MATLAB 5.2. ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ В СРЕДЕ WINDOWS: ПРАКТИЧЕСКОЕ ПОСОБИЕ.

Гультяев А.К.

В книге рассматриваются основы построения имитационных моделей и их применения в задачах принятия решений.

Основное внимание уделено инструментальнному средству визуального моделирования SIMULINK, входящему в состав популярного математического пакета MATLAB.

Изложение сопровождается многочисленными примерами, поясняющими технологию использования SIMULINK.

Для студентов и слушателей высших учебных заведений, обучающихся по техническим специальностям, слушателей курсов повышения квалификации по компьютерной подготовке, широкого круга пользователей персональных компьютеров.

Содержание

Предисловие

Имитационное моделирование — наиболее мощный и универсальный метод исследования и оценки эффективности систем, поведение которых зависит от воздействия случайных факторов. К таким системам можно отнести и летательный аппарат, и популяцию животных, и предприятие, работающее в условиях слабо регулируемых рыночных отношений.

В основе имитационного моделирования лежит статистический эксперимент (метод Монте-Карло), реализация которого практически невозможна без применения средств вычислительной техники. Поэтому любая имитационная модель представляет собой в конечном счете более или менее сложный программный продукт.

Конечно, как и любая другая программа, имитационная модель может быть разработана на любом универсальном языке программирования, даже на языке Ассемблера. Однако на пути разработчика в этом случае возникают следующие проблемы:

• требуется знание не только той предметной области, к которой относится исследуемая система, но и языка программирования, причем на достаточно высоком уровне.

• на разработку специфических процедур обеспечения статистического эксперимента (генерация случайных воздействий, планирование эксперимента, обработка результатов) может уйти времени и сил не меньше, чем на разработку собственно модели системы.

И наконец, еще одна, пожалуй, важнейшая проблема. Во многих практических задачах интерес представляет не только (и не столько) количественная оценка эффективности системы, сколько ее поведение в той или иной ситуации. Для такого наблюдения исследователь должен располагать соответствующими «смотровыми окнами», которые можно было бы при необходимости закрыть, перенести на другое место, изменить масштаб и форму представления наблюдаемых характеристик и т. д., причем не дожидаясь окончания текущего модельного эксперимента.

Реализация таких возможностей на универсальном языке программирования — дело очень не простое.

Вместе с тем в настоящее время на российском рынке компьютерных технологий есть продукт, позволяющий весьма эффективно решать указанные проблемы — пакет MATLAB (версии 5.*), содержащий в своем составе инструмент визуального моделирования SIMULINK.

SIMULINK — это инструмент для «ленивых» в хорошем смысле этого слова, т. е. для людей, умеющих ценить свое время и сравнивать ожидаемый эффект с затратами сил на его достижение.
Имеющиеся издания, посвященные описанию MATLAB, ориентированы на его применение для решения расчетных задач и аналитического моделирования. Знание этих особенностей, конечно, может во многих случаях оказаться полезным и даже необходимым. Однако основная задача предлагаемой книги — показать возможности и особенности использования SIMULINK как составной части пакета MATLAB в рамках технологии имитационного моделирования.

В соответствии с этим книга разделена на две относительно самостоятельные части: «Моделирование как наука» и «Моделирование как искусство». В первой из них описаны этапы процесса имитационного моделирования и те математические методы, которые положены в основу их реализации. Читатель, знакомый с этими вопросами, может материал первой части пропустить и обращаться к нему только в тех случаях, когда возникнет необходимость уточнить тот или иной термин, относящийся к теории имитационного моделирования. В связи с этим представляется уместным следующее замечание. Теория имитационного моделирования относительно молода, и в различных изданиях можно встретить несколько отличающееся толкование отдельных понятий и терминов. Те источники, на которые опирался автор, приведены в конце книги (в списке литературы к первой части). В них читатель может найти и более подробное изложение математических основ имитационного моделирования.

Вторая часть книги непосредственно посвящена технологии применения средств пакета MATLAB (в первую очередь — SIMULINK) для имитационного моделирования систем различных типов.

В настоящее время, когда круг пользователей персональных компьютеров постоянно расширяется, весьма сложно выбрать уровень изложения материала, который удовлетворил бы всех потенциальных читателей: одним достаточно получить представление о назначении основных инструментов и команд, другим требуется пояснить выполнение всех операций «от и до». Поэтому при изложении материала использован традиционный способ обучения «от простого к сложному». Представляется, что при таком подходе читателю проще самому выбрать тот уровень, который соответствует его подготовке. Другой принцип обучения — «делай, как я» — нашел свое отражение в многочисленных примерах, поясняющих использование различных инструментов SIMULINK. При этом построение каждой из рассматриваемых моделей происходит «на глазах у читателя», с привлечением все более сложных средств и методов.

Значительная часть приведенных в книге примеров относится к области вычислительной техники, однако описанные в ней приемы моделирования могут быть без больших затруднений перенесены в любую другую область деятельности.

ЧАСТЬ I
МОДЕЛИРОВАНИЕ КАК НАУКА

ГЛАВА 1
РОЛЬ МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ В ПРОЦЕССЕ ПРИНЯТИЯ РЕШЕНИЙ

— Всю жизнь ты одни глупости говоришь. И дурацкие советы даешь. Это меня не удивляет. А вот почему твои глупости всегда правильными бывают, этого я понять не могу.

Э. Успенский

1.1. ОБЩАЯ СХЕМА ПРОЦЕССА ПРИНЯТИЯ РЕШЕНИЙ

В течение всей своей жизни, от первых шагов и до последнего вздоха, человек вынужден принимать те или иные решения: какой подарок попросить у Деда Мороза, куда пойти учиться, как лучше потратить лишние (или последние) деньги и т. д.

Если определенная ситуация, требующая принятия решения, повторяется достаточно часто, то решение приходит само собой, автоматически. Если же ситуация недостаточно знакома или человек не располагает всей необходимой информацией, то принятие решения существенно усложняется. В таких случаях он вынужден, как правило, сравнивать между собой несколько возможных вариантов и выбирать тот, который кажется ему наиболее предпочтительным (или наименее опасным).

Например, представим себе такую ситуацию: некий добросовестный служащий опаздывает на работу; к остановке, которая находится на противоположной сторо-

не улицы, подходит автобус, а светофор сияет красным светом. У бедняги есть два варианта действий — пересечь улицу с риском для жизни на красный свет или поехать на следующем автобусе. Для принятия решения он должен знать, когда появится следующий автобус, а также соотнести потери от опоздания и возможного неудачного перехода улицы.

В приведенном примере служащий сам принимает решение, и сам же будет претворять его в жизнь. Значительно более важные последствия имеют так называемые ***управляющие решения***. Они характеризуются тем, что выбор и реализация решения возлагаются на различные элементы единой, как правило достаточно сложной, системы. Принимается решение управляющим органом, а реализуется — исполнительным. Система, средствами которой формируется и реализуется решение, может быть организационной, технической либо смешанной (комбинированной). Примером организационной системы может служить любое учебное заведение. В нем управляющим органом является ректорат, а исполнительным — профессорско-преподавательский и инженерно-технический состав. Беспилотный самолет-бомбардировщик — образец чисто технической системы (управляющий орган — бортовой компьютер, исполнительный — система бомбометания). Наиболее распространенными в настоящее время являются смешанные системы, которые иногда называют еще «человеко-машинными» системами. К таким системам относится, в частности, любое современное промышленное предприятие.

Одним из важнейших атрибутов сложной системы является наличие целенаправленного поведения. В процессе достижения цели система так или иначе взаимодействует с внешней средой, которая может быть либо «дружелюбной», либо «враждебной», либо нейтральной. Очевидно, чем сложнее система и чем сложнее ее взаимодействие со средой, тем больше существует различных вариантов движения к цели. Одни могут быть лучше, другие — хуже, третьи вообще могут привести к разрушению системы. А может отыскаться один, самый-самый, при котором и волки сыты, и овцы целы.

Итак, чтобы любая система жила долго и счастливо, необходимо уметь, во-первых, оценивать качество всех возможных способов достижения цели и, во-вторых, выбирать из них наилучший с точки зрения интересов системы (либо, по крайней мере, один из пригодных). Для решения указанных задач разработана специальная теория, которая так и называется — *теория принятия решений*. Для дальнейшего изложения нам потребуется ряд терминов и понятий, используемых в этой теории.

В основе принятия решения лежит ***исследование операции***. Под операцией в данном случае понимается процесс достижения цели системой (с учетом ее взаимодействия с внешней средой). Исследование операции заключается в оценке и сравнении возможных способов ее проведения с учетом имеющихся ограничений. Ограничения, как правило, связаны с временными, материальными, людскими или другими видами ***ресурсов***, которые находятся в распоряжении оперирующей стороны (***субъекта операции***). Таким образом, способ проведения операции опреде-

ляется стратегией использования имеющихся ресурсов. Поэтому вместо выражения «способ проведения операции» чаще используют термин ***стратегия***. Такая терминология обусловлена еще и тем, что появление этого раздела математики связано с исследованием военных операций. Стратегии, удовлетворяющие наложенным ограничениям, называются ***допустимыми***. Понятие «допустимая стратегия» является относительным: множество допустимых стратегий изменяется, если изменяются ограничения (или располагаемые ресурсы).

Реализация той или иной допустимой стратегии приводит к различным ***исходам операции***. Качество проведения операции, ее «успешность» оценивается с позиций ***лица, принимающего решение*** (ЛПР). ЛПР — это совсем не обязательно конкретный человек определенной национальности. Под этим термином в теории принятия решений понимается любой управляющий орган, персональный или коллегиальный, имеющий биологическое или техническое воплощение. В указанном смысле оценка качества проведения операции всегда является субъективной. Тем не менее для получения такой оценки должны использоваться объективные методы.

Мерой эффективности проведения операции служит ***показатель эффективности***. В общем случае он отражает результат проведения операции, который, в свою очередь, является функцией трех факторов: полезного эффекта операции *(q)*, затрат ресурсов на проведение операции *(c)* и затрат времени на ее проведение *(t)*. Значения величин q, c и t зависят от стратегии проведения операции *(u)*. В формальном виде сказанное можно записать так:

$$Yon = Y(q(u), c(u), t(u)). \qquad (1.1)$$

В зависимости от того, какие стороны планируемой операции интересуют ЛПР, список аргументов в выражении (1.1) может видоизменяться. Например, если эффективность операции не зависит от ее длительности, то фактор времени $t(u)$ может быть опущен. И наоборот, факторы, наиболее существенные с точки зрения ЛПР, должны быть детализированы. В частности, затраты на проведение операции $c(u)$ могут быть представлены в виде вектора $\{c1(u), c2(u), c3(u), ...\}$, каждая компонента которого соответствует определенному типу ресурсов.

Необходимо отметить, что факторы q, c и t могут носить не только количественный, но и качественный характер. Причем форма их описания зависит как от сферы деятельности, к которой относится рассматриваемая операция (или, как говорят, «предметной области»), так и от возможности и требований к точности их оценки. Разумеется, количественные оценки во многих случаях являются более объективными, однако при решении некоторых задач они либо просто не нужны, либо их получение является слишком трудоемким, а вводимые упрощения искажают сущность решаемой задачи.

В качестве иллюстрации к изложенному рассмотрим ситуацию, которая на протяжении многих лет повторяется в зимнее время на водоемах, окружающих Санкт-

Петербург. Речь идет об операциях по спасению рыбаков, уносимых от берега на оторванных льдинах. Очевидно, что каждая такая операция может быть проведена либо более, либо менее успешно в зависимости от выбранной стратегии поиска потерпевших, используемых сил и средств и т. д. При количественной оценке результата операции факторы в выражении (1.1) получают следующую интерпретацию:

$q(u)$ — число спасенных рыбаков;

$c(u)$ — стоимость спасательных работ (в рублях);

$t(u)$ — время на проведение операции (в часах).

Имея возможность рассчитать указанные величины, можно получить достаточно объективную оценку эффективности выбранной стратегии и проведения спасательной операции. Однако не менее реалистичную оценку можно получить и при использовании качественных значений тех же величин:

$q(u)$ — сохраняя тот же смысл, принимает только одно из двух значений — удалось спасти всех или нет;

$c(u)$ — также имеет два возможных значения — будет превышена смета на проведение работ или нет;

$t(u)$ — удастся ли закончить операцию до наступления темноты.

Несмотря на определенные преимущества качественного подхода к оценке результата операции, сравнение таких оценок связано со значительными трудностями. Для их преодоления создана специальная теория — ***теория отношений***. Описание используемых в ней методов выходит за рамки данной книги. Поэтому все последующее изложение относится к таким задачам поиска решения, в которых показатель эффективности имеет количественное выражение.

Итак, показатель эффективности (ПЭ) позволяет оценить (точнее, описать) результат операции, полученный при использовании конкретной стратегии. Однако даже если такие оценки будут получены для всего множества допустимых стратегий, этого еще не достаточно, чтобы выбрать одну из них, ту, которая будет реализована. Например, при оценке загруженности вычислительной сети оказалось, что коэффициент использования равен 0.7. Хорошо это или плохо? Чтобы ответить на подобный вопрос, необходимо сформулировать правило, позволяющее ЛПР сравнивать между собой стратегии, характеризующиеся различными значениями ПЭ. В одних случаях правило сравнения может быть очень простым, в других же его вообще не удается найти и приходится изменять (уточнять) показатель эффективности. Скажем, если автомобиль одной и той же марки в двух разных автосалонах продается по разным ценам (при прочих равных условиях), то правило выбора салона напрашивается само. Совсем другое дело, когда автомобили различаются стоимостью, фирмой-изготовителем, дизайном, организацией гарантийного обслуживания и т. д. При этом первый оказывается предпочтительнее по одним показателям, второй — по другим. В такой ситуации покупатель должен сначала опреде-

лить правило выбора и только после этого сравнивать между собой различные варианты.

В теории принятия решений правило, на основании которого производится выбор стратегии, отвечающей интересам ЛПР, называется ***критерием эффективности***.

Таким образом, показатель эффективности и критерий эффективности в совокупности отражают цели, которые преследует ЛПР при проведении данной операции, а также наиболее предпочтительный для него способ достижения этой цели.

Читателю, вероятно, известна трагедия буриданова осла, который погиб от голода, так и не выбрав одну из двух охапок сена. Почему это произошло? Да потому, что не было у него ни показателя ни критерия эффективности, отражающих его предпочтения.

Если система предпочтений ЛПР обладает свойствами полноты и направленности, то может быть построена ***модель предпочтений*** ЛПР. Слово «модель» в данном случае означает формализованное описание соответствующих категорий, которое обеспечивает повторение процедуры выбора в однотипных ситуациях при различных исходных данных. Кроме того, модель предпочтений ЛПР может быть использована для автоматизации процесса поиска решения.

Теперь поясним смысл свойств полноты и направленности.

Система предпочтений ЛПР обладает ***свойством полноты*** на множестве D элементов выбора , если она позволяет сравнить между собой любые два элемента $d1, d2 \in D$ и вынести одно из трех альтернативных суждений: а) $d1$ предпочтительнее $d2$; б) $d1$ и $d2$ равноценны; в) $d2$ предпочтительнее $d1$.

Свойство направленности означает следующее. Если, например, при сравнении элементов $d1$ и $d2$ ЛПР выносит суждение «$d1$ предпочтительнее $d2$», а при сравнении элементов $d2$ и $d3$ — «$d2$ предпочтительнее $d3$», то при сравнении элементов $d1$ и $d2$ его вывод должен быть однозначен: «$d1$ предпочтительнее $d3$».

Необходимо отметить, что на практике предпочтения ЛПР непостоянны и могут изменяться даже в одной и той же ситуации выбора. В связи с этим важное значение имеет понятие ***концепции рационального поведения*** ЛПР. Та линия поведения («концепция»), которой придерживается ЛПР, и определяет выбор правила, на основе которого будут сравниваться стратегии.

Согласно теории принятия решений, ЛПР может использовать одну из трех концепций рационального поведения: пригодности, оптимальности или адаптивности.

При использовании ***концепции пригодности*** приемлемой считается любая стратегия, обеспечивающая значение ПЭ не хуже заданного.

Концепция оптимальности требует, чтобы из всего множества допустимых стратегий была выбрана только та, которая приводит к наилучшему («экстремальному») значению ПЭ.

Концепция адаптивного поведения предполагает, что правило выбора может изменяться в соответствии с изменяющимися характеристиками рассматриваемой ситуации.

Поясним различие в использовании различных концепций рационального поведения, воспользовавшись приведенным ранее примером (оценка загруженности сети). Пусть имеются три альтернативные стратегии организации работы сети. Первая обеспечивает значение коэффициента загрузки (Кз), равное 0,6, вторая — 0,7, третья — 0,9.

При использовании концепции пригодности должно быть задано минимальное требуемое значение Кз. Если оно равно 0.7, а оценка стратегий проводилась в порядке их нумерации, то в качестве управляющего решения будет выбрана вторая (хотя при изменении порядка оценки на ее месте может оказаться и третья).

При использовании концепции оптимальности выбор в рассматриваемой ситуации будет всегда однозначен — в качестве стратегии управления работы сетью будет приниматься только третья.

Даже на фоне этого простого примера можно дать краткую сравнительную оценку первых двух концепций: концепция пригодности требует, как правило, меньших затрат времени на поиск решения и обладает определенной гибкостью, зато концепция оптимальности гарантирует выбор наилучшего решения из числа допустимых.

Еще большей гибкостью обладает концепция адаптивности. Для приведенного примера она может быть реализована так: допустимый уровень загрузки сети будет изменяться при изменении параметров сети (в частности, ее конфигурации), а вместе с ним будет изменяться и стратегия управления сетью.

И вот теперь мы вынуждены снова вернуться к понятию результата операции и связанному с ним показателю эффективности. Чтобы сравнивать между собой различные стратегии, необходимо располагать их количественными оценками (т. е. соответствующими значениями ПЭ). Каким образом они могут быть получены? Самый надежный способ — это измерение результата операции после ее реального проведения (при этом под измерением может пониматься подсчет, хронометраж, взвешивание и т. д.). Очевидно, что такой подход связан с целым рядом проблем.

Во-первых, далеко не всегда можно повторить операцию в одних и тех же условиях (погода изменилась, исполнители устали или вышли из строя и т. п.), что, естественно, не позволяет говорить об объективности выбора.

Во-вторых, многие операции просто невозможно провести повторно, используя другую стратегию (например, Курскую битву).

В-третьих, реальное воплощение системы, используемой при проведении операции, как правило является весьма дорогостоящим и трудоемким делом, а если речь идет о сравнении проектных или конструкторских решений, то затраты средств и времени возрастают пропорционально числу сравниваемых вариантов.

Список проблем можно было бы продолжить, но и приведенных вполне достаточно, чтобы сделать вывод: методу измерений должна существовать какая-то альтернатива. И вот здесь на первое место выходит моделирование.

Пока ограничимся достаточно общей трактовкой данного понятия.

Моделирование — это замещение исследуемого объекта (оригинала) его условным образом или другим объектом (моделью) и изучение свойств оригинала путем исследования свойств модели.

Достаточно очевидно, что действительная польза от моделирования может быть получена только при соблюдении двух условий:

- модель обеспечивает корректное (или, как говорят, адекватное) отображение свойств оригинала, существенных с точки зрения исследуемой операции;
- модель позволяет устранить проблемы, присущие проведению измерений на реальных объектах (перечислены выше).

О том, как обеспечить выполнение этих условий, пойдет речь в последующих главах. А пока еще одно предварительное замечание.

В зависимости от способа реализации все модели можно разделить на два больших класса: физические и математические.

Физические модели предполагают, как правило, реальное воплощение тех физических свойств оригинала, которые интересуют ЛПР. Например, при проектировании нового самолета создается его макет, обладающий теми же аэродинамическими свойствами; при планировании застройки архитекторы изготавливают макет, отражающий пространственное расположение ее элементов. В связи с этим физическое моделирование называют также макетированием.

Математическая модель представляет собой формализованное описание системы (или операции) с помощью некоторого абстрактного языка, например в виде совокупности математических соотношений или схемы алгоритма. По большому счету, любое математическое выражение, в котором фигурируют физические величины, можно рассматривать как математическую модель того или иного процесса или явления. В частности, известное всем еще из начальной школы уравнение $S = V \cdot t$, представляет собой модель равномерного прямолинейного движения. Именно математические модели мы и будем рассматривать в дальнейшем как основной инструмент оценки эффективности альтернативных стратегий.

Изложенное в данном разделе можно представить в виде следующей схемы (рис 1.1).

Наиболее примечательным в данной схеме является то, что процесс поиска (выбора) решения носит циклический характер. Имеется в виду, что любой из входящих в него этапов может повторяться неоднократно до тех пор, пока не будет найдено решение, удовлетворяющее требованиям ЛПР (либо не истечет время, отпущенное на принятие решения). При этом могут уточняться цели и условия проведения операции, корректироваться модель предпочтений ЛПР и модель самой операции. Очевидно, длительность и успех поиска зависят не только от знаний и навыков исследователя, но и от того, какие инструменты он использует в своей работе. Именно поэтому основное внимание в книге уделено описанию средств визуального моделирования — программе SIMULINK, входящей в состав универсального математического пакета MATLAB.

И все-таки даже самый хороший инструмент дает наилучший результат только в том случае, когда его обладатель знает больше, чем написано в инструкции по эксплуатации.

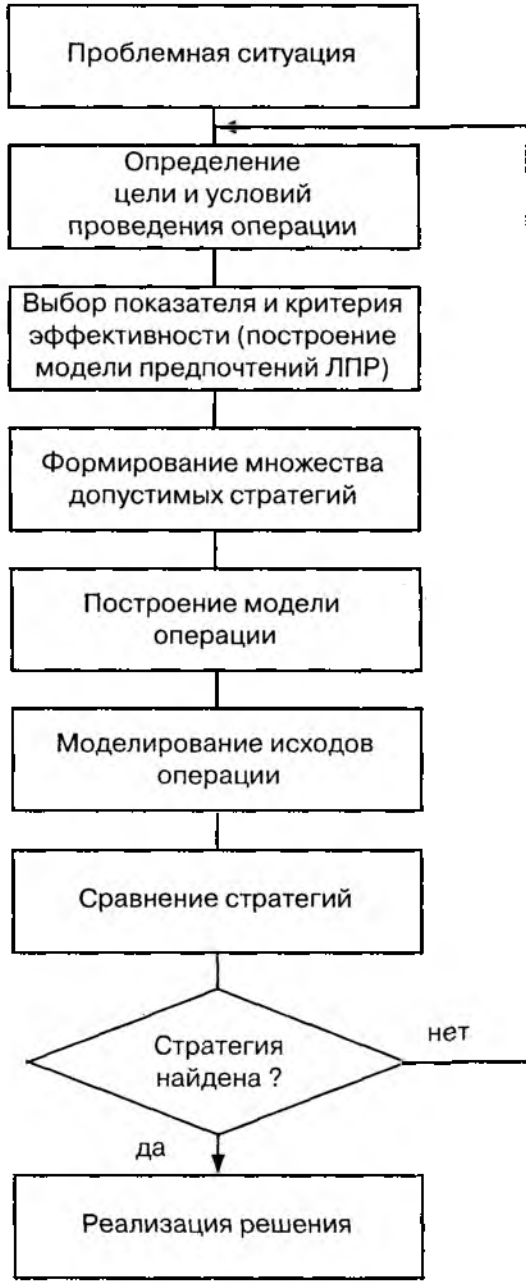


Рис.1.1. Общая схема процесса принятия решения

1.2. КЛАССИФИКАЦИЯ ЗАДАЧ ПРИНЯТИЯ РЕШЕНИЙ

Одно из наиболее важных условий сокращения затрат времени и сил при поиске решения — это умение правильно выбрать метод поиска. За время существования теории принятия решений для наиболее часто встречающихся задач были разработаны методы, учитывающие их характерные особенности.

Практически любая ситуация, требующая принятия решения, может быть отнесена к тому или иному известному классу, и исследователю остается только «узнать» ее. Для этого требуется, по крайней мере, иметь представление о характерных признаках различных классов. В настоящее время отсутствует единая универсальная классификационная схема задач принятия решений, однако практически во всех изданиях, посвященных этим вопросам, фигурируют следующие классификационные признаки:

- число лиц, принимающих решение;
- вид показателя эффективности;
- степень определенности информации о проблемной ситуации;
- зависимость характеристик проблемной ситуации от времени.

По признаку числа ЛПР различают задачи индивидуального и группового принятия решений. При групповом выборе решений определяющую роль играет проблема согласования индивидуальных предпочтений членов группы.

По виду ПЭ задачи принятия решения подразделяют на задачи со скалярным и векторным ПЭ.

При использовании скалярного ПЭ предполагается, что ЛПР интересует только одна из составляющих результата операции (т. е. только одна из характеристик стратегии), например, ее длительность. Это наиболее простой случай при выборе стратегии. Но это не означает, что определить значение скалярного ПЭ тоже легко. Практически все методы математического программирования предназначены для поиска решений именно по скалярному показателю. Необходимо отметить, что при использовании этих методов в роли показателя эффективности выступает целевая функция.

При сравнении стратегий по векторному ПЭ могут быть использованы специальные методы, позволяющие свести векторный показатель к скалярному. Некоторые из них будут рассмотрены в следующем разделе.

Вообще же выбор показателя эффективности является одним из наиболее важных и сложных этапов поиска решения и требует от исследователя не только опыта и знания рассматриваемой предметной области, но и элементов творчества. Наиболее распространенные и полезные в практическом смысле формы ПЭ также будут рассмотрены в следующем разделе.

По степени определенности информации о проблемной ситуации различают задачи принятия решений в условиях определенности и задачи принятия решений в условиях неопределенности.

Задачи принятия решений в ***условиях определенности*** характеризуются наличием полной и достоверной информации о проблемной ситуации, целях, ограничениях и последствиях принимаемых решений. В таких задачах заранее, до начала операции, известно, к какому исходу приведет каждая из стратегий. Это, в частности, означает, что все внешние факторы известны, учтены, и они не могут каким-либо непредвиденным образом повлиять на исход операции.

Характерная особенность всех задач ПР в ***условиях неопределенности*** состоит в том, что исход операции зависит не только от стратегий ЛПР и фиксированных факторов, но и от неопределенных факторов, не контролируемых ЛПР и не известных ему в момент принятия решения (или недостоверно известных). В результате каждая стратегия оказывается связанной с множеством возможных исходов операции, что существенно осложняет процесс выработки решения.

Задачи принятия решений в условиях неопределенности подразделяют на задачи стохастической и нестохастической неопределенности.

В случае ***стохастической неопределенности*** каждой стратегии соответствует некоторое конечное множество исходов, причем исследователю известны их вероятностные характеристики. Но даже если он будет ориентироваться на наиболее вероятный исход, это не означает, что операция будет развиваться именно по данному сценарию. Поэтому задачи такого типа называют также принятием решений в условиях риска. Они имеют место в тех случаях, когда на исход операции могут повлиять те или иные случайные факторы. Например, если встречаются две примерно равные по силам футбольные команды, то возможны три различных исхода матча (либо побеждает первая команда, либо вторая, либо встреча заканчивается вничью); вероятность каждого из исходов известна (равна 1/3), но какой именно будет реализован, остается загадкой практически до окончания игры.

Задачи ПР в условиях ***нестохастической неопределенности*** подразделяются, в свою очередь, на задачи ПР в условиях природной и поведенческой неопределенности.

Такие задачи возникают в тех случаях, когда ЛПР не располагает вероятностными характеристиками возможных исходов операции, либо они вообще не являются случайными. В лучшем случае известны лишь диапазоны их значений.

Если ограниченность информации обусловлена недостаточной изученностью природы рассматриваемых явлений, то говорят о задачах с «природной» неопределенностью. Если же недостаток информации обусловлен влиянием на ход операции других субъектов помимо ЛПР, то имеет место задача «поведенческой» неопределенности. Для решения задач с «поведенческой» неопределенностью используются методы теории игр.

По характеру зависимости проблемной ситуации от времени различают статические и динамические задачи ПР. В динамических задачах параметры (характеристики) проблемной ситуации изменяются во времени.

Классификация задач ПР по перечисленным признакам приводит к различным комбинациям типов задач. Например, выбор варианта атаки летчиком истребителя-перехватчика может быть классифицирован как динамическая скалярная задача индивидуального принятия решения в условиях поведенческой неопределенности, поскольку в данном случае в качестве ПЭ используется вероятность уничтожения цели и для любой выбранной стратегии исходы случайны, что обусловлено наличием фактора неопределенности — поведением противника.

1.3. ОПИСАНИЕ ПРЕДПОЧТЕНИЙ ЛИЦА, ПРИНИМАЮЩЕГО РЕШЕНИЕ

Как было показано в предыдущем разделе, модель предпочтений ЛПР служит основой для принятия решения, т. е. для выбора такой стратегии, которая в наибольшей степени соответствует его интересам в планируемой операции. В свою очередь, предпочтения ЛПР могут быть выражены посредством выбора соответствующих показателя и критерия эффективности.

В простейшем варианте каждой стратегии может быть поставлено в соответствие значение скалярного ПЭ (здесь применимы методы математического программирования). Более сложная (и более распространенная на практике) ситуация — когда каждая из допустимых стратегий характеризуется векторным ПЭ. Трудности выбора еще более возрастают в условиях неопределенности, при отсутствии однозначного соответствия между стратегиями и их векторными оценками.

Но в любом случае для успешного решения задачи выбора необходимо выявить и измерить предпочтения ЛПР.

Под ***выявлением предпочтений ЛПР*** понимают процесс получения информации о суждениях ЛПР относительно возможных исходов операции. Существует два подхода к выявлению предпочтений ЛПР:

- на основе информации о ранее принятых решениях (при многократном повторении выбора в неизменных условиях);
- до принятия решения ЛПР — посредством специальной процедуры опроса.

Измерение предпочтений есть отображение альтернативных вариантов решений на числовую ось.

В случае описания стратегий с помощью скалярного ПЭ измерение предпочтений не вызывает трудностей. Иногда может быть получена функциональная зависимость между частными ПЭ и результатом операции, что позволяет представить результат в виде скалярной величины.

В более сложных случаях используются другие способы измерения предпочтений.

Прежде чем перейти к их описанию, введем понятие шкалы измерений.

Шкала измерений — это система обозначений, позволяющая поставить в соответствие объекту некоторый признак и использовать его в дальнейшем для сравнения объектов между собой.

Наибольшее распространение получили метрические, порядковые и номинальные шкалы. Рассмотрим их в порядке возрастания возможностей.

Номинальная шкала (или шкала наименований) — это, по существу, качественная шкала. Ее применяют для обозначения принадлежности объектов к определенным классам. Она позволяет описать отношение эквивалентности и различия между объектами. Однако предпочтение между объектами и между классами не устанавливается. Числа в этой шкале используются только для обозначения класса объектов. Пример использования номинальной шкалы — номер цеха или производственного участка на предприятии.

Порядковая (ранговая) шкала применяется для измерения упорядоченности объектов по одному признаку или по их совокупности. Числа в ней задают только порядок следования объектов, но не позволяют определить, насколько один объект предпочтительнее другого.

Примерами применения ранговой шкалы являются: распределение мест в эстафетной гонке (когда интерес представляет только порядковый номер на финише, а не разница во времени); назначение приоритетов заявкам, поступающим на обслуживание в вычислительную систему и т. д.

Метрические шкалы являются наиболее совершенными. На практике применяются следующие их виды:

- *шкала интервалов, которая* используется для описания различия свойств объектов в виде разности. Измерения в данной шкале позволяют определить, насколько один объект лучше другого. При этом обязательно задаются масштаб измерений и начало отсчета. В шкале интервалов измеряются, например, сроки выполнения различных работ, гарантийные сроки службы устройств, объем затрат на проведение операции;
- *шкала отношений* — частный случай шкалы интервалов при выборе нулевой точки отсчета. Она является более совершенной, поскольку позволяет определять не только разность, но и отношение между значениями ПЭ. Показатели, измеренные в шкале отношений, наиболее распространены в технике и математике. К ним относятся, например, длина, масса, напряжение и т. д.;
- *абсолютная шкала*, которую принято считать наиболее совершенной. В этой шкале используется нулевая точка отсчета и единичный масштаб. Это означает, что измерения в ней могут быть произведены единственным способом (в отличие, например, от шкалы отношений).

В абсолютной шкале определяется, в частности, количество объектов (предметов, событий), которое может быть измерено единственным образом с помощью ряда натуральных чисел. Абсолютными являются, например, шкала температур по Кельвину, шкала значений вероятности события и т. п.

Ниже рассмотрены некоторые виды показателей эффективности, наиболее часто используемые на практике. Выбор конкретной формы ПЭ во многом зависит от способа описания исхода операции.

1. Цель операции описывается случайным событием A, наступление которого является желательным результатом (целью) операции. Вероятность наступления этого события $Pu(A)$ зависит от стратегии u. В этом случае ПЭ есть вероятность наступления события A: $W(u)=Pu(A)$.

В качестве примера такого ПЭ можно назвать вероятность выигрыша в программе «Русское лото».

Часто рассматриваемое событие A заключается в истинности одного из соотношений:

$$A=Yon(u) < Ymp \text{ или } A=Ymp1< Yon(u)<Ymp2,$$

т. е. операция считается успешной, если ее результат лежит в некотором заданном диапазоне (вспомните критерий пригодности).

При этом ПЭ трактуется как ***вероятностная гарантия*** выполнения поставленной задачи:

$$W(u)=P(Yon(u)\le Ymp).$$

Такой показатель может быть использован для оценки эффективности рассматриваемой операции с точки зрения сроков ее проведения: в качестве Ymp выступает заданная длительность выполнения работ, а в качестве Yon — реальные затраты времени.

2. Если цель операции выражается числовой переменной, зависящей от случайных факторов, то может быть использован еще один показатель эффективности — ***показатель среднего результата:*** $Wu(u)=M[Yon(u)]$.

Зная диапазон изменения ПЭ, исследователь может сравнивать значение среднего результата с граничными (допустимыми) значениями результата операции.

Как было отмечено ранее, выбор стратегии значительно усложняется, если исход операции оценивается с помощью ***векторного ПЭ*** (или множества скалярных ПЭ). Это объясняется тем, что некоторое решение может превосходить остальные по одним показателям и уступать им по другим. В таких условиях трудно определить, которая из стратегий более предпочтительна, не говоря уже об ее оптимальности. Если в задачах ПР по скалярному показателю основная сложность состоит в разработке или выборе метода поиска экстремума, то в задачах ПР по векторному показателю главное внимание уделяется выработке ***решающего правила***, основанного на компромиссе между значениями компонент векторного показателя.

Таким образом, сложность проблемы ПР по векторному показателю даже в условиях определенности связана не столько с трудностями вычисления, сколько с обоснованностью выбора «наилучшего» решения. Невозможно строго математически доказать, что выбранное решение является наилучшим; любое решение из

числа недоминируемых (не улучшаемых одновременно по всем показателям) может оказаться наилучшим для конкретного ЛПР в конкретных условиях. Это является основной аксиомой ПР по нескольким показателям.

В зависимости от способа формирования решающего правила методы ПР по векторному показателю можно условно разделить на две группы. К первой группе относятся ***эвристические методы***, в которых ЛПР определяет вид свертки компонент векторного ПЭ в скалярный на основе «здравого смысла» (или интуиции). Методы второй группы — ***аксиоматические*** — основаны на использовании дополнительной информации о компонентах векторного ПЭ.

Рассмотрим некоторые наиболее распространенные методы, относящиеся к первой группе.

1. Метод обобщенного показателя.

Если исходя из характера задачи можно допустить, что абсолютное уменьшение одного из показателей компенсируетя суммарным абсолютным увеличением других (показатели однородные), то в качестве обобщенного показателя может быть принята сумма такого вида:

$$W_0 = \sum_i^m g_i \cdot W_i,$$

где g_i — коэффициент относительной важности частного показателя W_i.

Иногда допустимой считается не абсолютная, а относительная компенсация изменения одних показателей другими.

Пример: При оценке эффективности боевых действий истребительной авиации могут быть использованы два частных показателя — математическое ожидание (МОЖ) числа уничтоженных целей (W_1) и МОЖ потерь своих самолетов (W_2):

$$W_0 = \sum_i g_i \cdot W_i,$$

где i=1, 2.

2. Метод «затраты — эффект».

Если из существа задачи следует, что одни показатели желательно увеличивать, а другие уменьшать, то в качестве обобщенного ПЭ используют следующее отношение:

$$W_0 = \prod_{i=1}^{m1} W_i \Big/ \prod_{i=m1+1}^{m} W_i,$$

где i=1, ... , $m1$,— номера показателей, значения которых желательно увеличивать;

i=$m1$+1, ... m — номера показателей, значения которых желательно уменьшать.

Часто первая группа показателей отождествляется с целевым эффектом, а вторая — с затратами на его достижение. При этом показатели не обязательно должны быть однородными.

3. Метод целевого программирования.

Основой метода является свертывание частных ПЭ в обобщенный показатель, имеющий смысл расстояния до «идеальной» точки в пространстве значений показателя эффективности. В качестве «идеальной» обычно выбирают точку, отвечающую представлениям ЛПР об идеальном исходе операции.

В этом случае W_0 вычисляется, как правило, следующим образом:

$$W_0=\sum_{i=1}^{m} g_i \cdot (W_i - W_{in})^2,$$

где W_{in} — идеальное значение частного ПЭ W_i.

В качестве ситуации, иллюстрирующей применение данного подхода, рассмотрим выбор одного из двух мест работы, каждое из которых характеризуется величиной зарплаты (W_1), длительностью отпуска (W_2) и временем, затрачиваемым на дорогу (W_3). Если выбирающий определил для себя желаемое значение для каждого из этих показателей (соответственно W_{1n}, W_{2n} и W_{3n}), то обобщенный ПЭ будет выглядеть так:

$$W_0=\sum_{i=1}^{3} g_i \cdot (W_i - W_{in})^2,$$

В этом случае будет выбрано такое место работы, для которого отклонение реальных характеристик от желаемых является наименьшим.

4. Метод главного показателя.

Если ЛПР считает, что целевой эффект достижим в основном вследствие увеличения одного («главного») частного ПЭ, то исходная задача может быть сведена к задаче оптимизации по этому показателю при условии, что значения остальных будут не ниже заданных.

Общим недостатком перечисленных методов является то, что они основаны на некоторых эвристических допущениях, задаваемых ЛПР, и могут не обеспечить действительно лучшей стратегии. Их достоинство — относительная простота реализации.

В тех случаях, когда невозможно произвести свертку векторного ПЭ, необходимо использовать один из аксиоматических методов.

Эти методы основаны на использовании понятия *парето-оптимальности* (по фамилии итальянского экономиста В. Парето). Стратегия называется парето-оптимальной (эффективной), если она по всем показателям не хуже любой стратегии из допустимого множества и лучше хотя бы по одному из них (при взаимной независимости частных ПЭ). Для того чтобы выбрать одну из нескольких эффективных стратегий, необходимо располагать дополнительной информацией. Эта информация должна позволить ЛПР упорядочить частные ПЭ по их важности, после чего могут быть использованы, например, методы свертки, рассмотренные выше.

1.4. ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ МОДЕЛИРОВАНИЯ

Итак, чтобы сравнить между собой различные стратегии проведения операции (или решения), необходимо получить для них ожидаемые значения показателя эффективности. Для этого, в свою очередь, полезно иметь математическую модель исследуемой операции. Таким образом, основная проблема заключается в том, где ее взять. Конечно, можно попросить у знакомых или у родственников, но вряд ли у них есть под рукой именно то, что вам нужно. В любом случае лучше всего рассчитывать на собственные силы, точнее — на собственные знания и опыт. И если опыт приходит только со временем, то соответствующие знания можно получить прямо сейчас.

Начнем с того, что рассмотрим основные принципы моделирования, в сжатой форме отражающие тот достаточно богатый опыт, который накоплен к настоящему времени в области разработки и использования математических моделей.

Принцип информационной достаточности. При полном отсутствии информации об исследуемой системе построение ее модели невозможно. При наличии полной информации о системе ее моделирование лишено смысла. Существует некоторый критический уровень априорных сведений о системе (уровень информационной достаточности), при достижении которого может быть построена ее адекватная модель.

Принцип осуществимости. Создаваемая модель должна обеспечивать достижение поставленной цели исследования с вероятностью, существенно отличающейся от нуля, и за конечное время. Обычно задают некоторое пороговое значение P_0 вероятности достижения цели моделирования $P(t)$, а также приемлемую границу t_0 времени достижения этой цели. Модель считают осуществимой, если одновременно выполнены два неравенства:

$$P(t) \geq P_0;\ t \leq t_0.$$

Принцип множественности моделей. Данный принцип, несмотря на его порядковый номер, является ключевым. Речь идет о том, что создаваемая модель должна отражать в первую очередь те свойства реальной системы (или явления), которые влияют на выбранный показатель эффективности. Соответственно при использовании любой конкретной модели познаются лишь некоторые стороны реальности. Для более полного ее исследования необходим ряд моделей, позволяющих с разных сторон и с разной степенью детальности отражать рассматриваемый процесс.

Принцип агрегирования. В большинстве случаев сложную систему можно представить состоящей из агрегатов (подсистем), для адекватного математического описания которых оказываются пригодными некоторые стандартные математические схемы. Принцип агрегирования позволяет, кроме того, достаточно гибко перестраивать модель в зависимости от задач исследования.

Принцип параметризации. В ряде случаев моделируемая система имеет в своем составе некоторые относительно изолированные подсистемы, характеризующиеся определенным параметром, в том числе векторным. Такие подсистемы можно заменять в модели соответствующими числовыми величинами, а не описывать процесс их функционирования. При необходимости зависимость значений этих величин от ситуации может задаваться в виде таблицы, графика или аналитического выражения (формулы). Принцип параметризации позволяет сократить объем и продолжительность моделирования. Однако надо иметь в виду, что параметризация снижает адекватность модели.

Степень реализации перечисленных принципов в каждой конкретной модели может быть различной, причем это зависит не только от желания разработчика, но и от соблюдения им технологии моделирования. А любая технология предполагает наличие определенной последовательности действий.

Слово «компьютер» пока в нашем повествовании не использовалось. Тем не менее рано или поздно оно должно было появиться. Начнем со словосочетания «компьютерное моделирование», которое все чаще используется в соответствующей литературе. Само по себе это понятие весьма широкое, и каждый автор трактует его по-своему. Встречаются, например, такие выражения: «компьютерное моделирование верхней одежды», «компьютерное моделирование причесок» и т. п. В связи с этим есть необходимость уточнить, что же мы будем понимать под этим термином. Так вот, в данном случае *компьютерное моделирование — это математическое моделирование с использованием средств вычислительной техники.* Соответственно, технология компьютерного моделирования предполагает выполнение следующих действий:

1) определение цели моделирования;
2) разработка концептуальной модели;
3) формализация модели;
4) программная реализация модели;
5) планирование модельных экспериментов;
6) реализация плана эксперимента;
7) анализ и интерпретация результатов моделирования.

Содержание первых двух этапов практически не зависит от математического метода, положенного в основу моделирования (и даже наоборот — их результат определяет выбор метода). А вот реализация остальных шести этапов существенно различается для каждого из двух основных подходов к построению модели. Именуются эти подходы в разных книгах по-разному, мы используем для их обозначения термины «аналитическое» и «имитационное» моделирование.

Аналитическое моделирование предполагает использование математической модели реального объекта в форме алгебраических, дифференциальных, интегральных и других уравнений, связывающих выходные переменные с входными, дополненных системой ограничений. При этом предполагается наличие однозначной вычислительной процедуры получения точного решения уравнений.

При ***имитационном моделировании*** используемая математическая модель воспроизводит логику («алгоритм») функционирования исследуемой системы во времени при различных сочетаниях значений параметров системы и внешней среды.

Примером простейшей аналитической модели может служить уже упоминавшееся уравнение прямолинейного равномерного движения. При исследовании такого процесса с помощью имитационной модели должно быть реализовано наблюдение за изменением пройденного пути с течением времени.

Очевидно, в одних случаях более предпочтительным является аналитическое моделирование, в других — имитационное (или сочетание того и другого). Чтобы выбор был удачным, необходимо ответить на два вопроса:

- с какой целью проводится моделирование;
- к какому классу может быть отнесено моделируемое явление.

Ответы на оба эти вопроса могут быть получены в ходе выполнения двух первых этапов моделирования.

Общая цель моделирования в процессе принятия решения была сформулирована в разделе 1.1 — это определение (расчет) значений выбранного показателя эффективности для различных стратегий проведения операции (или вариантов реализации проектируемой системы). При разработке конкретной модели цель моделирования должна уточняться с учетом используемого критерия эффективности. Для критерия пригодности модель, как правило, должна обеспечивать расчет значений ПЭ для всего множества допустимых стратегий. При использовании критерия оптимальности модель должна позволять непосредственно определять параметры исследуемого объекта, дающие экстремальное значение ПЭ.

Таким образом, цель моделирования определяется как целью исследуемой операции, так и планируемым способом использования результатов исследования.

Например, проблемная ситуация, требующая принятия решения, формулируется следующим образом: найти вариант построения вычислительной сети, который обладал бы минимальной стоимостью при соблюдении требований по производительности и по надежности. В этом случае целью моделирования является отыскание параметров сети, обеспечивающих минимальное значение ПЭ, в роли которого выступает стоимость.

Задача может быть сформулирована иначе: из нескольких вариантов конфигурации вычислительной сети выбрать наиболее надежный. Здесь в качестве ПЭ выбирается один из показателей надежности (средняя наработка на отказ, вероятность безотказной работы и т. д.), а целью моделирования является сравнительная оценка вариантов сети по этому показателю.

Приведенные примеры позволяют напомнить о том, что сам по себе выбор показателя эффективности еще не определяет «архитектуру» будущей модели, поскольку на этом этапе не сформулирована ее концепция, или, как говорят, не определена концептуальная модель исследуемой системы.

Концептуальная (содержательная) ***модель*** — это абстрактная модель, определяющая структуру моделируемой системы, свойства ее элементов и причинно-

следственные связи, присущие системе и существенные для достижения цели моделирования.

Построение концептуальной модели включает следующие этапы:

1) определение типа системы;

2) описание рабочей нагрузки;

3) декомпозицию системы.

На первом этапе осуществляется сбор фактических данных (на основе работы с литературой и технической документацией, проведения натурных экспериментов, сбора экспертной информации и т. д.), а также выдвижение гипотез относительно значений параметров и переменных, для которых отсутствует возможность получения фактических данных. Если полученные результаты соответствуют принципам информационной достаточности и осуществимости, то они могут служить основой для отнесения моделируемой системы к одному из известных типов (классов).

Наиболее важные в этом отношении классификационные признаки приведены ниже.

Одним из них является ***мощность множества состояний*** моделируемой системы. По этому признаку системы делят на статические и динамические. Система называется статической, если множество ее состояний содержит один элемент. Если состояний больше одного, и они могут изменяться во времени, система называется динамической.

Различают два основных типа динамических систем:

- с дискретными состояниями (множество состояний конечно или счетно);
- с непрерывным множеством состояний.

Возможны смешанные случаи.

Процесс смены состояний называется ***движением системы***.

Смена состояний может происходить либо в фиксированные моменты времени, множество которых дискретно и заранее определено (например, поступление новых партий товара на склад), либо непрерывно (изменение курсов валют в ходе торгов). При этом различают детерминированные системы и стохастические. В детерминированных системах новое состояние зависит только от времени и текущего состояния системы. Другими словами, если имеются условия, определяющие переход системы в новое состояние, то для детерминированной системы можно однозначно указать, в какое именно состояние она перейдет.

Для стохастической системы можно указать лишь множество возможных состояний перехода и, в некоторых случаях, вероятности перехода в каждое из этих состояний.

Рассмотренная схема классификации систем важна не сама по себе. На этапе разработки концептуальной модели она, во-первых, позволяет уточнить цели и задачи моделирования и, во-вторых, облегчает переход к этапу формализации модели. Кроме того, значительно позже, на этапе оценки качества разработанной модели, знание классификационных признаков дает возможность оценить степень ее соответствия первоначальному замыслу разработчика.

Необходимо отметить, что рассмотренные классификационные признаки применимы и для определения типа разрабатываемой модели. При этом исследуемая система и ее модель могут относится как к одному, так и к разным классам. Например, реальная система может быть подвержена воздействию случайных факторов и, соответственно, будет относиться к классу стохастических систем. Если разработчик модели считает, что влиянием этих факторов можно пренебречь, то создаваемая модель будет представлять собой детерминированную систему. Аналогичным образом возможно отображение системы с непрерывным временем смены состояний в модель с дискретными переходами и т. д. Разумеется, принадлежность реальной системы и ее модели к одному классу говорит о корректности модели, однако с точки зрения интересов исследования такое «зеркальное отображение» далеко не всегда является полезным (вспомните принцип множественности моделей). Подробнее этот вопрос будет рассмотрен при обсуждении этапа декомпозиции системы.

При исследовании эффективности операции весьма важную роль играет корректное описание условий ее протекания. Как правило, оно представляет собой перечень и характеристики внешних факторов, воздействующих на исполнительную подсистему, используемую ЛПР для достижения целей операции. Если при сравнении различных стратегий другие виды материальных ресурсов не рассматриваются, то задача исследования эффективности операции может быть сформулирована как задача оценки эффективности исполнительной подсистемы (именно в этом смысле ранее наряду с понятием «эффективность операции» использовалось понятие «эффективность системы»). В этом случае вместо условий проведения операции удобнее рассматривать рабочую нагрузку соответствующей системы.

Итак, ***рабочая нагрузка*** — это совокупность внешних воздействий, оказывающих влияние на эффективность применения данной системы в рамках проводимой операции.

Например, пусть оценивается производительность бортовой вычислительной системы (ВС) при управлении полетом космического корабля. В качестве параметров рабочей нагрузки такой ВС целесообразно рассматривать поток информации, подлежащей обработке, и поток отказов, приводящий к нарушению вычислительного процесса. Оценки производительности ВС будут иметь смысл только в том случае, если известно, для какой рабочей нагрузки они получены. Это утверждение справедливо для любой задачи принятия решения, к какой бы предметной области она ни относилась. Нельзя говорить о прочности моста, не указывая, на какую максимальную нагрузку он рассчитан; точно так же некорректно сообщать максимальную скорость автомобиля, не уточнив, в каких условиях она была достигнута.

Описание рабочей нагрузки является не только важной, но и достаточно сложной задачей. Особенно в тех случаях, когда приходится учитывать влияние случайных факторов, или когда речь идет о рабочей нагрузке проектируемой принци-

пиально новой системы. В связи с этим многие авторы вводят понятие модели рабочей нагрузки, подчеркивая сопоставимость уровня сложности описания собственно системы и ее рабочей нагрузки.

Модель рабочей нагрузки (РН) должна обладать следующими основными свойствами:

- совместимостью с моделью системы;
- представительностью;
- управляемостью;
- системной независимостью.

Свойство ***совместимости*** предполагает, что, во-первых степень детализации описания РН соответствует детализации описания системы; во-вторых, модель РН должна быть сформулирована в тех же категориях предметной области, что и модель системы. Например, если в модели системы исследуется использование ресурсов, то РН должна быть выражена в запросах на ресурсы;

Представительность модели РН определяется ее способностью адекватно представить РН в соответствии с целями исследования. Другими словами, модель РН должна отвечать целям исследования системы. Например, если оценивается пропускная способность, должна выбираться РН, «насыщающая» систему.

Под ***управляемостью*** понимается возможность изменения параметров модели РН в некотором диапазоне, определяемом целями исследования.

Системная независимость — это возможность переноса модели РН с одной системы на другую с сохранением ее представительности. Данное свойство наиболее важно при решении задач сравнения различных систем или различных модификаций одной системы. Если модель РН зависит от конфигурации исследуемой системы или других ее параметров, то использование такой модели для решения задачи выбора невозможно.

И наконец, обратимся к этапу, завершающему построение концептуальной модели системы — ее декомпозиции.

Декомпозиция системы производится исходя из выбранного уровня детализации модели, который, в свою очередь, определяется тремя факторами:

- целями моделирования;
- объемом априорной информации о системе;
- требованиями к точности и достоверности результатов моделирования.

Уровни детализации иногда называют ***стратами***, а процесс выделения уровней — ***стратификацией***.

Детализация системы должна производиться до такого уровня, чтобы для каждого элемента были известны или могли быть получены зависимости его выходных характеристик от входных воздействий, существенные с точки зрения выбранного показателя эффективности.

Повышение уровня детализации описания системы позволяет получить более точную ее модель, но усложняет процесс моделирования и ведет к росту затрат времени на его проведение.

Например, если моделируется дискретная система, то более детальное ее описание означает увеличение числа различных состояний системы, учитываемых в модели, и, как следствие — неизбежный рост объема вычислений.

Поэтому при выборе уровня описания системы целесообразно руководствоваться следующим правилом: в модель должны войти все параметры, которые обеспечивают определение интересующих исследователя характеристик системы на заданном временном интервале ее функционирования; остальные параметры по возможности следует исключить из модели.

При имитационном моделировании для оценки выбранного уровня детализации можно использовать специальные критерии.

Первый из них — отношение реального времени функционирования системы к времени моделирования (т. е. к затратам машинного времени, необходимого на проведение модельного эксперимента). Например, если при одних и тех же подходах к программной реализации модели моделирование одного часа работы системы требует в одном случае 3 минуты машинного времени, а в другом — 10 минут, то во втором случае степень детализации описания выше (соотношение 3:10).

Второй критерий — разрешающая способность модели, в том числе:

разрешающая способность *по времени* — может быть определена как кратчайший интервал модельного времени между соседними событиями;

разрешающая способность *по информации* — наименьшая идентифицируемая порция информации, представимая в модели (для вычислительных систем, например, такими порциями могут быть слово, страница, программа, задание).

Третий критерий — число различных моделируемых состояний системы (или типов событий).

Для тех компонентов, относительно которых известно или предполагается, что они сильнее влияют на точность результатов, степень детальности может быть выше других.

Необходимо отметить, что с увеличением детальности возрастает устойчивость модели, но возрастают и затраты машинного времени на проведение модельного эксперимента.

ГЛАВА 2
ОСНОВЫ ТЕХНОЛОГИИ ИМИТАЦИОННОГО МОДЕЛИРОВАНИЯ

— Я взмахну рукой и закричу: «А-а-а». Как только вы увидите мое махание... то есть машение... сразу пускайте вот этот старинный секундомер. Вы увидите, что «А-а-а» будет лететь до вас ровно три секунды. И те, которые не верят, сразу поверят.

Э. Успенский

2.1. ПОНЯТИЕ СТАТИСТИЧЕСКОГО ЭКСПЕРИМЕНТА

Содержание предыдущей главы можно, по большому счету, уместить в одной фразе: чтобы сравнить различные решения между собой, их требуется «измерить», а самым удобным измерительным инструментом является математическое моделирование. Какую роль в этом процессе играет эксперимент?

Когда вы после плотного обеда встаете на весы, чтобы проверить возникшее подозрение — это эксперимент, но когда речь идет о проведении расчетов на аналитической модели, слово «эксперимент» представляется не вполне подходящим. Действительно, трудно назвать экспериментом процедуру, в ходе которой раз за разом будет получаться один и тот же результат, сколько бы ее ни повторяли.

О каком же эксперименте идет речь?

Речь идет об оценке эффективности системы с помощью имитационного моделирования. В данном случае смысл термина в полной мере соответствует происходящему, поскольку имитационное моделирование представляет собой наблюдение поведения модели системы под влиянием входных воздействий. При этом часть из них (а может быть, и все) носят случайный характер. В результате такого наблюдения исследователь получает набор экспериментальных данных, на основе которых могут быть оценены характеристики системы.

Очевидно, что аналитические модели для проведения имитационного эксперимента не годятся, и здесь нужна специальная «имитационная» модель.

Такая модель должна отвечать двум основным требованиям:

- во-первых, отражать логику функционирования исследуемой системы во времени;
- во-вторых, обеспечивать возможность проведения статистического эксперимента.

Методы описания в модели логики функционирования системы будут рассмотрены немного позже.

Сейчас остановимся на понятии «статистический эксперимент».

В его основе лежит метод статистических испытаний (метод Монте-Карло). Суть его состоит в том, что результат испытания ставится в зависимость от значения некоторой случайной величины (СВ), распределенной по заданному закону. Поэтому результат каждого отдельного испытания также носит случайный характер.

Проведя серию испытаний, получают множество частных значений наблюдаемой характеристики (то есть выборку). Полученные статистические данные обрабатываются и представляются в виде соответствующих численных оценок интересующих исследователя величин (характеристик системы).

Теоретической основой метода статистических испытаний являются предельные теоремы теории вероятностей (теорема Чебышева, теорема Бернулли, теорема Пуассона). Принципиальное значение предельных теорем состоит в том, что они гарантируют высокое качество статистических оценок при весьма большом числе испытаний.

Важно отметить, что метод статистических испытаний применим для исследования как стохастических, так и детерминированных систем.

Еще одной важной особенностью данного метода является то, что его реализация практически невозможна без использования ЭВМ, или, как теперь принято говорить, компьютера.

В качестве иллюстрации к изложенному рассмотрим применение метода статистических испытаний для вычисления площади круга заданного радиуса. Данная задача явно относится к классу детерминированных, поскольку весьма сложно представить себе случайные факторы, под влиянием которых площадь неподвижной геометрической фигуры могла бы изменяться. И тем не менее...

Пусть круг имеет радиус $r=5$, и его центр находится в точке с координатами (1, 2). Уравнение соответствующей окружности имеет вид:

$$(x-1)^2 + (y-2)^2 = 25.$$

Для решения задачи методом Монте-Карло впишем круг в квадрат. Его вершины будут иметь координаты (−4,−3), (6,−3), (−4,7) и (6,7). Любая точка внутри квадрата или на его границе должна удовлетворять неравенствам $-4 < x < 6$ и $-3 < y < 7$.

При решении данной задачи естественно исходить из того, что все точки в этом квадрате могут появляться с одинаковой вероятностью, то есть x и y распределены равномерно с плотностями вероятности

$$f(x) = \begin{cases} 1/10 \text{ для } -4 \le x \le 6 \\ 0 \text{ в противном случае;} \end{cases}$$

$$f(y) = \begin{cases} 1/10 \text{ для } -3 \le y \le 7 \\ 0 \text{ в противном случае.} \end{cases}$$

Проведя некоторое количество испытаний (то есть получив множество случайных точек, принадлежащих квадрату), подсчитаем число точек, попавших внутрь

круга или на окружность. Если выборка состоит из n наблюдений и m из n точек попали внутрь круга или на окружность, то оценку площади круга можно получить из соотношения

$$S\text{кр} = S\text{кв} \cdot m/n = 100 \cdot m/n.$$

В таблице приведены оценки Sкр, полученные для разных значений n, причем для каждого n выполнялось 5 прогонов (точное значение Sкр = 78,54 см):

Таблица 2.1

Результаты оценки площади круга методом статических испытаний

Номер прогона	Оценки площади круга (Sкр)				
	Число испытаний (n)				
	100	200	1000	5000	10000
1	78	79,5	77	79,5	77
2	70	77	78	1,88	78,8
3	81	77,3	80,2	79,5	79,8
4	70	79,12	1,29	78,22	78,6
5	79	77,72	77,76	79	78,26
Среднее	75,6	0,3	78,77	78,23	78,88
Дисперсия	27,3	78,63	78,21	78,544	0,1

Прогоны отличаются друг от друга последовательностями случайных чисел, из которых формировались координаты точек.

На основании полученных результатов могут быть сделаны выводы, которые справедливы для любого имитационного эксперимента независимо от физической природы и типа моделируемой системы:

- Каждый прогон модели можно рассматривать как одно наблюдение в проводимом эксперименте на модели.
- С увеличением продолжительности прогона (то есть продолжительности наблюдения) отклонение измеряемой величины от ее точного значения уменьшается, поскольку наблюдаемая система переходит в стационарное состояние.
- Влияние переходных условий можно уменьшить, если увеличить количество прогонов модели (то есть количество экспериментов).
- Существует предел, за которым увеличение продолжительности прогона модели уже не дает существенного повышения точности результата, измеряемой дисперсией.

Основная цель рассмотренного примера — привлечь внимание к тому факту, что имитационное моделирование не ограничивается разработкой модели и напи-

санием соответствующей программы, а требует подготовки и проведения статистического эксперимента. В связи с этим результаты имитационного моделирования следует рассматривать как экспериментальные данные, требующие специальной обработки и анализа. В частности, для любого модельного эксперимента необходимо ответить на следующие вопросы:

1. Какова должна быть продолжительность прогона для достижения стационарных условий?
2. Как получить статистически независимые наблюдения?
3. Сколько наблюдений необходимо для обеспечения требуемой точности?

Ответы на эти вопросы будут даны в последующих разделах.

2.2. ОБЛАСТЬ ПРИМЕНЕНИЯ И КЛАССИФИКАЦИЯ ИМИТАЦИОННЫХ МОДЕЛЕЙ

До сих пор речь шла о том, что моделирование можно считать основным инструментом, обеспечивающим принятие своевременных и обоснованных решений. Тем не менее основной темой книги является не столько сам процесс принятия решений, сколько технология применения имитационного моделирования в различных областях человеческой деятельности. Поэтому пришло время несколько расширить круг обсуждаемых вопросов и поговорить о том, что же представляет собой полноценная имитационная модель и в каких случаях она может оказаться действительно полезной.

Начнем с того, что попытаемся сформулировать основные характерные черты нашей «героини».

Имитационная модель (ИМ) — это формальное (то есть выполненное на некотором формальном языке) описание логики функционирования исследуемой системы и взаимодействия отдельных ее элементов во времени, учитывающее наиболее существенные причинно-следственные связи, присущие системе, и обеспечивающее проведение статистических экспериментов.

При разработке ИМ остаются справедливыми основные принципы моделирования, рассмотренные в разделе 1.4.

В качестве следствия из этого утверждения необходимо отметить два важных обстоятельства:

1) взаимосвязь между отдельными элементами системы, описанными в модели, а также между некоторыми величинами (параметрами) может быть представлена в виде аналитических зависимостей (например, при моделировании полета управляемой ракеты отработка поступающих на борт команд может быть описана на уровне логики, а возникающие перегрузки рассчитываются аналитически);

2) модель можно считать реализуемой и имеющей практическую ценность только в том случае, если в ней отражены лишь те свойства реальной системы, которые влияют на значение выбранного показателя эффективности.

Как было отмечено выше, для ИМ практически отсутствуют ограничения на область их применения (по типу моделируемой системы), и речь может идти только о целесообразности использования ИМ в данной предметной области и об объеме трудозатрат на ее разработку.

Поскольку основой имитационного моделирования является метод статистических испытаний, наибольший эффект от его применения достигается при исследовании сложных систем, на функционирование которых существенное влияние оказывают случайные факторы.

Применение имитационного моделирования целесообразно также в следующих случаях:

1) если не существует законченной постановки задачи на исследование и идет процесс познания объекта моделирования;

2) если характер протекающих в системе процессов не позволяет описать эти процессы в аналитической форме;

3) если необходимо наблюдать за поведением системы (или отдельных ее компонентов) в течение определенного периода, в том числе с изменением скорости протекания процессов;

4) при изучении новых ситуаций в системе либо при оценке функционирования ее в новых условиях;

5) если исследуемая система является элементом более сложной системы, другие элементы которой имеют реальное воплощение;

6) когда необходимо исследовать поведение системы при введении в нее новых компонентов;

7) при подготовке специалистов и освоении новой техники (в качестве тренажеров).

Приведенный список возможных областей применения имитационных моделей (который одновременно можно рассматривать и как перечень их достоинств) невольно вызывает вопрос: а зачем же в таком случае нужны остальные виды моделей? К сожалению, имитационные модели имеют целый ряд недостатков. Первый, и весьма существенный, заключается в том, что разработка ИМ, как правило, требует больших затрат времени и сил. Кроме того, любая имитационная модель сложной системы значительно менее «объективна», чем аналитическая модель, поскольку она прежде всего отражает субъективные представления разработчика о моделируемой системе. Причем бывает достаточно сложно как опровергнуть, так и обосновать адекватность созданной ИМ, особенно если речь идет о проектируемой системе. И, наконец, еще одно обстоятельство. Результаты имитационного моделирования, как и при любом численном методе, всегда носят частный характер. Для получения обоснованных выводов необходимо проведение серии модельных экспериментов.

а обработка результатов требует применения специальных статистических процедур.

Каким же образом можно преодолеть указанные недостатки?

Во-первых, современное состояние вычислительной техники и ее программного обеспечения позволило вооружить «модельеров» такими мощными инструментальными средствами моделирования (к которым, в частности, относится пакет MATLAB), что по мере накопления опыта в их использовании возможные трудозатраты «стремятся к нулю»; то же самое можно сказать о средствах статистического анализа и визуализации полученных результатов.

Во-вторых, «объективность» создаваемой модели может быть обеспечена в том случае, когда разработчик ясно представляет себе, какие именно характеристики исследуемой системы его интересуют: длительности выполнения определенных операций, вероятность перехода системы в некоторое состояние, возможность конфликта между отдельными подсистемами и т.д. Дело в том, что для каждого варианта постановки задачи исследования может быть выбрана соответствующая схема построения модели.

В этом отношении знание существующих схем построения имитационных моделей является весьма полезным.

Наиболее важный признак — ***способ представления в модели динамики (движения) системы***. Она может быть описана посредством событий, работ (активностей), процессов и транзактов.

Другой важный признак — ***способ изменения модельного времени***. По этому признаку различают моделирование с постоянным шагом и моделирование по особым состояниям.

Все эти понятия являются основополагающими в теории имитационного моделирования, и их достаточно подробному толкованию посвящены соответствующие разделы книги.

В большинстве случаев конечной целью моделирования является оптимизация тех или иных параметров системы. Однако, как было отмечено выше, потенциальные возможности имитационного моделирования существенно шире. В зависимости от этапа и назначения проводимых исследований применяется один из трех наиболее распространенных видов имитационных экспериментов:

1) исследование относительного влияния различных факторов на значения выходных характеристик системы;

2) нахождение аналитической зависимости между интересующими исследователя выходными характеристиками и факторами;

3) отыскание оптимальных значений параметров системы (так называемый «экстремальный эксперимент»).

Вид эксперимента влияет не только на выбор схемы ее формализации, но также на построение плана эксперимента и выбор метода обработки его результатов.

С точки зрения организации взаимодействия исследователя с моделью в ходе эксперимента ИМ делятся на автоматические и диалоговые.

Автоматическими называются ИМ, взаимодействие пользователя с которыми сводится только к вводу исходной информации и управлению началом и окончанием работы моделей.

Диалоговыми называются ИМ, позволяющие исследователю активно управлять ходом моделирования.

2.3. ОПИСАНИЕ ПОВЕДЕНИЯ СИСТЕМЫ

Описание динамики системы, или, проще говоря, ее поведения, составляет основу любой имитационной модели. В качестве исходных посылок для решения этой задачи используются результаты, полученные на этапе разработки концептуальной модели системы. К ним относятся:

- определение принадлежности моделируемой системы одному из известных классов;
- описание рабочей нагрузки системы;
- выбор уровня детализации представления системы в модели и ее декомпозиция.

Все последующие действия исследователя по созданию модели могут быть отнесены к этапу ее формализации, который в общем случае предполагает:

- выбор метода отображения динамики системы (на основе событий, процессов или транзактов);
- формальное (математическое) описание случайных факторов, подлежащих учету в модели;
- выбор механизма изменения и масштаба модельного времени.

Начнем с определения понятий «процесс», «работа», «событие», «транзакт».

Работа (активность) — это единичное действие системы по обработке (преобразованию) входных данных. В зависимости от природы моделируемой системы под входными данными могут пониматься информационные данные или какие-либо материальные ресурсы.

Каждая из работ характеризуется временем выполнения и потребляемыми ресурсами.

Под ***процессом*** понимают логически связанный набор работ. Некоторые процессы могут рассматриваться, в свою очередь, как работы в процессе более высокого уровня.

Любой процесс характеризуется совокупностью статических и динамических характеристик.

К ***статическим*** характеристикам процесса относятся:

- длительность;
- результат;
- потребляемые ресурсы;
- условия запуска (активизации);
- условия останова (прерывания).

В общем случае статические характеристики процесса не изменяются в ходе его реализации, однако при необходимости любая из них может быть представлена в модели как случайная величина, распределенная по заданному закону.

Динамической характеристикой процесса является его состояние (активен или находится в состоянии ожидания).

Моделирование в терминах процессов производится в тех случаях, когда система оценивается по каким-либо временным показателям, либо с точки зрения потребляемых ресурсов.

Например, при оценке производительности вычислительной сети обработка заданий может быть представлена в модели как совокупность соответствующих процессов, использующих ресурсы сети (оперативную память, пространство на жестких дисках, процессорное время, принтеры и т. д.).

В том случае, если модель строится с целью изучения причинно-следственных связей, присущих системе, динамику системы целесообразно описывать в терминах событий.

Событие представляет собой мгновенное изменение некоторого элемента системы или состояния системы в целом.

Событие характеризуется:

- условиями (или законом) возникновения;
- типом, который определяет порядок обработки (дисциплину обслуживания) данного события;
- нулевой длительностью.

Обычно события подразделяют на две категории:

события следования, которые управляют инициализацией процессов (или отдельных работ внутри процесса);

события изменения состояний (элементов системы или системы в целом).

Как было отмечено, механизм событий используется в качестве основы построения моделей, предназначенных для исследования причинно-следственных связей в системах при отсутствии временных ограничений. К таким задачам можно отнести, например, некоторые задачи по оценке надежности.

Еще один способ имитационного моделирования систем основан на использовании понятия транзакта.

Транзакт — это некоторое сообщение (заявка на обслуживание), которое поступает извне на вход системы и подлежит обработке. В некоторых случаях, например, при моделировании автоматизированных систем управления, более удобно проследить функционирование системы именно относительно алгоритма обработки транзакта. В рамках одной ИМ могут рассматриваться транзакты нескольких типов. Каждый транзакт характеризуется соответствующим алгоритмом обработки и необходимыми для его реализации ресурсами системы. Учитывая это, прохождение транзакта по системе можно в некоторых случаях рассматривать как последовательную активизацию процессов, реализующих его обработку («обслуживание заявки»).

В связи с упоминанием термина «обслуживание заявки» уместно вспомнить о существовании теории массового обслуживания. Если читатель с ней знаком, то остается только добавить, что при разработке и исследовании имитационных моделей на основе транзактов целесообразно использовать методику и показатели, применяемые при анализе систем массового обслуживания. Во второй части книги приводятся примеры, поясняющие данный подход.

2.3.1. МОДЕЛИРОВАНИЕ СЛУЧАЙНЫХ ФАКТОРОВ

Еще раз отметим, что ИМ позволяет исследовать поведение различных систем с учетом влияния случайных факторов. Эти факторы в зависимости от их природы могут быть отражены в модели как случайные события, случайные величины (дискретные или непрерывные), или как случайные функции (процессы).

Например, если с помощью создаваемой имитационной модели предполагается исследовать надежность вычислительной системы, то возникновение отказа будет представлено в модели как случайное событие. Если же модель предназначена для оценки временных параметров процесса обслуживания клиентов в автомастерской, то интервал времени до появления очередного клиента удобнее всего описать как случайную величину, распределенную по некоторому закону.

В основе всех методов и приемов моделирования случайных факторов лежит использование случайных чисел, имеющих равномерное распределение на интервале [0; 1].

«Истинно» случайные числа формируются с помощью аналого-цифровых преобразователей на основе сигналов физических генераторов, использующих естественные источники случайных шумов (радиоактивный распад, шумы электронных и полупроводниковых устройств и т.п.).

Случайные числа, генерируемые аппаратно или программно на ЭВМ, называются псевдослучайными. Однако их статистические свойства совпадают со статистическими свойствами «истинно» случайных чисел. В состав практически всех современных систем программирования входят специальные функции генерации случайных чисел, которые обычно называют датчиками, или генераторами, случайных чисел.

Наиболее простой метод программной генерации случайных чисел — мультипликативный; в его основе лежит следующее рекуррентное соотношение:

$$a_i = (A \cdot a_{i-1} + C) \bmod M, \qquad (2.1)$$

где a_i, a_{i-1} — очередное и предыдущее случайные числа соответственно;

A, C — константы;

M — достаточно большое целое положительное число (чем больше М, тем длиннее неповторяемая последовательность).

Достоинство метода заключается в том, что при одних и тех же значениях входящих в выражение (2.1) величин можно полностью воспроизвести эксперимент.

Практика показывает, что результаты имитационного моделирования существенно зависят от качества используемых последовательностей псевдослучайных чисел. Поэтому применяемые в ИМ генераторы СЧ должны пройти тесты на пригодность.

Основные анализируемые характеристики генерируемых датчиком последовательностей:

- равномерность;
- стохастичность (случайность);
- независимость.

Рассмотрим методы проведения такого анализа, наиболее часто применяемые на практике.

Проверка равномерности может быть выполнена с помощью гистограммы относительных частот генерируемой СВ. Для ее построения интервал [0; 1] разбивается на m равных частей и подсчитывается относительное число попаданий значений СВ в каждый интервал. (Более подробно алгоритм построения гистограммы относительных частот рассматривается в разделе 2.6). Чем ближе огибающая гистограммы к прямой, тем в большей степени генерирумая последовательность отвечает требованию равномерности распределения (см. рис.2.1).

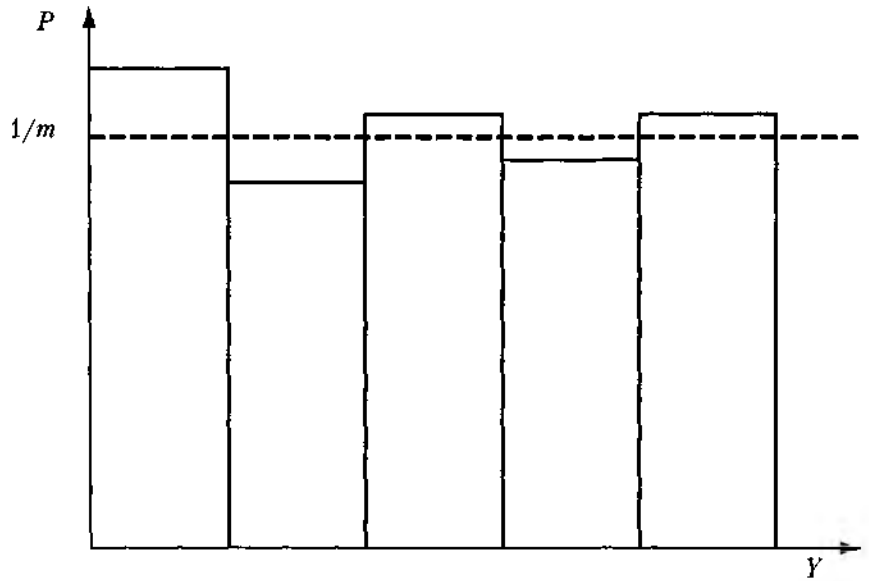


Рис. 2.1.Частотная гистограмма последовательности СЧ

Проверка стохастичности. Рассмотрим один из основных методов проверки – метод комбинаций. Суть его сводится к следующему. Выбирают достаточно большую последовательность случайных чисел x_i и для нее определяют вероятность появления в каждом из x_i ровно j единиц. При этом могут анализироваться как все

разряды числа, так и только l старших. Теоретически закон появления j единиц в l разрядах двоичного числа может быть описан как биномиальный закон распределения (исходя из независимости отдельных разрядов).

Тогда при длине выборки N ожидаемое число появлений случайных чисел x_i с j единицами в проверяемых l:

$$n_j = N \cdot C_l^j p^l(1),$$

где C_l^j — число комбинаций (сочетаний) j единиц в l разрядах,

$p^l(1)$ — вероятность появления единицы в двоичном разряде, $p(1) = 0.5$.

Для полученной последовательности определяется эта же характеристика. Проверка соответствия реального значения теоретическому выполняется с помощью одного из статистических критериев согласия.

Проверка независимости проводится на основе вычисления корреляционного момента.

Напомним, что две случайные величины a и b называются независимыми, если закон распределения каждой из них не зависит от того, какое значение приняла другая. Для независимых СВ корреляционный момент равен нулю.

Для оценки независимости элементов последовательности поступают следующим образом.

Вводят в рассмотрение дополнительную последовательность Y, в которой $y_i = x_1 + t$, где t — величина сдвига последовательности Y относительно исходной последовательности X.

Вычисляют коэффициент корреляции случайных величин X и Y, для чего используются специальные расчетные соотношения.

Еще одна важная характеристика датчика СЧ — длина отрезка апериодичности L.

Если в основу работы датчика положен мультипликативный метод, то оценить L несложно: она определяется величиной константы M.

Моделирование случайных событий. Для моделирования случайного события A, вероятность которого равна Pc, достаточно сформировать одно число r, равномерно распределенное на интервале $[0,1]$.

При попадании r в интервал $[0,Pc]$ считают, что событие A наступило, в противном случае — не наступило, т. е.:

$$A = \begin{cases} 1, \text{ если } r \in [0,Pc] \\ 0, \text{ если } r \in]Pc,1]. \end{cases}$$

Для моделирования полной группы N несовместных событий $A=\{A1, A2, ..., AN\}$ с вероятностями соответственно $P1, P2, ..., PN$ также достаточно одного значения r: событие A_i из группы A считается наступившим, если выполняется условие

$$A = A_i \Big| \sum_{i=0}^{n-1} P_i < r < \sum_{i=0}^{n} P_i ; n=1,N \qquad (1.3)$$

Если группа событий A не полна, то вводят фиктивное событие A_{N+1} с вероятностью P_{N+1}, такой, что

$$\sum_{i=0}^{N+1} P_i = 1,$$

т. е. дополняют группу A до полной.

После этого генерируют число r и проверяют условие (1.3). При $A=A_{N+1}$ считают, что ни одно событие из исходной группы A не наступило.

Моделирование дискретных случайных величин. Наиболее часто используются два метода:

- метод последовательных сравнений;
- метод интерпретации.

Метод последовательных сравнений. Число r последовательно сравнивают со значением суммы

$$P1 + P2 + \ldots,$$

где $P1$ — вероятность наименьшего значения СВ Y,

$P2$ — вероятность второго по величине значения.

При первом выполнении условия

$$r > \sum_{i=1}^{n} P_i$$

проверка прекращается и дискретная СВ Y считается принявшей значение y_i.

Процесс можно ускорить, применяя методы оптимизации перебора: дихотомии, ранжирования P и т. д.

Величины P_i рассчитывают по функциям распределения вероятности, соответствующим моделируемому закону.

Метод интерпретации основан на физической трактовке моделируемого закона распределения. Например, биномиальное распределение описывает число успехов в n независимых испытаниях с вероятностью успеха в каждом испытании P и вероятностью неудачи $g = 1-P$.

При моделировании этого распределения с помощью метода интерпретации выбирают n независимых СЧ, равномерно распределенных на интервале [0, 1], и подсчитывают количество тех из них, которые меньше P. Это число и является моделируемой СВ.

Моделирование непрерывных СВ. При моделировании непрерывных СВ с заданным законом распределения могут используются три метода:

- метод нелинейных преобразований;
- метод композиций;
- табличный метод.

Первые два метода требуют от разработчика модели весьма серьезной математической подготовки и в данной книге не рассматриваются. Третий метод, условно названный «табличным», основан на замене закона распределения непрерывной СВ специальным расчетным соотношением, которое позволяет вычислять значение СВ по значению случайного числа, равномерно распределенного все на том же интервале [0,1]. Такие соотношения получены практически для всех наиболее распространенных видов распределений и приведены в справочной литературе. В качестве примера ниже даны расчетные соотношения для двух законов распределения — показательного

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

где λ — параметр показательного распределения,

r — равномерно распределенное СЧ,

и нормального

$$x = m + s\left[\sum_{i=1}^{12} r_i - 6\right],$$

m, s — параметры нормального закона распределения,

r_i — равномерно распределенное СЧ.

В одной и той же имитационной модели могут фигурировать различные случайные факторы , одни могут быть представлены как случайные события, другие — как случайные величины; более того, если моделируется поведение достаточно сложной системы, то ее функционирование может быть связано с возникновением нескольких типов событий и учетом большого числа случайных величин, распределенных по различным законам. Если же моделирование всех случайных факторов основано на использовании одного датчика, генерирующего одну «общую» последовательность случайных чисел, то с математической точки зрения их нельзя считать независимыми. В связи с этим для моделирования каждого случайного фактора стараются использовать отдельный генератор, или, по крайней мере, обеспечивать создание новой последовательности случайных чисел. Во многих специализированных языках и пакетах моделирования такая возможность предусмотрена. Это относится, в частности, и к пакету MATLAB.

2.3.2. УПРАВЛЕНИЕ МОДЕЛЬНЫМ ВРЕМЕНЕМ

Приступая к изучению механизмов управления модельным временем, уместно поговорить о том, какую роль вообще играет время в имитационном моделировании. При знакомстве с имитационным экспериментом мы отмечали, что он представляет собой наблюдение за поведением системы в течение некоторого проме-

жутка времени. Конечно, далеко не во всех статистических испытаниях фактор времени играет ведущую роль, а в некоторых и вообще может не рассматриваться. Вспомните, например, задачу о вычислении площади круга: полученный результат не зависел от того, сколь долго мы «бомбили» квадрат случайными точками (речь в данном случае не идет о количестве этих точек). Но значительно больше таких задач, в которых оценка эффективности моделируемой системы напрямую связана с временными характеристиками ее функционирования. К ним относятся упоминавшиеся уже задачи по оценке производительности, некоторые задачи по оценке надежности, качества распределения ресурсов, а также все задачи, связанные с исследованием эффективности процессов обслуживания. Характерной особенностью большинства практических задач является то, что скорость протекания рассматриваемых в них процессов значительно ниже скорости реализации модельного эксперимента. Например, если моделируется работа авторемонтной мастерской в течение недели, вряд ли кому-то придет в голову воспроизводить этот процесс в модели в таком же масштабе времени.

С другой стороны, даже те имитационные эксперименты, в которых временные параметры работы системы не учитываются, требуют для своей реализации определенных затрат времени работы компьютера.

В связи с этим при разработке практически любой имитационной модели и планировании проведения модельных экспериментов необходимо соотносить между собой три представления времени:

- реальное время, в котором происходит функционирование имитируемой системы;
- модельное (или, как его еще называют, системное) время, в масштабе которого организуется работа модели;
- машинное время, отражающее затраты времени ЭВМ на проведение имитации.

С помощью механизма модельного времени решаются следующие задачи:

1) отображается переход моделируемой системы из одного состояния в другое;

2) производится синхронизация работы компонент модели;

3) изменяется масштаб времени «жизни» (функционирования) исследуемой системы;

4) производится управление ходом модельного эксперимента.

5) моделируется квазипараллельная реализация событий в модели;

Приставка «квази» в данном случае отражает последовательный характер обработки событий (процессов) в ИМ, которые в реальной системе возникают (протекают) одновременно.

Необходимость решения последней задачи связана с тем, что в распоряжении исследователя находится, как правило, однопроцессорная вычислительная система, а модель может содержать значительно большее число одновременно работающих подсистем. Поэтому действительно параллельная (одновременная) реализация всех компонент модели невозможна. Даже если используется так называемая

распределенная модель, реализуемая на нескольких узлах вычислительной сети, совсем необязательно число узлов будет совпадать с числом одновременно работающих компонент модели. Немного забегая вперед, следует отметить, что реализация квазипараллельной работы компонент модели является достаточно сложной технической задачей. Некоторые возможные методы ее решения рассматриваются в следующем разделе, а пути полного избавления от этой проблемы с помощью пакета MATLAB — во второй части книги.

Ранее были названы два метода реализации механизма модельного времени — с постоянным шагом и по особым состояниям.

Выбор метода реализации механизма модельного времени зависит от назначения модели, ее сложности, характера исследуемых процессов, требуемой точности результатов и т. д.

При использовании метода ***постоянного шага*** отсчет системного времени ведется через фиксированные, выбранные исследователем интервалы времени. События в модели считаются наступившими в момент окончания этого интервала. Погрешность в измерении временных характеристик системы в этом случае зависит от величины шага моделирования Δt.

Метод постоянного шага предпочтительнее, если:

- события появляются регулярно, их распределение во времени достаточно равномерно;
- число событий велико и моменты их появления близки;
- невозможно заранее определить моменты появления событий.

Данный метод управления модельным временем достаточно просто реализовать в том случае, когда условия появления событий всех типов в модели можно представить как функцию времени.

Пусть, например, событие состоит в том, что летящий самолет пересекает некоторый воздушный рубеж, расстояние до которого равно R. Если самолет движется по прямой с постоянной скоростью V, то можно вычислять путь, пройденный самолетом, с интервалом времени Δt: $S=S+V \cdot \Delta t$. Соответственно событие считается наступившим, если выполняется условие $S > R$, а момент времени наступления события принимается равным $n \cdot \Delta t$, где n — номер шага моделирования, на котором условие стало истинным.

В общем виде алгоритм моделирования с постоянным шагом представлен на рис. 2.2 (t_M — текущее значение модельного времени, T_M — интервал моделирования).

С целью некоторого «оживления» приведенной выше схемы вернемся к примеру с самолетом. Введем в рассматриваемую ситуацию еще одно событие, которое состоит в том, что диспетчер, наблюдающий за самолетом, вводит данные о нем в некую систему управления. Процесс ввода заключается в наборе на клавиатуре определенной текстовой информации. Известна длина текста (в символах) и средняя скорость ввода одного символа. В ходе модельного эксперимента требуется оп-

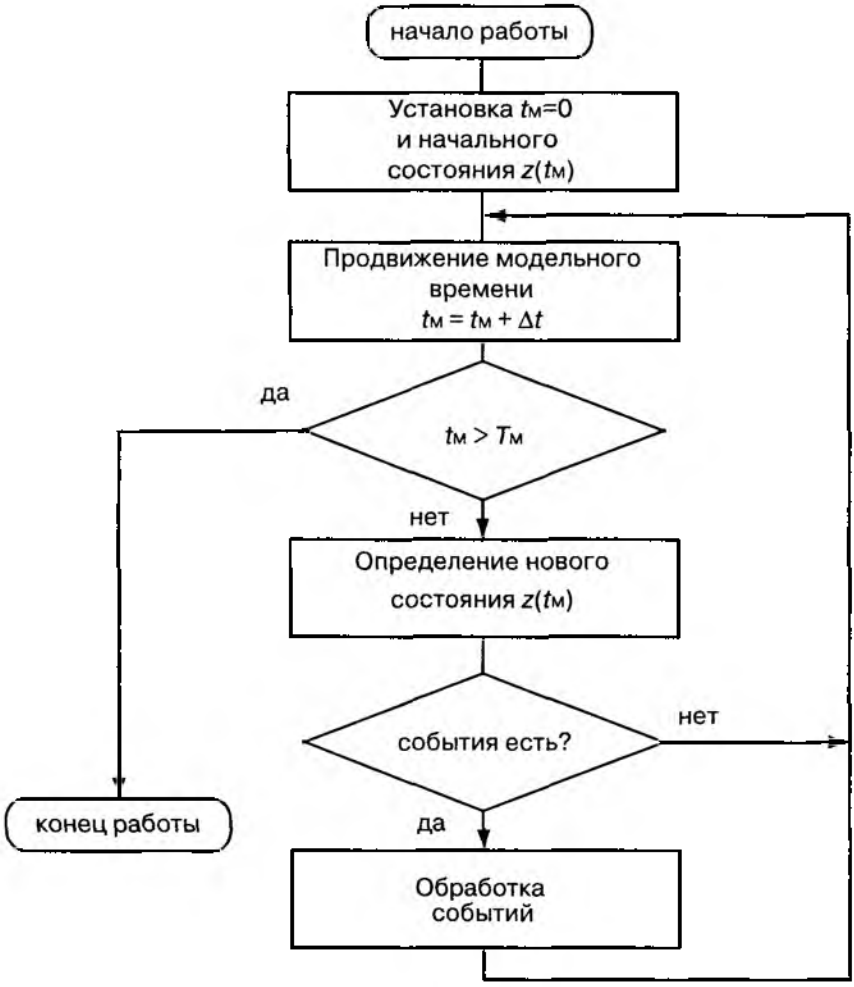


Рис. 2.2. Алгоритм моделирования с постоянным шагом

ределить, закончит ли диспетчер ввод текста до пересечения самолетом заданного рубежа. Алгоритм управления модельным временем для рассматриваемого примера показан на рис. 2.3.

Очевидно, что оба рассматриваемых процесса (полет самолета и ввод информации) должны быть «привязаны» к единой оси модельного времени. Вместе с тем каждый из них характеризуется различной скоростью, разной степенью дискретности и т. д. В такой ситуации для различных значений шага моделирования Δt эксперимент может дать разные результаты. Причем если шаг будет слишком большим, то результат, скорее всего, будет неверным: момент окончания ввода инфор-

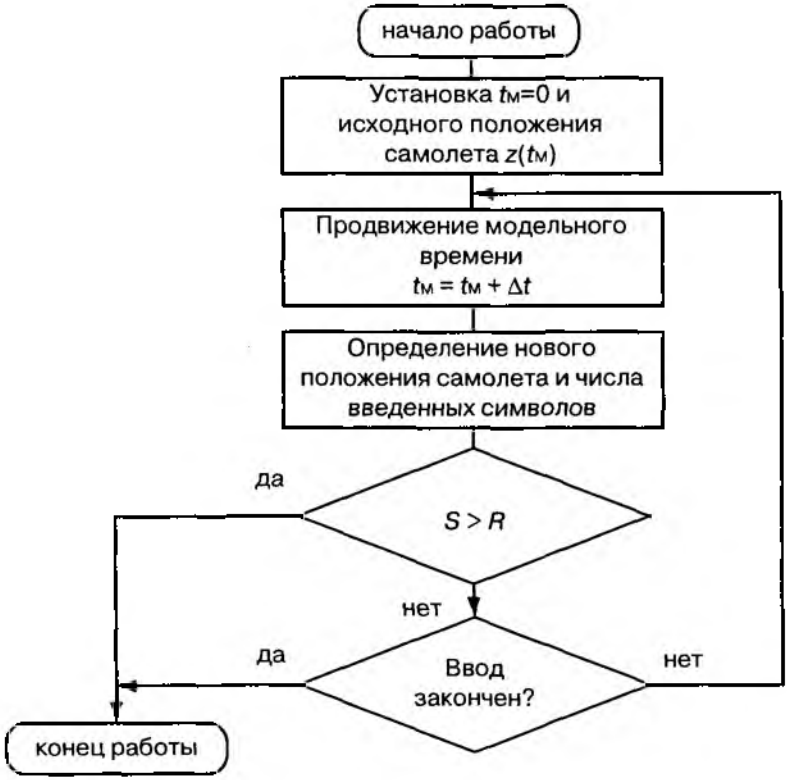


Рис. 2.3. Пример моделирования с постоянным шагом

мации будет всегда совпадать с моментом пересечения самолетом заданного рубежа. Такая ситуация показана на рис. 2.4.

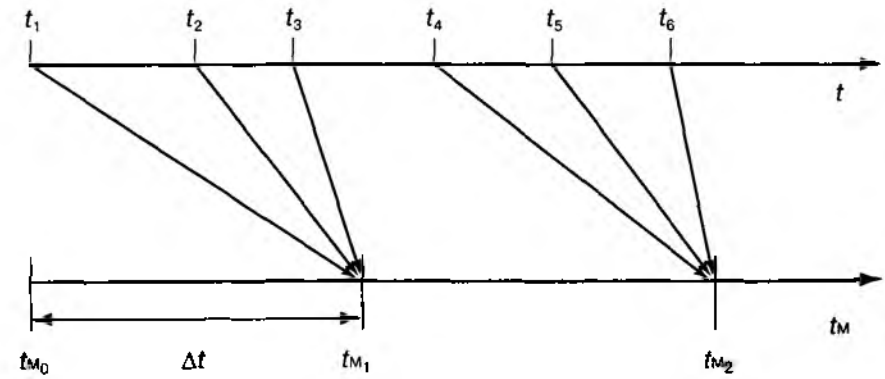


Рис.2.4. Пример привязки событий к оси модельного времени

Приведенный пример призван, кроме всего прочего, обратить внимание читателя на то, что выбор величины шага моделирования является нелегким и очень важным делом. Универсальной методики решения этой проблемы не существует, но во многих случаях можно использовать один из следующих подходов:

- принимать величину шага равной средней интенсивности возникновения событий различных типов;
- выбирать величину Δt равной среднему интервалу между наиболее частыми (или наиболее важными) событиями.

При моделировании ***по особым состояниям*** системное время каждый раз изменяется на величину, строго соответствующую интервалу времени до момента

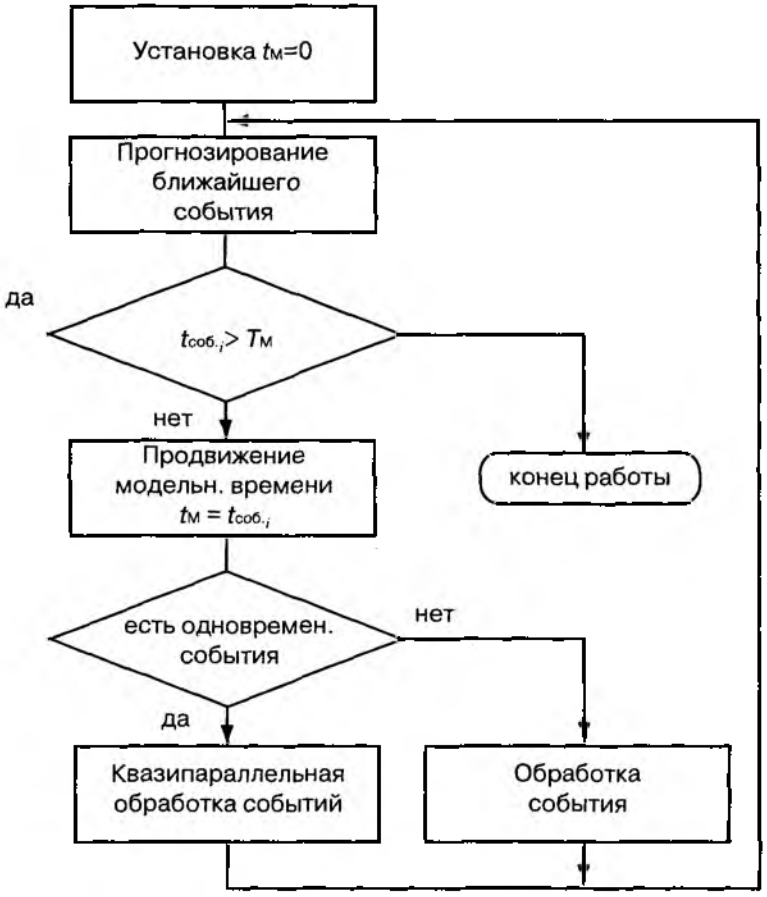


Рис. 2.5. Алгоритм моделирования по особым состояниям

наступления очередного события. В этом случае события обрабатываются в порядке их наступления, а одновременно наступившими считаются только те, которые являются одновременными в действительности.

Метод моделирования по особым состояниям сложнее в реализации, так как для него требуется разработка специальной процедуры планирования событий (так называемого календаря событий).

Моделирование по особым состояниям целесообразно использовать, если:

- события распределяются во времени неравномерно или интервалы между ними велики;
- предъявляются повышенные требования к точности определения взаимного положения событий во времени;
- необходимо реализовать квазипараллельную обработку одновременных событий.

Дополнительное достоинство метода заключается в том, что он позволяет экономить машинное время, особенно при моделировании систем периодического действия, в которых события длительное время могут не наступать.

Обобщенная схема алгоритма моделирования по особым состояниям представлена на рис. 2.5 ($t_{соб_i}$ — прогнозируемый момент наступления i-го события.

Чтобы читатель смог «почувствовать разницу» в использовании двух методов управления модельным временем, вернемся еще раз к примеру с летящим самолетом и сидящим диспетчером.

На этот раз перед разработчиком модели встает иная проблема: что понимать под «особыми состояниями», которые должны влиять на изменение модельного времени? На практике обычно вместо состояний рассматривают события, определяющие смену состояний моделируемого процесса. Для процесса ввода информации диспетчером такой переход выполняется достаточно просто: событие — это ввод очередного символа; другими словами, ввод очередного символа «продвигает» модельное время на соответствующий интервал. Все так просто потому, что процесс ввода является дискретным. А что делать с непрерывно летящим самолетом? Здесь возможны два варианта: либо увязать расчет нового положения самолета с моментом ввода очередного символа, либо изменить в модели представление полета с непрерывного на дискретное (например, рассматривать перемещение не самого самолета в воздухе, а его «образа» на экране индикатора).

Алгоритм работы модели для первого варианта приведен на рис. 2.6.

А как быть в том случае, когда и полет самолета, и работа диспетчера подвержены влиянию случайных факторов? Мы ответим на этот вопрос для не очень сложного, но весьма распространенного в практике моделирования варианта, когда рассматриваемые процессы могут быть описаны с помощью случайных величин, распределенных по заданному закону.

Для самолета такой величиной будет служить скорость полета, а для диспетчера — скорость ввода символов. В этом случае изменение модельного времени и расчет нового положения самолета также можно увязать с вводом очередного симво-

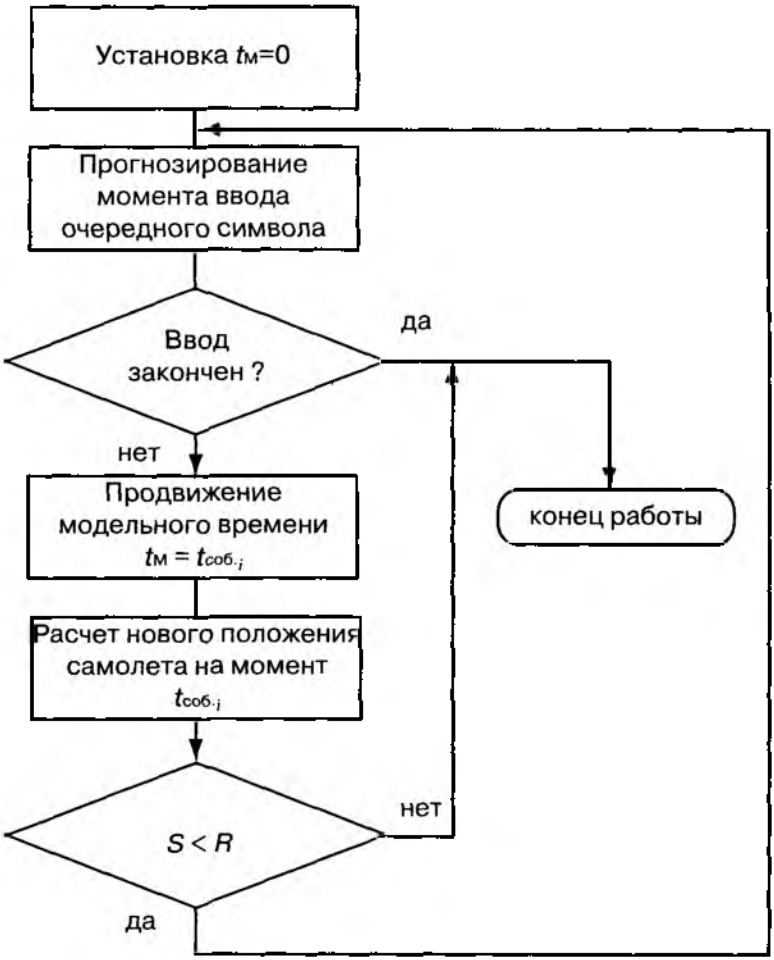


Рис. 2.6. Пример работы модели по особым состояниям

ла. Логика работы такой модели по-прежнему будет соответствовать алгоритму, изображенному на рис. 2.6. Отличие заключается лишь в том, что прогноз времени ввода очередного символа выполняется на основе функции распределения соответствующей случайной величины. Так, если скорость ввода символов подчиняется нормальному закону с параметрами m и d, то очередное i-е значение модельного времени $t_{м_i}$ определяется следующим образом:

$$t_{м_i} = t_{м_{i-1}} + \mathrm{norm}(m, d).$$

В этом выражении слагаемое $\mathrm{norm}(m, d)$ означает обращение к генератору случайных чисел, распределенных по нормальному закону.

Подведем итоги изложенному в этом разделе.

• Выбор механизма изменения модельного времени определяет и технологию реализации имитационной модели.

• На выбор метода моделирования влияет целый ряд факторов, однако определяющим является тип моделируемой системы: для дискретных систем, события в которых распределены во времени неравномерно, более удобным является изменение модельного времени по особым состояниям.

Если в модели должны быть представлены компоненты реальной системы, работа которых измеряется в разных единицах времени, то они должны быть предварительно приведены к единому масштабу.

2.4. МОДЕЛИРОВАНИЕ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ

Практически любая более или менее сложная система имеет в своем составе компоненты, работающие одновременно, или, как принято говорить на языке техники, параллельно. Наверняка читатель сам может привести массу примеров таких систем из близкой ему области деятельности. Мы же напомним только один, взятый из монолога М. Жванецкого, посвященного работе ликеро-водочного завода. Судя по сюжету монолога, все цеха этого предприятия работали абсолютно параллельно и независимо друг от друга.

Итак, если в составе системы имеются компоненты (подсистемы), выполняющие свои функции одновременно, то можно утверждать, что в такой системе существуют параллельные процессы. Параллельно работающие подсистемы могут взаимодействовать самым различным образом, либо вообще работать независимо друг от друга. Способ взаимодействия подсистем определяет вид параллельных процессов, протекающих в системе. В свою очередь, вид моделируемых процессов влияет на выбор метода их имитации.

2.4.1.ВИДЫ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ В СЛОЖНЫХ СИСТЕМАХ

Асинхронный параллельный процесс — такой процесс, состояние которого не зависит от состояния другого параллельного процесса (ПП).

Пример асинхронных ПП, протекающих в рамках одной системы: подготовка и проведение рекламной кампании фирмой и работа сборочного конвейера; пример из области вычислительной техники: выполнение вычислений процессором и вывод информации на печать.

Синхронный ПП — такой процесс, состояние которого зависит от состояния взаимодействующих с ним ПП.

Пример синхронного ПП: работа торговой организации и доставка товара со склада (нет товара — нет торговли).

Один и тот же процесс может быть синхронным по отношению к одному из активных ПП и асинхронным по отношению к другому. Так, при работе вычислительной сети по технологии «клиент-сервер» каждый из узлов сети синхронизирует свою работу с работой сервера, но не зависит от работы других узлов.

Подчиненный ПП — создается и управляется другим процессом (более высокого уровня).

Весьма характерным примером таких процессов является ведение боевых действий подчиненными подразделениями.

Независимый ПП — не является подчиненным ни для одного из процессов.

Скажем, после запуска неуправляемой зенитной ракеты ее полет можно рассматривать как независимый процесс, одновременно с которым самолет ведет боевые действия другими средствами.

Способ организации параллельных процессов в системе зависит от физической сущности этой системы.

Остановимся несколько подробнее на особенностях реализации параллельных процессов в вычислительных системах (ВС). Это обусловлено следующей причиной.

Разработка и использование любой ИМ предполагает ее программную реализацию и исследование с применением ВС. Поэтому для реализации моделей, имитирующих параллельные процессы, в некоторых случаях применимы механизмы, используемые при выполнении параллельных вычислений.

Вместе с тем, реализация параллельных процессов в ВС имеет свои особенности:

- на уровне задач вычислительные процессы могут быть истинно параллельными только в многопроцессорных ВС или вычислительных сетях;
- многие ПП используют одни и те же ресурсы, поэтому даже асинхронные ПП в пределах одной ВС вынуждены согласовывать свои действия при обращении к общим ресурсам;
- в ВС дополнительно используется еще два вида ПП: родительский и дочерний ПП; особенность их состоит в том, что процесс-родитель не может быть завершен, пока не завершатся все его дочерние процессы.

В силу перечисленных особенностей для организации взаимодействия параллельных процессов в ВС используются три основных подхода:

- на основе «взаимного исключения»;
- на основе синхронизации посредством сигналов;
- на основе обмена информацией (сообщениями).

«Взаимное исключение» предполагает запрет доступа к общим ресурсам (общим данным) для всех ПП, кроме одного, на время его работы с этими ресурсами (данными).

Синхронизация подразумевает обмен сигналами между двумя или более процессами по установленному протоколу. Такой «сигнал» рассматривается как

некоторое событие, вызывающее у получившего его процесса соответствующие действия.

Часто возникает необходимость передавать от одного ПП другому более подробную информацию, чем просто «сигнал-событие». В этом случае прорцессы согласуют свою работу на основе обмена сообщениями.

Перечисленные механизмы реализуются в ВС на двух уровнях — системном и прикладном.

Механизм взаимодействия между ПП на системном уровне определяется еще на этапе разработки ВС и реализуется в основном средствами операционной системы (частично — с использованием аппаратных средств).

На прикладном уровне взаимодействие между ПП реализуется программистом средствами языка, на котором разрабатывается программное обеспечение.

Наибольшими возможностями в этом отношении обладают так называемые языки реального времени (ЯРВ) и языки моделирования.

Языки реального времени — это языки, предназначенные для создания программного обеспечения, работающего в реальном масштабе времени, например для разработки различных автоматизированных систем управления (предприятием, воздушным движением и т. д.). К ним, в частности, относятся: язык *Ада*, язык *Модула* и практически единственный отечественный язык реального времени — *Эль-76* (использовавшийся в многопроцессорных вычислительных комплексах семейства «Эльбрус»).

2.4.2. МЕТОДЫ ОПИСАНИЯ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ В СИСТЕМАХ И ЯЗЫКАХ МОДЕЛИРОВАНИЯ

Языки моделирования по сравнению с языками реального времени требуют от разработчика значительно менее высокого уровня подготовки в области программирования, что обусловлено двумя обстоятельствами:

- во-первых, средства моделирования изначально ориентированы на квазипараллельную обработку параллельных процессов;
- во-вторых, механизмы реализации ПП относятся, как правило, к внутренней организации системы (языка) моделирования и их работа скрыта от программиста.

В практике имитационного моделирования одинаково широко используются как процессно-ориентированные языки (системы) моделирования, например *SIMULA*, так и языки, ориентированные на обработку транзактов (например, язык *GPSS*). В тех и других используются аналогичные методы реализации квазипараллелизма, основанные на ведении списков событий. В процессно-ориентированных системах используются списки событий следования, а в транзактных системах — списки событий изменения состояний.

Современные языки и системы моделирования, ориентированные на использование в среде многозадачных операционных систем типа Windows, частично ис-

пользуют их механизмы управления процессами, что делает их применение еще более эффективным. В пакете MATLAB также имеется собственный язык моделирования, и к нему в полной мере можно отнести сказанное выше. Тем не менее во многих случаях оказывается полезным знание общего механизма реализации ПП в языках моделирования.

Рассмотрим его применительно к моделированию на основе тракзактов.

В этом случае под событием понимается любое перемещение транзакта по системе, а также изменение его состояния (обслуживается, заблокирован и т. д.).

Событие, связанное с данным транзактом, может храниться в одном из следующих списков.

Список текущих событий. В этом списке находятся события, время наступления которых меньше или равно текущему модельному времени. События с «меньшим» временем связаны с перемещением тех транзактов, которые должны были начать двигаться, но были заблокированы.

Список будущих событий. Этот список содержит события, время наступления которых больше текущего модельного времени, то есть события, которые должны произойти в будущем (условия наступления которых уже определены — например, известно, что транзакт будет обслуживаться некоторым устройством 10 единиц времени).

Список прерываний. Данный список содержит события, связанные с возобновлением обработки прерванных транзактов. События из этого списка выбираются в том случае, если сняты условия прерывания.

В списке текущих событий транзакты расположены в порядке убывания приоритета соответствующих событий; при равных приоритетах — в порядке поступления в список.

Каждое событие (транзакт) в списке текущих событий может находиться либо в активном состоянии, либо в состоянии задержки. Если событие активно, то соответствующий транзакт может быть продвинут по системе; если продвижение невозможно (например, из-за занятости устройства), то событие (и транзакт) переводится в состояние задержки.

Как только завершается обработка (продвижение) очередного активного транзакта, просматривается список задержанных транзактов, и ряд из них переводится в активное состояние. Процедура повторяется до тех пор, пока в списке текущих событий не будут обработаны все активные события. После этого просматривается список будущих событий. Модельному времени присваивается значение, равное времени наступления ближайшего из этих событий. Данное событие заносится в список текущих событий. Затем просматриваются остальные события списка. Те из них, время которых равно текущему модельному времени, также переписываются в список текущих событий. Просмотр заканчивается, когда в списке остаются события, времена которых больше текущего модельного времени.

В качестве иллюстрации к изложеному рассмотрим небольшой пример.

Пусть в систему поступают транзакты трех типов, каждый из которых обслуживается отдельным устройством. Известны законы поступления транзактов в систему и длительность их обслуживания. Таким образом, в системе существуют три параллельных независимых процесса (*P*1, *P*2, *P*3).

Временная диаграмма работы системы при обслуживании одного транзакта каждого типа показана на рис.2.7.

На рисунке события, относящиеся к процессу *P*1, обозначены как $C1_i$, относящиеся к *P*2 и к *P*3 — соответственно как $C2_i$ и $C3_i$. Моменты времени $t_{вх}$ и $t_{вых}$ соответствуют началу и окончанию обслуживания транзакта.

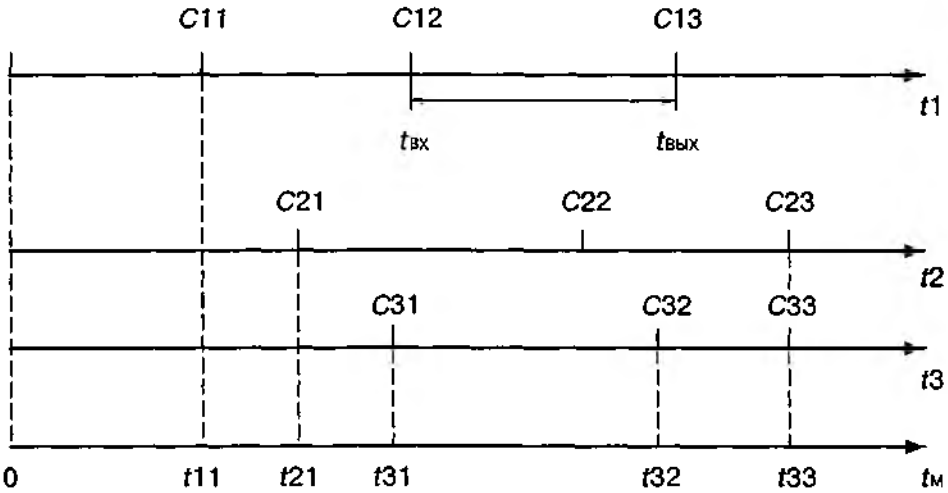


Рис. 2.7. Временная диаграмма параллельных процессов

Для каждого процесса строится своя цепь событий, однако списки событий являются общими для всей модели. Формирование списков начинается с заполнения списка будущих событий. Как было отмечено выше, в этот список помещаются события, время наступления которых превышает текущее значение модельного времени. Очевидно, что на момент заполнения списка время наступления прогнозируемых событий должно быть известно. На первом шаге $t_м$=0, и в список будущих событий заносятся события *C*11, *C*21, *C*31. Затем событие с наименьшим временем наступления — *C*11 — переносится в список текущих событий; если одновременных с ним событий нет, то оно обрабатывается и исключается из списка текущих событий. После этого вновь корректируется список будущих событий и т.д., пока не истечет заданный интервал моделирования.

Динамика изменения списков текущих и будущих событий для рассмотренного примера отражена в приведенной ниже таблице (табл. 2.2)

Таблица 2.2

Изменение списков событий

t	Список текущих событий	Список будущих событий
0	0	$C11$ $C21$ $C3.$
$t11$	$C11$	$C21$ $C31$ $C12$
$t21$	$C21$	$C31$ $C12$ $C22$
t31	$C31$	$C12$ $C22$ $C32$
$t12$	$C12$	$C22$ $C32$ $C13$
$t22$	$C22$	$C32$ $C13$ $C23$
$t32$	$C32$	$C13$ $C23$ $C33$
$t13$	$C13$	$C23$ $C33$
$t23$	$C23$ $C33$	

Многие авторы книг по имитационному моделированию, ставших уже классикой, считают, что знание механизма ведения списков событий просто необходимо разработчику модели; умение проследить в динамике цепь происходящих в модели событий во-первых, повышает уверенность создателя модели в том, что она работает правильно и, во вторых, существенно облегчает процесс отладки и модификации модели.

2.4.3. ПРИМЕНЕНИЕ СЕТЕВЫХ МОДЕЛЕЙ ДЛЯ ОПИСАНИЯ ПАРАЛЛЕЛЬНЫХ ПРОЦЕССОВ

Сведения, приведенные в предыдущем разделе, кроме всего прочего должны были убедить читателя в том, что разработчику имитационной модели необходимо иметь хотя бы минимальные знания и навыки в области программирования. И хотя вторая часть этой книги в основном посвящена тому, чтобы разве-

ять это впечатление, тем не менее существует проблема, которую мы не сможем обойти. Речь идет о том, что этапу программной реализации модели (т. е. ее описанию на одном из языков программирования) должен предшествовать так называемый этап алгоритмизации. Другими словами, прежде чем превратить имитационную модель в работающую компьютерную программу, ее создатель должен воспользоваться каким-то менее формальным и более наглядным средством описания логики работы будущей программы. Разумеется, это требование не является обязательным для всех разработчиков и для всех создаваемых моделей: при наличии достаточного опыта программа не очень сложной модели может быть написана сразу. Однако практика показывает, что человеческие возможности не безграничны, и при моделировании более сложных систем даже опытные разработчики бывают вынуждены немного «притормозить» на этапе алгоритмизации. Для описания логики работы модели могут быть использованы различные средства: либо русский язык (устный или письменный), либо традиционные схемы алгоритмов (такие, как на рисунках 1.3, 1.4), либо какие-то другие «подручные» средства. Первые два варианта являются, как правило, наиболее знакомыми и наиболее часто используемыми. Однако если вы попробуете описать в виде схемы алгоритма модель даже такой простой системы, которая использовалась в предыдущем примере, то это окажется напрасной тратой времени и сил. Прежде всего потому, что такие схемы совершенно не приспособлены для описания параллельных процессов.

Именно поэтому представляется необходимым познакомить читателя с одним из наиболее элегантных и весьма распространенных средств описания параллельных процессов — ***сетями Петри***. Им посвящено достаточно большое число изданий, поэтому мы ограничимся изложением только тех основных сведений, которые необходимы с точки зрения реализации технологии имитационного моделирования параллельных процессов.

Одно из основных достоинств аппарата сетей Петри заключается в том, что они могут быть представлены как в графической форме (что обеспечивает наглядность), так и в аналитической (что позволяет автоматизировать процесс их анализа).

При графической интерпретации сеть Петри представляет собой граф особого вида, состоящий из вершин двух типов — *позиций* и *переходов*, соединенных ориентированными дугами, причем каждая дуга может связывать лишь разнотипные вершины (позицию с переходом или переход с позицией). Вершины-позиции обозначаются кружками, вершины-переходы — черточками. С содержательной точки зрения, переходы соответствуют событиям, присущим исследуемой системе, а позиции — условиям их возникновения. Таким образом, совокупность переходов, позиций и дуг позволяет описать причинно-следственные связи, присущие системе, но в статике. Чтобы сеть Петри «ожила», вводят еще один вид объектов сети — так называемые *фишки*, или метки позиций. Переход считается активным (событие может произойти), если в каждой его входной позиции есть хотя бы одна фишка.

Расположение фишек в позициях сети называется ***разметкой сети*** (пример перемещения фишек по сети приведен на рис.2.8).

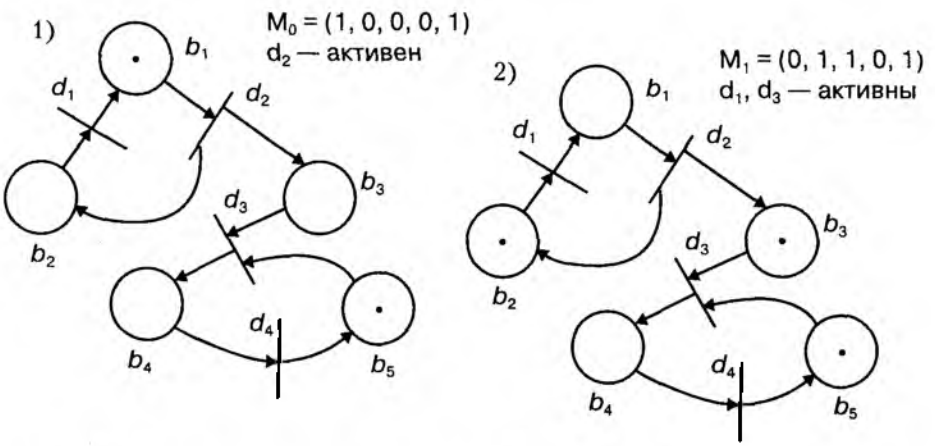


Рис. 2.8. Пример изменения разметки сети Петри при срабатывании переходов

В аналитической форме сеть Петри может быть представлена следующим образом:

$$P = (B, D, I, O, M),$$

где $B = \{b_i\}$ — конечное непустое множество позиций;

$D = \{d_i\}$ — конечное непустое множество переходов;

$I : BxD \rightarrow 0,1$ — входная функция (прямая функция инцидентности), которая для каждого перехода задает множество его входных позиций;

$O : DxB \rightarrow 0,1$ — выходная функция (обратная функция инцидентности), которая для каждого перехода задает множество его входных позиций;

M — функция разметки сети, $M : B \rightarrow 0, 1, 2,...$ — ставит каждой позиции сети в соответствие неотрицательное целое число.

С учетом введенных обозначений необходимое условие срабатывания перехода dj может быть записано следующим образом:

$\forall\, b_i \in I(d_j)\{ M(b_i) \geq 1\}$ (для всех входных позиций разметка должна быть ≥1).

Срабатывание перехода d_j изменяет разметку сети $M(B)$ на разметку $M'(B)$ по следующему правилу:

$$M'(B) = M(B) - I(d_j) + O(d_j),$$

то есть переход d_j изымает по одной метке из каждой своей входной позиции и добавляет по одной метке в каждую из выходных позиций. Смену разметки обозначают так:

$$M_0 \vdash^{d_j} M'.$$

Входная и выходная функции сети Петри (I и O) позволяют описать любую сеть с помощью двух матриц размера $m \times n$ (матриц входных и выходных позиций), имеющих следующую структуру:

	d_1	d_2	...	d_j	...	d_n
b_1	0	1	...	0	...	0
b_2	1	1	...	0	...	1
⋮				⋮		
b_j	0	1	...	0	...	1
⋮				⋮		
b_m	1	0	...	1	...	0

Основные направления анализа сети Петри следующие:

1. *Проблема достижимости:* в сети Петри с начальной разметкой M_0 требуется определить, достижима ли принципиально некоторая разметка M' из M_0.

С точки зрения исследования моделируемой системы, эта проблема интерпретируется как проблема достижимости (реализуемости) некоторого состояния системы.

2. *Свойство живости.* Под живостью перехода понимают возможность его срабатывания в данной сети при начальной разметке M_0.

Анализ модели на свойство живости позволяет выявить невозможные состояния в моделируемой системе (например, неисполняемые ветви в программе).

3. *Безопасность сети.* Безопасной является такая сеть Петри, в которой ни при каких условиях не может появиться более одной метки в каждой из позиций.

Для исследуемой системы это означает возможность функционирования ее в стационарном режиме. На основе анализа данного свойства могут быть определены требования к буферным накопителям в системе.

Итак, достоинства сетей Петри заключаются в том, что они:

1) позволяют моделировать ПП всех возможных типов с учетом вероятных конфликтов между ними;

2) обладают наглядностью и обеспечивают возможность автоматизированного анализа;

3) позволяют переходить от одного уровня детализации описания системы к другому (за счет раскрытия/закрытия переходов).

Вместе с тем, сети Петри имеют ряд недостатков, ограничивающих их возможности. Основной из них — время срабатывания перехода считается равным 0, что не позволяет исследовать с помощью сетей Петри временные характеристики моделируемых систем.

В результате развития аппарата сетей Петри был разработан ряд расширений. Одно из наиболее мощных — так называемые *E*-сети (evaluation — «вычисления», «оценка») — «оценочные сети».

В отличие от сетей Петри, в *E*-сетях:

1) имеются несколько типов вершин-позиций: простые позиции, позиции-очереди, разрешающие позиции;

2) фишки (метки) могут снабжаться набором признаков (атрибутов);

3) с каждым переходом может быть связана ненулевая задержка и функция преобразования атрибутов фишек.

4) введены дополнительные виды вершин-переходов.

5) в любую позицию может входить не более одной дуги и выходить также не более одной.

В связи с этим любой переход может быть описан тройкой параметров:

$$d_j = (S, t(d_j), \rho(d_j)),$$

где S — тип перехода,

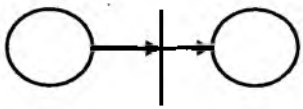
$t(d_j)$, — функция задержки,

$\rho(d_j)$ — функция преобразования атрибутов

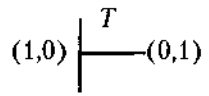
Базовые переходы *E*-сети описаны ниже.

***T*-переход** («исполнение», «простой переход»).

Его графическое представление аналогично представлению вершины-перехода сети Петри:



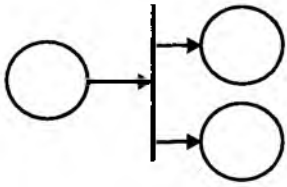
Переход срабатывает при наличии метки во входной позиции и отсутствии ее в выходной позиции. Формально это можно записать так:



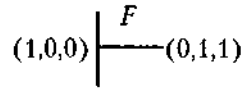
T-переход позволяет отразить в модели занятость некоторого устройства (подсистемы) в течение некоторого времени, определяемого параметром $t(d)$.

***F*-переход** («разветвление»).

Графическое представление:



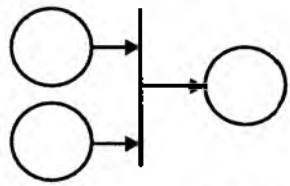
Срабатывает при тех же условиях, что и *T*-переход:



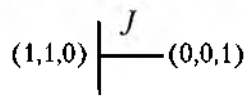
С содержательной точки зрения *F*-переход отображает разветвление потока информации (транзактов) в системе.

***J*-переход** («объединение»).

Графическое представление:



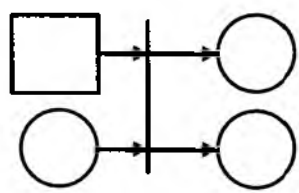
Переход срабатывает при наличии меток в обеих входных позициях:



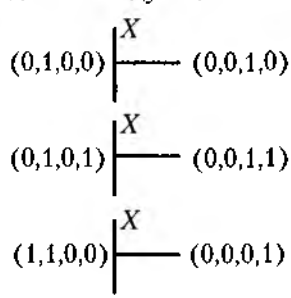
Он моделирует объединение потоков или наличие нескольких условий, определяющих некоторое событие.

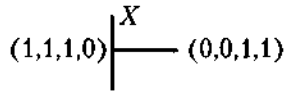
***X*-переход** («переключатель»)

По сравнению с тремя предыдущими типами переходов он содержит дополнительную управляющую («разрешающую») позицию, которая графически обозначается обычно либо квадратиком, либо шестиугольником:



Логика его срабатывания задается следующими соотношениями:

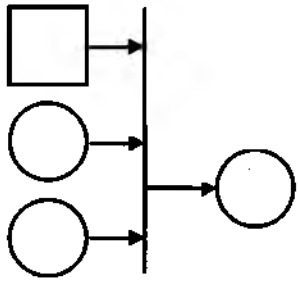




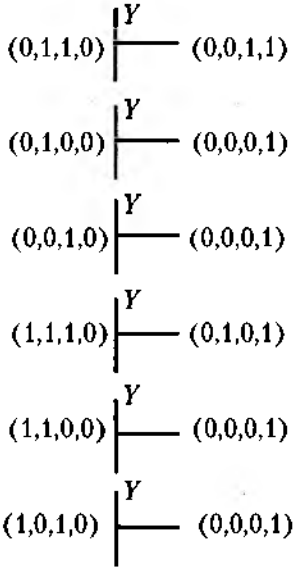
X-переход изменяет направление потока информации (транзактов). В общем случае разрешающая процедура может быть сколь угодно сложной, но сущность ее работы заключается в проверке выполнения условий разветвления потока (с точки зрения программиста разрешающая позиция аналогична условному оператору типа *if*).

***Y*-переход** («выбор», «приоритетный выбор»).

Этот переход также содержит разрешающую позицию:



Логика срабатывания Y-перехода:



Y-переход отражает приоритетность одних потоков информации (транзактов) по сравнению с другими. При этом разрешающая процедура может быть определена различным образом: как операция сравнения фиксированных приоритетов

меток; как функция от атрибутов меток (например, чем меньше время обслуживания, тем выше приоритет).

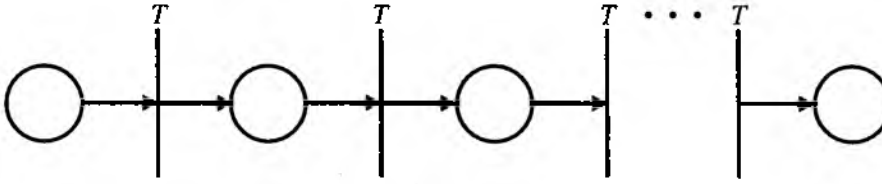
В некотором смысле он работает аналогично оператору выбора типа *case*.

В Е-сети все переходы обладают свойством безопасности. Это означает, что в выходных позициях (которые в свою очередь могут быть входными для следующего перехода) никогда не может быть более одной метки.

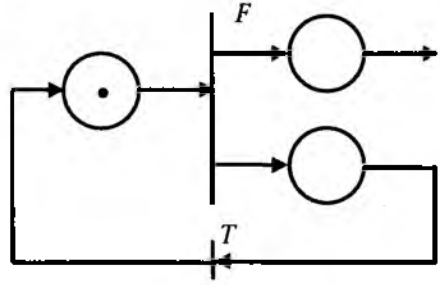
Вместе с тем, в Е-сетях существуют понятия ***макроперехода*** и ***макропозиции***, которые позволяют отображать в модели процессы накопления обслуживаемых транзактов в тех или иных узлах системы.

Рассмотрим некоторые из них.

Макропозиция ***очередь*** представляет собой линейную композицию вершин-позиций и *Т*-переходов; количество вершин-позиций определяет «емкость» очереди:

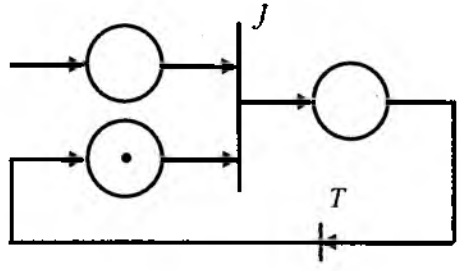


Макропозиция ***генератор*** позволяет представлять в сети источник меток (транзактов):



Если необходимо задать закон формирования меток, то «генератор» может быть дополнен разрешающей позицией.

Поскольку в *Е*-сети нельзя «накапливать» метки, то вводится макропозиция ***поглощения***:



В целях повышения компактности и наглядности *E*-сети для обозначения макропозиций используют специальные символы:


— «очередь»;

— «генератор»;

— «аккумулятор».

Аналогичным образом путем композиции N однотипных переходов могут быть получены макропереходы всех типов: X_N, Y_N, J_N.

Рассмотренные особенности *E*-сетей существенно расширяют их возможности для моделирования дискретных систем вообще и параллельных процессов в частности. Ниже приведен пример описания в виде E-сети мультипрограммной вычислительной системы. Обработка поступающих заданий организована в ней по принципу квантования времени: каждому заданию выделяется равный отрезок (квант) процессорного времени; если задание выполнено, то оно покидает систему, если же времени оказалось недостаточно, то задание встает в очередь и ждет повторного выделения кванта времени.

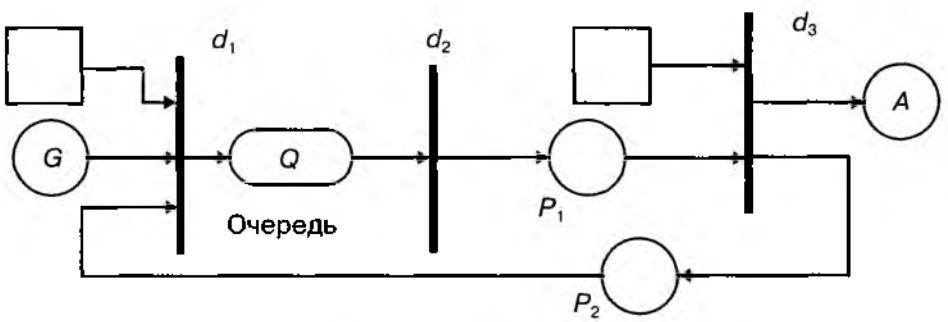


Рис. 2.9. Пример описания вычислительной системы в виде E-сети

На рисунке переходы, соответствующие определенным событиям в системе (d_i), имеют следующие обозначения.:

d_1 — постановка задания в очередь;

d_2 — выполнение задания в течение одного кванта времени;

d_3 — анализ степени завершенности задания.

Помимо очевидных достоинств E-сетей, проявленное к ним в первой части книги внимание объясняется еще и тем, что технология моделирования систем в виде E-сетей весьма эффективно реализуется с помощью инструмента SIMULINK, входящего в состав пакета MATLAB. Соответствующие вопросы рассмотрены во второй части книги.

2.5. ПЛАНИРОВАНИЕ МОДЕЛЬНЫХ ЭКСПЕРИМЕНТОВ

В качестве вступления хотелось бы привести небольшой пример, призванный смягчить некоторую сухость, появившуюся в изложении.

Предположим, три юных натуралиста решили изучить повадки некой птички, обитающей в средней полосе (какого-нибудь зяблика). Первый стал наблюдать за птичкой по утрам, второй — когда придется, а третий вообще отложил это занятие до осени. Как вы думаете, кто из них получит наиболее полное и достоверное жизнеописание зяблика? Если вы знакомы с орнитологией (наукой о птицах) на уровне автора, вряд ли вам удастся правильно ответить на этот вопрос. Почему? Потому, что мы слишком мало знаем о птицах вообще и о зябликах в частности. Может быть, эта птица отличается удивительным постоянством и независимо от сезона и времени суток занимается одним и тем же (например, высиживает птенцов). А может, взбалмошней ее в лесу никого нет: сегодня она в средней полосе, назавтра улетела в теплые края, а послезавтра оказалась еще где-нибудь. И каждый день придумывает себе новое занятие. Так в какое время за ней нужно наблюдать и насколько долго, чтобы получить реальную картину?

На подобные вопросы приходится отвечать не только юным натуралистам, но и тому, кто занимается имитационным моделированием.

Во-первых, исследователь и на этапе планирования эксперимента должен помнить, к какому классу относится моделируемая система (статическая или динамическая, детерминированная или стохастическая и т. д.).

Во-вторых, он должен определить, какой режим работы системы его интересует: стационарный (установившийся) или нестационарный.

В-третьих, необходимо знать, в течение какого промежутка времени следует наблюдать за поведением (функционированием) системы.

И, наконец, в-четвертых, хорошо было бы знать, какой объем испытаний (т. е. повторных экспериментов) сможет обеспечить требуемую точность оценок (в статистическом смысле) исследуемых характеристик системы.

Разумеется, можно пойти по такому пути: не особенно задумываясь над перечисленными вопросами, взять от модели все «по максимуму» — исследовать работу системы во всех режимах, для всех возможных сочетаний внешних и внутренних параметров и повторять каждый эксперимент по сотне раз. Однако польза от такого моделирования невелика, поскольку полученные данные будет очень сложно обработать и проанализировать, а еще труднее принять с их помощью какое-либо конкретное решение. Да и затраты времени на моделирование, даже с учетом быстродействия современных компьютеров, окажутся чрезмерными.

Таким образом, планирование модельных экспериментов преследует две основные цели:

- сокращение общего объема испытаний при соблюдении требований к достоверности и точности их результатов;
- повышение информативности каждого из экспериментов в отдельности.

Поиск плана эксперимента производится в так называемом факторном пространстве.

Факторное пространство — это множество внешних и внутренних параметров модели, значения которых исследователь может контролировать в ходе подготовки и проведения модельного эксперимента.

Поскольку факторы могут носить как количественный, так и качественный характер (например, отражать некоторую стратегию управления), значения факторов обычно называют *уровнями*. Если при проведении эксперимента исследователь может изменять уровни факторов, эксперимент называется *активным*, в противном случае — *пассивным*.

Введем еще несколько терминов, используемых в теории планирования эксперимента.

Каждый из факторов имеет верхний и нижний уровни, расположенные симметрично относительно некоторого нулевого уровня. Точка в факторном пространстве, соответствующая нулевым уровням всех факторов, называется ***центром плана***.

Интервалом варьирования фактора называется некоторое число J, прибавление которого к нулевому уровню дает верхний уровень, а вычитание — нижний.

Как правило, план эксперимента строится относительно одного (основного) выходного скалярного параметра Y, который называется ***наблюдаемой переменной***. Если моделирование используется как инструмент принятия решения, то в роли наблюдаемой переменной выступает показатель эффективности.

При этом предполагается, что значение наблюдаемой переменной, полученное в ходе эксперимента, складывается из двух составляющих:

$$y = f(x) + e(x),$$

где $f(x)$ — функция отклика (неслучайная функция факторов);

$e(x)$ — ошибка эксперимента (случайная величина);

x — точка в факторном пространстве (определенное сочетание уровней факторов);

Очевидно, что y является случайной переменной, так как зависит от случайной величины $e(x)$.

Дисперсия Dy наблюдаемой переменной, которая характеризует точность измерений, равна дисперсии ошибки опыта: $Dy = De$.

Dy называют дисперсией воспроизводимости эксперимента. Она характеризует качество эксперимента.

Эксперимент называется идеальным при $Dy = 0$.

Существует два основных варианта постановки задачи планирования имитационного эксперимента:

1. Из всех допустимых выбрать такой план, который позволил бы получить наиболее достоверное значение функции отклика $f(x)$ при фиксированном числе опытов.

2. Выбрать такой допустимый план, при котором статистическая оценка функции отклика может быть получена с заданной точностью при минимальном объеме испытаний.

Решение задачи планирования в первой постановке называется ***стратегическим планированием эксперимента***, во второй — ***тактическим планированием***.

2.5.1. СТРАТЕГИЧЕСКОЕ ПЛАНИРОВАНИЕ ИМИТАЦИОННОГО ЭКСПЕРИМЕНТА

Итак, цель методов стратегического планирования имитационных экспериментов — получение максимального объема информации об исследуемой системе в каждом эксперименте (наблюдении). Другими словами, стратегическое планирование позволяет ответить на вопрос, при каком сочетании уровней внешних и внутренних факторов может быть получена наиболее полная и достоверная информация о поведении системы (или птички зяблика).

При стратегическом планировании эксперимента должны быть решены две основные задачи:

1) идентификация факторов;

2) выбор уровней факторов.

Под ***идентификацией факторов*** понимается их ранжирование по степени влияния на значение наблюдаемой переменной (показателя эффективности).

По итогам идентификации целесообразно разделить все факторы на две группы — первичные и вторичные. ***Первичные*** — это те факторы, в исследовании влияния которых экспериментатор заинтересован непосредственно. ***Вторичные*** — факторы, которые не являются предметом исследования, но влиянием которых нельзя пренебречь.

Выбор уровней факторов производится с учетом двух противоречивых требований:

- уровни фактора должны перекрывать (заполнять) весь возможный диапазон его изменения;
- общее количество уровней по всем факторам не должно приводить к чрезмерному объему моделирования.

Отыскание компромиссного решения, удовлетворяющего этим требованиям, и является задачей стратегического планирования эксперимента.

Эксперимент, в котором реализуются все возможные сочетания уровней факторов, называется ***полным факторным экспериментом*** (ПФЭ).

Общее число различных комбинаций уровней в ПФЭ для k факторов можно вычислить так:

$$N = l_1 \cdot l_2 \cdot l_3 \cdot \ldots l_k$$

где l_i — число уровней i-го фактора.

Если число уровней для всех факторов одинаково, то $N = L^k$ (L — число уровней).

Недостаток ПФЭ — большие временные затраты на подготовку и проведение.

Например, если в модели отражены 4 фактора, влияющие на значение выбранного показателя эффективности, каждый из которых имеет 3 возможных уровня (значения), то план проведения ПФЭ будет включать 81 эксперимент ($N=3^4$). Если при этом каждый из них длится хотя бы одну минуту (с учетом времени на изменение значений факторов), то на однократную реализацию ПФЭ потребуется более часа.

Поэтому использование ПФЭ целесообразно только в том случае, если в ходе имитационного эксперимента исследуется взаимное влияние всех факторов, фигурирующих в модели.

Если такие взаимодействия считают отсутствующими или их эффектом пренебрегают, проводят ***частичный факторный эксперимент*** (ЧФЭ).

Известны и применяются на практике различные варианты построения планов ЧФЭ. Мы рассмотрим только некоторые из них.

1. ***Рандомизированный план*** — предполагает выбор сочетания уровней для каждого прогона случайным образом.

2. ***Латинский план*** («латинский квадрат»)— используется в том случае, когда проводится эксперимент с одним первичным фактором и несколькими вторичными. Суть такого планирования состоит в следующем. Если первичный фактор А имеет l уровней, то для каждого вторичного фактора также выбирается l уровней. Выбор комбинации уровней факторов выполняется на основе специальной процедуры, которую мы рассмотрим на примере.

Пусть в эксперименте используется первичный фактор А и два вторичных фактора — В и С; число уровней факторов l равно 4.

Соответствующий план можно представить в виде квадратной матрицы размером $l \times l$ (4×4) относительно уровней фактора А. При этом матрица строится таким образом, чтобы в каждой строке и в каждом столбце данный уровень фактора А встречался только один раз (табл. 2.2):

Таблица 2.3

Пример латинского плана

Значение фактора *B*	**Значение фактора *C***			
	*C*1	*C*2	*C*3	*C*4
*B*1	*A*1	*A*2	*A*3	*A*4
*B*2	*A*2	*A*3	*A*4	*A*1
*B*3	*A*3	*A*4	*A*1	*A*2
*B*4	*A*4	*A*1	*A*2	*A*3

В результате имеем план, требующий $4 \times 4 = 16$ прогонов, в отличие от ПФЭ, для которого нужно $4^3 = 64$ прогона.

3. ***Эксперимент с изменением факторов по одному.***

Суть его состоит в том, что один из факторов «пробегает» все l уровней, а остальные $n - 1$ факторов поддерживаются постоянными. Такой план обеспечивает исследование эффектов каждого фактора в отдельности. Он требует всего $N = l_1 + l_2 + ... + l_n$ прогонов (l_i — число уровней i-го фактора).

Для рассмотренного выше примера (3 фактора, имеющие по 4 уровня) $N=4 + 4 + 4=12$.

Еще раз подчеркнем, что такой план применим (как и любой ЧФЭ) только при отсутствии взаимодействия между факторами.

4. ***Дробный факторный эксперимент.***

Каждый фактор имеет два уровня — нижний и верхний, поэтому общее число вариантов эксперимента $N=2^k$, k — число факторов. Матрицы планов для $k = 2$ и $k = 3$ приведены ниже.

Таблица 2.4

Матрица плана дробного факторного эксперимента для $k = 2$

Номер эксперимента	Значение факторов	
	$x1$	$x2$
1	0	0
2	0	1
3	1	0
4	1	1

Таблица 2.5

Матрица плана дробного факторного эксперимента для $k = 3$

Номер эксперимента	Значение факторов		
	$x1$	$x2$	$x3$
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

Планы, построенные по такому принципу, обладают определенными свойствами (симметричности, нормированности, ортогональности и ротатабельности), обеспечивающими повышение качества проводимых экспериментов.

2.5.2. ТАКТИЧЕСКОЕ ПЛАНИРОВАНИЕ ЭКСПЕРИМЕНТА

Совокупность методов установления необходимого объема испытаний относят к тактическому планированию экспериментов.

Поскольку точность оценок наблюдаемой переменной характеризуется ее дисперсией, то основу тактического планирования эксперимента составляют так называемые методы понижения дисперсии.

В связи с этим для восприятия последующего материала читателю потребуются некоторые знания из области математической статистики.

Поскольку имитационное моделирование представляет собой статистический эксперимент, то при его проведении необходимо не только получить достоверный результат, но и обеспечить его «измерение» с заданной точностью.

Различие понятий «достоверный результат» и «точный результат» можно пояснить с помощью приведенного рисунка (рис. 2.10).

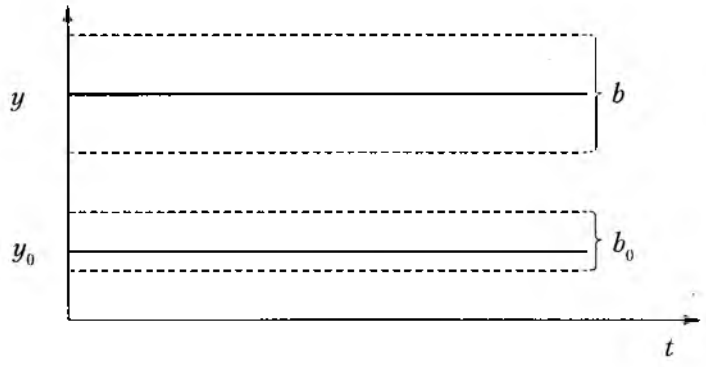


Рис. 2.10. Различие в понятиях «достоверный результат» и «точный результат».

На рисунке использованы следующие обозначения:

y, y_0 — истинное и ошибочное значения наблюдаемой переменной y;

b, b_0 — доверительные интервалы измерения величин y и y_0.

В общем случае объем испытаний (величина выборки), необходимый для получения оценок наблюдаемой переменной с заданной точностью, зависит от следующих факторов:

- вида распределения наблюдаемой переменной y (напомним, при статистическом эксперименте она является случайной величиной);
- коррелированности между собой элементов выборки;
- наличия и длительности переходного режима функционирования моделируемой системы.

Если исследователь не обладает перечисленной информацией, то у него имеется единственный способ повышения точности оценок истинного значения

наблюдаемой переменной — многократное повторение прогонов модели для каждого сочетания уровней факторов, выбранного на этапе стратегического планирования эксперимента. Такой подход получил название «формирование простой случайной выборки» (сокращенно — ПСВ). Другими словами, при использовании ПСВ каждый «пункт» стратегического плана просто выполняется повторно определенное число раз, и затем полученные результаты усредняются (вычисляются математическое ожидание и дисперсия наблюдаемой переменной). При таком подходе общее число прогонов модели, необходимое для достижения цели моделирования, равно произведению $Nc \times N_T$ (Nc — число сочетаний уровней факторов по стратегическому плану; N_T — число прогонов модели для каждого сочетания, вычисленное при тактическом планировании).

Например, если для полного факторного эксперимента $Nc = 81$, а для обеспечения требуемой точности оценок N_T должно быть равно 20, то общее число прогонов модели — 1620. Требуемое время для проведения всех испытаний (по минуте на каждое) — более 20 часов. То есть «модельер» должен трудиться почти сутки без сна и отдыха. Такой режим работы может вывести из строя не только человека, но и компьютер. Поэтому даже при использовании ПСВ до начала испытаний необходимо определить тот минимальный объем выборки, который обеспечит требуемую точность результатов.

Рассмотрим несколько основных вариантов вычисления необходимого объема испытаний (величину N_T).

1. Если случайные значения наблюдаемой переменной не коррелированы и их распределение не изменяется от прогона к прогону, то выборочное среднее можно считать нормально распределенным.

В этом случае число прогонов N_T, необходимое для того, чтобы истинное среднее y лежало в интервале $y \pm b$ с вероятностью $(1 - \alpha)$, определяется следующим образом:

$$N_T = \frac{Z^2 \cdot Dy}{b^2}, \qquad (2.2)$$

где Z — значение нормированного нормального распределения, которое определяется по справочной таблице при заданном уровне значимости $\alpha/2$;

Dy — дисперсия;

b — доверительный интервал.

Если требуемое значение дисперсии Dy до начала эксперимента неизвестно, целесообразно выполнить пробную серию из L прогонов и вычислить на ее основе выборочную дисперсию D:

$$D = \frac{1}{L-1} \sum_{i=1}^{L} (y_i - y_L)^2,$$

где y_L — выборочное среднее по результатам L прогонов.

Значение *Dy* подставляют в (2.2) и получают предварительную оценку числа прогонов *Nт*. Затем выполняют оставшиеся *N – L* прогонов, периодически уточняя оценку числа прогонов *Nт*.

2. Если наблюдаемая переменная — вектор, то оценку необходимого числа прогонов выполняют отдельно для каждой компоненты вектора. Наибольшее из полученных значений *Ni* принимают в качестве числа прогонов *Nт*.

Основной недостаток методов планирования, основанных на использовании простой случайной выборки — медленная сходимость выборочных средних к истинным средним с ростом объема выборки *Nт* (пропорционально значению $\sqrt{N_т}$). Это приводит к необходимости использования методов уменьшения ошибок, не требующих увеличения *Nт*. Такие методы называются ***методами понижения дисперсии*** и делятся на три группы:

активные (предусматривают формирование выборки специальным образом);

пассивные (применяются после того, как выборка уже сформирована);

косвенные (в которых для получения оценок наблюдаемой переменной используются значения некоторых вспомогательных величин).

Активных методов понижения дисперсии известно достаточно много. Выбор конкретного метода определяется, как правило, спецификой модели и целями эксперимента. Рассмотрим те из них, которые направлены на снижение влияния переходного периода. Выбор объясняется тем, что наличие и длительность переходного режима оказывает существенное влияние на качество результатов моделирования (в смысле точности). Вместе с тем, большинство ИМ используется для изучения функционирования системы в установившемся режиме.

Существует три основных метода уменьшения ошибок, обусловленных наличием переходного периода :

1. Значительное увеличение длительности прогона.

2. Исключение из рассмотрения переходного периода.

3. Инициализация модели при некоторых специально выбранных начальных условиях.

На практике снижения влияния переходного периода обычно добиваются одним из следующих способов:

- методом повторения;
- методом подынтервалов;
- методом циклов.

Метод повторения

При использовании этого метода каждое наблюдение получается при помощи отдельного прогона модели, причем все прогоны начинаются при одних и тех же начальных условиях, но используются различные последовательности случайных чисел.

Преимуществом метода является статистическая независимость получаемых наблюдений. Недостаток состоит в том, что наблюдения могут оказаться сильно смещенными под влиянием начальных условий .

Метод подынтервалов

Данный метод основан на разбиении каждого прогона модели на равные промежутки времени. Начало каждого интервала совпадает с началом очередного этапа наблюдений (рис. 2.11, Q – длина очереди заявок).

Рис. 2.11. Пример использования метода подынтервалов

Достоинство метода состоит в том, что влияние переходных условий со временем уменьшается и наблюдения точнее отражают поведение системы в стационарном режиме. Недостаток в том, что значения наблюдаемых переменных, полученные в начале очередного интервала, зависят от конечных условий предыдущего интервала (т. е. между интервалами существует автокорреляция).

Метод циклов

При использовании метода циклов влияние автокорреляции уменьшается за счет выбора интервалов таким образом, чтобы в их начальных точках условия были одинаковыми. Например, в качестве таких условий можно рассматривать длину очереди заявок на обслуживание. В этом случае удобно выбрать начало очередного интервала совпадающим с моментом, когда длина очереди становится равной нулю. Недостатком метода является меньшее по сравнению с методом подынтервалов число получаемых наблюдений (см.рис.2.12).

Рис. 2.12. Пример использования метода циклов

Пассивные методы влияют на подготовку и проведение эксперимента, но реализуются на этапе обработки и анализа результатов моделирования. Их довольно много; рассмотрим наиболее простой и распространенный — метод ***стратифицированной выборки.***

Суть метода состоит в следующем.

Выборка разделяется на части, называемые слоями (стратами). При этом необходимо, чтобы значения элементов выборки как можно меньше различались внутри одного слоя и как можно больше — между различными слоями. Внутри каждого слоя производят случайный отбор элементов и вычисляют среднее значение слоя y_i. Полученные оценки используют для вычисления МОЖ по выборке в целом:

$$y = \frac{1}{N} \sum_{i=1}^{k} N_i \cdot y_i$$

где N, N_i — объем всей выборки и i-го слоя соответственно,

k — число слоев.

Если считать, что оценки y_i независимы, то дисперсия по выборке в целом равна:

$$Dy = \frac{1}{N} \sum_{i=1}^{k} N_i \cdot D_{yi},$$

где Dy_i — дисперсия для i-го слоя.

При удачном выборе слоев величины Dy_i будут малы, а значит, и выборочная дисперсия Dy будет предпочтительнее, чем для оценки, полученной методами простой случайной выборки.

Косвенные методы понижения дисперсии основаны на том, что зачастую некоторые из выходных характеристик модели получить (вычислить) легче, чем другие. Их использование предполагает не только весьма глубокое знание сущности процессов, протекающих в системе, но и наличие формального описания взаимной зависимости параметров модели. Основные методы анализа такой зависимости рассматриваются в следующем, заключительном, разделе первой части книги.

2.6. ОБРАБОТКА И АНАЛИЗ РЕЗУЛЬТАТОВ МОДЕЛИРОВАНИЯ

Решения, принимаемые исследователем по результатам имитационного моделирования, могут быть конструктивными только при выполнении двух основных условий:

- полученные результаты обладают требуемой точностью и достоверностью;
- исследователь способен правильно интерпретировать полученные результаты и знает, каким образом они могут быть использованы.

Возможность выполнения первого условия закладывается, в основном, еще на этапе разработки модели и частично — на этапе планирования эксперимента. Достоверность результатов моделирования предполагает, что модель, с помощью которой они получены, не только является «правильной», но отвечает и некоторым дополнительным требованиям, предъявляемым к имитационным моделям. Эти требования и методы оценки соответствия им созданной модели рассматриваются ниже.

Способность исследователя правильно интерпретировать полученные результаты и принимать на их основе важные решения существенно зависит от степени соответствия формы представления результатов целям моделирования.

Если разработчик модели уверен, что полученные результаты будут использоваться в соответствии с одной, четко определенной целью, форма их представления может быть определена заранее. В этом случае преобразование экспериментальных данных к требуемому виду может производиться либо в ходе эксперимента, либо сразу после его завершения. Такой подход позволяет экономить память компьютера, необходимую для хранения большого количества необработанных данных, а также сократить время на анализ результатов и принятие решения.

Если же заранее конкретизировать цель моделирования сложно или целей несколько, данные должны накапливаться в базе данных и затем уже выдаваться в требуемой форме по запросу пользователя. Как правило, по такому принципу строятся системы автоматизации моделирования.

Во второй части данного раздела будет показано, как при правильной организации обработки экспериментальных данных могут быть получены дополнительные сведения о моделируемой системе.

2.6.1. ОЦЕНКА КАЧЕСТВА ИМИТАЦИОННОЙ МОДЕЛИ

Оценка качества модели является завершающим этапом ее разработки и преследует две цели:

1) проверить соответствие модели ее предназначению (целям исследования);

2) оценить достоверность и статистические характеристики результатов, получаемых при проведении модельных экспериментов.

При аналитическом моделировании достоверность результатов определяется двумя основными факторами:

1) корректным выбором математического аппарата, используемого для описания исследуемой системы;

2) методической ошибкой, присущей данному математическому методу.

При имитационном моделировании на достоверность результатов влияет целый ряд дополнительных факторов, основными из которых являются:

- моделирование случайных факторов, основанное на использовании датчиков СЧ, которые могут вносить «искажения» в поведение модели;
- наличие нестационарного режима работы модели;

• использование нескольких разнотипных математических методов в рамках одной модели;

• зависимость результатов моделирования от плана эксперимента;

• необходимость синхронизации работы отдельных компонентов модели;

• наличие модели рабочей нагрузки, качество которой зависит, в свою очередь, от тех же факторов.

Пригодность имитационной модели для решения задач исследования характеризуется тем, в какой степени она обладает так называемыми ***целевыми свойствами***. Основными из них являются:

• адекватность;

• устойчивость;

• чувствительность.

Ниже рассмотрены некоторые способы проведения оценки модели по каждому из них.

Оценка адекватности модели. В общем случае под адекватностью понимают степень соответствия модели тому реальному явлению или объекту, для описания которого она строится.

Вместе с тем, создаваемая модель ориентирована, как правило, на исследование определенного подмножества свойств этого объекта. Поэтому можно считать, что адекватность модели определяется степенью ее соответствия не столько реальному объекту, сколько целям исследования. В наибольшей степени это утверждение справедливо относительно моделей проектируемых систем (т.е. в ситуациях, когда реальная система вообще не существует).

Тем не менее во многих случаях полезно иметь формальное подтверждение (или обоснование) адекватности разработанной модели. Один из наиболее распространенных способов такого обоснования — использование методов математической статистики. Суть этих методов заключается в проверке выдвинутой гипотезы (в данном случае — об адекватности модели) на основе некоторых статистических критериев.

Замечание: при проверке гипотез методами математической статистики необходимо иметь в виду, что статистические критерии не могут доказать ни одной гипотезы: они могут лишь указать на отсутствие опровержения.

Итак, каким же образом можно оценить адекватность разработанной модели реально существующей системе?

Процедура оценки основана на сравнении измерений на реальной системе и результатов экспериментов на модели и может проводиться различными способами. Наиболее распространенные из них:

• по средним значениям откликов модели и системы;

• по дисперсиям отклонений откликов модели от среднего значения откликов системы;

• по максимальному значению относительных отклонений откликов модели от откликов системы.

Названные способы оценки достаточно близки между собой по сути, поэтому ограничимся рассмотрением первого из них.

При этом способе проверяется гипотеза о близости среднего значения наблюдаемой переменной Y среднему значению отклика реальной системы Y^*.

В результате No опытов на реальной системе получают множество значений (выборку) Y^*. Выполнив $Nм$ экспериментов на модели, также получают множество значений наблюдаемой переменной Y.

Затем вычисляются оценки математического ожидания и дисперсии откликов модели и системы, после чего выдвигается гипотеза о близости средних значений величин Y^* и Y (в статистическом смысле). Основой для проверки гипотезы является t-статистика (распределение Стьюдента). Ее значение, вычисленное по результатам испытаний, сравнивается с критическим значением $t_{кр}$, взятым из справочной таблицы. Если выполняется неравенство $t < t_{кр}$, то гипотеза принимается.

Необходимо еще раз подчеркнуть, что статистические методы применимы только в том случае, если оценивается адекватность модели существующей системе. На проектируемой системе провести измерения, естественно, не представляется возможным. Единственный способ преодолеть это препятствие заключается в том, чтобы принять в качестве эталонного объекта концептуальную модель проектируемой системы. Тогда оценка адекватности программно реализованной модели заключается в проверке того, насколько корректно она отражает концептуальную модель. Данная проблема сходна с проверкой корректности любой компьютерной программы, и ее можно решать соответствующими методами, например с помощью тестирования.

Оценка устойчивости модели. При оценке адекватности модели как существующей, так и проектируемой системе реально может быть использовано лишь ограниченное подмножество всех возможных значений входных параметров (рабочей нагрузки и внешней среды). В связи с этим для обоснования достоверности получаемых результатов моделирования большое значение имеет проверка устойчивости модели. В теории моделирования это понятие трактуется следующим образом.

Устойчивость модели — это ее способность сохранять адекватность при исследовании эффективности системы на всем возможном диапазоне рабочей нагрузки, а также при внесении изменений в конфигурацию системы.

Каким образом может быть оценена устойчивость модели? Универсальной процедуры проверки устойчивости модели не существует. Разработчик вынужден прибегать к методам «для данного случая», частичным тестам и здравому смыслу. Часто бывает полезна апостериорная проверка. Она состоит в сравнении результатов моделирования и результатов измерений на системе после внесения в нее изменений. Если результаты моделирования приемлемы, уверенность в устойчивости модели возрастает.

В общем случае можно утверждать, что чем ближе структура модели структуре системы и чем выше степень детализации, тем устойчивее модель.

Устойчивость результатов моделирования может быть также оценена методами математической статистики. Здесь уместно вспомнить основную задачу математической статистики. Она заключается в том, чтобы проверить гипотезу относитель-

но свойств некоторого множества элементов, называемого генеральной совокупностью, оценивая свойства какого-либо подмножества генеральной совокупности (т.е. выборки). В генеральной совокупности исследователя обычно интересует некоторый признак, который обусловлен случайностью и может иметь качественный или количественный характер.

В данном случае именно устойчивость результатов моделирования можно рассматривать как признак, подлежащий оценке. Для проверки гипотезы об устойчивости результатов может быть использован критерий Уилкоксона.

Критерий Уилкоксона служит для проверки того, относятся ли две выборки к одной и той же генеральной совокупности (т. е. обладают ли они одним и тем же статистическим признаком). Например, в двух партиях некоторой продукции измеряется определенный признак, и требуется проверить гипотезу о том, что этот признак имеет в обеих партиях одинаковое распределение; другими словами, необходимо убедиться, что технологический процесс от партии к партии изменяется несущественно.

При статистической оценке устойчивости модели соответствующая гипотеза может быть сформулирована следующим образом: при изменении входной (рабочей) нагрузки или структуры ИМ закон распределения результатов моделирования остается неизменным.

Проверку указанной гипотезы Н проводят при следующих исходных данных:

есть две выборки $X = (x_1, ..., x_n)$ и $Y = (y_1, ..., y_m)$, полученные для различных значений рабочей нагрузки; относительно законов распределения X и Y никаких предположений не делается.

Значения обеих выборок упорядочиваются вместе по возрастанию. Затем анализируется взаимное расположение x_i и y_j. В случае $y_j < x_i$ говорят, что пара значений (x_i, y_j) образует инверсию.

Например, пусть для $n = m = 3$ после упорядочивания получилась такая последовательность значений: $y_1, x_1, y_3, x_2, y_2, x_3$; тогда имеем инверсии: (x_1, y_1), (x_2, y_1), (x_2, y_2), (x_3, y_1), (x_3, y_2), (x_3, y_3).

Подсчитывают полное число инверсий U. Если гипотеза верна, то U не должно сильно отклоняться от своего математического ожидания M:

$$M = \frac{n \cdot m}{2} .$$

От гипотезы отказываются, если $|U - M| > U$кр (Uкр определяют по таблице для заданного уровня значимости).

Оценка чувствительности ИМ. Очевидно, что устойчивость является положительным свойством модели. Однако если изменение входных воздействий или параметров модели (в некотором заданном диапазоне) не отражается на значениях выходных параметров, то польза от такой модели невелика (ее можно назвать «бесчувственной»). В связи с этим возникает задача оценивания чувствительности модели к изменению параметров рабочей нагрузки и внутренних параметров самой системы.

Такую оценку проводят по каждому параметру X_k в отдельности. Основана она на том, что обычно диапазон возможных изменений параметра известен. Одна из наиболее простых и распространенных процедур оценивания состоит в следующем.

1) вычисляется величина относительного среднего приращения параметра X_k:

$$\Delta X_k = \frac{(X_{kmax} - X_{kmin}) \cdot 2}{(X_{kmax} + X_{kmin})} \cdot 100\% ;$$

2) проводится пара модельных экспериментов при значениях $X_k = X_{kmax}$ и $X_k = X_{kmin}$ и средних фиксированных значениях остальных параметров. Определяются значения отклика модели $Y_1 = f(X_{kmax})$ и $Y_2 = f(X_{kmin})$;

3) вычисляется ее относительное приращение наблюдаемой переменной Y:

$$\Delta Y = \frac{|Y_1 - Y_2| \cdot 2}{(Y_1 + Y_2)} \cdot 100\% .$$

В результате для k-го параметра модели имеют пару значений (ΔX_k, ΔY_k), характеризующую чувствительность модели по этому параметру.

Аналогично формируются пары для остальных параметров модели, которые образуют множество $\{\Delta X_k, \Delta Y_k\}$.

Данные, полученные при оценке чувствительности модели, могут быть использованы, в частности, при планировании экспериментов: большее внимание должно уделяться тем параметрам, по которым модель является более чувствительной.

Калибровка модели. Если в результате проведенной оценки качества модели оказалось, что ее целевые свойства не удовлетворяют разработчика, необходимо выполнить ее ***калибровку,*** т. е. коррекцию с целью приведения в соответствие предъявляемым требованиям.

Как правило, процесс калибровки носит итеративный характер и состоит из трех основных этапов:

1) глобальные изменения модели (например, введение новых процессов, изменение типов событий и т. д.);

2) локальные изменения (в частности, изменение некоторых законов распределения моделируемых случайных величин);

3) изменение специальных параметров, называемых калибровочными.

На первый взгляд, структурные изменения модели, как более сложные, должны рассматриваться только после того, как все попытки откалибровать модель путем изменения параметров и локальных модификаций окажутся безуспешными. Однако такая стратегия может скрыть структурное несоответствие или недостаточную степень детальности модели. В этом смысле начинать калибровку с внесения глобальных изменений значительно безопаснее.

Вообще целесообразно объединить оценку целевых свойств ИМ и ее калибровку в единый процесс. Именно такая стратегия принята в статистическом методе калибровки, описанном ниже.

Процедура калибровки состоит из трех шагов, каждый из которых является итеративным (рис. 2.13).

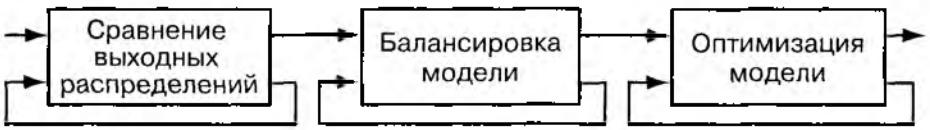


Рис.2.13. Схема процесса калибровки ИМ

Шаг 1. Сравнение выходных распределений.

Цель – оценка адекватности ИМ. Критерии сравнения могут быть различны. В частности, может использоваться величина разности между средними значениями откликов модели и системы.

Устранение различий на этом шаге основано на внесении глобальных изменений.

Шаг 2. Балансировка модели.

Основная задача – оценка устойчивости и чувствительности модели. По его результатам, как правило, производятся локальные изменения (но возможны и глобальные).

Шаг 3. Оптимизация модели.

Цель этого этапа – обеспечение требуемой точности результатов. Здесь возможны три основных направления работ:

дополнительная проверка качества датчиков СЧ;

снижение влияния переходного режима;

применение специальных методов понижения дисперсии.

2.6.2. ПОДБОР ПАРАМЕТРОВ РАСПРЕДЕЛЕНИЙ

В некоторых случаях имитационная модель сложной системы может быть реализована в виде набора отдельных моделей ее подсистем. При проведении экспериментов с такой моделью в целях сокращения затрат времени бывает необходимо заменять моделирование работы одной из подсистем некоторым числовым параметром (вспомните принцип параметризации), либо случайной величиной,распределенной по заданному закону. Чтобы такая замена была выполнена корректно, исследователь должен располагать описанием зависимости данного числового параметра от времени и других факторов, фигурирующих в модели.

При имитационном моделировании подбор законов распределений выполняется на основе статистических данных, полученных в ходе эксперимента.

В основе процедуры отыскания закона распределения некоторой величины по экспериментальным данным лежит проверка статистических гипотез.

Статистическая гипотеза — это утверждение относительно значений одного или более параметров распределения некоторой величины или о самой форме распределения.

Обычно выбирают две исходные гипотезы: основную — H_0 и альтернативную ей — H_1.

Статистическая проверка гипотезы — это процедура выяснения, следует ли принять основную гипотезу H_0 или отвергнуть ее.

Если в результате проверки гипотеза H_0 ошибочно отвергается, то имеет место ошибка I рода (характеризующаяся более тяжелыми последствиями); если гипотеза H_0 принимается при истинности H_1 — это ошибка второго рода.

Вероятности ошибок I и II рода (α и β) зависят от критерия, на основании которого будет выбираться одна из гипотез. Очевидно, что вероятности этих двух ошибок взаимосвязаны, то есть чем больше значение α, тем меньше β, и наоборот.

Обычное решение этой дилеммы состоит в том, что выбирают некоторое фиксированное значение α (как правило 0.05, 0.01, 0.001) и надеются, что β будет также мало. Фиксированное значение α называется **уровнем значимости**.

Для выбранного значения α определяется так называемая критическая область B, удовлетворяющая условию

$$P(Z \in B \mid H_0 \text{ верна}) \leq \alpha,$$

где Z — контрольная величина (критерий), представляющая собой некоторую функцию от выборки (результатов эксперимента).

Проверка гипотезы состоит в следующем. Производится выборка (проводится эксперимент), на основании чего вычисляется z — частное значение критерия Z. Если $z \in B$, то от гипотезы H_0 отказываются. Если z не лежит в B, то говорят, что полученные наблюдения **не противоречат** принятой гипотезе.

Разумеется, прежде чем выдвигать гипотезу относительно значений параметров распределения, необходимо определить вид самого закона распределения. Наиболее распространенный на практике и достаточно эффективный метод подбора закона распределения основан на использовании графического представления экспериментальных данных.

Они отображаются в виде так называемой гистограммы относительных частот, которая может быть построена как вручную, так и с помощью соответствующих инструментальных средств, входящих в состав большинства пакетов моделирования. Внешний вид гистограммы показан на рис. 2.14.

Для эффективного использования графических средств пакета MATLAB, которому посвящена большая часть книги, полезно знать методику построения гистограммы относительных частот.

Шаг 1. Вычисляется величина интервала гистограммы из следующего соотношения: $d = (y_{max} - y_{min})/n$, где $(y_{max} - y_{min})$ — диапазон изменения наблюдаемой переменной, n — число интервалов, выбранных исследователем.

Шаг 2. По результатам (или в процессе) моделирования определяется число попаданий значений y в i-й интервал.

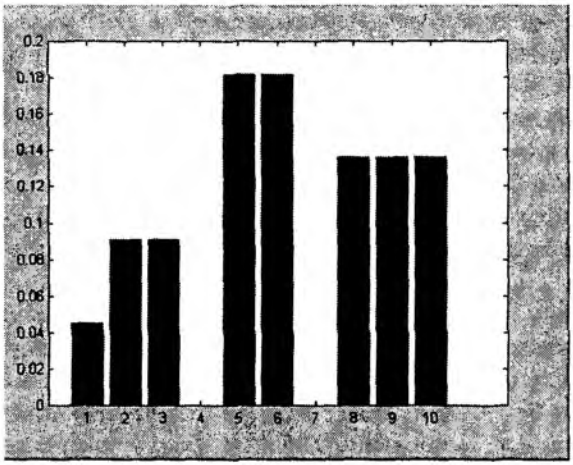


Рис. 2.14. Пример гистограммы относительных частот

Шаг 3. Вычисляется относительная частота попаданий наблюдаемой переменной в каждый интервал:

$$G_i = R_i/N,$$

где Ri — число попаданий в i-й интервал,

N — общее число измерений (объем выборки).

Шаг 4. На каждом i-м интервале строится прямоугольник со сторонами $d \times G_i$.

Сумма площадей прямоугольников гистограммы равна единице.

Для наиболее часто используемых статистических гипотез разработаны критерии, позволяющие проводить их проверку с наибольшей достоверностью. Рассмотрим основные из них.

t-критерий служит для проверки гипотезы о равенстве средних значений двух нормально распределенных СВ (X и Y) в предположении, что дисперсии их равны (хотя и неизвестны). Сравниваемые выборки могут иметь разный объем.

В качестве критерия используют величину

$$T = \frac{\bar{x} - \bar{y}}{\sqrt{(n_1 - 1)D_x + (n_2 - 1)D_y}} \sqrt{\frac{n_1 \cdot n_2 \cdot (n_1 + n_2 - 2)}{n_1 + n_2}}. \qquad (2.3)$$

Величина T подчиняется t-распределению Стьюдента.

Критическое значение для t-критерия определяется по таблице для выбранного значения α и числа степеней свободы $k = n_1 + n_1 - 2$.

Если вычисленное по (2.3) значение удовлетворяет неравенству $T \geq t$кр, то гипотезу $\boldsymbol{H_0}$ отвергают.

По отношению к предположению о «нормальной распределенности» величин x и y t-критерий не очень чувствителен. Его можно применять, если распределения СВ не имеют нескольких вершин и не слишком асимметричны.

F-критерий служит для проверки гипотезы о равенстве дисперсий D_x и D_y при условии, что x и y распределены нормально.

Гипотезы такого рода имеют большое значение в технике, так как дисперсия есть мера таких характеристик, как погрешности измерительных приборов, точность технологических процессов, точность наведения при стрельбе и так далее.

В качестве контрольной величины используется отношение дисперсий $F = D_x / D_y$ (или D_y / D_x – большая дисперсия должна быть в числителе).

Величина F подчиняется F-распределению (Фишера) с (m_1, m_2) степенями свободы ($m_1 = n_1 - 1$; $m_2 = n_2 - 1$). Проверка гипотезы состоит в следующем.

Для величины $a = \alpha/2$ и величин m_1, m_2 по таблице F-распределения выбирают значения $F_{a\,m1,m2}$. Если f, вычисленное по выборке, больше этого критического значения, гипотеза должна быть отклонена с вероятностью ошибки α.

Критерии согласия – это критерии, с помощью которых проверяют, удовлетворяет ли рассматриваемая СВ данному закону распределения.

Критерий согласия Пирсона (χ^2) служит для проверки гипотезы H_0 о том, что $F_y(y) = F_0(y)$, где $F_y(y)$ – истинное распределение СВ y; $F_0(y)$ – гипотетическое распределение.

Проверка производится следующим образом.

1) Область значений СВ y разбивается (произвольно) на k непересекающихся множеств («классов»).

2) В результате n опытов формируется выборка $(y_1, \ldots y_n)$.

3) Вычисляется контрольная величина χ^2:

$$\chi^2 = \left(\sum_{i=1}^{k} \frac{M_i^2}{np_i} \right) - n,$$

где M_i^2 – число значение y, попавших в i-й класс

p_i – теоретическая вероятность (для $F_0(y)$) попадания значения y в i-й класс.

4)По таблице χ^2-распределений находят критическое значение χ_α^2 для уровня значимости α и $m = k - 1$ степеней свободы. Если $\chi^2 \geq \chi_\alpha^2$, то гипотеза отвергается.

Критерий Колмогорова – Смирнова. При использовании данного критерия имеющуюся выборку $(y_1, \ldots, y_n)$ упорядочивают по возрастанию и строят следующую эмпирическую функцию распределения:

$$Fy(y) = \begin{cases} 0, & -\infty < y < y_1 \\ i/n & y_i \leq y < y_{i+1}\ i = 1, \ldots, n-1 \\ 1 & y_n \leq y < \infty \end{cases}$$

Контрольной величиной является

$$D = \max_{y} \left| \hat{F}(y) - F_0(y) \right|$$

Гипотеза H_0: $F_y(y) = F_0(y)$ отвергается, если вероятность попадания соответствующего критерия в критическую область оказывается меньше выбранного исследователем уровня значимости α.

Критическое значение критерия, как и в предыдущих случаях, находится по таблице.

Разумеется, проведение вручную расчетов, необходимых для проверки статистических гипотез, требует значительных затрат времени и сил. Поэтому многие современные математические пакеты имеют в своем составе средства, позволяющие свести к минимуму число операций, выполняемых пользователем вручную. Методика проверки статистических гипотез, используемая при работе с пакетом MATLAB, будет рассмотрена в пятой главе.

2.6.3. ОЦЕНКА ВЛИЯНИЯ И ВЗАИМОСВЯЗИ ФАКТОРОВ

Как правило, количественная оценка степени влияния того или иного фактора на значения наблюдаемой переменной (показателя эффективности) вызывает значительную сложность, особенно при наличии взаимного влияния факторов. Наиболее простой и доступный способ решения этой проблемы состоит в использовании результатов оценки чувствительности модели.

Однако эти результаты сложно представить в форме аналитической зависимости. Такое представление может оказаться весьма полезным для многих практических задач, связанных как с разработкой моделей (речь опять-таки идет о принципе параметризации), так и непосредственно с принятием решений по экспериментальным данным.

Отыскание аналитических зависимостей, связывающих между собой различные параметры, фигурирующие в модели, может быть основано на совместном использовании группы методов математической статистики: дисперсионного, корреляционного и регрессионного анализа. Подробному и строгому описанию соответствующих процедур посвящено огромное количество книг учебного, научного и справочного характера. Поэтому основная цель изложения последующего материала сводится к тому, чтобы показать роль и место указанных методов при проведении анализа данных, полученных в ходе имитационного эксперимента. Кроме того, наличие краткой характеристики методов статистического анализа данных в первой части книги «развяжет руки» автору при описании технологии анализа результатов моделирования средствами пакета MATLAB во второй ее части.

Однофакторный дисперсионный анализ

Его суть сводится к определению влияния на результат моделирования одного выбранного фактора.

Пусть, например, исследователя интересует средняя интенсивность отказов компьютера, и в созданной им модели учтены следующие факторы: интенсивность поступления заданий пользователей, интенсивность обращений в оперативную память, временные характеристики решаемых задач и интенсивность обращений к жесткому диску. Если предварительные данные говорят о том, что основной причиной отказов является ненадежная работа жесткого диска, то в качестве анализируемого фактора целесообразно выбрать интенсивность обращений к нему. Задача факторного анализа в данном случае состоит в том, чтобы оценить влияние указанного фактора на среднее число отказов.

Формально постановка задачи однофакторного дисперсионного анализа состоит в следующем. Пусть интересующий нас фактор x имеет l уровней. Для каждого из них получена выборка значений наблюдаемой переменной y: $y_j(1), y_j(2), \ldots y_j(l)$, $j=1, \ldots, n$, n — объем выборки (число наблюдений).

Необходимо проверить гипотезу $\boldsymbol{H_0}$ о равенстве средних значений выборок (т.е. о независимости значений y от значений исследуемого фактора x). Уравнение однофакторного дисперсионного анализа имеет вид:

$$y_{ij} = m + a_i + e_{ij},$$

где y_{ij} — j-е значение y в i-й серии опытов,

m — генеральное среднее случайной величины y (т. е. среднее значение наблюдаемой переменной, обусловленное ее «сущностью»),

a_i — неизвестный параметр, отражающий влияние фактора x («эффект» i-го значения фактора x),

e_{ij} — ошибка измерения y.

Для проверки гипотезы $\boldsymbol{H_0}$ используют F-критерий и переходят от проверки значимости различий средних к проверке значимости различий двух дисперсий:

- генеральной (обусловленной погрешностями измерений) — D_0;
- факторной (обусловленной изменением фактора x) — Dx.

Значение F-критерия вычисляется как отношение Dx/D_0 или D_0/Dx (в числителе должна стоять большая из дисперсий); затем по по таблице F-распределений находят его критическое значение Fкр для заданного уровня значимости и числа степеней свободы $m_1 = n - l$, $m_2 = l - 1$.

Если $F > F$кр, то гипотезу $\boldsymbol{H_0}$ отвергают, т. е. различия являются значимыми (фактор x влияет на значения y).

Многофакторный дисперсионный анализ(МДА) позволяет оценивать влияние на наблюдаемую переменную уже не одного, а произвольного числа факторов. Точнее, МДА позволяет выбрать из группы факторов, участвующих в эксперименте, те, которые действительно влияют на его результат.

Методику проведения многофакторного дисперсионного анализа рассмотрим применительно к частичному факторному эксперименту, проводимому в соответствии с латинским планом.

Пусть в эксперименте рассматриваются один первичный фактор и два вторичных, каждый из которых имеет n уровней (т. е. объем испытаний равен $N = n^2$).

Обозначим через y_{ijk} результат эксперимента при условии, что фактор $\boldsymbol{a}$ находился на уровне i, фактор $\boldsymbol{b}$ — на уровне j, фактор $\boldsymbol{c}$ — на уровне k. Множество значений, которые может принимать упорядоченная тройка (i, j, k), обозначим через L.

В этом случае уравнение дисперсионного анализа выглядит следующим образом:

$$y_{ijk} = m + a_i + b_j + g_k + e_{ijk},$$

где m — генеральное среднее случайной величины y,

a_i, b_j, g_k — неизвестные параметры («эффекты» соответствующих факторов).

Решение задачи дисперсионного анализа заключается в проверке гипотез о независимости результатов измерений от факторов $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$:

$$Ha: a_i = 0,\ i = \overline{1, n};$$
$$Hb: b_j = 0,\ j = \overline{1, n};$$
$$Hc: g_k = 0,\ k = \overline{1, n}.$$

Для этого по методу наименьших квадратов (МНК) находят оценки параметров m, a_i, b_j, g_k, минимизируя по указанным переменным (поочередно) функцию

$$SS = \sum_{L} (y_{ijk} - m + a_i + b_j + g_k)^2.$$

Затем по каждому фактору вычисляется F-статистика. Величина F есть мера потерь при принятии гипотезы $\boldsymbol{H_*}$. Чем больше F, тем хуже модель, отвергающая влияние соответствующего фактора. Таким образом, если вычисленное значение F больше Fкр, найденного по таблице для некоторого уровня значимости, то гипотеза отвергается.

Необходимо отметить, что дисперсионный анализ может использоваться для оценки влияния факторов, имеющих как количественный характер, так и качественный, поскольку в уравнении дисперсионного анализа фигурируют не сами факторы, а только их «эффекты».

В том случае, если все факторы носят количественный характер, взаимосвязь между ними и наблюдаемой переменной может быть описана с помощью уравнения регрессии.

Корреляционный и регрессионный анализ. Это два близких метода, которые обычно используются совместно для исследования взаимосвязи между двумя или более непрерывными переменными.

Методы ***корреляционного анализа*** позволяют делать статистические выводы о степени зависимости между переменными.

Величина линейной зависимости между двумя переменными измеряется посредством ***простого*** коэффициента корреляции, величина зависимости от нескольких — посредством ***множественного*** коэффициента корреляции.

В корреляционном анализе используется также понятие ***частного*** коэффициента корреляции, который измеряет линейную взаимосвязь между двумя переменными без учета влияния других переменных.

Если корреляционный анализ позволил установить наличие линейной зависимости наблюдаемой переменной от одной или более независимых, то форма зависимости может быть уточнена методами регрессионного анализа.

Для этого строится так называемое ***уравнение регрессии***, которое связывает зависимую переменную с независимыми и содержит неизвестные параметры. Если уравнение линейно относительно параметров (но необязательно линейно относительно независимых переменных), то говорят о ***линейной*** регрессии, в противном случае регрессия ***нелинейна***.

Рассмотрим ***простой корреляционный анализ***, т. е. метод определения взаимосвязи между двумя переменными.

Обозначим их x и y. Независимо от способа получения выборки имеются два предварительных шага для определения существования и степени линейной зависимости между x и y. Первый шаг заключается в графическом отображении точек (x_i, y_i) на плоскости (x, y) — т. е. в построении ***диаграммы рассеяния***. Анализируя диаграмму рассеяния, можно решить, допустимо ли предположение о линейной зависимости между x и y (см.рис.2.15).

Если r_{xy} не равен нулю, то на втором шаге вычисляется его точное значение.

Чем больше по абсолютному значению r_{xy}, тем сильнее линейная зависимость между переменными. При $|r_{xy}| = 1$ имеет место функциональная линейная зависимость между x и y вида $y = b_0 + b_1 x$, причем если $r_{xy} = +1$, то говорят о положительной корреляции, т.е. большие значения одной величины соответствуют большим значениям другой; при $r_{xy} = -1$ имеет место отрицательная корреляция; при $0 < r_{xy} < 1$ вероятна либо линейная корреляция с рассеянием (рис.2.15, *в*), либо нелинейная корреляция (рис.2.15, *г*).

При анализе результатов ИМ необходимо иметь в виду, что если даже удалось установить тесную зависимость между двумя переменными, это еще не является прямым доказательством их причинно-следственной связи. Возможно, имеет место стохастическая зависимость, обусловленная, например, коррелированностью последовательностей псевдослучайных чисел, используемых в имитационной модели.

Поэтому результаты корреляционного анализа целесообразно уточнить, проведя регрессионный анализ.

Регрессионный анализ позволяет решать две задачи:

1) устанавливать наличие возможной причинной связи между переменными;

2) предсказывать значения зависимой переменной по значениям независимых переменных. Эта возможность особенно важна в тех случаях, когда прямые измерения зависимой переменной затруднены.

Если предполагается линейная зависимость между x и y, то она может быть описана уравнением вида

$y_i = b_0 + b_1 \cdot x_i + e_i$ ($i = 1,..., n$, n — объем испытаний), которое называется ***простой линейной регрессией y по x***.

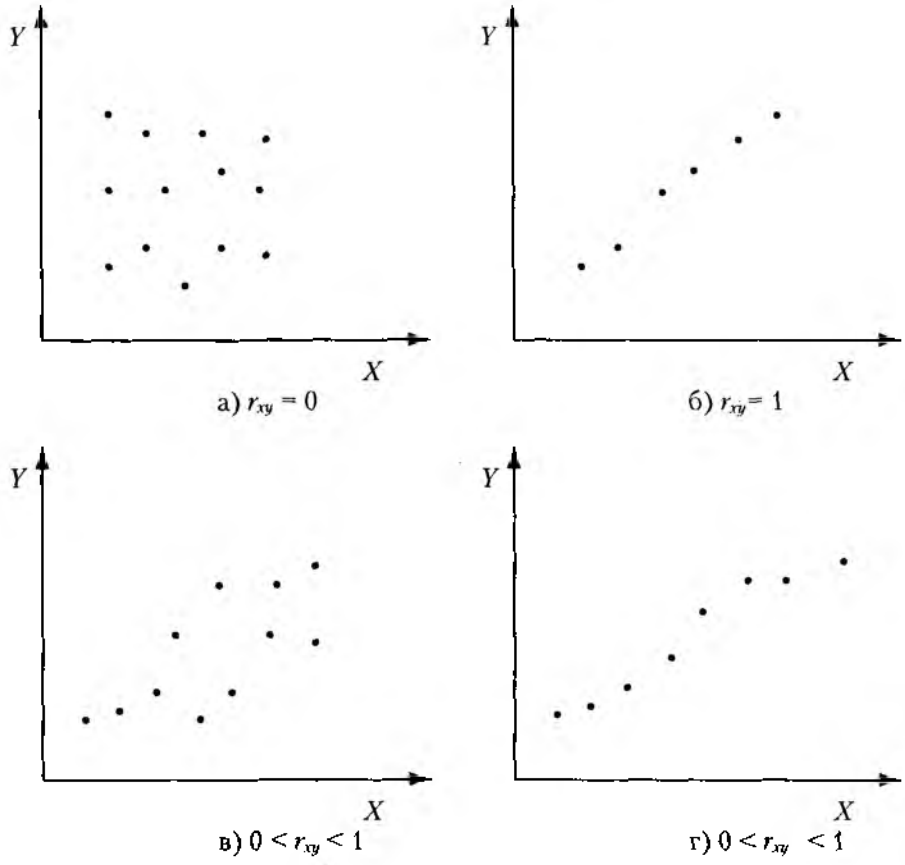


Рис. 2.15.Графическое отображение корреляции переменных

Величины b_0 и b_1 являются неизвестными параметрами, а e_i – случайные ошибки испытаний.

Цель регрессионного анализа – найти наилучшие в статистическом смысле оценки параметров b_0 и b_1 (величину b_1 обычно называют ***коэффициентом регрессии***).

Зная значения b_0 и b_1, можно найти оценку переменной y при $x = x_i$:

$$\hat{y}_i = \hat{b}_0 + \hat{b}_1 \cdot x_i.$$

Каким же образом полученное уравнение (или, как говорят, регрессионная модель) может быть использована для прогнозирования значений зависимой переменной y?

Чтобы ответить на этот вопрос, воспользуемся приводившимся уже примером, связанным с оценкой надежности компьютера. Предположим, исследова-

телю удалось посредством дисперсионного анализа установить наличие зависимости среднего числа отказов от интенсивности обращений к жесткому диску. Предположим также, что корреляционный анализ позволил определить линейный характер этой зависимости. В этом случае, имея уравнение регрессии, связывающее указанные величины, можно для каждого конкретного значения интенсивности обращений к диску «спрогнозировать» соответствующее среднее число отказов.

Разница между наблюдаемым и оцененным значением y при $x = x_i$ называется ***отклонением (или остатком)*** $d_i = y_i - y_i$. Величины отклонений могут быть использованы для проверки адекватности полученной модели. Для этого строится график $d = f(y)$ или $d=f(x)$ (см. рис.2.16), и по его виду делается предварительное заключение о степени адекватности регрессионной модели.

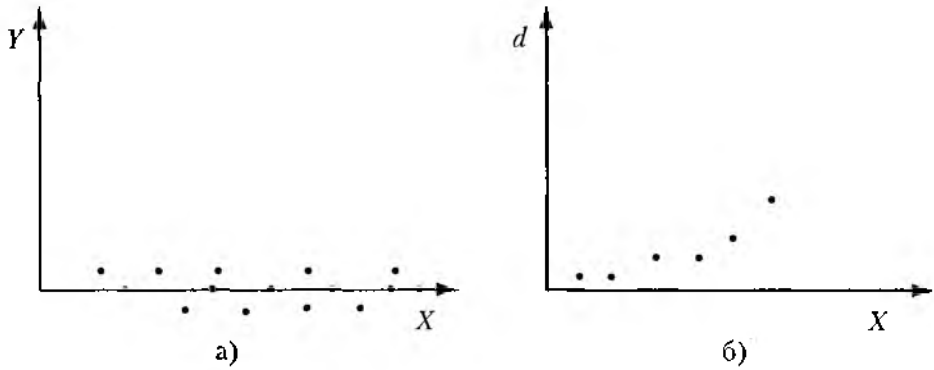


Рис.2.16. Графическое представление функции отклонений:
а — модель адекватна; *б* — необходимо введение дополнительной независимой переменной.

В случае нескольких независимых переменных имеет место ***множественная линейная регрессия***:

$$y = b_0 + b_1x_1 + ... + b_kx_k + e.$$

В этом случае для отыскания оценок b_j также используется метод наименьших квадратов.

В случае нелинейной регрессии основой для построения регрессионной модели опять-таки является МНК. Однако в этом случае для отыскания оценок b_j строится система нелинейных уравнений (относительно b_j), а для ее решения используются различные итерационные методы.

Как уже было отмечено ранее, эффективное использование процедур статистического анализа экспериментальных данных возможно только в том случае, если в распоряжении исследователя имеются соответствующие инструментальные средства, к описанию которых мы теперь можем перейти.

Но прежде подведем краткий итог изложенному в первой части книги.

• При сравнении альтернативных решений одним из наиболее эффективных инструментов их оценивания является математическое моделирование.

• В свою очередь, в тех случаях, когда значения оценок зависят от воздействия большого числа случайных факторов, либо интерес представляет развитие ситуации во времени, удобнее всего использовать имитационные модели.

• Создание имитационной модели сложной системы, функционирование которой предполагает наличие параллельных процессов, является весьма сложным делом, требующим от разработчика не только хорошего знания рассматриваемой предметной области, но достаточно прочных навыков в программировании.

• Результаты имитационного эксперимента могут быть использованы для принятия решения лишь при условии их корректной статистической обработки, что предъявляет к уровню подготовки исследователя целый ряд дополнительных требований.

И, наконец, главный вывод. Существенное повышение технологичности подготовки, проведения и анализа результатов имитационного моделирования возможно в том случае, если в распоряжении исследователя имеются соответствующие инструментальные средства.

Именно при выполнении этого условия скучноватая и потому не всегда понятная НАУКА превращается в ИСКУССТВО.

ЧАСТЬ II
МОДЕЛИРОВАНИЕ КАК ИСКУССТВО

ГЛАВА 3
ОСНОВНЫЕ ИНСТРУМЕНТЫ

> ...я взял листок бумаги и нарисовал на нем, что куда: где вход, где выход, где одеваться, где космонавта провожают и где кнопку нажимать... и я даже придумал, где ему умываться, и изобрёл для этого самовыдвигающиеся ведра, чтобы он в них собирал дождевую воду.
>
> *В. Драгунский*

3.1. ОБЩИЕ СВЕДЕНИЯ О ПАКЕТЕ MATLAB

История существования пакета MATLAB, название которого происходит от словосочетания Matrix Laboratory (Матричная лаборатория) насчитывает уже более двух десятков лет. Таким образом, можно считать, что развитие MATLAB «шло в ногу» с развитием средств вычислительной техники: от «больших» ЭВМ с маленькими интерактивными возможностями до настольных компьютеров, позволяющих использовать в работе все пять (а иногда и шесть) способов восприятия информации. И, несмотря на достаточно высокую скорость смены поколений вычислительной техники, MATLAB успевал впитывать все наиболее ценное от каждого из них.

В результате к настоящему времени MATLAB представляет собой весьма удачное сочетание возможностей математики с последними достижениями в области вычислительной техники.

Одно из основных достоинств пакета состоит в том, что для работы пользователю достаточно знать о нем ровно столько, сколько требует решаемая задача. Так, в простейшем случае MATLAB может сыграть роль обыкновенного калькулятора, для использования которого достаточно помнить знаки математических операций. Если же решаемая задача требует создания каких-либо специальных инструментов, MATLAB предоставляет в распоряжение пользователя практически универсальный язык объектно-ориентированного программирования в сочетании с интерактивными средствами отладки создаваемых программ.

И все-таки в первую очередь MATLAB — это средство математического моделирования, обеспечивающее проведение исследований практически во всех известных областях науки и техники. При этом структура пакета позволяет эффективно сочетать оба основных подхода к созданию модели: аналитический и имитационный.

Именно в сфере математического моделирования MATLAB позволяет наиболее полно использовать все современные достижения компьютерных технологий, в том числе средства визуализации и аудификации (озвучивания) данных, а также возможности обмена данными по сети Internet. Кроме того, пользователь имеет возможность создавать средствами MATLAB собственный графический интерфейс, отвечающий как его вкусам, так и требованиям решаемой задачи.

Как следует из названия пакета, он ориентирован в первую очередь на обработку массивов данных (матриц и векторов). Это позволило его разработчикам существенно повысить эффективность процедур, работающих с указанными типами данных, по сравнению с языками программирования «общего назначения» (*Pascal*, *C* и т. п.).

С точки зрения пользователя MATLAB представляет собой богатейшую библиотеку функций (в MATLAB 5.2 их около 800), единственная проблема работы с которой заключается в умении быстро отыскать те из них, которые нужны для решения данной задачи.

Для облегчения поиска библиотека функций разбита на разделы. Те из них, которые носят более общий характер и используются наиболее часто, входят в состав ядра MATLAB. Те же функции, которые являются специфическими для конкретной области, включены в состав соответствующих специализированных разделов. Эти разделы называются в MATLAB *Toolboxes* (*Инструменты*). Каждый из них имеет свое собственное название, отражающее его предназначение. Полная комплектация пакета MATLAB 5.2 содержит около 30 инструментальных приложений. В их число входят как достаточно стандартные для математических пакетов средства (решения дифференциальных и алгебраических уравнений, интегрального исчисления, символьных вычислений и т. д.), так и нетрадиционные, способные претендовать на определенную уникальность в своем роде: средства цифровой обработки изображений, поиска решений на основе нечеткой логики, аппарат построения и анализа нейронных сетей, средства финансового анализа и целый ряд других. Кроме того, имеются средства взаимо-

действия с популярными офисными продуктами фирмы Microsoft – MS Word и MS Excel (для включения в состав рабочей конфигурации пакета того или иного инструментального приложения требуется наличие соответствующего лицензионного соглашения).

Особое место среди инструментальных приложений занимает система визуального моделирования SIMULINK. В определенном смысле SIMULINK можно рассматривать как самостоятельный продукт фирмы Math Works (который даже в некоторых случаях продается в «именной» упаковке), однако он работает только при наличии ядра MATLAB и использует многие функции, входящие в его состав.

Именно поэтому в первых подразделах данной главы излагаются общие сведения о пакете MATLAB, необходимые для запуска и использования системы визуального моделирования SIMULINK. Более глубоко вопросы взаимодействия SIMULINK с ядром MATLAB и его инструментальными приложениями рассматриваются в Главе 5.

В свою очередь, дополнительные сведения о возможностях ядра MATLAB и некоторых инструментальных приложений, непосредственно не связанных с использованием SIMULINK, можно почерпнуть в справочных изданиях, список которых приведен в конце книги.

Необходимо отметить, что в MATLAB использована технология ассоциативной обработки файлов, поддерживаемая операционной системой Windows. Она заключается в том, что каждому типу файлов ставится в соответствие (ассоциируется с ним) определенное приложение, обеспечивающее обработку хранящихся в нем данных. Например, файлы с расширением .doc ассоциированы с текстовым редактором MS Word.

В MATLAB используется несколько типов файлов, для каждого из которых определен свой допустимый набор операций и реализующие их средства. При работе с SIMULINK в основном используются файлы трех типов:

M-файлы (имеющие расширение .m) – файлы, содержащие текст программы на языке MATLAB; в виде M-файлов реализованы все библиотечные функции MATLAB; по умолчанию M-файлы открываются с помощью собственного *Редактора/Отладчика* MATLAB;

Mdl-файлы (файлы с расширением .mdl) – файлы моделей SIMULINK; могут быть открыты либо с помощью SIMULINK (в виде графического окна с блок-диаграммой), либо с помощью *Редактора/Отладчика* MATLAB;

MAT-файлы (с расширением .mat) – файлы, содержащие данные в двоичном коде; доступ к ним возможен либо из командного окна MATLAB, либо с помощью специальных средств SIMULINK.

Необходимо также добавить, что хотя изложение материала ориентировано на работу с пакетом в среде Windows, MATLAB является платформно-независимой системой, и может работать под управлением и других операционных систем – UNIX и MacOS. При этом технология моделирования с помощью SIMULINK остается неизменной.

3.1.1. ИНСТАЛЛЯЦИЯ И ЗАГРУЗКА

Возможна либо сетевая, либо локальная установка пакета MATLAB. Объем памяти на жестком диске (в дальнейшем — HD), необходимый для установки ядра MATLAB, зависит от параметров файловой системы, установленных на вашем компьютере (точнее, от размера кластера) и составляет от 30 до 80 Мб.

Для установки MATLAB в полной комплектации необходимо около 350 Мб (включая файлы помощи, общий объем которых превышает 140 Мб; подробнее о средствах помощи будет сказано ниже). Пользователь может задать индивидуальную конфигурацию системы, оставив в предлагаемом списке только те разделы, которые его интересуют. Для «вычеркивания» раздела из списка достаточно убрать «птичку» в соответствующей строке, щелкнув на ней левой кнопкой мыши (ЛКМ). Наличие в списке ядра MATLAB при первоначальной установке является обязательным.

Для освоения технологии имитационного моделирования, изложенной в книге, достаточно установить три компоненты: MATLAB (ядро системы), SIMULINK (требует около 7 Мб на HD) и средства статистического анализа Statistics Toolbox (еще 2 Мб).

При изменении потребностей пользователя (или его интересов) исходная конфигурация может быть скорректирована. Для этого потребуется повторить процедуру инсталляции, внеся в список разделов MATLAB соответствующие изменения (при этом уже установленные разделы в список включать не нужно).

Необходимо отметить, что практически все разделы MATLAB содержат средства визуализации моделирования, реализуемые с помощью SIMULINK. Поэтому включение в рабочую конфигурацию любого раздела приводит к пополнению библиотеки блоков SIMULINK. Все блоки, включенные в состав библиотеки, являются «равноправными» и могут быть использованы при разработке любой модели независимо от того, к какому разделу MATLAB относится каждый из них.

При завершении инсталляции MATLAB на панели задач Windows автоматически создается соответствующая программная группа, содержащая три компоненты:

- ярлык запуска MATLAB;
- ярлык релиз-файла;
- ярлык программы удаления MATLAB из состава программного обеспечения компьютера (UnInstall MATLAB).

Для запуска MATLAB необходимо выбрать первую компоненту, дважды щелкнув на ней ЛКМ.

При интенсивном использовании MATLAB удобнее производить его запуск непосредственно с «Рабочего стола» Windows. Для этого на «Рабочем столе» следует создать соответствующий ярлык, выполнив следующие действия:

Шаг 1. Поместить курсор на свободное место «Рабочего стола» и щелкнуть правой кнопкой мыши (ПКМ).

Шаг 2. В появившемся списке действий выбрать команду *Создать*, а в открывшемся после этого дополнительном окне — *Ярлык* и нажать левую кнопку мыши. На экране появится диалоговое окно *Создание ярлыка*.

Шаг 3. В командной строке окна ввести (либо вручную, либо с помощью кнопки *Обзор*) полный путь к файлу matlab.exe, после чего «нажать» кнопку *Далее*.

Шаг 4. Ввести название ярлыка (по умолчанию — matlab.exe) и «нажать» кнопку *Готово*.

После закрытия диалогового окна на «Рабочем столе» появится ярлык со стандартной пиктограммой MATLAB в форме трехмерного графика.

3.1.2. ГЛАВНОЕ МЕНЮ. НАСТРОЙКА СИСТЕМЫ

Запуск MATLAB приводит к появлению на экране заставки (рис. 3.1). Затем открывается командное окно приложения (MATLAB Command Window) (рис. 3.2), состоящее из двух основных частей: главного меню пользователя и рабочего поля. Рассмотрим каждую из них в отдельности.

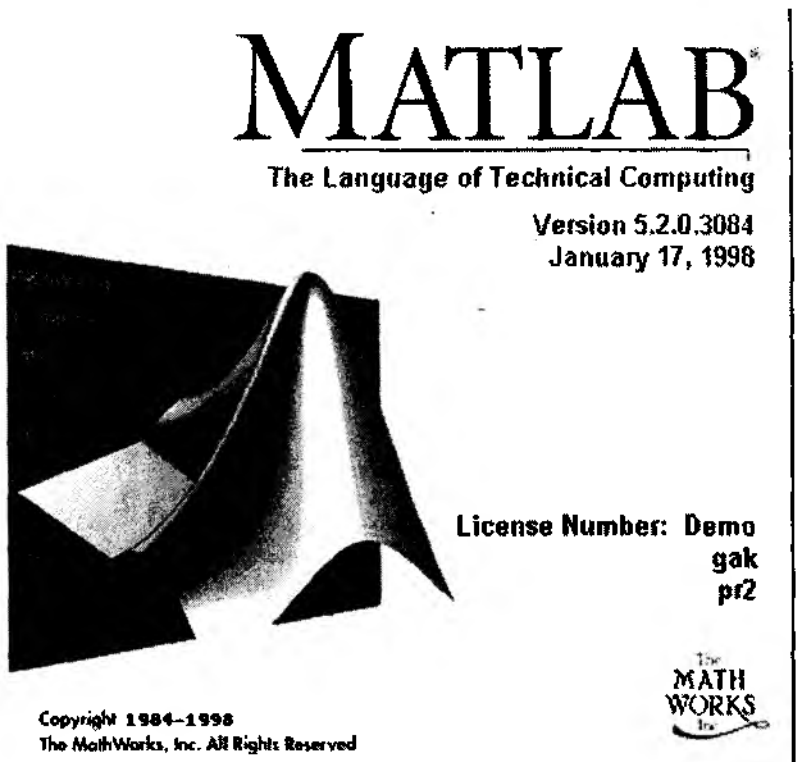


Рис. 3.1. Логотип пакета MATLAB

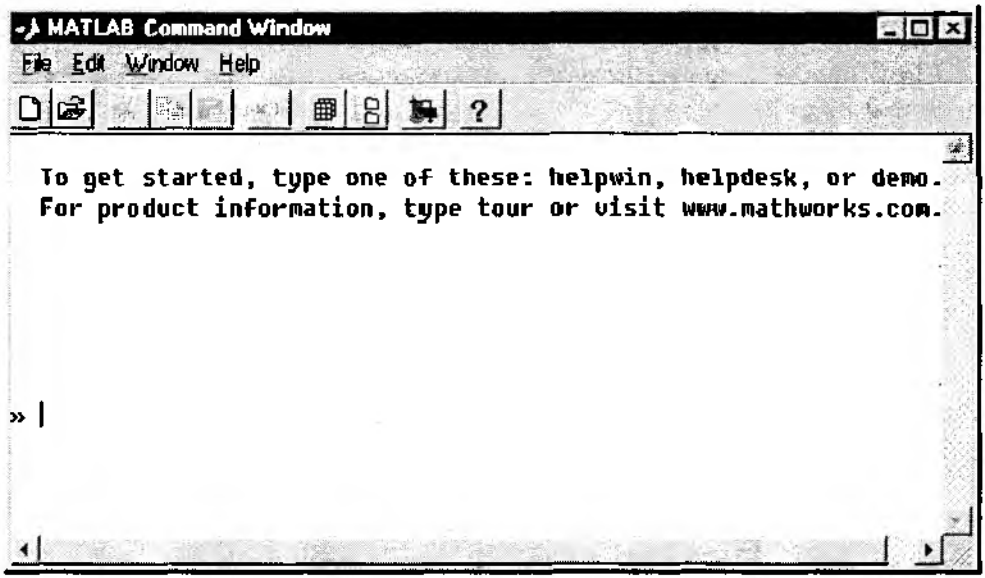


Рис. 3.2. Командное окно MATLAB

Главное меню по форме аналогично меню пользователя любого Windows-приложения.

В MATLAB главное меню содержит следующие разделы:

- ***File*** (команды работы с файлами и опции настройки системы);
- ***Edit*** (команды редактирования информации, отображенной в рабочем поле окна);
- ***Window*** (список открытых окон приложения);
- ***Help*** (команды вызова доступных средств помощи).

Ниже строки основного меню расположена панель кнопок, обеспечивающих быстрый доступ к наиболее часто используемым командам из разделов меню (рис 3.3).

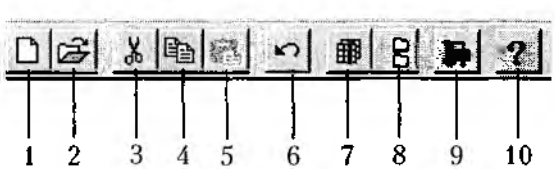


Рис. 3.3. Панель инструментов командного окна MATLAB

Каждая из кнопок снабжена «всплывающей» подсказкой, которая появляется на экране, когда курсор мыши находится на изображении соответствующей кнопки:

1 — создание нового М-файла;

2 — открытие существующего М-файла;

3...5 – стандартные операции редактирования (*вырезать, копировать, вставить*);

6 – *Undo* – отмена предыдущего действия (команды);

7 – *Workspace Browser* – просмотр содержимого рабочей области памяти MATLAB;

8 – *Path Browser* – вывод на экран диалогового окна настройки параметров работы с файлами и папками приложения;

9 – *New Simulink Model* – запуск SIMULINK;

10 – вызов справочника по функциям MATLAB (*Help Window*).

Из перечисленных выше разделов меню наибольший интерес представляет раздел ***File***. Содержащиеся в нем команды разбиты на несколько групп (рис. 3.4):

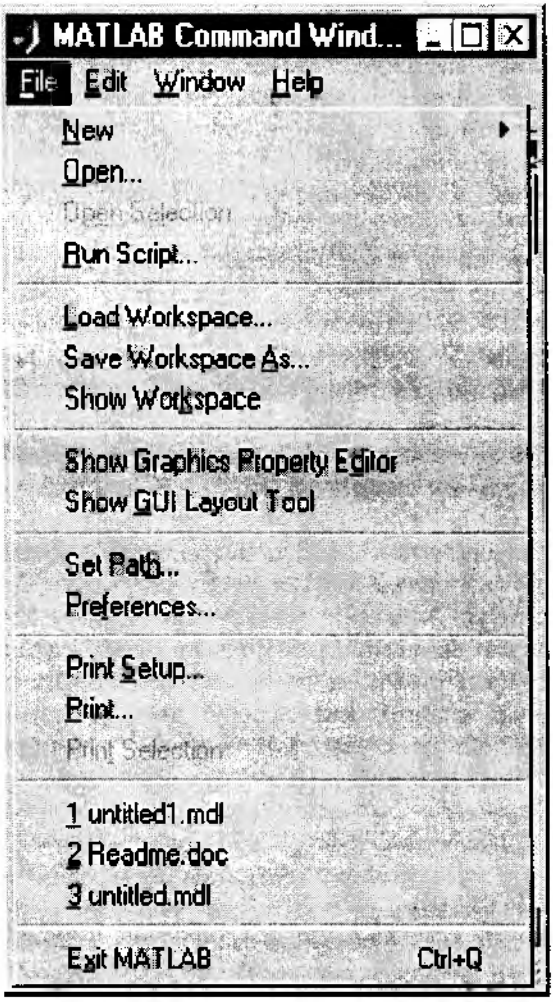


Рис. 3.4. Перечень команд раздела ***File***

- команды работы с файлами (*New, Open, Open Selection, Run Script*);
- команды работы с рабочей областью MATLAB;
- опции настройки системы;
- команды вывода на печать;
- список файлов, открывавшихся в последнее время;
- команда выхода из MATLAB.

В данном разделе будут рассмотрены только опции настройки системы. Особенности применения команд, относящихся к двум первым группам, излагаются в главах 4 и 5 применительно к работе с SIMULINK.

Вторая часть окна — рабочее поле. Оно является основным средством взаимодействия пользователя с системой и обеспечивает ведение диалога посредством командного языка MATLAB. Это означает, что любая информация, вводимая пользователем в активной строке (начинающейся символом «приглашение» >> и мерцающим курсором в форме вертикальной черты), воспринимается системой как команда, подлежащая исполнению.

В начале каждого сеанса работы с MATLAB в рабочем поле выводится сообщение-подсказка:

To get started, type one of these commands:
helpwin, helpdesk, or demo.

(Для начала введите одну из следующих команд: **helpwin**, **helpdesk** или **demo**).

Первая из указанных команд обеспечивает вызов справочника по функциям MATLAB, вторая — открывает файл помощи в формате HTML, а команда demo предназначена для демонстрации основных возможностей пакета MATLAB. Вопросы, связанные с использованием этих трех команд, обсуждаются подробнее в параграфе 3.1.3.

Кроме них в качестве команд можно использовать:

арифметические выражения (например 2+2; $x/3$);

имена библиотечных функций MATLAB (т. е. имена M-файлов) с соответствующими аргументами (например, *Sin*(3));

имена Mdl-файлов, расположенных в текущем директории;

некоторые операторы языка программирования MATLAB;

имена наборов данных (констант, скалярных переменных, матриц).

Если введенное выражение не является допустимой командой или содержит синтаксическую ошибку, система выдает соответствующее диагностическое сообщение.

Если команда не заканчивается символом (;), то она выполняется сразу же после нажатия клавиши <Enter>.

Использование (;) позволяет вводить в рабочем поле последовательность команд, результат выполнения которой будет выведен на экран только в том случае, если после очередной команды не стоит точка с запятой.

Если выполнение команды приводит к вычислению некоторого значения (скалярного или матрицы), то оно запоминается в рабочей области MATLAB в переменной с именем **ans** (от английского слова **answer** — ответ). Значение, занесенное в переменную ans, выводится на экран сразу после вычисления в форме **ans** =...

Переменная **ans** может быть использована при выполнении последующих команд. Например, выражение **ans+4** будет воспринято системой как команда, требующая увеличить значение переменной **ans** на 4. При необходимости результат выполнения команды может быть сохранен пользователем в рабочей области под любым именем с помощью операции присваивания. Например, выполнение команды **res = ans + 4** приводит к записи в рабочую область результата сложения под именем **res**. Аналогично может быть создан любой набор данных. В частности, команда

$$vector = [2\ 4\ 6\ 8]$$

позволяет создать и поместить в рабочую область массив из четырех элементов.

Для вывода на экран данных, хранящихся в рабочей области, достаточно набрать имя соответствующей переменной и нажать клавишу <Enter>.

Вывод информации в рабочее поле производится в режиме прокрутки, т. е. при выводе на экран очередного сообщения более старая информация сдвигается вверх. Вся информация, введенная пользователем или выведенная в рабочее поле системой, сохраняется в течение всего сеанса работы. При необходимости ее можно просмотреть, используя полосы вертикальной и горизонтальной прокрутки.

Если выведенная информация более не нужна, рабочее поле можно очистить с помощью команды *Clear Session* из раздела ***Edit*** главного меню.

Команды, вводимые пользователем в течение сеанса работы, сохраняются в буфере команд. Выполнявшуюся ранее команду можно выполнить без повторного набора, вызвав ее на экран с помощью клавиш управления курсором(↑ или ↓).

Настройка системы

Опции и команды настройки системы находятся в разделе ***File*** главного меню. К ним относятся:

- *Show GUI Layout Tool* (вызов средств разработки графического интерфейса пользователя: (GUI-Graphics User Interface);
- *Set Path...* (выбор рабочей папки);
- *Preferences...* (установка предпочтительных значений параметров системы).

Рассмотрим использование названных средств в порядке их усложнения.

Set Path. Опция обеспечивает выбор активного каталога (папки), который по умолчанию будет использоваться при выполнении операций работы с файлами.

Выбрав эту опцию в списке и дважды щелкнув левой кнопкой мыши, вы откроете соответствующее диалоговое окно, формат которого приведен на рис. 3.5.

Окно *MATLAB Path* содержит:

- редактируемую строку *Current Directory*, содержащую полный путь к активной папке;
- окно *Path*, в котором отображается список путей доступа к разделам функций MATLAB, входящих в рабочую конфигурацию;
- окно *Files in...*, в котором отображается список файлов, содержащихся в выделенном каталоге.

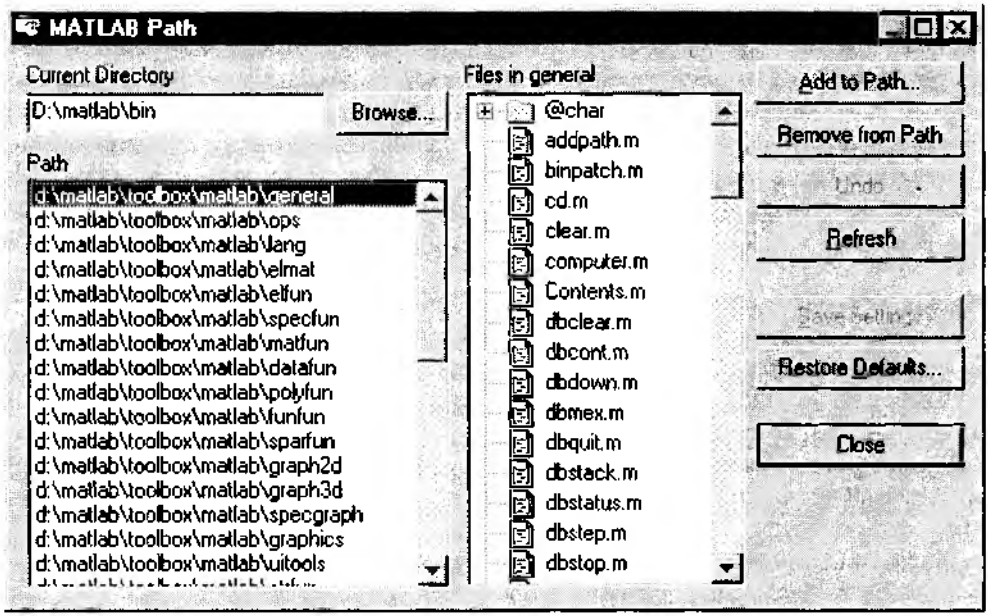


Рис. 3.5. Окно *MATLAB Path*

Выделить можно либо один из каталогов списка *Path* (установив курсор мыши в соответствующей строке и нажав левую кнопку мыши), либо текущий каталог в строке *Current Directory*, переведя на нее курсор и дважды нажав левую кнопку мыши.

Изменить текущий каталог в строке *Current Directory* можно либо вручную, введя путь с клавиатуры, либо с помощью кнопки *Browse...* . При ее «нажатии» открывается дополнительное диалоговое окно *Change Current Directory*, которое по структуре похоже на окно просмотра каталогов любого Windows-приложения. (рис. 3.6).

Отличие состоит в дополнительном переключателе *Add to Front / Add to Back*, который позволяет устанавливать приоритетность каталога, содержащего функции MATLAB (по умолчанию поиск библиотечных функций, используемых в приложениях, производится в каталоге *matlab\toolbox...*).

В правой части окна *MATLAB Path* расположен вертикальный ряд кнопок:

Add to Path... -- добавление раздела библиотеки функций MATLAB в список *Path*; название раздела и полный путь доступа к нему указывается в открывающемся диалоговом окне *Add to Path*, имеющим такой же формат, как и окно *Current Directory*.

Remove from Path — удаление выделенной строки из списка *Path*.

Undo — отмена предыдущей операции обновления списка *Path*; после «нажатия» название кнопки изменяется на *Redo* (т. е. можно произвести обратную отмену).

Refresh — вывод в окно *Path* обновленного списка.

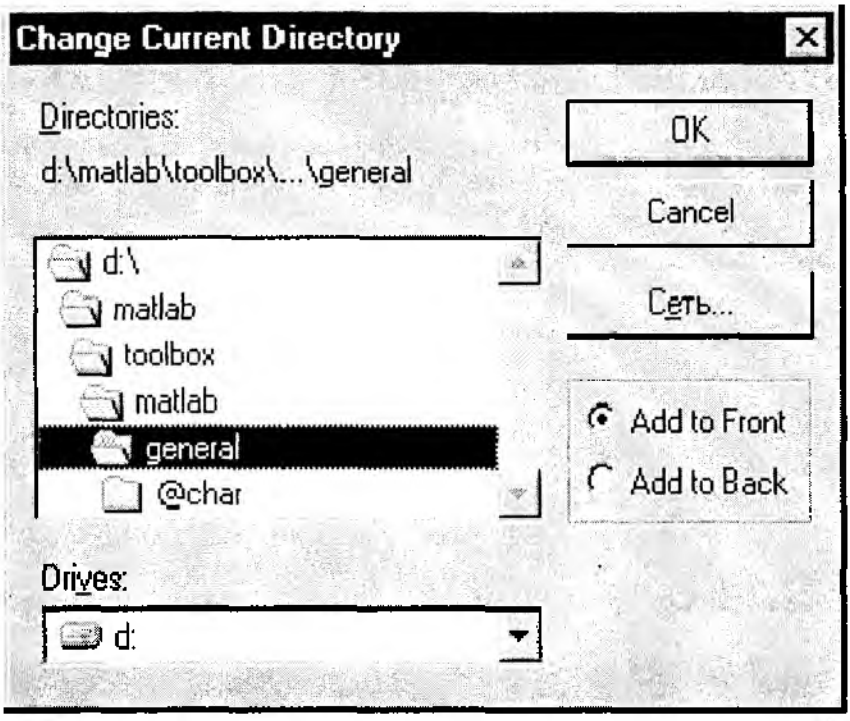


Рис. 3.6. Окно изменения текущего директория

Save Setting — сохранение внесенных изменений на HD (в файле *pathdef.m*).

Restore Defaults — восстановление предыдущих значений параметров опции *Path*.

Close — закрытие окна *MATLAB Path*.

Preferences... Опция обеспечивает выбор форматов представления числовой и текстовой информации в командном окне, формата копирования данных в буфер обмена и настройку ряда других параметров системы.

Все возможные настройки выполняются с помощью диалогового окна *Preferences*, открывающегося после двойного нажатия ЛКМ.

Окно имеет три вкладки: *General* (общие параметры системы), *Command Window Font* (выбор шрифта для командного окна MATLAB) и *Copying Options* (опции копирования).

Вкладка ***General*** (рис. 3.7) дает возможность произвести настройку следующих параметров.

1. Формат вывода числовых данных (*Numeric Format*); пользователю предлагается выбрать один из 10 вариантов:

- *Short* — вывод в формате с фиксированной точкой, выводится 5 значащих цифр (используется по умолчанию),

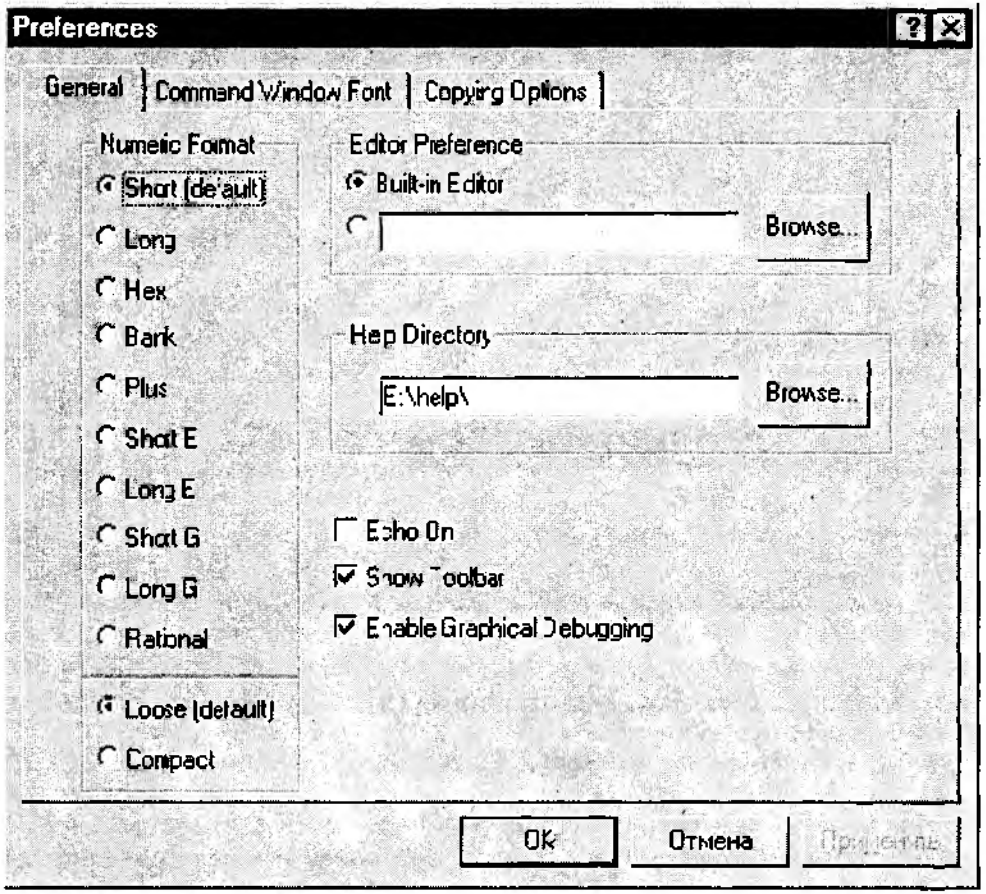


Рис. 3.7. Вкладка ***General*** окна установки параметров MATLAB

- *Long* – вывод в формате с фиксированной точкой, выводится 15 значащих цифр,
- *Hex* – шестнадцатеричный формат,
- *Bank* – формат для вывода денежных сумм с использованием знака доллара $,
- *Plus* – вывод положительных и отрицательных чисел со знаком (мнимая часть числа игнорируется),
- *Short E* – вывод дробных чисел в формате с плавающей запятой (в экспоненциальной форме), мантисса содержит 5 значащих цифр,
- *Long E* – вывод дробных чисел в экспоненциальной форме, в дробной части мантиссы выводится 15 значащих цифр;

для вывода порядка числа в двух последних форматах используется три знакоместа;

- *Short G* – «улучшенный» короткий формат, в дробной части числа выводится одна дополнительная значащая цифра;

- *Long G* — «улучшенный» длинный формат, в дробной части числа также выводится одна дополнительная значащая цифра;
- *Rational* — формат, при использовании которого дробные числа представляются в виде простых дробей (числитель/знаменатель); если числитель больше знаменателя, целая часть не выделяется.

2. Формат использования рабочего поля командного окна. Может быть указан либо *Loose* — «свободный» (используется по умолчанию), либо *Compact*.

При использовании формата *Loose* выводимые системой сообщения (ответы) разделяются одной пустой строкой. Формат *Compact* обеспечивает вывод сообщений в каждой строке рабочего поля.

3. *Editor Preference* (выбор используемого редактора программных файлов).

Можно использовать либо встроенный редактор MATLAB (если переключатель установлен в положении *Built-in*), либо указать какой-нибудь другой, введя имя соответствующего файла и путь доступа к нему в строке редактирования (или вручную, или применив кнопку *Browse...*).

4. *Help Directory*.

Опция позволяет указать каталог, содержащий help-файлы системы в формате html.

Путь доступа к каталогу может быть введен в строке редактирования либо вручную, либо с помощью кнопки *Browse*.

5. *Echo On* — флажок, позволяющий включить/выключить функции отображения в рабочем поле вводимых пользователем команд.

6. *Show Toolbar* — флажок, позволяющий показать/убрать панель кнопок главного меню.

7. *Enable Graphical Debugging* — флажок, позволяющий разрешить/не разрешить использование графических средств при работе отладчика.

Вкладка ***Command Window Font*** (рис. 3.8) обеспечивает выбор цвета, стиля и размера шрифта сообщений в командном окне MATLAB, а также цвета фона рабочего поля.

Выбор производится так же, как и в других Windows-приложениях — с помощью списков и выпадающих меню. В нижней части окна находится флажок *Display Fixed Pitch Fonts Only*, при установке которого в списке типов шрифтов окна параметров MATLAB остаются только шрифты с дискретным изменением размера символов.

Вкладка ***Copying Options*** позволяет установить параметры для операций копирования (рис. 3.9).

1. *Clipboard Format* — выбор формата копирования в буфер обмена:
- *Windows Metafile* — WMF-формат; используется для хранения векторных рисунков.
- *Windows Bitmap* — BMP-формат; обеспечивает хранение растровых изображений.

2. Флажок *Honor figure size properties* позволяет выбрать способ копирования графических окон MATLAB:

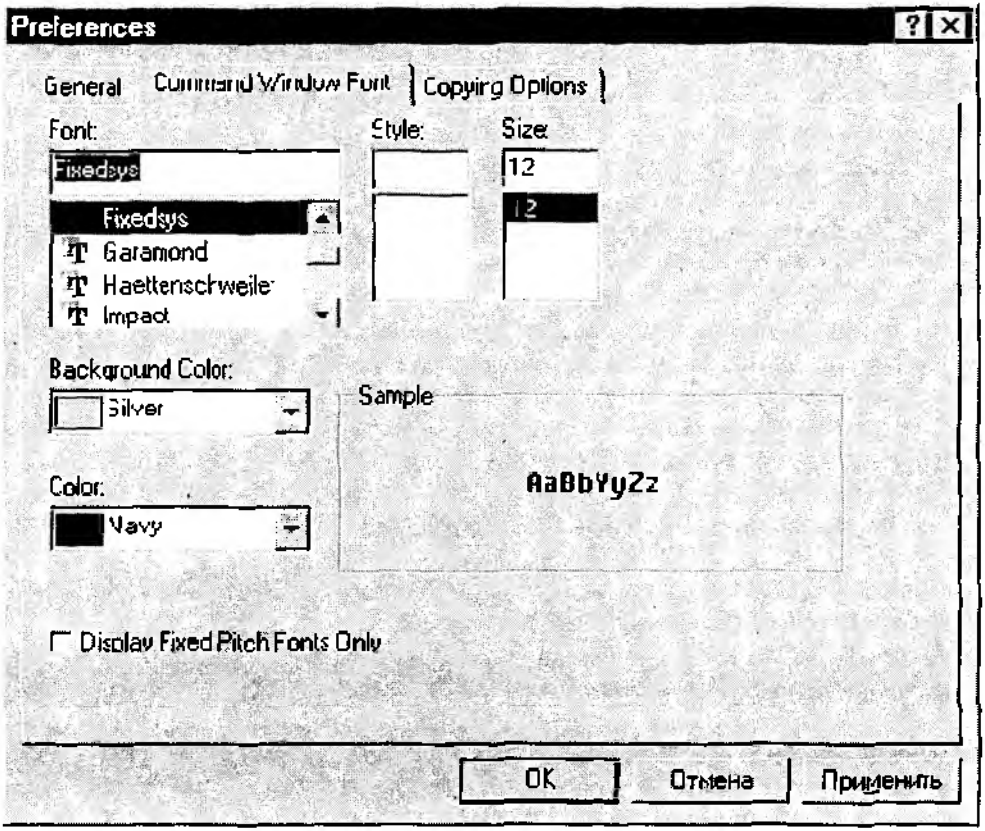


Рис. 3.8. Вкладка ***Command Window Font***

при установлении флажка пользователь может задать размер и расположение графика при печати;

если флажок снят, графическое окно будет выведено на печать в таком виде, как оно выглядит на экране.

3. Флажок *White background* устанавливает белый цвет фона графического окна; при снятом флажке цвет фона задается пользователем.

Разработка графического интерфейса пользователя (GUI) в среде MATLAB во многом сходна с технологией разработки приложений с помощью таких инструментов визуального программирования, как *Delphi, Visual Basic* или *Visual C*. И хотя создатели MATLAB постарались максимально облегчить пользователям работу по реализации собственного интерфейса, тем не менее для эффективного использования имеющихся инструментов требуются знания основ объектно-ориентированного программирования на языке MATLAB. Рассмотрение соответствующих вопросов выходит за рамки

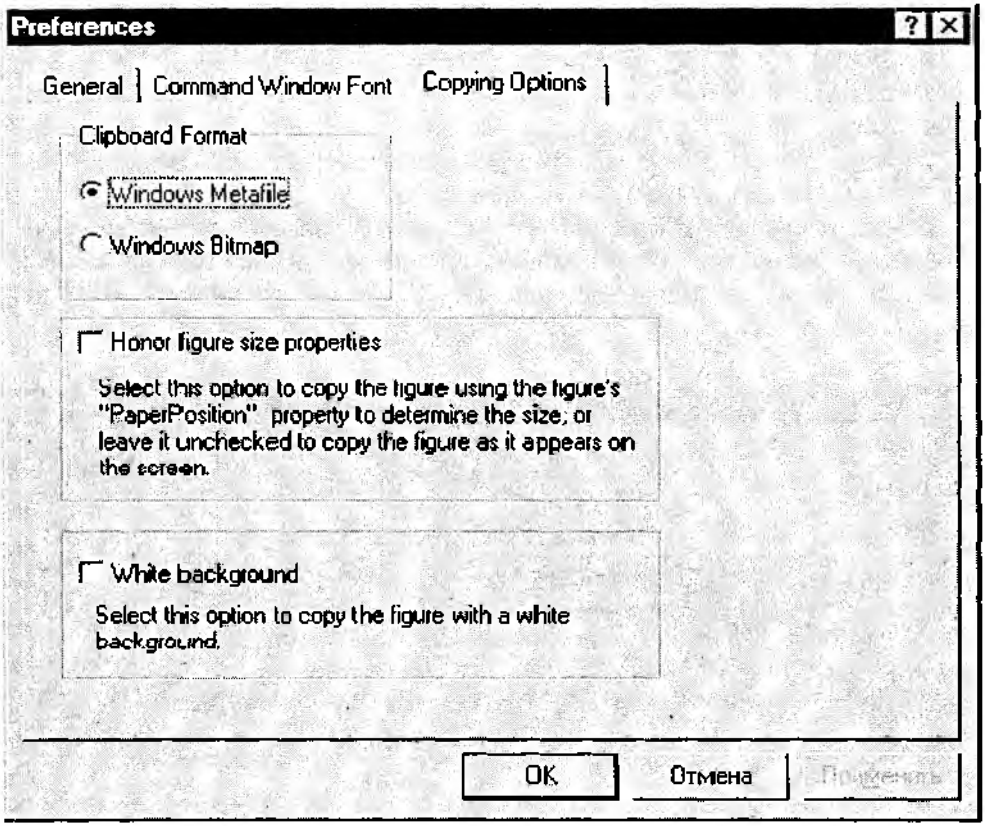


Рис. 3.9. Вкладка ***Copying Options*** окна установки параметров MATLAB

данной книги. Поэтому мы ограничимся лишь краткой характеристикой возможностей, которые предоставляет пользователям MATLAB при разработке GUI.

Как уже было сказано, вызов средств создания GUI возможен из командного окна MATLAB с помощью команды *Show GUI Layout Tool* меню *File*. В результате ее выполнения открывается диалоговое окно, содержащее все основные инструменты, необходимые для компоновки элементов GUI (рис. 3.10).

Кнопка *Property Editor* запускает редактор характеристик графических объектов (он может быть вызван и непосредственно из меню *File* командой *Show Graphics Property Editor*), который позволяет в дополнительном диалоговом окне изменять параметры компонентов GUI (размер, цвет, положение и т. д.).

Для создания нового графического окна (в MATLAB они называются *Figure*) необходимо «нажать» кнопку *Add Figure*.

При этом становятся доступны кнопки элементов GUI, расположенные в нижней части окна — осей для построения графиков, переключателей, меню и т. д.

Чтобы добавить элемент интерфейса в графическое окно, достаточно «нажать» с помощью мыши соответствующую кнопку, затем переместить курсор в графическое окно и щелкнуть ЛКМ в том месте, где должен находиться новый элемент. Изменение размеров и перемещение элементов интерфейса в пределах графического окна также производятся с помощью мыши.

При последующих вызовах средств создания интерфейса все ранее созданные пользователем графические окна автоматически выводятся в списке *Guide-Controlled Figure List*. Выбранное в нем окно становится доступным для редактирования.

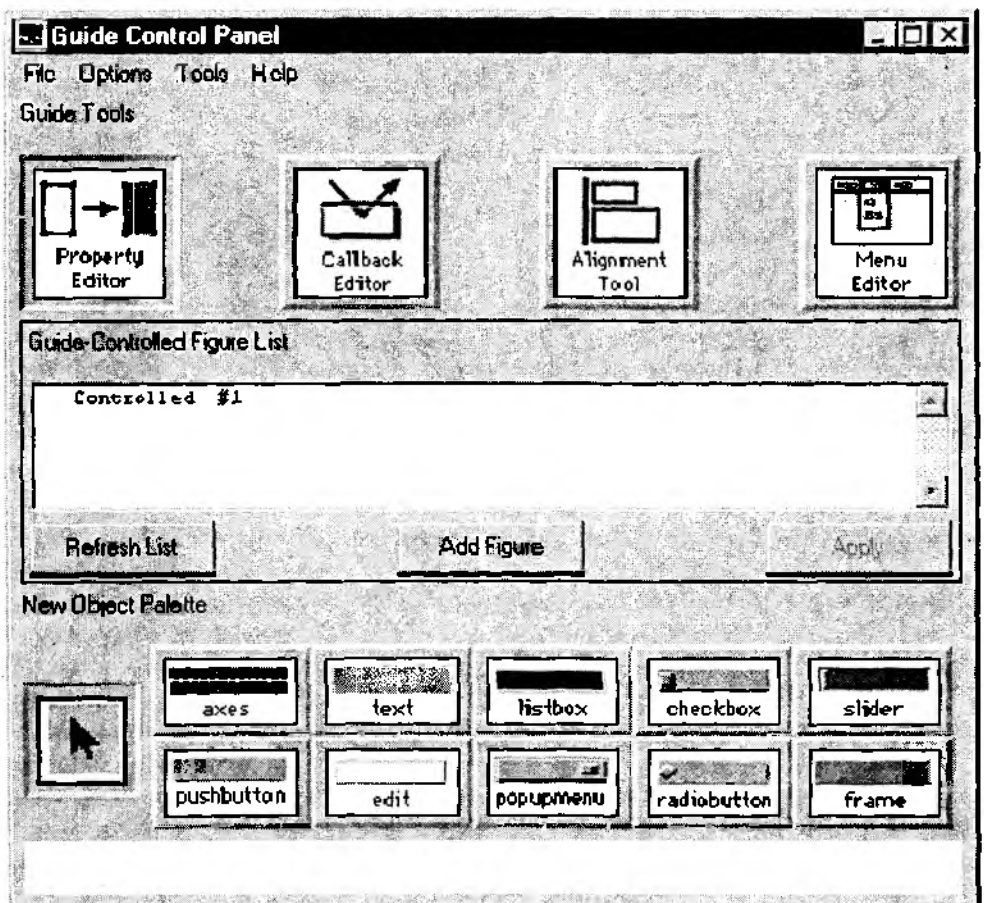


Рис. 3.10. Окно графического редактора GUI

3.1.3. ДЕМОНСТРАЦИЯ ВОЗМОЖНОСТЕЙ СИСТЕМЫ. СРЕДСТВА ПОМОЩИ ПОЛЬЗОВАТЕЛЮ

Самый простой и доступный способ познакомиться с основными возможностями системы MATLAB заключается в просмотре демонстрационных файлов, входящих в состав пакета. Для запуска процесса демонстрации используется команда *demo*, которую нужно ввести в активной строке командного окна.

Выполнение этой команды приводит к открытию окна *MATLAB Demos* (рис. 3.11). В левой части окна выводится список разделов и инструментальных приложений MATLAB, для которых можно просмотреть демонстрационный ролик. Этот список зависит от рабочей конфигурации MATLAB.

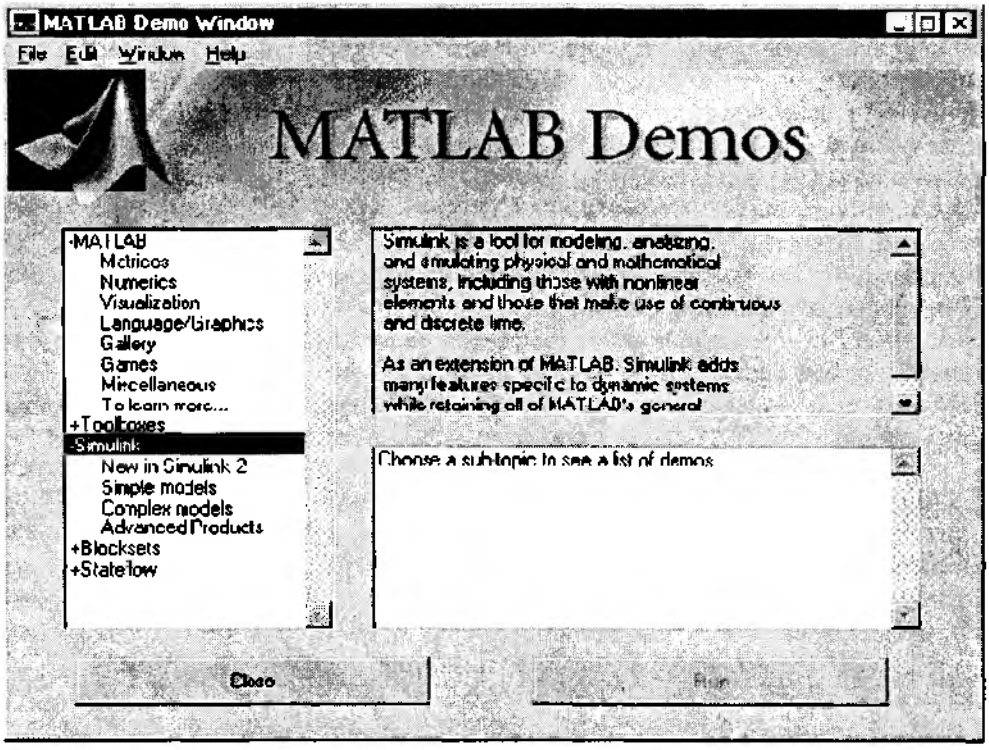


Рис. 3.11. Окно выбора демонстрационных файлов

Чтобы выбрать интересующий раздел, достаточно установить курсор на соответствующую строку и нажать ЛКМ. При этом обновляется содержимое окон, расположенных справа. В верхнем правом окне появляется текстовый комментарий к выбранному разделу, а в нижнем — список демонстрационных файлов, поясняю-

щих особенности его использования. Выбор файла в этом списке выполняется аналогично описанному выше. Имя выбранного файла появляется на кнопке *Run...*, расположенной под списком файлов. «Нажатие» ее приводит к запуску соответствующего демонстрационного ролика.

Выполнение практически всех демонстрационных файлов сопровождается открытием дополнительных диалоговых окон, с помощью которых пользователь может управлять процессом демонстрации.

Наряду со средствами демонстрации возможностей MATLAB, существенную помощь в его изучении могут оказать текстовые электронные справочные системы, входящие в комплект поставки. Таких систем три: гипертекстовая система на основе файлов в формате HTML (*HiperText Markup Language*), справочная документация в формате PDF (*Portable Document Format*) и встроенная справочная система по функциям MATLAB. Первые две из названных систем являются в достаточной степени автономными по отношению к MATLAB и пользователь при инсталляции может сам определить их состав, причем он не обязательно должен совпадать с перечнем устанавливаемых компонент пакета. В отличие от них, состав встроенного справочника MATLAB определяет автоматически, исходя из перечня устанавливаемых компонентов.

Справочная система в формате HTML

Для работы с ней на компьютере должно быть установлено одно из двух средств просмотра HTML-файлов: *Microsoft Explorer* (версии 3.0 и выше) или *Netscape Navigator*. Вызов данной справочной системы во время сеанса работы с MATLAB может быть выполнен либо непосредственно из командного окна путем ввода команды *Helpdesk*, либо с помощью той же команды, но выбранной из раздела меню Help. При необходимости эту команду можно выполнить, и работая со встроенным справочником MATLAB (подробнее об этом будет сказано немного позже).

Справочная система в формате PDF

Для просмотра документации в формате PDF необходимо установить специальную программу чтения файлов такого формата, которая называется *Acrobat Reader* из семейства продуктов Adobe фирмы Adobe Systems Inc. Эта программа входит в комплект поставки MATLAB и может быть установлена по желанию пользователя в процессе инсталляции пакета. Формат PDF является, пожалуй, наиболее удобным для получения «твердой» копии документации по интересующему разделу (если, конечно, у вас по какой-то причине не оказалось под рукой предоставляемого вместе с программным продуктом комплекта документации).

Справочник по функциям MATLAB

Справочник содержит сведения о назначении и параметрах, а также примеры использования функций MATLAB, входящих в состав рабочей конфигурации пакета. Изменение конфигурации приводит к изменению списка функций, по которым может быть получена справка.

Открыть справочник можно тремя способами:

- нажав соответствующую кнопку на панели меню командного окна;

• выбрав команду *HelpWindow* из раздела *Help* главного меню;
• введя команду *helpwin* в активной строке командного окна.

Формат основного окна справочника показан на рис. 3.12. При открытии справочника на его первой странице приводится список разделов библиотеки функций MATLAB, как входящих в ядро пакета, так и относящихся к установленным инструментальным приложениям, с краткой характеристикой каждого раздела.

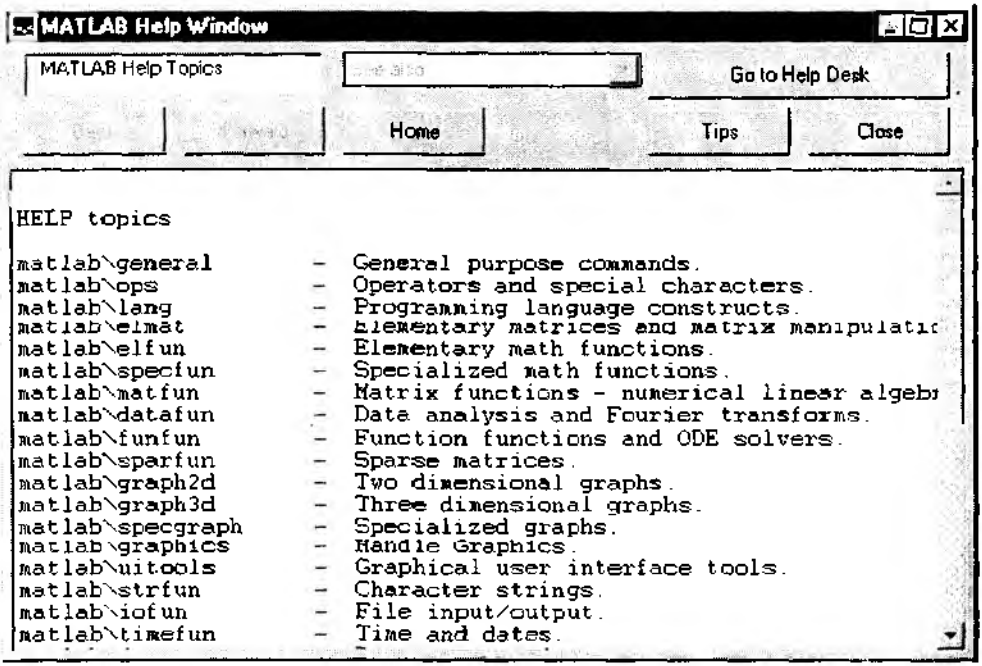


Рис. 3.12. Первая «страница» встроенного справочника MATLAB

Чтобы получить полную информацию по интересующему разделу, необходимо установить курсор на соответствующую строку и дважды нажать ЛКМ. В поле просмотра появится список функций, входящих в этот раздел, с указанием назначения каждой из них. Выбрав в списке нужную функцию и щелкнув ЛКМ, можно получить о ней более подробные сведения.

Название просматриваемого раздела справочника отображается в строке редактирования, расположенной в левом верхнем углу окна *MATLAB Help Window*.

Строку редактирования можно использовать для перехода к другому разделу справочника. Для этого следует набрать в ней название этого раздела (или имя функции), вывести курсор за пределы строки и нажать ЛКМ (либо клавишу *<Enter>* на клавиатуре).

Список дополнительной информации, относящейся к просматриваемому разделу, выводится в виде выпадающего меню в окне *See also* (*Смотри также*), расположенном правее строки редактирования. Чтобы вывести на экран эту информацию, нужно раскрыть меню и выбрать в нем нужный пункт.

Для перехода между разделами, которые были открыты ранее, используются кнопки *Back* (*Назад*) и *Forward* (*Вперед*), расположенные под строкой редактирования. Кнопка *Home* (*Домой*) позволяет вернуться на первую страницу справочника.

В правом верхнем углу окна имеются еще три кнопки:

Go To Help Desk — переход к головному файлу гипертекстовой системы в формате HTML;

Tips — открывает окно подсказки по трем командам вызова помощи: *Help*, *HelpWin* и *Helpdesk*.

Close — закрыть справочник по функциям.

Существует и другой способ доступа к информации, хранящейся во встроенном справочнике, — с помощью соответствующих команд, вводимых в командном окне. Таких команд две: *help* и *lookfor*.

Команда ***help*** имеет формат *help <имя М-функции>* и позволяет вывести непосредственно в командное окно информацию по запрашиваемой функции. Если имя функции опущено, в окно выводится список разделов справочника (в той же форме, как на его первой странице).

Команда ***lookfor*** обеспечивает поиск и выдачу справочной информации по ключевому слову. Если данная команда используется в формате *lookfor <ключевое слово>*, то при поиске просматривается только первая строка комментария, и она же выводится на экран, если в ней встретилось ключевое слово. Ввод команды в формате *lookfor <ключевое слово> — all* позволяет использовать при поиске и выводить на экран все строки комментария.

3.2. SIMULINK — ИНСТРУМЕНТ ВИЗУАЛЬНОГО МОДЕЛИРОВАНИЯ

3.2.1. ОБЩАЯ ХАРАКТЕРИСТИКА. ДЕМОНСТРАЦИЯ ВОЗМОЖНОСТЕЙ

Разработка моделей средствами SIMULINK (в дальнейшем *S*-моделей) основана на использовании технологии *Drag-and-Drop* (*Перетащи и Оставь*). В качестве «кирпичиков» для построения *S*-модели используются модули (или блоки), хранящиеся в библиотеке SIMULINK.

SIMULINK хорош тем, что, с одной стороны, обеспечивает пользователю доступ ко всем основным возможностям пакета MATLAB, а с другой — является достаточно самостоятельной его компонентой, в том смысле, что при работе с ним не

обязательно иметь навыки в использовании других инструментов, входящих в состав пакета.

Блоки, включаемые в создаваемую модель, могут быть связаны друг с другом как по информации, так и по управлению. Тип связи зависит от типа блока и логики работы модели. Данные, которыми обмениваются блоки, могут быть скалярными величинами, векторами или матрицами произвольной размерности.

Любая *S*-модель может иметь иерархическую структуру, то есть состоять из моделей более низкого уровня, причем число уровней иерархии практически не ограничено.

Наряду с другими параметрами моделирования пользователь может задавать способ изменения модельного времени (с постоянным или переменным шагом), а также условия окончания моделирования.

В ходе моделирования имеется возможность наблюдать за процессами, происходящими в системе. Для этого используются специальные «смотровые окна», входя-

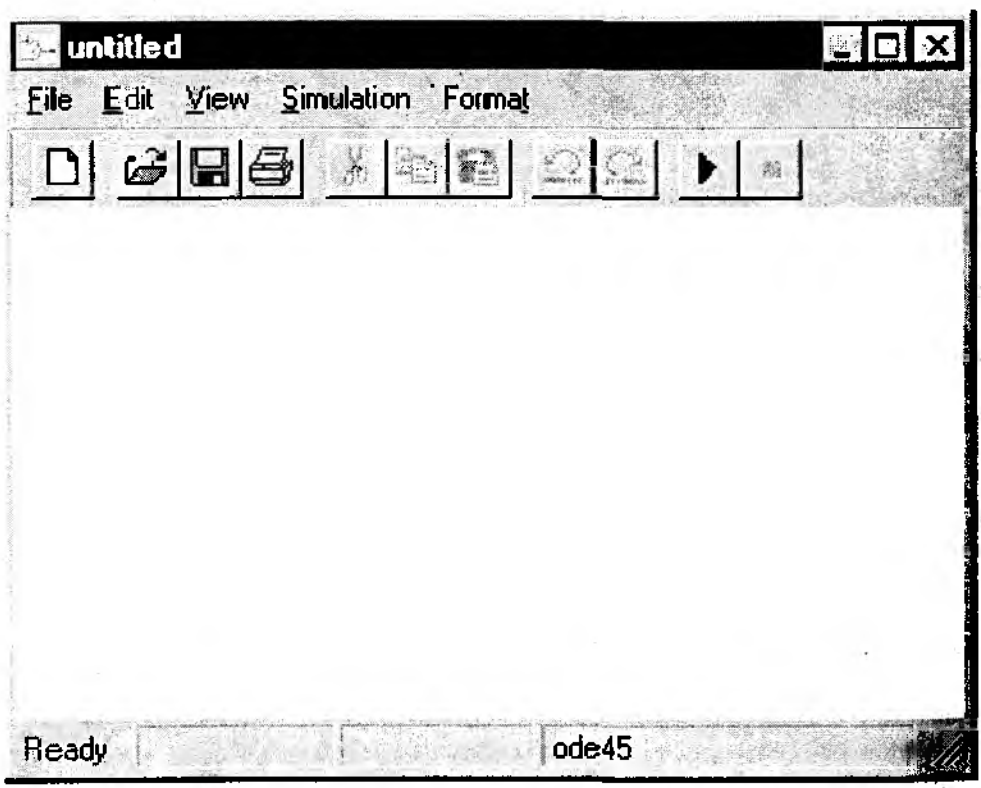


Рис. 3.13. Пустое окно для создания блок-диаграммы модели

щие в состав библиотеки SIMULINK. Интересующие пользователя характеристики системы могут быть представлены как в числовой, так и в графической форме. Кроме того, существует возможность включения в состав модели средств анимации.

Еще одно важное достоинство SIMULINK заключается в том, что он является открытой системой: состав библиотеки может быть пополнен пользователем за счет разработки собственных блоков.

Запуск SIMULINK можно произвести либо нажав соответствующую кнопку на панели меню командного окна, либо выбрав команду *New Model* в разделе *File* главного меню.

При запуске SIMULINK открываются два окна:

• пустое окно *untitled* (заготовка для создания новой *S*-модели) (рис. 3.13)

• окно *Library: simulink*, содержащее перечень основных разделов библиотеки SIMULINK (рис. 3.14).

Оба окна имеют сходную структуру и содержат строку меню, панель инструментов, строку состояния (только MATLAB 5.2) и рабочее поле.

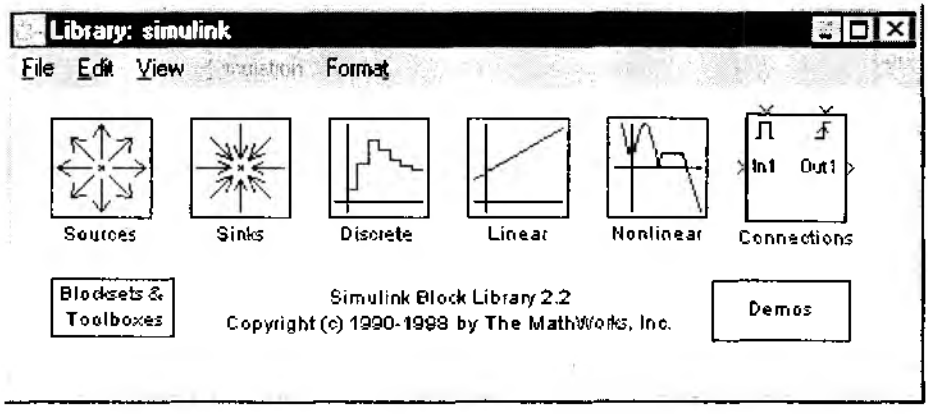


Рис. 3.14. Окно разделов библиотеки SIMULINK

Меню содержит следующие разделы:

File – команды работы с mdl-файлами,

Edit – команды редактирования блок-диаграммы и опции для работы с библиотекой,

View – команды изменения формата окна (показать/убрать панель инструментов и строку состояния);

Simulation – команды управления моделированием,

Format – команды редактирования формата (т. е. внешнего облика) блоков диаграммы и блок-диаграммы в целом.

На панель инструментов выведены следующие команды меню (рис. 3.15):

Рис. 3.15. Панель инструментов окна блок-диаграммы

1 – открыть новое (пустое) окно блок-диаграммы;
2 – открыть существующий mdl-файл;
3 – сохранить mdl-файл на диске;
4 – вывод на печать блок-диаграммы;
5...7 – команды редактирования блок-диаграммы (*вырезать*, *копировать*, *вставить*);
8 – отменить предыдущую операцию редактирования;
9 – восстановить предыдущую операцию редактирования;
10 – запуск модели на исполнение (команда *Start*); после запуска модели на изображении кнопки выводится символ ‖ и ей соответствует уже команда *Pause* (приостановить моделирование);
11 – закончить моделирование (команда *Stop*); кнопка становится доступной по истечении интервала моделирования и после выполнения команды *Pause*.

Если в состав рабочей конфигурации MATLAB включено приложение *Real-Time Workshop*, то меню дополняется разделом *Tools*, содержащим средства работы с ним.

Для версий MATLAB 5.0 и 5.1 единственным отличием окна библиотеки от окна блок-диаграммы является то, что в первом недоступен раздел меню *Simulation*.

В MATLAB 5.2 существует еще одно различие: в нижней части окна блок-диаграммы имеется строка состояния модели. Она содержит следующие поля (слева направо):

- текстовое поле состояния SIMULINK; может иметь значение *Ready* (*Готов*) или *Running* (*Выполнение*); кроме того, при «нажатой» кнопке панели инструментов в этом поле выводится ее назначение;
- индикатор степени завершенности сеанса моделирования («включается» при запуске модели);
- поле текущего значения модельного времени (выводится также только после запуска модели);
- используемый алгоритм расчета состояний модели.

Применение команд меню, не выведенных на панель инструментов, будет рассмотрено немного позже. Сейчас остановимся на описании основных возможностей SIMULINK.

Чтобы получить представление о том, что такое модель, разработанная с помощью SIMULINK, можно воспользоваться демонстрационными средствами MATLAB, о которых было сказано выше. Для открытия окна *MATLAB Demos* из

среды SIMULINK достаточно выбрать в окне *Library: Simulink* поле *Demos*, дважды щелкнув на нем ЛКМ.

При открытии окна *MATLAB Demos* из среды SIMULINK содержание demo-разделов отображается в несколько измененном виде (по сравнению с приведенным на рис. 3.11).

Пользователь имеет возможность выбрать один из следующих пунктов, входящих в раздел *Simulink-demo*:

- *New Simulink 2* – иллюстрация дополнительных возможностей версий Simulink 2.X по сравнению с версией Simulink 1, входящей в состав MATLAB 4.
- *Simple models* – примеры простых *S*-моделей.
- *Complex models* – примеры более сложных *S*-моделей.
- *Advanced Products* – вывод справки по использованию двух дополнительных модулей для работы с SIMULINK: *Accelerator* (средство повышения скорости моделирования) и RTW (*Real Time Workshop* – средство генерации кода на языке *C* из блок-диаграмм).

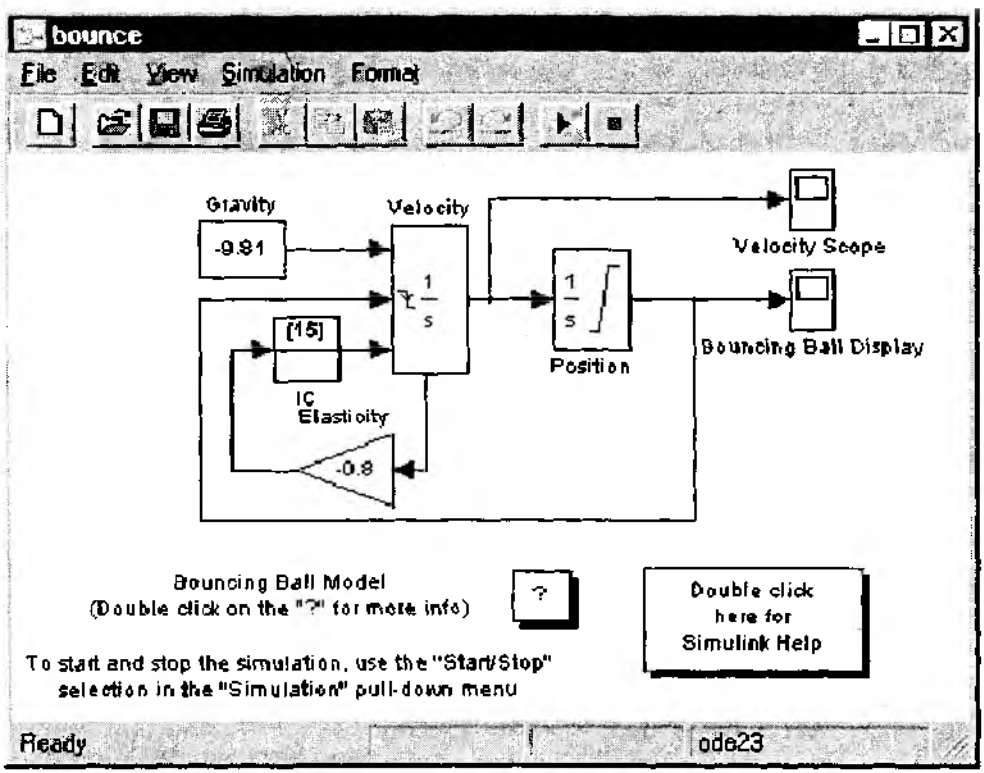


Рис. 3.16. Пример блок-диаграммы модели

При выборе одного из разделов в правой части окна выводится список входящих в него примеров *S*-моделей. В свою очередь название отмеченной в этом списке модели отображается на кнопке *Run...*, расположенной ниже. «Нажатие» этой кнопки приводит к тому, что открывается окно, содержащее блок-диаграмму выбранной модели.

Что же представляет собой *S*-модель, которой предстоит стать главным персонажем дальнейшего повествования? Чтобы составить о ней первое, пока весьма общее, впечатление, воспользуемся одним из примеров, включенных авторами MATLAB в раздел *Simple Models* (*Простые модели*). Выберите этот раздел в левом списке окна *MATLAB Demos*. Как следует из комментария, появляющегося при этом в окне справа, примеры из данного раздела помогут уяснить основные концепции SIMULINK, и если вы не были ранее с ним знакомы, начинать нужно именно отсюда. В списке примеров выберите модель *Traching a bouncing ball*, позволяющую получить траекторию прыгающего мяча. «Нажмите» кнопку с надписью *Run Traching a bouncing...* На экране появятся два окна. Первое из них содержит блок-диаграмму модели с комментариями (рис. 3.16), другое представляет собой пример одного из «смотровых окон», обеспечивающих наблюдение за поведением моделируемой системы. Это окно называется **Scope** и по виду напоминает экран электронного измерительного прибора (рис. 3.17). До начала моделирования на нем ничего нет, кроме

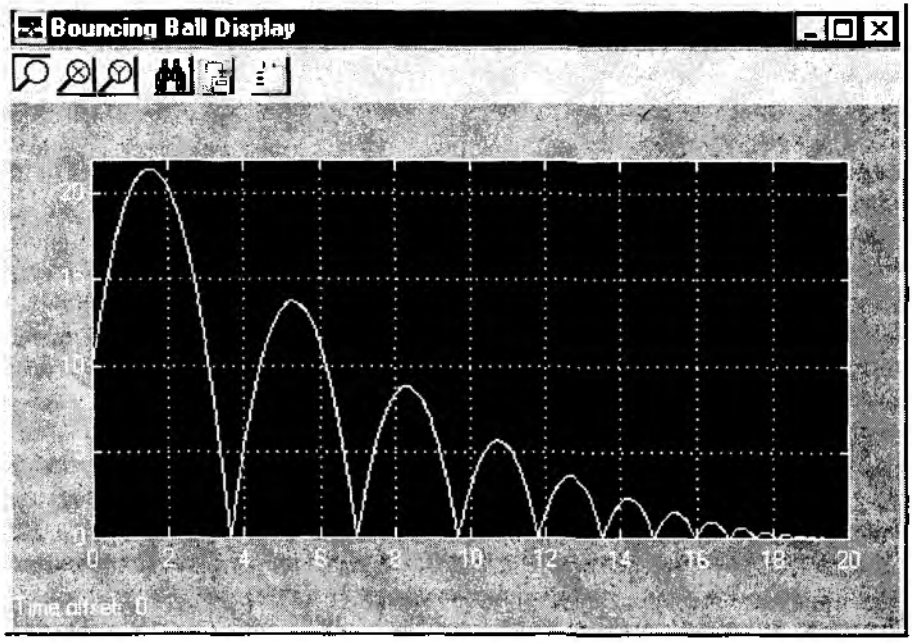


Рис. 3.17. «Смотровое окно», создаваемое блоком **Scope**

измерительной шкалы и кнопок панели управления. С точки зрения структуры *S*-модели такое «смотровое окно» — один из блоков диаграммы. Как управлять его параметрами, будет сказано немного позже.

Вернемся к окну, содержащему блок-диаграмму модели. Она представляет собой набор блоков, соединенных между собой линиями связи. Направление движения информационных и управляющих сигналов на диаграмме обозначено стрелками. Любая линия связи может иметь произвольное число ответвлений, начало каждого из которых обозначается точкой. Число входов и выходов блока определяется его типом и значениями параметров настройки блока. Для обеспечения наглядности диаграммы входящие в нее блоки не только различаются графическим представлением, но и снабжаются (при необходимости) индивидуальными именами, которые выбираются пользователем. На выбор имени не накладывается никаких ограничений, оно может представлять собой даже целую фразу.

Все блоки, входящие в блок-диаграмму, можно условно разделить на две группы: функциональные блоки и «смотровые окна».

В состав рассматриваемой *S*-модели входят два «смотровых окна» типа **Scope**. Одно из них уже упоминалось выше. Оно называется *Bouncing Ball Display* (*Демонстрация прыгающего мяча*) и открывается автоматически при открытии файла модели. Второе «смотровое окно» представлено на диаграмме блоком *Velocity Scope* (*Индикатор скорости*). Чтобы его открыть, нужно дважды щелкнуть ЛКМ на изображении блока. До начала моделирования это окно, так же, как и первое, пусто.

Остальные блоки модели являются функциональными. Двойной щелчок ЛКМ на любом из них приводит к тому, что открывается окно настройки параметров блока.

Пока будем считать, что значения параметров, заданные разработчиками модели, нас устраивают.

Следующий этап знакомства с *S*-моделью состоит в попытке провести с ней модельный эксперимент (то есть заставить ее работать).

Предварительно расположите все три окна модели (два окна **Scope** и блок-диаграмму) на экране так, чтобы они не перекрывали друг друга (при необходимости можно изменить размер каждого из них).

Сделайте активным окно с блок-диаграммой и запустите модель с помощью кнопки *Start*, расположенной на панели инструментов окна. После начала моделирования траектория движения прыгающего мяча отображается в «смотровом окне» *Bouncing Ball*, а изменение его скорости — в окне *Velocity Scope*.

Не дожидаясь, пока мяч перестанет «прыгать», выполните команду *Pause* («нажав» соответствующую кнопку на панели инструментов). При этом моделирование приостанавливается (мяч «зависает» в окне).

Во время паузы на исполнение может быть запущена другая модель или выполнены какие-либо действия в командном окне MATLAB.

При повторном «нажатии» все той же кнопки (теперь она соответствует команде *Continue* (продолжить) мяч продолжит движение. Модель закончит работу, когда мяч остановится (его скорость станет равна нулю).

В рассматриваемом примере условием окончания моделирования является наступление определенного события — остановка мяча. Но условием окончания модельного эксперимента может служить также истечение заданного интервала моделирования. SIMULINK позволяет использовать и такую возможность. Давайте попробуем ею воспользоваться.

При заданных параметрах модели (упругости мяча и начальной скорости полета) мяч перестает прыгать через 20 с. Этот отрезок времени совпадает в исходной модели с интервалом моделирования, длительность которого отображается на горизонтальной шкале окон **Scope**.

Изменим параметры движения мяча, оставив прежним интервал моделирования.

Найдите на блок-диаграмме элемент в форме треугольника с именем *Elasticity*. Он позволяет задавать значение коэффициента упругости мяча. Текущее значение этого коэффициента, равное –0.8, выводится внутри изображения блока. Очевидно, чем более упругий мяч нам попадется, тем дольше он будет прыгать. В данной модели увеличению упругости мяча соответствует увеличение абсолютного значения коэффициента *Elasticity*. Замените –0.8 на –0.9. Для этого щелкните дважды ЛКМ на изображении блока. В открывающемся окне настроек произведите соответствующие изменения (рис. 3.18).

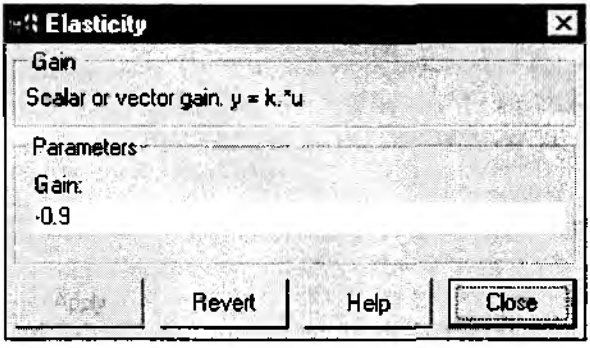


Рис. 3.18. Диалоговое окно для изменения упругости мяча

«Нажмите» последовательно кнопки *Apply* (*Применить*) и *Close* (*Закрыть*). Окно настроек закроется, новое значение коэффициента будет выведено на изображении блока. Запустите модель на исполнение командой *Start*. Для наблюдения за мячом опять воспользуемся окнами **Scope**. По истечении 20 с мяч еще полон энергии (кинетической), но... Время моделирования истекло, и работа модели остановлена. Это подтверждает и короткий звуковой сигнал, издаваемый компьютером.

На этом первое знакомство с *S*-моделью можно закончить и перейти к более детальному описанию возможностей SIMULINK.

Закройте окно с блок-диаграммой модели (без сохранения внесенных изменений). При этом «смотровые окна» закрываются автоматически. Окно *MATLAB Demo* также закройте, оставив на экране окно *Library: Simulink*, содержащее перечень основных разделов библиотеки.

3.2.2. БИБЛИОТЕКА МОДУЛЕЙ (БЛОКОВ)

Библиотека блоков SIMULINK представляет собой набор визуальных объектов, используя которые можно собирать, как из кубиков, произвольную конструкцию.

Для любого блока можно получать требуемое число копий и использовать каждую из них абсолютно автономно. Более того, практически для всех блоков существует возможность индивидуальной настройки: пользователь может изменить как внутренние параметры блоков (например, количество входов), так и внешнее оформление (размер, цвет, имя и т. д.). Подробнее эти вопросы обсуждаются в разделе 4.1.

На порядок соединения блоков друг с другом также не накладывается никаких ограничений. Конечно, при связывании блоков необходимо соблюдать определенные правила, о которых будет сказано чуть позже, однако они обусловлены в основном логикой работы самой модели, а не специальными требованиями SIMULINK.

Для удобства работы пользователя библиотека блоков разбита на семь разделов. Шесть из них являются базовыми и не могут изменяться пользователем (за исключением внешнего оформления):

- ***Sources*** (*Источники*),
- ***Sinks*** (*Получатели*),
- ***Discrete*** (*Дискретные элементы*),
- ***Linear*** (*Линейные элементы*),
- ***Nonlinear*** (*Нелинейные элементы*),
- ***Connections*** (*Соединения*).

Седьмой раздел — ***Blocksets & Toolboxes*** (*Наборы блоков и инструменты*) — содержит блоки, относящиеся к компонентам MATLAB, включенным пользователем в рабочую конфигурацию пакета. При этом для каждой компоненты создается свой подраздел библиотеки. При минимальной рабочей конфигурации, описанной в 3.1.1, в разделе *Blocksets & Toolboxes* имеется только один подраздел — *SIMULINK Extras* (*Дополнение к SIMULINK*). Этот подраздел, в свою очередь, разбит на шесть частей, три из которых являются дополнением к одноименным основным разделам библиотеки, а три других имеют самостоятельное значение (рис. 3.19). Это наборы блоков *Transformations* (блоки пересчета координат и шкал температуры), *Flip-Flops* (блоки, соответствующие основным типам триггеров) и *Linearization* (содержит единственный блок, реализующий функцию линейной апроксимации).

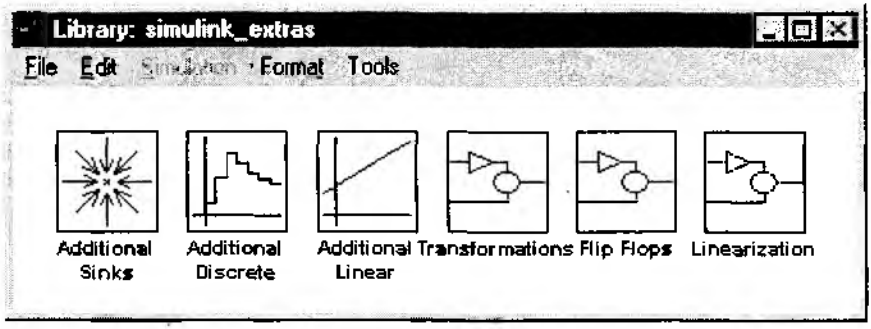


Рис. 3.19. Дополнительные разделы библиотеки

Чтобы открыть интересующий раздел библиотеки, достаточно дважды щелкнуть на нем ЛКМ.

Каждый блок, входящий в библиотеку SIMULINK, имеет по крайней мере один параметр настройки. Задавая требуемое значение параметра (или выбирая его из предлагаемого меню), пользователь имеет возможность скорректировать функцию, реализуемую данным блоком. Чтобы открыть окно настройки параметров, нужно дважды щелкнуть ЛКМ на изображении блока. Однако возможность изменять значения параметров появляется только после того, как блок будет помещен в окно блок-диаграммы.

Окна настройки параметров всех библиотечных блоков имеют идентичную структуру и содержат краткую характеристику блока, поля ввода (или выбора) значений параметров блока и 4 кнопки:

- *Apply* – применить;
- *Revert* – вернуть предыдущее значение параметров;
- *Help* – вызов файла помощи в формате html;
- *Close* – закрыть окно настроек.

Измененные значения параметров вступают в силу после «нажатия» кнопки *Apply*. Чтобы запустить модель на исполнение с новыми параметрами, закрывать окно настроек не обязательно.

Вернемся к основным разделам библиотеки SIMULINK. Как уже было сказано, их шесть.

Прежде чем перейти к описанию состава и назначения блоков, входящих в каждый из разделов, необходимо сделать следующее замечание.

Тот аспект визуального моделирования, который рассматривается в данной книге, далеко не исчерпывает всех возможностей SIMULINK. В частности, многие его блоки (а также используемая разработчиками SIMULINK терминология) ориентированы на моделирование систем автоматического управления и регулирования. Поэтому в дальнейшем основное внимание будет уделено тем блокам, которые необходимы для реализации подходов, изложенных в Части I книги.

Раздел *Sources* (*Источники*)

В окне *Library: Simulink* найдите раздел *Sources* и откройте его, дважды щелкнув на его изображении ЛКМ.

Блоки, входящие в этот раздел, предназначены для формирования сигналов, обеспечивающих управление работой *S*-модели в целом или отдельных ее частей. Все блоки-источники имеют по одному выходу и не имеют входов (рис. 3.20).

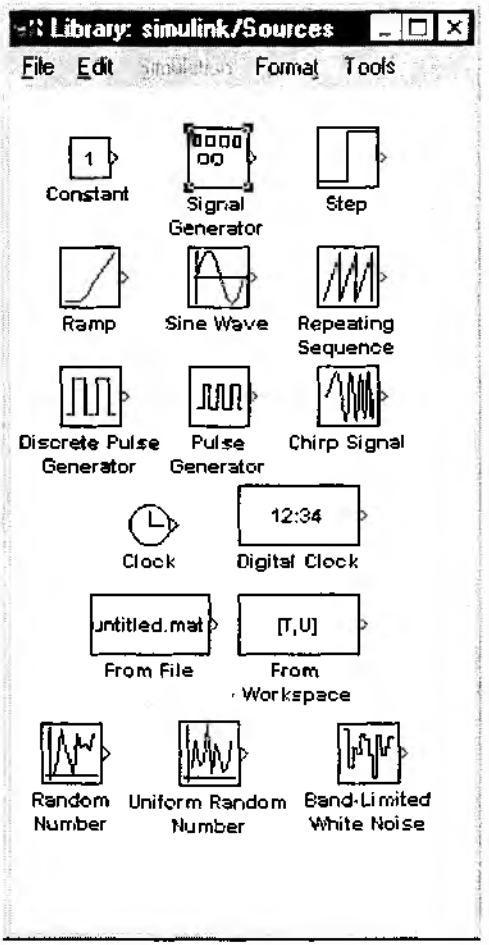


Рис. 3.20. Раздел *Source* (*Источники*)

Замечание. Терминология, используемая авторами SIMULINK для описания блоков этого и других разделов библиотеки, говорит о том, что в основном имеют-

ся в виду электрические сигналы. Тем не менее физическая интерпретация понятия «сигнал» в каждом конкретном случае различна и определяется в первую очередь физической природой моделируемой системы.

Итак, в качестве источников сигналов (входных величин) могут использоваться следующие блоки:

Constant — формирует постоянную величину (скаляр, вектор или матрицу);

Signal Generator — создает непрерывный сигнал произвольной формы;

Step — генерирует единичный дискретный сигнал с заданными параметрами;

Ramp — создает линейно возрастающий (убывающий) сигнал;

Sine Wave — генератор гармонических колебаний;

Discrete Pulse Generator — генератор дискретных импульсных сигналов;

Chirp Signal — генератор гармонических колебаний переменной частоты;

Clock — источник непрерывного временного сигнала;

Digital clock — формирует дискретный временной сигнал;

Random Number — источник дискретного сигнала, амплитуда которого является случайной величиной, распределенной по нормальному закону;

Uniform Random Number — источник дискретного сигнала, амплитуда которого является равномерно распределенной случайной величиной;

Band-Limited White Noise — генератор «белого шума» с ограниченной полосой.

Следующие два блока из раздела *Источники* отличаются от перечисленных тем, что обеспечивают использование в модели различных числовых данных, полученных ранее как с помощью SIMULINK, так и другими средствами MATLAB.

Первый из них — **From File** — предназначен для ввода в *S*-модель данных, хранящихся в MAT-файле.

Второй — **From Workspace** — обеспечивает ввод в модель данных непосредственно из рабочей области MATLAB.

Структура данных в MAT-файле представляет собой многомерный массив с переменным числом строк, которое определяется числом регистрируемых переменных. Элементы первой строки содержат последовательные значения модельного времени, элементы в других строках — соответствующие им значения переменных.

Как и другие библиотечные блоки, блоки — «источники» могут настраиваться пользователем (за исключением блока *Clock*, работа которого основана на использовании аппаратного таймера компьютера).

Рассмотрим особенности настройки тех блоков, которые понадобятся нам в дальнейшем при разработке собственных *S*-моделей.

Блок **Step**. Он обеспечивает формирование единичного управляющего сигнала, который может использоваться в любой точке модели. Блок имеет 3 параметра настойки (рис. 3.21):

- ***Step time*** (*Шаг времени*) — определяет длительность сигнала, исходное значение параметра равно 1.

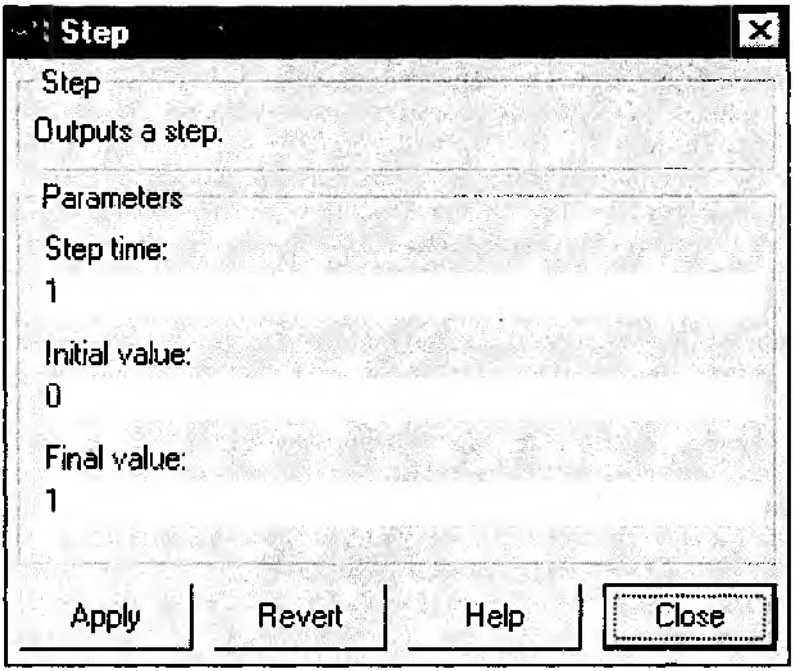


Рис. 3.21. Окно настроек блока **Step**

- ***Initial value*** (*Начальное значение*) — задает значение амплитуды сигнала в начальный момент времени; исходное значение — 0.
- ***Final value*** (*Конечное значение*) — задает значение амплитуды сигнала по истечении шага моделирования; исходное значение этого параметра равно 1.

Чтобы увидеть, что представляет собой сигнал, создаваемый блоком **Step**, и как на него влияют значения указанных параметров, воспользуемся уже знакомым нам окном **Scope**.

Последовательность действий для достижения поставленной цели такова.

1. В разделе *File* командного окна MATLAB выберите последовательность команд *New* → *Model*. Откроется (или станет активным) пустое окно для создания новой блок-диаграммы.

2. В разделе библиотеки *Источники* (он открыт) найдите блок **Step**, нажмите ЛКМ и, не отпуская ее, перстащите контур блока на свободное поле окна *untitled*. Отпустите ЛКМ. В окне появится блок **Step** — первый блок будущей диаграммы.

3. В окне библиотеки SIMULINKa откройте раздел *Sinks* (*Получатели*), дважды щелкнув на его изображении ЛКМ. Найдите в нем блок **Scope** и, нажав ЛКМ, перетащите его контур на свободное поле окна *untitled*. Отпустите ЛКМ. В окне появится блок **Scope**.

4. Для удобного соединения блоков расположите их так, чтобы вход блока **Scope** находился напротив выхода блока **Step** (на любом расстоянии).

5. Подведите курсор мыши к выходу блока **Step**. Курсор примет форму крестика. Нажмите ЛКМ и, не отпуская ее, переместите курсор к входу блока **Scope**. Как только крестик станет двойным, отпустите кнопку мыши. Между блоками образуется линия связи со стрелкой, указывающей направление передачи сигнала (рис. 3.22).

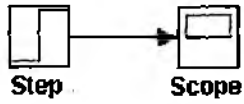


Рис. 3.22. Подключение блока **Scope**

6. Откройте окно **Scope**, дважды щелкнув на нем ЛКМ. Теперь можно «оживить» полученную блок-диаграмму, состоящую из двух элементов. Для этого, как и при наблюдении за «прыгающим мячом», нужно выполнить команду *Start*. Сформированный сигнал отображается в окне **Scope** (рис. 3.23). Из рисунка видно, что при исходных значениях параметров блока **Step** величину *Step time* можно интерпретировать как период задержки сигнала.

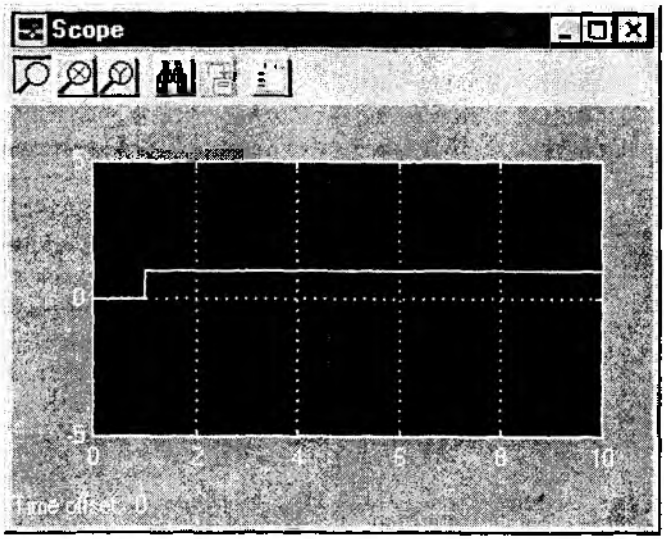


Рис. 3.23. Временная диаграмма работы блока **Step** со стандартными параметрами

Замените значение параметра *Initial value* на «1», а параметра *Final value* — на «0» и повторите моделирование. Форма сигнала изменилась (рис. 3.24). Теперь величина *Step time* определяет длительность управляющего сигнала. Именно та-

кой подход к использованию блока **Step** оказывается более удобным при разработке многих *S*-моделей.

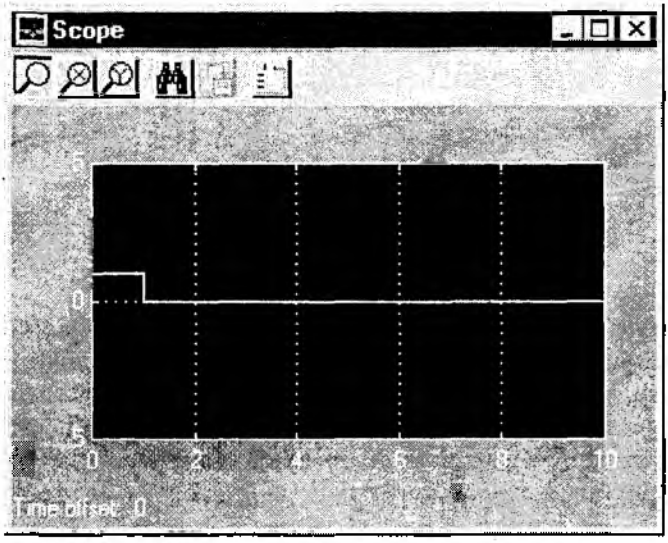


Рис. 3.24. Временная диаграмма работы блока **Step** с измененными параметрами

Необходимо отметить, что значения всех трех параметров блока могут задаваться не только в форме констант, но и в виде вычисляемых выражений, в том числе содержащих функции. Подробнее этот вопрос обсуждается в п. 4.3.

Блок **Discrete Pulse Generator**. Отыщите его в разделе *Источники* и с помощью двойного щелчка откройте окно настроек блока. Оно позволяет изменять значения пяти параметров (рис. 3.25):

- амплитуды сигнала (***Amplitude***),
- величины периода сигнала (***Period***),
- ширины импульса (***Pulse width***),
- величины задержки сигнала (***Phase delay***),
- шаг изменения модельного времени (***Sample time***).

Значения параметров 2...4 должны задаваться как целое число шагов модельного времени (*number of samples*). Поэтому выбор значений параметров блока целесообразно начинать с выбора величины *Sample time*. Размер шага можно указать как в форме константы, так и в форме вычисляемого выражения. Если вычисленное значение является дробным, оно округляется до целого. Аналогично может быть задано и значение амплитуды, но в случае дробной величины округление не производится.

Блок **Random Number** обеспечивает формирование сигналов, амплитуда которых является случайной величиной, распределенной по нормальному закону

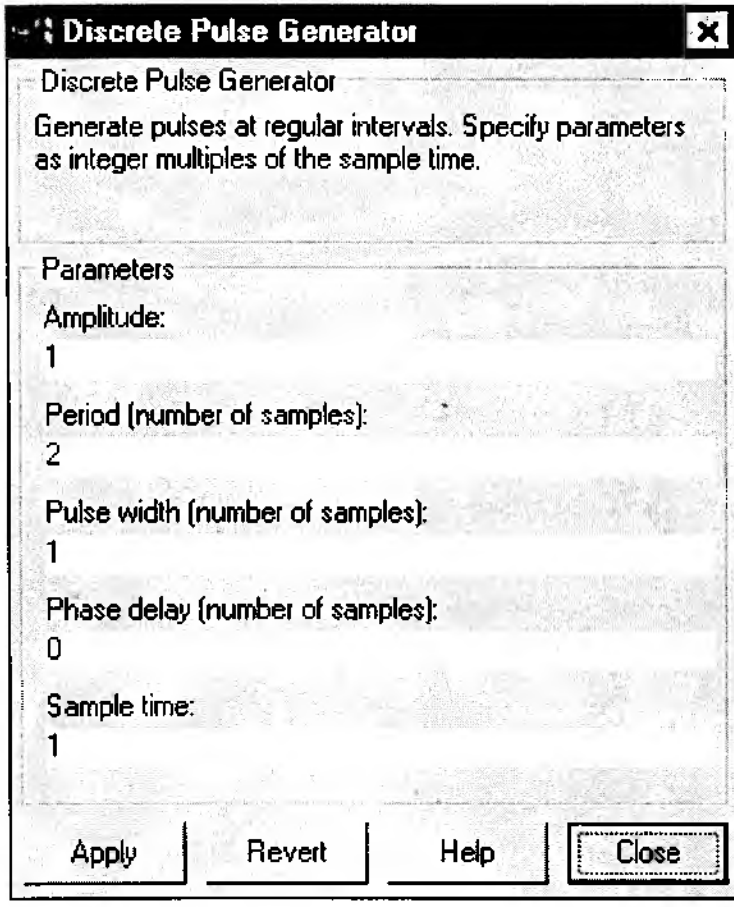


Рис. 3.25. Окно настроек блока **Discrete Pulse Generator**

с заданными параметрами. Блок имеет четыре параметра настройки (рис. 3.26). Первые два — ***Mean*** и ***Variance*** — являются параметрами нормального закона (среднее и дисперсия), третий — ***Initial seed*** — задает начальное значение для инициализации генератора последовательности случайных чисел. При фиксированном значении этого параметра генератор всегда вырабатывает одну и ту же последовательность СЧ. Четвертый параметр (***Sample time***) используется так же, как и в рассмотренных ранее блоках.

Блок **Uniform Random Number** обеспечивает формирование сигналов, амплитуда которых является случайной величиной, распределенной равномерно в заданном интервале.

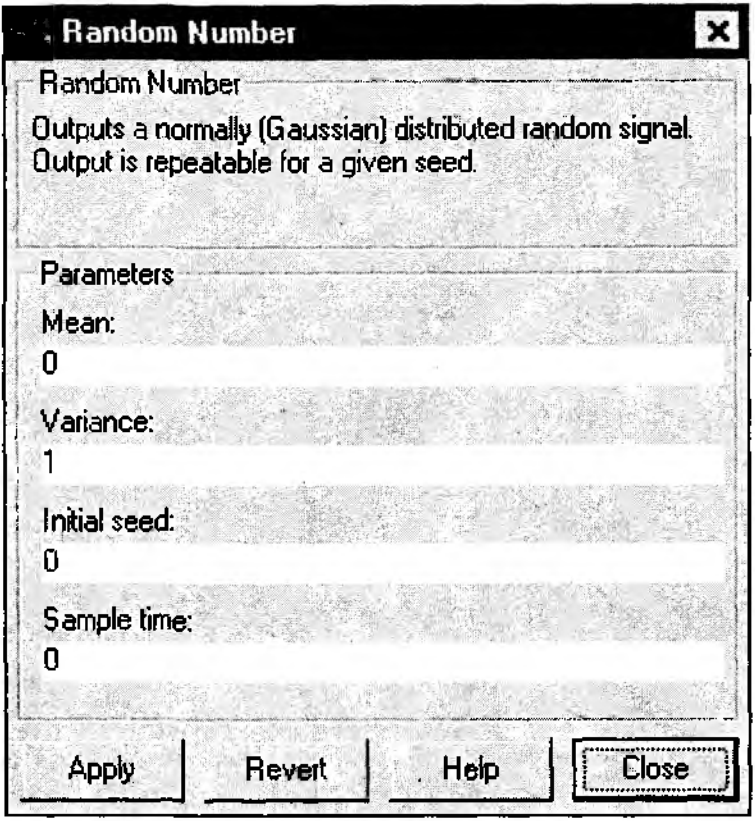


Рис. 3.26. Окно настроек блока **Random Number**

В блоке имеются такие же параметры настройки ***Initial seed*** и ***Sample time***, как и в **Random Number**. Еще два параметра (***Minimum*** и ***Maximum***) задают диапазон распределения СВ.

Блок **Digital Clock**. Этот блок имеет единственный параметр настройки — величину шага изменения модельного времени (***Sample time***). Величина шага задается таким же образом, как и в блоке **Discrete Pulse Generator**. Если величина шага задана выражением, то оно вычисляется только однажды, и полученное значение используется на всем интервале моделирования. Особенность блока **Digital Clock** состоит в том, что он не только формирует величину шага, но и вычисляет новое значение модельного времени, которое используется для проверки условия окончания моделирования. Очередное значение модельного времени вычисляется как сумма предыдущего значения и величины шага моделирования.

Блок **From File** имеет в качестве параметра настройки имя MAT-файла, из которого будут считываться требуемые данные. Если указывается только имя фай-

ла, то поиск файла производится в открытой папке. При необходимости можно указать полный путь доступа к файлу. После закрытия окна настроек имя файла выводится на изображении блока. Немного забегая вперед отметим, что в разделе библиотеки *Получатели* имеется блок **To File**, который выполняет запись результатов моделирования в MAT-файл.

Блок **From Workspace**, как было сказано ранее, обеспечивает использование в *S*-модели данных, хранящихся в рабочей области (собственной оперативной памяти) MATLAB. Блок имеет один составной параметр настройки, который представляет собой упорядоченный список используемых данных, заключенный в квадратные скобки (рис. 3.27).

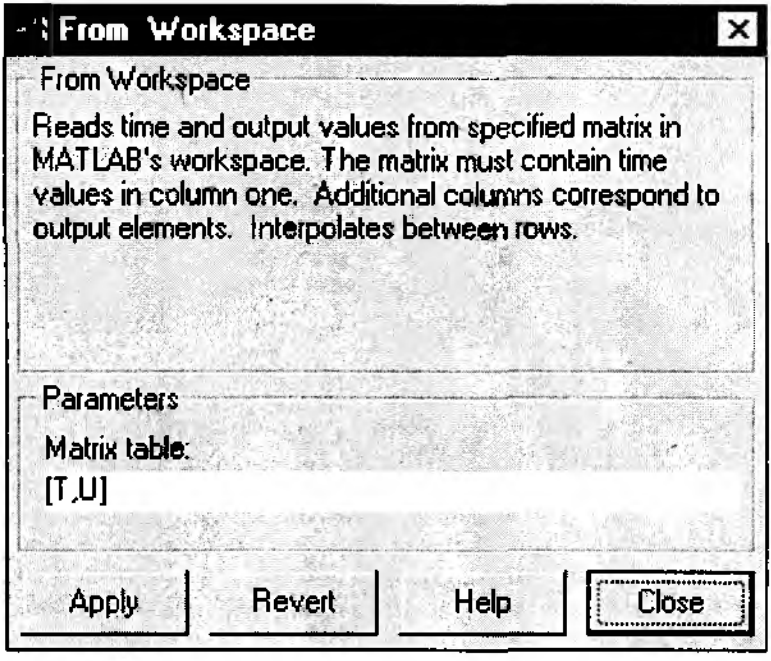


Рис. 3.27. Окно настроек блока **From Workspace**

Список должен содержать не менее двух элементов, первым из которых является вектор значений модельного времени с именем *tout*. Остальные элементы списка представляют собой векторы значений вводимых из рабочей области величин. Длина каждого из этих векторов должна быть равна длине вектора *tout*. Как правило, в качестве элементов списка данных используются величины, помещенные в рабочую область MATLAB с помощью блока **To Workspace**, входящего в раздел библиотеки *Sinks* (*Получатели*).

Раздел *Sinks* (*Получатели*)

Напомним, что раздел библиотеки *Sinks* мы уже открывали, чтобы «достать» из него блок **Scope**.

Блоки, собранные в этом разделе, достаточно существенно различаются по функциональному предназначению (рис. 3.28).

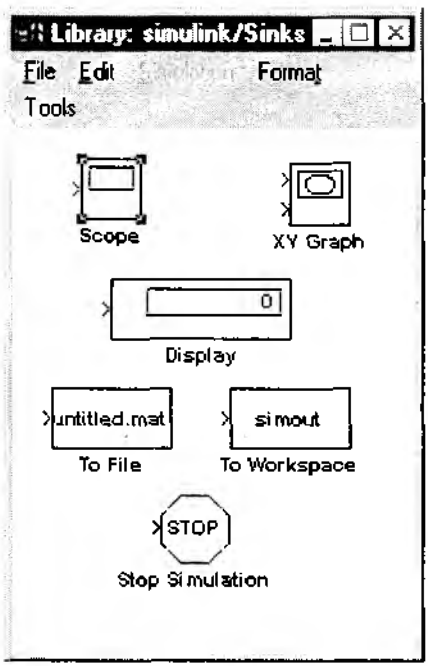


Рис. 3.28. Раздел библиотеки *Sinks*

Условно их можно разделить на три вида:

1) блоки, используемые при моделировании в качестве «смотровых окон». К ним относятся:

блок **Scope**, который уже упоминался выше;

блок **XYGraph**, обеспечивающий создание двумерных графиков в прямоугольной системе координат;

блок **Display**, предназначенный для отображения численных значений величин;

2) блоки, обеспечивающие сохранение промежуточных и/или выходных результатов моделирования:

блок **To File;**

блок **To Workspace;**

3) блок управления моделированием — **Stop Simulation**, который позволяет прервать моделирование при выполнении тех или иных условий. Блок

срабатывает в том случае, если на его вход поступает ненулевой сигнал. Поскольку этот блок не имеет параметров настройки, то мы к нему вернемся только в разделе 4.3, где речь пойдет о средствах управления моделированием. Особенности использования остальных блоков раздела рассмотрим подробнее уже сейчас.

Блок **Scope**. Напомним, что этот блок позволяет в процессе моделирования наблюдать динамику изменения интересующих исследователя характеристик системы. Создаваемое с его помощью «смотровое окно» напоминает экран измерительного прибора. Открыть окно **Scope** можно только после того, как блок помещен на поле блок-диаграммы (щелкнув дважды на его изображении ЛКМ). Размер и пропорции окна можно изменять произвольно, используя курсор мыши.

По оси ординат шкалы измерений откладываются значения наблюдаемой величины, по оси абсцисс — значения модельного времени. По умолчанию для оси ординат используется диапазон [–5; 5], а для оси модельного времени — [0; 10].

Блок-диаграмма может быть построена таким образом, чтобы на вход блока **Scope** поступала векторная величина. В этом случае для каждого элемента вектора в окне строится отдельная кривая, отражающая динамику его изменения. Выводимые кривые различаются цветом, который устанавливается автоматически. Одновременно в окне **Scope** может отображаться до 30 кривых.

Для управления параметрами окна **Scope** в нем имеется панель меню, содержащая семь кнопок (рис. 3.29):

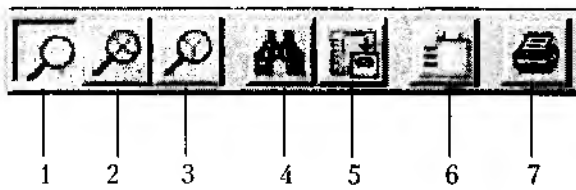


Рис. 3.29. Панель инструментов блока **Scope**

1 — изменение масштаба осей графика;
2 — изменение масштаба по оси абсцисс;
3 — изменение масштаба по оси ординат;
4 — автоматическая установка оптимального масштаба осей (автошкалирование);
5 — сохранение установленного масштаба осей;
6 — вызов диалогового окна настройки параметров блока **Scope**;
7 — печать содержимого окна **Scope**.

Кнопки 1...3 являются альтернативными, т. е. в каждый момент времени может быть «нажата» только одна из них.

Для изменения масштаба по выбранной оси координат необходимо:

- «нажать» соответствующую кнопку изменения масштаба;
- подвести курсор мыши к тому участку графика, который должен быть отображен в новом масштабе;
- нажать (один раз!) ЛКМ.

Первое нажатие ЛКМ приводит к четырехкратному увеличению масштаба, каждое последующее дает увеличение масштаба в два раза.

При «нажатии» кнопки 6 открывается окно *Properties: Scope*, содержащее две вкладки (рис. 3.30):

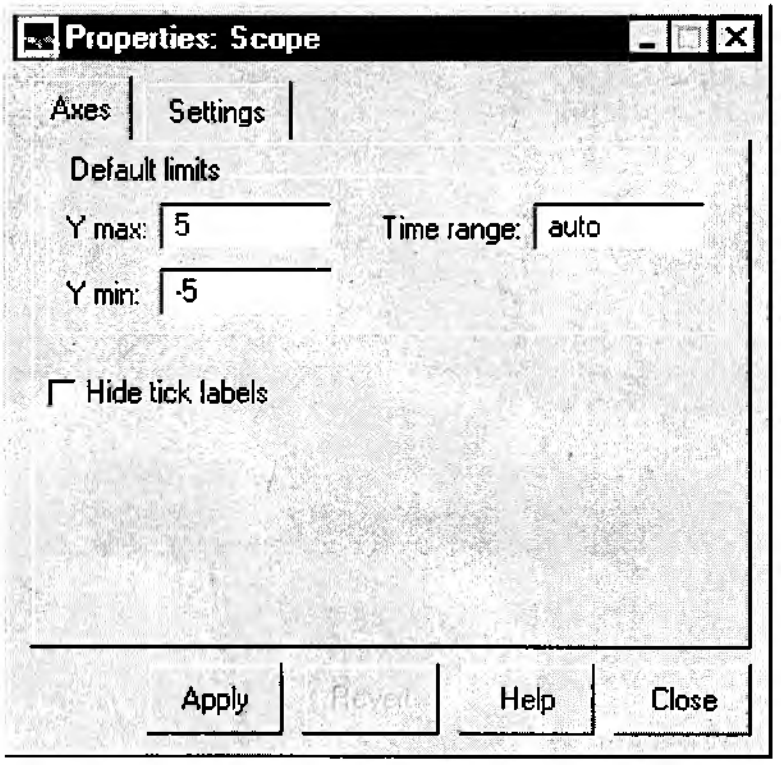


Рис. 3.30. Окно настроек блока **Scope** (вкладка *Axes*)

- ***Axes*** (*оси*), позволяющая устанавливать параметры осей графика;
- ***Setting*** (*установки*), предназначенная для ввода значений дополнительных параметров блока **Scope**.

В нижней части окна расположены кнопки, аналогичные кнопкам окон настройки параметров других библиотечных блоков: *Apply* (*Применить*); *Revert* (*Вернуть*

исходные значения); *Help* (*Вызов помощи* в формате HTML); *Close* (*Закрыть окно*).

На вкладке **Axes** имеются поле *Default limits* (*Диапазоны осей*) и флажок *Hide tick label* (*Скрыть обозначение осей*).

В поле *Default limits* устанавливаются верхняя (*Y max*) и нижняя (*Y min*) границы оси ординат, а также наибольшее отображаемое значение модельного времени на оси абсцисс (*Time range*).

Относительно оси времени необходимо сделать дополнительное пояснение.

Если величина заданного интервала моделирования не превышает значение *Time range* (и, следовательно, весь процесс «умещается» в окне **Scope**), то под графиком в строке *Time offset:* выводится 0.

Если же интервал моделирования превышает значение *Time range*, то в окне отображается только отрезок времени, равный $T_м - n \times$(*Time range*), где $T_м$ — длительность интервала моделирования, n — целое число. При этом в строке *Time offset* выводится величина «скрытого» интервала времени (длиной $n \times$(*Time range*)).

Например, если значение *Time range* равно 7, а длительность интервала моделирования составляет 16 единиц времени, то в окне **Scope** будет выведен график моделируемого процесса за последние 2 единицы времени, а строка под графиком будет иметь вид: *Time offset: 14*.

Флажок *Hide tick label* позволяет изменить форму вывода графика в окне **Scope**. Если он установлен, то оси графика не отображаются, и график занимает всю рабочую область окна.

Чтобы установленные значения параметров вступили в силу, необходимо «нажать» кнопку *Apply*, расположенную в нижней части окна *Properties*.

На вкладке ***Setting*** имеются следующие поля (рис. 3.31):

1. Поле *General* обеспечивает выбор дискретности измерения отображаемых величин (характеристик системы и модельного времени).

Установка дискретности измерения характеристик системы и времени выполняется раздельно. Выбор производится с помощью выпадающего меню, содержащего два пункта:

Decimation — установка дискретности измерения характеристик системы;

Sample time — установка дискретности измерения модельного времени.

Для ввода требуемых значений используется строка редактирования, расположенная справа от меню.

По умолчанию для измеряемых характеристик дискретность равна 1, а для модельного времени — нулю. Это означает, что значение наблюдаемой характеристики измеряется на каждом шаге моделирования, а модельное время считается непрерывным. Если параметр *Decimation* установить равным, например, 3, то значение исследуемой характеристики будет определяться только 1 раз в течение трех шагов моделирования. Параметр *Decimation* может принимать только целочисленные значения. Дискретность измерения модельного времени — это фактически величина шага моделирования. По умолчанию она равна 0,02. Для дискретных систем величина шага может быть задана либо в виде положительной константы, либо

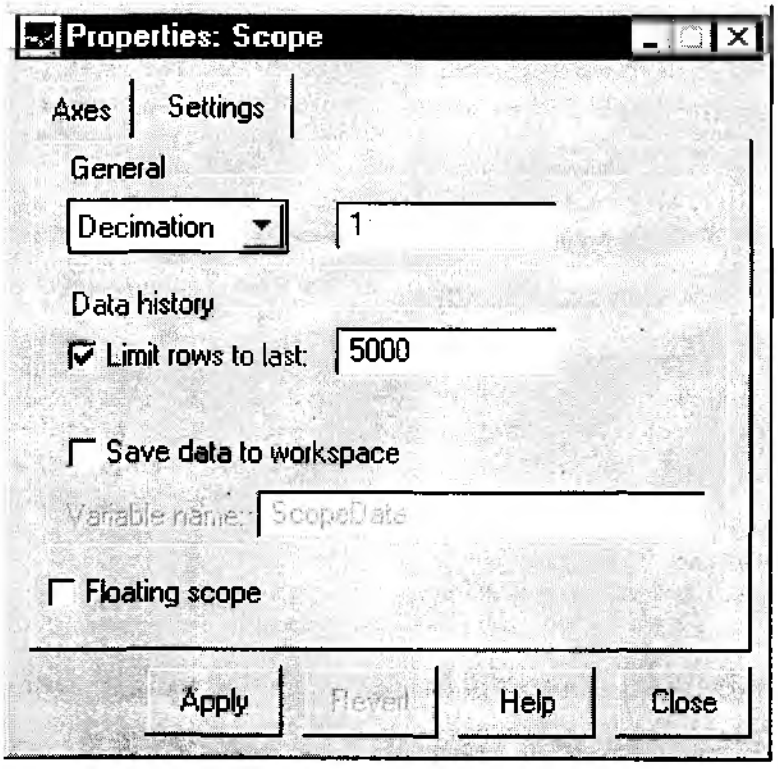


Рис. 3.31. Вкладка ***Settings*** окна настроек блока **Scope**

в форме вычисляемого выражения. Дискретность изменения модельного времени определяет момент окончания вывода данных в окно **Scope**: если очередной интервал заканчивается за пределами окна, то данные для него не выводятся.

Замечание. Параметр *Sample time* имеется практически во всех библиотечных блоках SIMULINK. Для каждого блока значение этого параметра устанавливается индивидуально. Если блок реализует некоторую функцию, то параметр *Sample time* определяет дискретность вычислений. Поэтому для корректной работы модели необходимо согласовывать установку параметра для взаимосвязанных блоков.

2. Поле *Data history* позволяет задавать максимальный объем и способ хранения отображаемых в окне данных. Объем сохраняемых данных (*Limit rows to last*) вводится в строке редактирования. Способ хранения указывается с помощью флажка *Save data to workspace*: если он установлен, то отображаемые в окне **Scope** данные сохраняются в рабочей области MATLAB в виде матрицы, аналогичной по структуре MAT-файлу. Имя матрицы указывается в строке редактирования (по умолчанию – *ScopeData*).

3. Флажок *Floating Scope* предназначен для изменения способа использования блока **Scope** в блок-диаграмме. При установленном флажке **Scope** отображается как блок без входа, и если он был связан по входу с другими блоками, то эти связи «обрываются».

Блок **XYGraph**. Этот блок также относится к «смотровым окнам». Он представляет собой упрощенный вариант блока **Scope** и обеспечивает построение графиков зависимостей произвольных величин, фигурирующих в модели. Блок имеет два входа, первый из которых предназначен для ввода аргумента, второй — для ввода значений функции этого аргумента.

На рис. 3.32 показан внешний вид окна **XYGraph**, содержащего график функции $sqrt(t)$, где t — текущее значение модельного времени. Окно **XYGraph** открывается автоматически при запуске модели. Оно имеет собственное меню, содержащее 4 раздела:

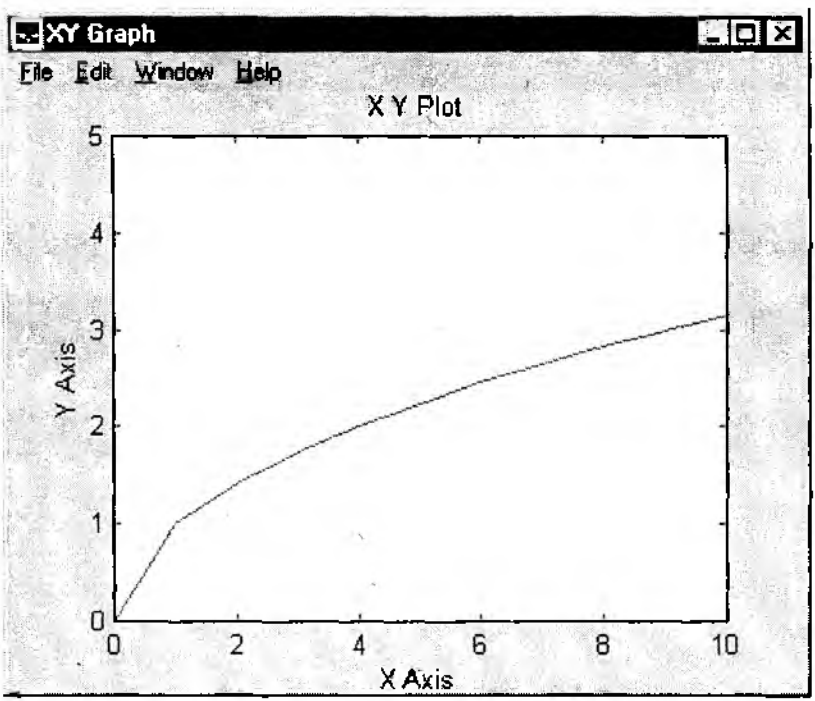


Рис. 3.32. Внешний вид окна **XYGraph**

- раздел ***File*** включает стандартные команды работы с файлами, а также опции вывода графиков на печать;
- раздел ***Edit*** содержит единственную доступную команду — *Copy Figure*, которая позволяет копировать содержимое окна ***XYGraph*** в буфер обмена;

• содержание разделов *Window* и *Help* совпадает с содержанием одноименных разделов меню командного окна MATLAB.

Блок **XYGraph** имеет следующие параметры настройки (рис. 3.33):

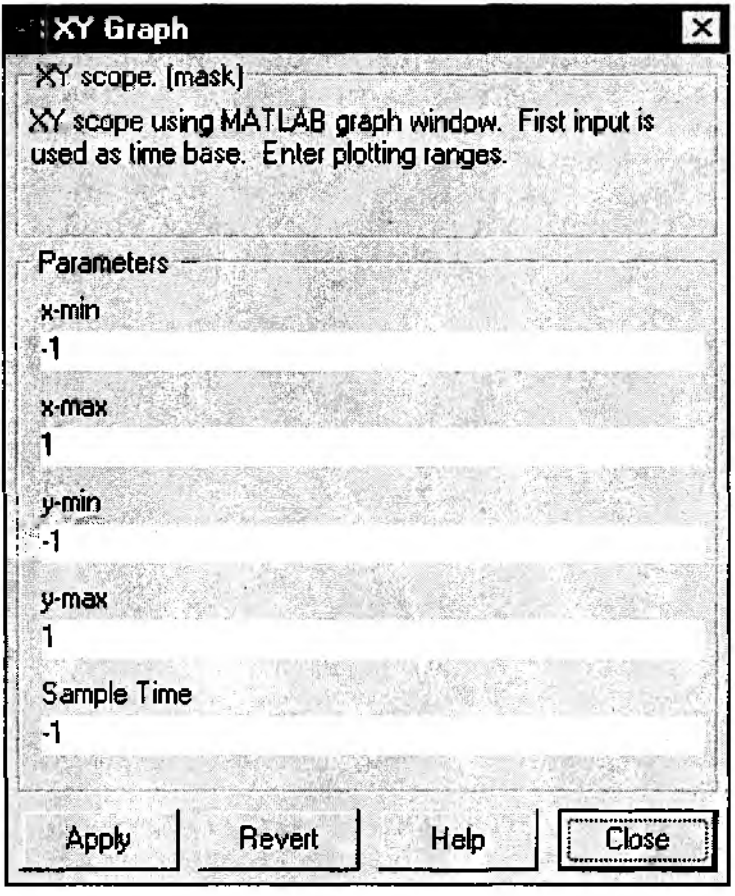


Рис. 3.33. Окно настроек блока **XYGraph**

• диапазоны осей графика (***X-min***, ***X-max*** – для оси абсцисс и ***Y-min***, ***Y-max*** – для оси ординат);

• шаг модельного времени (***Sample time***), по умолчанию его значение равно –1. Это означает, что величина шага совпадает с установленной для модели в целом (либо со значением одноименного параметра предшествующего блока).

Блок **Display**. Он предназначен для вывода на экран численных значений величин, фигурирующих в модели. Блок имеет 4 параметра настройки (рис. 3.34).

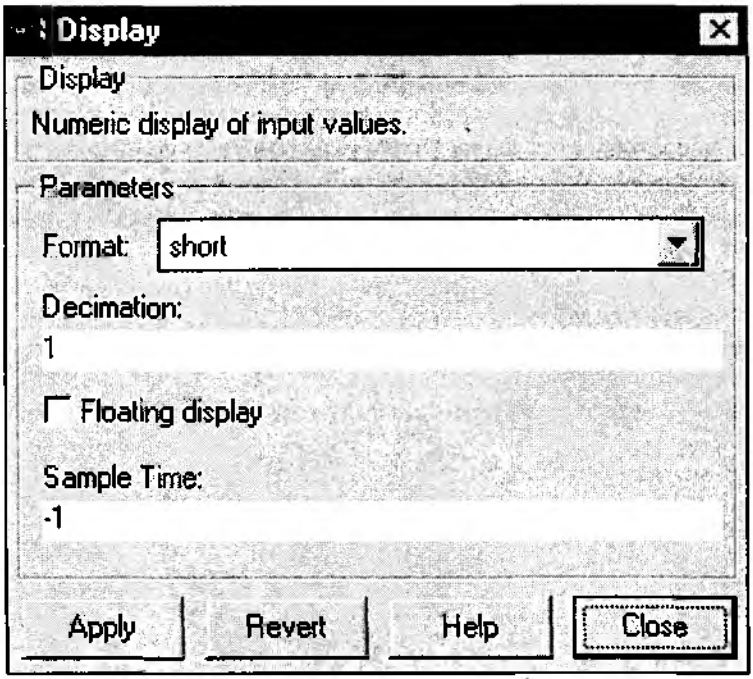


Рис. 3.34. Окно настроек блока **Display**

Первый — ***Format*** — задает формат вывода; формат выбирается с помощью выпадающего меню, содержащего 5 пунктов: *short*, *long*, *short_e*, *long_e*, *bank*. Предлагаемые форматы вывода аналогичны форматам, используемым в командном окне MATLAB. Следующие два параметра используются так же, как и одноименные параметры блока **Scope**:

Decimation — определяет периодичность вывода значений в окне **Display**;

Переключатель ***Floating display*** позволяет указывать способ использования блока **Display** в блок-диаграмме;

Поле ***Sample Time*** задает величину шага модельного времени, т. е. дискретность вывода данных в окно **Display**.

Блок **Display** может использоваться для вывода как скалярных, так и векторных данных. Если отображаемая величина является вектором, то исходный формат блока изменяется автоматически. Об изменении формата говорит маленький черный треугольник, появляющийся в нижнем правом углу блока. Для каждого элемента вектора создается свое мини-окно, но чтобы они стали видимы, необходимо «растянуть» изображение блока. Для этого следует выделить блок, подвести курсор мыши к одному из его углов (курсор при этом примет форму двойной стрелки), нажать ЛКМ, и, не отпуская ее, растянуть изображение блока. После того как

ЛКМ будет отпущена, на экране появятся дополнительные окна с выведенными в них значениями элементов вектора. Если хотя бы один элемент вектора остался «за кадром», на изображении блока **Display** по-прежнему присутствует черный треугольник. В этом случае операцию «растягивания» блока следует повторить. Убедитесь в возможностях блока ***Display*** самостоятельно, используя в качестве источника сигнала блок **Constant**.

Последовательность действий при этом должна быть почти такой же, как при знакомстве с блоком **Scope**:

- откройте раздел библиотеки *Sources* (*Источники*) и перенесите из него на свободное поле новой *S*-модели блок **Constant**;
- из раздела *Sinks* (*Получатели*) «перетащите» изображение блока **Display** и поместите его рядом с **Constant**;
- перемещая курсор от выхода **Constant** к выходу **Display**, соедините их между собой;
- выполните команду *Start*.

После завершения работы «модели» в окне ***Display*** появится цифра «1» (рис. 3.35).

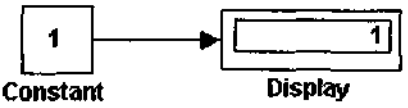


Рис. 3.35. Пример использования блока **Display**

Чтобы получить изображение элементов вектора, откройте окно настроек блока **Constant** и введите в качестве нового значения любую последовательность чисел, заключенную в квадратные скобки (числа должны быть разделены пробелами или запятыми). После «нажатия» кнопки *Apply* (*применить*) опять выберите команду *Start*. После завершения работы модели на изображении блока **Display** появится упоминавшийся выше черный треугольничек. «Растяните» блок и убедитесь, что значения вектора *Constant* выведены правильно (рис. 3.36).

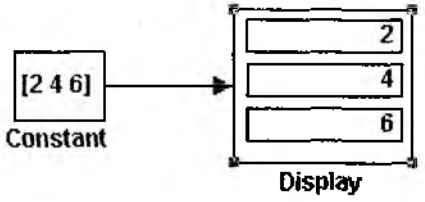


Рис. 3.36 Вывод с помощью блока **Display** векторных величин

Блок **To File**. Как уже было сказано, этот блок используется в паре с блоком **From File** из раздела *Sources*.

Он обеспечивает запись в MAT-файл данных, полученных в ходе моделирования. Блок имеет следующие параметры настройки (рис. 3.37):

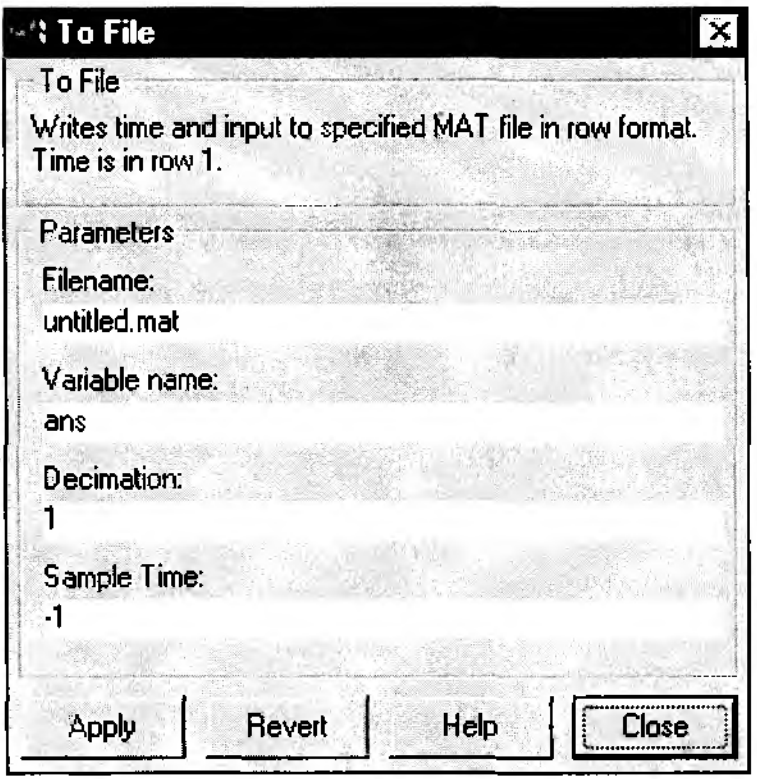


Рис. 3.37. Окно настроек блока **To File**

File name — имя MAT-файла, в который будут записываться данные (может быть указан полный путь доступа к файлу, по умолчанию — *untitled.mat*); имя файла выводится на изображении блока в блок-диаграмме;

Variable name — имя переменной, по которому можно обращаться к данным, записанным в файле (для того, например, чтобы просмотреть или изменить их в командном окне MATLAB); по умолчанию используется стандартное имя ***ans***;

Decimation — дискретность записи данных в файл; при *Decimation*=1 запись производится на каждом шаге моделирования; параметр может принимать только целочисленные значения;

Sample Time — величина шага моделирования для данного блока.

Блок **To Workspace.** Он также имеет «пару» из раздела *Sources* – блок **From Workspace** и предназначен для сохранения данных, полученных в процессе моделирования, в рабочей области MATLAB. Данные сохраняются в виде матрицы, структура которой отличается от структуры данных в MAT-файле тем, что:

- значения сохраняемых величин расположены по строкам, а не по столбцам;
- не регистрируются значения модельного времени.

Блок имеет 4 параметра настройки (рис. 3.38):

Variable name – имя, под которым данные сохраняются в рабочей области (по умолчанию – *Simout*);

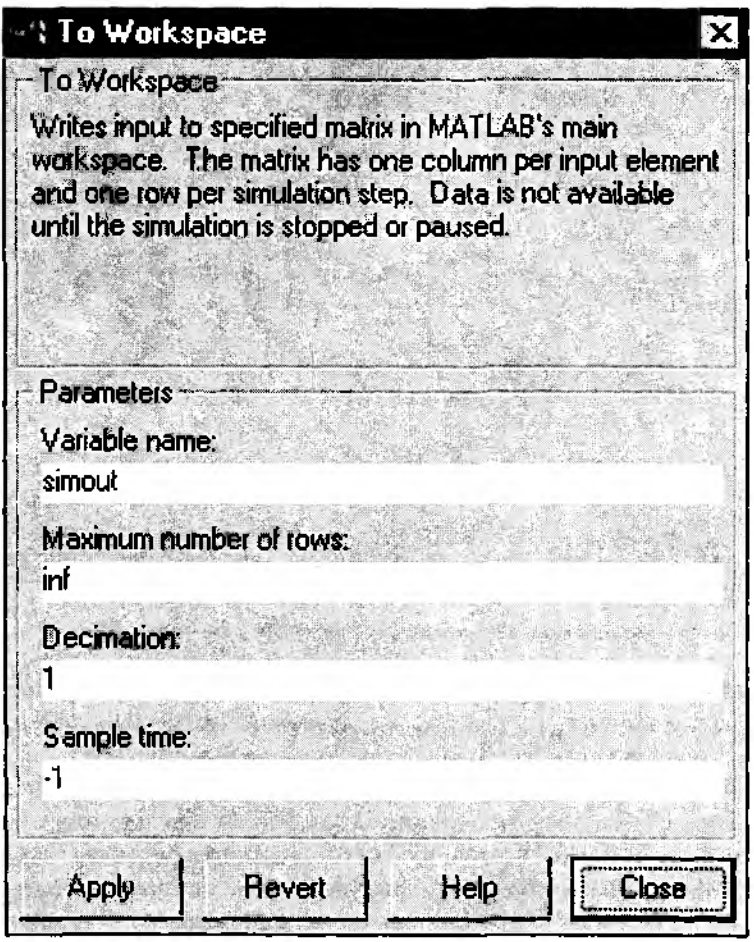


Рис. 3.38. Окно настроек блока **To Workspace**

Maximum number of rows (*Максимальное количество строк*) – предельно допустимое число шагов моделирования, для которого регистрируются данные (по умолчанию задается константой *inf*, то есть данные регистрируются на всем интервале моделирования);

Decimation – дискретность регистрации данных;

Sample time – величина шага (дискретность изменения) модельного времени.

Два последних параметра имеют тот же смысл, что и одноименные параметры блока **To File**.

Раздел *Discrete* (*Дискретные элементы*)

В этот раздел входят блоки, с помощью которых в модели может быть описано поведение дискретных систем. Напомним, что различают два основных типа та-

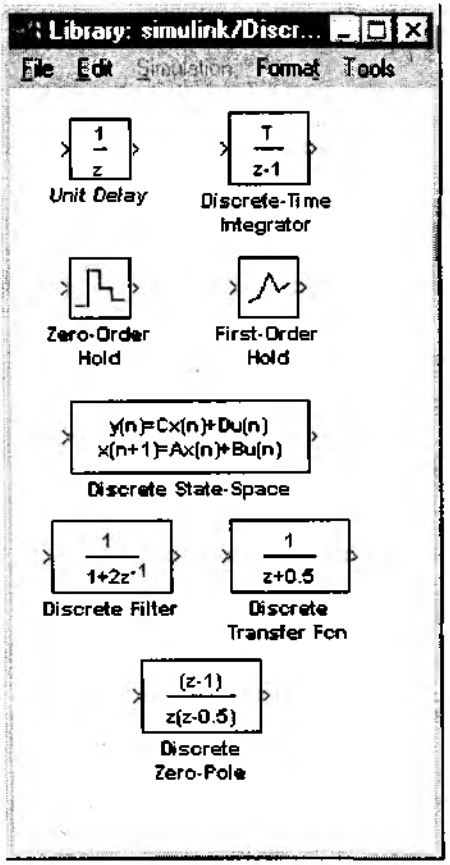


Рис. 3.39. Раздел библиотеки *Discrete*

ких систем: системы с дискретным временем и системы с дискретными состояниями (см. часть I гл. 1). Блоки, входящие в рассматриваемый раздел библиотеки SIMULINK, обеспечивают моделирование как тех, так и других.

Раздел содержит 8 блоков (рис. 3.39):

1. **Unit Delay** – блок задержки сигнала.
2. **Discrete-Time Integrator** – дискретный сумматор (счетчик) времен.
3. **Zero-Order Hold** – экстраполятор нулевого порядка.
4. **First-Order Hold** – экстраполятор первого порядка.
5. **Discrete State-Space** – блок формирования состояния системы.

Блоки, обеспечивающие Z-преобразование входного сигнала:

6. **Discrete Filter**.
7. **Discrete Transfer Fcn**.
8. **Discrete Zero-Pole**.

С точки зрения иллюстрации технологии имитационного моделирования интерес представляют первые два блока из рассматриваемого раздела. Познакомимся с ними поближе.

Блок **Unit Delay** обеспечивает задержку входного сигнала на заданное число периодов (шагов модельного времени).

Параметрами настройки для этого блока являются (рис. 3.40):

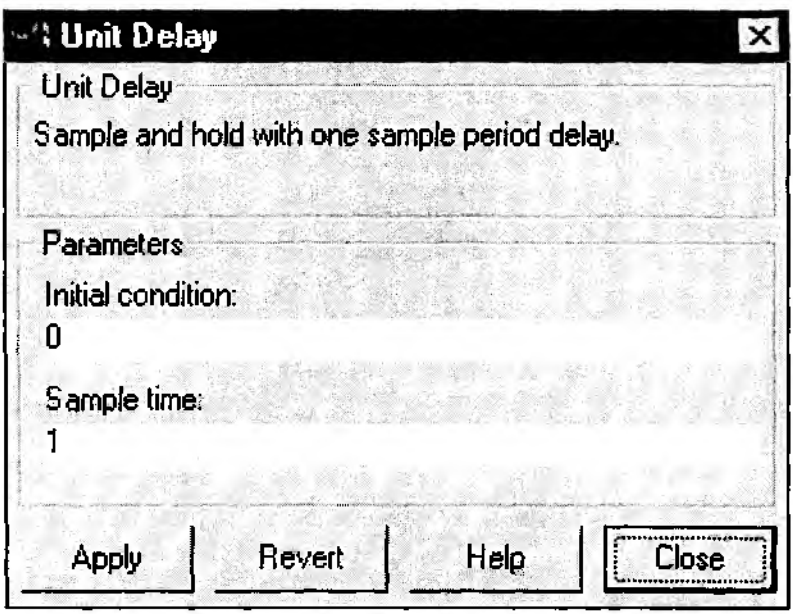


Рис. 3.40. Окно настроек блока **Unit Delay**

1. Начальное значение сигнала (***Initial condition***) — значение амплитуды сигнала в момент активизации блока; оно может быть задано либо в виде числовой константы, либо в виде вычисляемого выражения.

2. Величина задержки (***Sample time***) — определяет число шагов модельного времени, в течение которого сохраняется значение сигнала, поступившего на вход блока; как и первый параметр, может задаваться в любой форме, но значение параметра должно быть положительным.

Блок **Discrete-Time Integrator** выполняет суммирование интервалов времени между поступлениями входного сигнала. Блок может быть использован для управления логикой работы отдельных компонентов ИМ или модели в целом. В частности, суммарная длительность работы некоторой подсистемы может служить условием «досрочного» окончания моделирования.

Блок имеет следующие параметры настройки (рис. 3.41):

- используемый метод интегрирования (***Integrator method***); с помощью выпадающего меню пользователь может выбрать один из трех методов: прямой метод Эйлера; обратный метод Эйлера; метод трапеций;
- подключение дополнительного управляющего сигнала (***External reset***);
- использование внешней установки начального значения входного сигнала (***Initial condition source***).

Выбор значений двух последних параметров также производится с помощью «выпадающих» меню.

Параметр ***External reset*** может принимать следующие значения:

none — дополнительный управляющий сигнал не используется;

rising — для управления используется возрастающий сигнал;

falling — для управления используется ниспадающий сигнал;

either — на работу блока влияет любое изменение амплитуды управляющего сигнала.

Параметр ***Initial condition source*** принимает одно из двух значений:

internal — используется собственная установка начального значения сумматора;

external — установка начального значения производится извне.

Если выбранные пользователем значения двух рассматриваемых параметров предполагают наличие дополнительных входных сигналов, то на графическом изображении блока появляются дополнительные входные порты (после нажатия кнопки *Apply* в окне настроек блока);

- начальное состояние сумматора (***Initial condition***); значение вводится в строке редактирования либо как числовая константа, либо в виде вычисляемого выражения;
- флажок ***Limit output*** (*Ограничение выходного значения сумматора*) определяет, будут ли использоваться следующие ниже 4 параметра настройки;
- верхнее предельное значение времени (***Upper saturation limit***); по умолчанию — не ограничено (*inf*);

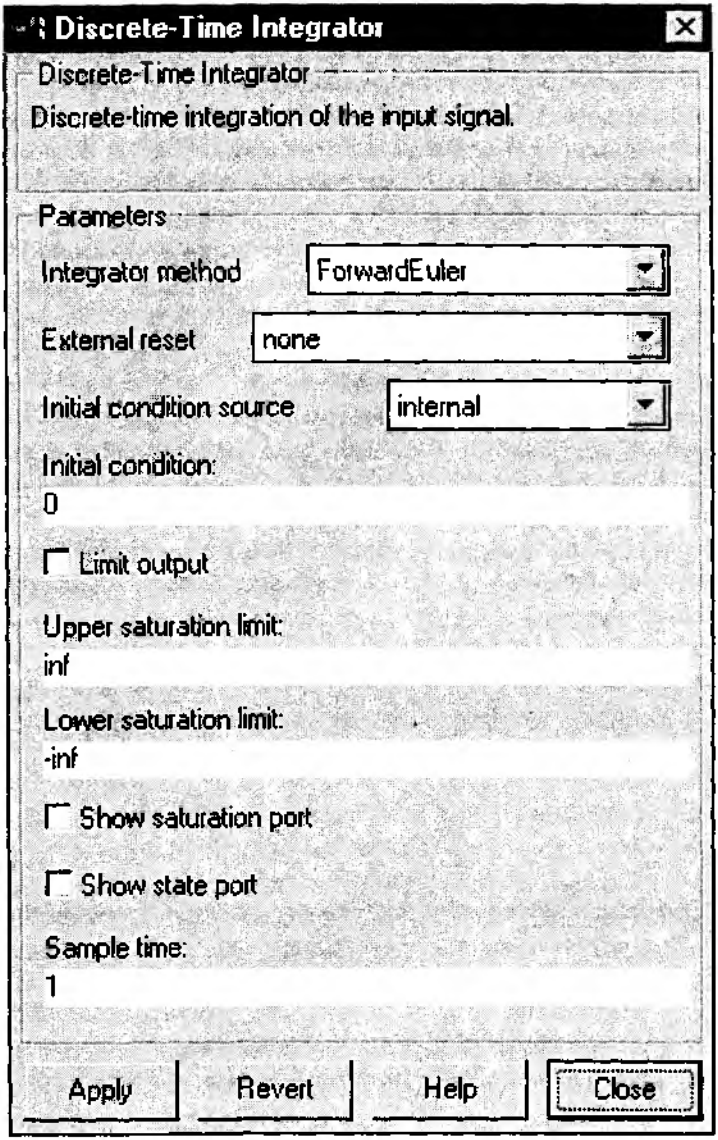


Рис. 3.41. Окно настроек блока **Discrete-Time Integrator**

- нижнее предельное значение времени (***Lower saturation limit***); по умолчанию параметр имеет значение *–inf*;
- флажок *показать порт насыщения* (***Show saturation port***);
- флажок *показать порт состояния* (***Show state port***).

Параметры 5...9 используются следующим образом. Если флажок *Limit output* установлен, то при переходе значения сумматора через верхний или нижний предел на дополнительных выходах блока (*saturation port* и *state port*) формируется единичный сигнал.

Чтобы этот сигнал можно было использовать для управления работой *S*-модели, флажки *Show saturation port* и *Show state port* должны быть включены (при этом на графическом изображении блока появляются обозначения портов). Особенность порта *state port* состоит в том, что снимаемый с него сигнал может быть использован только для прерывания алгебраического цикла или для согласования состояния подсистем модели.

Раздел *Linear* (*Линейные элементы*)

Раздел содержит блоки, которые можно условно разделить на две группы: блоки, непосредственно предназначенные для описания линейных непрерывных систем, и блоки общего назначения, которые могут быть использованы в модели любой системы (рис. 3.42).

К первой группе относятся:

1. **Gain** — «линейный усилитель» (умножитель).
2. **Transfer Fcn** — «передаточное звено».
3. **State-Space** — блок формирования состояния системы.
4. **Zero-Pole**.
5. **Derivative** — блок вычисления производной входного сигнала по времени (*du/dt*).
6. **Dot Product** — блок вычисления свертки (скалярного произведения) двух входных сигналов.
7. **Matrix Gain** — матричный усилитель (умножитель) входного сигнала.

Во вторую группу входят три блока:

1. **Integrator** — сумматор непрерывного времени.
2. **Sum** — блок суммирования входных сигналов.
3. **Slider Gain** — блок изменения коэффициента усиления.

В соответствии с общей концепцией книги более подробно будут рассмотрены блоки, относящиеся ко второй группе.

Блок **Integrator** производит вычисление «времени существования» входного сигнала, и, подобно блоку **Discrete-Time Integrator**, может быть использован для определения временных характеристик моделируемой системы (или отдельных ее подсистем).

Параметры настройки блока полностью идентичны параметрам настройки блока **Discrete-Time Integrator**, рассмотренным ранее. Некоторое отличие имеет лишь последний из них: вместо дискретности суммирования в данном случае требуется указать точность вычислений (*Absolute tolerance*).

Блок **Sum** может использоваться в двух режимах:

- сложения входных сигналов (в том числе с разными знаками);

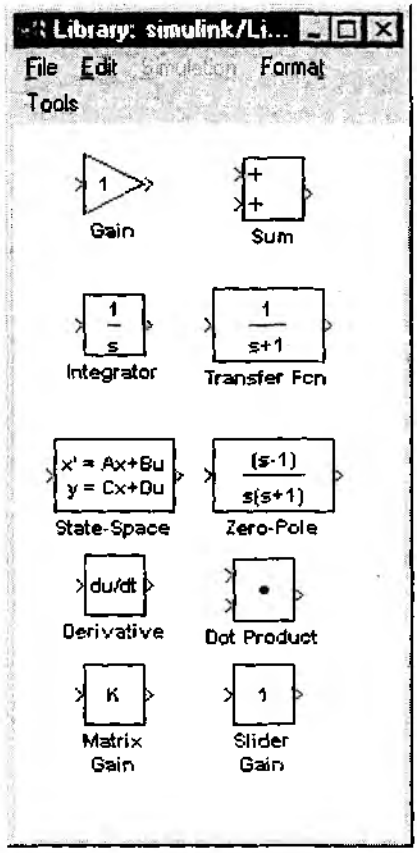


Рис. 3.42. Раздел библиотеки *Linear*

• суммирования элементов вектора, поступающего на вход блока.

Для управления режимами работы блока используется единственный параметр настройки — ***List of sings*** (*Список знаков*) (рис. 3.43).

Значения этого параметра могут задаваться одним из трех способов:

• в виде последовательности знаков «+» и «–»; при этом число знаков определяет число входов блока, а сам знак — полярность входного сигнала;

• в виде целой положительной константы (больше 1); ее значение равно числу входов блока, а все входы считаются положительными (например, ввод константы 4 аналогичен вводу «списка знаков» в форме ++++);

• в виде символа «1», который указывает на использование блока во втором режиме.

Блок **Slider Gain** является одним из элементов взаимодействия пользователя с моделью в процессе моделирования. В активном состоянии блок представляет со-

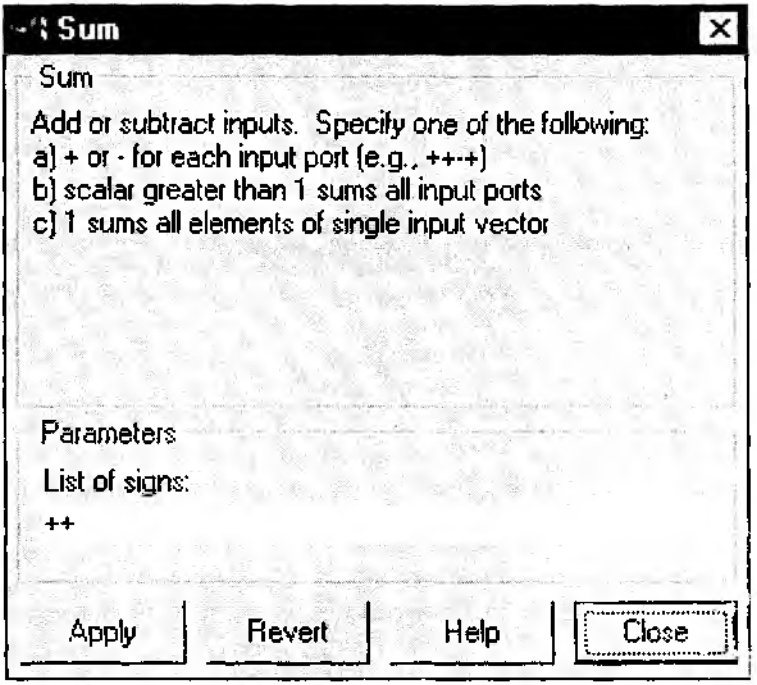


Рис. 3.43. Окно настроек блока **Sum**

бой диалоговое окно, обеспечивающее установку значения некоторого параметра модели с помощью «ползункового» регулятора (рис. 3.44).

Блок **Slider Gain** становится активным после того, как будет помещен в окно блок-диаграммы создаваемой модели. Чтобы открыть окно с регулято-

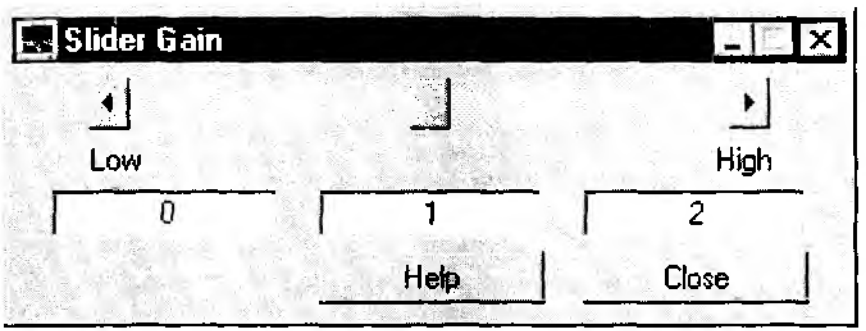


Рис. 3.44. Окно управления, создаваемое блоком **Slider Gain**

ром, необходимо дважды щелкнуть на изображении блока ЛКМ. Окно ***Slider Gain*** имеет три поля ввода: для указания нижнего уровня параметра (*Low*), верхнего уровня (*High*) и текущего значения. Текущее значение должно лежать внутри диапазона [*Low*, *High*]. Однако при выборе нового диапазона необходимо сначала указать новое значение параметра, а затем изменить границы диапазона.

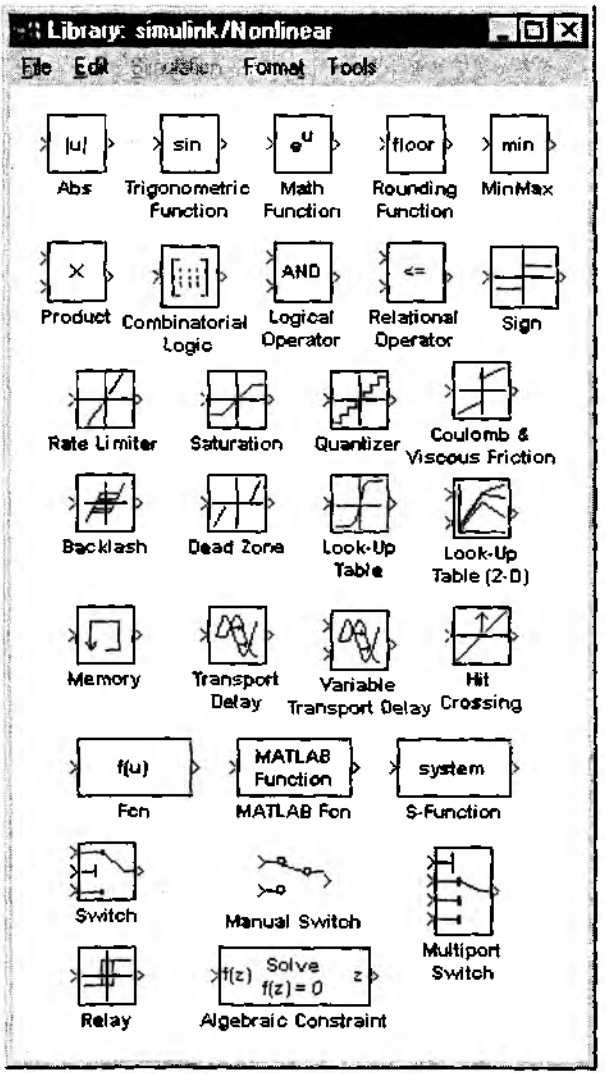


Рис. 3.45. Раздел библиотеки *Nonlinear*

Раздел *Nonlinear* (*Нелинейные элементы*)

По составу элементов это самый большой и, пожалуй, наиболее полезный для имитационного моделирования раздел библиотеки SIMULINK. Он содержит 30 блоков, которые условно можно разделить по назначению на несколько групп (рис. 3.45).

Первую группу образуют блоки, реализующие элементарные математические функции. К таким блокам относятся:

1. Блок **Abs** – формирует абсолютное значение входного сигнала (этот блок не имеет параметров настройки).

2. Блок **Trigonometric Function** обеспечивает преобразование входного сигнала с помощью одной из тригонометрических функций; выбор требуемой функции производится в окне настройки параметров блока с помощью «выпадающего» меню (оно становится доступным только после перемещения блока в поле блок-диаграммы).

3. Блок **Math Function** позволяет использовать для преобразования входного сигнала элементарные нетригонометрические функции (вычисление экспоненты, натурального и десятичного логарифмов, возведение в степень, извлечение квадратного корня и т. д.). Нужная функция выбирается с помощью «выпадающего» меню.

4. Блок **Rounding Function** содержит различные функции округления значения амплитуды входного сигнала; выбор конкретного метода округления выполняется также с помощью «выпадающего» меню.

5. Блок **MinMax** обеспечивает поиск минимального или максимального элемента входного вектора. Цель поиска задается в окне настроек блока. Второй параметр настройки – число входов блока.

Для блоков 2...5 имя выбранной функции выводится на графическом изображении блока.

6. Блок **Fcn** – это универсальный «вычислительный» блок; в качестве параметра настройки блока можно ввести любое вычисляемое выражение, аргументом которого является значение входного сигнала; особенность этого блока состоит в том, что аргумент выражения должен быть указан явно. Для обозначения входного сигнала используется символ *u*. Если входной сигнал является вектором, то для операций, выполняемых над отдельными его элементами, аргумент также должен быть задан явно. Например, сложение двух элементов входного сигнала должно быть записано в таком виде: $u(1)+u(2)$ (рис. 3.46).

Замечание. Вычисляемое выражение должно давать только скалярное значение. Это ограничение снимается при использовании блока **MATLAB Fcn**.

7. Блок **MATLAB Fcn** позволяет применить к входному сигналу любую подпрограмму обработки, реализованную в виде М-файла. Это может быть как библиотечная функция пакета MATLAB, так и подпрограмма, созданная разработчиком *S*-модели.

В первом случае использование блока **MATLAB Fcn** аналогично использованию блока **Fcn**. Вычисляемое выражение вводится в окне настроек блока в строке

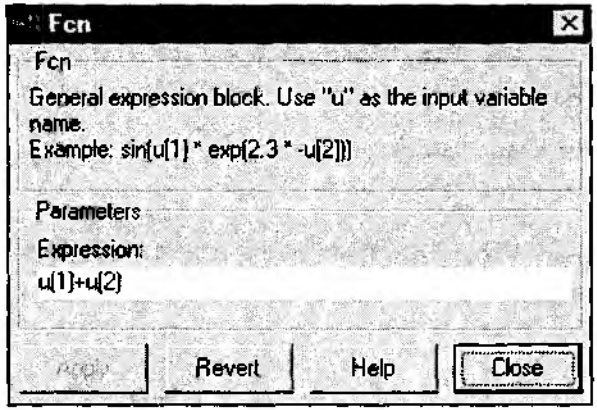


Рис. 3.46. Окно настроек блока **Fcn**

MATLAB function. Если оно содержит только обращение к библиотечной функции, то ее аргумент можно явно не указывать. Например, для вычисления квадратного корня входного значения достаточно ввести имя функции *sqrt*.

Другой, более существенной, особенностью блока **MATLAB Fcn** является наличие второго параметра настройки — ***Output width*** (*Ширина выходного сигнала*). Это означает, что результатом работы блока может быть не только скаляр, но и вектор. Число его элементов должно быть равно значению параметра *Output width*.

8. Блок **Product** позволяет выполнять умножение или деление нескольких входных сигналов (величин). В качестве параметров настройки могут указываться число входов блока и вид выполняемой операции.

Задание значений этих параметров аналогично настройке блока **Sum** из раздела *Линейные элементы*. В качестве знака операции умножения используется символ «*», а для указания операции деления — символ «/».

Если в качестве значения параметра настройки блока ввести «1», то будет вычисляться произведение элементов входного вектора (в этом случае на изображении блока выводится символ *P*).

Вторую группу образуют блоки, обеспечивающие логическую обработку входного сигнала. Наиболее важными и полезными из них являются следующие.

1. Блок **Logical Operator** содержит набор основных логических операций: *AND* (операция логического умножения «И»), *OR* (логическое сложение «ИЛИ»), *NAND* («И-НЕ»), *NOR* («ИЛИ-НЕ»), *XOR* (сложение по модулю 2), *NOT* (операция логического отрицания). Выбор требуемой функции выполняется с помощью «выпадающего» меню; имя функции отображается на иконке блока.

Другим параметром настройки блока является число аргументов логической операции, то есть число входных портов блока (***Number of input ports***). Его значение вводится в строке редактирования и должно быть натуральным числом. Максимально допустимое число входов блока практически не ограничено.

2. Блок **Relational Operator** реализует операции отношения над двумя входными сигналами: >, <, ≥, ≤, == (тождественно), != (не равно). Конкретная операция выбирается при настройке параметров блока посредством «выпадающего» меню. Знак операции выводится на изображении блока.

3. Блок **Combinatorial Logic** обеспечивает преобразование входного сигнала в соответствии с заданной таблицей истинности. По своим функциональным возможностям это очень мощный блок. С его помощью можно описать на уровне «вход-выход» логику работы любого устройства или системы. Единственное ограничение заключается в том, что входные данные и результат должны быть представимы в форме булевых величин. Напомним, что булева величина принимает только два значения: «1» («истина») или «0» («ложь»). На самом деле это очень «мягкое» ограничение, так как любое действие, утверждение или результат можно оценить по двухбалльной шкале: если действие произошло (или если некоторое утверждение нам нравится) — ставим ему «1», если же нет — оно получает оценку «0».

Например, включение настольной лампы можно описать с помощью законов физики, а можно — с помощью булевых величин:

«нажал выключатель» («1») — «лампа зажглась» («1»),

«не нажал выключатель» («0») — «лампа не зажглась» («0»),

или «нажал выключатель» («1») — «лампа не зажглась» («0»).

Последний вариант возможен, если лампа неисправна.

При таком подходе описываемое устройство рассматривается как «черный ящик», который при поступлении входного сигнала формирует некоторый выходной сигнал.

В кибернетике для описания и исследования работы «черных ящиков» создана специальная теория — теория автоматов. Теперь, возвращаясь к блоку **Combinatorial Logic**, можно сказать, что он представляет собой обобщенную модель конечного детерминированного автомата. Для такого автомата заранее известны все возможные значения выходного сигнала, и набор их ограничен. Причем каждое выходное значение однозначно соответствует определенному входному воздействию.

На вход блока может подаваться как скалярный, так и векторный сигнал. Скаляр в данном случае интерпретируется как одна булева величина, а вектор — как их совокупность. При этом любое ненулевое значение входного сигнала соответствует значению «истина».

Блок имеет единственный параметр настройки — ***Truth table*** (*таблица истинности*), который представляет собой список возможных выходных значений автомата. При задании таблицы истинности необходимо соблюдать два основных правила:

1) число строк таблицы должно быть равно 2^n, где n — число элементов (размерность) входного сигнала;

2) входы таблицы считаются заданными.

Например, если на вход блока **Combinatorial Logic** подается векторный сигнал с n=2, то параметр *Truth table* представляет собой список из четырех элементов, например такой: [0;1;0;1] (рис. 3.47).

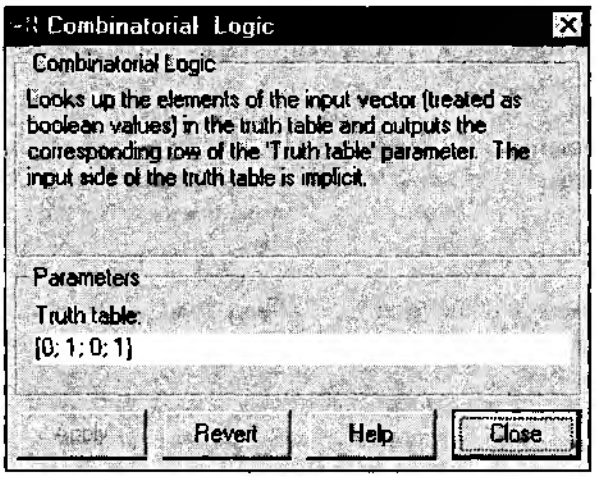


Рис. 3.47. Окно настроек блока *Combinatorial Logic*

Для приведенного примера полная таблица истинности будет выглядеть следующим образом (табл. 3.1):

Таблица 3.1.

Пример полной таблицы истинности для блока Combinatorial logic

Входной сигнал		Выходной сигнал
1-й элемент	2-й элемент	
0	0	0
0	1	1
1	0	0
1	1	1

Таким образом, параметр *Truth table* описывает только значения выходного сигнала.

Разрядность выходного сигнала, а также значение каждого его разряда выбираются пользователем на основе собственных представлений о логике работы создаваемой *S*-модели.

Рассмотрим содержательный пример, иллюстрирующий особенности использования блока.

Пусть к девушке Маше собираются посвататься два молодых человека — Вася и Петя. Не зная, кому отдать предпочтение, Маша решает: «Кто придет с букетом цветов, того и выберу. А если принесут оба — тогда Петю». В любом случае Маша должна сообщить о своем выборе родителям. Определив для себя линию

поведения, Маша начинает действовать как конечный детерминированный автомат. Описать его работу можно с помощью приведенной ниже таблицы истинности (табл. 3.2).

Таблица 3.2.

Таблица истинности, описывающая поведение Маши

На входе		На выходе		
У Васи цветы	У Пети цветы	Ответ Васе	Ответ Пете	Сообщение родителям
0	0	0	0	1
0	1	0	1	1
1	0	1	0	1
1	1	0	1	1

Для рассматриваемой ситуации значение параметра *Truth table* будет выглядеть следующим образом:

[001; 011; 101; 011].

Полезно еще раз отметить, что любое ненулевое значение сигнала в блоке **Combinatorial Logic** трактуется как «истина». Это относится и к входному, и к выходному сигналу. Поэтому при необходимости приведенная выше таблица истинности могла бы выглядеть несколько иначе. Если положительный ответ Маши избраннику требуется выразить через число поцелуев (например, 3), а сообщение родителям — через стоимость 1 минуты телефонного разговора (например, 4 рубля), то в строке *Truth table* нужно ввести:

[004; 034; 304; 034].

Сформированный выходной сигнал может быть или выведен в какое-либо «смотровое окно», или передан на другие блоки *S*-модели.

Следующие четыре блока можно рассматривать как вариант модификации блока **Combinatorial Logic** для непрерывного входного сигнала.

Dead Zone (*Мертвая зона*) — заменяет значение входного сигнала, лежащее в заданном диапазоне, нулем;

Look-up Table (*Таблица поиска*) — выполняет линейную интерполяцию входного сигнала в соответствии с заданной табличной функцией.

Look-up Table(2D) — производит линейную интерполяцию двумерного входного сигнала;

Hit Crossing (*Обнаружено пересечение*) — позволяет идентифицировать момент времени, когда входной сигнал «пересекает» некоторое значение: при появлении такой ситуации на выходе блока формируется единичный сигнал.

Блок **Hit Crossing** во многих случаях оказывается весьма удобным средством управления логикой работы *S*-модели, поэтому остановимся на нем немного подробнее.

Блок имеет три параметра настройки (рис. 3.48):

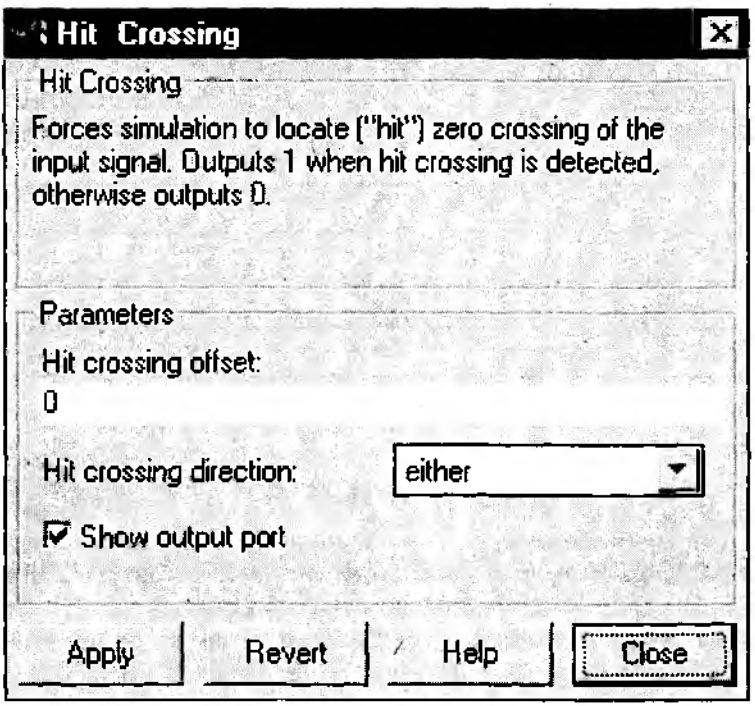


Рис. 3.48. Окно настроек блока **Hit Crossing**

Hit crossing offset – определяет значение, «пересечение» которого необходимо идентифицировать;

Hit crossing direction позволяет указать направление «*пересечения*», которое интересует разработчика модели; значение этого параметра выбирается с помощью «выпадающего» меню, которое содержит три пункта:

rising (*возрастание*),

falling (*убывание*),

either (*в обоих направлениях*);

Show output port (*показать выходной порт*) – флажок, с помощью которого можно выбрать формат использования блока.

При одновременном выполнении условий, задаваемых параметрами *Hit crossing offset* и *Hit crossing direction*, на выходе блока формируется единичный импульс. Его длительность определяется значением параметра *Sample time* блока, предшествующего в модели блоку **Hit crossing**. Если этот параметр отсут-

ствует, то единичный сигнал на выходе блока **Hit crossing** существует до его следующего срабатывания.

В третью группу можно объединить блоки, реализующие функцию задержки входного сигнала. Таких блоков в разделе ***Nonlinear*** три:

Memory (*Память*);

Transport Delay (*Задержка передачи*)

Variable Transport Delay (*Изменяемая задержка передачи*).

Блок **Memory** является наиболее «слабым» из них по своим возможностям. Он выполняет задержку входного сигнала только на один шаг модельного времени. Блок имеет два параметра настройки (см. рис. 3.49):

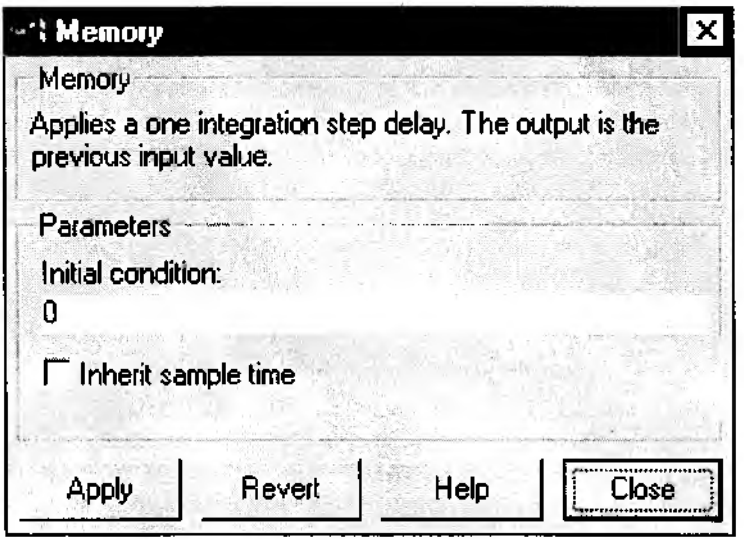


Рис. 3.49. Окно настроек блока **Memory**

Initial condition (*Начальное состояние*) — задает значение амплитуды входного сигнала на момент инициализации блока; в большинстве случаев целесообразно принимать это значение равным нулю; флажок ***Inherit sample time*** (*наследование шага времени*) позволяет выбрать величину шага, на который будет производиться задержка сигнала:

- если флажок снят, то используется минимальный шаг, равный 0.1 единицы модельного времени;
- если флажок установлен, то величина шага определяется значением *Sample time* блока, предшествующего блоку **Memory**.

Блок **Transport Delay** обеспечивает задержку сигнала на заданное количество шагов модельного времени, причем необязательно целое. Настройка блока производится с помощью трех параметров (рис. 3.50):

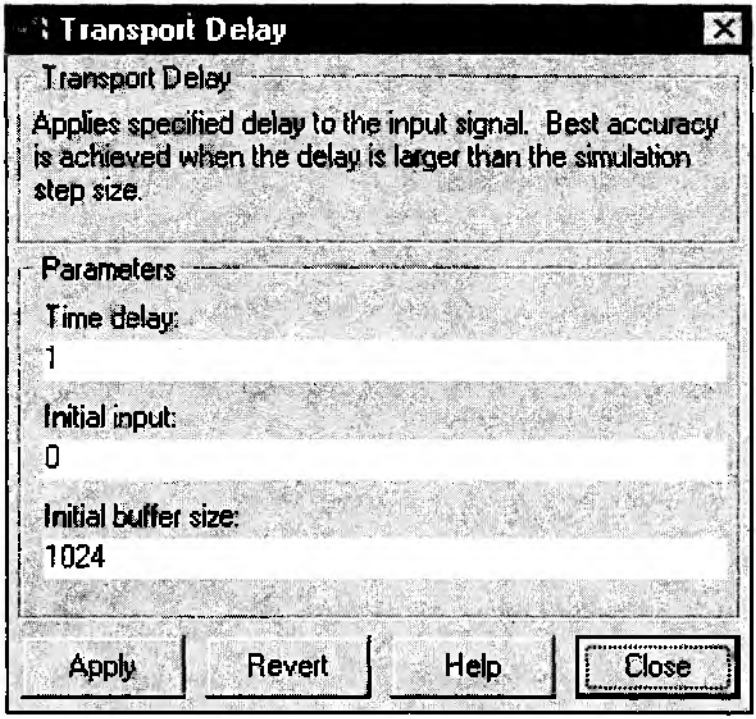


Рис. 3.50. Окно настроек блока **Transport Delay**

Time delay (*Время задержки*) — количество шагов модельного времени, на которое задерживается сигнал; может вводиться либо в числовой форме, либо в форме вычисляемого выражения;

Initial input (*Начальный ввод*) — значение амплитуды входного сигнала в момент инициализации блока (по умолчанию равно 0);

Initial buffer size (*Начальный размер буфера*) — объем памяти (в байтах), выделяемой в рабочей области MATLAB для хранения параметров задержанного сигнала; значение параметра должно быть кратно 8 (по умолчанию — 1024).

Блок **Variable Transport Delay** позволяет задавать управляемую извне величину задержки. С этой целью блок имеет дополнительный вход. Подаваемый на него сигнал определяет длительность задержки информационного сигнала, поступающего на основной вход блока.

Данный блок, как и предыдущий, имеет 3 параметра настройки: ***Maximum delay*** (*Максимальная задержка*), ***Initial input*** и ***Buffer size***. Назначение двух последних параметров идентично назначению одноименных параметров блока **Transport delay**.

Параметр *Maximum delay* определяет наибольшую допустимую величину задержки информационного сигнала. Если величина задержки, определяемая уп-

равляющим сигналом, превышает этот порог, то она принудительно устанавливается равной параметру *Maximum delay*. Его значение измеряется числом шагов модельного времени, может иметь дробное значение и вводиться либо в числовой форме, либо в форме вычисляемого выражения.

Четвертую группу образуют «блоки-переключатели», то есть блоки, управляющие направлением передачи сигнала. Таких блоков четыре:

Switch (*Переключатель*);
Manual Switch (*Ручной переключатель*);
Multiport Switch (*Многовходовый переключатель*)
Relay (*Реле*).

Блок **Switch** имеет три входа: два информационных (1-й и 3-й) и один управляющий (2-й). Логика работы блока состоит в следующем. Если амплитуда сигнала, поступающего на 2-й вход, не меньше заданного порогового значения, то на выход блока передается сигнал с 1-го входа, в противном случае — сигнал с 3-го входа. Блок имеет единственный параметр настройки — ***Threshold*** (*Порог*). Он может задаваться либо как числовая константа, либо как вычисляемое выражение. Периодичность срабатывания блока **Switch** определяется значением параметра *Sample time* блока, подсоединенного к его управляющему входу.

На рис. 3.51 показан пример использования блока *Switch* (значение параметра *Threshold* равно 5).

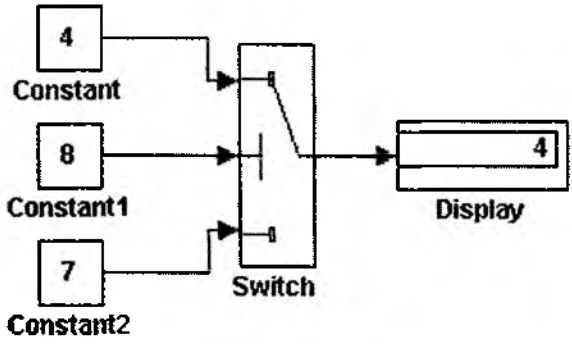


Рис. 3.51. Пример использования блока **Switch**

Необходимо иметь в виду, что при изменении направления передачи сигнала положение «перемычки» на иконке блока не изменяется.

Блок **Manual Switch** не имеет параметров настройки и позволяет «вручную» выбирать один из двух входных портов, сигнал с которого будет передаваться на выход блока. Для перемещения «перемычки», соединяющей выходной порт блока **Manual Switch** с входным, необходимо дважды щелкнуть ЛКМ на изображении

блока (предварительно, конечно, блок должен быть помещен в поле блок-диаграммы разрабатываемой модели).

Блок **Multiport Switch** обеспечивает передачу на выход сигнала, поступающего на один из информационных входов. Номер коммутируемого входа равен значению сигнала, подаваемого на управляющий вход блока. Если это значение является дробным числом, то оно округляется до целого по стандартным арифметическим правилам. Исключение составляют 2 случая: если значение управляющего сигнала меньше 1, то оно считается равным 1; если значение управляющего сигнала превышает число информационных входов, то оно принимается равным наибольшему номеру (входы нумеруются сверху вниз).

Блок имеет один параметр настройки — ***Number of inputs*** (*Число входов*), который устанавливает число информационных входов. Значение параметра может вводиться в форме числовой константы или в форме вычисляемого выражения.

Блок-диаграмма, поясняющая работу блока, приведена на рис. 3.52.

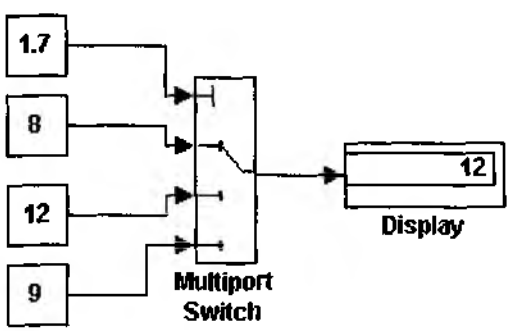


Рис. 3.52. Пример использования блока **Multiport Switch**

Блок **Relay** работает по аналогии с обычным реле: если входной сигнал превышает некоторое пороговое значение, то на выходе блока формируется «разрешающий» сигнал.

Блок имеет 4 параметра настройки (рис. 3.53):

Switch on point (*Точка включения*) — задает пороговое значение, при превышении которого происходит «включение» реле;

Switch off point (*Точка выключения*) — задает уровень сигнала, при котором реле «выключается»;

Output when on (*Выход при включенном состоянии*) — определяет значение амплитуды «разрешающего» сигнала;

Output when off (*Выход при выключенном состоянии*) — уровень сигнала на выходе реле, когда оно находится в состоянии «выключено».

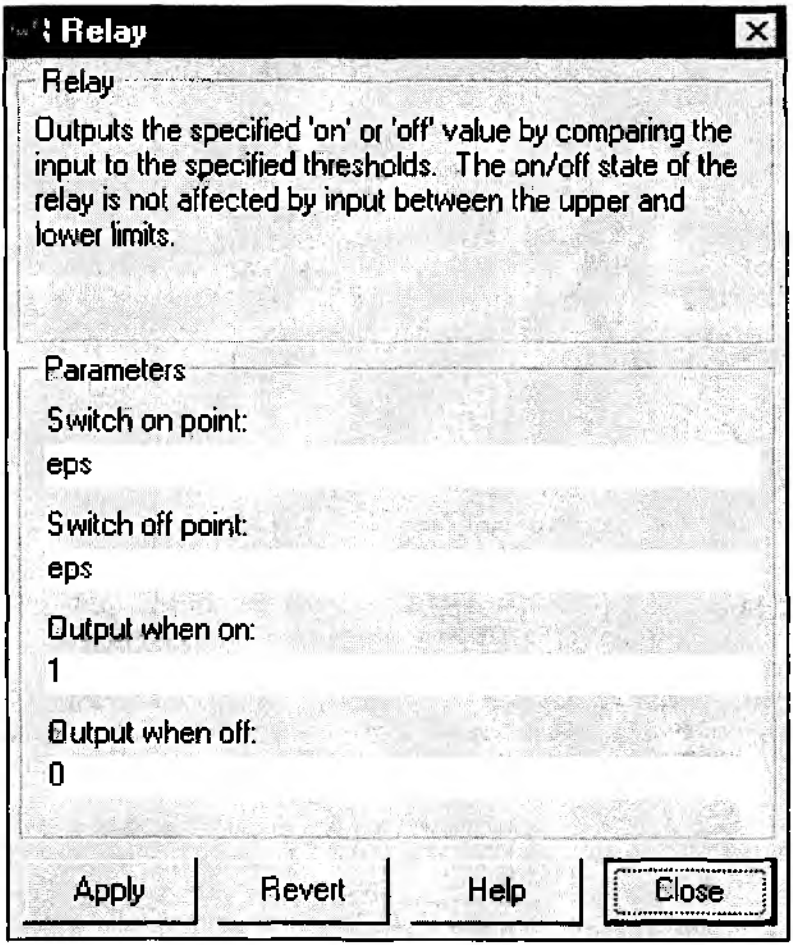


Рис. 3.53. Окно настроек блока **Relay**

Значения всех параметров блока могут вводиться либо в форме числовых констант, либо в форме вычисляемых выражений.

Остальные блоки раздела используются в основном при моделировании систем автоматического регулирования и в книге не рассматриваются.

Раздел *Connections* (*Соединительные узлы*)

Большинство блоков данного раздела предназначено для разработки *S*-моделей, содержащих модели более низкого уровня (подсистемы). Состав блоков раздела показан на рис. 3.54.

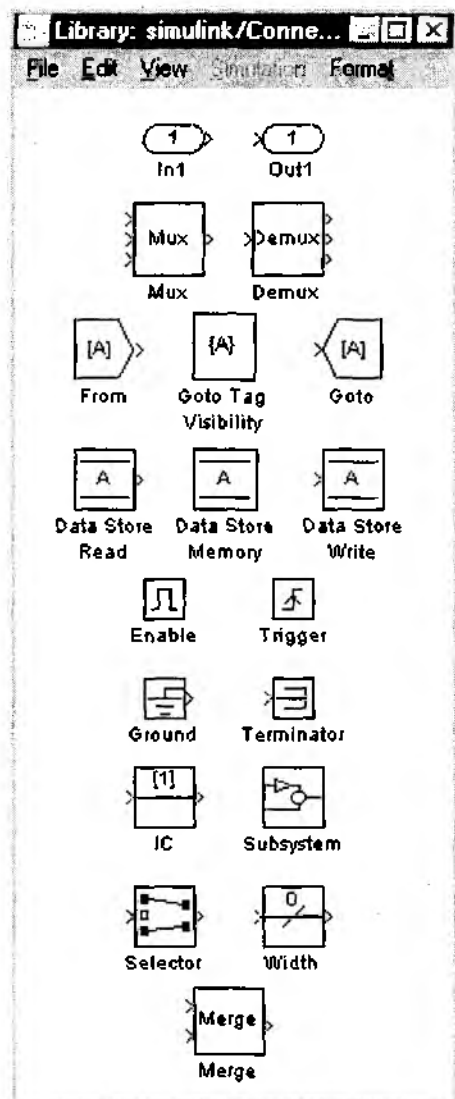


Рис. 3.54. Раздел библиотеки *Connections*

Технология создания и использования подсистем будет описана в следующей главе, поэтому пока ограничимся только краткой характеристикой блоков, относящихся к этой группе.

1. Блоки **In** (*Входной порт*) и **Out** (*Выходной порт*) обеспечивают «линейную» связь между подсистемами модели по информации.

2. Блоки **Goto Tag Visibility** (*Признак видимости*), **From** (*принять*) и ***Goto*** (*передать*) используются совместно и предназначены для обмена данными между различными компонентами *S*-модели с учетом доступности (видимости) этих данных.

3. Блоки **Data Store Memory** (*Память данных*), **Data Store Read** (*чтение данных*) и **Data Store Write** (*Запись данных*) также используются совместно и обеспечивают не только передачу данных, но и их хранение на интервале моделирования.

4. Блоки **Enable** (*Разрешить*) и **Trigger** (в данном случае уместен буквальный перевод этого слова — *защелка*) предназначены для логического управления работой *подсистем S-модели.*

5. Блок **Subsystem** (*Подсистема*) представляет собой «заготовку» для создания подсистемы. Подсистема — это достаточно самостоятельная *S*-модель более низкого уровня, которая, в свою очередь, может содержать подсистемы произвольного уровня вложенности.

Наряду с перечисленными выше, раздел *Connections* содержит еще несколько элементов, которые имеют самостоятельное значение и могут оказаться весьма полезными при создании многих практических приложений:

Блок **Mux** — выполняет объединение входных величин в один линейный вектор. При этом входные величины могут быть как скалярными, так и векторными. Размерность результирующего вектора равна суммарному количеству элементов, поступающих на входные порты блока. Например, если на 1-й вход блока подается матрица размером 2 × 2, содержащая четыре элемента:[[1,4],[6,7]], на 2-й и 3-й входы — константы 2 и 3 соответственно, то выходной вектор будет выглядеть так: [1 4 6 7 2 3] (рис. 3.55).

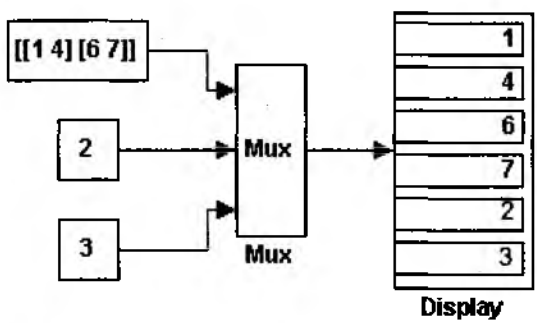


Рис. 3.55. Пример использования блока **Mux**

Блок **Mux** имеет один параметр настройки — ***Number of inputs*** (*Число входов*).

Блок **Demux** (*Разделитель*) выполняет функции, противоположные функциям блока **Mux**: разделяет входной вектор на заданное число компонентов. Данный блок также имеет единственный параметр настройки, который называется ***Number of***

outputs (*Число выходов*). Поскольку соотношение между размерностью входного вектора и числом выходов блока может быть различным, то размерность компонентов выходного вектора разработчику модели необходимо определять заранее самому. При этом следует учитывать следующие особенности работы блока **Demux**.

Если размерность входного вектора (M) равна значению параметра *Number of outputs*, т. е. числу выходов блока (N), то на всех выходах блока формируются скалярные величины — входной вектор просто распадается на отдельные элементы.

Если размерность входного вектора превышает число выходов блока ($M>N$), то на первых $(n-1)$-ом выходах формируются векторы, размерность которых равна целой части отношения M/N, а размерность вектора на последнем выходе равна остатку от деления (рис. 3.56).

Если $M<N$, то при запуске модели выдается сообщение об ошибке («несоответствие размерности портов»).

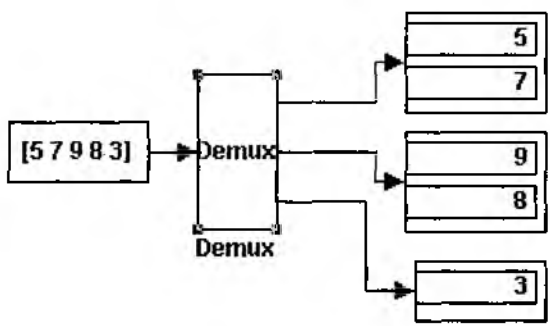


Рис. 3.56. Пример использования блока **Demux**

Блоки **Ground** (*Земля*) и **Terminator** (*Ограничитель*) могут использоваться в качестве «заглушек» для тех портов, которые по какой-либо причине оказались не подключенными к другим блокам S-модели (например, на этапе отладки модели). Блок **Ground** используется как «заглушка» для входных портов, а блок **Terminator** — для выходных. Применение указанных блоков позволяет избежать выдачи интерпретатором MATLAB предупреждения о том, что в S-модели имеются неподключенные порты.

Блок **IC** (*Initial Condition — начальное состояние*) позволяет устанавливать произвольное начальное состояние входного сигнала. Длительность пребывания системы в этом состоянии равна длительности шага моделирования. Параметром настройки блока является величина ***Initial Value*** (*Значение сигнала на момент входа в блок*).

Блок **Width** (*Размер*) вычисляет размерность сигнала, поступающего на его вход. Вычисленное значение выводится непосредственно на изображении блока. Параметров настройки он не имеет.

Блок **Selector** (*Селектор*) выбирает во входном векторе и передает на выход только те элементы, номера которых указаны в параметрах настройки блока.

Блок имеет два параметра настройки:

• ***Elements*** (*Элементы*) — список номеров элементов входного вектора, подлежащих отбору;

• ***Iuput port width*** (*Размерность входного порта*) — размерность входного вектора, который может быть «просеян» с помощью данного блока; значение этого параметра обязательно должно совпадать с числом элементов входного вектора.

Если на вход блока подается матрица, то в списке *Elements* используется сквозная нумерация элементов, причем индекс изменяется построчно. Например, если на вход блока поступает матрица размером 3×4, то ее элементы будут иметь номера с 1-го по 12-й, причем номера 1...3 будут принадлежать элементам первой строки, номера 4..6 — элементам 2-й строки и т. д.

На рис. 3.57. показана работа блока **Selector**, выполняющего «просеивание» входного массива, содержащего две строки (в первой — три элемента, во второй — два). На выход блока, в соответствии с его параметрами настройки (рис. 3.58), передаются только два элемента — первый и четвертый.

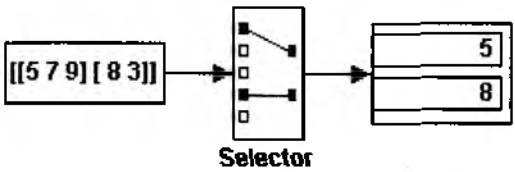


Рис. 3.57. Пример работы блока **Selector**

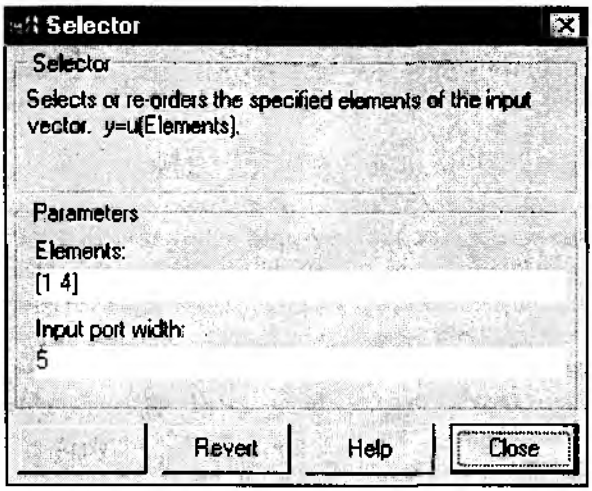


Рис. 3.58. Пример установки параметров блока **Selector**

Существенным достоинством блока является то, что значения его параметров настройки отображаются в графическом виде на иконке блока.

Блок **Merge** (*Слияние*) выполняет объединение поступающих на его входы сигналов в один. Блок имеет два параметра настройки (рис. 3.59):

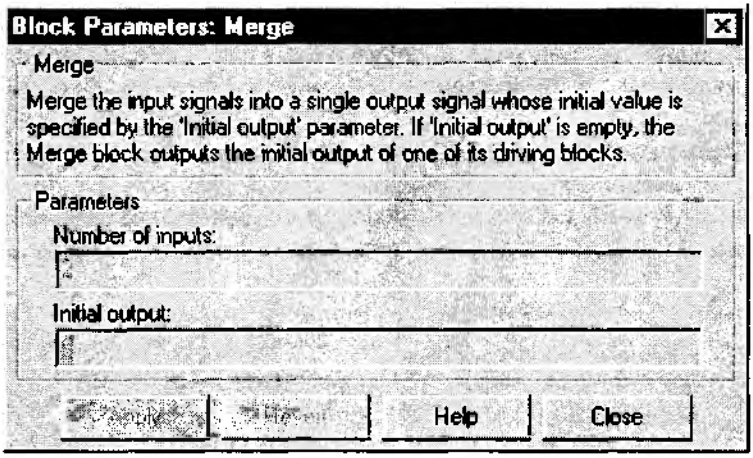


Рис. 3.59. Окно настроек блока **Merge**

Number of inputs (*Количество входов*), определяющий число входных сигналов, которое может быть подано на входы блока; для корректной работы блока все входные сигналы должны иметь одинаковую размерность (хотя при несоблюдении этого правила сообщение об ошибке не выдается);

Initial output (*Инициализация выхода*) — задает сигнал, на основе которого должно выполняться объединение; если значение параметра не задано, то на выход блока просто выдается один из входных сигналов.

Дополнительные разделы библиотеки (*Blocksets & Toolboxes*)

Еще раз отметим, что раздел *Blocksets & Toolboxes* является единственным разделом, содержимое которого изменяется в зависимости от установленной на вашем компьютере конфигурации пакета MATLAB.

Если из инструментальных средств пакета в рабочую конфигурацию включен только SIMULINK, то данный раздел содержит единственный подраздел — *Simulink Extras* (*Дополнения к Simulink*).

Этот подраздел, в свою очередь, разбит на шесть частей (их пиктограммы были показаны на рис. 3.19):

Additional Sinks (*Дополнение к разделу Sinks*) — включает дополнительные графические «смотровые окна»; четыре из них обеспечивают визуализацию спектрального анализа сигналов, другие два — отображение корреляционных характе-

ристик сигналов. Для использования двух последних блоков требуется включить в состав конфигурации пакета инструментальные средства обработки сигналов (*Signal Processing Toolbox*);

Additional Linear (*Дополнение к разделу Linear*) — блоки, входящие в эту часть библиотеки, предназначены для расчета характеристик линейных систем автоматического управления;

Transformations (*Преобразования*) — содержит блоки, выполняющие различные преобразования числовых величин:

- координат — из прямоугольной системы в полярную либо сферическую и обратно;
- температуры — из шкалы Фаренгейта в шкалу Цельсия и обратно;
- углов — из градусов в радианы и обратно.

Flip Flop (*Триггеры*) — содержит блоки, моделирующие работу основных типов триггеров:

- D-триггера,
- *S*-R-триггера,
- J-K-триггера;

Кроме того, в эту часть библиотеки включен блок **Clock** (*Часы*), генерирующий дискретный временной сигнал. Параметром настройки этого блока является величина периода сигнала;

Linearization (*Линеаризация*) — раздел содержит единственный блок, позволяющий вычислить производную входного сигнала по времени.

На этом мы завершим знакомство с базовой библиотекой блоков SIMULINK. В заключение еще раз отметим, что основная цель приведенного краткого обзора разделов библиотеки — дать читателю представление о потенциальных возможностях SIMULINK как инструмента визуального моделирования. Особенности организации взаимодействия блоков в рамках одной *S*-модели или при разработке семейства моделей, а также технология подготовки и проведения модельных экспериментов будут рассмотрены в следующих главах.

ГЛАВА 4
ОТ ПРОСТОГО К СЛОЖНОМУ

— Мы ведь только что позавтракали, — сказал Незнайка.

— Так разве это был настоящий завтрак? —возразил Пончик. — Этот завтрак был пробный, так сказать, *черновой, тренировочный... Зато теперь, когда тренировка закончена, мы можем позавтракать по-настоящему.*

Н. Носов

4.1. СОЗДАЕМ ПОЧТИ МОДЕЛЬ

В этом разделе мы рассмотрим основные технологические аспекты создания *S*-моделей на фоне приводившегося в предыдущей главе «случая из жизни».

Напомним, речь шла о том, что некая девушка Маша пытается выбрать одного из двух женихов. При использовании формального подхода к описанию этой ситуации оказалось, что поведение Маши аналогично поведению конечного детерминированного автомата. Попробуем построить *S*-модель, с помощью которой можно было бы исследовать поведение такого автомата в различных ситуациях.

Чтобы справиться с поставленной задачей, потребуется более детальное знакомство с командами меню окна блок-диаграммы.

4.1.1. МЕНЮ ПОЛЬЗОВАТЕЛЯ

Если создается новая *S*-модель (как в нашем случае), то в первую очередь необходимо открыть новое (пустое) окно блок-диаграммы. Это можно сделать либо из командного окна MATLAB, либо из окна *Library: Simulink*, выбрав из меню последовательно команды *File → New → Model*. Новое окно блок-диаграммы имеет по умолчанию имя *untitled* (*безымянное*), которое может быть изменено при записи файла модели на диск.

Меню окна блок-диаграммы (в дальнейшем для краткости будем называть его меню пользователя) содержит четыре основных раздела: ***File***, ***Edit***, ***Simulation*** и ***Format***.

Раздел *File*

В него входят следующие пункты (рис. 4.1).

New – создание нового окна блок-диаграммы; команда имеет два варианта: *Model* (открыть окно для создания *S*-модели) и *Library* (открыть окно для создания нового раздела библиотеки SIMULINK).

Open – открыть mdl-файл; при выборе данного пункта открывается стандартное диалоговое окно файловой системы Windows, с помощью которого можно найти и открыть требуемый файл, имеющий расширение mdl.

Close – закрыть окно блок-диаграммы (и соответствующий mdl-файл); если в блок-диаграмму вносились изменения, которые не были сохранены в файле на диске, то перед закрытием окна MATLAB запрашивает подтверждение на закрытие файла.

Save – сохранить (записать на диск) mdl-файл; если данный файл записывается впервые, то при выборе этой команды открывается диалоговое окно, с помощью которого пользователь может выбрать новое имя файла (вместо untitled) и каталог, в котором будет производиться запись; если же файл уже записывался на диск ранее, то при выполнении команды Save он будет сохранен под прежним именем и в том же каталоге (при этом диалоговое окно не открывается).

Save as... – сохранить как... (переписать) – команда позволяет сохранить файл под новым именем или в другом каталоге; при ее выполнении открывается диалоговое окно файловой системы Windows.

Замечание: при выполнении команд *Save as...* и *Save* (при первой записи) *mdl*-файл при необходимости может быть сохранен в формате M- или MAT-файла.

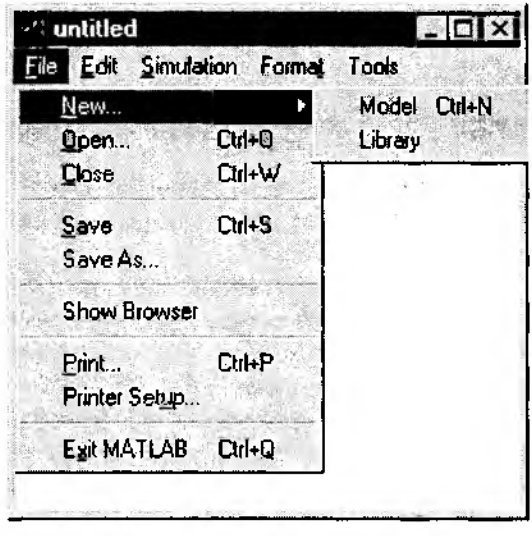


Рис. 4.1. Раздел ***File*** меню пользователя

Этим разработчику предоставляется возможность корректировать текст программы модели с помощью встроенного отладчика MATLAB. Более подробно соответствующие вопросы рассмотрены в Главе 5.

Show Browser — вызов специального средства просмотра структуры *S*-модели (броузера). Основное назначение этого средства — отображение в текстовой форме иерархической структуры модели. Его полезно использовать в том случае, если *S*-модель содержит подсистемы, относящиеся к различным уровням вложенности.

При запуске броузера открывается его диалоговое окно (рис. 4.2). Окно содержит два списка и элементы управления.

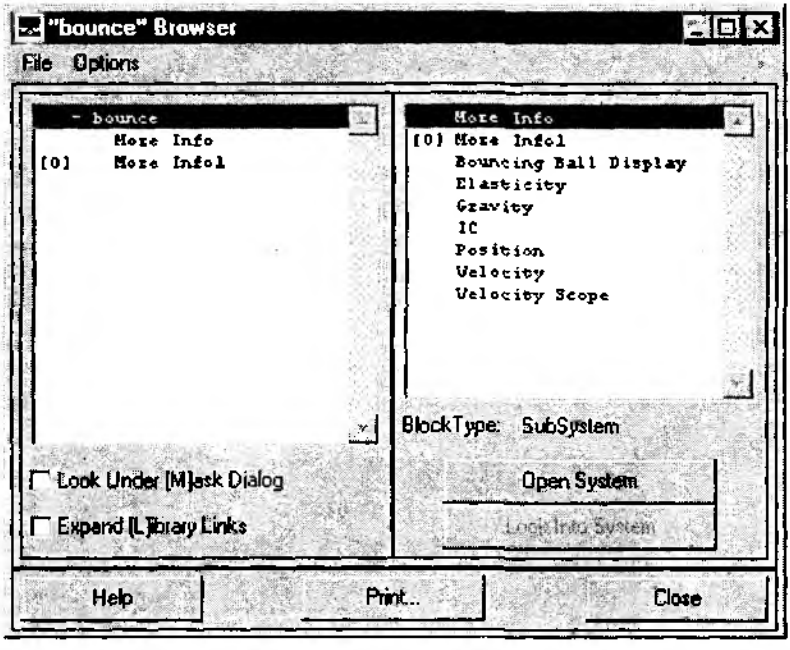


Рис. 4.2. Окно «броузера» блок-диаграммы

В левом списке выводятся имена подсистем, входящих в состав *S*-модели. Список может быть выстроен либо в соответствии с иерархией подсистем, либо в алфавитном порядке. Выбор способа упорядочения выполняется с помощью команды *Display* раздела *Options* меню броузера.

Для повышения наглядности при выводе списка подсистем используются специальные символы:

«+» — подсистема содержит подсистемы более низкого уровня, которые не выведены в списке; чтобы «развернуть» список, необходимо дважды щелкнуть ЛКМ на строке, помеченной символом «+». После этого имена вложенных подсистем появятся в списке, а символ «+» будет заменен символом «–»;

[M] — признак «маскированной» подсистемы; это означает, что непосредственно из блок-диаграммы содержимое такой подсистемы просмотреть нельзя; более подробно назначение и порядок создания маскированных подсистем будут рассмотрены в разделе 4.4;

[L] — признак того, что подсистема взята из библиотеки и не подлежит редактированию из окна блок-диаграммы;

[O] — соответствующая подсистема представляет собой информационное (справочное) окно;

[S] — признак подсистемы типа *Stateflow chart* (*Диаграмма состояний*). Такие подсистемы разрабатываются с помощью специального инструментального средства MATLAB — Stateflow. Некоторые вопросы, относящиеся к организации взаимодействия SIMULINK и Stateflow, рассмотрены в главе 5.

Если маскированная подсистема является в то же время и библиотечной, то ее имени предшествует символ [ML].

В правой части окна броузера выводится список блоков, входящих в состав подсистемы, выделенной в левом списке. Если какой-либо блок является подсистемой следующего уровня вложенности, то возле его имени также могут присутствовать пояснительные символы из числа рассмотренных выше (кроме знака +).

Двойной щелчок ЛКМ на имени блока в правом списке приводит к открытию окна настроек данного блока (т. е. это действие аналогично двойному щелчку ЛКМ на иконке блока в блок-диаграмме). Тот же результат дает и нажатие кнопки *Open System*, расположенной ниже.

Тип выбранного блока отображается под правым списком в строке *Block Type*. Информация о типе блока оказывается полезной в том случае, если разработчик *S*-модели использовал для обозначения блока в диаграмме «нестандартное» имя.

Кроме упомянутой выше кнопки *Open System*, диалоговое окно броузера содержит еще ряд элементов управления:

• флажок *Look Under [M]ask Dialog*, который позволяет изменять статус маскированных подсистем:

если он установлен, то такая подсистема считается обычным блоком, «заглянуть» в нее невозможно, и ее имя выводится только в списке блоков (в правой части окна);

если флажок снят, то имя подсистемы выводится в левом списке (с символом [M]), а ее состав отображается в правом списке;

• флажок *Expand [L]ibrary Links* (*Связи с расширением библиотеки*) позволяет изменять статус подсистем, которые взяты из расширенной библиотеки (т. е. из библиотеки пользователя); используется он так же, как флажок *Look Under [M]ask Dialog*;

• кнопка *Look Into System* (*Взгляд внутрь системы*) доступна только в том случае, если в правом списке выбрана маскированная подсистема (т. е. подсистема с признаком [M] или [ML]); при «нажатии» кнопки открывается окно блок-диаграммы выбранной подсистемы.

Каждому из четырех рассмотренных элементов управления соответствует одноименная команда (опция) из раздела *Options* меню броузера (*Open System*, *Look into system*, *Look Under* [*M*]*ask Dialog*, *Expand* [*L*]*ibrary Links*).

В этом разделе имеется еще одна команда, не упоминавшаяся ранее — *Expand All* (*Развернуть все*); она позволяет просмотреть полную иерархию *S*-модели. При ее выполнении для подсистем, помеченных в левом списке символом «+», выводятся имена вложенных подсистем всех уровней.

В нижней части окна броузера расположены три дополнительные «сервисные» кнопки:

Help — вывод справки по броузеру (открывается соответствующий раздел встроенной справочной системы пакета MATLAB);

Print — вывод на печать информации о модели; команда выполняется так же, как одноименная команда из раздела *File*; особенности ее использования будут рассмотрены ниже.

Close — закрытие окна броузера.

Пункт ***Print*** меню раздела *File* позволяет выводить на печать блок-диаграмму модели и некоторую дополнительную информацию по ней.

При выполнении этой команды открывается диалоговое окно, обеспечивающее настройку параметров печати (рис. 4.3). Окно состоит из двух основных полей: *Printer* и *Options*.

Первое поле содержит стандартные параметры настройки принтера из среды Windows (тип принтера, логическое имя порта вывода и т. д.).

Поле *Options* является специфическим именно для вывода информации об *S*-модели. На нем расположены следующие элементы управления:

- переключатели уровня иерархии, с помощью которых можно выбрать перечень компонентов *S*-модели (подсистем), которые будут распечатаны в виде блок-диаграммы; возможны следующие варианты:

Current system — печать только активного (текущего) окна блок-диаграммы;

Current system and above — печать текущего окна и всех подсистем более высокого уровня, в состав которых входит выбранная подсистема;

Current system and below — печать выбранной подсистемы и блок-диаграмм всех входящих в нее подсистем;

All systems — печать блок-диаграмм всех подсистем, входящих в состав *S*-модели.

- флажок *Include Print Log* (*Добавить печать реестра*) позволяет вывести на печать в текстовой форме список блоков и подсистем, входящих в состав модели;

- флажок *Look Under Mask Dialog* обеспечивает разрешение/запрет печати блок-диаграмм маскированных подсистем;

- флажок *Expand Unique Library Links* (*Раскрыть связи с личной библиотекой*) позволяет разрешить/запретить печать блок-диаграмм подсистем из личной (дополнительной) библиотеки пользователя; при этом для библиотечных блоков (подсистем) блок-диаграмма печатается лишь однажды, сколько бы раз они ни встречались в модели.

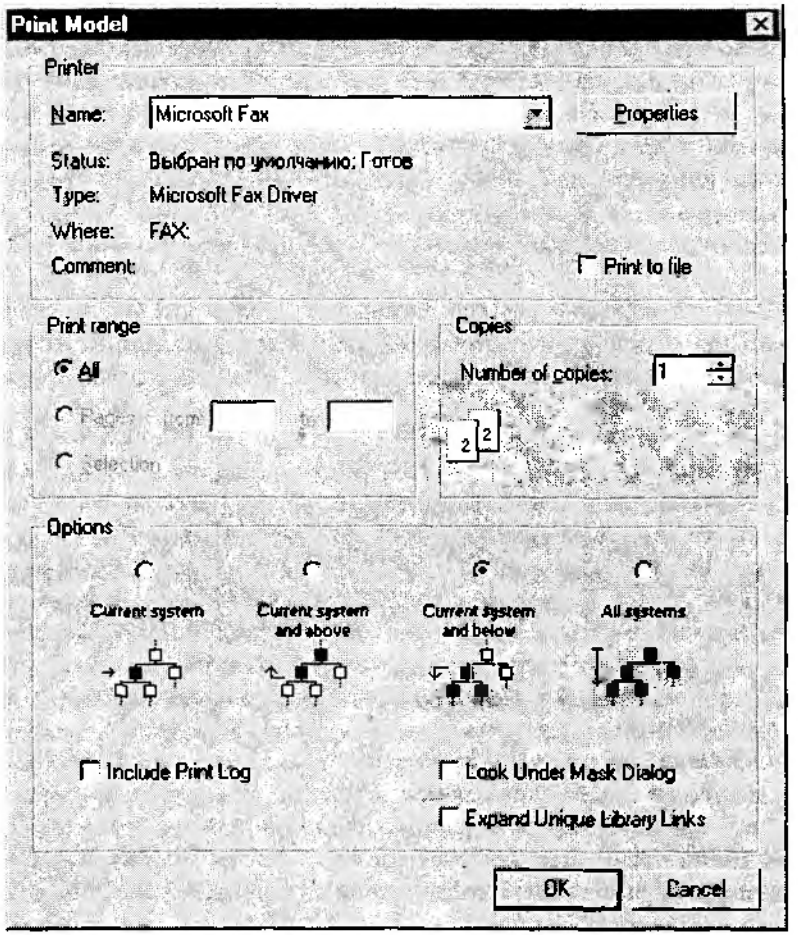


Рис. 4.3. Окно установки параметров печати блок-диаграммы

Два последних флажка становятся доступными только в том случае, если выбраны уровни иерархии печати *Current system and below* или *All systems*.

Print setup — обеспечивает настройку параметров вывода на печать с помощью стандартного окна Windows.

Раздел меню пользователя *Edit*

В данном разделе содержатся команды, обеспечивающие редактирование (изменение) структуры модели. Некоторые из них являются традиционными для Windows-приложений командами редактирования, другие относятся к специфическим командам работы с *S*-моделью (рис. 4.4).

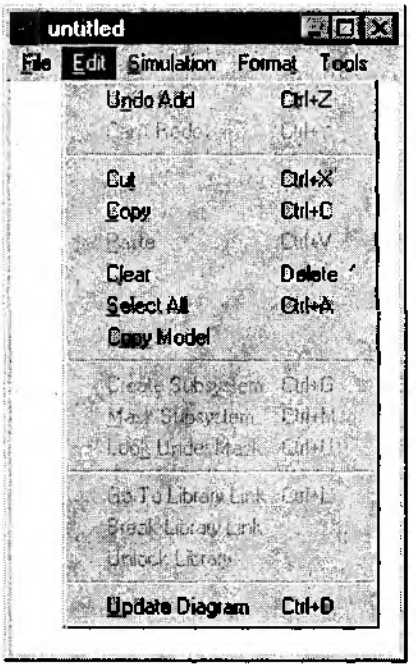


Рис. 4.4. Раздел ***Edit*** меню пользователя

В раздел ***Edit*** входят следующие пункты:

Undo – отменить предыдущую команду редактирования; в некоторых случаях команда Undo может конкретизироваться; например, после добавления в блок-диаграмму линии связи между блоками она называется *Undo Add Line* (*Отменить добавление линии*); если нельзя отменить предыдущее действие, то команда *Undo* заменяется сообщением *Can't Undo*.

Redo – отменить выполнение команды *Undo*; эта команда также может видоизменяться – либо конкретизироваться (например, *Redo Add Line*), либо сообщать о невозможности отмены (*Can't Redo*);

Cut – вырезать (забрать в буфер обмена) один или несколько блоков; соответствующие блоки должны быть выделены; чтобы выделить группу блоков, необходимо поместить курсор мыши рядом с одним из них и нажать ЛКМ, при этом курсор примет форму крестика; не отпуская кнопку, протащите курсор в сторону блоков, подлежащих выделению; за курсором потянется пунктирный прямоугольник, ограничивающий выделяемую область; после того, как в него попадут нужные блоки, отпустите ЛКМ; выделенные блоки, а также линии связи между ними будут помечены по углам черными прямоугольничками;

Copy – копировать один или несколько блоков; копируемые блоки должны быть предварительно выделены; данная команда используется совместно с командой ***Paste***.

Paste — вставить копируемый или удаленный в буфер обмена участок *S*-модели; перед выполнением этой команды необходимо предварительно щелкнуть ЛКМ в том месте окна блок-диаграммы, куда требуется произвести вставку (этой точке будет соответствовать верхний левый угол вставляемой области).

Замечание: команды *Cut*, *Copy* и *Paste* окна блок-диаграммы работают через собственный буфер обмена MATLAB, поэтому вставка копируемой или вырезанной области может выполняться многократно, пока не обновится содержимое буфера обмена; при этом вставку можно выполнять не только в пределах одного окна, но и в любую открытую блок-диаграмму; вместе с тем, передача содержимого собственного буфера MATLAB в другие приложения невозможна; для этого должны использоваться другие средства, которые будут рассмотрены немного позже.

Clear — очистить (удалить) выделенную область; (область в буфере обмена не сохраняется, но может быть восстановлена с помощью команды *Undo*);

Select All — выделить все элементы блок-диаграммы;

Copy Model — запись блок-диаграммы модели в буфер обмена Windows (*Clipboard*) для передачи в другие Windows-приложения (в качестве графического объекта); по умолчанию блок-диаграмма сохраняется в формате WMF (*Windows Metafile*), формат может быть изменен из командного окна MATLAB (раздел меню *File*, пункт *Preferences*).

Create Subsystem — создать подсистему; по этой команде выделенная часть *S*-модели (один или несколько блоков) «сворачиваются» в подсистему и заменяются в блок-диаграмме одним блоком — **Subsystem**; применение данной команды требует определенной осторожности, поскольку отменить результат невозможно (команда *Undo* в этом случае бессильна);

Mask Subsystem — вызов редактора «маски» подсистемы; команда доступна, если в блок-диаграмме выделена подсистема (блок типа Subsystem); технология создания маскированных подсистем подробно рассматривается в следующем разделе; если выделенная подсистема уже имеет «маску», т. е. является маскированной, то команда *Mask Subsystem* принимает вид *Edit Mask*;

Look Under Mask (*Заглянуть под маску*) — команда открывает окно блок-диаграммы маскированной подсистемы; доступна только в том случае, если выделенный блок является маскированной подсистемой;

Go To Library Link (*Перейти к связанной библиотеке*) — команда открывает раздел библиотеки, к которому относится выделенный блок; доступна только в том случае, если блок взят из библиотеки пользователя или из раздела *Simulink Extras*;

Break Library Link (*Разорвать связь с библиотекой*) — команда позволяет сделать библиотечный блок «самостоятельным», не связанным с библиотекой, что обеспечивает возможность его редактирования; данная команда работает для тех же разделов библиотеки, что и предыдущая;

Unlock Library (*Открыть библиотеку*) — команда доступна только в окне раздела библиотеки (*Library:*); после ее выполнения становится возможным редактирование соответствующего раздела, при этом на месте команды выводится при-

знак *Library Unlocked* (*Библиотека открыта*), который сохраняется до закрытия окна редактируемого раздела;

Update Diagram — обновить блок-диаграмму; команду необходимо использовать в следующих случаях:

- после изменения (редактирования) библиотечных блоков, копии которых используются в модели;
- после добавления в конфигурацию MATLAB нового раздела библиотеки, блоки из которого используются в открытой модели;
- после изменения параметров одного или нескольких блоков модели из командного окна MATLAB;
- после изменения *S*-функции, используемой в модели (при добавлении или удалении входных и/или выходных портов соответствующего блока).

Раздел меню пользователя *Simulation*

Данный раздел содержит команды управления сеансом моделирования. Раздел *Simulation* доступен только из окна блок-диаграммы модели или подсистемы и заблокирован в окне библиотеки (*Library*).

В его состав входят следующие команды (рис. 4.5).

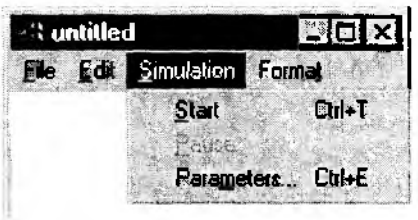


Рис. 4.5. Раздел *Simulation* меню пользователя

Start — запуск модели на исполнение; при запуске модели команда Start заменяется командой Stop, которая позволяет завершить моделирование досрочно (то есть либо до истечения заданного интервала моделирования, либо до реализации предусмотренных условий окончания сеанса моделирования);

Pause — приостановить сеанс моделирования; команда становится доступной после запуска модели на исполнение; при прерывании моделирования с помощью команды Pause она заменяется альтернативной командой — *Continue* (*Продолжить*);

Parameters... — по данной команде открывается диалоговое окно настроек параметров моделирования. Окно содержит три вкладки:

- *Solver* (*Установка параметров расчета модели*);
- *Workspace1/O* (*Установка параметров обмена данными с рабочей областью* MATLAB);
- *Diagnostics* (*Выбор уровня диагностики*).

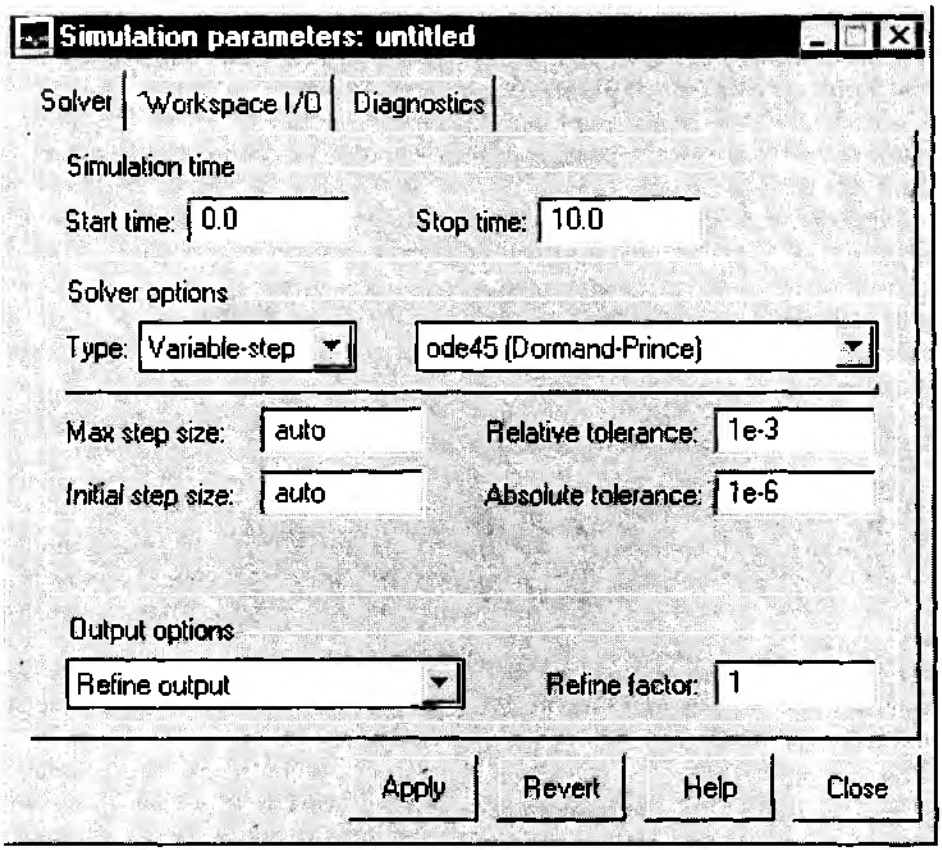


Рис. 4.6. Вкладка *Solver* диалогового окна установки параметров моделирования

На вкладке ***Solver*** могут быть произведены следующие установки (рис. 4.6):

Simulation time (*Интервал моделирования*); его величина задается посредством указания начального (*Start time*) и конечного (*Stop time*) значений модельного времени;

Solver options — выбор метода реализации (расчета) модели; речь идет о том, что имея структуру исследуемой системы в виде блок-диаграммы, разработчик может в ходе моделирования выбирать метод отображения динамики системы.

С помощью двух выпадающих меню система может быть реализована в следующих формах:

- с дискретными состояниями и дискретным временем перехода из одного состояния в другое;
- с дискретными состояниями и непрерывным временем переходов;
- с непрерывными состояниями и дискретным временем переходов;
- с непрерывными состояниями и непрерывным временем переходов.

Первое меню — *Type* — позволяет выбрать способ изменения модельного времени; оно содержит два пункта:

- *Variable-step* (*Моделирование с переменным шагом*);
- *Fixed-step* (*Моделирование с фиксированным шагом*).

Как правило, *Variable-step* используется для моделирования непрерывных систем, а *Fixed-step* — дискретных.

Второе меню, расположенное справа, позволяет выбрать метод расчета нового состояния системы. Первый пункт меню (*discrete*) обеспечивает расчет дискретных состояний системы (и для непрерывного, и для дискретного времени переходов из состояния в состояние).

Остальные 5 пунктов меню обеспечивают выбор метода расчета нового состояния для непрерывных систем. Эти методы различны для непрерывного времени (*Variable-step*) и для дискретного времени (*Fixed-step*), но основаны на единой методике — решении обыкновенных дифференциальных уравнений (ODE).

Подробное описание каждого из методов расчета состояний системы приведено во встроенной справочной системе MATLAB (раздел *matlab\fun\fun*).

Под окном меню *Type* находится строка редактирования, название которой изменяется в зависимости от выбранного способа изменения модельного времени. Для *Fixed-step* она называется *Fixed-step size* и позволяет указывать величину шага моделирования.

При выборе *Variable-step* данная строка получает имя *Max step size* (*Максимальная величина шага*) и, соответственно, содержит предельное допустимое значение шага моделирования.

По умолчанию величина шага моделирования для обоих способов изменения модельного времени устанавливается системой автоматически (*auto*) в соответствии с параметрами настройки блоков модели; требуемая величина шага может быть введена вместо значения auto либо в форме числовой константы, либо в виде вычисляемого выражения.

При моделировании непрерывных систем с использованием переменного шага (*Variable step*) необходимо указать точность вычислений: относительную (*Relative tolerance*) и абсолютную (*Absolute tolerance*). По умолчанию они равны соответственно $1 \cdot 10^{-3}$ и $1 \cdot 10^{-6}$. Кроме того, для указанного класса систем можно задать начальное значение шага моделирования (в поле *Initial step size*).

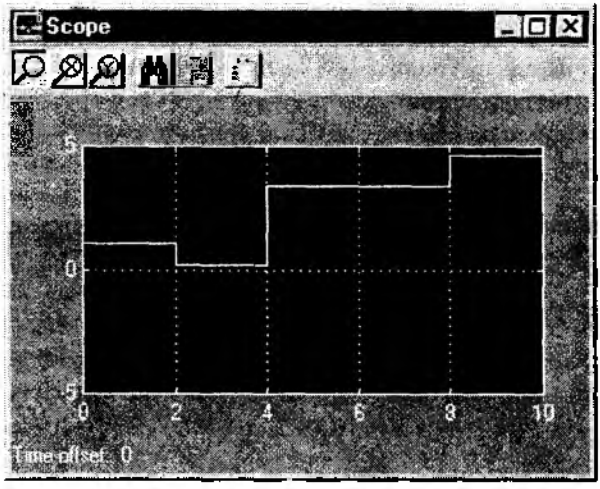
При моделировании с переменным шагом можно задавать опции вывода (*Options output*) выходных параметров моделируемой системы; соответствующие настройки выполняются с помощью двух элементов интерфейса: выпадающего меню и строки редактирования.

Меню, расположенное слева, содержит три пункта, особенности использования которых поясняются ниже.

Refine output (*Улучшенный вывод*) — при выборе этой опции можно изменять дискретность регистрации модельного времени и параметров модели (тех величин, которые сохраняются в рабочей области MATLAB с помощью блока **To Workspace**). Установка

дискретности выполняется в строке редактирования *Refine factor*, расположенной справа; по умолчанию значение *Refine factor* равно 1; это означает, что регистрация производится с шагом $\Delta t_M = 1$ (то есть для каждого очередного значения модельного времени: 0, 1, 2, 3 и т. д.); это наибольший возможный шаг регистрации. Значение *Refine factor* можно интерпретировать как относительную частоту регистрации: если, например *Refine factor* равен 2, это означает, что $\Delta t_M = 1/2$, если Refine factor равен 3, то $\Delta t_M = 1/3$ и т. д.; параметр *Refine factor* может быть только целым положительным числом.

а)



б)

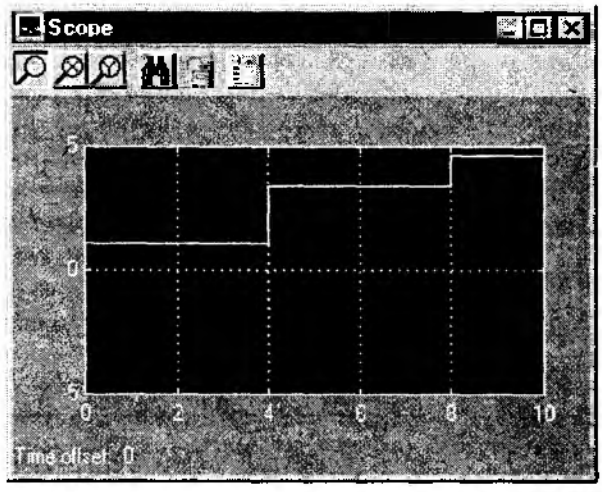


Рис. 4.7. Графики изменения параметров системы для двух значений опции *Options output*: а) *Refine factor* = 1; б) *Produce specified output only* = [4 8].

Второй пункт меню – *Produce additional output* (*создать дополнительный вывод*) – обеспечивает дополнительную регистрацию параметров модели в заданные моменты времени; их значения вводятся в строке редактирования (в этом случае она называется *Output times*) в виде списка, заключенного в квадратные скобки; при использовании дополнительных моментов регистрации базовый шаг регистрации (Δ_M) равен 1. Значения времени в списке *Output times* могут быть дробными числами и иметь любую точность.

Третий пункт меню – *Produce specified output only* (*формировать только заданный вывод*) – устанавливает вывод параметров модели только в заданные моменты времени, которые указываются в списке *Output times*.

На рис. 4.7 показано различие в расчете параметров модели для двух значений опции *Options output*.

Вкладка ***Workspace I/O*** содержит элементы управления вводом и выводом модельной информации в рабочую область MATLAB (рис. 4.8).

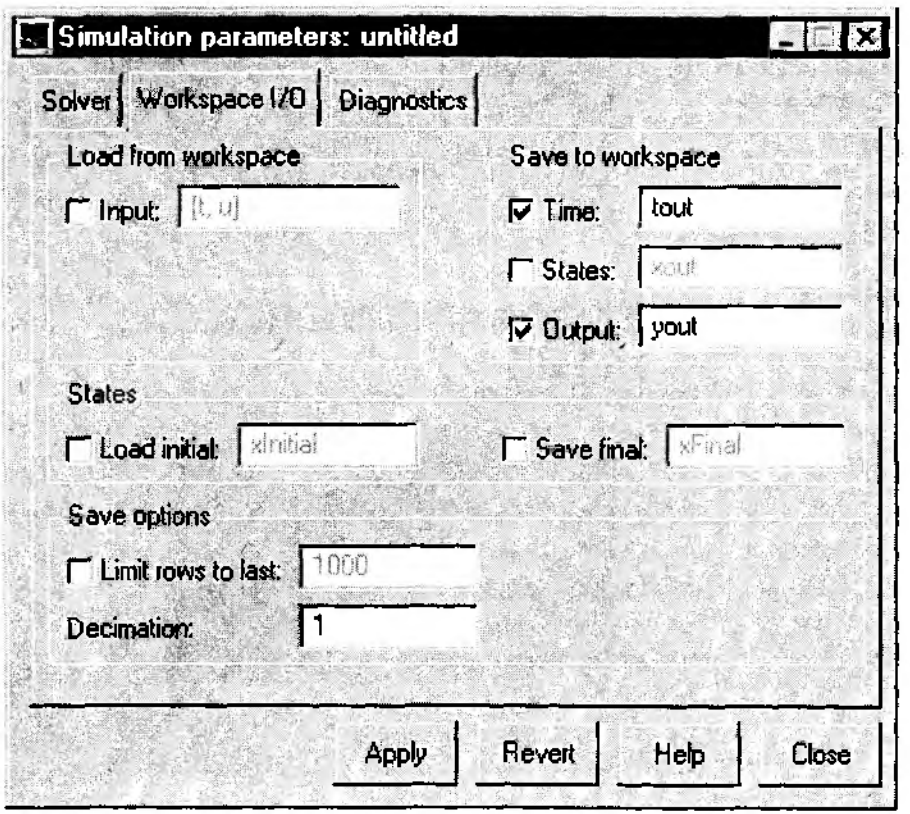


Рис. 4.8. Вкладка *Workspace I/O* диалогового окна установки параметров моделирования

Элементы управления вводом/выводом разбиты на 4 поля:
Load from workspace (*Загрузить из рабочей области*);
Save to workspace (*Записать в рабочую область*);
States (*Состояние*);
Save options (*Опции записи*).

Первое поле содержит флажок *Input* и строку ввода; при поставленном флажке можно ввести список параметров модели, которые будут считываться из рабочей области MATLAB; к списку предъявляются те же требования, что и при использовании блока **From Workspace**.

Поле *Save to workspace* содержит три флажка и связанные с ними строки ввода:

Time — позволяет указать имя вектора, в котором будут сохраняться регистрируемые моменты модельного времени (по умолчанию вектор имеет имя *tout*);

States — при установленном флажке в матрице *xout* сохраняется текущее состояние моделируемой системы. Оно описывается совокупностью значений сигналов на выходах блоков, формирующих (вычисляющих) текущее состояние системы; при моделировании дискретных систем такими блоками являются, в частности, все блоки раздела *Discrete*. SIMULINK автоматически определяет число указанных блоков в модели и на основании этого формирует матрицу xout. Число столбцов в матрице равно числу регистрируемых параметров текущего состояния системы, а число строк — числу актов регистрации; при желании разработчик может изменить имя матрицы;

Output — флажок записи в рабочую область — определяет возможность регистрации дополнительных параметров модели в ходе моделирования; если флажок установлен, то в соответствующей строке ввода необходимо указать имя регистрируемого параметра; в модели должен присутствовать блок **To Workspace**, обеспечивающий запись значений данного параметра в рабочую область MATLAB.

Поле *States* содержит два флажка, совмещенных со строками ввода:

Load initial (*загрузить начальное значение*) — при установленном флажке имеется возможность задать начальное значение параметров, характеризующих состояние моделируемой системы; указанные параметры считываются при запуске модели из вектора-строки *xInitial*. Число элементов этой строки (то есть число параметров состояния) должно быть равно числу столбцов (числу регистрируемых параметров) в матрице *xout*; значения элементов строки *xInitial* можно ввести в командном окне MATLAB следующим образом:

$$xInitial = [1, 2, 0.15];$$

в данном примере предполагается, что состояние системы описывается тремя параметрами, первый из них в момент запуска модели примет значение, равное 1, второй — равное 2, третий — равное 0.15; при необходимости имя *xInitial*, используемое по умолчанию, может быть изменено;

Save final (*Записать конечное состояние*) — при установленном флажке в рабочей области MATLAB сохраняются только последние значения параметров

состояния системы; они записываются в вектор-строку *xFinal*, которую можно либо вывести в командном окне MATLAB, либо использовать в качестве описания исходного состояния системы для последующих сеансов моделирования; имя *xFinal* также может быть изменено разработчиком модели;

Поле *Save options* (опции записи) позволяет задать ограничения на число строк матрицы *xout* (*Limit rows to last*), а также дополнительно указать необходимость «прореживания» (*Decimation*) регистрируемых состояний системы. Имеется в виду следующее. Если значение опции *Decimation* равно 1, то регистрация производится для всех моментов времени, заданных в поле *Output options* на вкладке *Solver*; если *Decimation* = 2, то регистрируется только каждое второе состояние системы, при *Decimation* = 3 — только каждое третье состояние и т. д.

В связи с этим необходимо подчеркнуть, что параметры, задаваемые в поле *Output options* на вкладке *Solver* и параметры, задаваемые в поле *Save options* на вкладке *Workspace I/O*, взаимосвязаны. Объем регистрируемой информации определяется выбранным сочетанием значений указанных параметров.

Например, если в поле *Output options* указано, что состояние системы должно регистрироваться в моменты времени [1, 5, 9, 11, 25], а в поле *Save options* заданы:

Limit rows to last: 4;
Decimation: 2,

то в матрице *xout* будут записаны состояния системы в моменты времени t=5 и t=11.

Вкладка ***Diagnostics*** (см. рис. 4.9.)позволяет изменять номенклатуру диагностических сообщений, выводимых SIMULINKом в командном окне MATLAB.

Указанные действия выполняются с помощью списка событий (*Events*), требующих реакции SIMULINK, и элементов управления. Переключатели в поле *Action* (*Действие*) становятся доступными, если в списке *Events* выбрано одно из событий. С помощью данных переключателей можно указать один из видов реакции на событие:

None — игнорировать;
Warning — выдать предупреждение;
Error — выдать сообщение об ошибке.

В поле *Debugging* (*Отладка*) можно дополнительно установить опцию порядок проверки согласованности компонент модели (*Consistency checking*) и идентификацию пересечения сигналом нулевого уровня (*Disable zero crossing detection*).

Раздел меню Format

В данном разделе меню пользователя собраны команды, позволяющие изменить внешнее представление (оформление) блок-диаграммы. По предназначению команды разделены на четыре группы (рис. 4.10).

Первую группу образуют команды, действие которых распространяется на выделенный блок (или группу блоков):

Font... — выбор шрифта для текстовой информации, выводимой на иконке блока, и для метки (имени) блока;

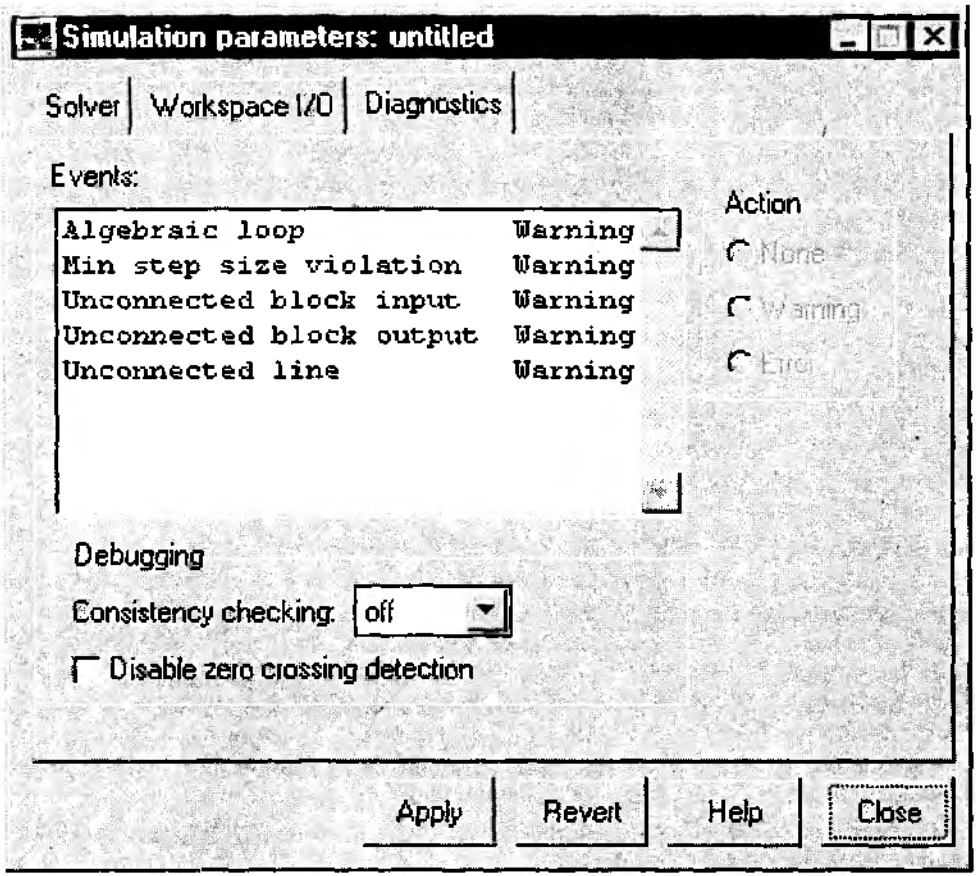


Рис. 4.9. Вкладка *Diagnostics* диалогового окна установки параметров моделирования

Flip Name — изменить положение имени блока (над или под иконой);

Hide Name/Show Name — скрыть/показать имя блока;

Flip Block — развернуть иконку блока относительно вертикальной оси симметрии на 180°;

Rotate Block — повернуть иконку блока относительно вертикальной оси симметрии на 90° (по часовой стрелке);

Show/Hide Drop Shadow — показать/скрыть «тень».

Ко второй группе отнесена единственная команда — ***Hide/Show Port Labels*** — скрыть/показать метки портов блока; данная команда применима только к блокам-подсистемам (**Subsystem**), содержащим внутренние входные или выходные порты, то есть блоки **In** и **Out**; если такие блоки имеются в подсистеме, то их метки по умолчанию выводятся на иконке блока-подсистемы.

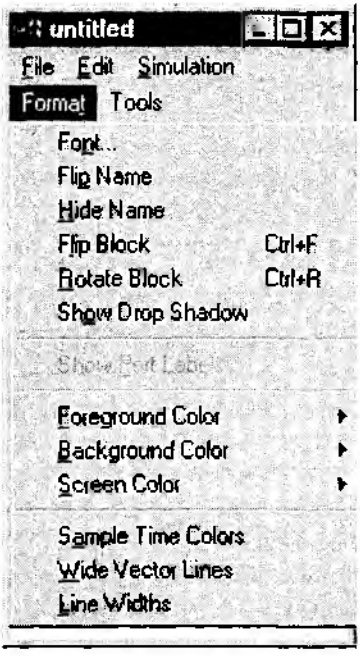


Рис. 4.10. Раздел ***Format*** меню пользователя

Третью группу образуют команды изменения цветовой палитры блок-диаграммы:

Foreground Color — выбор цвета контура и символов на иконке выделенного блока (группы блоков);

Background Color — выбор цвета фона иконки выделенного блока (группы блоков);

Screen Color — выбор цвета фона блок-диаграммы.

При выборе любой из этих трех команд на экране появляется список цветов, с помощью которого можно выполнить назначение.

Команды четвертой группы позволяют изменять некоторые параметры линий связи блок-диаграммы:

Sample Time Colors — «подсветка» линий связи, соединяющих блоки, работа которых зависит от величины шага модельного времени; при выполнении данной команды соответствующие линии связи (и сами блоки) выделяются на блок-диаграмме красным цветом;

Wide Vector Lines — линии связи, по которым передаются векторные величины, выводятся более «жирными»;

Line Widths — для всех линий связи выводится в цифровой форма «ширина» передаваемого сигнала (то есть число элементов соответствующего вектора), для скалярных сигналов «ширина» равна 1.

На рис. 4.11 показан результат применения двух последних опций к блок-диаграмме, приводившейся ранее (на рис. 3.55).

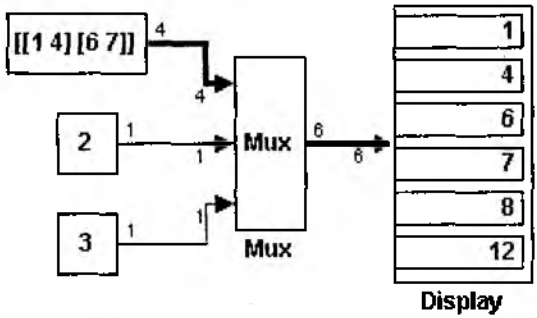


Рис. 4.11. Визуализация «векторных» линий связи

Разумеется, рассмотренные опции управления форматом вывода линий связи полезны не только с точки зрения дизайна создаваемой блок-диаграммы. Основное их назначение — оказание помощи разработчику *S*-модели на этапе ее отладки. В связи с этим необходимо отметить, что одной из весьма распространенных ошибок при построении *S*-модели, выявляемых SIMULINK, является несоответствие размерности входных и выходных портов соседних блоков. В случае ее обнаружения SIMULINK принудительно выводит на блок-диаграмме «ширину» векторных сигналов.

4.2.1. «ПЕРЕТАЩИ И ОСТАВЬ»

Возможность использования технологии *Drag-and-Drop* (*Перетащи и Оставь*) одно из основных достоинств практически всех Windows-приложений. Смысл ее заключается в том, что пользователь может «перетаскивать» (или, как говорят профессионалы, «буксировать») выделенный объект из одного окна в другое. При этом окна могут принадлежать как одному, так и разным приложениям.

Реализация технологии *Drag-and-Drop* возможна только благодаря особенностям ОС Windows: многозадачности и наличию графического (WIMP) интерфейса. Аббревиатура WIMP образована начальными буквами четырех английских слов: *Windows* (*Окна*), *Icons* (*Иконка, пиктограмма*), *Mouse* (*Мышь*), и *Pop-up* (*Выпадающее меню*). Эти термины и соответствующие им объекты составляют основу любого многооконного графического интерфейса.

Какая же связь между понятием WIMP-интерфейса и разработкой моделей с помощью SIMULINK?

Самая непосредственная: для визуального программирования необходимо наличие как минимум двух открытых окон. При начальном запуске SIMULINK та-

кими окнами, например, являются окно разделов библиотеки (*Library: Simulink*) и пустое окно блок-диаграммы модели (*untitled*).

После этих предварительных замечаний вернемся к решаемой задаче.

Итак, необходимо создать модель, описывающую поведение конечного детерминированного автомата по имени Маша. Как вы помните, простейший способ сделать это — воспользоваться блоком **Combinatorial Logic** из раздела библиотеки *Nonlinear*. Откройте указанный раздел (двойной щелчок ЛКМ) и отыщите в нем нужный блок. Поместите курсор на иконку блока, нажмите ЛКМ и, не отпуская ее, отбуксируйте изображение блока в поле *untitled*. Отпустите кнопку мыши. В здание будущей *S*-модели положен первый «кирпич». Поскольку размер «постройки» заранее неизвестен, целесообразно разместить его в центре окна блок-диаграммы. Если сразу это сделать не удалось, переместите блок в нужное место с помощью «мыши».

На следующем шаге целесообразно добавить в модель элементы, соответствующие двум другим персонажам рассматриваемой задачи — Васе и Пете. Пока будем считать, что их привычки характеризуются изрядным постоянством. Поэтому каждого из них в модели будет представлять блок **Constant** из раздела библиотеки *Sources*. Откройте этот раздел (двойной щелчок ЛКМ), найдите нужный блок и отбуксируйте его в левую часть поля блок-диаграммы. Чтобы поместить в блок-диаграмму второй блок **Constant**, есть три пути:

- повторить еще раз операцию буксировки блока из раздела библиотеки;
- воспользоваться командами *Копировать* (*Copy*) и *Вставить* (*Paste*) из раздела меню *Edit*;
- использовать дополнительные сервисные возможности SIMULINK.

Пойдем по третьему пути.

Поместите курсор на иконку блока **Constant** в блок-диаграмме создаваемой модели и нажмите правую клавишу мыши. Справа от курсора появится белый крестик. Не отпуская ЛКМ, переместите курсор в ту точку диаграммы, куда нужно вставить второй блок **Constant**. Отпустив ЛКМ, вы увидите его. Обратите внимание, что имя (метка) этого блока несколько изменилась: он называется **Constant1**. Объясняется это тем, что каждый блок, включенный в блок-диаграмму, должен иметь уникальное имя. Если бы в модели использовалось пять блоков **Constant**, то последний из них получил бы имя **Constant4**. Аналогичным образом именуются все блоки, используемые в модели.

Чтобы создаваемая модель более наглядно отражала существо рассматриваемой задачи, заменим метки блоков именами действующих персонажей. Начнем с блока **Combinatorial logic**. Поместите курсор на название блока и щелкните ЛКМ. Название блока окажется заключенным в прямоугольную рамку, а курсор примет форму вертикальной мерцающей черты. Используя клавиатуру, удалите стандартное имя блока и замените его именем «Маша». Щелкните ЛКМ вне текстовой области. Выделяющая рамка исчезнет. Выполните аналогичные действия с блоками **Constant** и **Constant1** (замените их названия именами «Вася» и «Петя» соответственно).

Замечание: все блоки в блок-диаграмме обязательно должны иметь имя — либо предложенное SIMULINK, либо назначенное разработчиком модели; поэтому нельзя сделать блок «безымянным», используя клавиши *Del* или *Backspace*; имя блока можно только «скрыть» с помощью команды *Hide Name* (из раздела меню *Format*).

Вернемся к созданию модели. Блок **Combinatorial logic** имеет ту особенность, что на его вход может поступать только один сигнал. А Маша должна получить информацию и о поведении Васи, и о поведении Пети. Как быть? Необходимо объединить два сигнала в один. Эту задачу можно решить с помощью блока **Mux** (раздел Connections). Найдите его и поместите в блок-диаграмме слева от блока *Маша*. Чтобы избавиться от лишнего входа в библиотечном варианте блока **Mux**, необходимо изменить его параметры настройки.

Поместите курсор на изображение блока и дважды щелкните ЛКМ. В открывшемся окне настроек измените значение параметра *Number of input* (*Число входов*) с 3 на 2. «Нажмите» кнопку *Apply*, затем — *Close*. Иконка блока изменится в соответствии с новым значением параметра настройки.

Теперь можно соединить блоки, помещенные в поле блок-диаграммы, в соответствии с логикой их взаимодействия: сигналы с выходов блоков *Вася* и *Петя* должны поступать на входы блока **Mux**, а объединенный сигнал с его выхода — передаваться на вход блока *Маша*. Для более удобного соединения блоков *Вася* и *Петя* с блоком **Mux** целесообразно немного «растянуть» последний по вертикали. Выделите указанный блок, подведите курсор к одному из его углов и «потяните» изображение в вертикальном направлении. Одновременно можно сократить блок по ширине. После соединения блоков модель будет выглядеть приблизительно так, как показано на рис. 4.12.

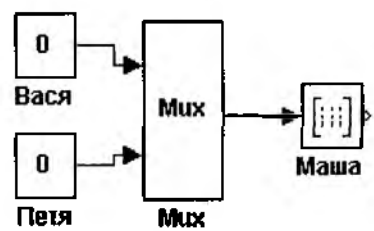


Рис. 4.12. Блок-диаграмма первой модели

Однако проводить эксперимент с полученной моделью пока рано:

- во-первых, мы не описали логику поведения Маши для различных значений входного сигнала;
- во-вторых, в модели отсутствуют средства, позволяющие увидеть результаты моделирования.

Сначала изменим таблицу истинности блока *Маша*. Откройте окно настроек параметров блока (двойной щелчок ЛКМ на иконке блока). В соответствии с пра-

вилами поведения героини, рассмотренными нами ранее, параметр *Truth table* должен быть задан следующим образом:

[001; 011; 101; 011].

Напомним, что значения входного сигнала в явном виде в таблице истинности не приводятся, задается только значение выходного сигнала.

Для отображения результатов работы модели используем блок **Display** из раздела *Sinks*. Отыщите указанный блок и поместите в поле блок-диаграммы, правее блока *Маша*. Соедините вход блока **Display** с выходом блока *Маша* (рис. 4.13).

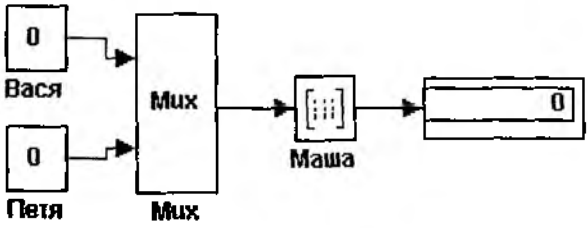


Рис. 4.13. Включение в блок-диаграмму блока **Display**

Создание модели завершено. Конечно, ее внешний вид и форму представления результатов можно улучшить, но этим мы займемся немного позже.

Сейчас давайте убедимся в том, что полученная модель действительно работоспособна.

Значения констант *Вася* и *Петя* оставим равными 0 (то есть будем считать, что ни тот, ни другой цветы принести не догадался).

Используя команду *Start* из раздела *Simulation* или соответствующую кнопку панели инструментов, запустите модель на исполнение.

Поскольку модель весьма проста, сеанс моделирования заканчивается очень быстро, о чем свидетельствует характерный звуковой «щелчок» компьютера. Кроме того, в правом нижнем углу блока **Display** должен появиться небольшой черный треугольник. Напомним, он служит признаком того, что выходной сигнал является векторной величиной. В рассматриваемом примере выходной вектор состоит из трех элементов (в соответствии с заданной таблицей истинности). Чтобы увидеть значения всех элементов вектора, необходимо «раздвинуть» блок **Display** по вертикали. Для этого блок следует выделить, подвести курсор к одному из его нижних углов и, нажав ЛКМ, переместить курсор вниз. Когда значения всех трех элементов станут видимы, отпустите кнопку мыши. Как и следовало ожидать, элементы вектора равны соответственно 0, 0, 1 (рис. 4.14).

В созданной нами модели практически все блоки использовались с параметрами, установленными по умолчанию. То же самое можно сказать и о параметрах модели в целом. В двух последующих разделах речь пойдет о том,

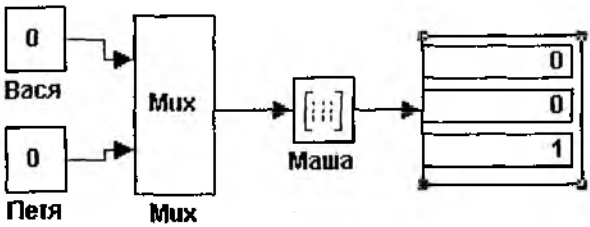


Рис. 4.14. Представление результатов работы модели с помощью блока **Display**

каким образом можно скорректировать параметры модели, чтобы они наиболее полно отвечали как интересам исследования, так и эстетическим пристрастиям разработчика.

Что можно изменить

Прежде всего необходимо сохранить созданную модель (она нам еще понадобится), записав файл с блок-диаграммой на жесткий диск или на дискету.

Воспользуйтесь для этого командой *Save as...* Выберите каталог (папку), куда будет записываться файл, и какое-нибудь имя для него, более содержательное, чем *untitled* например, *Choice* —(*Выбор*).

После сохранения файла модели сделайте активным командное окно MATLAB. Во время сеанса моделирования там появилось новое сообщение: «*Using variable step discrete time solver for model 'untitled'*» (*Использование переменного шага приращения дискретного времени при расчете модели 'untitled'*). Данное сообщение не является предупреждением о том, что мы сделали что-то неправильно. Это скорее тактичный намек на то, что при моделировании дискретных процессов и систем более корректным является использование фиксированного шага приращения модельного времени (*Fixed-step*).

Чтобы указанное сообщение в дальнейшем не появлялось, необходимо изменить параметры сеанса моделирования.

Откройте в окне блок-диаграммы модели раздел меню *Simulation* и выберите команду *Parameters...* На вкладке *Solver* в поле *Solver options* с помощью выпадающего меню *Type* замените *Variable-step* на *Fixed-step*. При этом в расположенном справа окне автоматически будет установлен метод расчета состояний *discrete* (рис. 4.15).

Еще один параметр, который может быть скорректирован — это величина интервала моделирования. Очевидно, что рассматриваемый конечный автомат формирует результат за один шаг работы модели. Поэтому изменим значение поля *Stop time* с 10 на 1 (на самом деле его можно сделать еще меньше, но об этом немного позже).

Чтобы установленные параметры вступили в силу, «нажмите» кнопку *Apply*, находящуюся в нижней части окна *Simulation parameters*.

Вернемся к блок-диаграмме модели *Выбор*. С любой моделью тем удобнее и приятнее работать, чем больше она содержит различных текстовых комментари-

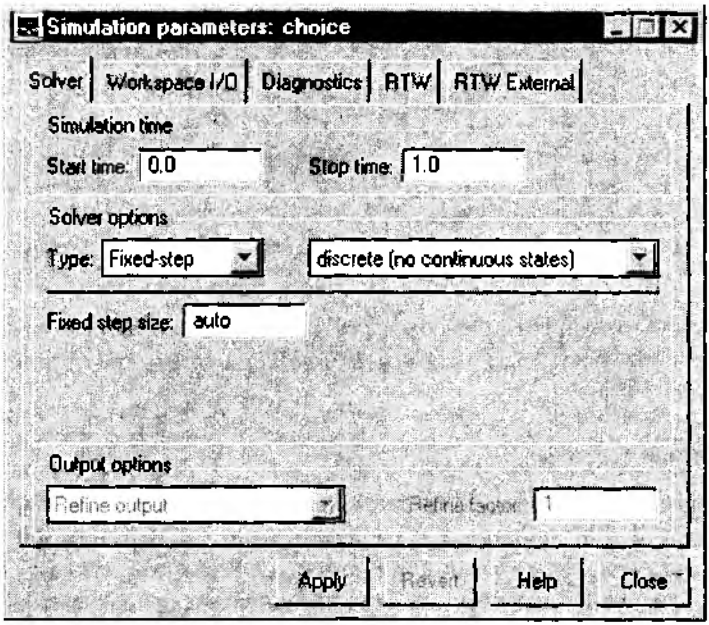


Рис. 4.15. Окно настройки параметров моделирования для модели *Choice* (*Выбор*)

ев. Причем, чем сложнее модель, тем больше должно быть пояснений по ее использованию. Созданная нами модель *Выбор* к сложным явно не относится, тем не менее снабдим ее дополнительной текстовой информацией.

Начнем с заголовка. Лучше всего разместить его там, где обычно находятся все заголовки — в верхней части окна. Если в вашем варианте модели места там недостаточно, выполните следующие подготовительные действия. В разделе *Edit* выберите команду *Select All*. В результате ее выполнения все элементы блок-диаграммы (и блоки, и линии связи) окажутся выделенными. Подведите курсор «мыши» к любому из элементов модели и нажмите ЛКМ. Не отпуская ее, переместите курсор вниз. Вы увидите, что контуры блоков диаграммы также будут перемещаться. Когда они займут нужное положение, отпустите ЛКМ. После этого диаграмма будет воспроизведена на новом месте.

Установите курсор мыши в ту точку окна, где, по вашему мнению, должен будет находиться центр заголовка, и щелкните дважды ЛКМ. В окне появятся прямоугольная выделяющая рамка и курсор, обозначающий позицию ввода первого символа. Введите с клавиатуры заголовок, который, на ваш взгляд, отображал бы назначение модели (например, «Исследование конечного автомата» или «Выбор жениха»). Закончив ввод, щелкните ЛКМ вне текстовой области. По умолчанию для ввода текста используется шрифт с размером символов 10 пикселов, что для заголовка явно недостаточно. Чтобы изменить тип шрифта, вновь поместите курсор в текстовую область

и щелкните ЛКМ (можно дважды). После этого в разделе меню *Format* выберите команду *Font...* Используя открывающееся диалоговое окно, установите подходящие тип, начертание и размер шрифта заголовка. Если после этого возникнет необходимость изменить его местоположение, выполните следующие действия: установите курсор на текстовую область, нажмите ЛКМ и, не отпуская ее, переместите курсор в нужном направлении; при этом контур текстовой области также будет перемещаться. Найдя удачное положение для заголовка, отпустите ЛКМ.

Аналогичным образом можно создать текстовые комментарии в любой точке окна блок-диаграммы (рис. 4.16).

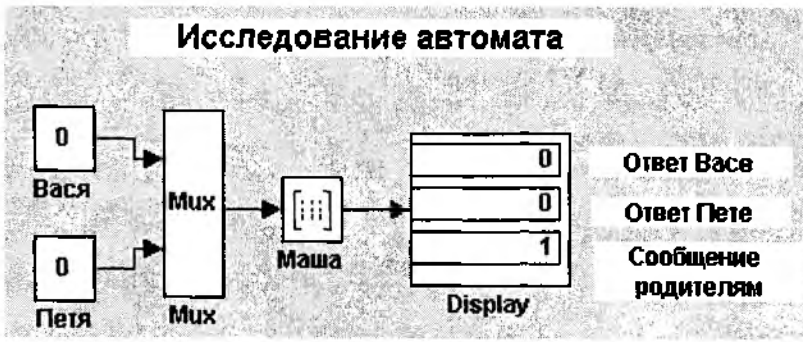


Рис. 4.16. Пример создания текстовых областей в блок-диаграмме

Теперь рассмотрим возможные способы изменения внешнего облика блоков диаграммы.

В большинстве *S*-моделей блок **Mux** имеет вспомогательное значение. Поэтому целесообразно несколько изменить иконку блока, приведя ее в соответствие с отводимой ему ролью.

Один из вариантов такой модификации приведен на рис. 4.17.

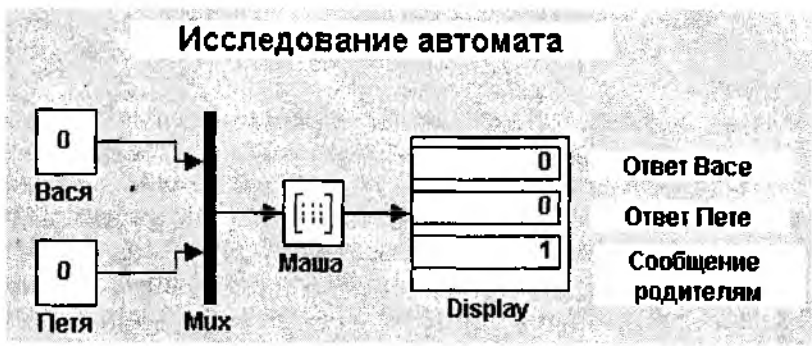


Рис. 4.17. Изменение формата блока **Mux**

Если Вы познакомились с материалом 2 главы, то, вероятно, заметили, что теперь блок **Mux** похож внешне на вершину-переход сети Петри (это наблюдение окажется весьма важным при рассмотрении ряда последующих примеров).

Блок **Mux** стал похож на вершину-переход благодаря двум операциям:

- первая — изменение геометрической формы иконки блока; эта операция вам уже знакома, она выполняется с помощью курсора «мыши»);
- вторая — изменение цвета фона иконки блока (*Background Color*); чтобы выполнить эту операцию, необходимо в разделе меню *Format* выбрать соответствующую команду, а в открывающемся дополнительном меню — цвет *Black* (*черный*).

Поскольку теперь блок **Mux** легко узнать на блок-диаграмме по внешнему виду, имя для него становится излишним. Чтобы убрать его, в том же разделе *Format* выберите команду *Hide Name*.

Имя блока **Display** в рассматриваемом примере также является мало информативным. Поэтому повторите операцию *Hide Name* и для указанного блока (блок должен быть предварительно выделен).

И наконец, последнее изменение, относящееся к внешнему облику блок-диаграммы. Так как в нашей модели используются векторные сигналы, целесообразно отразить этот факт визуально, воспользовавшись командой *Wide Vector Lines*. После применения данной команды линии связи между блоками **Mux**, *Маша* и **Display** станут вдвое толще других.

Замечание. Форматы линий связи и блока **Display** для векторного сигнала «проявляются» только после проведения хотя бы одного сеанса моделирования.

Наглядность блок-диаграмм, содержащих большое число блоков, во многом зависит от выбора формы и расположения всех элементов диаграммы, в том числе от конфигурации линий связи. Ниже описаны некоторые основные приемы их редактирования.

Для начала отметим, что редактируемая линия связи должна быть предварительно выделена. Чтобы выделить линию связи, достаточно подвести к ней курсор «мыши» и щелкнуть ЛКМ.

Необходимо также отметить, что при связывании блоков (то есть при создании новой линии связи) SIMULINK сам пытается подобрать наиболее удачную конфигурацию линии. При этом создаваемая линия может иметь два и более «изломов».

На выделенной линии связи «изломы» отмечаются черными прямоугольничками. Чтобы изменить форму линии, необходимо установить курсор «мыши» на точку излома (курсор превратиться в белую окружность), нажать ЛКМ и, не отпуская ее, переместить курсор в нужном направлении. Таким образом можно изменить длину прямых участков линии, либо вообще убрать лишние «изломы».

Чтобы изменить положение прямых отрезков, можно использовать другой способ. Установите курсор на редактируемый отрезок и нажмите ЛКМ; курсор примет форму черно-белого ромба; не отпуская ЛКМ, переместите курсор в нужном направлении (горизонтальном или вертикальном).

При создании новой линии связи или при перемещении связанных блоков SIMULINK старается выбрать для связи кратчайший маршрут. Если вы хотите самостоятельно указать маршрут связывания блоков, можно поступить следующим образом.

Подведите курсор к выходному порту блока, который вы хотите связать с одним из последующих блоков модели. Курсор примет форму черного крестика. Нажмите ЛКМ и, не отпуская ее, переместите курсор. За ним потянется контур будущей линии связи. Чтобы изменить направление движения, отпустите, затем вновь нажмите ЛКМ и продолжите движение в нужном направлении. Обратите внимание: перемещаемый конец линии связи оканчивается тонкой стрелкой. После того, как маршрут будет нарисован, входной порт присоединяемого блока нужно состыковать с упомянутой стрелкой. Если стыковка прошла успешно, стрелка примет уже знакомую вам форму.

Один и тот же блок может быть соединен по выходу с двумя или более блоками. Чтобы создать дополнительную линию связи, необходимо выполнить следующие действия.

1. Соединить блок с одним из блоков, на которые должен передаваться сигнал (созданную линию связи будем считать основной).

2. Подвести курсор к входному порту не подсоединенного блока и нажать ЛКМ; протащить контур новой линии связи до пересечения с основной; когда курсор примет форму двойного крестика, отпустить ЛКМ; в результате на основной линии связи появится точка соединения.

Если создается более одной дополнительной линии связи, то их можно замыкать как на уже созданную точку соединения, так и на любую другую точку основной или дополнительной линии связи.

При необходимости точка соединения может быть перемещена вдоль линии связи на другое место. Для этого следует установить на нее курсор, нажать ЛКМ и переместить точку соединения; выходящие из нее линии связи будут перерисованы автоматически.

Разумеется, к возможным изменениям созданной блок-диаграммы относятся также изменения цветовой гаммы и геометрических размеров входящих в нее блоков, использование «теней» и т. д. Особенности использования соответствующих команд читателю предлагается опробовать самостоятельно.

Что можно добавить

В данном подразделе речь пойдет уже не столько об изменении внешних атрибутов блок-диаграммы, сколько о возможных способах повышения технологичности модельного эксперимента.

Весьма существенным недостатком созданной нами модели является то, что она не позволяет сохранять результаты, полученные в ходе очередного сеанса моделирования.

Этот недостаток можно устранить с помощью блоков **To Workspace** и **To File**. Краткая характеристика их была дана при описании раздела библиотеки *Sinks*. Напомним, что блок **To Workspace** обеспечивает сохранение выходных и промежуточ-

ных результатов моделирования в рабочей области MATLAB, а блок **To File** — в файле на жестком диске или на дискете.

Рассмотрим особенности использования этих блоков применительно к модели *Выбор*.

Начнем с блока **To Workspace**.

Откройте раздел *Sinks*, найдите указанный блок и отбуксируйте его в блок-диаграмму модели. Подсоедините блок к линии связи, идущей от блока *Маша* к блоку **Display**, например так, как показано на рис. 4.18.

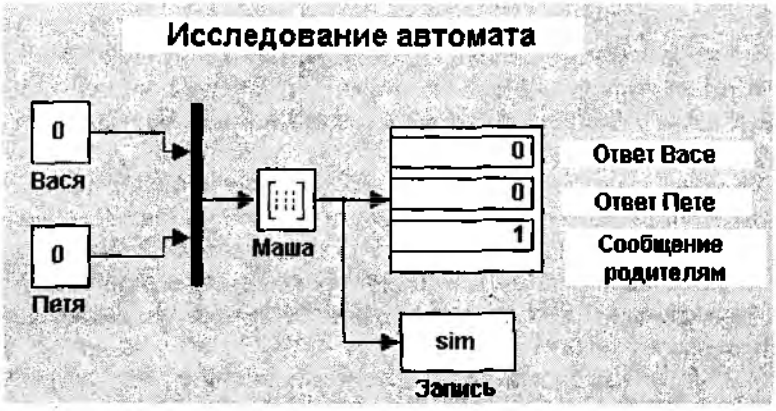


Рис. 4.18. Сохранение результатов моделирования с помощью блока **To Workspace**

Сохраняя русскоязычное направление в создании модели, заменим имя блока **To Workspace** словом *Запись*, после чего скорректируем параметры блока. Откройте окно настроек (двойной щелчок ЛКМ на иконке блока).

Первый параметр — *Variable name* — на работу блока не влияет и определяет только имя, под которым будет сохранена соответствующая величина (по умолчанию *simout*).

Имена переменных, используемых в *S*-модели, должны содержать только латинские символы. Поэтому в отношении первого параметра ограничимся тем, что сократим исходное имя до *Sim*. При нажатии кнопки *Apply* внесенное изменение отображается на иконке блока в блок-диаграмме модели.

Остальные параметры блока пока корректировать не будем. Закройте окно настроек и запустите модель на исполнение.

По окончании сеанса моделирования откройте командное окно MATLAB. В командной строке (т. е. в строке, содержащей приглашение >> и курсор), наберите имя переменной *Sim* и нажмите клавишу <Enter>.

В качестве ответа MATLAB выведет в окно содержимое матрицы *Sim*, имеющей размер 51×3.

Число элементов в строке (3) равно размерности выходного вектора, а число строк (51) определяется совокупностью параметров, задающих периодичность ре-

гистрации результатов моделирования. Поскольку все эти параметры сохранили значения, заданные по умолчанию, то мы получили максимально возможный объем регистрируемых данных: модельное время изменяется с шагом 0.02 и на каждом шаге производится регистрация результатов.

В нашем случае такой объем является избыточным, так как состояние модели, вычисленное на первом шаге, остается неизменным на всем интервале моделирования. Другими словами, вполне достаточно, чтобы матрица *Sim* содержала всего одну строку. Этого можно добиться двумя способами.

1) Задать нулевую длительность интервала моделирования; для этого в разделе меню *Simulation* необходимо выбрать команду *Parameters...* и на вкладке *Solver* установить значение *Stop time*, равное нулю;

2) В настройках блока **To Workspace** установить параметр *Maximum Number of rows* равным 1.

Во многих случаях требуется сохранять в рабочей области параметры, относящиеся к двум состояниям модели: начальному и конечному.

Для дискретной системы, подобной той, которая рассматривается в нашем примере, это можно выполнить несколькими способами:

1) задать величину шага модельного времени (поле *Fixed step size* на вкладке Solver), равную длительности интервала моделирования

2) установить шаг моделирования для блока **To Workspace** (параметр *Sample* time в окне настроек блока) равным интервалу моделирования;

3) указать максимальное число строк матрицы, сохраняемой в рабочей области (параметр *Maximum Number of rows* блока **To Workspace**), равное 2;

4) установить соответствующую дискретность записи данных в рабочую область (параметр *Decimation* блока **To Workspace**).

При необходимости можно просмотреть перечень и объем данных, сохраняемых в рабочей области, с помощью команды *Schow Workspace* из раздела меню *File* командного окна MATLAB (или использовав соответствующую кнопку панели меню этого окна). Вид окна рабочей области для рассматриваемого примера показан на рис. 4.19.

После просмотра содержимого рабочей области лишние данные могут быть удалены из нее. Для этого следует выделить переменные, подлежащие удалению (щелкнув на них ЛКМ), и затем «нажать» кнопку *Delete*, расположенную в нижней части окна рабочей области.

Кнопка *Open* позволяет загрузить данные из рабочей области в *Редактор/Отладчик* MATLAB в виде электронной таблицы, содержимое которой может корректироваться пользователем и затем вновь сохраняться в рабочей области (рис. 4.20).

Данные, необходимые для последующей работы, можно сохранить в MAT-файле с помощью команды *Save Workspace As...* из раздела меню *File* командного окна MATLAB.

В *S*-модели может использоваться либо только один из блоков **To Workspace** и **To File**, либо оба вместе. Однако, поскольку блок **To Workspace** является как бы ослабленной версией блока **To File**, то в сложной блок-диаграмме лучше стараться избегать дублирования.

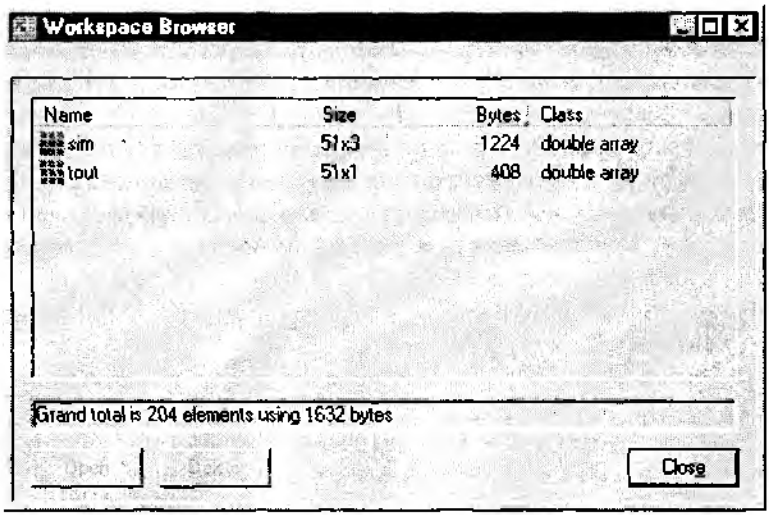


Рис. 4.19. Диалоговое окно рабочей области MATLAB после сохранения результатов моделирования

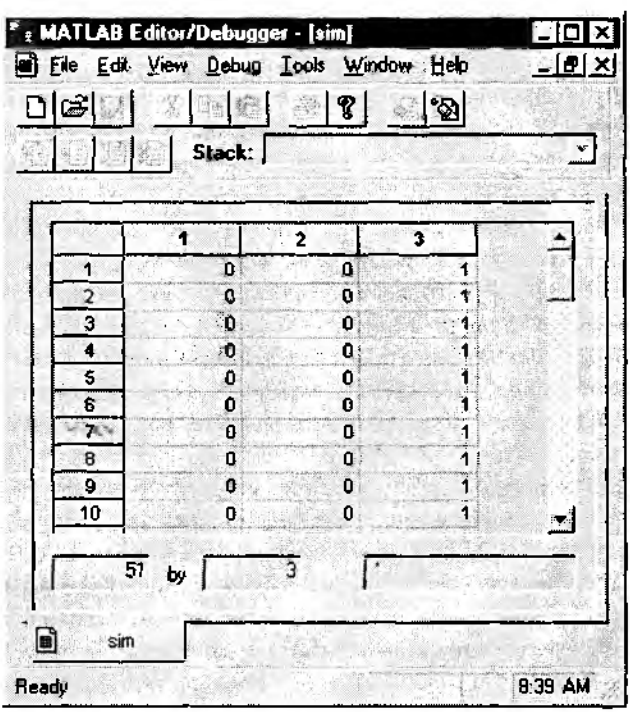


Рис. 4.20. Просмотр данных из рабочей области в окне *Редактора/Отладчика*

Чтобы разобраться с особенностями применения блока **To File**, заменим в модели *Выбор* блок **To Workspace** блоком **To File**; с этой целью выполните следующие действия:

• выделите в блок-диаграмме блок *Запись* (*To Workspace*) и нажмите на клавиатуре клавишу <Del>;

• в разделе *Sinks* отыщите блок **To File** и отбуксируйте его в окно блок-диаграммы модели;

пристыкуйте блок **To File** к свободному концу линии связи, оставшейся от блока **To Workspace** (рис. 4.21).

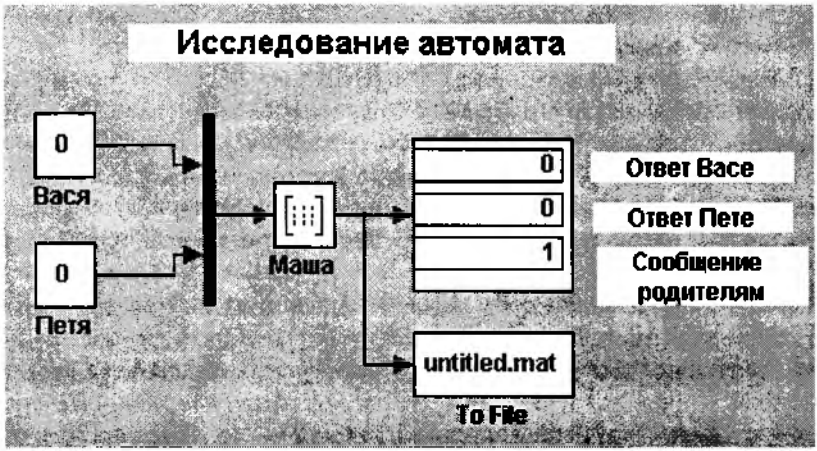


Рис. 4.21. Применение блока **To File** для сохранения результатов моделирования

Использование блока **To File** имеет следующие особенности:

1) при записи данных в файл они не дублируются в рабочей области MATLAB; поэтому для просмотра и редактирования сохраненных данных их предварительно необходимо загрузить в рабочую область с помощью команды *Load Workspace*, входящей в раздел меню *File* командного окна MATLAB);

2) структура матрицы, в которой сохраняются данные, несколько отличается от структуры матрицы, создаваемой при использовании блока **To Workspace**:

в первой строке матрицы записаны значения моментов времени регистрации данных, а остальные строки содержат значения регистрируемых величин; например, для рассматриваемой модели сохраняемая в файле матрица имеет размер 4×51 (при шаге времени регистрации, равном 0.02).

Вы можете изменять объем регистрируемых данных, используя практически те же средства, что и при работе с блоком **To Workspace**:

• увеличивая шаг моделирования *Fixed step size* на вкладке *Solver* в окне параметров моделирования;

• изменяя значения параметров *Decimation* и *Sample Time* блока **To File**.

До сих пор речь шла о сохранении результатов одного сеанса моделирования. Но, как вы знаете, для оценки поведения системы бывает полезно (и даже необходимо) иметь результаты всей серии модельных экспериментов.

MATLAB позволяет накапливать в рабочей области данные по нескольким экспериментам для их дальнейшей обработки и/или записи в файл.

Рассмотрим технологию накопления данных по серии экспериментов применительно к модели *Выбор*.

Поставим себе целью получить и зарегистрировать результаты моделирования для всех возможных сочетаний значений констант *Вася* и *Петя* (то есть, используя терминологию первой части книги, проведем полный факторный эксперимент). Поскольку каждая из констант может принимать одно из двух значений («0» или «1»), то общее число экспериментов будет $2^2=4$.

Итак, приступим к реализации плана эксперимента.

Замените в блок-диаграмме блок **To File** на блок **To Workspace**. Установите параметры моделирования и блока **To Workspace** таким образом, чтобы значение выходного вектора регистрировалось в матрице Sim только один раз.

Кроме того, в окне *Simulation parameters* на вкладке *Workspace I/0* в поле *Save to Workspace* снимите все флажки. Это предотвратит запись в рабочую область лишней информации.

Для начальных значений *Вася*=0 и *Петя*=0 выполните эксперимент. После его окончания откройте окно рабочей области и убедитесь, что регистрация прошла успешно.

Перед следующим запуском модели внесите в нее следующие изменения:

- установите новое значение для одной из констант (например, *Петя*=1);
- замените имя матрицы регистрации *Sim* на *Sim1* (в окне настроек блока **To Workspace**).

Изменяя аналогичным образом значения констант и имя матрицы регистрации, выполните два оставшихся эксперимента.

После окончания последнего эксперимента вновь откройте окно рабочей области; оно должно выглядеть так, как показано на рисунке 4.22.

Итак, в рабочей области сохранены результаты всех четырех экспериментов. Каким образом их можно использовать?

Во-первых, каждую из полученных записей можно просмотреть в командном окне MATLAB или в окне *Редактора/Отладчика*; если один из экспериментов оказался неудачным (например, вы забыли изменить значения констант), его результат можно удалить из рабочей области.

Во-вторых, все содержимое рабочей области можно сохранить в одном MAT-файле, используя команду *Save Workspace As...*

В-третьих, выполнить статистическую обработку результатов моделирования. О том, как это сделать, будет рассказано в Главе 5.

На любом этапе моделирования содержимое MAT-файла может быть вновь загружено в рабочую область с помощью команды *Load Workspace* и подвергнуто требуемой обработке.

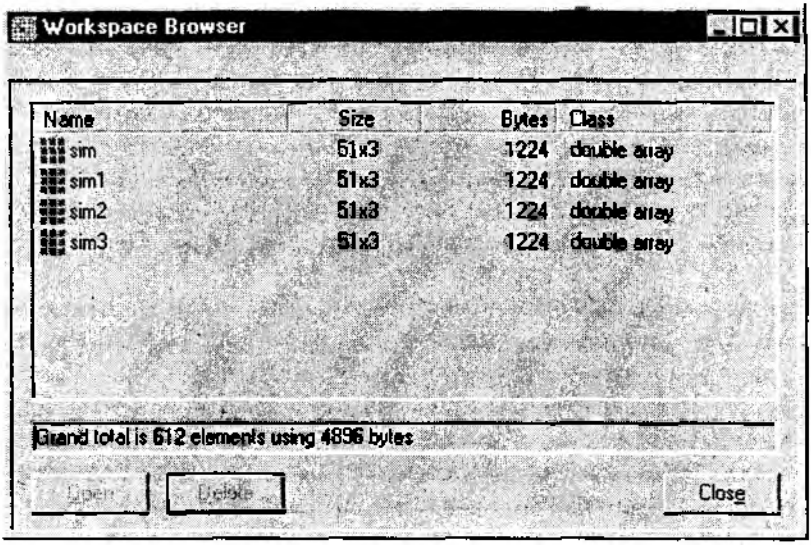


Рис. 4.22. Накопление результатов моделирования в рабочей области MATLAB

В частности, результаты двух или более экспериментов могут быть объединены в одну матрицу с помощью операции присваивания, выполняемой в командном окне MATLAB. Ниже приведен пример объединения данных по второму и третьему экспериментам в новой матрице *sim4*:

$$sim4 = [sim2, sim3].$$

Новая переменная также будет записана в рабочую область и может быть сохранена в MAT-файле вместо исходных записей *sim2* и *sim3*.

Еще один вариант сохранения результатов моделирования предоставляет блок **Scope**. Напомним, что данный блок представляет собой один из вариантов «смотровых окон» и обеспечивает отображение динамики изменения параметров модели в графической форме. Параметры настройки блока и некоторые особенности его использования были рассмотрены в главе 3. Сейчас остановимся только на одной из них, связанной с регистрацией данных.

Поместите блок **Scope** в блок-диаграмму модели *Выбор* и подсоедините его к выходу блока *Маша*. Откройте блок **Scope** (двойной щелчок ЛКМ на иконке блока). Обратите внимание: пределы измерения по оси времени (по оси *X*) совпадают с интервалом моделирования, установленным для модели *Выбор*.

Запустите модель на исполнение.

В окне **Scope** появятся прямые, соответствующие значениям элементов выходного вектора (рис. 4.23).

По умолчанию данные, выводимые в окне **Scope**, в рабочей области не сохраняются. Чтобы изменить установки, нажмите кнопку *Properties* в окне блока. На

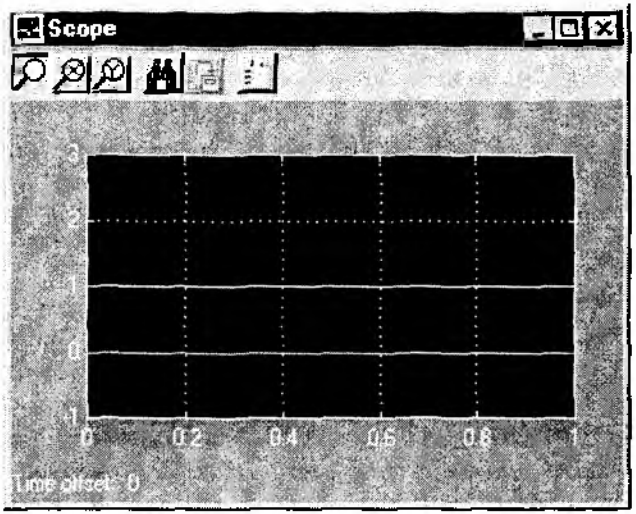


Рис. 4.23. Просмотр результатов моделирования в окне **Scope**

вкладке *Settings* поставьте флажок *Save data to workspace*. При этом станет доступным для редактирования поле ввода *Variable name* (имя, под которым будут сохранены данные в рабочей области). Его можно либо оставить без изменения — *Scope Data*, либо выбрать по своему усмотрению (например, просто *Data*) (рис. 4.24).

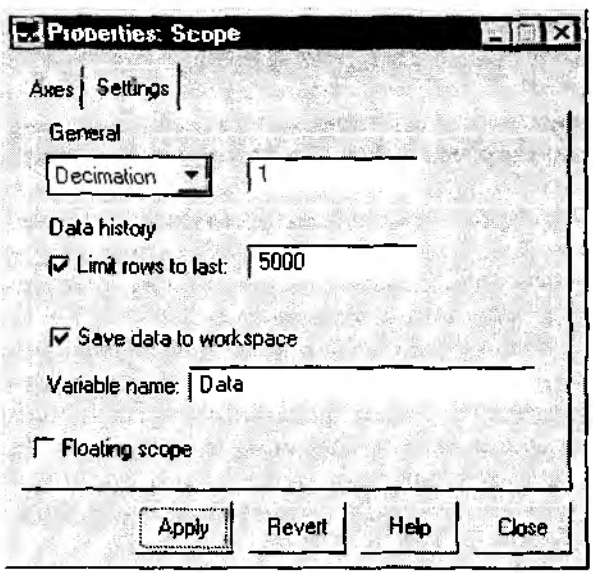


Рис. 4.24. Установка параметров сохранения результатов в рабочей области MATLAB

Нажмите кнопку *Apply* и повторите сеанс моделирования. Теперь данные, выводимые в окне **Scope**, будут сохранены в рабочей области в числовой форме. Просмотреть их можно стандартным образом: в командном окне MATLAB введите имя, под которым данные были сохранены, и нажмите клавишу <Enter>.

Полученный массив данных представляет собой матрицу размером 51 × 4 (если интервал моделирования равен 1). Число строк матрицы (51) определяется шагом регистрации данных. По умолчанию, как и для рассмотренных ранее блоков **To Workspace** и **To File**, регистрация производится через каждые 0.02 единицы модельного времени.

Число столбцов (4) на 1 больше числа регистрируемых величин, т. к. первый столбец используется для записи моментов времени регистрации (в порядке возрастания).

Управлять объемом регистрируемых данных можно как с помощью параметров моделирования (например, изменяя величину шага моделирования *Fixed step Size*), так и корректируя параметры самого блока **Scope**, расположенные на закладке Settings:

- изменяя периодичность регистрации данных, используя опции *Decimation* (*Прореживание*) и *Sample time* (*Величина шага регистрации*).
- ограничив максимальное число строк в матрице *ScopeData* (*Limit rows to last*); при этом в матрице будут записаны только результаты последних измерений.

Если указано предельное число строк, то данные выводятся в окне **Scope** только для тех интервалов времени, когда производится регистрация. На рис. 4.25 приведен вид окна **Scope** для случая, когда задано значение *Limit rows to last*, равное 10.

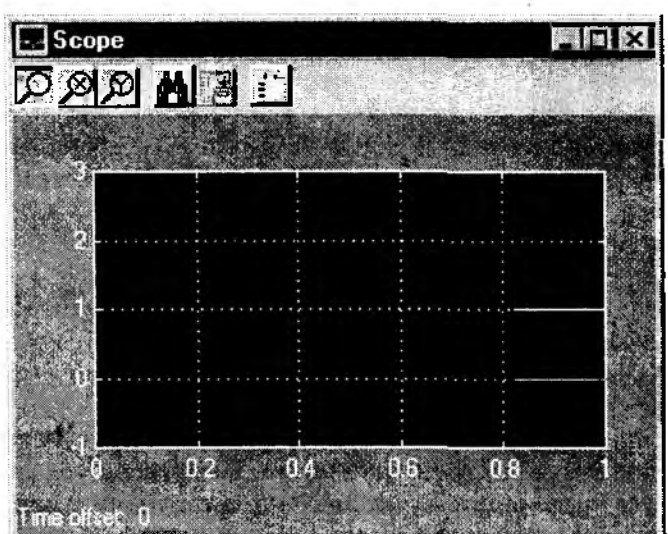


Рис. 4.25. Отображение в окне **Scope** регистрируемых данных

Существенным достоинством применения блока **Scope** для сохранения результатов моделирования является то, что формат матрицы *Scope Data* совпадает с форматом данных, используемых в блоке **From Workspace**.

Вспомнив о блоке **From Workspace**, мы тем самым перешли к обсуждению вопросов, связанных с использованием результатов моделирования.

Во многих реальных задачах результаты, полученные в текущем сеансе моделирования, используются в качестве исходных данных в последующих сеансах, либо во взаимодействующих *S*-моделях. Блоки **From Workspace** и **From File**, входящие в раздел библиотеки *Sources*, выполняют автоматическую загрузку данных из рабочей области MATLAB в исполняемую *S*-модель.

Но если форматы данных, используемых в блоках **To File** и **From File**, полностью совпадают, то в паре блоков **To Workspace** и **From Workspace** такая согласованность отсутствует.

Данные, загружаемые в модель с помощью блока **From Workspace**, должны иметь следующий формат:

$$\begin{bmatrix} t_1 U(1)_1 U(2)_1 \ldots U(m)_1 \\ t_2 U(1)_2 U(2)_2 \ldots U(m)_2 \\ \ldots \\ t_n U(1)_n U(2) n \ldots U(m)_n \end{bmatrix},$$

где t_i — моменты времени регистрации данных, а $U(j)_i$ — значение регистрируемой величины $U(j)$ в i-й момент времени.

При записи данных в рабочую область с помощью блока **To Workspace** моменты времени регистрации не запоминаются. Поэтому при использовании блока **From Workspace** необходимо либо предварительно сформировать вручную вектор-столбец, содержащий значения времени, либо применять для записи данных блок **Scope**.

Замечание: моменты регистрации данных в модели-источнике могут не совпадать с дискретностью расчета характеристик модели-получателя; в этом случае в модели-получателе выполняется линейная интерполяция данных, загруженных с помощью блока **From Workspace**.

С целью иллюстрации применения блока **From Workspace** несколько доработаем модель *Выбор*.

Сначала дадим содержательную трактовку последующего развития событий в рассматриваемой ситуации. Предположим, что Маша дает ответ своим претендентам не сразу, а через некоторое время, причем каждому в отдельности (в письменном виде), и уже потом звонит родителям.

Для описания такой ситуации создадим дополнительную *S*-модель, логика работы которой будет состоять в следующем:

1. Выходной сигнал, сформированный в модели *Выбор*, разделяется на три компоненты (*Ответ Васе*, *Ответ Пете* и *Сообщение родителям*).

2. Значения первых двух компонент выводятся одновременно в окнах двух блоков **Display** (каждое в своем).

3. На следующем шаге работы модели в третьем окне **Display** выводится значение компоненты *Сообщение родителям*.

В соответствии с описанной логикой работы новая модель должна содержать следующие блоки:

- **From Workspace**,
- **Demux**,
- **Display** (три экземпляра),
- **Memory**.

Чтобы ее создать, необходимо выполнить уже знакомую вам последовательность действий:

1. В разделе меню **File** блок-диаграммы модели *Выбор* выберите пункты ***New→Model***.

2. В новое (пустое) окно *untitled* перетащите из библиотеки SIMULINK перечисленные выше блоки (блок **Demux** находится в разделе *Connections*, а блок **Memory** – в разделе *Nonlinear*).

3. Измените формат (внешний вид) блока **Demux** аналогично тому, как был ранее изменен в модели *Выбор* блок **Mux**.

4. Соедините блоки в соответствии с логикой работы модели.

Блок-диаграмма модели должна выглядеть примерно так, как показано на рис. 4.26.

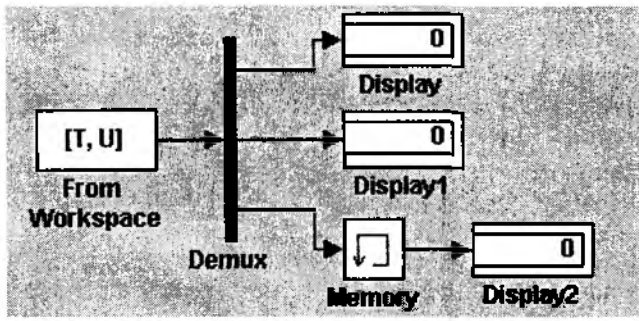


Рис. 4.26. Блок-диаграмма модели, использующей данные из рабочей области MATLAB

Если внешний вид модели вас удовлетворяет, сохраните ее в файле на диске, присвоив ему какое-либо осмысленное имя (например, *Answer* – **Ответ**).

Теперь займемся внутренними параметрами блоков, включенных в состав модели, точнее, одного из них – блока **From Workspace**. Он должен считывать из рабочей области MATLAB значение выходного сигнала модели *Выбор*, записанного туда

с помощью блока **Scope**. Согласование формата общих данных — весьма тонкий и ответственный момент, поэтому остановимся на нем несколько подробнее.

Для работы модели *Ответ* в качестве исходных данных необходимо и достаточно иметь конечный результат работы модели *Выбор*. Другими словами, упоминавшаяся ранее матрица *Data* должна содержать единственную строку, соответствующую значению выходного сигнала на момент окончания интервала моделирования. Для этого параметр *Limit rows to last* блока **Scope** должен быть равен 1.

Чтобы матрица *Data* стала доступна блоку **From Workspace**, необходимо указать ее имя в качестве параметра блока. Когда это будет выполнено, имя *Data* появится на иконке блока.

Остальные блоки модели в настройке не нуждаются. Но, вероятно, нуждается в пояснении роль блока **Memory**. Данный блок задерживает сигнал, поступающий на его вход, на один шаг модельного времени. В течение данного интервала на выход блока выдается значение, указанное в качестве его параметра *Initial condition*. По умолчанию оно равно нулю, что нас вполне устраивает. Правда, пока не известна величина шага моделирования. Неизвестны также длительность интервала моделирования, способ изменения модельного времени и используемый метод расчета состояний модели *Ответ*. Можно, конечно, предположить, какие значения получат указанные параметры по умолчанию. Но в данном случае лучше проявить (и закрепить) полученные ранее знания и навыки и установить самим требуемые значения параметров моделирования.

Итак, открываем окно настройки параметров модели *Ответ* (в разделе *Simulation* выберите пункт *Parameters...*).

На вкладке *Solver* устанавливаем:

- *Stop time*: 1
- *Type*: *Fixed step*;
- *Fixed step size*: 0.2 (чисто волюнтаристское решение).

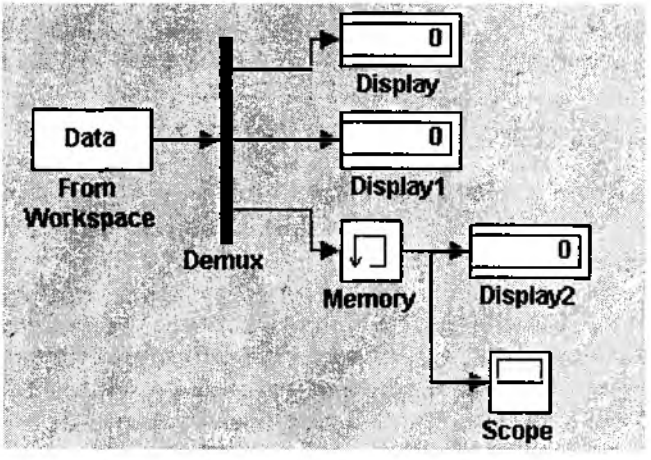


Рис. 4.27. Применение окна **Scope** для контроля за работой блока **Memory**

На вкладке *Workspace I/O* в поле *Save to workspace* можно поставить флажки *Time* и *Output* (хотя это необязательно).

На вкладке *Diagnostics* целесообразно убрать выдачу предупреждений при наличии неподсоединенных блоков и линий связи.

Внеся соответствующие изменения, «нажмите» кнопку *Apply* и закройте окно *Simulation parameters*.

Теперь можно приступать к моделированию. Запустите на исполнение сначала модель *Выбор*, затем — *Ответ*.

Сравните значения, выведенные в окнах блока **Display** модели *Выбор* с «показаниями» блоков **Display** модели **Ответ**. Если они совпадают, вы все сделали правильно. Правда, задержку *сообщения родителям* визуально заметить практически невозможно.

Чтобы убедиться в том, что блок **Memory** справляется со своей задачей, дополним модель *Ответ* смотровым окном в виде блока **Scope** (рис. 4.27).

Поместите его из раздела *Sinks* в поле блок-диаграммы и подключите к выходу блока *Memory* (параллельно с блоком **Display2**). Откройте окно **Scope** (двойной щелчок на иконке блока) и запустите модель *Ответ* на исполнение. График, выведенный в окне **Scope** (рис. 4.28), подтверждает корректную работу модели.

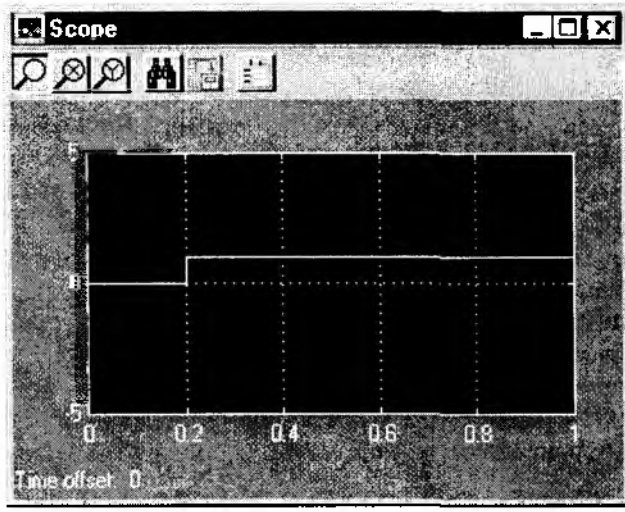


Рис. 4.28. Временная диаграмма работы блока **Memory**

Подводя итоги изложенному в данном разделе, можно сформулировать ***общую схему разработки моделей в среде SIMULINK***:

1. Описать на содержательном уровне логику поведения исследуемой системы.

2. Определить перечень блоков, соответствующих основным компонентам системы, а также необходимых для расчета выбранного показателя эффективности.

3. Поместить в окно блок-диаграммы требуемые блоки и соединить их между собой в соответствии с логикой работы модели.

4. Выполнить настройку и согласование параметров блоков.

5. Установить требуемые значения параметров моделирования для модели в целом.

6. Дополнить (обязательно!) блок-диаграмму модели необходимыми текстовыми комментариями.

7. Сохранить блок-диаграмму модели в файле на диске (до первого запуска модели на исполнение).

8. Выполнить сеанс моделирования.

По окончании последнего шага может оказаться, что вы чего-то не учли или сделали не совсем то, что хотели (или совсем не то). В связи с этим уместно еще раз напомнить, что моделирование – процесс творческий. Чтобы разработанная модель удовлетворяла предъявляемым к ней требованиям как по формальным, так и по субъективным критериям, может оказаться необходимым неоднократное повторное выполнение одного или даже нескольких шагов приведенной выше схемы.

4.2. УСЛОЖНЯЕМ ЗАДАЧУ

Имитационное моделирование в равной степени пригодно и для решения детерминированных задач, и для исследования ситуаций, зависящих от случайных факторов. В первой части книги обсуждались методы учета случайных факторов при разработке моделей различных типов систем. Настало время поговорить о том, насколько средства SIMULINK обеспечивают реализацию этих методов.

При создании моделей с помощью универсальных языков программирования аппаратный или программный датчик СЧ – единственный инструмент разработчика для моделирования всех видов случайных факторов – случайных событий, случайных величин и случайных процессов.

У того, кто использует SIMULINK, арсенал значительно шире. Если же и этих средств SIMULINK оказывается недостаточно, в модель могут быть включены инструменты, входящие в состав ядра MATLAB или других компонентов пакета.

4.2.1. ВВОДИМ СЛУЧАЙНОЕ СОБЫТИЕ

Как вы помните, случайное событие A считается наступившим, если сформированное датчиком случайное число r попало в диапазон $[0; P_A]$, где P_A – вероятность данного события.

Следовательно, модель должна содержать компоненты, обеспечивающие выполнение трех действий:

- генерацию случайного числа r;

• сравнение r с P_A;

• идентификацию момента возникновения события A.

В *S*-модели указанные действия можно реализовать, используя совместно блоки ***Uniform Random Number*** (раздел *Sources*) и *Relational Operator* (раздел *Nonlinear*).

Чтобы первый из названных блоков генерировал случайные числа, равномерно распределенные на интервале [0; 1], необходимо произвести соответствующую установку его параметров:

Minimum (нижняя граница диапазона): 0,

Maximum (верхняя граница): 1.

Третий параметр — *Sample time* — в данном случае позволяет задавать количество СЧ, которые будут сформированы блоком в течение интервала моделирования: при *Sample time*, равном 0, новое СЧ генерируется через каждые 0.02 единицы модельного времени; при *Sample time*, равном интервалу моделирования, генерируется только одно СЧ.

На рис. 4.29 приведен пример блок-диаграммы, позволяющей моделировать появление случайного события A при P_A=0.4. Блок **Scope** в данном случае играет чисто иллюстративную роль и обеспечивает графическое представление генерируемой последовательности случайных чисел.

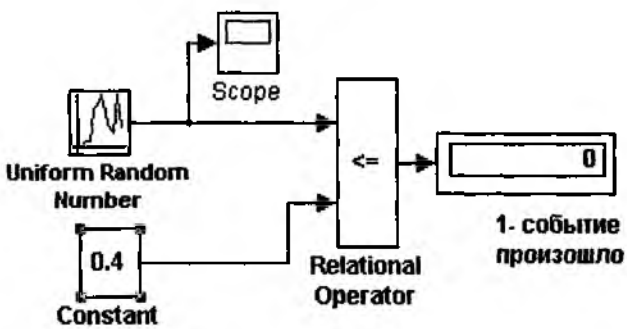


Рис. 4.29. Моделирование случайного события с заданной вероятностью наступления

Особенностью блока **Uniform Random Number** является то, что он в каждом сеансе моделирования генерирует одну и ту же последовательность СЧ. Для изменения генерируемой последовательности необходимо вручную изменить значение его параметра *Initial seed*. При проведении большого числа повторных экспериментов с целью накопления статистических данных это не очень удобно. Поэтому для моделирования случайных событий можно воспользоваться генераторами СЧ, входящими в состав компоненты MATLAB, которая называется *Statistics Toolbox* (*Статистические инструменты*). Прежде чем мы рассмотрим применение генераторов СЧ для решения указанной задачи, несколько слов о *Statistics Toolbox*.

Данная компонента доступна для использования в том случае, если она включена в рабочую конфигурацию пакета MATLAB. Если это не было сделано при начальной инсталляции пакета, ее можно добавить, выполнив повторно частичную установку.

Компонента *Statistics Toolbox*, как и другие инструментальные приложения MATLAB, представляет собой набор специализированных функций, реализованных в виде М-файлов. Ее особенностью является то, что у нее отсутствует собственный набор блоков, который включался бы в раздел библиотеки SIMULINK *Blocksets & Toolboxes*. Поэтому в процессе моделирования статистические функции следует использовать одним из двух способов:

выполнять в командном окне MATLAB;

включать в вычисляемое выражение в тех блоках *S*-модели, в которых разрешено использование М-функций.

Все функции, входящие в *Statistics Toolbox*, разбиты на несколько категорий: параметрические распределения, средства планирования эксперимента, генераторы случайных чисел и другие. Информация по ним приведена во встроенной справочной системе MATLAB (раздел *toolbox\stats*).

Наиболее важные с точки зрения имитационного эксперимента функции будут рассмотрены в Главе 5 книги.

Сейчас остановимся на использовании функций, относящихся к категории *Random Number Generators* (генераторы случайных чисел).

Всего таких генераторов более 20, каждый из них обеспечивает формирование значений случайной величины, распределенной по определенному закону с задаваемыми параметрами.

Генератор непрерывной СВ, равномерно распределенной в заданном интервале, называется *unifrnd*. Вызов данной функции имеет вид *unifrnd* (*A*, *B*, *M*, *N*), где *A*, *B* — границы диапазона распределения, а параметры *M* и *N* задают размер генерируемой матрицы СЧ.

При моделировании случайного события эта функция может быть указана в качестве параметра настройки следующих блоков:

MATLAB Fcn (раздел *Nonlinear*);

Fcn (раздел *Nonlinear*);

Constant (раздел *Sources*).

На рис. 4.30 приведен пример блок-диаграммы, обеспечивающей моделирование случайного события с помощью блока **Fcn**; рядом показано окно настройки блока, содержащее обращение к генератору *unifrnd*.

В каждом сеансе моделирования генератор формирует новое случайное число. В этом можно убедиться с помощью дополнительного блока **Display**, включенного в блок-диаграмму.

В качестве иллюстрации к изложенному выше воспользуемся все той же моделью *Выбор*. Будем считать, что наличие букета у каждого из претендентов на руку Маши — событие случайное. Чтобы внести «элемент случайности» в модель, заменим значения констант *Вася* и *Петя* обращениями к генератору *unifrnd*. При этом

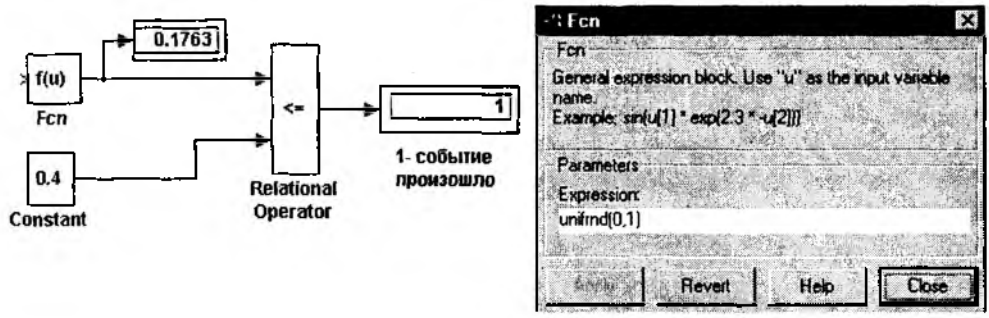


Рис. 4.30. Моделирование случайного события с помощью блока **Fcn**

внешний вид блок-диаграммы практически не изменится, за исключением того, что на иконках упомянутых блоков будет выведено имя генератора СЧ (рис. 4.31).

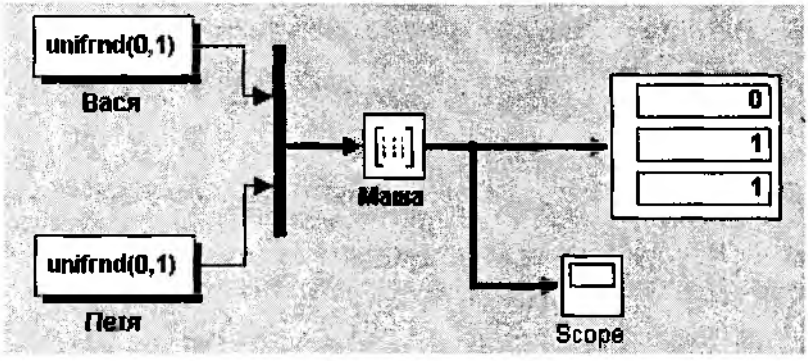


Рис. 4.31. Моделирование поведения «претендентов» как двух случайных событий

Теперь в каждом отдельном сеансе моделирования исход рассматриваемой ситуации действительно является случайным.

Появление случайного события можно интерпретировать еще одним способом: событие произошло, если некоторое связанное с ним число (признак) попало в заданный интервал или пересекло определенную границу.

Первый вариант может быть реализован с помощью блока **Relay**, второй — с помощью блока **Hit crossing** (оба блока относятся к разделу *Nonlinear*).

На рис. 4.32 приведена блок-диаграмма, которая позволяет подсчитать число попаданий некоторого признака в заданный интервал. Работа блок-диаграммы основана на использовании блока **Relay**. Остальные блоки выполняют следующие функции:

- **MATLAB Fcn** — генерирует последовательность СЧ, равномерно распределенных на интервале [0; 7];

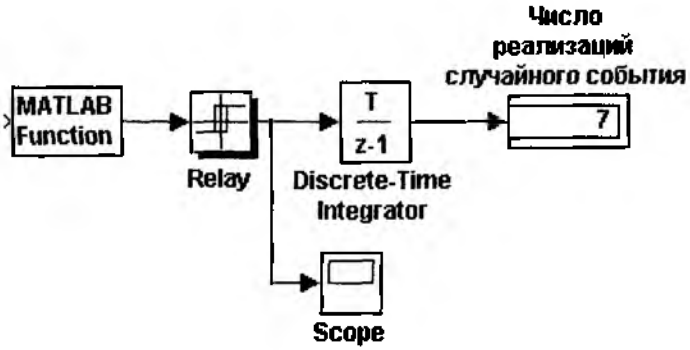


Рис. 4.32. Моделирование случайных событий с помощью блока **Relay**

- **Discrete-Time Integrator** — подсчитывает число попаданий СЧ в диапазон [0; 5];
- **Display** — выводит результат подсчета;
- **Scope** — позволяет наблюдать за работой блока **Relay**.

Для блока **Relay** установлены следующие значения параметров:

- *Switch on point*: 5
- *Switch off point*: 1
- *Output when on*: 0
- *Output when off*: 1.

Если исследователя интересует возможность хотя бы одной реализации случайного события на интервале моделирования, то удобнее воспользоваться блоком **Hit Crossing**.

На рис. 4.33 показан вариант блок-диаграммы, которая обеспечивает регистрацию момента превышения случайным числом заданного значения.

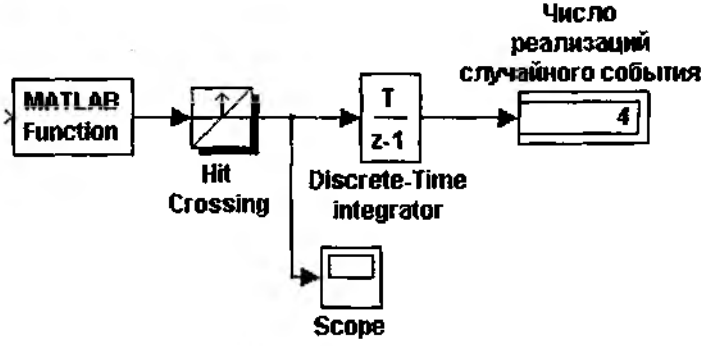


Рис. 4.33. Моделирование случайных событий с помощью блока **Hit crossing**

При этом параметры блока **Hit crossing** имеют значения:

- *Hit crossing offset*: 4;
- *Hit crossing detection*: *rising*.

Остальные блоки используются так же, как в предыдущей блок-диаграмме.

4.2.2. ПОДЧИНЯЕМ СЛУЧАЙНУЮ ВЕЛИЧИНУ ЗАДАННОМУ ЗАКОНУ

Использование случайных величин является наиболее универсальным и поэтому наиболее распространенным способом учета в модели случайных факторов, присущих реальным системам или процессам. Примерами случайных величин могут служить: интервал времени до появления очередного клиента, длительность проведения технического обслуживания автомобиля, объем данных, считываемых из оперативной памяти ЭВМ, и т. д.

Если закон распределения СВ известен, то она может быть достаточно адекватно представлена в имитационной модели. В первой части книги были рассмотрены методы, используемые для моделирования различных законов распределения СВ при разработке модели на одном из универсальных языков программирования.

При создании модели средствами MATLAB процедура отображения в ней СВ существенно упрощается. Разработчику достаточно иметь представление о том, какие генераторы случайных чисел входят в состав компоненты *Statistics Toolbox*. В предыдущем разделе уже было сказано, что таких генераторов более 20. Технология использования в *S*-модели любого из них одинакова и состоит в выполнении следующих действий:

- открыть встроенную справочную систему MATLAB (раздел *toolbox\stats*);
- в списке *Random Number Generator* выбрать функцию-генератор, соответствующую требуемому закону распределения;
- двойным щелчком ЛКМ на выбранной строке открыть страницу справочника, содержащую описание данного генератора; при этом в верхнем левом поле окна будет выведено название генератора; выделите его курсором мыши и скопируйте в буфер обмена (с помощью сочетания клавиш *<Ctrl>+<C>*);
- в блок-диаграмме выбрать блок, в котором будет использоваться генератор, и открыть окно его настроек;
- вставить из буфера обмена имя генератора (сочетание клавиш *<Ctrl>+<V>*);
- ввести требуемые значения параметров «запуска» генератора.

Конечно, при наличии достаточного опыта в разработке *S*-моделей, содержащих генераторы СВ, справочную систему MATLAB можно не использовать и вводить имя требуемого генератора вручную. Однако в этом случае существует опасность ошибиться хотя бы в одном символе, что приведет к ошибке в модели и к необходимости ее коррекции и повторного запуска. А если в создаваемой модели генератор используется в нескольких блоках?

В качестве примера использования генератора СЧ рассмотрим *S*-модель, содержащую случайную величину, распределенную по нормальному закону. Согласно теории вероятности, большинство случайных явлений и процессов, зависящих от многих одновременно действующих факторов, подчиняются именно этому закону.

Пусть имеется вычислительная система, содержащая 2 дисковых накопителя различной емкости: 600 Мб (назовем его *HD1*) и 800 Мб (*HD2*). Данные поступают на каждый из накопителей от своего источника. Объем очередной «порции» информации является случайной величиной, распределенной по нормальному закону.

Для первого источника закон распределения СВ имеет параметры m_1=10, v_1=3; для второго источника — m_2=20, v_2=7. Требуется сравнить эффективность использования накопителей. При этом показателем эффективности будет служить *коэффициент использования дискового пространства — Ки*. Эта величина может быть рассчитана как отношение объема памяти, использованного на интервале моделирования, к полной емкости накопителя.

Блок-диаграмма *S*-модели, позволяющей решить поставленную задачу, показана на рис. 4.34.

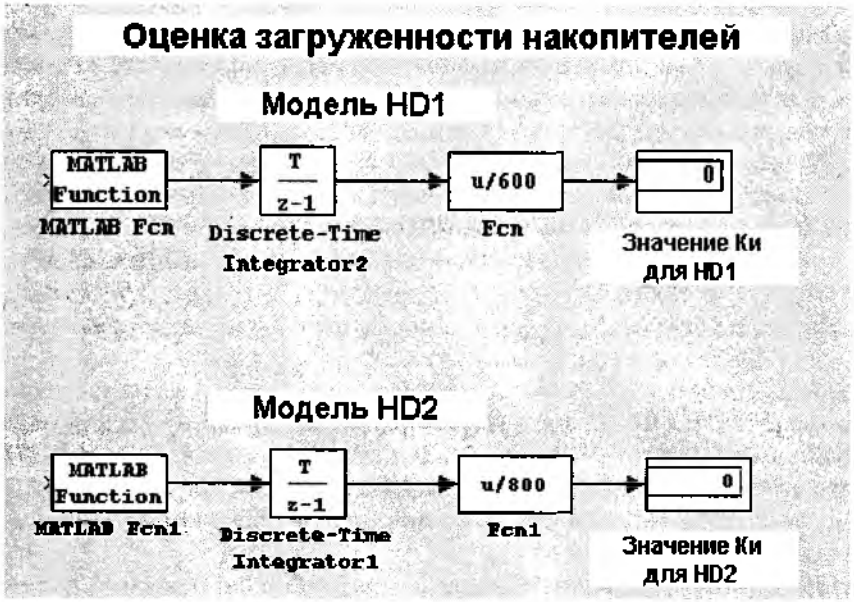


Рис. 4.34. Блок-диаграмма модели работы накопителей

Из рисунка видно, что блок-диаграмма состоит из двух достаточно самостоятельных частей. Каждая из них моделирует работу одного из накопителей. Поскольку обе части модели по структуре полностью идентичны, поясним назначение входящих в них блоков применительно к первой, моделирующей работу накопителя *HD1*.

Блок **MATLAB Fcn** играет роль источника данных (в терминах части I книги — это модель рабочей нагрузки накопителя); в качестве параметра настройки блока указано имя генератора нормального распределения с соответствующими аргументами: *normrnd*(10,3).

Блок **Discrete-Time Integrator** выполняет суммирование объема данных, поступающих от источника в накопитель; все параметры имеют значения, установленные по умолчанию.

Блок **Fcn** обеспечивает расчет значения *Ки* (вычисляемое выражение выводится на иконке блока) для текущего значения модельного времени;

Блок **Display** выводит на экран вычисленное значение Ки; по истечении интервала моделирования оно представляет собой итоговую оценку данного показателя для накопителя *HD1*.

Перед запуском модели на исполнение целесообразно установить способ изменения модельного времени *Fixed-step* (в поле *Type* на вкладке *Solver*), а также запретить выдачу предупреждений о неподключенных портах блоков (на вкладке *Diagnostics*).

Несколько забегая вперед, отметим, что SIMULINK имитирует одновременную работу обеих частей модели, в полном соответствии с технологией реализации квазипараллельных процессов, рассмотренной в I части книги (более подробно управление потоками событий в *S*-моделях рассматривается в следующем подразделе). Благодаря этому по окончании сеанса моделирования в окнах **Display** будут выведены значения *Ки* для обоих накопителей.

4.3. УПРАВЛЕНИЕ ВРЕМЕНЕМ

Корректное управление модельным временем, то есть «временем жизни» моделируемой системы — это одна из наиболее сложных, и вместе с тем наиболее важная задача, которая должна быть решена при подготовке модельного эксперимента.

Модельное время является «спинным мозгом», согласующим работу всех компонент *S*-модели, а при необходимости — и работу нескольких достаточно самостоятельных моделей.

При подготовке каждого модельного эксперимента должны быть получены ответы на три вопроса:

- какой способ изменения (приращения) времени целесообразно использовать (с постоянным или переменным шагом);
- с какой дискретностью следует изменять модельное время;
- какое событие будет служить признаком окончания эксперимента.

После того, как вы ответите себе на каждый из этих вопросов, следует подумать о том, как воплотить намеченное в жизнь, используя средства SIMULINK. Именно об этом идет речь далее.

4.3.1. ВЫБОР ШАГА МОДЕЛИРОВАНИЯ

В разделе 2.3. была дана сравнительная характеристика двух основных способов изменения модельного времени — с постоянным шагом и по особым состояниям. С точки зрения разработчиков SIMULINK ведущую роль при выборе одного из двух методов играет тип моделируемой системы: для непрерывных систем (с непрерывным временем смены состояний) по умолчанию используется переменный шаг приращения времени, а для дискретных систем следует устанавливать постоянный (фик-

сированный) шаг. Такой подход не всегда оправдывает себя, поскольку при моделировании непрерывных систем бывает удобнее определять очередное состояние системы как функцию времени, изменяющегося с заданной дискретностью. И наоборот, при моделировании дискретных систем величина очередного приращения времени зачастую определяется прогнозируемым моментом изменения состояния системы; причем смена состояний происходит, как правило, нерегулярно. Поэтому полезно знать, каким образом при разработке моделей дискретных систем можно заставить модельное время изменяться по особым состояниям.

С целью пояснения сказанного вернемся к примеру с накопителями. В разрабатываемой модели оценки загрузки накопителей фактор времени непосредственно не учитывался: была задана только длительность интервала моделирования, равная 10 единицам модельного времени. По умолчанию предполагалось, что данные поступают на каждый из накопителей через интервалы времени, определяемые дискретностью изменения модельного времени ($\Delta t = 0{,}2$). Вместе с тем, распределение входных воздействий во времени — важнейшая характеристика рабочей нагрузки системы.

Какие же средства SIMULINK позволяют управлять величиной шага моделирования при изменении модельного времени по особым состояниям?

В первую очередь это «связка» из двух уже знакомых вам блоков **MATLAB Fcn** и **Discrete-Time Integrator**. Блок **MATLAB Fcn** обеспечивает формирование отрезков времени, длина которых распределена по заданному закону, а блок **Discrete Time Integrator** выполняет суммирование этих отрезков, то есть «продвижение» модельного времени.

Таким образом, с помощью этих двух блоков можно реализовать механизм изменения модельного времени по особым состояниям для таких *S*-моделей, в которых условием наступления очередного события является истечение некоторого промежутка времени.

В рассматриваемом примере таким событием является поступление на вход системы очередного сообщения, интервал времени между соседними сообщениями является случайной величиной, распределенной по заданному закону. Пока для простоты будем считать, что запись информации на накопители происходит мгновенно, и величина приращения модельного времени определяется только промежутком времени до появления очередной порции информации.

С учетом внесенных дополнений модель работы накопителя будет выглядеть так, как показано на рис. 4.35 (для второго накопителя блок-диаграмма строится аналогичным образом).

Обратите внимание на следующее весьма важное обстоятельство: при такой реализации механизма изменения модельного времени собственный таймер SIMULINK играет роль «тактового генератора» и не определяет непосредственно длительность интервала моделирования. При таком подходе для визуального контроля за изменением модельного времени по-прежнему пригоден блок **Scope**. В этом случае на его вход следует подавать сигнал с блока **Discrete-Time Integrator**, значения которого будут отображаться в окне **Scope** как функция от номера шага

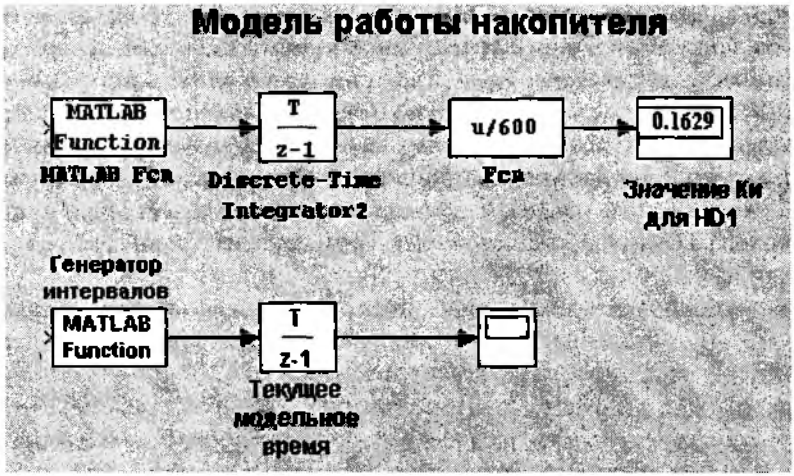


Рис. 4.35. Изменение модельного времени по особым состояниям

моделирования. Соответственно, длительность интервала моделирования будет равна последнему значению сигнала на выходе блока **Discrete-Time Integrator**.

Пусть, например, параметр *Stop time* на вкладке *Solver* по-прежнему будет равен 10, а интервалы времени между поступлениями порций информации распределены по нормальному закону с параметрами m=10; v=4. Тогда длительность интервала моделирования работы накопителей составит в среднем 90 с (рис. 4.36).

Рис. 4.36. Отображение модельного времени в блоке **Scope**

Для моделирования непрерывных систем, как вы помните, более удобным является изменение модельного времени с постоянным шагом. Чтобы реализовать этот механизм в создаваемой модели, вполне достаточно стандартных средств SIMULINK, имеющихся в разделе *Simulations Parameterters*. Хотя эти средства уже упоминались ранее, полезно рассмотреть их применение на конкретном примере, в качестве которого используем ситуацию, приведенную в первой части книги (раздел 2.4). Речь шла о том, что некий самолет движется прямолинейно и с постоянной скоростью, а следящий за ним диспетчер должен ввести информацию о самолете до того, как он пересечет заданный рубеж.

Блок-диаграмма модели, описывающей данную ситуацию, изображена на рис. 4.37.

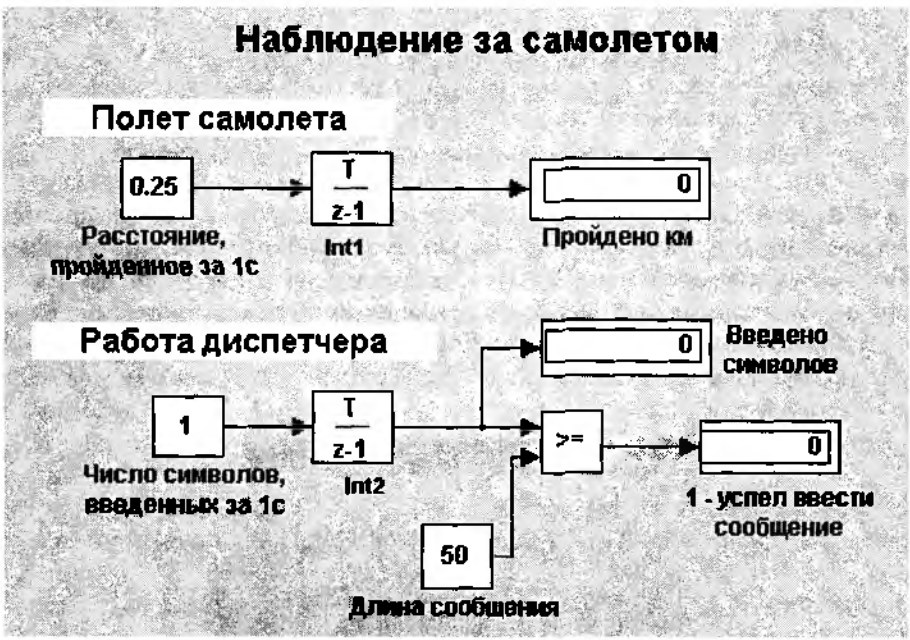


Рис. 4.37. Пример моделирования с постоянным шагом

Назначение блоков, входящих в модель, указано в самой диаграмме. Остается только добавить, что блок *Int1* вычисляет расстояние, пройденное самолетом, а блок *Int2* — количество введенных диспетчером символов.

Для определенности будем считать, что фигурирующие в модели величины имеют следующие значения:

скорость самолета: 900 км/ч;

расстояние до рубежа: 12,5 км;

скорость ввода информации: 1 символ в секунду;

длина текста: 50 символов.

Прежде чем установить параметры моделирования, мы должны:

- привести скорость полета самолета и работы оператора к единому масштабу времени;
- выбрать величину шага приращения модельного времени;
- определить приемлемую длительность интервала моделирования.

Первый пункт выполнить проще всего: выражаем скорость самолета в км/с и получаем 0,25 км/с.

Выполнение второго пункта требует сопоставления скоростей двух процессов; сравнив скорости, выберем $\Delta t = 2$ с.

Для выполнения третьего пункта также вполне достаточно выполнить однодва арифметических действия.

Конечно, простота рассматриваемого примера вызывает сильное желание посчитать все вручную , но нас в данном случае интересует не столько результат, сколько процесс создания модели.

Итак, устанавливаем следующие параметры моделирования:

- *Stop time*: 50;
- *Type*: *Fixed step*;
- *Fixed step size*: 2.

Последний параметр требует небольшого пояснения: для всех блоков, имеющих параметр *Sample time*, он также должен быть равен 2.

Выполнив сеанс моделирования, получаем ответ на поставленный ранее вопрос: диспетчер успеет (хотя и с трудом) ввести требуемую информацию.

Соответствие результата моделирования ручным расчетам говорит о том, что созданная модель работает корректно (всегда бы так!).

Казалось бы, при неизменных исходных данных такой результат должен получаться всегда. Но не следует забывать о той роли, которую играет выбор шага моделирования (т. е. величины Δt). Установите значение *Fixed step size*: 3 и повторите моделирование. Теперь диспетчер «не успевает» ввести сообщение, сколько бы раз вы не повторяли эксперимент.

4.3.2. УПРАВЛЕНИЕ ОКОНЧАНИЕМ МОДЕЛИРОВАНИЯ

Еще при знакомстве с *S*-моделями, включенными разработчиками MATLAB в состав демонстрационных файлов, мы отметили наличие нескольких способов управления окончанием моделирования.

Наиболее простой из них состоит в указании длительности интервала моделирования. При таком подходе сеанс моделирования завершается при достижении модельным временем значения, заданного в поле *Stop time* при настройке параметров моделирования (вкладка *Solver*).

Текущее состояние модели (или моделируемой системы) в данном случае не учитывается.

С целью расширения возможностей исследователя по управлению модельным экспериментом в состав библиотеки SIMULINK включен блок **Stop Simulation** (раздел *Sinks*). С его помощью можно увязывать окончание сеанса моделирования с выполнением тех или иных условий. Такими условиями, в частности, могут быть:

- переход моделируемой системы в определенное состояние;
- завершение выполнения какой-либо операции (системой в целом или одной из ее подсистем);
- достижение показателем эффективности заданного значения.

Условия окончания сеанса моделирования должны быть заданы в *S*-модели таким образом, чтобы при их наступлении на вход блока **Stop Simulation** поступал ненулевой сигнал (если сигнал векторный, то хотя бы один его элемент должен быть отличен от нуля).

На рис. 4.38 показан вариант моделирования работы накопителей *HD1* и *HD2* с использованием блока **Stop Simulation**. *Условием окончания* сеанса моделирования является исчерпание емкости хотя бы одного из накопителей (если накопитель заполнен, то Ки ≥ 1). Поскольку в рассматриваемом примере накопители заполняются достаточно медленно, целесообразно установить интервал моделирования существенно больше заданного по умолчанию (скажем, *Stop time* = 100).

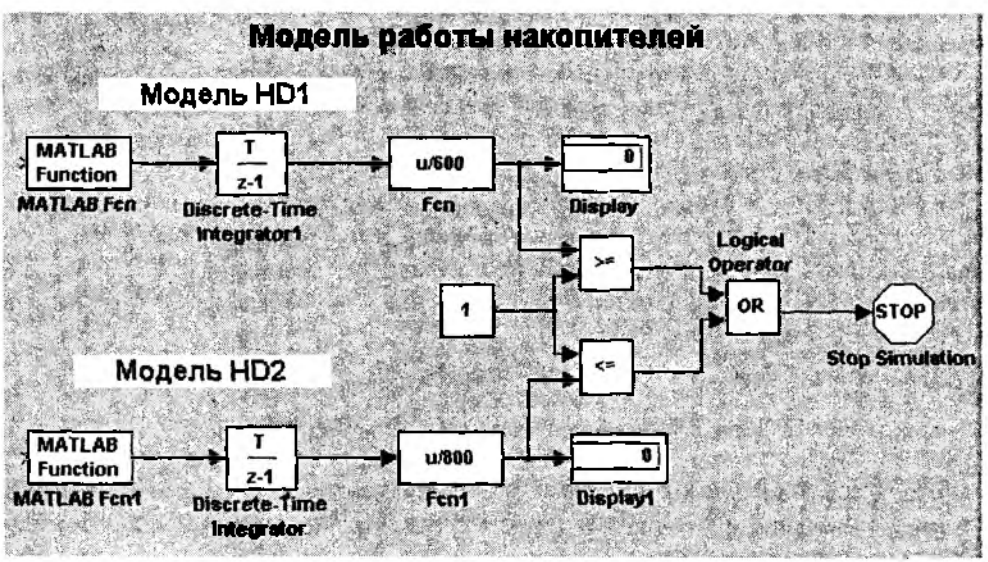


Рис. 4.38. Пример использования блока **Stop Simulation**

При моделировании сложных систем бывает необходимо задавать несколько различных условий окончания эксперимента. В таких случаях в состав *S*-модели может включаться несколько блоков **Stop Simulation**, каждый из которых связан со своим признаком завершения сеанса.

Пусть, например, при моделировании работы накопителя сеанс должен завершаться не только при исчерпании его емкости, но и при генерации сообщения, размер которого превышает заданный порог. Соответствующая блок-диаграмма показана на рис. 4.39.

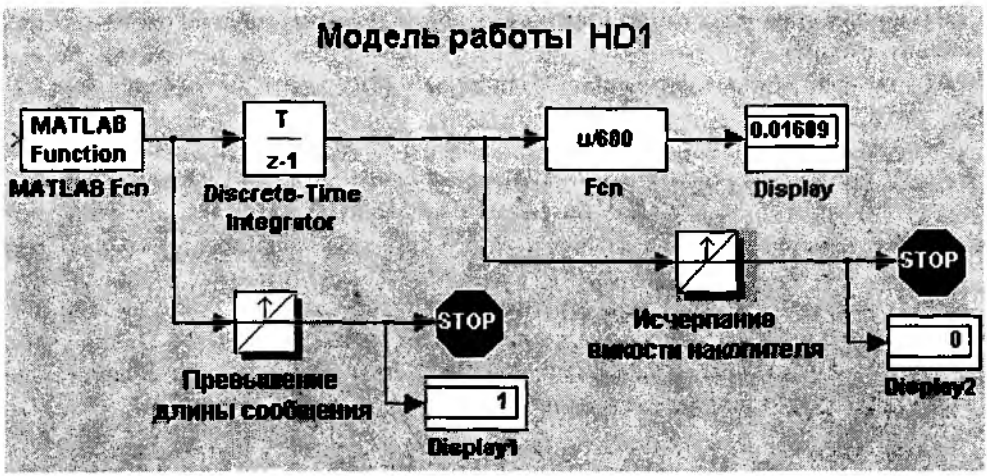


Рис. 4.39. Применение блока **Stop Simulation** при наличии нескольких условий

Для идентификации указанных событий используются блоки **Hit Crossing** (их имена поясняют назначение каждого из них); для визуализации причины окончания сеанса блока **Stop Simulation** связаны с блоками **Display**.

Как вы догадались, значение «1», выводимое в блоке **Display**, указывает на соответствующую причину завершения моделирования.

4.3.3. УПРАВЛЕНИЕ ПОТОКАМИ СОБЫТИЙ

Как было сказано в I части книги, каждый из отображаемых в модели типов событий характеризуются своими условиями (или законом) возникновения. Причем в одной модели могут отображаться события нескольких различных типов. С точки зрения управления модельным временем наибольшую сложность представляют события, связанные с запуском и завершением работ (или процессов), длительность которых являются случайными величинами.

Для конкретного управления модельным временем в таких моделях необходимо решить две задачи:

- обеспечить приращение модельного времени на величину длительности очередного завершенного процесса (работы);
- определить условия окончания сеанса моделирования.

Как было уже сказано, для формирования текущего значения модельного времени удобнее всего использовать блок **Discrete-Time Integrator**. Однако он имеет только один информационный вход. Поэтому при наличии в модели нескольких типов событий, влияющих на изменение модельного времени, соответствующие им интервалы времени должны предварительно суммироваться. Наиболее подходящий для этого блок — это блок **Sum**, число входов которого можно изменить.

На рис. 4.40 показана очередная модификация модели работы накопителя. На этот раз в нее внесены следующие изменения.

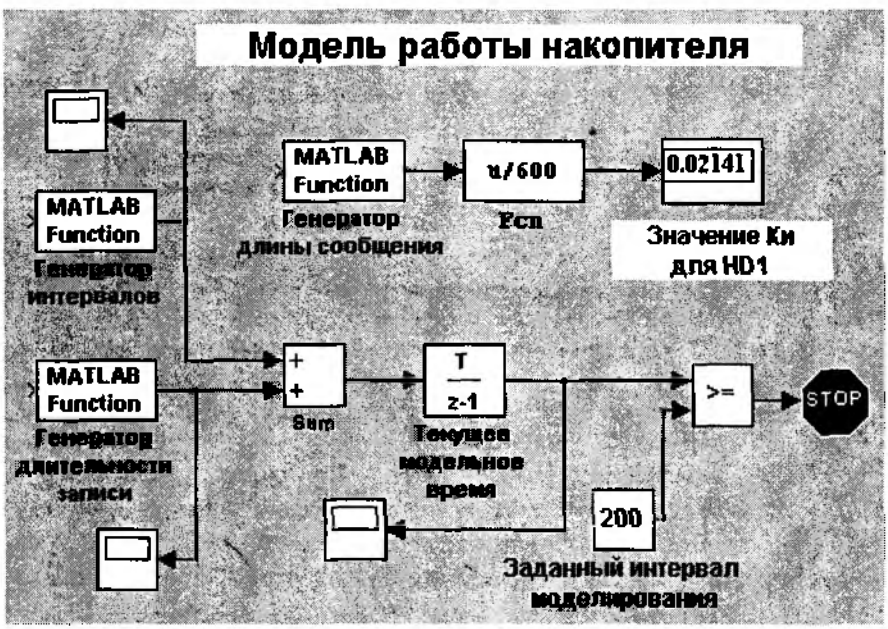


Рис. 4.40. Управление модельным временем при наличии двух типов событий

Предполагается, что время, необходимое для записи поступающего сообщения, является СВ, распределенной по нормальному закону с параметрами $m = 7$, $v = 3$. Сообщения поступают через случайные интервалы времени, величина которых распределяется по нормальному закону. Соответственно, продвижение модельного времени определяется двумя процессами, имеющими различные временные параметры:

- процессом поступления сообщений от источника;
- процессом записи сообщений на *HD*.

Замечание. На самом деле в данном случае более корректным был бы термин «работа», а не «процесс», поскольку и поступление сообщений, и их запись пред-

ставлены в модели как неделимые элементарные операции, образующие единый процесс — «запись сообщения на HD1».

Для отображения в модели временных параметров процесса записи сообщений в нее введены два блока **MATLAB Fcn** (на диаграмме они обозначены как «Генератор длительности записи» и «Генератор интервалов»). Блоки **Sum** и **Discrete-Time Integrator** обеспечивают продвижение модельного времени на соответствующую величину.

Как и в рассмотренном ранее примере, собственный таймер SIMULINK в данном случае играет роль «тактового генератора». Для наблюдения за изменением модельного времени и влияющими на него процессами в блок-диаграмму включены три блока **Scope**. На рис. 4.41 показаны выводимые в них данные.

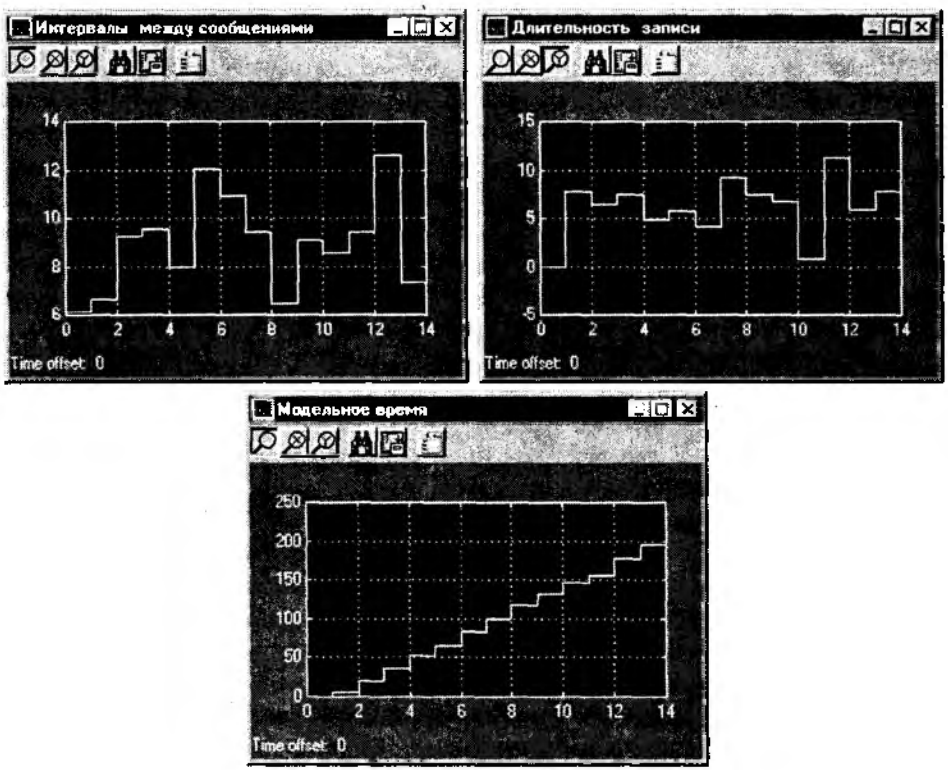


Рис. 4.41. Временные диаграммы работы модели

Еще раз подчеркнем, что в окне «Модельное время» текущее значение модельного времени отмечается на оси ординат, а на оси абсцисс указаны «номера тактов» работы модели (их можно трактовать как номера сообщений, поступающих от источника).

Чтобы использованный в данной модели механизм управления модельным временем работал корректно, необходимо также учитывать значения параметров моделирования, установленные на вкладке *Solver* окна *Simulation parameters*. В данном случае они должны быть такими:

Type: *Fixed-step*; *discrete*;

Fixed Step-size: 1.

Длительность интервала моделирования (параметр *Stop time*) может быть любой. В приведенной блок-диаграмме она подобрана таким образом, чтобы сеанс моделирования завершался при достижении модельным временем значения 200. Это условие реализуется, как вы видите, с помощью блока **Stop Simulation**.

Очевидно, в реальной системе процессы поступления сообщений и их записи на накопитель являются синхронными, так как очередное сообщение может начать записываться только после его поступления от источника. В модели такая строгая синхронизация не обеспечивается: мы не можем утверждать, что сначала SIMULINK добавляет к модельному времени отрезок, соответствующий интервалу до поступления очередного сообщения, и только после этого изменяет модельное время на длительность записи. Нам известно только то, что обе эти величины добавляются к текущему значению модельного времени в каждом такте моделирования (т. е. для каждого очередного сообщения). Пока этого вполне достаточно. Хотя в составе SIMULINK имеются средства синхронизации событий, но они относятся к механизму использования подсистем и будут рассмотрены в следующем разделе.

Сейчас же мы вернемся на некоторое время к первоначальной формулировке задачи об оценке работы двух накопителей. И обратим внимание на то, что оба они работают одновременно и совершенно независимо друг от друга, т. е. в рассматриваемой системе существуют два асинхронных параллельных процесса. Каким же образом можно «привязать» к единой оси модельного времени случайные события, происходящие в рамках каждого из них?

Для этого может быть использована та же пара блоков — **Sum** и **Discrete-Time Integrator**. Однако только этих двух блоков уже недостаточно. Дело в том, что если просто суммировать затраты времени на работу *HD1* с затратами времени на работу *HD2*, то это приведет к двойному учету времени (ведь накопители работают одновременно!).

В подобных ситуациях целесообразно использовать следующий способ продвижения модельного времени.

Будем исходить из того, что процесс обслуживания каждого сообщения состоит из двух составляющих:

- ожидания его поступления (в течение времени *t*ож);
- записи на носитель (за время *t*з).

Тогда *HD1* обслужит очередное сообщение за время

$$T_1 = t\text{ож}_1 + t\text{з}_1,$$

а накопитель *HD2* — за время

$$T_2 = t\text{ож}_2 + t\text{з}_2.$$

Еще одно важное допущение состоит в том, что начало обслуживания очередного сообщения каждым накопителем совпадает с очередным тактом работы модели, задаваемым таймером SIMULINK.

Тогда продвижение модельного времени в каждом такте будет определяться большей из величин T_1 и T_2:

$$t\text{м} = t\text{м} + \max\{T_1, T_2\}.$$

Соответствующая блок-диаграмма показана на рис. 4.42 (поскольку способ вычисления коэффициента использования накопителей *Ки* при этом не изменяются, соответствующие блоки в данном варианте диаграммы не показаны).

Для реализации описанного выше механизма управления временем в модели используются следующие блоки:

Sum1 – вычисляет значение T_1 для очередного такта моделирования (т. е. для очередного сообщения);

Sum2 – вычисляет значение T_2;

Sum3 – обеспечивает сравнение величин T_1 и T_2;

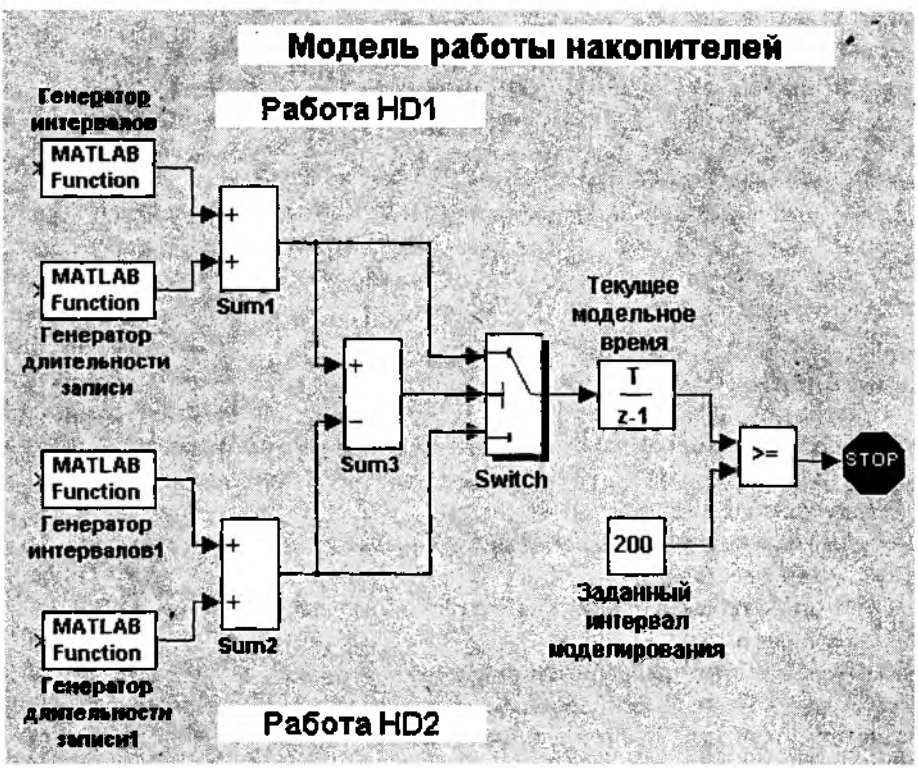


Рис. 4.42. Управление модельным временем при наличии двух параллельных процессов

Switch — пересылает на вход блока **Discrete-Time Integrator** большую из величин T_1 и T_2.

Discrete-Time Integrator — вычисляет новое значение модельного времени.

Как и в предыдущем варианте модели, сеанс завершается по истечении заданного интервала моделирования.

4.4. ИСПОЛЬЗОВАНИЕ ПОДСИСТЕМ

Как любил повторять незабвенный Козьма Прутков, нельзя объять необъятное. Но это утверждение справедливо только в том случае, если вы пытаетесь совершить указанный поступок «голыми руками». Чтобы необъятное перестало быть таковым, разработчики SIMULINK создали механизм подсистем. Он основан в первую очередь на использовании блока **Subsystem**, входящего в раздел библиотеки *Connections*. С его помощью любая сложная система может быть представлена как совокупность взаимодействующих компонентов, внутренняя структура которых при необходимости может быть скрыта.

Основными достоинствами механизма подсистем являются:

1. Повышение наглядности блок-диаграмм моделей сложных систем.
2. Возможность использования механизма синхронизации параллельно функционирующих подсистем.
3. Повышение технологичности разработки и модификации *S*-моделей.
4. Возможность включения в блок-диаграмму собственной «встроенной» справочной системы, в том числе содержащей демонстрационные средства.

О том, как реализовать и использовать каждое из перечисленных достоинств, будет рассказано в последующих разделах.

4.4.1. ВХОДЫ, ВЫХОДЫ И ПЕРЕХОДЫ

Знакомство с технологией использования подсистем начнем с обсуждения способов их создания.

Таких способов два:

- с помощью команды *Create Subsystem* (*Создать подсистему*), входящей в раздел *Edit* меню пользователя;
- путем копирования блока **Subsystem** из раздела библиотеки *Connections* в окно блок-диаграммы.

Замечание: разумеется, для создания копии или нового варианта существующей подсистемы могут быть использованы стандартные средства копирования, применимые к любому блоку *S*-модели. Но пока речь идет о создании новой, «пустой» подсистемы.

Команда *Create Subsystem* применяется к группе блоков или ко всей блок-диаграмме, находящейся в активном окне *S*-модели. Чтобы команда стала доступна, соответ-

ствующие блоки должны быть выделены (если в подсистему «сворачивается» вся блок-диаграмма, то удобнее воспользоваться командой *Select all — выделить все*). Недостатком команды *Create Subsystem*, как уже отмечалось, является то, что ее действие нельзя отменить. Это требует определенной осторожности при ее использовании.

Применение данной команды продемонстрируем для блок-диаграммы асинхронно работающих накопителей, изображенной на рис. 4.42.

Выделите на диаграмме 3 блока, относящиеся к работе накопителя *HD1*: *Генератор интервалов* (**MATLAB Fcn**), *Генератор длительности записи* (**MATLAB Fcn1**), и **Sum1**. После этого откройте раздел *Edit* меню пользователя и выберите команду *Create Subsystem*. Блок-диаграмма примет вид, показанный на рис. 4.43.

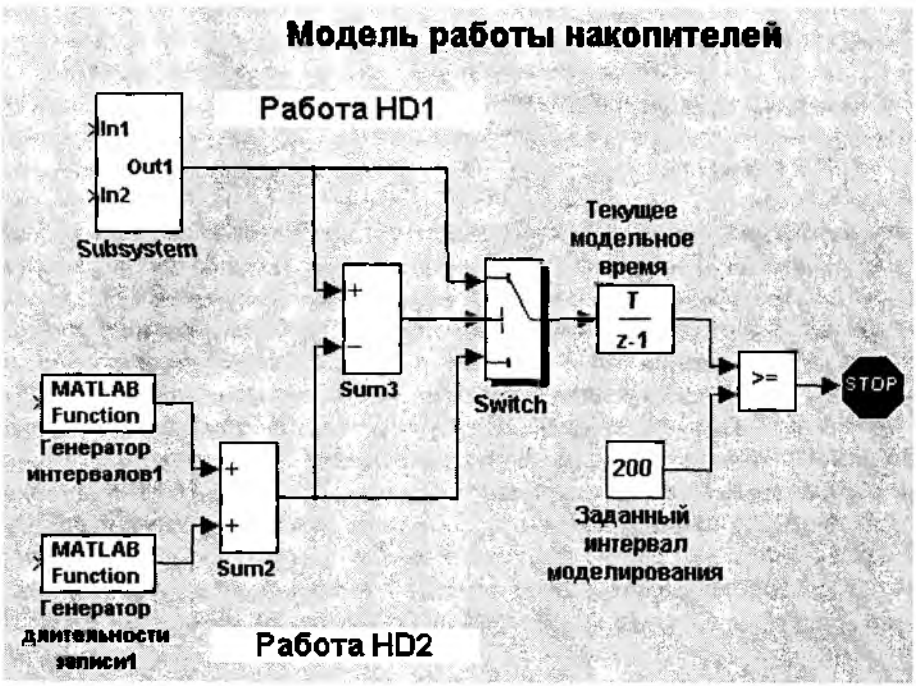


Рис. 4.43. Создание подсистемы с помощью команды *Create Subsystem*

Из рисунка видно, что иконка блока **Subsystem** существенно отличается от иконки его библиотечного варианта.

На иконке, появившейся в блок-диаграмме, обозначены два входных (*In*1 и *In*2) порта подсистемы и один выходной (*Out*1). Именно на них будет сосредоточено основное наше внимание в ближайшее время.

Давайте заглянем «вовнутрь» созданной подсистемы. Для этого нужно установить курсор мыши на ее иконку и дважды щелкнуть ЛКМ.

Блок-диаграмма, содержащаяся в открывшемся окне (рис. 4.44), вероятно, несколько отличается от того, что вы ожидали увидеть. Новое заключается в том, что наряду с тремя основными блоками, подсистема теперь содержит три дополнительных — **In1**, **In2** и **Out1**.

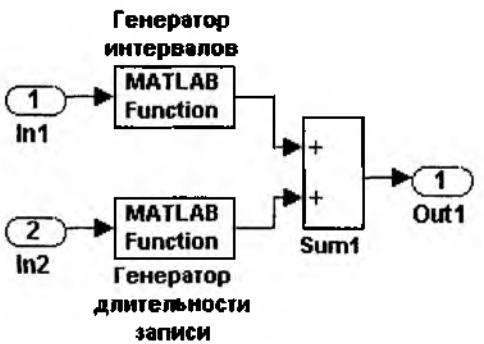


Рис. 4.44. Блок-диаграмма созданной подсистемы

Именно их присутствие в составе подсистемы и повлияло на изменение ее иконки.

Блоки **In** и **Out** входят в раздел библиотеки *Connections* и предназначены в первую очередь для обеспечения связи подсистемы с «внешним миром». Как следует из названия, блоки типа **In** выполняют прием входных сигналов, поступающих в подсистему, а блоки типа **Out** — выдачу результатов ее работы на другие блоки (в том числе и подсистемы) модели, либо их сохранение в рабочей области MATLAB.

Количество блоков **In** и **Out**, включаемых в состав подсистемы, не ограничено и определяется только логикой работы создаваемой модели. Эти блоки, как и блоки других типов, могут иметь в диаграмме произвольные имена, назначаемые разработчиком в соответствии с их содержательным смыслом. Эти имена выводятся на иконке подсистемы вместо стандартных обозначений **In** и **Out**, что, естественно, значительно повышает наглядность блок-диаграммы. Чтобы убедиться в этом, замените в окне подсистемы обозначение выходного порта **Out** именем *T*1 и закройте окно. Внесенные изменения сразу же отображаются на иконке подсистемы.

При создании подсистемы на основе копирования библиотечного блока **Subsystem** целесообразно придерживаться следующей последовательности действий.

Шаг 1. Отбуксируйте копию блока **Subsystem** в окно редактируемой (основной) блок-диаграммы.

Шаг 2. Откройте пустое окно диаграммы подсистемы (двойной щелчок ЛКМ на иконке блока) и разместить его на экране так, чтобы оно не перекрывалось окном основной блок-диаграммы.

Шаг 3. Способ реализации данного шага зависит от того, какие блоки должны войти в подсистему: если это часть основной блок-диаграммы, то ее можно пере-

нести в окно подсистемы с помощью команд редактирования *Cut* и *Paste*; если же создается новая блок-диаграмма, то на данном шаге выполняются те же действия, что и при создании любой другой блок-диаграммы.

Для определенности будем считать, что мы создаем подсистему, аналогичную полученной с помощью команды *Create Subsystem*, но для накопителя *HD2*.

С этой целью в модели работы накопителей необходимо выделить и «вырезать» (с помощью команды *Cut* или используя клавиши <Ctrl>+<X>) три блока – **Генератор интервалов 1**, **Генератор длительности записи 1** и **Sum2**, а также соединяющие их линии связи; после этого следует сделать активным окно подсистемы и вставить в него перемещаемый фрагмент (с помощью команды *Paste* или используя клавиши <Ctrl>+<V>).

Шаг 4. Подсоедините к входным портам, блоков **MATLAB Fcn** блоки **In**, а к выходному порту блока **Sum2** – блок **Out** (их можно отбуксировать из раздела *Connections* или скопировать из ранее созданной подсистемы).

Шаг 5. Закройте окно новой подсистемы (теперь она стала похожа на свою предшественницу), переместите блок **Subsystem1** в нужное место и подсоедините к входному порту блока **Sum3**.

Результат выполнения перечисленных действий показан на рис. 4.45.

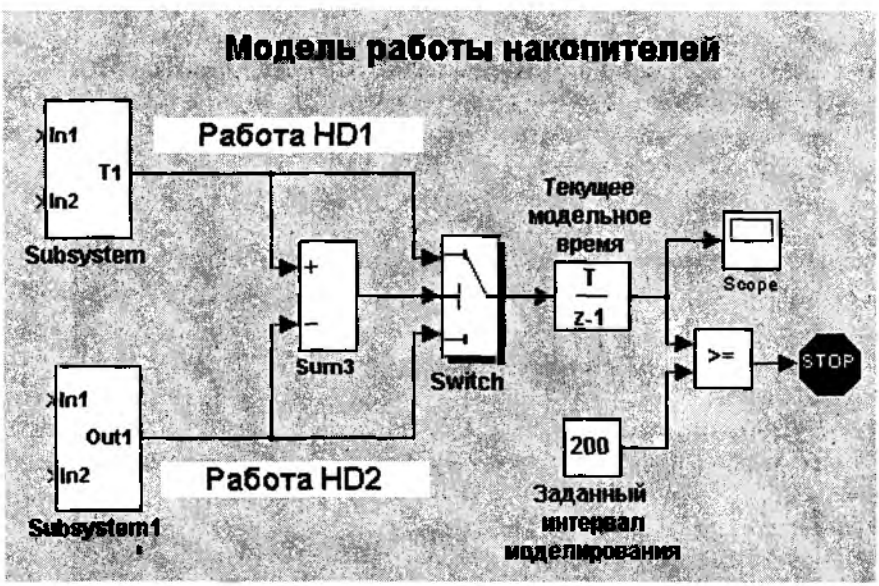


Рис. 4.45. Вид блок-диаграммы после создания второй подсистемы

Несколько слов об окне блок-диаграммы подсистемы. Внешне оно ничем не отличается от окна основной модели и его меню содержит те же разделы.

Однако область действия команд меню распространяется на всю *S*-модель, в которую входит данная подсистема. В частности, команда *Save* из раздела *File* обеспечивает запись на диск всей блок-диаграммы модели, включая подсистему, а команда *Start* запускает на исполнение основную модель. Исключение составляют только некоторые команды из раздела *Format* (например, команда изменения цвета заливки окна — *Screen color*).

После этих не очень существенных замечаний вернемся к блокам **In** и **Out**. Вернемся для того, чтобы пояснить весьма большое значение этих не очень примечательных блоков.

Необходимо отметить, что блоки **In** и **Out** могут использоваться и непосредственно в блок-диаграмме самого верхнего уровня. В этом случае они обеспечивают взаимосвязь *S*-модели с рабочей областью MATLAB.

Начнем с блока **In**. Он имеет 3 параметра настройки (рис. 4.46):

Port number определяет порядковый номер входного порта; по умолчанию его значение устанавливается SIMULINK для каждого нового блока **In** автоматичес-

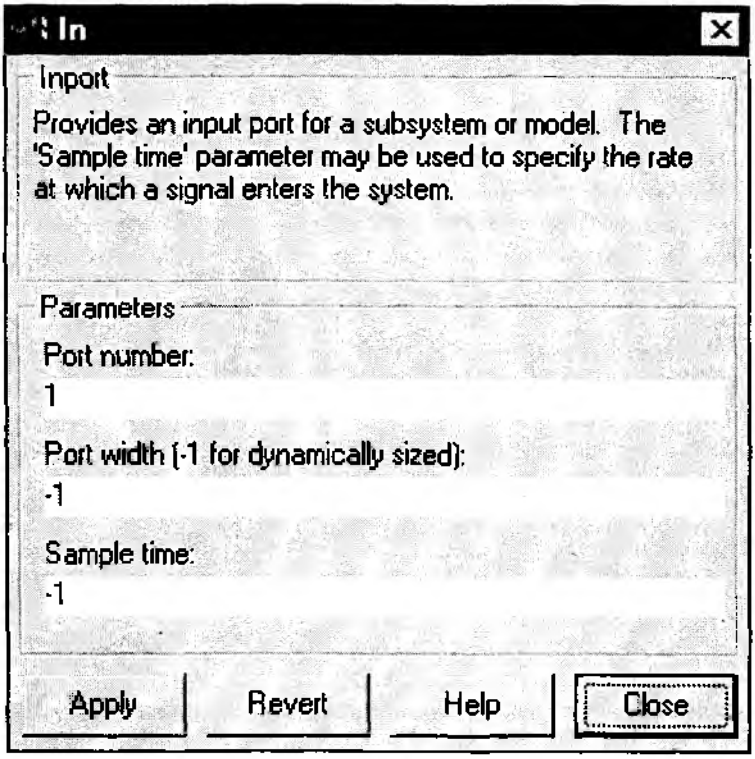


Рис. 4.46. Окно настройки параметров блока **In** (**Inport**)

ки в возрастающем порядке; при необходимости номер блока может использоваться для согласования с соответствующим блоком **Out**;

Port width (*Ширина порта*) устанавливает размерность входного сигнала, который может подаваться на данный порт; значение -1 обеспечивает автоматическую настройку порта на размерность входного сигнала;

Sample time (*Образец времени*) определяет дискретность времени, с которой блок **In** будет «реагировать» на изменение входного сигнала; при значении этого параметра, равном –1, дискретность определяется одноименным параметром предшествующего блока.

Если блок **In** включен в состав блок-диаграммы самого верхнего уровня, то с его помощью можно передавать в модель данные, находящиеся в рабочей области MATLAB. Для этого необходимо в окне установки параметров моделирования ***Parameters...*** на вкладке *Workspace I/O* установить флажок *Load Workspace*. Имена векторов времени и считываемых данных должны совпадать с теми именами, под которыми они записаны в рабочей области (по умолчанию – *tout* и *yout* соответственно).

Блок **Out** также имеет окно настройки, содержащее три параметра (рис. 4.47).

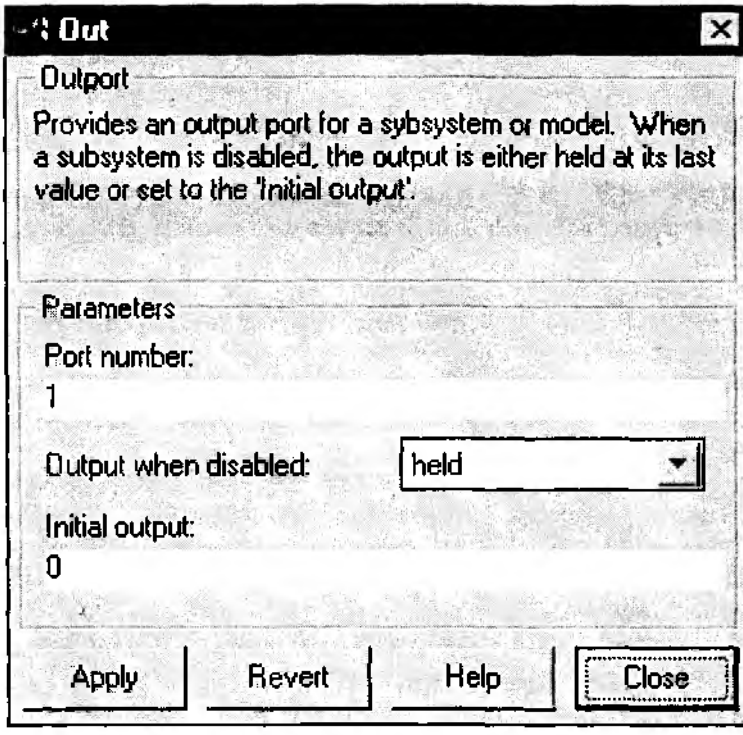


Рис. 4.47. Окно настройки параметров блока **Out (Outport)**

Port number — порядковый номер выходного порта; используется таким же образом, как и одноименный параметр блок **In**;

Output when disabled (*Значение на выходе при запрещенном состоянии*) — имеет смысл в том случае, если данный блок входит в состав управляемой подсистемы; его значение устанавливается с помощью выпадающего меню, содержащего два пункта:

held (*сохранение*) — на выход выдается сигнал, сформированный при последнем срабатывании подсистемы;

reset (*восстанавливать*) — на выход поступает сигнал, соответствующий исходному (начальному) состоянию подсистемы;

Initial output — начальное значение выходного сигнала; может быть задано в виде числового значения (скалярного или векторного), либо в виде вычисляемого выражения.

Если блок **Out** входит в блок-диаграмму самого верхнего уровня, то с его помощью можно сохранять в рабочей области MATLAB значение поступающего на него сигнала. Для этого на вкладке *Workspace I/O* окна ***Parameters*** должен быть установлен флажок *Output*. По умолчанию данные сохраняются под именем *yout*, которое может быть изменено пользователем. Если в диаграмме верхнего уровня имеется более одного блока **Out**, то переменная yout является матрицей, в которой каждый столбец соответствует блоку **Out** с таким же порядковым номером.

Используя переменные *tout* и *yout*, можно получить график изменения выходной характеристики *yout* во времени. Для этого в командном окне MATLAB достаточно ввести команду

plot(*tout*, *yout*).

Тем не менее блоки **In** и **Out** представляют собой самое слабое средство взаимосвязи подсистемы с другими компонентами модели, поскольку связывают подсистему по информации только с ее ближайшими «соседями».

Значительно более широкие возможности по обмену данными между компонентами модели представляют блоки **Goto**, **From** и **Goto Tag Visibility**, также входящие в раздел *Connections*.

Блок **Goto**, название которого в данном случае можно перевести как «передать», является основным среди них, поскольку определяет имя и ***область видимости*** передаваемых данных.

Для тех читателей, которые знакомы с программированием на одном из универсальных языков, термин «область видимости» в пояснениях не нуждается.

Для остальных же уточним, что область видимости переменной (или каких-либо данных) определяет те компоненты модели, в которых эта переменная доступна для использования.

Имя передаваемых данных и область их видимости указывается разработчиком модели с помощью окна настройки параметров блока **Goto**. Таких параметров два (рис. 4.48):

- *Tag* (*признак*) — имя передаваемых данных;
- *Tag visibility* (*признак видимости*).

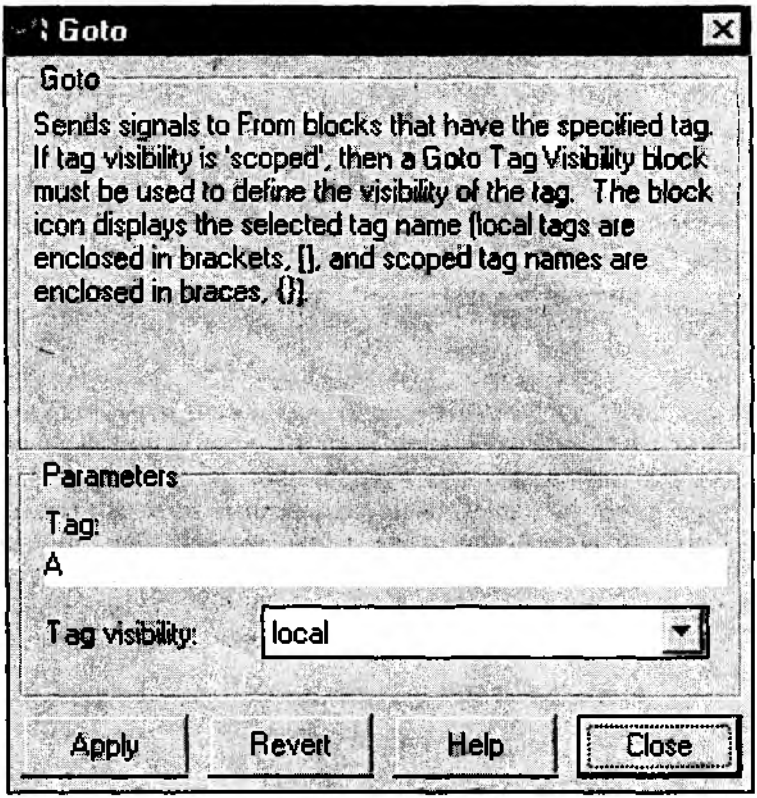


Рис. 4.48.Окно настроек блока **Goto**

Значение первого параметра вводится в строке редактирования вручную и представляет собой любую последовательность латинских букв, цифр и символов «подчеркивания», причем на первой позиции обязательно должна стоять буква. Длина имени практически не ограничена, но SIMULINK различает только первые 32 (по правилам языка *C*).

Значение второго параметра устанавливается с помощью «выпадающего» меню, которое содержит три пункта:

• *local* — переменная является локальной; это означает, что область ее видимости ограничено той подсистемой, в которой находится данный блок **Goto**;

• *scoped* — область видимости переменной распространяется дополнительно на все подсистемы более низких уровней иерархии, входящие в состав данной подсистемы;

• *global* — переменная является глобальной; это означает, что она доступна в любой компоненте (подсистеме) *S*-модели.

Установленные параметры отображаются на иконке блока: имя передаваемой переменной выводится либо в квадратных скобках (при локальной области види-

мости), либо в фигурных (для области видимости *scoped*), либо вообще без скобок (при глобальной области видимости).

Блок **From** (название можно перевести как «принять») обеспечивает прием данных от соответствующего блока **Goto**. Соответствие определяется по имени переменной, которое является единственным параметром настройки этого блока (рис. 4.49). Имя переменной, как и для блока **Goto**, выводится на иконке.

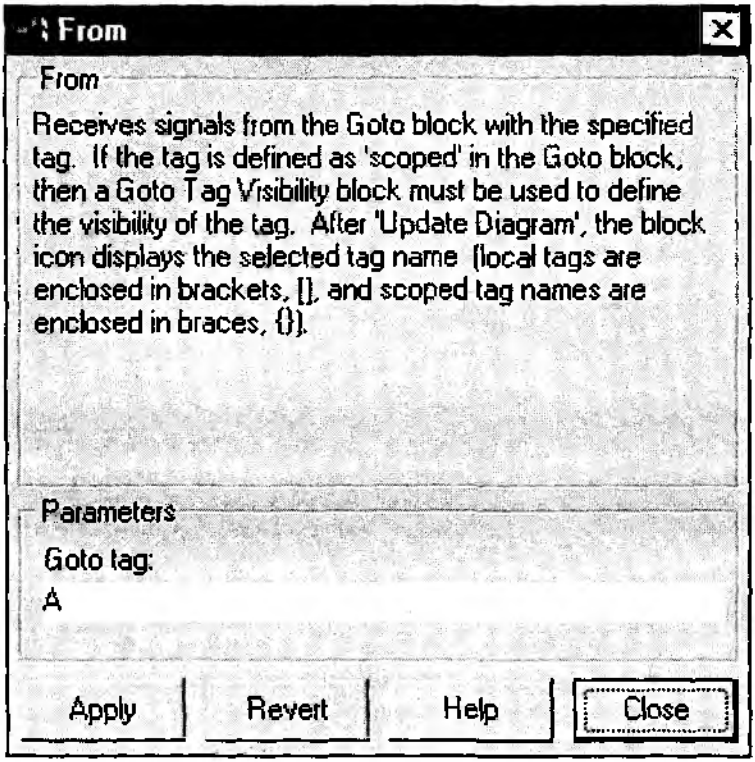


Рис. 4.49. Окно настройки параметров блока **From**

Блок **From** может быть связан только с одним блоком **Goto**. Зато последний может передавать значение переменной на произвольное число блоков **From**.

Блок **Goto Tag Visibility** необходимо включать в состав подсистемы только в том случае, если для передаваемых данных установлена область видимости *scoped*. Этот блок не имеет ни входных, ни выходных портов и не участвует в преобразовании данных. В модели он играет роль, аналогичную роли предупреждающего

дорожного знака. Его единственным параметром настройки является имя передаваемой переменной, которое выводится на иконке блока.

Один из возможных вариантов использования блоков **Goto** и **From** в модели работы накопителей показан на рис. 4.50.

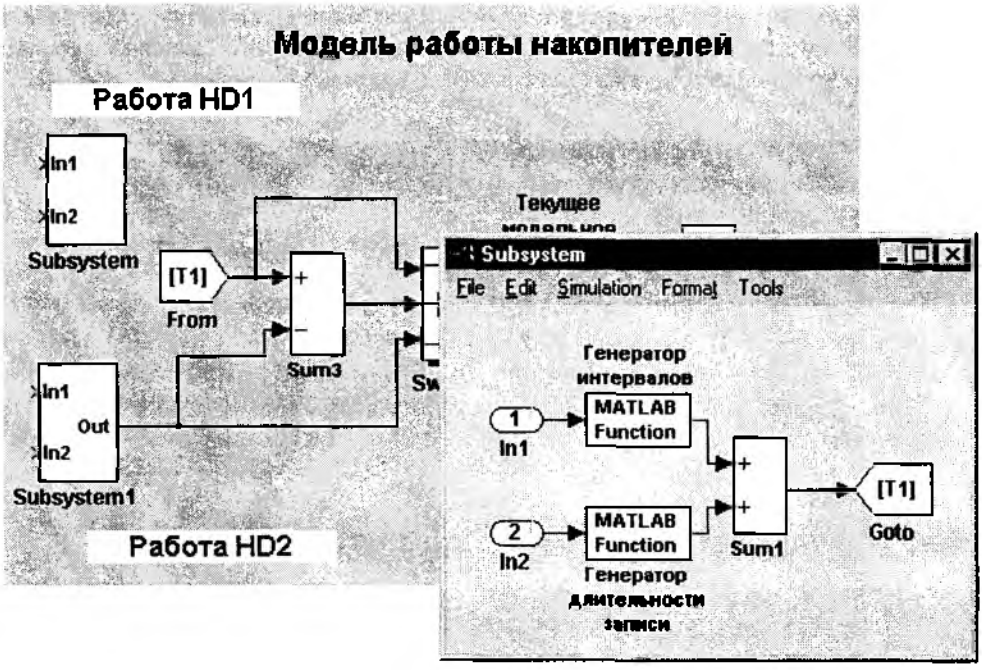


Рис. 4.50. Пример использования блоков **Goto** и **From**

В данном случае они обеспечивают передачу величины *T*1 (длительности обслуживания очередного сообщения накопителя *HD1*) из подсистемы *Subsystem* на входы блоков **Sum3** и **Switch**.

Те из читателей, кто занимался программированием, знают, что каждая переменная в программе характеризуется не только областью видимости, но и ***временем жизни***. Некоторые переменные «живут» на протяжении всего времени выполнения программы, другие «появляются на свет» только при запуске какой-либо подпрограммы и «умирают» при завершении ее работы.

С этой точки зрения блоки *S*-модели можно рассматривать как подпрограммы, а обрабатываемые в них данные (сигналы) остаются «живы» до тех пор, пока переходят от одного блока к другому. Другими словами, время жизни сигнала в *S*-модели определяется длиной той цепочки блоков, по которой он распространяется. Это утверждение справедливо и для тех данных, которые передаются с помощью блоков **Goto** и **From**.

Но в составе SIMULINK есть блоки, которые способны «продлить жизнь» любому сигналу (переменной) до окончания сеанса моделирования независимо от того, в каких еще блоках он используется. Это блоки **Data Store Memory**, **Data Store Read**, **Data Store Write**. Они используются только все вместе, поскольку отсутствие в модели любого из них делает ненужным присутствие двух других. Но наиболее важную роль играет блок **Data Store Memory**, поскольку он определяет имя, формат и область видимости сохраняемых данных. Блок имеет два параметра настройки (рис. 4.51):

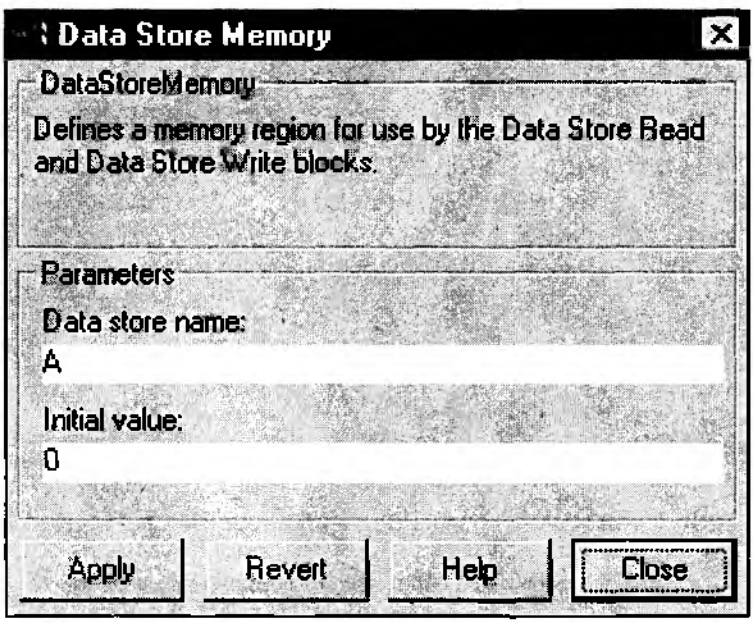


Рис. 4.51. Окно настройки параметров блока **Data Store Memory**

- ***Data Store name*** (*имя сохраняемых данных*);
- ***Initial value*** (*начальное значение*).

Правила для записи имени сохраняемых данных те же, что и для записи имени переменной в блоке **Goto**.

С помощью параметра *Initial value* задается не только начальное значение, но и формат сохраняемых данных. Например, если его значение установлено так:

[[2 4]] [3 7]],

то это означает, что сохраняемая переменная представляет собой массив размерностью 2 × 2. Этот формат должен обязательно учитываться при использовании блоков **Data Store Write** и **Data Store Read**.

Область видимости сохраняемой переменной определяется тем, где расположен блок **Data Store Memory**. Если он помещен в блок-диаграмме самого верхнего уровня, то сохраняемые данные доступны в любой компоненте *S*-модели. Если же блок **Data Store Memory** находится внутри одной из подсистем, то область видимости сохраняемых данных ограничивается этой подсистемой и входящими в нее подсистемами более низких уровней иерархии (если таковые имеются).

Окна настройки параметров блоков **Data Store Read** и **Data Store Write** идентичны и содержат по два параметра (рис. 4.52):

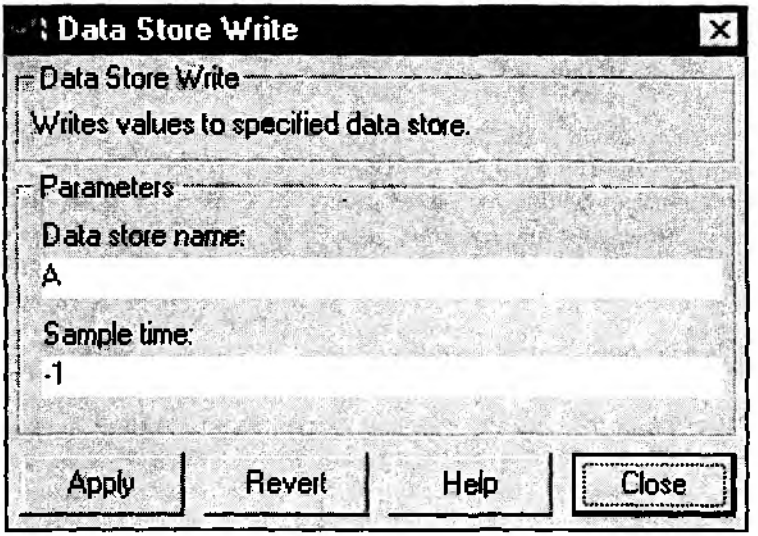


Рис. 4.52. Окно настройки параметров блока **Data Store Write**

• ***Data Store Name*** (*имя сохраняемых данных*) — его значение должно совпадать с именем, указанным в блоке **Data Store Memory**;

• ***Sample time*** — величина шага моделирования для данного блока.

SIMULINK разрешает создавать в модели любое число экземпляров блоков **Data Store Read** и **Data Store Write**, относящихся к одному и тому же блоку **Data Store Memory**. Однако при этом он не контролирует согласованность записи и считывания сохраняемых данных в различных точках блок-диаграммы. Ответственность за такое согласование возлагается на разработчика модели. Следует заметить, что недостаточно внимательное отношение к этому вопросу может привести к неожиданным и весьма неприятным последствиям.

На рис. 4.53 приведен пример некорректного использования блока **Data Store Write**. Некорректность заключается в том, что под одним и тем же именем (*A*)

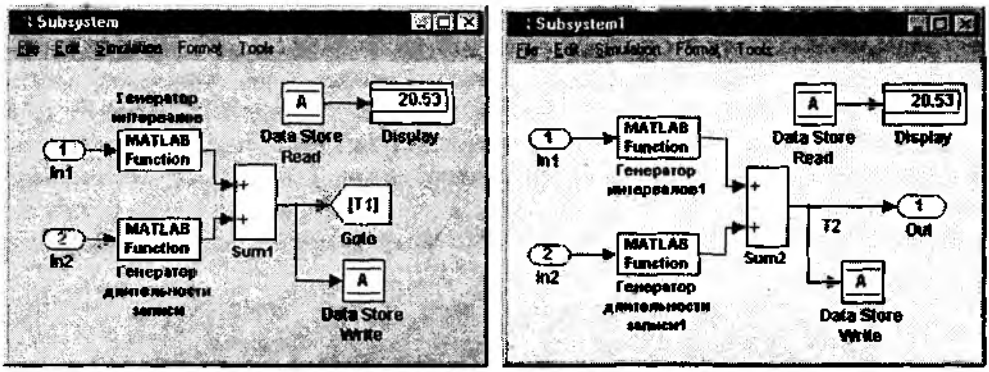


Рис. 4.53.Пример некорректного использования блока **Data Store Write**

сохраняются две разные величины — *T1* (время обслуживания очередного сообщения накопителем *HD1*) и *T2* (то же время, но для HD2).

В данном случае способность SIMULINK к квазипараллельной обработке событий может привести к ошибочному результату, поскольку неизвестно, какая именно из двух величин будет записана под именем *A*.

4.4.2. РАЗРЕШИТЬ – НЕ РАЗРЕШИТЬ

И при изложении материала I части, и при описании технологии создания *S*-моделей мы уже не раз затрагивали проблему синхронизации параллельных процессов. Наиболее остро она встает в тех случаях, когда *S*-модель содержит несколько подсистем, которые должны «согласовывать» свою работу друг с другом. Именно поэтому основные средства синхронизации в SIMULINK относятся к механизму использования подсистем. Эти средства реализованы в виде двух блоков, входящих в раздел библиотеки *Connections*:

Enable (*Разрешить*);

Trigger (в данном случае лучше перевести не как *Триггер*, а как *Защелка*).

Оба блока могут использоваться только внутри подсистем, поэтому если вы попытаетесь вставить любой из них в поле основной блок-диаграммы, SIMULINK не позволит это сделать и выдаст поясняющее сообщение.

Подсистему, которая содержит хотя бы один из блоков **Enable** и **Trigger**, будем называть ***управляемой***. Логика работы управляемой подсистемы зависит от того, какой именно блок входит в ее состав.

Рассмотрим особенности использования подсистем, содержащих блок **Enable** (будем в дальнейшем называть их *E*-подсистемами).

Иконка такой подсистемы имеет такой же символ, как и на самом блоке **Enable**. Кроме того, при включении этого блока в состав подсистемы ей авто-

матически добавляется управляющий входной порт, обозначение которого также выводится на иконке.

E-подсистема может содержать в своем составе любые блоки SIMULINK, как непрерывные, так и дискретные. Подсистема запускается в том случае, если управляющий входной сигнал имеет положительное значение.

Ели управляющий сигнал может изменять полярность, то *E*-подсистема начинает работу на том шаге моделирования, когда сигнал переходит нулевое значение в положительном направлении и работает до тех пор, пока он остается положительным.

Разработчик модели может в определенном смысле корректировать логику работы E-подсистемы, изменяя параметры настройки блока **Enable**. Таких параметров два (рис. 4.54):

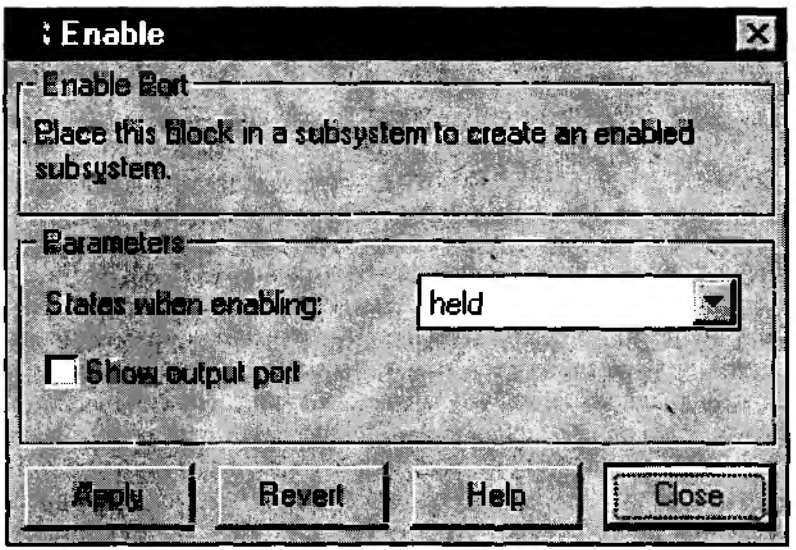


Рис. 4.54. Окно настройки параметров блока **Enable**

States when enabling (*Состояние при запуске*) определяет состояние подсистемы в момент очередного запуска; его значение выбирается с помощью выпадающего меню, которое содержит два пункта:

- *held* — использовать предыдущее состояние;
- *reset* — использовать начальное (исходное) состояние подсистемы;

Show output port (*Показать выходной порт*); при «включенном» переключателе блок **Enable** имеет выходной порт, сигнал с которого может быть использован для управления блоками, входящими в состав подсистемы.

Применение блока **Enable** поясним воспользовавшись уже знакомой вам моделью работы накопителя. Будем считать, что накопитель может начать записывать очередное сообщение только после того, как закончится запись предыдущего. Другими словами, изменение занятой емкости накопителя происходит только в том

случае, если интервал времени до появления очередного сообщения превышает длительность записи предыдущего.

Соответствующая блок-диаграмма показана на рис. 4.55.

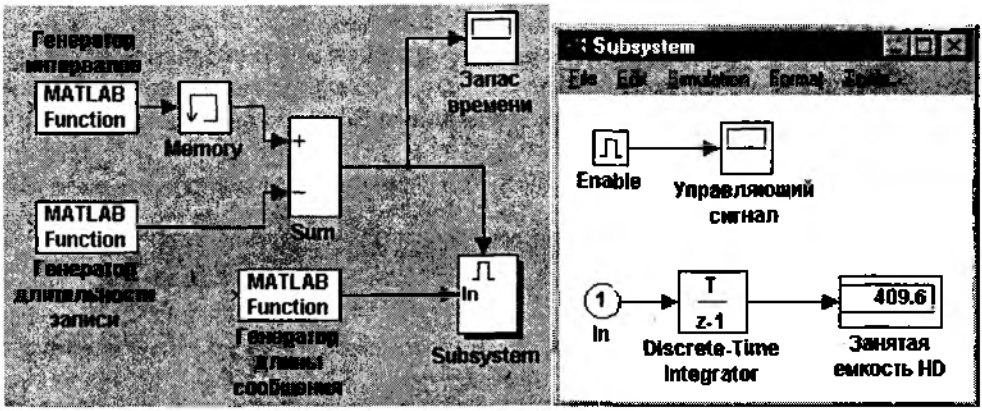


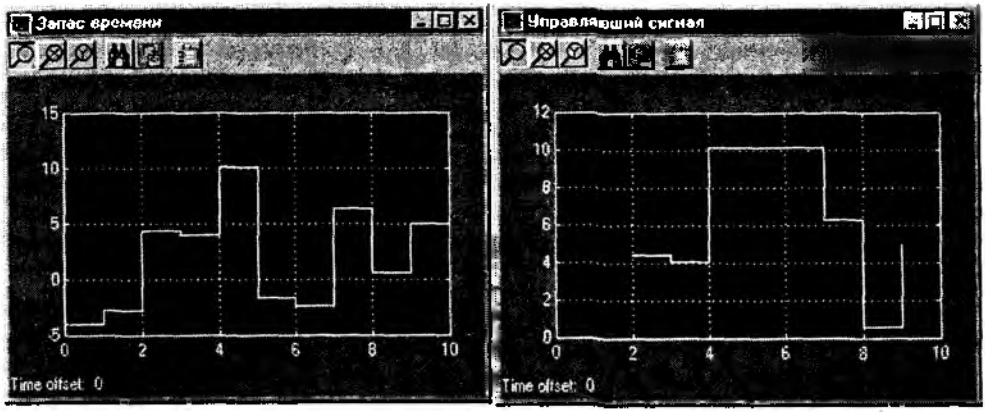
Рис. 4.55.Пример использования *E*-подсистемы

В данном варианте модели назначение трех блоков **MATLAB Fcn** осталось прежним: они используются в качестве генераторов соответствующих случайных величин. Блок **Memory** нужен для того, чтобы «задержать» появление очередного сообщения на один шаг моделирования по отношению к началу записи предыдущего. При этом предполагается, что за время записи сообщения в системе не может появиться более одного нового (выполнение этого условия обеспечивается за счет подбора значений параметров распределений соответствующих СВ).

Блок **Sum** вычисляет величину промежутка времени от начала записи сообщения до появления следующего; вычисленные с его помощью значения могут быть просмотрены в окне **Scope** с именем *Запас времени*. Сигнал *Запас времени* управляет работой подсистемы, состав которой показан на том же рисунке. По сути, она содержит единственный функциональный блок — **Discrete-Time Integrator**, который вычисляет занятую емкость накопителя. В соответствии с логикой работы модели, увеличение занятой емкости происходит только при положительном *Запасе времени*. В противном случае очередное поступившее сообщение «теряется» и его запись не производится. На рис. 4.56 приведены временные диаграммы управляющего сигнала на входе и на выходе блока **Enable**.

Замечание. В рассмотренном примере параметр *States when enabling* блока **Enable** обязательно должен иметь значение *held*, иначе величина занятой емкости *HD* будет вычисляться заново при каждом запуске подсистемы.

Работа подсистем, управляемых с помощью блока **Trigger** (в дальнейшем — *T*-подсистемы), несколько отличается от рассмотренной выше. Основное отличие состоит в том, что *T*-подсистема работает только на том шаге моделирования, на



а) б)

Рис. 4.56. Временные диаграммы входного (а) и выходного (б) сигналов блока **Enable**

котором произошло изменение полярности входного сигнала. Другое отличие заключается в том, что *T*-подсистема не возвращается в исходное состояние; ее текущее состояние сохраняется до очередного запуска.

Иконка *T*-подсистемы содержит символ блока **Trigger**, однако возможны три различные модификации его изображения, которые зависят от значения параметра ***Trigger type*** (*Тип триггера*) блока **Trigger**. Выбор значения этого параметра производится в окне настроек блока с помощью выпадающего меню (рис. 4.57).

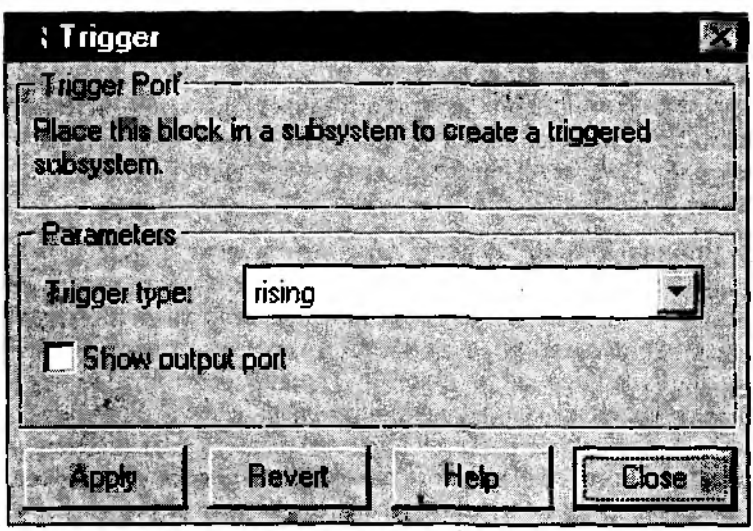


Рис. 4.57. Окно настройки параметров блока **Trigger**

Меню содержит четыре пункта:

rising обеспечивает запуск подсистемы при изменении полярности управляющего сигнала в положительном направлении;

falling — запуск подсистемы происходит при изменении полярности сигнала в отрицательном направлении;

either — запуск подсистемы разрешен при любом изменении полярности управляющего сигнала;

function-call — запуск системы определяется логикой работы заданной *S*-функции; подробнее понятие *S*-функции будет рассмотрено в Главе 5.

Вторым параметром настройки блока **Trigger** является переключатель ***Show output port***, который используется аналогично одноименному параметру блока **Enable**.

T-подсистемы, как и E-подсистемы, могут содержать любые блоки из библиотеки SIMULINK. Однако налагаются некоторые ограничения на использование тех блоков, которые имеют параметр *Sample time*: его значение должно быть установлено равным –1, которое указывает, что дискретность изменения времени «наследуется» от предшествующего блока. Кроме того, если в составе *T*-подсистемы имеются блоки, в параметрах настройки которых имеется флажок *Inherit sample time* (*Наследуемый шаг времени*), то он должен быть установлен.

К сожалению, с помощью *T*-подсистемы нельзя без специальных ухищрений решить ту же задачку о накопителе, которая была успешно решена с помощью *E*-подсистемы. Поэтому в качестве иллюстрации применения *T*-подсистемы рассмотрим совершенно новую ситуацию.

Предположим, что имеется система отопления, которая чутко реагирует на изменение температуры наружного воздуха и включается только в том случае, если температура опускается ниже нуля.

На рис. 4.58 показана *S*-модель, которая позволяет рассчитать стоимость отопления за 20 дней.

В модели роль датчика температуры исполняет блок **Uniform Random Number**; блок **Trigger** настроен на падение температуры (параметр *Trigger type* имеет значение *falling*); блок **Discrete-Time Integrator**, работающий в данном случае «Счетчиком», имеет параметр *Sample time* = –1.

Третий тип управляемой подсистемы — это подсистема, содержащая и блок **Enable**, и блок **Trigger** (назовем ее *ET*-подсистемой). На ее иконке выводятся символы обоих блоков и она имеет два управляющих входных порта. Правило запуска такой подсистемы состоит в следующем. Если на некотором шаге моделирования происходит изменение полярности «триггерного» сигнала, то SIMULINK проверяет значение сигнала на втором управляющем входе. Если оно больше нуля, то подсистема запускается и работает в течение данного шага моделирования. Если оба управляющих сигнала — векторы, подсистема выполняется в том случае, если по крайней мере один элемент каждого вектора отличен от нуля.

Пользователь может управлять логикой запуска *ET*-подсистемы, изменяя параметры настройки блоков **Enable** и **Trigger**.

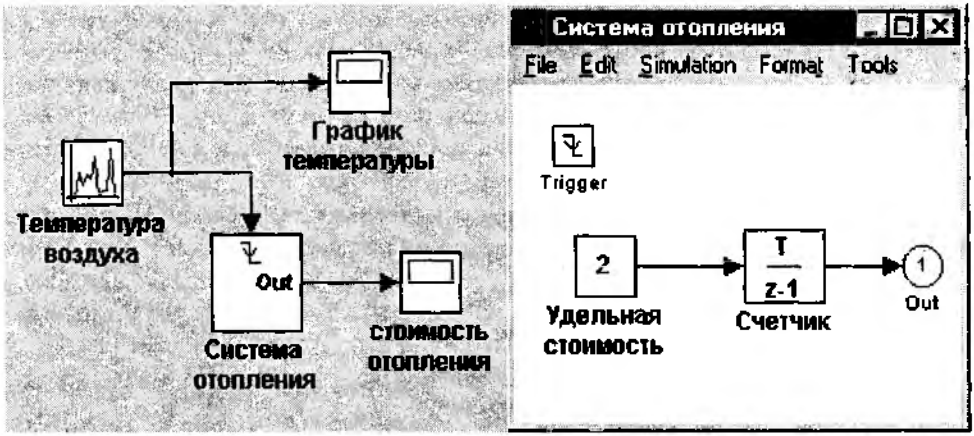


Рис. 4.58.Пример использования *T*-подсистемы

Наличие двух управляющих входов в *ET*-подсистеме позволяет описывать в модели достаточно сложные правила ее запуска.

4.4.3. МАСКИРОВАНИЕ ПОДСИСТЕМ

Маскированная подсистема (*Masked Subsystem*) — это такая подсистема, структуру которой нельзя увидеть непосредственно из блок-диаграммы; кроме того, она имеет собственное диалоговое окно настройки параметров и, как правило, собственную иконку.

Другими словами, маскированная подсистема — это аналог библиотечного блока SIMULINK.

Механизм использования маскированных систем обладает следующими достоинствами:

- значительно расширяет интерактивные возможности пользователя по управлению параметрами *S*-модели;
- позволяет создавать более понятный интерфейс за счет введения необходимых комментариев как к подсистеме в целом, так и к отдельным параметрам;
- обеспечивает повышение наглядности блок-диаграммы;
- повышает защищенность *S*-модели от неумышленной модификации (а в некоторых случаях — и от умышленной).

Создание маскированной подсистемы предполагает выполнение следующих действий:

- описание параметров подсистемы;
- определение способов изменения параметров;
- создание необходимых комментариев;
- создание собственной иконки подсистемы.

Все эти действия выполняются с помощью *Редактора Маски* (*Mask Editor*). Для запуска *Редактора Маски* необходимо выбрать в меню пользователя (раздел *Edit*) команду *Mask Subsystem*. Предварительно в блок-диаграмме *S*-модели должна быть выделена та подсистема, для которой вы собираетесь создать маску.

При запуске *Редактора Маски* открывается диалоговое окно, содержащее три вкладки: *Icon*, *Initialization*, *Documentation*.

Первая из них предназначена для создания иконки подсистемы, вторая обеспечивает создание диалогового окна настройки параметров подсистемы, а третья позволяет добавлять в диалоговое окно подсистемы необходимые комментарии.

В нижней части окна *Редактора Маски* расположены пять кнопок, которые являются общими для всех трех вкладок:

- *Apply* позволяет применить к маскируемой подсистеме опции, заданные на всех трех закладках;
- *Revert* обеспечивает восстановление тех значений параметров маски, которые были установлены на момент открытия *Редактора*;
- *Unmask* позволяет «демаскировать» подсистему; при этом окно *Редактора* закрывается; параметры «снятой» маски запоминаются SIMULINK и могут быть восстановлены с помощью команды *Mask Subsystem* из раздела меню пользователя *Edit* (соответствующая подсистема должна быть выделена); после закрытия *S*-модели информация о маске теряется;
- *Help* открывает раздел документации в формате .html, описывающий технологию маскирования подсистем;
- *Close* работает так же, как и кнопка *Apply*, но дополнительно закрывает окно *Редактора Маски*.

Теперь рассмотрим подробно те средства, которые предоставляет в распоряжение пользователя *Редактор Маски*.

Начнем с закладки *Initialization* (рис. 4.59).

Она содержит следующие элементы:

1. Строка редактирования *Mask type* (*Тип маски*) — предназначена для указания типа (названия) создаваемой маскированной подсистемы (аналог *типа* блока из библиотеки — **Constant**, **Sum** и т. д.); допускается использовать русскоязычные названия.

2. Окно, содержащее список параметров настройки подсистемы;

Для работы со списком служат расположенные слева от него четыре кнопки:

- *Add* — добавить элемент списка;
- *Delete* — удалить элемент списка;
- *Up* — поднять элемент списка на одну строку вверх;
- *Dawn* — опустить элемент на одну строку вниз.

Для использования последних трех кнопок необходимо предварительно выделить строку в списке, щелкнув на ней ЛКМ; расположение параметров в списке определяет их положение в диалоговом окне настройки. Список может содержать не более 14 параметров.

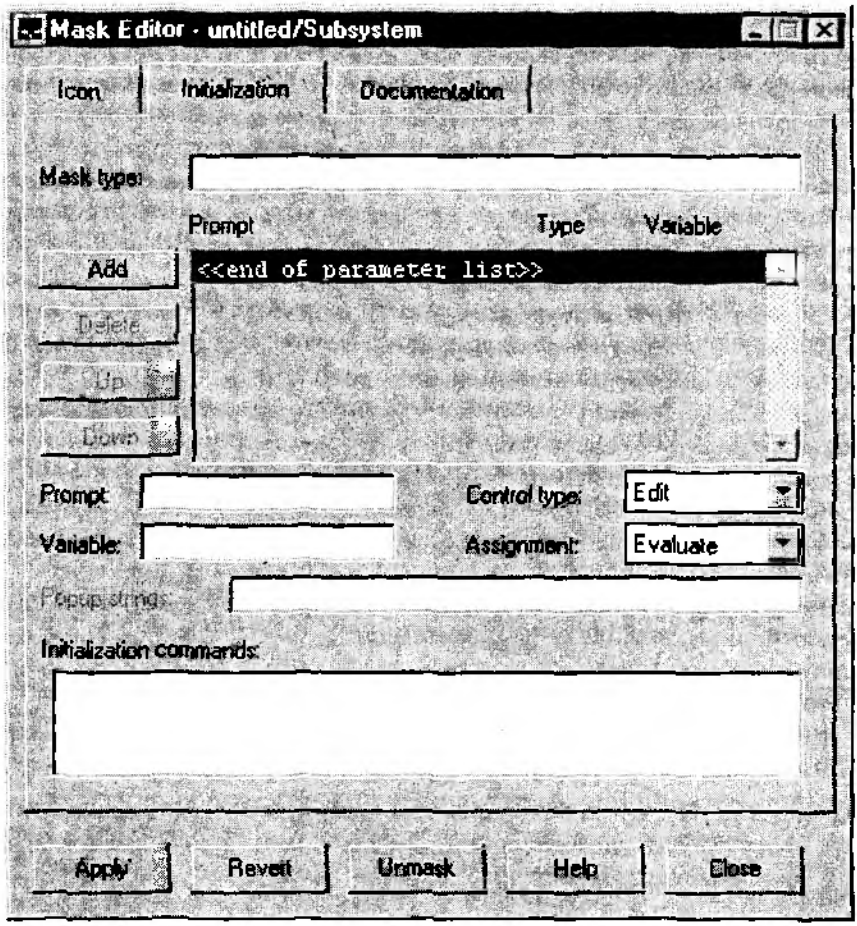


Рис. 4.59. Закладка *Initialization* окна *Редактора Маски*

3. Средства описания характеристик параметров подсистемы:

• строка редактирование *Prompt* (*подсказка, пояснение*); служит для ввода названия параметра (например, «Коэффициент усиления», «Первое слагаемое» и т. д.); допускается использовать русскоязычные названия , хотя в списке элементов они отображаются некорректно (требуется дополнительная настройка системных шрифтов);

• строка редактирования *Variable* (*Переменная*) предназначена для ввода имени переменной, в которой будет храниться значение соответствующего параметра; имя переменной может состоять только из латинских букв и цифр; в остальном выбор имени может быть произвольным;

• *выпадающее меню Control type* (*Тип управления*) позволяет устанавливать способ ввода значения для каждого параметра:

Edit — с помощью строки редактирования;

Checkbox — с помощью переключателя;

Popup — с помощью выпадающего меню; выпадающее меню *Assignment* (*Назначение*) — предназначено для указания типа параметра:

Evaluate — вычисляемый (т. е. имеющий численное значение);

Literal — символьный (воспринимается SIMULINK как строка символов).

4. Строка редактирования *Popup strings* (*Строки выпадающего меню*) служит для ввода пунктов выпадающего меню, используемого для выбора значений соответствующего параметра; становится доступной, когда этот параметр выбран в списке; вводимые в этой строке пункты меню должны быть разделены вертикальной чертой.

5. Область ввода *Initialization commands* (*Команды инициализации*), предназначенная для ввода списка команд инициализации маски.

Команды инициализации оперируют с переменными, находящимися в рабочей области маскированной подсистемы (*Mask Workspace*). Они представляют собой обычные команды MATLAB и могут содержать М-функции и операторы MATLAB. Другими словами, поле *Initialization commands* можно считать аналогом командного окна MATLAB, область действия которого ограничена рабочей областью маскированной подсистемы.

SIMULINK может выполнять команды инициализации в следующих случаях:

- при открытии окна блок-диаграммы модели;
- при запуске модели на исполнение;
- при обновлении блок-диаграммы (по команде *Update diagram*)
- при вращении блока маскированной подсистемы (с целью перерисовки иконки)
- для автоматического изменения иконки, зависящей от параметров блока.

Поясним технологию создания собственного окна настройки параметров на примере подсистемы, изображенной на рис. 4.60 (справа показана блок-диаграмма, в состав которой она включена).

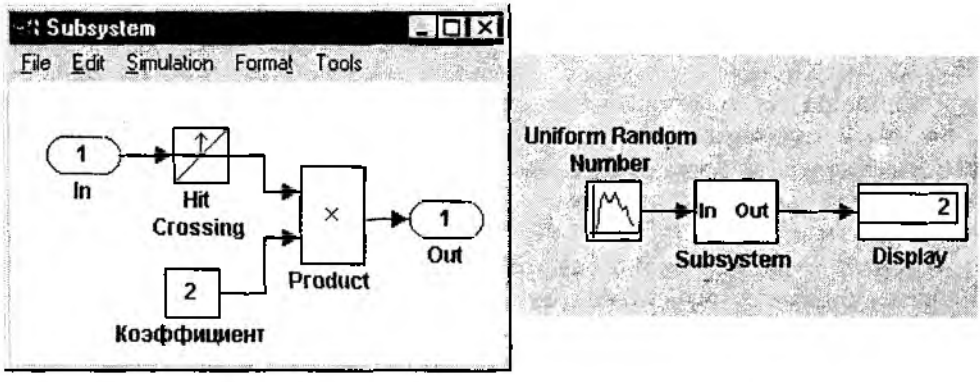


Рис. 4.60. Маскируемая подсистема (исходное состояние)

С ее помощью выполняются следующие действия: входной сигнал сравнивается с некоторым пороговым значением (посредством блока **Hit crossing**), и если порог превышен, то на выход подсистемы выдается число, которое задается константой *Коэффициент*.

Начнем с того, что выберем общее имя для подсистем такого типа, например «Умножитель». Введите его в строке *Mask type* и «нажмите» кнопку *Apply*. С этого момента подсистема стала маскированной — она получила свою «маску» в виде диалогового окна настроек. Чтобы ее увидеть, необходимо дважды щелкнуть ЛКМ на изображении блока подсистемы в окне блок-диаграммы.

Пока диалоговое окно («маска») содержит только название подсистемы и кнопки управления, назначение которых совпадает с назначением аналогичных кнопок в окне настроек любого библиотечного блока SIMULINK; правда, в данном случае доступна только одна из них — *Close* (рис. 4.61).

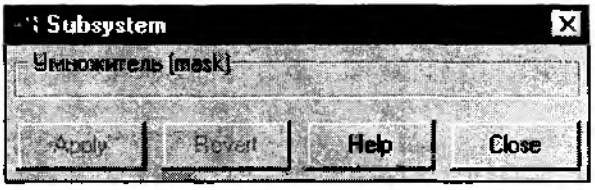


Рис. 4.61. Окно настроек для маскируемой подсистемы после выполнения первого шага

Следующим шагом в создании маски является описание параметров настройки подсистемы.

Пусть таких параметров будет два: значение порога, задаваемого в блоке **Hit Crossing**, и величина *Коэффициента*.

Для описания первого из них необходимо:

1. «Нажать» кнопку *Add* (чтобы внести первый элемент в список параметров);
2. В поле *Prompt* ввести «Порог» (пояснение к параметру; после нажатия клавиши <Enter> оно отображается в списке параметров);
3. В поле *Variable* ввести произвольное имя переменной, в которой будет храниться значение данного параметра, например *porog*; чтобы внести его в список параметров, следует еще раз нажать клавишу <Enter>.
4. В поле *Control type* оставить значение *Edit* (оно уже установлено по умолчанию в списке параметров).
5. В поле *Assignment* также оставить установленное по умолчанию значение *Evaluate*.

После «нажатия» кнопки *Apply* можно вновь открыть диалоговое окно подсистемы. Теперь оно выглядит несколько «солиднее» (рис. 4.62).

Диалоговое окно позволяет ввести значение параметра *Порог*, однако если это сделать и запустить модель, SIMULINK выдаст сообщение об ошибке. Дело в том, что он пока «не знает», каким образом параметр *Порог* связан с соответствующим параметром блока **Hit Crossing** (и связан ли вообще). Для установления

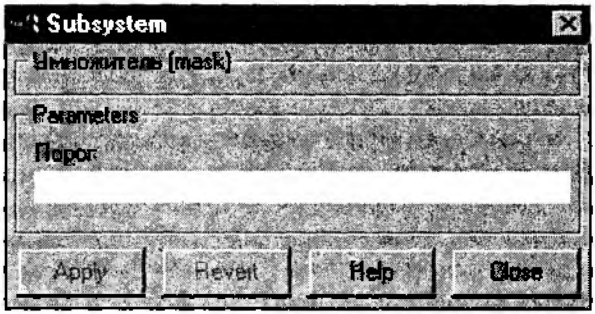


Рис. 4.62. Вид «маски» после описания первого параметра

такой связи как раз и используется имя переменной *porog*, относящейся к параметру *Порог*.

Связь между параметрами маскированной подсистемы и параметрами блоков, входящих в ее состав, реализуется через упоминавшуюся уже рабочую область подсистемы (*Mask Workspace*). Схематично эта связь показана на рис. 4.63.

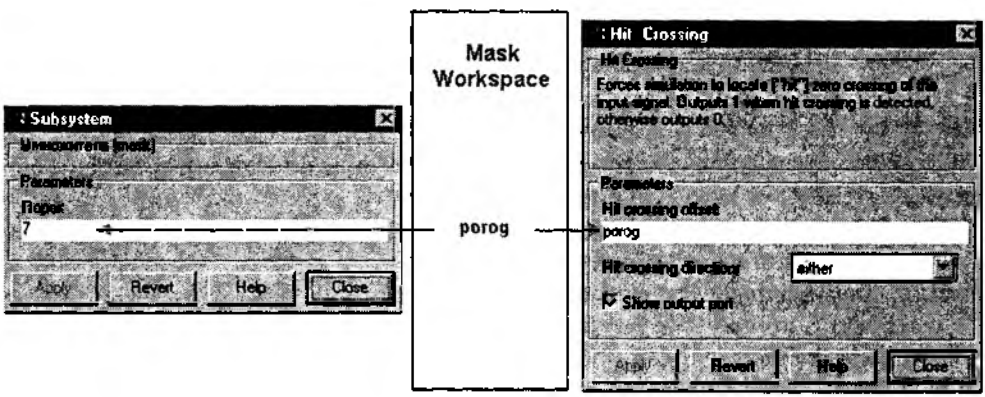


Рис. 4.63. Взаимосвязь параметров подсистемы и входящих в нее блоков

Таким образом, чтобы увязать параметр *Порог* с параметром настройки блока **Hit Crossing**, необходимо ввести в поле параметра *Hit Crossing offset* имя переменной *porog*.

Чтобы изменить значение параметра блока, входящего в маскированную подсистему, требуется «заглянуть ей под маску» (т. е. открыть окно блок-диаграммы подсистемы). Это можно сделать с помощью команды *Look Under Mask* из раздела *Edit* меню пользователя.

Несколько слов о том, что собой представляет рабочая область маскированной подсистемы.

Mask Workspace — это локальная рабочая область исполняемой подсистемы. Она создается SIMULINK в следующих случаях:

• если маска подсистемы содержит команды инициализации;

• если при описании маски заданы параметры настройки подсистемы (как в нашем примере).

Переменные, содержащиеся в рабочей области маскированной подсистемы, доступны всем блокам, входящим в ее состав. Другими словами, при создании маски можно определять параметры, которые должны иметь одинаковое значение для всех блоков, входящих в состав маскированной подсистемы (например, дискретность шага моделирования).

Продолжим создание маски для подсистемы «Умножитель» и опишем второй параметр, выполнив следующие действия:

• «Нажать» кнопку *Add* для добавления второй строки в список параметров;

• В поле *Prompt* ввести *Коэффициент*;

• В поле *Variable* указать имя переменной, например **k**.

Значения полей *Control type* и *Assignment* следует оставить без изменения.

После «нажатия» кнопки *Apply* можно открыть скорректированное окно настройки параметров подсистемы «Умножитель».

Чтобы изменить взаимное расположение полей «Коэффициент» и «Порог», следует поменять местами соответствующие строки в списке параметров окна *Редактора Маски* (с помощью кнопок *Up* или *Down*).

Параметр *Коэффициент*, как и параметр *Порог*, необходимо связать с соответствующим блоком подсистемы (блоком **Коэффициент**). Чтобы создать связь, требуется ввести в окне настройки этого блока имя переменной **k**.

После завершения описания обоих параметров вкладка *Initialization* выглядит так, как показано на рис. 4.64, а само диалоговое окно настройки параметров подсистемы — на рис. 4.65.

Вкладка *Icon* содержит средства создания индивидуальной иконки для маскируемой подсистемы. На ней расположены следующие элементы управления (рис. 4.66):

1. Строка редактирования *Mask type*, дублирующая одноименную строку на закладке *Initialization*.

2. Поле ввода *Drawing commands* (*Команды рисования*), предназначенное для ввода команд, создающих графическое изображение на иконке.

3. Выпадающее меню *Icon frame*, которое позволяет устанавливать способ вывода прямоугольной рамки иконки:

• *Visible* — рамка выводится,

• *Invisible* — рамка не выводится (*Невидима*).

4. Выпадающее меню *Icon transparance*, устанавливающее «прозрачность» нового образа иконки:

• *Opaque* — изображение, выводимое на иконке, скрывает стандартный «образ»;

• *Transparent* — новая иконка является «прозрачной», т. е. и новое, и стандартное изображения выводятся одновременно; например, если маскируемая подсистема имеет входной и выходной порты, то их названия (*In* и *Out*) будут «просвечивать» через новое изображение.

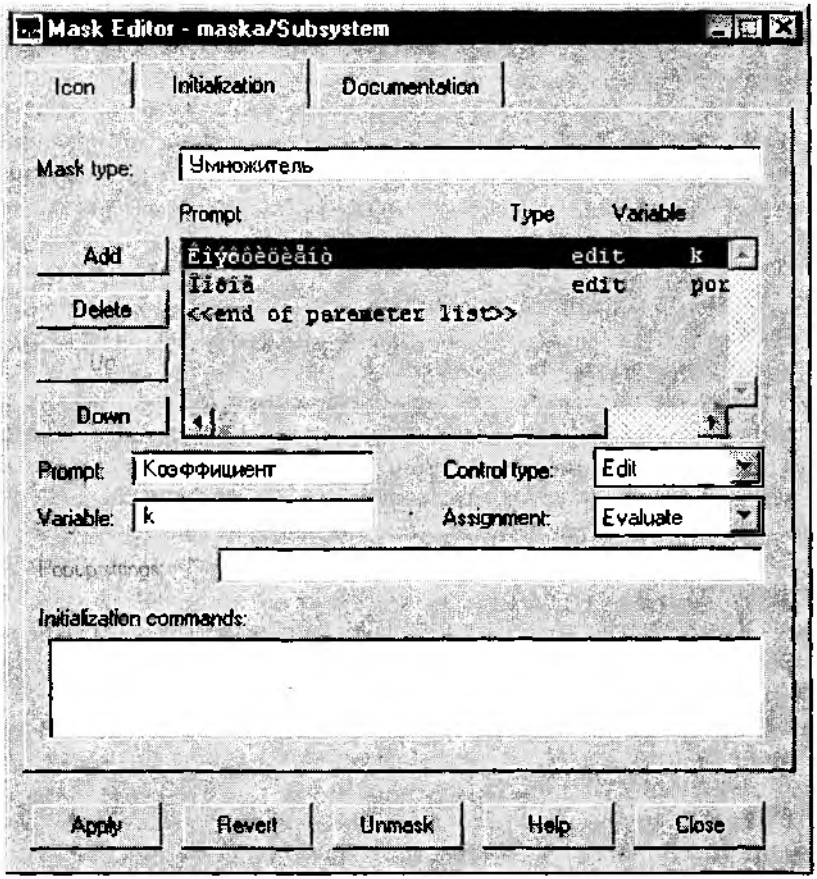


Рис. 4.64. Вкладка *Initialization* после завершения создания диалогового окна подсистемы

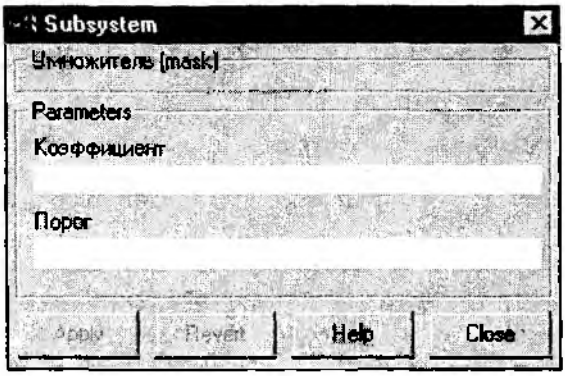


Рис. 4.65.Окончательный вид диалогового окна подсистемы

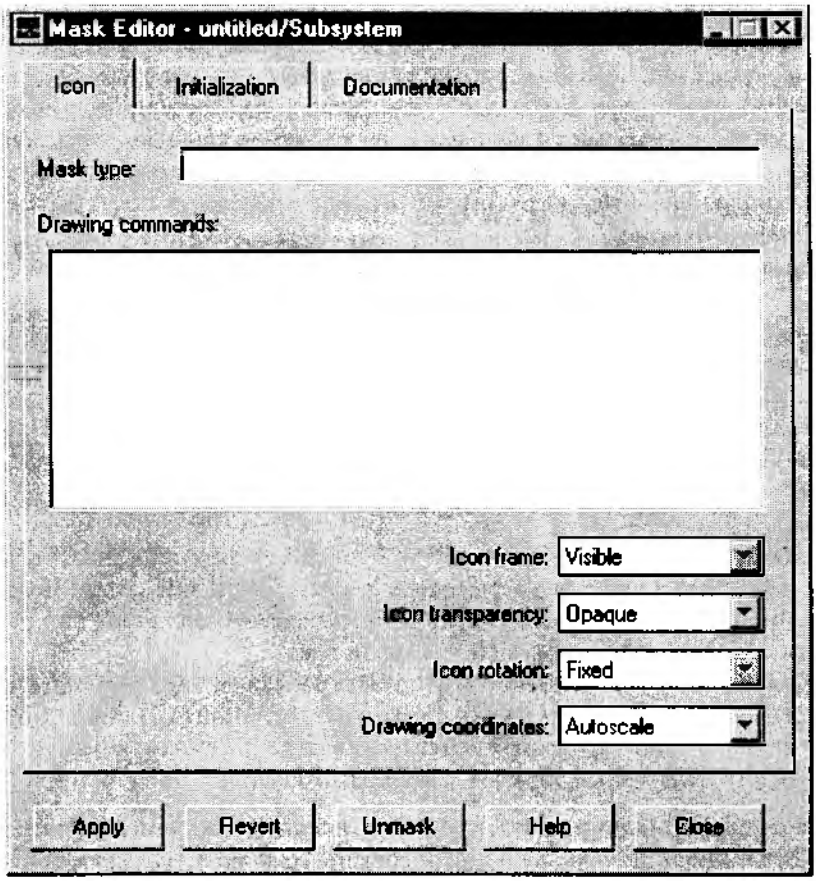


Рис. 4.66. Вкладка *Icon* диалогового окна *Редактора Маски* подсистемы

5. Выпадающее меню *Icon rotation* определяет, будет ли поворачиваться изображение на иконке при повороте блока подсистемы:

Rotates — изображение выводится с учетом поворота блока;

Fixed — изображение не корректируется.

6. Выпадающее меню *Drawing coordinates* позволяет выбрать систему координат для вывода изображения на иконке:

• *Autoscale* — автоматическая установка масштаба осей; при использовании данной опции размер иконки изменяется пропорционально изменению размеров контура блока;

• *Normalized* — для вывода иконки устанавливается постоянный масштаб; при этом нижний левый угол рамки иконки (контура блока) имеет координаты (0, 0), а правый верхний угол — координаты (1, 1); соответственно при выводе графика

на иконке значения координат *x* и *y* должны находиться в пределах [0, 1]; при изменении размеров блока размер иконки изменяется автоматически;

• *Pixel* — значения координат *x* и *y* для выводимого графика должны быть указаны в пикселах; при изменении размеров блока иконка не масштабируется; чтобы обеспечить ее автоматическое масштабирование, необходимо в командах «рисования» указывать дополнительные параметры: текущую высоту и ширину блока (*height* и *width*).

Теперь подробнее о том, какие именно *Команды рисования* могут быть введены в поле *Drawing Commands*.

Для вывода текста на иконке может использоваться одна из трех команд:

• *disp*('текст') — вывод указанного текста в центре иконки;

• *text*(*x*, *y*, 'текст') — вывод текста, начиная с позиции, задаваемой координатами *x* и *y*;

• *fprintf*('текст') — вывод форматированного текста по центру иконки.

Во всех трех командах выводимый текст может быть указан не непосредственно, а через имя переменной, в которой хранятся символьные данные. Например, команда disp может быть использована для вывода в иконке слова 'Подсистема' двумя способами:

• *disp*('Подсистема'),

• *disp*(*variable*), где *variable* — имя переменной, в которую предварительно был записан выводимый текст; эта переменная должна храниться в *Mask Workspace*, а ее значение может быть задано в поле *Initialization commands* (на закладке *Initialization*), например так: *variable* = 'Подсистема'.

Если выводимый в иконке текст должен быть размещен в виде нескольких строк, то конец каждой строки обозначается символом табуляции (\n). Например, чтобы вывести надпись «Маскированная подсистема» на двух строках, команда disp должна выглядеть следующим образом: *disp*('Маскированная\nподсистема').

Для вывода графиков на иконке могут быть использованы стандартные средства MATLAB, т. е. библиотечные функции рисования графиков. Наиболее универсальной из них является функция *plot*, которая в простейшем случае имеет формат *plot*(*x*, *y*), где *x* и *y* — векторы, содержащие равное число элементов. При создании иконки аргументы функции должны храниться в рабочей области подсистемы, а их значения могут быть заданы в поле *Initialization Commands*, например так:

$$x=[0\ 1\ 2\ 3];\ y=[1\ 4\ 1\ 4].$$

Если на иконке требуется отобразить вид функции, выполняющей преобразование входного сигнала (реализуемой с помощью данной подсистемы), то для этого используется функция *dpoly* с соответствующими параметрами.

SIMULINK позволяет также создавать иконки с изображениями, нарисованными «вручную». Такая иконка создается с помощью специального графического редактора. Для его запуска необходимо в командном окне MATLAB ввести команду *iconedit*.

Если данная команда вводится без параметров, то MATLAB запрашивает дополнительную информацию: название окна блок-диаграммы, в котором находит-

ся редактируемый блок (*Name of block window*) и имя блока, под которым он используется в диаграмме (*Name of block*).

Эти данные можно указать и непосредственно при вводе команды *iconedit*. Например, для маскируемой нами подсистемы она будет выглядеть так:

iconedit('*maska*', '*Subsystem*').

Замечание. Для корректной работы редактора *iconedit* редактируемый mdl-файл обязательно должен находиться в активном директории; соответствующая настройка производится с помощью опции *Set Path* из командного окна MATLAB.

При запуске редактора *iconedit* открывается окно, содержащее координатную сетку и визир, перемещаемый с помощью «мыши» (рис. 4.67). Нажатие ЛКМ приводит к появлению точки в выбранной позиции. Очередная точка соединяется с предыдущей прямой линией.

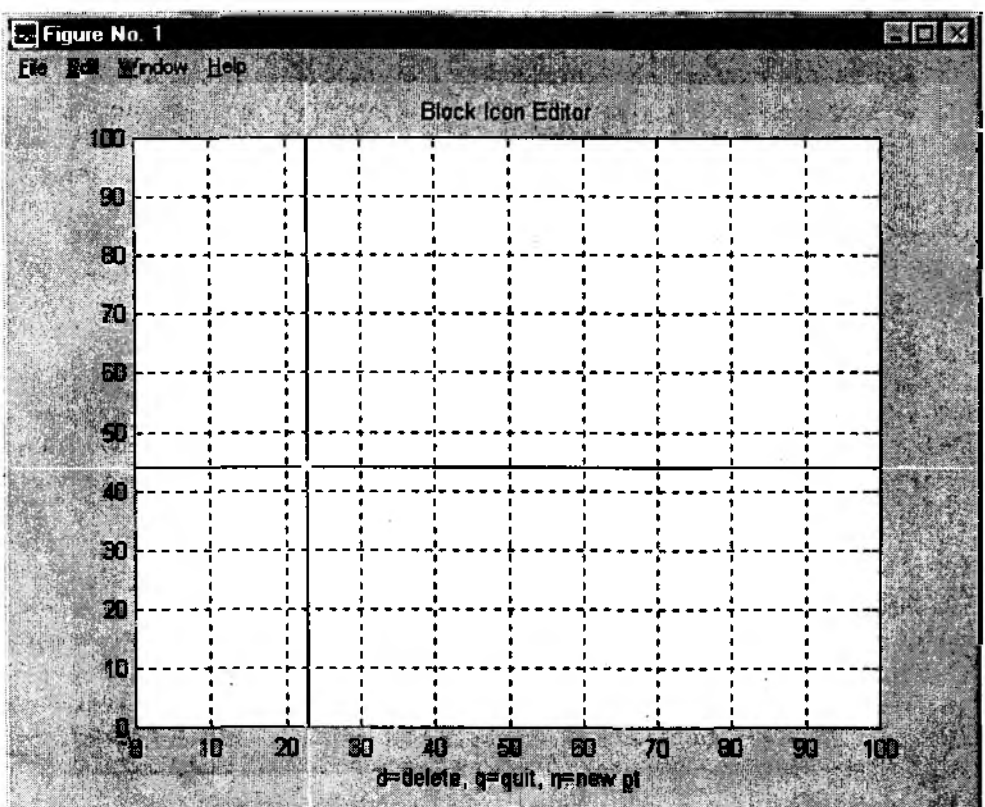


Рис. 4.67.Окно редактора иконки *iconedit*

При работе с редактором *iconedit* используются следующие команды, вводимые с клавиатуры:

- *d* — удаление последней точки;
- *n* — создание новой точки, не связанной с предыдущей;
- *q* — выход из редактора с автоматическим обновлением иконки блока.

После закрытия окна *iconedit* в поле *Drawing Commands* выводится команда *plot* с параметрами, соответствующими созданному изображению. При необходимости ее можно скопировать в буфер обмена *Clipboard* и использовать для создания аналогичного изображения на какой-либо другой иконке.

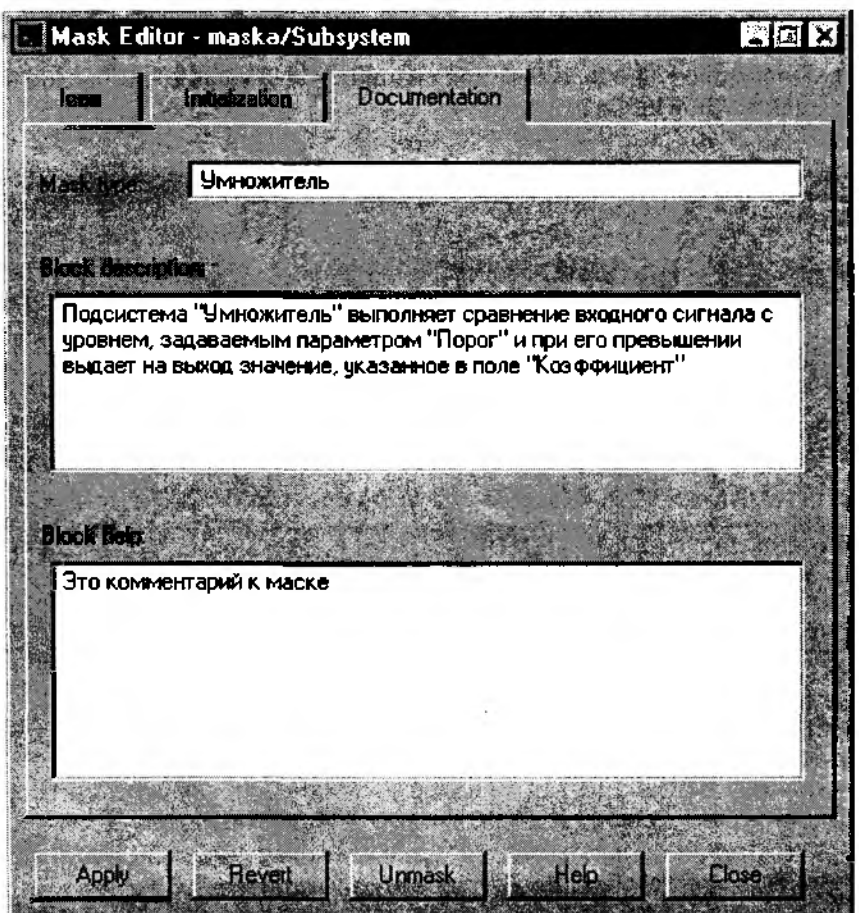


Рис. 4.68. Пример ввода справочной информации на вкладке *Documentation Редактора Маски*

Вкладка *Documentation Редактора Маски* содержит три области ввода:

• *Mask type*, дублирующее одноименные поля двух других закладок редактора;

• *Block description*, предназначенное для ввода пояснения к маскируемой подсистеме (это пояснение будет отображаться в окне настройки параметров подсистемы); форматирование вводимого в этом поле текста выполняется автоматически;

• *Block help*, обеспечивающее ввод информации, которая впоследствии будет доступна при «нажатии» кнопки *Help* в окне настройки параметров подсистемы; эта информация сохраняется в формате html, включается SIMULINK в состав его справочной системы и может быть просмотрена, например, с помощью *Internet Explorer*.

На рис. 4.68 приведен пример ввода информации на вкладке *Documentation*, а на рис. 4.69 показано, как эта информация отображается в соответствующих диалоговых окнах.

При необходимости в поле *Block help* вместо «статичного» текста могут содержаться другие средства помощи пользователю:

• адрес *Web*-узла, на котором можно найти дополнительные сведения, адрес электронной почты или другие *URL*-спецификации (*URL — Uniform Resource Locator*);

• команда *web*, запускающая *web*-броузер, установленный на вашем компьютере;

• команда *eval*, которая обеспечивает интерпретацию строк как исполняемых команд MATLAB; например, команда

eval('!*Word Book.doc*')

приводит к запуску редактора MS Word и открытию файла *Book.doc*.

Созданная маска может быть впоследствии изменена с помощью того же *Редактора Маски*. Его вызов в этом случае производится по команде *Edit Mask*, заменяющей для маскированной подсистемы команду *Mask Subsystem*.

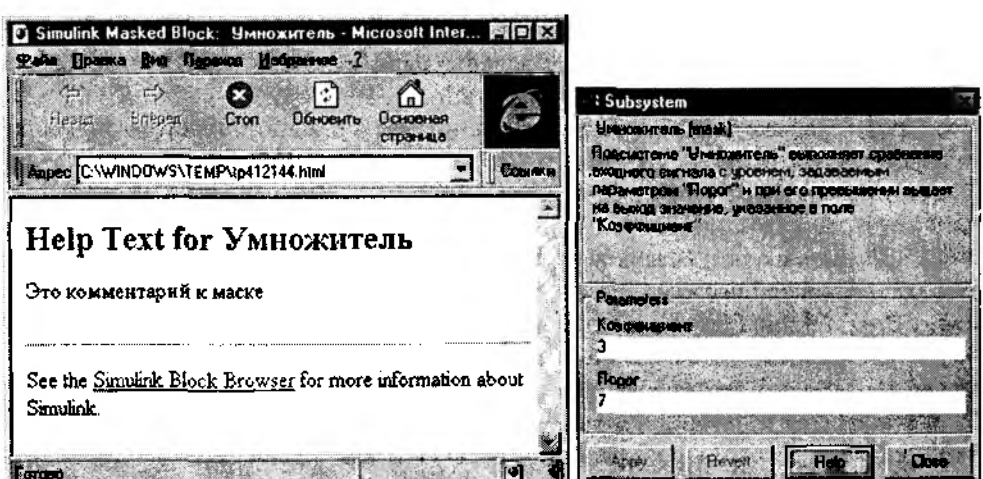


Рис. 4.69.Вывод справочной информации о маскированной подсистеме

Наличие в составе SIMULINK механизма маскирования делает его, как сейчас принято говорить, «открытой системой», т. е. доступной для доработки и расширения самими пользователями в соответствии с их потребностями. Именно этот механизм лежит в основе создания пользователями SIMULINK собственных библиотек блоков.

4.5. СОЗДАНИЕ СОБСТВЕННОЙ БИБЛИОТЕКИ БЛОКОВ

Как это ни парадоксально, основной недостаток любой универсальной системы заключается в ее универсальности. В том смысле, что решение конкретной задачи с помощью такой системы, как правило, требует определенной ее «настройки» на решаемую задачу. Если пользователь вынужден выполнять такую настройку многократно, то универсальность используемой системы начинает вызывать у него отрицательные эмоции. Именно поэтому наиболее популярными являются так называемые «открытые» системы, то есть такие, которые позволяют:

- во-первых, изменять характеристики системы (инструментальных средств) в достаточно широких пределах;
- во-вторых, сохранять внесенные изменения до тех пор, пока они остаются необходимы.

Возможность создания собственной библиотеки блоков — одно из проявлений «открытости» SIMULINK. Важную роль в реализации этой возможности играет механизм маскирования подсистем. Хотя в простейшем случае созданная пользователями библиотека может содержать один или несколько стандартных блоков SIMULINK, наиболее часто используемых в работе; при этом для каждого из них могут быть установлены требуемые значения параметров.

Для создания новой библиотеки необходимо:

1) в любом окне SIMULINK (блок-диаграммы или одного из существующих разделов библиотеки) выбрать в разделе *File* команду *New → Library*.

2) в открывшееся пустое окно поместить (в произвольном порядке)требуемые блоки, в том числе, возможно, маскированные подсистемы «собственного производства».

3) выделить в окне создаваемой библиотеки те блоки, которые были взяты из существующих библиотек; в разделе *Edit* этого же окна выбрать (и выполнить) команду *Break Library Link* — «разорвать связь с библиотекой»; в результате ее выполнения копии библиотечных блоков становятся «собственностью» новой библиотеки (и ее создателя); это, в свою очередь, приводит к трем важным последствиям:

- становится доступной команда *Look Under Mask*, что позволяет корректировать блок-диаграмму («содержимое») соответствующих блоков;
- становится доступной команда *Edit Mask*, что дает возможность изменять маску блоков, включенных в библиотеку;

• блоки *S*-моделей, создаваемых с помощью новой библиотеки, будут «связаны» только с ней.

Замечание: блоки, входящие в состав основной библиотеки SIMULINK, делятся на два вида: «базовые» и «производные»; «базовые» блоки включены в виде программного кода в состав ядра SIMULINK и их редактирование практически невозможно без применения «хаккерских» средств; команда *Break Library Link* для этих блоков неприменима; «производные» блоки представляют собой маскированные подсистемы и могут быть изменены описанным выше способом.

4) Сохранить созданную библиотеку на диске с помощью команды *Save as...*; библиотеки блоков, как и *S*-модели, сохраняются в файлах с расширением .mdl.

При последующих открытиях окна новой библиотеки (как и для всех других библиотек SIMULINK) становится доступной команда *Unlock Library* (*Открыть библиотеку*), входящая в раздел *Edit* меню пользователя. В результате ее выполнения библиотека становится доступной для редактирования (исключения имевшихся блоков и добавления новых, изменения форматов блоков и т. д.). При «открытии» библиотеки команда *Unlock Library* заменяется сообщением «*Library Unlocked*» (*Библиотека открыта*). В таком состоянии она остается до тех пор, пока не будет закрыто окно соответствующего mdl-файла.

Вы имеете возможность создавать «структурированные» библиотеки, т. е. состоящие из нескольких разделов (подобно основной библиотеке SIMULINK).

Раздел библиотеки создается на основе блока **Subsystem**. Как и при создании обычных подсистем, имеется два способа формирования раздела библиотеки:

• с помощью команды *Create Subsystem* (для этого в окне библиотеки должно быть выделено не менее двух блоков);

• путем копирования в окно библиотеки блока **Subsystem** из раздела *Connections*.

В обоих случаях вновь созданной подсистеме автоматически присваивается статус «библиотеки», о чем говорит соответствующий заголовок ее окна. В связи с этим имеется определенная «двойственность» при работе с разделами библиотеки:

• с одной стороны, с ними можно выполнять те же процедуры, что и с основной библиотекой (пополнять, «открывать» и т. д.);

• с другой стороны, к разделу библиотеки применимы практически все операции маскирования, рассмотренные в предыдущем разделе; в частности, вы можете создать для нее иконку, отражающую содержимое данного раздела библиотеки.

Созданный раздел может быть скопирован (или перемещен) в любую «открытую» библиотеку.

В качестве иллюстрации к изложенному в этом подразделе (и, частично, в предыдущем), на рис. 4.70 показана библиотека, содержащая некоторые элементы графического представления *E*-сети, а также вариант реализации с помощью этих элементов макропозиции «генератор».

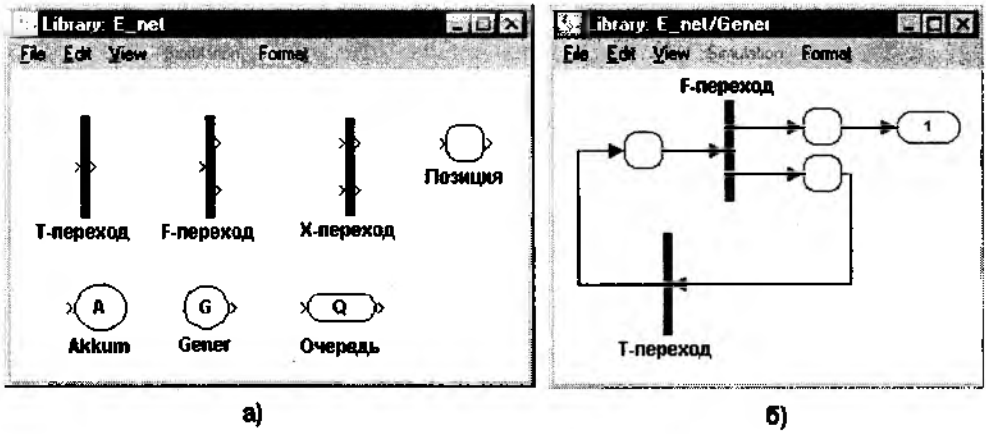


Рис. 4.70. Пример собственной библиотеки пользователя (а) и созданной с ее помощью блок-диаграммы (б)

4.6. ОТЛАДЧИК БЛОК-ДИАГРАММ (SIMULINK-DEBUGGER)

В логике и продуманности технических решений разработчикам SIMULINK не откажешь: если имеются визуальные средства построения моделей, то должны существовать и аналогичные средства отладки этих моделей.

Именно таким средством является отладчик блок-диаграмм *Simulink Debugger*.

Эта программа не создает собственного диалогового окна и использует для работы командное окно MATLAB и окно отлаживаемой блок-диаграммы; командное окно предназначено для ввода пользователем команд отладчика и вывода текстовых сообщений; в окне блок-диаграммы визуально отображается процесс работы отладчика.

Для запуска отладчика в командном окне следует ввести одну из двух команд:

sldebug '*<имя модели>*';
sim ('*<имя модели>*', [*Tstart Tfin*], *Simset*('*debug*', '*on*')).

Первую команду предпочтительнее использовать для версии MATLAB 5.2, вторую — для предыдущих версий пакета.

При выполнении команды запуска отладчика открывается окно блок-диаграммы указанной модели.

В блок-диаграмме «подсвечивается» блок, который является активным на первом шаге моделирования (при $tм=0$). Соответствующая информация выводится в текстовой форме в командном окне; после этого отладчик переводит модель в режим *Pause* (*Пауза*) и сам переходит в состояние ожидания следующей команды.

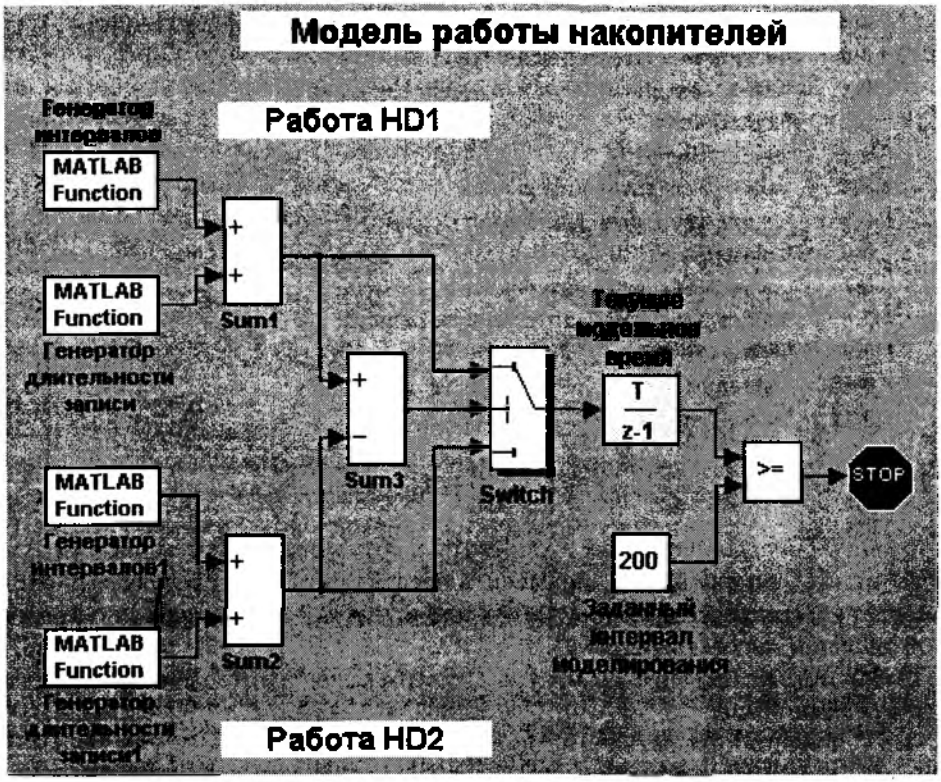


Рис. 4.71. Вид блок-диаграммы при запуске отладчика

Например, при отладке модели работы накопителей на первом шаге «подсвечивается» блок **Текущее модельное время** (рис. 4.71), а в командном окне выводится следующее сообщение:

```
[Tm=0          ]** Start** of system 'asinxr' outputs
(sldebug @ 0: 0 'asinxr' ('Текущее модельное время'):
```

В данной точке диалога вы можете обратиться за помощью к любой справочной системе MATLAB, просмотреть данные в рабочей области (если они там есть), выполнить любую команду MATLAB, либо ввести одну из команд отладчика.

Многие команды отладчика используют понятие «индекс блока». Он представляет собой номер блока внутри *S*-модели и отображается в команде отладчика в форме *s*: *b*,

где *s* — номер подсистемы *S*-модели (целое число),

b — номер блока в подсистеме (целое число).

Список индексов блоков можно получить с помощью команды *slist*. Данная команда не имеет параметров и применяется к активной *S*-модели. Результат ее работы выводится в командном окне MATLAB в следующем виде (рис. 4.72):

```
— Sorted list for 'asinxr' [12 blocks, 12 nonvirtual blocks, directFeed=0]
0:0    'asinxr/Текущее модельное время'
0:1    'asinxr/Заданный интервал моделирования'
0:2    'asinxr/Or'
0:3    'asinxr/Simu'
0:4    'asinxr/Генератор интервалов'
0:5    'asinxr/Генератор длительности записи'
0:6    'asinxr/Sum1'
0:7    'asinxr/Генератор интервалов1'
0:8    'asinxr/Генератор длительности записи1'
0:9    'asinxr/Sum2'
0:10   'asinxr/Sum3'
0:11   'asinxr/Switch'
```

Рис. 4.72. Список блоков *S*-модели, формируемый с помощью команды *slist*

Simulink Debugger позволяет проследить работу модели «шаг за шагом». В качестве таких «шагов» можно определить переход сигнала к следующему блоку, изменение модельного времени, либо промежуток работы модели до очередной «контрольной точки» (*breakpoint*).

Выбор типа «шага» определяется командой отладчика, которая может быть введена в режиме *Pause*:

step — продвижение сигнала на один блок;

next — продвижение на один шаг модельного времени;

continue — выполнение модели до следующей контрольной точки;

run — расчет модели до окончания заданного интервала моделирования (игнорируя контрольные точки).

Для завершения сеанса отладки служит команда *Stop*.

Рассмотрим подробнее особенности использования каждой из приведенных выше команд управления работой отладчика.

Команда *step*

При использовании этой команды отладчик выполняет текущий («подсвеченный») блок модели, приостанавливает ее работу и выделяет следующий блок.

Если следующий блок является подсистемой, то отладчик открывает ее блок-диаграмму и выделяет тот блок, с которого начинается работа подсистемы.

После выполнения блока отладчик выводит в командное окно значение входного (обозначается символом *U*) и выходного (обозначается *Y*) сигналов для выполненного блока и переходит в состояние ожидания следующей команды. Переменные *U* и *Y* выводятся с индексами, соответствующими порядковым номерам входных и выходных портов блока.

Если выполненный блок был последним для заданного интервала моделирования, то отладчик выбирает в качестве очередного первый блок модели, предоставляя возможность повторить сеанс моделирования. Чтобы обратить внимание пользователя на окончание предыдущего сеанса, в командном окне выводится также новое («обнуленное») значение модельного времени.

При использовании команды *step* у пользователя имеется возможность выбора «мелких» (*minor*) или «крупных» (*major*) шагов изменения модельного времени. Для выбора метода следует ввести в командном окне опцию *minor*.

Команда *next*

При использовании данной команды отладчик «проводит» сигнал через все блоки, которые должны быть выполнены на очередном шаге моделирования. Команду *next* целесообразно применять в тех случаях, когда разработчика интересует изменение состояния модели при изменении модельного времени; если очередной шаг является последним в данном сеансе моделирования, то отладчик делает активным первый блок модели и приостанавливает ее выполнение. После этого можно либо продолжить отладку, либо выполнить любую другую команду MATLAB.

Отладка на основе *контрольных точек*

Данный метод отладки является наиболее универсальным, поскольку контрольная точка может быть установлена практически в любом месте диаграммы модели. Между двумя контрольными точками моделирование выполняется без прерываний; для продолжения моделирования с последней контрольной точки (*КТ*) используется команда *Continue*.

Отладчик позволяет определять два типа контрольных точек: безусловные и условные. Прерывание в безусловной *КТ* происходит всякий раз, когда моделирование достигает блока или значения модельного времени, для которых задана *КТ*. Прерывание в условной *КТ* происходит в том случае, если возникли условия, установленные для данной *КТ*.

Для указания отладчику положения и типа контрольных точек используются следующие команды:

- *break <gcb|s:b>* — прерывание при входе в блок; блок, для которого устанавливается КТ, может быть задан либо с помощью индекса (*break s:b*), либо визуально; при втором способе следует выделить его в блок-диаграмме и ввести в командном окне команду *break gcb*;
- *bafter <gcb|s:b>* — прерывание при выходе из блока; как и предыдущая команда, может быть использована в одном из двух форматов:
  - *bafter s:b* — для указания блока с помощью индекса;
  - *bafter gcb* — при визуальном выборе блока.

Замечание: нельзя устанавливать *КТ* в виртуальном блоке. Виртуальный блок — это блок, имеющий чисто графические функции: он отображает группирование

или отношение подчиненности между функциональными блоками модели. При попытке установить *КТ* в таком блоке, отладчик выдает предупреждение. Список не виртуальных блоков можно получить с помощью команды *slist*;

• *tbreak <t>* — прерывание моделирования на заданном шаге; моделирование прерывается, как только текущее значение модельного времени становится больше (или равным) величины *t*; например, если введена команда *tbreak* 7, то при моделировании с непрерывным временем останов произойдет для t_M=7,02;

• для удаления «блочных» *КТ* используется команда *clear <gcb|s:b>* — которая также может использоваться в двух вариантах: «текстовом» и «визуальном»;

• *xbreak* — команда заставляет отладчик остановить моделирование, когда в модели используется переменный шаг (*Variable-step*) и встречается состояние, требующее ограничить величину шага;

• *zcbreak* — команда позволяет остановить моделирование, когда встречается непредусмотренное пересечение сигналом нулевого уровня; при обнаружении такой ситуации отладчик выводит в командное окно значение модельного времени и индекс блока, а также «тип пересечения» — в положительном направлении (*rising*) или в отрицательном (*failing*).

Управление выводом информации о процессе отладки

Получение дополнительной отладочной информации обеспечивается следующими командами:

• *probe* — вывод значений сигнала на входах и выходах блоков, выполняемых в течение очередного шага отладки; для вывода информации о конкретном блоке может использоваться в форме *probe <s:b>*; команда *probe* допустима в любом режиме отладки (по блокам, по модельному времени и по контрольным точкам);

• *disp <gcb|s:b>* — вывод значений сигнала на входе и выходе указанного блока при прерывании сеанса моделирования; действие команды может быть отменено командой *undisp<gcb|s:b>*;

• *trace <gcb|s:b>* — выполняет те же действия, что и команда *disp*, но без прерывания моделирования; действие ее отменяется командой *untrace <gcb|s:b>*;

• *states* — вывод информации о текущем состоянии модели.

В дополнение к отладочной информации отладчик позволяет получать некоторые сведения о структуре модели и входящих в нее блоках.

Основным средством для этого служит уже упоминавшаяся команда *slist*.

Кроме нее, могут быть использованы команды:

• *bshow <s:b>* — показать на блок-диаграмме блок, имеющий указанный индекс;

• *zclist* — вывод списка блоков, при выполнении которых может произойти «непредусмотренное» изменение полярности сигнала.

И, наконец, последняя команда — *status*, которая позволяет получить информацию о самом отладчике; она обеспечивает вывод заданных режимов отладки, а также перечень установленных контрольных точек.

ГЛАВА 5
SIMULINK + MATLAB

— Государь мой, что к чему привешено: хвост к собаке или собака к хвосту?

— Как, сударыня, приключится; ибо всякую собаку никому за хвост, как и за шею, приподнять невозбранно.

Из сочинений Козьмы Пруткова

Действительно, что же все-таки к чему «привешено»: MATLAB к SIMULINK или наоборот? Если, познакомившись с предыдущими главами, читатель решит, что MATLAB «привешен» к SIMULINK, то можно считать, что автор достиг поставленной цели. Но этого ему мало. Ему хочется показать, что SIMULINK — достаточно прочный «крючок», чтобы удержать на себе еще кое-что. Первый из этих «довесков» называется *Real-Time Workshop* (*Мастерская реального времени*); он обеспечивает создание на основе блок-диаграмм SIMULINK программного обеспечения, предназначенного для управления реальными аппаратными средствами в реальном масштабе времени.

Второй «довесок» — система событийного моделирования STATEFLOW. Это инструментальное средство снабжено собственным графическим интерфейсом, а созданные с его помощью модели могут включаться в виде блоков в состав блок-диаграмм SIMULINK.

Взаимодействие между названными инструментальными средствами можно схематично изобразить так (рис. 5.1):

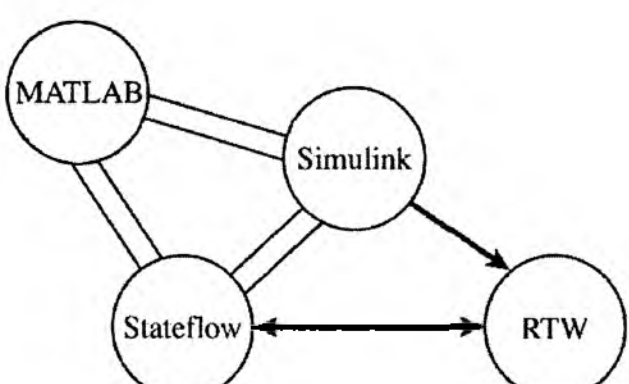


Рис. 5.1. Схема взаимодействия между основными компонентами пакета MATLAB

Так что же все-таки к чему «привешено»?...

Надеемся, материал последующих подразделов поможет читателю самостоятельно ответить на этот вопрос.

5.1. ПЛАНИРОВАНИЕ ЭКСПЕРИМЕНТОВ И ОБРАБОТКА РЕЗУЛЬТАТОВ МОДЕЛИРОВАНИЯ

Функции, обеспечивающие выполнение простейших процедур анализа данных, включены разработчиками MATLAB в состав его ядра (раздел *datafun*).

Некоторые дополнительные возможности в этом отношении предоставляет инструментальное приложение *Digital Signal Processing* (*Цифровая обработка сигналов*).

При его включении в конфигурацию MATLAB становится доступным соответствующий набор блоков, которые могут быть включены в диаграммы SIMULINK. Они позволяют, в частности, вычислять среднее значение и дисперсию элементов вектора.

Но все же в полной мере «ощутить власть» над данными позволяет только приложение *Statistics toolbox*.

Входящие в его состав М-функции разбиты на следующие категории:

- *Probability Distributions* (*Функции распределения вероятности*);
- *Parameter Estimation* (*Оценка параметров*);
- *Descriptive Statistics* (*Описательная статистика*);
- *Linear Models* (*Линейные модели*);
- *Nonlinear Models* (*Нелинейные модели*);
- *Hypothesis Tests* (*Проверка гипотез*);
- *Multivariate Statistics* (*Многомерная статистка*);
- *Statistical Plots* (*Статистические графики*);
- *Statistical Process Control* (*Управление статистическими процессами*);
- *Design of Experiments* (*Планирование экспериментов*).

Всего же данный раздел содержит более 200 функций, обеспечивающих проведение статистических экспериментов, а также обработку и анализ данных. Мы рассмотрим только те из них, с помощью которых может быть реализована технология подготовки и анализа результатов модельных экспериментов, описанная в первой части книги.

5.1.1. ПЛАНИРОВАНИЕ ЭКСПЕРИМЕНТОВ

В состав раздела *Design of Experiments* входят функции, обеспечивающие разработку всех трех основных видов стратегического плана эксперимента, рассмотренных в разделе 2.5:

- полного факторного эксперимента (ПФЭ)
- частичного факторного эксперимента (ЧФЭ)
- дробного факторного эксперимента (ДФЭ).

Для разработки первого из них предназначена функция с соответствующим названием — *fullfact*. В качестве ее параметров необходимо указать число уровней каждого фактора, участвующего в эксперименте.

Например, если факторы *A* и *B* имеют по 4 уровня, а фактор *C* имеет 2 уровня, то обращение к функции *fullfact* выглядит так:

fullfact ([4, 4, 2]).

Введя указанную конструкцию в командном окне MATLAB, можно получить список всех возможных комбинаций уровней факторов. Список выводится в командном окне, а также сохраняется в рабочей области под именем ***ans***. Он может быть использован в качестве подсказки либо в текущем сеансе работы с MATLAB, либо записан в отдельный MAT-файл для последующего применения.

Поскольку MATLAB использует переменную ans в качестве рабочей, целесообразно предварительно переписать план эксперимента в переменную с другим именем, например, *plan=ans*; после этого можно поочередно вызывать строки плана из рабочей области, используя операции индексирования. Для приведенного выше списка факторов содержимое седьмой строки можно считать в командное окно, введя команду

plan (7, 1:3).

Выполнив эксперимент для очередного сочетания факторов, можно удалить соответствующую строку плана, используя *Редактор/Отладчик* MATLAB.

Для формирования плана дробного фактора эксперимента используется функция *ff2n*. Параметром функции является число факторов. Результат ее выполнения представляет собой список возможных сочетаний уровней факторов; в списке значение «0» соответствует нижнему уровню фактора, а «1» — его верхнему уровню.

Например, команда *ff2n*(3) обеспечивает вывод в командное окно следующего списка:

```
ans=
     0     0     0
     0     0     1
     0     1     0
     0     1     1
     1     0     0
     1     0     1
     1     1     0
     1     1     1
```

При выборе метода планирования частичного факторного эксперимента разработчики MATLAB отдали предпочтение так называемым *D*-оптимальным планам. *D*-оптимальный план обладает той особенностью, что он обеспечивает мини-

мизацию взаимного влияния факторов, участвующих в эксперименте. В MATLAB предусмотрена возможность генерации нескольких вариантов *D*-оптимального плана, каждый из которых реализуется соответствующей М-функцией.

Мы рассмотрим способ формирования одного из видов ЧФЭ, приведенных в первой части книги. Речь идет о рандомизированном плане. И хотя он не гарантирует высокой информативности эксперимента, тем не менее во многих случаях такой вид планирования оказывается весьма полезным.

Для получения рандомизированного плана может быть использована функция *unidrnd*, представляющая собой генератор дискретной СВ, равномерно распределенной на интервале [1; *N*]. В общем случае она используется с тремя параметрами:

$$unidrnd(N, k, m),$$

где N — верхняя граница интервала распределения,

k, m — задают число строк и столбцов генерируемой случайной матрицы.

При генерации рандомизированного плана эти величины интерпретируются следующим образом:

N — число уровней факторов, участвующих в эксперименте (т. е. предполагается, что все факторы имеют равное число уровней, равное N);

k — выбранное пользователем число экспериментов (различных сочетаний уровней факторов);

m — число факторов.

Например, выполнение команды *unidrnd*(4, 5, 3) приводит к формированию такого плана эксперимента:

```
ans=
     4     4     3
     1     2     4
     3     1     4
     2     4     3
     4     2     1
```

5.1.2. ОБРАБОТКА И АНАЛИЗ РЕЗУЛЬТАТОВ МОДЕЛИРОВАНИЯ

Напомним, что в результате проведения серии модельных экспериментов исследователь получает совокупность статистических данных (выборку), на основании которых с помощью соответствующих статистических процедур могут быть получены ответы на следующие вопросы:

- зависит ли значение наблюдаемой переменной (показателя эффективности) от того или иного фактора, отображенного в модели (или группы факторов);
- можно ли описать существующую зависимость с помощью некоторого аналитического соотношения;

• можно ли считать наблюдаемую переменную случайной величиной, распределенной по некоторому закону;

• имеет ли место взаимное влияние двух или более факторов.

Кроме того, на этапе оценки качества разработанной имитационной модели анализ полученных результатов позволяет определить, в какой степени модель обладает требуемыми целевыми свойствами (адекватностью, устойчивостью, чувствительностью).

Ответ на каждый из этих вопросов может быть получен с помощью инструментов статистического анализа пакета MATLAB.

Подбор параметров распределения наблюдаемой переменной

Подбор (или оценка) параметров закона распределения наблюдаемой переменной, пожалуй, наиболее простая с математической точки зрения процедура. Особенно в том случае, если в распоряжении исследователя имеются средства визуализации полученных статистических данных. MATLAB такие средства предоставляет.

Для решения поставленной задачи в общем случае необходимо выполнить два действия:

• построить гистограмму относительных частот наблюдаемой переменной;

• подобрать вид функции распределения, в наибольшей степени отвечающий полученным экспериментальным данным.

Методика построения гистограммы «вручную» была подробно описана в разделе 2.6.

Для получения того же результата в MATLAB достаточно ввести две команды:

n = hist(y)

bar(n./length(y)).

Первая из них разбивает весь диапазон значений наблюдаемой переменной *y* на 10 равных интервалов и записывает в матрицу *n* число элементов, попавших в каждый интервал; при необходимости диапазон значений y можно разбить на произвольное число интервалов *m*, введя команду *n=hist(y, m)*.

С помощью второй команды вычисляется относительная частота попаданий в каждый интервал (*length(y)* — функция вычисления объема выборки *y*) и выводится графическое представление полученной гистограммы (рис. 5.2).

Для подбора одного из стандартных распределений, наиболее близкого к построенной гистограмме, целесообразно воспользоваться командой *disttool*.

При ее выполнении открывается диалоговое окно, обеспечивающее выбор и настройку параметров стандартных распределений (рис. 5.3).

Окно содержит следующие элементы:

• поле вывода графика выбранной функции распределения (*cdf*) или соответствующей плотности распределения вероятности (*pdf*); в поле отображается визир, обеспечивающий точное определение значений координат;

• выпадающее меню для выбора функции распределения;

• выпадающее меню для выбора типа функции (*cdf* ↔ *pdf*);

• ползунковые регуляторы для изменения значений параметров распределений (количество регуляторов и их обозначения изменяются в зависимости от вида распределения);

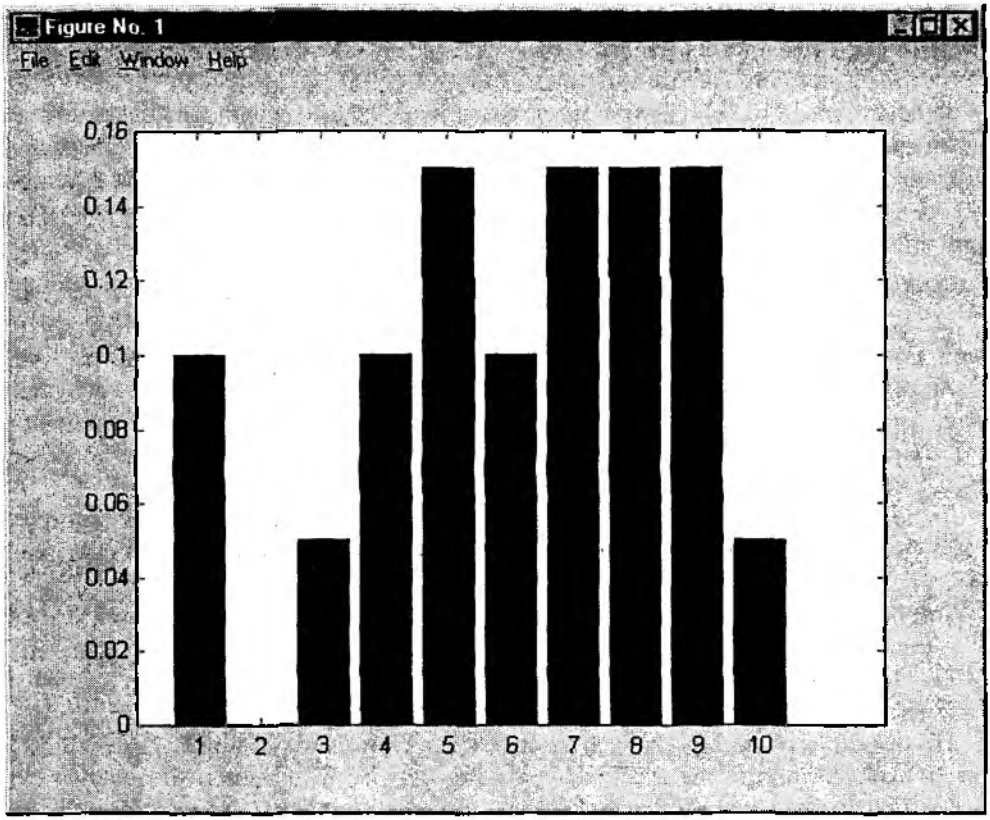


Рис. 5.2. Пример гистограммы относительных частот, построенной с помощью MATLAB

• окна ввода/отображения значений координат точек графика и параметров распределения.

Для отыскания подходящего вида распределения следует установить тип графика *pdf* и, используя собственные знания особенностей различных распределений, выбрать то из них, которое, на ваш взгляд, более других похоже на гистограмму относительных частот, построенную по экспериментальным данным. При этом необходимо учитывать, что вид графика существенно зависит от численных значений параметров функции распределения.

В первой части книги уже говорилось о том, что во многих случаях распределение наблюдаемой переменной оказывается близким к нормальному. Если такое предложение имеет место, то его можно проверить, используя дополнительные средства MATLAB: функция *normplot*(*y*) выводит пробит-график распределения случайной величины *y* (рис. 5.4).

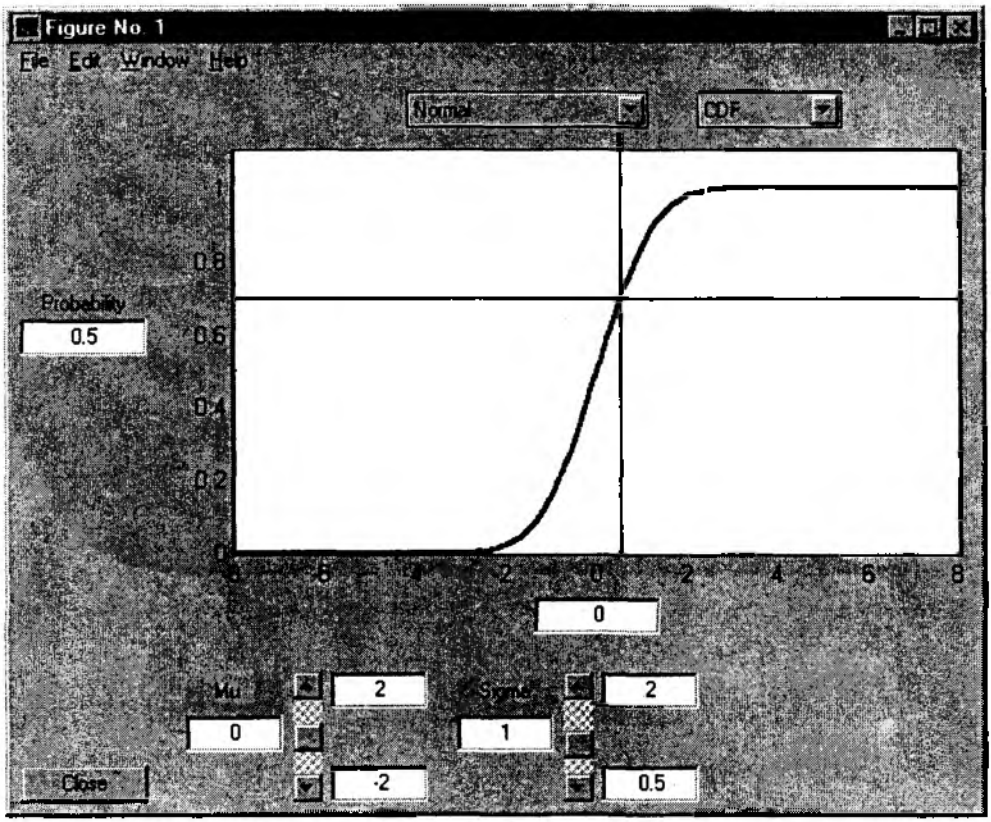


Рис. 5.3. Диалоговое окно, используемое при подборе параметров распределений

Чем лучше экспериментальные данные (обозначены на графике символами «+») согласуются с нормальным распределением, тем лучше они аппроксимируются прямой линией.

Проверка статистических гипотез

После визуального сравнения эмпирического распределения с теоретическими вы имеете полное право выдвинуть соответствующую статистическую гипотезу.

Функция *ztest* обеспечивает проверку гипотезы о том, что наблюдаемая переменная распределена по нормальному закону с математическим ожиданием, равным M; величина среднеквадратического отклонения СВ полагается известной.

Функция может вырабатывать одно из двух значений:

0 — гипотезу следует принять;

1 — гипотезу следует отвергнуть.

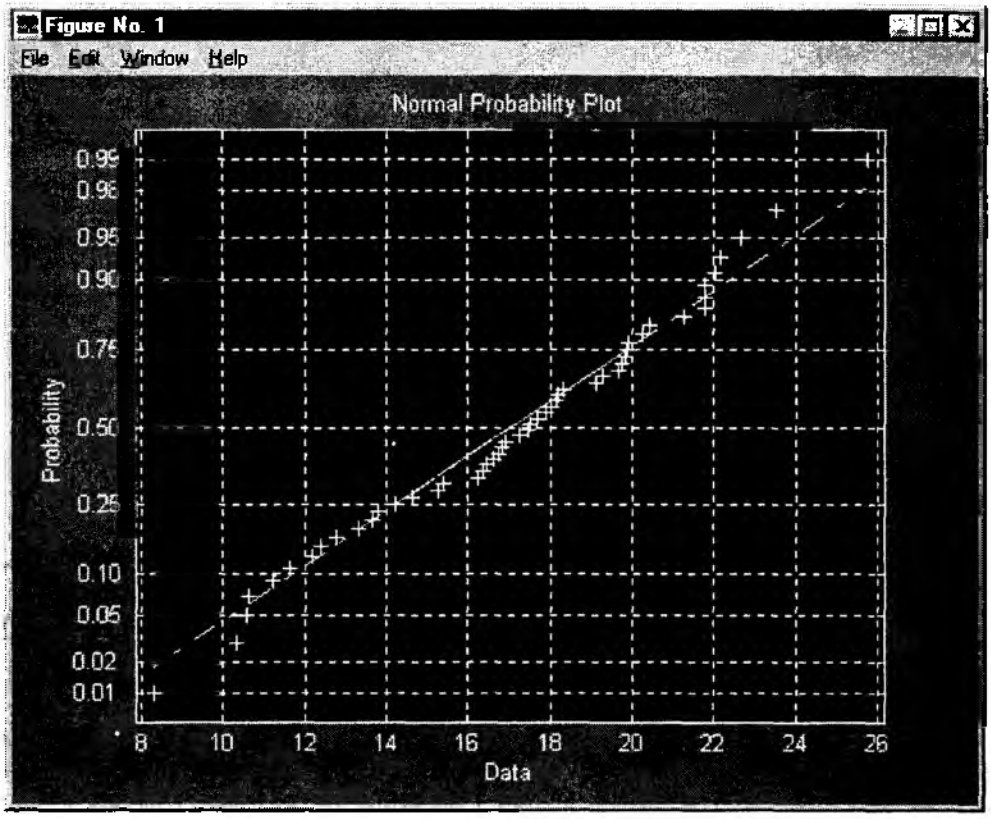


Рис. 5.4. Графический анализ экспериментальных данных

В качестве дополнительной информации функция *ztest* формирует величину *SIG*, которая отражает степень достоверности гипотезы H_0: чем меньше *SIG*, тем больше доверия гипотезе H_0.

Аналогичную задачу позволяет решить и функция *ttest*; в основе ее работы лежит расчет *t*-статистики; в качестве дополнительной информации функция *ttest* выдает *p*-значение и величину доверительного интервала. Понятие «*p*-значение» (*p*-value) широко используется в зарубежной литературе по математической статистике и, соответственно, нашло свое отражение практически во всех программных пакетах статистического анализа данных. *P*-значение представляет собой условную вероятность того, что используемый статистический критерий примет некоторое значение в предположении, что гипотеза H_0 — верна. Если *p*-значение оказывается меньше уровня значимости α, то гипотеза H_0 отвергается.

Пусть, например, в течение двух месяцев (марта и апреля) происходили колебания цен на видеокарты к компьютерам. И в том, и в другом месяце цены регист-

рировались 7 раз и были получены следующие данные:

$$mart = [16\ 22\ 14\ 25\ 18\ 25],$$
$$april = [18\ 25\ 14\ 28\ 20\ 23].$$

Требуется проверить гипотезу о том, что средняя цена в течение двух месяцев не изменилась. Для решения поставленной задачи вначале необходимо вычислить среднюю цену за март месяц:

$$c = mean(mart).$$

Величину c используем при обращении к функции ttest (уровень значимости α по умолчанию равен 0.05):

$$[H, SIG] = ttest(april, c).$$

В результате работы функции получаем:

H=0, SIG = 0,5815; это означает, что выдвинутую гипотезу следует принять, однако степень ее достоверности не очень велика.

Графическую интерпретацию статистических характеристик двух выборок (*mart* и *april*) можно получить с помощью функции *barplot*(y); параметр y представляет собой матрицу, столбцы которой содержат значения исследуемых выборок.

Средства дисперсионного анализа данных

В составе *Statistics Toolbox* имеются две функции, обеспечивающие проведение однофакторного и двухфакторного дисперсионного анализа: *anova1* и *anova2*. Название функций представляет собой общепринятое сокращение англоязычного термина *Analys of Variance* (*Дисперсионный анализ*).

Для проведения однофакторного дисперсионного анализа достаточно ввести команду *anova1*(y), где y — матрица, содержащая результаты наблюдений. Число столбцов матрицы должно быть равно числу уровней фактора, влияние которого оценивается. Соответственно, элементы каждого столбца представляют собой значения наблюдаемой переменной (показателя эффективности), полученные при данном уровне фактора.

Выполнение функции *anova1* приводит к открытию двух окон: первое из них содержит таблицу результатов дисперсионного анализа, второе — их графическую интерпретацию.

В качестве иллюстрации применения функции *anova1* вернемся к примеру с анализом цен на видеокарты. Предположим, что в качестве исследуемого фактора выступает среднемесячная температура воздуха t; в марте она имеет уровень «1», в апреле уровень «2»; тогда векторы mart и *april* можно рассматривать как значения цены, полученные для двух уровней фактора t. Гипотеза H_0 состоит в том, что цена не зависит от температуры воздуха t.

Объединив векторы *mart* и *april* в одну матрицу y и выполнив команду *anova1*(y), получим на экране два окна, упомянутые ранее (рис. 5.5).

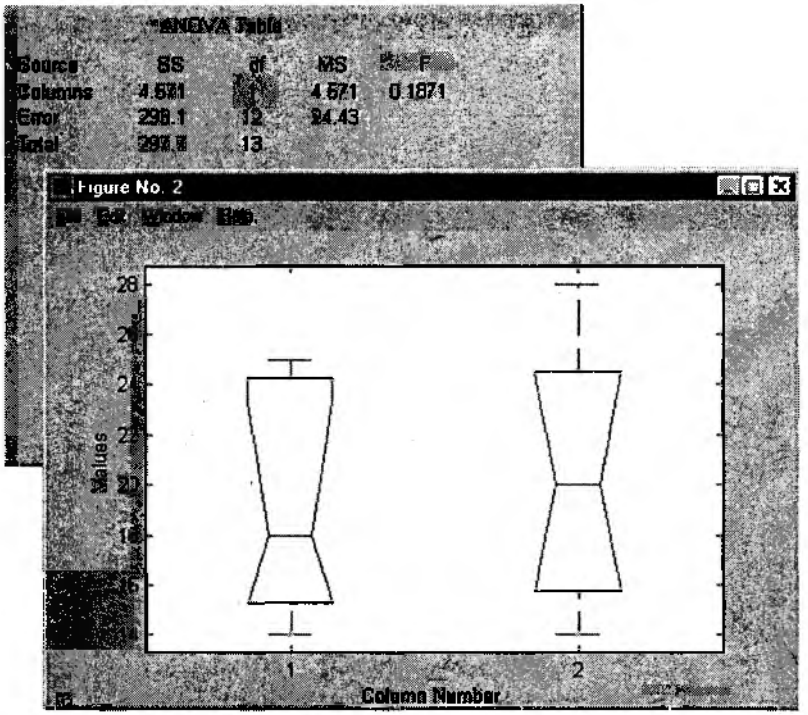


Рис. 5.5. Результат выполнения команды *anova1*

В таблице *ANOVA Table* представлены следующие величины:
SS — сумма квадратов;
df — число степеней свободы (*degrees of freedom*);
MS — средний квадрат (*SS*/*df*);
F — значение *F*-статистики.

На основании сравнения вычисленного значения *F*-статистики с критическим (взятым из справочника), можно сделать вывод о том, что гипотезу H_0 следует принять, т. е. стоимость видеокарт действительно не зависит от температуры воздуха.

Средства корреляционного и регрессионного анализа

Как вы, вероятно, помните, процедура корреляционного анализа начинается с построения диаграммы рассеяния, позволяющей отразить общий характер зависимости наблюдаемой переменной от исследуемого фактора.

При работе с MATLAB диаграмму рассеяния можно получить, использовав функцию *plot*(*x*, *y*). Чтобы выводимый график выглядел как совокупность не связанных между собой точек, следует задать дополнительный параметр *формат*, который представляет собой последовательность специальных символов, заключенных в апост-

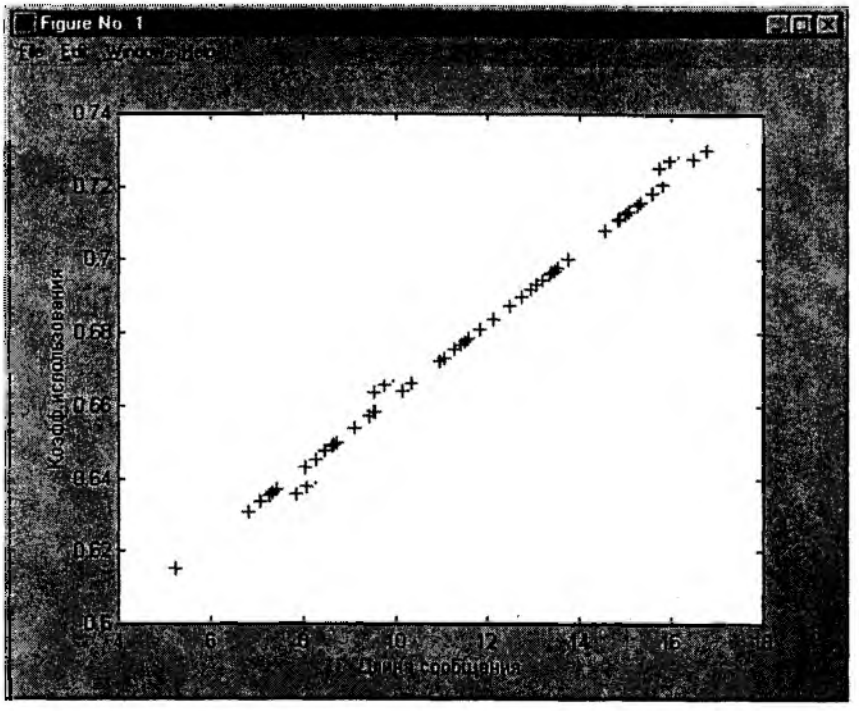


Рис. 5.6.Диаграмма рассеяния, полученная с помощью команды *plot*

рофы. Первый символ указывает цвет вывода точек, второй — способ их представления. Например, формат '*g*+' означает, что значения переменной *y* будут представлены на графике в виде символов «+» зеленого цвета (*g* — от английского *green*).

Реализацию процедуры корреляционного и регрессионного анализа средствами MATLAB рассмотрим на фоне неоднократно использовавшегося в Главе 4 примера моделирования работы жестких дисков (*HD*). Наблюдаемой переменной является коэффициент использования диска (*K*), а исследуемым фактором — длина сообщения (*l*).

Шаг 1 — построение диаграммы рассеяния:

plot (*l*, *K*, '*g*+'); результат показан на рис. 5.6.

Диаграмма рассеяния говорит о том, что между *K* и *l* существует линейная зависимость.

Шаг 2 — вычисление численного значения коэффициента корреляции:

corrcoef(*x*), где *x* — матрица, столбцы которой содержат значения величин *K* и *l*; для рассматриваемого примера коэффициент корреляции равен 1.

Шаг 3. Построение регрессионной модели; для выполнения этого шага удобнее всего воспользоваться командой *polytool*(*l*, *K*); она создает диалоговое графическое окно, с помощью которого можно оценить и при необходимости скорректировать полученную регрессионную модель.

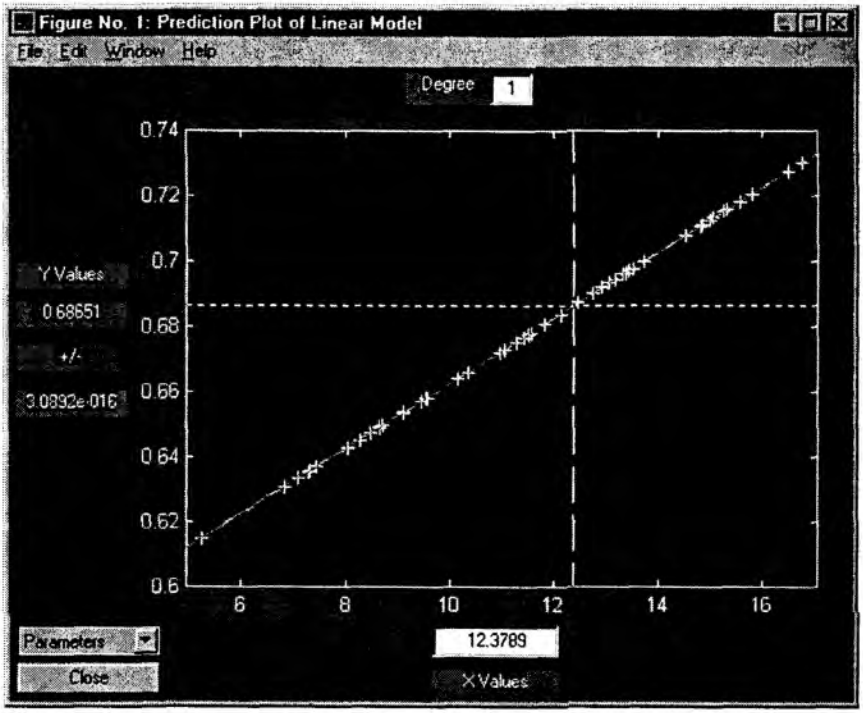


Рис. 5.7. Диалоговое окно для построения регрессионной модели

Окно содержит следующие элементы (рис. 5.7):

- поле отображения данных и графика регрессионной модели; оно содержит визир, с помощь которого для заданного значения независимой переменной может быть получено «прогнозируемое» значение зависимой переменной; если построенная регрессионная модель не очень точна, то здесь же дополнительно отображаются границы доверительного интервала, характеризующего точность модели (такая ситуация показана на рис. 5.8);
- поле выбора степени полинома для уравнения регрессии (*Degree*);
- поле вывода/установки значений независимой (*X Values*) и зависимой (*Y Values*) переменных;
- поле вывода численного значения ширины доверительного интервала;
- выпадающее меню *Export* для выбора величин, которые будут сохранены в рабочей области MATLAB:

Parameters — параметры регрессионной модели;
Prediction — «прогнозируемое» значение зависимой переменной;
Residuals — «остатки», характеризующий точность модели;
All — сохранение всех величин, отображаемых в окне.

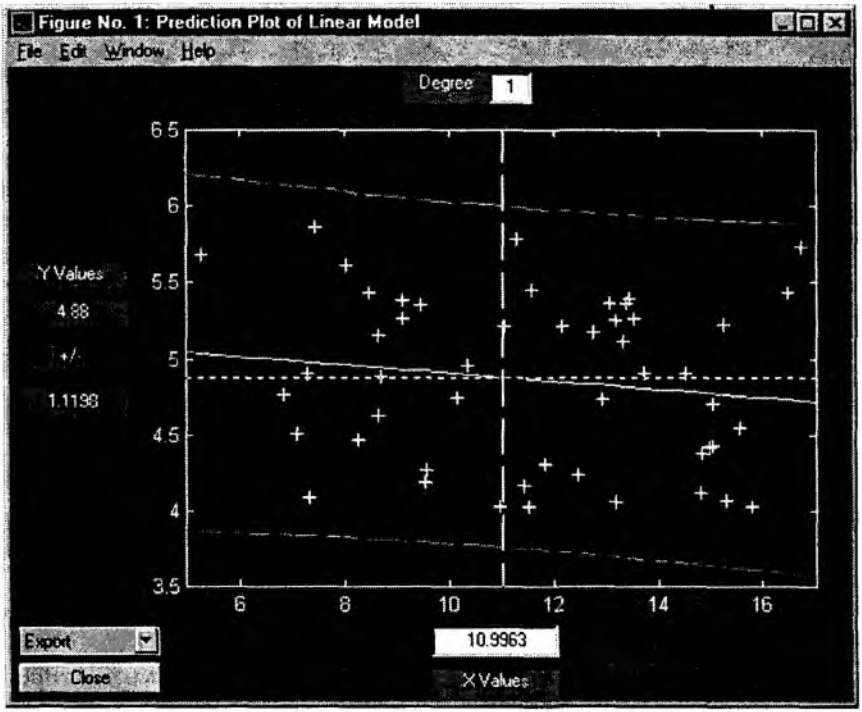


Рис. 5.8. Отображение в диалоговом окне доверительного интервала

5.1.3. СОЗДАНИЕ СЦЕНАРИЕВ АНАЛИЗА ДАННЫХ

Любая функция, входящая в состав Statistics Toolbox, может быть включена в состав S-модели посредством использования блока **MATLAB Fcn**. В качестве примера такого подхода можно привести применение генераторов случайных величин, распределенных по заданному закону.

Однако потребность во многих статистических функциях появляется после того, как моделирование завершено (или, по крайней мере, приостановлено). Причем при работе с одной и той же моделью, как правило, приходится выполнять одну и ту же последовательность статистических процедур. Другими словами, обработка и анализ результатов моделирования проводится по некоторому устоявшемуся «сценарию». В связи с этим уместно напомнить, что любая функция MATLAB представляет собой M-файл, то есть программу на языке MATLAB. Используя операторы этого языка и имеющиеся M-функции, можно описать сколь угодно сложную процедуру обработки данных и исполнять ее всякий раз, когда в этом возникает необходимость.

В зависимости от уровня сложности решаемой задачи, степени подготовленности в области программирования и, возможно, других факторов, пользо-

вателю предоставляется право выбрать один из двух вариантов построения М-файла:

- в виде М-сценария;
- в виде М-функции.

Между ними существует одно основное отличие:

М-сценарий не имеет входных параметров (аргументов); вместе с тем, входящие в него М-функции могут оперировать с данными, находящимися в рабочей области MATLAB.

Важным достоинством М-сценариев является то, что их разработка и использование практически не требует знаний и навыков в области программирования, в том числе и на языке MATLAB. Поэтому мы ограничимся описанием технологии создания именно этого вида М-файлов.

Основным средством разработки как М-сценариев, так и М-функций является *Редактор/Отладчик* MATLAB — *Editor/Debugger*, хотя для этих целей может быть использован любой текстовый редактор.

Несомненным достоинством *Редактора/Отладчика* является наличие весьма развитых (в том числе графических) инструментов отладки создаваемых М-файлов.

Для запуска *Редактора/Отладчика* можно использовать:

- команду *New→M-file* из раздела *File* командного окна MATLAB (если создается новый М-файл);
- команду *Open* из того же раздела (если вы хотите выполнить редактирование и отладку существующего М-файла);
- команду *edit<имя файла>*, введя ее непосредственно в командном окне MATLAB; если подлежащий редактированию файл не находится в активном директории, то следует указать полный путь доступа; при вводе команды *edit* без указания имени файла *Редактор/Отладчик* автоматически создает новый (пустой) файл.

Замечание: команда *Open...* позволяет открывать только файлы двух типов — М-файлы и mdl-файлы, с помощью же команды *edit* можно открывать файлы практически любого типа, в том числе текстовые, созданные, например, с помощью текстового редактора *Блокнот*.

Диалоговое окно *Редактора/Отладчика* содержит следующие элементы (рис. 5.9):

1. Меню пользователя, состоящее из разделов:
   - *Edit* — команды редактирования текста, а также команды поиска;
   - *View* — команды изменения формата окна *Редактора/Отладчика* и установки параметров его работы;
   - *Debug* — команды управления процессом отладки;
   - *Window* — команды размещения окон редактируемых файлов;
   - *Help* — вывод информации о *Редакторе/Отладчике* (номер версии, дата создания и т. д.);
2. Инструментальная панель, на которую выведены иконки основных команд из различных разделов меню;

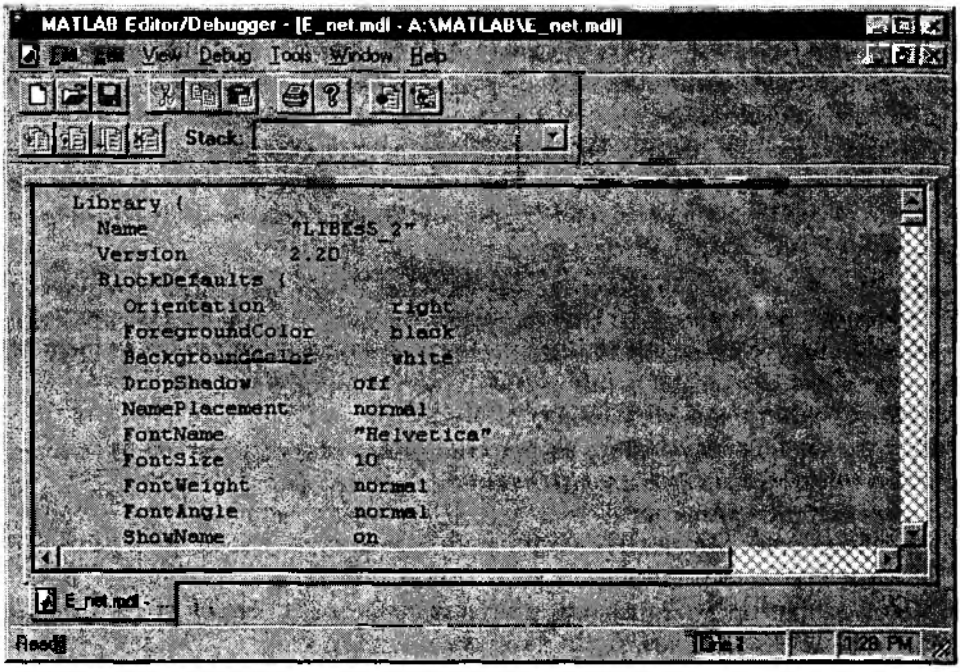


Рис. 5.9. Диалоговое окно *Редактора/Отладчика*

3. Окно выпадающего меню *Stack*, обеспечивающее выбор просматриваемой области памяти;

4. Рабочее поле *Редактора/Отладчика*, в котором отображается не только текст редактируемого файла, но и графические символы, позволяющие следить за процессом отладки;

5. Строка состояний, в которой выводятся: информация о состоянии *Редактора/Отладчика*; пояснения к командам меню; номер редактируемой строки; астрономическое время.

Поскольку круг наших интересов ограничивается только созданием М-сценариев, то в данном подразделе будут описаны лишь относящиеся к этой задаче функциональные возможности *Редактора/Отладчика*.

Прежде всего необходимо пояснить назначение основных команд управления процессом отладки, представленных на инструментальной панели (рис. 5.10):

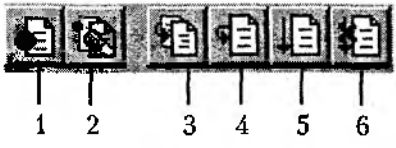


Рис. 5.10. Пиктограммы команд управления отладкой

1. *Set/Clear Breakpoint* — установить/убрать контрольную точку;
2. *Clear All Breakpoints* — убрать все установленные контрольные точки;
3. *Step In* — выполнить текущую строку М-файла, и если она содержит вызов функции, то загрузить ее;
4. *Single Step* — выполнить текущую строку М-файла;
5. *Continue* — продолжить выполнение М-файла, пока он не закончится, либо пока не встретится очередная контрольная точка;
6. *Quit Debug* — выйти из режима отладки.

Порядок использования перечисленных команд рассмотрим на примере создания и отладки сценария, обеспечивающего построение гистограммы относительных частот.

Напомним, что М-сценарий — это последовательность команд MATLAB; при необходимости она может быть дополнена текстовым комментарием (произвольный текст, начинающийся с символа '%').

Для улучшения визуального восприятия текста М-файла различные его компоненты имеют в окне *Редактора/Отладчика* разный цвет:

комментарий — зеленый;

ключевые слова MATLAB — синий;

остальные конструкции — черный.

Чтобы запретить дублирование исполняемых команд сценария в командном окне MATLAB, следует начать его с команды *echo off*.

В учебных целях гистограмму будем строить для вектора *y*, создание которого включено в этот же М-сценарий; дополнительно рассчитаем среднее значение элементов вектора *y*.

С учетом внесенных дополнений текст М-сценария будет выглядеть примерно так, как показано на рис. 5.11.

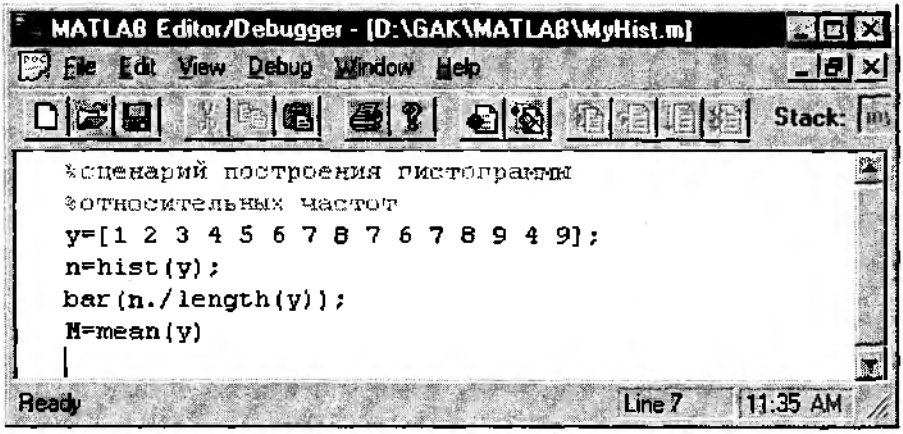


Рис. 5.11. Текст М-сценария

Сохраним его на диске под именем *MyHist*; с этого момента оно становится командой, которую можно вводить в командном окне MATLAB.

Отладку созданного сценария начнем с установки контрольных точек; пусть их будет две: в четвертой строке (*n=hist(y)*) и в шестой (*M=mean(y)*). Чтобы установить КТ, достаточно поместить курсор в требуемую строку и «нажать» соответствующую кнопку на инструментальной панели. КТ отображаются в виде крупных красных точек слева от выбранной строки (рис. 5.12).

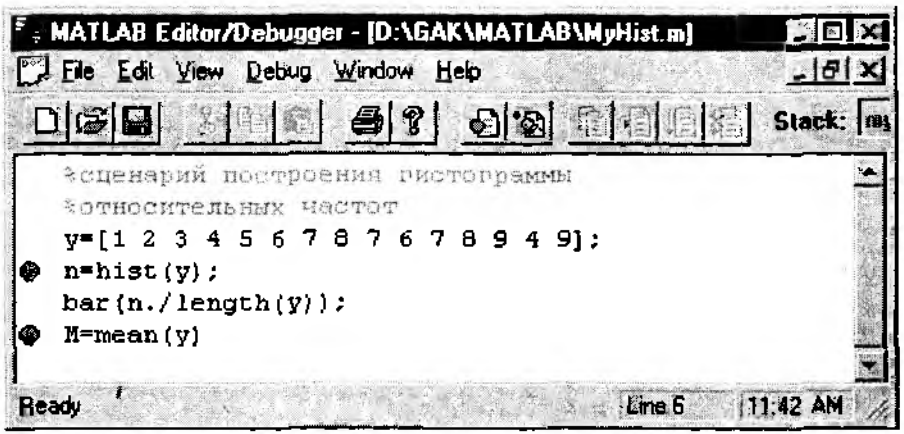


Рис. 5.12. Расстановка контрольных точек

Обратите внимание: пока доступны только две первые из шести команд управления отладкой.

Чтобы перейти в режим отладки, необходимо ввести в командном окне команду *MyHist*; в результате отладчик запускает М-сценарий, выполняет его до первой КТ и переходит в режим *Pause*; при этом в окне отладчика происходят следующие изменения (рис. 5.13):

- в строке, содержащей первую КТ, появляется желтая стрелка, указывающая место прерывания;
- становятся доступны все команды управления процессом отладки;
- становится доступным меню *Stack*; оно содержит два пункта:

1. *myhist* — имя рабочей области М-сценария;
2. *Base Workspace* — имя основной рабочей области MATLAB;

Замечание: для всех М-сценариев их локальная рабочая область совпадает с рабочей областью MATLAB (в отличие от М-функций).

В общем случае выбранная в меню *Stack* рабочая область определяет список переменных, которые доступны для просмотра. Чтобы получить значение интересующей переменной, необходимо выделить мышью ее имя в тексте М-файла и в разделе *View* выбрать команду *Evaluate Selection* (*Вычислить выбранное*). Значение соответствующей переменной выводится в командном окне MATLAB.

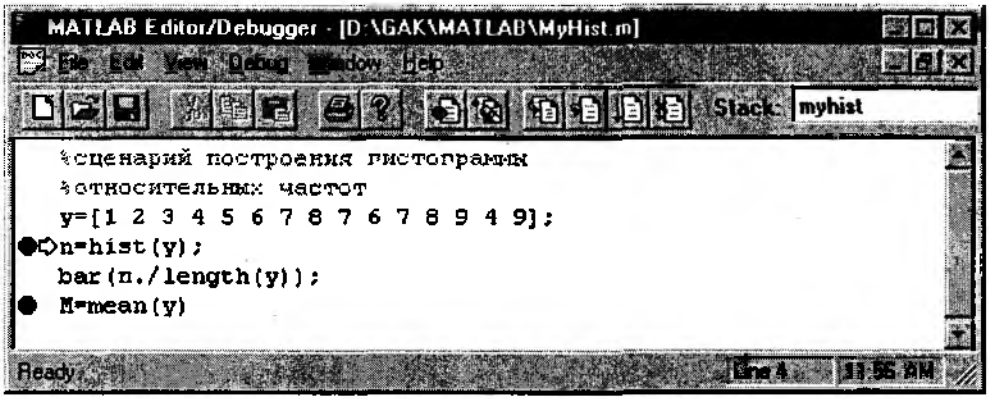


Рис. 5.13. Состояние диалогового окна в режиме отладки

Последовательный анализ значений переменных позволяет выявлять логические ошибки, допущенные при разработке М-файлов. Синтаксические же ошибки (неправильную запись конструкций языка MATLAB) отладчик обнаруживает сам еще до включения режима отладки. Попробуйте, например, добавить в 4-й строке лишнюю скобку и после этого установить в ней КТ. Отладчик заблокирует установку КТ и выдаст диагностическое сообщение.

В заключение рассмотрим работу отладчика в режиме *Step In*, который позволяет «заглянуть внутрь» М-функции, используемой в составленном сценарии.

Для этого в 5-ой строке заменим векторное деление (./) скалярным (/) и установим в ней КТ. Внесенное изменение должно привести к ошибке в работе функции *bar*. Чтобы проверить «бдительность» *Отладчика*, введите в командном окне команду *MyHist*, а после останова на КТ выберите опцию *Step In*. Отладчик не только обнаружит ошибку, но и укажет строку в функции *bar*, где она произошла (рис. 5.14).

После того, как М-сценарий (или М-функция) отлажен, он может быть сохранен не только как М-файл, но и в виде так называемого *псевдо-кода* (*P-code*) с помощью команды *pcode<имя файла>*. Р-файлы не подвергаются повторному синтаксическому анализу при их использовании, что позволяет повысить скорость их выполнения.

5.2. РАБОТА В МАСТЕРСКОЙ РЕАЛЬНОГО ВРЕМЕНИ

Мастерская Реального Времени (*Real-Time Workshop* — RTW) — это инструментальные средства разработки программного обеспечения для самого широкого класса аппаратных средств, работающих в реальном масштабе времени: датчиков, контроллеров, систем автоматического управления и регулирования и т. д.

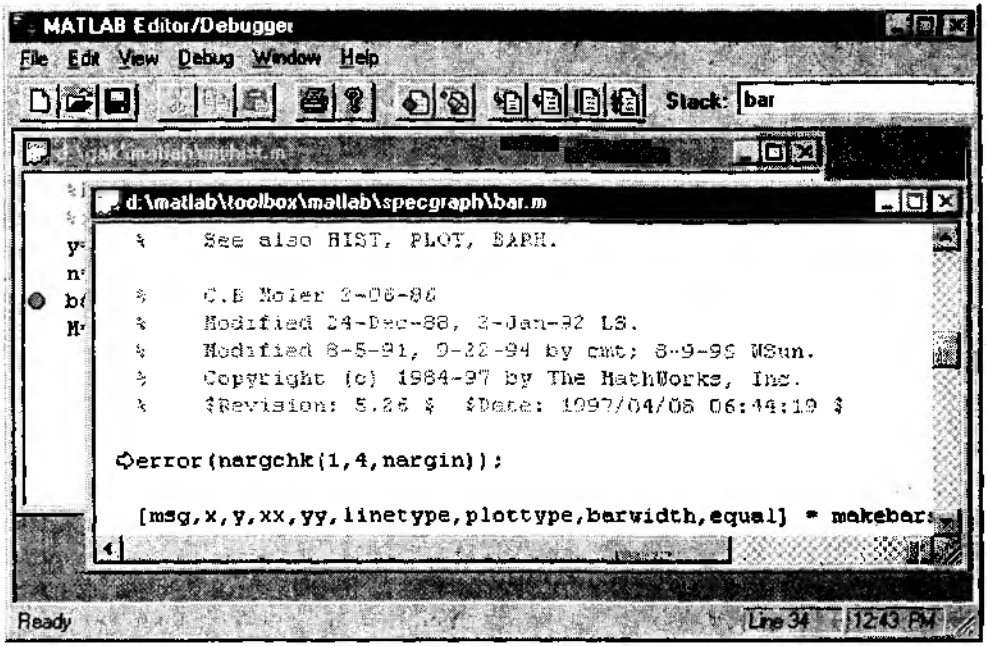


Рис. 5.14. Работа Отладчика в режиме *Step In*

Основными функциональными возможностями RTW являются:

- способность генерировать программный код из любой блок-диаграммы SIMULINK с фиксированным шагом изменения модельного времени (*Fixed-step*);
- использование расширяемой библиотеки драйверов устройств;
- автоматический и полностью настраиваемый процесс разработки программного обеспечения;
- объединение компонентов, работающих под управлением различных операционных систем;
- генерация программного кода практически для любого компилятора языка *C*, в том числе *Microsoft Visual C/C++*, *Watcom C/C++* и *UNIX*-версий.

Общая схема создания приложений с помощью RTW показана на приводимом ниже рисунке (рис. 5.15).

Поясним основные компоненты RTW и особенности их применения.

На первом шаге с помощью утилиты **Build** формируется описание блок-диаграммы в формате ASCII. Это описание используется как промежуточное представление блок-диаграммы и содержит, в частности, такие сведения, как значения параметров модели, размерность векторных сигналов, дискретность изменения модельного времени и т. п.; данная информация сохраняется в языково-независимом формате в файле с расширением .rtw.

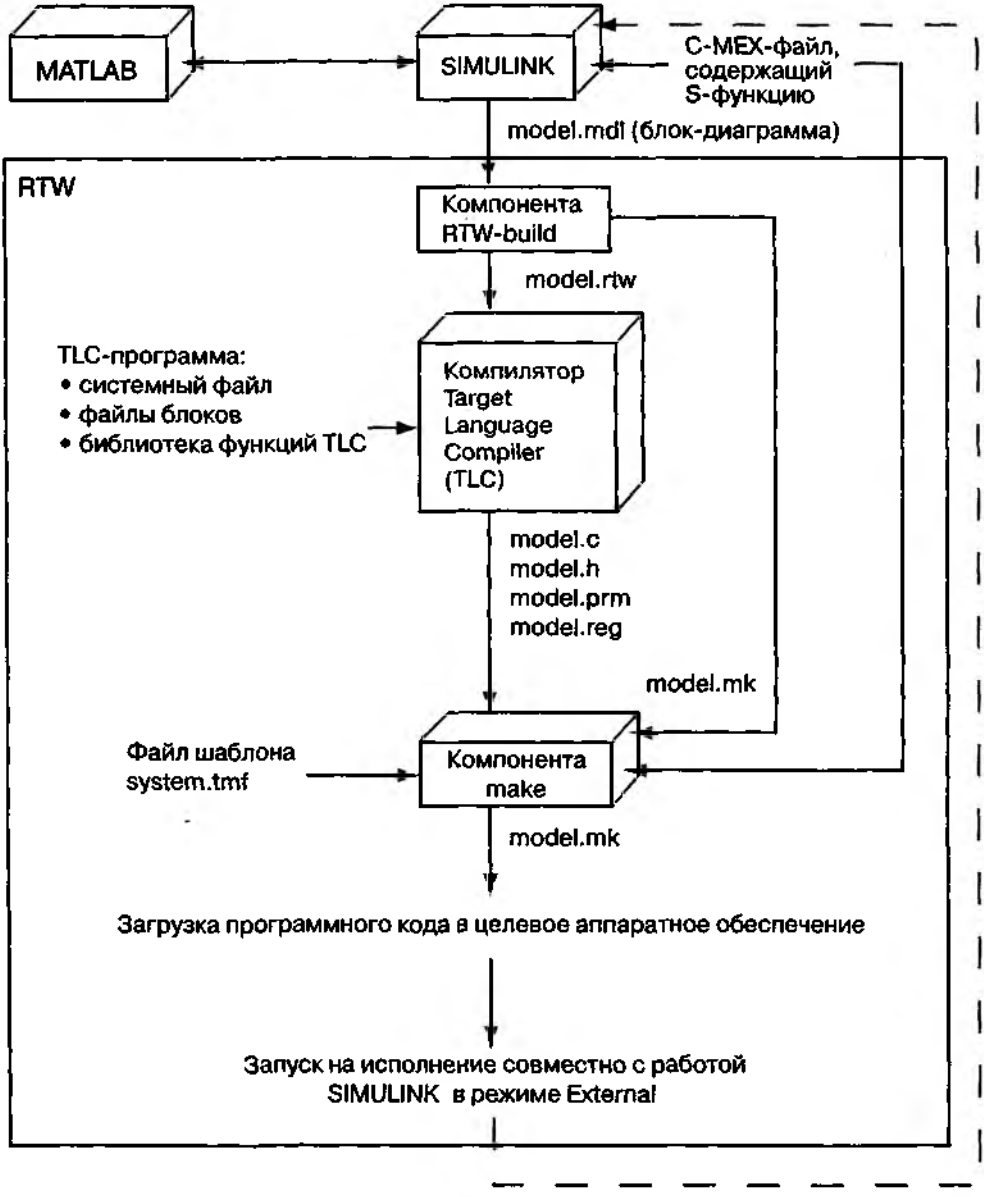


Рис.5.15. Общая схема создания программного кода с помощью RTW

На следующем шаге компилятор **TLC** (*компилятор объектного языка*) читает rtw-файл и с помощью *TLC*-программы, которая представляет собой набор «целевых файлов» (файлов с расширением .tlc), создает описание *S*-модели на языке *C*.

TLC-программа определяет, каким образом из промежуточного представления блок-диаграммы будет сформирован «целевой код». Она содержит:

- «точку входа» в программу, так называемый главный файл, который в данном случае именуется ***системным целевым файлом*** (*system target file*);
- набор библиотечных функций, которые TLC-программа использует при преобразовании rtw-файла в генерируемый код;
- набор целевых файлов блоков, которые определяют, каким образом каждый блок *S*-модели будет преобразован в целевой код.

Компилятор TLC создает на основе исходной *S*-модели 4 файла:

model.c — файл «точки входа» в программу;

model.h — «заголовочный» файл, который описывает связи между блоками модели;

model.prm — файл параметров блоков *S*-модели;

model.reg — файл описания параметров модельного времени *S*-модели.

Утилита **Make** используется для создания программного кода, выполняемого в реальном времени. Она производит компиляцию и связывание («линковку») перечисленных выше файлов и формирует так называемый исполняемый файл (с расширением .exe). Предварительно утилита **Make** создает на основе ***файла шаблона*** system.tmf объектный файл system.mk. Пользователь может управлять работой утилиты **Make**, модифицируя исходный шаблон по своему усмотрению. Имеющиеся в составе RTW файлы шаблонов перечислены в приведенной ниже таблице.

Таблица 5.1.

Шаблоны для создания программного кода

Тип создаваемого программного кода	Используемый компилятор	Файл шаблона
Исполняемый код реального времени для PC	MS Visual C/C++	grt_vc. tmf
Файл проекта для кода реального времени	MS *Visual C/C++*	grt_msvc. tmf
Исполняемый код реального времени для PC	Watcom C	grt_watc.tmf
Программный код реального времени для UNIX	Любой стандартизованный (ANSI) С-компилятор, поддерживаемый вашей UNIX-системой	grt_unix.tmf

На данном этапе к генерируемому коду добавляются коды *S*-функций, если они содержатся в блок-диаграмме SIMULINK.

Вы уже встречались с понятием *S*-функции при знакомстве с библиотекой SIMULINK: в разделе *Nonlinear* имеется блок, который так и называется — **S-function**. Сейчас уместно дать более подробное пояснение этого понятия.

S-функция — это достаточно самостоятельная программа, описывающая поведение некоторой системы или ее отдельной компоненты. *S*-функция может быть написана на языках *C*, *Фортран* или MATLAB. Если *S*-функция написана на языках *C* или *Фортран*, то она предварительно компилируется и преобразуется в MEX-файл. Эти действия выполняет специальная утилита mex, входящая в состав интерфейса прикладных программ (*Application Program Interface* — API). Созданный MEX-файл динамически связывается с другими компонентами модели, когда это необходимо.

Форма *S*-функции является очень общей и позволяет описывать непрерывные, дискретные и гибридные системы. Благодаря этому практически любая *S*-модель может быть представлена в виде *S*-функции.

При создании программного кода с помощью RTW пользователь может выбрать один из двух способов включения в него *S*-функции:

- непосредственное внедрение; в этом случае *S*-функции присваивается тип *inlining*;
- с помощью механизма API (используется по умолчанию).

Выбор метода выполняется с помощью соответствующих опций настройки RTW, которые будут рассмотрены немного позже.

Но вернемся к процессу генерации RTW-кода. Созданный утилитой *Make* исполняемый программный код может быть загружен в те аппаратные средства, для которых он разрабатывался; при запуске SIMULINK в режиме *External* возможна окончательная настройка параметров созданного программного обеспечения непосредственно в процессе его работы.

Такова общая схема использования Мастерской Реального Времени.

Теперь обсудим некоторые технические детали, без знания которых работа с RTW может вызвать затруднения.

Прежде всего следует, видимо, уточнить, что для работы с RTW данная компонента должна быть включена в рабочую конфигурацию пакета MATLAB.

Если это выполнено, то во всех mdl-файлах в меню пользователя добавляется еще один раздел — *Tools*. Он содержит две команды (рис. 5.16):

RTW Build — запуск утилиты RTW *Build*;

RTW Options — установка опций для работы с RTW; при выборе этой команды становятся доступны два диалоговых окна — *RTW* и *RTW External*, содержащие необходимые элементы настройки параметров RTW. Указанные окна реализованы в виде дополнительных вкладок диалогового окна *Simulations parameters* (рис. 5.17) и, следовательно, могут быть открыты также из раздела *Simulation* по команде *Parameters*.

Такое совмещение выполнено не случайно, поскольку, приступая к работе с RTW, необходимо еще раз проверить корректность установки параметров моделирования:

- начальное и конечное значения времени для интервала моделирования;
- способ изменения модельного времени (должен быть установлен *Fixed-step*);
- параметры сохранения результатов моделирования в рабочей области MATLAB.

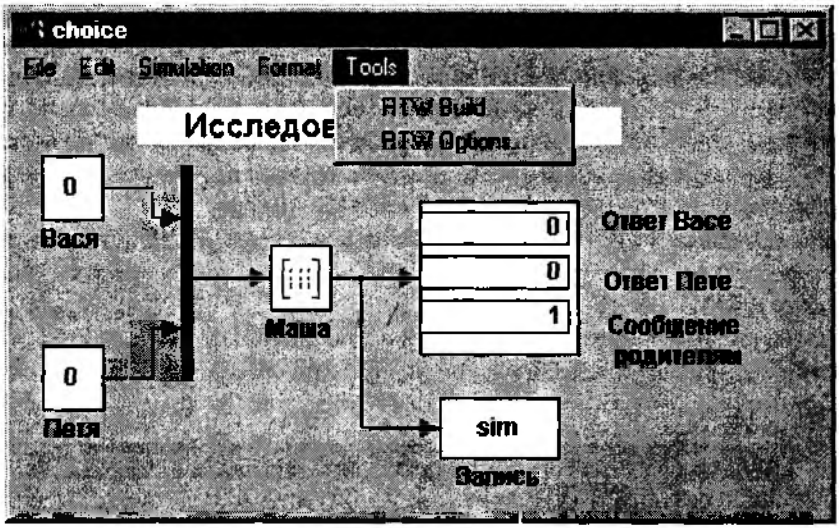


Рис. 5.16. Состав меню пользователя при работе с RTW

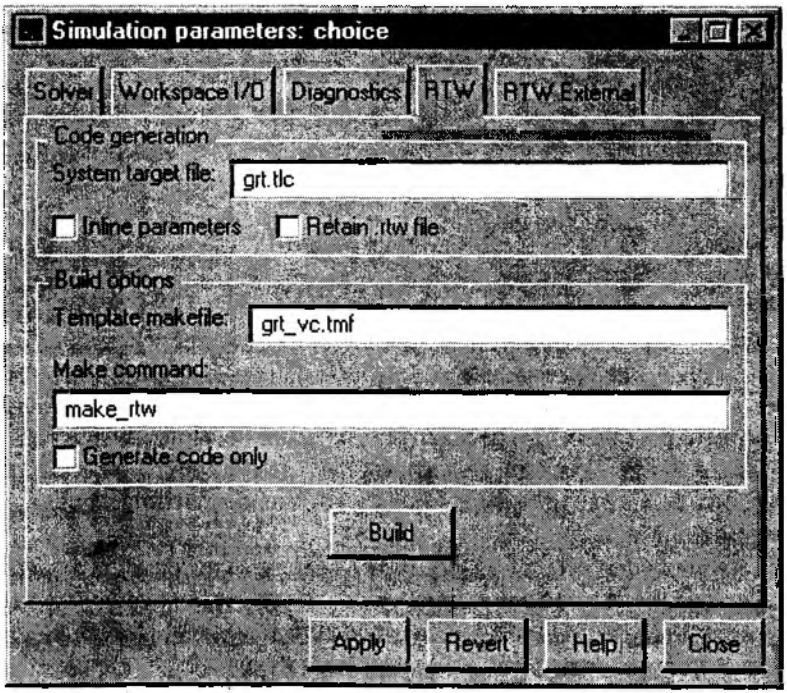


Рис. 5.17. Вид диалогового окна *Simulation Parameters* при использовании RTW

Последний пункт требует некоторого пояснения. Дело в том, что при исполнении программного кода, созданного с помощью RTW, регистрация данных выполняется так же, как при исполнении обычной *S*-модели. При этом создается стандартный MAT-файл, содержащий все переменные рабочей области MATLAB. Единственное отличие состоит в том, что к именам переменных добавляется префикс rt_. Это позволяет, кроме всего прочего, сравнить результаты моделирования с результатами, полученными с помощью RTW-кода.

При включении RTW в рабочую конфигурацию MATLAB происходит также изменение состава команд раздела *Simulation*: в нем появляются опции выбора режима работы SIMULINK (рис. 5.18):

Normal — обычный режим (установлен по умолчанию);

External — «внешний», используемый при совместной работе SIMULINK с RTW-кодом.

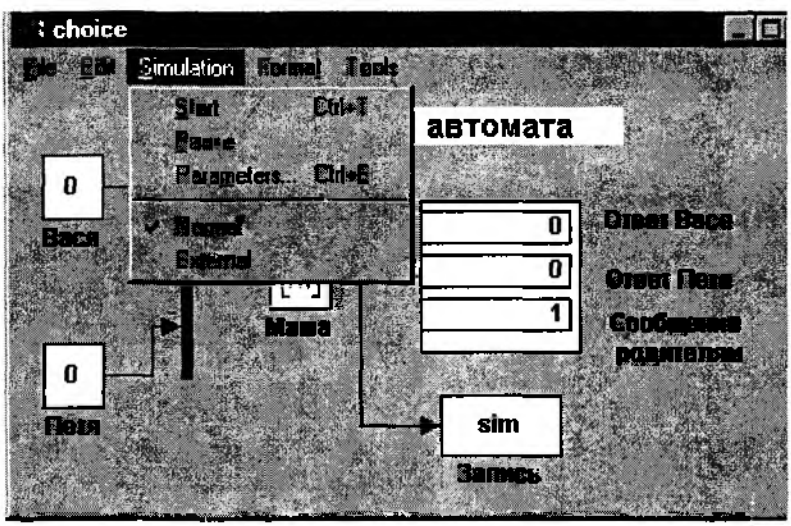


Рис. 5.18. Состав раздела *Simulation* при работе с RTW

Теперь вернемся к рис. 5.17 и рассмотрим назначение и порядок установки параметров работы RTW. Для определенности будем считать, что необходимо создать RTW-код для *S*-модели *Выбор* (*Choice*), описывающей поведение конечного автомата по имени Маша.

Окно установки параметров RTW содержит следующие диалоговые элементы:

• строку редактирования *System target file*, в которой выводится имя tlc-файла, используемого при создании RTW-кода (по умолчанию это файл grt.tlc); здесь же могут вводиться опции настройки работы компилятора TLC;

• флажок *Inline parameters*, определяющий способ представления параметров *S*-модели в создаваемом RTW-коде: если он установлен, то вместо имен переменных

непосредственно подставляются их значения; кроме того, в данном режиме используется единая величина шага модельного времени (параметра Sample time) во всех блоках *S*-модели, что повышает корректность работы создаваемого RTW-кода;

• флажок *Retain .rtw file* (*Сохранить rtw-файл*), позволяющий сохранить на диске (в активном директории) промежуточное описание *S*-модели в виде одноименного rtw-файла;

Замечание: для рассматриваемого примера целесообразно установить флажок *Retain .rtw file*, чтобы затем познакомиться с содержимым этого файла;

• строка редактирования *Template make file*, которая обеспечивает возможность выбора требуемого шаблона для создания *make*-файла;

• строка редактирования *Make command*, позволяющая вводить команду установки параметров используемого шаблона; например, команда

```
make_rtw OPTS='-DMULTITASKING'
```

разрешает использование многозадачности при формировании исполняемого кода;

• флажок *Generate code only* позволяет отказаться от создания объектных и исполняемых файлов: если данный флажок установлен, то работа RTW завершается генерацией файлов на языке *C* (для рассматриваемой модели это файлы choice.c, choice.h, choice.reg и choice.prm);

• кнопка *Build* обеспечивает запуск процесса создания RTW-кода в соответствии с установленными параметрами (чтобы они вступили в силу, предварительно должна быть «нажата» кнопка *Apply*); если параметры работы RTW были установлены ранее, или используются по умолчанию, то запуск RTW может быть произведен по команде *RTW Build* из раздела меню пользователя *Tools*.

Состояние процесса формирование RTW-кода отображается в командном окне MATLAB в виде текстовых сообщений. Их перечень для модели *Choice* показан на рис. 5.19.

```
### Starting RTW build procedure for model: choice
### Invoking Target Language Compiler on choice.rtw
### Loading TMW TLC function libraries
### Initial pass through model to cache user defined code
### Creating source file choice.c
### Creating part 1 of registration file choice.reg
### Creating parameter file choice.prm
### Creating model header file choice.h
### Creating part 2 of registration file choice.reg
### TLC code generation complete.
### Creating project marker file: rtw_proj.tmw
### Creating choice.mk from D:\MATLAB\rtw\c\grt\grt_vc.tmf
### Creating choice.exe
### Successful completion of RTW build procedure for model: choice
```

Рис. 5.19. Сообщения, выводимые в командном окне MATLAB при успешном создании RTW-кода

Замечание: с целью обеспечения корректной работы утилиты *Make* с файлами шаблонов и используемым компилятором языка *C*, необходимо убедиться в том, что:

1) определена переменная окружения *MsDevDir*;

2) утилита *nmake* находится в активном директории.

После того, как будет сформирован исполняемый (.exe) файл исходной модели, он может быть запущен на исполнение из командного окна MATLAB с помощью команды *!choice*; символ «!» означает обращение к операционной системе, установленной на вашем компьютере (другими словами, на время выполнения программы *choice* MATLAB передает управление операционной системе).

По завершении работы программы *choice.exe* ее результаты можно загрузить в рабочую область MATLAB из одноименного MAT-файла по команде *Load choice*.

Использование SIMULINK в режиме *External*

Данный режим обеспечивает возможность динамической настройки параметров сгенерированного RTW-кода в процессе его исполнения.

При этом наблюдение за исполнением RTW-кода и регистрация результатов осуществляется на основе механизма API, предоставляемого операционной системой компьютера.

Таким образом, в режиме *External* на базе MATLAB создается как бы новая среда разработки программного кода, получившая название ***среда быстрого макетирования.*** При работе в этой среде пользователь имеет возможность модифицировать параметры блоков исходной *S*-модели, изменять перечень входных и выходных параметров модели без перекомпиляции RTW-кода.

Взаимодействие программ в этой среде разработки основано на использовании клиент-серверной модели вычислений, где SIMULINK – клиент, который посылает запрос в сервер (внешней программе), чтобы установить новые значения того или иного параметра модели. Структурная организация среды быстрого макетирования делает ее расширяемой системой и позволяет использовать различные протоколы взаимодействия между ее компонентами.

Суть работы SIMULINK в режиме *External* состоит в следующем.

После запуска RTW-кода SIMULINK устанавливает начальные значения параметров блоков, переходит в состояние ожидания и остается в нем до тех пор, пока вы не измените параметры блок-диаграммы. После этого новые значения параметров пересылаются в исполняемую внешнюю программу. Значения параметров передаются как аргументы MEX-файла, который динамически вызывается SIMULINK.

Замечание: при работе в режиме *External* флажок *Inline parameters* на вкладке *RTW* обязательно должен быть установлен.

В общем случае при работе в режиме *External* нельзя изменять параметры, определяющие структуру *S*-модели, в том числе:

- число состояний, входов и выходов любого блока;
- величину шага модельного времени;

- алгоритм интегрирования (расчета состояний) для непрерывных систем;
- имена модели или любого ее блока;
- параметры блоков **Fcn**.

Для перевода SIMULINK в режим *External* следует выбрать соответствующую опцию в разделе *Simulation*. Установка параметров работы RTW в этом режиме производится в диалоговом окне *Simulation parameters* на вкладке *RTW External*.

Вкладка содержит два основных поля (рис. 5.20):

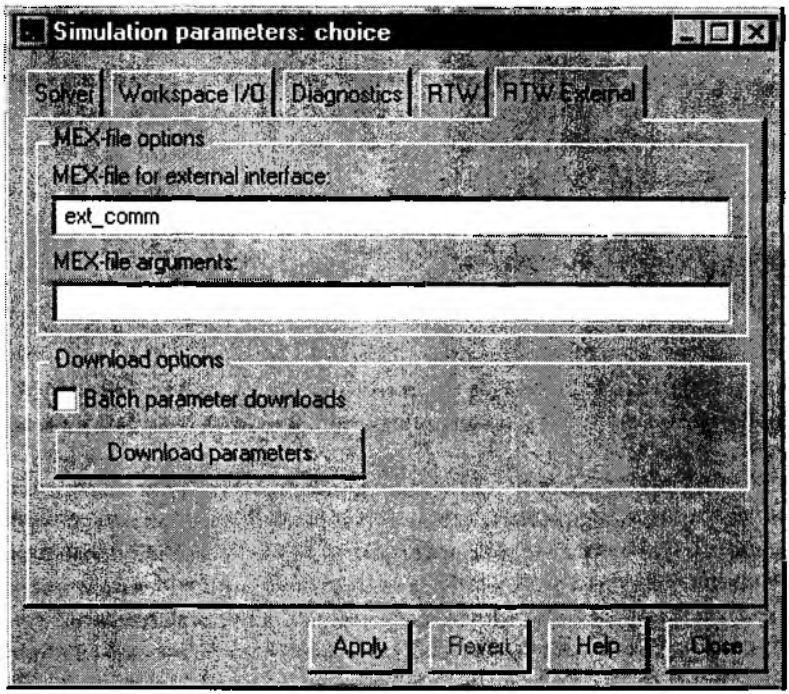


Рис. 5.20. Вкладка *RTW External*

Mex file options — установка параметров MEX-файла, используемого для передачи параметров во внешнюю программу;

Download options — опции загрузки изменяемых параметров блок-диаграммы во внешнюю программу.

Первое из них, в свою очередь, состоит из двух строк редактирования:

MEX-file for external interface — имя используемого MEX-файла;

MEX-file arguments — аргументы MEX-файла, определяющие порядок его использования (например, номер порта сервера, сетевое имя целевого оборудования и т. д.).

Поле опций загрузки параметров содержит флажок *Batch parameter downloads* (*Пакетная загрузка параметров*), который определяет способ передачи новых зна-

чений параметров во внешнюю программу: если флажок установлен, то передача производится только после «нажатия» кнопки *Downloads parameter*, расположенной ниже; в противном случае новые значения параметров передаются во внешнюю программу сразу после внесения изменений в блок-диаграмму (т. е. по одному).

Более подробное описание работы SIMULINK в режиме *External* предполагает знание читателем сетевых протоколов (в частности, TCP/IP) и механизма их использования в локальных вычислительных сетях. В связи с этим технические детали реализации режима *External* в данной книге не рассматриваются.

Завершая описание технологии использования *Мастерской Реального Времени*, необходимо отметить, что последняя версия данного инструментального средства содержит компоненту *RTW Ada Extension*, которая обеспечивает создание из блок-диаграмм RTW-кода на языке Ada. При этом порядок работы RTW аналогичен рассмотренному выше.

5.3. ВЗАИМОДЕЙСТВИЕ С ДРУГИМИ ИНСТРУМЕНТАЛЬНЫМИ ПРИЛОЖЕНИЯМИ MATLAB

В данном подразделе основное внимание будет уделено пакету STATEFLOW как наиболее «близкому по духу» рассмотренному в книге подходу к моделированию систем.

STATEFLOW представляет собой инструмент графического конструирования сценариев поведения различных систем. Используя предоставляемые интерактивные средства, разработчик может описывать поведение исследуемой системы как цепочку правил «если — то». Теоретической основой построения и исследования моделей в среде STATEFLOW является теория конечных автоматов.

Модели, разрабатываемые в STATEFLOW, могут использоваться как самостоятельно, так и в интеграции с другими компонентами MATLAB, в первую очередь — как особые подсистемы в блок-диаграммах SIMULINK. На основе STATEFLOW-моделей (*SF*-моделей) может быть сгенерирован исполняемый программный код.

При создании STATEFLOW разработчики ориентировались на возможность его применения в следующих предметных областях:

- планирование воздушного и автомобильного движения;
- проектирование периферийного компьютерного оборудования и программируемых логических контроллеров;
- разработка графического пользовательского интерфейса программного обеспечения.

STATEFLOW содержит следующие основные компоненты:

- графический редактор диаграмм *SF*-моделей;
- средства анализа *SF*-моделей (STATEFLOW *Explorer*);

• навигатор (STATEFLOW *Finder*), обеспечивающий поиск в *SF*-моделей требуемых объектов (событий, состояний, данных и т. д.);

• генератор кода, который преобразует *SF*-модель в промежуточный код, используемый для работы с RTW;

• отладчик *SF*-моделей.

При включении STATEFLOW в рабочую конфигурацию MATLAB раздел библиотеки SIMULINK *Blocksets&Toolboxes* дополняется подразделом Stateflow, содержащим три компоненты (рис. 5.21):

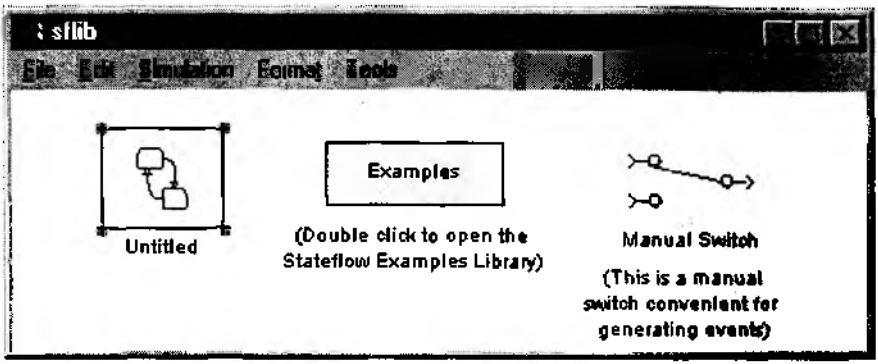


Рис. 5.21. Раздел библиотеки Stateflow

Untitled — окно графического редактора *SF*-моделей, которое в то же время является «заготовкой» для создания новой *SF*-модели;

Examples — примеры использования *SF*-моделей в составе блок-диаграмм SIMULINK;

Manual Switch (*Ручной переключатель*) — элемент блок-диаграммы, который может быть использован в качестве генератора (источника) событий для *SF*-моделей.

Технология совместного использования SIMULINK и STATEFLOW может быть весьма гибкой и определяется в первую очередь спецификой решаемой задачи.

В некоторых случаях удобнее сначала разработать (или скорректировать имеющуюся) *SF*-модель и затем на ее основе построить блок-диаграмму; иногда наоборот сначала создается блок-диаграмма, и только после этого один или несколько ее блоков заменяются соответствующими *SF*-моделями.

Второй подход удобнее в тех случаях, когда в состав блок-диаграммы входят блоки **Combinatorial logic**, отражающие, как вы помните, логику работы конечного автомата в форме таблицы истинности.

Если требуется более детальное моделирование поведения конечного автомата, то соответствующий блок **Combinatorial logic** заменяется *SF*-моделью, описывающей процесс смены состояний автомата. Такая технология обеспечи-

вает разработку модели по принципу *нисходящего проектирования*: сначала определяется общая структура модели, а затем отдельные ее компоненты уточняются и детализируются.

В качестве примера реализации этого принципа можно еще раз вернуться к модели *Выбор*, подробно описанной в 4-й главе. Определив первоначальную структуру модели, мы можем уточнить логику поведения Маши, описав ее (логику) с помощью *SF*-модели.

С этой целью заменим блок **Combinatorial logic** блоком **Untitled**, взятым из библиотеки STATEFLOW (назовем его *SF*-блок). В исходном состоянии *SF*-блок не имеет входных и выходных портов (рис. 5.22).

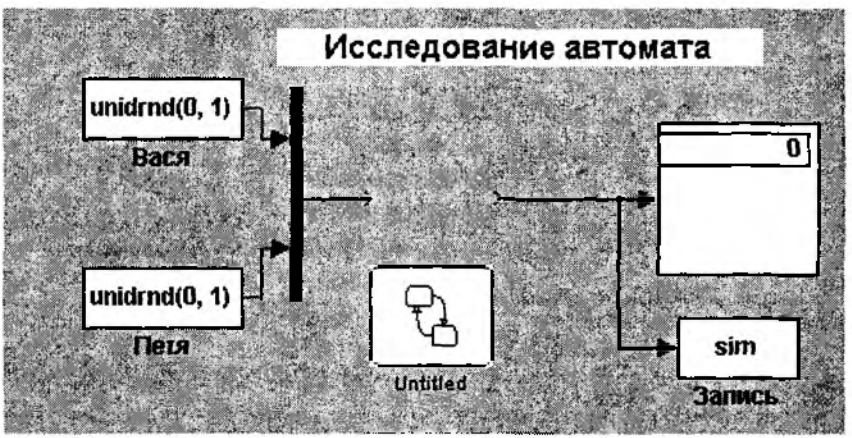


Рис. 5.22. Исходное состояние блок-диаграммы при включении в нее блока STATEFLOW

Поэтому сначала необходимо описать его интерфейс с блок-диаграммой SIMULINK. Описание заключается в указании входных и выходных портов, а также типа управления *SF*-блоком. Он может быть связан с другими блоками *S*-модели как по информации, так и по управлению. Связь по управлению осуществляется посредством обмена событиями, а для передачи данных используются информационные порты. Описание интерфейса *SF*-блока выполняется с помощью графического редактора STATEFLOW. Чтобы открыть окно, достаточно дважды щелкнуть ЛКМ на иконке *SF*-блока. Окно содержит меню, рабочее поле и набор инструментов для создания *SF*-модели, расположенный в левой части окна (рис. 5.23).

Для описания интерфейса с SIMULINK необходимо открыть раздел меню *Add*. Если создается информационный порт, то необходимо выбрать опцию *Data*, а в дополнительном меню – требуемый тип порта (рис. 5.24).

Для указания параметров создаваемого порта открывается соответствующее диалоговое окно (рис. 5.25). К таким параметрам, в частности, относится имя пе-

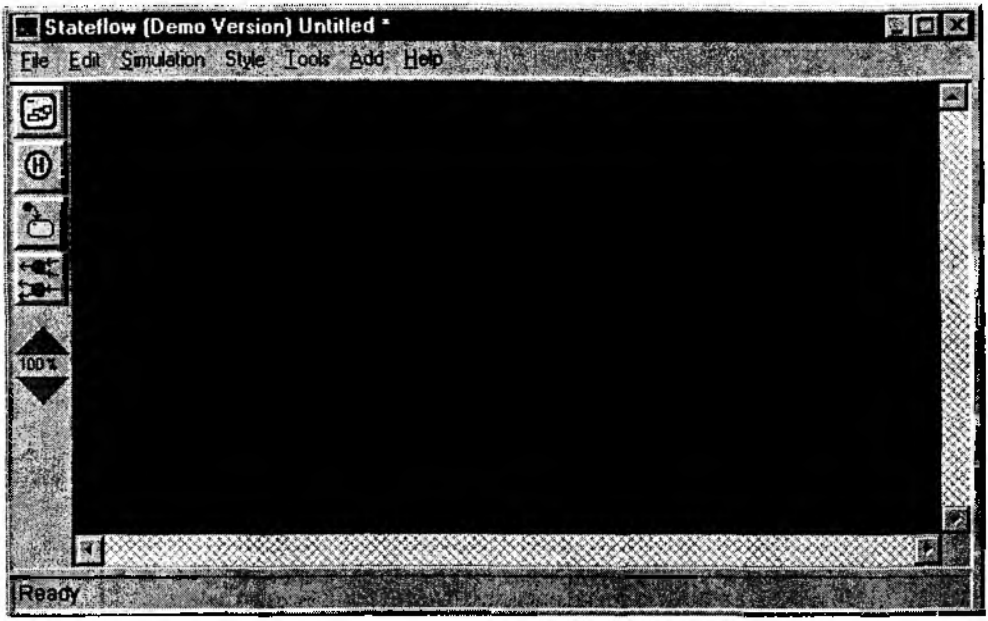


Рис. 5.23. Окно графического редактора *SF*-моделей

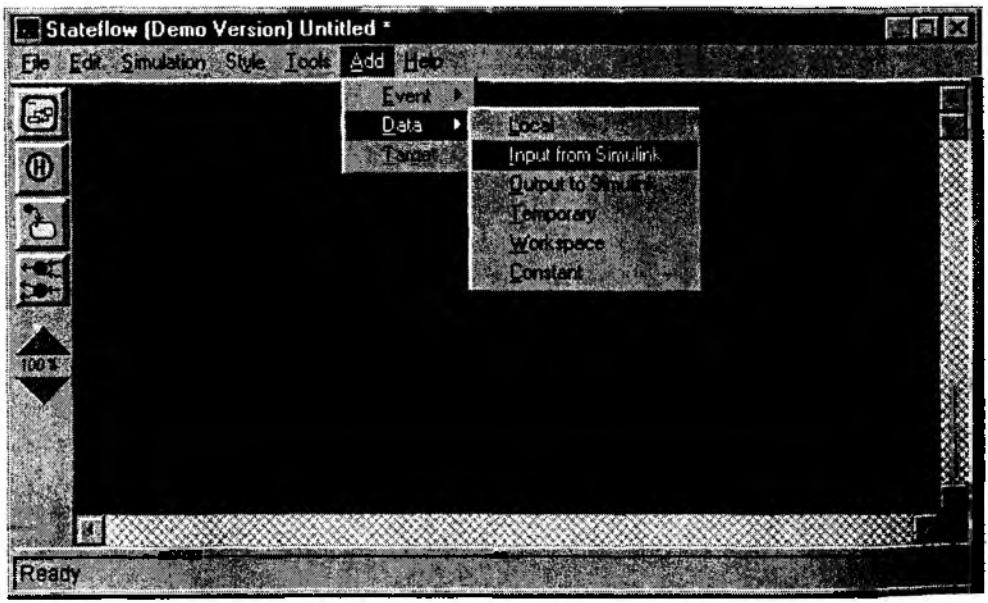


Рис. 5.24. Команды создания информационных портов *SF*-блока

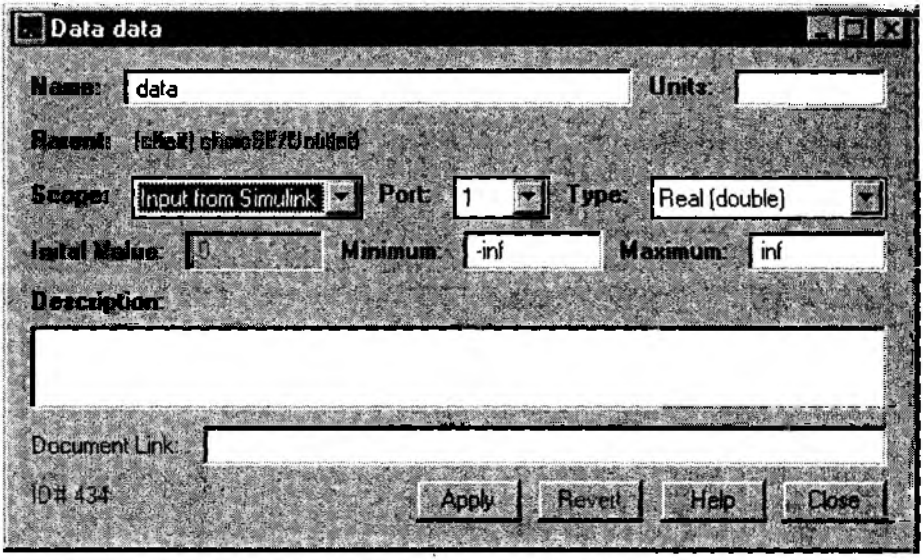


Рис. 5.25. Окно установки параметров информационных портов

редаваемых данных, их формат, диапазон значений и т. д. В рассматриваемом примере для всех параметров можно сохранить значения, установленные по умолчанию.

За один сеанс работы с окном *Data* может быть создан только один входной или выходной порт. Для описания выходного порта следует повторить ту же последовательность действий. После этого можно вернуться к блок-диаграмме модели *Выбор* и соединить *SF*-блок с другими ее блоками (рис. 5.26).

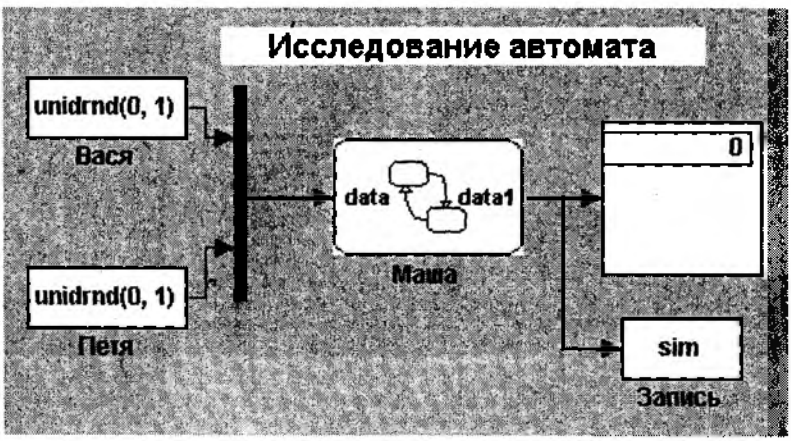


Рис. 5.26. Блок-диаграмма после включения в нее *SF*-блока

Для создания *SF*-модели используются визуальные компоненты, расположенные слева от диаграммы (рис. 5.27):

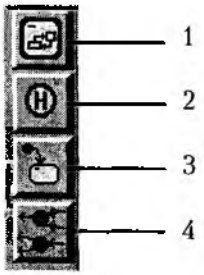


Рис. 5.27. Инструментальная панель графического редактора STATEFLOW

1 – добавить состояние;

2 – добавить «узел» (состояние более высокого уровня, остающееся активным при моделировании подчиненных состояний);

3 – добавить переход из одного состояния в другое;

4 – добавить разветвление/объединение переходов.

Чтобы внести любой из перечисленных элементов в *SF*-модель, достаточно щелкнуть на нем ЛКМ и переместить его графический образ в нужную позицию рабочего поля. Пример простейшей *SF*-модели приведен на рис. 5.28.

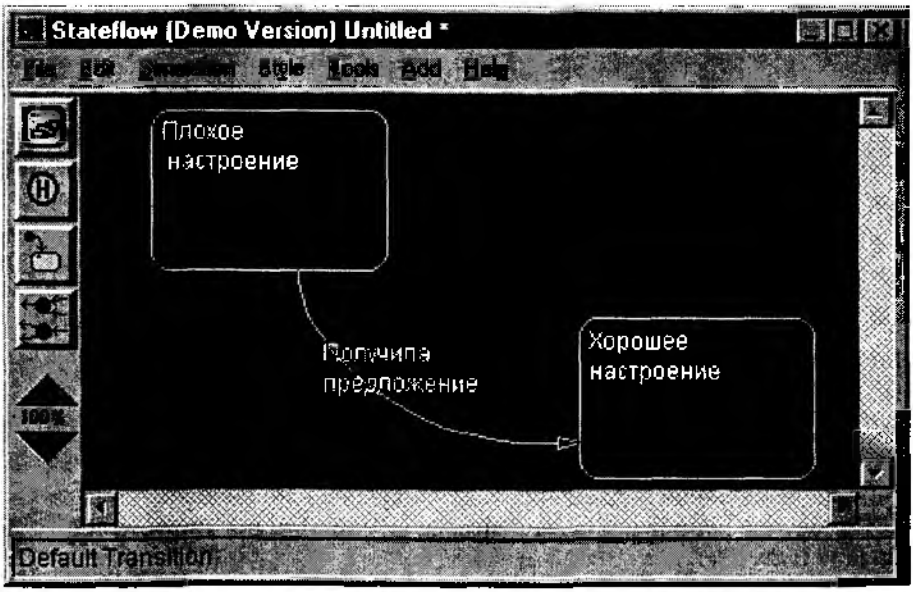


Рис. 5.28. Пример простейшей *SF*-модели

Дополнительные параметры элементов *SF*-модели устанавливаются с помощью опции *Properties*, входящей в раздел меню *Edit*.

На этапе отладки *SF*-модели она может исполняться автономно. Запуск модели производиться командой *Start* из раздела меню *Simulation*.

Из других инструментальных приложений MATLAB наибольший интерес (с точки зрения технологии поиска решений средствами имитационного моделирования) представляют два:

Fuzzy Logic Toolbox — инструментальные средства моделирования на основе методов нечетной логики;

Neural Network Toolbox — инструментальные средства поиска решений с помощью аппарата нейронных сетей.

Данные приложения наиболее эффективны (а иногда и просто незаменимы) в тех случаях, когда используемый показатель эффективности носит качественный характер и/или когда лицо, принимающее решение не может достаточно четко сформулировать правило, на основе которого это самое решение должно выбираться.

При включении обоих разделов в рабочую конфигурацию MATLAB дополняется и библиотека SIMULINK. Соответственно, разработчик получает возможность использовать в создаваемой модели как элементы нечеткой логики, так и средства описания системы в форме нейронной сети. Но это, как говорится, уже совсем другая история...

ЛИТЕРАТУРА

1. Надежность и эффективность в технике: Справочник: В 10 т./Ред. совет: Авдуевский В.С. (предс.) и др. — М.: Машиностроение, 1986.

2. *Питерсон Дж.* Теория сетей Петри и моделирование систем. — М.: Мир, 1984.

3. *Потемкин В. Г.* Система MATLAB 5 для студентов. — М.: ДИАЛОГ-МИФИ, 1998.

4. *Советов Б. Я.* Моделирование систем. — М.: Высшая школа, 1985.

5. *Саркисян С. А.* и др. Теория прогнозирования и принятия решений. — М.: Высшая школа, 1977.

6. *Эйзен С., Афифи А.* Статистический анализ: Подход с использованием ЭВМ. — М.: Мир, 1982.