Белорусский государственный университет информатики и радиоэлектроники

# Кафедра ПОИТ

### Отчет по лабораторной работе № 4

«Аналитическое моделирование непрерывно-стохастической СМО и построение её имитационной модели»

Выполнил:

студент группы 05050x

ololo

Проверил:

Огородник Р.В.

Минск 2013

1. **Задание.**

Двухфазная СМО с отказами

n1 n2

λ μ1=5 μ2=5

Варианты:

а) Построить зависимости Ротк, Ротк1, Ротк2 при изменении λ от 1 до 6 с шагом 0.5. Входной поток и потоки обслуживаний – простейшие, n1 = 2, n2 = 2.

б) Построить зависимости Ротк, Ротк1, Ротк2 при изменении n1 от 1 до 6. Входной поток и потоки обслуживаний - простейшие, n2 = 2, λ = 4.5.

1. **Вывод формул для аналитической модели.**

Каждая из фаз данной двухфазной СМО является одноканальной СМО с фиксированным числом мест ожидания.

Входной поток заявок простейший с интенсивностью λ, поток обслуживаний простейший с интенсивностью μ, количество мест ожидания n. Заявка, заставшая очередь полностью заполненной, теряется. Максимальное количество заявок, присутствующих в фазе – n+1 (n заявок в очереди и одна заявка в канале).

Диаграмма интенсивности переходов для каждой фазы:



Воспользовавшись правилом равенства встречных потоков вероятностей через сечение диаграммы и, введя обозначения P0= p и λ/μ = ω, получим:

λ P0 = μ P1, P1 = (λ/μ)P0 = ω p;

λ P1 = μ P2, P2 = (λ/μ)P1 = ω2 p;

λ P2 = μ P3, P3 = (λ/μ)P2 = ω3 p;

…

По индукции получаем Pi = ωi p.

Для определения значения P0 = p воспользуемся нормировочным уравнением:

Учитывая, что полученная сумма представляет собой сумму геометрической прогрессии, в которой n+2 слагаемых, получим:

Окончательно:

.

Если , то

**Вероятность отказа для 1-й фазы:**

,

где .

Интенсивность входного потока 2-й фазы есть абсолютная пропускная способность 1-й фазы:

*.*

**Вероятность отказа для 2-й фазы:**

,

где , *.*

**Вероятность отказа для СМО:**

.

1. **Выполнение задания варианта А.**

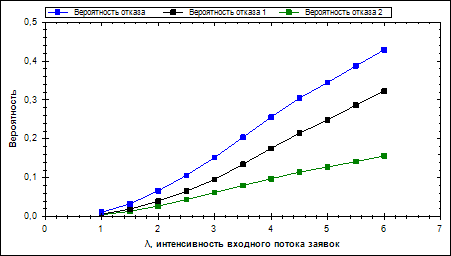
Зависимости Ротк, Ротк1, Ротк2 при изменении λ от 1 до 6 с шагом 0.5. Входной поток и потоки обслуживаний – простейшие, n1 = 2, n2 = 2.

***Аналитическая модель***



λ

***Имитационная модель***



Для сравнения при :

* в аналитической модели: Ротк1 = 0,25, Ротк2 = 0,15429, Ротк = 0,36571;
* в имитационной модели: Ротк1 = 0,25023, Ротк2 = 0,1299, Ротк = 0,3482;

Различие Ротк2 и Ротк в различных моделях объясняется тем, что очередь к 1-й фазе ограничена (n1 = 2). Таким образом выходной поток этой фазы не является простейшим. Но в аналитической модели входной поток 2-й фазы предполагается простейшим.

При увеличении мест ожидания выходной поток 1-й фазы будет приближаться к простейшему и результаты аналитического и имитационного моделирования для Ротк2 и Ротк будут почти одинаковые.

1. **Выполнение задания варианта Б.**

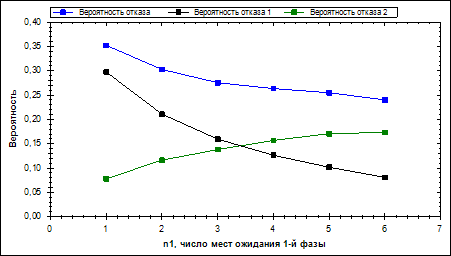
Зависимости Ротк, Ротк1, Ротк2 при изменении n1 от 1 до 6. Входной поток и потоки обслуживаний - простейшие, n2 = 2, λ = 4.5.

***Аналитическая модель***



n1

***Имитационная модель***



В аналитической модели:

* при n1 = 1: Ротк1 = 0,29889, Ротк2 = 0,11017, Ротк = 0,37614;
* при n1 = 2: Ротк1 = 0,21198, Ротк2 = 0,13866, Ротк = 0,32141;
* при n1 = 6: Ротк1 = 0,08398, Ротк2 = 0,18285, Ротк = 0,25147.

В имитационной модели:

* при n1 = 1: Ротк1 = 0,299075, Ротк2 = 0,0767, Ротк = 0,352862;
* при n1 = 2: Ротк1 = 0,211191, Ротк2 = 0,1141, Ротк = 0,3012;
* при n1 = 6: Ротк1 = 0,083685, Ротк2 = 0,17672, Ротк = 0,245616.

Как видно из приведённых результатов, при увеличении количества мест ожидания в первой фазе значения Ротк2 и Ротк в аналитической модели приближаются к значениям имитационной модели. Это происходит, потому что выходной поток 1-й фазы становится более похожим на простейший.

1. **Листинг программы.**

class Phase

{

private readonly double \_μ;

private readonly int \_maxQueueCount;

public int DenialCount { get; private set; } // Счётчик отказов

public Phase(double serviceRate, int maxQueueCount)

{

\_μ = serviceRate; // Интенсивность обработки заявок канала фазы

\_maxQueueCount = maxQueueCount; // Количество мест ожидания фазы

}

public List<double> Imitate(List<double> inputStream)

{

Random rnd = new Random(RandomProvider.Next() ^ Environment.TickCount);

double currentTime = 0; // // Время завершения последней обработки

int queueCount = 0;

int denialCount = 0;

List<double> outputStream = new List<double>();

int currentRequest = 0;

while (currentRequest < inputStream.Count)

{

if (inputStream[currentRequest] < currentTime)

{

if (queueCount == \_maxQueueCount)

denialCount++;

else queueCount++;

currentRequest++;

}

else

{

if (queueCount > 0)

{

currentTime += -Math.Log(rnd.NextDouble())/\_μ;

queueCount--;

}

else currentTime = inputStream[currentRequest++] +

-Math.Log(rnd.NextDouble())/\_μ;

outputStream.Add(currentTime);

}

}

DenialCount = denialCount;

return outputStream;

}

}

class QueuingSystem

{

private readonly double \_λ; // Интенсивность входного потока заявок

private Phase \_phase1, \_phase2;

// Счётчик отказов фазы 1

public int DenialCount1 { get { return \_phase1.DenialCount; } }

// Счётчик отказов фазы 2

public int DenialCount2 { get { return \_phase2.DenialCount; } }

// Счётчик отказов

public int DenialCount { get { return DenialCount1 + DenialCount2; } }

public int RequestsNumber { get; private set; } // Количество заявок

// Количество заявок, попавшее во 2-ю фазу

public int RequestsNumber2 { get; private set; }

public double DenialProbability1 { get { return (double)\_phase1.DenialCount / RequestsNumber; } } // Вероятность отказа фазы 1

public double DenialProbability2 { get { return (double)\_phase2.DenialCount / RequestsNumber2; } } // Вероятность отказа фазы 2

public double DenialProbability { get { return (double)(\_phase1.DenialCount + \_phase2.DenialCount) / RequestsNumber; } } // Вероятность отказа

public QueuingSystem(double λ, double μ1, double μ2, int maxQueueCount1, int maxQueueCount2)

{

\_phase1 = new Phase(μ1, maxQueueCount1);

\_phase2 = new Phase(μ2, maxQueueCount2);

\_λ = λ;

}

public void Imitate(int requestsNumber = 100000)

{

RequestsNumber = requestsNumber;

Random rnd = new Random(RandomProvider.Next() ^ Environment.TickCount);

// Генерация входящего потока

List<double> requestStream = new List<double>();

double currentTime = 0;

requestStream.Add(0);

for (int i = 1; i < requestsNumber; i++)

{

currentTime += -Math.Log(rnd.NextDouble()) / \_λ;

requestStream.Add(currentTime);

}

// 1-я фаза

// Формируем выходной поток / входной поток для 2-й фазы

List<double> requestStream2 = \_phase1.Imitate(requestStream);

// 2-я фаза

\_phase2.Imitate(requestStream2);

RequestsNumber2 = requestStream2.Count;

}

}