ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАНИЕ

Г. В. ОВЕЧКИН, П. В. ОВЕЧКИН

КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ

ACADEMIA

Москва
Издательский центр «Академия»
2014

УДК 004.94(075.32)
ББК 32.972вбя723
О-314

Рецензент —
старший научный сотрудник ГНУ ГОСНИТИ Россельхозакадемии
А. А. Соломашкин

Овечкин Г. В.

О-314 Компьютерное моделирование : учеб. для студ. учреждений сред. проф. образования / Г. В. Овечкин, П. В. Овечкин. — М. : Издательский центр «Академия», 2014. — 224 с.

ISBN 978-5-4468-1492-3

Учебник создан в соответствии с Федеральным государственным образовательным стандартом среднего профессионального образования по специальностям 230111 «Компьютерные сети» (ОП.11), 230401 «Информационные системы (по отраслям)» (ОП.09), 220703 «Автоматизация технологических процессов и производств (по отраслям)» (ЕН.02) (везде для углубленной подготовки).

Рассмотрены теория и практика построения компьютерных моделей. Подробно изложены вопросы построения генераторов случайных величин с заданными законами распределения, приведены рекомендации и методики проверки их качества. На конкретных примерах показаны основные приемы моделирования с помощью метода статистических испытаний. Подробно изложены вопросы моделирования систем массового обслуживания, построения моделей с помощью инструментальных средств моделирования Pilgrim. Рассмотрены основы стратегического и тактического планирования машинного эксперимента.

Для учащихся учреждений среднего профессионального образования, обучающихся по специальности 230111 «Компьютерные сети», а также для всех, желающих самостоятельно ознакомиться с вопросами построения компьютерных моделей систем.

УДК 004.94(075.32)
ББК 32.972вбя723

Оригинал-макет данного издания является собственностью Издательского центра «Академия», и его воспроизведение любым способом без согласия правообладателя запрещается

© Овечкин Г. В., Овечкин П. В., 2014
© Образовательно-издательский центр «Академия», 2014
© Оформление. Издательский центр «Академия», 2014

ISBN 978-5-4468-1492-3

Уважаемый читатель!

Данный учебник является частью учебно-методического комплекта для специальностей 230111 «Компьютерные сети» (ОП. 11.), 230401 «Информационные системы (по отраслям)» (ОП.09), 220703 «Автоматизация технологических процессов и производств (по отраслям)» (ЕН.02) (везде для углубленной подготовки).

Учебник предназначен для изучения общепрофессиональной дисциплины «Компьютерное моделирование» для специальностей 230111, 230401 и математического и общего естественно-научного цикла специальности 220703 той же дисциплины.

Учебно-методические комплекты нового поколения включают традиционные и инновационные учебные материалы, позволяющие обеспечить изучение общеобразовательных и общепрофессиональных дисциплин и профессиональных модулей. Каждый комплект содержит в себе учебники и учебные пособия, средства обучения и контроля, необходимые для освоения общих и профессиональных компетенций, в том числе с учетом требований работодателя.

Учебные издания дополняются электронными образовательными ресурсами. Электронные ресурсы содержат теоретические и практические модули с интерактивными упражнениями и тренажерами, мультимедийные объекты, ссылки на дополнительные материалы и ресурсы в Интернете. В них включен терминологический словарь и электронный журнал, в котором фиксируются основные параметры учебного процесса: время работы, результат выполнения контрольных и практических заданий. Электронные ресурсы легко встраиваются в учебный процесс и могут быть адаптированы к различным учебным программам.

Предисловие

Моделирование используется практически во всех сферах человеческой деятельности. Оно может применяться для того, чтобы лучше понять принципы работы какого-либо объекта, научиться им управлять, понять, как он будет работать в новых условиях, оптимизировать работу объекта и т. п. При этом моделирование позволяет проводить исследование объектов, реальные эксперименты над которыми по ряду причин (дорого, опасно для здоровья, очень большие или маленькие объекты, быстрые или медленные процессы) оказываются невозможными. Кроме того, моделирование позволяет изучать объекты, которые даже не существуют. Например, моделирование широко используется при конструировании новых летательных аппаратов.

Из всех видов моделирования нас будет интересовать имитационное моделирование, которое является едва ли не самым популярным средством, используемым для решения перечисленных задач. При имитационном моделировании для исследования системы используются программы, которые воссоздают в модельном времени динамику функционирования исследуемой системы. И написание таких программ требует от разработчика определенных знаний и умений.

Предлагаемый читателю учебник предназначен для изучения дисциплины «Компьютерное моделирование» учащимися специальностей 230111, 230401 и 220703, образовательный стандарт среднего профессионального образования которой предусматривает, что в результате изучения данной дисциплины обучаемый должен:

- *знать*: основные принципы построения математических моделей; основные типы математических моделей, используемых при описании сложных систем и при принятии решений; классификацию моделей, систем, задач и методов; методику проведения компьютерного эксперимента; методы исследования математических моделей разных типов;
- *уметь*: применять математические методы и вычислительные алгоритмы для решения практических задач; использовать инструментальные средства для построения и исследования математических моделей;
- *владеть*: навыками математического моделирования.

В связи с указанными требованиями стандарта дисциплины предлагается следующая структура учебника.

В *первой главе* изложены основные понятия математического моделирования, представлена классификация математических моделей, рассмотрены основы компьютерного и имитационного моделирования, перечислены типовые этапы, выполняемые при компьютерном моделировании систем.

Вторая глава посвящена рассмотрению вопросов, связанных с построением базовых генераторов псевдослучайных чисел, используемых при компьютерном моделировании. Основное внимание уделено арифметическим генераторам, обладающим рядом преимуществ при моделировании по сравнению с физическими и табличными генераторами.

Третья глава содержит описание методов, используемых при проверке качества генераторов случайных чисел. Рассмотрены критерии как проверки равномерности, так и независимости формируемых чисел. Описаны как графические, так и статистические тесты для проверки равномерности и независимости формируемых псевдослучайных чисел.

В *четвертой главе* изложены основные методы генерации непрерывных случайных величин с заданным законом распределения. Внимание уделено как универсальным методам генерации (при этом рассмотрены точные и приближенные методы), так и частным методам генерации случайных величин с наиболее часто используемыми законами распределения. Также изложены вопросы моделирования полной группы несовместных событий и сложных независимых и зависимых событий.

В *пятой главе* рассмотрены способы построения генераторов для дискретных случайных величин с произвольным и частными законами распределения. Уделено также внимание моделированию простых и сложных событий.

В *шестой главе* обсуждается моделирование систем с помощью метода статистических испытаний (метод Монте-Карло), а также представлен ряд примеров его использования. Рассмотрены классические модели одномерных и двумерных случайных блужданий.

Седьмая глава посвящена моделированию систем массового обслуживания. В ней представлена классификация подобных систем, рассмотрены их основные характеристики, а также даны методы

получения характеристик с помощью как аналитических расчетов, так и компьютерного моделирования.

В *восьмой главе* рассмотрены вопросы планирования компьютерного эксперимента. Основное внимание уделено тактическому планированию эксперимента, в рамках которого решаются вопросы обеспечения заданной точности и достоверности результатов моделирования.

В *девятой главе* представлены основы моделирования систем массового обслуживания с помощью инструментальных средств имитационного моделирования Pilgrim. Рассмотрены основные объекты системы и даны примеры их использования.

Десятая глава содержит девять лабораторных работ, снабженных индивидуальными вариантами заданий, которые рекомендуется выполнить учащимся, изучающим дисциплину «Компьютерное моделирование».

Следует отметить, что для упрощения изучения материала большая часть рассмотренных в учебнике методов и алгоритмов снабжена примерами их реализации на современном языке программирования C#. Кроме того, в конце каждой главы представлен список контрольных вопросов, которые можно использовать для оценки уровня усвоения материала.

Глава 1

ОБЩИЕ ПОНЯТИЯ КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ

1.1. ПОНЯТИЕ МОДЕЛИ И МОДЕЛИРОВАНИЯ

Перед человечеством постоянно возникает необходимость исследования объектов и систем в окружающем его мире. При этом могут преследоваться самые разные цели: лучше понять принципы работы систем, усовершенствовать их, научиться управлять системами, прогнозировать их поведение в заданных условиях и т. п. Одним из наиболее часто используемых в процессе исследования подходов является моделирование.

Дадим определения основным понятиям, используемым в учебнике.

Система представляет собой совокупность объектов, функционирующих и взаимодействующих друг с другом для достижения определенной цели [5]. Система может являться элементом другой системы более высокого порядка (надсистема) и включать в себя системы более низкого порядка (подсистемы).

Моделью называется объект любой природы, который создается исследователем с целью получения новых знаний об объекте-оригинале и отражает только существенные, с точки зрения разработчика, свойства оригинала [2].

Любая модель создается для проведения моделирования. Под *моделированием* будем понимать исследование каких-либо явлений, процессов или систем объектов путем построения и изучения их моделей.

Следует отметить, что одному объекту может соответствовать много моделей, поскольку для разных целей исследования наиболее существенные свойства оригинала могут отличаться. Например, в качестве модели человека могут выступать его скульптура, скелет, размер одежды, результаты анализов и т. п. Одна

модель при этом может соответствовать нескольким разным объектам.

Отметим также, что при моделировании важно использовать модель, адекватную исследуемой системе. Это означает, что существенные с точки зрения экспериментатора свойства модели и системы в достаточной для достижения целей исследования степени должны совпадать. Очевидно, что наиболее адекватные результаты будут получены при проведении экспериментов над самой системой. Однако существуют несколько веских причин, по которым в процессе исследования нужно использовать не реальную систему, а ее модель.

Во-первых, реальные системы часто очень сложны и число факторов, которые влияют на ее работу, просто огромно. В процессе проведения экспериментов контролировать и изменять все эти факторы оказывается невозможным. Поэтому исследователю необходимо упростить ситуацию с помощью модели, в результате чего разнообразие этих факторов уменьшается до желаемого уровня и существенно облегчается их контроль и изменение.

Во-вторых, проведение экспериментов над реальной системой во многих случаях оказывается невозможным по ряду причин, например, из-за слишком высокой стоимости эксперимента (экономика страны), его опасности для самой системы и окружающего мира (ядерный реактор), ограниченности современного уровня науки и техники. Также исследуемые системы могут быть очень велики (Солнечная система) или малы (атом) для проведения экспериментов, изучаемые процессы могут протекать слишком быстро (процессы в двигателе внутреннего сгорания) или слишком медленно (геологические процессы), исследуемый объект может уже (древняя цивилизация) или еще (новый летательный аппарат) не существовать. Использование модели позволяет проводить исследование таких систем.

В определении моделирования не указано, какие именно свойства оригинала следует считать существенными и переносить в модель. Общего и исчерпывающего ответа на этот вопрос нет. Выбор наиболее существенных свойств оригинала определяется на основе опыта исследователя, создающего модель, и во многом определяется целью моделирования. Типичными *целями применения моделирования* являются:

1) *исследование оригинала*. В данном случае исследователь занимается изучением сущности объекта или явления;

2) *анализ* («что будет, если ...»). При этом исследователь учится прогнозировать последствия различных воздействий на оригинал;

3) *синтез* («как сделать, чтобы ...»). При этом исследователь учится управлять оригиналом, оказывая на него воздействия;

4) *оптимизация* («как сделать лучше»). В данном случае исследователю необходимо выбрать наилучшее решение в заданных условиях.

1.2. КЛАССИФИКАЦИЯ ВИДОВ МОДЕЛИРОВАНИЯ

Упрощенная классификация различных видов моделирования представлена на рис. 1.1.

При *физическом моделировании* используется сама система или подобная ей, и имеющая ту же или другую физическую природу (например, гидродинамические исследования на уменьшенных моделях кораблей). Физическая модель может быть реализована в уменьшенном или увеличенном масштабе (масштабируемые модели).

Под *математическим моделированием* понимается процесс установления соответствия реальной системе математической модели и исследование этой модели, позволяющее получить характеристики реальной системы. Применение математического моделирования позволяет исследовать объекты, реальные эксперименты над которыми затруднены или невозможны (дорого, опасно для здоровья, однократные процессы, невозможны из-за

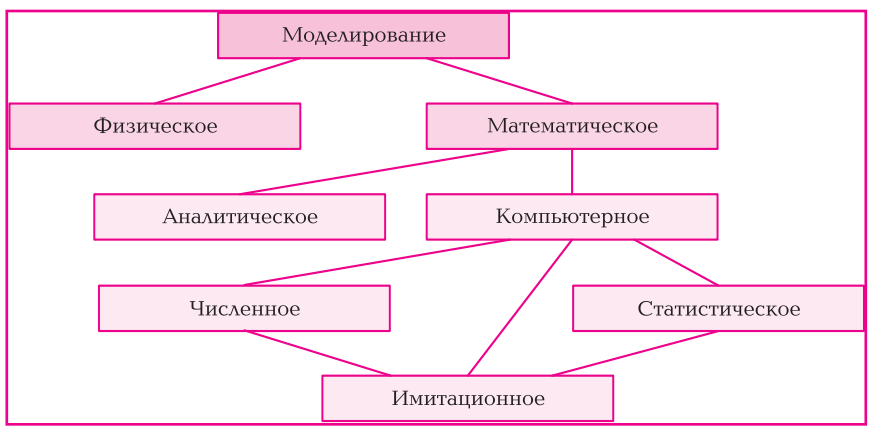


Рис. 1.1. Классификация видов моделирования

физических или временных ограничений — находятся далеко, еще или уже не существуют и т. п.). В зависимости от вида модели математическое моделирование подразделяют на аналитическое и компьютерное.

При *аналитическом моделировании* для представления процессов функционирования элементов системы используются математические соотношения (алгебраические, интегральные, дифференциальные, логические и т. д.). Аналитическая модель может быть исследована следующими методами [12]:

а) *аналитическим*, при котором устанавливаются явные зависимости для искомых характеристик;

б) *численным*, при котором получаются приближенные численные решения для заданных входных данных;

в) *качественным*, когда в явном виде можно найти лишь некоторые свойства решения, например, существование, единственность, характер зависимости.

Примером аналитической модели является модель колебания математического маятника (рис. 1.2).

Движение маятника описывается дифференциальным уравнением

$$ml^2\left[d^2\theta(t)/dt^2\right] + mgl\theta(t) = 0, \qquad (1.1)$$

где m, l — соответственно масса и длина подвеса маятника; g — ускорение свободного падения; $\theta(t)$ — угол отклонения маятника в момент времени t.

Используя данное уравнение, можно получить оценки интересующих исследователя характеристик.

Заметим, что аналитическое решение предпочтительнее всегда, но его обычно удается получить лишь после ряда упрощающих предположений, существенно ухудшающих адекватность модели.

При *компьютерном моделировании* математическая модель формулируется в виде алгоритма или программы для ЭВМ, что позволяет проводить над ней вычислительные эксперименты. Компьютерное моделирование в зависимости от используемых методов достаточно условно можно подразделить на численное, статистическое и имитационное.

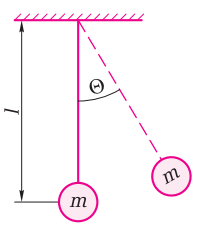


Рис. 1.2. Математический маятник

При *численном моделировании* используются методы вычислительной математики (численное решение уравнений, задающих модель, при различных наборах параметров модели).

При *статистическом моделировании* выполняется обработка данных о системе (модели) с целью получения статистических характеристик системы.

При *имитационном моделировании* процесс функционирования исследуемой системы воспроизводится на ЭВМ при соблюдении логической и временно́й последовательности протекания процессов, что позволяет узнать данные о состоянии системы или отдельных ее элементов в определенные моменты времени.

Именно имитационному моделированию будет посвящена оставшаяся часть учебника. Поэтому рассмотрим его более подробно.

1.3. ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ

В настоящее время имитационное моделирование является одним из наиболее эффективных методов исследования сложных систем, а часто и единственным практически доступным методом получения информации о поведении системы, особенно на этапе ее проектирования. Основное преимущество имитационного моделирования по сравнению с аналитическим моделированием состоит в возможности решения более сложных задач. Имитационные модели позволяют достаточно просто учитывать такие факторы, как наличие дискретных и непрерывных элементов, нелинейные характеристики элементов системы, многочисленные случайные воздействия и многие другие факторы, которые часто создают трудности при аналитических исследованиях.

Применение имитационного моделирования целесообразно при наличии определенных условий. Эти условия определяет Р. Шеннон [13].

1. Не существует законченной математической постановки данной задачи, либо еще не разработаны аналитические методы решения сформулированной математической модели. К этой категории относятся многие модели массового обслуживания, связанные с рассмотрением очередей.

2. Аналитические методы имеются, но математические процедуры столь сложны и трудоемки, что имитационное моделирование дает более простой способ решения задачи.

3. Кроме оценки определенных параметров желательно осуществить на имитационной модели наблюдение за ходом процесса в течение определенного периода.

Имитационное моделирование также можно успешно применять в сфере образования и профессиональной подготовки. Разработка и использование имитационной модели дает возможность обучаемым видеть и разыгрывать на модели реальные процессы и ситуации.

Вместе с тем в процессе применения имитационного моделирования возникает целый ряд проблем [13], которые могут привести к получению недостоверных сведений об исследуемом объекте, поэтому исследователь должен знать эти проблемы и попытаться их решить.

Первая проблема состоит в нахождении компромисса между точностью и сложностью модели. По мнению Р. Шеннона, искусство моделирования в основном состоит в умении находить и отбрасывать факторы, не влияющие или незначительно влияющие на исследуемые характеристики системы. Если модель слишком упрощена и в ней не учтены некоторые существенные факторы, то высока вероятность получить по этой модели ошибочные результаты. Если же модель слишком сложная и в нее включены факторы, оказывающие малое влияние на изучаемую систему, то существенно повышаются затраты на создание такой модели и возрастает риск ошибки в логической структуре модели. Поэтому перед созданием модели необходимо проделать большой объем работы по анализу структуры системы и взаимосвязей между ее элементами, изучению совокупности входных воздействий, тщательной обработке имеющихся статистических данных об исследуемой системе. При выборе наиболее существенных факторов обычно руководствуются принципом Парето, который говорит о том, что 20 % наиболее важных факторов определяют 80 % свойств системы, а оставшиеся 80 % факторов определяют лишь 20 % ее свойств.

Вторая проблема заключается в искусственном воспроизводстве случайных воздействий окружающей среды. Этот вопрос очень важен, поскольку результаты, полученные на модели, неверно воспроизводящей случайные воздействия, с большой вероятностью не будут соответствовать действительности.

В модели для формирования случайности обычно используются так называемые генераторы псевдослучайных чисел, качество которых должно быть тщательно проверено перед использованием модели.

Третьей проблемой является оценка адекватности модели и полученных с ее помощью результатов. Адекватность моделей может быть оценена методом экспертных оценок, сравнением с другими моделями, уже подтвердившими свою достоверность, сравнением полученных результатов с реальными данными по системе.

Следует отметить, что существует много типов имитационных моделей, подходящих для моделирования разнообразных систем. Рассмотрим дихотомическую классификацию имитационных моделей в соответствии с некоторыми признаками. Имитационные модели делятся на следующие виды [12].

1. *Статические и динамические*. Статическая имитационная модель — это система в определенный момент времени (снимок системы) или же система, в которой время просто не играет никакой роли. Примерами статической имитационной модели являются модели, созданные по методу Монте-Карло. Динамическая имитационная модель представляет собой систему, меняющуюся во времени, например систему массового обслуживания (СМО).

2. *Детерминированные и стохастические*. Если имитационная модель не содержит вероятностных (случайных) компонентов, она называется детерминированной. Примером является моделирование движения молекул в газе, когда каждая молекула представляется в виде шарика с определенным направлением и скоростью движения. Взаимодействие двух молекул или молекулы со стенкой сосуда происходит согласно законам абсолютно упругого столкновения и легко описывается алгоритмически. В стохастических имитационных моделях ряд входных компонентов является случайным. Большинство систем массового обслуживания и управления запасами являются стохастическими. Стохастические имитационные модели выдают результат, который является случайным сам по себе, и поэтому он может рассматриваться лишь как оценка истинных характеристик модели.

3. *Непрерывные и дискретные*. В дискретных моделях переменные, характеризующие состояние системы, меняются мгновенно, а в непрерывных — непрерывно во времени. Примером непрерывной модели является модель двигающегося автомобиля, переменные состояния которого (координаты, скорость, направление движения) мгновенно измениться не могут. Примером дискретной модели выступает модель СМО, для которой переменные состояния (число клиентов в очереди, состояние устройств обслуживания) меняются мгновенно во времени.

К хорошим имитационным моделям предъявляются следующие основные требования:

- модели должны быть адекватны соответствующим системам;
- должна обеспечиваться необходимая точность получаемых результатов;
- должно обеспечиваться удобство работы пользователя с моделью;
- модель должна иметь достаточную скорость работы;
- необходимо обеспечить наглядность полученных результатов;
- должна обеспечиваться приемлемая стоимость разработки и использования средств моделирования.

1.4. ОСНОВНЫЕ ЭТАПЫ СОЗДАНИЯ И ИСПОЛЬЗОВАНИЯ КОМПЬЮТЕРНЫХ МОДЕЛЕЙ

Процесс последовательной разработки имитационной модели начинается с создания простой модели, которая затем постепенно усложняется в соответствии с предъявляемыми решаемой проблемой требованиями.

Несмотря на то что универсальной схемы создания и использования модели не существует, можно выделить некоторые типичные этапы данного процесса [5].

1. *Формулировка задачи и планирование исследования системы*. На данном этапе формулируется проблема, стоящая перед исследователем, и определяются основные цели исследования, которые должны быть достигнуты в результате моделирования. При этом также выполняется первоначальное изучение объекта моделирования, обсуждаются ресурсы, имеющиеся в распоряжении экспериментатора, выбираются применяемые средства разработки модели, составляется укрупненный план исследования.

2. *Сбор данных и определение модели*. Исследователем осуществляется построение концептуальной модели системы. При этом выполняется детальное изучение системы, определяются ее границы, устанавливаются необходимые упрощения и приближения, выявляются существенные элементы и свойства системы, устанавливаются единицы измерения, диапазоны изменения, законы распределения, функциональные зависимости переменных и параметров модели. На данном этапе также по возможности собираются данные о рабочих характеристиках реальной системы для последующей проверки адекватности разработанной модели. Концептуальную модель желательно представить в виде известных и

хорошо разработанных типовых систем: массового обслуживания, управления запасами, авторегулирования и т. д.

3. *Определение адекватности концептуальной модели*. Выполняется структурный анализ концептуальной модели с ее предоставлением на рассмотрение аудитории, которая состоит из руководителей, аналитиков, а также экспертов по изучаемой тематике. Анализ проводится до начала программирования, чтобы в дальнейшем избежать перепрограммирования модели. Он помогает убедиться, что допущения, принятые для модели, верны и ничего не упущено. В том случае, если модель оказалась не адекватной системе, исследователь возвращается к этапу 2.

4. *Создание компьютерной программы и ее проверка*. Выполняется программирование и отладка модели. При этом может использоваться либо универсальный язык программирования (C++, C# и т. п.), либо применяться программное обеспечение для моделирования (ReThink, Arena, GPSS, Pilgrim и др.). Преимущество использования языков программирования заключается в том, что они обладают большей гибкостью, позволяя реализовать любые особенности системы, и созданная с их помощью модель требует меньше времени для выполнения. Использование же программного обеспечения моделирования уменьшает время программирования и позволяет снизить вероятность ошибки при реализации модели.

5. *Выполнение предварительных прогонов*. На данном этапе выполняются предварительные прогоны имитационной модели с целью осуществления проверки ее адекватности системе на следующем этапе исследования.

6. *Проверка соответствия программной модели*. Выполняется проверка адекватности программной модели исследуемой системе. При этом если есть существующая система, то можно сравнить характеристики модели и существующей системы при одинаковых исходных данных. Кроме того, результаты моделирования могут быть показаны аналитикам и экспертам по изучаемой теме с целью определения их правильности. Если программная модель не пройдет данную проверку, то исследователю необходимо вернуться к этапу 2 с целью уточнения концептуальной модели системы.

7. *Планирование экспериментов*. Выполняется стратегическое и тактическое планирование компьютерного эксперимента. Основной целью данного этапа является получение такого плана эксперимента, который позволит получить ответы на все поставленные вопросы за минимальное время с требуемой достоверностью. В результате определяются те наборы исходных данных, для

которых нужно получить результаты моделирования, а также определяются параметры моделирования для конкретных наборов исходных данных, обеспечивающие заданные точность и достоверность результатов.

8. *Выполнение рабочих прогонов*. Получение результатов моделирования в соответствии с планом эксперимента, составленным на предыдущем этапе. Данные результаты будут анализироваться на следующем этапе.

9. *Анализ выходных данных*. Выполняется обработка результатов моделирования с целью определения абсолютных характеристик известной конфигурации системы или проведения относительного сравнения альтернативных конфигураций системы. При этом могут использоваться методы регрессионного, дисперсионного анализа, градиентные и другие методы оптимизации. На основании полученных результатов делаются выводы, достаточные для принятия решений по поставленным в начале исследования вопросам.

10. *Документальное представление и использование результатов*. Документальное оформление допущений компьютерной программы и результатов исследования для использования в текущем и будущих проектах. Выполняется представление результатов исследования для их обсуждения с заказчиками исследования, а также использование результатов в процессе принятия решений, если они действительны и достоверны.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое модель и моделирование? В каких случаях они применяются?
2. Какие виды моделирования вы знаете?
3. Каковы основные преимущества использования имитационного моделирования?
4. С какими трудностями сталкивается исследователь в процессе применения имитационного моделирования?
5. Какие требования предъявляются к хорошим имитационным моделям?
6. Какие виды имитационных моделей вам известны?
7. Каковы типичные этапы исследования систем с помощью имитационного моделирования?

Глава 2

ПРОГРАММИРОВАНИЕ БАЗОВЫХ ГЕНЕРАТОРОВ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ

2.1. ОБЩИЕ СВЕДЕНИЯ

При имитационном моделировании стохастических систем выполняется реализация алгоритмов, описывающих процесс функционирования системы во времени. При этом воспроизводятся элементарные явления, происходящие в системе, с учетом их логической и временно́й последовательности. Для моделирования таких явлений, обычно носящих случайный характер, используются случайные числа. Следовательно, при имитационном моделировании необходимо уметь генерировать случайные величины с требуемыми параметрами и характеристиками.

Большинство системных программных средств общего пользования имеют в своем составе специальные стандартные подпрограммы для генерации случайных величин. Однако во многих случаях оказывается, что нужные пользователю стандартные подпрограммы либо вообще отсутствуют, либо не отвечают поставленным требованиям, поэтому необходимо уметь самостоятельно разрабатывать и проверять генераторы случайных величин.

Базовым генератором для получения последовательностей случайных величин с любым заданным распределением является генератор равномерно распределенной случайной величины на интервале [0; 1). Такая непрерывная случайная величина X имеет функцию распределения

$$F_x(x) = \begin{cases} 0, \text{при } x < 0; \\ x, \text{при } 0 \le x < 1; \\ 1, \text{ при } x \ge 1 \end{cases}$$

и функцию плотности распределения вероятности

$$f_x(x) = \begin{cases} 1, \text{при } 0 \le x < 1; \\ 0, \text{при других } x. \end{cases}$$

Математическое ожидание (матожидание) такой равномерно распределенной случайной величины X составляет $M(X) = 1/2$, дисперсия $D(X) = 1/12$. Графики функции плотности вероятности $f_x(x)$ и функции распределения $F_x(x)$ изображены на рис. 2.1. В дальнейшем равномерно распределенные на интервале [0; 1) случайные величины будем называть *случайными числами*.

Равномерное распределение на интервале [0; 1) очень важно, поскольку случайные величины из всех других распределений могут быть получены путем преобразования независимых случайных чисел некоторым способом, определяемым нужным распределением. Некоторые из таких преобразований будут рассмотрены в гл. 4.

Для генерации случайных чисел может применяться один из следующих способов.

1. *Аппаратный*. В основе работы аппаратного генератора лежит какой-либо физический эффект (например, шумы в электронных устройствах или другие явления). Достоинством такого генератора является то, что случайные числа, получаемые с его помощью, являются абсолютно случайными. Также запас чисел аппаратного генератора неограничен. В качестве недостатка аппаратного генератора можно отметить, что данный способ не гарантирует качество получаемой последовательности случайных чисел непосредственно во время моделирования. Кроме того, с помощью аппаратного генератора нельзя получать одинаковые последовательности случайных чисел, которые желательно использовать при отладке

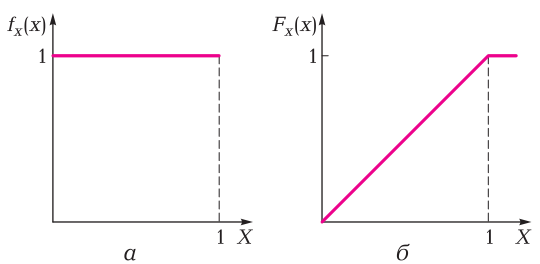


Рис. 2.1. Графики функций:

а — плотности вероятности; *б* — распределения для равномерного распределения на интервале [0; 1)

моделирующих программ и сравнении альтернативных вариантов организации системы. Поэтому при моделировании аппаратные генераторы практически не используются.

2. *Табличный*. При использовании табличного генератора заранее полученные и проверенные случайные числа оформлены в виде таблицы в памяти ЭВМ. Данный способ свободен от отмеченных недостатков аппаратного генератора. Однако при использовании табличного генератора запас случайных чисел ограничен размером таблицы. Кроме того, неэффективно используются ресурсы ЭВМ. Поэтому данный способ при моделировании используется сравнительно редко.

3. *Арифметический* (*численный*). При использовании арифметического генератора каждое новое число определяется одним или несколькими предшествующими числами в соответствии с заданной математической формулой. Данный способ свободен от описанных недостатков ранее рассмотренных генераторов. Поэтому при моделировании чаще всего используются именно арифметические генераторы.

Отметим, что при получении последовательности случайных чисел на ЭВМ с помощью арифметических генераторов используются алгоритмы. Поэтому такие последовательности, являющиеся по сути детерминированными, но для стороннего наблюдателя выглядящие как случайные, называются *псевдослучайными*. Заметим также, что любая ЭВМ оперирует только n-разрядными числами. В связи с этим на ЭВМ вместо непрерывной совокупности равномерных случайных чисел из интервала [0; 1) может получиться лишь дискретная последовательность не более чем 2^n случайных чисел из того же интервала. Поэтому закон распределения такой дискретной последовательности называется *квазиравномерным*.

К идеальному генератору случайных чисел предъявляются следующие основные требования [5]:

- последовательность должна состоять из квазиравномерно распределенных чисел;
- числа должны быть статистически независимыми;
- последовательности случайных чисел должны быть воспроизводимыми;
- последовательности должны быть непериодическими;
- последовательности должны получаться с минимальными затратами вычислительных ресурсов;
- генераторы должны занимать минимальный объем машинной памяти.

Далее рассмотрим основные типы арифметических генераторов случайных чисел. Заметим, что иногда генераторы случайных чисел в тексте учебника называются датчиками.

2.2. АРИФМЕТИЧЕСКИЕ ГЕНЕРАТОРЫ СЛУЧАЙНЫХ ЧИСЕЛ

Известно большое число алгоритмов, процедур и методов получения случайных (псевдослучайных) чисел. В общем случае программные генераторы формируют последовательность чисел по некоторой рекуррентной формуле

$$x_{n+1} = \varphi(x_n, x_{n-1}, ..., x_{n-r}), \qquad (2.1)$$

где φ — совокупность операций, которые необходимо проделать над числами x_n, x_{n-1}, ..., x_{n-r}, чтобы получить x_{n+1}. При этом качество последовательности x_{r+1}, ..., x_n в значительной мере зависит от начальных (исходных) чисел x_0, x_1, ..., x_r.

В результате многократного применения процедуры (2.1) получается последовательность чисел, которая носит детерминированный характер, но в определенных границах она удовлетворяет свойствам равномерного распределения и свойству случайности.

2.2.1. Конгруэнтный (линейный) метод

Наиболее широко применяемые в настоящее время программные генераторы равномерно распределенных чисел в общем виде можно представить с помощью следующей формулы:

$$y_{n+1} = \left(\sum_{i=0}^{r} a_i y_{n-i} + \mu\right) \bmod m, \qquad (2.2)$$

где a_0, a_1, ..., a_r, μ и m, а также числа y_0, y_1, ... являются целыми числами. Образование равномерно распределенных псевдослучайных чисел на интервале [0; 1) реализуется с помощью выражения

$$x_{n+1} = \frac{y_{n+1}}{m}. \qquad (2.3)$$

Формулы (2.2) и (2.3) определяют конгруэнтный метод, частные случаи которого задают тот или иной датчик случайных чисел.

Алгоритм Лемера (мультипликативный датчик). Датчик случайных чисел, основанный на использовании алгоритма Лемера, является частным случаем линейного конгруэнтного метода. Подставляя в формулу (2.2)

$$\mu = a_1 = ... = a_j = 0$$

и полагая, что $a = a_0 > 0$, получим

$$y_{n+1} = a y_n \bmod m. \qquad (2.4)$$

В алгоритме Лемера задаются два соответствующим образом подобранных целых числа: множитель a и модуль m, а также начальное значение y_0. Последовательность случайных чисел вычисляется таким образом.

1. Число y_i известно с предыдущего шага. Вычисляется произведение ay_i.
2. Получается остаток y_{i+1} от деления ay_i на m.
3. Так как y_{i+1} является целым числом из интервала [0; m), то нужно его еще разделить на m, чтобы получить число из интервала [0; 1):

$$x_{i+1} = \frac{y_{i+1}}{m}.$$

Следует отметить, что с помощью формулы (2.4) можно получить всего m различных значений y_i. Поэтому на некотором шаге T обязательно получится y_T, которое уже было получено ранее. А так как следующее значение, формируемое генератором, зависит только от предшествующего, то формируемая далее последовательность начнет повторяться. Количество чисел T, после получения которых последовательность начинает повторяться, называется *периодом генератора*.

Значения y_0, a и m выбираются чаще всего исходя из требования получения максимально возможного периода.

Например, если $m = 2^l$, где $l > 2$ — число двоичных разрядов для задания максимального значения целой константы в ЭВМ (например, l может быть равно 15 или 31). Данный выбор определяется тем, что при таком m путем вычислений легко получить остаток от деления по модулю m. Максимальный период в этом случае достигается при нечетном y_0 и значении a mod 8, равном 3 или 5, что выполняется, когда $a = 5^{2p+1}$, $p = 0, 1, 2, ...$ (или $a = 2^s + 3$, $s = 3, 4, 5, ...$). Значение максимально достижимого периода в данном случае $T_{max} = 2^{l-2} = m/4$.

В том случае, если в качестве m выбирается наибольшее простое число, меньшее 2^l, можно получить генератор с периодом $m - 1$. Такой генератор называется мультипликативным линейным конгруэнтным генератором с простым модулем. Например, если $b = 31$, то $m = 2^{31} - 1 = 2\,147\,483\,647$. Для такого m можно получить период $m - 1$, если a — первообразный элемент по модулю m, т. е. наименьшее целое число l, для которого $a^l - 1$ делится на m, составляет $l = m - 1$.

Заметим, что при построении такого генератора сложно выбирать параметр a. Хорошие свойства имеют генераторы с $m = 2^{31} - 1 = 2\,147\,483\,647$ и $a = a_1 = 16\,807$ или $a = a_2 = 630\,360\,016$, причем генератор с $a = a_2$ лучше [5]. Отметим также, что данный генератор работает несколько медленнее из-за более сложного получения остатка от деления на $m - 1$.

В качестве примера приведем консольную программу на языке C#, реализующую и использующую функцию для генерации случайных чисел. Данная функция реализует мультипликативный генератор с параметрами: $y_0 = 2\,451$, $m = 4\,096$, $a = 5$.

```
namespace Mult_RND
{
   class Program
   {
      // Количество случайных чисел
      private const int N = 1500;
      // Параметр мультипликативного генератора
      private const int A = 5;
      // Модуль
      private static int _Mm;
      // Случайное число
      private static int _Y;
      /// <summary> Генератор случайных чисел
      // </summary>
      /// <returns> Случайное значение </returns>
      private static double Rnd()
      {
         _Y = (A * _Y) % _Mm;
         return (double)_Y / _Mm;
      }
      /// <summary> Основная программа </summary>
      /// <param name = "args"> Аргументы </param>
      static void Main(string[] args)
      {
         _Mm = 4096;
         _Y = 2451;
         // Массив для случайных чисел
         double[] x = new double[N];
         for (int i = 0; i < N; i++)
         {
            // получение случайного числа
            x[i] = Rnd();
            … // обработка случайного числа
         }
         … // обработка массива случайных чисел
      }
   }
}
```


Смешанный генератор. Если в формуле (2.2) положить $a_1 = a_2 = a_3 = \ldots = a_r = 0$ и $a = a_0 > 0$, $\mu > 0$, то получается выражение для смешанного генератора:

$$y_{n+1} = (ay_n + \mu) \bmod m. \qquad (2.5)$$

Целые числа a, μ и m для получения максимального периода генератора, равного m, должны удовлетворять следующим условиям:

1) $a \bmod 8 = 5$;

2) $m / 100 < a < m - \sqrt{m}$;

3) двоичные знаки числа a не должны иметь очевидного шаблона;

4) $\mu / m \approx 1 / 2 - \sqrt{3} / 6$;

5) μ — нечетное;

6) m — целое число, максимальное для данной ЭВМ.

В случае 32-разрядного представления чисел можно использовать следующие значения параметров [5]:

$$a = 16\,070\,093;\ \mu = 453\,816\,693;\ m = 2^{31} = 2\,147\,483\,648.$$

Запишем пример функции, реализующей смешанный генератор:

```
private double Rnd()
{
   _Y = (A * _Y + _Mu) % _Mm;
   return (double)_Y / _Mm;
}
```

Для использования этой функции в основной программе необходимо задать *A*, *_Mu*, *_Mm* и начальное значение *_Y*, например так:

```
…
_A = 165;
_Mu = 3463;
_Mm = 4096 * 4;
_Y = 3887;
double[] x = new double[N];
for (int i = 0; i < N; i++)
{
   x[i] = Rnd();
   …
}
…
```

Аддитивные генераторы. Если в формуле (2.2) положить $a_0 = a_1 = 1$ и $\mu = a_2 = \ldots = a_r = 0$, то полученный генератор называется аддитивным или генератором Фибоначчи:

$$y_{n+1} = (y_n + y_{n-1}) \bmod m. \qquad (2.6)$$

В отличие от ранее рассмотренных генераторов в генераторе Фибоначчи задаются два начальных значения y_0 и y_1.

Для аддитивного генератора можно использовать следующую функцию:

```
private double Rnd()
{
   int y = (_Y0 + _Y1) % _Mm;
   _Y0 = _Y0;
   _Y1 = y;
   return (double)y / _Mm;
}
```

При этом *_Mm* и начальные значения *_Y*0, *_Y*1 должны быть определены в основной программе, например так:

```
…
_Mm = 4096 * 4;
_Y0 = 3971;
_Y1 = 1013;
```



```
double[] x = new double[N];
for (int i = 0; i < N; i++)
{
   x[i] = Rnd();
   …
}
…
```

Обобщенный аддитивный генератор. Генератор вида (2.2), где все $a_i = 1$, $i = 0, 1, \ldots, r$, называется обобщенным аддитивным генератором. Для получения псевдослучайной последовательности чисел $x_1, x_2, \ldots, x_N$, равномерно распределенных на интервале [0; 1), обобщенный аддитивный генератор реализуется в виде следующего алгоритма:

$$x_{n+1} = \{x_n + x_{n-1} + \ldots + x_{n-r}\}. \qquad (2.7)$$

Дробная часть числа ψ

$$\{\psi\} = \psi - \lfloor\psi\rfloor,$$

где $\lfloor\psi\rfloor$ — целая часть ψ.

Начальные значения $x_0, x_1, \ldots, x_r$ выбираются из таблицы случайных чисел. Количество слагаемых в формуле (2.7) может быть любым. В практических случаях ограничиваются $r = 6 \ldots 8$. Формулу (2.7) реализуют следующие основные фрагменты программы:

```
…
// Количество случайных чисел, необходимых для
// получения нового случайного числа
private const int R = 6;
…
private static double Rnd()
{
   double s = 0.0;
   for (int k = 0; k < R; k++)
   {
      s += _Rand[k]; // сумма случайных чисел
   }
   for (int k = 1; k < R; k++)
   {
      // сдвиг случайных чисел
      _Rand[k - 1] = _Rand[k];
   }
```

```
   s -= Math.Truncate(s);
   // новое случайное число
   _Rand[R - 1] = s;
   return s;
}
...
static void Main(string[] args)
{
   // инициализация стандартного датчика
   Random random = new Random();

   for (int i = 0; i < R; i++)
   {
      // генерация первых случайных чисел
      _Rand[i] = random.NextDouble();
   }
   for (int i = 0; i < N_MAX; i++)
   {
      double x = Rnd();
      ...
   }
   ...
}
```

2.2.2. Комбинации генераторов случайных чисел

С целью предотвращения использования при моделировании циклично повторяющихся последовательностей (для увеличения длины периода) широко используют комбинацию генераторов случайных чисел.

Рассмотрим простейший пример такого генератора, использующего два простых генератора. С помощью них получают две последовательности случайных чисел: одну с периодом N_1, другую с периодом N_2. Обозначим через y_{ij} j-й член ($j = 1, 2, ...$) i-й последовательности ($i = 1, 2$) и через z_j — j-й член новой последовательности, образующейся по формуле

$$z_j = (y_{1j} + y_{2j}) \bmod m. \qquad (2.8)$$

Если периоды N_1 и N_2 — взаимнопростые числа, то результирующая последовательность z_j — периодическая с периодом $N = N_1 N_2$.

Процедуру (2.8) можно обобщить путем увеличения числа первоначальных последовательностей. Например, если число первоначальных последовательностей равно r, то j-й член результирующей последовательности получается по формуле

$$z_j = \left(\sum_{i=1}^{r} y_{ij} \right) \bmod m.$$

Еще в качестве одного примера рассмотрим универсальный датчик, в котором для вычисления случайного числа используются три идентичных мультипликативных датчика. Каждый из первоначальных датчиков имеет вид

$$y_{k,n+1} = \left| a_k \left(y_{k,n} \bmod B_k \right) - \frac{c_k y_{k,n}}{B_k} \right|; \qquad (2.9)$$

$$x_{k,n+1} = \frac{y_{k,n+1}}{d_k},$$

где $x_{k,n+1}$ — случайное число, принадлежащее интервалу [0; 1); $k = 1, 2, 3$ — номер первоначального датчика; a_k, b_k, c_k, d_k — целые числа; $y_{k,n}$ — n-е случайное число, генерируемое k-м датчиком и изменяющееся от 0 до $(d_k - 1)$.

Результирующее случайное число вычисляется по формуле

$$z_n = \left\{ x_{1,n} + x_{2,n} + x_{3,n} \right\}.$$

В качестве исходных данных можно взять следующие целые константы:

$$a_1 = 171;\ b_1 = 177;\ c_1 = 2;\ d_1 = 30\,269$$

$$a_1 = 172;\ b_2 = 176;\ c_2 = 35;\ d_2 = 30\,307;$$

$$a_3 = 170;\ b_3 = 63;\ c_3 = 63;\ d_3 = 30\,323.$$

Функция, реализующая универсальный датчик, имеет следующий вид:

```
private double Rnd()
{
   double s = 0.0;
   double[] x = new double[K];
   int[] yN = new int[K];
   for (int i = 0; i < K; i++)
   {
```

```
      yN[i] = (int) Math.Abs(_A[i]*(_Y[i]%_B[i]) –
      _C[i] * (double)_Y[i] / _B[i]);
      _Y[i] = yN[i];
      x[i] = (double) yN[i] / _D[i];
      s += x[i];
   }
   return Math.Abs(s - Math.Truncate(s));
}
```

При этом в основной программе должны быть объявлены и определены вспомогательные массивы и константа K.

```
…
// Количество первоначальных датчиков
private const int K = 3;
// Вспомогательные массивы
private int[] _A = new int[K];
private int[] _B = new int[K];
private int[] _C = new int[K];
private int[] _D = new int[K];
…
```

Метод Макларена — Марсальи основан на использовании двух последовательностей случайных чисел $x_{1,1}, x_{1,2}, \ldots, x_{1,n}, \ldots$ и $x_{2,1}, x_{2,2}, x_{2,1}, x_{2,2}, \ldots, x_{2,n}, \ldots$, получаемых с помощью двух начальных датчиков.

Первоначально с помощью первого генератора вырабатывают последовательность $x_{1,1}, x_{1,2}, \ldots, x_{1,k}$ из k чисел, обычно $k = 64, 128, 256$.

В дальнейшем с помощью первого и второго датчиков генерируют очередные $x_{1,n+k}$ и $x_{2,n}$ числа. Число z_n комбинированной последовательности получают в результате выполнения операций

$$\begin{aligned} M &= [x_{2,n}k] + 1; \\ z_n &= x_{1,M}; \\ x_{1,M} &= x_{1,n+k}. \end{aligned} \qquad (2.10)$$

Операции (2.10) повторяются необходимое число раз. В результате с помощью значений, полученных вторым генератором, реализуется случайный выбор числа из последовательности k чисел $x_{1,1}, x_{1,2}, \ldots, x_{1,k}$, а также ее случайное заполнение новыми числами.

Для реализации метода Макларена — Марсальи необходимы следующие основные фрагменты программы:

```
// Объем выборки
private const int N_MAX = 4000;
// Размер 1-й последовательности случайных чисел
private const int K = 64;
// 1-я случайная последовательность
private static double[] _Z1 = new double[K];
// Стандартный датчик
private static Random _Random = new Random();
…
private static double Rnd()
{
   // случайное число 1-й последовательности
   double g1 = _Random.NextDouble();
   // случайное число 2-й последовательности
   double g2 = _Random.NextDouble();
   int m = (int) (g2 * K);
   double res = _Z1[m];
   // заполнение элемента 1-й последовательности
   // новым числом
   _Z1[m] = g1;
   return res;
}
…
static void Main(string[] args)
{
   for (int i = 0; i < K; i++)
   {
      // генерация первых K случайных чисел
      _Z1[i] = _Random.NextDouble();
   }
   double x;
   for (int i = 0; i < N_MAX; i++)
   {
      x = Rnd();
      …
   }
   …
}
```

2.2.3. Алгоритмы на основе нелинейных формул

Квадратичный конгруэнтный метод. Обобщением линейного конгруэнтного метода является квадратичный конгруэнтный метод формирования случайной последовательности чисел

$$y_{n+1} = (Ay_n^2 + By_n + C) \bmod m, \qquad (2.11)$$

где $m = 2^l$. Если $l \geq 2$, то наибольшее значение периода квадратического конгруэнтного датчика $T_{\max} = 2^l$, что достигается при четном A, нечетном C и если нечетное B удовлетворяет условию $B \bmod 4 = (A+1) \bmod 4$.

Функция, реализующая формулу (2.11), имеет следующий вид:

```
private double Rnd()
{
   _Y = (_A * _Y * _Y + _B * _Y + _C) % _Mm;
   return (double) _Y / _Mm;
}
```

При этом в основной программе должны быть заданы *_A*, *_B*, *_C*, *_Mm* и начальное значение *_Y*, например:

```
…
_A = 6;
_B = 7;
_C = 3;
_Mm = 4096;
_Y = 4001;
for (int i = 0; i < N; i++)
{
   double x = Rnd();
   …
}
…
```

Квадратичный метод Ковэю. При использовании данного метода вычисление очередного числа осуществляется по формуле

$$y_{n+1} = y_n (y_n + 1) \bmod m, \qquad (2.12)$$

где $m = 2^l$, а начальное значение y_0 удовлетворяет равенству $y_0 \bmod 4 = 2$.

При этом длина периода случайной последовательности такого датчика составляет 2^{l-2}.

Функция, реализующая формулу (2.12), имеет следующий вид:

```
private double Rnd()
{
   _Y = _Y * (_Y + 1) % _Mm;
   return (double)_Y / _Mm;
}
```

При этом в основной программе должны быть заданы *_Mm* и *_Y*, например:

```
…
_Mm = 512;
_Y = 2135;
for (int i = 0; i < N; i++)
{
   double x = Rnd();
   …
}
…
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Каким образом задается равномерно распределенная случайная величина?
2. Какие основные способы генерации случайных чисел вы знаете?
3. В чем заключается конгруэнтный метод генерации равномерно распределенных случайных чисел?
4. Какие разновидности конгруэнтного метода генерации вам известны?
5. Что такое период псевдослучайной последовательности? Опишите способы увеличения длины периода.
6. Каковы основные принципы выбора составляющих датчиков случайных чисел для формирования сложного датчика? Сформулируйте их.

Глава 3

МЕТОДЫ ПРОВЕРКИ ГЕНЕРАТОРОВ СЛУЧАЙНЫХ ЧИСЕЛ

3.1. ГИСТОГРАММА ЧАСТОТ И СТАТИСТИЧЕСКАЯ ФУНКЦИЯ РАВНОВЕРОЯТНОГО РАСПРЕДЕЛЕНИЯ

Эффективность статистического моделирования систем на ЭВМ и достоверность получаемых результатов существенным образом зависят от качества базовых последовательностей псевдослучайных чисел. Поэтому прежде чем приступать к реализации моделирующих алгоритмов на ЭВМ, необходимо убедиться в том, что исходная последовательность псевдослучайных чисел удовлетворяет предъявляемым к ней требованиям.

При проверке генераторов можно оценить основные статистические характеристики последовательности псевдослучайных чисел (матожидание, дисперсия и др.), построить гистограмму частот и статистическую функцию распределения и визуально сравнить их с теоретическими функциями.

Для формальной проверки качества случайных чисел дополнительно разработан ряд критериев, в основе которых обычно лежит многократная проверка различных частей выработанных программными генераторами последовательностей случайных величин различного объема N, причем N должно быть довольно большим. Рассмотренные далее критерии применимы для случайных величин с любым законом распределения, однако ниже они используются в процессе проверки качества равновероятных случайных чисел.

Статистическая совокупность случайных величин $x_1,\ldots,x_N$ при большом числе N наблюдений перестает быть удобной формой записи статистического материала. Для придания ему большей компактности строится так называемый статистический ряд. С этой целью весь диапазон возможных значений случайной величины (в случае обсуждаемых генераторов для равномерного закона имеем диапазон от 0 до 1) делят на k равных частей, каждая длиной

$\Delta = 1/k$. В практических случаях выбирают k равным 10...20. Также k можно оценить по формуле Стерджеса

$$k = \lfloor 1 + \log_2(N) \rfloor.$$

Далее по имеющейся выборке $x_1,\ldots,x_N$ подсчитывают количество значений ν_i, приходящихся на i-ю часть. Оценка вероятности $\hat{p}_i$ попадания случайной величины в i-й интервал определяется в соответствии с выражением

$$\hat{p}_i = \frac{\nu_i}{N}.$$

Графическое представление статистического ряда $\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_k$ называется *гистограммой частот*. Очевидно, что сумма частот $\hat{p}_i$ всех интервалов должна быть равна единице.

Примерный вид гистограммы частот для равновероятного распределения показан на рис. 3.1. По внешнему виду гистограммы можно делать предположения о законе распределения, лежащем в основе последовательности чисел $x_1, \ldots, x_N$.

Для удобства сравнения гистограммы частот с плотностью вероятностей можно использовать нормализованную гистограмму частот (рис. 3.2), значения для построения которой определяются по формуле

$$\hat{f}_i = \frac{\nu_i}{\Delta N}.$$

Отметим, что при увеличении числа опытов можно задавать более мелкие интервалы Δ. При этом нормализованная гистограмма приближается к плотности распределения вероятностей $f_x(x)$.

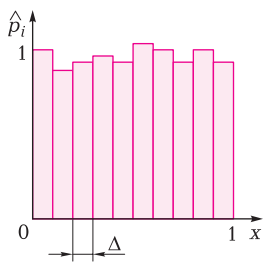


Рис. 3.1. Гистограмма частот

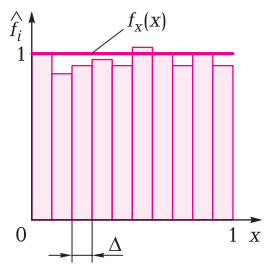


Рис. 3.2. Нормализованная гистограмма частот

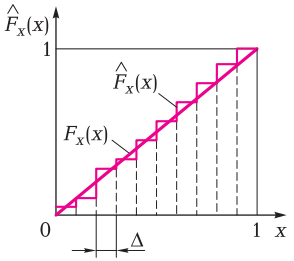


Рис. 3.3. Статистическая функция распределения

Пользуясь данными статистического ряда, можно приближенно построить статистическую (эмпирическую) функцию распределения $\hat{F}_X(x)$ случайной величины X.

Эмпирической функцией распределения называют функцию, определяющую для каждого значения x относительную частоту события $X < x$:

$$\hat{F}_X(x) = \frac{N_X}{N},$$

где N_X — число значений из совокупности $x_1,\ldots,x_N$, меньших x; N — объем выборки.

Различие между эмпирической $\hat{F}_X(x)$ и теоретической $F_X(x)$ функциями состоит в том, что теоретическая функция $F_X(x)$ определяет вероятность события $X < x$, а эмпирическая функция $\hat{F}_X(x)$ — относительную частоту этого же события.

В качестве примера на рис. 3.3 показана типичная статистическая функция для равновероятного распределения.

3.2. СТАТИСТИЧЕСКИЕ ОЦЕНКИ ПАРАМЕТРОВ РАСПРЕДЕЛЕНИЯ

Обработка данных, содержащихся в выборке x_i, $i = \overline{1,N}$, определяет в качестве основных два класса задач: оценка параметров распределения по выборке; проверка статистических гипотез.

Задачи второго класса рассматриваются в следующих подразделах применительно к равновероятному распределению. Задачи первого класса позволяют по известной выборке оценить какую-нибудь числовую характеристику (математическое ожидание, дисперсию, моменты и т.д.) распределения, т.е. указать функцию от выборки

$$\hat{\theta} = w(x_i),$$

предназначенную для использования вместо параметра θ в качестве его приближения. Статистику $\hat{\theta}$ называют оценкой параметра θ.

Например, в качестве несмещенной оценки математического ожидания можно использовать *выборочное среднее*

$$\bar{X}(N) = \frac{1}{N}\sum_{i=1}^{N} x_i.$$

Для оценки дисперсии используется *выборочная дисперсия*

$$S^2(N) = \frac{1}{N}\sum_{i=1}^{N}\left[x_i - \bar{X}(N)\right]^2. \qquad (3.1)$$

Заметим, что использование формулы (3.1) при проведении машинных экспериментов является неудобным, так как для получения оценки $S^2(N)$ требуются две реализации (две выборки), первая из которых служит для получения оценки $\bar{X}(N)$, а вторая — оценки $S^2(N)$. Можно использовать одну и ту же выборку, но обрабатывать ее приходится дважды.

С целью исключения этого недостатка применяют следующую формулу для определения выборочной дисперсии:

$$S^2(N) = \frac{1}{N}\sum_{i=1}^{N} x_i^2 - \bar{X}^2(N),$$

где *второй начальный момент*

$$\hat{\alpha}_2(X) = \frac{1}{N}\sum_{i=1}^{N} x_i^2$$

может быть оценен параллельно с математическим ожиданием.

Начальные моменты более высоких порядков имеют оценки

$$\hat{\alpha}_m(X) = \frac{1}{N}\sum_{i=1}^{N} x_i^m, \quad m = 3, 4, \ldots$$

Заметим, что выборочная дисперсия является смещенной оценкой дисперсии. Для получения несмещенной оценки используется *исправленная выборочная дисперсия*

$$S^2 = \frac{N}{N-1}S^2(N) = \frac{1}{N-1}\sum_{i=1}^{N}\left[x_i - \bar{X}(N)\right]^2.$$

Отметим также, что при достаточно больших объемах выборки выборочная и исправленная выборочная дисперсии отличаются мало. Поэтому на практике исправленную дисперсию используют только при небольших объемах выборки (например, при $N < 30$).

Дополнительную информацию о характере распределения можно получить с помощью асимметрии, которая позволяет оценить симметричность распределения относительно математического ожидания. Асимметрия оценивается как

$$\hat{\nu}(X) = \frac{\sum_{i=1}^{N}[x_i - \bar{X}(N)]^3}{N[S^2]^{3/2}}.$$

Если асимметрия больше нуля, то распределение смещено вправо, если меньше нуля, то влево, если же равно нулю, то распределение симметрично.

Данные оценки можно использовать в случае независимости случайных величин. Однако выходные данные, полученные в результате моделирования, почти всегда являются коррелированными. В этом случае выборочное среднее $\bar{X}(N)$ остается несмещенной оценкой μ, а исправленная выборочная дисперсия S^2 не будет несмещенной оценкой σ^2, поскольку

$$E\left[S^2\right] = \sigma^2\left[1 - 2\frac{\sum_{j=1}^{N-1}(1 - j/N)\rho_j}{N-1}\right],$$

где ρ_j — коэффициент корреляции.

Таким образом, если $\rho_j > 0$ (положительная корреляция), как это часто бывает на практике, исправленная выборочная дисперсия S^2 будет иметь отрицательное смещение $S^2 < \sigma^2$.

Коэффициент корреляции можно оценить так:

$$\hat{\rho}_j = \frac{\hat{C}_j}{S^2},$$

где $\hat{C}_j$ — оценка ковариации, определяемая по формуле

$$\hat{C}_j = \frac{\sum_{i=1}^{N-j}\left[x_i - \bar{X}(N)\right]\left[x_{i+j} - \bar{X}(N)\right]}{n - j}.$$

Отметим, что оценки ρ_j получаются плохими, если N не принимает очень большие значения и значение j мало по отношению к N.

Далее приведены фрагменты программы, используемые для получения данных для гистограммы, статистической функции распределения, а также статистических оценок некоторых параметров распределения на примере мультипликативного генератора псевдослучайных чисел:

```
namespace Mult_RND_test
{
  class Program
  {
  // Количество случайных чисел
  private const int _N = 1500;
  // Параметр мультипликативного датчика
  private static int _A;
  // Модуль
  private static int _Mm;
  // Случайное число
  private static int _Y;
  // Число интервалов разбиения
  private static int _K;
  /// <summary> Генератор случайных чисел
  // </summary>
  /// <returns> Случайное значение </returns>
  private static double Rnd()
  {
    _Y = (_A * _Y) % _Mm;
    return (double)_Y / _Mm;
  }
  /// <summary> формировать набор данных
  // </summary>
  /// <param name = "parValues"> Массив случайных
  /// чисел </param>
  static void GenerateData(out double[]parValues)
  {
    parValues = new double[_N];
    // получение случайных чисел
    for (int i = 0; i < _N; i++)
      parValues [i] = Rnd();
  }
```

```
  /// <summary> Получить массивы оценок для плотно-
  сти и функции
  /// распределения </summary>
  /// <param name = "parValues"> Массив случайных
  /// чисел </param>
  /// <param name = "parDataPlot"> Массив частот < /
  param>
  /// <param name = "parDataFunc"> Массив функции
  /// распределения </param>
  /// <param name = "parMin"> Левая граница интерва-
  ла
  /// </param>
  /// <param name = "parMax"> Правая граница интер-
  вала
/// </param>
private void MakeData(double[] parValues,
out double[] parDataPlot, out double[] parDataFunc,
double parMin, double parMax)
{
double delta = (parMax - parMin) / _K;
parDataPlot = new double[_K];
parDataFunc = new double[_K];
for (int i = 0; i < _N; i++)
{
int j = (int) ((parValues[i] - parMin) / delta);
if (j >= _K) j = _K - 1;
else if (j < 0) j = 0;
parDataPlot[j]++;
}
for (int i = 0; i < _K; i++)
parDataPlot[i] /= _N;
parDataFunc[0] = parDataPlot[0];
for (int i = 1; i < _K; i++)
  parDataFunc[i] = parDataFunc[i - 1] +
  parDataPlot[i];
}
/// <summary> Получить статистические оценки </
summary>
/// < param name = "parValues"> Случайные числа </
param>
/// <param name = "parMx"> Математическое ожидание
/// </param>
```



```
/// <param name = "parS2"> Исправленная выборочная
/// дисперсия </param>
private void Estimate(double[] parValues,
out double parMx, out double parDx)
{
  double m2 = 0;
  parMx = 0;
  for (int i = 0; i < _N; i++)
  {
    parMx += parValues[i];
    m2 += parValues[i] * parValues[i];
  }
  parMx /= _N;
  m2 /= _N;
  parS2 = (m2 - parMx * parMx) * _N / (_N - 1);
}
/// <summary> Основная программа </summary>
/// <param name = "args"> Аргументы </param>
static void Main(string[] args)
{
  double[] values, dataPlot, dataFunc;
  _Mm = 4096;
  _A = 5;
  _Y = _Mm - 5;
  _K = 15;
  GenerateData(out values);
  MakeData(values, out dataPlot,out dataFunc,0.0,1.0);
  … // построение гистограммы частот и
       // статистической функции распределения
  double mx, dx;
  Estimate(values, out mx, out dx);
  … // обработка статистических параметров
    }
  }
}
```

3.3. РАСПРЕДЕЛЕНИЕ НА ПЛОСКОСТИ

Для определения зависимости между элементами выборки $x_1, x_2, \ldots, x_N$ можно использовать тест распределения на плоскости. При проведении данного теста на плоскость наносят точки с

координатами (x_i, x_{i+1}), где $i = 1 \ldots N$. После этого полученный рисунок анализируется. Если на плоскости присутствуют явные «узоры», то последовательность чисел в выборке не является случайной. Если же распределение точек на плоскости хаотично, то между элементами выборки зависимость отсутствует.

Для примера на рис. 3.4 показано распределение на плоскости для генераторов, формирующих зависимые (*а*) и независимые (*б*) псевдослучайные числа.

Заметим, что решение о качестве генератора при использовании рассмотренных способов проверки принимает пользователь. Вследствие этого возможны различия в трактовке результатов. Далее рассмотренные оценочные тесты (критерии согласия) характеризуются тем, что они выдают численную характеристику, которая позволяет однозначно сказать, пройден тест или нет.

Критерий согласия — это статистический критерий для проверки гипотезы, применяемый, чтобы формально оценить, являются ли наблюдения $x_1, x_2, \ldots, x_N$ независимой выборкой из определенного распределения с функцией распределения $F(x)$. Критерий согласия используется для проверки следующей нулевой гипотезы:

H_0: X_i — независимые и одинаково распределенные случайные величины с функцией распределения $F(x)$.

Рассмотрим наиболее часто используемые критерии согласия, такие как критерий «хи-квадрат» (критерий Пирсона) и критерий Колмогорова.

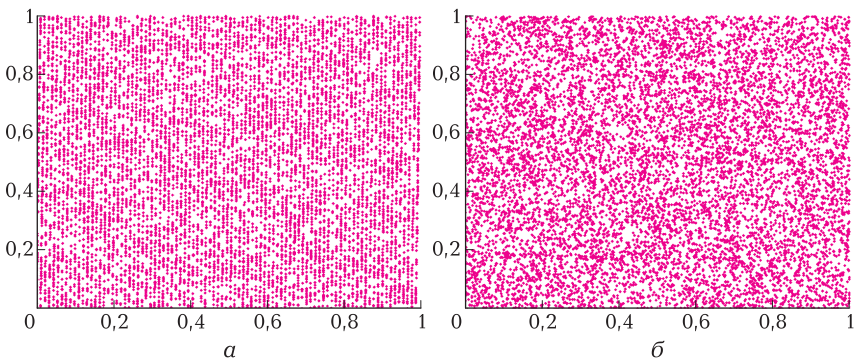


Рис. 3.4. Пример распределения на плоскости для генераторов, формирующих зависимые (*а*) и независимые (*б*) псевдослучайные числа

3.4. КРИТЕРИЙ χ^2 ПИРСОНА

Пусть имеется выборка из N случайных чисел $x_1, x_2, \ldots, x_N$, полученных по результатам машинных экспериментов.

Предположим также, что полученные значения сведены в k разрядов (интервалов) и оформлены в виде статистического ряда:

$$\begin{matrix} (a_1; a_2) & (a_2; a_3) & \cdots & (a_k; a_{k+1}) \\ \hat{p}_1 & \hat{p}_2 & \cdots & \hat{p}_k \end{matrix},$$

где $\hat{p}_j = m_i / N$; m_i — число попаданий случайной величины X в j-й интервал $(a_j; a_{j+1})$

Требуется проверить, действительно ли случайная величина X имеет некоторые плотность $f(x)$ и закон $F(x)$ распределения.

Зная плотность распределения $f(x)$ можно определить ожидаемые вероятности p_i попадания случайной величины в j-й интервал $(i_j = 1 \ldots k)$, если бы осуществлялась выборка из теоретического распределения $f(x)$.

При этом

$$p_j = \int_{a_j}^{a_{j+1}} f(x)\mathrm{d}x = F(a_{j+1}) - F(a_j).$$

В случае равновероятного распределения и одинаковых длинах интервалов

$$p_1 = p_2 = \ldots = p_k = 1/k.$$

Мерой расхождения между теоретическими вероятностями p_i и наблюдаемыми частотами $\hat{p}_j$ считают сумму квадратов их отклонений с некоторыми весами, т. е. величину

$$U = \sum_{j=1}^{k} c_j (\hat{p}_j - p_j)^2.$$

К. Пирсон показал, что если

$$c_j = \frac{N}{p_j},$$

то при больших N закон распределения случайной величины обладает следующими свойствами:

- практически не зависит от вида $f(x)$ и числа опытов N;

- определяется только числом k разрядов и при увеличении N приближается к так называемому распределению χ^2 (хи-квадрат).

При этом мера расхождения между экспериментальной и теоретической плотностями распределения обозначается

$$\chi^2 = N\sum_{j=1}^{k}\frac{(\hat{p}_j - p_j)^2}{p_j} = \sum_{j=1}^{k}\frac{(m_j - Np_j)^2}{Np_j}. \qquad (3.2)$$

Поскольку Np_j — ожидаемое число величин x_i, попадающих в j-й интервал, то статистика χ^2 критерия будет небольшой при хорошем соответствии между теоретическим распределением $f(x)$ и данными. В этом случае гипотеза H_0 принимается. Если же статистика χ^2 имеет слишком большое значение, то гипотеза H_0 отклоняется.

Распределение χ^2 зависит от параметра r, называемого числом «степеней свободы» распределения, равного числу разрядов k минус число независимых условий, накладываемых на частоты $\hat{p}_j$. Отметим, что при проверке равновероятного распределения необходимо учитывать только лишь одно условие:

$$\sum_{j=1}^{k}\hat{p}_j = 1.$$

Это условие — ограничение, накладываемое во всех случаях.

Для критерия «хи-квадрат» составлены специальные таблицы, пользуясь которыми можно для каждого значения числа r степеней свободы и уровня значимости α найти критическое значение $\chi^2_{r,1-\alpha}$. Сокращенный вариант такой таблицы представлен в табл. 3.1.

Схема применения критерия χ^2 к оценке согласованности теоретического и статистического распределений сводится к следующему.

1. Определяется мера расхождения χ^2 по формуле (3.2). Для равновероятного распределения при одинаковых интервалах

$$p_1 = p_2 = \ldots = p_k = 1/k.$$

На практике обычно принимают

$$k = 20 \ldots 50,\ N = (10^2 \ldots 10^3)k.$$

При этом желательно, чтобы в каждый разряд попало не менее 10 ... 20 значений, что обеспечивает достаточную точность построения статистической плотности распределения.

Таблица 3.1. Критические значения $\chi^2_{r,1-\alpha}$ для критерия «хи-квадрат»

r	$1-\alpha$												
	0,990	0,975	0,950	0,900	0,800	0,700	0,600	0,500	0,400	0,300
5	0,55	0,83	1,15	1,61	2,34	3,00	3,66	4,35	5,13	6,06
6	0,87	1,24	1,64	2,20	3,07	3,83	4,57	5,35	6,21	7,23
7	1,24	1,69	2,17	2,83	3,82	4,67	5,49	6,35	7,28	8,38
8	1,65	2,18	2,73	3,49	4,59	5,53	6,42	7,34	8,35	9,52
9	2,09	2,70	3,33	4,17	5,38	6,39	7,36	8,34	9,41	10,66
10	2,56	3,25	3,94	4,87	6,18	7,27	8,30	9,34	10,47	11,78
11	3,05	3,82	4,57	5,58	6,99	8,15	9,24	10,34	11,53	12,90
12	3,57	4,40	5,23	6,30	7,81	9,03	10,18	11,34	12,58	14,01
13	4,11	5,01	5,89	7,04	8,63	9,93	11,13	12,34	13,64	15,12
14	4,66	5,63	6,57	7,79	9,47	10,82	12,08	13,34	14,69	16,22
15	5,23	6,26	7,26	8,55	10,31	11,72	13,03	14,34	15,73	17,32

Окончание табл. 3.1

r	$1-\alpha$									
	0,990	0,975	0,950	0,900	0,800	0,700	0,600	0,500	0,400	0,300
16	5,81	6,91	7,96	9,31	11,15	12,62	13,98	15,34	16,78	18,42
17	6,41	7,56	8,67	10,09	12,00	13,53	14,94	16,34	17,82	19,51
18	7,01	8,23	9,39	10,86	12,86	14,44	15,89	17,34	18,87	20,60
19	7,63	8,91	10,12	11,65	13,72	15,35	16,85	18,34	19,91	21,69
20	8,26	9,59	10,85	12,44	14,58	16,27	17,81	19,34	20,95	22,77
25	11,52	13,12	14,61	16,47	18,94	20,87	22,62	24,34	26,14	28,17
50	29,71	32,36	34,76	37,69	41,45	44,31	46,86	49,33	51,89	54,72
100	70,06	74,22	77,93	82,36	87,95	92,13	95,81	99,33	102,95	106,91

2. Рассчитывается число степеней свободы $r = k - m - 1$, где m — число параметров распределения, определяемых по сгруппированной выборке. В случае проверки равномерности равновероятного распределения $r = k - 1$.

3. Для выбранного уровня α и числа r степеней свободы устанавливается критическое значение $\chi^2_{r,\,1-\alpha}$. Если полученное значение χ^2 превзошло $\chi^2_{r,\,1-\alpha}$, то гипотеза H_0 отвергается, иначе принимается.

Следует особо отметить, что если полученное значение χ^2 много меньше $\chi^2_{r,\,1-\alpha}$, то генератор не удовлетворяет требованию случайного распределения с функцией плотности $f(x)$, так как наблюдаемые частоты $\hat{p}_i$ слишком близки к теоретическим p_i и не могут рассматриваться как случайные.

Основную проблему при применении критерия «хи-квадрат» вызывает выбор числа и размеров интервалов так, чтобы сохранялась высокая мощность критерия для всех возможных распределений и всех объемов выборки.

Обычно размеры интервалов рекомендуется выбирать так, чтобы выполнялось условие $p_1 = p_2 = \ldots = p_k$ (равновероятный подход). Однако в некоторых случаях этого сложно добиться. Приемлемых результатов можно добиться при выполнении описанных далее условий.

Пусть $a = \min\limits_{1 \le j \le k} Np_j$ и пусть $y(5)$ будет числом интервалов, для которых $Np_j < 5$. Тогда желательно выбирать число и длину интервалов так, чтобы выполнялись условия $k \ge 3$, $a \ge 5y(5)/k$.

Фрагмент функции на языке C# вычисления χ^2 для критерия Пирсона выглядит следующим образом:

```
/// <summary> Функция вычисления xi2 для
/// критерия Пирсона </summary>
/// <param name = "hst"> Число попаданий
/// случайной величины в интервалы </param>
/// <param name = "pt"> Теоретическая вероятность
/// попадания случайной величины в интервалы
// </param>
/// <param name = "k"> Число интервалов разбиения
/// </param>
/// <param name = "n"> Объем выборки </param>
/// <returns> Результат вычисления xi2
// </returns> double Xi2(double[] hst, double[] pt,
byte k, long n)
```

```
{
   double xi = 0.0;
   for (int i = 0; i < k; i++)
      xi += Math.Pow(hst[i] - n * pt[i], 2) / (n *
      pt[i]);
   return xi;
}
```

3.5. КРИТЕРИЙ КОЛМОГОРОВА

Недостаток критерия «хи-квадрат», сравнивающего между собой гистограмму данных с плотностью вероятностей, состоит в сложности выбора интервалов.

Критерий Колмогорова, предназначенный для сравнения функции эмпирического распределения с предполагаемой функцией распределения, свободен от указанного недостатка, поскольку для него не требуется группировать данные. Этот критерий достоверен при любом объеме выборки N.

В качестве меры расхождения между теоретическим и статистическим распределениями А. Н. Колмогоров предложил использовать максимальное значение модуля разности между статистической функцией распределения $\hat{F}(x)$ и соответствующей теоретической:

$$D = \max\left|\hat{F}(x) - F(x)\right|.$$

А. Н. Колмогоров показал, что, какова бы ни была функция распределения $F(x)$ непрерывной случайной величины X, при неограниченном возрастании числа опытов N вероятность неравенства

$$D\sqrt{N} \geq \lambda$$

сводится к пределу

$$P(\lambda) = 1 - \sum(-1)^k e^{-2k^2\lambda^2}. \qquad (3.3)$$

Значения вероятности $P(\lambda)$ подсчитанные по формуле (3.3), приведены в табл. 3.2.

Схема применения критерия А. Н. Колмогорова сводится к выполнению следующих этапов.

1. Строятся статистическая функция распределения $\hat{F}(x)$ и теоретическая функция распределения $F(x)$ и определяется максимум D модуля разности между ними, являющийся статистикой критерия Колмогорова (рис. 3.5).

Таблица 3.2. Таблица значений вероятности $P(\lambda)$

λ	$P(\lambda)$	λ	$P(\lambda)$	λ	$P(\lambda)$
0,0	1,000	0,7	0,711	1,4	0,040
0,1	1,000	0,8	0,544	1,5	0,022
0,2	1,000	0,9	0,393	1,6	0,012
0,3	1,000	1,0	0,270	1,7	0,006
0,4	0,997	1,1	0,178	1,8	0,003
0,5	0,964	1,2	0,112	1,9	0,002
0,6	0,864	1,3	0,068	2,0	0,001

Статистическая функция распределения $\hat{F}(x)$ определяется по формуле

$$\hat{F}(x) = \frac{M(x)}{N},$$

где $M(x)$ — число величин X_i из выборки, не превосходящих x. Данная функция может быть легко найдена как $\hat{F}(X_{(i)}) = i/N$ для $i = 1, 2, \ldots, N$, где $X_{(i)}$ — отсортированные в порядке возрастания X_i, т. е. $X_{(1)} < X_{(2)} < \ldots < X_{(N)}$.

Статистику критерия можно определить как

$$D = \max\{D^+, D^-\},$$

где

$$D^+ = \max_{1 \leq i \leq n}\left\{\frac{i}{n} - F(X_{(i)})\right\}, \quad D^- = \max_{1 \leq i \leq n}\left\{F(X_{(i)}) - \frac{i-1}{n}\right\}.$$

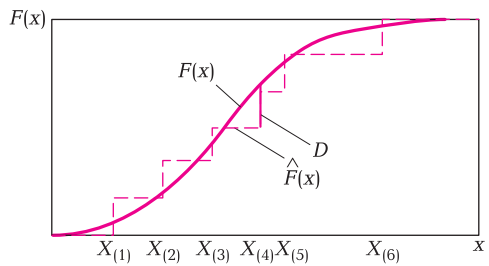


Рис. 3.5. Нахождение максимальной разницы между теоретической и статистической функциями распределения

2. Вычисляется величина $\lambda = D\sqrt{N}$.

3. По табл. 3.2 находится вероятность $P(\lambda)$.

4. Если вероятность $P(\lambda)$ достаточно велика, то считают, что величина X распределена по закону $F(x)$. При малых значениях вероятности следует признать, что распределение величины X далеко от закона распределения $F(x)$.

Фрагмент функции на языке C# вычисления λ для критерия А. Н. Колмогорова выглядит следующим образом:

```
/// <summary> Функция вычисления Lambda для
/// критерия Колмогорова </summary>
/// <param name = "x"> Отсортированная в порядке
/// возрастания выборка </param>
/// <param name = "n"> Объем выборки </param>
/// <returns> Вычисленное значение Lambda
// </returns>
double Lambda(double[] x, int n)
{
   double dMax = 0.0;
   for (int i = 0; i < n; i++)
   {
      double dp = Math.Abs((double) (i + 1) / n -
      Ft(x[i]));
      double dm = Math.Abs(Ft(x[i]) - (double) i/ n);
      // Здесь Ft — теоретическая функция распределения
      if (dp > dMax) dMax = dp;
      if (dm > dMax) dMax = dm;
   }
   return dMax * Math.Sqrt(n);
}
```

3.6. КРИТЕРИЙ СЕРИЙ

Критерий серий используется для проверки независимости случайных чисел.

Пусть $x_1, x_2, ..., x_N$ — последовательность случайных чисел с функцией распределения $F(x)$.

Разделим область этих случайных чисел на два обособленных подмножества S_0 и S_1.

Используя это распределение, построим новую случайную величину Y со значениями 0 и 1:

$$Y = \begin{cases} 0, \text{ если } x \in S_0; \\ 1, \text{ если } x \in S_1. \end{cases} \quad (3.4)$$

Пусть $P(Y = 0) = p$, тогда $P(Y = 1) = 1 - p$.

В результате выполнения преобразования (3.4) получим, например, такую последовательность значений случайной величины Y:

001011101000 ... 0110001.

Серией называется любой отрезок последовательности, состоящий из одинаковых элементов. Значит, в данной последовательности имеются серии

00, 1, 0, 111, 0, 1, 000, ..., 0, 11, 000, 1.

Число R таких серий является случайной величиной с некоторым распределением и числовыми характеристиками:

$$M(R) = 2Np(1-p) + p^2 + (1-p)^2; \quad (3.5)$$

$$D(R) = 4Np(1-p)[1-3p(1-p)] - 2p(1-p)[3-10p(1-p)]. \quad (3.6)$$

На основании теоремы Муавра — Лапласа при больших N распределение числа серий R аппроксимируется нормальным распределением с параметрами, определяемыми формулами (3.5) и (3.6).

Проверка гипотезы о независимости случайных величин x_1, $x_2, ..., x_N$ с надежностью $\beta = 1 - \alpha$ (на уровне значимости α) на основе числа R серий сводится к определению двух критических значений $R_н$ и $R_в$ таким образом, чтобы имело место равенство

$$P(R < R_н) = P(R > R_в) = \frac{\alpha}{2}.$$

Если наблюдаемое число серий меньше $R_н$ или больше $R_в$, то основная гипотеза о независимости случайных чисел $x_1, x_2, ..., x_N$ отклоняется. В противном случае есть все основания считать числа $x_1, x_2, ..., x_N$ независимыми.

Применение теста числа серий (для проверки независимости случайных чисел) сводится к реализации следующих этапов.

1. Из последовательности $x_1, x_2, ..., x_N$ формируется последовательность $y_1, y_2, ..., y_N$ и подсчитывается число R наблюдаемых серий. В качестве разделительного элемента выбирается некоторое значение $p (0 < p < 1)$.

2. По формулам (3.5) и (3.6) определяются математическое ожидание $M(R)$, дисперсия $D(R)$ и среднее квадратическое отклонение $\sigma = \sqrt{D(R)}$.

Таблица 3.3. Значения квантиля стандартного нормального распределения t_β

Надежность β	Уровень значимости α	t_β
0,90	0,10	1,65
0,95	0,05	1,96
0,96	0,04	2,06
0,07	0,03	2,18
0,98	0,02	2,33
0,99	0,01	2,58
0,999	0,001	3,30

3. По табл. 3.3 на уровне значимости α (для надежности $\beta = 1 - \alpha$) находится значение квантиля стандартного нормального распределения t_β.

4. Определяются критические значения

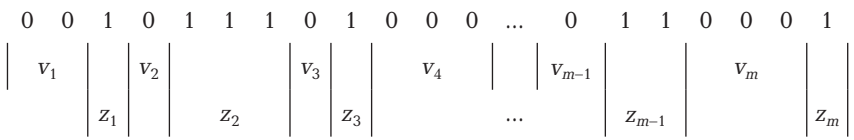
$$R_н = M(R) - t_\beta \sigma(R);\ \ R_в = M(R) + t_\beta \sigma(R).$$

5. Если $R_н \le R \le R_в$, то нет оснований подвергать сомнению гипотезу о независимости случайных чисел $x_1, x_2, ..., x_N$.

Фрагмент программы вычисления R для критерия серий на языке C# выглядит следующим образом:

```
/// <summary> Функция вычисления R для
/// критерия серий </summary>
/// <param name = "x"> массив случайных чисел, сге-
нерированных
/// тестируемым генератором </param>
/// <param name = "n"> Объем выборки </param>
/// <param name = "p"> Разделительный элемент </
param>
/// <returns> Вычисленное значение R </returns>
int getR(double[] x, int n, double p)
{
  bool y1, y2;
  int R = 0;
  // определяем 0-й элемент последовательности y
  if (x[0] < p) y1 = false;
  else y1 = true;
  for (int i = 1; i < n; i++)
  {
    // определяем i-й элемент последовательности y
    if (x[i] < p) y2 = false;
    else y2 = true;
    if (y1 != y2) R++; // увеличиваем число R
    y1 = y2;
  }
  return R;
}
```


Разновидностью критерия серий можно считать тест длины серий. В этом случае в случайной последовательности $y_1, y_2, ..., y_N$ выделяют серии элементов одного вида. Длина V серии «нулей» и длина Z серии «единиц» в последовательности $y_1, y_2, ..., y_N$ являются случайными величинами с некоторыми распределениями $P(V = v)$ и $P(Z = z)$. Для примера представим обработку ранее приведенной последовательности:



В результате такой обработки формируются две случайные последовательности $v_1, v_2, ..., v_m$ и $z_1, z_2, ..., z_n$.

Так как случайные числа 0 и 1 независимы и сформированы из последовательности $x_1, x_2, ..., x_N$, равномерно распределенной на интервале [0; 1), то теоретическая вероятность появления серии нулей длиной v определяется геометрическим распределением (с учетом того, что серия всегда начинается с 0):

$$P(V = v) = (1 - p)^{v-1} p,$$

где p — вероятность появления 1.

Тогда математическое ожидание и дисперсия числа нулей в серии определяются по следующим формулам:

$$M(V) = \frac{1 - p}{p} + 1; \qquad (3.7)$$

$$D(V) = \frac{1 - p}{p^2}. \qquad (3.8)$$

Для оценки среднего числа нулей в серии в процессе обработки последовательности $x_1, x_2, ..., x_N$ необходимо определить число N_0 нулей и число K_0 серий нулей. Тогда оценка среднего числа нулей в серии осуществляется так:

$$M_v = \widehat{M}(V) = \frac{N_0}{K_0}. \qquad (3.9)$$

Применение теста длины серии нулей сводится к реализации следующих этапов.

1. Из последовательности $x_1, x_2, ..., x_N$ формируется последовательность $y_1, y_2, ..., y_N$ и подсчитываются число N_0 нулей и число K_0 серий нулей.

2. По формулам (3.7), (3.8) и (3.9) определяется математическое ожидание $M(V)$, дисперсия $D(V)$ и оценка средней длины M_v серии нулей.

3. По табл. 3.3 на уровне значимости $\alpha = 1 - \beta$ находится t_β.

4. Определяются критические значения

$$V_н = M(V) - t_\beta\sqrt{\frac{D(V)}{K_0}};\ V_в = M(V) + t_\beta\sqrt{\frac{D(V)}{K_0}}.$$

5. Если $V_н \le M_v \le V_в$, то нет оснований подвергать сомнению гипотезу о независимости случайных величин $x_1, x_2, ..., x_N$.

Если используется тест длины серии единиц, то в процессе обработки последовательности $y_1, y_2, ..., y_N$ выделяются серии единиц $z_1, z_2, ..., z_n$ и вычисляются значения

$$M(Z) = \frac{p}{1-p};\ D(Z) = \frac{p}{(1-p)^2};\ M_z = \widehat{M}(Z) = \frac{N_1}{K_1},$$

где N_1 — число единиц в последовательности $y_1, y_2, ..., y_N$; K_1 — число серий единиц.

В результате проведенного анализа гипотеза о независимости исходной случайной последовательности принимается, если

$$M(Z) - t_\beta\sqrt{\frac{D(Z)}{K_1}} \le M_z \le M(Z) + t_\beta\sqrt{\frac{D(Z)}{K_1}}.$$

Далее представлен фрагмент программы определения числа K_0 серий нулей в массиве y, используемый в тесте длины серий нулей.


```
//обнуляем число переходов из 0 в 1
K0 = 0;
//считаем число переходов из 0 в 1
for (int i = 1; i < n; i++)
  if ((y[i - 1] == 0) && (y[i] = = 1)) K0++;
//учитываем последнюю серию, если она есть
if (y[n - 1] == 0) K0++;
```

Критерии, основанные на выделении последовательности серий, базируются на выводах теоремы Муавра — Лапласа, когда полученные в результате обработки случайные значения аппроксимируются нормальным распределением. Аналогичный подход можно использовать и в следующем критерии.

3.7. ПРОВЕРКА РАВНОМЕРНОСТИ ПО КОСВЕННЫМ ПРИЗНАКАМ

Генерируемая последовательность чисел $x_1, x_2, ..., x_N$ разбивается на две последовательности:

$$x_1, x_3, x_5, ...$$

$$x_2, x_4, x_6, ...$$

Затем проводится следующий эксперимент. Если выполняется условие

$$x_{2i-1}^2 + x_{2i}^2 < 1; i = 1, 2, ..., \qquad (3.10)$$

то фиксируется наступление некоторого события C и в счетчик событий добавляется единица. После $N/2$ опытов, когда сгенерировано N чисел, в счетчике будет число $N_C \le N/2$.

Геометрически условие (3.10) означает, что точка находится внутри четверти круга радиусом $R = 1$, что иллюстрируется рис. 3.6.

В общем случае точка (x_{2i-1}, x_{2i}) всегда попадает внутрь единичного квадрата, поэтому при равномерном распределении исходной последовательности при больших N частота

$$\hat{p}_k = \frac{2N_C}{N}$$

сходится к вероятности попадания этой точки в четверть круга

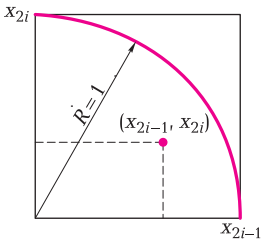


Рис. 3.6. Иллюстрация метода проверки генератора по косвенным признакам

$$p_k = S_{1/4\text{круга}} / S_{\text{квадрата}} = (\pi R^2/4)/R^2 = \pi/4.$$

С уровнем значимости $\alpha = 1 - \beta$ (который определяет значение t_β) гипотезу о равномерном распределении исходной последовательности принимают, если выполняются неравенства

$$\frac{\pi}{4} - t_\beta\sqrt{\frac{\pi(4-\pi)}{8N}} \le \frac{2N_C}{N} \le \frac{\pi}{4} + t_\beta\sqrt{\frac{\pi(4-\pi)}{8N}}.$$

Подпрограмма вычисления числа N_C наступлений события C на языке C# выглядит следующим образом:

```
/// <summary> Функция вычисления Nc для проверки
/// равномерности по косвенным признакам
// </summary>
/// <param name = "x"> массив случайных чисел, сге-
нерированных
/// тестируемым генератором </param>
/// <param name = "n"> Объем выборки </param>
/// <returns> Вычисленное значение R </returns>
int getNc(double[] x, int n)
{
   //определяем количество пар значений в массиве x
   int Nc = 0;
   for (int i = 0; i < n / 2; i++)
      if (Math.Pow(x[2*i],2) + Math.Pow(x[2*i + 1],2)
      < 1) Nc++;
   return Nc;
}
```

3.8. КОМБИНАТОРНЫЕ ТЕСТЫ

Комбинаторные тесты основаны на специальных критериях, в основу которых положены комбинации некоторых событий. Для вновь сформированных случайных величин определяется некоторая функция распределения, которая сравнивается со статистической функцией распределения. Согласие между ними, как правило, исследуется с помощью критерия χ^2.

3.8.1. Покер-тест

Данный тест сконструирован специально для проверки случайности появления цифр в вырабатываемых последовательностях чисел. Пусть $x_1, x_2, ..., x_N$ — последовательность случайных величин с функцией распределения $F(x)$. Разделим область значений случайной величины X на k равновероятных интервалов с помощью точек $a_0 < a_1 < ... < a_{k-1} < a_k$. Если X действительно имеет функцию распределения $F(x)$, то для каждого интервала $[a_{j-1}; a_j)$ имеем

$$P(a_{j-1} \le X < a_j) = \frac{1}{k}.$$

Используя $y_i = i$ для $x_i \in [a_j; a_{j+1})$, $j = 0, 1, ..., k-1$ получаем новую последовательность случайных чисел y_i, $i = 1, 2, ...$, принимающих одно из значений $0, 1, ..., k-1$ с одинаковой вероятностью.

Далее всю последовательность $y_1, y_2, ...$ можно разбить на пятерки $(y_1, y_2, ..., y_5)$ $(y_6, y_7, ..., y_{10}, ...)$.

Возможны семь классов пятерок, отличающихся различным содержанием цифр:

abcde — все цифры в пятерке различны;
aabcd — две цифры одинаковы, остальные различны;
aabbc — две пары цифр одинаковы;
aaabc — три цифры одинаковы, две различны;
aaabb — три цифры одинаковы и две одинаковы;
aaaab — четыре цифры одинаковы;
aaaaa — все цифры одинаковы.

Предполагая, что каждая из цифр $0, 1, ..., k-1$ в последовательности $y_1, y_2, ...$ появляется с одинаковой вероятностью и что отдельные члены этой последовательности независимы, можно определить распределение вероятностей различных пятерок $P(abcde)$, $P(aabcd)$, $P(aabbc)$, $P(aaabc)$, $P(aaabb)$, $P(aaaab)$, $P(aaaaa)$. Значения этих вероятностей указаны в табл. 3.4 для наиболее часто используемых $k = 2$, 8 и 10.

Согласие распределения пятерок различных типов с наблюдаемым распределением проверяется с помощью критерия χ^2.

Таблица 3.4. Вероятности событий для покер-теста

Вероятность	$k = 2$	$k = 8$	$k = 10$
$P(abcde)$	—	0,20518	0,3024
$P(aabcd)$	—	0,51270	0,5040
$P(aabbc)$	—	0,15381	0,1080
$P(aaabc)$	—	0,10254	0,0720
$P(aaabb)$	0,6250	0,01709	0,0090
$P(aaaab)$	0,3125	0,00854	0,0045
$P(aaaaa)$	0,0625	0,00024	0,0001

Для примера далее представлен вариант фрагмента программы для определения вероятностей появления различных классов пятерок при проверке генератора равномерно распределенных чисел.

```
// x — массив случайных чисел, сгенерированных тестируемым генератором
// n — количество случайных чисел в массиве x
// K — количество равновероятных интервалов
// получаем последовательность y
int i, j, m;
int []numN = new int[5];
int []repN = new int[5];
int []y = new int[n];
for (i = 0; i < n; i++) y[i] = (int)(x[i] * K);
// определяем количество пятерок
int N5 = n / 5;
// определяем количество пятерок разного класса
int Nabcde = 0, Naabcd = 0, Naabbc = 0;
int Naaabc = 0, Naaabb = 0, Naaaab = 0, Naaaaa = 0;
for (i = 0; i < N5 - 1; i++)
{
  // обнуляем массив, в котором будем хранить
  // количество повторений j-го числа в пятерке
  for (j = 0; j < 5; j++) numN[j] = 0;
  for (j = 0; j < 5; j++)
    for (m = 0; m < 5; m ++)
      if (y[j + i * 5] == y[m + i * 5]) numN[j]++;
  // обнуляем массив, в котором будем хранить
  // сколько раз встретилось j повторяющихся чисел
  for (j = 0; j < 5; j++) repN[j] = 0;
  for (j = 0; j < 5; j++) repN[numN[j] - 1]++;
  // определяем класс пятерки
  if (repN[0] == 5) Nabcde++;
  else
    if (repN[0] == 3) Naabcd++;
    else
      if (repN[1] == 4) Naabbc++;
      else
        if ((repN[1] == 2) && (repN[2] == 3))
        Naaabb++;
        else
          if (repN[3] == 4) Naaaab++;
          else
            if (repN[4] == 5) Naaaaa++;
            else Naaabc++;
}
// Оцениваем вероятности появления пятерок разных
// классов
double Pabcde = (double)Nabcde / N5;
double Paabcd = (double)Naabcd / N5;
double Paabbc = (double)Naabbc / N5;
double Paaabc = (double)Naaabc / N5;
double Paaabb = (double)Naaabb / N5;
double Paaaab = (double)Naaaab / N5;
double Paaaaa = (double)Naaaaa / N5;
```

3.8.2. Критерий коллекционера

Тест реализует процесс, аналогичный деятельности коллекционера, пополняющего свою коллекцию. При применении этого критерия образуемая так же, как и при использовании покер-теста, последовательность $y_1, y_2, \ldots$ наблюдается так долго, пока в ней появятся все k цифр $0, 1, \ldots, k-1$. Пусть R_1 — длина наблюдаемого отрезка последовательности. После этого для оставшейся части последовательности снова запускается процесс «коллекционирования», в результате которого получается новая длина последо-

вательности R_2. Данные действия продолжаются до тех пор, пока не будет обработана вся последовательность $y_1, y_2, \ldots$ В результате получается последовательность случайных значений R_1, R_2, ... Случайная величина R имеет распределение вероятностей, определяемое как

$$P(R = r) = \frac{1}{k^{r-1}} \sum_{j=0}^{k-2} (-1)^j C_{k-1}^j (k - 1 - j)^{r-1}, \qquad (3.11)$$

где $r = k, k+1, \ldots$; C_n^m — число сочетаний из n элементов по m;

$$C_n^m = \frac{n!}{m!(n-m)!}$$

Число сочетаний из n элементов по m.

Согласие распределения (3.11) с наблюдаемым распределением R_1, R_2, ... исследуется с помощью критерия χ^2.

Приведем фрагмент программы для определения длины R наблюдаемого отрезка последовательности.

```
// x — массив случайных чисел, сгенерированных
// тестируемым генератором
// n — количество случайных чисел в массиве x
// K — количество равновероятных интервалов
int i, y;
// обнуляем текущую длину последовательности
int R = 0;
// создаем и обнуляем массив, в котором будем отме-
//чать
// появляющиеся цифры
int []numN = new int[K];
for (i = 0; i < K; i++) numN[i] = 0;
//обнуляем счетчик числа появившихся чисел
int M = 0;
while ((M != K) && (R < n))
{
  // получаем элемент последовательности y
  y = (int)(x[R] * K);
  // если такого элемента не было
  if (numN[y] == 0)
  { // то отмечаем его в массиве
    numN[y] = 1;
    // и увеличиваем счетчик числа элементов
    M++;
  };
  // переходим к следующему элементу последователь-
  //ности
  R++;
}
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое гистограмма частот? Каковы способы ее построения?
2. Что такое статистическая функция распределения? Каковы способы ее построения?
3. Какие основные числовые характеристики случайных величин вы знаете? Каким образом можно оценить их значение по известной выборке?
4. Что позволяет проверять тест распределения на плоскости? В чем он заключается?
5. Для чего нужны критерии проверки датчиков псевдослучайных чисел?
6. В чем состоит сущность критерия χ^2 Пирсона?
7. При выполнении каких условий возможно применение критерия χ^2 Пирсона?
8. Каким образом определяется число степеней свободы для критерия χ^2?
9. В чем заключается критерий Колмогорова?
10. С помощью какого критерия можно проверить независимость псевдослучайных величин, формируемых датчиком случайных чисел? В чем состоит данный критерий?
11. С помощью каких критериев можно проверить случайность цифр в генерируемой последовательности?

Глава 4

МЕТОДЫ МОДЕЛИРОВАНИЯ СЛУЧАЙНЫХ ВЕЛИЧИН С ЗАДАННЫМ ЗАКОНОМ РАСПРЕДЕЛЕНИЯ

4.1. МЕТОД ОБРАТНЫХ ФУНКЦИЙ

В процессе построения имитационных моделей стохастических систем часто возникает необходимость имитирования случайных процессов или задания случайных воздействий на некоторую систему. Законы распределений этих случайных величин в ряде случаев отличаются от равномерного. Подобные ситуации определяют необходимость программной реализации случайных величин с некоторым заданным законом распределения. При этом обычно существуют несколько альтернативных вариантов генерации случайной величины с нужным распределением, различающихся точностью, эффективностью и сложностью. *Точность* определяет насколько точно распределение генерируемых случайных величин соответствует требуемому. *Эффективность* обусловливается требуемым объемом памяти и быстродействием, а *сложность* — сложностью понимания и реализации метода.

Рассмотрим способы решения задачи моделирования случайной величины с требуемым законом распределения.

Пусть требуется смоделировать непрерывную случайную величину X с плотностью распределения $f(x)$ Задача решается путем моделирования последовательности случайных чисел, равномерно распределенных на интервале [0; 1), и преобразования последовательности случайных чисел r_1, r_2, ..., r_n в последовательность $x_1, x_2, ..., x_n$ с помощью некоторой функции

$$x = \Psi(r), \tag{4.1}$$

связывающей случайные числа с равномерным распределением со случайными числами с заданным распределением. Преобразование (4.1) может быть выполнено различными методами.

Метод обратных функций используется для генерирования случайной величины X, заданной функцией распределения $F(x)$.

Данный метод основан на следующей лемме.

Лемма. Если случайная величина ξ имеет плотность распределения $f(\xi)$, то случайная величина $\gamma = \int_{-\infty}^{\xi} f(x)dx = F(\xi)$ имеет равномерный закон распределения на интервале [0; 1).

Из данной леммы следует, что значение x случайной величины X с функцией распределения $F(x)$ является решением уравнения

$$F(x) = r,$$

где r — значение случайной величины R, равномерно распределенной на интервале [0; 1). Таким образом

$$x = F^{-1}(r). \tag{4.2}$$

Отметим, что в выражении (4.2) верхний индекс «–1» является признаком обратной функции.

Для наглядности получение случайных величин методом обратных функций можно изобразить графически (рис. 4.1). Если по оси ординат брать случайные числа r_i в диапазоне от 0 до 1 и находить для них, как показано стрелкой, соответствующие числа на оси абсцисс, то полученные значения x_i будут являться значениями случайной величины X с функцией распределения $F(x)$.

Таким образом, процедура получения значений случайной величины с заданной функцией распределения $F(x)$ включает в себя два этапа:

1) реализация случайной величины R, равномерно распределенной на интервале [0; 1);

2) вычисление значений случайной величины X по формуле (4.2).

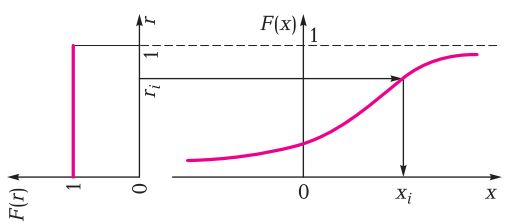


Рис. 4.1. Иллюстрация метода обратных функций

Пример. Случайная величина X имеет равномерное распределение на интервале $[a; b)$. Составить программу моделирования случайной величины X на интервале $[a; b)$.

Решение. Функция распределения случайной величины X имеет вид

$$F(x)=\begin{cases}0, & x<a;\\ \dfrac{x-a}{b-a}, & a\le x<b;\\ 1, & x\ge b.\end{cases}$$

Согласно методу обратных функций

$$F(x)=\frac{x-a}{b-a}=r,$$

откуда

$$x=a+r(b-a).$$

Составим программу моделирования случайных величин на языке C#:

```
static void Main(string[] args)
{
   Console.Write("Задайте левую границу интервала: ");
   double a = double.Parse(Console.ReadLine());
   Console.Write("Задайте правую границу интервала: ");
   double b = double.Parse(Console.ReadLine());
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double r, x;
   for (int i = 0; i < n; i++)
   {
      r = random.NextDouble();
      x = a + r * (b - a);
      Console.WriteLine("x[{0}] = {1:f3}", i + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

Пример. Случайная величина X имеет экспоненциальный (показательный) закон распределения

$$f(x)=\lambda e^{-\lambda x},\ x\ge 0,$$

где λ – параметр распределения, $\lambda>0$.

Составить программу моделирования случайной величины X.

Решение. Функция распределения случайной величины X

$$F(x)=\int_{-\infty}^{x}f(x)\mathrm{d}x=\int_{0}^{x}\lambda e^{-\lambda x}\mathrm{d}x=1-e^{-\lambda x}.$$

Согласно методу обратных функций

$$F(x)=1-e^{-\lambda x}=r.$$

Получим

$$e^{-\lambda x}=1-r;$$

$$-\lambda x=\ln(1-r);$$

$$x=-\frac{1}{\lambda}\ln(1-r).$$

Так как случайная величина $(1-R)$ имеет то же самое распределение, что и R, то при нахождении значений случайной величины X пользуются формулой

$$x=-\frac{1}{\lambda}\ln r.$$

Составим программу моделирования случайных величин на языке C#:

```
static void Main(string[] args)
{
   Console.Write("Задайте параметр распределения Lambda: ");
   double lambda = double.Parse(Console.ReadLine());
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double r, x;
   for (int i = 0; i < n; i++)
   {
      r = random.NextDouble();
      x = -Math.Log(r) / lambda;
      Console.WriteLine("x[{0}] = {1:f3}", i + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

Отметим, что метод обратных функций является универсальным и точным методом генерирования случайной величины X, заданной функцией распределения $F(x)$. Однако он не применим в

случае, если отсутствует конечная форма записи функции распределения. Кроме того, не всегда возможно аналитически решить уравнение (4.2).

4.2. МЕТОД КУСОЧНО-ЛИНЕЙНОЙ АППРОКСИМАЦИИ

Метод кусочно-линейной аппроксимации предложен Н. П. Бусленко и используется в тех случаях, когда одного выражения для обратной функции достичь не удается. Функция распределения $F(x)$ заменяется функцией $F^*(x)$, состоящей из аппроксимированных отрезков прямых (или других подходящих простых функций, для которых можно найти обратную функцию). Число подынтервалов разбиения зависит от требуемой точности конечных результатов. На каждом интервале моделируется случайная величина с равномерным законом распределения (или другим аппроксимирующим простым распределением). Для получения выражения для реализации случайной величины X необходимо решить уравнение $r = F(x)$ относительно переменной X на каждом аппроксимирующем интервале.

Пример. Составить программу моделирования случайной величины X, функция распределения которой $F(x)$ аппроксимирована тремя линейными функциями:

$$F(x) = \begin{cases} F_1(x) = 0{,}5x, & x \in [0;1); \\ F_2(x) = 0{,}3x + 0{,}2, & x \in [1;2); \\ F_3(x) = 0{,}1x + 0{,}6, & x \in [2;4]. \end{cases}$$

Решение. Для каждого подынтервала решим уравнение $r = F(x)$ относительно переменной x. Получим:

$$x = F^{-1}(r) = \begin{cases} 2r, & F_1(0) \le r < F_1(1); \\ \dfrac{r - 0{,}2}{0{,}3}, & F_2(1) \le r < F_2(2); \\ 10r - 6, & F_3(2) \le r \le F_3(4). \end{cases}$$

Схема алгоритма реализации случайной величины приведена на рис. 4.2.

Программа на языке C# имеет следующий вид:

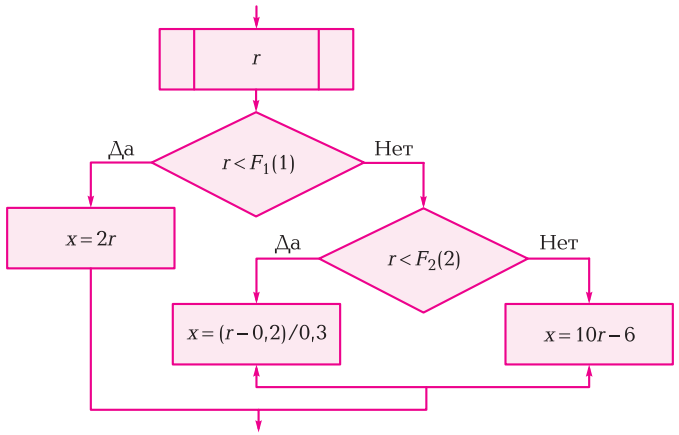


Рис. 4.2. Схема алгоритма генерации случайной величины

```
static void Main(string[] args)
{
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double r, x;
   for (int i = 0; i < n; i++)
   {
      r = random.NextDouble();
      if (r < 0.5) x = 2 * r;
      else if (r < 0.8) x = (r - 0.2) / 0.3;
      else x = 10 * r - 6;
      Console.WriteLine("x[{0}] = {1:f3}", i + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

Пример. Составить программу моделирования случайной величины X, функция распределения которой $F(x)$ аппроксимирована двумя простыми функциями:

$$F(x) = \begin{cases} F_1(x) = x^2, & x \in [0;\ 0{,}7071); \\ F_2(x) = \sqrt{x - 0{,}4571}, & x \in [0{,}7071;\ 1{,}4571]. \end{cases}$$

Решение. Для каждого подынтервала решим уравнение $R = F(x)$ относительно переменной x. Получим:

$$x = F^{-1}(r) = \begin{cases} \sqrt{r}, & F_1(0) \le r < F_1(0{,}7071); \\ r^2 + 0{,}4571, & F_2(0{,}7071) \le r \le F_2(1{,}4571). \end{cases}$$

Программа на языке C# имеет вид:

```
static void Main(string[] args)
{
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double r, x;
   for (int i = 0; i < n; i++)
   {
      r = random.NextDouble();
      x = r < 0.5 ? Math.Sqrt(r) : r * r + 0.4571;
      Console.WriteLine("x[{0}] = {1:f3}", i + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

4.3. МОДЕЛИРОВАНИЕ СЛУЧАЙНОЙ ВЕЛИЧИНЫ ПО ЭМПИРИЧЕСКИМ ДАННЫМ

Метод применяется в случае, когда неизвестно выражение для функции распределения $F(x)$ и имеются статистические данные. Пусть эти данные представлены в виде статистического ряда, т. е. известны интервалы $(x_i; x_{i+1})$, $i = 0 \ldots k-1$, и оценки вероятности $\hat{p}_i$ попадания случайной величины в них.

Для генерации случайной величины по эмпирическим данным реализуется случайная величина R, равномерно распределенная на интервале [0; 1). Если значение r_j попадает в интервал $\left[\sum_{i=0}^{k-1} p_i; \sum_{i=0}^{k} p_i\right)$, то в качестве случайной величины X принимается значение $x_j = \dfrac{x_{k-1} + x_k}{2}$ (рис. 4.3).

Пример. Пусть имеется статистический ряд

$(x_i; x_{i+1})$	(0; 0,1)	(0,1; 0,2)	(0,2; 0,3)	(0,3; 0,4)	(0,4; 0,5)	(0,5; 0,6)
$\hat{p}_i$	0,1	0,3	0,25	0,2	0,1	0,05

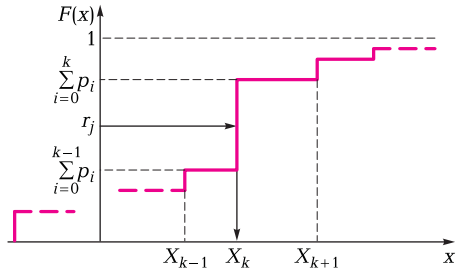


Рис. 4.3. Иллюстрация метода генерации случайной величины по эмпирическим данным

Составить программу моделирования случайной величины X.

Решение. Программа на языке C# имеет следующий вид:

```
static void Main(string[] args)
{
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   double[] xt = new double[]
      {
         0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6
      };
   double[] p = new double[]
      {
         0.1, 0.3, 0.25, 0.2, 0.1, 0.05
      };
   Random random = new Random();
   double r, s, x;
   int k;
   for (int i = 0; i < n; i++)
   {
      r = random.NextDouble();
      s = p[0];
      for (k = 1; s < r; k++)
         s += p[k];
      x = (xt[k] + xt[k - 1]) / 2;
      Console.WriteLine("x[{0}] = {1:f3}", i + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

Отметим, что при использовании описанного способа будут получаться только дискретные значения случайной величины, рав-

ные серединам интервалов. В ряде случаев необходимо, чтобы получаемая величина принимала любое значение из множества допустимых.

Для моделирования таких величин можно применить метод обратных функций к непрерывной кусочно-линейной эмпирической функции распределения. При этом данную функцию можно строить как по сгруппированным, так и по несгруппированным данным. Рассмотрим процесс построения.

Пусть имеется n наблюдений $X_1, X_2, ..., X_n$ случайной величины. Тогда можно определить непрерывную кусочно-линейную функцию распределения F, сначала отсортировав величины X_i в порядке возрастания.

Пусть $X_{(i)}$ обозначает i-е наименьшее значение величин $X_1, X_2, ..., X_n$ так, чтобы $X_{(1)} < X_{(2)} < ... < X_{(n)}$. Тогда F задается как

$$F(x) = \begin{cases} 0, \text{ если } x < X_{(1)}; \\ \dfrac{i-1}{n-1} + \dfrac{x - X_{(i)}}{(n-1)(X_{(i+1)} - X_{(i)})}, \text{ если } X_{(i)} \le x < X_{(i+1)}; \\ 1, \text{ если } X_{(n)} \le x. \end{cases}$$

На рис. 4.4 показан пример графика функции распределения для $n = 6$.

Если имеются отдельные наблюдения, т. е. определена функция F, то для генерации случайных величин можно использовать следующий алгоритм.

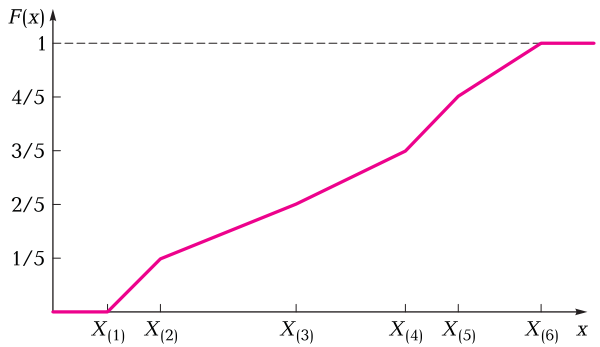


Рис. 4.4. Функция непрерывного кусочно-линейного эмпирического распределения, полученного из исходных данных

1. Генерируется R, имеющая равномерное распределение на интервале [0; 1).

2. Вычисляется $P = (n - 1)R$ и $I = \lfloor P \rfloor + 1$.

3. Возвращается $X = X_{(I)} + (P - I + 1)(X_{(I+1)} - X_{(I)})$.

Если данные сгруппированы, то нужно использовать другой подход к определению эмпирического распределения, поскольку значения отдельных величин X_i неизвестны.

Пусть n величин сгруппированы в k смежных интервалов $[a_0; a_1)$, $[a_1, a_2)$, ..., $[a_{k-1}; a_k)$ и j-й интервал содержит n_j наблюдений, где $n_1 + n_2 + ... + n_k = n$. Пусть $G(a_0) = 0$, $G(a_j) = (n_1 + n_2 + ... + n_j)/n$ для $j = 1, 2, ..., k$. Тогда можно определить функцию кусочно-линейного эмпирического распределения

$$G(x) = \begin{cases} 0, \text{ если } x < a_0; \\ G(a_{j-1}) + \dfrac{x - a_{j-1}}{a_j - a_{j-1}}\left[G(a_j) - G(a_{j-1})\right], \text{ если } a_{j-1} \le x < a_j; \\ 1, \text{ если } a_k \le x. \end{cases}$$

Если данные сгруппированы, т. е. определена функция G, для генерации случайных величин можно использовать следующий алгоритм.

1. Генерируется R, имеющая равномерное распределение на интервале [0; 1).

2. Определяется неотрицательное целое число J ($0 \le J \le k - 1$), так что $G(a_J) \le R < G(a_{J+1})$.

3. Возвращается $X = a_J + \left[R - G(a_J)\right](a_{J+1} - a_J) / \left[G(a_{J+1}) - G(a_J)\right]$.

Один очевидный недостаток генерации случайных величин по эмпирическому распределению состоит в том, что случайные значения, сгенерированные таким образом, никогда не могут быть меньше $X_{(1)}$ или больше $X_{(n)}$. Кроме того, среднее $F(x)$ не равно выборочному среднему величин X_i.

4.4. МЕТОД ОТБОРА

Если для генерации случайных величин нельзя применить точные методы, можно использовать приближенные. Одним из них является метод отбора (другие названия — метод режекций, метод Неймана). Данный метод позволяет генерировать случайные величины, заданные с помощью функции плотности вероятностей $f(x)$, определенной на некотором ограниченном интервале $[a; b)$.

Метод отбора предложен Дж. Нейманом, и сущность его сводится к тому, что из области задания случайной величины X «отбирается» точка с координатами, являющимися функциями случайных чисел с равномерным распределением. Если эта точка не может быть использована для расчета X, то происходит ее «отбрасывание» и «отбирается» новая.

Пусть необходимо смоделировать одномерную случайную величину X, заданную на интервале $[a; b)$ плотностью $f(x)$. Вне этого интервала $f(x) = 0$ и, кроме того, плотность распределения ограничена сверху, т. е. $f(x) \le C$, где C — постоянная.

Метод отбора получения значений случайной величины X графически показан на рис. 4.5 и заключается в следующем:

1) получают два независимых значения r_1 и r_2 случайной величины R с равномерным распределением на интервале $[0; 1)$;

2) строят точку с координатами (z_1, z_2), где

$$z_1 = a + r_1(b - a);\ z_2 = r_2 C;$$

3) если $z_2 < f(z_1)$ (т. е. точка попадает в область, ограниченную кривой $f(x)$, и прямыми $x = a$, $x = b$ и $y = 0$), то полагают, что величина X приняла значение z_1; иначе точка $z = (z_1, z_2)$ отбрасывается и вычисления повторяются с получением новой пары случайных чисел $(r_1; r_2)$ по пункту 1.

Пример. Составить программу формирования случайной величины с распределением, заданным функцией плотности вероятности

$$f(x) = \begin{cases} 2(x^2 + x^5), x \in [0,1]; \\ 0, x \notin [0; 1]. \end{cases}.$$

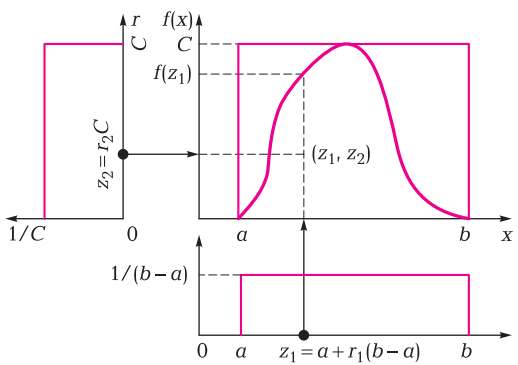


Рис. 4.5. Иллюстрация метода отбора

Решение. Сначала определим параметры a, b и C метода отбора. Функция $f(x)$ имеет ненулевое значение на интервале от 0 до 1. Следовательно, параметр метода отбора $a = 0$, а параметр $b = 1$. Данная функция на интервале [0; 1] является монотонно возрастающей. Следовательно, максимальное значение функция принимает на правой границе интервала, т. е. параметр $C = f(1) = 4$.

Далее представим пример программы на языке C# для генерации случайных величин, использующий данный метод.

```
static void Main(string[] args)
{
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   // задаем пределы, в которых функция плотности
   // вероятности ненулевая
   double a = 0, b = 1;
   // задаем максимальное значение для функции плотности
   // вероятности
   double C = 4;
   double x, r1, r2 z1, z2;
   bool f;
   for (int i = 0; i < n; i++)
   {
      f = true;
      do
      {
         r1 = random.NextDouble();
         r2 = random.NextDouble();
         z1 = a + r1 * (b – a);
         z2 = C * r2;
         if (z2 < 2 * (Math.Pow(z1, 2) + Math.Pow(z1, 5)))
         {
            x = z1;
            Console.WriteLine("x[{0}] = {1:f3}", i + 1, x);
            f = false;
         }
      }
      while (f);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

Эффективностью метода отбора называют вероятность того, что точка $z = (z_1, z_2)$ будет использована для расчета X. В рассмо-

тренной процедуре эффективность метода характеризуется отношением площади, ограниченной кривой $f(x)$, осью x, прямыми $x = a$ и $x = b$ к площади прямоугольника $C(b - a)$.

4.5. ГЕНЕРАЦИЯ НОРМАЛЬНО РАСПРЕДЕЛЕННЫХ СЛУЧАЙНЫХ ВЕЛИЧИН

Рассмотрим методы генерации специальных законов распределения.

Среди различных теоретических законов особое место занимает закон нормального распределения непрерывной случайной величины, так как является основным во многих практических исследованиях и им описывается большинство случайных явлений, связанных с производственными и другими процессами. К случайным явлениям, подчиняющимся нормальному закону распределения, относятся, например, ошибки измерений производственных параметров, распределение технологических погрешностей изготовления, рост и вес большинства биологических объектов, распределение параметров пленочных резисторов.

В связи с этим при моделировании различных явлений возникает потребность иметь в распоряжении последовательность случайных величин, отвечающих нормальному закону распределения.

Нормальный закон распределения задается функцией плотности

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

Нормальное распределение характеризуется параметрами μ и σ и обозначается как $N(\mu, \sigma^2)$. Отметим, что математическое ожидание нормально распределенной случайной величины равно μ, а дисперсия равна σ^2. График функции плотности вероятности при различных значениях μ показан на рис. 4.6 (при изменении μ график сдвигается), а при различных значениях σ — на рис. 4.7 (при изменении σ график растягивается или сужается).

Нормально распределенная случайная величина с $\mu = 0$ и $\sigma = 1$ называется стандартной нормально распределенной случайной величиной и обозначается как $N(0, 1)$. Важность данной величины

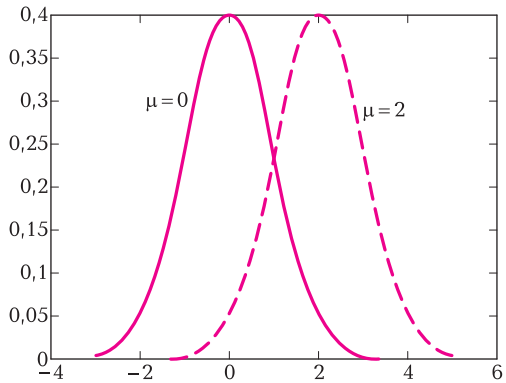


Рис. 4.6. Функция плотности нормального распределения при разных μ и $\sigma = 1$

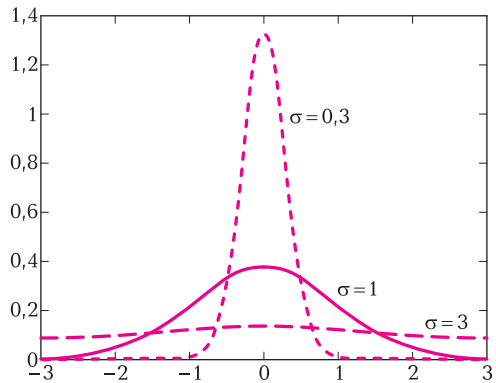


Рис. 4.7. Функция плотности нормального распределения при разных σ и $\mu = 0$

объясняется тем, что нормально распределенную случайную величину X' с параметрами μ и σ^2 всегда можно получить из стандартной нормально распределенной случайной величины X с помощью следующего преобразования:

$$X' = \mu + \sigma X.$$

Поэтому далее будем рассматривать только способы генерирования случайной величины с распределением $N(0, 1)$.

4.5.1. Метод аппроксимации

Случайная величина с нормальным распределением $N(0,1)$ может быть реализована с помощью классических общих методов моделирования. В качестве примера рассмотрим один из таких методов — метод аппроксимации.

Метод аппроксимации применяется в том случае, когда в формуле $x = F^{-1}(r)$ обращение функции распределения представляет собой сложную численную задачу. Однако для обратной функции F^{-1} можно найти достаточно хорошую аппроксимацию.

Пример. Составить программу моделирования случайной величины *X*, имеющей нормальное распределение с параметрами *N*(0, 1).

Решение. Обратную функцию плотности нормального распределения нельзя представить в виде простых, легко вычисляемых формул. Если же для случайной величины с плотностью

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}, \quad -\infty < x < \infty,$$

воспользоваться аппроксимацией

$$\sqrt{\frac{2}{\pi}} e^{-\frac{x^2}{2}} \approx \frac{2ke^{-kx}}{\left(1+e^{-kx}\right)^2}, \quad x > 0,$$

где $k = \sqrt{\frac{8}{\pi}}$, то для положительной полуоси можно получить следующую аппроксимацию:

$$r = F(x) = \int_0^{\infty} \frac{2ke^{-kx}}{\left(1+e^{-kx}\right)^2} dx = \frac{2}{1+e^{-kx}} - 1.$$

Тогда обратная функция имеет вид

$$x \approx \frac{1}{k} \ln \frac{1+r}{1-r}, \quad 0 \le r < 1.$$

Если значениям *x* присваивать с вероятностью 0,5 знак « + » и с вероятностью 0,5 знак «–», то полученная последовательность может рассматриваться как реализация случайной величины с распределением *N*(0, 1). Программа на языке C# имеет следующий вид:

```
static void Main(string[] args)
{
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double r, x;
   double k = Math.Sqrt(8.0 / Math.PI);
   for (int i = 0; i < n; i++)
   {
      r = random.NextDouble();
      x = Math.Log((1 + r) / (1 - r)) / k;
      r = random.NextDouble();
      if (r < 0.5) x = -x;
      Console.WriteLine("x[{0}] = {1:f3}", i + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```


4.5.2. Использование центральной предельной теоремы

Один из самых известных методов программной реализации нормально распределенной случайной величины основан на *центральной предельной теореме*: распределение суммы независимых случайных величин приближается к нормальному при неограниченном увеличении *n*, если выполняются следующие условия:

1) все величины имеют конечные математические ожидания и дисперсии;

2) ни одна из величин по своему значению резко не отличается от всех остальных.

Согласно этой теореме можно сконструировать алгоритм реализации случайной величины *X* на основе аппроксимации распределения $N(0,1)$ суммой независимых случайных величин $r_1, r_2, \ldots, r_n$, равномерно распределенных на интервале [0; 1). Так как каждая из них имеет математическое ожидание $M(r_i) = 1/2$ и дисперсию $D(r_i) = 1/12$, $i = 1, 2, \ldots, n$, то

$$M\left(\sum_{i=1}^{n} r_i\right) = \frac{n}{2}, \quad D\left(\sum_{i=1}^{n} r_i\right) = \frac{n}{12}.$$

Согласно центральной предельной теореме распределение случайной величины

$$x = \frac{\sum_{i=1}^{n} r_i - \frac{n}{2}}{\sqrt{\frac{n}{12}}} \qquad (4.3)$$

при достаточно больших n приближается к $N(0,1)$. При конечном n распределение приближенно нормальное.

Практика показывает, что при $n = 12$ получаем достаточно хорошее и удобное для расчета приближение.

Формула для моделирования имеет вид

$$x = \sum_{i=1}^{12} r_i - 6.$$

Пример. Составить программу моделирования случайной величины X, отвечающей распределению $N(0{,}1;\ 2{,}25)$.

Решение. Программа на языке C# имеет следующий вид:

```
static void Main(string[] args)
{
   Console.Write("Математическое ожидание m: ");
   double m = double.Parse(Console.ReadLine());
   Console.Write("Дисперсия d: ");
   double d = double.Parse(Console.ReadLine());
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double s, r;
   for (int i = 0; i < n; i++)
   {
      s = 0;
      for (int j = 0; j < 12; j++)
      {
         r = random.NextDouble();
         s += r;
      }
      double x = s - 6;
      x = m + x * Math.Sqrt(d);
      Console.WriteLine("x[{0}] = {1:f3}", i + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

К недостатку метода, основанного на центральной предельной теореме, можно отнести то, что с его помощью, как следует из выражения (4.3), получаются случайные числа из конечного диапазона, в то время как нормально распределенные числа могут принимать любые вещественные значения.

4.5.3. Метод Бокса и Маллера

В другом известном генераторе нормально распределенных случайных величин используется точный обратный метод Бокса и Маллера. Метод предназначен для реализации двумерной случайной величины $[X, Y]$, заданной совместной функцией плотности $f(x, y)$. Если случайные величины X и Y независимы, то $f(x, y) = (x)f(y)$ и моделирование распределения двумерной случайной величины сводится к моделированию пары нормированных нормальных величин x и y через независимые, равномерно распределенные на интервале [0; 1) случайные числа R_1 и R_2 с помощью следующих уравнений:

$$x = \sqrt{-2\ln r_1}\cos(2\pi r_2);$$
$$y = \sqrt{-2\ln r_1}\sin(2\pi r_2).$$

Метод дает хорошие результаты, легко программируется и достаточно быстро работает.

Пример. Составить программу моделирования двух последовательностей случайных величин, отвечающих распределениям $N(-1{,}5;\ 0{,}9)$ и $N(2{,}5;\ 1{,}96)$.

Решение. Программа на языке C# имеет следующий вид:

```
static void Main(string[] args)
{
   Console.WriteLine("Параметры первого распределения");
   Console.Write("математическое ожидание m1: ");
   double m1 = double.Parse(Console.ReadLine());
   Console.Write("дисперсия d1: ");
   double d1 = double.Parse(Console.ReadLine());
   Console.WriteLine("Параметры второго распределения");
   Console.Write("математическое ожидание m2: ");
   double m2 = double.Parse(Console.ReadLine());
   Console.Write("дисперсия d2: ");
   double d2 = double.Parse(Console.ReadLine());
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double r1, r2, x, y;
   for (int i = 0; i < n; i++)
   {
      r1 = random.NextDouble();
      r2 = random.NextDouble();
```

```
      x = Math.Sqrt(-2 * Math.Log(r1)) *
         Math.Cos(2 * Math.PI * r2);
      x = m1 + x * Math.Sqrt(d1);
      y = Math.Sqrt(-2 * Math.Log(r1)) *
         Math.Sin(2 * Math.PI * r2);
      y = m2 + y * Math.Sqrt(d2);
      Console.WriteLine("x[{0}] = {1:f3}\ty[{0}] = {2:f3}",
         i + 1, x, y);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

4.5.4. Метод Марсальи и Брея

Модификацией метода Бокса и Маллера является метод Марсальи и Брея. Этот метод предназначен для реализации двумерной случайной величины $[X, Y]$, заданной совместной функцией плотности $f(x, y) = f(x)f(y)$ при условии, что случайные величины X и Y независимы. Данный метод является более быстрым за счет исключения вычислений синусов и косинусов. Процедура получения случайных величин заключается в следующем:

1) генерируются независимые, равномерно распределенные на интервале $[-1; 1)$, случайные величины U_1 и U_2:

$$U_1 = -1 + 2R_1;$$

$$U_2 = -1 + 2R_2,$$

где R_1, R_2 — независимые, равномерно распределенные на интервале $[0; 1)$, случайные числа;

2) проверяется условие $U_1^2 + U_2^2 \le 1$. Если условие не выполняется, то цикл повторяют снова. Если условие верно, то реализуют случайные величины по формулам

$$X = U_1\sqrt{\frac{-2\ln(U_1^2 + U_2^2)}{U_1^2 + U_2^2}},\ Y = X\frac{U_2}{U_1}.$$

Метод дает хорошие результаты и легко программируется. В то же время данный метод обладает следующими недостатками:

- достаточно медленный, поскольку в нем используются тригонометрические функции, которые долго вычисляются;
- при r_1, близких к нулю, возможно переполнение разрядной сетки ЭВМ.

Данные недостатки являются серьезными в случае использования генератора в стохастических моделях, требующих получения миллионов случайных величин.

Пример. Составить программу моделирования двух последовательностей случайных величин, отвечающих распределениям $N(-1,5; 0,9)$ и $N(2,5; 1,96)$.

Решение. Программа на языке C# имеет следующий вид:

```
static void Main(string[] args)
{
   Console.WriteLine("Параметры первого распределения");
   Console.Write("математическое ожидание m1: ");
   double m1 = double.Parse(Console.ReadLine());
   Console.Write("дисперсия d1: ");
   double d1 = double.Parse(Console.ReadLine());
   Console.WriteLine("Параметры второго распределения");
   Console.Write("математическое ожидание m2: ");
   double m2 = double.Parse(Console.ReadLine());
   Console.Write("дисперсия d2: ");
   double d2 = double.Parse(Console.ReadLine());
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double r1, r2, u1, u2, s, x, y;
   int i = 0;
   while (i < n)
   {
      r1 = random.NextDouble();
      r2 = random.NextDouble();
      u1 = -1 + 2 * r1;
      u2 = -1 + 2 * r2;
      s = u1 * u1 + u2 * u2;
      if (s <= 1)
      {
         x = u1 * Math.Sqrt(-2 * Math.Log(s) / s);
         y = u2 * Math.Sqrt(-2 * Math.Log(s) / s);
         x = m1 + x * Math.Sqrt(d1);
         y = m2 + y * Math.Sqrt(d2);
         Console.WriteLine("x[{0}] = {1:f3}\ty[{0}] = {2:f3}",
         i + 1, x, y);
         i++;
      }
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

4.6. ГЕНЕРАЦИЯ СЛУЧАЙНЫХ ВЕЛИЧИН СО СПЕЦИАЛЬНЫМИ ЗАКОНАМИ РАСПРЕДЕЛЕНИЯ

Для некоторых законов распределения случайных величин, наиболее часто встречающихся в приложениях, разработаны частные приемы моделирования. Частные приемы не исключают возможностей использования общих методов. Решение каждой конкретной задачи требует индивидуального подхода к выбору алгоритма моделирования. Одним из показателей является *эффективность метода*, под которой понимают отношение количества псевдослучайных величин требуемого распределения к количеству применяемых для этого равномерно распределенных случайных чисел.

4.6.1. Моделирование экспоненциального распределения

Экспоненциальное, или показательное, распределение — это непрерывное распределение, моделирующее время между двумя последовательными свершениями одного и того же события, время безотказной работы устройства.

Функция плотности экспоненциального распределения задается выражением

$$f(x) = \lambda e^{-\lambda x}, \quad x \geq 0,$$

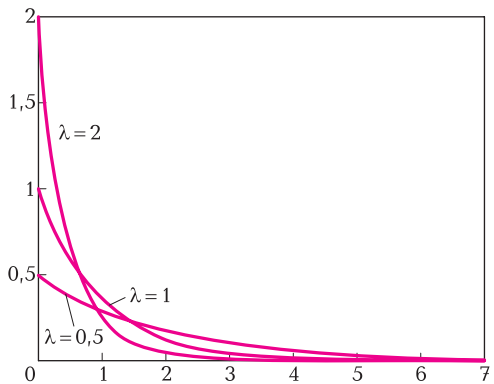


Рис. 4.8. Функция плотности экспоненциального распределения

где λ – параметр распределения, $\lambda > 0$.

Для экспоненциального распределения матожидание и дисперсия определяются в соответствии с выражениями

$$M(X) = \lambda;$$

$$D(X) = \lambda^2.$$

Графики функции плотности вероятностей для различных значений параметра λ показаны на рис. 4.8.

Формула для моделирования случайных величин с экспоненциальным распределением с параметром λ имеет вид

$$x = -\frac{1}{\lambda}\ln r.$$

Пример моделирования был рассмотрен в подразд. 4.1.

4.6.2. Моделирование бета-распределения

Бета-распределение является непрерывным распределением с конечными верхним и нижним граничными значениями. Такие ограничения могут существовать во множестве реальных ситуаций, поэтому бета-распределение можно применять для аппроксимации реального распределения до того, как получено достаточно большое количество информации. Бета-распределение в достаточной мере соответствует большинству данных.

Плотность бета-распределения на интервале (0; 1) определяется формулой

$$f_{p,m}(x) = \frac{x^{p-1}(1-x)^{m-1}}{\mathrm{B}(p, m)}, \quad 0 < x < 1,$$

где $p > 0$, $m > 0$ — параметры распределения.

Бета-функция имеет вид

$$\mathrm{B}(p,m) = \int_0^1 x^{p-1}(1-x)^{m-1}\,\mathrm{d}x.$$

Для бета-распределения матожидание и дисперсия определяются в соответствии с выражениями

$$M(X) = \frac{p}{p+m};$$

$$D(X) = \frac{pm}{(p+m)^2(p+m+1)}.$$

Графики функции плотности вероятностей для различных значений параметров m и p показаны на рис. 4.9.

Если m — целое, p — нецелое, формула для моделирования случайной величины $X_{p,m}$, имеющей бета-распределение с параметрами p и m, имеет вид

$$x_{p,m} = \prod_{k=1}^{m} r_k^{\frac{1}{p+k-1}} = \exp\left(\sum_{k=1}^{m} \frac{\ln(r_k)}{p+k-1}\right),$$

где $r_1, r_2, ..., r_m$ — случайные числа, равномерно распределенные на интервале [0; 1).

Формулу можно использовать и в случае, когда m — нецелое, p — целое, сделав предварительно замену переменных вида $y = 1 - x$.

Для моделирования бета-распределения с нецелыми параметрами $p > 0$, $m > 0$ можно использовать метод суперпозиций. Тогда формула для моделирования имеет вид

$$x_{p,m} = \exp\left(\sum_{k=1}^{\lfloor m \rfloor + 1} \frac{\ln(r_k)}{\nu + p + k - 1}\right),$$

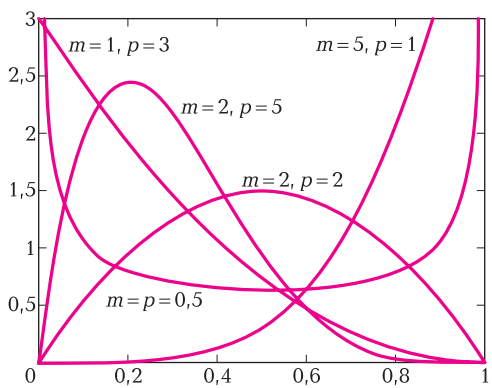


Рис. 4.9. Функция плотности бета-распределения

где $\lfloor m \rfloor$ — целая часть m; v — целочисленная случайная величина, имеющая дискретное распределение, заданное рядом распределения $p(v = k) = p_k$, $k = 0, 1, 2, ...$;

$$p_k = \frac{\lfloor m \rfloor!}{B(p,m)} \cdot \frac{a(a+1)...(a+k-1)}{k!(k+p)(k+p+1)...(k+p+\lfloor m \rfloor)}, \quad a = \lfloor m \rfloor + 1 - m.$$

Более простую процедуру моделирования случайных величин, отвечающих бета-распределению, предложил австрийский математик Йонк. Процедура заключается в следующем:

1) генерируются случайные значения r_1 и r_2, равномерно распределенные на интервале [0; 1);

2) проверяется условие $r_1^{1/p} + r_2^{1/m} \le 1$. Если условие не выполняется, то цикл повторяется. Если условие верно, то реализуется случайная величина по формуле

$$r_{p,m} = \frac{r_1^{1/p}}{r_1^{1/p} + r_2^{1/m}}.$$

Пример. Составить программу моделирования случайных чисел, отвечающих бета-распределению с параметрами $m = 3$, $p = 2{,}5$.

Решение. Воспользуемся общей формулой для моделирования при m — целом, p — нецелом. Программа на языке C# имеет следующий вид:

```
static void Main(string[] args)
{
   Console.WriteLine("Параметры распределения");
   Console.Write("p: ");
   double p = double.Parse(Console.ReadLine());
   Console.Write("m: ");
   int m = int.Parse(Console.ReadLine());
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double s, r, x;
   for (int i = 0; i < n; i++)
   {
      s = 0.0;
      for (int k = 0; k < m; k++)
      {
         r = random.NextDouble();
         s += Math.Log(r) / (p + k);
      }
      x = Math.Exp(s);
```

```
      Console.WriteLine("x[{0}] = {1:f3}", i + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

4.6.3. Моделирование гамма-распределения

Гамма-распределение широко применяется в экономике и менеджменте, теории и практике надежности и испытаний, в различных областях техники, метеорологии и т.д. В частности, гамма-распределению подчинены во многих ситуациях такие величины, как общий срок службы изделия, время достижения изделием предельного состояния при коррозии, время наработки до k-го отказа, $k = 1, 2, \ldots$, и т.д. Данное распределение подходит для описания спроса в экономико-математических моделях управления запасами (логистики).

Функция плотности гамма-распределения задается выражением

$$f_{\alpha,\beta}(x) = \frac{x^{\alpha-1}e^{-\frac{x}{\beta}}}{\beta^{\alpha}\Gamma(z)}, \quad x > 0,$$

где $\alpha > 0$, $\beta > 0$ — параметры распределения; $\Gamma(z)$ — гамма-функция;

$$\Gamma(Z) = \int_0^{\infty} t^{z-1}e^{-t}\mathrm{d}t.$$

В частном случае для целого Z

$$\Gamma(Z+1) = Z!.$$

Для гамма-распределения матожидание и дисперсия определяются в соответствии с выражениями

$$M(X) = \alpha\beta;$$

$$D(X) = \alpha\beta^2.$$

Графики функции плотности вероятностей для различных значений параметров α и β показаны на рис. 4.10.

Следует отметить, что в случае значения параметра $\alpha = 1$, гамма-распределение совпадает с ранее рассмотренным экспоненциальным распределением с параметром β.

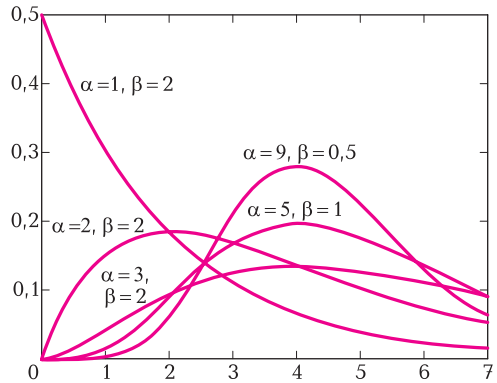


Рис. 4.10. Функция плотности гамма-распределения

Формула для моделирования случайных чисел с гамма-распределением с целочисленным параметром $\alpha = n > 1$ и параметром $\beta = 1$ имеет вид

$$x = -\sum_{k=1}^{n}\ln(r_k) = -\ln\left(\prod_{k=1}^{n} r_k\right).$$

Для того чтобы получить случайную величину с гамма-распределением с целочисленным параметром α и $\beta \neq 1$ достаточно умножить формулу (4.4) на β.

Пример. Составить программу моделирования случайных величин, отвечающих гамма-распределению с параметрами $\alpha = n$, $\beta = 1$.

Решение. Программа на языке C# имеет следующий вид:

```
static void Main(string[] args)
{
   Console.Write("Параметр распределения lambda: ");
   int lambda = int.Parse(Console.ReadLine());
   Console.Write("Количество случайных величин n: ");
   int m = int.Parse(Console.ReadLine());
   Random random = new Random();
   double p, r, x;
   for (int i = 0; i < n; i++)
   {
      p = 1;
      for (int k = 0; k < lambda; k++)
      {
         r = random.NextDouble();
         p *= r;
```

```
      }
      x = -Math.Log(p);
      Console.WriteLine("x[{0}] = {1:f3}", i + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

4.6.4. Моделирование логарифмически-нормального закона распределения

Случайная величина X имеет логарифмически-нормальное распределение с параметрами μ, σ, если $X = \exp(Y)$, где Y имеет нормальное распределение с параметрами μ, σ. С помощью данного распределения часто описывается время выполнения какой-либо задачи, величины, являющиеся произведением большого числа других величин.

Логарифмически-нормальный закон распределения случайной величины описывается следующей функцией плотности:

$$f(x) = \frac{1}{x\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(\ln x - \mu)^2}{2\sigma^2}\right],$$

где $\sigma > 0$ и μ — параметры распределения.

Для данного распределения матожидание и дисперсия определяются в соответствии с выражениями

$$M(X) = e^{\mu+\sigma^2/2};$$

$$D(X) = e^{2\mu+\sigma^2}(e^{\sigma^2} - 1).$$

Графики функции плотности вероятностей для $\mu = 0$ и различных значений параметра σ показаны на рис. 4.11.

Для моделирования случайной величины с логарифмически нормальным распределением используется выражение

$$x = \exp(\sigma y + \mu),$$

где y — стандартная нормально распределенная случайная величина с параметрами $\mu = 0$, $\sigma = 1$.

Пример. Составить программу моделирования случайной величины X, отвечающей логарифмически-нормальному распределению.

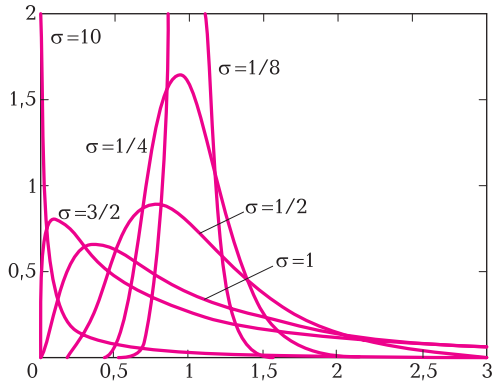


Рис. 4.11. Функция плотности логарифмически нормального распределения при $\mu = 0$

Решение. Программа на языке C# имеет следующий вид:

```
static void Main(string[] args)
{
   Console.WriteLine("Параметры распределения");
   Console.Write("c: ");
   double c = double.Parse(Console.ReadLine());
   Console.Write("b: ");
   double b = double.Parse(Console.ReadLine());
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double s, r, x;
   for (int i = 0; i < n; i++)
   {
      s = 0;
      for (int j = 0; j < 12; j++)
      {
         r = random.NextDouble();
         s+ = r;
      }
      x = s - 6;
      x = Math.Exp(c * x + b);
      Console.WriteLine("x[{0}] = {1:f3}", i + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

4.6.5. Моделирование распределения Вейбулла

С помощью распределения Вейбулла часто описывается время выполнения какой-либо задачи, величины, время безотказной работы устройства.

Функция плотности для распределения Вейбулла имеет вид

$$f(x) = \frac{cx^{c-1}}{b^c}\exp\left(-\left(\frac{x}{b}\right)^c\right),$$

где $0 \le x < \infty$; $c > 0$; $b > 0$; b, c — целые.

Для распределения Вейбулла матожидание и дисперсия определяются в соответствии с выражениями

$$M(X) = \frac{b}{c}\Gamma\left(\frac{1}{c}\right);$$

$$D(X) = \frac{b^2}{c}\left[2\Gamma\left(\frac{2}{c}\right) - \frac{1}{c}\left[\Gamma\left(\frac{1}{c}\right)\right]^2\right].$$

Здесь Γ( ) — гамма функция.

Графики функции плотности вероятностей для $b = 1$ и различных значений параметра c показаны на рис. 4.12.

Распределение Вейбулла может быть смоделировано путем использования следующей формулы:

$$x = b[-\ln(r)]^{1/c},$$

где r — равномерно распределенная на интервале [0; 1) случайная величина. Заметим, что при моделировании значение r должно быть ограничено снизу малой величиной.

Пример. Составить программу моделирования случайной величины X, отвечающей распределению Вейбулла.

Решение. Программа на языке C# имеет следующий вид:

```
static void Main(string[] args)
{
   Console.WriteLine("Параметры распределения");
   Console.Write("c: ");
   double c = double.Parse(Console.ReadLine());
   Console.Write("b: ");
   double b = double.Parse(Console.ReadLine());
```

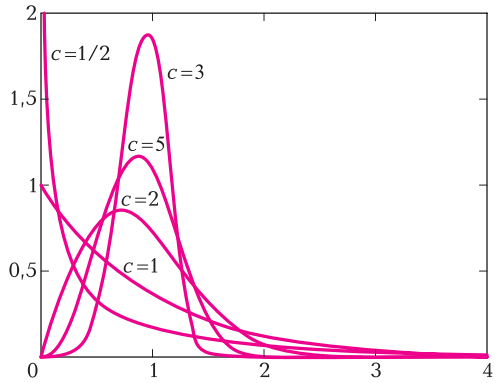



Рис. 4.12. Функция плотности распределения Вейбулла при $b = 1$

```
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double r, x;
   for (int i = 0; i < n; i++)
   {
      r = random.NextDouble();
      x = b * Math.Pow(-Math.Log(r), 1 / c);
      Console.WriteLine("x[{0}] = {1:f3}", i + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем заключается метод обратных функций?
2. В каких случаях возможно применение метода обратных функций?
3. В чем состоит метод кусочно-линейной аппроксимации? В каких случаях он применяется?
4. Какой метод формирования случайных чисел необходимо использовать в том случае, если неизвестно выражение для функции распределения? В чем он заключается?
5. В чем заключается метод отбора? В каких случаях он применяется?
6. Какова геометрическая интерпретация метода отбора?

7. Как выглядит функция плотности для нормального закона распределения?
8. Как выглядит функция распределения для нормального закона распределения?
9. Какие существуют способы формирования последовательности случайных величин, отвечающих нормальному закону распределения?
10. В чем заключается метод аппроксимации для моделирования нормально распределенных случайных величин?
11. Каким образом используется центральная предельная теорема для формирования последовательности случайных величин, отвечающих нормальному закону распределения?
12. В чем состоит сущность метода Бокса и Маллера?
13. В чем заключается сущность метода Марсальи и Брея?
14. Как выглядит функция плотности бета-распределения?
15. Как выглядит функция плотности гамма-распределения?
16. Как выглядит функция плотности логарифмически-нормального распределения?
17. Как выглядит функция плотности распределения Вейбулла?
18. Каким образом осуществляется моделирование случайных величин, имеющих бета-распределение?
19. Каким образом осуществляется моделирование случайных величин, имеющих гамма-распределение?
20. Каким образом осуществляется моделирование случайной величины, имеющей логарифмически-нормальное распределение?
21. Какой метод используется для моделирования распределения Вейбулла?

Глава 5

МОДЕЛИРОВАНИЕ ДИСКРЕТНЫХ СОБЫТИЙ И РАСПРЕДЕЛЕНИЙ

5.1. МОДЕЛИРОВАНИЕ ПРОИЗВОЛЬНОГО ДИСКРЕТНОГО РАСПРЕДЕЛЕНИЯ

При моделировании часто приходится генерировать *дискретные случайные величины*, принимающие множество дискретных (счетных) значений. Примером подобных величин является число очков, выпавших при бросании игральной кости, число пассажиров автобуса и т. п.

Пусть требуется получить последовательность возможных значений $x_1, x_2, \ldots x_n$ дискретной случайной величины X, заданной рядом распределения $p(X = X_i) = p_i$:

X	x_1	x_2	...	x_n
$P(X)$	p_1	p_2	...	p_n

При этом выполняется условие $\sum_{i=1}^{n} p_i = 1$.

Для генерирования такой дискретной случайной величины используем непрерывную случайную величину R, равномерно распределенную в интервале [0; 1). Рассмотрим интервал [0; 1) и разобьем его на n подынтервалов, длины которых равны $p_1, p_2, \ldots, p_n$. Координаты точек деления будут $y_1 = p_1$, $y_2 = p_1 + p_2$, ..., $y_{n-1} = p_1 + p_2 + \ldots + p_{n-1}$ (рис. 5.1).

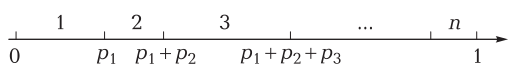


Рис. 5.1. Пример разбиения на подынтервалы

Случайная величина R равномерно распределена в интервале [0; 1), а значит, вероятность того, что R окажется в некотором интервале, равна длине этого интервала:

$$P\{0 \le R < p_1\} = p_1;$$
$$P\{p_1 \le R < p_1 + p_2\} = p_2;$$
$$\vdots$$
$$P\{p_1 + p_2 + \ldots + p_{n-1} \le R < 1\} = p_n.$$

В результате можно считать, что если случайное число R попало в интервал с номером i, то случайная величина X приняла значение x_i.

Таким образом, алгоритм генерирования дискретной случайной величины, заданной рядом распределения $p_i = P(X = x_i)$, будет следующим.

1. Генерируется R, имеющее равномерное распределение на интервале [0; 1).

2. Определяется такое I, для которого $\sum_{i=1}^{I-1} p_i \le R < \sum_{i=1}^{I} p_i$.

3. Возвращается $X = I$ (предполагается, что $X_i = i$).

Пример. Составить программу моделирования случайной величины X, которая принимает значения k_i с вероятностью p_i:

k_i	0	1	2	3	4	5
p_i	0,1	0,3	0,25	0,2	0,1	0,05

Решение. Программа на языке C#, работающая в соответствии с рассмотренным алгоритмом, имеет следующий вид:

```
static void Main(string[] args)
{
   Console.Write("Количество случайных величин N: ");
   int n = int.Parse(Console.ReadLine());
   double[] p = new double[]
      {
         0.1, 0.3, 0.25, 0.2, 0.1, 0.05
      };
   Random random = new Random();
   double r, s;
   int x;
   for (int i = 0; i < n; i++)
   {
      r = random.NextDouble();
```



```
      s = 0.0;
      int k;
      for (k = 0; s < r; k++)
         s += p[k];
      x = k-1;
      Console.WriteLine("x[{0}] = {1}", i + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

Рассмотренный подход позволяет моделировать любую дискретно распределенную случайную величину. Однако он не всегда является наиболее эффективным для моделирования некоторых частных законов распределения. Поэтому далее рассмотрим наиболее часто встречающиеся дискретные распределения и способы их моделирования.

5.2. МОДЕЛИРОВАНИЕ РАСПРЕДЕЛЕНИЯ БЕРНУЛЛИ

Распределение Бернулли описывает дискретную случайную величину, имеющую два возможных значения (0 и 1). Вероятностная мера (ряд распределения) такой случайной величины описывается выражением

$$p(x) = \begin{cases} 1 - p, & \text{если } x = 0; \\ p, & \text{если } x = 1; \\ 0, & \text{в противном случае,} \end{cases}$$

где p — параметр распределения.

Для распределения Бернулли матожидание и дисперсия определяются в соответствии с выражениями

$$M(X) = p;$$
$$D(X) = p(1 - p).$$

Для генерации случайной величины, имеющей распределение Бернулли, можно воспользоваться следующим алгоритмом.

1. Генерируется R, имеющее равномерное распределение на интервале [0; 1).

2. Если $R < p$, то возвращается $X = 1$. В противном случае возвращается $X = 0$.

Пример. Составить программу моделирования случайных величин, отвечающих распределению Бернулли с параметром p.

Решение. В соответствии с изложенным ранее алгоритмом моделирования программа на языке C# имеет следующий вид:

```
static void Main(string[] args)
{
   Console.Write("Вероятность появления события p: ");
   double p = double.Parse(Console.ReadLine());
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double r;
   int x;
   for (int j = 0; j < n; j++)
   {
      r = random.NextDouble();
      if (r < p) x = 1;
      else x = 0;
      Console.WriteLine("x[{0}] = {1}", j + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

5.3. МОДЕЛИРОВАНИЕ БИНОМИАЛЬНОГО РАСПРЕДЕЛЕНИЯ

Биномиальное распределение описывает дискретную случайную величину, представляющую собой число успешных испытаний в t независимых испытаниях Бернулли, вероятность успеха в каждом из которых постоянна и равна p. Вероятностная мера биномиального распределения задается выражением

$$p(x) = \begin{cases} C_t^x p^x (1-p)^{t-x}, \text{ если } x \in \{0, 1, 2, ..., t\}; \\ 0, \text{ в противном случае.} \end{cases}$$

Здесь p, t — параметры распределения.

Так как сумма t независимых и одинаково распределенных величин с распределением Бернулли имеет биномиальное распределение, то алгоритм генерации случайной величины с биномиальным распределением таков.

1. Генерируются случайные значения $Y_1, Y_2, ..., Y_t$, имеющие распределение Бернулли с параметром p.

2. Возвращается $X = \sum_{i=1}^{t} Y_i$.

Пример. Составить программу моделирования случайных величин, отвечающих биномиальному распределению с параметрами $t = 10$, $p = 0{,}35$.

Решение. В соответствии с изложенным ранее алгоритмом моделирования программа на языке C# имеет следующий вид:

```
static void Main(string[] args)
{
   Console.Write("Вероятность появления события p: ");
   double p = double.Parse(Console.ReadLine());
   Console.Write("Число независимых испытаний t: ");
   int t = int.Parse(Console.ReadLine());
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double r;
   int x;
   for (int j = 0; j < n; j++)
   {
      int s = 0;
      for (int i = 0; i < t; i++)
      {
         r = random.NextDouble();
         if (r < p) s++;
      }
      x = s;
      Console.WriteLine("x[{0}] = {1}", j + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

Заметим, что при больших значениях t и малых p для уменьшения вычислительной сложности моделирование случайной величины с биномиальным распределением выполняют на основе общего алгоритма моделирования дискретных случайных величин, заданных рядом распределения $p_i = P(X = x_i)$. При этом $p_i = C_t^i p^i (1-p)^{t-i}$.

Получаем следующий алгоритм.

1. Генерируется R, имеющее равномерное распределение на интервале [0; 1). Устанавливается $S = 0$, $k = 0$.

2. Определяется $S = S + p_k$.

3. Если $S \le R$, то возвращается $X = k$. В противном случае $k = k + 1$ и переходят к шагу 2.

Заметим, что для вычисления p_k можно использовать рекуррентное соотношение

$$p_k = p_{k-1}\frac{t-k}{k+1}\frac{p}{1-p}.$$

Начальное значение $p_0 = (1-p)^t$.

В результате случайная величина X принимает значение, равное числу итераций k, которое нужно выполнить, чтобы удовлетворялось неравенство

$$R > \sum_{i=0}^{k} p_i.$$

Пример. Составить программу моделирования случайных величин, отвечающих биномиальному распределению с параметрами $n = 5\,000$, $p = 0{,}01$.

Решение. Программа на языке C# имеет следующий вид:

```
static void Main(string[] args)
{
   Console.Write("Вероятность появления события p: ");
   double p = double.Parse(Console.ReadLine());
   Console.Write("Число независимых испытаний t: ");
   int t = int.Parse(Console.ReadLine());
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double pk, s, r;
   int x;
   for (int i = 0; i < n; i++)
   {
      pk = Math.Pow(1 - p, t);
      s = pk;
      r = random.NextDouble();
      int k;
      for (k = 0; r > s; k++)
      {
         pk *= (t - k) * p / (k + 1) / (1 - p);
         s += pk;
      }
      x = k;
      Console.WriteLine("x[{0}] = {13}", i + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```


5.4. МОДЕЛИРОВАНИЕ СЛУЧАЙНОЙ ВЕЛИЧИНЫ С ГЕОМЕТРИЧЕСКИМ РАСПРЕДЕЛЕНИЕМ

Геометрическое распределение описывает дискретную случайную величину, представляющую собой число неудачных испытаний до первого успешного в последовательности независимых испытаниях Бернулли, вероятность успеха в каждом из которых равна p. Вероятностная мера такого распределения задается выражением

$$p(x) = \begin{cases} p(1-p)^x, \text{ если } x \in \{0, 1, 2, \ldots\}; \\ 0, \text{ в противном случае.} \end{cases}$$

Здесь p — параметр распределения.

Исходя из смысла случайной величины с геометрическим распределением, можно использовать следующий алгоритм генерации такой случайной величины.

1. $k = 0$.

2. Генерируется Y, имеющая распределение Бернулли с параметром p.

3. Если $Y = 0$, то устанавливается $k = k + 1$ и переходят к шагу 2. В противном случае возвращается $X = k$.

Пример. Составить программу моделирования случайных величин, отвечающих геометрическому распределению с параметром $p = 0{,}1$.

Решение. Программа на языке C# имеет следующий вид:

```
static void Main(string[] args)
{
   Console.Write("Вероятность появления события p: ");
   double p = double.Parse(Console.ReadLine());
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double r;
```

```
   int x, k;
   bool Y;
   for (int i = 0; i < n; i++)
   {
      k = 0;
      do
      {
         r = random.NextDouble();
         Y = r < p;
         if (Y) k++;
      }
      while (Y);
      x = k;
      Console.WriteLine("x[{0}] = {1}", i + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

Отметим, что при малых значениях p для получения одной случайной величины в среднем требуется достаточно большое число итераций алгоритма, из-за чего такой генератор становится медленным. Для генерации случайной величины с геометрическим распределением при малых значениях p можно использовать следующий алгоритм.

1. Генерируем R, имеющую равномерное распределение на интервале [0; 1).

2. Возвращаем $X = \lfloor \ln R / \ln(1-p) \rfloor$

5.5. МОДЕЛИРОВАНИЕ РАСПРЕДЕЛЕНИЯ ПУАССОНА

Распределение Пуассона описывает дискретную случайную величину, представляющую собой число событий в интервале времени, когда события происходят с одинаковой интенсивностью.

Вероятностная мера случайной величины, распределенной по закону Пуассона, определяется выражением

$$p(x) = \begin{cases} \dfrac{\lambda^x e^{-\lambda}}{x!}, \text{ если } x \in \{0, 1, \ldots\}; \\ 0, \text{ в противном случае.} \end{cases}$$

Здесь λ — параметр распределения, определяющий интенсивность наступления событий (число событий в единицу времени).

Моделирующий алгоритм основывается на следующем известном утверждении: если случайные величины y_1, y_2, ... независимы и все имеют экспоненциальное распределение с математическим ожиданием, равным 1, то неотрицательное целое число k, для которого выполняется неравенство

$$\sum_{i=1}^{k} y_i \le \lambda < \sum_{i=1}^{k+1} y_i,$$

имеет распределение Пуассона с параметром λ.

В связи с тем что $y_i = -\ln(r_i)$, где r_i — случайная величина, равномерно распределенная на интервале [0; 1), условие можно записать в виде

$$\prod_{i=1}^{k+1} r_i \le e^{-\lambda} < \prod_{i=1}^{k} r_i.$$

Процедура получения случайных величин, распределенных по закону Пуассона, с параметром λ, заключается в следующем:

1) реализуется последовательность $r_1, r_2, \ldots, r_n$ независимых случайных величин, равномерно распределенных на интервале [0; 1);

2) вычисляются произведения $r_1, r_1 r_2, r_1 r_2 r_3, \ldots$ до тех пор, пока не выполнится условие

$$\prod_{i=1}^{k+1} r_i \le e^{-\lambda} < \prod_{i=1}^{k} r_i.$$

В качестве значения случайной величины X принимается число k. Если неравенству удовлетворяет первое из равномерно распределенных чисел r_1, то $x = 0$.

Пример. Составить программу моделирования случайных величин, отвечающих распределению Пуассона с параметром $\lambda = 2$.

Решение. Программа на языке C# имеет следующий вид:

```
static void Main(string[] args)
{
   Console.Write("Параметр распределения Lambda: ");
   double lambda = double.Parse(Console.ReadLine());
   Console.Write("Количество случайных величин n: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   double r, pr;
```

```
   int x;
   for (int i = 0; i < n; i++)
   {
      r = random.NextDouble();
      pr = r;
      int k;
      for (k = 0; pr > Math.Exp(-lambda); k++)
      {
         r = random.NextDouble();
         pr *= r;
      }
      x = k;
      Console.WriteLine("x[{0}] = {1}", i + 1, x);
   }
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

5.6. МОДЕЛИРОВАНИЕ ПРОСТОГО СОБЫТИЯ

В процессе моделирования часто требуется имитировать случайные события с заранее известными вероятностями. *Случайным* называется событие, которое при определенной совокупности условий во время испытаний может произойти или не произойти [2]. Каждому событию из множества возможных соответствует вероятность события. Вероятность достоверного события, которое обязательно должно произойти, равна единице. Вероятность невозможного события равна нулю. Рассмотрим способ моделирования отдельного события.

Пусть имеется событие A, вероятность наступления которого равна p_A (например, стрелок попадает по мишени с вероятностью 0,85). Требуется выработать правило, при многократном использовании которого частота появления события A стремилась бы к его вероятности.

Выберем с помощью датчика случайных чисел, равномерно распределенных в интервале [0; 1), некоторое число z и определим вероятность того, что $z < p_A$. Для случайной величины z с равномерным распределением на интервале [0; 1) справедлива следующая зависимость:

$$P(z < p_A) = \int_{-\infty}^{p_A} f(x)\mathrm{d}x = p_A.$$

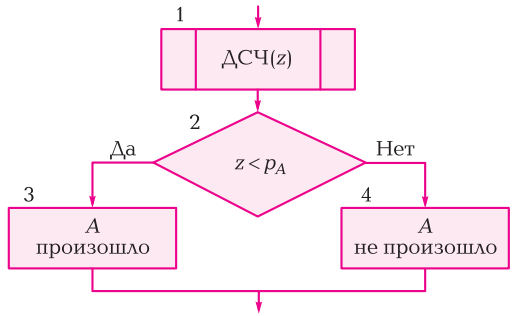


Рис. 5.2. Алгоритм моделирования простого события

Таким образом, вероятность попадания случайной величины z в интервал [0; p_A) равна величине p_A. Поэтому если число z, полученное с помощью датчика равномерно распределенных случайных чисел, попало в этот интервал, то следует считать, что событие A произошло. Противоположное событие (не A) произойдет с вероятностью $(1 - p_A)$, т. е. в том случае, если $z \geq p_A$.

Процедура моделирования простого события в имитационной модели описывается алгоритмом, схема которого показана на рис. 5.2.

На данном рисунке в блоке 1 осуществляется обращение к датчику случайных чисел ДСЧ, генерирующему случайную величину z. Блок 2 проверяет условие $z < p_A$. Если оно выполняется, считается, что произошло событие A. В противном случае считается, что произошло противоположное событие (не A).

Программа на языке C#, использующая данный алгоритм для имитации события A, выглядит следующим образом:

```
static void Main(string[] args)
{
   Console.Write("Количество экспериментов N: ");
   int n = int.Parse(Console.ReadLine());
   Console.Write("Вероятность появления события A: ");
   double pA = double.Parse(Console.ReadLine());
   Random random = new Random();
   for (int i = 0; i < n; i++)
   {
      double r = random.NextDouble();
      if (r < pA)
```

```
        {
            … //действия при появлении события A
        }
        else
        {
            … //действия при появлении события не A
        }
    }
    Console.WriteLine("Моделирование завершено");
    Console.ReadLine();
}
```

5.7. МОДЕЛИРОВАНИЕ ПОЛНОЙ ГРУППЫ НЕСОВМЕСТНЫХ СОБЫТИЙ

Пусть имеется полная группа несовместных событий A_1, A_2, ..., A_k. Данный термин означает, что никакие два и более события не могут происходить одновременно (т. е. в одном испытании), но в каждом испытании обязательно происходит одно из них. Предположим, что вероятности появления событий A_1, A_2, ..., A_k равны соответственно p_1, p_2, ..., p_k. При этом выполняется условие

$$\sum_{i=1}^{k} p_i = 1.$$

Для моделирования полной группы несовместных событий разделим интервал [0; 1) на k отрезков, длины которых составляют p_1, p_2, ..., p_k (рис. 5.3).

Если случайное число z, сгенерированное датчиком случайных чисел с равномерным распределением в интервале [0; 1), попало, например, на участок p_i, то это должно означать, что произошло событие A_i.

Тогда процедура моделирования полной группы несовместных событий описывается алгоритмом, схема которого показана на рис. 5.4. Здесь $L_i = \sum_{i=1}^{L} p_j$.

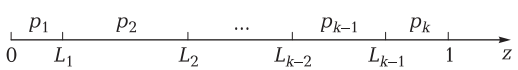


Рис. 5.3. Иллюстрация разбиения на подынтервалы

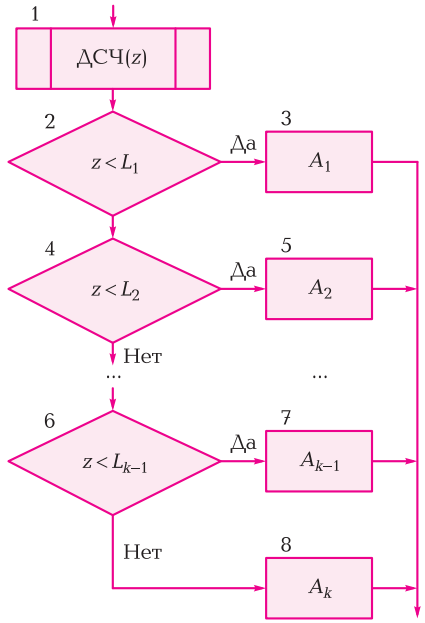


Рис. 5.4. Алгоритм моделирования полной группы несовместных событий

В блоке 1 происходит обращение к датчику случайных чисел с равномерным распределением в интервале [0; 1). Условный оператор 2 проверяет условие попадания случайной величины z в интервал $[0; L_1)$. Если это условие выполняется, то считается, что произошло событие A_1. Если условие в операторе 2 не выполняется, то алгоритм осуществляет проверку условий попадания случайной величины в другие интервалы. Одно из событий A_1, A_2, ..., A_k обязательно произойдет.

Пример. Пусть при попадании в мишень в тире можно набрать от 0 до 5 очков. Вероятность p того, что стрелок одним выстрелом наберет k очков, задана далее.

k	0	1	2	3	4	5
p	0,1	0,15	0,2	0,25	0,2	0,1

Смоделировать серию из пяти выстрелов и оценить вероятность того, что стрелок за пять выстрелов наберет более 15 очков.

Решение. Программа на языке C#, разработанная в соответствии со схемой алгоритма, представленной на рис. 5.4, имеет следующий вид:

```
static void Main(string[] args)
{
   Console.Write("Количество экспериментов N: ");
   int n = int.Parse(Console.ReadLine());
   double[] L = new double[]
      {// задаем границы интервалов разбиения
      0.1, 0.25, 0.45, 0.7, 0.9, 1.0
      };
   Random random = new Random();
   int m = 0;
   for (int i = 0; i < n; i++)
   {
      int si = 0;
      for (int j = 0; j < 5; j++)
      {
         double r = random.NextDouble();
         if (r < L[0]) si += 0;
         else if (r < L[1]) si++;
         else if (r < L[2]) si += 2;
         else if (r < L[3]) si += 3;
         else if (r < L[4]) si += 4;
         else si += 5;
      }
      if (si > 15) m++;
   }
   Console.WriteLine("Искомая вероятность p = {0:f4}",
   (float) m / n);
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

5.8. МОДЕЛИРОВАНИЕ СЛОЖНЫХ СОБЫТИЙ

Результатом испытания может быть появление нескольких, следующих друг за другом событий. Рассмотрим случай, когда происходят два *независимых* события A и B с вероятностями p_A и p_B. Возможные исходы испытаний в этом случае AB, $A\bar{B}$, $\bar{A}B$ и $\bar{A}\bar{B}$. События AB, $A\bar{B}$ $\bar{A}B$ и $\bar{A}\bar{B}$ образуют полную группу несовместных событий с вероятностями

$$P(AB) = p_A p_B;$$

$$P(A\bar{B}) = p_A(1 - p_B);$$

$$P(\bar{A}B) = (1 - p_A)p_B;$$

$$P(\bar{A}\bar{B}) = (1 - p_A)(1 - p_B). \qquad (5.1)$$

Для моделирования совместных испытаний могут быть использованы два варианта процедуры.

Первый вариант можно построить по аналогии с полной группой несовместных событий (алгоритм на рис. 5.4) как определение одного из исходов в соответствии с вероятностями (5.1). В этом случае можно ограничиться одним случайным числом, однако сравнений в общем случае может потребоваться больше.

Второй вариант состоит в последовательной проверке условия, аналогичного моделированию простого события с заданной вероятностью (алгоритм на рис. 5.2) относительно событий A, B, и требует использования двух случайных чисел и двух сравнений. В среднем второй вариант оказывается более экономным, чем первый.

Теперь рассмотрим случай, когда события A и B являются *зависимыми* и известны: p_A, p_B и $p_{B/A}$.

Для первого варианта процедуры моделирования еще раз заметим, что события AB, $A\bar{B}$ $\bar{A}B$ и $\bar{A}\bar{B}$ составляют полную группу. Их вероятности можно найти следующим образом:

$$P(AB) = p_A p_{B/A};$$

$$P(A\bar{B}) = p_A(1 - p_{B/A});$$

$$P(\bar{A}B) = (1 - p_A)p_{B/\bar{A}};$$

$$P(\bar{A}\bar{B}) = (1 - p_A)(1 - p_{B/\bar{A}}),$$

где

$$p_{B/\bar{A}} = \frac{p_B - p_A p_{B/A}}{1 - p_A}$$

определяется из формулы полной вероятности

$$p_B = p_A p_{B/A} + p_{\bar{A}} p_{B/\bar{A}}.$$

Второй вариант процедуры в этом случае выглядит следующим образом. Из последовательности равномерно распределенных чисел $\{z_i\}$ извлекается очередное число z_k и проверяется справедливость неравенства

$$z_k < p_A. \qquad (5.2)$$

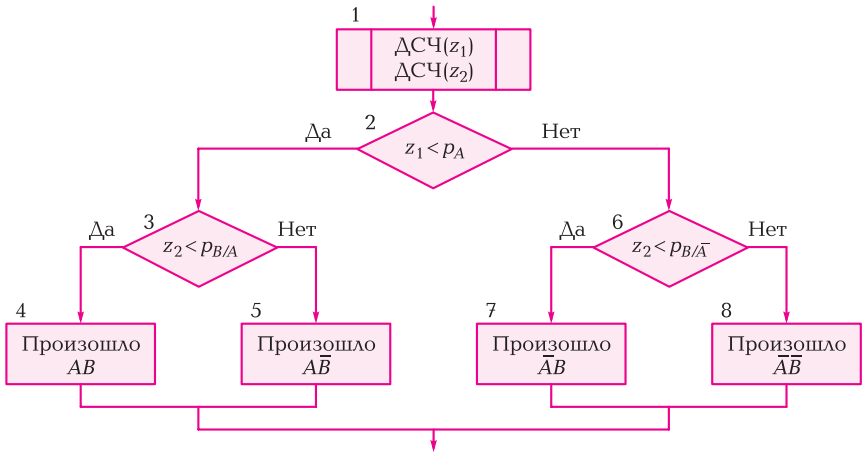


Рис. 5.5. Алгоритм моделирования зависимых событий

Если оно оказалось справедливым, то наступило событие A. Из совокупности $\{z_i\}$ берется очередное число z_{k+1} и проверяется условие

$$z_{k+1} < p_{B/A}.$$

В зависимости от его справедливости исходом испытания является AB или $A\bar{B}$.

Если неравенство (5.2) не выполняется, то это значит, что наступило событие $\bar{A}$. Поэтому для испытания, связанного с событием B, необходимо использовать вероятность $p_{B/\bar{A}}$. Выберем из совокупности $\{z_i\}$ число z_{k+1} и проверим справедливость неравенства

$$z_{k+1} < p_{B/\bar{A}}.$$

В зависимости от того, справедливо или нет это неравенство, получим в результате испытания $\bar{A}B$ или $\bar{A}\bar{B}$.

Алгоритм второго варианта процедуры имитации двух зависимых событий для одного испытания изображен на блок-схеме на рис. 5.5.

Следует отметить, что если при моделировании требуется имитировать возникновение большого числа зависимых событий, то второй вариант процедуры имитации оказывается более простым для реализации.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем состоит отличие между дискретно и непрерывно распределенными случайными величинами?
2. Каким образом осуществляется моделирование дискретной случайной величины, заданной рядом распределения?
3. Каким образом задается распределение Бернулли? Приведите примеры случайных величин, для описания которых можно использовать распределение Бернулли. Как их моделировать?
4. Каким образом задается биномиальное распределение? Приведите примеры случайных величин, для описания которых можно использовать биномиальное распределение. Как их моделировать?
5. Каким образом задается геометрическое распределение? Приведите примеры случайных величин, для описания которых можно использовать геометрическое распределение. Как их моделировать?
6. Каким образом задается распределение Пуассона? Приведите примеры случайных величин, для описания которых можно использовать распределение Пуассона. Как их моделировать?
7. Каким образом осуществляется моделирование полной группы несовместных событий?
8. Какие события называются зависимыми? Как их моделировать?

Глава 6

МОДЕЛИРОВАНИЕ С ПОМОЩЬЮ МЕТОДА СТАТИСТИЧЕСКИХ ИСПЫТАНИЙ

6.1. МЕТОД МОНТЕ-КАРЛО

В процессе построения и реализации имитационных моделей стохастических систем широко используется метод статистических испытаний (Монте-Карло). *Метод Монте-Карло* — общее название группы численных методов, основанных на получении большого числа реализаций стохастического (случайного) процесса, который формируется таким образом, чтобы его вероятностные характеристики совпадали с аналогичными величинами решаемой задачи. Метод часто используется для решения задач в различных областях физики, химии, математики, экономики, оптимизации, теории управления и др.

Идея метода Монте-Карло заключается в следующем. Вместо того чтобы описывать исследуемый случайный процесс аналитически, составляется алгоритм, имитирующий этот процесс. В алгоритм включаются специальные процедуры для моделирования случайности. Конкретное осуществление алгоритма складывается каждый раз по-иному, со своими результатами. Обработав результаты множества реализаций алгоритма методами математической статистики, можно получить любые характеристики: вероятности событий, матожидания, дисперсии случайных величин. Вероятность того, что эти характеристики отличаются от истинных не более чем на заданную величину, является функцией количества испытаний.

Обобщенная схема алгоритма, реализующего метод Монте-Карло, представлена на рис. 6.1. На первом этапе работы алгоритма выполняется инициализация статистических переменных. В данных переменных, например при оценке вероятности наступления некоторого события, будет накапливаться число экспериментов, в кото-

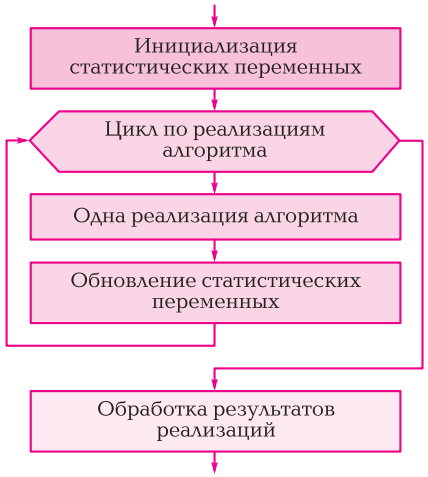


Рис. 6.1. Обобщенный алгоритм моделирования по методу Монте-Карло

рых интересующее нас событие произошло. Затем организуется цикл, в теле которого выполняются отдельные реализации алгоритма (например, выполняется моделирование события) и обновляются статистические переменные (например, если событие произошло, то соответствующая статистическая переменная увеличивается на 1). В последнем блоке алгоритма выполняется обработка результатов моделирования. При этом, например, может оцениваться вероятность интересующего нас события.

Поясним сущность метода статистического моделирования следующими примерами.

Пример. Найти значение числа π.

Решение. Квадрат со стороной r, например, нарисованный на листе бумаги, показан на рис. 6.2. Его площадь $S_1 = r^2$. В квадрат вписана четверть круга радиуса r площадью $S_2 = \pi r^2/4$. Отношение площадей четверти круга и квадрата равно $S_2/S_1 = \pi/4$.

Это отношение, а следовательно число π, можно приближенно получить, проделав следующие статистические испытания. Будем бросать на лежащий на столе лист бумаги мелкие крупинки так, чтобы они равномерно рассыпались по поверхности листа случайным образом. Крупинки вне квадрата учитывать не будем. Подсчитаем N_1 — число крупинок, попавших в квадрат, и N_2 — число крупинок в пределах четверти круга. Поскольку крупинки имеют одинаковую вероятность попасть в любой участок рисунка, то отношение N_2/N_1 при достаточно большом числе бросаний будет приближенно равным отношению площадей, т. е. $\pi/4$.

Можно немного изменить процесс нахождения числа π. Для этого проведем вдоль сторон квадрата координатные оси. Масштаб выберем таким, чтобы сторона квадрата равнялась единице. Вместо крупинок будем наносить на этот квадрат точки со случайными координатами (x, y). Под случайными координатами будем понимать числа с равномерным распределением в единичном отрезке. Вероятность того, что точка окажется внутри круга, равна отношению площадей $\pi/4$. Составим алгоритм для нахождения числа π. Одно испытание (одна реализация) состоит из проверки неравенства $x^2 + y^2 < 1$ для координат x и y на истинность (рис. 6.3).

Повторим его N раз, тогда N_2 — количество точек, для которых неравенство выполняется, и N_2/N — частота этого события, зная которую можно найти значение числа π.

Результаты расчетов числа π, полученные с помощью программы, реализующей описанный алгоритм, при различном числе точек N представлены в табл. 6.1. Отметим, что с помощью метода Монте-Карло можно получить лишь приближенное значение числа π, и чем больше экспериментов выполняется, тем меньше погрешность получаемых результатов. Дополнительно отметим, что на результаты моделирования существенное влияние оказывает качество генератора случайных чисел. Это определяется тем, что если точки будут размещаться в пределах квадрата неравномерно, то и результаты моделирования будут неправильными даже при большом числе экспериментов.

Пример. Происходит дуэль между участниками A и B. Участник A стреляет первым с вероятностью $p = 0{,}9$. При стрельбе участник A попадает в B с вероятностью $p_A = 0{,}5$, а участник B в A с вероятностью $p_B = 0{,}8$. Участники делают не более одного выстрела. Оценить вероятность победы участника A, B и вероятность ничьей.

Решение. Поскольку данная задача достаточно простая, то можно выполнить аналитический расчет требуемых вероятностей:

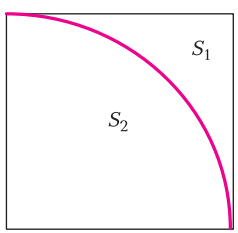


Рис. 6.2. Иллюстрация для нахождения числа π

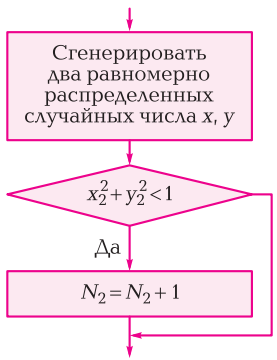


Рис. 6.3. Реализация проверки неравенства

Таблица 6.1. Результаты экспериментов по определению значения числа π

Количество точек (N)	Значение π
100	3,2000
1 000	3,1120
10 000	3,1140
100 000	3,1374
1 000 000	3,1391
10 000 000	3,1409

$$P(A) = pp_A + (1-p)(1-p_B)p_A = 0{,}46;$$

$$P(B) = p(1-p_A)p_B + (1-p)p_B = 0{,}44;$$

$$P(H) = (1-p_A)(1-p_B) = 0{,}1.$$

Эти результаты можно использовать для проверки правильности результатов моделирования.

При моделировании нам нужно много раз разыграть дуэль между участниками, моделируя отдельные события дуэли (выбор первого стреляющего, стрельба первого стреляющего, стрельба второго стреляющего) и определяя тип результата (победа участника A, участника B или ничья).

После выполнения ряда экспериментов подсчитывается число событий каждого типа, и затем оцениваются вероятности соответствующих событий.

Обобщенная схема алгоритма решения задачи была приведена на рис. 6.1. Алгоритм розыгрыша одного эксперимента (одной дуэли) представлен на рис. 6.4. В данном алгоритме используются три статистических счетчика (N_A, N_B, N_H), обнуляемых на этапе инициализации статистических переменных и используемых для подсчета числа побед участника A, участника B и числа ничьих.

В начале работы данного алгоритма необходимо смоделировать событие, заключающееся в выборе первого стреляющего. Алгоритм моделирования такого простого события был рассмотрен в подразд. 5.6. В том случае, если первым стреляет участник A, моделируется его стрельба. Для этого разыгрывается событие, исходом которого будет промах (с вероятностью $1 - p_A$) или попадание (с вероятностью p_A) участника A. Если он попал, то в разыгрываемой дуэли победил участник A и мы увеличиваем на единицу число его побед.

Если же игрок A промахнулся, то необходимо смоделировать стрельбу игрока B. Если он попал, то увеличиваем число побед участника B. В противном случае произошла ничья, и мы увеличиваем на единицу счетчик

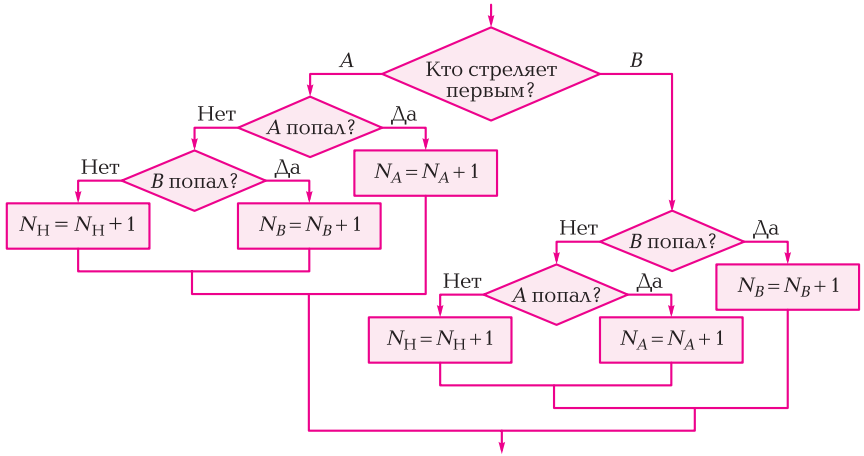


Рис. 6.4. Алгоритм розыгрыша одного эксперимента

N_H. Похожим образом развивается ситуация в том случае, если первым стреляет участник *B*.

После розыгрыша *N* дуэлей можно оценить вероятность искомых событий с помощью следующих выражений:

$$P(A) = N_A/N;$$

$$P(B) = N_B/N;$$

$$P(H) = N_H/N.$$

Выполним моделирование рассматриваемой системы вручную. Полученные результаты по 20 экспериментам (дуэлям), случайные числа для которых формировались с помощью функции СЛЧИС() MS Excel, представлены в табл. 6.2.

Для оценки вероятности победы *A* считаем, сколько раз он победил (в данном случае 7), и делим полученную сумму на общее число экспериментов (20). В результате получаем, что оценка вероятности победы участника *A* равна 0,35. Похожим образом можно оценить вероятность победы участника *B* (получаем 0,50) и вероятность ничьей (получаем 0,15). Данные оценки похожи на точные значения вероятностей, однако погрешность полученных результатов из-за небольшого объема эксперимента очень высока.

Для уменьшения погрешности моделирования, как будет показано в гл. 8, следует увеличить объем эксперимента.

Пример программы на языке C#, реализующей оценку интересующих нас вероятностей с помощью метода Монте-Карло, имеет следующий вид:

Таблица 6.2. Пример моделирования методом Монте-Карло

Номер эксперимента	Случайное число	Кто стреляет первым	Случайное число	Результат	Случайное число	Результат	Кто победил
1	0,3216	*A*	0,3401	Попал	—	—	*A*
2	0,1922	*A*	0,3069	Попал	—	—	*A*
3	0,6360	*A*	0,5904	Промах	0,7010	Попал	*B*
4	0,6234	*A*	0,7095	Промах	0,9393	Промах	Ничья
5	0,0983	*A*	0,5505	Промах	0,7170	Попал	*B*
6	0,1811	*A*	0,6871	Промах	0,0626	Попал	*B*
7	0,0562	*A*	0,6631	Промах	0,6491	Попал	*B*
8	0,3536	*A*	0,4897	Попал	—	—	*A*
9	0,4534	*A*	0,0713	Попал	—	—	*A*
10	0,4586	*A*	0,5352	Промах	0,0795	Попал	*B*
11	0,2774	*A*	0,2644	Попал	—	—	*A*

Окончание табл. 6.2

Номер эксперимента	Случайное число	Кто стреляет первым	Случайное число	Результат	Случайное число	Результат	Кто победил
12	0,0836	*A*	0,6348	Промах	0,8685	Промах	Ничья
13	0,8119	*A*	0,5507	Промах	0,3903	Попал	*B*
14	0,1563	*A*	0,4240	Попал	—	—	*A*
15	0,1373	*A*	0,6166	Промах	0,0428	Попал	*B*
16	0,5001	*A*	0,0578	Попал	—	—	*A*
17	0,9247	*B*	0,8491	Промах	0,4729	Попал	*B*
18	0,4176	*A*	0,7501	Промах	0,9962	Промах	Ничья
19	0,5981	*A*	0,7187	Промах	0,0926	Попал	*B*
20	0,4908	*A*	0,5251	Промах	0,6681	Попал	*B*


```
static void Main(string[] args)
{
   Console.Write("Вероятность того, что игрок A стреляет
   первым: ");
   double p = double.Parse(Console.ReadLine());
   Console.Write("Вероятность попадания игрока A: ");
   double pA = double.Parse(Console.ReadLine());
   Console.Write("Вероятность попадания игрока B: ");
   double pB = double.Parse(Console.ReadLine());
   Console.Write("Количество экспериментов N: ");
   int n = int.Parse(Console.ReadLine());
   Random random = new Random();
   int nA = 0, nB = 0, nH = 0;
   double r;
   for (int i = 0; i < n; i++)
   {
      r = random.NextDouble();
      if (r < p)
      { // игрок A стреляет первым
         r = random.NextDouble();
         if (r < pA) nA++;
         else
         {
            r = random.NextDouble();
            if (r < pB) nB++;
            else nH++;
         }
      }
      else
      { // игрок B стреляет первым
         r = random.NextDouble();
         if (r < pB) nB++;
         else
         {
            r = random.NextDouble();
            if (r < pA) nA++;
            else nH++;
         }
      }
   }
   Console.WriteLine("Искомые  вероятности");
   Console.WriteLine("pA = {0:f4}",(double)nA/n);
   Console.WriteLine("pB = {0:f4}",(double)nB/n);
   Console.WriteLine("pH = {0:f4}",(double)nH/n);
   Console.WriteLine("Моделирование завершено");
   Console.ReadLine();
}
```

6.2. ОДНОМЕРНОЕ СЛУЧАЙНОЕ БЛУЖДАНИЕ

Рассмотрим задачи о случайных блужданиях, которые часто возникают при исследовании различных процессов в физике, биологии, химии, экономике и многих других науках. Подобные задачи можно решать с помощью метода статистических испытаний.

Наиболее простой является одномерная задача по движению пьяного пешехода, который начинает движение от фонарного столба, расположенного в точке $x = 0$ (рис. 6.5). Все шаги, совершаемые пешеходом, имеют одинаковую длину l. Направление каждого шага пешехода не зависит от направления предыдущего. Пешеход делает шаг вправо с вероятностью p и шаг влево с вероятностью $q = 1 - p$. В данной задаче необходимо определить вероятность того, что после M шагов пешеход окажется на расстоянии x от фонарного столба, или среднее смещение пешехода от начальной точки после M шагов.

Исследование задачи одномерного случайного блуждания можно провести аналитически, воспользовавшись теорией вероятности. Полученные при этом результаты могут быть использованы в процессе отладки имитационной модели и проверки ее правильности. Среднее смещение μ и дисперсию σ_x^2 можно рассчитать с помощью следующих аналитических выражений:

$$\mu = (p - q)Ml;$$
$$\sigma_x^2 = 4pqMl^2.$$

Далее приведен фрагмент программы на языке C# моделирования методом Монте-Карло одномерного случайного блуждания, в результате которого оцениваются матожидание и дисперсия смещения L пешехода от начальной точки. Для получения результата выполняется ряд экспериментов, в каждом из которых имитируется движение одного пешехода и определяется его смещение от фонарного столба после m шагов. Результаты проведенных экспериментов затем усредняются. Заметим, что в данном случае также получаются лишь приближенные результаты, точность которых определяется числом экспериментов.

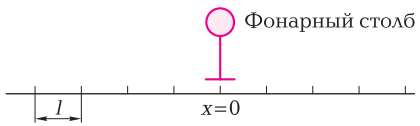


Рис. 6.5. Иллюстрация для задачи по движению пьяного пешехода


```
…
// n — число испытаний
// p — вероятность шага вправо
// m — число шагов
Random random = new Random();
x1 = 0.0;
x2 = 0.0;
for (int i = 0; i < n; i++)
{
   x = 0.0;
   for (int j = 0; j < m; j++)
   {
      double r = random.NextDouble();
      if (r < p) x++;
      else x--;
   }
   x1 += x;
   x2 += x * x;
}
L = x1 / n;
D = x2 / n — L * L;
…
```

6.3. ДВУМЕРНОЕ СЛУЧАЙНОЕ БЛУЖДАНИЕ

Более сложным является блуждание в многомерном пространстве. Например, при двумерном случайном блуждании на прямоугольной решетке на каждом шаге по времени осуществляется движение случайным образом равновероятно в одном из четырех возможных направлений: на север, юг, восток или запад (рис. 6.6). Движение начинается из начальной точки H, обычно совпадающей с началом координат ($x = 0$; $y = 0$), и заканчивается через M шагов в точке K. В данном случае часто требуется определить удаление точки K, полученной после M шагов, от начала координат.

Отметим, что решетки могут иметь и другую форму. Например, встречаются задачи случайного блуждания на треугольной решетке. При таком блуждании на каждом шаге движение осуществля-

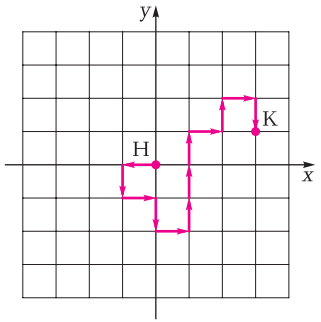


Рис. 6.6. Иллюстрация для задачи двумерного случайного блуждания

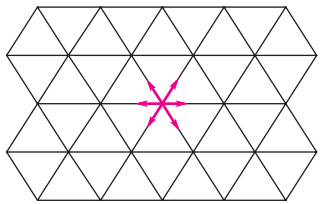


Рис. 6.7. Иллюстрация для задачи случайного блуждания на треугольной решетке

ется равновероятно в одном из шести возможных направлений (рис. 6.7).

Известна задача случайного блуждания на сотах. При таком блуждании на каждом шаге движение осуществляется равновероятно в одном из трех возможных направлений (рис. 6.8).

Фрагмент программы на языке C#, имитирующей двумерное случайное блуждание на прямоугольной решетке и определяющей среднее удаление L точки K от начала координат после выполнения m шагов, выглядит следующим образом:

```
...
// n – число испытаний
// m – число шагов
Random random = new Random();
double L = 0;
for (int i = 0; i < n; i++)
{
  x = y = 0;
```

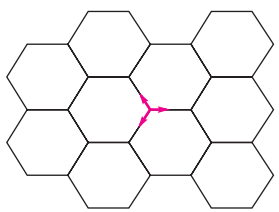


Рис. 6.8. Иллюстрация для задачи блуждания на сотах


```
  for (int j = 0; j < m; j++)
  {
    double r = random.NextDouble();
    if (r < 0.25) x++;
    else if (r < 0.5) x--;
    else if (r < 0.75) y--;
    else y++;
  }
  L += Math.Sqrt(x * x + y * y);
}
L /= n;
...
```

6.4. ПРОСТОЕ СЛУЧАЙНОЕ БЛУЖДАНИЕ С ПОГЛОЩАЮЩИМИ ЭКРАНАМИ

В ранее рассмотренных задачах область блуждания не была ничем ограничена, т.е. пешеход (частица) мог удалиться на любое расстояние от начальной точки. В ряде практических задач такие ограничения есть. Примером является классическая задача о разорении игрока. В ней два человека играют в кости, имея начальный капитал a и b руб. После каждого броска игральной кости один из игроков выигрывает 1 руб., а второй проигрывает 1 руб. Необходимо узнать, как долго в среднем они будут играть, пока капитал одного из них не исчерпается. Данная задача представляет собой простое случайное блуждание с поглощающими экранами.

Пусть задана одномерная решетка, у которой имеются «поглощающие» узлы (экраны) в точках $x = a$ и $x = b$ ($b > a$). Движение частицы начинается из точки x_0 ($a < x_0 < b$). На каждом шаге частица перемещается влево с вероятностью p и вправо с вероятностью $q = 1 - p$. Величина шага равна 1. При достижении частицей любого из экранов она поглощается. Необходимо узнать среднее время блуждания частицы.

Фрагмент программы на языке C# моделирования данной ситуации с определением среднего времени блуждания T имеет следующий вид:

```
// n – число испытаний
// a, b – положения экранов
// x0 – начальное положение
// p – вероятность шага вправо
```

```
Random random = new Random();
double T = 0;
for (int i = 0; i < n; i++)
{
   double x = x0;
   int time = 0;
   while ((x != a) && (x != b))
   {
      double r = random.NextDouble();
      if (r < p) x++;
      else x--;
      time++;
   }
   T += time;
}
T /= n;
…
```

6.5. МОДЕЛЬ ПАДЕНИЯ ДОЖДЕВОЙ КАПЛИ

Еще одним вариантом модели случайных блужданий с ограничениями является модель падения дождевой капли. При воздействии случайных порывов легкого ветра падение дождевой капли можно моделировать случайным блужданием на квадратной решетке (рис. 6.9). Движение начинается с узла, расположенного на расстоянии h от горизонтальной линии (над поверхностью земли). Вероятности скачков вниз, вверх, влево и вправо равны $p_{\downarrow}$, $p_{\uparrow}$, $p_{\leftarrow}$, $p_{\rightarrow}$. При этом вероятность $p_{\downarrow}$ шага вниз больше вероятности $p_{\uparrow}$ шага вверх. В таком случае после выполнения некоторого числа шагов частица достигнет поверхности земли. Необходимо оценить среднее время падения капли.

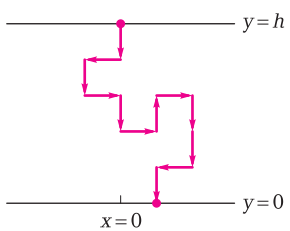


Рис. 6.9. Иллюстрация для задачи о падении дождевой капли

Фрагмент программы на языке C# моделирования данной ситуации с определением среднего времени падения T имеет следующий вид:

```
…
// n — число испытаний
// pDown, pUp, pLeft, pRight — вероятности шага вниз,
//вверх, влево и вправо
// h — высота
Random random = new Random();
double T = 0;
for (int i = 0; i < n; i++)
{
   double y = h;
   double x = 0;
   int time = 0;
   while (y != 0)
   {
      double r = random.NextDouble();
      if (r < pDown) y--;
      else if (r < pDown + pUp) y++;
         else if (r < pDown + pUp + pLeft) x--;
            else x++;
      time++;
   }
   T += time;
}
T /= n;
…
```

6.6. ПЕРСИСТЕНТНОЕ СЛУЧАЙНОЕ БЛУЖДАНИЕ

В ранее рассмотренных моделях случайных блужданий направление следующего шага никак не было связано с направлением предыдущего шага. В персистентном случайном блуждании вероятность перехода (или скачка) зависит от последнего перехода.

При персистентном одномерном случайном блуждании шаги совершаются только в ближайшие соседние узлы. Предположим, что сделано $k - 1$ шагов. Далее k-й шаг делается в том же направ-

лении с вероятностью α, а шаг в противоположном направлении делается с вероятностью 1 – α. Необходимо определить среднее удаление частицы от исходного положения за *M* шагов.

Фрагмент программы на языке C# моделирования данной ситуации с определением среднего удаления частицы *L* от исходного положения имеет следующий вид:

```
…
// n — число испытаний
// alpha — вероятность шага в том же направлении
// M — число шагов
Random random = new Random();
double r = random.NextDouble();
int dir;// направление движения
//выбираем случайное начальное направление движения
if (r < 0.5) dir = 1;
else dir = -1;
double x;
double L = 0.0;
for (int i = 0; i < n; i++)
{
  x = 0.0;
  //выполняем M шагов
  for (int j = 0; j < M; j++)
  {
    r = random.NextDouble();
    if (r > = alpha) dir = -dir;//меняем направление
    x += dir;//делаем шаг в нужном направлении
  }
  L += Math.Abs(x);
}
L /= n;
…
```

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В чем состоит сущность метода Монте-Карло?
2. От чего зависит точность результатов, полученных с помощью метода Монте-Карло?
3. Каким образом осуществляется моделирование одномерного случайного блуждания?
4. Каким образом осуществляется моделирование двумерного случайного блуждания?
5. Каким образом осуществляется моделирование простого случайного блуждания с поглощающими экранами?
6. Каким образом осуществляется моделирование персистентного случайного блуждания?

Глава 7

МОДЕЛИРОВАНИЕ СИСТЕМ МАССОВОГО ОБСЛУЖИВАНИЯ

7.1. ОСНОВНЫЕ ХАРАКТЕРИСТИКИ СИСТЕМ МАССОВОГО ОБСЛУЖИВАНИЯ

Достаточно часто при моделировании исследователь сталкивается с задачей определения характеристик систем, являющихся системами массового обслуживания. *Системы массового обслуживания* (СМО) — это такие системы (рис. 7.1), в которые в случайные моменты времени поступают заявки или требования на обслуживание. При этом поступившие заявки обслуживаются с помощью имеющихся в распоряжении системы каналов обслуживания.

Примерами СМО могут служить кассы в банке и магазине, персональные компьютеры, решающие вычислительные задачи, сервисные центры, автозаправочные станции (АЗС), телефонные станции, сборочные конвейеры на заводах и т. п.

В процессе функционирования в СМО поступают заявки на обслуживание. Поступив в СМО, требование присоединяется к очереди ранее поступивших требований. Канал обслуживания в соответствии с некоторым правилом выбирает требование из очереди и приступает к его обслуживанию. После завершения процедуры

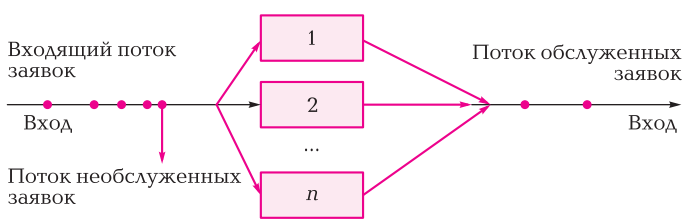


Рис. 7.1. Графическое представление типичной СМО

обслуживания очередного требования канал обслуживания приступает к обслуживанию следующего требования, если таковое имеется в очереди. Данный процесс повторяется многократно в течение всего периода работы обслуживающей системы.

Основными компонентами СМО любого вида являются [1]:

- входной поток поступающих требований или заявок на обслуживание;
- дисциплина очереди;
- механизм обслуживания.

Входной поток заявок определяется законом распределения случайной величины, обычно задающей интервал времени между поступлением требований и числом заявок в каждом требовании. Обозначим через A_i время между поступлением требования $(i - 1)$ и требования i. Тогда величина $\lambda = 1/E(A)$ называется интенсивностью поступления требований. Здесь $E(A)$ — среднее время между поступлением требований.

Дисциплина очереди определяется правилом, которое устройство обслуживания использует при выборе из очереди очередного требования для обслуживания. Чаще всего используются типичные дисциплины очереди, определяемые следующими правилами:

- первым пришел — первый обслуживаешься (First In First Out — FIFO);
- пришел последним — обслуживаешься первым (Last In First Out — LIFO);
- случайный отбор заявок;
- отбор заявок по критерию приоритетности.

Механизм обслуживания определяется структурой обслуживающей системы и характеристиками процедуры обслуживания. Структура обслуживающей системы обусловливается тем, сколько каналов обслуживания присутствует в СМО, одинаковые эти каналы или нет, как эти каналы взаимодействуют друг с другом. Процедура обслуживания обычно описывается вероятностным законом, определяющим время обслуживания заявок. Обозначим через S_i время обслуживания i-го требования. Тогда величина $\mu = 1/E(S)$ называется интенсивностью обслуживания требований. Здесь $E(S)$ — среднее время обслуживания требований.

В качестве характеристик эффективности функционирования СМО можно выбрать три основные группы (обычно средних) показателей [1].

1. Показатели эффективности использования СМО:

1.1. Абсолютная пропускная способность СМО — среднее число заявок, которое сможет обслужить СМО в единицу времени.

1.2. Относительная пропускная способность СМО — отношение среднего числа заявок, обслуживаемых СМО в единицу времени, к среднему числу поступивших заявок за это же время.

1.3. Коэффициент использования СМО — средняя доля времени, в течение которого СМО занята обслуживанием заявок, и т. п.

2. Показатели качества обслуживания заявок:

2.1. Среднее время ожидания заявки в очереди.

2.2. Среднее время пребывания заявки в СМО.

2.3. Вероятность отказа заявке в обслуживании без ожидания.

2.4. Вероятность того, что поступившая заявка немедленно будет принята к обслуживанию.

2.5. Среднее число заявок, находящихся в очереди.

2.6. Среднее число заявок, находящихся в СМО, и т. п.

3. Показатели эффективности функционирования пары «СМО — потребитель», где под «потребителем» понимают всю совокупность заявок или некий их источник (например, средний доход, приносимый СМО в единицу времени).

Рассмотрим основные подходы к классификации СМО [1].

1. По характеру потока заявок (требований) и длительности обслуживания различают *марковские* и *немарковские* СМО. В марковских системах входящий поток требований и выходящий поток обслуженных требований (заявок) являются *пуассоновскими*. Для такого потока число требований k для любого интервала времени распределено по закону

$$P_k(t) = \frac{(\lambda t)^k}{k!} e^{-\lambda t},\ k \geq 0,\ t \geq 0,$$

где λ — интенсивность потока требований (число требований за единицу времени).

Пуассоновские потоки позволяют легко описать и построить математическую модель СМО, которая допускает достаточно простые аналитические решения. В случае немарковских процессов получение характеристик СМО значительно усложняется и обычно требует применения имитационного моделирования.

В том случае, если пуассоновский поток заявок (входной или выходной) является стационарным, т. е. не изменяющимся во времени (такие потоки также называются простейшими), то случайная величина, представляющая собой промежуток времени между любыми двумя соседними событиями (между приходом заявок или между началом и окончанием обслуживания заявки), распределена по показательному закону с плотностью вероятностей

$$f(t) = \lambda e^{-\lambda t},\ t \geq 0,$$

где λ — интенсивность потока.

2. В зависимости от дисциплины обслуживания выделяют три основных вида СМО:

- *системы с отказами*, в которых заявка, поступившая в систему в момент, когда все каналы заняты, получает отказ и сразу же покидает очередь;
- *системы с ожиданием* (очередью), в которых заявка, поступившая в момент, когда все каналы обслуживания заняты, становится в очередь и ждет, пока не освободится один из каналов. В данных системах заявка, стоящая в очереди, ждет обслуживание неограниченно долго, т. е. пока не подойдет очередь;
- *системы смешанного типа* (с ограниченным ожиданием), в которых на пребывание заявки в очереди накладываются некоторые ограничения. Например, может ограничиваться длина очереди или время пребывания в очереди.

3. В зависимости от числа каналов обслуживания выделяют *одноканальные* и *многоканальные* (n-канальные) системы. Обычно считают, что каждый канал одновременно может обслуживать только одну заявку и каждая заявка может обслуживаться только одним каналом. Многоканальные СМО могут состоять из однородных каналов, либо из разнородных, отличающихся длительностью обслуживания одной заявки.

4. По количеству этапов обслуживания все СМО можно подразделить на *однофазные* и *многофазные* системы. Если каналы СМО однородны, т. е. выполняют одну и ту же операцию обслуживания, то такие СМО называются однофазными. Если же каналы обслуживания расположены последовательно и они неоднородны, т. е. выполняют различные операции обслуживания, то СМО называется многофазной. Примером многофазной СМО может служить поликлиника, в которой сначала нужно получить талон к врачу в регистратуре и только потом пройти прием.

5. По месту нахождения источника требований СМО делятся на *разомкнутые*, когда источник находится вне системы, и *замкнутые*, когда источник находится в самой системе. К последнему виду относится, например, автобусный парк, в котором автобусы являются источником неисправностей, а следовательно, и требований на их обслуживание (ремонт).

При этом в замкнутых СМО, в отличие от разомкнутых, интенсивность потока требований зависит от числа требований, находящихся в данный момент в очереди или на обслуживании.

7.2. СИСТЕМЫ С ОДНИМ УСТРОЙСТВОМ ОБСЛУЖИВАНИЯ

Рассмотрим вопросы аналитической оценки характеристик простейших СМО, подробно изложенные в [1].

Сначала рассмотрим одноканальную (с одним устройством обслуживания) СМО, показанную на рис. 7.2. Пусть в данной СМО потоки заявок и обслуживания являются простейшими с интенсивностью λ и μ соответственно (т. е. заявки в СМО приходят в среднем через $1/\lambda$ и обслуживаются в среднем в течение $1/\mu$ единиц времени). Предположим также, что рассматриваемая СМО является системой с ограниченным ожиданием, причем ограничения накладываются на длину очереди (длина очереди $\le m$). Такая система при $m = 0$ превращается в СМО без ожидания, а при $m = \infty$ — в СМО с неограниченным ожиданием.

Приведем формулы для аналитической оценки наиболее важных показателей эффективности описанной СМО. Сразу заметим, что в том случае, если потоки заявок или обслуживания не будут простейшими, то многие из представленных далее выражений перестают быть справедливыми. Для небольшого числа других потоков заявок и обслуживания существуют выражения для оценки характеристик СМО. В более сложных случаях для этих целей необходимо применение компьютерного моделирования.

Коэффициент использования обсуждаемой СМО определяется так:

$$\rho = \frac{\lambda}{\mu}.$$

Выражения для определения вероятности наличия в СМО k заявок P_k имеют вид

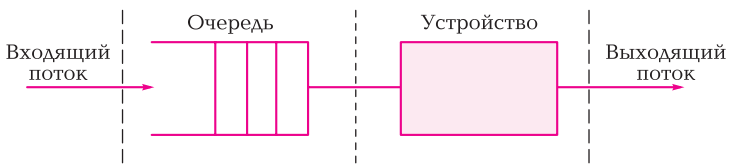


Рис. 7.2. Одноканальная СМО

$$P_0 = \begin{cases} \dfrac{1-\rho}{1-\rho^{m+2}}, & \text{если } \rho \neq 1; \\ \dfrac{1}{m+2}, & \text{если } \rho = 1, \end{cases}$$

$$P_k = \rho^k P_0;\ k = 1, \ldots, m+1.$$

Вероятность отказа в обслуживании, т. е. вероятность того, что при появлении заявки на обслуживание в системе уже будет находиться $m + 1$ заявка, определяется по формуле

$$P_{\text{отк}} = P_{m+1} = \begin{cases} \dfrac{\rho^{m+1}(1-\rho)}{1-\rho^{m+2}}, & \text{если } \rho \neq 1; \\ \dfrac{1}{m+2}, & \text{если } \rho = 1. \end{cases} \qquad (7.1)$$

Используя формулу (7.1), можно определить вероятность того, что заявка будет принята в систему:

$$P_{\text{сис}} = 1 - P_{\text{отк}}.$$

Относительная пропускная способность СМО

$$Q = P_{\text{сис}} = 1 - P_{\text{отк}} = \begin{cases} \dfrac{1-\rho^{m+1}}{1-\rho^{m+2}}, & \text{если } \rho \neq 1; \\ \dfrac{m+1}{m+2}, & \text{если } \rho = 1. \end{cases}$$

Абсолютная пропускная способность СМО

$$A = \lambda Q.$$

Среднее число заявок в очереди

$$\bar{N}_{\text{оч}} = \begin{cases} P_0 \dfrac{\rho^2\left[1-\rho^m(m+1-m\rho)\right]}{(1-\rho^{m+2})(1-\rho)}, & \text{если } \rho \neq 1; \\ \dfrac{m(m+1)}{2(m+2)}, & \text{если } \rho = 1. \end{cases}$$

Среднее число заявок, находящихся в процессе обслуживания,

$$\bar{N}_{\text{об}} = \rho Q = \begin{cases} \dfrac{\rho\left(1-\rho^{m+1}\right)}{1-\rho^{m+2}}, & \text{если } \rho \neq 1; \\ \dfrac{m+1}{m+2}, & \text{если } \rho = 1. \end{cases}$$

Среднее число заявок в СМО

$$\bar{N}_{\text{сис}} = \bar{N}_{\text{оч}} + \bar{N}_{\text{об}}.$$

Среднее время ожидания заявки в очереди

$$\bar{T}_{\text{оч}} = \frac{\bar{N}_{\text{оч}}}{\lambda P_{\text{сис}}}.$$

Среднее время пребывания заявки в системе

$$\bar{T}_{\text{сис}} = \frac{\bar{N}_{\text{сис}}}{\lambda P_{\text{сис}}}.$$

Напомним, что представленные ранее формулы применимы для оценки характеристик одноканальной СМО с простейшим потоком требований на обслуживание. Если поток заявок не простейший, то для получения характеристик СМО часто необходимо использовать компьютерное моделирование.

Рассмотрим пример расчета характеристик одноканальной СМО с ограниченным ожиданием.

Пример. Нотариальная контора представляет собой одноканальную СМО. Число мест в комнате ожидания очереди к нотариусу ограничено и равно двум. Если все места в комнате ожидания заняты, то вновь прибывший клиент в очередь не становится. Поток клиентов, прибывающий на консультацию, является простейшим с интенсивностью $\lambda = 8$ клиентов в час. Время обслуживания распределено по экспоненциальному закону со средним временем обслуживания $t = 7$ мин.

Определить вероятностные характеристики нотариальной конторы, работающей в стационарном режиме.

Решение. 1. Сначала определим интенсивность потока обслуживания клиентов. Результат получим в такой же размерности, что и λ:

$$\mu = \frac{1}{\bar{t}} = \frac{1}{7/60} = 8{,}571.$$

2. Коэффициент использования СМО рассчитаем как отношение интенсивностей λ и μ:

$$\rho = \frac{\lambda}{\mu} = \frac{8}{8{,}571} = 0{,}933.$$

3. Вычислим вероятности наличия в СМО k клиентов:

$$P_0 = \frac{1-\rho}{1-\rho^{m+2}} = \frac{1-0{,}933}{1-0{,}933^4} \approx 0{,}276;$$

$$P_1 = \rho^1 P_0 = 0{,}933 \cdot 0{,}276 \approx 0{,}258;$$

$$P_2 = \rho^2 P_0 = 0{,}933^2 \cdot 0{,}276 \approx 0{,}241;$$

$$P_3 = \rho^3 P_0 = 0{,}933^3 \cdot 0{,}276 \approx 0{,}225.$$

4. Вероятность отказа в обслуживании клиента:

$P_{\text{отк}} = P_{m+1} = 0{,}225.$

5. Относительная пропускная способность нотариальной конторы:

$$Q = P_{\text{сис}} = 1 - P_{\text{отк}} = 1 - 0{,}225 = 0{,}775.$$

6. Абсолютная пропускная способность нотариальной конторы, клиентов в час:

$$A = \lambda \cdot Q = 8 \cdot 0{,}775 = 6{,}2.$$

7. Среднее число клиентов, находящихся в очереди:

$$\bar{N}_{\text{оч}} = P_0 \frac{\rho^2\left[1-\rho^m(m+1-m\rho)\right]}{(1-\rho^{m+2})(1-\rho)} =$$

$$= 0{,}276 \frac{0{,}933^2\left[1-0{,}933^2(2+1-2\cdot 0{,}933)\right]}{(1-0{,}933^4)(1-0{,}933)} \approx 0{,}69.$$

8. Среднее число клиентов, находящихся на обслуживании:

$$\bar{N}_{\text{об}} = \rho Q = 0{,}933 \cdot 0{,}775 \approx 0{,}72.$$

9. Среднее число клиентов, находящихся в системе:

$$\bar{N}_{\text{сис}} = \bar{N}_{\text{оч}} + \bar{N}_{\text{об}} = 0{,}69 + 0{,}72 = 1{,}41.$$

10. Среднее время пребывания клиента в очереди, мин:

$$\bar{T}_{\text{оч}} = \frac{\bar{N}_{\text{оч}}}{\lambda P_{\text{сис}}} = \frac{0{,}69}{8 \cdot 0{,}775} = 0{,}1113 \text{ ч} = 6{,}68.$$

11. Среднее время пребывания клиента в системе, мин:

$$\bar{T}_{\text{сис}} = \frac{\bar{N}_{\text{сис}}}{\lambda P_{\text{сис}}} = \frac{1{,}41}{8 \cdot 0{,}775} = 0{,}228 \text{ ч} = 13{,}68.$$

Работу рассмотренной нотариальной конторы можно считать удовлетворительной, так как она не обслуживает клиентов в среднем в 22,5 % случаев ($P_{\text{отк}} = 0{,}225$). С помощью аналитического расчета можно показать, что за счет введения дополнительного места в очереди можно уменьшить вероятность отказа в обслуживании до 17 %.

7.3. МНОГОКАНАЛЬНЫЕ СИСТЕМЫ МАССОВОГО ОБСЛУЖИВАНИЯ

Многоканальная СМО с n ($n \geq 2$) одинаковыми устройствами обслуживания изображена на рис. 7.3. Как и для рассмотренной одноканальной системы, пусть данная СМО является системой с ограниченным ожиданием (длина очереди $\leq m$), входящие потоки заявок и обслуживания являются простейшими с интенсивностью λ и μ соответственно.

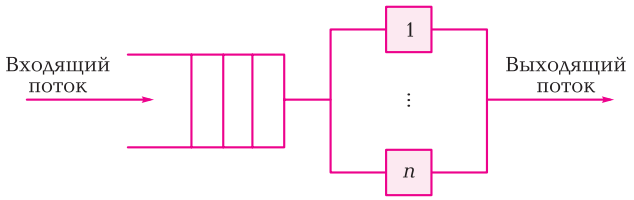


Рис. 7.3. Многоканальная СМО

Для данной СМО коэффициент нагрузки определяется следующим образом:

$$\rho = \frac{\lambda}{\mu}.$$

Коэффициент нагрузки, приходящийся на один канал, рассчитывается так:

$$\psi = \frac{\rho}{n}.$$

Вероятность того, что все каналы свободны

$$P_0 = \begin{cases} \left[\sum_{k=0}^{n} \frac{n^k}{k!} \psi^k + \frac{n^n}{n!} \frac{\psi^{n+1}\left(1-\psi^m\right)}{1-\psi} \right]^{-1}, \psi \neq 1; \\ \left[\sum_{k=0}^{n} \frac{n^k}{k!} + \frac{n^n}{n!} m \right]^{-1}, \psi = 1. \end{cases}$$

Вероятность присутствия в СМО k заявок вычисляется так:

$$P_k = \begin{cases} \frac{n^k}{k!} \psi^k P_0, k = 1, \ldots, n; \\ \frac{n^n}{n!} \psi^k P_0, k = n+1, \ldots, n+m. \end{cases}$$

Вероятность отказа заявке в обслуживании равна вероятности наличия в системе $n + m$ заявок:

$$P_{\text{отк}} = P_{n+m} = \frac{n^n}{n!} \psi^{n+m} P_0. \qquad (7.2)$$

Используя формулу (7.2), можно определить вероятность того, что заявка будет принята в систему:

$$P_{\text{сис}} = 1 - P_{\text{отк}} = 1 - \frac{n^n}{n!} \psi^{n+m} P_0.$$

Относительная пропускная способность СМО

$$Q = P_{\text{сис}} = 1 - P_{\text{отк}} = 1 - \frac{n^n}{n!} \psi^{n+m} P_0.$$

Абсолютная пропускная способность СМО

$$A = \lambda Q.$$

Среднее число занятых каналов (т. е. среднее число заявок, находящихся под обслуживанием)

$$\bar{N}_{\text{об}} = \frac{A}{\mu} = \rho Q = \rho \left(1 - \frac{n^n}{n!} \psi^{n+m} P_0 \right).$$

Среднее число заявок в очереди

$$\bar{N}_{\text{оч}} = \begin{cases} \frac{n^n}{n!} \psi^{n+1} \frac{1-(m+1)\psi^m + m\psi^{m+1}}{(1-\psi)^2} P_0, \psi \neq 1; \\ \frac{n^n}{n!} \frac{m(m+1)}{2} P_0, \psi = 1. \end{cases}$$

Среднее число заявок в СМО

$$\bar{N}_{\text{сис}} = \bar{N}_{\text{оч}} + \bar{N}_{\text{об}}.$$

Среднее время ожидания заявки в очереди

$$\bar{T}_{\text{оч}} = \frac{\bar{N}_{\text{оч}}}{\lambda P_{\text{сис}}}.$$

Среднее время пребывания заявки в системе

$$\bar{T}_{\text{сис}} = \frac{\bar{N}_{\text{сис}}}{\lambda P_{\text{сис}}}.$$

Приведенные формулы, как уже упоминалось, позволяют выполнять расчеты для СМО с простейшими потоками заявок и обслуживания и сравнивать их с полученными результатами имитационного моделирования.

Рассмотрим пример расчета характеристик многоканальной СМО, немного изменив приведенный ранее пример с одноканальной СМО.

Пример. Нотариальная контора представляет собой двухканальную СМО. Число мест в комнате ожидания очереди к нотариусу ограничено и равно трем. Если все места в комнате ожидания заняты, то вновь прибывший клиент в очередь не становится. Поток клиентов, прибывающий на консультацию, является простейшим с интенсивностью $\lambda = 12$ клиентов в час. Время обслуживания распределено по экспоненциальному закону со средним временем обслуживания $t = 7$ мин.

Определить вероятностные характеристики нотариальной конторы, работающей в стационарном режиме.

Решение. 1. Интенсивность потока обслуживания клиентов, клиентов в час:

$$\mu = \frac{1}{\bar{t}} = \frac{1}{7} \cdot 60 = 8{,}571.$$

2. Коэффициент нагрузки СМО:

$$\rho = \frac{\lambda}{\mu} = \frac{12}{8{,}571} = 1{,}4.$$

3. Коэффициент нагрузки СМО на один канал:

$$\psi = \frac{\rho}{n} = \frac{1{,}4}{2} = 0{,}7.$$

4. Вероятности наличия в СМО k клиентов:

$$P_0 = \left[\sum_{k=0}^{n} \frac{n^k}{k!}\psi^k + \frac{n^n}{n!}\frac{\psi^{n+1}(1-\psi^m)}{1-\psi}\right]^{-1} =$$

$$= \left[\sum_{k=0}^{2} \frac{2^k}{k!}0{,}7^k + \frac{2^2}{2!}\frac{0{,}7^{2+1}(1-0{,}7^3)}{1-0{,}7}\right]^{-1} \approx 0{,}205;$$

$$P_1 = \frac{n^1}{1!}\psi^1 P_0 = \frac{2^1}{1!}0{,}7^1 \cdot 0{,}205 \approx 0{,}287;$$

$$P_2 = \frac{n^2}{2!}\psi^2 P_0 = \frac{2^2}{2!}0{,}7^2 \cdot 0{,}205 \approx 0{,}201;$$

$$P_3 = \frac{n^3}{n!}\psi^3 P_0 = \frac{2^3}{2!}0{,}7^3 \cdot 0{,}205 \approx 0{,}140;$$

$$P_4 = \frac{n^4}{n!}\psi^4 P_0 = \frac{2^4}{2!}0{,}7^4 \cdot 0{,}205 \approx 0{,}098;$$

$$P_5 = \frac{n^5}{n!}\psi^5 P_0 = \frac{2^5}{2!}0{,}7^5 \cdot 0{,}205 \approx 0{,}069.$$

5. Вероятность отказа в обслуживании клиента:

$P_{\text{отк}} = P_{m+n} = 0{,}069.$

6. Относительная пропускная способность нотариальной конторы:

$Q = P_{\text{сис}} = 1 - P_{\text{отк}} = 1 - 0{,}069 = 0{,}931.$

7. Абсолютная пропускная способность нотариальной конторы, клиентов в час:

$A = \lambda Q = 12 \cdot 0{,}931 = 11{,}17.$

8. Среднее число клиентов, находящихся в очереди:

$$\bar{N}_{\text{оч}} = \frac{n^n}{n!}\psi^{n+1}\frac{1-(m+1)\psi^m + m\psi^{m+1}}{(1-\psi)^2}P_0 =$$

$$= \frac{2^2}{2!}0{,}7^{2+1}\frac{1-(3+1)0{,}7^3 + 3 \cdot 0{,}7^{3+1}}{(1-0{,}7)^2}0{,}205 \approx 0{,}544.$$

9. Среднее число клиентов, находящихся на обслуживании:

$\bar{N}_{\text{об}} = \rho Q = 1{,}4 \cdot 0{,}931 = 1{,}303.$

10. Среднее число клиентов, находящихся в системе:

$\bar{N}_{\text{сис}} = \bar{N}_{\text{оч}} + \bar{N}_{\text{об}} = 0{,}544 + 1{,}303 = 1{,}847.$

11. Среднее время пребывания клиента в очереди, мин:

$$\bar{T}_{\text{оч}} = \frac{\bar{N}_{\text{оч}}}{\lambda P_{\text{сис}}} = \frac{0{,}544}{12 \cdot 0{,}931} = 0{,}048 \text{ ч} = 2{,}92.$$

12. Среднее время пребывания клиента в системе, мин:

$$\bar{T}_{\text{сис}} = \frac{\bar{N}_{\text{сис}}}{\lambda P_{\text{сис}}} = \frac{1{,}847}{12 \cdot 0{,}931} = 0{,}165 \text{ ч} = 9{,}92.$$

Работу рассмотренной нотариальной конторы можно считать удовлетворительной, так как она не обслуживает клиентов в среднем всего в 6,9 % случаев ($P_{\text{отк}} = 0{,}069$).

7.4. ЗАМКНУТЫЕ СИСТЕМЫ МАССОВОГО ОБСЛУЖИВАНИЯ

В рассмотренных СМО интенсивность λ входящего потока заявок не зависела от состояния системы. В этом случае источник заявок являлся внешним по отношению к СМО и генерировал неограниченный поток требований. В ряде случаев интенсивность потока заявок λ зависит от состояния системы, причем источник требований является внутренним и генерирует ограниченный поток заявок. Подобные системы называются *замкнутыми системами массового обслуживания*.

Примером замкнутой СМО является автобусный парк, состоящий из N автобусов, которые при поломках ремонтируются R бригадами автомехаников ($N > R$). При этом каждый автобус ремонтируется только одной бригадой. В данной системе автобусы являются источниками заявок на обслуживание, а бригады автомехаников — обслуживающими каналами. Неисправный автобус после обслуживания используется на маршруте и становится потенциальным источником возникновения требований на обслуживание.

Очевидно, что интенсивность потока заявок λ зависит от того, сколько автобусов в данный момент работает ($N - K$) и сколько автобусов ремонтируется или стоит в очереди на обслуживание (K).

В рассматриваемой модели емкость источника требований следует считать ограниченной. Входящий поток требований исходит из ограниченного числа эксплуатируемых машин ($N - K$), которые в случайные моменты времени выходят из строя и требуют обслуживания.

При этом каждая машина из ($N - K$), находящаяся в эксплуатации, генерирует пуассоновский поток требований с интенсивностью λ независимо от других объектов. Тогда общий суммарный входящий поток имеет интенсивность $\lambda(N - K)$. Требование, поступившее в систему в момент, когда свободен хотя бы один канал, немедленно идет на обслуживание. Если требование застает все каналы занятыми обслуживанием других требований, то оно не покидает систему, а становится в очередь и ждет, пока один из каналов не станет свободным.

Приведем формулы для аналитической оценки наиболее важных показателей эффективности замкнутой СМО. Данные оценки представлены в [1].

Вероятность того, что в системе k объектов находится на обслуживании или в очереди, определяют как

$$P_k = P_0 \varphi_k,$$

где P_0 — вероятность того, что ни один объект не находится на обслуживании;

$$\varphi_k = \begin{cases} \dfrac{N!\rho^k}{k!(N-k)!}, & \text{для } 1 \le k < R; \\ \dfrac{N!\rho^k}{R!R^{k-R}(N-k)!}, & \text{для } R \le k \le N, \end{cases}$$

$$\rho = \frac{\lambda}{\mu},$$

Здесь λ — интенсивность потока требований в расчете на один обслуживаемый объект; μ — интенсивность обслуживания объектов одним каналом обслуживания.

Вероятность P_0 находят из равенства

$$\sum_{k=0}^{N} P_k = 1.$$

Получим

$$\sum_{k=0}^{N} P_k = P_0 + P_0\varphi_1 + P_0\varphi_2 + \ldots + P_0\varphi_N = P_0(1 + \varphi_1 + \varphi_2 + \ldots + \varphi_N) = 1,$$

откуда

$$P_0 = \frac{1}{1 + \varphi_1 + \varphi_2 + \ldots + \varphi_N}.$$

Среднее число объектов в очереди на обслуживание

$$\bar{N}_{\text{оч}} = \sum_{k=R}^{N} (k - R)P_k.$$

Среднее число объектов, находящихся в системе (на обслуживании и в очереди),

$$\bar{N}_{\text{сис}} = \sum_{k=0}^{N} kP_k.$$

Среднее число простаивающих каналов обслуживания

$$\bar{R}_{\text{п}} = \sum_{k=0}^{R-1} (R - k)P_k.$$

Коэффициент простоя обслуживаемого объекта

$$\alpha_1 = \frac{\bar{N}_{\text{оч}}}{N}.$$

Коэффициент использования объекта

$$\alpha_2 = 1 - \frac{\bar{N}_{\text{сис}}}{N}.$$

Коэффициент простоя обслуживающих каналов

$$\alpha_3 = \frac{\bar{R}_п}{R}.$$

Среднее время ожидания объекта в очереди

$$\bar{T}_{оч} = \frac{1}{\lambda}\left(\frac{1-\alpha_2}{\alpha_2}\right) - \frac{1}{\mu}.$$

Рассмотрим пример расчета характеристик замкнутой СМО.

Пример. Рассмотрим вычислительный центр, включающий в себя 15 компьютеров, которые при поломках ремонтируют два специалиста по ремонту ПК. При этом каждый компьютер ремонтируется только одним специалистом. Неисправный компьютер после ремонта используется в вычислительном центре и становится потенциальным источником возникновения требований на обслуживание. Интенсивность потока отказов от одного компьютера равна 0,5 раза за 100 ч, среднее время ремонта компьютера составляет 10 ч.

Определить характеристики вычислительного центра.

Решение. Для нашей задачи $N = 15$, $R = 2$, $\lambda = 0{,}5$, $\mu = 10$.

1. Коэффициент ρ определим как отношение интенсивностей λ и μ:

$$\rho = \frac{\lambda}{\mu} = \frac{0{,}5}{20} = 0{,}05.$$

2. Вычислим вероятности того, что в системе k компьютеров находится на обслуживании или в очереди.

Для этого сначала найдем φ_k для $k = 1 \,..\, N$:

$$\varphi_1 = \frac{10!0{,}05^1}{(10-1)!} = 0{,}5; \quad \varphi_2 = \frac{10!0{,}05^2}{2!2^{2-2}(10-2)!} = 0{,}1125;$$

$$\varphi_3 = \frac{10!0{,}05^3}{2!2^{3-2}(10-3)!} = 0{,}0225; \quad \varphi_4 = \frac{10!0{,}05^4}{2!2^{4-2}(10-4)!} = 3{,}9\cdot 10^{-3};$$

$$\varphi_5 = \frac{10!0{,}05^5}{2!2^{5-2}(10-5)!} = 5{,}9\cdot 10^{-4}; \quad \varphi_6 = \frac{10!0{,}05^6}{2!2^{6-2}(10-6)!} = 7{,}4\cdot 10^{-5};$$

$$\varphi_7 = \frac{10!0{,}05^7}{2!2^{7-2}(10-7)!} = 7{,}6\cdot 10^{-6}; \quad \varphi_8 = \frac{10!0{,}05^8}{2!2^{8-2}(10-8)!} = 5{,}5\cdot 10^{-7};$$

$$\varphi_9 = \frac{10!0{,}05^9}{2!2^{9-2}(10-9)!} = 2{,}8\cdot 10^{-8}; \quad \varphi_{10} = \frac{10!0{,}05^{10}}{2!2^{10-2}(10-10)!} = 6{,}9\cdot 10^{-10}.$$

Вычислим P_0:

$$P_0 = \frac{1}{1+\varphi_1+\varphi_2+\ldots+\varphi_N} = \frac{1}{1{,}6396} = 0{,}6099.$$

Рассчитаем P_k:

$P_1 = P_0\varphi_1 = 0{,}3050$; $P_2 = P_0\varphi_2 = 0{,}0686$;

$P_3 = P_0\varphi_3 = 0{,}0137$; $P_4 = P_0\varphi_4 = 2{,}4\cdot 10^{-3}$;

$P_5 = P_0\varphi_5 = 3{,}6\cdot 10^{-4}$; $P_6 = P_0\varphi_6 = 4{,}5\cdot 10^{-5}$;

$P_7 = P_0\varphi_7 = 4{,}5\cdot 10^{-6}$; $P_8 = P_0\varphi_8 = 3{,}4\cdot 10^{-7}$;

$P_9 = P_0\varphi_9 = 1{,}7\cdot 10^{-8}$; $P_{10} = P_0\varphi_{10} = 4{,}2\cdot 10^{-10}$.

3. Среднее число компьютеров в очереди на обслуживание

$$\bar{N}_{оч} = \sum_{k=2}^{N}(k-R)P_k = 0{,}0198.$$

4. Среднее число компьютеров, находящихся на обслуживании или в очереди,

$$\bar{N}_{сис} = \sum_{k=0}^{N} kP_k = 0{,}4951.$$

5. Среднее число простаивающих специалистов по ремонту

$$\bar{R}_п = \sum_{k=0}^{R-1}(R-k)P_k = 2P_0 + P_1 = 1{,}5248.$$

6. Коэффициент простоя компьютера

$$\alpha_1 = \frac{\bar{N}_{оч}}{N} = \frac{0{,}0198}{10} = 1{,}98\cdot 10^{-2}.$$

7. Коэффициент использования компьютера

$$\alpha_2 = 1 - \frac{\bar{N}_{сис}}{N} = 1 - \frac{0{,}4951}{10} = 0{,}9505.$$

8. Коэффициент простоя специалиста по ремонту

$$\alpha_3 = \frac{\bar{R}_п}{R} = \frac{1{,}5248}{2} = 0{,}7624.$$

9. Среднее время ожидания компьютера в очереди на ремонт

$$\bar{T}_{оч} = \frac{1}{\lambda}\left(\frac{1-\alpha_2}{\alpha_2}\right) - \frac{1}{\mu} = \frac{1}{0{,}05}\left(\frac{1-0{,}9505}{0{,}9505}\right) - \frac{1}{10} = 4{,}2\cdot 10^{-3}.$$

Отметим, что полученное среднее время ожидания в очереди измеряется в сотнях часов, т. е. в среднем компьютеры ждут ремонта 0,42 ч.

7.5. КОМПЬЮТЕРНОЕ МОДЕЛИРОВАНИЕ СИСТЕМ МАССОВОГО ОБСЛУЖИВАНИЯ

В том случае, если для исследуемой СМО невозможно получить аналитические выражения для оценки ее характеристик или данные выражения оказываются слишком сложными для практического применения, можно выполнить исследование эффективности СМО с помощью компьютерного моделирования. Различные подходы к моделированию СМО на конкретных примерах рассмотрены в [2].

Для моделирования СМО могут применяться следующие основные способы построения алгоритмов [2]:

- повременного моделирования с переменным шагом (или способ продвижения модельного времени от события к событию);
- повременного моделирования с постоянным шагом;
- последовательной проводки заявок;
- поэтапной последовательной проводки заявок.

Данные алгоритмы отличаются способом продвижения модельного времени, хранящегося в так называемых часах модельного времени, от одного значения к другому. Кратко рассмотрим суть первых двух алгоритмов, после чего более подробно опишем способ последовательной проводки заявок, чаще всего применяемый при моделировании СМО.

При использовании *продвижения времени от события к событию* часы модельного времени в исходном состоянии устанавливаются в нуль и определяется время возникновения будущих событий, приводящих к изменению состояния системы [5]. После этого часы модельного времени переходят на время возникновения ближайшего события, и в этот момент обновляется состояние системы с учетом произошедшего события, а также сведения о времени возникновения будущих событий. Затем часы модельного времени продвигаются ко времени возникновения следующего ближайшего события, обновляется состояние системы и определяется время будущих событий и т. д. Процесс продвижения модельного времени от времени возникновения одного события ко времени возникновения другого продолжается до тех пор, пока не будет выполнено какое-либо условие останова, указанное заранее. Поскольку в такой имитационной модели все изменения происходят только во время возникновения событий, периоды бездействия системы просто пропускаются и часы переводятся от времени возникновения одного события на время возникновения другого.

При *продвижении модельного времени с постоянным шагом* часы модельного времени продвигаются точно на Δt единиц времени для конкретного значения Δt. После каждого обновления часов выполняется проверка с целью определить, произошли ли какие-либо события в течение предыдущего интервала времени Δt. Если на этот интервал запланированы одно или несколько событий, считается, что данные события происходят в конце интервала, после чего состояние системы и статистические счетчики соответствующим образом обновляются.

Продвижение времени посредством постоянного шага имеет два недостатка:

- возникновение ошибок, связанных с обработкой событий в конце интервала, в течение которого они происходят;
- необходимость решать, какое событие обрабатывать первым, если события, в действительности происходящие в разное время, рассматриваются как одновременные.

Подобного рода проблемы можно частично решить, сделав интервалы Δt менее продолжительными, но тогда возрастает число проверок возникновения событий, что приводит к увеличению времени выполнения задачи. Из-за этого продвижение времени с помощью постоянного шага не используют при моделировании СМО, когда интервалы времени между последовательными событиями могут значительно отличаться по своей продолжительности.

При моделировании процессов обслуживания заявок в СМО в ряде случаев удобно строить моделирующий алгоритм *по способу последовательной проводки заявок* [2]. Этот способ может использоваться, если события, происходящие в системе, не зависят друг от друга. Моделирующий алгоритм последовательно воспроизводит истории отдельных заявок в порядке их поступления в систему. Алгоритм обращается к сведениям о других заявках лишь в том случае, если это необходимо для решения вопроса о дальнейшем порядке обслуживания данной заявки.

Существуют две разновидности способа последовательной проводки:

- проводка одиночных заявок;
- проводка потоков заявок.

Данные способы отличаются лишь порядком формирования и обслуживания заявок. Сначала рассмотрим способ *проводки одиночных заявок* [2] применительно к моделированию одноканальной СМО с ожиданием. В процессе моделирования необходимо сформировать очередную заявку (для этого генерируется случай-

ное число, определяющее интервал времени между приходом заявки, и случайное время ее обслуживания) и выполнить ее обслуживание, определив время начала и окончания обслуживания.

В начале моделирования часы модельного времени устанавливаются в нуль. Пусть первая заявка поступила в систему в момент времени $t_1 = A_1$. Здесь A_1 — реализация случайной величины, представляющей собой время между поступлением заявок. Поскольку в момент времени t_1 канал обслуживания был свободен, обслуживание заявки начинается в момент времени $t_{н1} = t_1$. Затем нужно получить реализацию случайной величины времени обслуживания первой заявки S_1. Тогда момент окончания обслуживания 1-й заявки $t_{к1} = t_{н1} + S_1$. При этом к счетчику числа обслуженных заявок прибавляется единица.

Далее определяется время поступления следующей 2-й заявки. Для этого формируется время между поступлением заявок A_2 и определяется момент времени поступления 2-й заявки $t_2 = t_1 + A_2$. Если в данный момент канал, например, занят, то для 2-й заявки начинается период ожидания продолжительностью $\Delta T_{ож} = t_{к1} - t_2$. Обслуживание этой заявки начинается в момент $t_{н2} = t_{к1}$. Далее определяется случайная величина времени обслуживания 2-й заявки S_2. Тогда момент окончания обслуживания 2-й заявки $t_{к2} = t_{н2} + S_2$. К счетчику числа обслуженных заявок прибавляется единица. Аналогичным образом обслуживаются последующие заявки.

Процесс формирования и обслуживания четырех заявок проиллюстрирован на рис. 7.4. Верхняя временна́я ось показывает время прихода (левая граница прямоугольников) и обслуживания (длина прямоугольников) заявок, а нижняя — время начала (левая граница прямоугольников) и окончания (правая граница прямоугольников) обслуживания заявок.

Вторым вариантом способа последовательной проводки заявок является способ, при котором вначале формируется весь *поток*

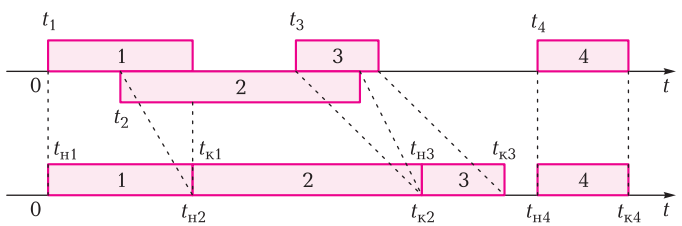


Рис. 7.4. Иллюстрация способа последовательной проводки заявок

заявок, а затем начинается процесс обслуживания заявок [2]. При этом предполагается, что все заявки однородные. В этом случае для хранения данных о заявках в памяти ЭВМ необходимо создавать массивы чисел. Поскольку размерность такого массива заранее неизвестна, приходится выбирать ее с определенным запасом, что приводит к напрасному расходу памяти.

Далее представлены фрагменты текста программы на языке C# моделирования одноканальной СМО с неограниченным ожиданием, в которых будет применяться второй вариант способа последовательной проводки заявок.

Формирование входящего потока заявок:

```
…
double time = 0;//Обнуляем модельное время
int i, Nin = 0;//Обнуляем счетчик поступивших заявок
   // формируем входящие заявки
   for (i = 0; i < Zmax; i++)
   {
   //Формируем время поступления очередной заявки
   time = time + rnd_expo(lambda);
   //Если время моделирования не закончено
   if (time < modelTime)
   {
      // то увеличить количество поступивших заявок
      Nin++;
      // запомнить время прихода заявки
      z_arrive[i] = time;
      // сформировать продолжительность обслуживания
      //заявки
      z_serve[i] = rnd_expo(mu);
   }
   else break; //выход при окончании времени модели-
   //рования
}
…
```

Используемые в программе константы имеют следующее назначение:

lambda — интенсивность входящего потока заявок;

mu — интенсивность потока обслуживания заявок;

Zmax — максимальное количество обслуживаемых заявок (определяет размер массивов z_arrive и z_serve, в которых хранится информация о времени прихода и обслуживания заявок);

modelTime — время моделирования.

В представленном фрагменте программы использовалась пользовательская функция rnd_expo, генерирующая экспоненциально распределенную случайную величину с параметром, определенным аргументом функции.

Далее приведен фрагмент программы проводки заявок.

```
time = 0; // Обнуляем модельное время
int Nserve = 0; // Обнуляем счетчик числа обслужен-
//ных заявок
//перебираем все поступившие заявки
for (i = 0; i < Nin; i++)
{
   // Определяем начало обслуживания текущей заявки
   if (z_arrive[i] < time) timeServe = time;
   else timeServe = z_arrive[i];
   //если время моделирования не закончилось
   if (timeServe < modelTime)
   {
      // то запоминаем время начала обслуживания за-
      //явки
      z_begin[i] = timeServe;
      time = timeServe + z_serve[i];
      //запоминаем время окончания обслуживания
      z_end[i] = time;
      //увеличиваем количество обслуженных заявок
      Nserve++;
   }
   else break; //выход при завершении модельного вре-
   //мени
}
//время окончания обслуживания последней заявки
double timeEnd = time;
```

Отметим, что при разработке программы предполагалось, что заявка, начавшая обслуживаться до окончания модельного времени, будет обязательно обслужена. Следующие за ней заявки обслужены не будут.

Теперь представим фрагмент программы для вычисления и вывода на экран характеристик функционирования СМО.

```
double timeWait = 0; // время ожидания заявок в оче-
//реди
```



```
double timeSys = 0; // время пребывания заявок в си-
//стеме
double timeWork = 0; // время работы системы
//перебираем обслуженные заявки
for (i = 0; i < Nserve; i++)
{
   // накапливаем время ожидания в очереди
   timeWait += (z_begin[i] - z_arrive[i]);
   // накапливаем время пребывания в системе
   timeSys += (z_end[i] - z_arrive[i]);
   // накапливаем время работы системы
   timeWork += z_serve[i];
}
Console.WriteLine("Число поступивших заявок:{0}", Nin);
Console.WriteLine("Число обслуженных заявок:{0}",
Nserve);
Console.WriteLine("Среднее время ожидания в
очереди:{0:f4}",
         (double) timeWait / Nserve);
Console.WriteLine("Среднее время пребывания в системе:
         {0:f4}", (double) timeSys / Nserve);
Console.WriteLine("Коэффициент использования СМО:
{0:f4}",
         (double) timeWork/timeEnd);
```

Рассмотренный способ последовательной проводки заявок можно применять и для создания моделей многоканальных СМО. Для примера на рис. 7.5 проиллюстрирован процесс моделирова-

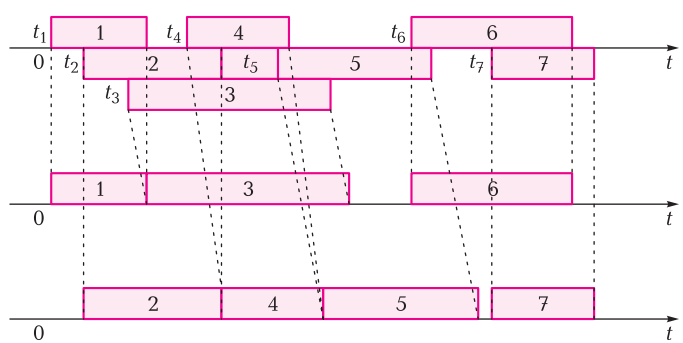


Рис. 7.5. Иллюстрация способа последовательной проводки заявок для моделирования двухканальной СМО

ния работы двухканальной СМО. Как видно из рисунка, в данном случае входящий поток заявок (верхняя временна́я ось) распределяется на два устройства обслуживания (нижние временны́е оси). Причем, если в момент прихода заявки есть незанятые устройства, то она обслуживается любым из них (обычно выбирается первое свободное). Если же все устройства обслуживания заняты, то заявка направляется в то устройство, которое раньше освободится.

Далее представлен фрагмент программы обслуживания заявок и определения характеристик в многоканальной СМО с ожиданием с количеством каналов servCount (формирование заявок при этом полностью совпадает с формированием заявок в одноканальной СМО).

```
// сначала все обслуживающие приборы свободны
for (j = 0; j < servCount; j++)
  endTime[j] = 0;
int Nserve = 0; // Обнуляем счетчик числа обслужен-
//ных заявок
double timeWait = 0; // время ожидания заявок в оче-
//реди
double timeSys = 0; // время пребывания заявок в си-
//стеме
double timeWork = 0; // время работы системы
//перебираем поступившие заявки
for (i = 0; i < Nin; i++)
{
  //Определяем номер канала ind, который освободится
  //раньше
  ind = 0;
  for (j = 1; j < servCount; j++)
    if (endTime[j] < endTime[ind])
      ind = j;
  // определяем время начала обслуживания
  // если все приборы обслуживания заняты
  if (z_arrive[i] < endTime[ind])
    timeServe = endTime[ind];
  else
    timeServe = z_arrive[i];
  // если время не закончилось
  if (timeServe < modelTime)
  {
```



```
    // то запоминаем время начала обслуживания за-
    явки
    z_begin[i] = timeServe;
    // запоминаем время окончания обслуживания
    z_end[i] = timeServe + z_serve[i];
    // накапливаем время работы системы
    timeWork += z_serve[i];
    endTime[ind] = z_end[i];
    // увеличиваем количество обслуженных заявок
    Nserve ++;
    // накапливаем время ожидания в очереди
    timeWait += (z_begin[i] - z_arrive[i]);
    // накапливаем время пребывания в системе
    timeSys += (z_end[i] - z_arrive[i]);
    if (timeEnd < endTime[ind]) timeEnd =
    endTime[ind];
  }
  else break; //завершаем моделирование
}
Console.WriteLine("Число поступивших заявок:{0}", Nin);
Console.WriteLine("Число обслуженных заявок:{0}",
Nserve);
Console.WriteLine("Среднее время ожидания в
очереди:{0:f4}",
              (double) timeWait / Nserve);
Console.WriteLine("Среднее время пребывания в системе:
              {0:f4}", (double) timeSys / Nserve);
Console.WriteLine("Коэффициент использования СМО:
{0:f4}",
              (double) timeWork/timeEnd/servCount);
```

Последний обсуждаемый способ моделирования СМО — *способ поэтапной последовательной проводки заявок*, предназначен для моделирования СМО, в которых есть заявки различных приоритетов. Данный способ основан на способе последовательной проводки заявок и заключается в том, что сначала моделируется процесс обслуживания заявок с более высоким приоритетом, а потом процесс обслуживания заявок более низкого приоритета. При этом учитывается, что часть времени работы устройства обслуживания уже занята обслуживанием заявок более высокого приоритета. Другими словами, заявки более низкого приоритета должны обслуживаться только во время простоя устройства обслуживания.

При этом реализуется так называемый абсолютный приоритет, когда заявка более высокого приоритета, пришедшая в момент обслуживания заявки более низкого приоритета, вытесняет заявку более низкого приоритета. Данная заявка после освобождения устройства обслуживания дообслуживается.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое СМО? Приведите примеры таких систем.
2. Каковы основные компоненты СМО? Какими параметрами они задаются?
3. Каковы основные подходы к классификации СМО? Дайте классификацию СМО.
4. Какие основные характеристики СМО вы знаете?
5. Что понимают под одноканальной СМО? Опишите ее.
6. Как аналитически рассчитать характеристики одноканальной СМО?
7. Что понимают под многоканальной СМО? Опишите ее.
8. Как аналитически рассчитать характеристики многоканальной СМО?
9. Какие СМО называются замкнутыми?
10. В чем заключается способ моделирования с продвижением модельного времени от события к событию?
11. В чем состоит способ моделирования с продвижением модельного времени с постоянным шагом?
12. В чем заключается способ последовательной проводки заявок?
13. В чем состоит способ поэтапной последовательной проводки заявок?

Глава 8

ПЛАНИРОВАНИЕ КОМПЬЮТЕРНОГО ЭКСПЕРИМЕНТА

8.1. ОСНОВНЫЕ ПОНЯТИЯ

После того как установлены цели эксперимента и принято решение использовать машинное имитационное моделирование, полезно провести предварительное планирование предстоящего эксперимента.

Исследователь должен иметь подробный план для целенаправленного и эффективного получения требуемых данных. Плановые ограничения на время и затраты должны быть приведены в соответствие с имеющимися в распоряжении исследователя ресурсами. Чем дороже и сложнее эксперимент, тем большее внимание следует обращать на этот этап.

План эксперимента определяет:

- объем вычислений на компьютере;
- порядок проведения вычислений на компьютере;
- способы накопления и статистической обработки результатов моделирования.

Планирование экспериментов имеет следующие цели:

- сокращение общего времени моделирования при соблюдении требований к точности и достоверности результатов;
- увеличение информативности каждого наблюдения;
- создание структурной основы процесса исследования.

Рассмотрим основные понятия теории планирования экспериментов, подробно изложенные в [12]. Наиболее подходящей моделью системы при планировании эксперимента является абстрактная схема, называемая «черным ящиком» (рис. 8.1).

При таком подходе различают входные и выходные переменные: x_1, x_2, ..., x_k; y_1, y_2, ..., y_m. Любая из переменных является либо фактором, либо реакцией.

Рис. 8.1. Модель системы в виде «черного ящика»

Пусть, например, имеют место только две переменные: x и y. Тогда если цель эксперимента — изучение влияния переменной x на переменную y, то x — фактор, а y — реакция.

Каждый фактор x_i, $i = 1 \ldots k$ может принимать в эксперименте одно из нескольких значений, называемых *уровнями*. Фиксированный набор уровней всех факторов определяет одно из возможных состояний рассматриваемой системы.

Каждому фиксированному набору уровней факторов соответствует определенная точка в многомерном пространстве, называемом *факторным пространством*.

Существует вполне определенная связь между уровнями факторов и реакцией системы, которую можно представить в виде соотношения

$$y_j = \psi_j(x_1, x_2, \ldots, x_k),\ j = \overline{1,m}.$$

Функцию ψ_j, связывающую реакцию с факторами, называют *функцией реакции*, а соответствующий ей геометрический образ — *поверхностью реакции*. Знание функций реакции дает исследователю полную информацию о работе системы. Он может легко научиться управлять интересующими характеристиками системы, подобрать оптимальные значения факторов в соответствии с заданным критерием оптимальности. Поэтому исследователю в процессе моделирования важно узнать, каким образом реакции зависят от факторов.

Вместе с тем исследователю заранее неизвестен вид зависимостей ψ_j, $\varphi = 1 \ldots m$, поэтому используют приближенные соотношения:

$$\tilde{y}_j = \varphi_j(x_1, x_2, \ldots, x_k),\ \varphi = \overline{1,m}.$$

В качестве зависимостей φ_l обычно выбираются полиномы первой или второй степени, коэффициенты которых находятся по данным эксперимента (например, с помощью метода наименьших квадратов), который необходимо поставить так, чтобы при минимальных затратах вычислительных ресурсов (например, минимальном числе испытаний), варьируя по специально сформулированным правилам значения входных переменных, построить математическую модель системы и оценить ее характеристики.

При планировании экспериментов необходимо определить основные свойства факторов. Факторы при проведении экспериментов могут быть управляемыми и неуправляемыми, наблюдаемыми и ненаблюдаемыми, изучаемыми и неизучаемыми, количественными и качественными, фиксированными и случайными.

Фактор называется *управляемым*, если его уровни целенаправленно выбираются исследователем в процессе эксперимента. Фактор называется *наблюдаемым*, если его значения наблюдаются и регистрируются. Фактор относится к *изучаемым*, если он включен в модель для изучения свойств системы, а не для вспомогательных целей, например для увеличения точности эксперимента. Фактор будет *количественным*, если его значения — числовые величины, влияющие на реакцию, а в противном случае фактор называется *качественным*. Например, в СМО количественными факторами являются интенсивности входящих потоков заявок, интенсивности потоков обслуживания, емкости накопителей, число обслуживающих каналов и т. д., а качественными факторами — дисциплины постановки в очередь, выбора из очереди, обслуживания заявок каналами и т. д.

Фактор называется *фиксированным*, если в эксперименте исследуются все интересующие экспериментатора значения фактора, а если экспериментатор исследует только некоторую случайную выборку из совокупности интересующих значений факторов, то фактор называется *случайным*. На основании случайных факторов могут быть сделаны вероятностные выводы и о тех значениях факторов, которые в эксперименте не исследовались.

Основными требованиями, предъявляемыми к факторам, являются требование управляемости фактора, требование непосредственного воздействия на объект, требования совместимости и независимости факторов.

Под *управляемостью фактора* понимается возможность установки и поддержания выбранного нужного уровня фактора постоянным в течение всего испытания или изменяющимся в соответствии с заданной программой. Требование *непосредственного воздействия на объект* имеет большое значение в связи с тем, что трудно управлять фактором, если он является функцией других факторов. *Совместимость факторов* означает, что все их комбинации осуществимы, а *независимость* соответствует возможности установления фактора на любом уровне независимо от уровней других.

При планировании машинных экспериментов можно выделить две составляющие: стратегическое и тактическое планирование.

Стратегическое планирование ставит своей основной целью решение задачи получения необходимой информации о системе с помощью модели, реализованной на ЭВМ, с учетом ограничений на ресурсы, имеющиеся в распоряжении экспериментатора. К числу частных задач, решаемых при планировании машинных экспериментов, относятся задачи уменьшения затрат машинного времени на моделирование, увеличения точности и достоверности результатов моделирования, проверки адекватности модели.

Тактическое планирование представляет собой определение способа проведения каждой серии испытаний машинной модели, предусмотренных планом эксперимента.

8.2. СТРАТЕГИЧЕСКОЕ ПЛАНИРОВАНИЕ ЭКСПЕРИМЕНТА

Процесс построения стратегического плана моделирования состоит из двух основных этапов: построение структурной модели эксперимента; построение функциональной модели.

При построении *структурной модели* выбирается число факторов и число уровней для каждого фактора. Выбор данных параметров определяется целями эксперимента, точностью измерений факторов, интересом к нелинейным эффектам и т. п. На этот выбор не должна влиять ограниченность числа возможных измерений, возникающая вследствие ограниченности ресурсов, т. е. структурная модель выбирается исходя из того, что должно быть сделано.

Структурная модель, следовательно, имеет вид

$$N_s = q_1 q_2 \ldots q_k,$$

где N_s — число элементов эксперимента; k — число факторов эксперимента; q_i — число уровней i-го фактора.

Здесь под элементом понимается простейший эксперимент, проводимый при конкретном наборе значений факторов.

При выборе числа и вида необходимых факторов последовательно решаются такие вопросы:

1) определяют, какие отклики (выходные переменные) нас интересуют в первую очередь;

2) устанавливают, какие факторы влияют на эти переменные. Поскольку большинство систем работает в соответствии с принципом Парето, согласно которому 20 % факторов определяют 80 % свойств системы, а остальные 80 % факторов определяют лишь 20 % ее свойств, на данном этапе выделяются наиболее существенные факторы;

3) выделяют измеряемые и управляемые факторы и определяют их возможные уровни. Число уровней для каждого фактора следует выбирать минимально возможным, но достаточным для достижения целей эксперимента. Определение для всех факторов одинакового числа q равноотстоящих уровней (два-три) существенно упрощает исследование модели. В этом случае структурная модель имеет вид

$$N_s = q^k.$$

Функциональная модель обусловливает количество элементов структурной модели, которые должны служить действительными измерителями отклика, т. е. определять, сколько необходимо иметь различных информационных точек. Подобные функциональные модели могут быть либо совершенными, либо несовершенными. Функциональная модель называется *совершенной*, если в измерении отклика участвуют все ее элементы, т. е. $N_f = N_s$. Функциональная модель называется *несовершенной*, если число имеющих место откликов меньше числа элементов, т. е. $N_f < N_s$. В идеале функциональная модель совпадает со структурной. Но так как большинство модельных исследований имеет ограничения, наложенные на время, денежные средства и производительность вычислительных машин, при построении функциональной модели необходимо выбрать приемлемый компромисс между желаниями и ресурсами. Дальнейшие действия при построении функциональной модели зависят от целей проводимых исследований. Например, в случае необходимости исследования поверхности реакции осуществляется выбор полного или неполного факторного эксперимента, достаточного для определения коэффициентов функций зависимости $\tilde{y}_j = \varphi_j(x_1, x_2, \ldots, x_k),\ j = \overline{1,m}.$

Более полную информацию о построении стратегических планов можно получить в [5, 12].

8.3. ТАКТИЧЕСКОЕ ПЛАНИРОВАНИЕ КОМПЬЮТЕРНОГО ЭКСПЕРИМЕНТА

Тактическое планирование эксперимента с машинной моделью системы связано с вопросами эффективного использования выде-

ленных для эксперимента машинных ресурсов и определения конкретных способов проведения испытаний модели, намеченных планом эксперимента, построенным при стратегическом планировании. При этом решаются проблемы:

- определения начальных условий и их влияния на достижение установившегося результата при моделировании;
- обеспечения точности и достоверности результатов моделирования;
- выбора правил автоматической остановки имитационного эксперимента с моделями систем.

8.3.1. Начальные условия и их влияние на достижение установившегося режима

Первая проблема при проведении машинного эксперимента заключается в искажении начального периода работы машинной модели из-за влияния начальных условий запуска модели. При этом типичное поведение оцениваемого параметра $\hat{p}$ (в качестве параметра $\hat{p}$ взята вероятность приема заявки к обслуживанию) при изменении объема N выборки показано на рис. 8.2.

Существуют три основных пути уменьшения возможного влияния начального периода на получаемые данные.

1. Использование достаточно длинных вычислительных прогонов, чтобы число данных переходного периода было незначительно по сравнению с числом данных установившегося режима. Однако при этом существенно увеличивается время моделирования.

2. Исключение из рассмотрения начального периода прогона, требуемого для достижения моделью установившегося режима работы. В данном случае сложность составляет определение времени переходного процесса. Если выбрать его слишком малым, то останется влияние начальных условий. Если же выбрать его слишком большим, то увеличивается время моделирования за счет того, что часть полезных данных будет отброшена.

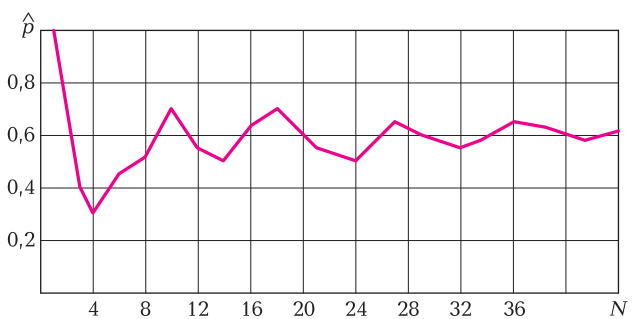


Рис. 8.2. Зависимость изменения оценки параметра от объема выборки

3. Выбор для установившегося состояния такого начального условия, которое ближе к типичному, и тем самым уменьшение переходного периода. В данном случае также возникает проблема определения типичного состояния системы при моделировании.

Таким образом, все эти приемы позволяют только уменьшить, но не свести к нулю влияние переходного процесса при проведении машинного эксперимента с моделью.

Далее рассмотрим вопросы определения объема выборки, позволяющего обеспечить заданные точность и достоверность результатов моделирования, описанные в [10].

8.3.2. Оценивание среднего значения выборочной совокупности

Задача состоит в получении оценки неизвестного математического ожидания μ некоторого исследуемого параметра по выборочной средней $\bar{x}$ за минимальное время с заданной точностью и достоверностью. Выборочную среднюю $\bar{x}$, полученную при нескольких прогонах имитационной модели с различными начальными условиями, можно рассматривать как случайную величину X ($\bar{x}$ изменяется от выборки к выборке) и выборочные значения признака $x_1, x_2, \ldots, x_N$ — как одинаково распределенные независимые случайные величины $X_1, X_2, \ldots, X_N$ с математическим ожиданием μ и среднеквадратическим отклонением σ. Сначала будем использовать предположение о независимости и нормальном распределении откликов модели. Это предположение основано на применении *центральной предельной теоремы* теории вероятностей, сущность которой состоит в утверждении, что распределение случайной величины, являющейся суммой большого числа независимых величин с одинаковыми распределениями вероятностей, близко к нормальному распределению. В более поздних работах показано, что требования независимости и одинаковой распределенности не являются необходимыми. Часто бывает достаточно, чтобы отклик представлял собой сумму большого числа небольших эффектов. Поэтому переменная отклика сложной имитационной модели, являющаяся результатом аддитивного действия

большого числа случайных переменных, распределена приблизительно нормально. В результате этого, если случайная величина X распределена нормально, то выборочная средняя $\bar{x}$, найденная по независимым наблюдениям, также распределена нормально с математическим ожиданием

$$M(\bar{x}) = \mu$$

и среднеквадратическим отклонением

$$\sigma(\bar{x}) = \sigma/\sqrt{N}.$$

Потребуем, чтобы выполнялось соотношение

$$P(\mu - d \le \bar{x} < \mu + d) = \beta, \qquad (8.1)$$

где $\bar{x}$ — выборочное значение средней; d — точность оценки; β — вероятность того, что интервал $\mu \pm d$ содержит $\bar{x}$ (надежность, задаваемая исследователем).

Задаваемая надежность β однозначно связана с уровнем значимости α получаемой оценки $\bar{x}$ формулой

$$\beta = 1 - \alpha.$$

Если требуется оценить математическое ожидание μ с наперед заданной точностью d и надежностью β (уровнем значимости α), то минимальный объем выборки, который обеспечит эту точность, находят по формуле

$$N = \frac{t_\beta^2 \sigma^2}{d^2},$$

где число t_β определяется из равенства $2\Phi(t_\beta) = \beta$ или $\Phi(t_\beta) = \beta/2$ по таблице функции Лапласа (здесь $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_0^x e^{-\frac{z^2}{2}} dz$ — функция Лапласа).

Если неизвестны точность d и среднеквадратическое отклонение σ, то задачу можно поставить так: каков должен быть размер выборки, чтобы искомая оценка с вероятностью $\beta = 1 - \alpha$ находилась в пределах $\mu \pm \Delta\sigma$, где $\Delta\sigma$ — некоторая доля σ (например, $\Delta\sigma = \sigma/2$; $\sigma/4$; $\sigma/6$; $\sigma/10$; $\sigma/20$; ...).

Например, при $\Delta\sigma = \sigma/4$ и $\beta = 0{,}95$ ($\sigma = 0{,}05$) имеем $t_\beta = 1{,}96$, тогда

$$N = \frac{(1{,}96\sigma)^2}{(\sigma/4)^2} = 61.$$

В других случаях следует определить дисперсию выхода с помощью пробного эксперимента и получить оценку s^2 дисперсии, а затем вычислить полное число необходимых наблюдений. Тогда размер выборки определяется выражением

$$N = \frac{t_\beta^2 S^2(n)}{d^2},$$

где оценка дисперсии получена по формуле

$$S^2(n) = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \left(\frac{1}{n}\sum_{i=1}^{n} x_i\right)^2,$$

где n — размер выборки пробного эксперимента.

8.3.3. Применение теоремы Чебышева

Неравенство Чебышева говорит, что при заданном числе k (не меньше единицы) и произвольной выборке $x_1, x_2, \ldots, x_N$ размера N по меньшей мере $1 - 1/k^2$ доля измерений находится вблизи среднего значения на расстоянии не более k среднеквадратических отклонений. Это неравенство справедливо для любых распределений совокупности. Если не исходить из предположения о нормальном распределении выхода (которое не всегда выполняется с достаточной точностью), можно воспользоваться неравенством Чебышева, которое имеет вид

$$P(|x - \mu| > k\sigma) \le \frac{1}{k^2}.$$

Зададимся условием, при котором оценка $\bar{X}$ должна находиться в интервале $\mu \pm \sigma/m$ (m = 2, 4, 6, 10, ...) с вероятностью $\beta = 1 - \alpha$:

$$P\left(|\bar{x} - \mu| > \frac{\sigma}{m}\right) \le \sigma,$$

тогда, используя уравнение (8.1), получим

$$P\left(|\bar{x} - \mu| > \frac{\sqrt{N}}{m}\frac{\sigma}{\sqrt{N}}\right) \le \alpha = \frac{m^2}{N},$$

так как

$$k = \frac{\sqrt{N}}{m};\ \frac{1}{x^2} = \frac{m^2}{N}.$$

Таблица 8.1. Размеры выборки при различных значениях погрешности

Отклонение	Необходимый объем выборки N	
	по центральной предельной теореме	по теореме Чебышева
σ/2	15	80
σ/4	61	320
σ/6	138	720
σ/8	246	1 280
σ/10	384	2 000
σ/12	553	2 880
σ/20	1 537	8 000

Отсюда находим требуемый размер выборки машинного эксперимента

$$N = \frac{m^2}{\alpha}.$$

Полученный размер выборки существенно больше того, который оказывается достаточным в случае нормального распределения совокупности. Размеры выборки при различных значениях d при $\alpha = 0{,}05\,(\beta = 0{,}95)$ приведены в табл. 8.1.

8.3.4. Оценивание процентных отношений

Во многих случаях цель машинного эксперимента — получение оценки $\hat{p}$ вероятности появления $p = P(A)$ некоторого события A, определяемого состояниями процесса функционирования исследуемой системы. В качестве оценки вероятности p выступает частота $\hat{p} = \theta/N$, где θ — число положительных исходов эксперимента.

Представим частоту $\hat{p} = \theta/N$ в виде

$$\hat{p} = \frac{1}{N}\sum_{i=1}^{N} x_i,$$

где количество наблюдений события A в данной реализации из N наблюдений является случайной величиной, принимающей значения $x_i = 1$ с вероятностью p и $x_i = 0$ с дополнительной вероятностью $1 - p$.

Тогда

$$\bar{X}(N) = p;$$

$$S^2(N) = \frac{p(1-p)}{N}.$$

По ***теореме Лапласа*** частоту θ/N при достаточно больших N можно рассматривать как случайную величину, описываемую нормальным законом распределения вероятностей с математическим ожиданием $\bar{X}(N)$ и выборочной дисперсией $S^2(N)$. Поэтому точность ε и достоверность β оценок связаны с количеством N реализаций равенством

$$P\left(p - \varepsilon < \frac{\theta}{N} < p + \varepsilon\right) = \Phi\left(\frac{p + \varepsilon - p}{\sqrt{p(1-p)}}\sqrt{N}\right) - \Phi\left(\frac{p - \varepsilon - p}{\sqrt{p(1-p)}}\sqrt{N}\right) = \beta,$$

из которого объем выборки, необходимый для получения оценки $\hat{p}$ с точностью ε и достоверностью β, определяется в виде

$$N = \frac{t_\beta^2 p(1-p)}{\varepsilon^2}. \qquad (8.2)$$

При тактическом планировании эксперимента, когда значение p неизвестно, проводят предварительное моделирование для произвольно выбранного небольшого значения n. По результатам предварительного моделирования определяют $p_0 = \theta/n$, а затем по соотношению (8.2) вычисляют необходимое количество реализаций N, используя вместо p значение p_0. Такая процедура оценки N может выполняться несколько раз в ходе машинного эксперимента.

При отсутствии возможности получения каких-либо предварительных сведений о вероятности p использование абсолютной точности теряет смысл. В таких случаях целесообразно задавать относительную точность результатов моделирования

$$\varepsilon_0 = \frac{\varepsilon}{p}.$$

Тогда соотношение (8.2) примет вид

$$N = \frac{t_\beta^2 (1-p)}{\varepsilon_0^2 p}. \qquad (8.3)$$

Формула (8.3) наглядно иллюстрирует специфику статистического моделирования редких событий, выражающуюся в том, что для оценивания малых вероятностей p с высокой точностью необходимо очень большое число реализаций. В практических случаях для оценивания вероятностей порядка 10^{-k} целесообразно количество реализаций выбирать равным хотя бы 10^{k+1}.

8.3.5. Оценивание дисперсии совокупности

В качестве показателя эффективности системы может выступать дисперсия σ^2, оценку s^2 которой требуется получить в результате проведения машинных экспериментов с некоторой надежностью $\beta = 1 - \alpha$.

Потребуем, чтобы выполнялось соотношение

$$P\left(s^2 - d \le \sigma^2 \le s^2 + d\right) = \beta.$$

Преобразуем двойное неравенство

$$s^2 - d \le \sigma^2 \le s^2 + d$$

в равносильное

$$s^2(1-q) \le \sigma^2 \le s^2(1+q),$$

где q — число, характеризующее степень близости оценки s^2 к истинной дисперсии σ^2; $q = d/s^2$.

Для удобства пользования введем в рассмотрение статистику

$$\chi^2 = \frac{s^2(N-1)}{\sigma^2},$$

которая распределена по закону χ^2 («хи-квадрат») с $N-1$ степенями свободы.

Плотность распределения χ^2 имеет вид

$$k_N(x) = \begin{cases} \dfrac{1}{2^{N/2}\Gamma\left(\dfrac{N}{2}\right)} x^{\frac{N}{2}-1} e^{-x/2}, x > 0;\ N \ge 1; \\ 0, x \le 0. \end{cases}$$

Здесь $\Gamma(\lambda)$ — гамма-функция, определенная как

$$\Gamma(\lambda) = \int_0^{\lambda} t^{\lambda-1} e^t \mathrm{d}t.$$

В частности при целом значении аргумента $\Gamma(n+1) = n!$

При больших N согласно центральной предельной теореме распределение нормированной величины $\left(\chi^2 - N\right)/\sqrt{2N}$ стремится к стандартному нормальному распределению

$$P\left(\frac{\chi^2 - N}{\sqrt{2N}} < x\right) \approx \Phi(x),$$

откуда можно получить уравнение

$$t_\beta^2 = \frac{q^2(N-1)}{2}$$

или

$$N = 1 + \frac{2t_\beta^2}{q^2},$$

где t_β — квантиль стандартного нормального распределения, значение которого находится из равенства $2\Phi\left(t_\beta\right) = \beta$ по таблице функции Лапласа.

8.3.6. Автокоррелированные данные

Если выборочные значения на выходе модели автокоррелированы, т. е. последующее выборочное значение зависит от предыдущих, то в такой выборке содержится меньше информации, чем в выборке из независимых данных. При этом необходимый размер выборки весьма чувствителен к величине автокорреляции.

Рассмотрим способ оценки автокорреляционной функции и использование ее при построении оценок параметров, которые в случае коррелированных данных принимают вид

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i;$$

$$\sigma_N^2 = \frac{\sigma_x^2}{N}\left[1 + 2\sum_{p=1}^{m}\left(1 - \frac{p}{m+1}\right)\rho_{p,x}\right],$$

где σ_x^2 — дисперсия совокупности; m — максимальная глубина рассматриваемой автокорреляции; $\rho_{p,x}$ — p-й коэффициент автокорреляции; $p = 1, 2, \ldots, m$.

Для оценивания коэффициентов корреляции необходимо провести эксперимент с пробной выборкой и применить формулу

$$\rho_{p,x} = \frac{\sum_{i=1}^{N-p}\left[(x_i - \bar{x})(x_{i+p} - \bar{x})\right]}{S^2},$$

где S^2 — исправленная выборочная дисперсия, полученная на основе N измерений в пробном эксперименте.

Минимально необходимый размер выборки можно вычислить по формуле

$$N = \frac{t^2 S^2\left[1 + 2\sum_{p=1}^{m}\left(1 - \frac{p}{m+1}\right)\rho_{p,x}\right]}{(\bar{x}d)^2},$$

где d — относительная погрешность истинного среднего.

Максимальную глубину m вычисляемых коэффициентов корреляции выбирают на практике равной примерно 10 % числа N измерений.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое планирование машинного эксперимента?
2. В чем состоит сущность тактического планирования компьютерного эксперимента?
3. Как влияют начальные условия на результаты моделирования?
4. Каким образом может решаться задача определения размера выборки, обеспечивающей заданную точность и минимальную стоимость эксперимента?
5. Каким образом осуществляется оценка среднего значения выборочной совокупности с заданной точностью?
6. Как использовать неравенство Чебышева для оценивания оптимального объема выборки?
7. Каким образом можно определить объем выборки при оценивании вероятности наступления события с заданной точностью?
8. Каким образом можно определить объем выборки при оценивании дисперсии с заданной точностью?
9. Каким образом оценивается объем выборки при автокоррелированных данных?

Глава 9

ПАКЕТЫ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

9.1. НАЗНАЧЕНИЕ ПАКЕТОВ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

При создании имитационных моделей систем можно выделить некоторые типичные функциональные возможности, необходимые для их разработки [5].

К таким возможностям относятся:

- генерирование случайных чисел, равномерно распределенных на интервале от 0 до 1;
- генерирование случайных величин с заданным распределением вероятностей;
- организация продвижения модельного времени;
- сбор выходных статистических данных и создание отчета с результатами и т. п.

Наличие общих функциональных возможностей в моделирующих программах привело к разработке специальных программных пакетов или систем имитационного моделирования, которые позволяют существенно упростить и ускорить процесс разработки имитационных моделей.

Использование пакета имитационного моделирования в сравнении с применением универсального языка программирования дает несколько преимуществ [5].

1. Пакеты имитационного моделирования автоматически предоставляют большинство функциональных возможностей, требующихся для создания имитационной модели, что позволяет существенно сократить время, необходимое для программирования, и общую стоимость проекта.

2. Пакеты имитационного моделирования обеспечивают естественную среду для создания имитационных моделей. Их основные моделирующие конструкции больше подходят для имитаци-

онного моделирования, чем соответствующие конструкции в универсальных языках программирования.

3. Имитационные модели, созданные с помощью пакетов моделирования, обычно проще модифицировать и использовать.

4. Пакеты имитационного моделирования обеспечивают более совершенные механизмы обнаружения ошибок, поскольку они выполняют автоматический поиск ошибок многих типов. И так как модель не требует большого числа структурных компонентов, уменьшаются шансы совершить какую-либо ошибку.

В то же время применение универсальных языков программирования также имеет ряд преимуществ [5].

1. Языки программирования знает большинство разработчиков, чего нельзя сказать о пакетах имитационного моделирования.

2. Скорость выполнения имитационных моделей, написанных на универсальных языках программирования, обычно выше, чем моделей, созданных с помощью пакетов имитационного моделирования.

3. При программировании универсальные языки обеспечивают бóльшую гибкость, чем пакеты имитационного моделирования.

4. Стоимость применяемого программного обеспечения при использовании универсальных языков программирования обычно ниже стоимости пакетов имитационного моделирования.

Таким образом, использование обоих методов имеет свои преимущества, поэтому разработчику моделей следует очень обдуманно подходить к выбору каждого из них.

На сегодняшний день разработано большое число пакетов имитационного моделирования, отличающихся друг от друга степенью универсальности, функциональными возможностями, областью применения и многими другими параметрами. Далее рассмотрим один из пакетов имитационного моделирования под названием Pilgrim, обладающий рядом преимуществ по сравнению с другими пакетами моделирования.

9.2. ПАКЕТ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ PILGRIM

Пакет Pilgrim обладает широким спектром возможностей имитации временно́й, пространственной и финансовой динамики моделируемых объектов [3]. С его помощью можно создавать дискретно-непрерывные модели. В текст модели можно вставлять любые блоки с помощью стандартного языка C++. Модели в системе Pilgrim компилируются и поэтому имеют высокое быстродействие, что очень важно для отработки управленческих решений и адаптивного выбора вариантов в сверхускоренном масштабе времени. Полученный после компиляции объектный код можно встраивать в разрабатываемые программные комплексы или передавать (продавать) заказчику, так как при эксплуатации моделей инструментальные средства пакета Pilgrim не используются.

Приведем представленное в [3] описание пакета имитационного моделирования Pilgrim.

9.2.1. Основные объекты модели

Существуют шесть основных понятий, на которых базируется концепция моделирующей системы.

1. Схему любой имитационной модели можно представить в виде направленного *графа модели*, вершины которого являются компонентами элементарных процессов, а дуги определяют направление потоков заявок и управляющих воздействий в моделируемой системе. Пример графа, описывающего простейшую СМО, показан на рис. 9.1.

2. *Транзакт* — это формальный запрос на какое-либо обслуживание. Транзакт в отличие от обычных заявок, которые рассматриваются при анализе моделей массового обслуживания, имеет набор динамически изменяющихся особых свойств и параметров. Пути миграции транзактов по графу модели определяются логикой функционирования компонентов модели в узлах графа.

Транзакт может выполнять следующие действия:

- порождать группы (семейства) других транзактов;
- поглощать другие транзакты конкретного семейства;
- захватывать ресурсы и использовать их некоторое время, а затем освобождать;
- определять времена обслуживания, накапливать информацию о пройденном пути и иметь информацию о своем дальнейшем пути и о путях других транзактов.

К основным параметрам транзактов относят:

Рис. 9.1. Пример графа модели

- уникальный идентификатор транзакта;
- идентификатор (номер) семейства, к которому принадлежит транзакт;
- наборы различных ресурсов, которые транзакт может захватывать и использовать какое-то время;
- время жизни транзакта;
- приоритет — неотрицательное число; чем больше приоритет, тем приоритетнее транзакт (например, в очереди);
- параметры обслуживания в каком-либо обслуживающем устройстве (включая вероятностные характеристики).

Примерами транзактов являются: требование на перечисление денег; заказ на выполнение работ в фирме; покупатель в магазине.

3. *Узлы графа сети* представляют собой центры обслуживания транзактов (но необязательно массового обслуживания). В узлах транзакты могут задерживаться, обслуживаться, порождать семейства новых транзактов, уничтожать другие транзакты. С точки зрения вычислительных процессов в каждом узле порождается независимый процесс. Вычислительные процессы выполняются параллельно и координируют друг друга. Они реализуются в едином модельном времени, в одном пространстве, учитывают временну́ю, пространственную и финансовую динамику.

Примеры узлов: производственный (ремонтный) участок; касса для продажи билетов; транспортное средство, которое перемещает ресурсы из одной точки пространства в другую; склад ресурсов.

4. *Событием* называется факт выхода из узла одного транзакта. События всегда происходят в определенные моменты времени. Они могут быть связаны и с точкой пространства. Интервалы между двумя соседними событиями в модели — это, как правило, случайные величины. Разработчик модели практически не может управлять событиями вручную (например, из программы). Поэтому функция управления событиями отдана специальной управляющей программе — координатору, автоматически внедряемому в состав модели.

5. *Ресурс* независимо от его природы в процессе моделирования может характеризоваться тремя общими параметрами: мощностью, остатком и дефицитом. Мощность ресурса — это максимальное число ресурсных единиц, которые можно использовать для различных целей. Остаток ресурса — число незанятых на данный момент единиц, которые можно использовать для удовлетворения транзактов. Дефицит ресурса — количество единиц ресурса

в суммарном запросе транзактов, стоящих в очереди к данному ресурсу.

Обычно можно выделить три основных типа ресурсов: материальные, информационные и денежные.

6. *Пространство* — географическое, декартова плоскость (можно ввести и другие). Узлы, транзакты и ресурсы могут быть привязаны к точкам пространства и мигрировать в нем.

Внутренняя реализация модели использует объектно-ориентированный способ представления экономических процессов. Транзакты, узлы, события и ресурсы — основные объекты имитационной модели. Взаимодействие этих объектов определяется структурой графа.

В различных моделирующих системах имеются разные способы представления узлов графа. Это связано с отличительными свойствами таких систем. Например, в системе GPSS узлы называются блоками; причем количество различных типов блоков более сотни, что затрудняет восприятие графа модели. В пакете Pilgrim имеется всего 17 типов узлов, которые функционально перекрывают все возможности блоков GPSS и предоставляют дополнительные средства, которые в GPSS отсутствуют:

- возможность работы с непрерывными процессами;
- моделирование пространственной динамики;
- работу с ресурсами, представляющими собой деньги и материальные ценности, счета бухгалтерского учета, банковские счета.

Имеется система обозначений узлов, помогающая «читать» граф модели. Каждый узел имеет графическое обозначение, функциональное наименование, произвольный уникальный номер и произвольное название (например: наименование — serv, номер — 123, название — «Мастерская»). Пути транзактов обозначаются дугами — сплошными линиями со стрелкой на одном конце.

9.2.2. Типы узлов имитационной модели

Узлы имитационной модели характеризуются параметрами, задаваемыми пользователем при написании моделирующей программы. Параметры узлов могут иметь как постоянное, так и переменное значение. Первым параметром p_1 любого узла является его символическое имя в модели, отражающее логическое назначение узла. Далее описаны некоторые узлы пакета Pilgrim и назначение их параметров.

Узел ag — генератор транзактов. Узел ag представляет собой генератор транзактов с бесконечной емкостью. Он предназначен для создания транзактов имитационной модели. Транзакты генерируются по одному через определенные пользователем промежутки времени.

Для задания генератора используется функция ag(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8). Параметры имеют следующий смысл:

p_1 — символическое имя узла (строка длиной до 14 символов);

p_2 — номер узла-генератора (int);

p_3 — приоритет (int), назначаемый каждому сгенерированному транзакту, — число в диапазоне 1...32 767 (чем число больше, тем приоритет выше); если приоритет не нужен, то p_3 = none;

p_4 — тип функции распределения интервала времени между двумя последовательно сгенерированными транзактами. Используются условные обозначения: norm — нормальное распределение, unif — равномерное, expo — экспоненциальное, erln — распределение Эрланга, beta — треугольное распределение, none — интервал, являющийся детерминированной величиной;

p_5 — математическое ожидание интервала времени (float) между двумя последовательно сгенерированными транзактами (p_4 = norm, unif, expo), либо математическое ожидание одного слагаемого этого интервала (p_4 = erln), либо минимальное значение интервала (p_4 = = beta), либо постоянная величина этого интервала (p_4 = none);

p_6 — величина, зависимая от типа функции распределения (float): среднеквадратичное отклонение (p_4 = norm), либо максимальное отклонение от среднего (p_4 = unif), либо значение zero (p_4 = expo, none), либо число слагаемых интервала (p_4 = erln, в этом случае $p_6 > 0$), либо наиболее вероятное значение интервала времени между двумя последовательно сгенерированными транзактами (p_4 = beta);

p_7 — величина, также зависящая от типа функции распределения (float): либо значение zero (p_4 = norm, unif, expo, erln, none), либо максимальное значение интервала времени между двумя последовательно сгенерированными транзактами (p_4 = beta);

p_8 — номер узла, в который передается сгенерированный транзакт.

Параметры генератора могут быть изменены во время работы модели с помощью команды cheg.

Узел serv — обслуживающий многоканальный прибор с абсолютными приоритетами. Узел serv — это обслуживающий прибор (сервер), осуществляющий какое-либо обслуживание транзактов в течение модельного времени, отличного от нуля. С точки зрения

имитационной модели «обслуживание» заключается в задержке транзакта на заданный промежуток времени. Сервер — это одно- или многоканальный обслуживающий прибор, работающий по правилам абсолютных приоритетов или без них и имеющий стек для «прерванных» транзактов (правило относительных приоритетов реализуется в узле типа queue — очередь).

Сервер задается с помощью узловой функции serv(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8). Параметры имеют следующий смысл:

p_2 — число обслуживающих каналов (int), $1 \le p_2 \le 32\,767$;

p_3 — дисциплина обслуживания: abs — приоритетная, с прерыванием обслуживания менее приоритетного транзакта, либо none — бесприоритетная;

Назначение параметров p_1, $p_4 \ldots p_8$ совпадает с назначением соответствующих параметров узла ag.

Если задать p_3 = abs, то имеются две возможности для работы с прерванными неприоритетными транзактами: либо они «дообслуживаются», либо характер прерывания был таков, что нужно возобновить обслуживание сначала. Например, если осуществляется набор текста на компьютере, то внезапное отключение питания (интерпретируется как приоритетный транзакт) вынуждает начать набор текста (неприоритетный транзакт) заново. По умолчанию «обслуживание» транзакта будет продолжено после прерывания в режиме дообслуживания.

Однако если перед функцией serv будет выполнен оператор присваивания $t \rightarrow$ ga = again, то транзакт получает признак обслуживания заново. После выхода из serv этот признак теряется.

Следует отметить, что среднее время обслуживания транзактов в узле serv и среднее время задержки в нем — разные времена. При прерываниях обслуживание транзакта — это чистое время обслуживания, а во время задержки неприоритетного транзакта в узле входит как время обслуживания, так и время пребывания этого транзакта в прерванном состоянии (т. е. во время обслуживания более приоритетного транзакта).

Узел queue — очередь с относительными приоритетами. Узел queue моделирует очередь транзактов. Эта очередь строится по одному из двух правил: либо транзакты упорядочены в порядке поступления, либо, кроме того, более приоритетные транзакты находятся ближе к началу очереди, а менее приоритетные — ближе к концу. Во втором случае работает правило относительных приоритетов. Для описания данного узла используется функция queue(p_1, p_2, p_3). Причем параметры p_1 и p_3 определяют символическое имя узла и номер следующего узла на графе, а параметр p_2

задает тип организации очереди (если p_2 = prty, то используется очередь с приоритетами, а если p_2 = none, то без).

Узел term — терминатор транзактов. Узел term — терминатор транзактов, назначение которого заключается в том, что он удаляет из модели входящий в него транзакт и фиксирует время его существования, начиная с момента выхода этого транзакта из генератора. Для описания данного узла используется функция term(p_1), параметр p_1 которой определяет символическое имя узла.

Узел key — клапан на пути транзактов. Узел key — клапан, или ключ, — работает по принципу «шлагбаума». Когда клапан закрыт, транзакт не может в него войти из предшествующего узла. Если же клапан открыт, то транзакт проходит через него в следующий узел без задержки. Среднее время пребывания такого узла в закрытом состоянии подсчитывается автоматически. Для управления этим клапаном или ключом существуют команды hold и rels.

Описывается данный узел с помощью функции key(p_1, p_2), параметры которой определяют символическое имя узла и номер следующего узла на графе.

Узел creat — управляемый генератор транзактов. Узел creat предназначен для создания нового семейства транзактов. Заметим, что все транзакты принадлежат какому-то семейству. Транзакты, выходящие из обычного генератора ag, принадлежат к семейству с номером 0. Для описания управляемого генератора используется узловая функция creat(p_1, p_2, p_3, p_4, p_5, p_6). Логика функционирования данного узла такова:

- через узел проходит порождающий транзакт, который принадлежит семейству f_1, и поступает в следующий узел p_6;
- одновременно с этим в узле генерируются p_3 новых транзактов, принадлежащих семейству с номером $f_2 = p_2$, которые будут направлены в узел p_5. Номера семейств f_1 и f_2 в общем случае могут совпадать.

Приведем назначение оставшихся параметров. Параметр p_1 определяет символическое имя узла. Параметр p_4 имеет следующие значения: либо copy — для тиражирования параметров порождающего транзакта каждому порожденному, либо none — для присвоения каждому порожденному транзакту в качестве параметров нулевых значений.

Узел delet — управляемый терминатор транзактов. Узел delet предназначен для уничтожения группы транзактов, принадлежащих некоторому диапазону семейств. Данный узел описывается функцией delete(p_1, p_2, p_3, p_4, p_5, p_6).

Логика функционирования этого узла такова: в узел входит уничтожающий транзакт семейства p_4 и находится там до тех пор, пока в него не поступят p_5 транзактов семейств $p_2 \le$ Number $\le p_3$, которые он мгновенно уничтожает. В данном узле фиксируется время существования поглощенных транзактов. После поступления требуемого количества транзактов заданных семейств уничтожающий транзакт переходит в узел p_6.

Узлы send и direct — операции с бухгалтерскими счетами. Основные объекты пакета Pilgrim (узел, транзакт, событие) очень хорошо подходят для описания финансовой динамики на счетах бухгалтерского учета предприятия (фирмы). Для описания счета (субсчета) бухгалтерского учета используется узел send; предположим, что номер этого узла i. Транзакт, вошедший в узел i, — это запрос на проводку со счета i определенной суммы на какой-то другой счет. Для осуществления проводки необходимо, чтобы на счете i (т. е. в узле i) была сумма не менее требуемой. При отсутствии такой суммы транзакт становится в ожидание момента поступления на счет i достаточных средств. Другими словами, узел с номером i, который формирует запрос на бухгалтерскую проводку, — это специальная очередь транзактов.

Следующим за узлом send обязательно должен стоять узел direct, играющий роль «финансового директора». Узел direct является своеобразным клапаном, через который могут пройти только те транзакты, проводки по которым можно выполнить. В принципе, один узел direct может обслуживать все узлы send, имеющиеся в модели.

Для описания узла send необходимо использовать функцию send(p_1, p_2, p_3, p_4, p_5), параметры которой имеют следующие значения:

p_1 — символическое имя узла;

p_2 — узел-счет, на который переводится заданная сумма;

p_3 — размер заданной суммы (тип double). Единицы измерения финансовых средств — любые (рубли, доллары и т. д.). После точки обязательно необходимо указывать одно или два числа — доли используемых единиц измерений. Например: 1 000 000.00 (один млн руб. 00 коп.), 110.2 (сто десять и две десятые млн руб.);

p_4 — параметр, определяющий возможность работы с приоритетами (значение prty) или строгое упорядочивание проводок по моментам возникновения необходимости в этих проводках (значение none). Если указано значение prty, то чем меньше сумма проводки, тем запрос приоритетнее;

p_5 — номер узла типа direct, который осуществляет финансовый менеджмент и выполняет проводки по мере необходимости.

В каждом узле типа send имеется внутренний атрибут saldo, который отражает остаток средств на счете *i*. Дефицит средств на счетах бухгалтерского учета содержится в атрибуте defic. Если атрибут saldo в узле *i* имеет нулевое значение и в этом узле имеются транзакты (один или несколько), запросившие проводки, то суммарный дефицит затребованных этими транзактами сумм автоматически отражается в атрибуте defic.

Для управления узлом send используется команда assign. С ее помощью можно изменять остаток средств на соответствующем узлу бухгалтерском счете.

Обслуживание очередей типа send возможно с помощью одного или нескольких узлов типа «финансовый директор». Для описания такого узла используется функция direct(p_1, p_2). Здесь параметр p_1 определяет символическое имя узла, а параметр p_2 задает номер узла-приемника транзакта, выполнившего бухгалтерскую проводку.

Узлы attach и manage — работа со складами ресурсов. При построении имитационных моделей в пакете Pilgrim можно учитывать не только время протекания моделируемых процессов, но и использование этими процессами различного вида ресурсов, необходимых для их работы. Под ресурсом могут пониматься любые объекты окружающего мира (сырье, оборудование, персонал и т. д.). Объем ресурса в модели измеряется в целых числах. В модели все виды используемых процессами ресурсов хранятся в специальных узлах-складах attach. При посещении транзактом узла attach ему выдается некоторое количество единиц хранящегося там ресурса. Если ресурса на складе меньше, чем требуется транзакту, он остается в узле до тех пор, пока объем имеющегося ресурса не станет больше, т. е. узел attach является очередью запросов на получение ресурса. Очередь запросов может быть обыкновенной, а может учитывать приоритет запроса. Приоритет тем выше, чем меньше объем требуемого ресурса.

После узла attach в модели должен обязательно стоять узел manage — управляющий складом. Этот узел проверяет наличие требуемого количества ресурса для каждого транзакта, пришедшего в attach. Если ресурса достаточно, транзакт проходит через узел manage дальше. Иначе транзакт остается в узле attach.

Узел attach задается функцией attach(p_1, p_2, p_3, p_4). Смысл ее параметров следующий:

p_1 — символическое имя узла;

p_2 — требуемое число единиц ресурса;

p_3 — флажок учета приоритета запросов — prty или none. Как и для узла send, чем меньше объем запроса, тем выше его приоритет;

p_4 — номер узла типа manage, управляющего данным складом.

Остаток ресурса и дефицит фиксируется в атрибутах узла rsal и rdef. Дефицит равен суммарному объему запросов всех стоящих в очереди в узле attach транзактов (тех, которые не могут быть обслужены из-за нехватки ресурса).

Обслуживанием транзактов занимается узел типа manage — управляющий складом. Функция, описывающая данный узел, имеет вид manage(p_1, p_2). Параметр p_1 данной функции определяет символическое имя узла, а параметр p_2 — номер узла-приемника транзакта.

Полученный на складе ресурс транзакт «носит» по модели и все, что он может с ним сделать — это вернуть обратно на склад. Вернуть ресурс транзакт может из любого узла модели с помощью команды detach. Возвращать ресурс необязательно. Изменить объем имеющегося на складе ресурса можно с помощью команды supply.

9.2.3. Команды управления узлами

Команды управления узлами могут использоваться в любом узле по мере необходимости. Каждый узел модели может управлять и любым другим узлом, и самим собой. Команда срабатывает каждый раз в момент входа транзакта в узел, в котором она указана.

Освобождение заблокированного узла delet — команда freed. Команда freed(p_1) предназначена для борьбы с блокировкой узла delet. Она срабатывает в соответствии со следующей логикой.

Допустим, что в узле delet с номером p_1 «застрял» транзакт (т. е. долгое время не приходит необходимое количество уничтожаемых транзактов). Текущий транзакт, который проходит через функцию freed, мгновенно генерирует вспомогательный транзакт и направляет его в узел p_1. Этот вспомогательный транзакт «выталкивает» застрявший, приводит delet в нормальное состояние, а сам погибает. Факт посещения вспомогательным транзактом фиксируется в delet и отражается на статистических результатах, но время его жизни равно нулю. Застрявший транзакт будет направлен в следующий за delet узел.

Перенастройка генератора транзактов ag — команда cheg. Команда cheg(p_2, p_3, p_4, p_5, p_6, p_7, p_8) предназначена для настройки генератора с номером p_2 на новые значения параметров. Такая перенастройка в смысле модельного времени произойдет «мгновенно». Все параметры этой команды имеют то же назначение, что и параметры настройки генератора ag (символическое имя генератора в команде не указывается).

Управление клапаном key — команды hold и rels. Команды hold(p_1) и rels(p_1) предназначены для управления клапаном с номером p_1 из любых других узлов. После выполнения hold клапан p_1 принимает состояние «закрыт», а после rels — в состояние «открыт».

Управление бухгалтерским счетом — команда assign. Команда assign(p_1, p_2, p_3) помещает денежную сумму p_3 (double) на счет p_1. Если значение кода операции p_2 равно add, то сальдо счета увеличивается на сумму p_3 (режим добавления); если p_2 = none, то сальдо счета становится равным p_3 (режим замещения).

Управление складом ресурсов — команды supply и detach. Команда supply(p_1, p_2, p_3) помещает p_3 (long) единиц ресурса на склад p_1. Если значение кода операции p_2 равно add, то запас ресурса на складе увеличивается на величину p_3 (режим добавления); если p_2 = none, то запас ресурса становится равным p_3 (режим замещения).

Команда detach(p_1, p_2) возвращает на склад с номером p_1 число единиц ресурса, равное p_2. Для корректной работы модели необходимо, чтобы указанный объем ресурса был ранее взят возвращающим транзактом на указанном в команде складе (возвращающим является транзакт, активизирующий команду detach).

9.2.4. Параметры транзактов и параметры состояния узлов

Транзакты имеют параметры, некоторые из которых доступны пользователю. Обращение к параметру транзакта возможно только в момент его входа в узел модели и производится через системную переменную *t*, являющуюся именем входящего в узел транзакта.

Основными параметрами транзакта являются:

$t \to$ iu0, $t \to$ iu1, $t \to$ iu2, $t \to$ iu3 — произвольные целочисленные параметры пользователя (используются для хранения информации о состоянии транзакта);

$t \to$ ru0, $t \to$ ru1, $t \to$ ru2, $t \to$ ru3 — произвольные параметры пользователя, имеющие вид переменной с плавающей точкой;

$t \to$ ga — признак обслуживания транзакта вновь, который устанавливается операцией присваивания ему глобальной константы again, как это было показано при рассмотрении функции serv;

$t \to$ pr — приоритет транзакта;

$t \to$ ft — номер семейства, к которому принадлежит транзакт.

Изменять, не опасаясь непредсказуемого поведения модели, можно только $t \to$ iu0, $t \to$ iu1, $t \to$ iu2, $t \to$ iu3, $t \to$ ru0, $t \to$ ru1, $t \to$ $\to$ ru2, $t \to$ ru3, $t \to$ ga и, иногда, $t \to$ pr и $t \to$ ft. Изменение других параметров следует доверить только Pilgrim.

Узлы модели в пакете Pilgrim также имеют параметры. Некоторые параметры узлов доступны пользователю для анализа (но не для их изменения). Параметры содержатся в системном массиве addr. Доступ к параметру осуществляется с помощью выражения вида

```
addr[<n>]-> <параметр>,
```

Таблица 9.1. Параметры узлов

Имя	Назначение параметра
nc	Число каналов в узле
na	Число транзактов, прошедших через узел на данный момент модельного времени
tn	Число транзактов, находящихся в узле на данный момент модельного времени
ts	Среднее время обслуживания, подсчитанное на данный момент
op	Признак состояния узла типа key. Это целочисленная переменная, принимающая значение true, если клапан открыт, или false, если он закрыт
se	Признак состояния узла типа delet. Это целочисленная переменная, принимающая значение nil, если в узле нет уничтожающего транзакта
saldo, defic	Остаток и дефицит средств на бухгалтерском счете, связанном с узлом типа send
rsal, rdef	Остаток и дефицит ресурса на складе, связанном с узлом типа attach

где <n> — номер узла в модели.

Имена и назначения параметров узлов приведены в табл. 9.1.

Для анализа параметров любого узла можно использовать его номер <n>. Например, если транзакту необходимо проанализировать состояние узла типа delet, имеющего номер 5, и, в случае отсутствия уничтожающего транзакта, выполнить какой-то оператор, он должен пройти через следующее выражение:

```
if (addr[5]->se == nil) <оператор>;
```

При этом сам транзакт в это время может находиться в произвольном узле.

9.2.5. Табличные результаты моделирования

Отчет о результатах моделирования выводится в виде таблицы, состоящей из одной или нескольких страниц. Таблица содержится в отдельном файле, имя которого указывается автором модели в моделирующей программе. Расшифровка полей этой таблицы дана в табл. 9.2.

Кроме этих стандартных средств Pilgrim у пользователя всегда есть возможность организовать печать своих документов либо вывода своих результатов как в процессе моделирования, так и по его завершении. Это делается с помощью пользовательских операторов или функций, написанных на языке C++ и размещенных в надлежащих точках Pilgrim-программы.

Таблица 9.2. Назначение полей таблицы с результатами моделирования

Столбец	Расшифровка
№	Номер узла
Тип узла	Тип узла (ag, serv, key, queue, term, creat, delet, proc, dyna, send, direct, attach, manage)
Точка	Номер последней точки пространства, в которой находится узел типа creat, delet или proc в момент окончания моделирования
Загрузка, % Путь, км	Подсчитывается коэффициент использования транзактами узлов типа serv, key или proc в процентах. Узел key используется, когда через него проходят транзакты. Для узла attach — это коэффициент использования ресурса

Окончание табл. 9.2

Столбец	Расшифровка
M [t] среднее	Среднее значение времени задержки транзакта в узле или иной интервал времени, зависящий от типа узла: 1) для serv — это среднее время пребывания в узле (оно может быть больше времени обслуживания у неприоритетных транзактов при использовании правила абсолютных приоритетов); 2) для queue, send и attach — это среднее время задержки в очереди; 3) для ag — это среднее время между двумя сгенерированными транзактами; 4) для term или delet — это среднее время существования уничтоженных транзактов; 5) для key — это среднее время пребывания в закрытом состоянии
C^2 [t] вариация	Отношение дисперсии времени задержки к квадрату среднего значения времени задержки (коэффициент вариации, возведенный в квадрат)
Счетчик входов	Число транзактов, прошедших через узел, либо число сгенерированных транзактов (для ag или creat), либо уничтоженных (для term или delet). Для key — число выданных команд hold
Количество каналов	Число каналов в узле
Остаток транзактов	Количество транзактов, которые остались в узле в момент завершения моделирования
Состояние узла в этот момент	Состояние узла в момент окончания прогона модели: узел может быть «открыт» или «закрыт» для входа очередного транзакта (узлы serv и key). Сервер открыт, есть в нем есть хотя бы один свободный канал. Для узлов send и attach в данном поле таблицы указывается остаток и дефицит ресурса (литеры «S» и «D» соответственно)

9.2.6. Язык описания моделей

Общая структура Pilgrim-программы. Структура программы, описывающей имитационную модель в пакете Pilgrim, имеет следующий вид:

```
#include <pilgrim.h>
[описание констант пользователя]
[описание переменных пользователя]
forward
{
  modbeg(…);
  ag(…);
  …
  [начальные команды assign и supply]
  [стартовые функции пользователя]
  network(…)
  {
    <блок описания узла>
    …
    fault(…);
  }
  [финишные функции пользователя]
  modend(…);
  return 0;
}
```

Первой строкой модели является команда подключения системного файла Pilgrim. Далее пользователь может описать необходимые ему константы и переменные в соответствии с правилами языка C++.

После функции modbeg следуют описания всех генераторов ag. Другие узлы модели описываются с помощью блоков описания узла. Описывать узлы можно в любом порядке, независимо от нумерации.

Инициализация модели — функция modbeg. Оператор modbeg(p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9) осуществляет первоначальную настройку моделирующих программ. Аргументы этой функции имеют следующий смысл:

p_1 — символическое имя модели;

p_2 — максимальный номер узла модели(int), $2 \le p_2 \le 1\,024$;

p_3 — модельное время, в течение которого необходимо производить моделирование (float);

p_4 — произвольное целое число, используемое для настройки датчиков псевдослучайных величин (long). В качестве p_4 полезно использовать значение таймера ЭВМ, обращение к которому имеет следующий вид: (long)time(NULL). В этом случае результаты моделирования, проводимого в различное время, будут разными;

p_5 —признак режима пространственной имитации: earth — поверхность Земли, plane — декартова плоскость, cosmos — произвольное пространство. Можно использовать none, если пространственная имитация не используется;

p_6 — номер одной из очередей (узел типа queue), которую необходимо контролировать во времени для анализа динамики задержек в этой очереди с графическим отображением результатов. Например, если указать число 3 и в модели действительно имеется очередь с таким номером, то можно получать изображение динамики этой очереди непосредственно в процессе моделирования;

p_7 — номер одного из процессов (узел типа proc), который необходимо контролировать как в пространстве, так и во времени с графическим отображением результатов. Если нет необходимости в графической интерпретации пространственной имитации, то указывается none;

p_8 — номер терминатора (узел типа term), на входе которого необходимо наблюдать интенсивность потока транзактов во время моделирования. Если такой необходимости нет, то указывается none;

p_9 — точность (число int от 1 до 6 либо none), т.е. количество знаков после запятой при выводе результатов, соответствующих интервалам времени. Если p_9 = none, то результаты будут округляться до целых значений.

Запуск координатора модели — функция network. Функция network(p_1, p_2) осуществляет диспетчеризацию транзактов в ветвях модели, планирует события в едином модельном времени и активизирует непрерывные компоненты модели. С точки зрения синтаксиса языка программирования эта функция является структурным оператором, за которым следует фигурная скобка {, открывающая блок с описанием графа модели. Обращение к этой функции делается только после того, как выполнены функция modbeg и все функции ag. Аргументы p_1 и p_2 — это имена (адреса) подпрограмм, производящих интегрирование, решение разностных уравнений, вычисление по формулам и т.д. В простейших моделях эти функции не нужны и в качестве параметров p_1 и p_2 указывается слово dummy. Выполнение всего моделирующего ком-

плекса планируется либо до первой ошибки, либо пока не истечет время, заданное аргументом p_3 в функции modbeg.

Обработчик ошибок — функция fault(err). В самом конце описания графа необходимо записать функцию fault(err), которой управление будет передано в том случае, если из-за невнимательности пользователя произойдет попытка перехода транзакта в несуществующий узел. После такой попытки моделирование прекратится и будет завершено с кодом err, где err — произвольно задаваемое пользователем число, обычно в пределах от 101 до 32 767. При этом пользователю будет выдано сообщение, в каком узле произошла эта ошибка и чему равен ошибочный номер следующего узла.

Конец модели — функция modend. Функция modend(p_1, p_2, p_3, p_4) должна выполняться после того, как будет нарушено внутреннее условие noerr, определяющее, что в модели либо зафиксирована грубая ошибка, либо истекло время моделирования, указанное в операторе modbeg. Она удаляет все созданные в процессе моделирования управляющие структуры из памяти ЭВМ. Кроме того, эта функция позволяет просмотреть на экране монитора графические результаты и выводит итоговые результаты в файл-отчет. Значения параметров:

p_1 — имя файла, содержащего таблицу результатов моделирования;

p_2 — номер первой страницы отчета;

p_3 — число строк на каждой странице;

p_4 — имеет два значения: либо page, если в файле-отчете необходимо проставить символ перевода страницы, либо none.

Стартовые и финишные функции пользователя. Между функциями modbeg и network, когда модельный таймер содержит нулевое значение, а также после описания блока узлов графа, законченного закрывающей фигурной скобкой } — перед функцией modend, когда модельный таймер содержит конечное значение, — можно использовать любую программу на языке C++ или обратиться к любой функции, написанной на C++, которые сработают до или соответственно после прогона основной модели.

Блок описания узла. Структура блока описания узла имеет следующий вид:

```
top( <n>): [ <операторы пользователя>]
             <тип узла>(<параметры узла>);
             [clcode <оператор пользователя>]
             place;
```


Начиная с top(<n>) транзакт проходит все операторы без задержек (в смысле модельного времени) и попадает в оператор place, где находится до тех пор, пока не появятся условия для перехода в следующий узел. В некоторых узлах одновременно могут находиться несколько транзактов. Все они находятся в одном и том же операторе place. Задержка транзакта в place — это и есть время пребывания в узле. Операторы, стоящие перед указанием типа узла (узловой функцией), — это те действия, которые транзакт должен выполнить перед входом в узел. Операторы, стоящие после метки clcode, — это действия, которые нужно выполнить после входа. Все переменные — параметры узла должны быть определены перед входом транзакта в узел.

Операторы, стоящие перед типом узла, могут быть выполнены несколько раз при попытках транзакта войти в запертый узел (занятый сервер или закрытый ключ). В процессе работы модели координатор сети периодически «опрашивает» запертые узлы, проверяя, не освободились ли они. Во время «опроса» происходит автоматическое выполнение операторов, стоящих перед типом узла. Таким образом, в этом месте нельзя писать те операторы, повторное выполнение которых приведет к ошибке в работе модели. Операторы, стоящие после clcode, выполняются только один раз.

В блоке описания узлов можно помещать любые наборы операторов языка C++ и команд пакета Pilgrim, а также производить обращения к параметрам транзактов и параметрам состояния узлов, фрагментам программ, к функциям, написанным на языке C++, к системным вызовам. Однако это нужно делать при соблюдении определенных правил.

Во-первых, нельзя использовать операцию языка C++ типа goto не только внутри одного узла (между меткой-функцией top и соответствующим оператором place), но и для перехода в другой узел. Для принудительного завершения моделирования вместо goto можно использовать только операцию

```
error = код,
```

где «код» — любой код ошибки пользователя, удовлетворяющий условию «код» > 100.

После такой операции управление будет отобрано у координатора и передано оператору, следующему за блоком описания графа (например, функции modend).

Во-вторых, вследствие возможности многократного выполнения, между меткой top и типом узла запрещено использовать рекурсию типа $x = f(x)$ или $x = x + 1$ и т. д., а также нежелательно пи-

сать ресурсоемкие вычисления, требующие существенных затрат времени процессора ЭВМ. Желательно использовать только операцию присваивания и, если необходимо, операцию if или switch.

Например:

```
top(3): if (a>0) b = 4;
        else b = 5;
        key("Ключ",b);
        place;
```

Если имеется необходимость производить ресурсоемкие вычисления, использовать рекурсии с некоторыми переменными (например, $x = f(x)$ или $x = x + 1$), работать с системными вызовами, обращаться к ресурсоемким вычислительным функциям, то это делается между функцией определения типа узла и соответствующим оператором place с помощью специального средства clcode.

После оператора определения типа можно записать блок любых операторов языка C++, оформленный в виде

```
clcode
{
  <блок операторов пользователя>
}
```

или отдельный оператор

```
clcode <оператор пользователя>
```

9.3. ГРАФИЧЕСКИЙ КОНСТРУКТОР GEM

При разработке модели в системе имитационного моделирования Pilgrim моделирующую программу удобно создавать с помощью графического конструктора Gem, который позволяет создать граф модели, настроить общие параметры моделирования и параметры узлов графа, сформировать код модели на языке C++. В дальнейшем этот код используется в проекте, созданном в системе Microsoft Visual Studio. Рассмотрим основные принципы разработки модели с помощью графического конструктора Gem.

При запуске графического конструктора появляется главное окно программы (рис. 9.2). Данное окно содержит главное меню и две панели инструментов, на одной из которых расположены эле-

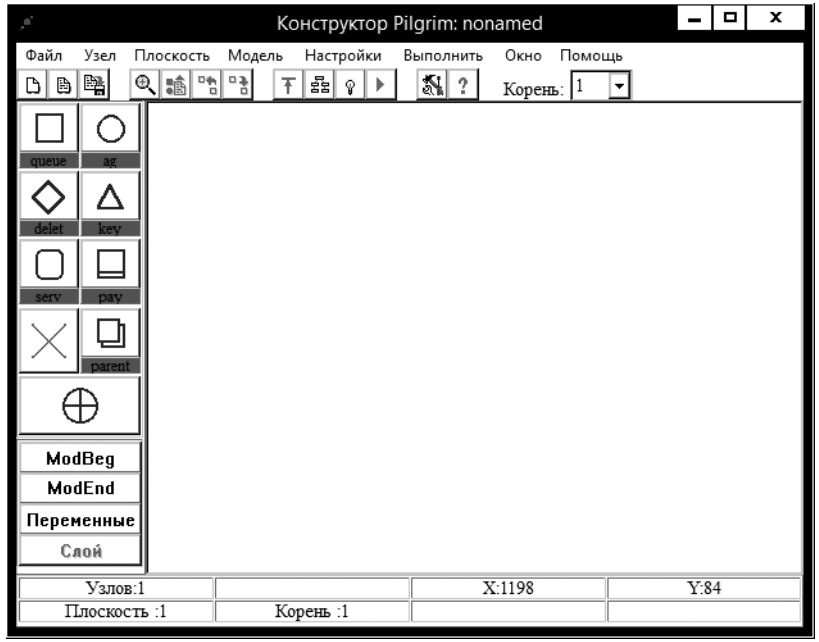


Рис. 9.2. Главное окно графического конструктора Gem

менты, из которых может состоять граф модели. При этом одна кнопка может соответствовать нескольким типам узлов (конкретный тип узла выбирается в параметрах модели). Например, кнопка *queue* позволяет задать узлы типа queue, sent и attach, кнопка *ag* — задать узлы типа ag и term, кнопка *delete* — задать узлы типа delete и create, кнопка *key* — задать узлы типа key, manager и direct, кнопка *pay* — задать узлы типа pay, rent и down.

Для создания новой модели необходимо выполнить команду меню *Файл/Создать* или нажать кнопку *Новая модель* на панели инструментов. После этого будет создана новая плоскость, на которой можно создавать граф модели. Для того чтобы поместить узел модели на плоскости, необходимо перетащить его с панели инструментов на плоскость. После этого с помощью команды *Параметры узла* контекстного меню блока (или с помощью двойного щелчка мыши на блоке) можно открыть окно по настройке параметров узла. Для задания связей между узлами необходимо использовать кнопку с синим крестом в круге. Эту кнопку нужно перетащить на узел, из которого выходит связь, а затем щелкнуть

мышью на узле, в который эта связь входит. Для удаления элементов графа используется кнопка с красным крестом. Ее нужно перетащить на объект, который необходимо удалить.

Для задания общих параметров моделирования используется кнопка *ModBeg* на панели инструментов, при нажатии которой открывается окно с параметрами (рис. 9.3). Здесь задаются название модели (поле *Название*), время моделирования в единицах модельного времени (поле *Время*), номер узла типа queue, динамика задержек в которой будет отображаться при моделировании (поле *Задержка*), номер узла типа term, динамика входного потока в который будет отображаться при моделировании (поле *Поток*), задается число знаков после запятой для получаемых результатов (поле *Точность*) и указываются команды C++ , которые выполняются на начальном этапе работы модели (поле *Начальный C++ текст*).

С помощью кнопки *ModEnd* вызывается окно, в котором можно задать имя файла, в который будут выводиться результаты моделирования, и указать параметры вывода.

С помощью кнопки *Переменные* можно определить пользовательские переменные, которые будут использоваться при моделировании, и указать их тип и начальное значение.

Пример графа модели, созданного в графическом конструкторе Gem, показан на рис. 9.4.

После создания графа модели можно проверить его корректность с помощью команды меню *Выполнить/Проверка корректности графа* и сгенерировать C++-файл с помощью кнопки *Выполнить/Генерировать C++-файл*. Данный C++-файл можно использовать в проекте Visual C++ .

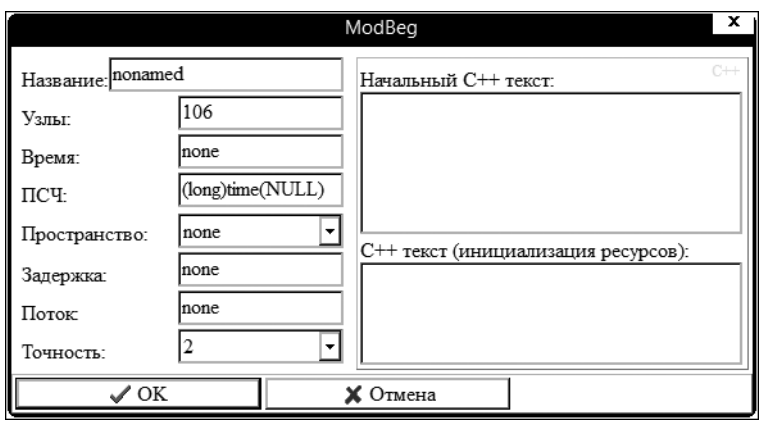


Рис. 9.3. Окно задания общих параметров работы модели

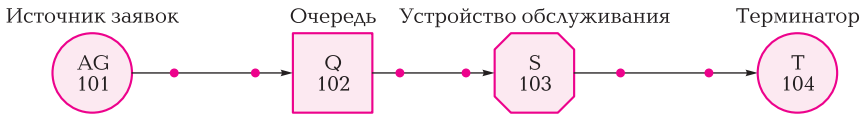


Рис. 9.4. Пример графа модели СМО, созданного в графическом конструкторе Gem

Сформированную модель можно сохранить с помощью команды меню *Файл/Сохранить* в виде pgf-файла для дальнейшего использования.

9.4. ПРИМЕР РАЗРАБОТКИ ИМИТАЦИОННОЙ МОДЕЛИ

Рассмотрим процесс разработки модели СМО (нотариальная контора с двумя нотариусами), характеристики которой были вычислены аналитически в подразд. 7.3. Для удобства еще раз приведем описание данной системы.

Пусть нотариальная контора представляет собой двухканальную СМО. Число мест в комнате ожидания очереди к нотариусу ограничено и равно трем. Если все места в комнате ожидания заняты, то вновь прибывший клиент в очередь не становится. Поток клиентов, прибывающий на консультацию, является простейшим с интенсивностью $\lambda = 12$ клиентов в час. Время обслуживания распределено по экспоненциальному закону со средним временем обслуживания $t = 7$ мин.

Сначала выполним анализ задачи и создадим граф модели. После этого проведем вычислительный эксперимент и анализ его результатов.

В данной задаче есть заявки только одного типа — клиенты. Эти заявки будут формироваться с помощью узла — генератора транзактов. Поскольку поток клиентов простейший, то закон распределения интервала времени между приходом клиентов является экспоненциальным со средним значением $60/\lambda$ мин.

Часть из сформированных в генераторе транзактов заявок попадают в очередь на обслуживание (если число клиентов в очереди меньше трех), а часть сразу покидают систему, попадая в терминатор необслуженных заявок. Для организации такого выбора

можно использовать вспомогательный узел, например, типа key, заявки из которого будут попадать в очередь, если поле tn параметров узла очереди меньше трех, и в терминатор необслуженных транзактов в противном случае (эти условия задаются в параметрах клапана в поле *Условие перехода* при выборе соответствующего перехода, как показано для выхода на очередь на рис. 9.5).

После узла очередь заявки попадают на устройство обслуживания, имеющее два канала обслуживания. По условию задачи время обслуживания распределено по экспоненциальному закону со средним временем обслуживания 7 мин (исходные данные задаются в одном модельном времени). После устройства обслуживания заявки должны попасть в терминатор обслуженных заявок.

На основе проведенного анализа задачи с помощью графического конструктора Gem можно построить граф модели, показанный на рис. 9.6.

После этого задаем общие параметры модели с помощью кнопки *ModBeg* (рис. 9.7) и, при необходимости, параметры для вывода результата с помощью кнопки *ModEnd*.

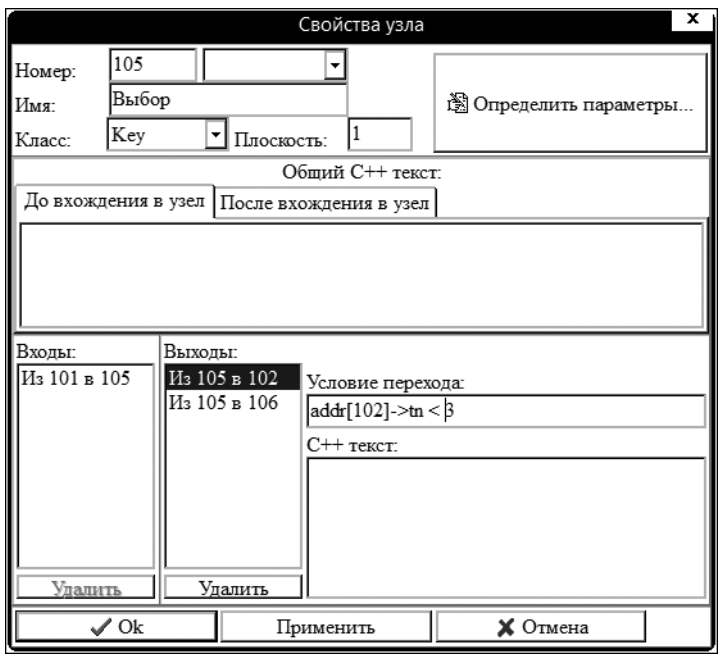


Рис. 9.5. Параметры клапана

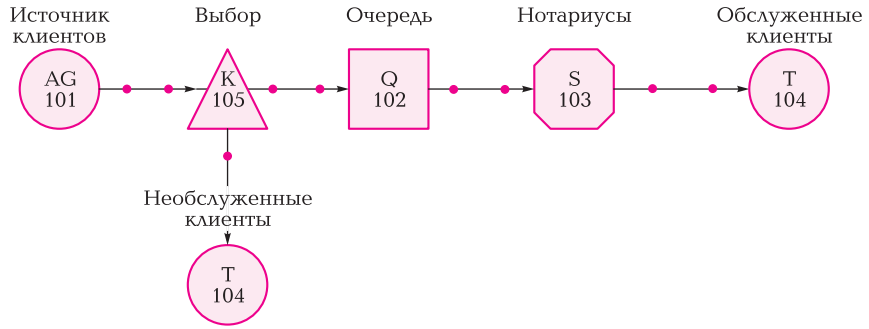


Рис. 9.6. Граф модели

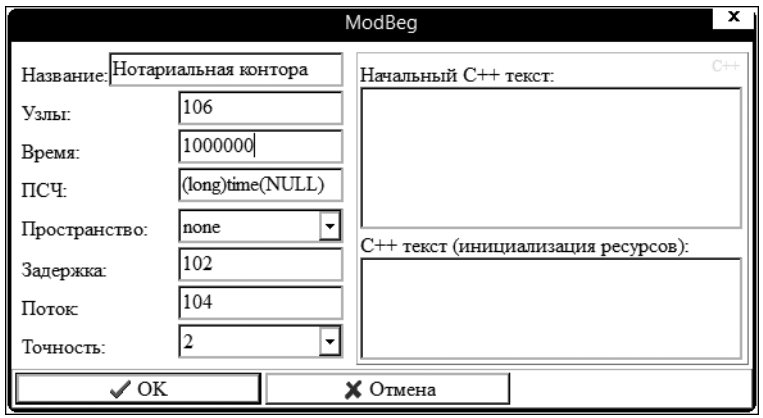


Рис. 9.7. Общие параметры модели

Далее проверяем корректность графа модели с помощью команды меню *Выполнить/Проверка корректности графа* и при отсутствии ошибок формируем C++-файл с помощью команды меню *Выполнить/Генерировать C++-файл*. Модель желательно сохранить в pgf-файле с помощью команды меню *Файл/Сохранить* для того, чтобы в дальнейшем при необходимости ее можно было изменять.

Текст полученного C++-файла представлен далее.

```
#include <Pilgrim.h>
forward
{
  int fw;
```

```
  modbeg("Нотариальная контора", 107, 1000000,
         (long)time(NULL), none, 102, none, 104, 2);
  ag("Источник клиентов", 101, none, expo, 60.0 / 12,
         none, none, 105);
  network(dummy, dummy)
  {
    top(102):
      queue("Очередь", none, 103);
      place;
    top(103):
      serv("Нотариусы", 2, none, expo, 7.0, none,
        none, 104);
      place;
    top(104):
      term("Обслуженные клиенты");
      place;
    top(105):
      if(addr[102]->tn < 3 )
      {
        fw = 102;
      }
      else
      {
        fw = 106;
      }
      key("Выбор", fw);
      place;
    top(106):
      term("Необслуженные клиенты");
      place;
    fault(123);
  }
  modend("pilgrim.rep", 1, 8, page);
  return 0;
}
```

Создание и запуск проекта модели Pilgrim. Имитационная модель создается в среде Microsoft Visual C++. Рассмотрим процесс создания модели на примере Visual Studio 2000. Сначала необходимо создать проект модели. Для этого выбирается подменю *New...* меню *File* оболочки Visual C++ , после чего в появившемся окне на вкладке *Projects* выбирается Win32 Application. В этом же окне следует указать имя проекта и его местоположение на диске. Затем нажатием кнопки *OK* создается пустой проект.

Далее в проект нужно добавить файл, содержащий текст модели. Для этого можно выбрать подменю *New...* меню *File* оболочки Visual C++ , поле чего в появившемся окне на вкладке *Files* выбрать *C++ Source File*, ввести имя файла, отметить галочкой флажок *Add to project* и после этого щелкнуть кнопку *OK*. В созданном файле необходимо набрать текст модели (например, представленный ранее) или сканировать его из заранее подготовленного файла.

Затем к проекту подключаются библиотеки Comctl32.lib (C:\Program Files\Microsoft Visual Studio\VC98\Lib\COMCTL32.LIB), Pilgrim.lib (C:\Program Files\Microsoft Visual Studio\VC98\Lib\PILGRIM.LIB), файл ресурсов Pilgrim.res (C:\Program Files\Microsoft Visual Studio\VC98\projects\PILGRIM.RES) и заголовочные файлы Pilgrim.h (C:\Program Files\Microsoft Visual Studio\VC98\Include\Pilgrim.h) и Simulate.h (C:\Program Files\Microsoft Visual Studio\VC98\Include\ SIMULATE.H). Для подключения данных файлов можно выбрать меню *Project* → *Add To Project* → *Files...*, после чего в появившемся окне выбрать нужный файл и нажать *OK*. Для Visual Studio 2005 и 2008 в свойствах проекта (вызывается комбинацией клавиш Alt + F7) выбрать свойство *Configuration Properties/Linker/Input* и в поле *Ignore Specific Library* занести текст «libc.lib».

Далее выполняются компиляция и сборка модели: меню *Build* → *Rebuild All*. Если в тексте модели нет ошибок, то будет создан исполняемый файл, который можно запускать как самостоятельно, так и из среды Visual C++ с помощью меню *Build* → → *Execute*.

При запуске модели появляется главное окно (рис. 9.8), с помощью элементов меню которого можно управлять процессом моделирования. Далее приведено назначение наиболее важных пунктов меню.

Моделирование → *Запуск модели* (F4) — начать моделирование.

Моделирование → *Приостановка* (F5) — остановить моделирование.

Моделирование → *Продолжение* (F6) — возобновить моделирование.

Результаты → *Параметры узла* (F9) — выбрать узел, параметры которого будут отображаться в правой части окна модели.

Результаты → *Динамика задержек в очереди* (F10) — позволяет отображать динамику задержек в очереди, заданной параметром p_6 оператора modbeg (рис. 9.9).

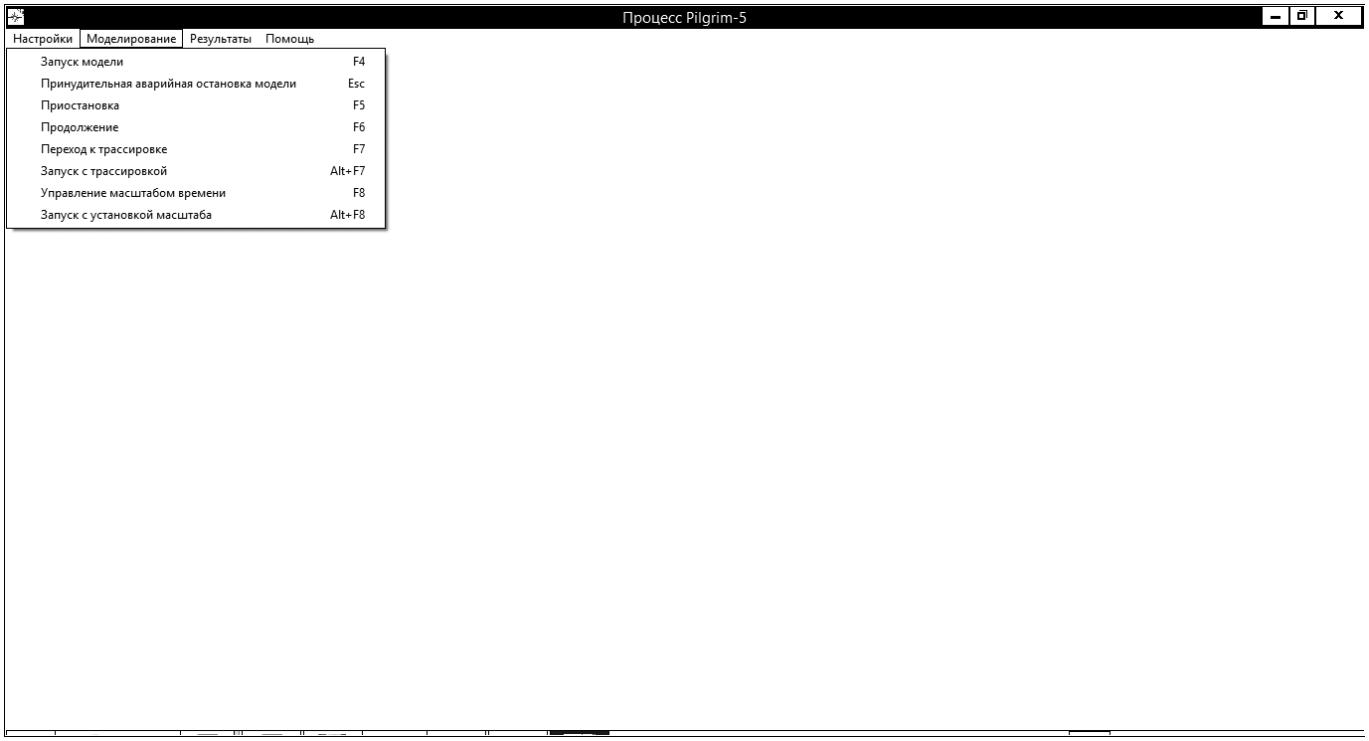


Рис. 9.8. Окно модели Pilgrim

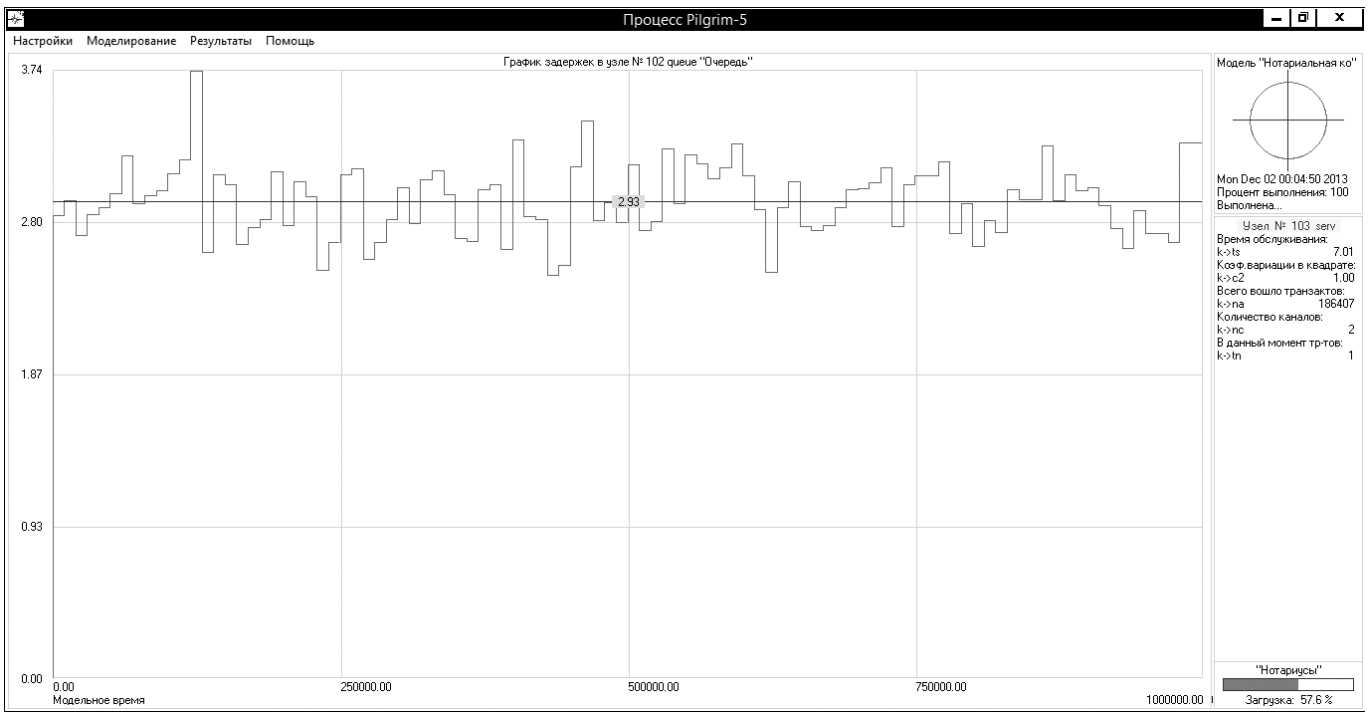


Рис. 9.9. Окно модели Pilgrim

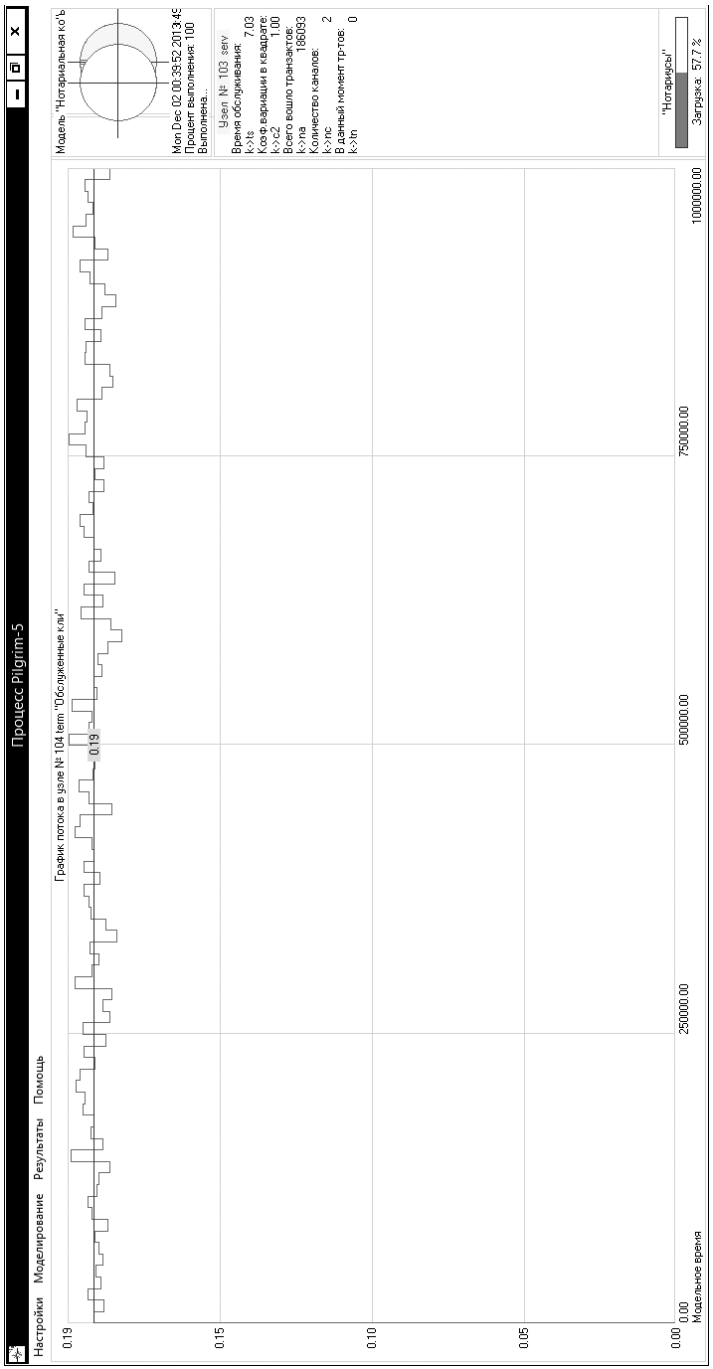


Рис. 9.10. Динамика потока в узле *Очередь*


```
*-------------------------------------------------------------------------------------------------*
|  НАЗВАНИЕ МОДЕЛИ:     Нотариальная контора                                                      |
|  ВРЕМЯ МОДЕЛИРОВАНИЯ: 1000003.70                                                  Лист: 1       |
|-------------------------------------------------------------------------------------------------|
|    |               |      |    |       |         |    2     |       |    |    |               |
| No | Наименование  | Тип  |Точ-|Загруз-|  M [t]  |  C [t]   |Счетчик|Кол.|Ост.|Состояние узла|
|узла|     узла      | узла | ка |ка(%=),|  среднее |квадрат   |входов |кан.|тр. |в этот момент |
|    |               |      |    |Путь(км)|  время  |коэф.вар. |и hold |    |    |               |
|----+---------------+------+----+-------+---------+----------+-------+----+----+---------------|
|    |               |      |    |       |         |          |       |    |    |               |
| 101 Источник клиент ag       -      -        5.00       1.00  200166    1    1         открыт |
| 102 Очередь         queue    -      -        2.93       2.73  186407    1    0         открыт |
| 103 Нотариусы       serv     -  %= 57.6      7.01       1.00  186407    2    1         открыт |
| 104 Обслуженные кли term     -      -        9.94       0.73  186406    0    0         открыт |
| 105 Выбор           key      -  %=  0.0      0.00       1.00       0    1    1         открыт |
| 106 Необслуженные к term     -      -        0.00     999.99   13758    0    0         открыт |
|    |               |      |    |       |         |          |       |    |    |               |
*-------------------------------------------------------------------------------------------------*
```

Рис. 9.11. Результаты моделирования

Результаты → *Динамика потока* (F11) — позволяет отображать динамику потока транзактов на входе терминатора, определенного параметром p_8 оператора modbeg (рис. 9.10).

Табличные результаты моделирования для созданной модели приведены на рис. 9.11.

Анализ результатов моделирования. В таблице результатов прогона модели (см. рис. 9.11) мы видим статистические данные о работе модели за период, равный 1000003,7 ч. Для проверки исходных данных можно узнать среднее время между приходом клиента и среднее время обслуживания. Они составили 5 мин и 7,01 мин (поле $M[t]$ узлов 101 и 103), что соответствует исходным данным. За это время средняя загрузка нотариуса составила 57,6 % (загрузка узла 103). Среднее время ожидания клиентом в очереди составило 2,93 мин ($M[t]$ узла 102). Среднее время пребывания в системе для обслуженных заявок оказалось равным 9,94 мин (поле $M[t]$ узла 104).

Всего в нотариальную контору пришло 200 166 клиентов (поле *Счетчик входов* узла 101). Из них отказ в обслуживании получили 13 758 (поле *Счетчик входов* узла 106). Таким образом, вероятность отказа в обслуживании составила 13 758/200 166 = 0,0687.

Напомним, что аналитический расчет давал следующие результаты.

1. Среднее время пребывания в очереди — 2,92 мин.
2. Среднее время пребывания в системе — 9,92 мин.

3. Вероятность отказа в обслуживании — 0,069.

Таким образом, полученные результаты моделирования хорошо согласуются с ранее выполненными результатами аналитического расчета.

Следует отметить, что модель, созданная в пакете моделирования Pilgrim, позволяет легко изменять параметры модели. Например, можно увеличить или уменьшить число каналов обслуживания, число мест в очереди, изменить закон распределения времени между приходом клиентов и закон распределения времени обслуживания и их параметры. Причем сделать это можно как в графическом конструкторе Gem, заново сгенерировав C++-файл, так и в уже сформированном коде программы, поменяв значение параметров соответствующих узловых функций.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего нужны пакеты имитационного моделирования?
2. Какие преимущества и недостатки имеет использование пакетов имитационного моделирования и универсальных языков программирования при разработке имитационных моделей?
3. Какими возможностями обладает пакет имитационного моделирования Pilgrim?
4. Из каких основных объектов состоит модель в пакете Pilgrim?
5. Какие основные типы узлов модели в пакете Pilgrim вы знаете?
6. Какими параметрами обладает транзакт в пакете Pilgrim?
7. Какими параметрами обладает узел в пакете Pilgrim?
8. Какова общая структура программы в пакете Pilgrim?
9. В каком виде выводятся результаты работы модели в пакете Pilgrim?
10. Что позволяет делать графический конструктор Gem?

Глава 10

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

ЛАБОРАТОРНАЯ РАБОТА № 1. ИЗУЧЕНИЕ БАЗОВЫХ ГЕНЕРАТОРОВ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ

Составьте и отладьте программу (подпрограмму) генерирования псевдослучайных чисел с равновероятным распределением на интервале [0; 1). Вариант выполняемого задания выберите из табл. 10.1, в которой указаны тип генератора случайных чисел, начальные условия и пр. Для заданных объема выборки и числа участков разбиения интервала [0; 1) постройте гистограмму частот и статистическую функцию распределения, получите программным способом оценки матожидания, дисперсии, второго и третьего моментов. Выполните анализ полученных результатов.

Таблица 10.1. Вариант заданий к лабораторной работе № 1

Номер варианта	Тип датчика	Начальные данные	Объем выборки	Число участков разбиения
1	Мультипликативный, формула (2.4)	$Y_0 = 4\,003$, $M = 4\,096$	256	16
2	Универсальный, формула (2.9), $k = 3$	Y_k — любые	500	16
3	Аддитивный, формула (2.6)	$Y_1 = 3\,215$, $Y_2 = 4\,073$, $m = 4\,096 \cdot 4$	5 000	10
4	Универсальный, формула (2.9), $k = 1$	Y — любое	1 600	18

Продолжение табл. 10.1

Номер варианта	Тип датчика	Начальные данные	Объем выборки	Число участков разбиения
5	Квадратичный конгруэнтный метод, формула (2.11)	Y — любое, $I = 12$	5 000	25
6	Квадратичный метод Ковэю, формула (2.12)	$I = 9$, Y_1 из условия $Y_1 \bmod 4 = 2$	5 000	12
7	Квадратичный конгруэнтный метод, формула (2.11)	Y — любое, $I = 12$	7 000	16
8	Метод Макларена — Марсальи	$k = 256$	1 000	21
9	Смешанный, формула (2.5)	Y — любое	4 000	16
10	Аддитивный, формула (2.6)	$Y_1 = 4\,091$, $Y_2 = m - 5$, $m = 4\,096 \cdot 4$	1 000	16
11	Обобщенный аддитивный, формула (2.7)	$r = 6$, $x_1, \ldots, x_6$ из таблицы случайных чисел	6 000	16
12	Обобщенный аддитивный, формула (2.7)	$r = 10$, $x_1, \ldots, x_{10}$ из таблицы случайных чисел	6 000	16
13	Аддитивный, формула (2.6)	$Y_1 = 3\,971$, $Y_2 = 1\,013$, $m = 4\,096 \cdot 4$	2 000	21
14	Метод Макларена — Марсальи	$k = 64$	2 000	16
15	Смешанный, формула (2.5)	$Y = 3\,845$	1 000	16
16	Обобщенный аддитивный, формула (2.7)	$r = 8$, $x_1, \ldots, x_8$ из таблицы случайных чисел	6 000	26

Окончание табл. 10.1

Номер варианта	Тип датчика	Начальные данные	Объем выборки	Число участков разбиения
17	Метод Макларена — Марсальи	$k = 128$	5 000	26
18	Смешанный, формула (2.5)	$Y = 4\,001$	1 500	16
19	Универсальный, формула (2.9), $k = 2$	Y_k — любое	4 000	21
20	Мультипликативный, формула (2.4)	$Y_0 = 3\,091$, $M = 4\,096$	2 000	21

ЛАБОРАТОРНАЯ РАБОТА № 2. ПРОВЕРКА КАЧЕСТВА ГЕНЕРАТОРОВ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ

Используя результаты, полученные при выполнении лабораторной работы 1, проверьте качество последовательности псевдослучайных чисел с помощью критерия Пирсона, Колмогорова, а также критерия, указанного в табл. 10.2.

Таблица 10.2. Варианты заданий к лабораторной работе № 2

Номер варианта	Критерий
1	Проверка качества по косвенным признакам
2	Критерий числа серий, разделительный элемент $p = 0{,}25$
3	Тест длины серий нулей, разделительный элемент $p = 0{,}3$
4	Тест длины серий единиц, разделительный элемент $p = 0{,}4$
5	Покер-тест, $k = 2$
6	Критерий коллекционера

Окончание табл. 10.2

Номер варианта	Критерий
7	Покер-тест, $k = 8$
8	Критерий числа серий, разделительный элемент $p = 0{,}5$
9	Тест длины серий нулей, разделительный элемент $p = 0{,}5$
10	Тест длины серий единиц, разделительный элемент $p = 0{,}25$
11	Критерий коллекционера
12	Тест длины серий нулей, разделительный элемент $p = 0{,}4$
13	Тест числа серий, разделительный элемент $p = 0{,}45$
14	Тест длины серий нулей, разделительный элемент $p = 0{,}45$
15	Тест длины серий единиц, разделительный элемент $p = 0{,}45$
16	Тест числа серий, разделительный элемент $p = 0{,}6$
17	Покер-тест, $k = 10$
18	Проверка качества по косвенным признакам
19	Тест длины серий нулей, разделительный элемент $p = 0{,}65$
20	Тест длины серий единиц, разделительный элемент $p = 0{,}65$

ЛАБОРАТОРНАЯ РАБОТА № 3. ГЕНЕРИРОВАНИЕ СЛУЧАЙНЫХ ВЕЛИЧИН С ЗАДАННЫМ ЗАКОНОМ РАСПРЕДЕЛЕНИЯ

Составьте подпрограмму генерирования случайных величин в соответствии с вариантом задания, определяемым табл. 10.3. По полученной с помощью подпрограммы выборке постройте гистограмму частот и статистическую функцию распределения, оцените математическое ожидание и дисперсию случайной величины. Соответствие эмпирических данных теоретическому распределению проверьте с помощью критерия Пирсона или критерия Колмогорова. Объем выборки случайных величин не менее 1 000. Количество интервалов разбиения $k = 15$ или $k = 25$.

Таблица 10.3. Варианты заданий к лабораторной работе № 32

Номер варианта	Закон распределения	Способ построения
1	$F(x) = \begin{cases} 0{,}4(x-1)^3 + 0{,}4, & x \in [0; 0{,}5); \\ 0{,}3x + 0{,}2, & x \in [0{,}5; 1{,}5); \\ 0{,}4(x-1)^3 + 0{,}6, & x \in [1{,}5; 2] \end{cases}$	Метод отбора
2	$F(x) = \begin{cases} \sqrt{0{,}25 - (x-0{,}5)^2}, & x \in [0; 0{,}5); \\ 0{,}3125x + 0{,}34375, & x \in [0{,}5; 1{,}3); \\ 1{,}25x - 0{,}875, & x \in [1{,}3; 1{,}5] \end{cases}$	Метод обратных функций
3	$F(x) = \begin{cases} 0{,}25x^2, & x \in (0; 1); \\ 1{,}14x - 0{,}89, & x \in [1; 1{,}5); \\ 1 - 0{,}08(x-3)^2, & x \in [1{,}5; 3] \end{cases}$	Метод отбора
4	$F(x) = \begin{cases} x, & x \in [0; 0{,}5); \\ 0{,}5, & x \in [0{,}5; 1); \\ 2(x-1)^2 + 0{,}5, & x \in [1; 1{,}5] \end{cases}$	Метод обратных функций
5	$F(x) = \begin{cases} 0{,}3x, & x \in [0; 0{,}5); \\ 3x - 1{,}35, & x \in [0{,}5; 0{,}7); \\ 0{,}25x + 0{,}575, & x \in [0{,}7; 1{,}7] \end{cases}$	Метод отбора
6	$F(x) = \begin{cases} x^2, & x \in [0; 0{,}5); \\ 1{,}1x - 0{,}3, & x \in [0{,}5; 1); \\ 0{,}4x + 0{,}4, & x \in [1; 1{,}5] \end{cases}$	Метод обратных функций

Продолжение табл. 10.3

Номер варианта	Закон распределения	Способ построения
7	Треугольное: $f(x)=\begin{cases}0, & x\le a;\\ \dfrac{2(x-a)}{(b-a)(c-a)}, & a<x\le c;\\ \dfrac{2(b-x)}{(b-a)(b-c)}, & c<x\le b;\\ 0, & x>b.\end{cases}$ $a=1;\ b=5;\ c=2$	Метод отбора
8	$F(x)=\begin{cases}0{,}2\cdot 10^{x}-0{,}2, & x\in[0;0{,}3);\\ 1{,}5x-0{,}25, & x\in[0{,}3;0{,}7);\\ 0{,}25x+0{,}625, & x\in[0{,}7;1{,}5]\end{cases}$	Метод обратных функций
9	$F(x)=\begin{cases}0{,}15x, & x\in[0;1);\\ 0{,}35x-0{,}2, & x\in[1;2);\\ 0{,}875x-1{,}25, & x\in[2;2{,}4);\\ 0{,}15x-0{,}49 & x\in[2{,}4;3{,}4]\end{cases}$	Метод отбора
10	$F(x)=\begin{cases}0{,}8x^{2}, & x\in[0;0{,}5);\\ 0{,}7x-0{,}15, & x\in[0{,}5;1);\\ 1-e^{-0{,}8x}, & x\in[1;\infty)\end{cases}$	Метод обратных функций
11	Треугольное (см. задание 7) $a=0;\ b=10;\ c=5$	Метод отбора
12	$F(x)=\begin{cases}0{,}05x^{3}, & x\in[0;2);\\ 1-2e^{-0{,}602x}, & x\in[2;\infty)\end{cases}$	Метод обратных функций
13	$F(x)=\begin{cases}\sqrt{x}, & x\in[0;0{,}25);\\ 0{,}25x+0{,}4375, & x\in[0{,}25;2{,}25]\end{cases}$	Метод отбора

Окончание табл. 10.3

Номер варианта	Закон распределения	Способ построения
14	$F(x)=\begin{cases}\dfrac{2(e-1)}{e}x, & x\in[0;0{,}5);\\ 1-e^{-2x}, & x\in[0{,}5;\infty)\end{cases}$	Метод обратных функций
15	Треугольное (см. задание 7) $a=4;\ b=5;\ c=4{,}7$	Метод отбора
16	$F(x)=\begin{cases}1-e^{-2x}, & x\in[0;1);\\ \dfrac{x-3+2e^{2}}{2e^{2}}, & x\in[1;3]\end{cases}$	Метод обратных функций
17	$f(x)=\sqrt{R^{2}-(x-a)^{2}};$ $R=\sqrt{\pi/2};\ a=-2$	Метод отбора
18	$F(x)=\begin{cases}0{,}4(x-1)^{3}+0{,}4, & x\in[0;0{,}5);\\ 0{,}3x+0{,}2, & x\in[0{,}5;1{,}5);\\ 0{,}4(x-1)^{3}+0{,}6, & x\in[1{,}5;2]\end{cases}$	Метод обратных функций
19	$f(x)=\dfrac{\pi}{2}\sin(\pi x),\quad 0<x\le 1$	Метод отбора
20	Треугольное (см. задание 5) $a=-3;\ b=10;\ c=0$	Метод обратных функций

ЛАБОРАТОРНАЯ РАБОТА № 4. ГЕНЕРИРОВАНИЕ СЛУЧАЙНЫХ ВЕЛИЧИН С НОРМАЛЬНЫМ ЗАКОНОМ РАСПРЕДЕЛЕНИЯ

Составьте подпрограмму генерирования случайных величин с нормальным законом распределения методом, основанным на центральной предельной теореме, а также методом, определенным в соответствии с вариантом задания (табл. 10.4). Параметры закона распределения указаны в виде $N(\mu; \sigma^2)$. По полученной с помощью подпрограммы выборке постройте гистограмму частот и статистическую функцию распределения, оцените математическое ожидание и дисперсию случайной величины. Соответствие эмпирических данных теоретическому распределению проверьте с помощью критерия Пирсона или критерия Колмогорова. Объем выборки случайных величин не менее 1 000. Количество интервалов разбиения $k = 15$ или $k = 25$.

Таблица 10.4. Варианты заданий к лабораторной работе № 4

Вариант	Закон распределения	Способ построения
1	Нормальный, $N(3; 1)$	Метод аппроксимации
2	Нормальный, $N(0; 1)$	Метод Бокса и Малера
3	Нормальный, $N(3; 1)$	Процедура Марсальи и Брея
4	Нормальный, $N(-2; 0{,}81)$	Метод аппроксимации
5	Нормальный, $N(4.3; 0{,}5)$	Метод Бокса и Малера
6	Нормальный, $N(2; 0{,}9)$	Процедура Марсальи и Брея
7	Нормальный, $N(3.5; 0{,}9)$	Метод аппроксимации
8	Нормальный, $N(2; 0{,}2)$	Метод Бокса и Малера
9	Нормальный, $N(-1{,}5; 1{,}7)$	Процедура Марсальи и Брея
10	Нормальный, $N(0; 0{,}1)$	Метод аппроксимации
11	Нормальный, $N(2; 1)$	Метод Бокса и Малера
12	Нормальный, $N(-2; 1)$	Процедура Марсальи и Брея
13	Нормальный, $N(2; 1)$	Метод аппроксимации

Окончание табл. 10.4

Вариант	Закон распределения	Способ построения
14	Нормальный, $N(1; 0{,}7)$	Метод Бокса и Малера
15	Нормальный, $N(5; 1)$	Процедура Марсальи и Брея
16	Нормальный, $N(4{,}7; 0{,}6)$	Метод аппроксимации
17	Нормальный, $N(1{,}5; 0{,}1)$	Метод Бокса и Малера
18	Нормальный, $N(-4; 0{,}1)$	Процедура Марсальи и Брея
19	Нормальный, $N(3; 0{,}1)$	Метод аппроксимации
20	Нормальный, $N(2{,}75; 2{,}8)$	Метод Бокса и Малера

ЛАБОРАТОРНАЯ РАБОТА № 5. ГЕНЕРИРОВАНИЕ СЛУЧАЙНЫХ ВЕЛИЧИН С ЧАСТО ИСПОЛЬЗУЕМЫМИ ЗАКОНАМИ РАСПРЕДЕЛЕНИЯ

Составьте подпрограммы генерирования случайных величин, подчиненных бета-распределению, гамма-распределению, логарифмически-нормальному распределению и распределению Вейбулла. По полученной с помощью подпрограммы выборке постройте гистограмму частот и статистическую функцию распределения, оцените математическое ожидание и дисперсию случайной величины. Соответствие эмпирических данных теоретическому распределению проверьте с помощью критерия Пирсона или критерия Колмогорова. Объем выборки случайных величин не менее 1 000. Количество интервалов разбиения $k = 15$ или $k = 25$.

ЛАБОРАТОРНАЯ РАБОТА № 6. МОДЕЛИРОВАНИЕ МЕТОДОМ МОНТЕ-КАРЛО

Составьте программу решения задачи, определенной в соответствии с вариантом задания, с помощью машинного моделирования (метод Монте-Карло). Постройте доверительный интервал для

полученных оценок, накрывающий точное значение оцениваемых вероятностей с надежностью $\beta = 0{,}95$. Правильность результатов проверьте аналитическим решением задачи.

Варианты заданий

1. Истребитель, вооруженный двумя ракетами, посылается на перехват воздушной цели. Вероятность вывода истребителя в такое положение, из которого возможна атака цели, равна p_1. Если истребитель выведен в такое положение, он выпускает по цели обе ракеты, каждая из которых независимо от другой выводится в окрестность цели с вероятностью p_2. Если ракета выведена в окрестность цели, она поражает ее с вероятностью p_3. Оцените вероятность того, что цель будет поражена.

2. Производится стрельба двумя снарядами по k бакам с горючим ($k > 2$), расположенным рядом друг с другом в одну линию. Каждый снаряд независимо от других попадает в первый бак с вероятностью p_1, во второй — с вероятностью p_2 и т.д. Для воспламенения баков требуется два попадания в один и тот же бак или два попадания в соседние баки. Оцените вероятность воспламенения баков.

3. Производится стрельба по цели тремя снарядами. Снаряды попадают в цель независимо друг от друга. Для каждого снаряда вероятность попадания в цель равна p_0. Если в цель попал один снаряд, он поражает цель (выводит ее из строя) с вероятностью p_1; если два снаряда — с вероятностью p_2; если три снаряда — с вероятностью p_3. Оценить полную вероятность поражения цели.

4. Происходит воздушный бой между двумя самолетами: истребителем и бомбардировщиком. Стрельбу начинает истребитель: он дает по бомбардировщику один выстрел и сбивает его с вероятностью p_1. Если бомбардировщик этим выстрелом не сбит, он стреляет по истребителю и сбивает его с вероятностью p_2. Если истребитель этим выстрелом не сбит, он еще раз стреляет по бомбардировщику и сбивает его с вероятностью p_3. Оцените вероятности следующих исходов боя:

A — сбит бомбардировщик;

B — сбит истребитель;

C — сбит хотя бы один из самолетов.

5. N стрелков независимо один от другого ведут стрельбу каждый по своей мишени. Каждый из них имеет боезапас k патронов. Вероятность попадания в мишень при одном выстреле для i-го стрелка равна p_i ($i = 1, 2, ..., N$). При первом же попадании в свою мишень стрелок прекращает стрельбу. Оцените вероятности следующих событий:

A — у всех стрелков вместе останется неизрасходованным хотя бы один патрон;

B — ни у кого из стрелков не будет израсходован весь боезапас;

C — какой-либо один из стрелков израсходует весь боезапас, а все остальные — не весь.

6. N стрелков стреляют поочередно по одной мишени. Стрельба ведется до первого попадания. Вероятность попасть в мишень для каждого стрелка равна p_i ($i = 1, 2, ..., N$). Выигравшим считается тот стрелок, который первым попадет в мишень. У каждого стрелка в запасе имеется n патронов. Оцените вероятность того, что выиграет i-й стрелок.

7. Происходит воздушный бой между бомбардировщиком и двумя атакующими его истребителями. Стрельбу начинает бомбардировщик; он дает по каждому истребителю один выстрел и сбивает его с вероятностью p_1. Если данный истребитель не сбит, то он независимо от судьбы другого стреляет по бомбардировщику и сбивает его с вероятностью p_2.

Оцените вероятности следующих исходов боя:

A — сбит бомбардировщик;

B — сбиты оба истребителя;

C — сбит хотя бы один истребитель;

D — сбит хотя бы один самолет;

E — сбит ровно один истребитель;

F — сбит ровно один самолет.

8. Человек, принадлежащий к определенной группе населения, с вероятностью p_1 оказывается брюнетом, с вероятностью p_2 — шатеном, с вероятностью p_3 — блондином и с вероятностью p_4 — рыжим. Выбирается наугад группа из шести человек. Оцените вероятности следующих событий:

A — в составе группы не меньше четырех блондинов;

B — в составе группы хотя бы один рыжий;

C — в составе группы равное число блондинов и шатенов.

9. Прибор состоит из трех узлов. При включении прибора с вероятностью p_1 появляется неисправность в первом узле, с вероятностью p_2 — во втором узле, с вероятностью p_3 — в третьем узле. Неисправности в узлах возникают независимо друг от друга. Каждый из трех узлов, безусловно, необходим для работы прибора. Для того чтобы узел отказал, необходимо, чтобы в нем было не менее двух неисправностей. Оцените вероятность того, что прибор благополучно выдержит n включений.

10. Группа самолетов в составе: один ведущий и два ведомых, направляется на бомбометание по объекту. Каждый из них несет по одной бомбе. Ведущий самолет имеет прицел, ведомые — не имеют и производят бомбометание по сигналу ведущего. По пути к объекту группа проходит зону противовоздушной обороны, в которой каждый из самолетов независимо от других сбивается с вероятностью p. Если к цели подойдет ведущий самолет с обоими ведомыми, они поразят объект с вероятностью $P_{1,2}$. Ведущий самолет, сопровождаемый одним ведомым, поразит объект с вероятностью $P_{1,1}$. Один ведущий самолет без ведомых поразит объект с вероятностью $P_{1,0}$. Если ведущий самолет сбит, то каждый из ведомых, если он сохранился, выходит к объекту и поражает его с вероятностью $P_{0,1}$. Оцените полную вероятность поражения объекта с учетом противодействия.

11. Завод изготовляет изделия, каждое из которых с вероятностью p имеет дефект. В цехе имеются три контролера; изделие осматривается только одним контролером, с одинаковой вероятностью первым, вторым или третьим. Вероятность обнаружения дефекта (если он имеется) для i-го контролера равна p_i (i = l, 2, 3). Если изделие не было забраковано в цехе, то оно попадет в отдел технического контроля (ОТК) завода, где дефект, если он имеется, обнаруживается с вероятностью p_0.

Оцените вероятности следующих событий:

A — изделие будет забраковано;

B — изделие будет забраковано в цехе;

C — изделие будет забраковано в ОТК завода.

12. Радиолокационная станция ведет наблюдение за объектом, который может применять или не применять помехи. Если объект не применяет помех, то за один цикл обзора станция обнаруживает его с вероятностью p_0; если применяет — с вероятностью $p_1 < p_0$. Вероятность того, что во время цикла будут применены помехи, равна p и не зависит от того, как и когда применялись помехи в остальных циклах. Оцените вероятность того, что объект будет обнаружен хотя бы один раз за n циклов обзора.

13. Группа, состоящая из трех самолетов-разведчиков, высылается в район противника с целью уточнить координаты объекта, который предполагается подвергнуть обстрелу ракетами. Для поражения объекта выделено n ракет. При уточненных координатах объекта вероятность его поражения одной ракетой равна p_1, при неуточненных — p_2. Каждый разведчик перед выходом в район объекта может быть сбит противовоздушными средствами противника; вероятность этого p_3. Если разведчик не сбит, он сообщает координаты объекта по радио. Радиоаппаратура разведчика имеет надежность p_4. Для уточнения координат достаточно приема сообщения от одного разведчика. Оцените вероятность поражения объекта с учетом деятельности разведки.

14. Из N стрелков можно выделить четыре группы: a_1 отличных стрелков, a_2 хороших, a_3 посредственных и a_4 плохих. Вероятность попадания в мишень при одном выстреле для стрелка i-и группы равна p_i (i = 1, 2, 3, 4). Вызываются наугад два стрелка, которые стреляют по одной и той же мишени. Оцените вероятность хотя бы одного попадания в мишень.

15. Мишень состоит из яблока и двух колец. При одном выстреле вероятность попадания в яблоко равна p_0, в первое кольцо — p_1, во второе — p_2; вероятность непопадания в мишень p_3. По мишени произведено пять выстрелов. Оцените вероятность того, что они дадут два попадания в яблоко и одно — во второе кольцо.

16. Прибор состоит из 10 узлов. Надежность (вероятность безотказной работы в течение времени t) для каждого узла равна p. Узлы выходят из строя независимо один от другого. Оцените вероятность того, что за время t: а) откажет хотя бы один узел; б) откажет ровно один узел; в) откажут ровно два узла; г) откажет не менее двух узлов.

17. Производится четыре независимых выстрела по цели. Вероятности попадания при разных выстрелах различны и равны: p_1, p_2, p_3, p_4. Оцените вероятности $P_{0,4}$; $P_{1,4}$; $P_{2,4}$; $P_{3,4}$; $P_{4,4}$ ни одного, одного, двух, трех, четырех попаданий; вероятность $R_{1,4}$ хотя бы одного попадания; вероятность $R_{2,4}$ не менее двух попаданий.

18. Завод изготовляет изделия, каждое из которых должно подвергаться четырем видам испытаний. Первое испытание изделие проходит благополучно с вероятностью p_1; второе — с вероятностью p_2; третье — с вероятностью p_3 и четвертое — с вероятностью p_4. Оцените вероятность того, что изделие пройдет благополучно:

A — все четыре испытания;

B — ровно два испытания (из четырех);

C — не менее двух испытаний (из четырех).

19. Завод изготовляет изделия, каждое из которых с вероятностью r (независимо от других) оказывается дефектным. При осмотре дефект, если он имеется, обнаруживается с вероятностью p. Для контроля из продукции завода выбирается n изделий. Оцените вероятность следующих событий:

A — ни в одном из изделий не будет обнаружено дефекта;

B — среди n изделий ровно в двух будет обнаружен дефект;

C — среди n изделий не менее чем в двух будет обнаружен дефект.

20. Самолет обстреливается n независимыми выстрелами; каждый из выстрелов с вероятностью p_1 попадает в зону, где он поражает самолет немедленно; с вероятностью p_2 попадает в топливный бак и с вероятностью p_3 не попадает в самолет. Снаряд, попавший в топливный бак, оставляет в нем пробоину, через которую вытекает k литров горючего в час. Потеряв M литров горючего, самолет становится небоеспособным. Оцените вероятность того, что через час после обстрела самолет не будет боеспособен.

ЛАБОРАТОРНАЯ РАБОТА № 7. МОДЕЛИРОВАНИЕ СЛУЧАЙНЫХ БЛУЖДАНИЙ

Каждое задание предполагает разработку программной имитационной модели случайного блуждания, с помощью которой могут быть получены необходимые результаты. В результате проведения определенного количества экспериментов постройте статистическое распределение исследуемого параметра (гистограмму и эмпирическую функцию распределения) и определите целесообразность аппроксимации полученного распределения одним из известных законов (нормальным, экспоненциальным, логарифмически-нормальным и др.).

Варианты заданий

1. *Одномерное случайное блуждание*. Составьте модель определения расстояния, на которое удалится пешеход за $M = 20$ шагов.

2. *Двумерное случайное блуждание*. Составьте модель определения расстояния, на которое удалится пешеход за $M = 10$ шагов.

3. *Простое случайное блуждание с поглощающими экранами*. Составьте машинную модель для определения времени блуждания.

4. «*Пчелы*» *на квадратной решетке*. «Рой» из N «пчел» изначально расположен в единичном круге с центром в начале координат. На каждом шаге по времени каждая «пчела» движется случайным образом равновероятно в одном из четырех направлений: на север, юг, восток и запад. Определите расстояние, на которое удаляется отдельная «пчела» за $M = 8$ шагов. В течение каждого временно́го интервала каждая «пчела» делает шаг единичной длины. Усреднение выполняется по N «пчелам».

5. *Блуждание на треугольной решетке*. Составьте имитационную модель случайного блуждания «пчелы» на треугольной решетке. На каждом шаге по времени «пчела» движется равновероятно в одном из шести возможных направлений. На какое расстояние удаляется «пчела» за $M = 8$ шагов?

6. *Модель падения дождевой капли*. При воздействии случайных порывов легкого ветра падение дождевой капли можно моделировать случайным блужданием на квадратной решетке (см. рис. 6.9). Движение начинается с узла, расположенного на расстоянии h над горизонтальной линией (поверхностью земли). Вероятность $p_\downarrow$ шага «вниз» больше вероятности $p_\uparrow$ шага «вверх». Вероятности скачков целесообразно выбирать равными $p_\downarrow = 0{,}5$; $p_\uparrow = 0{,}1$; $p_\leftarrow = p_\rightarrow = 0{,}2$.

Определите время τ, за которое капля достигает горизонтальной прямой, и функциональную зависимость τ от h (4…6 значений).

7. *Ограниченные случайные блуждания*. Задача простого случайного блуждания с поглощающими экранами может быть рассмотрена в форме следующей модификации. Пусть одномерная решетка имеет поглощающие узлы (ловушки) в точках $x = 0$ и $x = a$ ($a > 0$). Частица начинает движение из точки x_0 ($0 < x_0 < a$) и с равной вероятностью переходит в ближайшие соседние узлы. Определите время τ прохода частицы до ее поглощения.

8. *Блуждание на сотах*. Составьте имитационную модель случайного блуждания на сотах. На каждом шаге по времени пчела движется равновероятно в одном из трех направлений. На какое расстояние удаляется «пчела» за $M = 8$ шагов?

9. *Случайные блуждания на трехмерной решетке*. Оцените расстояние, на которое удаляется частица, равновероятно блуждающая по трехмерной решетке. Число шагов блуждания $M = 10$. Параллельно исследуемому процессу определите удаление от начального состояния отдельно по всем трем составляющим координатам.

10. *Персистентное случайное блуждание*. В персистентном случайном блуждании вероятность перехода, или «скачка», зависит от последнего перехода. Рассмотрите одномерное случайное блуждание частицы, в котором шаги совершаются только в ближайшие соседние узлы. Предположим, что сделано $k - 1$ шагов. Далее k-й шаг делается в том же направлении с вероятностью α, а шаг в противоположном направлении делается с вероятностью $1 - \alpha$. Опре-

делите удаление частицы от исходного положения за $M = 8$ шагов при $\alpha = 0{,}2$ и $\alpha = 0{,}4$.

11. *Случайные блуждания с переменным шагом*. Рассмотрите одномерное случайное блуждание со всеми допустимыми длинами прыжков. Вероятность того, что длина шага равна j, имеет вид $P(j) = \exp(-j)$. Определите удаление от начального положения после 10 шагов.

12. *Решетка с переменным шагом*. Рассмотрите случайное блуждание на решетке, как в задании 11, с вероятностью распределения длины шагов $P(j) = a/j^2$. Здесь $a = 6/\pi^2$. Определите удаление от начального положения после восьми шагов.

13. *Непрерывное случайное блуждание*. Одна из первых непрерывных моделей случайного блуждания предложена Рейли в 1919 г. В модели Рейли длина каждого шага a является случайной величиной, распределенной с плотностью вероятности $p(a)$, и случайным направлением каждого шага.

Определите удаление пешехода от исходной точки за пять шагов, если плотность вероятности $p(a)$ является равновероятной в пределах от 0,5 до 1,5 и направление движения выбирается равновероятно с точностью до 1°.

ЛАБОРАТОРНАЯ РАБОТА № 8. МОДЕЛИРОВАНИЕ СИСТЕМ МАССОВОГО ОБСЛУЖИВАНИЯ

Составьте программу моделирования СМО, определенной вариантом задания. Оцените основные характеристики эффективности функционирования СМО. Результаты сравните с точными значениями, полученными при аналитическом расчете по приведенным в гл. 7 формулам.

Варианты заданий

1. Одноканальная СМО с отказами представляет собой одну телефонную линию. Заявка (вызов), пришедшая в момент, когда линия занята, получает отказ. Все потоки событий простейшие. Интенсивность потока $\lambda = 0{,}95$ вызова в минуту. Средняя продолжительность разговора $t = 1$ мин. Определите вероятностные характеристики СМО в установившемся режиме работы.

2. В одноканальную СМО с отказами поступает простейший поток заявок с интенсивностью $\lambda = 0{,}5$ заявки в минуту. Время обслуживания заявки имеет показательное распределение с $t = 1{,}5$ мин. Определите вероятностные характеристики СМО в установившемся режиме работы.

3. В вычислительном центре работает пять персональных компьютеров. Простейший поток задач, поступающих на вычислительный центр, имеет интенсивность $\lambda = 10$ задач в час. Среднее время решения задачи равно 12 мин. Заявка получает отказ, если все компьютеры заняты. Найдите вероятностные характеристики системы обслуживания (вычислительного центра).

4. В аудиторскую фирму поступает простейший поток заявок на обслуживание с интенсивностью $\lambda = 1{,}5$ заявки в день. Время обслуживания распределено по показательному закону и равно в среднем трем дням. Аудиторская фирма располагает пятью независимыми бухгалтерами, выполняющими аудиторские проверки (обслуживание заявок). Очередь заявок не ограничена. Определите вероятностные характеристики аудиторской фирмы как СМО, работающей в стационарном режиме.

5. На пункт техосмотра поступает простейший поток заявок (автомобилей) интенсивностью $\lambda = 4$ машины в час. Время осмотра распределено по показательному закону и равно в среднем 17 мин, в очереди может находиться не более пяти автомобилей. Определите вероятностные характеристики пункта техосмотра в установившемся режиме.

6. В бухгалтерии предприятия имеются два кассира, каждый из которых может обслужить в среднем 30 сотрудников в час. Поток сотрудников, получающих заработную плату, — простейший, с интенсивностью, равной 40 сотрудников в час. Очередь в кассе не ограничена. Время обслуживания подчинено экспоненциальному закону распределения. Вычислите вероятностные характеристики СМО в стационарном режиме и определите целесообразность приема третьего кассира на предприятие, работающего с такой же производительностью, как и первые два.

7. В инструментальном отделении сборочного цеха работают три кладовщика. В среднем за 1 мин за инструментом приходят 0,8 рабочих ($\lambda = 0{,}8$). Обслуживание одного рабочего занимает у кладовщика $t = 1{,}0$ мин. Очередь не имеет ограничения. Известно, что поток рабочих за инструментом — пуассоновский, а время обслуживания подчинено экспоненциальному закону распределения. Стоимость 1 мин работы рабочего равна 30 д.е., а кладовщика — 15 д.е. Найдите средние потери цеха при данной организации об-

служивания в инструментальном отделении (стоимость простоя) при стационарном режиме работы.

8. Билетная касса работает без перерыва. Билеты продает один кассир. Среднее время обслуживания — 2 мин на каждого человека. Среднее число пассажиров, желающих приобрести билеты в кассе в течение одного часа, $\lambda = 20$ пассажиров в час. Все потоки в системе простейшие. Определите характеристики СМО в условиях стационарного режима работы кассы.

9. Пост диагностики автомобилей представляет собой одноканальную СМО с отказами. Заявка на диагностику, поступившая в момент, когда пост занят, получает отказ. Интенсивность потока заявок на диагностику $\lambda = 0{,}5$ автомобиля в час. Средняя продолжительность диагностики $t = 1{,}2$ ч. Все потоки событий в системе простейшие. Определите в установившемся режиме характеристики системы.

10. Автозаправочная станция представляет собой СМО с одним каналом обслуживания и одной колонкой. Площадка при АЗС допускает пребывание в очереди на заправку не более трех автомобилей одновременно. Если в очереди уже находится три автомобиля, очередной автомобиль, прибывший к станции, в очередь не становится, а проезжает мимо. Поток автомобилей, прибывающих для заправки, имеет интенсивность $\lambda = 0{,}7$ автомобиля в минуту. Процесс заправки продолжается в среднем 1,25 мин. Все потоки простейшие. Определите вероятностные характеристики СМО в стационарном режиме.

11. На железнодорожную сортировочную горку прибывают составы с интенсивностью $\lambda = 2$ состава в час. Среднее время, в течение которого горка обслуживает состав, равно 0,4 ч. Составы, прибывающие в момент, когда горка занята, становятся в очередь и ожидают в парке прибытия, где имеются три запасных пути, на каждом из которых может ожидать один состав. Состав, прибывший в момент, когда все три запасных пути в парке прибытия заняты, становится в очередь на внешний путь. Все потоки событий простейшие. Определите характеристики СМО в условиях стационарного режима работы.

12. Рассматривается работа АЗС, на которой имеются три заправочные колонки. Заправка одной машины длится в среднем 3 мин. В среднем на АЗС каждую минуту прибывает машина, нуждающаяся в заправке бензином. Число мест в очереди не ограничено. Все машины, вставшие в очередь на заправку, дожидаются своей очереди. Все потоки в системе простейшие. Определите вероятностные характеристики работы АЗС в стационарном режиме.

13. На станцию технического обслуживания автомобилей каждые два часа подъезжает в среднем одна машина. Станция имеет шесть постов обслуживания. Очередь автомобилей, ожидающих обслуживания, не ограничена. Среднее время обслуживания одной машины — 2 ч. Все потоки в системе простейшие. Определите характеристики станции технического обслуживания автомобилей.

14. Имеется двухканальная простейшая СМО с отказами. На ее вход поступает поток заявок с интенсивностью $\lambda = 3$ заявки в час. Среднее время обслуживания одной заявки $t = 0{,}5$ ч. Каждая обслуженная заявка приносит доход 5 д.е. Содержание канала обходится в 3 д.е./ч. Решите, выгодно ли в экономическом отношении увеличить число каналов СМО до трех.

15. В магазине работает один продавец, который может обслужить в среднем 30 покупателей в час. Поток покупателей простейший с интенсивностью, равной 60 покупателей в час. Все покупатели «нетерпеливые» и уходят, если в очереди стоит пять человек (помимо обслуживаемых). Все потоки событий простейшие. Определите характеристики магазина для стационарного режима работы.

16. На вход телефонной станции, имеющей девять каналов обслуживания, поступает в среднем 120 заявок в час. Заявка получает отказ, если все каналы заняты. Среднее время обслуживания в одном канале равно 4 мин. Все потоки в системе простейшие. Определите характеристики телефонной станции.

17. Рассматривается работа АЗС, на которой имеется пять заправочных колонок. Заправка одной машины длится в среднем 4 мин. В среднем на АЗС каждую минуту прибывает машина, нуждающаяся в заправке бензином. Число мест в очереди не ограничено. Все машины, вставшие в очередь, дожидаются своей очереди. Все потоки событий простейшие. Определите вероятностные характеристики АЗС для стационарного режима.

18. Подсчитайте вероятностные характеристики для простейшей одноканальной СМО с тремя местами в очереди при условиях $\lambda = 4$ заявки в час, $t = 0{,}5$ ч. Выясните, как эти характеристики изменятся, если увеличить число мест в очереди до четырех.

19. Одноканальная СМО — ЭВМ, на которую поступают заявки на расчеты. Поток заявок простейший со средним интервалом времени между заявками $t = 10$ мин. Время обслуживания распределено по экспоненциальному закону с математическим ожиданием $t_{обс} = 8$ мин. Определите среднее число заявок в СМО, среднее число заявок в очереди, среднее время пребывания заявки в системе и в очереди.

20. Система массового обслуживания — билетная касса с тремя окошками (с тремя кассирами) и неограниченной очередью. Пассажиров, желающих купить билет, приходит в среднем пять человек за 20 мин. Поток пассажиров можно считать простейшим. Кассир в среднем обслуживает трех пассажиров за 10 мин. Время обслуживания подчинено показательному закону распределения. Определите вероятностные характеристики СМО в стационарном режиме.

ЛАБОРАТОРНАЯ РАБОТА № 9. ТАКТИЧЕСКОЕ ПЛАНИРОВАНИЕ ЭКСПЕРИМЕНТА

Каждое задание предполагает разработку программной имитационной модели случайного блуждания, с помощью которой могут быть получены необходимые результаты по двум частям лабораторной работы. В первой части работы выполняется планирование и получение результатов эксперимента по оценке среднего значения указанного параметра. Во второй части выполняется планирование и получение результатов эксперимента для оценки с заданной точностью выборочной дисперсии указанного параметра. В процессе проведения пробного эксперимента при планировании строится статистическое распределение исследуемого параметра и определяется целесообразность аппроксимации этого распределения нормальным законом. В зависимости от этого выбирается подход для определения объема эксперимента. Варианты заданий берутся из лабораторной работы № 7.

ЛАБОРАТОРНАЯ РАБОТА № 10. МОДЕЛИРОВАНИЕ СИСТЕМ МАССОВОГО ОБСЛУЖИВАНИЯ С ПОМОЩЬЮ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ МОДЕЛИРОВАНИЯ PILGRIM

Выполните моделирование СМО, определенной вариантом задания из лабораторной работы № 8, с помощью инструментальных средств моделирования Pilgrim. Результаты моделирования сравните с результатами, полученными в лабораторной работе № 8.

Список литературы

1. *Бережная Е. В.* Математические методы моделирования экономических систем : учеб. пособие / Е. В. Бережная, В. И. Бережной. — М. : Финансы и статистика, 2002.
2. *Варфоломеев В. И.* Алгоритмическое моделирование элементов экономических систем: Практикум : учеб. пособие / В. И. Варфоломеев. — М. : Финансы и статистика, 2000.
3. *Емельянов А. А.* Имитационное моделирование экономических процессов : учеб. пособие / А. А. Емельянов, Е. А. Власова, Р. В. Дума. — М. : Финансы и статистика, 2002.
4. *Золотарев В. В.* Компьютерное моделирование : учеб. пособие / В. В. Золотарев, Г. В. Овечкин, П. В. Овечкин. — Рязань : РГРТУ, 2008.
5. *Кельтон В.* Имитационное моделирование. Классика CS / В. Кельтон, А. Лоу. — 3-е изд. — СПб. : Питер ; Киев : BHV, 2004.
6. *Кнут Д.* Искусство программирования. Том 2. Получисленные алгоритмы / Д. Кнут. — 3-е изд. — М. : Вильямс, 2007.
7. *Лабскер Л. Г.* Теория массового обслуживания в экономической сфере : учеб. пособие / Л. Г. Лабскер, Л. О. Бабешко. — М. : Банки и биржи ; ЮНИТИ, 1998.
8. *Нейлор Т.* Машинные имитационные эксперименты с моделями экономических систем / Т. Нейлор. — М. : Мир, 1975.
9. Основы компьютерного моделирования систем / Д. Е. Артемкин, В. В. Баринов, Г. В. Овечкин, И. М. Степнов ; под ред. А. Н. Пылькина. — М. : Бином. Лаборатория знаний, 2004.
10. *Пузикова Л. А.* Основы имитационного моделирования на ЭВМ. Методические указания к лабораторным работам / Л. А. Пузикова. — Рязань : РГРТУ, 1991.
11. *Рыжиков Ю. И.* Имитационное моделирование. Теория и технология / Ю. И. Рыжиков. — СПб. : КОРОНА принт ; М. : Альтекс-А, 2004.
12. *Советов Б. Я.* Моделирование систем : учебник / Б. Я. Советов, С. А. Яковлев. — М. : Высш. шк., 2001.
13. *Шеннон Р.* Имитационное моделирование систем: наука и искусство / Р. Шеннон. — М. : Мир, 1978.

Оглавление

Учебное издание

Овечкин Геннадий Владимирович,
Овечкин Павел Владимирович

Компьютерное моделирование

Учебник

Редактор *Л. В. Толочкова*
Компьютерная верстка: *Л. М. Беляева*
Корректор *Л. В. Гаврилина*

Изд. № 101115958. Подписано в печать 00.00.2014. Формат 60 × 90/16.
Гарнитура «Балтика». Бумага офсетная. Печать офсетная. Усл. печ. л. 14,0.
Тираж 0000 экз. Заказ №

Издательский центр «Академия». www.academia-moscow.ru
125252, Москва, ул. Зорге, д. 15, корп. 1, пом. 26б.
Адрес для корреспонденции: 129085, г. Москва, пр-т Мира, д. 101в, стр. 1, а/я 48.
Тел. 8(495)648-05-07, факс 8(495)616-00-29.
Санитарно-эпидемиологическое заключение № 77.99.60.953.Д.007831.07.09 от 06.07.2009.

ОАО «Тверской полиграфический комбинат». 170024, г. Тверь, пр-т Ленина, 5.
Телефон: (4822) 44-52-03, 44-50-34. Телефон/факс (4822) 44-42-15
Home page — www.tverpk.ru