток пакетов;

- U – управление.

Для реализации системы я выбрал язык программирования java, являющийся современным высокоуровневым языком, имеющий удобные инструменты программирования и позваляющий запускать программу на большенстве платформах скомпилировав код только один раз. Блок-схема общего алгоритма работы системы показана на рисунке 2.

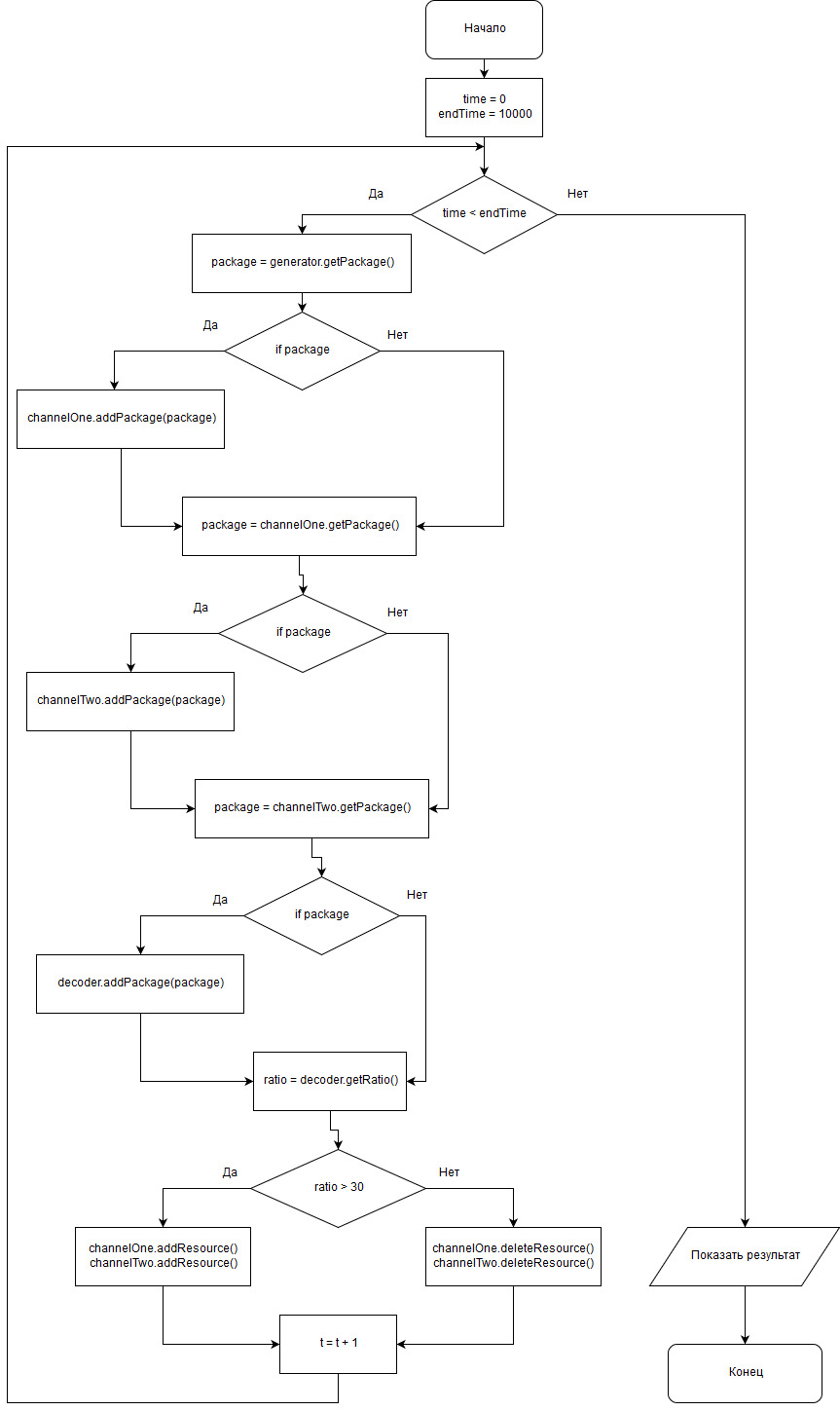


Рисунок 2. Блок-схема общего алгоритма работы системы.

Система поделена на модули (классы): канал, генератор, пакет, ресурс, декодер. Время системы представлено переменной time, где каждую еденицу берем за 1 мс. Каждый тик время увеличивается на 1мс. Генератор генерирует пакеты, используя функцию рандомизации встроенную в стандартную библиотеку языка Java. Получив рандомное число от 3 до 9 (по условию), генератор сохраняет его и каждый тик уменьшает на 1. Как только число станет равно 0, генератор сгенерирует пакет и получит новое рандомное число. В сгенерированном пакете есть счетчик, который увеличивается на 1 в каждый тик. Сгенерированный пакет отправляет в первый канал, где будет находится либо 5 секунд если очередь пуста и ресурс не подключен, либо 4 если ресурс подключен, либо будет ждать своей очереди. На первый пакет в очереди в канале устанавливается счетчик имитируя передачу этого пакета по каналу, каждый тик счетчки уменьшается и как только будет равен 1 пакет будет считать приняты и может передаваться далее во второй канал. Второй канал работает также как и первый. После него пакет передаётся в декодер. Декодер проверяет счетчик пришедшего пакета, это время за которое доставлен пакет. В декодере имеется 3 переменные: общее число пакетов, количество уничтоженых пакетов и процент уничтоженных пакетов. Если время больше 10мс, то декодер считает пакет негодным и увеличивает счетчки уничтоженых пакетов на 1, и увеличивает счетчик общего числа пакетов на 1 в любом случае. Далее проверяется соотношение уничтоженых пакетов и если оно больше 30%, то в каналы добавляется ресурс, тем самым задавая скорость канала 4 мс вместо стандартных 5 мс. Когда соотношение уничтоженых пакетов станет меньше 30%, то ресурс из каналов удаляется и время передачи опять становится 5мс. Программа завершается когда time станет больше определенного нами времени окончании работы равного 10с (10 000мс). И в завершение показывает отчет о работе моделирования. Интерфейс системы показан на рисунке 3.

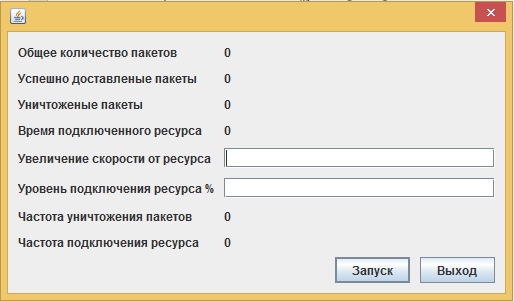


Рисунок 3. Основное окно системы.

# Имитационное исследование

Цель: испытать и исследовать свойства модели передачи цифровой информации, выявить оптимальное значение одного из свойств.

Задачи: провести 10 прогонов системы и получить характеристики свойств модели, рассчитать доверительный интервал для этих свойств, провести анализ чувствительности, спланировать и провести направленный вычислительный эксперимент и выявить оптимальное значение одного из свойств.

# Испытание и исследование свойств имитационной модели

Проведем 10 прогонов модели, данные запишем в таблицу 1.

Таблица 1. Результаты испытаний имитационной модели.

|  |  |
| --- | --- |
| Частота уничтожения пакетов | Частота подключения ресурса |
| 0.31 | 0.18 |
| 0.31 | 0.23 |
| 0.311 | 0.206 |
| 0.31 | 0.168 |
| 0.31 | 0.144 |
| 0.31 | 0.199 |
| 0.31 | 0.182 |
| 0.308 | 0.196 |
| 0.309 | 0.164 |
| 0.31 | 0.18 |

Оценим точность результатов имитационного моделирования.

Математическое ожидание:

Таблица 2. Математическое ожидание.

|  |  |
| --- | --- |
| Частота уничтожения пакетов | Частота подключения ресурса |
| 0.31 | 0.185 |

Доверительный интервал:

Таблица 3. Доверительный интервал.

|  |  |
| --- | --- |
| Частота уничтожения пакетов | Частота подключения ресурса |
| 0.31 +- 0.001 | 0.185 +- 0.016 |

Оценим чувствительность результатов имитационного моделирования на изменение времени прихода пакетов с 5+-3 мс до 7+-3 мс в среднем.

Таблица 4. Чувствительность к изменению времени прихода пакетов.

|  |  |  |
| --- | --- | --- |
| Частота уничтожения пакетов | Частота подключения ресурса | Время прихода пакетов |
| 0.311 | 0.712 | х = 5 +- 3 |
| 0.176 | 0 | х = 7 +- 3 |
| -0.554 | -2.000 | dx |

По результатам видно, что чем больше время генерации пакетов, тем меньше пакетов уничтожается в декодере и тем меньше частота подключения ресурса.

Оценим чувствительность к изменению уровня уничтожения пакетов

Таблица 5. Чувствительность к изменению уровня уничтожения пакетов.

|  |  |  |
| --- | --- | --- |
| Частота уничтожения пакетов | Частота подключения ресурса | Уровень уничтожения пакетов % |
| 0.209 | 0.384 | x=20 |
| 0.401 | 0.039 | x=40 |
| 0.630 | -1.631 | dx |

По результатам видно, что чем ниже уровень уничтожения пакетов, тем меньше пакетов уничтожается и тем больше частота подключения ресурса.

# Планирование и проведение направленного вычислительного эксперимента модели

План эксперимента:

Фактор изменения - уровень уничтоженных пакетов, после которого подключается дополнительный ресурс. Контролируемые факторы: частота уничтоженных пакетов и частота подключения ресурса. Определить оптимальный уровень уничтоженных пакетов.

Диапазон уровня уничтоженных пакетов 22%-40% шаг 2%. Количество прогонов 100.

Результаты проведения вычислительного эксперимента модели представлены в таблице 6.

Таблица 6. Результаты проведения направленного вычислительного эксперимента.

|  |  |  |
| --- | --- | --- |
| Уровень уничтоженных пакетов % | Частота уничтожения пакетов | Частота подключения ресурса |
| 22 | 0.23 | 0.34 |
| 24 | 0.25 | 0.304 |
| 26 | 0.269 | 0.26 |
| 28 | 0.289 | 0.224 |
| 30 | 0.309 | 0.18 |
| 32 | 0.328 | 0.155 |
| 34 | 0.349 | 0.124 |
| 36 | 0.367 | 0.097 |
| 38 | 0.385 | 0.066 |
| 40 | 0.403 | 0.042 |

# Интерпретация результатов эксперимента

Оценим точность.

Математическое ожидание.

Таблица 7. Математическое ожидание.

|  |  |
| --- | --- |
| Частота уничтожения пакетов | Частота подключения ресурса |
| 0.318 | 0.179 |

Доверительный интервал.

Таблица 8. Доверительный интервал.

|  |  |
| --- | --- |
| Частота уничтожения пакетов | Частота подключения ресурса |
| 0.318 +- 0.038 | 0.179 +- 0.066 |

Построим график с результатами эксперимента.

Рисунок 4. Результаты эксперимента.

Корреляция между уровнем уничтожения пакетов и частотой уничтожения пакетов 0.999. Корреляция между уровнем уничтожения пакетов и частотой подключения ресурса -0.996.

Как видно из графика оптимальным для частоты уничтожения пакетов и частоты подключения ресурса будет решение снизить уровень уничтоженных пакетов до 26%.

# Заключение

Результатом данной работы стало построение программы, моделирующей процесс функционирования заданной системы и построение схемы моделирующего алгоритма, проведение имитационного исследования работы данной модели.

Также был спланирован и проведен вычислительный эксперимент, с помощью которого мы выяснили, что корреляция между изменением уровня уничтоженных пакетов и частотой уничтоженных пакетов 0.999, корреляция между уровнем уничтожения пакетов и частотой подключения ресурса -0.996. Мы выяснили что, оптимальным уровнем уничтоженных пакетов, после которого следует подключать ресурс является уровень 26%.

# Список литературы

1. Бусленко Н.П. Моделирование сложных систем. -М.: Наука, 1978. – 399 с.
2. Василенков В.П. Математическое моделирование социально-экономических процессов: практический курс для студентов специальностей «Менеджмент организации» и «Государственное и муниципальное управление» / В.П.Василенков, И.Б.Болотин; Смол.гос.ун-т. – Смоленск: Изд-во СмолГУ, 2009.
3. Варфоломеев В.И. Алгоритмическое моделирование элементов экономических систем. Практикум.-М.: Финансы и статистика, 2004. – 265 с.
4. Введение в математическое моделирование: Учеб.пособие / Под ред. П.В. Трусова. – М.: Университетская книга, Логос, 2007. – 440 с.
5. Емельянов А.А. Структурный анализ и динамические имитационные модели в экономике. - М.: Финансы и статистика, 1998. – 358 с.
6. Емельянов А.А. Имитационное моделирование экономических процессов. – М., 2004.
7. Интрилигатор М. Математические методы оптимизации и экономическая теория / Пер. с англ. Г. И. Жуковой, Ф. Я. Кельмана. – М.: Айрис-прсс, 2002. – 576 с.: ил. – (Высшее образование).
8. Калянов Г.Н. CASE структурный системный анализ (автоматизация и применение).-М.:Лори, 1996. – 243 с.

# Приложение.

Исходный код программы

Class Main

**package main;**

**import g.Gui;**

**public class Main {**

**public int all = 0;**

**public int fault = 0;**

**public int resource = 0;**

**public int success = 0;**

**public void start(int runs, int gen\_base, int res\_help, int procent) {**

**int total\_runs\_all = 0;**

**int total\_runs\_fault = 0;**

**int total\_runs\_successs = 0;**

**int total\_runs\_resource = 0;**

**for (int i = 0; i < runs; i++) {**

**long time = 0;**

**long endTime = 10 \* 1000;**

**Generator generator = new Generator(gen\_base);**

**Channel channelOne = new Channel(res\_help);**

**Channel channelTwo = new Channel(res\_help);**

**Decoder decoder = new Decoder();**

**Resource resource = new Resource();**

**while (time < endTime || generator.all() > decoder.all) {**

**Package newPackage = null;**

**if (time < endTime) {**

**newPackage = generator.getPackage();**

**}**

**if (newPackage != null) {**

**channelOne.addPackage(newPackage);**

**}**

**Package package1 = channelOne.getPackage();**

**if (package1 != null) {**

**channelTwo.addPackage(package1);**

**}**

**Package package2 = channelTwo.getPackage();**

**if (package2 != null) {**

**decoder.addPackage(package2);**

**}**

**int ratio = decoder.getGetDeletedRatio();**

**if (ratio > procent) {**

**channelOne.addResource();**

**channelTwo.addResource();**

**resource.call();**

**} else {**

**channelOne.deleteResource();**

**channelTwo.deleteResource();**

**}**

**time += 1;**

**channelOne.updatePackagesTime();**

**channelTwo.updatePackagesTime();**

**}**

**total\_runs\_all += generator.all();**

**total\_runs\_fault += decoder.fault;**

**total\_runs\_successs += decoder.success;**

**total\_runs\_resource += resource.getCalled();**

**}**

**this.all = total\_runs\_all/runs;**

**this.fault = total\_runs\_fault/runs;**

**this.success = total\_runs\_successs/runs;**

**this.resource = total\_runs\_resource/runs;**

**}**

**public static void main(String[] args) {**

**Gui dialog = new Gui();**

**dialog.pack();**

**dialog.setVisible(true);**

**System.exit(0);**

**}**

**}**

Class Channel

**package main;**

**import java.util.LinkedList;**

**import java.util.Queue;**

**public class Channel {**

**private Queue<Package> packages;**

**private int res\_help;**

**private int speed;**

**private int counter;**

**private boolean resourceAdded;**

**public Channel(int res\_help) {**

**this.res\_help = res\_help;**

**this.speed = 5;**

**this.counter = this.speed;**

**this.packages = new LinkedList<Package>();**

**}**

**public void addPackage(Package newPackage) {**

**if (this.packages.isEmpty()) {**

**this.counter = this.speed;**

**}**

**this.packages.add(newPackage);**

**}**

**public Package getPackage() {**

**if (this.counter < 1) {**

**this.counter = this.speed;**

**this.counter -= 1;**

**return this.packages.poll();**

**}**

**this.counter -= 1;**

**return null;**

**}**

**public void updatePackagesTime()**

**{**

**for (Package pack : this.packages) {**

**pack.tick();**

**}**

**}**

**public void addResource() {**

**if (this.resourceAdded) {**

**return;**

**}**

**this.resourceAdded = true;**

**this.speed = this.speed - this.res\_help;**

**if (this.counter > 0) {**

**this.counter -= this.res\_help;**

**}**

**}**

**public void deleteResource() {**

**this.speed = 5;**

**this.resourceAdded = false;**

**}**

**}**

Class Decoder

**package main;**

**public class Decoder {**

**public int all;**

**public int fault;**

**public int success;**

**public Decoder() {**

**this.all = 0;**

**this.fault = 0;**

**this.success = 0;**

**}**

**public void addPackage(Package pack) {**

**if (pack.getTime() > 10) {**

**this.fault += 1;**

**}**

**this.all += 1;**

**this.success = this.all - this.fault;**

**}**

**public int getGetDeletedRatio() {**

**if (this.fault > 0) {**

**int result = (this.fault \* 100) / this.all;**

**return result;**

**}**

**return 0;**

**}**

**}**

Class Generator

**package main;**

**import java.util.Random;**

**public class Generator {**

**private Random random;**

**private int timer;**

**private int all;**

**private int bound;**

**private int upperbound;**

**private int lowerbound;**

**public Generator(int base) {**

**this.random = new Random();**

**this.bound = base;**

**this.upperbound = this.bound + 3 + 1;**

**this.lowerbound = this.bound - 3;**

**this.timer = 1;**

**this.all = 0;**

**}**

**public Package getPackage() {**

**this.timer -= 1;**

**if (this.timer <= 0) {**

**this.timer = this.random.nextInt(upperbound-lowerbound) + lowerbound;**

**this.all += 1;**

**return new Package();**

**} else {**

**return null;**

**}**

**}**

**public int all() {**

**return this.all;**

**}**

**}**

Class Package

**public class** Package {  
  
 **private int time**;  
  
 **public** Package() {  
 **this**.**time** = 0;  
 }  
  
 **public void** tick() {  
 **this**.**time** += 1;  
 }  
  
 **public int** getTime() {  
 **return this**.**time**;  
 }  
}

Class Resource

**public class** Resource {  
 **private int called**;  
  
 **public** Resource() {  
 **this**.**called** = 0;  
 }  
 **public void** call()  
 {  
 **this**.**called** += 1;  
 }  
 **public int** getCalled()  
 {  
 **return this**.**called**;  
 }  
}