Серия «Библиотека профессионала»

В. П. Дьяконов

MATLAB 6.5 SP1/7 + Simulink 5/6®
в математике и моделировании

Москва
СОЛОН-Пресс
2005

УДК 621.396.218
ББК 32.884.1
Д93

В. П. Дьяконов

Д93 **MATLAB 6.5 SP1/7 + Simulink 5/6® в математике и моделировании.**
Серия «Библиотека профессионала». — М.: СОЛОН-Пресс, 2005. — 576 с.: ил.

ISBN 5-98003-209-6

В этой второй, но вполне самостоятельной справочной монографии по новейшим системам MATLAB 6.5 SP1/7 + Simulink 5/6 описано их применение в математических расчетах и математическом моделировании. Особое внимание уделено анализу, моделированию и проектированию систем и устройств, в частности электрорадиотехнических и телекоммуникационных. Дано описание многих пакетов расширения Blockset и Toolbox последних реализаций, относящихся к математике и математическому моделированию. Ряд пакетов в нашей литературе описан впервые. Значительное внимание уделено визуализации результатов работы и описанию сотен примеров применения системы MATLAB и ее расширений.

Для инженеров, научных работников, студентов и преподавателей университетов и вузов.

MATLAB and Simulink are registered trademark of The MathWorks Inc.
Blockset, Toolbox and it components are trademark of The MathWorks Inc.

КНИГА — ПОЧТОЙ

Книги издательства «СОЛОН-Пресс» можно заказать наложенным платежом (оплата при получении) по фиксированной цене. Заказ оформляется одним из двух способов:

1. Послать открытку или письмо по адресу: 123242, Москва, а/я 20.
2. Передать заказ по электронной почте на адрес: *magazin@solon-r.ru*.

Бесплатно высылается каталог издательства по почте.

При оформлении заказа следует правильно и полностью указать адрес, по которому должны быть высланы книги, а также фамилию, имя и отчество получателя. Желательно указать дополнительно свой телефон и адрес электронной почты.

Через Интернет вы можете в любое время получить свежий каталог издательства «СОЛОН-Пресс». Для этого надо послать пустое письмо на робот-автоответчик по адресу: *katalog@solon-r.ru*.

Получать информацию о новых книгах нашего издательства вы сможете, подписавшись на рассылку новостей по электронной почте. Для этого пошлите письмо по адресу: *news@solon-r.ru*. В теле письма должно быть написано слово SUBSCRIBE.

По вопросам приобретения обращаться:
ООО «Альянс-книга»
Тел: (095) 258-91-94, 258-91-95, **www.abook.ru**

ISBN 5-98003-209-6

© Макет и обложка «СОЛОН-Пресс», 2005
© В. П. Дьяконов, 2005

Введение

В наши дни компьютерная математика получила должную известность и интенсивно развивается как передовое научное направление на стыке математики и информатики [1]. Среди ряда современных СКМ выгодно выделяется математическая матричная лаборатория MATLAB®. Она имеют мощные средства математически ориентированного программирования, диалога, графики и *комплексной визуализации* [2—23].

Популярности системы MATLAB способствует ее мощное расширение Simulink, предоставляющее пользователю удобные и простые средства, в том числе визуального объектно-ориентированного программирования, для блочного моделирования линейных и нелинейных динамических систем, а также множество других пакетов расширения системы. Некоторые из них являются наиболее доступными и эффективными инструментальными средствами для развития и применения новейших направлений науки и техники, таких как средства анализа и обработки сигналов и изображений (включая новейшую вейвлет-технологию), методы и инструменты анализа, идентификации и моделирования систем, нечеткая логика, нейронные сети и т. д.

Эта книга является вторым томом нового четырехтомника автора, в которой, в отличие от трехтомника [11—13], описаны базовые возможности двух последних реализаций системы MATLAB + Simulink, появившихся летом 2003 и 2004 г. Тем не менее данная книга является вполне самостоятельной благодаря размещению в ней первой главы, посвященной обзорному описанию систем MATLAB 6.5 SP1 и Simulink 5. Эта глава может рассматриваться как краткий вводный курс по этой реализации системы MATLAB + Simulink. Все остальные главы книги посвящены описанию практических возможностей системы MATLAB + Simulink и ее основных пакетов расширения. Это описание базируется на многочисленных примерах решения математических задач и математического моделирования самых разнообразных систем и устройств.

Книга дает наглядное представление о возможностях множества пакетов расширения системы MATLAB + Simulink, входящих в инструментальные «ящики» системы Blockset и Toolbox. Однако следует отметить несколько иной характер данной книги по сравнению с многочисленной литературой, посвященной описаниям тех или иных конкретных расширений системы [2—23]:

- в данной книге впервые описаны новейшие реализации пакетов расширения систем MATLAB + Simulink двух последних версий;
- достаточно подробно описаны расширения инструментального «ящика» Blockset, поскольку его средства широко используются во многих расширениях «ящика» Toolbox;
- детально описаны расширения Toolbox Notebook и Symbolic Math, придающие системе MATLAB принципиально новые качества — интеграции с текстовым процессором Word и мощной системой компьютерной алгебры Maple;
- в одной книге описаны разнообразные расширения системы MATLAB + Simulink самого общего характера, относящиеся к анализу, идентификации, проектированию и моделированию самых различных систем и устройств как общего характера, так и относящихся к отдельным отраслям науки и

техники, таким как энергетические системы, событийно управляемые системы, связные и телекоммуникационные системы и др.;

- в книге не ставилась задача полного справочного описания всех пакетов расширения Toolbox, ибо она в рамках одной книги недостижима из-за огромного объема документации по этим расширениям (многие тысячи страниц);
- описание применений MATLAB + Simulink дается на многочисленных примерах, как новых, так и взятых из фирменных описаний и ранее опубликованных книг, при этом возможности расширений Toolbox использовались по мере предметной необходимости;
- предпочтение отдавалось примерам, реализуемым в командном режиме работы и оформленным в виде m-файлов, что придает книге характер практического пособия по технике программирования в среде MATLAB;
- особое внимание уделялось сложным примерам, например решению математических задач с высокой размерностью и моделированию систем и устройств;
- несмотря на справочно-монографический характер книги она может использоваться студентами и преподавателями вузов для подготовки учебных курсов (в частности специальных), выполнения курсовых и дипломных работ.

Оглавление книги может служить подробным тематическим указателем. В целом книга имеет вполне законченный характер и полезна всем, кто собирается изучать или уже использует новые реализации систем MATLAB 6.5 SP1/7 + Simulink 5/6. В то же время в эту книгу не вошли пакеты расширения специального характера (например, по нечеткой логике, нейронным сетям, вейвлетам и др.), поскольку объем данной книги недостаточен для их детального описания. По пакетам расширения, предназначенным для обработки сигналов и изображений, готовится третий том данного четырехтомника.

Предупреждения

В книгах, описывающих новейшие реализации сложных программных продуктов, к коим относится система MATLAB + Simulink, трудно полностью избежать опечаток и даже отдельных ошибок. Как говорится, «и на солнце есть пятна». Автор считает нужным предупредить читателей об этом, хотя он сам и издательство сделали все возможное, чтобы свести ущерб от спешки в подготовке книги к разумному минимуму.

Автор считает нужным отметить, что в данной книге имеется ряд материалов из ранее изданных книг автора. Это естественная практика публикации серьезных справочников и описаний новых версий программных систем. Покупающий новый холодильник или телевизор покупатель вряд ли согласится приобрести вместо целой вещи лишь ее части, отличные от предыдущих моделей. В наше время книга тот же товар и вряд ли стоит полагать, что вместо справочника по той или иной системе, содержащего ее достаточно полное описание, серьезный покупатель предпочтет брошюрку с описанием только отличий новой версии программной системы от ее предыдущих версий.

Работа с такими мощными математическими системами, как MATLAB 6.5 SP1/7 и Simulink 5/6, с их расширениями, несомненно, требует от читателя знания основ математики и теории и техники математического моделирования. Без

этого невозможно гарантировать правильное применение используемых в этих системах методов и получение корректных результатов. В связи с этим следует отметить, что данная книга не является справочником по математике, численным методам вычислений самим по себе и даже не справочникам по многим пакетам расширений. Она представляет справочник по применению СКМ MATLAB + Simulink в математических расчетах и в математическом моделировании, основанный на описании большого числа практических примеров применения их в этих областях. Нужные теоретические сведения по ним можно найти в специальной литературе, например [24—39].

Мы вынуждены предупредить читателя, что автор и издательство не несут никакой ответственности за неудачи читателей в освоении такой сложной и мощной системы, как MATLAB с ее расширениями, и за моральный или даже экономический ущерб, который может иметь место вследствие ошибок и неудачного выбора математической системы для обучения или ее применения при решении конкретных задач пользователя.

Сказанное ни в коей мере не означает, что в данной книге или в системах MATLAB + Simulink заведомо имеются серьезные ошибки и недочеты. Просто такое предупреждение отвечает юридическим нормам современного права в отношении сложных программных продуктов и сопровождающей их документации.

Благодарности и адреса для связи

Автор выражает особую благодарность представителям фирмы MathWotks Naomi Fernandes и Courtney Esposito. Благодаря им подготовка данной книги была включена в планы поддержки корпорацией MathWorks изданий книг по системе MATLAB в разных странах мира и была обеспечена легальными программными средствами и обширной документацией по ним задолго до официального появления систем на рынке.

Автор благодарит доктора технических наук, профессора Владимира Круглова, кандидата физико-математических наук, доцента Романа Кристаллинского и кандидатов технических наук, доцентов Ирину Абраменкову и Андрея Пенькова за постоянный обмен мнениями и полезные замечания.

Автор благодарит также генерального директора ЗАО «Смоленский Телепорт» (www.keytown.com) Григория Рухамина за предоставление услуг спутникового Интернета в ходе работы над книгой, что позволило посредством прямой оперативной связи с сайтом фирмы MathWorks, Inc. быть в курсе обновлений системы MATLAB и использовать самую свежую информацию.

С автором можно связаться по электронной почте vpdyak@keytown.com. Автор заранее выражает признательность всем читателям, которые готовы сообщить свое мнение о данной книге и поделиться соображениями по ее улучшению. Кроме электронной почты, замечания можно направлять по следующему адресу: 214000, г. Смоленск, ул. Пржевальского, 4, СГПУ. Вы можете отправлять свои письма и по адресу издательства «СОЛОН-Пресс», выпустившего книгу. Консультаций по применению описанных в книге средств в учебном процессе автор не дает, поскольку это почетный долг преподавателей, ведущих занятия со студентами, и вмешиваться в их работу со стороны автора неэтично.

Связаться с фирмой MathWorks вы можете, посетив сайт www.mathworks.com. Ее официальные почтовые реквизиты следующие:

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA, 01760-2098 USA
Tel: 508-647-7000
Fax: 508-647-7101
E-mail: info@mathworks.com

С официальным дилером фирмы в России — корпорацией SoftLine — можно связаться через сайт http://www.softline.ru. Консультации по системе MATLAB читатель может получить на сайте www.matlab.ru и www.exponenta.ru.

Глава 1. Работа с MATLAB и Simulink

Эта глава является кратким курсом по работе в системах MATLAB 6.5 SP1/7 + Simulink 5/6. Она включена в данную книгу, чтобы сделать ее вполне самостоятельной. Кратко описаны установка систем, особенности реализации различных версий, пользовательский интерфейс системы и начало работы с ней. Рассмотрено решение типовых задач и применение средств визуализации результатов решения. В то же время полного описания встроенных функций системы MATLAB в ней, естественно, нет. Информацию о них легко получить из первой книги данного четырехтомника и справки по указанным системам. Данная глава дает объем сведений, достаточный для использования описанных далее пакетов по математическим расчетам и моделированию.

1.1. Назначение и особенности системы MATLAB

1.1.1. Назначение системы MATLAB

MATLAB — одна из старейших, тщательно проработанных и проверенных временем систем автоматизации математических расчетов [2—23], построенная на расширенном представлении и применении матричных операций [24—25]. Это нашло отражение в названии системы — MATrix LABoratory — *матричная лаборатория*.

Матрицы широко применяются в сложных математических расчетах, например, при решении задач линейной алгебры и математического моделирования статических и динамических систем и объектов [23—26]. Они являются основой автоматического составления и решения уравнений состояния динамических объектов и систем. Примером программы, выполняющей это, может служить расширение MATLAB — Simulink [7, 15, 22]. Оно существенно повышает интерес к системе MATLAB, вобравшей в себя лучшие достижения в области быстрого решения матричных задач за послевоенное время.

MATLAB поставляется как комплекс MATLAB + Simulink + Toolbox + Blockset, где разделами системы Toolbox и Blockset корпорация MathWorks называет пакеты расширения для систем MATLAB и Simulink, соответственно. Предшествующие версии этой системы были описаны в четырехтомнике автора [11—13]. Данная книга является вторым томом нового четырехтомника, посвященного двум новейшим и последним реализациям систем MATLAB 6.5 SP1/7 + Simulink 5/6, которые появились на рынке летом 2003 и 2004 г. Третий том нового четырехтомника посвящен обработке сигналов и изображений.

Одной из основных задач системы MATLAB всегда было предоставление пользователям мощного *языка программирования*, ориентированного на технические и математические расчеты и способного превзойти возможности традиционных языков программирования, которые многие годы использовались для реализации численных методов. При этом особое внимание уделялось как повышению скорости вычислений, так и адаптации системы к решению самых разнообразных задач пользователей.

Важными достоинствами системы являются ее *открытость* и *расширяемость*. Большинство команд и функций системы реализованы в виде m-файлов текстового формата (с расширением .m) и файлов на языке Си, причем все файлы доступ-

ны для модификации. Пользователю дана возможность создавать не только отдельные файлы, но и библиотеки файлов для реализации специфических задач. Такие файлы можно готовить как в простом и удобном редакторе m-файлов системы MATLAB, так и в любом другом текстовом редакторе — например Microsoft Word 95/97/2000/XP. Более того, такие файлы можно перенести с помощью буфера в командную строку MATLAB и тут же исполнить. А в новой реализации MATLAB 6.5 любой набор команд в справке можно тут же исполнить с помощью команды `Evaluate Selection` контекстного меню правой клавиши мыши.

Объем документации по пакетам расширения Toolbox составляет десятки тысяч страниц. Поэтому эта книга описывает достаточно полно только наиболее важные из пакетов расширения, которые используются для обработки сигналов и изображений. В этой главе описаны начала работы с базовой системой MATLAB (причем впервые в нашей литературе описана версия MATLAB 6.5 + Service Pack 1.0, которая является переходной к будущей реализации системы R14) и основным пакетом расширения Simulink 5. Это позволит читателю уверенно пользоваться описанными далее пакетами расширения. Однако для более полного знакомства с новыми реализациями MATLAB + Simulink надо взять все книги нового четырехтомника.

1.1.2. Новые реализации MATLAB 6.*

Казалось бы, совсем недавно (в 2001 г.) появилась новая версия этой системы — MATLAB 6.0 (реализация R12). Но уже спустя некоторое время вышла ее модификация MATLAB 6.1 (R12.1) с существенно измененными и расширенными пакетами применений — в частности с новой версией главного пакета расширения Simukink 4, обеспечивающего визуально-ориентированное блочное имитационное моделирование. Однако корпорация MathWorks не сочла ее очередной реализацией. И вот уже в 2002 году появилась очередная версия этой системы — MATLAB 6.5 (реализация R13).

Ожидалось, что к лету 2003 года ко дню независимости США выйдет реализация R14. Но вместо нее появилась промежуточная версия MATLAB + Service Pack 1.0. В ней устранено множество недоделок прежней версии, а главное — существенно переработаны многие пакеты расширения.

Поразительная легкость модификации системы и возможность ее адаптации к решению специфических задач науки и техники привели к созданию десятков пакетов прикладных программ, вошедших в инструментальный ящик системы Toolbox и намного расширивших сферы применения системы. Это особенно характерно для новых реализаций системы. Некоторые из пакетов расширения, например, Simulink (моделирование динамических систем и устройств, заданных в виде системы блоков), Notebook (интеграция с текстовым процессором Word и подготовка «живых» электронных книг) и Symbolic (символьные вычисления с применением ядра системы Maple) настолько органично интегрировались с системой MATLAB, что стали ее составными частями.

1.2. Возможности систем MATLAB

1.2.1. Возможности ранних версий MATLAB

Уже первые, ориентированные на Microsoft Windows, версии системы (MATLAB 4.x) обладали мощными средствами выполнения научно-технических вычис-

лений и их визуализации. В их числе задание и применение обычных векторов и матриц, разреженных матриц, многомерных массивов, массивов ячеек и структур. Уже системы MATLAB 5.* обладали обширными возможностями в визуализации результатов вычислений и имели десятки пакетов расширения. Однако, поскольку эти возможности детально описывались в книгах [1—9] мы опустим их описание и отметим возможности только новейших реализаций MATLAB.

Ввиду множества особенностей интерфейса последующих версий системы MATLAB и дополнения их многими новыми функциями и возможностями, отражение их в данной книге стало не целесообразным. «Любителям старины», пользующимся версиями MATLAB 4.*/5.*, вряд ли стоит приобретать данную книгу, если, конечно, они не планируют все же перейти на более новые версии системы.

1.2.2. Возможности версий MATLAB 6.0/6.1

Версия системы MATLAB 6.0 не только позволяет заменить предшествующие версии, но и характеризуется целым рядом новых и важных возможностей:

- доведенное до более чем 600 число встроенных функций и команд;
- новый интерфейс с набором инструментов для управления средой, включающий в себя окно команд (Command Window), окно истории команд (Command History), браузер рабочей области (Workspace Browser) и редактор массивов (Array Editor);
- новые инструменты, позволяющие при помощи мыши интерактивно редактировать и форматировать графики, оптимизировать их коды и затраты памяти на графические команды и атрибуты;
- улучшенные алгоритмы на основе оптимизированной библиотеки LAPACK;
- новая библиотека FFTW (быстрых преобразований Фурье) Массачусетского технологического института;
- ускоренные методы интегральных преобразований;
- новые, более мощные и точные алгоритмы интегрирования дифференциальных уравнений и квадратур;
- новые современные функции визуализации: вывод на экран двумерных изображений, поверхностей и объемных фигур в виде прозрачных объектов;
- новая инструментальная панель Camera для управления перспективой и ускорение вывода графики с помощью OpenGL;
- новый интерфейс для вызова Java-процедур и использования Java-объектов непосредственно из MATLAB;
- новые, современные инструменты проектирования графического пользовательского интерфейса;
- обработка (регрессия, интерполяция, аппроксимация и вычисление основных статистических параметров) графических данных прямо из окна графики;
- новое приложение MATLAB для системы разработки Visual Studio, позволяющее автоматически, непосредственно из Microsoft Visual Studio, преобразовывать Си и Си++ коды в выполняемые MATLAB файлы (MEX-файлы);
- интеграция с системами контроля версий кода, такими как Visual Source Safe;

- новый интерфейс (последовательный порт) для обмена данными с внешним оборудованием из MATLAB;
- новый пакет управления измерительными приборами (Instrument Control ToolBox) для обмена информацией с приборами, подключенными к Каналу общего пользования (GPIB, HP-IB, IEEE-488) или к шине VXI через адаптер VXI — GPIB (только в версиях для Windows и Sun Solaris) и последовательному интерфейсу RS-232, RS-422, RS-485 (также и для Linux-версии), в том числе в соответствии со стандартом VISA (Virtual Instruments Systems Application — Применение виртуальных измерительных приборов);
- существенно обновленные пакеты расширения, в частности, новые версии пакета моделирования динамических систем Simulink 4 и Real Time Workshop 4;
- интеграция с системами управления потребностями, например, DOORS.

Поставляемый с системой MATLAB 6.0 новый пакет расширения Simulink 4 также имеет ряд новинок. Они перечислены ниже по категориям.

Усовершенствование пользовательского интерфейса:

- новый графический отладчик для интерактивного поиска и диагностики ошибок в модели;
- усовершенствован навигатор моделей (Model Browser, Windows 95/98/Me/2000/NT4);
- новый однооконный режим для открытия подсистем;
- контекстное меню для блок-диаграмм (открывается щелчком правой кнопки мыши) как в Windows, так и в Unix версиях;
- новый диалог Finder для поиска моделей и библиотек.

Simulink поступает к пользователям с более 100 встроенными блоками, в состав которых входят наиболее необходимые функции моделирования. Блоки сгруппированы в библиотеки в соответствии с их назначением: источники сигнала, приемники, дискретные, непрерывные, нелинейные, математика, функции и таблицы, сигналы и системы. В дополнение к обширному набору встроенных блоков Simulink имеет расширяемую библиотеку блоков благодаря функции создания пользовательских блоков и библиотек.

Реализованы новые и улучшенные возможности блоков:

- наряду с существовавшей ранее поддержкой скалярных и векторных сигналов обеспечена поддержка матричных сигналов многими блоками Simulink;
- блоки Product, Multiplication, Gain и Math Function теперь поддерживают матричные операции на матричных сигналах;
- Mux и Demux блоки теперь поддерживают мультиплексирование матричных сигналов;
- новый блок Reshape изменяет размер матрицы своего входного сигнала;
- блок Probe теперь по умолчанию выводит размер матрицы сигнала, подаваемого на вход;
- новый блок Bitwise Logical Operator (логические операции по битам) накладывает маску, инвертирует или производит логические операции с отдельными битами целочисленного сигнала без знака;
- четыре новых блока Look-Up Table (просмотра таблиц);
- новый Polynomial блок выводит полиномиальную функцию от входного сигнала.

Расширенная поддержка для крупных приложений:

- новые объекты данных Simulink позволяют создавать специфические для приложений типы данных MATLAB;
- новый графический пользовательский интерфейс Simulink Explorer для наблюдения и редактирования объектов данных Simulink;
- усовершенствование блока Configurable Subsystems (конфигурируемые подсистемы);
- новое меню выбора блока конфигурируемой подсистемы;
- поддержка защиты интеллектуальной собственности с помощью S-функций, позволяющая не передавать исходный код S-функций (требуется Real-Time Workshop 4.0 (Лаборатория реального времени));
- поддержка S-функций, кодируемых на языке ADA (требуется новый отдельный пакет Real Time Workshop Ada Coder);
- улучшенная интеграция со Stateflow — пакетом инструментов моделирования систем, управляемых событиями, значительно усовершенствованный Stateflow Coder для генерации кода;
- run-time сервер MATLAB для запуска программ MATLAB без установленной системы MATLAB;
- улучшенная версия xPC Embedded Target для записи генерируемого кода не только на переносимые носители, но и в постоянные запоминающие устройства, твердотельные диски и на жесткий диск управляющего компьютера. Наряду с xPC поддерживаются другие платформы встроенных управляющих систем, включая VxWorks/Tornado (причем как UNIX, так и Windows хостом с MATLAB), Real Time Windows Target; Lynx Embedded OSEK Target, стандартизированную в автомобилестроении, DOS Target на управляющем компьютере Intel386 и старше (последняя только со снятым с производства компилятором Watcom Си/Си++ для DOS и несовместима с приложениями Windows). Но возможность работы без хоста с системой MATLAB (Stand-Alone) имеется только в xPC;
- поддержка xPC Target стандартной полевой шины промышленной автоматизации CAN, возможность синхронизации xPC сигналами, поступающими по этой шине;
- web-сервер, встроенный в xPC Target, позволяющий осуществлять управление встроенными компьютерами и просмотр их состояния при помощи браузеров Интернета (Microsoft Explorer 4.0 и старше и Netscape Navigator 4.5 и старше).

Все это говорит о том, что двенадцатый выпуск системы (MATLAB 6.0 + Simulink 4.0 + Stateflow 4.0 +...) подвергся не косметической, а самой серьезной переработке, выдвигающей эту систему на новый уровень развития и применения.

Вышедшая вслед за MATLAB 6.0 версия MATLAB 6.1 претерпела несущественные изменения. В нее, в частности, добавлены всего три новые функции — одна по обработке строк и две по реализации звуковых возможностей. Однако существенно обновился набор пакетов расширения этой системы. Тем не менее считать MATLAB 6.1 новой реализацией нельзя и не случайно MathWorks объявила MATLAB 6.1 лишь модификацией MATLAB 6.0.

1.2.3. Возможности версии MATLAB 6.5

В следующую версию системы MATLAB 6.5, объявленную уже новой 13-й реализацией [11, 12], введены следующие усовершенствования:

- расширенная поддержка виртуальной машины Java Virtual Machine;
- применение новой версии основного пакета моделирования систем Simulink 5;
- обновление полутора десятков пакетов применения;
- поддержка новых компьютерных платформ на базе операционных систем Windows XP и Unix, а также интеграция с Office XP;
- улучшенные опции печати и истории сессии;
- возможность стирания текущей директории в браузере директорий с помощью комбинации клавиш Shift + Delete (директория поступает в корзину Windows);
- опция автоматической записи Autosave в редакторе-отладчике m-файлов;
- улучшенный браузер системы помощи (Help Browser) и работа из него с демонстрационными примерами (отдельное окно последних удалено);
- отдельные усовершенствования в интерфейсе пользователя;
- поддержка импорта файлов в формате HDF с помощью нового инструмента HDF Import Tool;
- расширенная поддержка работы с файловыми серверами FTP и функцией ZIP;
- несколько новых математических функций, например логарифмической производной гамма-функции, функции тройного интегрирования и др.;
- улучшенный синтаксис ряда функций;
- улучшенная диагностика ошибок в программах;
- усовершенствованный синтаксис функций пользовательского интерфейса GUI;
- улучшенные средства общения с внешними приложениями;
- новые логические функции true и false.

Большинство нововведений в новейшей версии системы относится к интерфейсу пользователя и его возможностям. Новые возможности версии MATLAB 6.5 мы будем оговаривать особо.

1.2.4. Особенности версии MATLAB 6.5 + Service Pack 1.0

Летом 2003 года к очередному дню независимости США ожидалось появление новейшей 14-й реализации системы MATLAB с кардинально обновленными пакетами расширениями, включая Simulink. Однако огромную работу по подготовке новой реализации системы к назначенной дате выполнить не удалось, и была выпущена промежуточная серия системы с модным ныне названием MATLAB 6.5.1 + Service Pack 1. Если быть точными, то речь идет о реализации R13SP1.

О масштабности доработки системы говорит тот факт, что в ходе ее были написаны заново или существенно переработаны более 100 000 блоков исходного кода системы. Исчезли проблемы со шрифтами, в том числе кириллическими, заметно повысилась стабильность работы, исчезло подвисание Windows при выключении после сеанса работы с MATLAB.

К новым возможностям системы MATLAB 6.5.1 + Service Pack 1 относится:

- улучшенная поддержка динамических библиотек — MATLAB Interface to Generic DDLs;
- поддержка операторов отношения, таких как <, >, <=, >=, ~= и ==, для новых типов данных int64 и utit 64;
- поддержка считывания файлов формата HDF5 (Hierarchical Data Format, Version 5), предложенного Национальным Центром суперкомпьютерных приложений (NCSA);
- расширенная поддержка файлов формата TIF, в частности запись и считывание L*a*b цветового пространства TIF-файлов;
- устранение множества недоработок предшествующей версии;
- обеспечение работы функции sqrt с комплексными данными;
- улучшение работы с COM-интерфейсом;
- существенная переработка более чем полутора десятков пакетов расширения системы MATLAB, включая Simulink;
- поддержка получения из Интернета новых реализаций пакетов расширения;
- расширенная поддержка компьютерных платформ (см. ниже).

Версия MATLAB 6.5.1 + Service Pack 1 может работать на большом числе компьютерных платформ:

- Windows 98 (оригинальная и second edition);
- Windows ME;
- Windows 2000 (Service Pack 1, 2, or 3);
- Windows NT 4.0 (Service Pack 5, 6 или 6a);
- Windows XP (Service Pack 1);
- Linux ix86 2.2.x or 2.4.x, glibc (libc6) 2.1.2, glibc (libc6) 2.2.5 (рекомендуется);
- Sun Solaris 2.6, 2.7, 2.8 и 2.9;
- Alpha TRU64;
- UNIX 4.0f, 4.0g, 5.0, 5.1, or 5.1a;
- HP-UX 10.20, 11.0 или 11i;
- SGI IRIX/IRIX64 6.5.8 — 6.5.12, 6.5.15f;
- IBM AIX 4.3.3 or 5.1;
- Macintosh OS X 10.1.5 и 10.2.

Хотя в этой книге описаны примеры только для реализации MATLAB под Windows, большая часть материала книги полностью пригодна и для реализаций на других компьютерных платформах.

1.3. Особенности матричных систем MATLAB

1.3.1. Интеграция с другими программными системами

С системой MATLAB могут интегрироваться такие популярные математические системы, как Mathcad, Maple V и Mathematica. Есть тенденция и к объединению математических програм с современными текстовыми процессорами. Так, новое средство последних версий MATLAB — Notebook — позволяет готовить документы в текстовом процессоре Word 95/97/2000/XP со вставками в виде документов MATLAB и результатов вычислений, представленных в численном, таб-

личном или графическом виде. Таким образом, становится возможной подготовка «живых» электронных книг, в которых демонстрируемые примеры могут быть оперативно изменены. Так, вы можете менять условия задач и тут же наблюдать изменение результатов их решения.

В MATLAB задачи расширения системы решаются с помощью специализированных *пакетов* (Toolbox). Многие из них содержат специальные средства для интеграции с другими программами, поддержки объектно-ориентированного и визуального программирования, для генерации различных приложений. В состав системы MATLAB входит ядро одной из самых мощных, популярных и хорошо апробированных систем символьной математики (компьютерной алгебры) Maple V. Оно используется пакетами расширения Symbolic Math Toolbox и Extended Symbolic Math Toolbox, благодаря которым в среде MATLAB стали доступны принципиально новые возможности символьных и аналитических вычислений.

Новые свойства системе MATLAB придала ее интеграция с программной системой Simulink, созданной для моделирования блочно заданных динамических систем и устройств. Базируясь на принципах визуально-ориентированного программирования, Simulink позволяет выполнять моделирование сложных устройств с высокой степенью достоверности и прекрасными средствами представления результатов. Помимо естественной интеграции с пакетами расширения Symbolic Math и Simulink, MATLAB интегрируется с десятками мощных пакетов расширения.

В свою очередь, многие другие математические системы, например Mathcad и Maple, допускают установление объектных и динамических связей с системой MATLAB, что позволяет использовать в них эффективные средства MATLAB для работы с матрицами. Эта прогрессивная тенденция интегрирования компьютерных математических систем, несомненно, будет продолжена.

1.3.2. Ориентация на матричные операции

Напомним, что двумерный массив чисел или математических выражений принято называть *матрицей*. А одномерный массив называют *вектором*. Примеры векторов и матриц даны ниже:

$$\begin{vmatrix} 1 & 2 & 3 & 4 \end{vmatrix}$$ Вектор из 4 элементов

$$\begin{vmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 8 & 7 & 6 \end{vmatrix}$$ Матрица размера 3 × 4

$$\begin{vmatrix} a & a+b & a+b/c \\ x & y*x & z \\ 1 & 2 & 3 \end{vmatrix}$$ Матрица с элементами разного типа

Векторы и матрицы характеризуются *размерностью* и *размером*. Размерность определяет структурную организацию массивов в виде строки (размерность 1), страницы (размерность 2), куба (размерность 3) и т. д. Так что вектор является одномерным массивом, а матрица представляет собой двумерный массив с размерностью 2. MATLAB допускает задание и использование многомерных массивов, но в этой главе мы ограничимся только одномерными и двумерными массивами — векторами и матрицами.

Размер вектора — это число его элементов, а размер матрицы определяется числом ее строк m и столбцов n. Обычно размер матрицы указывают как $m \times n$.

Матрица называется квадратной, если $m = n$, то есть число строк матрицы равно числу ее столбцов.

Векторы и матрицы могут иметь имена, например, **V** — вектор или **M** — матрица. В данной книге имена векторов и матриц набираются полужирным шрифтом. Элементы векторов и матриц рассматриваются как *индексированные переменные*, например:

$\mathbf{V}_2$ — второй элемент вектора **V**;

$\mathbf{M}_{23}$ — третий элемент второй строки матрицы **M**.

Интересно отметить, что даже обычные числа и переменные в MATLAB рассматриваются как матрицы размера 1 × 1, что дает единообразные формы и методы проведения операций над обычными числами и массивами. Это также означает, что большинство вычислительных функций может работать с аргументами в виде векторов и матриц, вычисляя значения для каждого их элемента. Данная операция обычно называется *векторизацией* и обеспечивает упрощение записи операций, производимых одновременно над всеми элементами векторов и матриц, и существенное повышение скорости их выполнения.

1.3.3. Расширяемость системы

MATLAB — расширяемая система, и ее легко приспособить к решению нужных пользователю специфических классов задач. Ее огромное достоинство заключается в том, что это расширение достигается естественным путем и реализуется в виде так называемых m-файлов (с расширением .m). Иными словами, расширения системы хранятся на жестком диске компьютера и в нужный момент вызываются для использования точно так же, как встроенные в MATLAB (внутренние) функции и процедуры.

Благодаря текстовому формату m-файлов пользователь может ввести в систему любую новую команду, оператор или функцию и затем пользоваться ими столь же просто, как и встроенными операторами или функциями. При этом, в отличие от таких языков программирования, как Бейсик, Си или Паскаль, не требуется никакого объявления этих новых функций. Это роднит MATLAB с языками Лого и Форт, имеющими словарную организацию операторов и функций и возможности пополнения словаря новыми определениями-словами. Но, поскольку новые определения в системе MATLAB хранятся в виде файлов на диске, это делает набор операторов и функций практически неограниченным.

В базовый набор слов системы входят спецзнаки, знаки арифметических и логических операций, арифметические, алгебраические, тригонометрические и некоторые специальные функции, функции быстрого преобразования Фурье и фильтрации, векторные и матричные функции, средства для работы с комплексными числами, операторы построения графиков в декартовой и полярной системах координат, трехмерных поверхностей и т. д. Словом, MATLAB предоставляет пользователю обширный набор готовых средств (большая часть из них — это внешние расширения в виде M-файлов).

Дополнительный уровень системы образуют ее *пакеты расширения*. Они позволяют быстро ориентировать систему на решение задач в той или иной предметной области: в специальных разделах математики, в физике и в астрономии, в области нейтронных сетей и средств телекоммуникаций, в математическом моделировании, проектировании событийно управляемых систем и т. д. Благодаря этому MATLAB обеспечивает высочайший уровень адаптации к решению задач конечного пользователя.

1.3.4. Мощные средства программирования

Многие математические системы создавались, исходя из предположения, что пользователь будет решать свои задачи, практически не занимаясь программированием. Однако с самого начала было ясно, что подобный путь имеет недостатки и по большому счету порочен. Многие задачи нуждаются в развитых средствах программирования, которые упрощают запись их алгоритмов и порою открывают новые методы создания последних.

С одной стороны, MATLAB содержит огромное число встроенных операторов и функций (приближающееся к тысяче), которые успешно решают множество практических задач, для чего ранее приходилось готовить достаточно сложные программы. К примеру, это функции обращения или транспонирования матриц, вычисления значений производной или интеграла и т. п. Число таких функций с учетом пакетов расширения системы уже достигает многих тысяч и непрерывно увеличивается.

Но, с другой стороны, система MATLAB с момента своего создания создавалась как мощный математико-ориентированный на технические вычисления язык программирования *высокого уровня*. И многие вполне справедливо рассматривали это как важное достоинство системы, свидетельствующее о возможности ее применения для решения новых, наиболее сложных математических задач.

Система MATLAB имеет *входной язык*, напоминающий Бейсик (с примесью средств Фортрана и Паскаля). Запись программ в системе традиционна и потому понятна для большинства пользователей компьютеров. К тому же система дает возможность редактировать программы с помощью любого, привычного для пользователя текстового редактора. Имеет она и собственный редактор с отладчиком.

Язык системы MATLAB в части программирования математических вычислений намного богаче любого универсального языка программирования высокого уровня. Он реализует почти все известные средства программирования, в том числе объектно-ориентированное и визуальное программирование. Это дает опытным программистам необъятные возможности для самовыражения.

1.3.5. Визуализация и графические средства

В последнее время создатели математических систем уделяют огромное внимание *визуализации* решения математических задач. Говоря проще, это означает, что постановка и описание решаемой задачи и результаты решения должны быть предельно понятными не только тем, кто решает задачи, но и тем, кто в дальнейшем их изучает или просто просматривает. Большую роль в визуализации решения математических задач играет графическое представление результатов, причем как конечных, так и промежуточных.

Визуализация постановки задачи в MATLAB решается применением приложения Notebook [12] и назначением именам функций достаточно ясных имен (идентификаторов). А визуализация результатов вычислений достигается применением обширных средств графики, в том числе анимационной, а также использованием (там, где это нужно) средств символьной математики.

Новая версия MATLAB напрочь избавилась от некоторой примитивности графиков, которая была присуща первым версиям этой системы. Теперь новые графические средства Handle Graphics (дескрипторная или описательная графика) позволяют создавать полноценные объекты графики высокого разрешения, как геометрического, так и цветового. Возможности этой графики поддерживаются

объектно-ориентированным программированием, средства которого также имеются в языке программирования системы MATLAB.

Реализуются, причем с повышенной скоростью, построения графиков практически всех известных в науке и технике типов. Широко практикуется функциональная закраска сложных поверхностей, в том числе с интерполяцией по цвету и свойства прозрачности, основанные на применении графических средств OpenGL. Возможен учет разнообразных световых эффектов — вплоть до бликов на поверхности сложных фигур при освещении их различными источниками света и с учетом свойств материалов отражающих поверхностей и прозрачности. Применение дескрипторной (описательной) графики позволяет создавать и типовые элементы пользовательского интерфейса — кнопки, меню, информационные и инструментальные панели и т. д., то есть реализовать элементы *визуально-ориентированного программирования*.

Графики выводятся отдельно от текстов в отдельных окнах. На одном графике можно представить множество кривых, отличающихся цветом (при цветном дисплее) и отличительными символами (кружками, крестиками, прямоугольниками и т. д.). Графики можно выводить в одно или в несколько окон. Наконец, в статьях и книгах формата Notebook, реализованных при совместной работе системы MATLAB с популярным текстовым процессором Word 95/97, графики могут располагаться вместе с текстом, формулами и результатами вычислений (числами, векторами и матрицами, таблицами и т. д.). В этом случае степень визуализации оказывается особенно высокой, поскольку документы класса Notebook по существу являются превосходно оформленными электронными книгами с действующими (вычисляемыми) примерами.

Особенно привлекательной выглядит возможность построения трехмерных поверхностей и фигур. По сравнению с системой Mathcad, построение трехмерных фигур средствами MATLAB происходит почти на порядок быстрее. Кроме того, при построении таких графиков используется достаточно совершенный алгоритм удаления невидимых линий, что наряду с большими размерами графиков и возможностью интерполяции по цвету делает построенные трехмерные поверхности и фигуры весьма эстетичными и наглядными. Уже в MATLAB 5.3.1 была введена эффектная возможность быстрого вращения графиков мышью в любом направлении. В MATLAB 6.* она улучшена — теперь вращать в пространстве можно даже плоскость с двумерными графиками.

Введен также ряд средств на основе графического интерфейса пользователя (GUI — Graphic User Interface), привычного для операционных систем Windows 95/98/NT. Это панели инструментов, редактор и отладчик m-файлов, красочная демонстрация возможностей и т. д. Есть и возможность создавать свои средства пользовательского интерфейса.

1.3.6. О компиляции программ системы MATLAB

Язык программирования систем MATLAB относится к классу интерпретаторов. Это обеспечивает простой интерактивный характер взаимодействия пользователя с системой — любая функция системы или набор их (кодов программы) может исполняться после ввода в командной строке или исполнения M-файла (с расширением .m). Это существенно облегчает отладку программных модулей.

Однако M-файлы могут исполняться только в среде системы MATLAB. Полностью законченных исполняемых файлов с расширением .exe или .com система не создает. Между тем MATLAB весьма громоздкая система и ее наличие на компью-

тере — тяжкое бремя. Многие пользователи предпочли бы иметь свои программы, созданные на языке MATLAB, в виде самостоятельных исполняемых модулей, в случае необходимости сопровождаемых динамическими библиотеками.

MATLAB предусматривает такую возможность за счет включения в его состав специального компилятора и дополнения системы языком программирования высокого уровня C или C++. К сожалению, поставки MATLAB в таком виде крайне редки и стоят очень дорого. Самостоятельное расширение системы в принципе возможно, но, как показывает практика, требует привлечения для этого весьма опытных программистов и, соответственно, больших средств на оплату их работы. В связи с этим ограничимся лишь упоминанием возможности компиляции программных модулей системы MATLAB и в дальнейшем будем рассматривать все примеры ее применения только в среде системы MATLAB, т. е. в интерпретирующем режиме ее работы.

1.4. Установка и файловая система MATLAB 6.*

1.4.1. Системные требования

Новые версии системы MATLAB — весьма громоздкие программные комплексы, которые (при полной установке) требуют каждый до 1000–1500 Мбайт дисковой памяти (в зависимости от конкретной поставки, полноты справочной системы и числа устанавливаемых пакетов прикладных программ). Поэтому они поставляются на двух компакт-дисках (CD-ROM), на одном из которых размещены системные файлы, а на другом PDF-файлы документации. В версии MATLAB 6.5 SR1 предусмотрена возможность работы с PDF-файлами документации прямо с CD-ROM (третьего диска из трех, на которых поставляется система).

Для успешной установки MATLAB 6.5 SP1 необходимы следующие минимальные средства:

- компьютер с микропроцессором не ниже Pentium и математическим сопроцессором, рекомендуются процессоры Pentium PRO, Pentium II, Pentium III, Pentium 4 или AMD Athlon (учтите, что для установки на ПК с процессором Pentium 4 годятся только версии, начиная с MATLAB 6.1);
- устройство считывания компакт-дисков (привод CD-ROM) (для установки), мышь, 8-разрядный графический адаптер и монитор, поддерживающие не менее 256 цветов;
- операционная система Windows 95/98 (оригинальная или второе издание) /Me (Millennium Edition)/2000 (допускается также NT4 с сервис-пакетами 5 или 6a, а также Windows XP для MATLAB 6.5);
- ОЗУ емкостью 64 Мбайт для минимального варианта системы (рекомендуется иметь память не менее 128 Мбайт);
- до 1500 Мбайт дискового пространства при полной установке всех расширений и всех справочных систем (при ограничении справки англоязычными документами MATLAB 6.5 снижает свои «аппетиты» и устанавливается примерно в 1000 Мбайт дискового пространства).

Для использования расширенных возможностей системы нужны графический ускоритель, Windows-совместимые звуковая карта и принтер, текстовый процессор Microsoft Word 95/97/2000 для реализации Notebook, компиляторы языков Си/Си++ и/или ФОРТРАН для подготовки собственных файлов расширения и

браузер Netscape Navigator 4.0 и выше или Microsoft Internet Explorer 4.0 и выше. Для просмотра файлов справочной системы в формате PDF нужна программа Adobe Reader или Adobe Acrobat 3.0 и выше.

Отличия между платформами, на которых может работать система MATLAB, в основном связаны со скоростью выполнения операций, в особенности при выводе трехмерной графики, при расчете сцены и рендеринге с новым, введенным только в данной версии, механизмом Open GL, и с отдельными деталями интерфейса. Как гарантирует MathWorks, отличия совсем (или для платформ HP и IBM почти совсем) не затрагивают базового набора возможностей ядра и пакетов прикладных программ. Поэтому читатели, работающие с MATLAB 6.* на любой платформе, могут пользоваться всеми или большей частью материалов данной книги.

1.4.2. Инсталляция систем MATLAB 6.*

Установка систем MATLAB в среде Windows 95/98/NT обычно не имеет никаких специфических особенностей и подобна установке других программных продуктов. Возможны типичная установка и выборочная, в ходе которой вам предлагается выбор компонентов системы. Последняя предпочтительнее, так как из-за огромного объема системы ее полная установка не всегда возможна.

Для установки системы на ПК достаточно вставить установочный компакт-диск в устройство. Диск запускается автоматически и выводит временное окно — заставку системы. При этом копируются вспомогательные файлы MathWorks Installer (Мастера установки). При выборочной установке в окне инсталлятора (рис. 1.1) надо отметить те пакеты расширения, которые желательно загрузить.

Полное описание установки можно найти в [5, 6], но, как показывает практика, даже начинающий пользователь в наше время справляется с ней без особых затруднений. Учтите лишь, что нередко установка занимает 2—3 часа (меньше только на современных компьютерах со скоростными микропроцессорами и CD-ROM-драйвами).

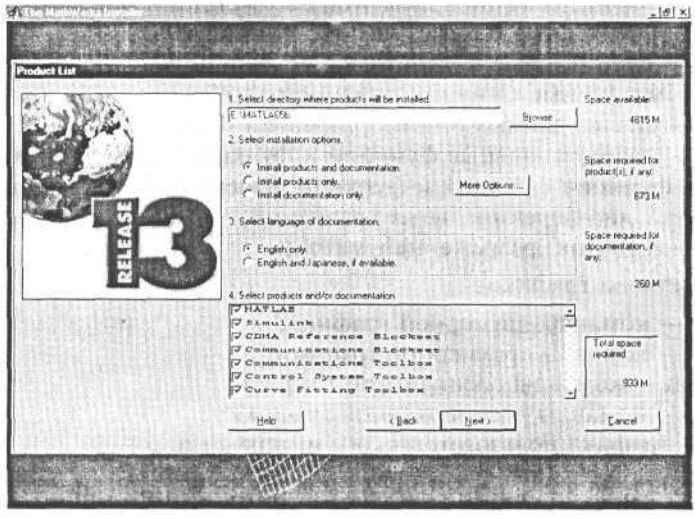

Рис. 1.1. Окно инсталлятора MATLAB с перечнем расширений

1.4.3. Файловая система MATLAB

Система MATLAB состоит из многих тысяч файлов, находящихся в множестве папок. Полезно иметь представление о содержании основных папок, поскольку это позволяет быстро оценить возможности системы — например, узнать, какие операторы, функции или графические команды входят в систему.

В MATLAB особое значение имеют файлы двух типов — с расширениями .mat и .m. Первые являются бинарными файлами, в которых могут храниться значения переменных. Вторые представляют собой текстовые файлы, содержащие внешние программы, определения команд и функций системы. Именно к ним относится большая часть команд и функций, в том числе задаваемых пользователем для решения своих специфических задач. Нередко встречаются и файлы с расширением .c (коды на языке Си), файлы с откомпилированными кодами MATLAB с расширением .mex и другие. Исполняемые файлы имеют расширение .exe.

Особое значение имеет папка MATLAB/TOOLBOX/MATLAB. В ней содержится набор стандартных m-файлов системы. Просмотр этих файлов позволяет детально оценить возможности поставляемой конкретной версии системы. Ниже перечислены основные подпапки с этими файлами (деление на категории условно, на самом деле все подпапки находятся в общей папке MATLAB/TOOLBOX/MATLAB).

Подпапка команд общего назначения:

General — команды общего назначения: работа со справкой, управление окном MATLAB, взаимодействие с операционной системой и т. д.

Подпапки операторов, конструкций языка и системных функций:

ops — операторы и специальные символы;
lang — конструкции языка программирования;
strfun — строковые функции;
iofun — функции ввода/вывода;
timefun — функции времени и дат;
datatypes — типы и структуры данных.

Подпапки основных математических и матричных функций:

elmat — команды создания элементарных матриц и операций с ними;
elfun — элементарные математические функции;
specfun — специальные математические функции;
matfun — матричные функции линейной алгебры;
datafun — анализ данных и преобразования Фурье;
polyfun — полиномиальные функции и функции интерполяции;
funfun — функции функций и функции решения обыкновенных дифференциальных уравнений;
soarfun — функции разреженных матриц.

Подпапки команд графики:

graph2d — команды двумерной графики;
graph3d — команды трехмерной графики;
specgraph — команды специальной графики;
graphics — команды дескрипторной графики;
uitools — графика пользовательского интерфейса.

Полный состав файлов каждой подпапки (их список содержится в файле contents.m) можно вывести на просмотр с помощью команды `help имя`, где `имя` — название соответствующей подпапки.

1.5. Начало работы с MATLAB

1.5.1. Запуск MATLAB и работа в режиме диалога

MATLAB обычно запускается из главного меню операционной системы Windows 98 со стандартным видом рабочего стола, подобным использованному в Windows 95. Для раскрытия главного меню активизируется кнопка Пуск (Start), расположенная внизу рабочего стола слева, или можно щелкнуть на значке с логотипом системы на рабочем столе Windows.

После запуска MATLAB на экране появляется основное окно системы, показанное на рис. 1.2. Обычно это окно раскрыто не полностью и занимает часть рабочего стола. Вы можете его увеличить, щелкнув на средней из трех кнопок, расположенных в конце титульной (верхней) строки окна. Левая кнопка сворачивает окно в кнопку с именем приложения, помещаемую в панель задач Windows 95/98, а правая закрывает окно и прекращает работу с MATLAB.

Система готова к проведению вычислений в *командном режиме*. При этом вы можете не обращать внимания на новации пользовательского интерфейса, привнесенные операционными системами Windows 95 и 98/Me/2000/NT4, в виде расширяемого окна и панели инструментов. Мы обсудим их роль позже.

Для явного уточнения версии системы следует вывести окно с информацией о системе (команда About MATLAB (О MATLAB) в меню Help (Помощь). Это окно представлено на рис. 1.3 в центре. Из него видно, что в данном случае запущена версия 6.5.1 SP1 от 4 августа 2003 года. Поскольку номер лицензии имеет конфиденциальный характер, вместо него на рисунке проставлен 0. Аналогичным образом легко получить данные о той версии, которая используется пользователем на его ПК.

Полезно знать, что в начале запуска автоматически выполняется команда `matlabrc`, которая исполняет загрузочный файл matlabrc.m и файл startup.m, если

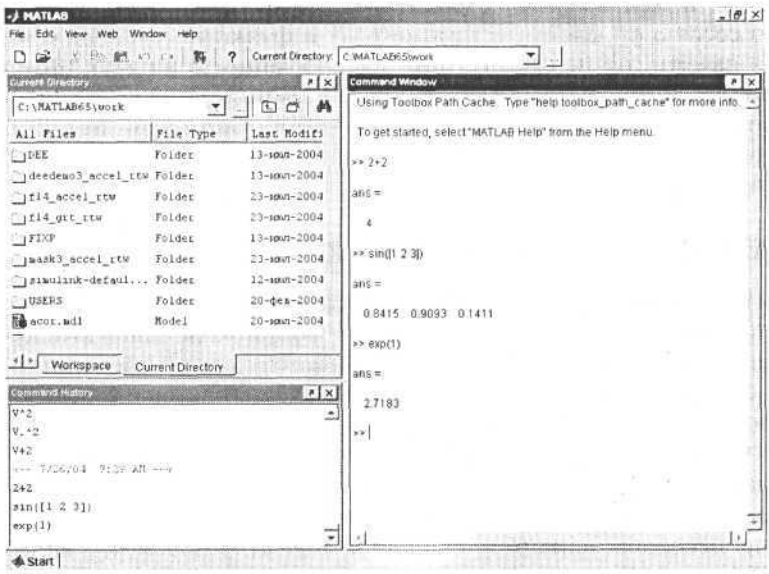

Рис. 1.2. Окно системы MATLAB после запуска и выполнения простых вычислений

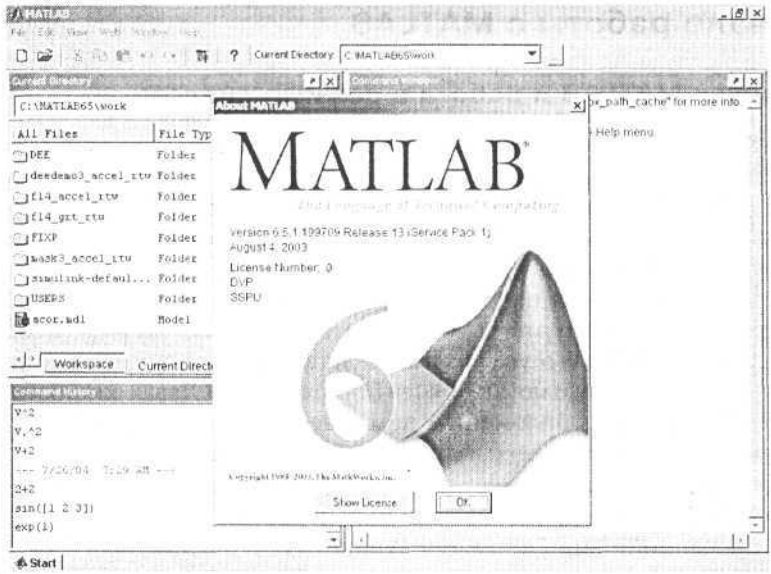

Рис. 1.3. Окно с логотипом системы MATLAB

таковой существует. Эти файлы выполняют начальную настройку терминала системы и задают ряд ее параметров. В частности, могут быть заданы пути доступа к другим файлам, необходимым для корректной работы системы MATLAB. Поскольку указанные файлы имеют текстовый формат, их легко просмотреть с помощью какого-либо текстового редактора или с помощью команды `type` в командном режиме работы MATLAB.

Сеанс работы с MATLAB принято именовать *сессией* (session). Сессия, в сущности, является текущим документом, отражающим работу пользователя с системой MATLAB. В ней имеются строки ввода, вывода и сообщений об ошибках. Входящие в сессию определения переменных и функций, расположенные в рабочей области памяти, но не саму сессию, можно записать на диск (файлы формата .mat), используя команду `save` (Сохранить). Команда `load` (Загрузить) позволяет считать с диска данные рабочей области. Фрагменты сессии можно оформить в виде дневника с помощью команды `diary` (Дневник). Позже мы обсудим эти команды подробно.

1.5.2. MATLAB 6.5 в среде Windows XP

Одной из важных отличительных черт MATLAB 6.5 является возможность работы в среде новейшей операционной системы нового поколения — Windows XP. Надо сразу предупредить читателя, что особых оснований к использованию Windows XP для работы с системой MATLAB нет. Как показал опыт автора, серьезного повышения производительности вычислений в этой операционной системе нет, зато есть некоторые (к счастью несерьезные) проблемы с цветовыми палитрами.

Основанием к применению Windows XP является, пожалуй, лучшая поддержка этой операционной системой новых возможностей потокового видео и новых типов микропроцессоров, например Pentium 4. Качество просмотра видеофильмов в формате MP4 в новой операционной системе явно выше, чем в случае применения Windows 95/98/NT. Многим по вкусу придется и вид окна новой операцион-

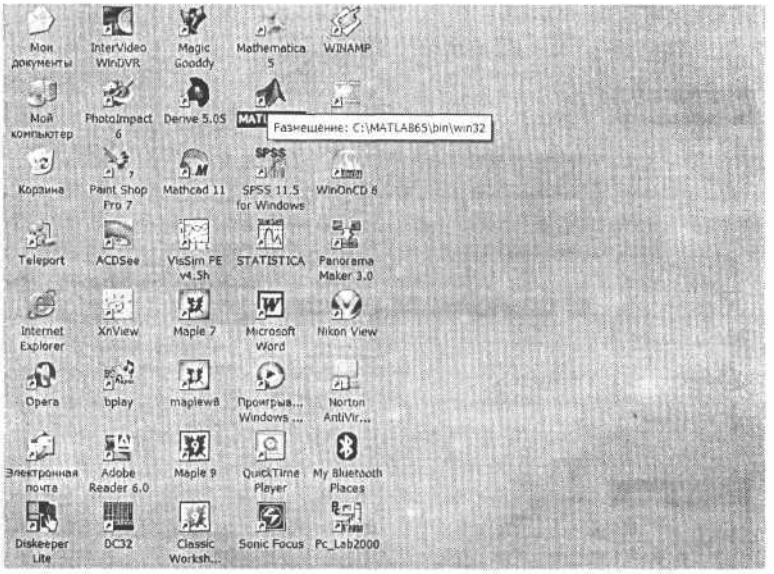

Рис. 1.4. Рабочий стол операционной системы Windows XP с ярлыком системы MATLAB 6.5

ной системы (рис. 1.4), которое меньше загружено ярлыками и папками и имеет более приятный для глаза вид. В частности, благодаря применению плавных переходов цвета и скругления очертаний многих окон. На рис. 1.4 можно увидеть ярлык системы MATLAB 6.5, появляющийся после ее инсталляции.

Инсталляция MATLAB 6.5/6.5 SP1 в среде Windows XP проходит аналогично описанной для Windows 98. На рабочем столе появляется ярлык системы MATLAB, видный на рис. 1.4. Обратите внимание на одно из многочисленных усовершенствований Windows XP — при наведении курсора мыши на ярлык появляется поле с указанием полного пути к исполняемому файлу, представленному этим ярлыком.

1.5.3. Запуск MATLAB 6.5 в среде Windows XP

Как и в Windows 98, запуск системы MATLAB 6.5 в среде Windows XP возможен двумя основными способами — активизацией ярлыка системы на рабочем столе и из рабочего меню. Второй способ иллюстрируется рис. 1.5, на котором представлено рабочее меню системы Windows XP с полным путем доступа к исполняемому файлу системы MATLAB.

Полезно обратить внимание на возможность использования контекстного меню правой клавиши мыши в момент выделения той или иной позиции рабочего меню. Контекстное меню для исполняемого файла системы MATLAB 6.5 представлено на рис. 1.4. На рис. 1.6 показаны окна системы MATLAB 6.5 + SP1. Обратите внимание на окно с данными о системе и отделенное окно браузера файловой системы.

Интерфейс системы является настраиваемым. Основные средства его настройки представлены в позиции `View` меню. Стандартный вид окна MATLAB 6.5 SP1 представлен на рис. 1.7.

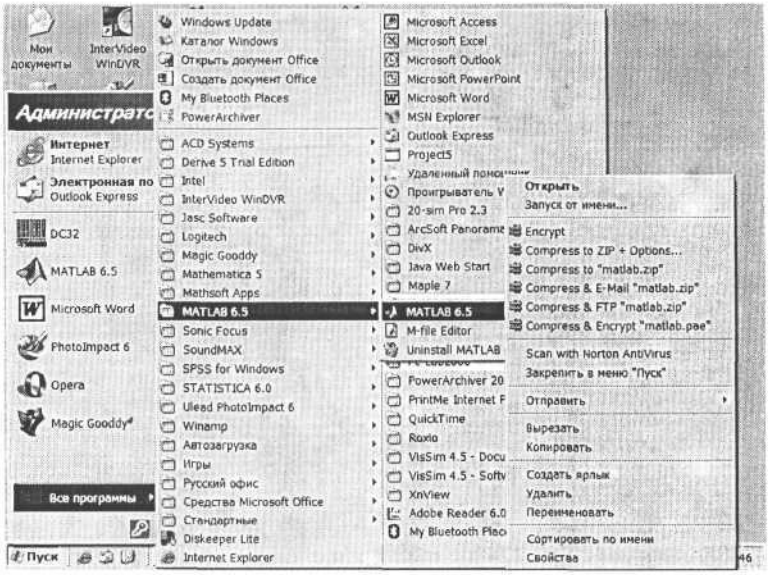

Рис. 1.5. Рабочее меню Windows XP с доступом к исполняемому файлу системы MATLAB 6.5 SP1

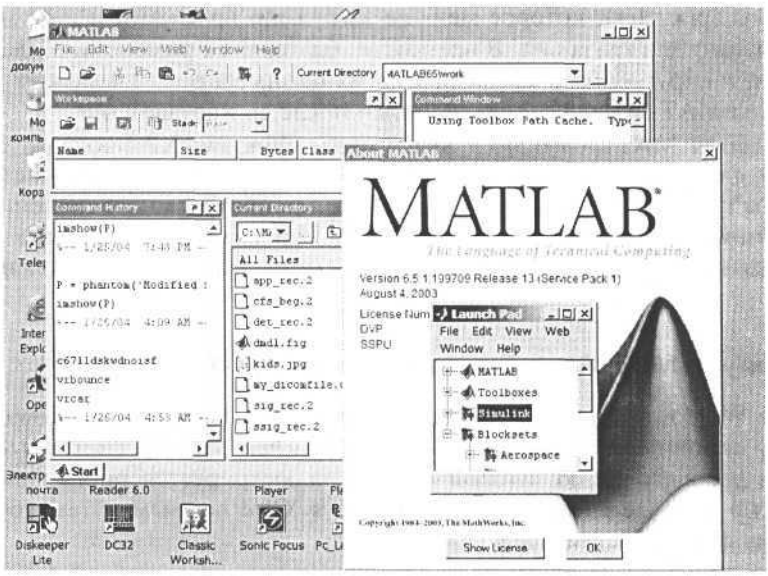

Рис. 1.6. Вид рабочего Windows XP при первом запуске системы MATLAB 6.5

Версия MATLAB 6.5 SP1 отличается повышенной стабильностью, хотя отдельные примеры (крайне редко) в ней не выполнялись из-за ошибок в M-файлах. Впрочем, речь идет о комплексе, поставленном автору легально от корпорации MathWorks и размещаемом на трех CD-ROM. Как будет работать пиратская копия системы, которую уже ухитрились разместить на одном диске, наши пользователи могут прочувствовать на своей «собственной шкуре».

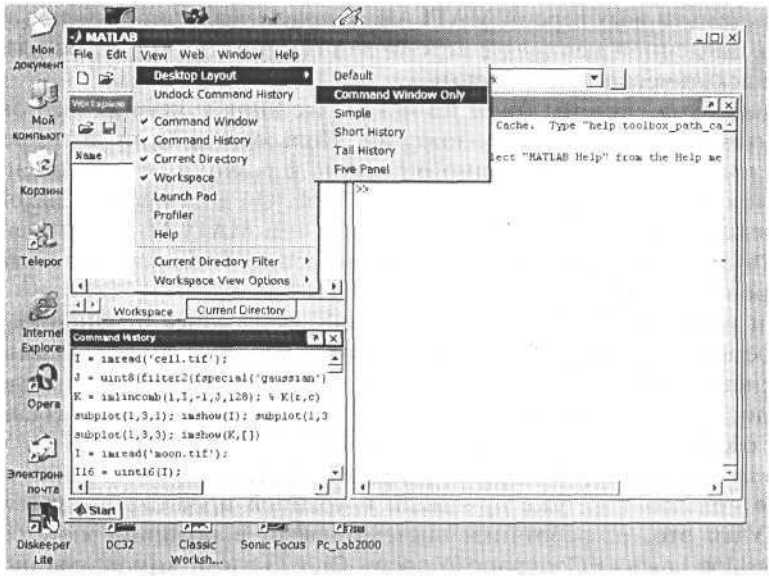

Рис. 1.7. Вид рабочего окна MATLAB 6.5 SP1 на рабочем столе Windows XP после настройки

Нетрудно заметить, что вид окон самой системы MATLAB один и тот же вне зависимости от примененной операционной системы. В связи с этим в дальнейшем мы не будем акцентировать внимание на том, в какой операционной системе используется MATLAB.

1.5.4. О работе в MATLAB 7

Особенности новейшей реализации системы MATLAB 7 детально описаны в первом томе данного четырехтомника. Планируется также описание вводного курса по этой системе в третьем томе. Напоминаем, что эта глава является вводным курсом по реализации MATLAB 6.5 SP1, отличительные особенности системы MATLAB 7 описаны в конце этой главы.

Данная книга посвящена пакетам расширения, ориентированным на математические расчеты и математическое моделирования. Все приводимые в ней примеры ориентированы на применение MATLAB в командном режиме работы, когда достаточно работать в основном окне системы MATLAB. Работа в нем совершенно идентична для любой версии MATLAB, начиная с MATLAB 5. В связи с этим описание в этом томе особенностей реализации MATLAB 7 лишено практического смысла. Мы рассмотрим особенности применения системы MATLAB на предшествующей реализации MATLAB 6.5 SP1, которая более известна, чем MATLAB 7, и в течение по крайней мере одного-двух лет будет находить более широкое применение.

1.6. Начало работы с MATLAB

1.6.1. Новый и старый облик системы MATLAB

Пользователи, уже имеющие опыт работы с системами MATLAB, будут приятно (а кое-кто, напротив, неприятно) удивлены новациями пользовательского ин-

терфейса системы новых версий MATLAB. В новой реализации пользовательского интерфейса, введенной начиная с версии MATLAB 6.0 (R12), не осталось и следа от прежней аскетической скромности.

Теперь вид окна системы (рис. 1.3 и 1.6) уже вполне отвечает канонам современного интерфейса Windows-приложений. Пользовательский интерфейс многооконный и имеет ряд средств прямого доступа к различным компонентам системы. Бросается в глаза новый пункт меню — Web. Он дает прямой выход в Интернет. Это средство себя не слишком оправдало, и в MATLAB 7 эта позиция меню уже отсутствует. Более удобным оказалась работа в Интернете с помощью обычных браузеров Internet Explorer, Opera и др. [41—43].

В панели инструментов добавлена позиция ввода ранее отмененной операции и меню просмотра файловой системы с кнопкой его открытия. Но самые решительные изменения произошли в левой части общего окна системы — здесь появились окна доступа к компонентам системы Launch Pad/Workspace (Панель запуска/Рабочая область) и окно Current Directory (текущей папки).

Надо прямо признать, что особой необходимости в этих новациях нет, поскольку старые пользователи уже привыкли к крайней простоте интерфейса систем MATLAB. Учтя это, разработчики системы ввели в позицию View (Вид) меню команду Desktop Layout ▸ Command Windows Only (Только командное окно). Стоит ее исполнить, как вид окна системы будет очень напоминать добрый старый интерфейс версий MATLAB 5.* — см. рис. 1.8.

Рис. 1.8. Упрощенный интерфейс системы MATLAB (для всех версий)

Если вы все же хотите вкусить прелести нового интерфейса, то исполните в той же позиции меню команду Desktop Layout ▸ Default (Интерфейс по умолчанию). Там же вы найдете и другие возможности модификации вида интерфейса системы MATLAB 6.0.

1.6.2. Операции строчного редактирования

При работе с MATLAB в командном режиме действует простейший строчный редактор. Его команды перечислены в табл. 1.1.

Таблица 1.1

Команды строчного редактора MATLAB

Комбинация клавиш	Назначение
→ или Ctrl + b	Перемещение курсора вправо на один символ
← или Ctrl + f	Перемещение курсора влево на один символ
Ctrl + → или Ctrl + r	Перемещение курсора вправо на одно слово
Ctrl + ← или Ctrl + l	Перемещение курсора влево на одно слово
Home или Ctrl + a	Перемещение курсора в начало строки
End или Ctrl + e	Перемещение курсора в конец строки
↑ и ↓ или Ctrl + p и Ctrl + n	Перелистывание предыдущих команд вверх или вниз для подстановки в строку ввода
Del или Ctrl + d	Стирание символа справа от курсора
← или Ctrl + h	Стирание символа слева от курсора
Ctrl + k	Стирание до конца строки
Esc	Очистка строки ввода
Ins	Включение/выключение режима вставки
PgUp	Перелистывание страниц сессии вверх
PgDn	Перелистывание страниц сессии вниз

Эти возможности кажутся примитивными, но позволяют пользователю быстро работать в стиле первых версий MATLAB для MS-DOS. Они обеспечивают важное свойство новых версий систем — их совместимость со старыми версиями в части преемственности навыков работы. Позже вы увидите, что в новых версиях есть вполне современный редактор со средствами отладки создаваемых документов — m-файлов.

Обратите особое внимание на применение клавиш ↑ и ↓. Они используются для подстановки после маркера строки ввода » ранее введенных строк, например для их исправления, дублирования или дополнения. При этом указанные клавиши обеспечивают перелистывание ранее введенных строк снизу вверх или сверху вниз. Такая возможность существует благодаря организации специального стека, хранящего строки с исполненными ранее командами.

1.6.3. Команды управления окном

Полезно сразу усвоить некоторые команды управления окном командного режима:

- `clc` — очищает экран и размещает курсор в левом верхнем углу пустого экрана;
- `home` — возвращает курсор в левый верхний угол окна;
- `echo <file_name> on` — включает режим вывода на экран текста Script-файла (файла-сценария);
- `echo <file_name> off` — выключает режим вывода на экран текста Script-файла;

- `echo <file_name>` — меняет режим вывода на противоположный;
- `echo on all` — включает режим вывода на экран текста всех m-файлов;
- `echo off all` — отключает режим вывода на экран текста всех m-файлов;
- `more on` — включает режим постраничного вывода (полезен при просмотре больших m-файлов);
- `more off` — отключает режим постраничного вывода (в этом случае для просмотра больших файлов надо пользоваться линейкой прокрутки).

В новых версиях MATLAB обе команды `clc` и `home` действуют аналогично — очищают экран и помещают курсор в левый верхний угол окна командного режима работы. Команды `echo` позволяют включать или выключать отображение текстов m-файлов при каждом обращении к ним. Как правило, отображение текста файлов сильно загромождает экран и часто не является необходимым. При больших размерах файлов начало их текста (листинга) убегает далеко за пределы области просмотра (текущего окна командного режима). Поэтому для просмотра длинных листингов файлов полезно включить постраничный вывод командой `more on`. Различие между m-файлами сценариев и функций мы обсудим позже.

1.6.4. MATLAB в режиме прямых вычислений

Система MATLAB создана таким образом, что любые (подчас весьма сложные) вычисления можно выполнять в режиме *прямых вычислений*, то есть без подготовки программы. Это превращает MATLAB в необычайно мощный калькулятор, который способен производить не только обычные для калькуляторов вычисления (например, выполнять арифметические операции и вычислять элементарные функции), но и операции с векторами и матрицами, комплексными числами, рядами и полиномами. Можно почти мгновенно задать и вывести графики различных функций — от простой синусоиды до сложной трехмерной фигуры.

Работа с системой в режиме прямых вычислений (или в командном режиме) носит диалоговый характер и происходит по правилу «задал вопрос, получил ответ». Пользователь набирает на клавиатуре вычисляемое выражение, редактирует его (если нужно) в командной строке и завершает ввод нажатием клавиши ENTER. В качестве примера на рис. 1.2 уже были показаны простейшие вычисления — вычисление выражения 2 + 2 и значения sin([1 2 3]) и exp(1).

Даже из таких простых примеров можно сделать некоторые поучительные выводы:

- для указания ввода исходных данных используется символ >>;
- данные вводятся с помощью простейшего строчного редактора;
- для блокировки вывода результата вычислений некоторого выражения после него надо установить знак ; (точка с запятой);
- если не указана переменная для значения результата вычислений, то MATLAB назначает такую переменную с именем `ans`;
- знаком присваивания является привычный математикам знак равенства =, а не комбинированный знак :=, как во многих других языках программирования и математических системах;
- встроенные функции (например `sin`) записываются строчными буквами, и их аргументы указываются в *круглых скобках*;
- результат вычислений выводится в строках вывода (без знака »);
- диалог происходит в стиле «задал вопрос — получил ответ».

Следующий пример (он показан на рис. 1.8) иллюстрирует применение системы MATLAB для выполнения простых векторных операций. В этом примере задается четырехэлементный вектор **V** со значениями элементов 1, 2, 3 и 4. Далее (сосредоточьте на этом внимание!) вычисляются функции синуса и экспоненты с аргументом в виде *вектора*, а не скаляра.

Две записи для вектора — `V=[1 2 3 4]` и `V=[1,2,3,4]` — являются идентичными. Таким образом, векторы задаются списком своих элементов, разделяемых пробелами или запятыми. Список заключается в квадратные скобки. Для выделения *n*-го элемента вектора `V` используется выражение `V(n)`. Оно задает соответствующую индексированную переменную.

В большинстве математических систем вычисление `sin(V)` и `exp(V)`, где `V` — вектор, сопровождалось бы выдачей ошибки, поскольку функции sin и exp должны иметь аргумент в виде скалярной величины. Однако MATLAB — матричная система, а вектор является разновидностью матрицы с размером 1 × n. Поэтому в нашем случае результат вычислений будет вектором того же размера, что и аргумент `V`, но элементы возвращаемого вектора будут синусами или экспонентами от элементов вектора `V`.

Матрица в MATLAB задается в виде ряда векторов, представляющих ее строки и заключенных в квадратные скобки. Для разделения элементов векторов используется пробел или запятая, а для отделения одного вектора от другого — точка с запятой. Для выделения отдельного элемента матрицы `M` используется выражение вида `M(j,i)`, где `M` — имя матрицы, `j` — номер строки и `i` — номер столбца.

Приведем примеры вычислений в командном режиме:

```
>> 2+3
ans =
 5
>> sin(1)
ans =
 0.8415
>> type sin
sin is a built-in function.
>> help sin
SINSine.
 SIN(X) is the sine of the elements of X.
Overloaded methods
 help sym/sin.m
>>
>> V=[1 2 3 4]
V =
  1  2  3  4
>> sin(V)
ans =
 0.8415 0.9093 0.1411 -0.7568
>> 3*V
ans =
  3  6  9  12
>> V^2
??? Error using ==> ^
Matrix must be square.
```

```
>> V.^2
ans =
  1   4   9   16
>> V+2
ans =
  3   4   5   6
>>
```

Обратите внимание на форму ответов при выполнении простых операций без указания переменной, которой присваивается результат. В таких случаях MATLAB сам назначает переменную ans, которой присваивается результат и значение которой затем выводится на экран.

Сравните эти записи с записями в реальных сессиях (рис. 1.2 и 1.8). Вы наверняка отметите, что они практически идентичны. Разве что в текстовых вариантах примеров для экономии бумаги, на которой печаталась эта книга, убраны пропуски между строками. Мы будем показывать представление сессий в виде прямых копий экрана только в том случае, когда это связано со спецификой проведения вычислений, например, когда они сопровождаются выводом графиков или демонстрацией элементов пользовательского интерфейса. В остальных случаях будет использоваться представление сессии прямо в тексте книги в представленном выше текстовом формате — основном для командного режима работы с системой MATLAB. При этом строки ввода будут отмечаться маркером ввода » в их начале. Ради компактности записи пустые строки будут опускаться.

1.6.5. О переносе строки в сессии

В некоторых случаях вводимое математическое выражение может оказаться настолько длинным, что для него не хватит одной строки. В этом случае часть выражения можно перенести на новую строку с помощью знака многоточия «...» (3 или более точек), например:

```
s = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 - ...
1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```

Этот прием может быть весьма полезным для создания наглядных документов, у которых предотвращается заход строк в невидимую область окна. Вообще говоря, максимальное число символов в одной строке командного режима — 4096, а в m-файле — не ограничено, но со столь длинными строками работать неудобно. В ранних версиях в одной строке было не более 256 символов.

1.7. Основные объекты MATLAB

1.7.1. Понятие о математическом выражении

Центральным понятием всех математических систем является *математическое выражение*. Оно задает то, что должно быть вычислено в численном (реже символьном) виде. Вот примеры простых математических выражений:

```
2+3
2.301*sin(x)
4+exp(3)/5
sqrt(y)/2
```

```
sin(pi/2)
```

Математические выражения строятся на основе чисел, констант, переменных, операторов, функций и разных спецзнаков. Ниже даются краткие пояснения сути этих понятий.

1.7.2. Действительные и комплексные числа

Число — простейший объект языка MATLAB, представляющий количественные данные. Числа можно считать константами, имена которых совпадают с их значениями. Числа используются в общепринятом представлении о них. Они могут быть целыми, дробными, с фиксированной и плавающей точкой. Возможно представление чисел в хорошо известном научном формате с указанием мантиссы и порядка числа. Ниже приводятся примеры представления чисел:

```
0
2
-3
2.301
0.00001
123.456e-24
-234.456e10
```

Как нетрудно заметить, в мантиссе чисел целая часть отделяется от дробной не запятой, а точкой, как принято в большинстве языков программирования. Для отделения порядка числа от мантиссы используется символ `e`. Знак «плюс» у чисел не проставляется, а знак «минус» у числа называют *унарным минусом*. Пробелы между символами в числах не допускаются.

Числа могут быть *комплексными*: $z = \mathrm{Re}(x) + \mathrm{Im}(x)*i$. Такие числа содержат действительную $\mathrm{Re}(z)$ и мнимую $\mathrm{Im}(z)$ части. Мнимая часть имеет множитель i или j, означающий корень квадратный из -1:

```
3i
2j
2+3i
-3.141i
-123.456+2.7e-3i
```

Функция `real(z)` возвращает действительную часть комплексного числа, $\mathrm{Re}(z)$, а функция `imag(z)` — мнимую, $\mathrm{Im}(z)$. Для получения модуля комплексного числа используется функция `abs(z)`, а для вычисления фазы — `angle(Z)`. Ниже даны простейшие примеры работы с комплексными числами:

```
>> i
ans =
  0 + 1.0000i
>> j
ans =
  0 + 1.0000i
>> z=2+3i
z =
  2.0000 + 3.0000i
>> abs(z)
```

```
ans =
  3.6056
>> real(z)
ans =
  2
>> imag(z)
ans =
  3
>> angle(z)
ans =
  0.9828
```

В MATLAB операции над числами обычно выполняются в формате, который принято считать форматом *с двойной точностью*. Такой формат удовлетворяет подавляющему большинству требований к численным расчетам, но совершенно не подходит для символьных вычислений с произвольной (абсолютной) точностью. Символьные вычисления MATLAB может выполнять с помощью специального пакета расширения Symbolic Math Toolbox [12].

1.7.3. Форматы чисел

По умолчанию MATLAB выдает числовые результаты в *нормализованной форме* с четырьмя цифрами после десятичной точки и одной до нее. Многих такая форма представления не всегда устраивает. Поэтому при работе с числовыми данными можно задавать различные *форматы* представления чисел. Однако в любом случае все вычисления проводятся с предельной, так называемой *двойной*, точностью. Для установки формата представления чисел используется команда

```
>> format name
```

где `name` — имя формата. Для числовых данных `name` может быть следующим сообщением:

- `short` — короткое представление в фиксированном формате (5 знаков);
- `short e` — короткое представление в экспоненциальном формате (5 знаков мантиссы и 3 знака порядка);
- `long` — длинное представление в фиксированном формате (15 знаков);
- `long e` — длинное представление в экспоненциальном формате (15 знаков мантиссы и 3 знака порядка);
- `hex` — представление чисел в шестнадцатеричной форме;
- `bank` — представление для денежных единиц.

Для иллюстрации различных форматов рассмотрим вектор, содержащий два элемента-числа:

```
x=[4/3 1.2345e-6]
```

В различных форматах их представления будут иметь следующий вид:

```
format short    1.3333                  0.0000
format short e  1.3333E+000             1.2345E-006
format long     1.333333333333338       0.000001234500000
format long e   1.333333333333338E+000  1.234500000000000E-006
format bank     1.33                    0.00
```

Задание формата сказывается только на *форме вывода* чисел. Вычисления все равно происходят в формате двойной точности, а ввод чисел возможен в любом удобном для пользователя виде.

1.7.4. Константы и системные переменные

Константа — это предварительно определенное числовое или символьное значение, представленное уникальным именем. Числа (например, 1, −2 и 1.23) являются безымянными *числовыми константами*.

Другие виды констант в MATLAB принято назвать *системными переменными*, поскольку, с одной стороны, они задаются системой при ее загрузке, а с другой — могут переопределяться. Основные системные переменные, применяемые в системе MATLAB, указаны ниже:

`i` или `j` — мнимая единица (корень квадратный из −1);
`pi` — число $\pi = 3.1415926...$;
`eps` — погрешность операций над числами с плавающей точкой (2^{-52});
`realmin` — наименьшее число с плавающей точкой (2^{-1022});
`realmax` — наибольшее число с плавающей точкой (2^{1023})
`inf` — значение машинной бесконечности;
`ans` — переменная, хранящая результат последней операции и обычно вызывающая его отображение на экране дисплея;
`NaN` — указание на нечисловой характер данных (Not-a-Number).

Вот примеры применения системных переменных:

```
>> 2*pi
ans =
  6.2832
>> eps
ans =
  2.2204e-016
>> realmin
ans =
  2.2251e-308
>> realmax
ans =
  1.7977e+308
>> 1/0
Warning: Divide by zero.
ans =
 Inf
>> 0/0
Warning: Divide by zero.
ans =
 NaN
```

Как отмечалось, системные переменные могут *переопределяться*. Можно задать системной переменной `eps` иное значение, например `eps=0.0001`. Однако важно то, что их значения по умолчанию задаются сразу после загрузки системы. Поэтому неопределенными, в отличие от обычных переменных, системные переменные не могут быть никогда.

Символьная константа — это цепочка символов, заключенных в апострофы, например:

```
'Hello my friend!'
'Привет'
'2+3'
```

Если в апострофы помещено математическое выражение, то оно *не вычисляется* и рассматривается просто как цепочка символов. Так что `'2+3'` не будет возвращать число 5. Однако с помощью специальных функций преобразования символьные выражения могут быть преобразованы в вычисляемые. Соответствующие функции преобразования будут рассмотрены в дальнейшем.

1.7.5. Текстовые комментарии

Поскольку MATLAB используется для достаточно сложных вычислений, важное значение имеет наглядность их описания. Она достигается, в частности, с помощью текстовых комментариев. *Текстовые комментарии* вводятся с помощью символа `%`, например так:

```
%It is factorial function
```

ПРИМЕЧАНИЕ *В каждой большой программе есть свои «ложки дегтя». В MATLAB 6.* такой «ложкой дегтя» является перевод строки при вводе буквы «с» русского алфавита в командной строке. В итоге ввод комментариев в командной строке на русском языке превращается в проблему. Пока автор может порекомендовать заменять русское «с» на английское, что на виде текстового комментария никак не сказывается. Не рекомендуется вводить русскоязычные комментарии и в тесты m-файлов при подготовке их в редакторе/отладчике (он будет описан позже). Это нередко делает программы неработоспособными. Жаловаться тут бесполезно — MATLAB англоязычный продукт и официальной, локализованной под Россию версии этой системы пока нет.*

Обычно первые строки m-файлов служат для описания их назначения, которое выводится на экран дисплея после команды

```
>> help Имя_файла
```

Считается правилом хорошего тона вводить в m-файлы достаточно подробные текстовые комментарии. Без таких комментариев даже разработчик программных модулей быстро забывает о сути собственных решений. В текстовых комментариях и в символьных константах могут использоваться буквы русского алфавита — при условии, что установлены содержащие эти буквы наборы шрифтов (см. примечание выше).

1.7.6. Переменные и присваивание им значений

Переменные — это имеющие имена объекты, способные хранить некоторые, обычно разные по значению данные. В зависимости от этих данных переменные могут быть числовыми или символьными, векторными или матричными.

В системе MATLAB можно задавать переменным определенные значения. Для этого используется операция *присваивания*, вводимая знаком равенства =

```
Имя_переменной = Выражение
```

Типы переменных заранее не декларируются. Они определяются выражением, значение которого присваивается переменной. Так, если это выражение — вектор или матрица, то переменная будет векторной или матричной.

Имя переменной (ее *идентификатор*) может содержать сколько угодно символов, но запоминается и идентифицируется только 31 начальный символ. Имя любой переменной не должно совпадать с именами других переменных, функций и процедур системы, то есть оно должно быть уникальным. Имя должно начинаться с буквы, может содержать буквы, цифры и символ подчеркивания _. Недопустимо включать в имена переменных пробелы и специальные знаки, например +, -, *, / и т. д., поскольку в этом случае правильная интерпретация выражений становится невозможной.

Желательно использовать содержательные имена для обозначений переменных, например `speed_1` для переменной, обозначающей скорость первого объекта. Переменные могут быть обычными и *индексированными*, то есть элементами векторов или матриц (см. выше). Могут использоваться и *символьные* переменные, причем символьные значения заключаются в апострофы, например `s='Demo'`.

1.7.7. Уничтожение определений переменных

В памяти компьютера переменные занимают определенное место, называемое *рабочей областью* (workspace). Для очистки рабочей области используется функция `clear` в разных формах, например:

`clear` — уничтожение определений всех переменных;
`clear x` — уничтожение определения переменной x;
`clear a, b, c` — уничтожение определений нескольких переменных.

Уничтоженная (стертая в рабочей области) переменная становится неопределенной. Использовать неопределенные переменные нельзя, и такие попытки будут сопровождаться выдачей сообщений об ошибке. Приведем примеры задания и уничтожения переменных:

```
>> x=2*pi
x =
   6.2832
>> V=[1 2 3 4 5]
V =
   1   2   3   4   5
>> MAT
??? Undefined function or variable 'MAT'.
>> MAT=[1 2 3 4; 5 6 7 8]
MAT =
   1   2   3   4
   5   6   7   8
>> clear V
>> V
??? Undefined function or variable 'V'.
>> clear
>> x
??? Undefined function or variable 'x'.
>> M
??? Undefined function or variable 'M'.
```

Обратите внимание на то, что сначала выборочно стерта переменная V, а затем командой clear без параметров стерты все остальные переменные.

1.7.8. Операторы и функции

Оператор — это специальное обозначение для определенной операции над данными — *операндами*. Например, простейшими арифметическими операторами являются знаки суммы +, вычитания –, умножения * и деления /. Операторы используются совместно с операндами. Например, в выражении 2 + 3 знак + является оператором сложения, а числа 2 и 3 — операндами.

Следует отметить, что большинство операторов относится к матричным операциям, что может служить причиной серьезных недоразумений. Например, операторы умножения * и деления / вычисляют произведение и частное от деления двух массивов, векторов или матриц. Есть ряд специальных операторов, например оператор \ означает деление *справа налево*, а операторы .* и ./ означают, соответственно, *поэлементное* умножение и *поэлементное* деление массивов.

Следующие примеры поясняют сказанное на примере операций с векторами:

```
>> V1=[2 4 6 8]
V1 =
  2  4  6  8
>> V2=[1 2 3 4]
V2 =
  1  2  3  4
>> V1/V2
ans =
  2
>> V1.*V2
ans =
 2  8  18  32
>> V1./V2
ans =
  2  2  2  2
```

Полный список операторов можно получить, используя команду

```
>> help ops
```

Постепенно мы рассмотрим все операторы системы MATLAB и обсудим особенности их применения. А пока приведем только часть полного списка операторов, содержащую арифметические операторы:

```
>> help ops
Operators and special characters.
Arithmetic operators.
Plus - Plus  +
Uplus - Unary plus  +
Minus - Minus  -
Uminus - Unary minus  -
Mtimes - Matrix multiply  *
times - Array multiply  .*
mpower - Matrix power  ^
power - Array power  .^
```

```
mldivide - Backslash or left matrix divide  \
mrdivide - Slash or right matrix divide  /
ldivide - Left array divide  .\
rdivide - Right array divide  ./
kron - Kronecker tensor product
```

Функции — это имеющие уникальные имена объекты, выполняющие определенные преобразования своих аргументов и при этом возвращающие результаты этих преобразований. *Возврат результата* — отличительная черта функций. При этом результат вычисления функции с одним выходным параметром подставляется на место ее вызова, что позволяет использовать функции в математических выражениях, например функцию sin в `2*sin(pi/2)`.

Функции в общем случае имеют список аргументов (параметров), заключенный в круглые скобки. Например, функция Бесселя записывается как `bessel(NU,X)`. В данном случае список параметров содержит два аргумента — `NU` в виде скаляра и `X` в виде вектора. Многие функции допускают ряд форм записи, отличающихся списком параметров. Если функция возвращает несколько значений, то она записывается в виде

```
[Y1, Y2,...]=func(X1, X2,...)
```

где `Y1, Y2,...` — список *выходных* параметров и `X1, X2,...` — список *входных* аргументов (параметров).

Со списком элементарных функций можно ознакомиться, выполнив команду `help elfun`, а со списком специальных функций — с помощью команды `help specfun`. Функции могут быть *встроенными* (внутренними) и *внешними*, или *m-функциями*. Так, встроенными являются наиболее распространенные элементарные функции, например, `sin(x)` и `exp(y)`, тогда как функция `sinh(x)` является внешней функцией. Внешние функции содержат свои определения в m-файлах. Задание таких функций с помощью специального редактора m-файлов мы рассмотрим в главе 2. Встроенные функции хранятся в откомпилированном ядре системы MATLAB, в силу чего они выполняются предельно быстро.

1.7.9. Применение оператора : (двоеточие)

Очень часто необходимо произвести формирование упорядоченных числовых последовательностей. Такие последовательности нужны для создания векторов или значений абсциссы при построении графиков. Для этого в MATLAB используется оператор : (двоеточие):

```
Начальное_значение:Шаг:Конечное_значение
```

Данная конструкция порождает возрастающую последовательность чисел, которая начинается с начального значения, идет с заданным шагом и завершается конечным значением. Если `Шаг` не задан, то он принимает значение 1. Если конечное значение указано меньшим, чем начальное значение, — выдается сообщение об ошибке. Примеры применения оператора : даны ниже:

```
>> 1:5
ans =
     1     2     3     4     5
>> i=0:2:10
i =
```

```
    0     2     4     6     8    10
>> j=10:-2:2
j =
    1    0     8     6     4     2
>> V=0:pi/2:2*pi;
>> V
V =
       0    1.5708    3.1416    4.7124    6.2832
>> X=1:-.2:0
X =
    1.0000    0.8000    0.6000    0.4000    0.2000 0
>> 5:2
ans =
    Empty matrix: 1-by-0
```

Как отмечалось, принадлежность MATLAB к матричным системам вносит коррективы в назначение операторов и приводит, при неумелом их использовании, к казусам. Рассмотрим следующий характерный пример:

```
>> x=0:5
x =
    0     1     2     3     4     5
>> cos(x)
ans =
    1.0000    0.5403   -0.4161   -0.9900   -0.6536    0.2837
>> sin(x)/x
ans =
    -0.0862
```

Вычисление массива косинусов здесь прошло корректно. А вот вычисление массива значений функции $\sin(x)/x$ дает неожиданный, на первый взгляд, эффект — вместо массива с шестью элементами вычислено единственное значение!

Причина «парадокса» здесь в том, что оператор / вычисляет отношение двух *матриц*, *векторов* или *массивов*. Если они одной размерности, то результат будет одним числом, что в данном случае и выдала система. Чтобы действительно получить вектор значений $\sin(x)/x$, надо использовать специальный оператор *поэлементного* деления массивов — ./. Тогда будет получен массив чисел:

```
>> sin(x)./x
Warning: Divide by zero.
ans =
 NaN 0.8415  0.4546  0.0470  -0.1892  -0.1918
```

Впрочем, и тут без особенностей не обошлось. Так, при $x = 0$ значение $\sin(x)/x$ дает устранимую неопределенность вида $0/0 = 1$. Однако, как и всякая численная система, MATLAB классифицирует попытку деления на 0 как ошибку и выводит соответствующее предупреждение. А вместо ожидаемого численного значения выводится символьная константа NaN, означающая, что неопределенность 0/0 — это все же не обычное число.

Выражения с оператором : могут использоваться в качестве аргументов функций для получения множественных их значений. Например, в приводимом ниже примере вычислены функции Бесселя порядка от 0 до 5 со значением аргумента 0.5:

```
>> bessel(0:1:5,1/2)
ans =
 0.9385  0.2423  0.0306  0.0026  0.0002  0.0000
```

А в следующем примере вычислено шесть значений функции Бесселя нулевого порядка для значений аргумента от 0 до 5 с шагом 1:

```
>> bessel(0,0:1:5)
ans =
 1.0000  0.7652  0.2239  -0.2601  -0.3971  -0.1776
```

Таким образом, оператор : является весьма удобным средством задания регулярной последовательности чисел. Он широко используется при работе со средствами построения графиков. В дальнейшем мы расширим представление о возможностях этого оператора.

1.7.10. Функции пользователя

Хотя в ядро MATLAB последних версий встроено уже около 900 операторов и функций, пользователю всегда может понадобиться та или иная функций, простая или сложная, отсутствующая в ядре. Язык программирования систем MATLAB предоставляет ряд эффективных возможностей для задания функций пользователя.

Одна из таких возможностей заключается в применении функции `inline`, аргументом которой надо в апострофах задать выражение, задающее функцию одной или нескольких переменных. В приведенном ниже примере задана функция двух переменных — суммы квадратов sin(x) и cos(y):

```
>> sc2=inline('sin(x).^2+cos(y)^.2')
sc2 =
Inline function:
sc2(x,y) = sin(x).^2+cos(y).^2
```

Можно также задавать свои функции в виде m-файлов. Подробно это описано в главе 10. Например, можно в окне редактора m-файлов (открывается командой New в меню File) создать m-файл с именем sc2 и листингом:

```
function y=sc2(x,y)
y=sin(x).^2+cos(y).^2
```

Записав его на диск, можно командой `type sc2` вывести листинг созданной функции:

```
>> type sc2
function y=sc2(x,y)
y=sin(x).^2+cos(y).^2
```

Обращение к функции, созданной описанными методами, задается как sc2(x,y), где на место x и y подставляются значения переменных — аргументов функции пользователя. Например:

```
>> sc2(1,2)
ans =
    0.8813
>> sc2(2,1)
y =
```

```
      1.1187
ans =
      1.1187
```

Можно также создать так называемую handle-функцию с помощью оператора @:

```
>> fh=@sc2;
```

К такой функции можно обращаться с помощью функции исполнения функций `feval(fh,x,y)`:

```
>> feval(fh,1,2)
y =
    0.8813
ans =
    0.8813
>> feval(fh,2,1)
y =
    1.1187
ans =
    1.1187
```

1.7.11. Сообщения об ошибках и исправление ошибок

Важное значение при диалоге с системой MATLAB имеет *диагностика ошибок*. Вряд ли есть пользователь, помнящий точное написание тысяч операторов и функций, входящих в систему MATLAB и в пакеты прикладных программ. Поэтому никто не застрахован от ошибочного написания математических выражений или команд. MATLAB диагностирует вводимые команды и выражения и выдает соответствующие сообщения об ошибках или предупреждения. Пример вывода сообщения об ошибке (деление на 0) только что приводился.

Рассмотрим еще ряд примеров. Введем, к примеру, ошибочное выражение

```
>> sqr(2)
```

и нажмем клавишу ENTER. Система сообщит об ошибке:

```
??? Undefined function or variable 'sqr'.
```

Это сообщение говорит о том, что не определена переменная или функция, и указывает, какая именно — `sqr`. В данном случае, разумеется, можно просто набрать правильное выражение. Однако в случае громоздкого выражения лучше воспользоваться редактором. Для этого достаточно нажать клавишу ↓ для перелистывания предыдущих строк. В результате в строке ввода появится выражение

```
>> sqr(2)
```

с курсором в его конце. В версии MATLAB 6.* можно теперь нажать клавишу Tab. Система введет подсказку, анализируя уже введенные символы. Из предложенных системой трех операторов выбираем sqrt. Теперь с помощью клавиши ↓ вновь выбираем нужную строку и, пользуясь клавишей ←, устанавливаем курсор после буквы r. Теперь нажмем клавишу T, а затем клавишу ENTER. Выражение примет следующий вид:

```
>> sqrt(2)
ans =
 1.4142
```

Если бы был только один вариант окончания введенных символов, то после нажатия клавиши Tab система бы закончила наш ввод без перевода строки. Вычисления дают ожидаемый результат — значение квадратного корня из двух.

В системе MATLAB внешние определения используются точно так же, как и встроенные функции и операторы. Никаких дополнительных указаний на их применение делать не надо. Достаточно лишь позаботиться о том, чтобы используемые определения действительно существовали в виде файлов с расширением .m. Впрочем, если вы забудете об этом или введете имя несуществующего определения, то система отреагирует на это звуковым сигналом (звонком) и выводом сообщения об ошибке:

```
>> hsin(1)
??? Undefined function or variable 'hsin'.
>> sinh(1)
ans =
 1.1752
```

В этом примере мы забыли (нарочно), какое имя имеет внешняя функция, вычисляющая гиперболический синус. Система подсказала, что функция или переменная с именем `hsin` не определена ни как внутренняя, ни как m-функция. Зато далее мы видим, что функция с именем `sinh` есть в составе функций системы MATLAB — она задана в виде M-функции. Между тем в последнем примере мы не давали системе никаких указаний на то, что следует искать именно внешнюю функцию! И это вычисление прошло так же просто, как вычисление встроенной функции, такой как `sin`. Разумеется, скорость вычислений по внешним определениям несколько ниже, чем по встроенным функциям или операторам.

При этом вычисления происходят следующим образом: вначале система быстро определяет, имеется ли введенное слово среди служебных слов системы. Если да, то нужные вычисления выполняются сразу, если нет, система ищет m-файл с соответствующим именем на диске. Если файла нет, то выдается сообщение об ошибке и вычисления останавливаются. Если же файл найден, он загружается с жесткого диска в память машины и исполняется.

Иногда в ходе вывода результатов вычислений появляется сокращение `NaN` (от слов Not a Number — не число). Оно обозначает неопределенность, например вида `0/0` или `Inf/Inf`, где `Inf` — системная переменная со значением машинной бесконечности. Могут появляться и различные предупреждения об ошибках (на английском языке). Например, при делении на 0 конечного числа появляется предупреждение «`Warning: Devide by Zero.`» («Внимание: деление на ноль»). Диапазон чисел, представимых в системе, лежит от 10^{-308} до 10^{+308}.

Вообще говоря, в MATLAB надо отличать *предупреждение* об ошибке от *сообщения* о ней. *Предупреждения* (обычно после слова `Warning`) не останавливают вычисления и лишь предупреждают пользователя о том, что диагностируемая ошибка способна повлиять на ход вычислений. *Сообщение* об ошибке (после знаков `???`) останавливает вычисления.

1.8. Формирование векторов и матриц

1.8.1. Особенности задания векторов и матриц

MATLAB — система, специально предназначенная для проведения сложных вычислений с векторами, матрицами и массивами — в том числе многомерными.

При этом она по умолчанию предполагает, что каждая заданная переменная — это вектор, матрица или массив. Все определяется конкретным значением переменной. Например, если задано `X=1`, то это значит, что `X` — это вектор с единственным элементом, имеющим значение 1. Если надо задать вектор из трех элементов, то их значения следует перечислить в квадратных скобках, разделяя пробелами. Так, например, присваивание

```
>> V=[1 2 3]
V =
 1  2  3
```

задает вектор `V`, имеющий три элемента со значениями 1, 2 и 3. После ввода вектора система выводит его на экран дисплея.

Задание матрицы требует указания нескольких строк. Для разграничения строк используется знак `;` (точка с запятой). Этот же знак в конце ввода предотвращает вывод матрицы или вектора (и вообще любой операции) на экран дисплея. Так, ввод

```
>> M=[1 2 3; 4 5 6; 7 8 9];
```

задает квадратную матрицу, которую можно вывести:

```
>> M
M =
  1  2  3
  4  5  6
  7  8  9
```

Возможен ввод элементов матриц и векторов в виде арифметических выражений, содержащих любые доступные системе функции, например:

```
>> V= [2+2/(3+4) exp(5) sqrt(10)];
>> V
V =
 2.2857  148.4132  3.1623
```

Для указания отдельного элемента вектора или матрицы используются выражения вида `V(i)` или `M(i, j)`. Например, если задать

```
>> M(2, 2)
ans =
     5
```

то результат будет равен 5. Если нужно присвоить элементу `M(i, j)` новое значение `x`, следует использовать выражение

```
M(ij)=x
```

Например, если элементу `M(2, 2)` надо присвоить значение 10, следует записать

```
>> M(2, 2)=10
```

Вообще говоря, в тексте программ MATLAB лучше не использовать буквы i и j как индексы, так как i и j — обозначение квадратного корня из −1. Но можно использовать I и J.

Выражение M(i) с одним индексом дает доступ к элементам матрицы, развернутым в один столбец. Такая матрица образуется из исходной, если подряд выписать ее столбцы. Следующий пример поясняет такой доступ к элементам матрицы M:

```
>> M=[1 2 3; 4 5 6; 7 8 9]
M =
 1  2  3
 4  5  6
 7  8  9
>> M(2)
ans =
 4
>> M(8)
ans =
 6
>> M(9)
ans =
 9
>> M(5)=100;
>> M
M =
 1  2   3
 4  100 6
 7  8   9
```

Возможно задание векторов и матриц с комплексными элементами, например:

```
>> i=sqrt(-1);
>> CM = [1 2; 3 4] + i*[5 6; 7 8]
```

или

```
>> CM = [1+5*i 2+6*i; 3+7*i 4+8*i]
```

Это создает матрицу:

```
CM =
1.0000 + 5.0000i   2.0000 + 6.0000i
3.0000 + 7.0000i   4.0000 + 8.0000i
```

Наряду с операциями над отдельными элементами матриц и векторов система позволяет производить операции умножения, деления и возведения в степень сразу над всеми элементами, то есть над массивами. Для этого перед знаком операции ставится точка. Например, оператор * означает умножение для векторов или матриц, а оператор .* — поэлементное умножение всех элементов массива. Так, если M — матрица, то M.*2 даст матрицу, все элементы которой умножены на скаляр — число 2. Впрочем, для умножения матрицы на скаляр оба выражения — M*2 и M.*2 — оказываются эквивалентными.

Имеется также ряд особых функций для задания векторов и матриц. Например, функция magic(n) задает магическую матрицу размера $n \times n$, у которой сумма всех столбцов, всех строк и даже диагоналей равна одному и тому же числу:

```
>> M=magic(4)
M =
```

```
  16    2    3    13
   5    11   10   8
   9    7    6    12
   4    14   15   1
>> sum(M)
ans =
   34   34   34   34
>> sum(M')
ans =
   34   34   34   34
>> sum(diag(M))
ans =
   34
>> M(1,2)+M(2,2)+M(3,2)+M(4,2)
ans =
   34
```

Уже сама по себе возможность создания такой матрицы с помощью простой функции `magic` заинтересует любителей математики. Но векторных и матричных функций в системе множество, и мы их детально рассмотрим в дальнейшем. Напомним, что для стирания переменных из рабочей области памяти служит команда `clear`.

1.8.2. Объединение малых матриц в большую

Описанный способ задания матриц позволяет выполнить операцию *конкатенации* — объединения малых матриц в большую. Например, создадим вначале магическую матрицу размера 3 × 3:

```
>> A=magic(3)
A =
   8   1   6
   3   5   7
   4   9   2
```

Теперь можно построить матрицу, содержащую четыре матрицы:

```
>> B=[A A+16;A+32 A+16]
B =
   8   1   6   24   17   22
   3   5   7   19   21   23
   4   9   2   20   25   18
  40  33  38   24   17   22
  35  37  39   19   21   23
  36  41  34   20   25   18
```

Полученная матрица имеет уже размер 6 × 6. Вычислим сумму ее столбцов:

```
>> sum(B)
ans =
   126   126   126   126   126   126
```

Любопытно, что она одинакова для всех столбцов. А для вычисления суммы строк используем команду

```
>> sum(B.')
ans =
 78  78  78  174  174  174
```

Здесь запись B.' означает транспонирование матрицы B, то есть замену строк столбцами. На этот раз сумма оказалась разной. Это отвергает изначально возникшее предположение, что матрица B тоже является магической. Для истинно магической матрицы суммы столбцов и строк должны быть одинаковыми:

```
>> D=magic(6)
D =
 35   1   6   26   19   24
 3    32  7   21   23   25
 31   9   2   22   27   20
 8    28  33  17   10   15
 30   5   34  12   14   16
 4    36  29  13   18   11
>> sum(D)
ans =
 111   111   111   111   111   111
>> sum(D.')
ans =
 111   111   111   111   111   111
```

Более того, для магической матрицы одинаковой является и сумма элементов по основным диагоналям (главной диагонали и главной антидиагонали).

1.8.3. Удаление столбцов и строк матриц

Для формирования матриц и выполнения ряда матричных операций возникает необходимость удаления отдельных столбцов и строк матрицы. Для этого используются пустые квадратные скобки [ ]. Проделаем это с матрицей M:

```
>> M=[1 2 3; 4 5 6; 7 8 9]
M =
 1  2  3
 4  5  6
 7  8  9
```

Удалим второй столбец используя оператор : (двоеточие):

```
>> M(:,2)=[ ]
M =
 1  3
 4  6
 7  9
```

А теперь, используя оператор : (двоеточие), удалим вторую строку:

```
>> M(2,:)=[ ]
M =
  1  3
  7  9
```

1.9. Операции с рабочей областью и текстом сессии

1.9.1. Дефрагментация рабочей области

По мере задания одних переменных и стирания других рабочая область перестает быть непрерывной и начинает содержать «дыры» и всякий «мусор». Это рано или поздно может привести к ухудшению работы системы или даже к нехватке оперативной памяти. Подобная ситуация становится возможной, если вы работаете с достаточно большими массивами данных.

Во избежание непроизводительных потерь памяти при работе с объемными данными (а векторы, матрицы и массивы относятся к таковым) следует использовать команду `pack`, осуществляющую дефрагментацию рабочей области. Эта команда переписывает все определения рабочей области на жесткий диск, очищает рабочую область и затем заново считывает все определения без «дыр» и «мусора» в рабочую область.

1.9.2. Сохранение рабочей области сессии

Переменные и определения новых функций в системе MATLAB хранятся в особой области памяти, именуемой рабочей областью. MATLAB позволяет сохранять значения переменных в виде бинарных файлов с расширением .mat. Для этого служит команда `save`, которая может использоваться в ряде форм:

`save fname` — записывается рабочая область всех переменных в файле бинарного формата с именем fname.mat;
`save fname X` — записывает только значение переменной `X`;
`save fname X Y Z` — записывает значения переменных `X`, `Y` и `Z`.

После параметров команды `save` можно указать ключи, уточняющие формат записи файлов:

`-mat` — двоичный MAT-формат, используемый по умолчанию;
`-ascii` — ASCII-формат единичной точности (8 цифр);
`-ascii -double` — ASCII-формат двойной точности (16 цифр);
`-ascii -double -tabs` — формат с разделителем и метками табуляции;
`V4` — запись MAT-файла в формате версии MATLAB 4;
`-append` — добавление в существующий MAT-файл.

Возможно использование слова `save` и в формате функции, а не команды, например:

```
save('fname','var1','var2')
```

В этом случае имена файлов и переменных задаются строковыми константами.

Следует отметить, что возможности сохранения всего *текста сессии*, формируемой в командном режиме, команда `save` не дает. И не случайно! Дело в том, что сессия является результатом проб и ошибок, и ее текст наряду с правильными определениями содержит сообщения об ошибках, переопределения функций и переменных и много прочей «шелухи». Необходимости сохранять такое «творчество» обычно нет. А если есть — для этого служит команда `diary`, описанная чуть ниже. Вы можете также воспользоваться редактором и отладчиком, которые позволяют (после отладки программы) получить документ в корректной форме без синтаксических и иных ошибок и сохранить его в виде файла с расширением .m текстового формата.

1.9.3. Ведение дневника

Мы отмечали, что сессии не записываются на диск стандартной командой `save`. Однако, если такая необходимость есть, можно воспользоваться специальной командой для ведения так называемого *дневника* сессии:

`diary file_name` — ведет запись на диск всех команд в строках ввода и полученных результатов в виде текстового файла с указанным именем;
`diary off` — приостанавливает запись в файл;
`diary on` — вновь начинает запись в файл.

Таким образом, чередуя команды `diary off` и `diary on`, можно сохранять нужные фрагменты сессии в их формальном виде. Команду `diary` можно задать и в виде функции `diary('file')`, где строка `'file'` задает имя файла. Следующий пример поясняет технику применения команды `diary`:

```
>> diary myfile.m
>> 1+2
ans =
 3
>> diary off
>> 2+3
ans =
 5
>> diary on
>> sin(1)
ans =
 0.8415
>> diary off
```

Нетрудно заметить, что в данном примере первая операция — 1 + 2 = 3 — будет записана в файл myfile.m, вторая — 2 + 3 = 5 — не будет записана, третья операция — sin(1) = 0.8415 — снова будет записана. Таким образом, будет создан Script-файл следующего вида:

```
1+2
ans =
  3
diary off
sin(1)
ans =
  0.8415
diary off
```

Он приведен в том виде, как записан, то есть с пробелами между строк. Одна из распространенных ошибок начинающих пользователей — попытка запустить подобный файл в командной строке указанием его имени:

```
>> myfile
??? ans =
Missing variable or function.
Error in ==> C:\MATLAB\bin\myfile.m
On line 3 ==> ans =
```

Обычно это приводит к ошибкам, так как данный файл — это просто текстовая запись команд и результатов их выполнения, не проверяемая на корректность и содержащая ряд строк, ошибочных с позиций синтаксиса языка программирования MATLAB — например, выражения `ans =`. Зато команда `type` позволяет просмотреть текст такого файла со всеми записанными действиями:

```
>> type myfile
1+2
ans =
 3
diary off
sin(1)
ans =
 0.8415
diary off
```

Во избежание отмеченных казусов рекомендуется записывать файл с расширением, отличным от .m, например .txt. Это позволит встраивать подобные текстовые файлы дневника сессии в документы, содержащие ее описание.

1.9.4. Загрузка рабочей области сессии

Для загрузки рабочей области ранее проведенной сессии (если она была сохранена) можно использовать команду `load`:

`load fname ...` — загрузка ранее сохраненных в файле fname.mat определений со спецификациями на месте многоточия, подобными описанным для команды `save` (включая ключ `-mat` для загрузки файлов с расширением .mat обычного бинарного формата, используемого по умолчанию);

`load('fname',...)` — загрузка файла fname.mat в форме функции.

Если команда (или функция) `load` используется в ходе проведения сессии, то произойдет замена текущих значений переменных теми значениями, которые были сохранены в считываемом MAT-файле.

Для задания имен загружаемых файлов может использоваться знак `*`, означающий загрузку всех файлов с определенными признаками. Например, `load demo*.mat` означает загрузку всех файлов с началом имени demo, например demo1, demo2, demoa, demob и т. д. Имена загружаемых файлов можно формировать с помощью операций над строковыми выражениями.

1.9.5. Работа с редактором/отладчиком M-файлов

Любая последовательность команд может быть оформлена в виде M-файлов. Для этого можно использовать любой текстовый редактор, поддерживающий формат ASCII-файлов — например WordPad или Word. Для профессиональной подготовки, редактирования и отладки M-файлов служит специальный редактор системы MATLAB. Он выполнен как типичное приложение под Windows. Редактор можно вызвать командой `edit` из командной строки MATLAB или командой New ▸ M-file из меню File. После этого в окне редактора можно создавать свой файл, пользоваться средствами его отладки и запуска. Для запуска файла его необходимо записать на диск, используя команду Save as в меню File редактора.

На рис. 1.9 приведен пример работы с редактором/отладчиком. Задано вычисление выражения 2 + 3 и построение графика синусоиды. Окно с текстом этого простого файла в окне редактирования и отладки представлено справа на фоне окна MATLAB. Подготовленный текст файла (это простейшая и первая наша программа на языке программирования MATLAB) надо записать на диск, например в виде файла demo. Для этого используется обычная команда Save As в окне редактора.

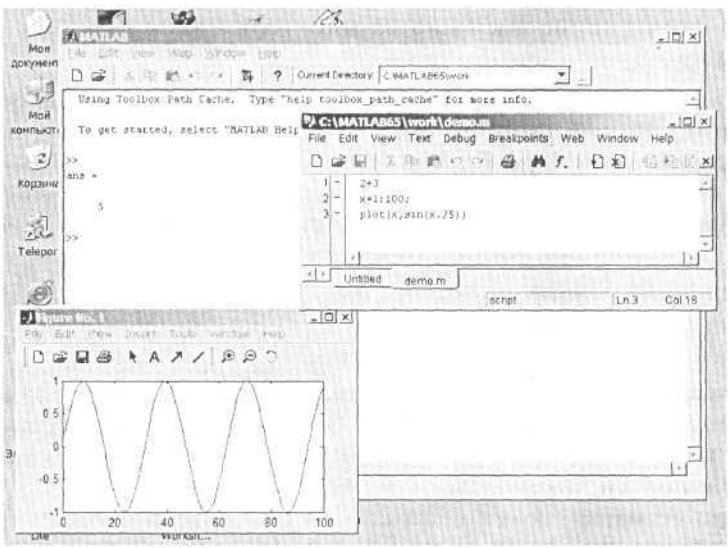

Рис. 1.9. Выполнение m-файла прямо из окна редактора/отладчика этих файлов

После записи файла на диск можно заметить, что команда Run в меню Debug редактора становится активной (до записи файла на диск она пассивна) и позволяет произвести запуск файла. Запустив команду Run, можно наблюдать исполнение М-файла — в нашем случае это вычисление числа 5 и построение графика синусоиды в окне, показанном на рис. 1.9 слева.

Редактор имеет и другие важные отладочные средства — он позволяет устанавливать в тексте файла специальные метки, именуемые контрольными точками. При их достижении вычисления приостанавливаются, и пользователь может оценить промежуточные результаты вычислений (например, значения переменных), проверить правильность выполнения циклов и т. д. Наконец, редактор позволяет записать файл в текстовом формате и увековечить ваши труды в файловой системе MATLAB.

После сохранения М-файла редактор/отладчик можно закрыть. При этом созданный М-файл можно запустить уже из командной строки MATLAB.

Редактор/отладчик М-файлов выполняет синтаксический контроль текстов (листингов) файлов по мере ввода текста. При этом используются следующие цветовые выделения:

- ключевые слова языка программирования — синий цвет;
- операторы, константы и переменные — черный цвет;
- комментарии после знака % — зеленый цвет;
- символьные переменные (в апострофах) — зеленый цвет;
- синтаксические ошибки — красный цвет.

Благодаря цветовым выделениям, а точнее их учету программистом, вероятность синтаксических ошибок резко снижается.

1.9.6. Файлы-сценарии и файлы-функции

Здесь полезно отметить, что М-файлы, используемые в MATLAB, делятся на два класса:

- файлы-сценарии, не имеющие входных параметров;
- файлы-функции, имеющие входные параметры.

Видимый в окне редактора на рис. 1.9 файл является файлом-сценарием (Script-файлом). Данный файл не имеет списка входных параметров и является примером простой процедуры без параметров. Он использует *глобальные переменные*, то есть такие переменные, значения которых могут быть изменены в любой момент сеанса работы и в любом месте программы. Структура этого файла следующая:

```
%Основной комментарий
%Дополнительный комментарий
Тело файла с любыми выражениями
```

Основным комментарием является первая строка текстовых комментариев, а дополнительным — последующие строки. Основной комментарий выводится при выполнении команд `lookfor` и `help имя_каталога`. Полный комментарий выводится при выполнении команды `help Имя_файла`.

Тело файла-сценария может содержать любые математические и логические выражения, а также управляющие структуры (например, циклы и условные выражения), которые присущи MATLAB языку программирования. Он реализует модульный и объектно-ориентированный подход к подготовке программ и имеет средства для создания элементов интерфейса пользователя.

М-файл-*функция* является типичным объектом — модулем языка программирования системы MATLAB. Структура такого модуля с одним выходным параметром выглядит следующим образом:

```
function var=f_name(Список_параметров)
%Основной комментарий
%Дополнительный комментарий
Тело файла с любыми выражениями
var=выражение
```

М-файл-функция имеет следующие свойства:

- он начинается с объявления `function`, после которого указывается имя переменной `var` — выходного параметра, имя самой функции и список ее входных параметров;
- функция возвращает свое значение и может использоваться в виде `name(Список_параметров)` в математических выражениях;
- все переменные, имеющиеся в теле файла-функции, являются локальными, то есть действуют только в пределах тела функции;
- файл-функция является самостоятельным программным модулем, который общается с другими модулями через свои входные и выходные параметры;
- правила вывода комментариев те же, что у файлов-сценариев;
- файл-функция служит средством расширения системы MATLAB;

- при обнаружении файла-функции он компилируется и затем исполняется, а созданные машинные коды хранятся в рабочей области системы MATLAB.

Последняя конструкция `var=выражение` вводится, если требуется, чтобы функция возвращала результат вычислений. Если выходных параметров больше, то они указываются в квадратных скобках после слова `function`. При этом структура модуля имеет следующий вид:

```
function [var1,var2,...]=f_name(Список_параметров)
%Основной комментарий
%Дополнительный комментарий
Тело файла с любыми выражениями
var1=выражение
var2=выражение
...
```

Переменные в файлах-функциях — по умолчанию *локальные*. Это значит, что они, меняя свое значение внутри модуля файла-функции, не сохраняют его при выходе из файла-функции. Нередко применение глобальных переменных в программных модулях может приводить к побочным эффектам. Применение локальных переменных устраняет эту возможность и отвечает требованиям структурного программирования.

Однако передача данных из модуля в модуль в этом случае происходит только через входные и выходные параметры, что требует тщательного планирования такой передачи. Команда `global var1 var2...` позволяет объявить переменные модуля-функции глобальными. Таким образом, внутри функции могут использоваться и глобальные переменные, если это нужно по условиям решения вашей задачи.

Итак, файл-функция отличается от файла-сценария прежде всего тем, что созданная им функция имеет *входные параметры*, список которых указывается в круглых скобках. Вот пример задания простой функции, вычисляющей сумму квадратов x и y:

```
function z=fun(x,y)
z=x^2+y^2;
return
```

Имя этой функции z и под таким именем ее надо записать в виде М-файла. `(x,y)` — список параметров (`x` и `y`). Оператор `return` задаст возврат функцией значения в ответ на обращение к ней по имени с указанием фактических параметров. Например, задав `z(2,3)`, получим 13.

В MATLAB можно включать *подфункции*. Они объявляются и записываются в теле основных функций и имеют идентичную им конструкцию — см. пример ниже:

```
function [mean,stdev] = statv(x)
%STATV Interesting statistics.
%Пример функции с встроенной подфункций
n = length(x);
mean = avg(x,n);
stdev = sqrt(sum((x-avg(x,n)).^2)/n);
%-----------------------------
function m = avg(x,n)
%Mean subfunction
m = sum(x)/n;
```

В этом примере среднее значение элементов вектора x вычисляется с помощью подфункции `avg(x,n)`, тело которой записано в теле основной функции `statv`. Пример использования функции `statv` представлен ниже:

```
>> V=[1 2 3 4 5]
V =
     1     2     3     4     5
>> [a,m]=statv(V)
a =
     3
m =
    1.4142
>> statv(V)
ans =
     3
>> help statv
 STATV Interesting statistics.
 Пример функции с встроенной подфункций
```

Подфункции определены и действуют локально, то есть только в пределах M-файла, определяющего основную функцию. Команда `help name` выводит комментарий, относящийся только к основной функции, тогда как команда `type name` выводит весь листинг M-файла. Так что заданные в некотором M-файле подфункции нельзя использовать ни в командном режиме работы, ни в других M-файлах.

1.9.7. Панель инструментов редактора/отладчика

Редактор/отладчик M-файлов имеет свое меню и свою инструментальную панель, которая может перемещаться мышью в любое подходящее место. Внешний вид инструментальной панели показан на рис. 1.9. По стилю данная панель похожа на панель инструментов окна командного режима работы, но имеет несколько иной набор кнопок.

Назначение кнопок панели инструментов редактора/отладчика следующее:

- New — создание нового m-файла;
- Open — вывод окна загрузки файла;
- Save — запись файла на диск;
- Cut — вырезание выделенного фрагмента и перенос его в буфер;
- Copy — копирование выделенного объекта в буфер;
- Paste — размещение фрагмента из буфера в позиции текстового курсора;
- Print — печать содержимого текущего окна редактора;
- About — вывод данных о версии редактора;
- Set/Clear Breakpoint — установка/сброс контрольной точки;
- Clear All Breakpoint — сброс всех контрольных точек;
- Step In — пошаговая трассировка с заходом в вызываемые m-файлы;
- Single Step — пошаговая трассировка без захода в вызываемые m-файлы;
- Continue — продолжение работы программы без трассировки;
- Quit Debugging — завершение отладки.

С назначением первых восьми кнопок вы уже знакомы, поскольку оно аналогично описанному для основного окна MATLAB. А вот о назначении других кнопок надо поговорить.

1.9.8. Работа с точками останова

Основным приемом отладки M-файлов является установка в их тексте контрольных точек остановки (Breakpoints). Они устанавливаются (и сбрасываются) с помощью кнопки Set/Clear Breakpoint. Сброс всех контрольных точек обеспечивается кнопкой Clear All Breakpoints.

При остановке в контрольной точке вы можете провести контроль значений переменных как «вручную», так и с помощью организации вывода на просмотр перед контрольной точкой. Вы можете задать выполнение программы без захода (кнопка Single Step) и с заходом в вызываемые M-файлы (кнопка Step In). Кнопка Continue продолжает исполнение программы после останова, а кнопка Quit Debugging прекращает операции отладки.

1.9.9. Завершение вычислений и работы с системой

Иногда из-за ошибок в программе или из-за сложности решаемой задачи MATLAB «зацикливается» и перестает выдавать результаты либо непрерывно выдает их, хотя в этом уже нет необходимости. Для прерывания вычислений в этом случае достаточно нажать одновременно клавиши Ctrl и C (латинское).

Для завершения работы с системой можно использовать команды `quit`, `exit` или комбинацию клавиш Ctrl + Q. Если необходимо сохранить значения всех переменных (векторов, матриц) системы, то перед этим следует дать команду `save` нужной формы. Команда `load` после загрузки системы считывает значения этих переменных и позволяет начать работу с системой с того момента, когда она была прервана.

На этом мы закончим начальную экскурсию в технику матричных вычислений системы MATLAB. Мы расширим представления о ней в последующих уроках.

1.10. Двумерная графика

1.10.1. Особенности графики системы MATLAB

Начиная с версии MATLAB 4.0, впервые ориентированной на Windows, графические средства системы MATLAB были существенно улучшены. Основные отличительные черты графики в новых версиях MATLAB 6.*:

- существенно улучшенный интерфейс графических окон;
- введение новой панели инструментов Camera для интерактивного изменения условий видимости объекта;
- расширенные возможности форматирования графики;
- возможность создания графики в отдельных окнах;
- возможность вывода нескольких графических окон;
- возможность перемещения окон по экрану и изменения их размеров;
- возможность перемещения области графики внутри графического окна;
- задание различных координатных систем и осей;

- высокое качество графики;
- широкие возможности использования цвета;
- легкость установки графических признаков — атрибутов;
- снятие ограничений на число цветов;
- обилие параметров команд графики;
- возможность получения естественно выглядящих трехмерных фигур и их сочетаний;
- простота построения трехмерных графиков с их проекцией на плоскость;
- возможность построения сечений трехмерных фигур и поверхностей плоскостями;
- функциональная многоцветная и полутоновая окраска;
- возможность имитации световых эффектов при освещении фигур точечным источником света;
- возможность создания анимационной графики;
- возможность обработки данных графиков (аппроксимация, регрессия и др.) прямо в графическом окне;
- возможность создания объектов для типового интерфейса пользователя.

С понятием графики связано представление о *графических объектах*, имеющих определенные свойства. В большинстве случаев об объектах можно забыть, если только вы не занимаетесь объектно-ориентированным программированием задач графики. Связано это с тем, что большинство команд высокоуровневой графики, ориентированной на конечного пользователя, автоматически устанавливает свойства графических объектов и обеспечивает воспроизведение графики в нужной системе координат, палитре цветов, масштабе и т. д.

На более низком уровне решения задач используется ориентированная на опытного программиста *дескрипторная графика* (Handle Graphics), при которой каждому графическому объекту в соответствие ставится особое описание — *дескриптор*, на который возможны ссылки при использовании графического объекта. Дескрипторная графика позволяет осуществлять визуальное программирование объектов пользовательского интерфейса — управляющих кнопок, текстовых панелей и т. д. Команды дескрипторной графики могут использоваться в высокоуровневой графике, например, для удаления осей, изменения цвета и т. д. в уже построенных графических объектах. Эти обширные возможности делают графику MATLAB одной из лучших среди графических подсистем компьютерной математики (СКМ).

1.10.2. Построение графика функций одной переменной

В режиме непосредственных вычислений доступны практически все возможности системы. Широко используется, например, построение графиков различных функций, дающих наглядное представление об их поведении в широком диапазоне изменения аргумента. При этом графики строятся в отдельных масштабируемых и перемещаемых окнах.

Возьмем вначале простейший пример — построение графика синусоиды. Следует помнить, что MATLAB (как и другие СКМ) строит графики функций по ряду точек, соединяя их отрезками прямых, т. е. осуществляя линейную интерполяцию функции в интервале между смежными точками. Зададим интервал изменения аргумента x от 0 до 10 с шагом 0,1. Для построения графика достаточно вначале за-

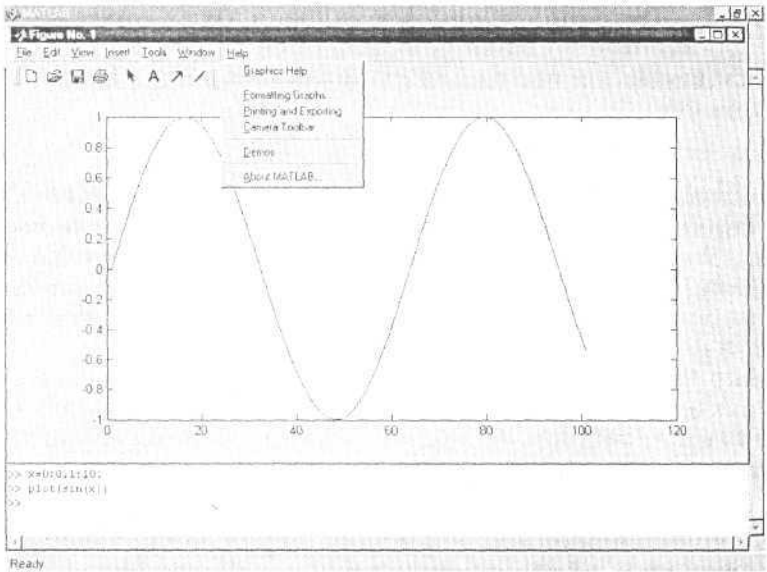

Рис. 1.10. Пример построения графика синусоиды

дать вектор `x=0:0.1:10`, а затем использовать команду построения графиков `plot(sin(x))`. Это показано на рис. 1.10. Само задание и выполнение команд видно в левом нижнем углу окна системы MATLAB.

Вектор x задает интервал изменения независимой переменной от 0 до 10 с шагом 0,1. Почему взят такой шаг, а не, скажем, 1? Дело в том, что `plot` строит не истинный график функции sin(x), а лишь заданное числом элементов вектора x число точек. Эти точки затем просто соединяются отрезками прямых, то есть осуществляется кусочно-линейная интерполяция данных графика. При 100 точках полученная кривая глазом воспринимается как вполне плавная, но при 10—20 точках она будет выглядеть состоящей из отрезков прямых.

Графики MATLAB строит в отдельных окнах, называемых *графическими окнами*. С первого взгляда видны отличия графического окна, показанного на рис. 1.10, от командного окна MATLAB. В главном меню окна появилась позиция Tools (Инструменты), которая позволяет вывести или скрыть инструментальную панель, видимую в верхней части окна графики на рис. 1.10. Средства этой панели (мы их рассмотрим полнее в дальнейшем) позволяют легко управлять параметрами графиков и наносить на них текстовые комментарии в любом месте.

1.10.3. Построение в одном окне графиков нескольких функций

Пойдем дальше и попытаемся построить графики сразу трех функций: sin(*x*), cos(*x*) и sin(*x*)/*x*. Прежде всего, отметим, что эти функции могут быть обозначены переменными, не имеющими явного указания аргумента в виде y(x):

```
>> y1=sin(x); y2=cos(x); y3=sin(x)/x;
```

Такая возможность обусловлена тем, что эти переменные являются векторами — как и переменная x. Теперь можно использовать одну из ряда форм команды `plot`:

```
plot(a1,f1,a2,f2,a3,f3,...),
```

где a1, a2, a3, ... — векторы аргументов функций (в нашем случае все они — x), а f1, f2, f3, ... —векторы значений функций, графики которых строятся в одном окне. В нашем случае для построения графиков указанных функций мы должны записать следующее:

```
>> plot(x,y1,x,y2,x,y3)
```

Можно ожидать, что MATLAB в этом случае построит, как обычно, точки графиков этих функций и соединит их отрезками линий. Но, увы, если мы выполним эти команды, то никакого графика не получим вообще. Не исключен даже сбой в работе программы. Причина этого казуса уже обсуждалась в предыдущем уроке — при вычислении функции y3=sin(x)/x, если x представляет собой массив (вектор), то нельзя использовать оператор матричного деления /.

Этот пример еще раз наглядно указывает на то, что чисто поверхностное применение даже такой мощной системы, как MATLAB, иногда приводит к досадным срывам. Чтобы все же получить график, надо вычислять отношение $\sin(x)$ к x с помощью оператора поэлементного деления массивов ./. Этот случай поясняет рис. 1.11.

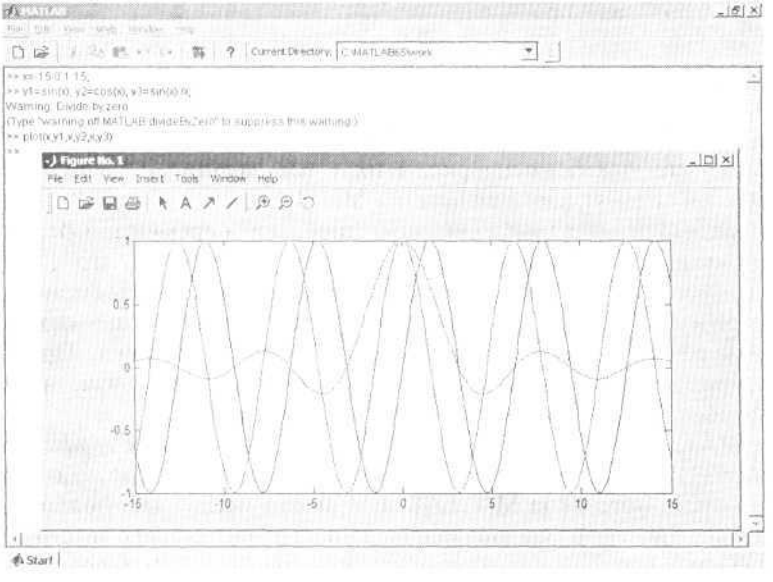

Рис. 1.11. Построение графиков трех функций

Обратите внимание на то, что хотя на этот раз MATLAB построил графики всех трех функций, в окне командного режима появилось предупреждение о делении на 0 — в момент, когда x = 0 — «Warning: Divide by zero». Таким образом plot «не знает» о том, что неопределенность sin(x)/x = 0/0 устранимая, и дает 1. Это недостаток практически всех систем для численных вычислений.

1.10.4. Графическая функция fplot

Разумеется, MATLAB имеет средства для построения графиков и таких функций, как sin(x)/x, которые имеют устранимые неопределенности. Не обсуждая эти средства подробно, просто покажем, как это делается, с помощью другой графической команды — fplot:

```
fplot('f(x)', [xmin xmax])
```

Она позволяет строить график функции f(x), заданной в символьном виде, в интервале изменения аргумента x от xmin до xmax без фиксированного шага изменения x. Один из вариантов ее применения демонстрирует рис. 1.12. Хотя в процессе вычислений предупреждение об ошибке (деление на 0) выводится, но график строится правильно, при x = 0 sinx/x = 1. Обратите также внимание на две используемые команды: clear (очистить) — очистка графического окна и grid on (сетка) — включение отображения сетки, которая строится пунктирными линиями.

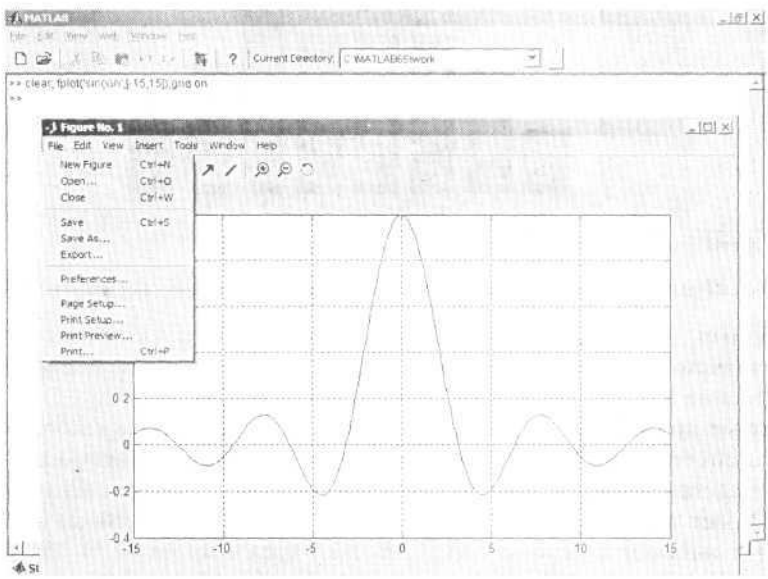

Рис. 1.12. Построение графика sin(x) / x функцией fplot

На рис. 1.12 представлено также меню File (Файл) окна графики. Нетрудно заметить, что оно содержит типовые файловые операции. Однако они относятся не к файлам документов, а к файлам графиков. В частности, можно присваивать имя записываемым на диск рисункам с графиками.

Позже мы более подробно рассмотрим возможности различных графических команд. Мы также покажем, как можно задавать определенный цвет и стиль линий, как менять вывод координатных осей, наносить на графики различные текстовые надписи и выполнять множество иных операций форматирования графиков для придания им более наглядного вида, соответствующего требованиям пользователя. Мы также обсудим множество новых форм применения графических команд, резко расширяющих их возможности построения графиков всех мыслимых типов.

1.10.5. Столбцовые диаграммы

В прикладных расчетах часто встречаются графики, именуемые столбцовыми диаграммами, отражающие содержание некоторого вектора V. При этом каждый элемент вектора представляется столбцом, высота которого пропорциональна зна-

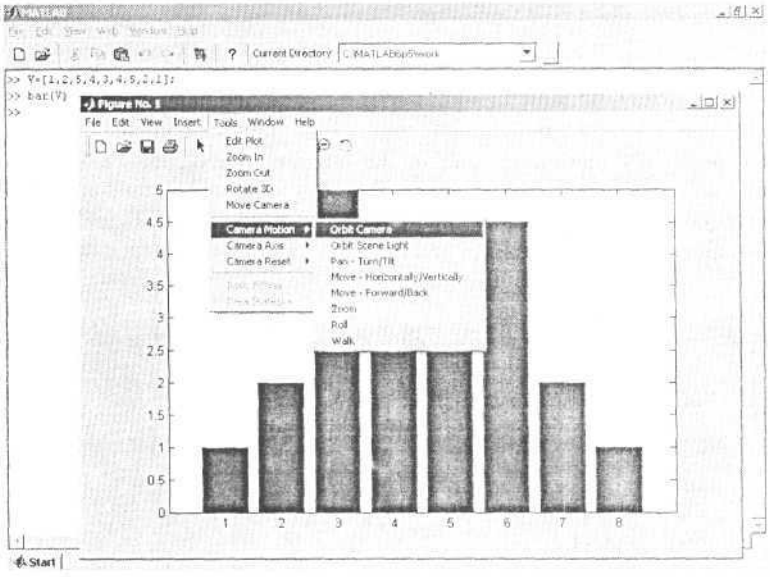

Рис. 1.13. Построение столбцовой диаграммы значений элементов вектора

чению элемента. Столбцы нумеруются и масштабируются по отношению к максимальному значению наиболее высокого столбца. Выполняет построение такого графика команда `bar(V)` (рис. 1.13).

Столбцовые диаграммы — лишь один из многих типов графиков, которые может строить система MATLAB. Особенно часто столбцовые диаграммы используются при представлении данных финансово-экономических расчетов.

Рис. 1.13 дает также представление о меню Tools (Инструменты) окна графики, появившемся начиная с версии MATLAB 5.3.1 (реализация 11.1). Нетрудно заметить, что кроме возможности вывода инструментальной панели здесь имеется целый ряд других команд, которые будут рассмотрены в дальнейшем. Это команды вывода свойств графических объектов, изменения масштаба графика, добавления осей и т. д.

1.11. Трехмерная графика

1.11.1. Построение трехмерных графиков

Столь же просто обеспечивается построение графиков сложных поверхностей. Надо только знать, какой командой реализуется тот или иной график. Например, для построения графика поверхности и ее проекции в виде контурного графика на плоскость под поверхностью достаточно использовать следующие команды:

```
>> [X,Y]=meshgrid(-5:0.1:5);
>> Z=X.*sin(X+Y);
>> meshc(X,Y,Z)
```

Окно с построенным графиком показано на рис. 1.14. Раньше пришлось бы убить много дней на составление и отладку нужной для построения такого графика программы. В MATLAB же можно в считанные секунды изменить задающую по

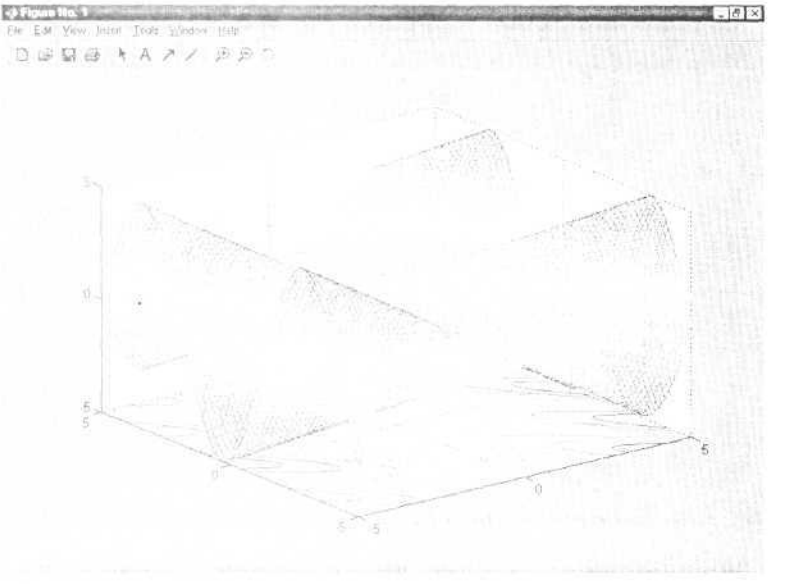

Рис. 1.14. Окно с графиками поверхности и ее проекции на плоскость под фигурой

верхность функцию *Z(X, Y)* и тут же получить новый график поверхности с окраской, в данном случае заданной вектором Z, и с ее проекцией на плоскость XY. На рис. 1.14 показано также открытое меню Help (Помощь) окна трехмерной графики.

Мы ограничимся этими примерами построения графиков как достаточно простыми и типовыми. Из них следует важный вывод — для решения той или иной частной задачи надо знать соответствующие команды и функции. В этом вам помогут как данная книга, так и справочная система MATLAB.

1.11.2. Вращение графиков мышью

Можно поворачивать построенную фигуру мышью и наблюдать ее под разными углами. Рассмотрим эту возможность на примере построения логотипа системы MATLAB — мембраны. Для этого, введя команду `membrane`, получим исходный график, представленный на рис. 1.15.

Для вращения графика достаточно активизировать последнюю справа кнопку панели инструментов с изображением пунктирной окружности со стрелкой. Теперь, введя курсор мыши в область графика и нажав левую кнопку мыши, можно круговыми движениями заставить график вращаться вместе с обрамляющим его параллелепипедом (рис. 1.16).

Любопытно, что в версии MATLAB 6 вращать можно и двумерные графики, наблюдая поворот плоскости, в которой они построены. Никакого программирования такое вращение не требует.

1.11.3. Контекстное меню графиков

Для переключения в режим редактирования графика нужно щелкнуть на кнопке Edit Plot (Редактировать график) с изображением курсора-стрелки. В этом режиме графиком можно управлять с помощью контекстного меню, вызываемого

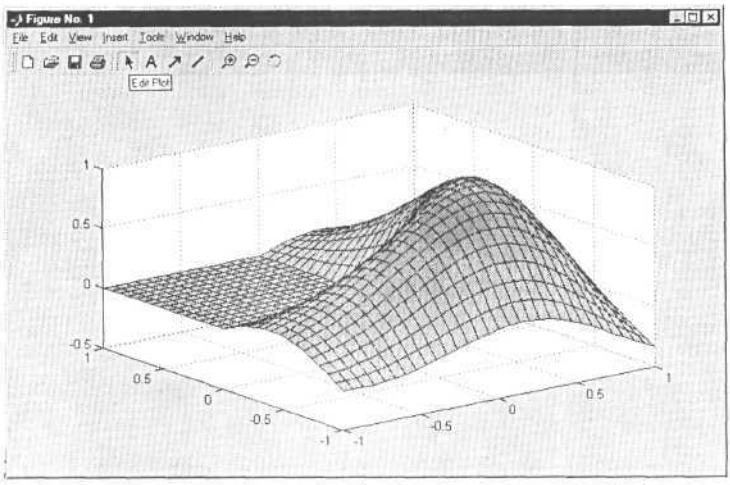

Рис. 1.15. Построение мембраны — логотипа системы MATLAB

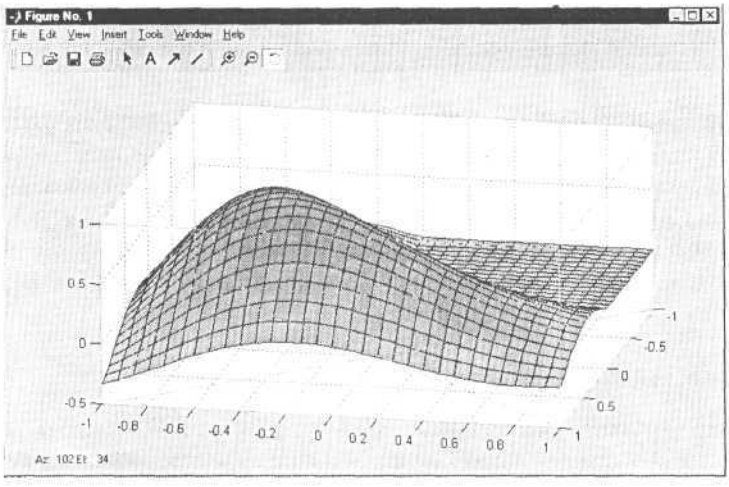

Рис. 1.16. Вращение трехмерной фигуры мышью

щелчком правой кнопки мыши. Вид этого меню при курсоре, расположенном в области трехмерного графика вне построенных трехмерных графических объектов, показан на рис. 1.17. С помощью мыши можно также выделить график. Щелчок левой клавишей выводит рамку вокруг рисунка (см. рис. 1.17). Теперь на график можно наносить стрелки, поясняющие надписи (кнопка с буквой A) и т. д.

1.12. Основы форматирования графиков

1.12.1. Форматирование двумерных графиков

Графики в системе MATLAB строятся обманчиво просто. Связано это с тем, что многие свойства графиков установлены по умолчанию. К таким свойствам относятся вывод или скрытие координатных осей, положение их центра, цвет линии

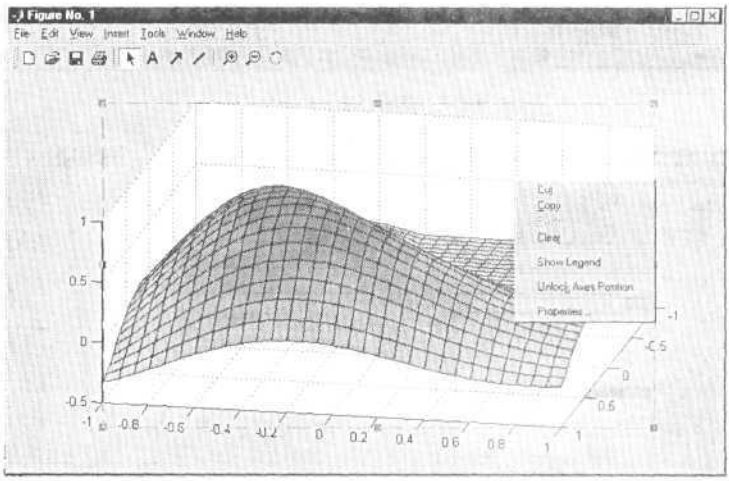

Рис. 1.17. График в состоянии редактирования и контекстное меню

графика, ее толщина и т. д. и т. п. Позже будет показано, как свойства и вид графиков можно менять в широких пределах с помощью параметров команд графики. Однако этот путь требует хорошего знания деталей языка программирования и дескрипторной графики системы MATLAB.

В новых версиях MATLAB 6.* для изменения свойств графиков (их форматирования) используются принципы визуального контроля за стилем (видом) всех объектов графиков. Это позволяет легко, просто и наглядно придать графикам должный вид перед записью их в виде файлов на диск. Можно сказать, что в этой части реализованы отдельные принципы визуально-ориентированного программирования графических средств.

Здесь мы рассмотрим возможности форматирования графиков, которые, образно говоря, лежат на поверхности. Систематизированное описание интерфейса систем MATLAB 6.*, в том числе интерфейса графических окон, дается в первой книге данного четырехтомника.

1.12.2. Форматирование линий графиков

MATLAB имеет возможность легко настраивать и корректировать свойства графиков с помощью специальных средств. В новых версиях MATLAB 6.* они существенно изменены. Так, в предшествующих версиях MATLAB 5.* для настройки (форматирования) графиков использовался специальный редактор свойств — Graphics Properties Editor (Редактор свойств графики). Его можно было вызвать из меню File окна командного режима MATLAB с помощью команды Show Graphics Properties Editor (Показать редактор свойств графики).

В новой версии MATLAB форматирование графиков стало более строгим и удобным. При этом ранее упомянутый редактор свойств графиков перестал так именоваться, и команда Show Graphics Properties в новых версиях отсутствует. Ее заменяют команды Figure Properties (Свойства фигуры) и Axis Properties (Свойства осей) со всеми необходимыми настройками.

При построении графиков появляется графическое окно. Иногда оно бывает скрыто ранее имеющимися окнами как системы MATLAB, так и других работающих в среде Windows 95/98/Me/2000/NT4 приложений. Если вы не увидели гра-

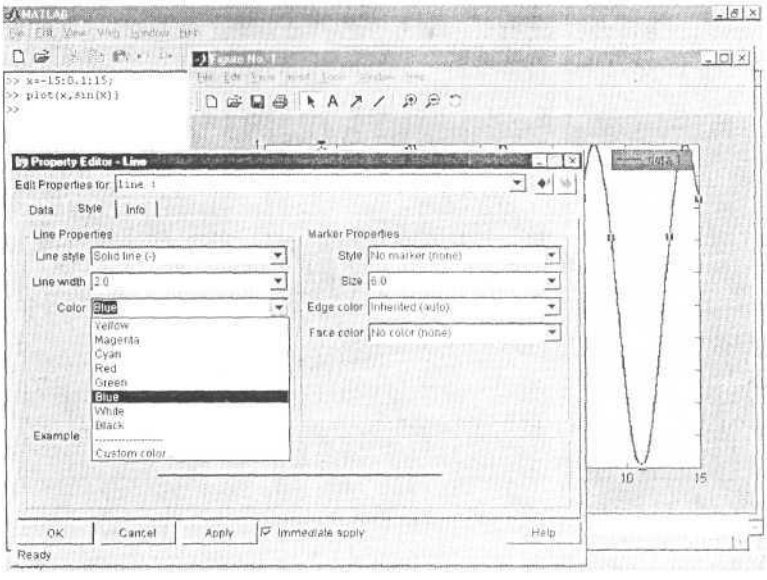

Рис. 1.18. Окно графика (справа) и окно форматирования линий (слева)

фика, заданного для построения, то поищите его в списке открытых окон (приложений), нажимая клавиши Alt + Tab, и выберите из списка нужное окно. Окна графики имеют изображение логотипа системы MATLAB. По умолчанию они выводятся с панелью инструментов с рядом кнопок вполне очевидного назначения.

Щелкнув на кнопке Edit Plot (Редактировать график) в панели инструментов окна графики и щелкнув по графику, можно заметить, что график выделился: вокруг него появилась рамка. Теперь, указав курсором мыши на тот или иной объект графика и щелкнув снова левой клавишей, можно наблюдать выделение объекта и появление окна его форматирования.

Например, указав в режиме редактирования мышью на линию графика (и дважды быстро щелкнув левой клавишей), можно увидеть окно форматирования линий графика, показанное на рис. 1.18 слева. Часть окна графики с выделенным графиком видна справа. Обратите внимание на появление на линии графика ряда черных квадратиков — они используются для указания курсором мыши именно на линию графика, а не на другие объекты.

В этом окне открыта главная для операций форматирования вкладка — Style (Стиль). Она устанавливает стиль отображения линии, то есть ее вид (например сплошная линия или пунктирная), ширину и цвет, а также параметры маркеров отмечающих опорные точки графиков.

Полезно знать, что кнопка Apply (Применить) позволяет применить сделанные установки к графику до закрытия окна диалога. Кнопка OK вводит сделанные установки и закрывает окно диалога. Назначение других кнопок очевидно.

1.12.3. Форматирование маркеров опорных точек

В нашем случае опорные точки задаются ранжированной переменной x, имеющей ряд значений от −15 до +15 с шагом 0.1. Эти точки появляются на графике если в поле свойств маркера Marker Properties (Свойства маркера) из меню Style

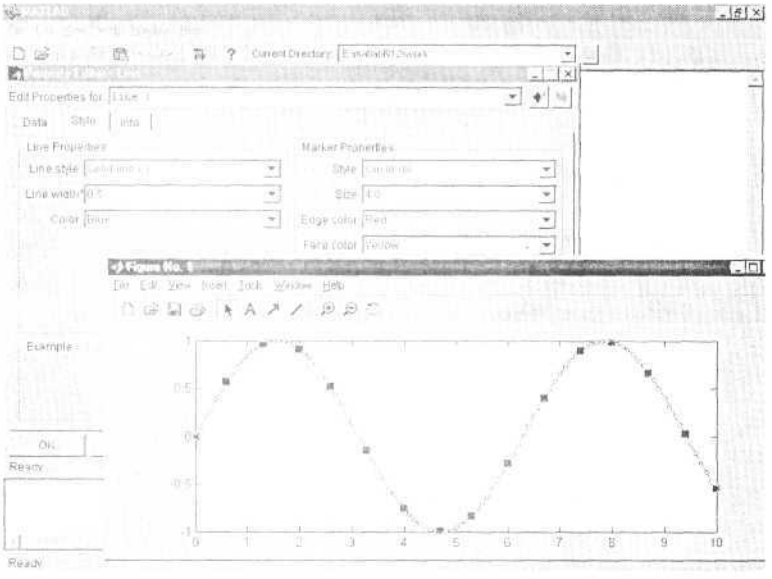

Рис. 1.19. Пример задания параметров маркеров и построения графика с ними

(Стиль) выбрать стиль маркера. На рис. 1.19, к примеру, показано построение графика с маркерами опорных точек в виде окружностей.

Можно задавать размеры маркеров, цвет их закраски и цвет окантовки. Так, на рис. 1.19 при его просмотре на экране цветного дисплея маркеры имеют вид окружностей с условным размером 4, цветом окантовки красным и цветом закраски желтым. Маркеры можно задавать в виде окружностей, прямоугольников, крестиков, ромбиков и так далее. Применение маркеров делает графики более наглядными. Но, к сожалению, в графиках с маркерами вы не сможете использовать возможности Open GL.

1.12.4. Форматирование линий и маркеров для графика нескольких функций

Если строится график нескольких функций, то можно форматировать линии и маркеры каждой кривой отдельно. Выполним следующие команды:

```
>> x=-6:.1:6;
>> plot(x,sin(x),x,sin(x).^3,x,sin(x).^5);
```

Рис. 1.20 показывает пример такого форматирования для графика, полученного исполнением этих команд.

Кстати, обратите внимание на то, как заданы степени синуса. Записать эти выражения в виде `sin(x)^2` и `cos(x)^2` будет грубейшей ошибкой, поскольку x здесь вектор. Операторы `.^` в данном случае дают поэлементное возведение в степень, что и нужно для построения графиков этих функций.

1.12.5. Форматирование осей графиков

Аналогично описанным выше правилам выполняется форматирование и других объектов графиков. Например, указав курсором мыши на оси графиков (на

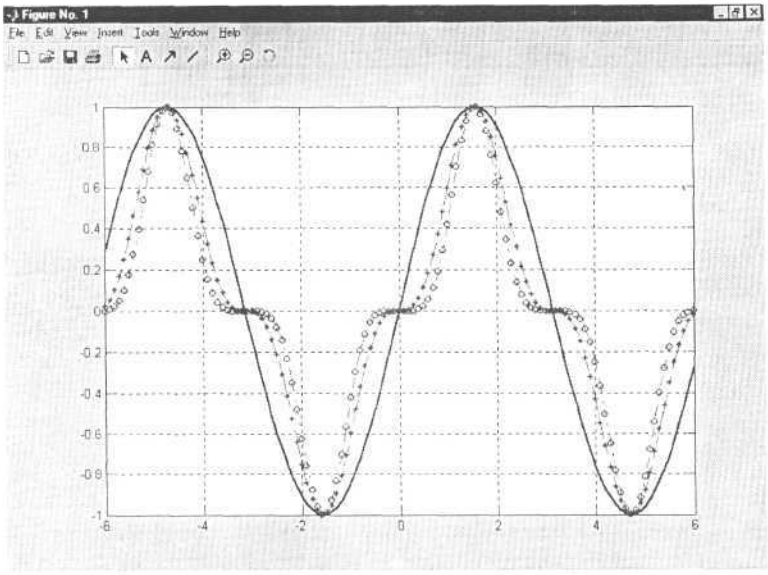

Рис. 1.20. Пример форматирования для графика трех функций

них тоже есть метки в виде черных квадратиков) и дважды щелкнув левой клавишей мыши, можно увидеть появление окна форматирования объектов дескрипторной графики Property Editor (Редактор свойств, Графический редактор свойств) (рис. 1.21), настроенного на форматирование осей.

Окно графического редактора свойств дескрипторной графики имеет множество вкладок, настройки которых довольно очевидны, и ничто не мешает читателю поэкспериментировать с ними несколько минут. Это позволит понять простоту и

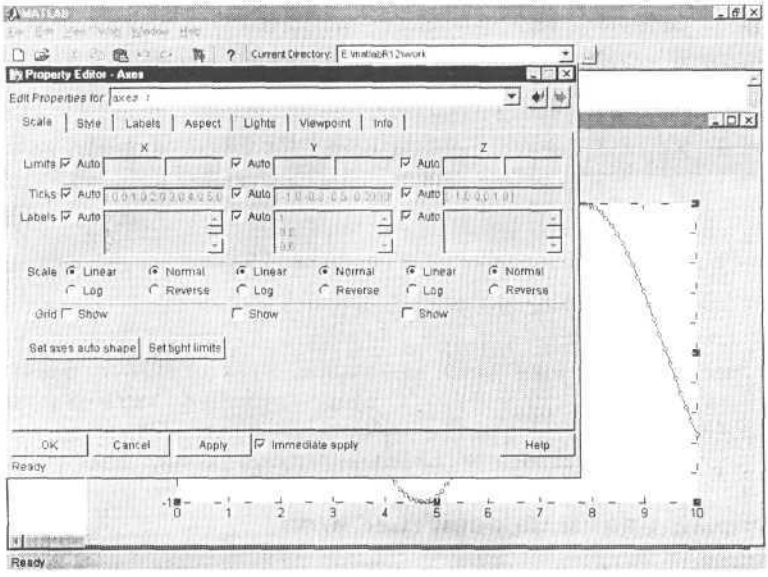

Рис. 1.21. Пример форматирования осей графика

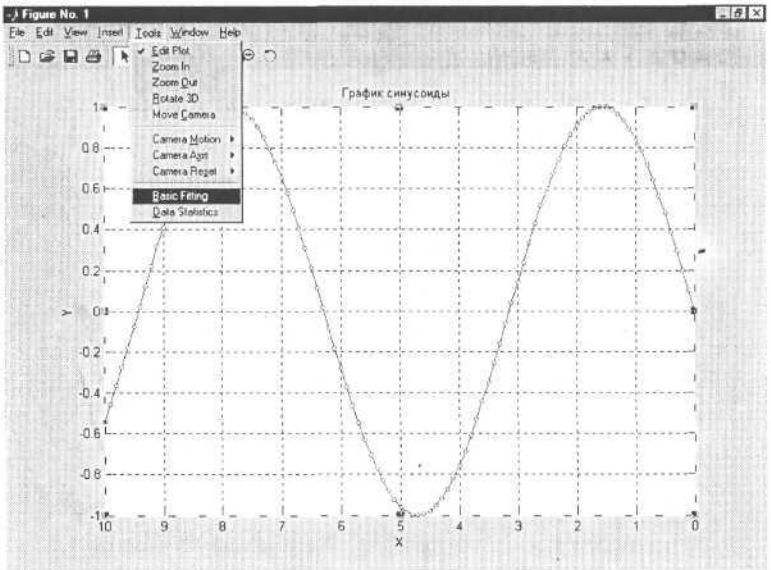

Рис. 1.22. Пример построения графика синусоиды после форматирования осей

одновременно высокую эффективность средств форматирования объектов графики. Например, вы можете задать линейный или логарифмический масштаб осей (вкладка Scale (Масштаб), открытая на рис. 1.21), нормальное или инверсное направление осей (X, У, а, в случае трехмерных графиков, и Z), показ сетки (параметр Grid Show), изменить стиль осей и цвета фона (вкладка Style (Стиль)), нанести у осей надписи (вкладка Label (Ярлык)) и прочее.

Рис. 1.22 показывает график синусоиды после некоторых операций по форматированию осей. Здесь (кстати, как и на рис. 1.21) задано построение сетки Grid по осям X и Y, построение надписей (просто буквы X и Y) по координатным осям и построение титульной надписи. Заодно на рис. 1.22 показано в открытом виде меню расширенных инструментальных средств графического окна. Его команды подробно обсуждаются в следующей главе.

Словом, с объектами графики можно сделать все что угодно! Некоторые из возможностей форматирования объектов графики мы рассмотрим позже, по мере описания типов графиков.

Если компьютер оснащен должным набором шрифтов, то надписи на графиках могут быть сделаны на русском языке — рис. 1.22 хорошо иллюстрирует эту важную для наших пользователей возможность. На нем титульная надпись сделана на русском языке. Средства форматирования надписей дают обширные возможности по выбору набора шрифтов, их стиля, размеров символов и их цвета.

1.12.6. Нанесение надписей и стрелок прямо на график

Дополнительно на график можно нанести надписи с помощью кнопки панели инструментов с буквой A. Место надписи фиксируется щелчком мыши. На рис. 1.23 показан отформатированный график с текстовым блоком, созданным таким образом в левой верхней части поля графика.

Там же показано контекстное меню правой клавиши мыши, поясняющее выбор размера символов надписи (и другие возможности этого меню). Напоминаем, что

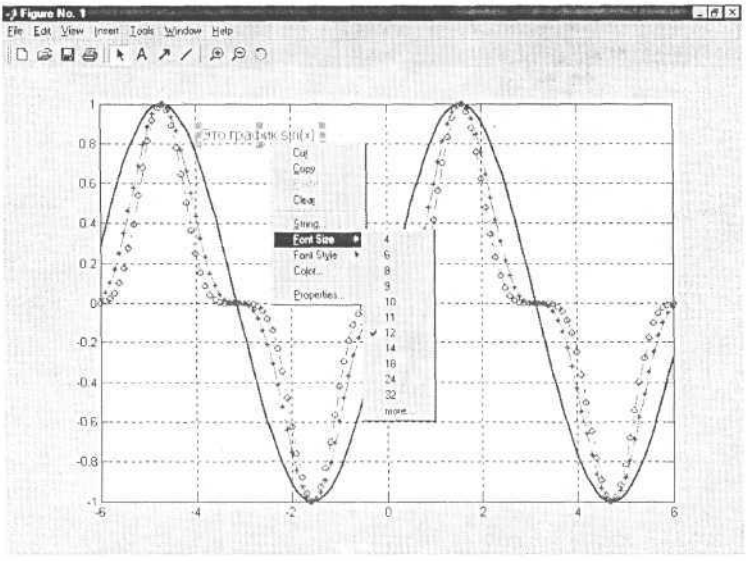

Рис. 1.23. Нанесение надписи на отформатированный график

это меню появляется при щелчке правой кнопки мыши на заданном объекте. В этом меню имеются все команды, доступные для данного объекта в данной ситуации.

Полученную таким образом надпись можно выделить и перенести мышью в любое другое место. Рис. 1.24 показывает процесс создания еще двух надписей с переносом их текстового блока в нужное место. Надписи сделаны с разным размером символов и разным стилем. Особенно приятно, что при задании на надписи возведения в степень знаком ^ надпись на экране отображается в естественном математическом виде (степень в виде верхнего индекса).

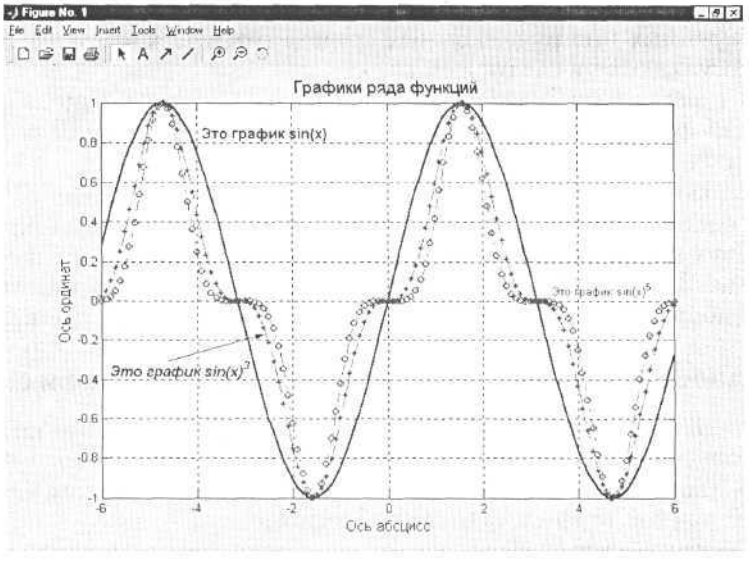

Рис. 1.24. Окончательно отформатированный график трех функций

На рис. 1.24, в частности, показано задание надписей разным стилем, а также задание стрелки с помощью соответствующей кнопки панели инструментов. Это стрелку в режиме редактирования графика можно перемещать и вращать мышью, а также менять ее длину. Можно также наносить на график и обычные линии (без стрелки).

1.12.7. Построение легенды и шкалы цветов на графике

Дополнительно можно изменить размеры графика (см. меню Tools (Инструменты) и его команды Zoom In (Увеличить) и Zoom Out (Уменьшить)), начать поворот графика мышью (команда Rotate 3D), добавить отрезок прямой или иной графический примитив (подменю Add) и подключить к графику *легенду* — пояснение в виде отрезков линий со справочными надписями, размещаемое внутри графика или около него. Поскольку наш график содержит три кривые, то легенда представляет собой обозначение этих трех линий в правом верхнем углу рисунка (рис. 1.25). Каждая линия имеет тот же цвет, что и на графике (и тот же стиль). Возможен также вывод шкалы цветов.

Следует еще раз отметить, что все описанные возможности форматирования графиков доступны и программным способом, путем задания соответствующих графических команд, параметров и примитивов. Например, команда `text(x,y,'legend')` позволяет задать надпись `'legend'` с началом, имеющим координаты (x,y). Если после первого апострофа перед текстом поместить параметр `\leftarrow`, то надпись (легенда) появится после стрелки с острием, обращенным влево. Аналогично, параметр `\rightarrow` после надписи задает вывод стрелки после надписи с острием, обращенным вправо. Эта возможность позволяет помечать не только кривые, но и отдельные точки на них. Возможно также применение команды `legend('s1','s2',...)`, выводящей легенду обычного вида — отрезки линий графиков с поясняющими надписями `'s1'`, `'s2'` и т. д.

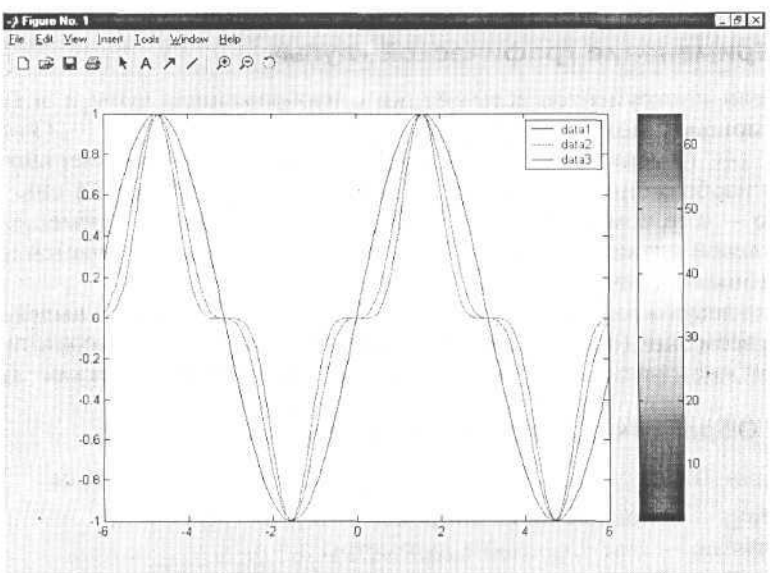

Рис. 1.25. График трех функций с легендой и шкалой цветов

1.12.8. Перемещение графика в графическом окне

Обычно график занимает фиксированное положение в центре графического окна. Однако в режиме редактирования графиков, когда курсор мыши находится в области графика, в контекстном меню правой клавиши мыши есть команда Unlock Axes Position (Отключить позиционирование осей). Она снимает фиксацию положения координатных осей графика и позволяет двигать его мышью вместе с осями. Это иллюстрирует рис. 1.26.

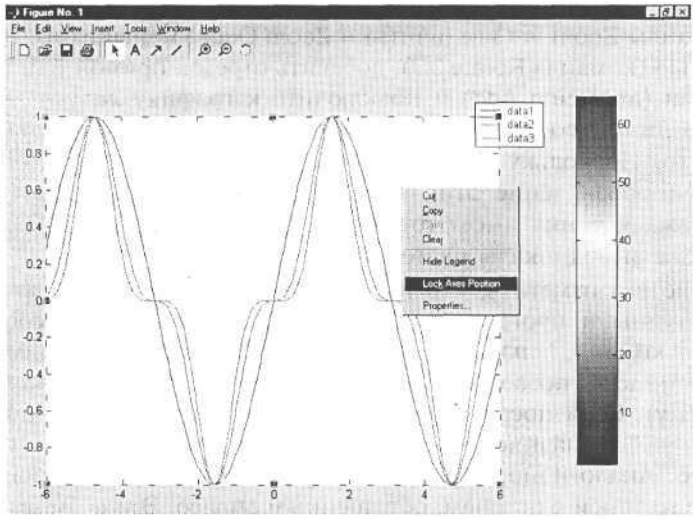

Рис. 1.26. Пример перемещения графика

Интересно, что при перемещении графика его легенда и цветовая диаграмма остаются на прежнем месте.

1.12.9. Применение графической «лупы»

На панели инструментов есть кнопки с изображением лупы и знаками + и -. С их помощью можно исполнять команды `Zoom In` (+) (Увеличить) и `Zoom Out` (-) (Уменьшить). Это позволяет увеличивать или уменьшать масштаб просмотра изображения. При этом команда `Zoom In` интересна еще одной возможностью — с ее помощью можно выделять часть графика перемещением мыши с нажатой левой клавишей (рис. 1.27). Область выделения отмечается прямоугольником из тонких точечных линий.

Отпустив левую клавишу мыши, можно наблюдать построение выделенной части графика на всем окне (рис. 1.28). С помощью команды `Zoom Out` можно восстановить график в прежнем масштабе. Таким образом реализуется графическая «лупа».

1.12.10. Обработка данных в графическом окне

В позиции Tools графического окна имеется две новые команды:

Basic Fiting — основные виды аппроксимации (регрессии);

Data Statistics — статистические параметры данных.

Команда Basic Fiting открывает окно, дающее доступ к ряду видов аппроксимации и регрессии: сплайновой, эрмитовой и полиномиальной со степенями от 1

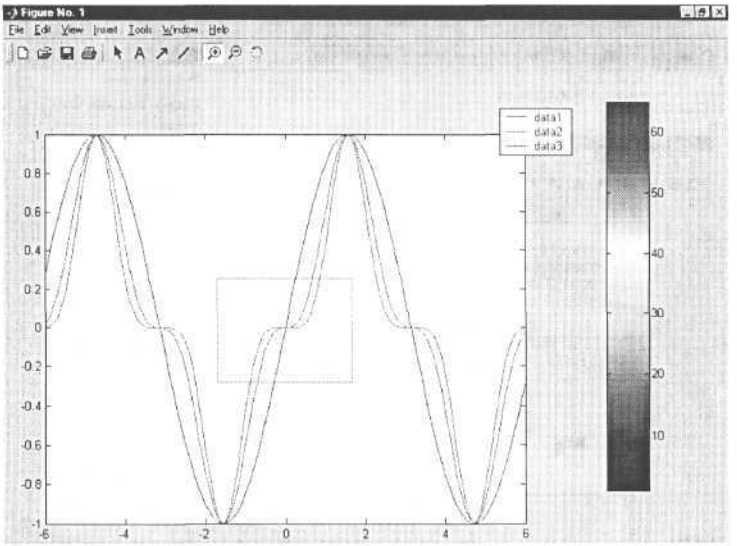

Рис. 1.27. Пример выделения части графика

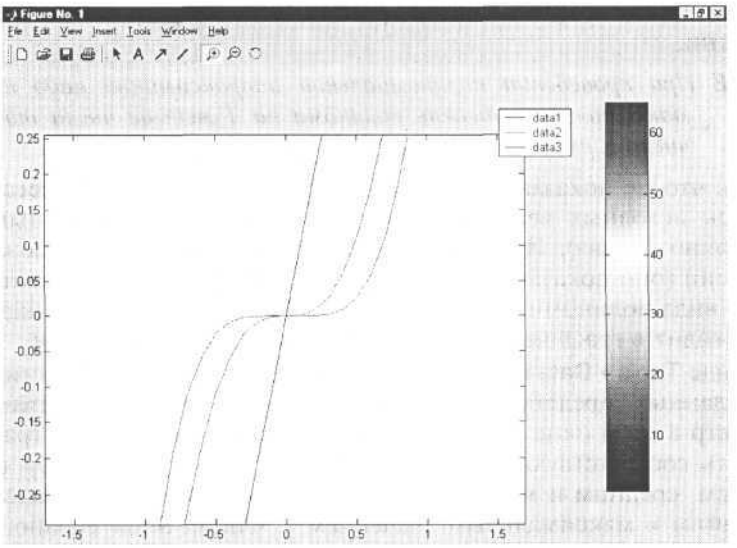

Рис. 1.28. Пример просмотра части графика

(линейная аппроксимация) до 10. В том числе со степенью 2 (квадратичная аппроксимация) и 3 (кубическая аппроксимация). Команда Data Statistics открывает окно с результатами простейшей статистической обработки данных.

Пусть некая зависимость y(x) она задана векторами координат ее точек:

```
>> X=[2,4,6,8,10,12,14];
>> Y=[3.76,4.4,5.1,5.56,6,6.3,6.7];
>> plot(X,Y,'o');
```

Рис. 1.29 показывает пример выполнения полиномиальной регрессии (аппроксимации) для степеней полинома 1, 2, 3 и 4. Иными словами, выполняется линей-

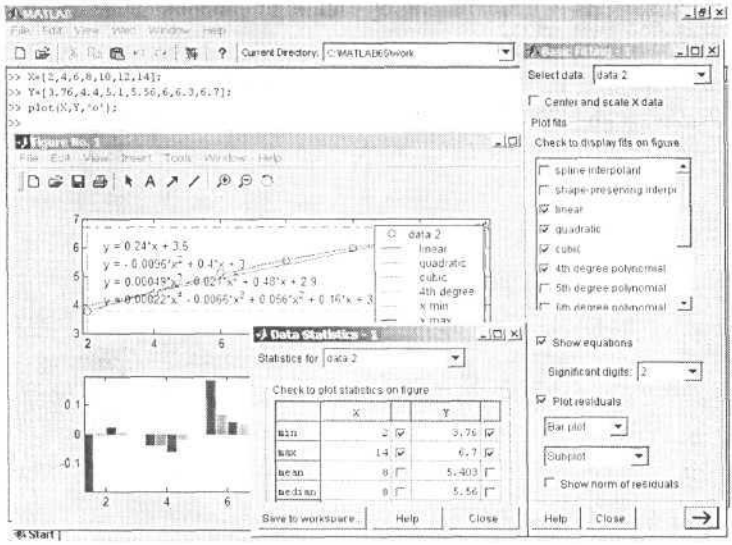

Рис. 1.29. Пример обработки табличных данных в графическом окне

ная, параболическая и кубическая регрессия, а также регрессии полиномом четвертой степени.

ВНИМАНИЕ *При проведении полиномиальной аппроксимации надо помнить, что максимальная степень полинома на 1 меньше числа точек, то есть числа элементов в векторах X и Y.*

Поясним, что же показано на рис. 1.29. В левом верхнем углу сессии MATLAB видна запись исходных векторов и команды построения заданных ими точек кружками (окно справа). Исполнив команду Tools ▸ Basic Fiting, можно получить окно регрессии (оно показано справа). В этом окне птичкой отмечены три упомянутых выше вида полиномиальной регрессии. Установка птички у параметра Show equations выводит в графическом окне записи уравнений регрессии.

По команде Tools ▸ Data Statistics выводится окно с рядом статистических параметров для данных, представленных векторами X и Y. Отметив птичкой тот или иной параметр в этом окне (оно показано на рис. 1.29 под окном графики), можно наблюдать соответствующие построения на графике, например вертикалей с минимальным, средним и максимальным значением y и горизонталей с минимальным, средним и максимальным значением x. Опция Show equation выводит на график уравнения регрессии, а опция Plot residuals строит столбиковую диаграмму погрешностей (видна под графиком). Команда Tools ▸ Data Statistics выводит окно для задания построения статистических параметров данных.

О других возможностях оперативной обработки данных в графическом окне можно прочитать в [11] или в справке.

1.12.11. Графики разного типа в одном окне

Бывает, что в одном окне надо расположить несколько координатных осей с различными графиками без наложения их друг на друга. Для этого используются команды:

- `subplot` — создает новые объекты класса `axes` (подокна);

- `subplot(m,n,p)` или `subplot(mnp)` — разбивает графическое окно на $m \times n$ подокон, при этом `m` — число подокон по горизонтали, `n` — число подокон по вертикали, а `p` — номер подокна, в которое будет выводиться текущий график (подокна отсчитываются последовательно по строкам);
- `subplot(H)`, где `H` — дескриптор для объекта `axes`, дает альтернативный способ задания подокна для текущего графика;
- `subplot('position',[left bottom width height])` — создает подокно с заданными нормализованными координатами (в пределах от `0.0` до `1.0`);
- `subplot(111)` и `clf reset` — удаляют все подокна и возвращают графическое окно в обычное состояние.

Следующий пример иллюстрирует применение команды `subplot`:

```
>> x=-5:0.1:5;
subplot(2,2,1),plot(x,sin(x))
subplot(2,2,2),plot(sin(5*x),cos(2*x+0.2))
subplot(2,2,3),contour(peaks)
subplot(2,2,4),surf(peaks)
```

В этом примере последовательно строится четыре графика различного типа, размещаемых в разных подокнах (рис. 1.30).

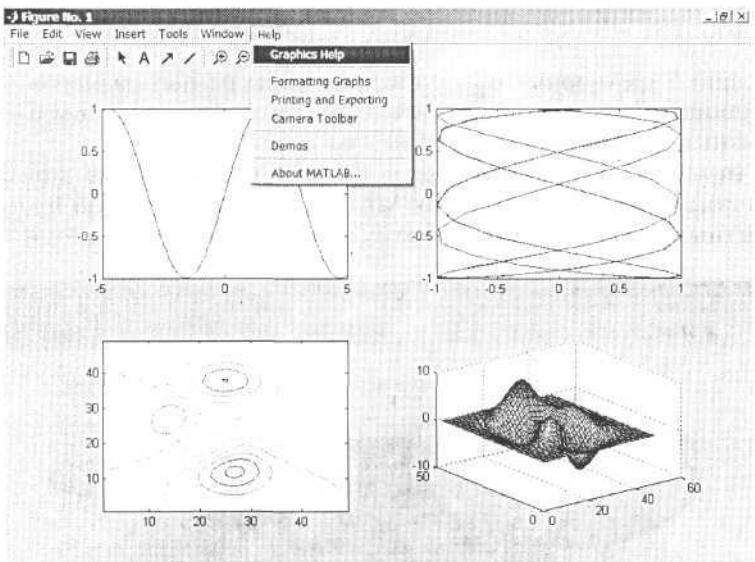

Рис. 1.30. Четыре графика различного типа в одном окне

Следует отметить, что для всех графиков возможна индивидуальная установка дополнительных объектов, например титульных надписей, надписей по осям и т. д. Данный пример дает хорошее представление о создании графиков разного типа. В том числе встроенного объекта `peaks` (пики).

1.12.12. Низкоуровневая дескрипторная графика

Дескрипторная (описательная) графика является низкоуровневой. Она создается с помощью графических объектов, имеющих свойства наследования и иерар-

хию. Для иллюстрации возможностей дескрипторной графики создадим файл ms1.m:

```
[x,y] = meshgrid([-2:.4:2]);
Z =sin(x.^2+y.^2);
fh = figure('Position',[350 275 400 300],'Color','w');
ah = axes('Color',[.8 .8 .8],'XTick',[-2 -1 0 1 2],...
'YTick',[-2 -1 0 1 2]);
sh = surface('XData',x,'YData',y,'ZData',Z,...
'FaceColor',get(ah,'Color')+.1,...
'EdgeColor','k','Marker','o',...
'MarkerFaceColor',[.5 1 .85])
```

В этом файле заданы три объекта: прямоугольник `fh` — объект класса `figure`, оси с метками `ah` — объект класса `axes` и трехмерная поверхность `sh` — объект класса `surface`. Теперь создадим второй файл — ms2.m:

```
h(1) = axes('Position',[0 0 1 1]); sphere
h(2) = axes('Position',[0 0 .4 .6]); peaks;
h(3) = axes('Position',[0 .5 .5 .5]); sphere
h(4) = axes('Position',[.5 0 .4 .4]); sphere
h(5) = axes('Position',[.5 .5 .5 .3]); cylinder([0 0 0.5])
set(h,'Visible','off'); set(gcf,'Renderer','painters')
```

Здесь задано 5 трехмерных объектов: три сферы разных размеров, поверхность `peaks` и цилиндр. Запустив последовательно эти файлы, получим сложный многокомпонентный объект, представленный на рис. 1.31.

Следует отметить, что дескрипторная графика рассчитана не столько на конкретных пользователей, использующих MATLAB как прикладную программу, сколько на опытных разработчиков программного обеспечения для этой системы.

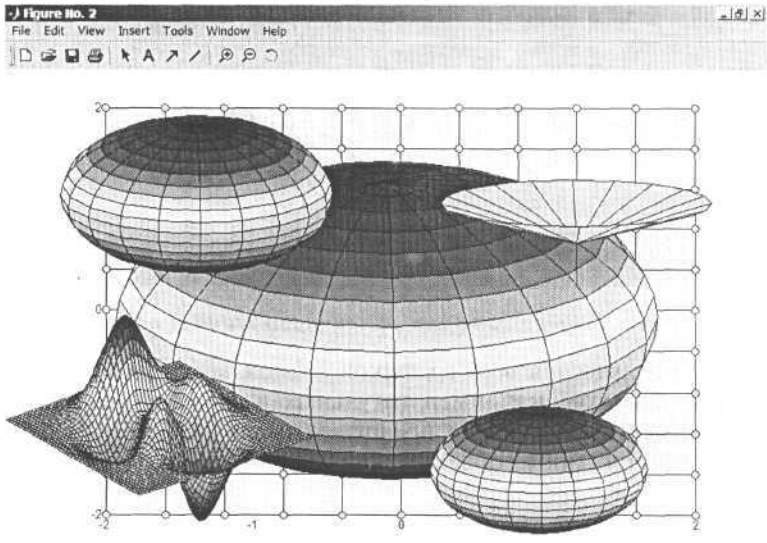

Рис. 1.31. Комбинированный рисунок, полученный при запуске файлов ms1 и ms2

1.12.13. Работа с камерой 3D-графики

В отличие от двумерных (2D) графиков форматирование трехмерных графиков содержит ряд дополнительных возможностей. Покажем их на простом примере построения 3D-графики с помощью следующих простых команд:

```
>> Z=peaks(40);
>> mesh(Z);
```

Здесь первая команда создает массив точек поверхности с помощью одного из ряда встроенных в ядро системы MATLAB готовых описаний таких поверхностей. Вторая команда просто строит эту поверхность по опорным точкам с использованием интерполяции для промежуточных точек. Таким образом создается цветная каркасная поверхность, как бы сотканная из разноцветных проволок. На рис. 1.32 показано построение этой поверхности вместе со специальной панелью инструментов трехмерной графики, названной в оригинале Camera (Камера).

Несмотря на множество кнопок пользование панелью инструментов 3D-графики достаточно просто, если представить себе, что вы смотрите на предмет через объектив фотокамеры. Наглядные рисунки на кнопках поясняют смысл их действия — это перемещение и вращение 3D-рисунков относительно тех или иных координатных осей, включение отображения перспективы, изменение цветовой схемы и др.

Рис. 1.33 показывает, что приемы форматирования двумерной графики можно использовать при работе с трехмерной графикой — вывод надписи на график, вывод легенды (кстати, теперь объемной) и шкалы цветов.

Для управления положением и вращением трехмерного графика можно использовать клавиши перемещения курсора. Эффект вращения графика иллюстрирует рис. 1.34, где показан график рис. 1.33 после его поворота при нажатой клавише →. В отличие от поворота мышью (также возможного), перемещение и повороты с помощью клавиш курсора при выбранном типе перемещения дают

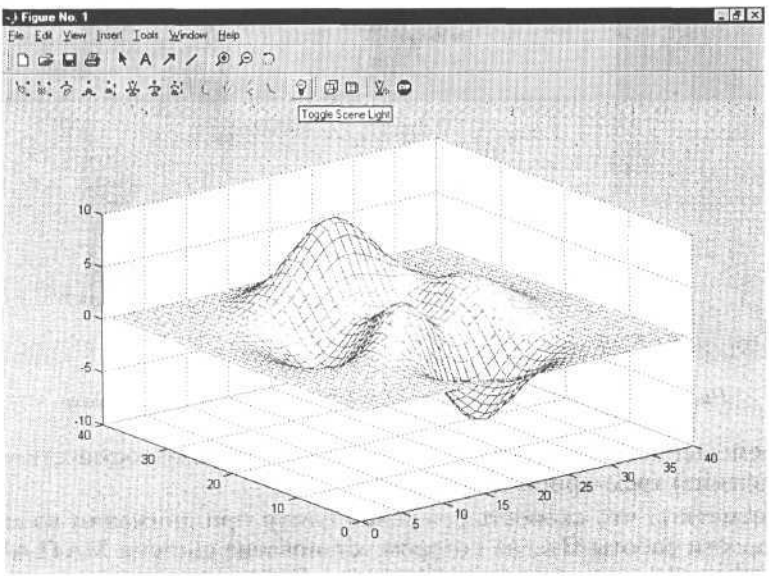

Рис. 1.32. Пример построения каркасного 3D-графика

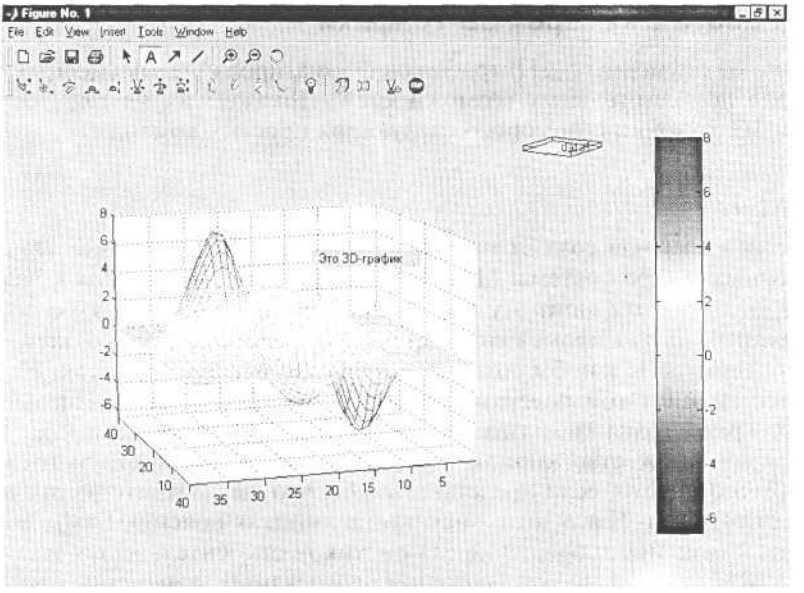

Рис. 1.33. Пример форматирования трехмерного графика

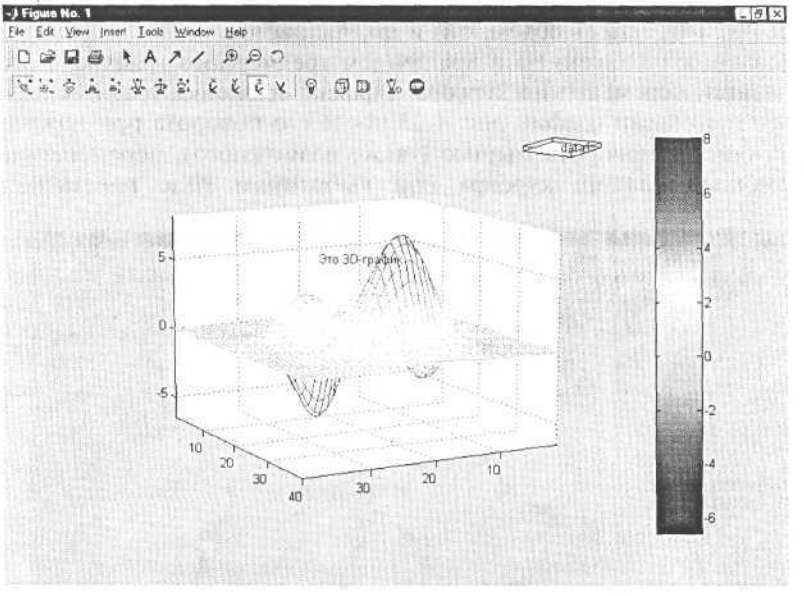

Рис. 1.34. Стоп-кадр вращения трехмерного графика

плавное перемещение или вращение фигуры. Таким образом осуществляется анимация (оживление) трехмерной графики.

Следует отметить, что скорость вращения фигур при анимации во многом зависит от скорости работы ПК, на котором установлена система MATLAB, от примененной видеокарты и ее настроек. На старых ПК она может оказаться довольно низкой.

1.13. Демонстрационные примеры MATLAB

В системах MATLAB 6.0/6.1 имеется демонстрационное окно Demos с обширным списком демонстрационных примеров. Рекомендуется просмотреть хотя бы те из них, которые относятся к сфере интересов пользователя. В новейшей версии MATLAB 6.5 окна Demos уже нет, но есть подобный раздел в справочной базе данных. Кроме того, доступ к демонстрационным примерам возможен из командной строки.

Рис. 1.35 дает пример работы с MATLAB 6.5 SP1 в среде Windows XP. Здесь в командной строке MATLAB задана команда `bench`, запускающая тест MATLAB на быстродействие. После тестирования по ряду параметров выдаются окна с гистограммой сравнительного быстродействия различных ПК и данного компьютера — в нашем случае это ПК на процессоре Intel Pentium 4 2,6 ГГц с технологией Hyper Threading.

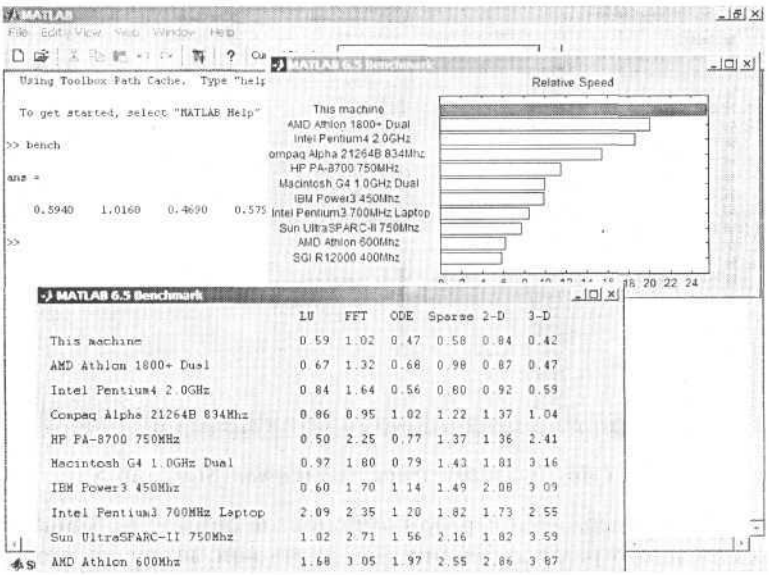

Рис. 1.35. Результаты выполнения теста bench системы MATLAB 6.5 SP1 в среде Windows XP

В ряде книг приводился этот тест на ПК с разными микропроцессорами. Как правило, по скорости вычислений эти ПК были в середине данных тестирования разных типов ПК. Тест на рис. 1.35 дан для новейшего (конец 2003 года) ПК на базе микропроцессора Pentium 4 Hyper Threading с рабочей частотой 2,6 ГГц. По производительности он уверенно вышел на первое место. Ради интереса тестирование было проведено и при отключенной технологии Hyper Threading (одновременное выполнение двух потоков). Неожиданно результат оказался совершенно одинаковым, как при отключении в BIOS этой технологии, так и при ее включении. Это говорит о том, что пока даже новейшая версия MATLAB 6.5 SP1 технологию Hyper Threading, похоже, не поддерживает.

1.14. Работа с основным пакетом расширения Simulink

1.14.1. Доступ к пакету расширения Simulink

Пакет блочного ситуационного моделирования Simulink является неразрывной частью системы MATLAB + Simulink. Он автоматически интегрируется с базовой системой MATLAB. Внешне это выражается появлением кнопки Simulink в панели инструментов (перед кнопкой ?) системы MATLAB (рис. 1.36). При нажатии этой кнопки открывается окно интегрированного браузера библиотек, показанное в левой части рис. 1.36.

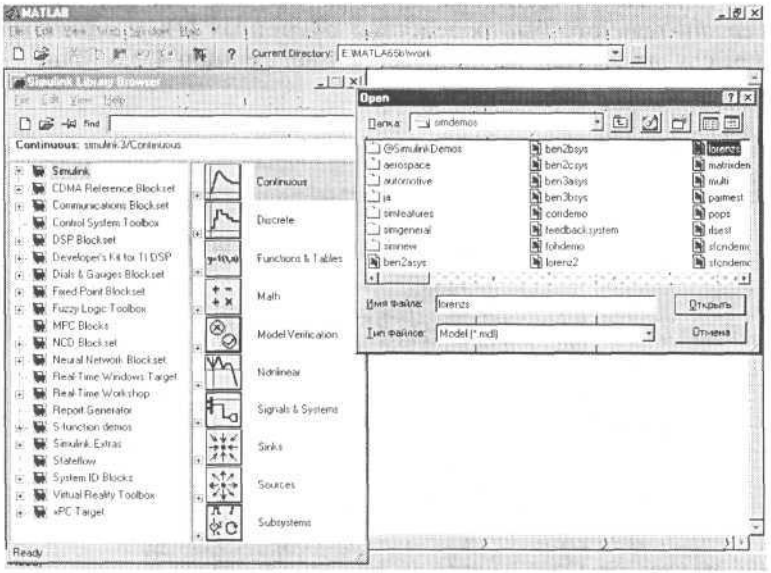

Рис. 1.36. Окно браузера библиотек Simulink 5

В окне браузера содержится дерево компонентов библиотек Simulink. Для просмотра того или иного раздела библиотеки достаточно выделить его мышью — в правой части окна Simulink Browser Library появится набор пиктограмм компонентов активного раздела библиотеки. На рис. 1.36 показан основной раздел библиотеки Simulink. Число разделов библиотеки различно у разных версий Simulink — у Simulink 5 оно возросло с 8 до 10. Новыми являются подразделы верификации моделей Model Verification и построения субсистем Subsystems. В остальном окна моделей версий Simulink 4 и 5 идентичны.

С помощью меню браузера или кнопок его панели инструментов можно открыть окно для создания новой модели или загрузить существующую модель. Работа с Simulink происходит на фоне открытого окна системы MATLAB, в котором нередко можно наблюдать за выполняемыми операциями — если их вывод предусмотрен программой моделирования.

1.14.2. Запуск моделей Simulink из среды MATLAB

Обычно Simulink запускается соответствующей кнопкой из панели инструментов окна MATLAB, что было описано выше, после чего все последующие дейст-

вия выполняются в среде MATLAB + Simulink. Можно также запустить Simulink, исполнив в командной строке MATLAB команду

```
» simulink
```

Для вывода полного перечня команд Simulink надо выполнить команду

```
» help simulink
```

Дополнительную информацию можно получить, используя команды `help blocks` и `help simdemos`. Первая команда дает информацию об основных библиотеках Simulink и примерах применения S-функций, а вторая выводит список демонстрационных примеров. Запуск этих примеров дает возможность практически познакомиться с возможностями пакета Simulink и оценить степень сложности систем и устройств, которые могут моделироваться с помощью этого пакета.

1.14.3. Особенности интерфейса Simulink

Интерфейс Simulink полностью соответствует стилю интерфейса типичных приложений Windows 95/98/NT/2000 (для Simulink 5 возможна работа и в Windows XP), в том числе интерфейсу системы MATLAB. В то же время он концептуально строг, чтобы не досаждать пользователю многочисленными «излишествами» стандартного интерфейса Windows 95/98/NT/2000. Меню системы содержит следующие пункты:

- File — работа с файлами моделей и библиотек (их создание, сохранение, считывание и печать);
- Edit — операции редактирования, работа с буфером обмена и создание подсистем;
- View — управление отображением панели инструментов и строки состояни;
- Simulation — управление процессом моделирования (старт, пауза, вывод окна настройки параметров моделирования;
- Format — операции форматирования модели (смена шрифтов, редактирование надписей, повороты блоков, использование тени от блоков, операции с цветами линий блоков, их фоном и общим фоном);
- Tools — управление видом анализа (в линейной области и в режиме реального времени RTW).

Первые три пункта главного меню содержат общепринятые для Windows-приложений команды и операции, поэтому мы не будем их обсуждать. Остальные будут рассмотрены по мере знакомства с системой Simulink.

Как уже отмечалось, вместе с рабочим окном Simulink выводится окно с перечнем разделов основной библиотеки компонентов. Это окно — важная часть интерфейса Simulink. Оно открывает доступ к другим пакетам компонентов (Blocksets&Toolboxes) и примерам их применения (Demos). Это дает пользователю возможность постепенно знакомиться с новыми областями применения Simulink.

1.14.4. Поиск и загрузка модели

Даже незнакомый с Simulink пользователь может быстро оценить возможности этого пакета, воспользовавшись интересными и поучительными примерами, входящими в поставку Simulink. Они находятся в папке MATLAB/TOOLBOX/SIMULINK/SIMDEMOS. Попутно отметим, что в папке MATLAB/SIMULINK располагаются служебные файлы.

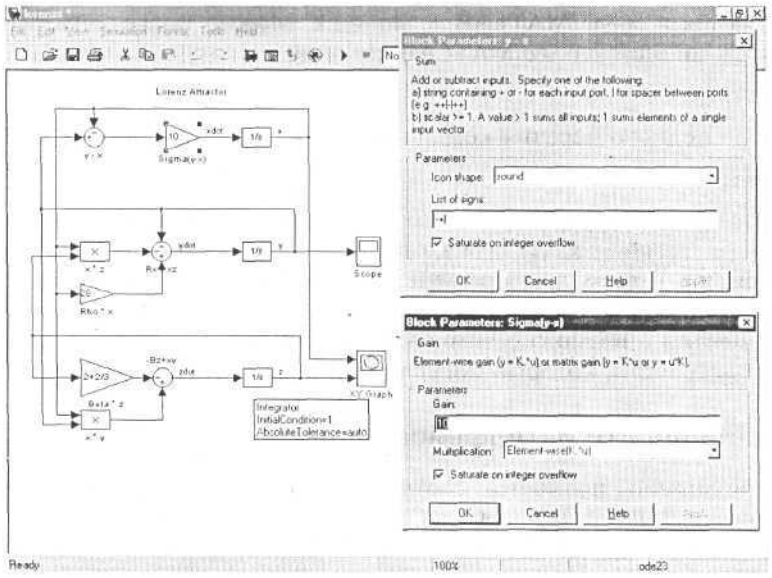

Рис. 1.37. Модель аттрактора Лоренца в окне редактирования пакета Simulink

Для загрузки примеров используется (как обычно) команда Open в меню File браузера библиотеки Simulink. В обычном окне загрузки файлов с названием Open надо войти в указанную папку и выбрать подходящий пример. При открытии нужного примера (в данном случае — файл lorenzs) появляется окно редактирования графической модели устройства (рис. 1.37).

Как можно заметить из примера, графическая модель содержит ряд блоков. Каждый блок имеет наглядное общепринятое обозначение в виде прямоугольника, треугольника и т. д. Блоки имеют входы и выходы и описываются различными математическими зависимостями. Блоки соединяются друг с другом линиями со стрелками, причем стрелка указывает направление от выходов одних блоков ко входам других. Имеются также текстовые комментарии и средства для вывода подсказок и открытия окон справочной системы.

1.14.5. Установка параметров компонентов модели

Для пользователя, имеющего хотя бы начальное представление о моделировании, вполне ясно, что любой компонент модели имеет какие-то параметры или хотя бы один параметр. Из самой графической модели не видно, какими параметрами обладает тот или иной компонент. Каким же образом узнать, что за параметры компонентов установлены в той или иной модели?

Для того чтобы вызвать окно модификации параметров компонента, нужно навести курсор мыши на изображение компонента и дважды щелкнуть левой кнопкой мыши. На рис. 1.34 справа от модели показаны два таких окна для компонентов с именами y – x и Sigma(y – x).

Кроме того, можно просто просмотреть параметры компонента. Для этого нужно, указав курсором мыши интересующий вас компонент, задержать курсор на 2—3 секунды. Под ним появится подсказка с указанием параметров этого компонента — на рис. 1.34 показана подсказка для компонента 1/s.

1.14.6. Установка параметров моделирования

Прежде чем запустить загруженную модель, стоит ознакомиться с установкой общих параметров моделирования. Для этого выполним команду `Simulation Parameters...` в меню Simulation окна Simulink. Появится окно установки параметров моделирования, показанное на рис. 1.38.

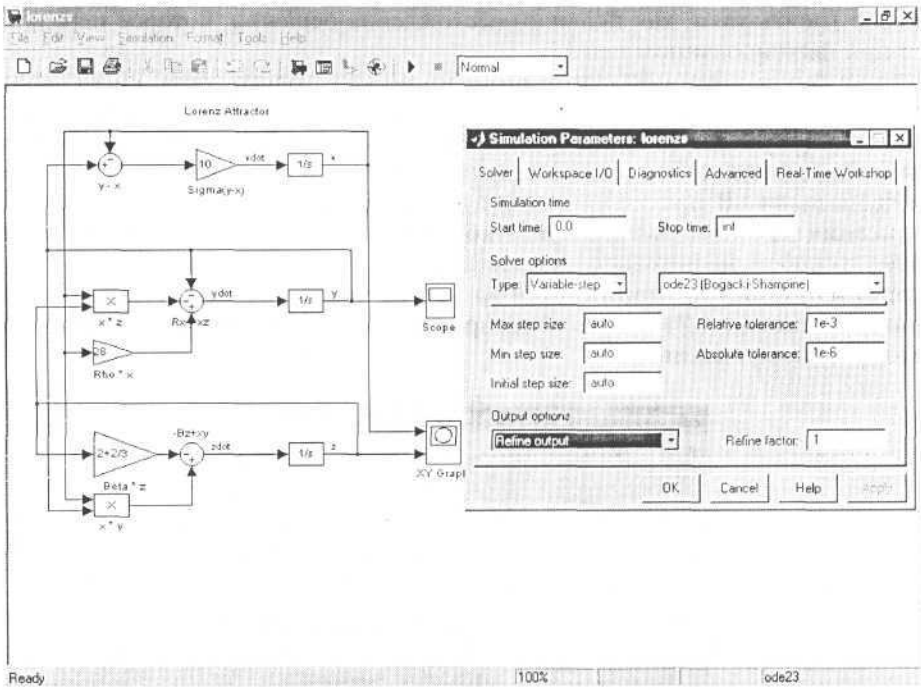

Рис. 1.38. Установка параметров модели аттрактора Лоренца

Это окно имеет ряд вкладок с довольно большим числом параметров. Мы постепенно рассмотрим их все. Но пока остановимся на вкладке, открытой по умолчанию, — Solver (Решатель). Эта вкладка позволяет установить параметры решающего устройства системы моделирования Simulink.

К числу важнейших параметров решателя относится время моделирования — Simulation time. Оно задается начальным временем Start time (обычно 0) и конечным временем Stop time (в нашем случае бесконечность inf). Равенство Stop time бесконечности означает, что моделирование будет происходить бесконечно долго, пока мы не прервем его. Однако в этом случае трудно получить различимые осциллограммы работы устройства, поэтому рекомендуется задавать конечные значения Stop time.

Стоит отметить, что время моделирования — величина довольно условная. Не следует думать, что Stop time = 50 означает моделирование в течение 50 секунд. Точного соответствия между временем моделирования в секундах и заданным значением нет. Реальное время моделирования сильно зависит от быстродействия компьютера, на котором выполняется моделирование.

Первостепенное значение имеют две опции решателя в поле Solver options: тип решения и метод решения. Возможно два типа решения:

- Variable-step solvers — решение с переменным шагом;
- Fixed-step solvers — решение с фиксированным шагом.

Как правило, лучшие результаты дает решение с переменным шагом (обычно по времени, но не всегда). В этом случае шаг автоматически уменьшается, если скорость изменения результатов в процессе решения возрастает. И, напротив, если результаты меняются слабо, шаг решения автоматически увеличивается. Это исключает (опять-таки, как правило) расхождение решения, которое нередко случается при фиксированном шаге.

Метод с фиксированным шагом стоит применять только тогда, когда фиксированный шаг обусловлен спецификой решения задачи, например, если ее цель заключается в получении таблицы результатов с фиксированным шагом. Этот метод дает неплохие результаты, если поведение системы описывается почти монотонными функциями. В противном случае шаг времени придется сильно уменьшать для описания наиболее быстрых участков изменения результатов моделирования, что ведет за собой значительное возрастание времени моделирования.

Вторая из указанных опций — выбор метода моделирования. Рис. 1.39 показывает окно установки параметров моделирования отдельно с открытым меню выбора метода моделирования.

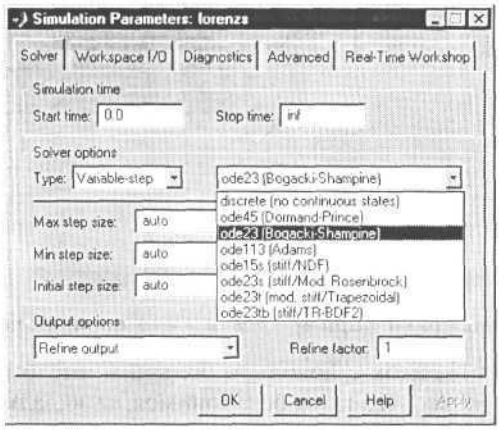

Рис. 1.39. Установка метода моделирования

Для решения дифференциальных уравнений можно выбрать следующие методы: discrete (дискретный), ode45, ode23 (три варианта, включая метод Розенброка), rk45 (метод Дорманда—Принса), ode113 (метод Адамса) и ode15s. Методы, в наименовании которых имеется слово stiff, служат для решения жестких систем дифференциальных уравнений.

Следующие три параметра — значения опции auto — обычно задаются автоматически, но в особых случаях их можно ввести явно.

- Max step size — максимальный шаг интегрирования системы однородных дифференциальных уравнений;
- Min step size — минимальный шаг интегрирования;
- Initial step size — начальный шаг интегрирования;

Важен и такой параметр моделирования, как точность интегрирования:

- Relative tolerance — относительная погрешность интегрирования;

- Absolute tolerance — абсолютная погрешность интегрирования.

По умолчанию они заданы, соответственно, равными 10^{-3} и 10^{-6}. Если, например, графики результатов моделирования выглядят составленными явно из отдельных фрагментов, это указывает на необходимость уменьшения указанных значений погрешности. Однако слишком малые погрешности могут вызвать значительное увеличение времени вычислений. Не оптимально выбранные значения погрешности (как очень малые, так и очень большие) могут вызвать неустойчивость и даже «зацикливание» процесса моделирования.

С остальными параметрами и вкладками окна параметров моделирования мы познакомимся в дальнейшем.

1.14.7. Запуск процесса моделирования

В конце панели инструментов Simulink находятся две важные кнопки управления моделированием. Одна из них, в виде черного треугольника (Start/Pause Simulation), запускает или приостанавливает начатый процесс моделирования, а другая, в виде черного квадратика (Stop), останавливает его. Все, что нужно для запуска моделирования, — это нажать кнопку с изображением треугольника. Рис. 1.40 показывает результат запуска выбранной модели. Вместо кнопок можно использовать команды Start и Pause в меню Simulation окна модели.

В данном случае результаты моделирования представлены в виде довольно сложного и неординарного фазового портрета колебаний, построенного с помощью виртуального графопостроителя, и осциллограммы временной зависимости колебаний, полученной с помощью виртуального осциллографа.

Результат моделирования показывает, что даже в такой сравнительно простой нелинейной системе, каковой является аттрактор Лоренца, возникают сложные и отчасти хаотические колебания. Вы можете познакомиться подобным образом с множеством других демонстрационных примеров применения пакета расширения Simulink.

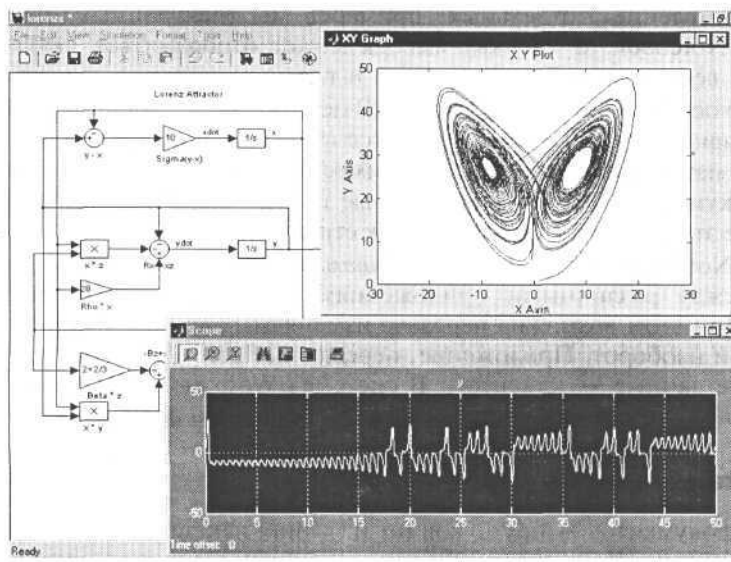

Рис. 1.40. Результаты моделирования аттрактора Лоренца

Глава 2. Подготовка электронных документов

2.1. Пакет расширения Notebook

2.1.1. Назначение приложения Notebook

Notebook (Блокнот) — это специальное приложение системы MATLAB, позволяющее готовить с помощью текстового процессора (редактора) Microsoft Word 6.0/7.0/8.0/2000/XP [40] электронные документы (книги) с полноценным текстовым описанием, с различным стилевым оформлением и «живыми» примерами. Таким образом, это средство — очередное достижение разработчиков MATLAB в визуализации всех этапов работы с системой. Во всех новейших версиях MATLAB используется Notebook версии 1.5.2. Далее рассматривается работа с этим приложением в среде операционной системы Microsoft Window XP.

Как вы могли убедиться из первой главы, основой MATLAB является решатель математических задач с довольно скромным интерфейсом и скромными возможностями стилизации текстов. В части последнего неоспоримым преимуществом обладают текстовые процессоры класса Word, которые позволяют в рамках одного документа создавать описания с любым стилем, цветом и размером символов, включать в это описание рисунки и иллюстрации, математические формулы и графики функций. Однако эти объекты не могут видоизменяться при изменении исходных данных описываемых задач. Можно сказать, что текстовые процессоры позволяют готовить обычные «мертвые» книги по математическим расчетам.

Notebook обеспечивает объединение возможностей текстовых процессоров класса Word с возможностями системы MATLAB путем включения в произвольные тексты документов, создаваемых этими редакторами, действующих ячеек ввода и вывода. При этом изменение исходных данных в ячейках ввода ведет к изменению результатов вычислений в связанных с ними ячейках вывода. Это и обеспечивает «оживление» отдельных примеров и электронных книг на базе приложения Notebook. В ячейках вывода может отображаться любая информация — числа, векторы, матрицы, рисунки и т. д.

Здесь надо особо отметить, что MATLAB не позволяет готовить документы, в которых размещены рисунки — они создаются в отдельных окнах. Нельзя в MATLAB отображать и математические формулы в их естественном виде. Это затрудняет чтение таких документов. Документы, которые готовит приложение Notebook, могут содержать все эти элементы и выглядят как страницы книги.

В основе Notebook лежит механизм *динамической связи* (DDE — Dynamic Data Exchange) между различными приложениями в операционных системах Windows 95/98. При этом возможна передача изменяемых данных из одного приложения в другое и наоборот. Приложение, передающее данные, называют сервером, а принимающее данные — клиентом. В системе «Word — MATLAB», по существу реализованной в Notebook, обе программы могут играть роли сервера и клиента.

2.1.2. Инсталляция Notebook

Создание документов — ноутбуков (от названия пакета) в MATLAB решено довольно оригинально. В частности, в ходе этого процесса в явной форме отсутствует процесс создания объектной связи между приложениями с помощью команды

Insert Object (Вставка объекта). Такая связь устанавливается автоматически — стоит лишь загрузить файл с именем readme.doc из папки PC в папке NOTEBOOK системы MATLAB. Когда Word (в нашем случае это Word 2003) выдаст сообщение о том, что текст загружаемого документа содержит макросы, надо подтвердить их загрузку — иначе приложение Notebook работать не будет.

Можно также дать команду `notebook` из окна MATLAB — при этом произойдет загрузка текстового редактора Word той версии, которая установлена на данном ПК. Однако в первый раз надо выполнить установку связи MATLAB с Word, исполнив команду:

```
>> notebook -setup
Welcome to the utility for setting up the MATLAB Notebook
for interfacing MATLAB to Microsoft Word
Choose your version of Microsoft Word:
[1] Microsoft Word 97
[2] Microsoft Word 2000
[3] Microsoft Word 2002 (XP)
[4] Exit, making no changes
```

Теперь надо указать версию Word в виде номера:

```
Microsoft Word Version: 3
```

Диалог нормально завершается выводом сообщения:

```
Notebook setup is complete.
```

Приведенные далее примеры даны для Microsoft Office Word 2003. Он хотя не входит в указанный список версий, тем не менее благополучно стыкуется с MATLAB при указании версии [3] (это не литературная ссылка).

Более простым способом инсталляции приложения Notebook является загрузка в Word файла readme.doc из папки PC в папке NOTEBOOK системы MATLAB. Начало загрузки этого файла представлено на рис. 2.1. Здесь видно окно загрузки файла и сообщение о том, что он содержит макросы.

Этот файл (как и любой файл документа класса Notebook) обеспечивает следующее:

- запускает систему MATLAB;
- устанавливает динамическую объектную связь DDE между MATLAB и Word;
- задает макросы для обработки специальных типов ячеек Notebook;
- создает новое меню Notebook в строке меню Word;
- поддерживает стили ячеек Notebook и текста Word.

Поскольку файл readme.doc содержит нужные для работы Notebook макросы, то нужно исполнить команду «Не отключать макросы». При этом содержимое файла будет видно в окне просмотра окна загрузки файла (рис. 2.2).

Нажав кнопку Открыть, можно загрузить файл в текстовый процессор Word. Однако, если система MATLAB не была запущена, то в окне с текстом файла появится сообщение об этом, представленное на рис. 2.3. Возможно также появление и окна с предложением обеспечить доступ к файлу запуска MATLAB. Во избежание этих осложнений проще запустить MATLAB перед загрузкой файла readme.doc.

После успешного запуска MATLAB обеспечена совместная работа этой системы с приложением Notebook.

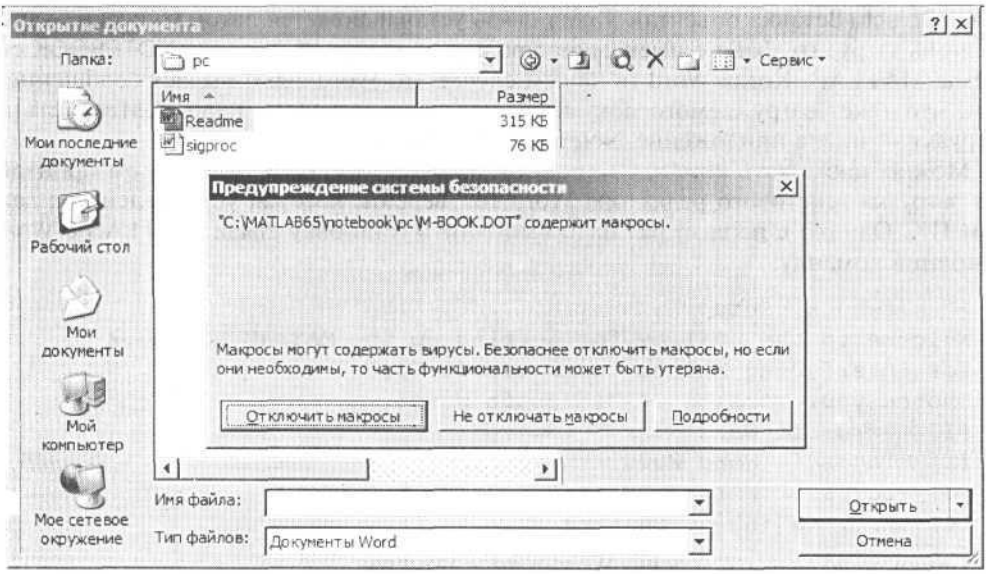

Рис. 2.1. Начало загрузки файла readme.doc в текстовый процессор Word

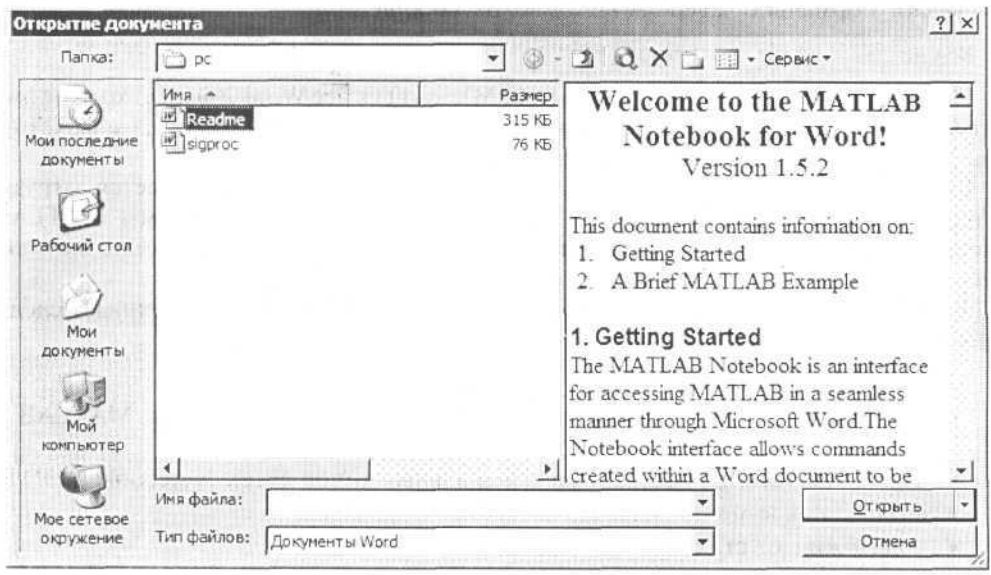

Рис. 2.2. Просмотр загружаемого файла readme.doc

2.2. Начало работы с Notebook

2.2.1. Изменения интерфейса текстового процессора Word

В начале процесса загрузки файла вы увидите момент загрузки системы MATLAB — появление рисунка с ее логотипом. В конце процесса загрузки появится текст файла readme, как показано на рис. 2.4.

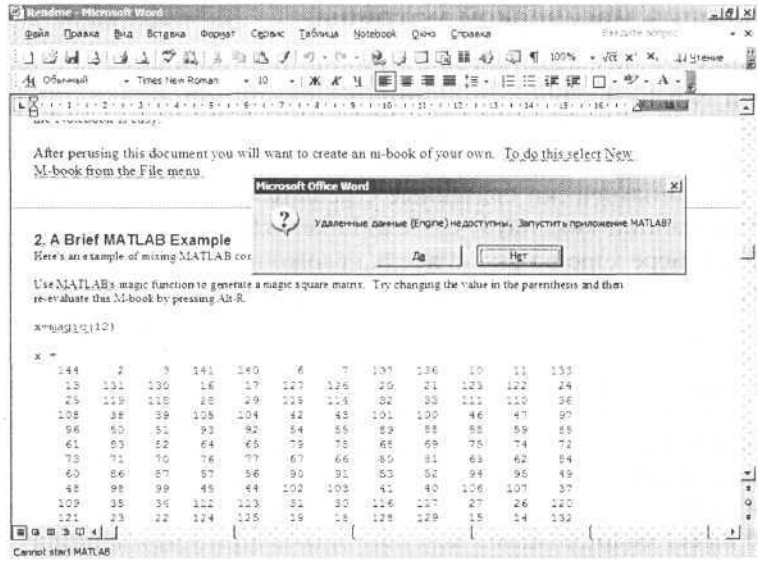

Рис. 2.3. Текст загруженного файла readme.doc с сообщением о запуске MATLAB

Рис. 2.4. Окно текстового процессора Word с загруженным файлом readme

Внимательный читатель тут же отметит некоторые необычные свойства окна текстового процессора Word. Первое, что бросается в глаза при сравнении рис. 2.1 и 2.4, это появление в строке меню нового меню Notebook, которое на рис. 2.1 отсутствует. Это меню содержит множество команд, относящихся к приложению Notebook, созданному на основе текстового процессора Word. В раскрытом виде меню Notebook показано на рис. 2.4. Кроме того, в позиции Файл меню Word появляется новая позиция New M-book, открывающая окно для подготовки новых Notebook-документов.

2.2.2. Эволюция магической матрицы

Файл readme.doc содержит несколько наглядных примеров для демонстрации возможностей Notebook. Для оценки этих возможностей достаточно просмотреть файл и остановиться на разделе «A Brief MATLAB Example». Часть документа (рис. 2.5), заключенная в жирные квадратные скобки, представляет собой ячейки ввода и вывода, динамически связанные с решателем системы MATLAB. Скобки можно убрать, исполнив команду Hide Cell Markers в новой позиции меню Notebook.

В данном примере хорошо видна ячейка ввода, в которой определена операция задания большой магической матрицы:

```
x=magic(12)
```

Под ней показана ячейка вывода, реализующая вывод магической матрицы размера 12 × 12. На рис. 2.5 виден также обычный англоязычный текст, набранный в редакторе Word разными стилями. Напоминаем, что под стилем понимается совокупность параметров текста: используемые наборы шрифтов, их размеры и стили, цвета символов, межстрочные расстояния, отступы абзацев и другие параметры. Word имеет обширные возможности в создании текстов различного стиля.

Теперь покажем, что ячейки MATLAB в тексте документа способны к изменению. Для этого обычным путем поместим маркер ввода в ячейку ввода и заменим параметр функции `magic`, равный 12, на значение 4. Не выводя маркера из этой ячейки, нажмем одновременно клавиши Ctrl и Enter. Вы тут же увидите, что ячейка вывода изменится — вместо магической матрицы размера 12 × 12 появится магическая матрица меньшего размера — 4 × 4. Это и есть пример эволюции ячеек Notebook (рис. 2.6).

Далее вы увидите, что имеется также возможность вводить команды MATLAB в середину строк, создавать объединенные ячейки и осуществлять эволюцию как отдельных ячеек, так и всех ячеек документа одновременно.

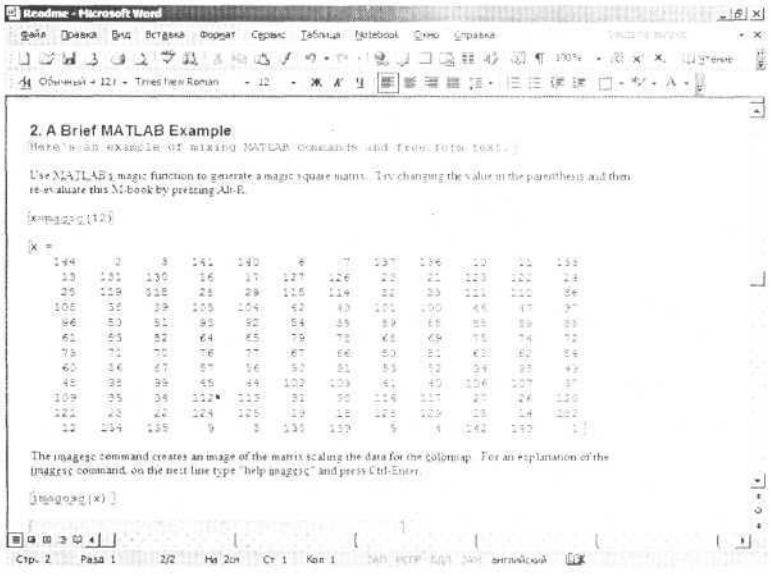

Рис. 2.5. Пример формирования магической матрицы размером 12 × 12

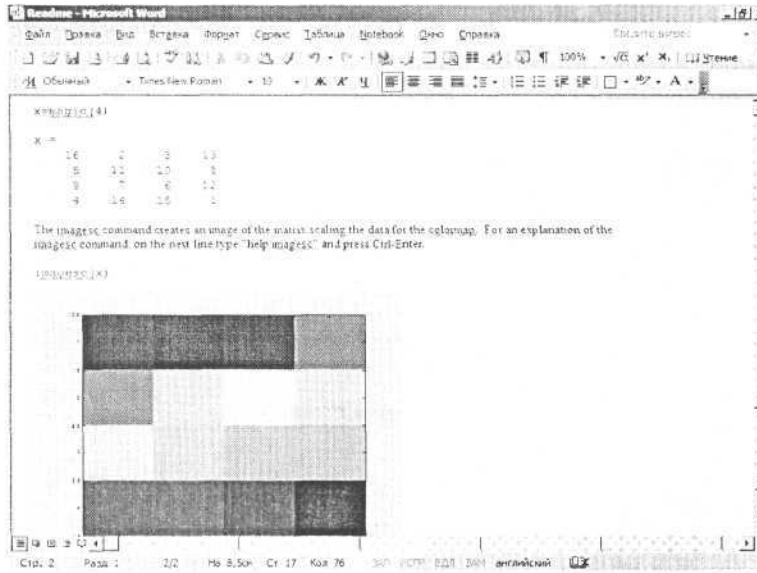

Рис. 2.6. Пример изменения размера магической матрицы

2.2.3. Эволюция рисунка

Пролистав документ readme чуть дальше, мы увидим команду `images(x)`, которая дает графическое представление содержимого магической матрицы. При этом каждый ее элемент отображается квадратом с функциональной окраской, зависящей от значения элемента матрицы. Все это для матрицы размера 4 × 4 показано на рис. 2.7.

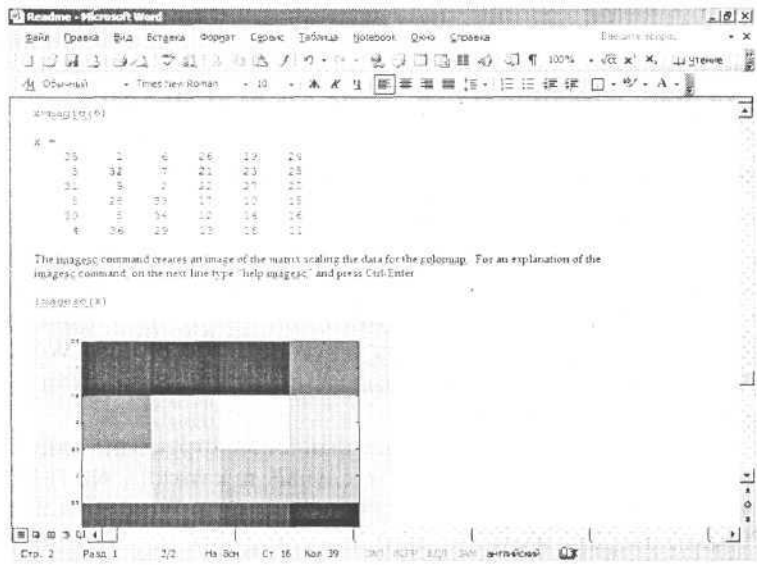

Рис. 2.7. Магическая матрица размера 4 × 4 и ее графическое представление

Теперь измените размер матрицы, например на 6 × 6, и создайте новую матрицу так, как это было описано выше. Далее поместите в ячейку с командой `images` маркер ввода и нажмите клавиши Ctrl + Enter. На рис. 2.8 видно, как изменилась картинка — число квадратиков на ней возросло. Это и есть пример эволюции рисунка.

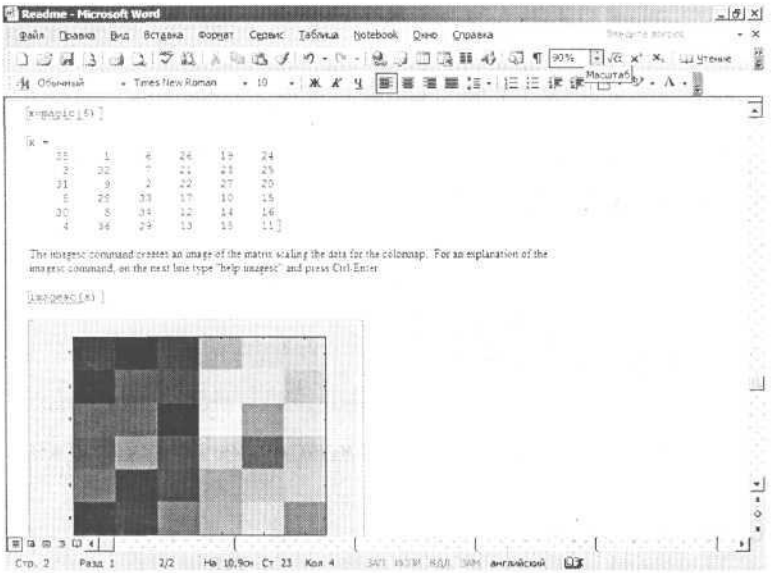

Рис. 2.8. Магическая матрица размера 6 × 6 и ее графическое представление

Таким образом, вы убедились, что эволюция при изменении исходных данных возможна для выходных ячеек разного типа — в нашем случае для вывода матрицы и ее графического представления. И все это происходит на фоне обычного текстового оформления документа.

2.3. Создание документов класса Notebook

2.3.1. Открытие нового документа класса Notebook

Для создания своего Notebook откройте меню Файл текстового процессора при загруженном в него файле readme. Вы обнаружите в этом меню новую команду New M-book (Создание новой М-книги) — рис. 2.9. Выполнив ее, вы увидите пустое окно, в котором можно вводить обычный текст. При этом правила ввода текста полностью совпадают с таковыми для текстового процессора Word. В частности, вы можете задавать любые стили и выделения в текстовой части создаваемого Notebook.

Иногда из-за неточностей в установке файловой структуры такой способ не дает положительного результата. На этот случай в каталоге NOTEBOOK предусмотрено два «пустых» файла шаблонов: M-book.dot (для Word 97 и более поздних версий) и Mbook95.dot (для Word 95). Загрузив эти файлы, можно установить Word для подготовки документа в виде Notebook, причем окно документа будет пустым и готовым для задания нового документа в формате шаблона.

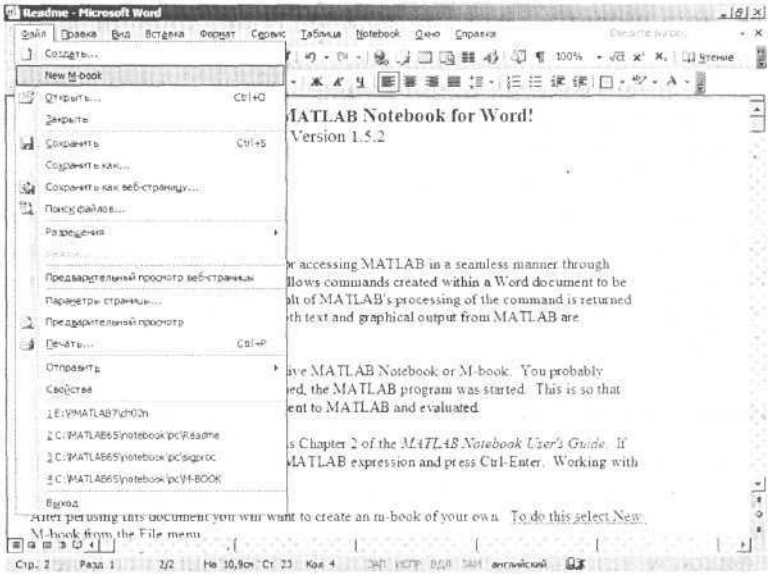

Рис. 2.9. Подготовка к созданию нового документа класса Notebook

2.3.2. Пример создания документа класса Notebook

На рис. 2.10 показан созданный Notebook, который поясняет, как выполнить построение графика трех функций. Вначале в нем введен абзац текста с описанием создания ячейки ввода.

Для создания ячейки ввода надо установить маркер ввода на свободную строку и исполнить команду Define Input Cell в меню Notebook текстового процессора

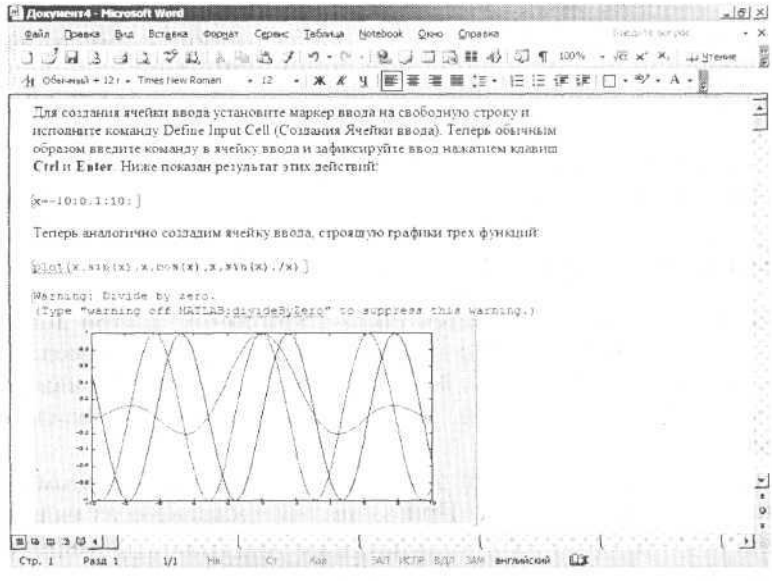

Рис. 2.10. Созданный новый Notebook

Word. После этого вводится текст команды MATLAB, который затем фиксируется одновременным нажатием клавиш Ctrl и Enter (или исполнением команды Evaluate Cell из меню Notebook). Если ячейка ввода должна порождать ячейку вывода, то она тут же появится. В первой команде `x=-10:0.1:10;` оператор «;» блокирует вывод, поэтому ячейка вывода не появляется.

Далее аналогично введем команду построения графика трех функций (см. рис. 2.10). Теперь, после нажатия клавиш Ctrl + Enter, тут же появится ячейка вывода с графиком трех функций. Установив в нее курсор мыши и дважды щелкнув левой кнопкой, можно наблюдать выделение графика и появление в его рамке прямоугольников, за которые можно «уцепиться» курсором мыши, чтобы изменить размеры графика.

2.3.3. Ячейки ввода MATLAB в тексте Word

Ячейки ввода MATLAB можно ввести прямо в текст документа. Для этого текст ячейки ввода набирается в Word как обычный текст. Затем он выделяется (с помощью мыши или клавиш перемещения по горизонтали при нажатой клавише Shift) и фиксируется нажатием клавиш Ctrl + Enter или исполнением команды Evaluate Cell из меню Notebook.

2.3.4. Преобразование текстов Word в ячейки ввода

Иногда желательно создать Notebook из самого обычного файла редактора Word. Для этого надо создать новый Notebook и загрузить нужный файл, используя команду Файл (File) меню Вставка (Insert). В созданный таким образом шаблон Notebook можно добавить ячейки ввода MATLAB.

2.3.5. Математические формулы в Notebook

В документы Notebook можно вводить математические формулы любого вида. Для этого достаточно воспользоваться встроенным в Word редактором математических формул. Наряду с файлом readme.doc в новых версиях MATLAB обычно поставляется файл sigprog.doc, содержащий документ с описанием свертки матриц для обработки сигналов. Начало этого документа представлено на рис. 2.11 и дает прекрасное представление о подготовке документов с математическими формулами.

2.3.6. Сохранение документов класса Notebook

Созданный Notebook записывается так же, как любой другой документ Word. Если нужно сохранение Notebook с заданным именем, то надо исполнить команду Save As (Сохранить как) из меню File (Файл) главного меню. Появится стандартное окно записи файла (рис. 2.12), и его можно записать как файл документа с расширением .doc.

Если использовался шаблон, то это будет окно шаблона. В этом случае файл записывается с расширением .dot. При закрытии последнего из ряда документов приложение Notebook создает окно с запросом о том, нужно ли закрыть MATLAB. Для редактирования уже созданного Notebook достаточно загрузить его с помощью команды Открыть из меню Файл.

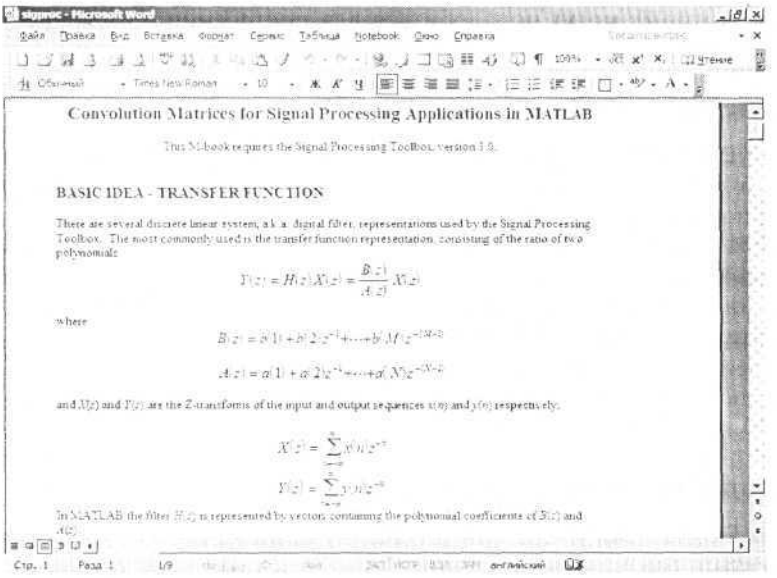

Рис. 2.11. Документ класса Notebook с математическими формулами

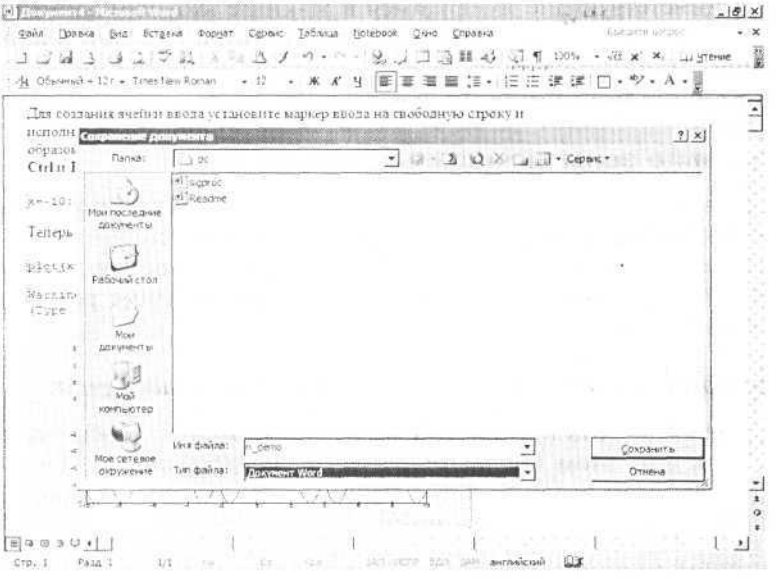

Рис. 2.12. Запись Notebook на жесткий диск

2.4. Меню Notebook

Возможности создания М-книг, или документов класса Notebook, намного шире описанных выше. Далее дается полное описание операций и команд меню Notebook, появляющегося в строке меню текстового процессора Word. Их применение позволяет создавать М-книги или документы типа Notebook с особыми

свойствами, которые могут понадобиться для построения обучающих или демонстрационных программ.

2.4.1. Создание ячейки ввода

Команда Define Input Cell (Alt + D) формирует ячейку ввода. Если маркер ввода находится в начале абзаца, то весь абзац преобразуется в ячейку ввода. Если команда используется при наличии выделенного фрагмента текста, то этот фрагмент преобразуется в ячейку ввода. Ячейка автостарта (см. ниже) также преобразуется в ячейку ввода, если в ней размещен маркер ввода. Пустая строка с маркером ввода становится ячейкой ввода после набора нужного выражения и его фиксации нажатием клавиш Ctrl + Enter.

Текст ячейки ввода обрамляется жирными квадратными скобками — [Текст]. Используется стиль Input с жирным шрифтом Courier New темно-зеленого цвета размером 10 пунктов.

2.4.2. Создание ячейки автостарта

Команда Define AutoInit Cell создает ячейку автостарта. Это ячейка, которая будет исполняться сразу после загрузки М-книги в текстовый процессор Word. Тут уместно отметить, что обычные ячейки (без автостарта) не эволюционируют без специальной команды. Ячейки автостарта обязательно эволюционируют и выдают результаты, соответствующие имеющимся в М-книге входным данным.

Правила применения этой команды те же, что были описаны для предшествующей команды. Текст соответствующей ячейки стиля AutoInit имеет темно-синий цвет (шрифт Courier New, размер 10 пунктов).

2.4.3. Создание зоны вычислений

Команда Define Calc Zone превращает выделенный текст (с ячейками ввода и вывода) в некоторую *зону вычислений*, решающую определенную задачу. Таких зон в М-книге может быть много, и они могут использоваться для решения ряда задач. Примером, где такие зоны полезны, являются сборники различных задач с действующими примерами.

2.4.4. Преобразование ячеек MATLAB в обычный текст

Команда Undefine Cells преобразует выделенные ячейки в обычный текст. Обрамления ячеек при этом убираются, а текст представляется стилем Обычный (Normal). Если выделений текста нет, а маркер ввода стоит на ячейке MATLAB, то именно эта ячейка преобразуется в текст.

2.4.5. Удаление ячеек вывода

Команда Purge Output Cell удаляет ячейки вывода. Если надо удалить одну ячейку вывода, достаточно разместить в ней маркер ввода и исполнить данную команду. Для удаления нескольких ячеек их надо предварительно выделить. При этом можно выделить и весь текст М-книги, содержащей ячейки вывода.

2.4.6. Создание многострочной ячейки ввода

Команда Group Cells объединяет все ячейки ввода в выделенной части документа в *группу* ячеек ввода. При этом выделенный текст размещается после этой группы,

за исключением той части текста, которая размещена до первой ячейки. Ячейки вывода, имеющиеся в выделенном тексте, устраняются. Если первая ячейка ввода имеет статус ячейки автостарта, то такой статус приобретают все ячейки группы. Группе ячеек можно придать этот статус и с помощью команды AutoInit Cell.

2.4.7. Преобразование группы ячеек в ячейки ввода

Команда Ungroup Cells преобразует группу ячеек в обычные ячейки ввода или ячейки автостарта. Ячейки вывода, связанные с преобразуемыми ячейками, удаляются. Для преобразования надо указать группу путем размещения маркера ввода в конце строки, завершающей группу, или в ячейке вывода, связанной с выделенной группой ячеек.

2.4.8. Управление показом маркеров

Как уже указывалось, обычно ячейки ввода и вывода MATLAB отмечаются жирными серыми квадратными скобками (маркерами), отличными от обычных квадратных скобок. Маркеры видны только на экране дисплея. Команда Hide/Show Cell Markers позволяет убрать или, напротив, включить показ маркеров. При печати М-книг маркеры не печатаются независимо от того, видны они на экране дисплея или нет.

2.4.9. Пуск оценки ячеек

Команда Evaluate Cell (Оценка Ячейки) направляет текущую ячейку ввода или группу ячеек в решатель MATLAB для проведения необходимых вычислений или обработки данных. Этот процесс принято называть *оцениванием* (evaluate) или попросту *вычислением* ячейки. Результат, в том числе и в виде сообщений об ошибках (они выводятся красным цветом), направляется в ячейку вывода текстового редактора Word. Ячейка ввода (или группа ячеек) считается текущей, если маркер ввода находится в ее поле, в конце ее строки или в связанной с ней ячейке вывода. Выделенная ячейка также считается текущей.

2.4.10. Пуск оценки зоны

Команда Evaluate Calc Zone (Оценка вычисляемой зоны) вызывает пересчет текущей зоны вычислений. Зона считается текущей, если в ней размещен маркер ввода. Для каждой ячейки ввода в зоне создается ячейка вывода.

2.4.11. Пуск оценки всей М-книги

Команда Evaluate M-book вызывает пересчет всех ячеек ввода или групп ячеек для текущей М-книги. Текущей является та книга, текст которой (или его часть) виден в активном окне текстового процессора Word. Заметим, что Word может работать с несколькими М-книгами поочередно. При этом все они загружены в свои окна, но лишь одно окно (видимое на экране) является активным и текущим.

Вычисления начинаются от начала М-книги и продолжаются до ее конца. Результаты вычислений поступают в ячейки вывода, а если каких-либо выходных ячеек еще нет, то они создаются. Рекомендуется применять эту команду в конце отладки и редактирования М-книги, чтобы соблюсти соответствие между модифицированными ячейками ввода и их ячейками вывода.

2.4.12. Циклическая оценка

Выделенную ячейку ввода или группу ячеек можно выполнить циклически с помощью команды Evaluate Loop (Циклическая оценка). Рис. 2.13 показывает подготовку к такой оценке. В первой строке ввода задано значение переменной `i=0`. Во второй строке заданы вычисления `i=i+1` и `magic(i)`. Первоначально, таким образом, выдается магическая матрица размера 1 × 1.

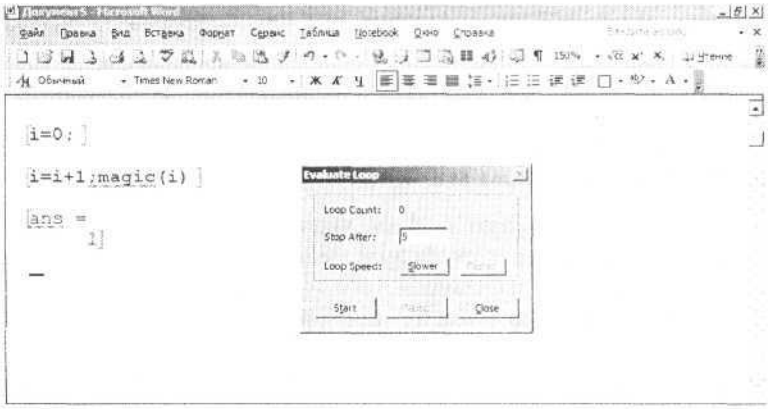

Рис. 2.13. Подготовка к циклической оценке заданной ячейки ввода

В правой части окна рис. 2.13 видно диалоговое окно Evaluate Loop, которое появляется при исполнении команды Evaluate Loop (Циклическая оценка). В нем имеется поле Stop After (Остановить после), задающее число циклов оценки. Кнопки Slower (Медленно) и Faster (Быстро) задают (условно) медленную и быструю оценку текущей ячейки ввода — в нашем случае это вторая ячейка из двух, показанных на рис. 2.13.

Запустив циклический просмотр нажатием кнопки Start, можно наблюдать построение матриц постоянно увеличивающегося размера `ii`. По завершении циклов можно увидеть результаты, показанные на рис. 2.14. На ваших глазах прои-

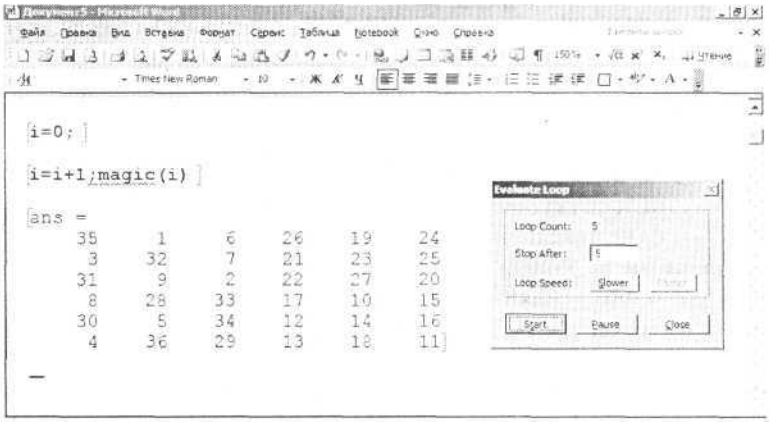

Рис. 2.14. Конец циклической оценки

зойдет вывод магических матриц с размером от 1 × 1 до 6 × 6, так что в конечном счете будут выведена такая матрица размера 6 × 6.

Можно приостановить циклическую оценку (если успеете!) нажатием кнопки Pause. А чтобы убрать окно управления циклической оценкой, следует нажать его кнопку Close (Закрыть).

2.4.13. Вывод окна MATLAB на передний план

Команда Bring MATLAB to Front выводит на передний план командное окно MATLAB (рис. 2.15).

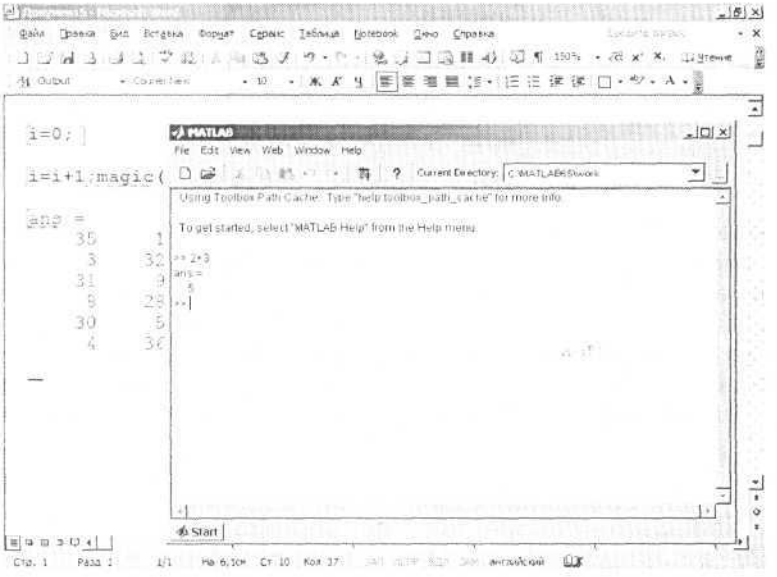

Рис. 2.15. Пример вывода окна системы MATLAB на передний план

Это может понадобиться, например, для выполнения тех или иных вычислений в ходе создания документа класса Notebook.

2.4.14. Установка опций Notebook

Последняя команда меню Notebook — Notebook Options — выводит окно установки опций, в основном связанных с параметрами вывода результатов вычислений. Это окно показано на рис. 2.16.

Окно содержит две панели. Первая панель — Numetic Format — позволяет задать вывод чисел в различных форматах. Для изменения формата служит выпадающий список, содержащий набор возможных форматов. Форматы представления чисел мы обсуждали в разделе «Форматы чисел — команда format» Урока 1. Опция Loose добавляет пробел между ячейками, а опция Compact — устраняет его. В последнем случае представление ячеек получается более компактным.

Следующая панель — Figure Optoins — содержит опции отображения графики и рисунков. Установка флажка Embed Figures in M-book обеспечивает размещение рисунков прямо в тексте книги — как разновидность ячеек вывода. Снятый флажок означает, что рисунки будут выведены в отдельных окнах.

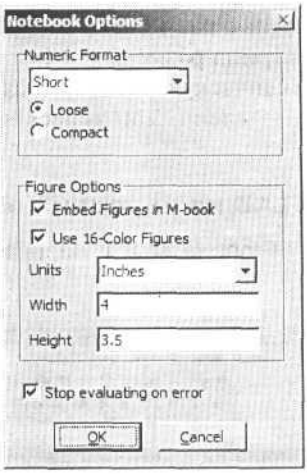

Рис. 2.16. Окно установки опций Notebook

Опция Use 16-Color Figures задает отображение рисунков в формате с 16 цветами, в противном случае отображается 256 цветов. Опция Units задает единицы измерения размеров графиков — в дюймах, сантиметрах или пикселях. Ширину и высоту рисунков задают соответственно параметры Width и Height.

Дополнительная опция Stop Evaluating on error останавливает оценку ячеек при возникновении ошибки. Если эта опция отключена, то эволюция продолжается даже при наличии ошибки с выдачей соответствующих предупреждающих сообщений.

Глава 3. Символьные вычисления

3.1. Назначение пакета Symbolic Math

В систему MATLAB 6.5 SP1 входит новая версия 3.0.1 пакета расширения Symbolic Math Toolbox (в дальнейшем сокращенно Symbolic), которая базируется на ядре символьной математической системы Maple 8, лидирующей в области автоматизации аналитических решений [44—46]. Объективности ради надо отметить, что используется не последняя реализация Maple — Maple 9. Сейчас уже есть версия Maple 9. Заметно обновленная версия Symbolic Math Toolbox 3.1, применяемая в системе MATLAB 7, имеет ряд новых функций и также работает с ядром Maple 8 [46]. Новые функции этой системы описаны в конце этой главы.

Пакет Symbolic Math Toolbox придает системе MATLAB новые качества системы символьных (аналитических) вычислений, которые по общности результатов и точности вычислений часто намного и принципиально превосходят численные вычисления. Символьные вычисления пока используются намного реже, чем численные. Но возможность их применения это именно то, чего системе MATLAB не хватало с момента ее появления.

С помощью команды

```
>> help symbolic
```

можно получить перечень входящих в пакет команд и функций. Он представлен ниже:

```
Symbolic Math Toolbox.
  Version 3.0.1 (R13SP1) 27-Dec-2002
  Calculus.
    diff        - Differentiate.
    int         - Integrate.
    limit       - Limit.
    taylor      - Taylor series.
    jacobian    - Jacobian matrix.
    symsum      - Summation of series.
  Linear Algebra.
    diag        - Create or extract diagonals.
    triu        - Upper triangle.
    tril        - Lower triangle.
    inv         - Matrix inverse.
    det         - Determinant.
    rank        - Rank.
    rref        - Reduced row echelon form.
    null        - Basis for null space.
    colspace    - Basis for column space.
    eig         - Eigenvalues and eigenvectors.
    svd         - Singular values and singular vectors.
    jordan      - Jordan canonical (normal) form.
    poly        - Characteristic polynomial.
    expm        - Matrix exponential.
```

```
Simplification.
  simplify   - Simplify.
  expand     - Expand.
  factor     - Factor.
  collect    - Collect.
  simple     - Search for shortest form.
  numden     - Numerator and denominator.
  horner     - Nested polynomial representation.
  subexpr    - Rewrite in terms of subexpressions.
  subs       - Symbolic substitution.
Solution of Equations.
  solve      - Symbolic solution of algebraic equations.
  dsolve     - Symbolic solution of differential
               equations.
  finverse   - Functional inverse.
  compose    - Functional composition.
Variable Precision Arithmetic.
  vpa        - Variable precision arithmetic.
  digits     - Set variable precision accuracy.
Integral Transforms.
  fourier    - Fourier transform.
  laplace    - Laplace transform.
  ztrans     - Z transform.
  ifourier   - Inverse Fourier transform.
  ilaplace   - Inverse Laplace transform.
  iztrans    - Inverse Z transform.
Conversions.
  double     - Convert symbolic matrix to double.
  poly2sym   - Coefficient vector to symbolic
               polynomial.
  sym2poly   - Symbolic polynomial to coefficient
               vector.
  char       - Convert sym object to string.
Basic Operations.
  sym        - Create symbolic object.
  syms       - Short-cut for constructing symbolic
               objects.
  findsym    - Determine symbolic variables.
  pretty     - Pretty print a symbolic expression.
  latex      - LaTeX representation of a symbolic
               expression.
  ccode      - C code representation of a symbolic
               expression.
  fortran    - FORTRAN representation of a symbolic
               expression.
Special Functions.
  sinint     - Sine integral.
  cosint     - Cosine integral.
  zeta       - Riemann zeta function.
```

```
    lambertw   - Lambert W function.
    gcd        - Greatest common divisor.
    lcm        - Least common multiple.
  String handling utilities.
    isvarname  - Check for a valid variable name (MATLAB
                 Toolbox).
    vectorize  - Vectorize a symbolic expression.
  Pedagogical and Graphical Applications.
    rsums      - Riemann sums.
    ezcontour  - Easy to use contour plotter.
    ezcontourf - Easy to use filled contour plotter.
    ezmesh     - Easy to use mesh (surface) plotter.
    ezmeshc    - Easy to use combined mesh/contour
                 plotter.
    ezplot     - Easy to use function, implicit, and
                 parametric curve plotter.
    ezplot3    - Easy to use spatial curve plotter.
    ezpolar    - Easy to use polar coordinates plotter.
    ezsurf     - Easy to use surface plotter.
    ezsurfc    - Easy to use combined surface/contour
                 plotter.
    funtool    - Function calculator.
    taylortool - Taylor series calculator.
  Demonstrations.
    symintro     - Introduction to the Symbolic Toolbox.
    symcalcdemo  - Calculus demonstration.
    symlindemo   - Demonstrate symbolic linear algebra.
    symvpademo   - Demonstrate variable precision arithmetic
    symrotdemo   - Study plane rotations.
    symeqndemo   - Demonstrate symbolic equation solving.
  Access to Maple. (Not available with Student Version.)
    maple        - Access Maple kernel.
    mfun         - Numeric evaluation of Maple functions.
    mfunlist     - List of functions for MFUN.
    mhelp        - Maple help.
    procread     - Install a Maple procedure. (Requires
                   Extended Toolbox.)
```

Он соответствует названиям заголовков этой главы. Для получения справки по любой команде или функции можно использовать команду

```
>> help sym/name.m
```

где `name` — это имя соответствующей команды или функции, а `name.m` — имя m-файла, задающего данную команду или функцию.

Ниже представлен пример на получение справки по функции интегрирования int:

```
>> help sym/int
 INT Integrate.
    INT(S) is the indefinite integral of S with respect to its symbolic
      variable as defined by FINDSYM. S is a SYM (matrix or scalar).
      If S is a constant, the integral is with respect to 'x'.
```

```
INT(S,v) is the indefinite integral of S with respect to v. v is a
   scalar SYM.
INT(S,a,b) is the definite integral of S with respect to its
   symbolic variable from a to b. a and b are each double or
   symbolic scalars.
INT(S,v,a,b) is the definite integral of S with respect to v
   from a to b.
Examples:
   syms x x1 alpha u t;
   A = [cos(x*t),sin(x*t);-sin(x*t),cos(x*t)];
   int(1/(1+x^2))          returns    atan(x)
   int(sin(alpha*u),alpha) returns    -cos(alpha*u)/u
   int(besselj(1,x),x)     returns    -besselj(0,x)
   int(x1*log(1+x1),0,1)   returns    1/4
   int(4*x*t,x,2,sin(t))   returns    2*sin(t)^2*t-8*t
   int([exp(t),exp(alpha*t)]) returns [exp(t), 1/alpha*exp(alpha*t)]
   int(A,t)                returns     [sin(x*t)/x, -cos(x*t)/x]
                                       [cos(x*t)/x, sin(x*t)/x]
```

Как нетрудно заметить, справка довольно детальная. Особенно приятно наличие ряда примеров применения функции интегрирования. Подобный подход использован и в справках по другим функциям.

Пакет Symbolic добавил системе MATLAB качественно новое свойство — возможность выполнения *символьных вычислений* и преобразований, которые ранее были доступны только в системах принципиально иного класса, относящихся к *компьютерной алгебре*. До введения этого пакета система MATLAB считалась наиболее мощной при решении математических задач и математического моделирования в численном виде [24—41]. Теперь она, с учетом новых средств, становится в полной мере *универсальной*.

Последняя реализация системы символьной математики Maple 8 в своем ядре и в расширениях имеет свыше 3000 функций [46]. Система MATLAB с пакетом Symbolic, включающим в себя чуть больше сотни символьных команд и функций, намного уступает Maple по количеству таких команд и функций. Однако в данный пакет включены лишь наиболее важные и широко распространенные функции. Кроме того, есть специальная команда, которая дает доступ к ядру Maple, что заметно расширяет круг используемых функций.

Помимо типовых *аналитических вычислений* (таких как символьное дифференцирование и интегрирование, упрощение математических выражений, подстановка и т. д.) пакет Symbolic позволяет реализовать *арифметические операции с произвольной точностью*.

О возможностях MATLAB в области символьной математики наглядно свидетельствуют заголовки разделов данной главы. Учитывая наличие ряда книг по системе Maple [43—45], мы рассмотрим символьные возможности MATLAB конспективно, но достаточно полно.

3.2. Демонстрационные примеры

3.2.1. Наборы примеров, вызываемые из командной строки

Для ознакомления с пакетом Symbolic можно использовать следующие команды:

- `symintro` — начальное знакомство с Symbolic;

- `symcalcdemo` — демонстрация символьных вычислений;
- `symlindemo` — демонстрация применения пакета Symbolic в задачах линейной алгебры;
- `symvpademo` — демонстрация операций арифметики с произвольной точностью;
- `symrotdemo` — изучение вопросов вращения плоскости;
- `symeqndemo` — демонстрация решения уравнений в символьном виде.

С каждым из этих примеров связан m-файл, запуск которого осуществляется с помощью одной из указанных команд. Текст пояснений в этих примерах приводится на английском языке. Для знакомства используется просмотр в окне командного режима, а возможности графики практически не используются в силу специфики символьных вычислений.

3.2.2. Примеры из вкладки Demos справки

С возможностями пакета Symbolic можно познакомиться также по примерам, которые имеются на вкладке Demos справки системы MATLAB. Доступ к этим примерам показан на рис. 3.1.

Из вкладки Demos доступны четыре группы примеров:

- Introduction — примеры вводного курса по применению пакета;
- Command line demos — примеры работы из командной строки;
- Calculus — примеры на вычисления;
- Function Calculator — применение калькулятора.

При выделенной гиперссылке Introduction в правой половине окна справки формируется описание этого раздела демонстрационных примеров. Активизация гиперссылки Run this demo выводит окно демонстрационных примеров, показан-

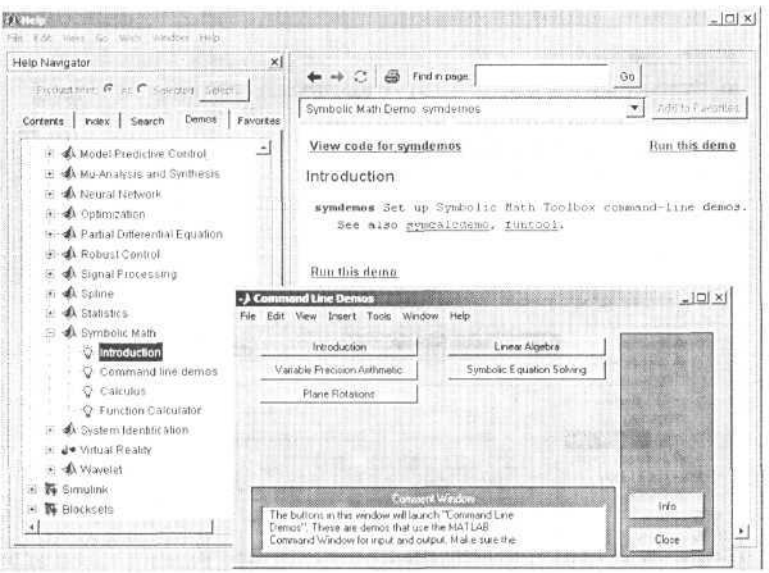

Рис. 3.1. Доступ к демонстрационным примерам со вкладки Demos справки MATLAB

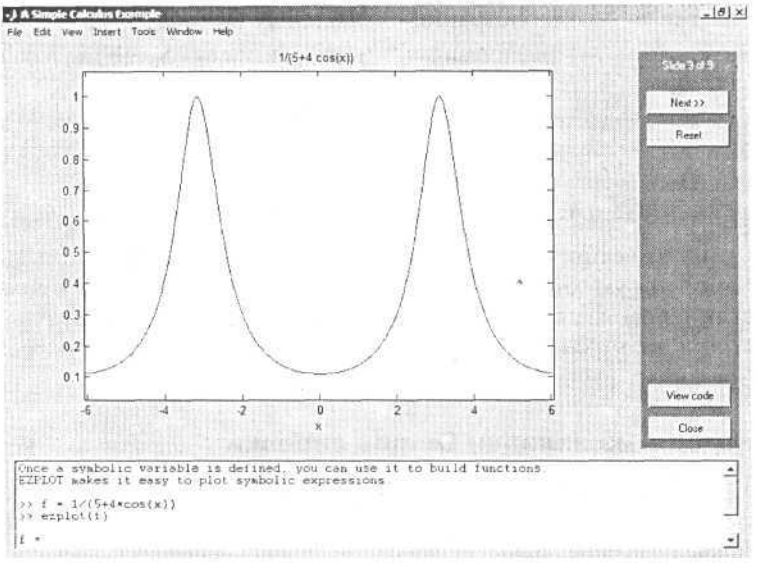

Рис. 3.2. Пример на построение графика функции

ное на рис. 3.1 в левом нижнем углу. Нажимая мышью его кнопки можно просмотреть демонстрационные примеры по простым вычислениям, линейной алгебре, точной арифметике, решению в символьном виде уравнений и по вращению плоскости. На рис. 3.2 показан один из примеров на построение графика функции.

Примеры выполняются в окне MATLAB с применением для продолжения просмотра примеров клавиши пробела. Аналогично выполняются примеры двух последующих разделов. Раздел Function Calculator выводит окна калькулятора, которые позволяют вычислять различные функции и строить их графики (рис. 3.3).

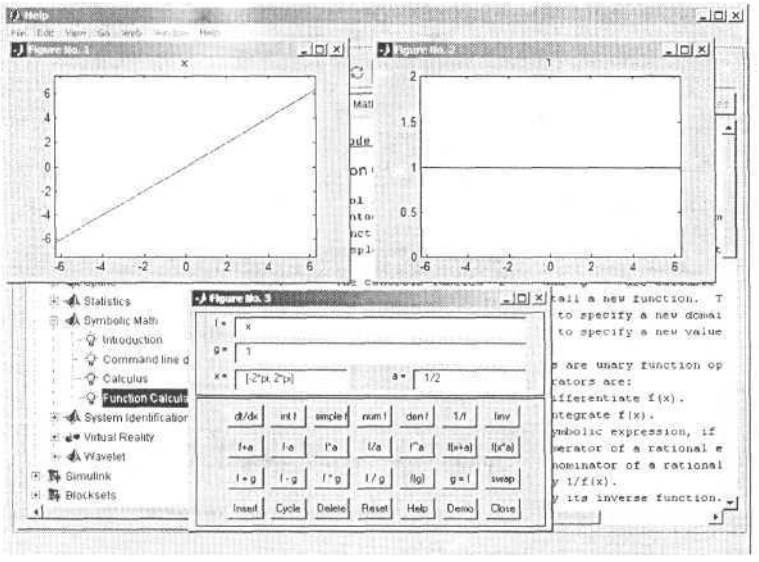

Рис. 3.3. Доступ к демонстрационным примерам на применение калькулятора

К сожалению, приходится отметить, что применение демонстрационных примеров для изучения пакета Symbolic не очень наглядно. Это обусловлено тем, что в окне MATLAB математические выражения представлены в MATLAB-формате строчной записи, который сильно отличается от обычных выражений. В связи с этим гораздо удобнее знакомиться с этим пакетом по приведенному ниже описанию.

3.3. Работа с числами, объектами и переменными

3.3.1. Задание символьных переменных

Поскольку переменные системы MATLAB по умолчанию не определены и традиционно задаются как векторные, матричные, числовые и т. д., то есть не имеющие отношения к символьной математике, для реализации символьных вычислений нужно прежде всего позаботиться о создании специальных *символьных переменных*. В простейшем случае их можно определить как строковые переменные, заключив имена в апострофы. Следующие примеры иллюстрируют действие этого приема на примере вычисления выражения $\sin(x)^2 + \cos(x)^2$:

```
>> sin(x)^2+cos(x)^2
??? Undefined function or variable 'x'.
>> sin('x')^2+cos('x')^2
ans =
     1
```

Итак, в первом случае система MATLAB «возмутилась» нашей небрежностью и сообщила, что функция или переменная `x` не определена и ни о каких вычислениях синуса и косинуса речи быть не может. Вместе с тем она подсказала, как надо сделать это определение — заключить имя переменной в апострофы, ибо таким образом система получает информацию о необходимости включить символьный режим вычислений. Поэтому во второй раз получен вполне осмысленный результат — сумма квадратов синуса и косинуса переменной `'x'` выдана равной 1. Это соответствует известному математическому равенству $\sin(x)^2 + \cos(x)^2 = 1$ для любого x.

Этот результат сразу демонстрирует характерную особенность символьных вычислений: они возвращают результаты даже в том случае, когда переменные не определены. Правила выполняемых при этом преобразований заложены в ядре символьных операций, в нашем случае это ядро системы Maple 8. Подробное описание ее можно найти в [46].

Символьные переменные имеют много общего со строковыми, которые также задаются с использованием апострофов. Однако различия между ними все же имеются, и самое серьезное из них — это возможность *эволюции* символьных выражений, то есть их вычисления и преобразования с помощью специальных символьных функций. По этой причине мы отделяем символьные переменные, хранящие символьные выражения, от строковых переменных, хранящих не эволюционирующий текст.

3.3.2. Функция создания символьных переменных sym

Для создания символьных переменных или объектов используется также функция `sym`:

- `S = sym(A)` — возвращает символьный объект `S` класса `'sym'` для входного параметра `A`. Если `A` — строка, то будет получена символьная строка или символьная переменная, а если `A` — это число (скаляр) или матрица, то будут получены их символьные представления.
- `x = sym('x')` — возвращает символьную переменную с именем `'x'` и записывает результат в `x`.
- `x = sym('x','real')` — возвращает символьную переменную вещественного типа, так что `conj(x)` эквивалентно `x`.

Возможны также следующие очевидные формы применения этой функции:

```
alpha = sym('alpha')
r = sym('Rho','real')
x = sym('x','unreal')
pi = sym('pi')
delta = sym('1/10')
```

Для преобразования чисел или матриц в символьную форму применяется функция `sym` в виде

```
S = sym(A,flag)
```

Второй аргумент — `flag` — задается в следующем виде:

- `'f'` — число с плавающей точкой `'1.F'*2^(e)` или `'-1.F'*2^(e)`, где `F` является строкой из 13 шестнадцатеричных чисел, а `e` — целым числом;
- `'r'` — число в рациональной форме (задано по умолчанию);
- `'e'` — число в рациональной форме плюс оценка машинной погрешности;
- `'d'` — число в расширенной десятичной форме с числом верных цифр, заданных функцией `digits`.

Примеры применения функции `sym` представлены ниже:

```
>> sym(1/50,'f')
ans =
'1.47ae147ae147b'*2^(-6)
>> sym(4/6,'r')
an+s =
2/3
>> sym(4/6,'e')
ans =
2/3-eps/6
>> sym(4/6,'d')
ans =
.66666666666666662965923251249478
>> digits(12)
>> sym(4/6,'d')
ans =
.666666666667
>> S=sym([1 2;3 4])
S =
[1, 2]
[3, 4]
>> 2*S
```

```
ans =
[2, 4]
[6, 8]
```

Обратите внимание на то, что результат символьных преобразований отображается *без отступа*, которым сопровождается выдача иных результатов. Это позволяет сразу опознавать его как символьный, в отличие от обычных численных результатов.

Ниже представлена еще тройка примеров, демонстрирующих различия вывода обычного и символьного:

```
>> sqrt(3)
ans =
    1.7321
>> sqrt(sym(3))
ans =
3^(1/2)
>> exp(1)
ans =
    2.7183
>> exp(sym(1))
ans =
exp(1)
>> pi/2
ans =
    1.5708
>> sym(pi/2)
ans =
pi/2
```

В отличие от обычного вывода в формате чисел с плавающей точкой символьный вывод старается представить числа в формате рациональных чисел и не вычисляет фундаментальные константы, например e = exp(1), а старается представить их в точном аналитическом виде.

3.3.3. Функция создания группы символьных объектов syms

Для создания группы символьных объектов служит функция `syms`:

- `syms arg1 arg2 ...` — создает группу символьных объектов, подобную выражениям arg1 = sym('arg1'); arg2 = sym('arg2'); ...;
- `syms arg1 arg2 ... real` и `syms arg1 arg2 ... unreal` — создают группы символьных объектов с вещественными (`real`) и невещественными (`unreal`) значениями. Последнюю функцию можно использовать для отмены задания вещественности объектов.

Приведем пример на применение функции `syms`:

```
>> syms x y1 y2
>> y1=sin(x)^2;
>> y2=cos(x)^2;
  >> y1+y2
  ans =
```

```
sin(x)^2+cos(x)^2
>> simplify(y1+y2)
ans =
1
```

Имена параметров (аргументов) должны начинаться с буквы и содержать только буквы и цифры. Применение в них спецзнаков, таких как +, –, *, ^ и т. п., недопустимо, поскольку такие знаки воспринимаются как операторы — сигналы к действию.

3.3.4. Функция создания списка символьных переменных findsum

В математических выражениях могут использоваться как обычные, так и символьные переменные. Функция `findsym` позволяет выделить символьные переменные в составе выражения `S`:

- `findsym(S)` — возвращает в алфавитном порядке список всех символьных переменных выражения `S`. При отсутствии таковых возвращается пустая строка.
- `findsym(S,N)` — возвращает список `N` символьных переменных, ближайших к `'x'` в порядке упорядочения по алфавиту.

Примеры:

```
>> a=2;b=4;
>> findsym(a*x^2+b*y+z)
ans =
x, y, z
>> findsym(x+y*i)
ans =
x, y
>> findsym(a+b+x+y+z,2)
ans =
x,y
>> findsym(a+b+x+y+z,4)
ans =
x, y, z
```

Функция `findsym` позволяет упростить запись многих функций, поскольку она автоматически находит используемые в них символьные переменные.

3.3.5. Работа с обычными числами

Пакет Symbolic может работать с числами различных, применяемых в MATLAB, форматов. При этом для представления результатов вычислений с вещественными числами в характерном для символьных вычислений виде используется функция sym — см. примеры ниже:

```
>> 1/2+3/4
ans =
    1.2500
>> sym(1/2+3/4)
ans =
```

```
5/4

>> pi/2
ans =
    1.5708

>> sym(pi/2)
ans =
pi/2

>> sin(pi/3)
ans =
    0.8660

>> sym(sin(pi/3))
ans =
sqrt(3/4)

>> exp(1)
ans =
    2.7183

>> sym(exp(1))
ans =
6121026514868074*2^(-51)
```

Тогда как MATLAB отдает предпочтения вычислениям с представлением чисел в формате с плавающей точкой (по умолчанию типа doubly), пакет Symbolic стремится представить числа в наиболее точном виде, предпочитая при этом целые и рациональные числа. Это видно и на примере вывода матрицы Гильберта:

```
>> MH=hilb(4)
MH =
    1.0000    0.5000    0.3333    0.2500
    0.5000    0.3333    0.2500    0.2000
    0.3333    0.2500    0.2000    0.1667
    0.2500    0.2000    0.1667    0.1429
>> sym(hilb(4))
ans =
[   1, 1/2, 1/3, 1/4]
[ 1/2, 1/3, 1/4, 1/5]
[ 1/3, 1/4, 1/5, 1/6]
[ 1/4, 1/5, 1/6, 1/7]
```

3.3.6. Работа с комплексными числами

Для работы с комплексными числами существует ряд возможностей. Прежде всего это применение опций 'real', 'imag' и 'unreal'. Например:

```
>> x=sym('x','real'); y=sym('y','real');
>> z=x+i*y
z =
x+i*y
>> real(z)
ans =
```

```
x
>> imag(z)
ans =
y
```

Нетрудно заметить, что функции real(z) и imag(z) обеспечивают выделение действительной и мнимой частей комплексного числа z. Конструкция

```
syms x unreal или x=sym('x','unreal')
```

делает переменную x не вещественной.

Для получения комплексно-сопряженного с z = x + i*y числа используется функция conj(z):

```
>> conj(z)
ans =
x-i*y
```

С комплексными числами могут работать также некоторые функции символьных преобразований, которые будут описаны в этой главе позже.

3.3.7. Математические выражения и функции

В части применения математических выражений пакет Symbolic Math не вносит ничего нового. За исключением того, что в выражениях должны использоваться символьные переменные. Выражения строятся на основе переменных, констант, арифметических операторов и функций — как встроенных, так и задаваемых пользователем.

В выражениях могут использоваться широко распространенные арифметические операторы +, −, *, / и ^. В общем случае это векторные и матричные операторы. Если необходимы операторы поэлементные, то надо использовать операторы .+, .−, .*, ./ и .^. Пример:

```
>> sin([1 2 3])
ans =
    0.8415    0.9093    0.1411
>> sym x
ans =
x
>> 2.*sin([1 2 3])
ans =
    1.6829    1.8186    0.2822
```

В выражениях доступны встроенные функции системы MATLAB. Так, в приведенном примере использована функция синуса. Возможно также задание абстрактных функций:

```
>> f=sym('f(x)')
f =
f(x)
```

Можно также задать различные операции с такими функциями, например операцию дифференцирования:

```
>> df=(subs(f,'x','x+h')-f)/'h'
```

```
df =
(f(x+h)-f(x))/h
>> syms x h
>> df=(subs(f,x,x+h)-f)/h
df =
(f(x+h)-f(x))/h
```

3.3.8. Матрицы с символьными элементами

После задания символьных переменных их можно использовать в качестве элементов матриц:

```
>> syms a b c d
>> M=[a b;c d]
  M =
  [ a, b]
  [ c, d]
```

С такими матрицами можно выполнять различные операции, например инвертировать матрицу или вычислять ее определитель (детерминант):

```
>> inv(M)
ans =
[ d/(a*d-b*c), -b/(a*d-b*c)]
[ -c/(a*d-b*c), a/(a*d-b*c)]
>> det(M)
ans =
a*d-b*c
```

Как нетрудно заметить, результат получается в аналитическом (символьном) виде.

3.4. Вывод и преобразования символьных выражений

3.4.1. Вывод символьного выражения

К сожалению, в отличие от современных систем символьной математики (Mathcad, Maple или Mathematica), MATLAB пока не способна выводить выражения и результаты их преобразований в естественной математической форме — с использованием общепринятых спецзнаков для отображения интегралов, сумм, произведений и т. д. Этот недостаток, однако, часто оборачивается достоинством — запись осуществляется в простом текстовом формате, облегчающем ввод выражений. Да и вывод символьных выражений в таком формате поддерживается практически всеми устройствами вывода.

Тем не менее некоторые ограниченные текстовым форматом возможности близкого к математическому виду вывода обеспечивает функция `pretty`:

- `pretty(S)` — дает вывод выражения `S` в формате, приближенном к математическому;
- `pretty(S,n)` — аналогична предшествующей функции, но задает вывод в `n` позициях строки (по умолчанию n = 79).

Примеры:

```
>> x=sym('x');
>> pretty(x^2)
                                    2
                                   x
>> pretty(int(x^2,x))
                                      3
                                   1/3 x
>> pretty(x^3,10)
     3
     x
>> syms x y z
>> pretty(int(sin(x),x))
                                    -cos(x)
>> S=(1+x^2)/(y^2-z^2);
>> pretty(y)
                                       y
>> pretty(S)
                                        2
                                   1 + x
                                   -------
                                    2    2
                                   y  - z
```

Нетрудно заметить, что в выведенных выражениях используются надстрочные показатели степени и (в необходимых случаях) знаки деления в виде горизонтальной черты. Однако для простых констант (например, 1/3) по-прежнему используется знак деления в виде наклонной черты. В целом такой вывод является лишь пародией на запись математических выражений в обычной математической форме.

3.4.2. Представления выражений в форме LaTeX

При подготовке материалов математической литературы давно используется специальный язык LaTeX, позволяющий задавать математические выражения с высокой точностью разметки. Функция `latex(S)` возвращает выражение `S` в форме языка LaTeX. Примеры применения этой функции:

```
>> syms x y z
>> latex((1+x^2)/(y^2-z^2))
ans =
{\frac {1+{x}^{2}}{{y}^{2}-{z}^{2}}}
>> S=x^2*y^3/z^4;
>> latex(S)
ans =
{\frac {{x}^{2}{y}^{3}}{{z}^{4}}}
```

Следует отметить, что LaTeX — это, в сущности, язык программирования разметки математических выражений и текстов, а его применение с использованием функции `latex(S)` не требует от пользователя знакомства с этим языком.

3.4.3. Представление выражений в кодах языка C

Система MATLAB написана на языке C, одном из лучших языков системного программирования. Существует возможность использования написанных на языке C библиотек расширения системы. А с помощью функции `ccode (S)` можно представить выражения языка MATLAB S в форме, принятой в языке C. Проиллюстрируем это примерами:

```
>> syms x y z
>> ccode((1+x^2)/(y^2-z^2))
ans =
      t0 = (1.0+x*x)/(y*y-z*z);
>> S=x^2*y^3/z^4;
>> ccode(S)
ans =
      t0 = x*x*y*y*y/pow(z,4.0);
```

Применение функции `ccdode` облегчает программистам создание библиотек программ на языке C и уменьшает вероятность ошибок в математических выражениях.

3.4.4. Представление выражений в кодах языка Fortran

Язык Fortran и сегодня не потерял своей привлекательности у создателей математических программ — в основном из-за своей почетной родословной и наличия множества библиотечных программ по реализации математических алгоритмов и численных методов. Функция `fortran(S)` обеспечивает преобразование выражения MATLAB `S` в форму, соответствующую записи на языке Fortran. Примеры:

```
>> syms x y z
>> fortran((1+x^2)/(y^2-z^2))
ans =
      t0 = (1+x**2)/(y**2-z**2)
>> S=x^2*y^3/z^4;
>> fortran(S)
ans =
      t0 = x**2*y**3/z**4
```

Функция `fortran` призвана облегчить работу специалистов, работающих с программами на языке Fortran. Большинство пользователей вряд ли оценят полезность этой функции, поскольку они обращаются к системе MATLAB именно с целью избавления от необходимости программирования на языках высокого уровня.

3.5. Контроль допустимости имен

Функция `isvarname(S)` возвращает логическое значение 1, если имя `S` допустимо в системе MATLAB, и 0, если оно недопустимо. Допустимое имя должно начинаться с буквы и может иметь до 32 символов (букв и цифр). Примеры:

```
>> isvarname('xxx')
ans =
     1
```

```
>> isvarname('1x')
ans =
     0
>> isvarname('a+b')
ans =
     0
```

3.6. Векторизация символьных выражений

Векторизация означает проведение почленного преобразования элементов матриц и векторов. В MATLAB функция `vectorize(S)` для символьного выражения `S` вставляет знак «.» после всех следующих символов: у «^», «*» или «/». Результатом будет строка, содержащая операторы для почленного вычисления выражения:

```
>> syms x
>> vectorize(1+2*x+3*x^2)
ans =
1+2.*x+3.*x.^2
```

3.7. Арифметика произвольной точности

3.7.1. Установка количества знаков чисел

Арифметикой произвольной точности называют вычисления, у которых все числа результатов являются точными. Функция `digits` служит для установки числа цифр в числах арифметики произвольной точности. Она используется в следующем виде:

- `digits` — возвращает число значащих цифр в числах арифметики произвольной точности (по умолчанию 32);
- `digits(D)` — устанавливает заданное число цифр `D` для чисел арифметики произвольной точности (`D` — целое число, строка или переменная типа `sym`).

Примеры:

```
>> digits
Digits = 32
>> vpa pi
ans =
3.1415926535897932384626433832795
>> digits(6)
>> pi
ans =
    3.1416
```

3.7.2. Вычисления в арифметике произвольной точности

MATLAB обычно ведет вычисления с числами, представленными в формате плавающей точки с двойной точностью. Это довольно высокая точность, обеспе-

чивающая потребности практических вычислений в прикладных задачах. Однако ряд задач теории чисел, численного кодирования и некоторых других требует выполнения вычислений вообще без какой-либо погрешности или со сколь угодно малой погрешностью. Такие вычисления не очень удачно называют *арифметикой произвольной точности* — правильнее говорить просто о *точной* арифметике.

Для проведения вычислений в арифметике произвольной точности служит функция vpa:

- R=vpa(S) — возвращает результат вычислений каждого элемента символьного массива S, используя арифметику произвольной точности с текущим числом цифр D, установленным функцией digits. Результат R имеет тип sym;
- vpa(S,D) — возвращает результат вычислений каждого элемента массива S, используя арифметику произвольной точности с количеством знаков чисел D.

Примеры:

```
>> vpa(exp(1),50)
 ans =
 2.7182818284590450907955982984276488423347473144531
 >> vpa([2*pi exp(1) log(2)],10)
 ans =
 [ 6.283185308, 2.718281828, .6931471806]
```

3.7.3. Обращения к функциям Maple на примере вычисления факториала

Иногда при точной арифметике возникают трудности в вычислении некоторых функций. Например, в комбинаторике большую роль играет точное вычисление факториала. Факториал в MATLAB вычисляется с применением фунции произведения:

```
>> prod(1:100)
ans =
  9.3326e+157
```

Такое вычисление выглядит очень грубым, ибо результат представлен в форме с плавающей точкой и с мантиссой, содержащей всего 5 цифр. Не очень помогает подобное вычисление с помощью функций sym и prod:

```
>> sym(prod(1:100))
 ans =
 7653246495856088*2^(472)
```

Самый разумный путь вычисления факториала со всеми точными цифрами заключается в использовании для этого средств вычисления факториала системы символьной математики Maple:

```
>> nfact=sym('n!')
 nfact =
n!
 >> sym n
 ans =
 n
```

```
>> subs(nfact,10)
ans =
3628800
>> subs(nfact,100)
ans =
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000000
```

Здесь использована функция подстановки subs для подстановки аргумента в виде числа в параметр функции nfact. При вычислении 100! вычислены все точные цифры весьма громоздкого результата. Позже мы вернемся к более подробному описанию функции подстановки.

3.8. Символьные операции с матрицами

3.8.1. Задание или извлечение диагональных элементов матриц

Линейная алгебра — конек системы MATLAB. В этом разделе описаны функции линейной алгебры, обеспечивающие символьные преобразования и вычисления векторов и матриц, существенно расширяющие типовые численные средства линейной алгебры системы MATLAB.

Для создания диагональных матриц и извлечения из них диагональных элементов служит функция `diag`:

- `diag(V,K)` — если `V` — вектор с `N` компонентами, то формируется квадратная матрица с размером `N+ABS(K)`, в которой на `K`-й диагонали размещен вектор `V`. При `K=0` вектор `V` располагается на главной диагонали, при `K>0` на `K`-й диагонали сверху, а при `K<0` — снизу относительно главной диагонали;
- `diag(V)` — формирует диагональную матрицу с вектором `V` на главной диагонали;
- `v=diag(X,K)` — извлекает `K`-ю диагональ из квадратной матрицы `X`;
- `v=diag(X)` — извлекает главную диагональ из матрицы `X`.

Помимо заданных, остальные элементы матриц — нули. Примеры применения функции `diag` приводятся ниже:

```
>> syms a b c d;
>> diag([a b c d])
ans =
[ a, 0, 0, 0]
[ 0, b, 0, 0]
[ 0, 0, c, 0]
[ 0, 0, 0, d]
>> diag([a b],-2)
ans =
[ 0, 0, 0, 0]
[ 0, 0, 0, 0]
[ a, 0, 0,0 ]
[ 0, b, 0, 0]
```

```
>> M=diag([a b c],1)
M =
[ 0, a, 0, 0]
[ 0, 0, b, 0]
[ 0, 0, 0, c]
[ 0, 0, 0, 0]
>> V=diag(M,1)
V =
[ a]
[ b]
[ c]
```

3.8.2. Формирование верхней треугольной матрицы

Для формирования верхней треугольной матрицы используются функции:

- `triu(X)` — оставляет в матрице `X` верхнюю треугольную часть;
- `triu(X ,K)` — оставляет в матрице `X` верхнюю треугольную часть от `K`-й диагонали (правила задания `K` были описаны выше).

Остальные элементы матрицы `X` заменяются нулями. Примеры применения функции `triu` представлены ниже:

```
>> syms a b c;
>> X=[a b c; b c a; c b a];
>> X=[a b c; b c a; c b a]
X =
[ a, b, c]
[ b, c, a]
[ c, b, a]
>> triu(X)
ans =
[ a, b, c]
[ 0, c, a]
[ 0, 0, a]
>> triu(M,1)
ans =
[ 0, a, 0, 0]
[ 0, 0, b, 0]
[ 0, 0, 0, c]
[ 0, 0, 0, 0]
```

3.8.3. Формирование нижней треугольной матрицы

Для формирования нижней треугольной матрицы используются функции:

- `tril(X)` — оставляет в матрице `X` нижнюю треугольную часть;
- `tril(X ,K)` — оставляет в матрице `X` нижнюю треугольную часть от `K`-й диагонали (правила задания `K` были описаны выше).

Остальные элементы матрицы `X` заменяются нулями. Примеры применения функции `tril`:

```
>> syms a b c;
```

```
>> X=[a b c; b c a; c b a]
X =
[ a, b, c]
[ b, c, a]
[ c, b, a]
ans =
[ a, 0, 0]
[ b, c, 0]
[ c, b, a]
>> tril(X,-1)
ans =
[ 0, 0, 0]
[ b, 0, 0]
[ c, b, 0]
```

3.8.4. Обращение матрицы

Напоминаем, что обращением квадратной матрицы **X** называют результат деления единичной матрицы `E` того же размера, что и исходная матрица, на эту матрицу, то есть $\mathbf{X}^{-1} = \mathbf{E}/\mathbf{X}$. Для обращения (инвертирования) матрицы в символьном виде используется функция `inv`:

```
>> syms a b c d;
>> inv([a b;c d])
ans =
[ d/(a*d-b*c), -b/(a*d-b*c)]
[ -c/(a*d-b*c), a/(a*d-b*c)]
```

3.8.5. Вычисление детерминанта и ранга матрицы

Функция `det(X)` вычисляет детерминант квадратной матрицы в символьном виде. Пример:

```
>> syms a b c d;
>> det([a b; c d])
ans =
a*d-b*c
```

Для вычисления ранга квадратной матрицы используется функция `rank`:

- `rank(A,tol)` — число сингулярных значений матрицы `A`, *определенных* с погрешностью `tol`;
- `rank(A)` — аналогично предшествующей функции при `tol=max(size(A))*norm(A)*eps`.

Пример:

```
>> syms a b c d;
>> rank([a b;c d])
ans =
2
```

3.8.6. Приведение матриц к треугольным формам

Для приведения матрицы к верхней треугольной форме используется функция `rref`:

- `rref(A)` — осуществляет приведение матрицы к треугольной форме, используя метод исключения Гаусса с частичным выбором ведущего элемента. По умолчанию значение порога допустимости для незначительного элемента столбца принимается равным `(max(size(A))*eps *norm(A,inf))`.
- `[R,jb] = rref(A)` — дополнительно возвращает вектор `jb`, такой что
  - `r = length(jb)` может служить оценкой ранга матрицы `A`;
  - `x(jb)` — связанные переменные в системе линейных уравнений `Ax = b`;
  - `A(:,jb)` — базис матрицы `A`;
  - `R(1:r,jb)` — единичная матрица размера `r×r`.
- `[R,jb] = rref(A,tol)` — осуществляет приведение матрицы к треугольной форме, используя метод исключения Гаусса с частичным выбором ведущего элемента для заданного значения порога допустимости `tol`.

3.8.7. Нуль-пространство матрицы

Функция `Z = null(A)` возвращает матрицу `Z`, столбцы которой являются базисом нуль-пространства целочисленной матрицы `A`. Число столбцов матрицы `Z` задает размер нуль-пространства. При этом `A*Z=0`, а если матрица `A` имеет полный ранг, то матрица `Z` будет пустой. Пример:

```
>> syms a b c;
>> A=[a b c; a b c;a b c]
A =
[ a, b, c]
[ a, b, c]
[ a, b, c]
>> null(A)
ans =
[ -1/a*c, -1/a*b]
[      0,      1]
[      1,      0]
```

3.8.8. Базис-пространство столбцов — colspace

Функция `colspace(A)` возвращает матрицу, столбцы которой являются образующими базиса пространства. Ранг целочисленной матрицы `A` равен `size(B,2)`. Примеры:

```
>> syms a b c;
>> A=[a b c; a b c; a b c]
A =
[ a, b, c]
[ a, b, c]
[ a, b, c]
>> colspace(A)
ans =
[ 1]
[ 1]
```

```
[ 1]
>> colspace(sym(magic(3)))
ans =
[ 1, 0, 0]
[ 0, 1, 0]
[ 0, 0, 1]
```

3.8.9. Вычисление собственных значений и векторов матриц

Для вычисления собственных значений и собственных векторов матриц используется функция eig, имеющая ряд форм записи:

- LAMBDA=eig(A) — формирует символьный вектор LAMBDA собственных значений квадратной матрицы A;
- [V,D]=eig(A) — возвращает матрицу V, столбцы которой являются векторами собственных значений матрицы A, и диагональную матрицу D собственных значений. Если размеры V и A одинаковы, то A имеет полную систему независимых собственных векторов. При этом A*V = V*D;
- [V,D,P]=eig(A) — дополнительно к сказанному возвращает вектор индексов P, длина которого равна числу линейно независимых векторов. При этом A*V = V*D(P,P);
- LAMBDA=eig(VPA(A)) и [V,D]= eig(VPA(A)) — возвращают численные значения собственных векторов и собственных значений в формате арифметики с произвольной точностью. Если матрица A не имеет полной системы собственных векторов, то столбцы матрицы V будут линейно зависимыми.

Примеры:

```
>> syms a b c d
>> A=[a b; c d]
A =
[ a, b]
[ c, d]
>> eig(A)
ans =
[ 1/2*a+1/2*d+1/2*(a^2-2*a*d+d^2+4*b*c)^(1/2)]
[ 1/2*a+1/2*d-1/2*(a^2-2*a*d+d^2+4*b*c)^(1/2)]
>> M=[1 2 3; 4 5 6; 9 8 7];
>> L=eig(M)
L =
   15.3459
   -2.3459
   -0.0000
>> [V,D]=eig(M)
V =
  -0.2437      0.4781      -0.4082
  -0.5553      0.3846      0.8165
  -0.7951      -0.7896     -0.4082
D =
  15.3459           0              0
        0          -2.3459         0
        0           0         -0.0000
```

3.8.10. Сингулярное разложение матриц — svd

Для сингулярного разложения матриц используется функция svd в ряде форм:

- SIGMA=svd(A) — возвращает вектор сингулярных значений символьной матрицы A;
- SIGMA=svd(VPA(A)) — возвращает численные сингулярные значения в формате арифметики произвольной точности;
- [U,S,V]=svd(A) и [U,S,V]=svd(VPA(A)) — возвращает унитарные матрицы U и V и диагональную матрицу S сингулярных значений, для которых A = U*S*V'. Эти вычисления возможны только в численной форме.

Примеры:

```
>> A=sym(magic(3))
A =
[ 8, 1, 6]
[ 3, 5, 7]
[ 4, 9, 2]
>> svd(A)
ans =
[        15]
[ 2*3^(1/2)]
[ 4*3^(1/2)]
>> digits(6)
>> [U,S,V]=svd(A)
U =
[-.577350,      -.707107,      -.408248]
[-.577350,  .152046e-15,       .816497]
[-.577350,       .707107,      -.408248]
S =
[15.0000,             0,             0]
[ 0,            6.92820,             0]
[ 0,                  0,       3.46410]
V =
[-.577350,      -.408248,      -.707107]
[-.577350,       .816497,      -.194726e-16]
[-.577350,      -.408248,       .70710 ]
```

3.8.11. Вычисление канонической формы Жордана

Функция jordan(A) возвращает каноническую форму Жордана для символьной или численной матрицы A. Матрица A должна задаваться точно (элементы должны быть целыми или рациональными числами), поскольку даже малая погрешность способна исказить структуру клеток Жордана. В форме [V,J]=jordan(A) вычисляются как каноническая форма Жордана J, так и матрица подобия V, так что V\A*V = J. Столбцы матрицы V являются обобщенными собственными векторами. Примеры:

```
>> A=sym(magic(3))
A =
[ 8, 1, 6]
```

```
[ 3, 5, 7]
[ 4, 9, 2]
>> J=jordan(A)
J =
[15,              0,           0 ]
[ 0,    -2*6^(1/2),           0 ]
[ 0,              0,   2*6^(1/2)]
>> [V,J]=jordan(A)
V =
[1/3,   -1/8*6^(1/2)+1/3,   1/8*6^(1/2)+1/3 ]
[1/3,   1/12*6^(1/2)-1/6,   -1/12*6^(1/2)-1/6 ]
[1/3,   1/24*6^(1/2)-1/6,   -1/24*6^(1/2)-1/6 ]
J =
[15,              0,              0 ]
[ 0,     -2*6^(1/2),              0 ]
[ 0,              0,     2*6^(1/2) ]
```

3.8.12. Вычисление характеристического полинома матриц — poly

Для вычисления характеристического полинома матриц используется функция `poly`:

- `poly(A)` — возвращает характеристический полином матрицы `A`, используя (по умолчанию) переменную `'x'` или `'t'`;
- `poly(A,v)` — действует аналогично, но позволяет задать переменную `'v'` полинома.

Пример:

```
>> syms a b c d;
>> A=[a b;c d];
>> poly(A,'p')
ans =
p^2-p*d-a*p+a*d-b*c
```

3.8.13. Вычисление матричного экспоненциала

Для вычисления матричного экспоненциала матрицы `A` используется функция `expm(A)`. Рассмотрим пример ее применения:

```
>> syms t;
>> A=[1 0; 0 -1]
A =
     1      0
     0     -1
>> expm(t*A)
ans =
[   exp(t),          0]
[        0,    exp(-t)]
```

3.9. Символьные операции математического анализа

3.9.1. Вычисления производных

Для вычисления в символьном виде производных от выражения S служит функция diff, записываемая в формате diff(S,'v') или diff(S,sym('v')). Она возвращает символьное значение первой (n=1) производной от символьного выражения или массива символьных выражений S по переменной v. Эта функция возвращает

$$S'(v) = \frac{dS}{dv}.$$

- diff(S,n) — возвращает n-ю (n — целое число) производную от символьного выражения или массива символьных выражений S по переменной v.
- diff(S,'v',n) и diff(S,n,'v') — возвращает n-ю производную S по переменной v, то есть значение

$$S^n(v) = \frac{d^n S}{dv^n}.$$

Примеры:

```
>> x=sym('x');y=sym('y');
>> diff(x^y)
ans =
x^y*y/x
>> diff(x^y,x)
ans =
x^y*y/x
>> simplify(ans)
ans =
x^(y-1)*y
>> diff(sin(y*x),x,3)
ans =
-cos(y*x)*y^3
>> diff([x^3 sin(x) exp(x)],x)
ans =
[ 3*x^2, cos(x), exp(x)]
```

3.9.2. Вычисления интегралов

В практической работе часто возникает необходимость вычисления неопределенных и определенных интегралов вида

$$I = \int f(x)dx \quad \text{и} \quad I = \int_a^b f(x)dx.$$

Здесь $f(x)$ — подынтегральная функция независимой переменной x, a — нижний и b — верхний пределы интегрирования для определенного интеграла.

- int(S) — возвращает символьное значение неопределенного интеграла от символьного выражения или массива символьных выражений S по пере-

менной, которая автоматически определяется функцией `findsym`. Если `S` — скаляр или матрица, то вычисляется интеграл по переменной `'x'`.

- `int(S,v)` — возвращает неопределенный интеграл от `S` по переменной `v`.
- `int(S,a,b)` — возвращает определенный интеграл от `S` с пределами интегрирования от `a` до `b`, причем пределы интегрирования могут быть как символьными, так и числовыми.
- `int(S,v,a,b)` — возвращает определенный интеграл от `S` по переменной `v` с пределами от `a` до `b`.

Примеры применения этой функции приводятся ниже:

```
>> x=sym('x');
>> int(x^2,x)
ans =
1/3*x^3
>> int(sin(x)^3,x)
ans =
-1/3*sin(x)^2*cos(x)-2/3*cos(x)
>> int(log(2*x),x)
ans =
log(2*x)*x-x
>> int((x^2-2)/(x^3-1),x,1,2)
ans =
-inf
>> int((x^2-2)/(x^3-1),x,2,5)
ans =
-2/3*log(2)+2/3*log(31)+2/3*3^(1/2)*atan(11/3*3^(1/2))-...
2/3*log(7)-2/3*3^(1/2)*atan(5/3*3^(1/2))
>> int([x^3 sin(x) exp(x)],x)
ans =
[ 1/4*x^4, -cos(x), exp(x)]
```

С помощью функции `int` можно вычислять имеющие аналитическое решение сложные интегралы, например с бесконечными пределами (или одним из пределов), а также кратные интегралы. Ниже представлен пример вычисления одного из таких интегралов:

$$\int_0^{\infty} xe^{-x}dx.$$

```
>> syms a x y z
>> int(x*exp(-x),x,0,inf)
 ans =
 1
```

Аппарат точной арифметики позволяет вычислять определенные интегралы, в вычислении которых существуют трудности из-за вычислений больших чисел. К примеру, интеграл

$$\int_0^1 x^{20}\exp(-x)dx$$

часто дает нулевое значение вместо 0,01835... MATLAB, правда, честно сообщает, что такой интеграл он вычислить не в состоянии из-за отсутствия соответствующего метода:

```
>> int(exp(1)^20*exp(-x),x,0,1)
??? No appropriate methods for function int.
```

Попробуем выяснить, в чем тут дело. Вычислим для этого неопределенный интеграл

$$\int x^{20}\exp(-x)dx,$$

используя функцию int, но в символьном виде:

```
>> syms x
>> int(x^20*exp(-x),x)
ans =
-x^20*exp(-x)-20*x^19*exp(-x)-380*x^18*exp(-x)-
6840*x^17*exp(-x)-116280*x^16*exp(-x)-1860480*x^15*exp(-x)-
27907200*x^14*exp(-x)-390700800*x^13*exp(-x)-
5079110400*x^12*exp(-x)-60949324800*x^11*exp(-x)-
670442572800*x^10*exp(-x)-6704425728000*x^9*exp(-x)-
60339831552000*x^8*exp(-x)-482718652416000*x^7*exp(-x)-
3379030566912000*x^6*exp(-x)-20274183401472000*x^5*exp(-x)-
101370917007360000*x^4*exp(-x)-405483668029440000*x^3*exp(-x)-
1216451004088320000*x^2*exp(-x)-2432902008176640000*x*exp(-x)-
2432902008176640000*exp(-x)
```

Нетрудно заметить, что интеграл представлен рядом экспонент с большими множителями перед ними. При обычной точности вычислений погрешности велики и приводят к нулевому результату из-за вычитания больших близких чисел в условиях ограничения разрядности. Но MATLAB с пакетом Symbolic Math благополучно выдает точное и приближенное значение нашего интеграла:

```
>> a=int(x^20*exp(-x),x,0,1)
a =
-661331331924808001*exp(-1)+2432902008176640000
>> double(a)
ans =
    0.0184
```

Следующий пример относится к вычислению тройного интеграла:

$$\int_0^a\int_0^a\int_0^a (x^2+y^2)zdxdydz = \frac{1}{3}a^6.$$

Здесь можно трижды использовать функцию int:

```
>> syms x y z a
>> int(int(int((x^2+y^2)*z,x,0,a),y,0,a),z,0,a)
 ans =
 1/3*a^6
```

Итак, один из приведенных примеров вычисляет интеграл с бесконечным верхним пределом, другой — довольно «каверзный» интеграл, а третий — тройной интеграл.

3.9.3. Вычисление пределов

Вычисление пределов функций представляет собой важный раздел математического анализа.

Число L называется пределом функции $F(x)$ в точке a, если при x, стремящемся к a (или $x \to a$), значение функции неограниченно приближается к L. Это обозначается следующим образом:

$$\lim_{x \to a} F(x) = L.$$

Предел может быть конечным числом, положительной или отрицательной бесконечностью.

Есть функции (например, разрывные в точке $x = a$), у которых нет предела в самой точке $x = a$, но есть предел при $x \to a - 0$ или при $x \to a + 0$, где под 0 подразумевается очень малое число. В первом случае говорят о существовании предела *слева* от точки $x = a$, а во втором — *справа* от этой точки. Если эти пределы равны, то существует предел функции в точке $x = a$.

Для вычисления пределов аналитически (в символьном виде) заданной функции $F(x)$ служит функция `limit`, которая записывается в следующих вариантах:

- `limit(F,x,a)` — возвращает предел символьного выражения `F` в точке `x → a`;
- `limit(F,a)` — возвращает предел для независимой переменной, определяемой функцией `findsym`;
- `limit(F)` — возвращает предел при `a=0`;
- `limit(F,x,a,'right')` или `limit(F,x,a,'left')` — возвращает предел в точке `a` справа или слева.

Ниже представлены примеры применения этих функций:

```
>> syms a x
>> limit(sin(a*x)/(a*x))
ans =
1
>> limit(sin(a*x)/x)
ans =
a
>> limit(2*sin(x)/x)
ans =
2
>> limit(2+sin(x)/x,0)
ans =
3
>> limit(tan(x),pi)
ans =
0
>> limit(tan(x), pi/2)
ans =
```

```
NaN
>> limit(tan(x),x,pi/2,'right')
ans =
-inf
>> limit(tan(x),x,pi/2,'left')
ans =
inf
>> limit([sin(x)/x, (1+x)/(2+x)],x,0)
ans =
[ 1, 1/2]
```

3.9.4. Разложение выражения в ряд Тейлора

В задачах аппроксимации и приближения функций $f(x)$ важное место занимает их разложение в ряд Тейлора в окрестности точки a:

$$f(x) = \sum_{n=0}^{\infty} (x-a)^n \frac{f^n(a)}{n!}.$$

Частным случаем этого ряда при $a = 0$ является ряд Маклорена:

$$f(x) = \sum_{n=0}^{\infty} x^n \frac{f^n(0)}{n!}.$$

Для получения разложений аналитических функций в ряд Тейлора (и Маклорена) служит функция `taylor`:

- `taylor(f)` — возвращает шесть членов ряда Маклорена (ряд Тейлора в точке $x = 0$). В любом разложении можно задавать число членов ряда `n`, точку `a`, относительно которой ищется разложение, и переменную `x`, по которой ищется разложение, например `taylor(f,n,x,a)`;
- `taylor(f,n)` — возвращает члены ряда Маклорена до `(n-1)`-го порядка;
- `taylor(f,a)` — возвращает ряд Тейлора в окрестности точки `a`;
- `taylor(f,x)` — возвращает ряд Тейлора для переменной `x`, определяемой функцией `findsym(f)`.

Примеры разложения функций в ряд:

```
>> x=sym('x');
>> F=sin(x);
>> taylor(F)
ans =
x-1/6*x^3+1/120*x^5
>> taylor(F,10)
ans =
x-1/6*x^3+1/120*x^5-1/5040*x^7+1/362880*x^9
>> taylor(exp(x),1)
ans =
1
>> taylor(cos(x),-pi/2,6)
ans =
x+1/2*pi-1/6*(x+1/2*pi)^3+1/120*(x+1/2*pi)^5
```

3.9.5. Вычисления матрицы Якоби

Матрица Якоби вычисляется для системы функций и в общем случае представляет собой прямоугольную матрицу. Число строк равно числу функций в системе, число столбцов — числу аргументов. Матрица Якоби записывается в виде

$$J = \begin{vmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \cdots & \frac{\partial F_2}{\partial x_{n1}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1} & \frac{\partial F_n}{\partial x_2} & \cdots & \frac{\partial F_n}{\partial x_{n1}} \end{vmatrix}.$$

Для ее вычисления используется функция `jacobian`:

- `jacobian(f,v)` — возвращает матрицу Якоби для скаляра или вектора `f` по вектору переменных `v`. Каждый (i, j)-й элемент матрицы представляет собой частную производную $\partial f_i/\partial v_j$.

Примеры:

```
>> syms x y z;
>> F=[x^2;x+y/z;x*z];
>> v=[x,y,z];
>> J=jacobian(F,v)
J =
[2*x,      0,         0]
[ 1,    1/z,    -y/z^2]
[ z,      0,         x]
>> v=[x,y];
>> jacobian(F,v)
ans =
[ 2*x,     0]
[   1,   1/z]
[   z,     0]
>> jacobian(x*y,v)
ans =
[ y, x]
```

3.9.6. Вычисление сумм рядов

В математическом анализе часто приходится вычислять суммы некоторой функции $f(i)$ для целочисленных значений аргумента i от a до b:

$$Sum = \sum_{i=a}^{b} f(i).$$

Такие суммы принято назвать *конечными*. При $b = \infty$ можно говорить о *бесконечной* сумме (в смысле бесконечности числа членов ряда).

Для аналитического вычисления суммы ряда служит команда `symsum`:

- `symsum(S)` — возвращает символьное значение суммы бесконечного ряда по переменной, найденной автоматически с помощью функции `findsym`;
- `symsum(S,v)` — возвращает сумму бесконечного ряда по переменной `v`;
- `symsum(S,a,b)` и `symsum(S,v,a,b)` — возвращает конечную сумму ряда в пределах номеров слагаемых от `a` до `b`.

Примеры на вычисление сумм даны ниже:

```
>> x=sym('x');
>> symsum(x^2)
ans =
1/3*x^3-1/2*x^2+1/6*x
>> symsum(x^2,6)
>> symsum(1/x^4)
ans =
-1/6*Psi(3,x)
>> symsum(1/x^4,1,5)
 ans =
 14001361/12960000
>> symsum([x,x^2,x^3],1,5)
 ans =
 [ 15, 55, 225]
```

3.9.7. Решение алгебраических уравнений

Для решения систем алгебраических уравнений и одиночных уравнений служит функция `solve`:

- `solve(expr1,expr2,...,exprN,var1,var2,...varN)` — возвращает значения переменных `varI`, при которых соблюдаются равенства, заданные выражениями `exprI`. Если в выражениях не используются знаки равенства, то полагается `exprI=0`;
- `solve(expr1,expr2,...,exprN)` — аналогична предшествующей функции, но переменные, по которым ищется решение, определяются функцией `findsym`.

При отсутствии аналитического решения и числе неизвестных, равном числу уравнений, ищется только одно *численное* решение, а не все решения. Результат решения возможен в следующих формах:

- для одного уравнения и одной переменной решение возвращается в виде одномерного или многомерного массива ячеек;
- при одинаковом числе уравнений и переменных решение возвращается в упорядоченном по именам переменных виде;
- для систем с одним выходным аргументом решение возвращается в виде массива записей.

Примеры:

```
>> syms x y;
>> solve(x^3-1,x)
ans =
[                  1]
[ -1/2+1/2*i*3^(1/2)]
```

```
[ -1/2-1/2*i*3^(1/2)]
>> solve(x^2-x-9,x)
ans =
[ 1/2+1/2*37^(1/2)]
[ 1/2-1/2*37^(1/2)]
>> syms a b c
>> solve(a*x^2+b*x+c)
ans =
[ 1/2/a*(-b+(b^2-4*a*c)^(1/2))]
[ 1/2/a*(-b-(b^2-4*a*c)^(1/2))]
>> S=solve('x+y=3','x*y^2=4',x,y)
S =
    x: [3x1 sym]
    y: [3x1 sym]
>> S.x
ans =
[ 4]
[ 1]
[ 1]
>> S.y
ans =
[ -1]
[ 2]
[ 2]
>> solve('sin(x)=0.5',x)
ans =
.52359877559829887307710723054658
```

Обратите внимание на то, что для задания уравнений в явном виде используются строковые выражения, когда уравнения заключаются в апострофы. Рекомендуется использовать графики функций или графики левой и правой части уравнений для получения графической интерпретации решений. Это особенно полезно в том случае, если решение носит множественный характер, поскольку функция `solve` дает только одно решение.

3.9.8. Решение дифференциальных уравнений

Для решения дифференциальных уравнений в форме Коши MATLAB имеет следующую функцию:

- `dsolve('eqn1','eqn2', ...)` — возвращает аналитическое решение системы дифференциальных уравнений с начальными условиями. Они задаются равенствами `eqnI` (вначале задаются уравнения, затем начальные условия).

По умолчанию независимой переменной считается переменная `'t'`, обычно обозначающая время. Можно использовать и другую переменную, добавив ее в конец списка параметров функции `dsolve`. Символ `D` обозначает производную по независимой переменной, то есть `d/dt`, при этом `D2` означает d^2/dt^2 и т. д. Имя независимой переменной не должно начинаться с буквы `D`.

Начальные условия задаются в виде равенств `'y(a)=b'` или `'Dy(a)=b'`, где `y` — независимая переменная, `a` и `b` — константы. Если число начальных условий меньше, чем число дифференциальных уравнений, то в решении будут присутст-

вовать произвольные постоянные C1, C2 и т. д. Правила вывода подобны приведенным для функции solve.

Примеры применения функции dsolve:

```
>> dsolve('D2x=-2*x')
ans =
C1*cos(2^(1/2)*t)+C2*sin(2^(1/2)*t)
>> dsolve('D2y=-2*x+y','y(0)=1','x')
 ans =
(2*x*exp(x)+(-C2+1)*exp(x)^2+C2)/exp(x)
```

3.10. Интегральные преобразования

В этом разделе описаны важные интегральные преобразования. Преобразования Фурье составляют основу метода спектрального анализа сигналов и вопросов, связанных с анализом радиотехнических цепей. Трудно переоценить и преобразования Лапласа, составляющие основу символического метода расчета электрических и радиотехнических цепей. Z-преобразования широко применяются в теории автоматического управления. Все эти преобразования (прямые и обратные) рассматриваются в данном разделе.

3.10.1. Преобразования Фурье

Прямым преобразованием Фурье является следующее выражение:

$$F(w) = \int_{-\infty}^{\infty} f(x) e^{-iwx} dx,$$

где $f(x)$ — скалярная функция независимой переменной x. Оно реализуется функцией fourier:

- F=fourier(f) — возвращает $F(w)$ и определяет независимую переменную с помощью функции findsym (по умолчанию это x). Если $f = f(w)$, то возвращается функция $F = F(t)$. Таким образом, преобразование имеет вид $f = f(x) \Rightarrow F = F(w)$;
- F=fourier(f,v) — аналогична ранее приведенной функции, но заменяет аргумент возвращаемой функции F (по умолчанию — w) на v, то есть реализует преобразование Фурье по формуле

$$F(v) = \int_{-\infty}^{\infty} f(x) e^{ivx} dx;$$

- fourier(f,u,v) — аналогична исходной функции, но заменяет аргумент x в f(x) на u, а аргумент в F(w) на v, то есть дает следующее преобразование:

$$F(v) = \int_{-\infty}^{\infty} f(u) e^{-ivu} du.$$

Примеры прямого преобразования Фурье:

```
>> syms f F x w u v
```

```
>> fourier(0.1*x)
ans =
1/5*i*pi*Dirac(1,w)
>> F=fourier(sin(x),v)
F =
-i*pi*Dirac(v-1)+i*pi*Dirac(v+1)
>> syms t
>> f=1/t^2;
>> F=fourier(f,v)
F =
pi*v*(Heaviside(-v)-Heaviside(v))
>> fourier(exp(-x^2),x,t)
ans =
pi^(1/2)*exp(-1/4*t^2)
>> fourier(diff(sym('F(x)')),x,w)
ans =
i*w*fourier(F(x),x,w)
```

Обратное преобразование Фурье обычно реализуется формулой:

$$f(x) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(w)e^{iwx}dw.$$

Для его осуществления используется функция `ifourier`:

- `f=ifourier(F)` — возвращает результат обратного преобразования Фурье над скалярной символьной функцией `F` независимой переменной `w`. По умолчанию возвращается функция `F(x)`. Таким образом, преобразование имеет вид $F = F(w) \Rightarrow f = f(x)$. Если $F = F(x)$, то данная функция возвращает функцию переменной t: $f = f(t)$.

Существуют и другие формы обратного преобразования Фурье, указанные ниже.

- `f=fourier(F,u)` — осуществляет обратное преобразование Фурье с заменой в $f(x)$ x на u. Таким образом, реализуется следующая формула преобразования:

$$f(u) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(w)e^{iwu}dw.$$

- `f=ifourier(F,v,u)` — осуществляет обратное преобразование Фурье, заменяя в $f(x)$ x на u и в $F(w)$ w на v, реализуя следующую формулу преобразования:

$$f(u) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(v)e^{ivu}dv.$$

Примеры обратного преобразования Фурье приведены ниже:

```
>> syms t x u w
>> ifourier(sin(w))
ans =
1/2*i*(-Dirac(x+1)+Dirac(x-1))
```

```
>> ifourier(w*exp(-2*w)*sym('Heaviside(w)'))
ans =
1/2/(-2+i*x)^2/pi
>> ifourier(1/(1 + 2*w),u)
ans =
-1/4*i*exp(-1/2*i*u)*(-Heaviside(u)+Heaviside(-u))
>> ifourier(sym('fourier(f(x),x,w)'),w,x)
ans =
f(x)
```

Безусловно, отсутствие нормальной математической нотации в выводах несколько снижает ценность таких вычислений. Но, в принципе, они реализуются. Если значительная часть научного проекта пользователя основана на символьных вычислениях, то лучше перейти к их реализации в системах класса Maple и Mathematica.

3.10.2. Преобразования Лапласа

Прямое преобразование Лапласа осуществляется по основной формуле:

$$L(s) = \int_0^\infty f(t)e^{-st}dt.$$

Для осуществления этого преобразования используется функция `laplace`:

- `L=laplace(F)` — обеспечивает прямое преобразование Лапласа для скалярной символьной функции $f(t)$ с независимой переменной t. Результат — функция $L(s)$. Если $f = f(s)$, то возвращается функция $L = L(t)$;
- `L=laplace(F,t)` — обеспечивает прямое преобразование Лапласа по модифицированной формуле

$$L(t) = \int_0^\infty f(x)e^{-tx}dx;$$

- `L=laplace(F,w,z)` — обеспечивает преобразование Лапласа по формуле

$$L(z) = \int_0^\infty f(w)e^{-zw}dw.$$

Примеры прямого преобразования Лапласа:

```
syms a s t w x
```

`laplace(t^5)`	возвращает	`120/s^6`
`laplace(exp(a*s))`	возвращает	`1/(t-a)`
`laplace(sin(w*x),t)`	возвращает	`w/(t^2+w^2)`
`laplace(cos(x*w),w,t)`	возвращает	`t/(t^2+x^2)`
`laplace(x^sym(3/2),t)`	возвращает	`s*pi^(1/2)/t^(5/2)`
`laplace(diff(sym('F(t)')))`	возвращает	`laplace(F(t),t,s)*s-F(0)`

Обратное преобразование Лапласа выполняется по следующей главной формуле:

$$f(t) = \frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} L(s)e^{st}ds,$$

где c — действительное число, такое что все особенности функции $L(s)$ расположены слева от вертикали $s = c$, i — мнимая единица. Данное преобразование осуществляется функцией `ilaplace`:

- `F=ilaplace(L)` — возвращает результат обратного преобразования Лапласа для скалярной символьной функции `L` с независимой переменной по умолчанию `s`. Если `L = L(t)`, то возвращается функция `F = F(x)`.

Есть еще две формы преобразования:

- `F=ilaplace(L,y)` — возвращает результат обратного преобразования Лапласа по формуле

$$f(y) = \frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} L(s)e^{sy}ds.$$

- `F=ilaplace(L,y,x)` — выполняет результат обратного преобразования Лапласа по формуле

$$f(y) = \frac{1}{2\pi i}\int_{c-i\infty}^{c+i\infty} L(x)e^{xy}dx.$$

Примеры обратного преобразования Лапласа:

```
syms s t w x y
```

`ilaplace(1/(s-1))`	возвращает	`exp(t)`
`ilaplace(1/(t^2+1))`	возвращает	`sin(x)`
`ilaplace(t^(-sym(5/2)),x)`	возвращает	`4/3/pi^(1/2)*x^(3/2)`
`ilaplace(y/(y^2 + w^2),y,x)`	возвращает	`cos(w*x)`
`ilaplace(sym('laplace(F(x),x,s)'),s,x)`	возвращает	`F(x)`

3.10.3. Z-преобразования

Z-преобразование особенно широко используется в теории автоматического управления. Оно описывается следующим соотношением:

$$F(z) = \sum_{n=0}^{\infty} \frac{f(n)}{Z^n},$$

которое вычисляет z-преобразование для скалярной функции f независимой переменной n (по умолчанию). В MATLAB оно реализуется функцией `ztrans`:

- `F=ztrans(f)` — обеспечивает прямое Z-преобразование вида `f` $= f(n) \Rightarrow$ `F` $= F(z)$, где `n` — символьная переменная, определяемая функцией `findsym`. Если `f` $= f(z)$, то `ztrans(f)` возвращает `F` $= F(w)$;
- `F=ztrans(f,w)` — возвращает `F`, заменяя аргумент по умолчанию `z` на `w`, то есть осуществляет преобразование

$$F(w) = \sum_{n=0}^{\infty} \frac{f(n)}{w^n}.$$

Наконец, еще одна функция — `F=ztrans(f,k,w)` — дает z-преобразование по формуле

$$F(w) = \sum_{k=0}^{\infty} \frac{f(k)}{w^k}.$$

Примеры прямого z-преобразования:

```
syms k n w z
```

`Ztrans(2^n)`	возвращает	`z/(z-2)`
`Ztrans(sin(k*n),w)`	возвращает	`sin(k)*w/(1-2*w*cos(k)+w^2)`
`Ztrans(cos(n*k),k,z)`	возвращает	`z*(-cos(n)+z)/(-2*z*cos(n)+z^2+1)`
`Ztrans(cos(n*k),n,w)`	возвращает	`w*(-cos(k)+w)/(-2*w*cos(k)+w^2+1)`
`Ztrans(sym('f(n+1)'))`	возвращает	`z*ztrans(f(n),n,z)-f(0)*z`

Обратное z-преобразование для функции $F(n)$ задается выражением

$$f(n) = \frac{1}{2\pi i} \oint_{|z|=R} F(z) z^{n-1} dz,$$

где $n = 1, 2, \ldots$, а R — положительное число, определяющее аналитичность функции $F(z)$ вне круга $|z| = R$. Такое преобразование реализуется следующей функцией:

- `f=iztrans(F)` — возвращает результат обратного z-преобразования для скалярной символьной функции `F` независимой переменной `z`. Это преобразование определяется как `F` = $F(z) \Rightarrow$ `f` = $f(n)$. Если `F` = $F(n)$, то `iztrans` возвращает функцию `f` = $f(k)$.

Другая функция — `f=iztrans(F,k)` — дает то же, но заменяет аргумент возвращаемой функции по умолчанию `n` на `k`. Таким образом, она реализует преобразование по формуле

$$f(k) = \frac{1}{2\pi i} \oint_{|z|=R} F(z) z^{k-1} dz \text{ , где } k = 1, 2, \ldots$$

Наконец, функция `f=iztrans(F,w,k)` делает еще одну замену, заменяя аргумент исходной функции (по умолчанию — `z`) на `v`, реализуя соотношение

$$f(k) = \frac{1}{2\pi i} \oint_{|v|=R} F(v) v^{k-1} dv.$$

Эти преобразования можно проверить по следующим примерам:

`iztrans(z/(z-2))`	возвращает	`2^n`
`iztrans(exp(x/z),z,k)`	возвращает	`x^k/k!`

3.11. Символьные операции с выражениями

3.11.1. Упрощение выражений

Функция `simplify(S)` поэлементно упрощает символьные выражения массива `S`. Если упрощение невозможно, то возвращается исходное выражение. Примеры:

```
>> syms a b x;
>> V=[sin(x)^2+cos(x)^2 log(a*b)]
V =
[ sin(x)^2+cos(x)^2, log(a*b)]
>> simplify(V)
ans =
[1, log(a*b)]
>> simplify((a^2-2*a*b+b^2)/(a-b))
ans =
a-b
```

К сожалению, возможности проведения упрощений с помощью команды `simplify` в пакете Symbolic Math не вполне соответствуют возможностям системы Maple, в связи с отсутствием опций, определяющих путь упрощения.

Функция `simple(S)` выполняет различные упрощения для элементов массива `S` и выводит как промежуточные результаты, так и самый короткий конечный результат. В другой форме — `[R,HOW] = simple(S)` — промежуточные результаты не выводятся. Результаты упрощений содержатся в векторе `R`, а в строковом векторе `HOW` указывается выполняемое преобразование. Следующие примеры иллюстрируют работу функции:

S	R	HOW
`cos(x)^2+sin(x)^2`	`1`	`combine(trig)`
`2*cos(x)^2-sin(x)^2`	`3*cos(x)^2-1`	`simplify`
`cos(x)^2-sin(x)^2`	`cos(2*x)`	`combine(trig)`
`cos(x)+(-sin(x)^2)^(1/2)`	`cos(x)+i*sin(x)`	`radsimp`
`cos(x)+i*sin(x)`	`exp(i*x)`	`convert(exp)`
`(x+1)*x*(x-1)`	`x^3-x`	`collect(x)`
`x^3+3*x^2+3*x+1`	`(x+1)^3`	`factor`
`cos(3*acos(x))`	`4*x^3-3*x`	`expand`

3.11.2. Расширение выражений

Функция `expand(S)` расширяет выражения, входящие в массив `S`. Рациональные выражения она раскладывает на простые дроби, полиномы — на полиномиальные разложения и т. д. Функция работает со многими алгебраическими и тригонометрическими функциями. Ниже представлены примеры использования функции `expand`:

```
>> syms a b x;
>> S=[(x+2)*(x+3)*(x+4) sin(2*x)];
```

```
>> expand(S)
ans =
[ x^3+9*x^2+26*x+24, 2*sin(x)*cos(x)]
>> expand(sin(a+b))
ans =
sin(a)*cos(b)+cos(a)*sin(b)
>> expand((a+b)^3)
ans =
a^3+3*a^2*b+3*a*b^2+b^3
```

3.11.3. Разложение выражений на простые множители

Функция `factor(S)` поэлементно разлагает выражения вектора `S` на простые множители, а целые числа — на произведение простых чисел. Следующие примеры иллюстрируют применение функции:

```
>> x=sym('x');
>> factor(x^7-1)
ans =
(x-1)*(x^6+x^5+x^4+x^3+x^2+x+1)
>> factor(x^2-x-1)
ans =
x^2-x-1
>> factor(sym('123456789'))
ans =
(3)^2*(3803)*(3607)
```

3.11.4. Комплектование по степеням

Функция `collect(S,v)` обеспечивает комплектование выражений в составе вектора или матрицы `S` по степеням переменной `v`. А функция `collect(S)` выполняет аналогичные действия относительно переменной, определяемой функцией `findsym`. Примеры применения данной функции:

```
>> syms x y
>> S=[x^3*y^2+x^2*y+3*x*y^2 x^4*y-y*x^2];
>> collect(S,x)
ans =
[ x^3*y^2+x^2*y+3*x*y^2,          x^4*y-x^2*y]
>> collect(S,y)
ans =
[ (x^3+3*x)*y^2+x^2*y,            (x^4-x^2)*y]
>> collect(S)
ans =
[ x^3*y^2+x^2*y+3*x*y^2,           x^4*y-x^2*y]
```

3.11.5. Приведение к рациональной форме

Функция `[N,D]=numden(A)` преобразует каждый элемент массива `A` в рациональную форму в виде отношения двух неприводимых полиномов с целочисленными коэффициентами. При этом `N` и `D` — числители и знаменатели каждого преобразованного элемента массива. Примеры:

```
>> [n,d]=numden(sym(8/10))
n =
4
d =
5
>> syms x y
>> [n,d]=numden(x*y+y/x)
n =
y*(x^2+1)
d =
x
```

3.11.6. Приведение к схеме Горнера

Функция `horner(P)` возвращает символьный полином или массив символьных полиномов `P`, преобразованный по схеме Горнера, минимизирующей число операций умножения. Пример:

```
>> x=sym('p');
>> horner(x^5-2*x^4-3*x^3-2*x^2-5*x-6)
ans =
-6+(-5+(-2+(-3+(-2+p)*p)*p)*p)*p
>> horner([x^3+x^2+x,(x^3+1)*(x^2+2)*(x+3)])
ans =
[ (1+(1+x)*x)*x, (x^3+1)*(x^2+2)*(x+3)]
```

3.11.7. Запись с подстановками

Функция

`[Y,SIGMA]=subexpr(X,SIGMA)` или `[Y,SIGMA]=subexpr(X,'SIGMA')`

преобразует символьное выражение `X`, обеспечивая при этом подстановку `SIGMA`. Для представления подвыражений используются обозначения `%1`, `%2` и т. д.

Пример:

```
>> t=solve('a*x^3+b*x+c=0');
>> [r,s]=subexpr(t,'s')
r =
[ 1/6/a*s^(1/3)-2*b/s^(1/3) ]
[ -1/12/a*s^(1/3)+b/s^(1/3)+1/2*i*3^(1/2)*(1/6/a*s^(1/3)+2*b/s^(1/3)) ]
[ -1/12/a*s^(1/3)+b/s^(1/3)-1/2*i*3^(1/2)*(1/6/a*s^(1/3)+2*b/s^(1/3)) ]
s =
(-108*c+12*3^(1/2)*((4*b^3+27*c^2*a)/a)^(1/2))*a^2
```

3.11.8. Обеспечение подстановок

Одной из самых эффектных и часто используемых операций символьной математики является операция подстановки. Она реализуется функцией `subs`, имеющей ряд форм записи:

- `subs(S)` — заменяет в символьном выражении `S` все переменные их символьными значениями, которые берутся из вычисляемой функции или рабочей области системы MATLAB;

- subs(S,NEW) — заменяет все свободные символьные переменные в S из списка NEW;
- subs(S,OLD,NEW) — заменяет OLD на NEW в символьном выражении S. При одинаковых размерах массивов OLD и NEW замена идет поэлементно. Если S и OLD — скаляры, а NEW — числовой массив или массив ячеек, то скаляры расширяются до массива результатов.

Если подстановка subs(S,OLD,NEW) не меняет S, то выполняется подстановка subs(S,NEW,OLD).

- subs(S,OLD,NEW,0) — исключает попытку обратной подстановки.

Примеры:

```
>> syms a b x y;
>> subs(x-y,y,1)
ans =
x-1
>> subs(sin(x)+cos(y),[x,y],[a,b])
ans =
sin(a)+cos(b)
>> subs(exp(a*x),a,-magic(3))
ans =
[ exp(-8*x),      exp(-x),    exp(-6*x)]
[ exp(-3*x),    exp(-5*x),    exp(-7*x)]
[ exp(-4*x),    exp(-9*x),    exp(-2*x)]
```

3.11.9. Обращение функции

Часто возникает необходимость в задании функции, обратной по отношению к заданной функции *f*. Для этого в Symbolic имеется функция обращения inverse, которая задается в двух формах:

- g=finverse(f) — возвращает функцию, обратную к f. Считается, что f — функция одной переменной, например 'x'. Тогда g(f(x)) = x;
- g=finverse(f,v) — возвращает функцию, обратную к f, относительно заданной переменной v, так что g(f(v)) = v. Эта форма используется, если *f* — функция нескольких переменных.

Примеры:

```
>> syms x
>> finverse(sinh(x))
ans =
asinh(x)
>> finverse(exp(x))
ans =
log(x)
```

3.11.10. Суперпозиция функций

К числу часто встречающихся в символьной математике манипуляций с функциями относится суперпозиция функций, реализуемая функциями compose:

- `compose(f,g)` — возвращает $f(g(y))$, где `f` = $f(x)$ и `g` = $g(y)$. Независимые переменные `x` и `y` находятся с помощью функции `findsym`;
- `compose(f,g,z)` — возвращает $f(g(z))$, где `f` = $f(x)$, `g` = $g(y)$;
- `compose(f,g,x,z)` — возвращает $f(g(z))$ и при этом рассматривает x как независимую переменную для функции f. Так, если `f` = $\cos(x/t)$, то `compose(f,g,x,z)` возвращает $\cos(g(z)/t)$, а `compose(f,g,t,z)` возвращает $\cos(x/g(z))$;
- `compose(f,g,x,y,z)` — возвращает $f(g(z))$ и рассматривает x как независимую переменную для функции f и y как независимую переменную для функции g. Для $f = \cos(x/t)$ и $g = \sin(y/u)$ `compose(f,g,x,y,z)` возвращает $\cos(\sin(z/u)/t)$, а `compose(f,g,x,u,z)` возвращает $\cos(\sin(y/z)/t)$.

Следующие примеры поясняют применение функции `compose`:

```
syms x y z t u;
f = 1/(1 + x^2); g = sin(y); h = x^t; p = exp(-y/u);
```

`compose(f,g)`	возвращает	`1/(1+sin(y)^2)`
`compose(f,g,t)`	возвращает	`1/(1+sin(t)^2)`
`compose(h,g,x,z)`	возвращает	`sin(z)^t`
`compose(h,g,t,z)`	возвращает	`x^sin(z)`
`compose(h,p,x,y,z)`	возвращает	`exp(-z/u)^t`
`compose(h,p,t,u,z)`	возвращает	`x^exp(-y/z)`

3.12. Преобразования объектов

При практической работе с системами символьной математики зачастую необходимы взаимные преобразования различных объектов, например численных в символьные и наоборот. К примеру, если вы аналитически получили производную функции, то для построения графика производной вам надо преобразовать формулу для нее в вычисленные численные значения. В этом разделе рассматриваются такие преобразования, введенные в пакет расширения Symbolic. Рассматриваются также специальные математические функции, калькулятор, средства графики и доступа в ядро символьных операций.

3.12.1. Преобразование символьной матрицы в числовую

В связи с использованием в символьных вычислениях символьных выражений возникает необходимость в преобразовании их в обычные числа с плавающей точкой и обычной двойной точностью. Для этого служит функция `double(S)`, которая преобразует символьную матрицу `S` поэлементно в матрицу вычисленных значений символьных выражений. Примеры такого преобразования даны ниже:

```
>> double('sin(1)')
ans =
    115   105   110    40    49    41
>> double(sym('sin(1)'))
ans =
```

```
    0.8415
>> double(sym('1+2*sin(1)'))
ans =
    2.6829
```

3.12.2. Преобразование вектора коэффициентов полинома в символьный полином

Если задан полином, коэффициенты которого хранятся в векторе C, то функция poly2sym(C) преобразует его в символьное представление полинома в стандартной форме записи с независимой переменной x. Другие функции — poly2sym(C,'V') и poly2sym(C,SYM('V') — делают то же, но позволяют задать независимую переменную полинома как V. Примеры:

```
>> poly2sym([3 2 1])
ans =
3*x^2+2*x+1
>> poly2sym([3 2 1],'p')
ans =
3*p^2+2*p+1
>> poly2sym([sin(1) 3 2 1])
ans =
3789648413623927/4503599627370496*x^3+3*x^2+2*x+1
```

Обратите внимание на то, что в последнем примере коэффициент полинома при x^3 оказался представленным значением sin(1) с дробно-рациональным видом.

3.12.3. Преобразование символьного полинома в вектор его коэффициентов

Если задан символьный полином P, то функция sym2poly(P) возвращает вектор его коэффициентов. Это поясняют следующие примеры:

```
>> syms x
>> sym2poly(3*x^2+2*x+1)
ans =
     3        2        1
>> sym2poly(exp(1)*x^2+sin(1)*x+1)
ans =
    2.7183    0.8415    1.0000
```

Как показывает последний пример, коэффициенты исходного полинома могут быть арифметическими выражениями, которые при преобразовании вычисляются.

3.12.4. Преобразование символьного объекта в строковый

Функция char(A) преобразует символьный объект A в строку. Если A — вектор или матрица, результат представляется в форме 'array([[...]])'. Примеры преобразования даны ниже:

```
>> Y=char('1+2*sin(1)')
Y =
```

```
1+2*sin(1)
>> double(sym(Y))
ans =
2.6829
```

3.13. Вычисление специальных функций

3.13.1. Интегральные синус и косинус

Для вычисления интегрального синуса

$$Si(z) = \int_0^t \frac{\sin(t)}{t} dt$$

служит функция `sinint(z)`. Примеры ее применения:

```
>> sinint(1)
ans =
    0.9461
>> sinint(2+3i)
ans =
    4.5475 + 1.3992i
```

Интегральный косинус определяется выражением

$$Ci(z) = \gamma + \ln(z) + \int_0^z \frac{\cos(t) - 1}{t} dt$$

при $|\arg(z)| < \pi$. Здесь γ — постоянная Эйлера (0,5772...). Примеры:

```
>> cosint(1)
ans =
    0.3374
>> cosint(pi/4)
ans =
    0.1853
>> cos(2+3i)
ans =
    -4.1896 - 9.1092i
```

3.13.2. Дзета-функция Римана

Дзета-функция Римана определяется выражением:

$$\varsigma(s) = \sum_{k=1}^{\infty} k^{-s}$$

при $\mathrm{Re}(s) > 0$. Примеры ее применения показаны ниже:

```
>> zeta(2)
ans =
     1.6449
>> zeta(pi+2i)
```

```
ans =
    0.9794 - 0.1336i
>> zeta([1 2 3])
ans =
   Inf   1.6449   1.2021
```

3.13.3. W-функция Ламберта

W-функция Ламберта является решением трансцендентного уравнения $w*\exp(w) = x$ и задается функцией `lambertw(X)` или `lambertw(K,X)`.

В последнем виде функция находит K-ю комплексную ветвь для многозначной функции. Примеры на применение этой функции:

```
>> lambertw(2+3i)
ans =
    1.0901 + 0.5301i
>> lambertw(3,2+3i)
ans =
   -1.6214 +18.1726i
```

3.13.4. Суммы Римана

Функции `rsums(f)` и `rsums f` вычисляют приближение Римана к интегралу с подынтегральной функцией $f(x)$ и строят в виде столбцовой диаграммы график функции и площадей под кривой.

Пример применения данной функции представлен на рис. 3.4. Здесь используется функция f(x) = tan(x). Число термов можно менять от 2 до 256.

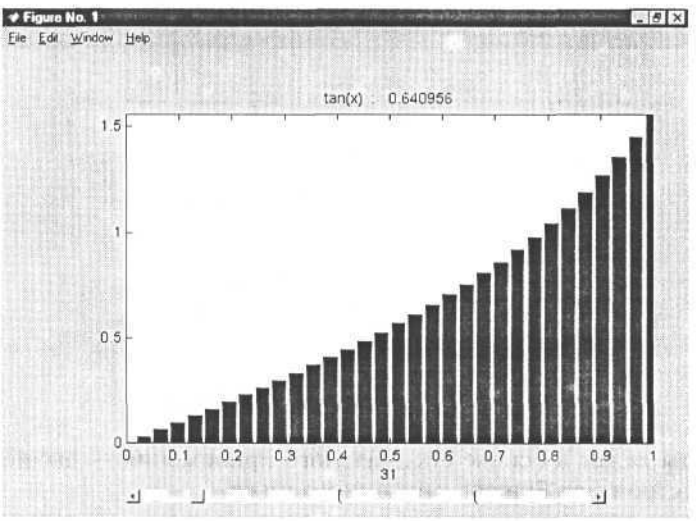

Рис. 3.4. Пример применения функции rsums

Обратите внимание на характерный ползунковый регулятор под графиком представления площадей. Меняя положения движка, можно изменять в широких пределах число сумм (термов) и следить за тем, насколько точно они представляют выбранную функцию.

3.14. Графические возможности пакета

3.14.1. Графики символьных функций

Чтобы избавить пользователя от возни с вполне возможным построением графиков функций с помощью стандартных средств (например команды `plot`), в пакет Symbolic введены довольно удобные команды класса `ezplot`:

- `ezplot(f)` — строит график символьно заданной функции $f(x)$ независимой переменной `'x'` в интервале `[-2*pi, 2*pi]`;
- `ezplot(f,xmin,xmax)` или `ezplot(f,[xmin,xmax])` — делает то же, но позволяет задать диапазон изменения независимой переменной `x` от `xmin` до `xmax`;
- `ezplot(f,[xmin xmax],fig)` — обеспечивает спецификацию графика с помощью параметра `fig`.

Команды класса `ezplot` позволяют строить графики функций, имеющих особенности. С примером построения графика функции $\sin(x)/x$ с особенностью при $x = 0$ мы уже знакомились в главе 1. Другой пример такого рода — построение графика функции $\tan(x)$, имеющего разрывы:

```
>> ezplot('tan(x)',0,20)
>> grid on
```

На рис. 3.5 показан вид построенного графика. Он выглядит намного естественнее, чем аналогичный график, построенный командой `plot`.

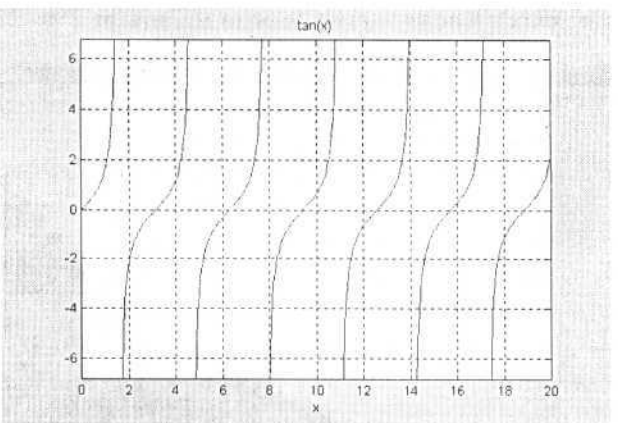

Рис. 3.5. График функции tan(x)

Эти команды лежат в основе специального приложения — вычислителя функций и графопостроителя Funtool, описанного ниже.

3.14.2. Калькулятор и графопостроитель

Команда `funtool` создает интерактивный графический калькулятор, позволяющий быстро построить две функции одной переменной — $f(x)$ и $g(x)$. Например, одна может задавать собственно функцию, а другая — ее производную. Функции обозначаются как `'f = '` и `'g = '` и после знака равенства можно набрать функ-

ции с помощью клавиш калькулятора в его нижней части. С помощью полей 'x = ' и 'a = ' можно задать диапазон изменения переменной x и значение масштабирующего параметра a.

При запуске команды funtool появляются окна для двух функций и окно калькулятора (рис. 3.6.) По умолчанию заданы функции $f(x) = x$ и $g(x) = 1$, предел изменения x от -2π до 2π и $a = 1/2$. Все окна масштабируемые и перемещаемые.

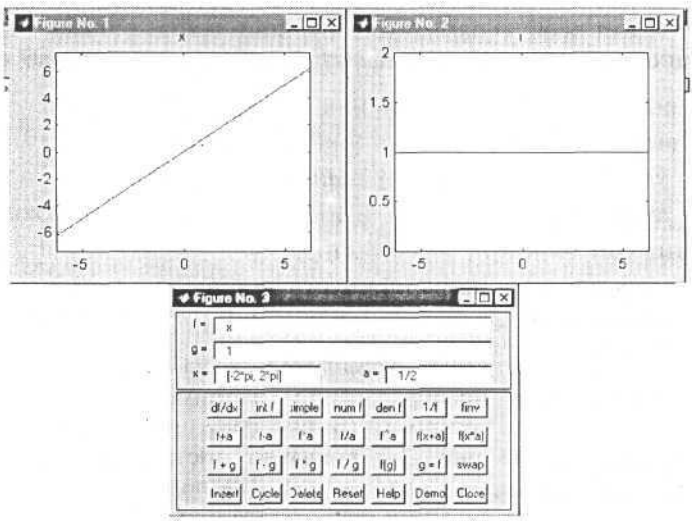

Рис. 3.6. Внешний вид вычислителя функций и окон двух графиков

Верхний ряд кнопок вычислителя относится только к функции $f(x)$ и задает следующие операторы:

- df/dx — символьное дифференцирование $f(x)$;
- int f — символьное интегрирование $f(x)$ при наличии замкнутой формы;
- simple f — упрощение выражения, если таковое возможно;
- num f — выделение числителя рационального выражения;
- den f — выделение знаменателя рационального выражения;
- 1/f — замена $f(x)$ на $1/f(x)$;
- finv — замена $f(x)$ инверсной функцией.

Второй ряд клавиш выполняет операции масштабирования и сдвига $f(x)$ с применением параметра 'a':

- f + a — заменяет $f(x)$ на $f(x) + a$;
- f – a — заменяет $f(x)$ на $f(x) - a$;
- f * a — заменяет $f(x)$ на $f(x) * a$;
- f / a — заменяет $f(x)$ на $f(x) / a$;
- f ^ a — заменяет $f(x)$ на $f(x)$ ^ a;
- f(x+a) — заменяет $f(x)$ на $f(x + a)$;
- f(x*a) — заменяет $f(x)$ на $f(x * a)$.

Третий ряд клавиш ряд предназначен для осуществления бинарных операций над функциями $f(x)$ и $g(x)$:

- f + g — заменяет $f(x)$ на $f(x) + g(x)$;

- f – g — заменяет $f(x)$ на $f(x) - g(x)$;
- f * g — заменяет $f(x)$ на $f(x) * g(x)$;
- f / g — заменяет $f(x)$ на $f(x) / g(x)$;
- f(g) — заменяет $f(x)$ на $f(g(x))$;
- g = f — заменяет $g(x)$ на $f(x)$;
- swap — меняет $f(x)$ и $g(x)$ местами.

Калькулятор имеет особую память для функций в виде списка `fxlist`. Четвертый ряд клавиш служит для работы с памятью калькулятора и иных операций:

- Insert — помещает текущую функцию в список функций;
- Cycle — выполняет текущую функцию из списка;
- Delete — удаляет выделенную функцию из списка;
- Reset — устанавливает `f`, `g`, `x`, `a` и `fxlist` в исходное состояние;
- Help — выводит описание калькулятора;
- Demo — запускает демонстрационный пример;
- Close — завершает работы с калькулятором.

Благодаря описанным средствам вычислитель позволяет задать интересующую вас функцию, выполнить ее преобразования (например, дифференцирование и интегрирование) и, наконец, построить график функции и результатов ее преобразования. Для примера на рис. 3.7 показано построение графика функции $f(x) = \sin(x)/x$ и ее производной — график $g(x)$. Для получения этих графиков потребовался ввод $f(x)$, нажатие клавиши df/dx и клавиши swap.

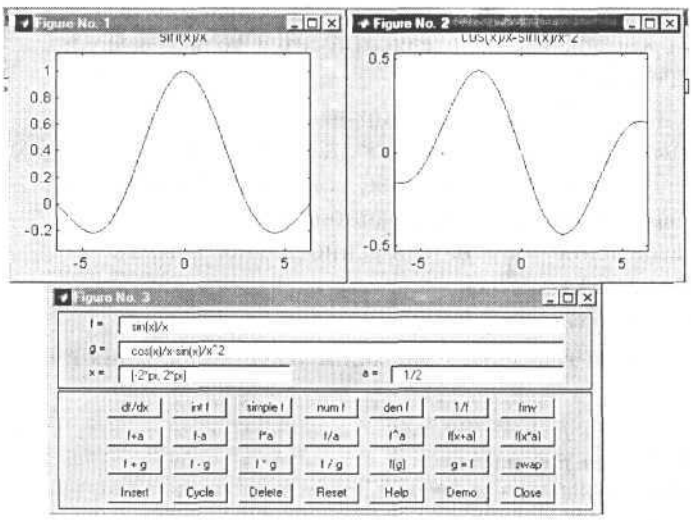

Рис. 3.7. Графики функции $sin(x)/x$ и ее производной

Кнопка Cycle позволяет просмотреть графики ряда интересных функций (рис. 3.8) в качестве примера. Кнопкой Demo можно запустить демонстрацию возможностей вычислителя и графопостроителя. Список демонстрируемых функций можно изменить или пополнить, в частности, оставив в нем только интересующие вас функции.

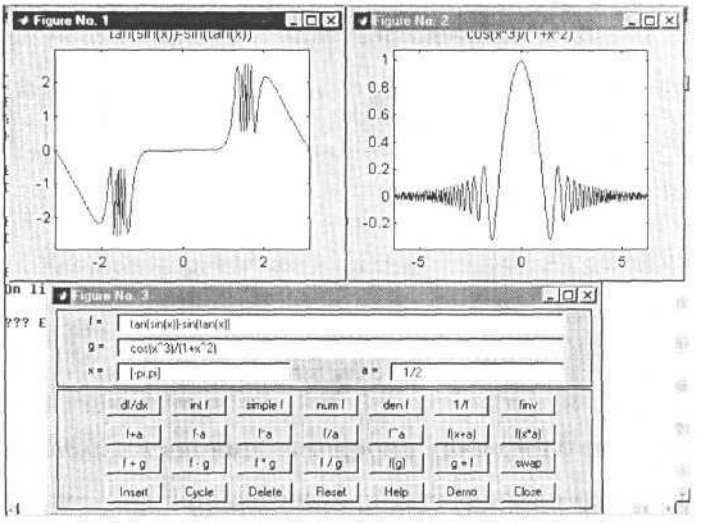

Рис. 3.8. Фрагмент демонстрации построения функций из списка fxlist

Вычислитель с графопостроителем и впрямь является весьма удобным средством визуализации графиков самых различных функций. Эта утилита заменяет множество объемных книг по графикам элементарных и специальных функций.

3.14.3. Контурные графики

Помимо упомянутых графических функций, пакет Symbolic поддерживает построение графиков различных типов. Так, функция `ezcontour` служит для построения контурных графиков функций вида $f(x,y)$. Эта функция записывается в следующем виде:

- `ezcontour(f)` — строит контурный график с настройкой по умолчанию;
- `ezcontour(f,domain)` — строит контурный график с заданными параметром `domain` пределами изменения x и y;
- `ezcontour(...,n)` — обеспечивает ранее указанные построения при явном задании числа линий `n`.

Следующий пример

```
>> syms x y
>> ezcontour(sin(x*y),[-3,3],30)
```

строит контурный график функции $\sin(x*y)$, представленный на рис. 3.9.

Похожая на рассмотренную выше функция `ezcontourf` строит контурные графики с функциональной окраской (fill) областей между линиями равного уровня:

- `ezcontourf(f)` — строит контурный график с окраской и настройкой по умолчанию;
- `ezcontourf(f,domain)` — строит контурный график окраской с заданными параметром `domain` пределами изменения x и y;
- `ezcontourf(...,n)` — обеспечивает ранее указанные построения при явном задании числа линий `n`.

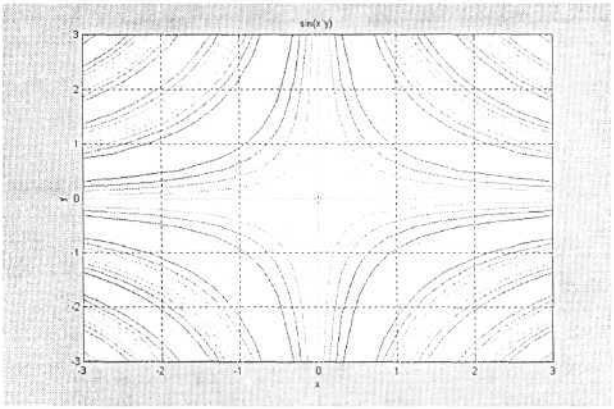

Рис. 3.9. Пример построения контурного графика командой ezcontour

Следующий пример иллюстрирует применение этой функции для построения контурного графика выражения sin(*x*)*sin(*y*):

```
>> syms x y
>> ezcontourf(sin(x)*sin(y),[-3,3],50)
```

График данной функции представлен на рис. 3.10.

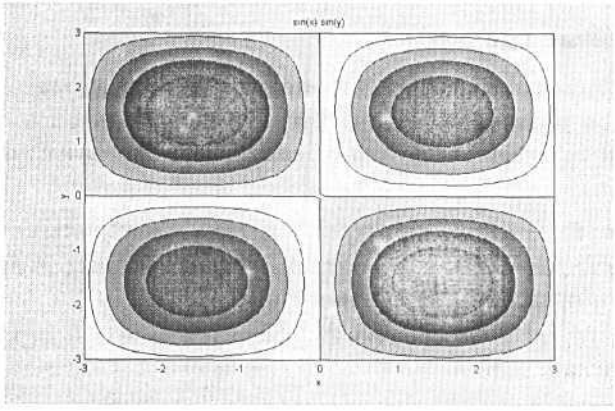

Рис. 3.10. Пример построения контурного графика с функциональной окраской командой ezcontourf

3.14.4. Трехмерные графики параметрически заданных функций

Для построения трехмерных графиков параметрически заданных функций служит команда `ezplot3`:

- `ezplot3(x,y,z)` — строит трехмерный график функции, заданной параметрически уравнениями $x(t)$, $y(t)$, $z(t)$ при настройке по умолчанию;
- `ezplot3(x,y,z,[tmin tmax])` — строит трехмерный график функции, заданной параметрически уравнениями $x(t)$, $y(t)$, $z(t)$ при изменении аргумента t от `tmin` до `tmax`;

- `ezplot3(...,'animate')` — аналогична предшествующим командам, но обеспечивает анимацию при построении графика.

Следующий пример показывает, как можно построить пространственную спираль с элементами анимации:

```
>> syms t; ezplot3(cos(t),sin(t),t,[0 20],'animate')
```

Заключительная стадия построения показана на рис. 3.8. При наличии опции `'animate'` по спирали движется шарик, который также можно видеть на рис. 3.11. Можно повторить движение шарика, нажав кнопку Repeat в графическом окне. Эта кнопка и шарик отсутствуют при отсутствии опции `'animate'`.

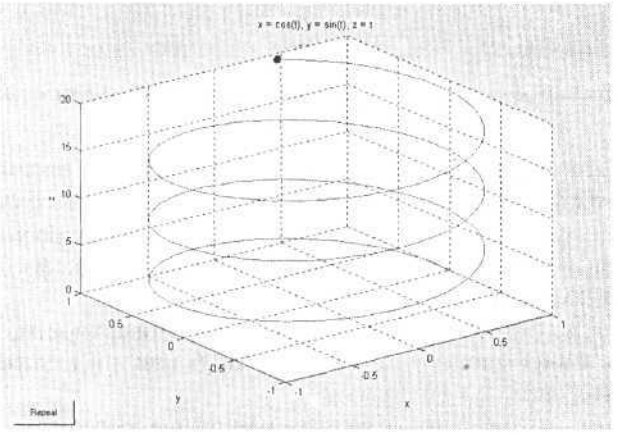

Рис. 3.11. Построение объемной спирали и движущегося по ней шарика

3.14.5. Полярные графики

График функции $f(t)$ в полярной системе координат строит команда `ezpolar`:

- `ezpolar(f)` — строит график функции $f(t)$ в полярной системе координат при изменении угла t от 0 до 2π;
- `ezpolar(f,[a b])` — строит график функции $f(t)$ в полярной системе координат при изменении угла t от a до b.

Применение этой команды иллюстрирует следующий пример:

```
>> syms t
>> ezpolar(sin(2*t))
```

Окно с построенным в этом примере графиком представлено на рис. 3.12.

3.14.6. Графики поверхностей

Команда `ezsurf` служит для построения графиков поверхностей, задаваемых функциями двух переменных $f(x,y)$:

- `ezsurf(f)` — построение поверхности $f(x,y)$ с параметрами x и y, меняющимися по умолчанию от -2π до 2π;
- `ezsurf(f,domain)` — построение поверхности $f(x,y)$ с пределами изменения x и y, заданными параметром `domain`;

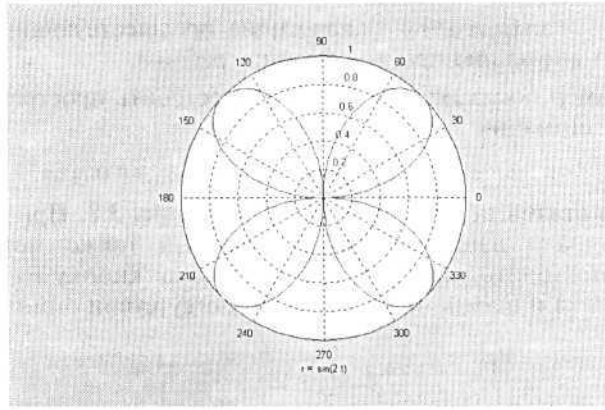

Рис. 3.12. Пример построения графика в полярной системе координат командой ezpolar

- `ezsurf(x,y,z)` — построение поверхности, заданной параметрически зависимостями $x(s,t)$, $y(s,t)$, $z(s,t)$ при s и t, меняющихся в интервале от -2π до 2π.
- `ezsurf(x,y,z,[smin, smax, tmin, tmax])` — построение поверхности, заданной параметрически зависимостями $x(s,t)$, $y(s,t)$, $z(s,t)$ при s и t, меняющихся в заданном интервале;
- `ezsurf(x,y,z,[min, max])` — построение поверхности, заданной параметрически зависимостями $x(s,t)$, $y(s,t)$, $z(s,t)$ при s и t, меняющихся в одинаковом интервале от `min` до `max`;
- `ezsurf(...,n)` — аналогична описанным выше командам, но с заданным числом линий сетки `n`;
- `ezsurf(...,'circ')` — аналогична описанным выше командам, но вписывает поверхность в окружность.

Следующий пример показывает действие этой команды:

```
>> syms x y
>> ezsurf(real(asec(x+i*y)))
```

Построенный в этом примере график представлен на рис. 3.13.

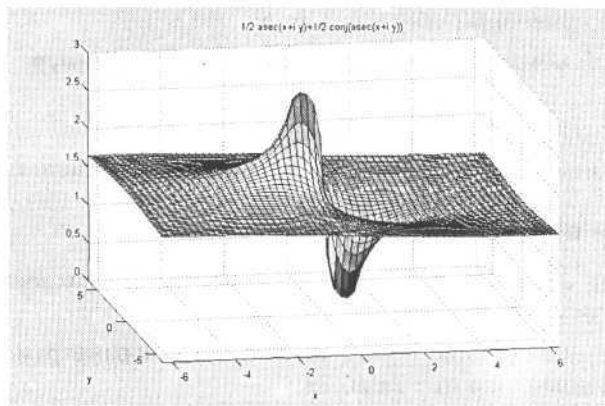

Рис. 3.13. Пример построения поверхности командой ezsurf

Есть совершенно аналогичная по синтаксису записи группа команд `ezsurfc`. В отличие от предшествующей группы команд в этом случае строится еще и контурный график поверхности на плоскости, лежащей под поверхностью. Следующий пример поясняет такое построение:

```
>> syms x y
>> ezsurfc(real(asec(x+i*y)))
```

Сами графики (поверхности в пространстве и контурный на плоскости) показаны на рис. 3.14.

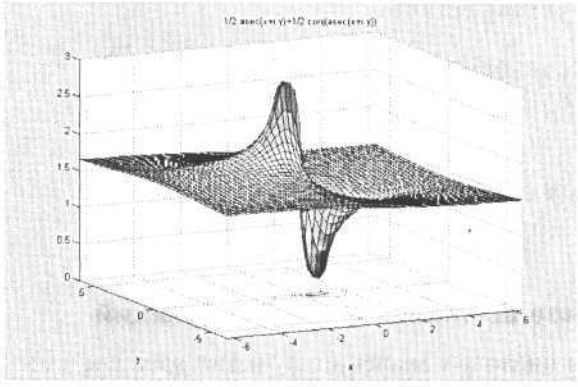

Рис. 3.14. Пример построения поверхности командой ezsurfc и контурного графика под поверхностью

Описанные выше команды графики похожи на соответствующие команды MATLAB. Однако, благодаря применению средств символьной математики, они задаются в более естественной форме и обычно упрощают построение сложных математических графиков.

3.15. Доступ к ресурсам ядра системы Maple

В этом разделе описаны функции, дающие доступ к ресурсам ядра системы символьной математики Maple, включенным в систему MATLAB. В студенческой версии MATLAB эти возможности отсутствуют. Применение возможностей системы Maple совместно с возможностями системы MATLAB придает последней особую гибкость и резко расширяет возможности в решении сложных математических задач, где целесообразно объединять аналитические (символьные) методы с численными вычислениями.

3.15.1. Доступ к ядру системы Maple

Хотя в пакет Symbolic включено множество (около сотни) функций символьной математики, есть возможность получить доступ к многим другим функциям системы Maple, ядро которой включено в MATLAB. Для этого используется функция `maple` в той или иной форме:

- `maple(STATEMENT)` — устанавливает выражение `STATEMENT` для ядра Maple. `STATEMENT` является строкой, записанной в формате команд Maple. Возвращает результат в форме строки с нотацией системы Maple V;

- `maple('function',ARG1,ARG2,...)` — дает доступ к Maple-функциям с заданными аргументами;
- `[RESULT,STATUS]=maple(...)` — возвращает результат и статус сообщений/ошибок, если они возникают в ходе вычисления заданной функции.

Примеры:

```
>> maple('sin(1)')
ans =
sin(1)
>> maple('evalf(sin(1))')
ans =
.84147098480789650665250232163030
>> maple('sqrt',2)
ans =
2^(1/2)
>> maple('evalf(sin)',1)
ans =
sin(1)
```

3.15.2. Численное вычисление Maple-функций

Для вычисления числовых значений функций ядра системы Maple используется функция `mfun`:

- `mfun('fun',p1,p2,...,pk)` — возвращает в численном виде значение Maple-функции с именем `'fun'` и списком параметров функции `p1`, `p2`, ..., `pk`. Если вычисление функции невозможно (например из-за сингулярных точек), то возвращается константа `NaN`.

```
>> mfun('sin',1)
ans =
    0.8415
>> mfun('sqrt',2+3i)
ans =
    1.6741 + 0.8960i
>> mfun('ln',0)
ans =
    NaN
>> mfun('ln',2)
ans =
    0.6931
```

3.15.3. Вызов списка функций Maple

Команда `mfunlist` выводит лист с перечнем функций ядра Maple. Ниже показан первый десяток этих функций:

```
>> mfunlist
bernoulli    n       Bernoulli Numbers
bernoulli    n,z     Bernoulli Polynomials
BesselI      x1,x    Bessel Function of the First Kind
BesselJ      x1,x    Bessel Function of the First Kind
BesselK      x1,x    Bessel Function of the Second Kind
```

```
BesselY      x1,x    Bessel Function of the Second Kind
Beta         z1,z2   Beta Function
binomial     x1,x2   Binomial Coefficients
LegendreKc   x       Complete Elliptic Integral of First Kind
LegendreEc   x       Complete Elliptic Integral of Second Kind
...
```

3.15.4. Получение справки по ядру Maple

Для получения справки по Maple-функциям непосредственно из среды MATLAB служат команда и функция mhelp:

mhelp topic или mhelp('topic')

Пример:

```
>> help LCM
 LCM    Least common multiple.
   LCM(A,B) is the least common multiple of corresponding
 elements of
   A and B. The arrays A and B must contain positive integers
   and must be the same size (or either can be scalar).
   See also GCD.
```

3.15.5. Инсталляция Maple-процедур

С помощью команды procread(FILENAME) можно готовить процедуры с синтаксисом языка системы Maple. Например, подготовим процедуру с предполагаемым именем файла check.src и следующим текстом:

```
check := proc(A)
    # check(A) computes A*inverse(A)
    local X;
    X := inverse(A):
    evalm(A &* X);
end;
```

Создав процедуру, можно проинсталлировать ее с помощью команды

```
procread('check.src')
```

После этого возможно использование этой процедуры в виде

```
maple('check',magic(3))
```

или

```
maple('check',vpa(magic(3)))
```

Применение процедур с синтаксисом Maple расширяет возможности программирования системы MATLAB и свидетельствует об углублении процесса интеграции систем MATLAB и Maple.

3.16. Комплексный пример на исследование функции

3.16.1. Задание функции и построение ее графика

В заключение этой главы рассмотрим комплексный пример из справки на исследование функции одной переменной.

Вначале зададим некоторую функцию f, вычислим ее вторую производную f2 и построим ее график:

```
>> f = 1/(5+4*cos(x))
   f =
   1/(5+4*cos(x))
   >> f2 = diff(f,2)
   f2 =
   32/(5+4*cos(x))^3*sin(x)^2+4/(5+4*cos(x))^2*cos(x)
   >> ezplot(f2)
axis([-2*pi 2*pi -5 2])
title('Graph of f2')
```

График функции f2 представлен на рис. 3.15. Эту функцию мы и будем использовать для последующих вычислений.

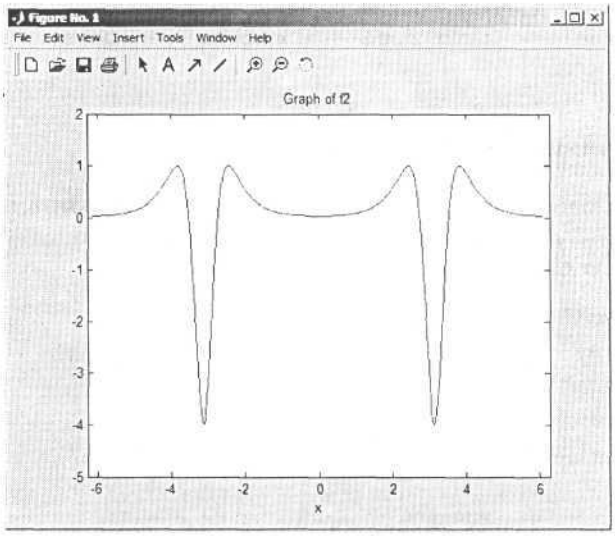

Рис. 3.15. График функции f2

Теперь найдем третью производную от f — функцию f3:

```
>> f3 = diff(f2); pretty(f3);
```

Убрав точку с запятой в конце, вы можете просмотреть довольно громоздкое выражение для третьей производной в формате pretty, имитирующем математическую нотацию.

3.16.2. Поиск и визуализация корней третьей производной

Теперь найдем корни третьей производной — функции f3:

```
>> zeros = solve(f3)
zeros =
[                                                          0]
[       atan((-255-60*19^(1/2))^(1/2),10+3*19^(1/2))]
[      atan(-(-255-60*19^(1/2))^(1/2),10+3*19^(1/2))]
[  atan((-255+60*19^(1/2))^(1/2)/(10-3*19^(1/2)))+pi]
[ -atan((-255+60*19^(1/2))^(1/2)/(10-3*19^(1/2)))-pi]
```

Выведем их в формате с плавающей точкой при 5 цифрах:

```
>> format; % Default format of 5 digits
zerosd = double(zeros)
zerosd =
        0
        0 + 2.4381i
        0 - 2.4381i
  2.4483
  -2.4483
```

Построим график функции f3 с найденными корнями:

```
>> ezplot(f3)
hold on;
plot(zerosd,0*zerosd,'ro') % Plot zeros
plot([-2*pi,2*pi], [0,0],'g-.'); % Plot x-axis
title('Graph of f3')
Warning: Imaginary parts of complex X and/or Y arguments ignored.
```

Этот график представлен на рис. 3.16. Заметим, что корни f3 соответствуют локальным экстремумам функции f2.

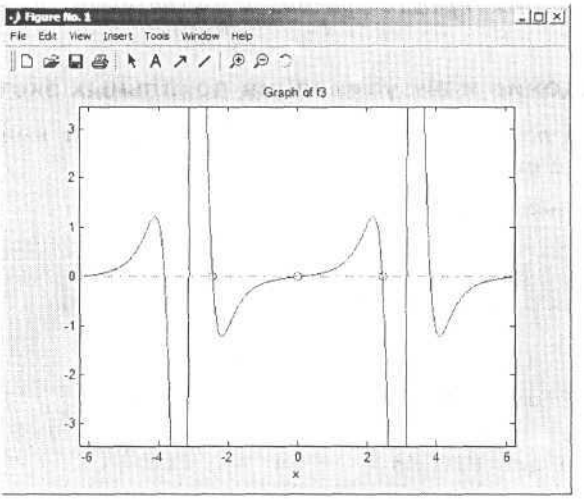

Рис. 3.16. График функции f3 с найденными нулями

Можно попытаться расширить область поиска корней, что иллюстрирует следующий пример:

```
>> zerosd = [zerosd(1) zerosd(4) zerosd(5) pi];
>> ezplot(f3)
hold on;
plot(zerosd,0*zerosd,'ro')
plot([-2*pi,2*pi], [0,0],'g-.'); % Plot x-axis
title('Zeros of f3')
hold off;
```

Для этого случая график f3 с найденными корнями представлен на рис. 3.18. Нетрудно заметить, что на нем есть еще один корень.

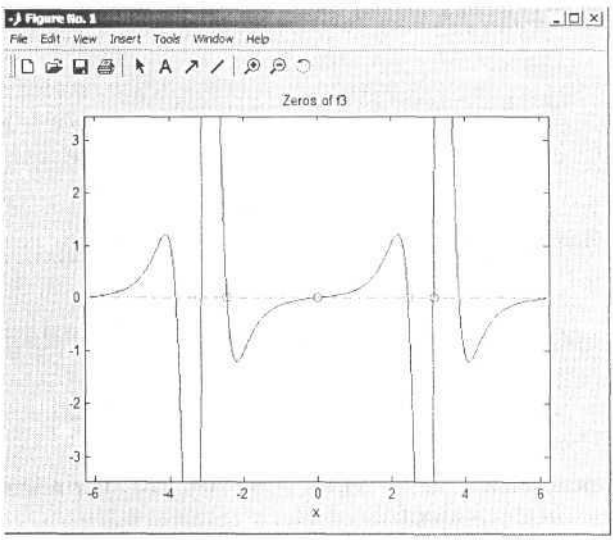

Рис. 3.17. График функции f3 с найденными нулями

3.16.3. Нахождение и визуализация локальных экстремумов

Теперь найдем локальные экстремумы функции f2 и выполним построение графика функции с ними:

```
>> [zerosd; subs(f2,zerosd)]
ans =
         0      2.4483     -2.4483      3.1416
    0.0494      1.0051      1.0051     -4.0000
>> clf
ezplot(f2)
axis([-2*pi 2*pi -4.5 1.5])
ylabel('f2');
title('Maxima and Minima of f2')
hold on
plot(zerosd, subs(f2,zerosd), 'ro')
text(-4, 1.25, 'Absolute maximum')
text(-1,-0.25,'Local minimum')
text(.9, 1.25, 'Absolute maximum')
```

```
text(1.6, -4.25, 'Absolute minimum')
hold off;
```

Этот график представлен на рис. 3.18 и дает наглядное представление о положении найденного локального максимума и двух локальных минимумов. Таким образом анализ функции можно считать завершенным.

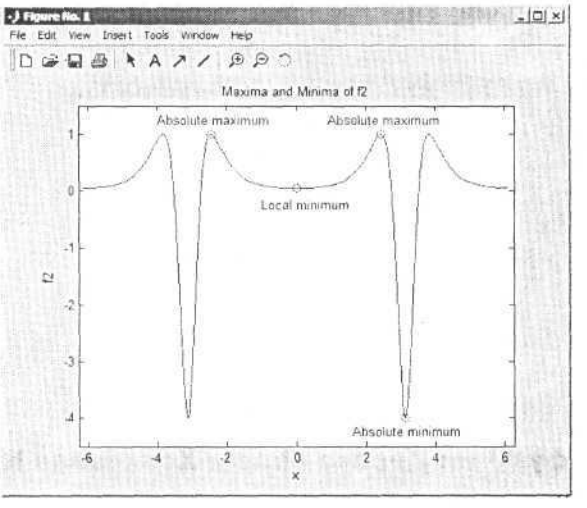

Рис. 3.18. График функции f2 с найденными локальными экстремумами

Объективно оценивая возможности пакета Symbolic Math, нельзя не отметить, что система Mapl 8 имеет удобные и законченные средства для подобного и даже расширенного анализа функций одной переменной.

3.17. Новые функции пакета Symbolic Math Toolbox 3.1

3.17.1. Новые функции округления чисел

В новейшую реализацию Symbolic Math Toolbox 3.1, поставляемую в составе комплекса MATLAB 7 + Simulink 6 введены пять функций округления чисел:

`ceil(x)` — округление числа x до ближайшего большего целого;
`fix(x)` — округление числа отбросом дробной части;
`floor(x)` — округление числа до ближайшего меньшего целого;
`frac(x)` — вычисление дробной части числа;
`round(x)` — округление числа до ближайшего целого.

Следующие примеры демонстрируют применение этих функций:

```
>> x = sym(-7/2); [fix(x) floor(x) round(x) ceil(x) frac(x)]
 ans =
[   -3,   -4,   -4,   -3,   -1/2]
>> x = sym(-7/3); [fix(x) floor(x) round(x) ceil(x) frac(x)]
 ans =
[   -2,   -3,   -2,   -2,   -1/3]
```

Функции явно относятся к числовым.

3.17.2. Новая функция quorem

Новая функция

```
[Q,R] = quorem (A,B)
```

вычисляет частное Q и остаток R для матриц A и B с целочисленными или полиномиальными элементами. Простые примеры применения этой функции представлены ниже:

```
>> syms x; p = x^3-3*x+7; [q,r] = quorem(x^5,p)
 q =
 x^2+3
 r =
 -7*x^2-21+9*x
 >> [q,r] = quorem(10^4,subs(p,'10'))
 q =
 10
 r =
 230
```

3.17.3. Новые функции Дирака dirac и Хевисайда heaviside

Две новые функции широко используются для представления и обработки сигналов:

- dirac(x) — дельта-функция Дирака, возвращающая 0 при x < 0 и 1 в ином случае (функция создает единичный скачок в момент t = 0);
- heaviside(x) — функция Хевисайда, возвращающая 0 при x < 0, 1 при x > 0 и NaN при x = 0.

Примеры на применение этих функций:

```
>> syms x a; a = 5; int(dirac(x-a)*sin(x),-inf, inf)
ans =
sin(5)
>> diff(heaviside(x),x)
ans =
dirac(x)
```

3.17.4. Новая функция сортировки sort

Для сортировки массивов с символьными данными введена функция sort(v). Ее применение вполне очевидно:

```
>> syms a b c d e x
>> sort([x d c e b a])
ans =
[ a, b, c, d, e, x]
```

Стоит отметить, что символы кириллицы применять в массивах для этой функции нельзя. Уже функция syms отреагирует на это ошибкой.

3.17.5. Новая функция вычисления коэффициентов полиномов с многими переменными

Функция

```
C = coeffs(P) C = coeffs(P,X) [C,T]=coeffs(p,x)
```

Выполняет вычисление коэффициентов полиномов p относительно переменной x. Примеры применения этой функции представлены ниже:

```
>> syms x y;
>> z = 2*x^2*y^2 + 3*x*y^3;
>> [c,t] = coeffs(z,y)
 c =
 [ 2*x^2, 3*x]
 t =
 [ y^2, y^3]
```

3.17.6. Новая функция reshape задания символьных массивов

Новая функция reshape служит для задания (формирования) символьных массивов. Это поясняется следующим примером:

```
>> syms x; A = reshape (x.^(1:9),1,3,3).
A(:,:,1) =
[ x, x^2, x^3]
A(:,:,2) =
[ x^4, x^5, x^6]
A(:,:,3) =
[ x^7, x^8, x^9]
```

3.17.7. Новые функции преобразования

Набор функций

```
int8(S) int16(S) int32(S) int64(S)
```

преобразует символьную матрицу в матрицу с целочисленными элементами соответствующей разрядности.

Функция

```
single(S)
```

преобразует символьную матрицу в матрицу с элементами в виде чисел одинарной точности.

А функции

```
uint8(S) uint16(S) uint32(S) uint64(S)
```

преобразуют символьную матрицу в матрицу с целочисленными элементами соответствующей разрядности без знака.

Интересно, что в справке нет примеров применения этих функций. А между тем они дают вовсе не тривиальные результаты, например:

```
>> int8('[1 2; 3 4]')
ans =
    91   49   32   50   59   32   51   32   52   93
```

```
>> uint8('[1 -2; 3 -4]')
ans =
     91   49   32   45   50   59   32   51   32   45   52   93
```

3.17.8. Функции вычисления логарифмов

Две новые функции введены для вычисления логарифмов по основанию 2 и 10:

```
log10(X) и log2(X)
>> log2(10)
```

Примеры их применения вполне очевидны:

```
ans =
     3.3219
>> log10(2)
ans =
     0.3010
```

Менее очевидно применение этих функция в пакете расширения по символьной математики, поскольку функции явно относятся к численным.

3.17.9. Функция вычисления по модулю

Функция C = mod(A, B) для символьных матриц A и B с целочисленными элементами вычисляет положительный остаток от поэлементного деления A на B. Для матриц с полиномиальными entries, mod(A, B) возвращает их коэффициенты. Например:

```
>> syms x; mod(x^3-2*x+799,5)
ans =
x^3+3*x+4
```

В целом надо отметить, что эти нововведения практически не меняют возможностей данного пакета в области символьных вычислений.

Глава 4. Решение задач оптимизации

4.1. Пакет оптимизации Optimization Toolbox

4.1.1. Назначение и возможности пакета

Возможности базовой системы MATLAB в решении практически весьма важных задач оптимизации довольно ограничены. Поэтому для решения таких задач интересен пакет оптимизации Optimization Toolbox. Ниже описана реализация пакета 2.3 (R13 SP1). Уточнить версию продукта можно, исполнив команду ver.

В MATLAB 7 используется версия 3.0, которая в пределах рассмотрения в данной главе идентична версии 2.3. В новую версию введена функция целочисленного бинарного программирования `bintprog`, усовершенствованы алгоритмы функций `fminunc` и `exitflag` и введены новые опции выхода.

Основные типы таких задач, решаемых системой MATLAB с этим пакетом, представлены в табл. 4.1. С методами оптимизации можно познакомиться по многочисленной литературе, например [47—50].

Таблица 4.1

Типы задач, решаемых пакетом Optimization Toolbox

Тип задачи	Математическая запись	Используемая функция MATLAB
	Задачи минимизации	
Скалярная (одномерная) минимизация	$\min_a f(a),\ a_1 < a < a_2$	`fminbnd`
Безусловная минимизация (без ограничений)	$\min_x f(x)$	`fminunc, fminsearch`
Линейное программирование	$\min_x f^T x$ при условиях $A \cdot x \le b,\ Aeq \cdot x = beq,$ $x_L \le x \le x_U$	`linprog`
Квадратичное программирование	$\min_x \frac{1}{2} x^T H x + f_T x$ при условиях $A \cdot x \le b,\ Aeq \cdot x = beq,$ $x_L \le x \le x_U$	`quadprog`
Минимизация при наличии ограничений	$\min_x f(x)$ при условиях $c(x) \le 0,\ ceq(x) = 0,$ $A \cdot x \le b,\ Aeq \cdot x = beq,$ $x_L \le x \le x_U$	`fmincon`

Тип задачи	Математическая запись	Используемая функция MATLAB
Достижение цели	$\min_{x,\gamma}\gamma$ при условиях $F(x) - w\gamma \le goal$, $c(x) \le 0$, $ceq(x) = 0$, $A \cdot x \le b$, $Aeq \cdot x = beq$, $x_L \le x \le x_U$	`fgoalattain`
Минимакс	$\min_x \max_{\{F_i\}}\{F_i(x)\}$ при условиях $c(x) \le 0$, $ceq(x) = 0$, $A \cdot x \le b$, $Aeq \cdot x = beq$, $x_L \le x \le x_U$	`fminimax`
Полубесконечная минимизация	$\min_x f(x)$ при условиях $K(x,w) \le 0$ для всех w, $c(x) \le 0$, $ceq(x) = 0$, $A \cdot x \le b$, $Aeq \cdot x = beq$, $x_L \le x \le x_U$	`fseminf`
Нахождение решений уравнений		
Линейные уравнения	$C \cdot x = d$, n уравнений, n переменных	\ (оператор левого деления, `slash`)
Нелинейное уравнение одной переменной	$f(a) = 0$	`fzero`
Нелинейные уравнения многих переменных	$F(x) = 0$, n уравнений, n неизвестных	`fsolve`
Задачи аппроксимации («подгонки» кривых)		
Линейный метод наименьших квадратов (МНК)	$\min_x \|C \cdot x - d\|_2^2$, m уравнений, n переменных	\ (оператор левого деления, `backslash`)
Неотрицательный линейный МНК	$\min_x \|C \cdot x - d\|_2^2$ при условии $x \ge 0$	`lsqnonneg`
Линейный МНК при наличии ограничений	$\min_x \|C \cdot x - d\|_2^2$ при условиях $A \cdot x \le b$, $Aeq \cdot x = beq$, $x_L \le x \le x_U$	`lsqlin`

Тип задачи	Математическая запись	Используемая функция MATLAB
Нелинейный МНК	$\min_x \frac{1}{2}\|F(x)\|_2^2 = \frac{1}{2}\sum_i f_i^2(x)$ при условии $x_L \le x \le x_U$	`lsqnonlin`
Нелинейная «подгонка» кривой	$\min_x \frac{1}{2}\|F(x,xdata) - ydata\|_2^2$ при условии $x_L \le x \le x_U$	`lsqcurvefit`

Принятые обозначения:

- a — скалярный аргумент; x, γ — в общем случае векторные аргументы;
- $f(a)$, $f(x)$ — скалярные функции; $F(x)$, $c(x)$, $ceq(x)$, $K(x,w)$ — векторные функции,
- A, Aeq, C, H — матрицы;
- b, beq, d, f, w, $goal$, $xdata$, $ydata$ — векторы;
- x_L, x_U — соответственно нижняя и верхняя граница области изменения аргумента.

4.1.2. Применяемые алгоритмы

В рамках пакета Optimization Toolbox все задачи оптимизации делятся на две группы:

- задачи малой и средней размерности;
- задачи большой размерности.

Соответственно, такое же деление принято для алгоритмов решения данных задач. Разумеется, это не означает, что для решения задач средней размерности нельзя применять алгоритмы большой размерности и наоборот. Просто алгоритмы той или иной группы более эффективны для задач своей размерности.

В пакете Optimization Toolbox реализован широкий набор алгоритмов для решения задач оптимизации средней и малой размерности:

- основными для задач без ограничений являются симплексный метод Нелдера—Мида и квазиньютоновские методы;
- для решения задач с ограничениями, минимакса, достижения цели и полубесконечной оптимизации использованы алгоритмы квадратичного программирования;
- задачи, сводящиеся к нелинейным МНК, решаются с помощью алгоритмов Ньютона—Рафсона и Левенберга—Марквардта;
- вспомогательные процедуры одномерной (скалярной) оптимизации используют алгоритмы квадратичной (параболической) и кубической интерполяции.

Рассмотрим перечисленные алгоритмы подробнее.

4.1.3. Общая формулировка задачи параметрической оптимизации

Задача параметрической оптимизации формулируется как задача нахождения набора параметров $x = \{x_1, x_2, ..., x_n\}$, который является оптимальным в смысле некоторого критерия. В простейшем случае такая задача сводится к минимизации или максимизации некоторой (целевой) функции без каких-либо ограничений. В более сложных ситуациях на отмеченные параметры могут быть наложены некоторые ограничения в виде равенств $g_i(x) = 0$ ($i = 1, 2, ..., m_e$), неравенств $g_i(x) \le 0$ ($i = m_e+1, ..., m$) и/или параметрических границ x_L, x_U.

Общая формулировка задачи параметрической оптимизации представляется следующим образом: требуется найти вектор x, обеспечивающий

$$\min_{x \in R^n} f(x)$$

при ограничениях

$$g_i(x) = 0,\ i = 1, 2, ..., m_e,$$

$$g_i(x) \le 0,\ i = m_e+1, ..., m,$$

$$x_L \le x \le x_U,$$

где x — вектор оптимизируемых параметров ($x \in R^n$), $f(x)$ — скалярная целевая функция (критерий) векторного аргумента ($f(x)$: $R^n \to R$), $g_i(x)$ — также некоторые скалярные функции векторного аргумента (заметим, что задача максимизации сводятся к задаче минимизации заменой $f(x)$ на $-f(x)$).

Эффективность и точность решения данной задачи зависит как от числа параметров и ограничений, так и от вида целевой функции. При линейных ограничениях и целевой функции приведенная задача оптимизации называется задачей линейного программирования, при линейных ограничениях, но при квадратичной (по аргументам) целевой функции — задачей квадратичного программирования, в общем случае это задача нелинейного программирования.

4.1.4. Безусловная оптимизация

Существующие алгоритмы безусловной оптимизации могут быть разделены на две группы — алгоритмы, базирующиеся на использовании производных минимизируемой функции (градиентные и методы второго порядка) и алгоритмы, использующие только значения функции (безградиентные).

Безградиентные методы (например, симплексный метод Нелдера—Мида) более пригодны для задач, где минимизируемая функция является существенно нелинейной или имеет разрывы. Градиентные методы (методы первого порядка) обычно эффективны в случаях целевых функций, непрерывных вместе с первыми производными. Методы второго порядка, такие как метод Ньютона, применяются реже, поскольку требуют больших вычислительных затрат для расчета матриц вторых производных.

Градиентные методы используют информацию о наклоне функции для выбора направления поиска экстремума. В одном из таких методов — наискорейшего спуска — на каждой итерации движение к точке минимума осуществляется в направлении $-\nabla f(x)$ (где $\nabla f(x)$ — вектор-градиент целевой функции $f(x)$). Этот метод неэффективен в ситуациях, когда поверхность целевой функции имеет узкие «овраги», как, например, у известной функции Розенброка

$$f(x) = 10(x_1 - x_2^2)^2 + (1 - x_1)^2.$$

Минимальное значение данной функции, как нетрудно видеть, равно нулю при $x_1 = x_2 = 1$. Между тем численные эксперименты показывают, что зачастую метод наискорейшего спуска не обеспечивает нахождение точки экстремума даже после сотен и тысяч итераций.

Отметим, что указанную функцию из-за своеобразной формы ее линий равного уровня часто называют «банановой» функцией (как видно, банановыми бывают не только республики!) и используют как тестовую при проверке эффективности различных оптимизационных алгоритмов.

Квазиньютоновские алгоритмы. Среди алгоритмов, использующих информацию о градиенте, наиболее распространенными являются квазиньютоновские. В этих (итерационных) алгоритмах целевая функция в окрестностях произвольной точки аппроксимируется квадратичной функцией, при этом на каждой итерации решается задача локальной минимизации

$$\min_x \frac{1}{2} x^T \mathbf{H} x + c^T x + b,$$

где $\mathbf{H}$ — симметричная и положительно определенная матрица вторых частных и смешанных производных (матрица Гессе или гессиан), c — постоянный вектор, b — константа.

Оптимальное решение приведенной задачи соответствует нулевым значениям первых производных, то есть

$$\nabla f(x^*) = \mathbf{H} x^* + c = 0,$$

откуда

$$x^* = -\mathbf{H}^{-1} c.$$

4.1.5. Ньютоновские алгоритмы

Ньютоновские алгоритмы (в отличие от квазиньютоновских) непосредственно вычисляют $\mathbf{H}$ (как отмечалось, прямое вычисление матрицы $\mathbf{H}$ требует больших вычислительных затрат) и осуществляют движение в рассчитанном на очередной итерации направлении уменьшения целевой функции до достижения минимума (с использованием методов одномерного поиска). В квазиньютоновских алгоритмах такое вычисление не производится, а используется некоторая аппроксимация $\mathbf{H}$.

Среди подобных алгоритмов одним из наиболее популярных и используемым в пакете Optimization Toolbox является так называемый BFGS-алгоритм, получивший свое название по фамилиям предложивших его авторов (Broyden, Fletcher, Goldfarb, Shanno), в котором аппроксимация $\mathbf{H}$ производится итерационно, по формуле

$$H_{k+1} = H_k + \frac{q_k q_k^T}{q_k^T s_k} - \frac{H_k s_k s_k^T H_k}{s_k^T H_k s_k},$$

где

$$s_k = x_{k+1} - x_k,$$

$$q_k = \nabla f(x_{k+1}) - \nabla f(x_k).$$

Заметим, что более удобно иметь аппроксимацию не матрицы **H**, а обратной к ней матрицы $\mathbf{H}^{-1}$; приведенное рекуррентное соотношение подобную замену допускает, при этом сам алгоритм становится практически идентичным хорошо известному отечественным исследователям *алгоритму Давидона—Флетчера—Пауэлла*, за тем исключением, что в последнем векторы q заменены на векторы s и наоборот.

4.1.6. Алгоритмы Ньютона—Гаусса и Левенберга—Марквардта

Алгоритмы Ньютона—Гаусса и Левенберга—Марквардта используются в функциях рассматриваемого пакета, предназначенных для решения задачи нелинейного метода наименьших квадратов (МНК). При отсутствии ограничений указанная задача формулируется следующим образом:

$$\min_x \frac{1}{2}\|F(x)\|_2^2 = \frac{1}{2}\sum_i F_i^2(x).$$

Скалярная оптимизация в данных алгоритмах производится, соответственно, вдоль направлений

- $d_k = -\left(J^T(x_k)J(x_k)\right)^{-1} J(x_k)F(x_k)$ — для алгоритма Ньютона—Гаусса,
- $d_k = -\left(J^T(x_k)J(x_k) + \lambda_k I\right)^{-1} + J(x_k)F(x_k)$ — для алгоритма Левенберга—Марквардта,

где $\mathbf{J}(x)$ — матрица-якобиан размера $m \times n$ (в документации к пакету Optimization Toolbox под якобианом понимается матрица первых частных производных вектор-функции $F(x)$ по векторному аргументу x, а не определитель этой матрицы, как обычно принято в математической литературе; будем придерживаться терминологии документации пакета), λ_k — параметр алгоритма, определяемый в процессе линейной (скалярной) оптимизации вдоль выбранного направления.

4.1.7. Минимизация при наличии ограничений

В задачах оптимизации с ограничениями (см. табл. 4.1) обычный подход к нахождению решения состоит в замене исходной задачи с ограничениями на задачу без ограничений (задачу безусловной оптимизации), например, с помощью метода штрафных функций. В настоящее время, однако, более эффективным считается применение так называемых уравнений Куна—Таккера, которые на основании вышеприведенной формулировки задачи параметрической оптимизации и при некоторых дополнительных предположениях о характере ограничений записываются в виде:

$$f(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(x^*) = 0,$$

$$\nabla g_i(x^*) = 0\ ,\ i = 1, 2, \ldots, m_e,$$

$$\lambda_i^* = 0\ ,\ i = m_e+1, \ldots, m,$$

где λ_i — множители Лагранжа.

Для решения данных уравнений в пакете Optimization Toolbox использован алгоритм так называемого последовательного квадратичного программирования (в оригинале — Sequential Quadratic Programming или SQP), представляющий собой, по сути, разновидность квазиньютоновского метода. Основная идея SQP заключается в применении квадратичной аппроксимации функции Лагранжа (учитывающей ограничения)

$$L(x,\lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x),$$

так что на каждой итерации решается задача оптимизации

$$\min_{d \in R^n} \frac{1}{2} d^T H_k d + \nabla f^T(x_k) d,$$

$$\nabla g_i^T(x_k) d + g_i(x_k) = 0,\ i = 1, 2, \ldots, m_e,$$

$$\nabla g_i^T(x_k) d + g_i(x_k) \le 0,\ i = m_e+1, \ldots, m.$$

В последней формулировке данная задача может быть решена любым методом решения задач квадратичного программирования. В пакете Optimization Toolbox для этой цели использован комбинированный алгоритм, объединяющий алгоритм BFGS и так называемый метод проекций.

4.1.8. Многокритериальная оптимизация

Достаточно часто в реальных ситуациях качество работы исследуемого объекта или системы оценивается не единственным критерием или показателем качества, а совокупностью таких критериев, причем представляющихся одинаково значимыми. Это приводит к задаче оптимизации с векторной целевой функцией $F(x) = \{F_1(x), F_2(x), \ldots, F_m(x)\}$ и формулировкой

$$\min_{x \in R^n} F(x)$$

при ограничениях

$$g_i(x) = 0,\ i = 1, 2, \ldots, m_e,$$

$$g_i(x) \le 0,\ i = m_e+1, \ldots, m,$$

$$x_L \le x \le x_U,$$

получившей название задачи многокритериальной или векторной оптимизации.

Известно, что решение подобной задачи сводится к нахождению множества точек не улучшаемых решений (Парето-множества), для чего используются такие,

например, методы, как метод взвешенной суммы частных критериев или метод ε-ограничения.

В рассматриваемом пакете для решения задачи многокритериальной оптимизации применен так называемый *метод достижения цели*, предложенный Gembicki и математически описывающийся соотношениями

$$\min_{\gamma \in R,\, x \in \Omega} \gamma$$

при ограничениях

$$F_i(x) - w_i\gamma \le F_i^*,\ i = 1, 2, \ldots, m,$$

где Ω — область допустимых значений x, w_i — некоторые весовые коэффициенты, F_i^* — некоторые устанавливаемые «цели» (goals).

В приведенной формулировке задача подобна задаче однокритериальной оптимизации и может решаться с помощью вышеотмеченных алгоритмов.

4.1.9. Алгоритмы большой размерности

Задачи оптимизации с большим числом оптимизируемых факторов (десятки, сотни, тысячи) и/или сложным, нелинейным характером целевой функции достаточно плохо решаются рассмотренными методами ввиду существенных вычислительных затрат (например, при определении аппроксимации матрицы Гессе), что привело к выделению в отдельную группу алгоритмов, оказывающихся эффективными в подобных ситуациях — алгоритмов большой размерности.

Данные алгоритмы основаны на идее так называемой области доверия — области N вблизи некоторой точки x, в которой рассматриваемая целевая функция $f(x)$ может быть адекватно аппроксимирована более простой функцией $q(s)$, так что исходная задача оптимизации сводится к задаче

$$\min_{s \in N} q(s).$$

Текущая точка x обновляется, то есть заменяется точкой $x + s$, если $f(x + s) < f(x)$, и не изменяется в противоположном случае.

В стандартном подходе предполагается, что как функция $q(s)$, так и область доверия описываются квадратичными функциями, что приводит к задаче оптимизации

$$\min_s \frac{1}{2} s^T \mathbf{H} s + s^T G$$

при условии

$$\|\mathbf{D}s\| \le \Delta,$$

где $G = \nabla f(x)$ — градиент в текущей точке x, $\mathbf{D}$ — некоторая диагональная матрица весовых коэффициентов, $\|\bullet\|$ — обозначение нормы, Δ — положительный скаляр.

В пакете Optimization Toolbox используется двумерный вектор $s = [s_1, s_2]$, в котором элемент s_1 соответствует направлению градиента G, а элемент s_2 — направлению, выбираемому по одному из соотношений

$$\mathbf{H}s_2 = -G,$$

$$s_2^T \mathbf{H} s_2 < 0.$$

Очевидно, если направления s_1 и s_2 установлены, локальная оптимизация производится в двумерной области, что не вызывает никаких проблем с нахождением соответствующего решения. Наибольшую сложность здесь вызывает именно определение указанных направлений. Решение данной вспомогательной задачи производится модификацией метода сопряженных градиентов, так называемого метода PCG (Preconditioned Conjugate Gradients).

Для решения частных задач большой размерности (задачи с линейными ограничениями, линейного и нелинейного МНК, задач линейного и квадратичного программирования) в рассматриваемом пакете применены специальные модификации изложенного подхода, в том числе использующие операции с разреженными матрицами, подробности о которых можно выяснить в справочной системе MATLAB.

4.2. Практика оптимизации

4.2.1. Функции пакета Optimization Toolbox

Пакет имеет 19 функций, реализующих все его возможности. Функции пакета разделяются на пять групп:

- функции минимизации;
- решения уравнений;
- реализации метода наименьших квадратов (подбора кривых);
- утилиты;
- демонстрационные.

Подробная запись функций с расшифровкой обозначений ниже дана только для функций минимизации, для других функций она аналогична. В приведенных примерах дополнительные комментарии при выводе, как правило, опущены. Примеры даны для новейших версий MATLAB 6.5 SP1/7 и пакета Optimization Toolbox 2.3. В других версиях возможны расхождения в пределах машинной точности вычислений.

4.2.2. Решение задач максимизации

Функции пакета `fminbnd`, `fminsearch`, `fminunc`, `fmincon`, `fgoalattain`, `fminimax`, `lsqcurvefit` и `lsqnonlin` решают задачу *минимизации* целевой функции $f(x)$. Максимизация достигается, если решать задачу минимизации для целевой функции, взятой со знаком «минус», то есть $-f(x)$. Аналогичный результат достигается в задачах линейного и квадратичного программирования: там со знаком «минус» необходимо применять матрицу $\mathbf{H}$ и вектор f (то есть использовать $-\mathbf{H}$ и $-f$).

4.2.3. Приведение ограничений-неравенств к стандартному виду

В пакете Optimization Toolbox предполагается некоторая стандартная форма записи ограничений-неравенств: $c_i(x) \le 0$.

Неравенства иного вида должны быть приведены к данному (умножением обеих частей на -1, переносом констант в левую часть и т. п.), например:

$$c_i(x) \geq 0 \rightarrow -c_i(x) \leq 0,$$

$$c_i(x) \geq b \rightarrow -c_i(x) + b \leq 0.$$

4.2.4. Введение дополнительных аргументов

В списке аргументов рассмотренных функций пакета могут присутствовать дополнительные, обозначенные через `P1`, `P2` и т. д., например:

```
[x,fval] = fsolve('objfun',x0,options,P1,P2,...)
```

Эти дополнительные аргументы выполняют роль глобальных переменных для вызываемых целевых функций.

Пусть, например, требуется найти нули эллиптической функции Якоби `ellipj(u,m)`. Данная функция зависит от аргумента `u` и параметра `m` (указываемого как второй аргумент). Поиск ее нуля вблизи значения `u0 = 3` и при $m = 0.5$ может быть организован следующим образом:

```
>> m = 0.5;
>> options = optimset('Display','off'); % Отключение вывода
текущей информации
>> x = fsolve('ellipj',3,options,m) % Поиск решения
x =
    3.7081
```

Легко убедиться в том, что найденное решение корректное:

```
>> f = ellipj(x,m)
f =
-3.0042e-008
```

то есть оно действительно является нулем рассматриваемой функции.

Следующий пример относится к использованию функции `fgoalattain`:

```
>> fun = inline('sort(eig(A+B*x*C))','x','A','B','C'); %
Задание целевой функции
>> x = fgoalattain(fun, -ones(2,2), [-5,-3,-1], [5, 3, 1],...
[ ], [ ], [ ], [ ], -4*ones(2), 4*ones(2), [ ], [ ], A, B, C);
% Поиск решения
```

4.2.5. Многомерная минимизация с ограничениями

Для решения задач многомерной оптимизации с ограничениями в пакет Optimization Toolbox входит 9 функций:

- `x = fgoalattain(fun,x0,goal,w)` — возвращает решение задачи многомерной оптимизации при заданных векторе целевых функций `fun`, начальном приближении `x0`, заданных векторе целей `goal` и векторе весов `w`;

- `x = fgoalattain(fun,x0,goal,w,A,b)` — позволяет ввести ограничения в форме линейных неравенств $A \cdot x \le b$;
- `x = fgoalattain(fun,x0,goal,w,A,b,Aeq,beq)` — позволяет ввести дополнительные ограничения в форме равенств $Aeq \cdot x = beq$; если ограничения в форме неравенств отсутствуют, задаются `A=[ ]` и `b=[ ]`;
- `x = fgoalattain(fun,x0,goal,w,A,b,Aeq,beq,lb,ub)` — позволяет ввести дополнительные граничные ограничения $lb \le x \le ub$;
- `x = fgoalattain(fun,x0,goal,w,A,b,Aeq,beq,lb,ub,nonlcon)` — позволяет ввести дополнительные ограничения в форме нелинейных неравенств или равенств $c(x) \le 0$, $ceq(x) = 0$; при отсутствии граничных ограничений задаются `lb=[ ]` и/или `ub=[ ]`;
- `x = fgoalattain(fun,x0,goal,w,A,b,Aeq,beq,...lb,ub,nonlcon, options)` — позволяет задавать опций (см. ниже);
- `x = fgoalattain(fun,x0,goal,w,A,b,Aeq,beq,...lb,ub,nonlcon, options,P1,P2,...)` — позволяет задавать дополнительные параметры `P1,P2,...`, относящиеся к функциям-аргументам;
- `[x,fval] = fgoalattain(...)` — возвращает не только оптимальное значение векторного аргумента, но и значение целевой функции в точке минимума `fval`;
- `[x,fval,attainfactor] = fgoalattain(...)` — возвращается еще и коэффициент достижения цели `attainfactor`;
- `[x,fval,attainfactor,exitflag] = fgoalattain(...)` —возвращает еще и информацию о характере завершения вычислений `exitflag`;
- `[x,fval,attainfactor,exitflag,output] = fgoalattain(...)` — возвращается еще информация о результатах оптимизации (выходная структура) `output`;
- `[x,fval,attainfactor,exitflag,output,lambda] = fgoalattain(...)` — возвращаются еще и множители Лагранжа `lambda`.

Аргументы функции:

- `fun` — векторная функция векторного аргумента. Должна быть задана либо с помощью оператора @ (например `@sin(x.*x)`), функции `inline` (см. ограничения, указанные выше), например

```
>> fun = inline(' sin(x.*x)');
```

либо как m-файл, например,

```
function F = myfun(x)
F = ...
```

Если задано вычисление градиента (функцией `options = optimset( 'GradObj','on')`), то m-файл должен возвращать не только значение функции F, но и значения градиентов G:

```
function [F,G] = myfun(x)
F = ... % Вычисление векторной функции
G = ... % Вычисление градиента
```

- `goal` — вектор задаваемых целевых значений той же размерности, что и вектор `fun`;

- `w(weight)` — вектор весов той же размерности, что и вектор целей, часто принимается равным `abs(goal)`;
- `nonlcon` — функция, возвращающая значения функций-ограничений, а при необходимости (если задано `options = optimset('GradConstr','on')`) и их градиентов;
- `options` — опции (их можно изменять, используя функцию `optimset`):
- `DerivativeCheck` — задает проверку соответствия производных, определенных пользователем, их вычисленным оценкам в виде первых разностей;
- `Diagnostics` — вывод диагностической информации о минимизируемой функции;
- `DiffMaxChange` — максимальные значения изменений переменных при определении первых разностей;
- `DiffMinChange` — минимальные значения изменений переменных при определении первых разностей;
- `Display` — уровень отображения: `'off'` — вывод информации отсутствует, `'iter'` — вывод информации о поиске решения на каждой итерации, `'final'` — вывод только итоговой информации;
- `GoalExactAchieve` — определяет количество целей, которые должны быть достигнуты «точно»;
- `GradConstr` — использование градиентов для ограничений (опция имеет смысл в случае применения аргумента `nonlcon`, см. выше), возможные значения — `'off'` и `'on'`;
- `GradObj` — использование градиента для целевой функции, определяемого пользователем (возможные значения — `'off'` и `'on'`);
- `MaxFunEvals` — максимальное число вычислений функции;
- `MaxIter` — максимальное допустимое число итераций;
- `MeritFunction` — устанавливает вид функции оценки качества достижения цели (возможные значения `'multiobj'` или `'singleobj'`);
- `TolCon` — допуск останова вычислений при нарушении ограничений;
- `TolFun` — допуск останова вычислений по величине изменений функции;
- `TolX` — допуск останова вычислений по величине изменений `x`;
- `attainfactor` — коэффициент достижения цели, усредненное значение несоответствий заданным целям, выраженное в долевом (процентном) виде. Если данный коэффициент отрицательный, цели были «перекрыты», если положительный — цели не достигнуты;
- `exitflag` — информация о характере завершения вычислений: если эта величина положительна, то вычисления завершились нахождением решения `x`, если она равна нулю, то останов произошел в результате выполнения предельного числа итераций, если данная величина отрицательна, то решение не найдено;
- `lambda` — множители Лагранжа, соответственно, для различных типов ограничений:
  - `lambda.lower` — для нижней границы `lb`;
  - `lambda.upper` — для верхней границы `ub`;
  - `lambda.ineqlin` — для линейных неравенств;
  - `lambda.eqlin` — для линейных равенств;
  - `lambda.ineqnonlin` — для нелинейных неравенств;
  - `lambda.eqnonlin` — для нелинейных равенств;

- output — информация о результатах оптимизации:
  - output.iterations — число выполненных итераций;
  - output.funcCount — число вычислений функции;
  - output.algorithm — используемый алгоритм.

4.2.6. Скалярная нелинейная минимизация с ограничениями

Для нелинейной скалярной минимизации с ограничениями вида $x_1 < x < x_2$ служит функция fminbnd. Алгоритм базируется на методе золотого сечения и квадратичной (параболической) интерполяции. Запись этой функции следующая:

```
x = fminbnd(fun,x1,x2,options,P1,P2,...)
[x,fval,exitflag,output] = fminbnd(...)
```

Приведем несколько примеров. Пусть точка минимума функции sin(x) в интервале (0, 2π) определяется следующим образом:

```
>> x = fminbnd('sin',0,2*pi)
x = 4.7124
```

Значение функции при этом

```
>> y = sin(x)
y = -1.0000
```

Для минимизации функции $f(x) = (x - 3)^2 - 1$ на интервале (0, 5) сначала необходимо представить данную функцию в виде m-файла:

```
function f = myfun(x)
f = (x-3).^2-1;
```

а затем использовать функцию fminbnd:

```
>> x = fminbnd('myfun',0,5)
x = 3
```

4.2.7. Решение задачи нелинейного программирования

Для решения задачи нелинейного программирования служит функция fmincon. Она обеспечивает поиск минимума скалярной функции многих переменных и записывается в виде:

```
x = fmincon(fun,x0,A,b,Aeq,beq,lb,ub,nonlcon,options,P1,P2, ...)
[x,fval,exitflag,output,lambda,grad,hessian] = fmincon(...)
```

Здесь grad — градиент функции в точке минимума. Есть возможность задания вычисления гессиана **H** (вводом функции options = optimset('Hessian', 'on').

Пусть требуется найти минимум функции $f(x) = -x_1x_2x_3$ при начальном значении $x = [10; 10; 10]$ и при наличии ограничений $0 \le x_1 + 2x_2 + 2x_3 \le 72$. Для решения этой задачи создадим m-файл, определяющий целевую функцию:

```
function f=myfun1(x)
f =-x(1)*x(2)*x(3);
```

Затем запишем ограничения в виде неравенств:

$$-x_1 - 2x_2 - 2x_3 \le 0,$$

$$x_1 + 2x_2 + 2x_3 \le 72,$$

или в матричной форме:

$$A \cdot x \le b,$$

где

$$A = \begin{bmatrix} -1 & -2 & -2 \\ 1 & 2 & 2 \end{bmatrix}, \; b = \begin{bmatrix} 0 \\ 72 \end{bmatrix}.$$

Теперь нахождение решения представляется следующим образом.

```
>> A=[-1 -2 -2; 1 2 2]; b=[0;72];
>> x0 = [10; 10; 10]; % Стартовое значение
>> [x,fval] = fmincon('myfun1',x0,A,b)
x = 24.0000
  12.0000
  12.0000
fval = -3.4560e+003
```

4.2.8. Решение минимаксных задач

Для решения минимаксных задач служит функция `x = fminimax(fun,x0)`. Она возвращает значение `x`, минимизирующее максимальное значение функций из заданного набора `fun` при стартовой точке поиска `x0`. Другие представления и аргументы аналогичны таковым для функции `fgoalattain`.

Пусть требуется решить минимаксную задачу для набора из 5 функций $[f_1(x), f_2(x), f_3(x), f_4(x), f_5(x)]$, где

$$f_1(x) = 2x_1^2 + x_2^2 - 48x_1 - 40x_2 + 304,$$

$$f_2(x) = -x_1^2 - 3x_2^2,$$

$$f_3(x) = x_1 + 3x_2 - 18,$$

$$f_4(x) = -x_1 - x_2,$$

$$f_5(x) = x_1 + x_2 - 8.$$

Для получения решения сначала представим данные функции в виде m-файла

```
function f = myfun(x)
f(1)= 2*x(1)^2+x(2)^2-48*x(1)-40*x(2)+304;
f(2)= -x(1)^2-3*x(2)^2;
f(3)= x(1)+3*x(2)-18;
f(4)= -x(1)-x(2);
f(5)= x(1)+x(2)-8;
```

Затем используем функцию `fminimax`:

```
>> x0 = [0.1; 0.1]; % Стартовое значение
>> [x,fval] = fminimax('myfun2',x0)
x =
   4.0000
   4.0000
fval =
   0.0000   -64.0000   -2.0000   -8.0000   -0.0000
```

4.2.9. Поиск минимума без ограничений симплексным методом

Функция fminsearch позволяет найти минимум функции нескольких переменных без ограничений, то есть решение задачи безусловной оптимизации с использованием симплексного метода:

```
x = fminsearch(fun,x0,options,P1,P2,...)
[x,fval,exitflag,output] = fminsearch(...)
```

Пример. Ниже демонстрируется нахождение минимума функции одного аргумента $f(x) = \sin(x) + 3$:

```
>> f = inline('sin(x)+3'); x = fminsearch(f,2)
x = 4.7124
```

Функция fminunc предназначена для тех же целей, что и предыдущая функция, но, в отличие от последней, имеет большее число представлений. Самые полные из них следующие:

```
x = fminunc(fun,x0,options,P1,P2,...)
[x,fval,exitflag,output,grad,hessian] = fminunc(...)
```

Найдем минимум функции $f(x) = 3x_1^2 + 2x_1x_2 + x_2^2$.

Способ 1. Создадим m-файл

```
function f = myfun3(x)
f = 3*x(1)^2 + 2*x(1)*x(2) + x(2)^2;
```

Затем используем рассматриваемую функцию при начальном значении [1, 1]. Результат возвращается буквально после пары итераций:

```
>> x0 = [1,1];[x,fval] = fminunc('myfun3',x0)
x = 1.0e-008 *
   -0.7512   0.2479
fval = 1.3818e-016
```

Как видно, найденное значение достаточно близко к истинной точке минимума [0, 0].

Способ 2. Проделаем то же самое, но с заданием информации о градиенте целевой функции. Вот соответствующий m-файл:

```
function [f,g] = myfun4(x)
f = 3*x(1)^2 + 2*x(1)*x(2) + x(2)^2;
if nargout > 1
g(1) = 6*x(1)+2*x(2);
g(2) = 2*x(1)+2*x(2);
```

Решение задачи:

```
>> options = optimset('GradObj','on');
>> x0 = [1,1]; [x,fval] = fminunc('myfun4',x0,options)
Optimization terminated successfully:
First-order optimality less than OPTIONS.TolFun, and no
 negative/zero curvature detected
x = 1.0e-015 *
  0.1110 -0.8882
fval = 6.2862e-031
```

Результат оптимизации в комментариях не нуждается.

4.2.10. Полубесконечная минимизация с ограничениями

Функция `fseminf` предназначена для нахождения решения задачи полубесконечной минимизации с ограничениями. Ее полная запись имеет вид:

```
fseminf(fun,x0,ntheta,seminfcon,A,b,Aeq,beq,...lb,ub,
options,P1,P2,...)
[x,fval,exitflag,output,lambda] = fseminf(...)
```

Новые аргументы этой функции следующие: `ntheta` — число полубесконечных ограничений вида $K_i(x, w_i) \le 0$, где переменные w_i — некоторые задаваемые векторы и `seminfcon` — функция (имя функции), возвращающая значения векторов ограничений при заданном `x`.

Пример. Рассмотрим задачу минимизации функции

$$f(x) = (x_1 - 0.5)^2 + (x_2 - 0.5)^2 + (x_3 - 0.5)^2$$

при наличии ограничений

$$K_1(x, w_1) = \sin(w_1 x_1)\cos(w_1 x_2) - \frac{1}{1000}(w_1 - 50)^2 - \sin(w_1 x_3) - x_3 - 1 \le 0,$$

$$K_2(x, w_1) = \sin(w_2 x_2)\cos(w_2 x_1) - \frac{1}{1000}(w_2 - 50)^2 - \sin(w_2 x_3) - x_3 - 1 \le 0,$$

где w_1 и w_2 принадлежат отрезку [1, 100].

Для решения задачи создадим два m-файла, первый (с именем myfun5) — для вычислений целевой функции, второй (с именем mycon) — для вычислений левых частей ограничений:

```
function f = myfun5(x,s)
f = sum((x-0.2).^2);
function [c,ceq,K1,K2,s] = mycon(X,s)
% Initial sampling interval
if isnan(s(1,1)),
   s = [0.2 0; 0.2 0];
end
% Sample set
w1 = 1:s(1,1):100;
w2 = 1:s(2,1):100;
% Semi-infinite constraints
K1 = sin(w1*X(1)).*cos(w1*X(2)) - 1/1000*(w1-50).^2 -...
     sin(w1*X(3))-X(3)-1;
K2 = sin(w2*X(2)).*cos(w2*X(1)) - 1/1000*(w2-50).^2 -...
     sin(w2*X(3))-X(3)-1;
% No finite nonlinear constraints
c = []; ceq=[];
% Plot a graph of semi-infinite constraints
plot(w1,K1,'-',w2,K2,':'),title('Semi-infinite constraints')
drawnow
```

В листинге этой функции сохранены англоязычные комментарии. Здесь уместно напомнить, что использование русскоязычных комментариев в новых реализациях MATLAB не всегда возможно.

Решение представленной задачи иллюстрируется следующим образом:

```
>> x0 = [0.5; 0.2; 0.3];
>> [x,fval] = fseminf(@myfun5,x0,2,@mycon)
Optimization terminated successfully:
 Search direction less than 2*options.TolX and
  maximum constraint violation is less than options.TolCon
Active Constraints:
     7
     19
     21
x =
     0.6535
     0.2821
     0.4013
fval = 0.2530
```

На рис. 4.1 приведен графический вид функций в правых частях полубесконечных ограничений, возвращаемый m-файлом mycon для точки минимума.

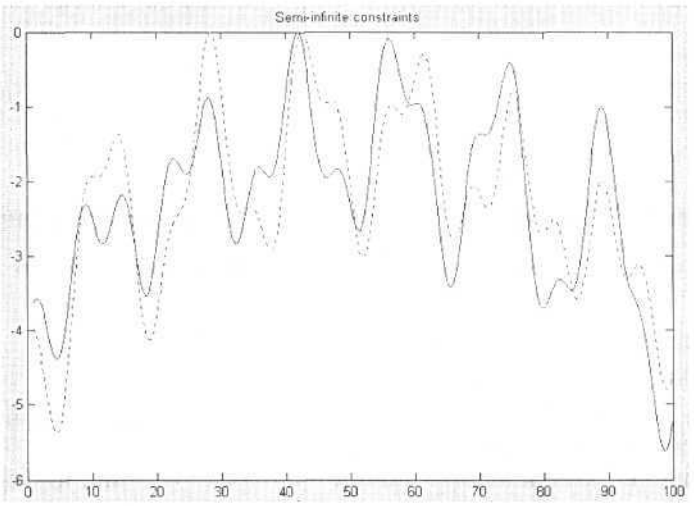

Рис. 4.1. Графический вид функций в правых частях полубесконечных ограничений

Как видно из рисунка, ограничения удовлетворяются.

4.3. Решение задач линейного и квадратичного программирования

4.3.1. Решение задачи линейного программирования

Функция `linprog` обеспечивает решение типовой задачи линейного программирования:

```
x = linprog(f,A,b,Aeq,beq,lb,ub,x0,options)
```

```
[x,fval,exitflag,output,lambda] = linprog(...)
```

В этой функции введен дополнительный аргумент `f` — вектор коэффициентов линейной целевой функции. Функция может использовать алгоритм большой размерности LIPSOL или алгоритм средней размерности (метод проекций).

Пусть требуется найти решение задачи линейного программирования, описывающейся соотношениями

$$f(x) = -5x_1 - 4x_2 - 6x_3,$$

$$x_1 - x_2 + x_3 \le 20$$

$$3x_1 + 2x_2 + 4x_3 \le 42,$$

$$3x_1 + 2x_2 \le 30$$

$$0 \le x_1, 0 \le x_2, 0 \le x_3.$$

Решение задачи приведено ниже.

```
>> f=[-5; -4; -6]; A=[1 -1 1;3 2 4;3 2 0];
>> b=[20; 42; 30]; lb = zeros(3,1);
>> [x,fval,exitflag,output,lambda] = linprog(f,A,b,[],[],lb)
x =
   0.0000
   15.0000
   3.0000
fval = -78.0000
```

4.3.2. Решение задачи квадратичного программирования

Функция `quadprog` возвращает решение задачи квадратичного программирования:

```
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0,options)
[x,fval,exitflag,output,lambda] = quadprog(...)
```

Аргументы (за исключением матрицы `H`) и возвращаемые величины аналогичны рассмотренным для функции `fmincon`.

Найдем решение задачи квадратичного программирования, имеющей описание

$$f(x) = \frac{1}{2}x_1^2 + x_2^2 - x_1x_2 - 2x_1 - 6x_2,$$

$$x_1 + x_2 \le 2,$$

$$-x_1 + 2x_2 \le 2,$$

$$2x_1 + x_2 \le 3,$$

$$0 \le x_1, 0 \le x_2.$$

В данном случае

$$H = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}, f = \begin{bmatrix} -2 \\ -6 \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

и решение задачи представляется в следующем виде:

```
>> H=[1 -1; -1 2]; f=[-2; -6];
>> A=[1 1; -1 2; 2 1]; b=[2; 2; 3]; lb = zeros(2,1);
>> [x,fval,exitflag,output,lambda] = quadprog(H,f,A,b,[],[],lb)
```

Приведем основную часть вывода:

```
x =
   0.6667
   1.3333
fval =
   -8.2222
```

Из опущенной выходной информации следует, что количество выполненных итераций равно 3 и использован алгоритм средней размерности.

4.4. Дополнительные примеры решения типовых оптимизационных задач

4.4.1. Минимизация без ограничений

Рассмотрим задачу нахождения значений переменных x_1 и x_2, обеспечивающих решение задачи минимизации

$$\min_x f(x) = e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 1).$$

Нахождение решения производится в соответствии со следующими этапами.

Составим m-файл с именем objfun, реализующий вычисление значения целевой функции:

```
function f = objfun(x)
f = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
```

Решим задачу с использованием подходящей функции пакета fminunc:

```
>> x0=[-1,1]; options = optimset('LargeScale','off');
>> [x,fval,exitflag,output] = fminunc('objfun',x0,options)
Optimization terminated successfully:
Current search direction is a descent direction, and magnitude of
directional derivative in search direction less than
 2*options.TolFun
x =
    0.5000 -1.0000
fval = 1.3028e-010
exitflag = 1
output =
       iterations: 7
        funcCount: 40
         stepsize: 1
    firstorderopt: 8.1997e-004
        algorithm: 'medium-scale: Quasi-Newton line search'
```

Значения `x = [0.5000 -1.0000]` и `fval = 1.3028e-010` — искомое решение задачи. Значение `exitflag=1` дает информацию, что найдена точка минимума (возможно, локального). Остальные характеристики выхода вполне очевидны.

4.4.2. Минимизация с ограничениями — нелинейными неравенствами

Теперь рассмотрим задачу минимизации той же целевой функции, но при наличии ограничений в форме нелинейных неравенств:

$$x_1x_2 - x_1 - x_2 \le -1.5,$$

$$x_1x_2 \ge -10.$$

Данная задача (см. табл. 4.1) может быть решена с применением функции `fmincon` в соответствии с теми же этапами, что и предыдущая. Поскольку при использовании данной функции нелинейные ограничения должны быть представлены в виде $c(x) \le 0$, перепишем их соответствующим образом:

$$x_1x_2 - x_1 - x_2 + 1.5 \le 0,$$

$$-x_1x_2 - 10 \le 0.$$

Для решения задачи составим m-файл с именем confun, возвращающий значения левых частей ограничивающих неравенств:

```
function [c, ceq] = confun(x)
c = [1.5 + x(1)*x(2) - x(1) - x(2); -x(1)*x(2) - 10];
ceq = [];
```

Для получения решения выполним команды:

```
>> x0=[-1,1]; options = optimset('LargeScale','off');
>> [x,fval]=fmincon('objfun',x0,[],[],[],[],[],[],'confun',
 options)
Optimization terminated successfully:
First-order optimality measure less than options.TolFun and
 maximum constraint violation is less than options.TolCon
x =
    -9.5474    1.0474
fval = 0.0236
```

4.4.3. Минимизация с ограничениями на диапазоны изменения переменных

Функция `fmincon` может быть применена и для поиска решения в задаче минимизации, в которой допустимые значения переменных ограничены некоторыми диапазонами: $lb \le x \le ub$. Продолжим рассмотрение текущего примера, введя дополнительные ограничения $x_1 \ge 0$, $x_2 \ge 0$, что эквивалентно заданию `lb = [0 0]`, `ub = [ ]`.

Программа оптимизации и результаты вычислений для данного случая приведены ниже.

```
>> x0 = [-1,1]; lb = [0, 0]; ub = [ ];
>> options = optimset('LargeScale','off');
```

```
>> [x,fval] = fmincon('objfun',x0,[],[],[],[],lb,ub,
'confun',options)
Optimization terminated successfully:
First-order optimality measure less than options.TolFun and
maximum constraint violation is less than options.TolCon
Active Constraints:
     1
     3
x =
         0       1.5000
fval = 8.5000
```

Найденное решение удовлетворяет ограничениям, наложенным на диапазоны изменения переменных. Проверим теперь выполнение ограничений в форме неравенств:

```
>> [c, ceq] = confun(x)
c =
     0
     -10
ceq =
     []
```

Как следует из приведенного результата, ограничения в форме неравенств выполнены (ограничения в форме равенств отсутствуют).

4.4.4. Использование вектора-градиента, аналитически задаваемого пользователем

По умолчанию функции пакета заменяют точные значения производных целевой функции и ограничений (в требуемых случая) их приближенными оценками в виде первых и/или вторых разностей. Иногда целесообразно задать аналитическое вычисление производных — это может привести к ускорению процесса поиска решения. Рассмотрим такой подход при задании векторов первых производных — векторов-градиентов.

Продолжая решение в условиях приведенного примера, получим следующие аналитические выражения:

- градиент целевой функции:

$$\frac{df(x)}{dx} = \begin{bmatrix} e^{x_1}(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1) + e^{x_1}(8x_1 + 4x_2) \\ e^{x_1}(4x_1 + 4x_2 + 2) \end{bmatrix},$$

- матрица, столбцы которой являются градиентами функций в левых частях ограничений-неравенств:

$$\begin{bmatrix} \frac{\partial c_1}{\partial x_1} & \frac{\partial c_2}{\partial x_1} \\ \frac{\partial c_1}{\partial x_2} & \frac{\partial c_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} x_2 - 1 & -x_2 \\ x_1 - 1 & -x_1 \end{bmatrix}.$$

Используя данные выражения, создадим требуемые m-файлы. Вначале создадим m-файл с именем objfungrad для расчета значений целевой функции и ее градиента:

```
function [f,G] = objfungrad(x)
f = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
t = exp(x(1))*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
G = [ t + exp(x(1)) * (8*x(1) + 4*x(2)),
exp(x(1))*(4*x(1)+4*x(2)+2)];
```

Затем создадим m-файл для нелинейных ограничений и его градиент:

```
function [c,ceq,DC,DCeq] = confungrad(x)
c(1) = 1.5 + x(1) * x(2) - x(1) - x(2);
c(2) = -x(1) * x(2)-10;
DC= [x(2)-1, -x(2); x(1)-1, -x(1)];
ceq=[]; DCeq = [ ];
```

Для решения задачи исполним команды:

```
>> x0 = [-1,1]; options = optimset('LargeScale','off');
>> options = optimset(options,'GradObj','on','GradConstr', 'on');
>> lb = [ ]; ub = [ ];
>> [x,fval] = fmincon('objfungrad',x0,[],[],[],[],lb,ub,
'confungrad',options)
Optimization terminated successfully:
First-order optimality measure less than options.TolFun and
maximum constraint violation is less than options.TolCon
x =
    -9.5474    1.0474
fval = 0.0236
```

Проверка выполнения ограничений:

```
>> c
c =
       0
     -10
```

Как видно, в точке решения заданные ограничения выполняются.

4.4.5. Задача достижения цели

Рассмотрим следующий пример. Пусть некоторая замкнутая линейная динамическая система управления 3-го порядка описывается уравнениями

$$\dot{x} = (\mathbf{A} + \mathbf{BKC})x + \mathbf{B}u,$$

$$y = \mathbf{C}x,$$

где матрицы

$$\mathbf{A} = \begin{bmatrix} -0.5 & 0 & 0 \\ 0 & -2 & 10 \\ 0 & 1 & -2 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 1 & 0 \\ -2 & 2 \\ 0 & 1 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

отражающие динамические свойства объекта управления, являются заданными, а матрица **K** регулятора — изменяемой.

Как известно, качество работы подобных систем (качество переходных процессов) определяется расположением на комплексной плоскости собственных чисел матрицы **A** + **BKC**. Поставим задачу оптимизации таким образом: при задании диапазона возможных изменений элементов матрицы **K** от −4 до +4 подобрать эти элементы таким образом, чтобы указанные собственные числа равнялись величинам [−5, −3, −1]. В подобной формулировке задача является многокритериальной (3 критерия) и соответствует задаче достижения цели, приведенной в табл. 4.1.

С целью решения этой задачи создадим m-файл с именем eigfun для вычисления и упорядочивания по величине собственных чисел матрицы **A** + **BKC**:

```
function F = eigfun(K,A,B,C)
F = sort(eig(A+B*K*C));
```

Решим данную задачу:

```
>> A = [-0.5 0 0; 0 -2 10; 0 1 -2];
>> B = [1 0; -2 2; 0 1]; C = [1 0 0; 0 0 1];
>> K0 = [-1 -1; -1 -1]; goal = [-5,-3,-1]; w = abs(goal);
>> lb = -4*ones(size(K0)); ub = 4*ones(size(K0));
>> options = optimset('Display','iter');
>> [K,fval,attainfactor] = fgoalattain('eigfun',K0, goal,
 w,[],[],[],[],lb,ub,[],options,A,B,C)
Attainment                    Directional
 Iter     F-count     factor     Step-size     derivative
                                               Procedure
     1          13      1.061             1           1.03
     2          20     0.4211             1         -0.679
     3          27   -0.06352             1         -0.523
     4          34    -0.1571             1         -0.053
     5          41    -0.3489             1         -0.133
     6          48    -0.3643             1        -0.00768
     7          55    -0.3645             1       -4.25e-005
     8          62    -0.3674             1        -0.00303
     9          69    -0.3806             1         -0.0213
    10          76    -0.3862             1         0.00266
    11          83    -0.3863             1       -2.73e-005
    12          90    -0.3863             1       -1.21e-013
Optimization terminated successfully:
Search direction less than 2*options.TolX and
maximum constraint violation is less than options.TolCon
Active Constraints:
     1
     2
     4
     9
    10
K =
    -4.0000    -0.2564
```

```
    -4.0000    -4.0000
fval =
    -6.9313
    -4.1588
    -1.4099
attainfactor = -0.3863
```

Полученные результаты не слишком близки к целевым. Поэтому для получения более приемлемого результата внесем коррекцию в программу оптимизации — зададим опцию точного достижения всех трех целей:

```
>> options = optimset('GoalsExactAchieve',3);
```

после чего повторим поиск оптимального решения:

```
>> [K,fval,attainfactor] = fgoalattain
( 'eigfun',K0,goal,w, [],[],[],[],lb,ub,[],options,A,B,C)
Optimization terminated successfully:
Magnitude of directional derivative in search direction
less than 2*options.TolFun and maximum constraint violation
is less than options.TolCon
Active Constraints:
     9
    10
    13
    14
K =
    -1.5954    1.2040
    -0.4201    -2.9046
fval =
    -5.0000
    -3.0000
    -1.0000
attainfactor = 1.6941e-021
```

Очевидно, теперь результат является вполне удовлетворительным. Отметим, что подробную демонстрацию приведенного примера можно получить, используя функцию `goaldemo`.

4.5. Решение минимизационных задач высокой размерности

4.5.1. Решение системы нелинейных уравнений с заданием якобиана

Перейдем к решению задач на основе алгоритмов большой размерности. Рассмотрим задачу нахождения решения системы $F(x) = 0$ из $n = 1000$ уравнений вида

$$F(1) = 3x_1 - 2x_1^2 - 2x_2 + 1,$$

...

$$F(i) = 3x_i - 2x_i^2 - x_{i-1} - 2x_{i+1} + 1,$$

...

$$F(n) = 3x_n - 2x_n^2 - x_{n-1} + 1,$$

используя функцию `fsolve` с применением алгоритма большой размерности (заметим, что якобиан данной системы функций, как нетрудно показать, является разреженной матрицей).

Составим m-файл (под именем nlsf1) для вычисления целевых функций и якобиана:

```
function [F,J] = nlsf1(x);
n = length(x); F = zeros(n,1); i = 2:(n - 1);
F(i) = (3 - 2*x(i)).*x(i) - x(i -1) - 2*x(i+1)+ 1;
F(n) = (3 - 2*x(n)).*x(n) - x(n -1) + 1;
F(1) = (3 - 2*x(1)).*x(1) - 2*x(2) + 1;
d = - 4*x + 3*ones(n,1); D = sparse(1:n,1:n,d,n,n);
c = - 2*ones(n-1,1); C = sparse(1:n -1,2:n,c,n,n);
e = - ones(n-1,1); E = sparse(2:n,1:n -1,e,n,n);
J = C + D + E;
```

Для поиска решения выполним команды:

```
>> xstart = -ones(1000,1); fun = 'nlsf1';
>> options = optimset('Display','iter','Jacobian','on');
>> [x,fval,exitflag,output] = fsolve(fun,xstart,options);
```

Iteration	Func-count	f(x)	Norm of step	First-order optimality	Trust-region radius
0	1	1011		19	1
1	2	774.963	1	10.5	1
2	3	343.695	2.5	4.63	2.5
3	4	2.93752	5.20302	0.429	6.25
4	5	0.000489408	0.590027	0.0081	13
5	6	1.62688e-011	0.00781347	3.01e-006	13
6	7	6.7055e-0[illegible]	1.41828e-006	5.85e-013	13

Для получения всех 1000 элементов решений в командной строке необходимо набрать `x` и нажать клавишу Enter. Заметим, что в данном случае алгоритм большой размерности применяется по умолчанию, в то время как использование пользовательского якобиана задано с помощью функции `optimset`.

4.5.2. Решение системы нелинейных уравнений с представлением оценки якобиана в виде разреженной матрицы

Иногда выражения для элементов якобиана столь сложны, что лучше доверить нахождение их оценок (в виде первых разностей) самой функции минимизации. Для сокращения вычислений в алгоритмах большой размерности в этом случае задается специальная опция — опция представления разреженного образа якобиана.

Данный разреженный образ есть матрица того же размера, что и якобиан, при этом часть элементов — ненулевые, остальные — нулевые. Наличие *ij*-го ненулевого элемента означает, что при выполнении расчетов будет находиться оценка соответствующего элемента якобиана; для нулевых элементов такие расчеты производиться не будут. Указанную матрицу следует заранее подготовить. Будем полагать, продолжая рассмотрение предыдущего примера, что такая матрица под именем `Jstr` подготовлена и сохранена в файле nlsdat1.mat.

Создание m-файла (под именем nlsf1a) для вычисления значений векторной функции:

```
function F = nlsfla(x);
n = length(x); F = zeros(n,1); i = 2:(n -1);
F(i) = (3 - 2*x(i)).*x(i) - x(i -1) - 2*x(i+1) + 1;
F(n) = (3 - 2*x(n)).*x(n) - x(n -1) + 1;
F(1) = (3 - 2*x(1)).*x(1) - 2*x(2) + 1;
```

Для оптимизации исполним команды:

```
>> xstart = -ones(1000,1); fun = 'nlsfla';
>> load nlsdat1
>> options = optimset('Display','iter','JacobPattern',Jstr,
'PrecondBandWidth',1);
>> [x,fval,exitflag,output] = fsolve(fun,xstart,options);
                                    Norm of      First-order     Trust-region
Iteration  Func-count     f(x)          step       optimality      radius
    0          1            1011                      19             1
    1          2         774.963             1         10.5             1
    2          3         343.695           2.5         4.63           2.5
    3          4         2.93752       5.20302        0.429          6.25
    4          5     0.000489408      0.590027       0.0081            13
    5          6    1.62688e-011     0.00781347     3.01e-006            13
    6          7     6.7055e-026     1.41828e-006   5.85e-013            13
Optimization terminated successfully:
First-order optimality is less than options.TolFun.
```

Данные результаты по точности и по времени счета сопоставимы с полученными в предыдущем случае (но здесь якобиан не задавался).

4.5.3. Нелинейный МНК с вычислением оценок всех элементов якобиана

Алгоритмы большой размерности, применяемые в функциях `lsqnonlin`, `lsqcurvefit` и `fsolve`, могут использоваться для решения задач малой и средней размерности вообще без задания якобиана пользователя или разреженного образа якобиана. Такую возможность рассмотрим на примере задачи минимизации суммы 10 функций, зависящих от 2 переменных

$$\sum_{k=1}^{10}(2+2k-e^{kx_1}-e^{kx_2})^2,$$

ранее уже приведенном для иллюстрации функции `lsqnonlin` (см. выше).

Поскольку якобиан не вычисляется в m-файле myfun.m и по умолчанию его использование не задано в опциях, функция `lsqnonlin` автоматически запускает алгоритм большой размерности с заданием разреженного образа якобиана в виде матрицы `Jstr = sparse(ones(10,2))`, то есть матрицы, все элементы которой — единицы. Когда подпрограмма определения оценки якобиана вызывается первый раз, она обнаруживает, что матрица `Jstr` не является разреженной, то есть нет никакой необходимости рассматривать и хранить в памяти данную матрицу как разреженную, и в дальнейшем все вычисления производятся как для обычных матриц.

4.5.4. Минимизация нелинейной функции с использованием градиента и гессиана

Пусть требуется найти минимум функции вида

$$f(x) = \sum_{i=1}^{n-1}\left[\left(x_i^2\right)^{x_{i+1}^2+1} + \left(x_{i+1}^2\right)^{x_i^2+1}\right]$$

при $n = 1000$.

В соответствии с табл. 4.1 данная задача является задачей безусловной оптимизации, и ее целесообразно решать с помощью функции `fminunc`. Для этого составим файл с именем brownfgh, вычисляющий значения целевой функции, ее градиента и разреженной трехдиагональной матрицы Гессе:

```
function [f,g,H] = brownfgh(x)
n=length(x); y=zeros(n,1); i=1:(n-1);
y(i)=(x(i).^2).^(x(i+1).^2+1)+(x(i+1).^2).^(x(i).^2+1); f=sum(y);
i=1:(n-1); g = zeros(n,1);
g(i)= 2*(x(i+1).^2+1).*x(i).*((x(i).^2).^(x(i+1).^2))+2*x(i).*
((x(i+1).^2).^(x(i).^2+1)).*log(x(i+1).^2);
g(i+1)=g(i+1)+2*x(i+1).*((x(i).^2).^(x(i+1).^2+1)).*log(x(i).^2)+2*(x(i)
.^2+1).*
x(i+1).*((x(i+1).^2).^(x(i).^2)); v=zeros(n,1); i=1:(n-1);
v(i)=2*(x(i+1).^2+1).*((x(i).^2).^(x(i+1).^2))+4*(x(i+1).^2+1).*(x(i+1).
^2).*(x(i).^2)
.*((x(i).^2).^((x(i+1).^2)-1))+2*((x(i+1).^2).^(x(i).^2+1)).*
(log(x(i+1).^2));
v(i)=v(i)+4*(x(i).^2).*((x(i+1).^2).^(x(i).^2+1)).*((log(x(i+1).^2)).^2);
v(i+1)=v(i+1)+ 2*(x(i).^2).^(x(i+1).^2+1).*(log(x(i).^2))+4*
(x(i+1).^2).*((x(i).^2).^
(x(i+1).^2+1)).*((log(x(i).^2)).^2)+2*(x(i).^2+1).*((x(i+1).^2).^
(x(i).^2));
(i+1)=v(i+1)+4*(x(i).^2+1).*(x(i+1).^2).*(x(i).^2).*((x(i+1).^2).^
(x(i).^2-1));
v0=v; v=zeros(n-1,1);
v(i)=4*x(i+1).*x(i).*((x(i).^2).^(x(i+1).^2))+4*x(i+1).*
x(i+1).^2+1).*x(i).*
((x(i).^2).^(x(i+1).^2)).*log(x(i).^2);
v(i)=v(i)+ 4*x(i+1).*x(i).*((x(i+1).^2).^(x(i).^2)).*log(x(i+1).^2);
v(i)=v(i)+4*x(i).*((x(i+1).^2).^(x(i).^2)).*x(i+1);
vl=v; i=[(1:n)';(1:(n-1))']; j=[(1:n)';(2:n)'];
s=[v0;2*vl]; H=sparse(i,j,s,n,n); H=(H+H')/2;
```

Выполним решение:

```
>> n=1000; xstart = -ones(n,1); xstart(2:2:n,1) = 1;
>> options = optimset('GradObj','on','Hessian','on');
>> [x,fval,exitflag,output] =
 fminunc('brownfgh',xstart,options);
Optimization terminated successfully:
First-order optimality less than OPTIONS.TolFun, and no
 negative/zero curvature
>> exitflag
exitflag =
       1
```

```
>> fval
fval =
  2.8709e-017
>> output.iterations
ans =
     8
>> output.cgiterations
ans =
     7
>> output.firstorderopt
ans =
  4.7948e-010
```

4.5.5. Нелинейная минимизация с ограничениями в виде линейных равенств

Продолжим рассмотрение задачи с минимизацией введенной функции, добавив в нее ограничения в виде 100 линейных равенств

$$\mathbf{Aeq} \cdot x = \mathbf{beq},$$

то есть матрица **Aeq** имеет размер 100 × 1000.

Будем полагать, что матрица **Aeq** и вектор **beq** содержатся в файле browneq.mat.

В данном случае будем использовать ранее подготовленный m-файл brownfgh. Тогда решение задается командами

```
>> fun = 'brownfgh'; load browneq; n = 1000;
>> xstart = -ones(n,1); xstart(2:2:n) = 1;
>> options = optimset('GradObj','on','Hessian','on', ...
'PrecondBandWidth', inf);
>> [x,fval,exitflag,output] = fmincon('brownfgh',xstart,
[],[], Aeq,beq, [],[],[],options);
```

Результаты расчета представляются удовлетворительными (об этом можно судить, например, по величине нормы невязки `norm(Aeq*x-beq)`):

```
>> exitflag
exitflag =
  1
>> fval
fval =
  205.9313
>> output
output =
  iterations: 16
  funcCount: 16
  cgiterations: 14
  firstorderopt: 2.1434e-004
  algorithm: 'large-scale: projected trust-region Newton'
  The linear equalities are satisfied at x
  norm(Aeq*x-beq)
ans =
    1.1922e-012
```

4.5.6. Квадратичное программирование с ограничениями на диапазоны изменений переменных

Для минимизации квадратичной формы, зависящей от большого числа переменных, следует использовать функцию quadprog. Рассмотрим подобную задачу при числе переменных 400 и симметричной трехдиагональной матрице **H**, сохраненной в файле qpbox1.mat.

Соответствующая программа и результаты вычислений приведены ниже.

```
>> load qpbox1
>> lb = zeros(400,1); lb(400) = -inf;
>> ub = 0.9*ones(400,1); ub(400) = inf;
>> f = zeros(400,1); f([1 400]) = -2;
>> xstart = 0.5*ones(400,1);
>> [x,fval,exitflag,output] =
quadprog(H,f,[],[],[],[],lb,ub,xstart);
>> exitflag
exitflag =
  1
>> output
output =
firstorderopt: 9.8393e-006
       iterations: 19
     cgiterations: 1670
        algorithm: 'large-scale: reflective trust-region
```

4.5.7. Решение задачи линейного программирования

Рассмотрим задачу линейного программирования при исходных данных, имеющихся в тестовом файле sc50b.mat. Задача включает 48 переменных, 30 неравенств и 20 равенств, так что вполне может быть отнесена к задачам большой размерности. Для решения задачи с использованием функции linprog исполним команды:

```
>> load sc50b
>> [x,fval,exitflag,output] = ...
linprog(f,A,b,Aeq,beq,lb,[],[],optimset('Display','iter'));
Residuals:     Primal       Dual       Duality       Total
               Infeas       Infeas     Gap           Rel
               A*x-b        A'*y+z-f   x'*z          Error
-------------------------------------------------------------
Iter    0:   1.50e+003     2.19e+001   1.91e+004   1.00e+002
Iter    1:   1.15e+002     2.94e-015   3.62e+003   9.90e-001
Iter    2:   1.16e-012     2.21e-015   4.32e+002   9.48e-001
Iter    3:   3.24e-012     1.08e-015   7.78e+001   6.88e-001
Iter    4:   5.69e-011     2.68e-016   2.38e+001   2.69e-001
Iter    5:   1.11e-010     3.17e-016   5.05e+000   6.89e-002
Iter    6:   1.06e-010     2.15e-016   1.64e-001   2.34e-003
Iter    7:   4.82e-012     2.13e-016   1.09e-005   1.55e-007
Iter    8:   7.90e-014     5.67e-013   1.09e-011   3.82e-012
Optimization terminated successfully.
>> exitflag
```

```
exitflag =
     1
>> fval
fval =
  -70.0000
>> output
output =
      iterations: 8
    cgiterations: 0
       algorithm: 'lipsol'
```

4.6. Демонстрационные примеры на оптимизацию

4.6.1. Выход к демонстрационным примерам

Для дополнительного знакомства с техникой оптимизации можно просмотреть демонстрационные примеры на нее, доступ к которым открывает пункт меню Demos. Он открывает соответствующую вкладку окна справки в MATLAB. Нужные примеры имеются в разделе Toolbox Optimization (рис. 4.2).

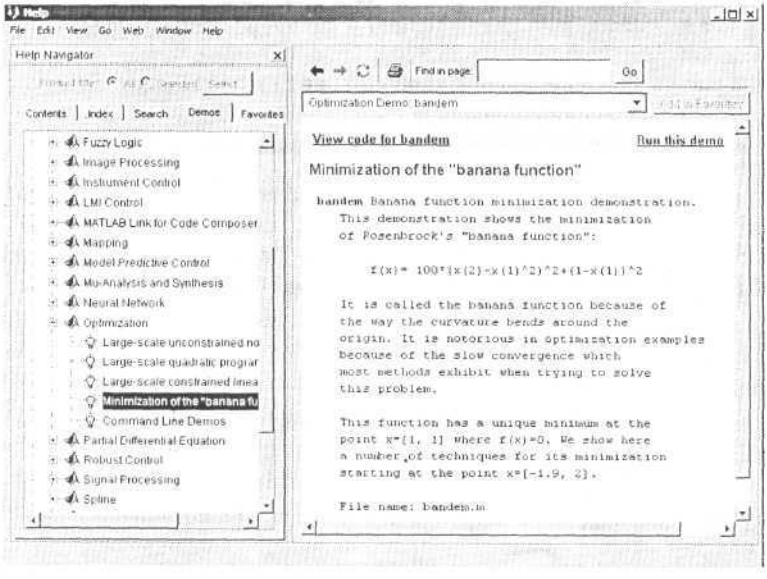

Рис. 4.2. Доступ к демонстрационным примерам пакета Toolbox Optimization

Поскольку каждый пользователь системой MATLAB может легко ознакомиться с этими примерами, ограничимся кратким описанием их с приведением лишь наиболее характерных кадров.

4.6.2. Минимизация тестовой функции Розенброка

Рис. 4.3 показывает наглядное решение классической оптимизационной задачи — нахождение минимума тестовой функции Розенброка, именуемой также «банановой» функцией (файл bandem.m).

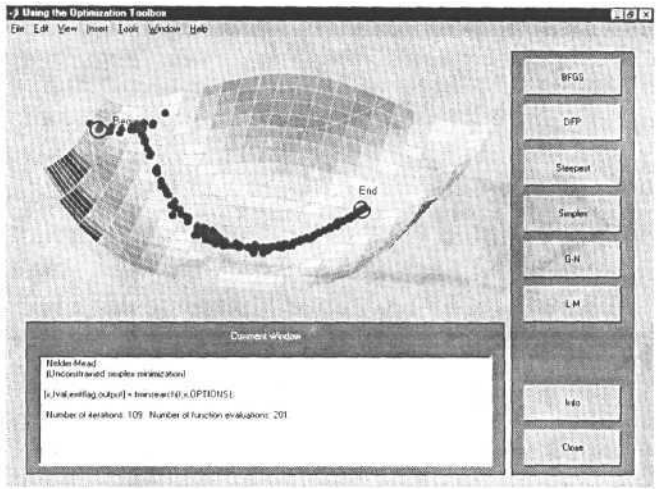

Рис. 4.3. Оптимизация тестовой функции Розенброка

Эта функция имеет вид

$$f(x) = 100*(x(2) - x(1)^2)^2 + (1 - x(1))^2$$

и является типичной тестовой функцией двух переменных x(1) и x(2). Профиль дна этой функции, в которой расположена точка минимума x = [1 1] имеет вид пологого оврага, чем затрудняет поиск минимума. Приведенный пример интересен тем, что он демонстрирует в динамике переход из начальной точки поиска минимума Begin с x = [−1,9 2], в конечную End с показом каждой промежуточной точки на каждой итерации. Случай, представленный на рис. 4.3, соответствует поиску минимума симплекс-методом. Это достаточно надежный, но далеко не самый быстрый метод поиска минимума данной функции. Как видно из окна этого примера, можно опробовать минимизацию любым из 7 методов, что позволяет наглядно сравнить их эффективность.

4.6.3. Минимизация упругой поверхности

Еще один из показательных примеров — построение поверхности, которая образуется, когда упругая тонкая пленка набрасывается на вертикально стоящие 5 столбиков (по углам и в центре квадрата). Рис. 4.4 показывает первый кадр этой демонстрации, на котором вид пленки и указана точка экстремума, находящаяся над центральным столбиком. Это типичная задача квадратичного программирования (файл circustent.m).

А на рис. 4.5 дан один из промежуточных кадров этой демонстрации. На нем видно, как пленка, разбитая на многоугольные ячейки, — сетка набрасывается на столбики и как она выглядит после оптимизации и построения поверхности.

Вид конечного кадра демонстрации представлен на рис. 4.6. Хотя хорошо видно неплохое качество моделирования прогиба поверхности в середине между столбцами, на них самих поверхность выглядит не столь естественно — не заметна выпуклость поверхности. Нетрудно также понять, что это связано с большим размером ячеек, на которые разбивается поверхность. Таким образом, в данном примере явно видна необходимость решения задачи оптимизации большой размерности.

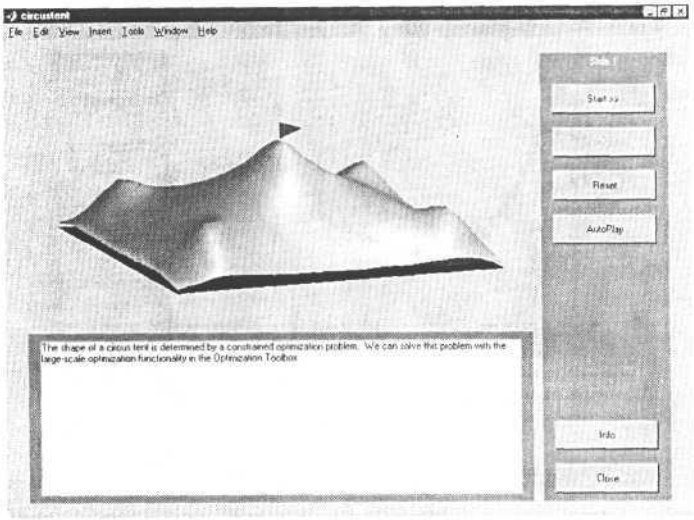

Рис. 4.4. Оптимизируемая поверхность и точка ее экстремума

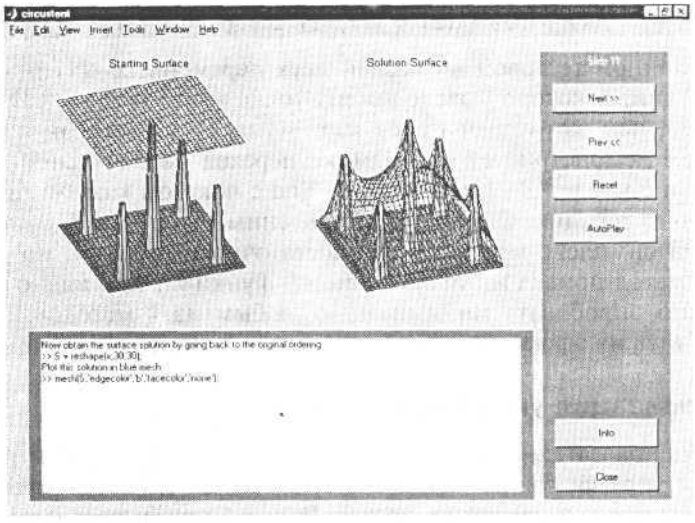

Рис. 4.5. Один из кадров демонстрации моделирования процесса наложения упругой поверхности на столбики

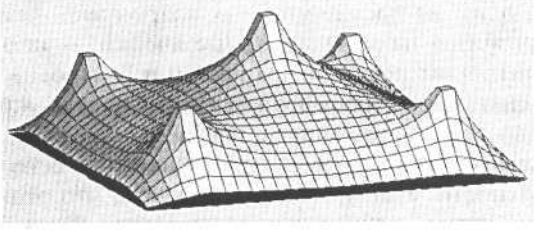

Рис. 4.6. Результат моделирования процесса укладки упругой пленки на столбики

4.6.4. Минимизация размера изображения

Еще одна классическая задача оптимизации большой размерности при наличии ограничений — уменьшение размера изображения. На рис. 4.7 показан начальный кадр с изображением автомобиля типа graycsale (полутоновое изображение). Такое изображение можно представить как множество ячеек с яркостью от 0 до 1. Цель этого примера (файл optdeblur.m) — показать возможность представления изображения в виде разреженной матрицы, дающей размытое изображение автомобиля, максимально верно приближающее исходное изображение.

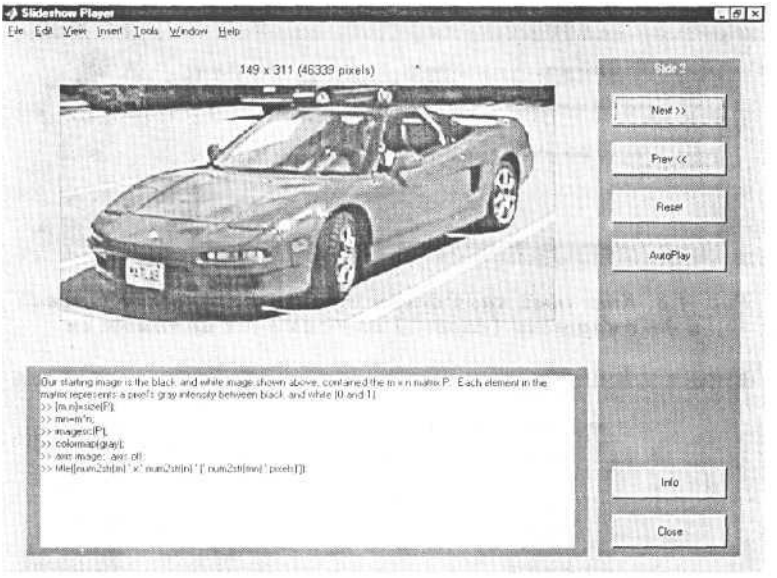

Рис. 4.7. Начальный кадр демонстрации компрессии изображения автомобиля

По мере выполнения этого примера можно наблюдать технику матричного описания и преобразования изображения, а затем и его восстановления — см. конечный кадр демонстрации, показанный на рис. 4.8. Слева показано размытое изображение автомобиля, полученное с применением разреженных матриц, а справа — восстановленное изображение. Нетрудно заметить, что техника восстановления дает существенное улучшение четкости размытого изображения — операция DeBlurred.

4.6.5. Проблема конструирования молекулы

Интересной задачей крупноразмерной оптимизации без ограничений является конструирование молекулы, атомы которой надо разместить таким образом, чтобы расстояние между смежными атомами было равно экспериментально заданным результатам. Суть задачи на примере малой молекулы поясняет рис. 4.9.

Рис. 4.10 показывает начало создания молекулы (инициализация) и завершение этого процесса — представление молекулы. Промежуточные кадры заинтересованный пользователь может просмотреть самостоятельно.

Как видно из этих демонстрационных примеров, GUI дает наглядное представление об этапах решаемых задач. Кадры демонстрации можно выполнять как

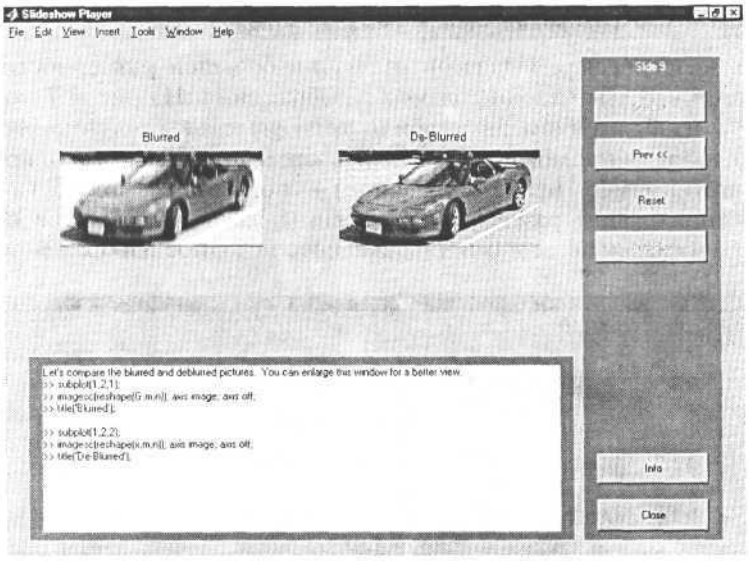

Рис. 4.8. Конечный кадр демонстрации компрессии (слева) и декомпрессии (справа) изображения автомобиля

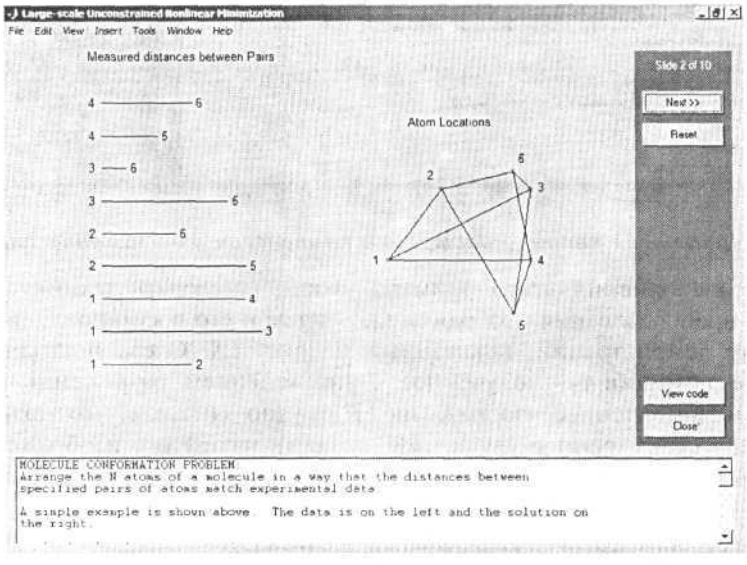

Рис. 4.9. Второй кадр задачи на решение проблемы конструирования молекулы

поодиночке, так и в автоматическом режиме — кнопка AutoPlay. В специальных окнах под результатами можно наблюдать фрагменты m-файлов, описывающих демонстрационные примеры. Наконец, в окнах Demos, активизируя гиперссылку View code..., можно просмотреть коды программы на языке MATLAB, решающей соответствующую задачу.

Несмотря на высокую степень визуализации демонстрационных примеров они все же менее полезны, чем многочисленные примеры решения оптимизационных задач в командном режиме. Хотя бы потому, что эти примеры не отягощены дета-

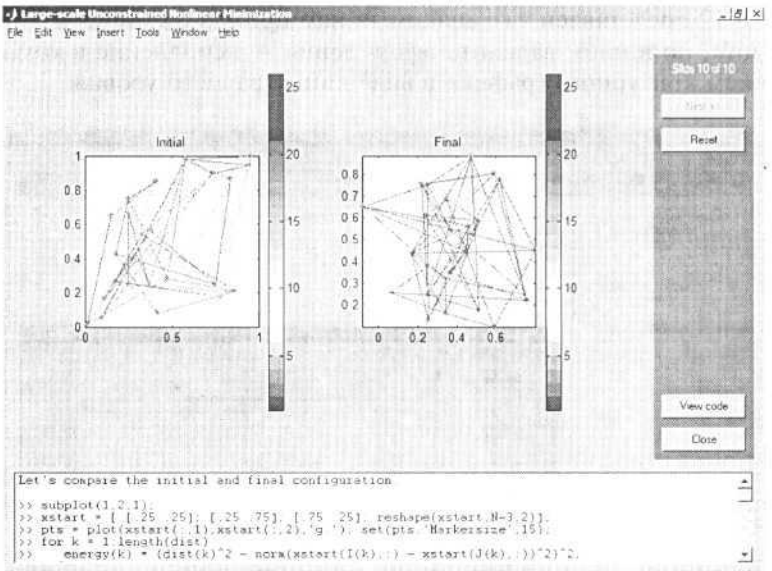

Рис. 4.10. Завершающий кадр задачи на решение проблемы конструирования молекулы

лями GUI и могут легко модифицироваться пользователем. Кстати говоря, такая модификация для демонстрационных примеров не предусмотрена.

4.6.6. Примеры, выполняемые в командном окне

Один из демонстрационных примеров Command Line Demos выводит скромное окно с наборами примеров на оптимизацию, выполняемых (и наблюдаемых) в командном окне MATLAB. Это окно представлено на рис. 4.11.

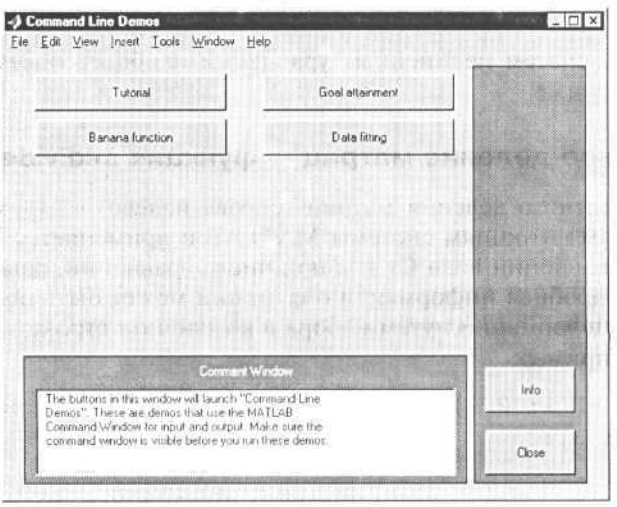

Рис. 4.11. Окно демонстрационных примеров, выполняемых в командном окне MATLAB

На рис. 4.12 представлен уже знакомый нам пример на минимизацию «банановой» функции, но в ином варианте оформления — исполнение в командном окне с построением контурного графика в виде линий равного уровня.

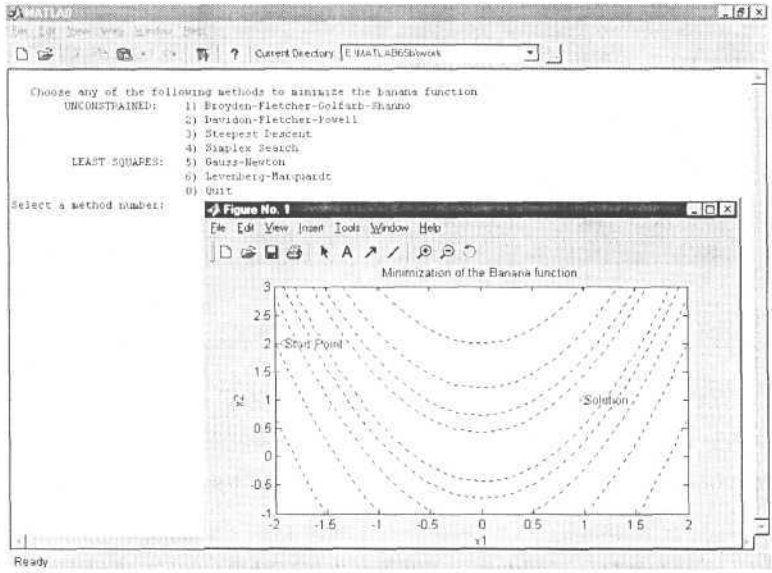

Рис. 4.12. Окно примера на минимизацию «банановой» функции, выполняемого в командном окне MATLAB

Полезно просмотреть и другие примеры, поскольку они дают дополнительное (к уже описанному) представление о технике оптимизации.

4.7. Функции решения систем уравнений

Для решения систем нелинейных уравнений наиболее полезны три функции, представленные ниже.

4.7.1. Обратное деление матриц — функция mldivide

Функция обратного деления матриц (справа налево) `mldivide` (или `\`) относится к основным функциям системы MATLAB и применяется при решении систем линейных уравнений вида $Cx = d$ при числе уравнений, равном числу неизвестных. Более подробная информация о функции может быть получена при запросе справочной информации путем набора в командной строке `help mldivide`.

Рассмотрим пример:

```
>> C=[1 1; -1 2; 2 1];
>> d=[2; 2; 3];
>> x=C\d
x =
    0.7714
    1.3714
```

4.7.2. Решение систем нелинейных уравнений

Функция `fsolve` возвращает решение системы нелинейных уравнений $F(x) = 0$:

```
x = fsolve(fun,x0,options,P1,P2, ... )
[x,fval,exitflag,output,jacobian] = fsolve(...)
```

Здесь `jacobian` — якобиан вектора целевых функций в точке найденного решения (напомним, что в терминах рассматриваемого пакета якобиан — это транспонированная матрица, составленная из столбцов-градиентов частных целевых функций).

Найдем решение системы нелинейных уравнений

$$2x_1 - x_2 - e^{-x_1} = 0,$$
$$-x_1 + 2x_2 - e^{-x_2} = 0$$

при начальном приближении (стартовом значении) `x0 = [-5 -5]`.

Составим вначале соответствующий m-файл с именем myfun для вычисления значений функций в левых частях уравнений:

```
function F = myfun6(x)
F = [2*x(1)- x(2)- exp(-x(1));-x(1) + 2*x(2)- exp(-x(2))];
```

Затем организуем процесс вычислений:

```
>> x0=[-5; -5]; options=optimset('Display','iter');
>> [x,fval] = fsolve('myfun6',x0,options)
Norm of      First-order     Trust-region
Iteration  Func-count  f(x)         step          optimality
radius
    0         3         47071.2    2.29e+004 1
    1         6         12003.4      1         5.75e+003          1
    2         9          3147.02     1          1.47e+003         1
    3        12         854.452      1                388         1
    4        15         239.527      1                107         1
    5        18         67.0412      1               30.8         1
    6        21         16.7042      1               9.05         1
    7        24          2.42788     1               2.26         1
    8        27        0.032658     0.759511         0.206      2.5
    9        30     .03149e-006     0.111927        0.00294     2.5
   10        33    3.29525e-013     0.00169132      6.36e-007 2.5
Optimization terminated successfully:
First-order optimality is less than options.TolFun.
x =
     0.5671
     0.5671
fval = 1.0e-006 *
    -0.4059
    -0.4059
x =
     0.5671
     0.5671
fval = 1.0e-008 *
    -0.5320
```

```
    -0.5320
```

Итак, процесс поиска решения прошел нормально.

А теперь найдем матрицу x, удовлетворяющую уравнению

$$x \cdot x \cdot x = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

со стартовым значением x0 = [1, 1; 1, 1].

Для нахождения решения, как и раньше, составим m-файл (с именем myfun) для вычисления значений функции:

```
function F = myfun7(x)
F = x*x*x-[1,2;3,4];
```

Решение задается командами:

```
>> x0 = ones(2,2); options = optimset('Display','off');
>> [x,Fval,exitflag] = fsolve('myfun7',x0,options)
x =
   -0.1291    0.8602
    1.2903    1.1612
Fval = 1.0e-009 *
   -0.1619    0.0776
    0.1161   -0.0469
exitflag =
     1
```

Судя по итоговому сообщению (exitflag=1), решение достигнуто и определяется выведенными матрицами x и Fval (эта матрица в данном случае характеризует ошибку решения).

4.7.3. Поиск корней функции одной переменной

Функция fzero предназначена для поиска корней (нулей) функции одной переменной, то есть для решения задачи

$$a: f(a) = 0.$$

Функция реализует одноименный метод, объединяющий методы бисекции и секущих, предложенный ван Вайнгаарденом, Деккером и другими сотрудниками Математического центра в Амстердаме. Запись функции:

```
x = fzero(fun,x0,options,P1,P2,...)
[x,fval,exitflag,output] = fzero(...)
```

Найдем ноль функции $f(x) = x^3 - 2x - 5$ вблизи точки x0 = 2:

```
>> z = fzero('x.^3-2*x-5',2)
z =
    2.0946
```

Более полный результат можно получить, применяя функцию MATLAB roots

```
>> roots([1 0 -2 -5])
ans =
    2.0946
   -1.0473 + 1.1359i
   -1.0473 - 1.1359i
```

Глава 5. Регрессия и статистика

5.1. Регрессия в пакете Optimization Toolbox

Система MATLAB может выполнять любые статистические расчеты. Однако для их проведения в настоящее время создан целый ряд специальных статистических программ, например Statistica, SPSS, S-Plus и др., которые имеют более удобный интерфейс и средства для таких расчетов. Поэтому реализация статистических расчетов в системе MATLAB ориентирована прежде всего на научно-технические сферы применения.

Задача подбора (подгонки) кривых, наилучшим образом приближающих облако точек некоторых исходных данных, является одной из наиболее распространенных задач статистической обработки данных и проведения их математического анализа [53, 54]. Часто эта задача решается на основе применения метода наименьших квадратов (МНК), при котором критерием наилучшего приближения является минимальная среднеквадратическая погрешность для всех точек исходных данных. Более коротко этот метод называют регрессией [38, 39]. Для ее применения в пакете Optimization Toolbox есть ряд функций, описанных ниже.

5.1.1. Решение задачи линейного МНК без ограничений

Функция обратного деления матриц (справа налево) `mldivide` (или `\`) используется для реализации линейного МНК в следующей постановке. Имеется переопределенная линейная система из m уравнений с n переменными ($m > n$)

$$C \cdot x = d.$$

Требуется найти вектор решения x. Такое решение, получаемое с использованием линейного МНК, определяется формулой

$$x = (\mathbf{C}^{\mathrm{T}}\mathbf{C})^{-1}\mathbf{C}^{\mathrm{T}}d,$$

которая в системе MATLAB реализуется операцией `x=C\d`.

Пример:

```
>> C=[1 2;2 3;3 4]; d=[1; 2; 3]; x=C\d
x =
1
0
```

5.1.2. Решение задачи линейного МНК при наличии ограничений

Функция `lsqlin` возвращает решение задачи линейного МНК при наличии ограничений:

```
x = lsqlin(C,d,A,b,Aeq,beq,lb,ub,x0,options)
[x,resnorm,residual,exitflag,output,lambda] = lsqlin(...)
```

Аргументы и опции соответствуют ранее рассмотренным. Возвращаемые величины:

`x` — найденное решение,

`resnorm` — сумма квадратов невязок $\|C \cdot x - d\|_2^2$ (для найденного значения x),
`residual` — вектор невязок $C \cdot x - d$ (также для конечного значения x),
`exitflag`, `output`, `lambda` — см. выше.

Найдем решение переопределенной системы линейных уравнений

$$C \cdot x = d$$

при наличии ограничений

$$A \cdot x \le b,\ lb \le x \le ub.$$

Пример решения приведен ниже:

```
>> C = [
        0.9501    0.7620    0.6153    0.4057
        0.2311    0.4564    0.7919    0.9354
        0.6068    0.0185    0.9218    0.9169
        0.4859    0.8214    0.7382    0.4102
        0.8912    0.4447    0.1762    0.8936];
>> d = [
0.0578
0.3528
0.8131
0.0098
0.1388];
>> A =[
       0.2027    0.2721    0.7467    0.4659
       0.1987    0.1988    0.4450    0.4186
       0.6037    0.0152    0.9318    0.8462];
>> b =[
0.5251
0.2026
0.6721];
>> lb = -0.1*ones(4,1); ub = 2*ones(4,1);
>> [x,resnorm,residual,exitflag,output,lambda] = ...
lsqlin(C,d,A,b,[ ],[ ],lb,ub);
Warning: Large-scale method can handle bound constraints only;
switching to medium-scale method.
>> In E:\MATLAB65B\toolbox\optim\lsqlin.m at line 192
Optimization terminated successfully.
ans =
    0.1000
    0.1000    0.2152    0.3502
```

Судя по сообщению `Optimization terminated successfully` (Оптимизация успешно закончена), возвращенное решение является «правильным».

5.1.3. Регрессия для заданной функции

Функция `lsqcurvefit` для экспериментальных данных `xdata`, `ydata` возвращает параметры функции заданного вида, обеспечивающие ее наименьшее среднеквадратическое отклонение от указанных данных:

```
x = lsqcurvefit(fun,x0,xdata,ydata,lb,ub,options,P1,P2,...)
[x,resnorm,residual,exitflag,output,lambda,jacobian] =
lsqcurvefit(...)
```

Пусть нужно приближение следующей нелинейной функцией:

```
function F = myfun8(x,xdata)
F = x(1)*xdata.^2 + x(2)*sin(xdata) + x(3)*xdata.^3;
```

Для точек, представленых данными:

```
>> xdata = [3.6 7.7 9.3 4.1 8.6 2.8 1.3 7.9 10.0 5.4];
>> ydata = [16.5 150.6 263.1 24.7 208.5 9.9 2.7 163.9 325.0
54.3];
```

Зададим вектор начальных значений переменных и команды для решения данной задачи:

```
>> x0 = [10, 10, 10];
>> [x,resnorm] = lsqcurvefit('myfun8',x0,xdata,ydata)
Optimization terminated successfully:
Relative function value changing by less than OPTIONS.TolFun
x =
0.2269 0.3385 0.3021
resnorm =
6.2950
```

5.1.4. Решение задачи нелинейного МНК при средней и большой размерности

Функция `lsqnonlin` возвращает решение задачи нелинейного МНК:

$$\min_x f(x) = f_1^2(x) + f_2^2(x) + \ldots + f_m^2(x) + const = \min_x \frac{1}{2}\|F(x)\|_2^2 = \frac{1}{2}\sum_i f_i^2(x).$$

Использует алгоритмы средней или большой размерности. Запись функции:

```
x = lsqnonlin(fun,x0,options,P1,P2, ... )
[x,resnorm,residual,exitflag,output,lambda,jacobian] =
lsqnonlin(...)
```

Для примера найдем вектор x, минимизирующий функцию

$$\sum_{k=1}^{10}(2 + 2k - e^{kx_1} - e^{kx_2})^2$$

при начальном значении `x0 = [0.3, 0.4]`.

Для решения задачи вначале составим соответствующий m-файл:

```
function F = myfun9(x)
k = 1:10;
F = 2 + 2*k-exp(k*x(1))-exp(k*x(2));
```

Для решения задачи исполним команды:

```
>> x0 = [0.3 0.4];[x,resnorm] = lsqnonlin('myfun9',x0)
Optimization terminated successfully:
Norm of the current step is less than OPTIONS.TolX
```

```
x =
0.2578 0.2578
resnorm =
124.3622
```

В данном примере итерационный процесс успешно закончился нахождением решения x = [0.2578 0.2578] при сумме квадратов невязок, равной 124.3622.

5.1.5. Решение задачи неотрицательного линейного МНК

Функция lsqnonneg возвращает решение так называемой задачи неотрицательного линейного МНК:

```
x = lsqnonneg(C,d,x0,options)
[x,resnorm,residual,exitflag,output,lambda] = lsqnonneg(...)
```

Пример:

```
>> C=[
        0.0372    0.2869
        0.6861    0.7071
        0.6233    0.6245
        0.6344    0.6170];
>> d=[
0.8587
0.1781
0.0747
0.8405];
>> [C\d, lsqnonneg(C,d)]
ans =
    2.5627         0
3.1108    0.6929
>> [norm(C*(C\d)-d), norm(C*lsqnonneg(C,d)-d)]
ans =
0.6674    0.9118
```

Видно, что линейный МНК обеспечил результат с меньшими невязками, зато применение функции lsqnonneg позволило получить (наилучшее) неотрицательное решение x = [3.1108 0.6929].

5.2. Пакет подгонки кривых Curve Fitting Toolbox

5.2.1. Назначение пакета Curve Fitting Toolbox

В новых реализациях MATLAB существенно усилено внимание к технике подгонки кривых с использованием МНК. К средствам, присущим собственно системе MATLAB, была добавлена возможность реализации регрессии, а также многоинтервальной сплайновой и эрмитовой аппроксимаций прямо в окне двумерной графики. В продолжение этой тенденции в MATLAB 6.5 был введен новый пакет подгонки кривых Curve Fitting Toolbox. Он обеспечивает следующие возможности:

- импорт данных;
- выполнение подгонки кривых — регрессии;

- анализ (в том числе графический) результатов подгонки;
- запись сессии и результатов подгонки.

5.2.2. Приближение для численности населения США

Для знакомства с возможностями нового пакета можно исполнить в командной строке команду `census`. Она запускает файл, который является законченным GUI-приложением, позволяющим реализовать различные методы приближения данных, представляющих зависимость численности населения США в период с 1790 по 1990 гг. с интервалом в 10 лет. Окно GUI этого приложения представлено на рис. 5.1.

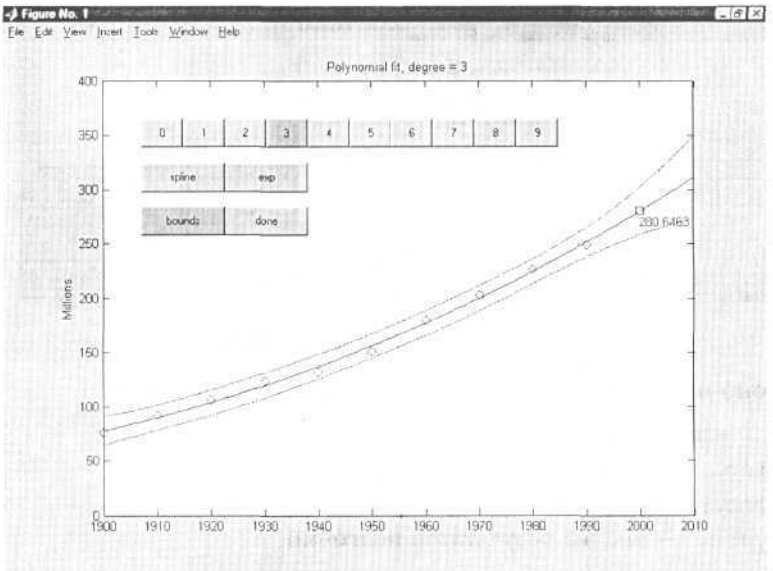

Рис. 5.1. Окно примера census

Как нетрудно заметить, приложение `census` позволяет строить точки исходных данных и графики различных приближений — полиномом степени от 0 до 9 (выбор верхним рядом кнопок), сплайнами и с экспоненциальной функцией с возможностью построения кривых верхней и нижней оценок. Кнопка `Done` завершает работу с этим приложением.

Командой `type census` можно вывести листинг m-файла данного приложения. Поскольку он довольно большой, ограничимся только указанием той части листинга, в которой заданы исходные данные в виде векторов-столбцов, получаемых из векторов-строк (операция '):

```
t = (1900:10:1990)';
p = [75.995 91.972 105.711 123.203 131.669 150.697 179.323 203.212 226.505 249.633]'
```

5.2.3. Начало работы с пакетом Curve Fitting

Работу с пакетом надо начинать с загрузки данных. На первых порах для этого можно использовать команду `load census`, которая загружает данные из файла `census` без его запуска как самостоятельного приложения. После этого командой

cftool можно открыть основное окно Curve Fitting Tool пакета. Это окно показано на рис. 5.2.

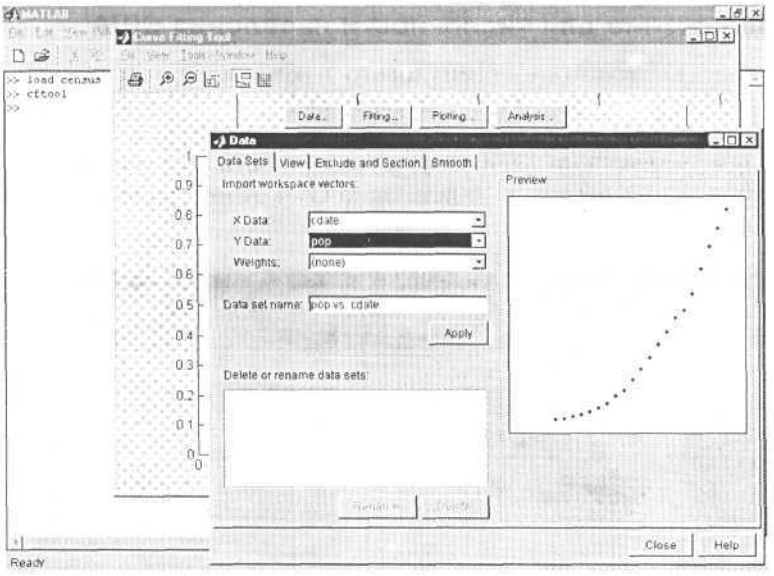

Рис. 5.2. Окно примера census

Вначале оно имеет пустое окно графики и 4 кнопки над ним:

- Data... – загрузка данных;
- Fitting... — выполнение подгонки;
- Ploting... — графическая визуализация подгонки;
- Analysis... – анализ результатов подгонки.

5.2.4. Оценка данных

Обычно работа с окном Curve Fitting Tool начинается с уточнения типа загружаемых данных, для чего необходимо активизировать кнопку Data.... Это ведет к появлению окна импорта данных Data. Это окно показано на рис. 5.3 поверх окна Curve Fitting Tool. В этом окне с помощью соответствующих вкладок можно установить тип множеств данных, просмотреть их, скорректировать и сгладить. Нажатие кнопки Apply и закрытие окна приводит к загрузке скорректированных данных в основное окно Curve Fitting Tool. В результате точки данных появляются в поле графики этого окна (рис. 5.3).

5.2.5. Выполнение регрессии заданного типа

Следующий шаг заключается в установке типа и параметров подгонки (регрессия). Окно редактора подгонки Fitting показано на рис. 5.4.

В этом окне имеются следующие средства GUI:

- New fit — выполнение нового сеанса подгонки;
- Copy fit — копирование сеанса подгонки;
- Fit name — задание имени подгонки;

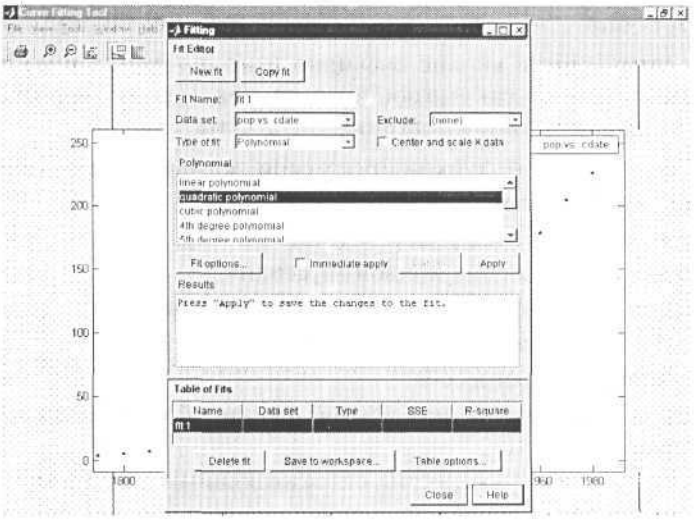

Рис. 5.3. Окно пакета Curve Fitting Tool и окно работы с данными Data

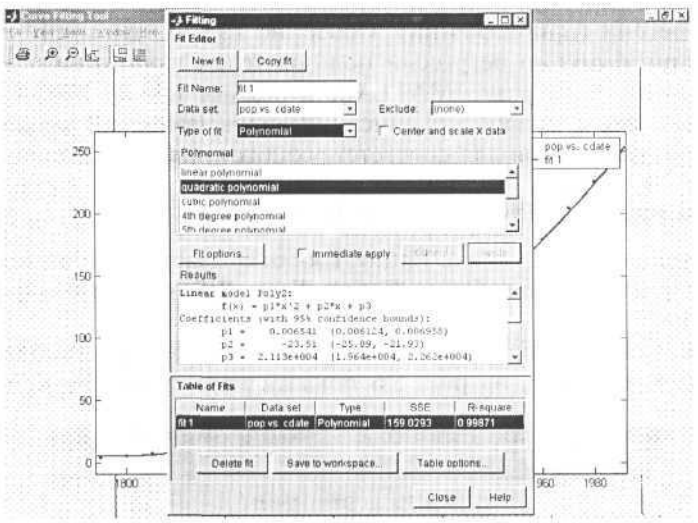

Рис. 5.4. Окно редактора подгонки Fitting

- `Data set` — задание (коррекция) типа множества данных;
- `Type of fit` — тип подгонки (вид регрессии);
- `Fit options` — вывод окна задания опций подгонки;
- `Cancel` — отмена работы с окном;
- `Apply` — применение заданных параметров (характеристик);
- `Result` — поле вывода результатов текущей подгонки;
- `Table fit` — поле таблицы сеансов подгонки;
- `Delete fit` — удаление из таблицы выделенной позиции списка;
- `Save to workspace...` — запись результатов подгонки в рабочую область памяти;

- Table options ... — вывод окна установки опций таблицы подгонки.

Назначение этих средств вполне очевидно и не нуждается в подробном пояснении. Отметим лишь, что открывающийся список Type of fit позволяет установить любой из следующих типов подгонки (регрессии):

- Custom equations — функция регрессии, задаваемая пользователем;
- Exponential — экспоненциальная регрессия;
- Fourier — приближение рядом Фурье;
- Gaussian — приближение кривой гауссиана;
- Interpolant — интерполяция с выбором метода;
- Polynomial — полиномиальная регрессия;
- Power — степенная регрессия;
- Rational — регрессия рациональной функцией с выбором числителя ее и знаменателя;
- Smoothing Spline — регрессия сглаживающими сплайнами;
- Sum of Sin Func — регрессия суммой синусоидальных функций;
- Weibull — регрессия кривой Вейбулла.

Как видно из этого перечисления, представлен внушительный набор возможных видов регрессии (приближений). Особенно важно, что среди них есть возможность задания функции регрессии пользователем любого вида — разумеется, на основе встроенных функций MATLAB. В разделе Library Models (Библиотека моделей) можно найти подробное описание видов (моделей) регрессии, а главное — расчетные выражения для соответствующих кривых регрессии.

5.2.6. Графическая визуализация регрессии

Кнопка Plotting... окна Curve Fitting Tool открывает окно установок графики, показанное на рис. 5.5 окна Curve Fitting Tool. Работа с этим простым окном вполне очевидна.

Как уже отмечалось, в окне Curve Fitting Tool можно выбрать различные функции регрессии. К примеру, на рис. 5.6 показан пример приближения заданного

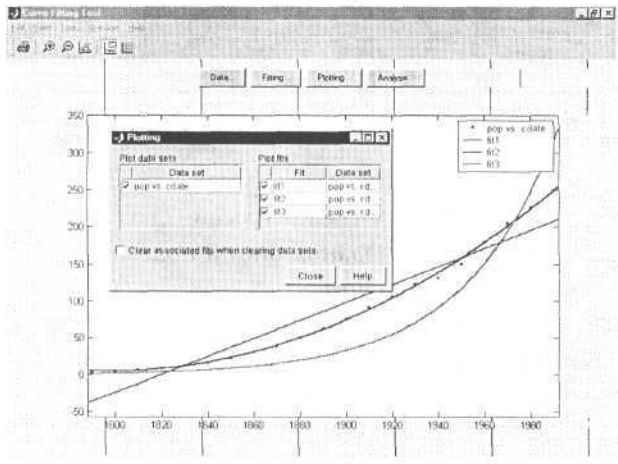

Рис. 5.5. Окно установок графики

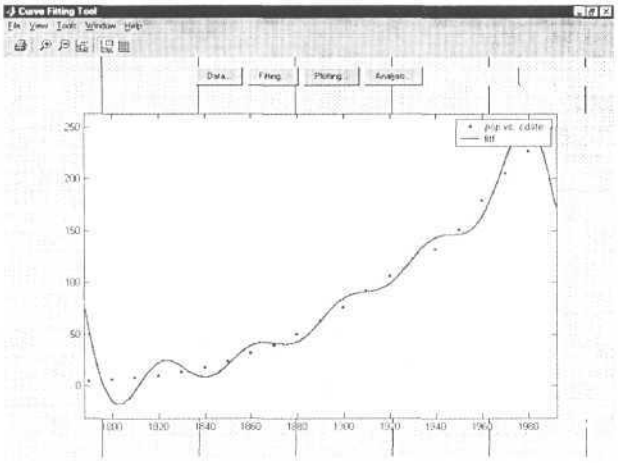

Рис. 5.6. Пример регрессии рядом Фурье

множества исходных точек рядом Фурье с 5 членами. Нетрудно заметить, что для этого множества такое приближение не является удачным — кривая регрессии сильно осциллирует, особенно вблизи начальной и конечной точек. Такие осцилляции известны как эффект Гиббса и обусловлены ограничением числа членов ряда (числа гармоник при гармоническом синтезе).

Рис. 5.7 показывает окно Fitting... для приведенного выше случая регрессии рядом Фурье. Нетрудно заметить, что ряд данных этого окна (в частности в поле описания функции регрессии) изменился в соответствии с изменением функции регрессии. Кроме того, на рис. 5.7 представлено окно опций таблицы Options Table, с помощью которого можно изменять набор табличных данных (установкой знака птички на соответствующие опции).

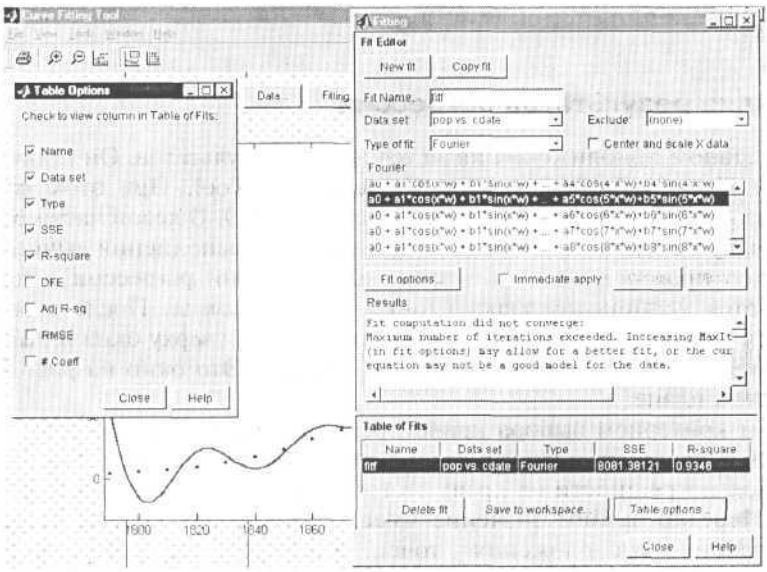

Рис. 5.7. Окно редактора подгонки Fitting регрессии рядом Фурье

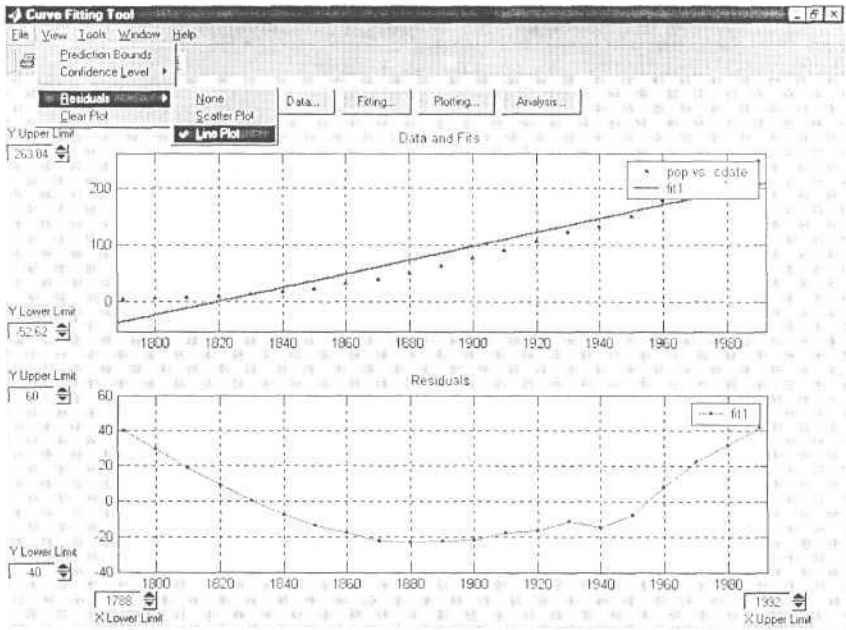

Рис. 5.8. Модификация окна Curve Fitting Tool

Окно Curve Fitting Tool имеет свое меню. Работа с ним достаточно очевидна и в подробном описании не нуждается. Отметим лишь, что с помощью позиции View (Вид) меню можно менять вид окна в широких пределах. Это иллюстрирует рис. 5.8, на котором это окно показано после включения опций вывода графика погрешности приближения Residuals, масштабной сетки и пределов по вертикальной оси графиков. Можно также установить вывод линий границ регрессии, изменить тип линий графика и т. д. Рис. 5.8 показывает позицию меню View в открытом виде.

5.2.7. Анализ результатов регрессии

Важным этапом приближения является анализ результатов. Он проводится при активизации кнопки Analysis... окна Curve Fitting Tool. При этом открывается окно анализа Analysis, частично показанное на рис. 5.9. В левой части окна имеется ряд поначалу не задействованных опций анализа: выполнения вычислений, вывода графиков производных и интеграла для функции регрессии и др. Нужные опции задаются установкой знака птички у их названия. После этого нажатие кнопки Apply формирует таблицу результатов анализа сверху окна и выводит окно с графическим представлением результатов анализа. Это окно на рис. 5.9 показано на переднем плане.

Основным критерием выбора кривой регрессии является минимум погрешности Residuals в заданном диапазоне изменения аргумента, отсутствие резких выбросов кривой погрешности на краях этого диапазона или в отдельных его областях и т. д. Нередко важное значение имеет соответствие кривой регрессии возможному закону расположения точек исходных данных. Рекомендуется опробовать несколько кривых регрессии для заданного набора точек (см. пример на рис. 5.5).

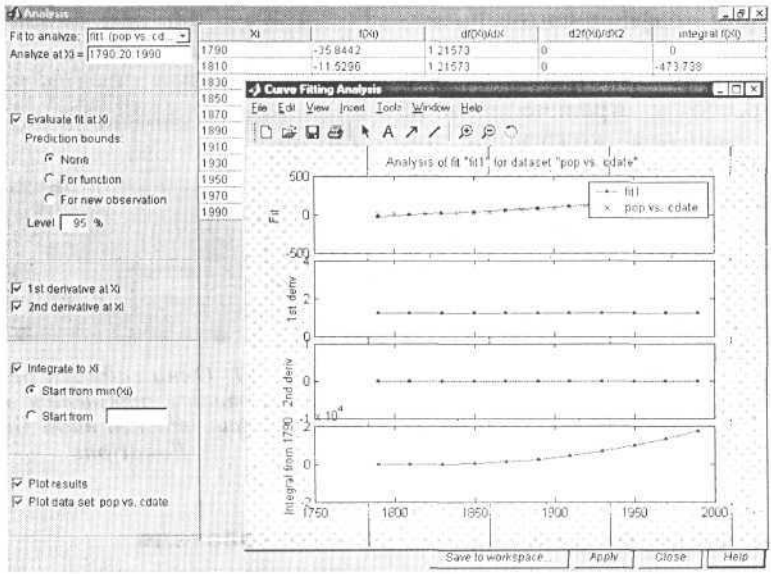

Рис. 5.9. Окна установок анализа и его графического представления

5.2.8. Регрессия с уравнением пользователя

Два обстоятельства придают пакету Curve Fitting Toolbox подлинную универсальность:

- возможность задания пользователем произвольного уравнения регрессии;
- возможность задания своего массива данных в виде векторов X и Y одного размера или сдвоенного вектора Y.

Для задания своего уравнения регрессии в списке видов регрессии надо выбрать позицию Custom equations. Это приводит к появлению окна задания регрессии Create Custom Equations (рис. 5.10).

Это окно имеет две вкладки. На первой вкладке Linear Equations можно задать параметры уравнения, линейного относительно коэффициентов регрессии. Заметим, что сама зависимость y(x) при этом может быть нелинейной — пример на рис. 5.10 дан для синусоидального терма такой зависимости.

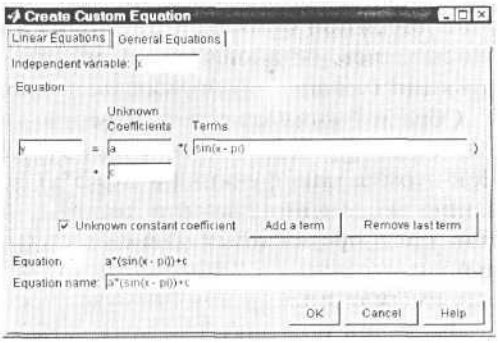

Рис. 5.10. Окно задания регрессии пользователя с открытой вкладкой линейной регрессии Linear Equations

На другой вкладке General Equations можно установить произвольное нелинейное уравнение регрессии, т. е. осуществить нелинейную регрессию. Вид окна Custom equations при открытой вкладке General Equations показан на рис. 5.11. Помимо задания самого уравнения можно задать начальные значения коэффициентов уравнения регрессии и диапазон их изменения в ходе подгонки — вычис-

ления значений коэффициентов по минимуму среднеквадратической погрешности. Таким образом становится возможной регрессия с ограничениями на возможные значения коэффициентов. По умолчанию ограничения сняты заданием пределов изменения коэффициентов от минус бесконечности (−Inf) до бесконечности (Inf).

В демонстрационных примерах пакета можно найти реализацию рациональной и тригонометрической регрессии, непараметрической регрессии и регрессии гауссианом, регрессии функцией пользователя (полиномом Лежандра) и др.

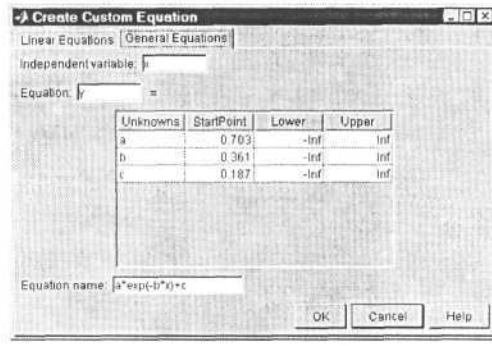

Рис. 5.11. Окно задания регрессии пользователя с открытой вкладкой регрессии общего вида General Equations

5.2.9. Пример регрессии с данными пользователя

Теперь рассмотрим полный цикл подгонки для некоторых данных пользователя. Пусть эти данные представлены следующим m-файлом с именем mydata:

```
Xdata=[2 3 4 5 6 7 8 9 10]
Ydata=[3.5 5 6.2 9 13 16 23 30 40]
```

Чтобы начать регрессию, исполним файл mydata для размещения векторов данных Xdata и Ydata в рабочую область MATLAB:

```
>> mydata
Xdata =
2   3    4    5    6    7    8    9    10
Ydata =
   3.5000   5.0000   6.2000   9.0000   13.0000   16.0000
 23.0000   30.0000   40.0000
```

Теперь, исполнив команду `cftool`, вызовем основное окно регрессии. В нем, как описывалось выше, зададим данные Xdata и Ydata и приступим к проведению четырех видов регрессии — квадратичным полиномом, степенной функцией, экспоненциальной функцией и функцией пользователя. Поскольку эти действия уже описывались, остановимся на представлении сессии подгонки в момент задания функции (уравнения) пользователя (рис. 5.12).

Обратите внимание, что четвертая кривая для подгонки задается нами как произвольная функция на вкладке General Equations окна CreateCustom Equations. При этом задана функция a*exp(b*x) + c*x^2, которая представляет собой комбинацию экспоненциальной и квадратичной зависимостей. Такая комбинация в состав предопределенных зависимостей не входит. Поскольку обе составляющие приблизительно представляют исходные данные, то можно ожидать, что из заданных именно эта зависимость будет наилучшей.

Завершив подгонку для этих четырех случаев, выведем графики исходных точек, наших четырех кривых (рис. 5.13 сверху) и погрешностей (рис. 5.13 снизу).

Опытный пользователь сразу сделает вывод о том, что введенная функция пользователя дает в данном случае лучшие результаты, чем три первые встроенные

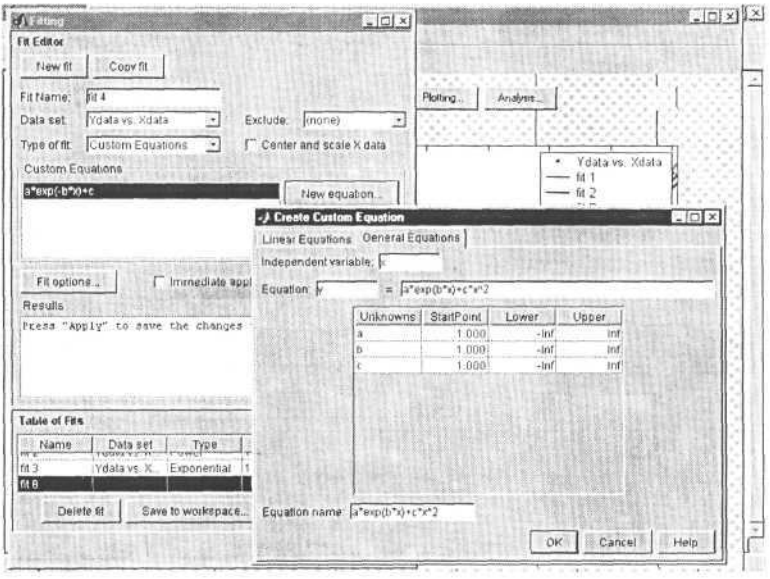

Рис. 5.12. Конец сессии подгонки по четырем кривым

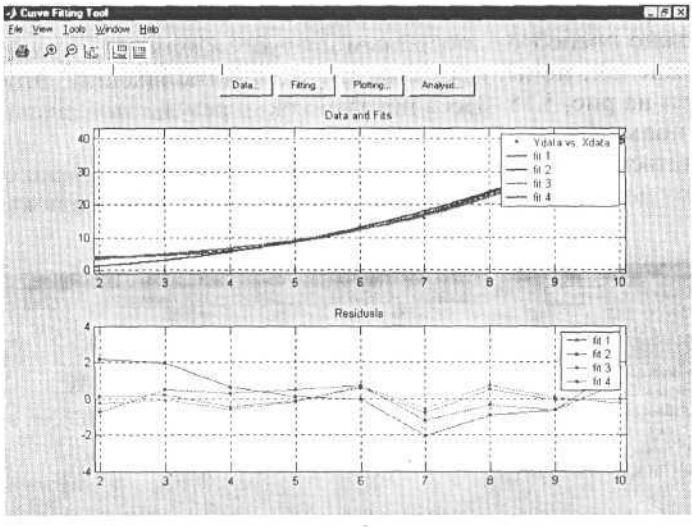

Рис. 5.13. Кривые и точки подгонки (сверху) и графики погрешности (снизу)

функции. Это следует из того (увы, на черно-белом рис. 5.13 заметить это трудно), что кривая функции пользователя ближе всего расположена к большинству точек, а кривая погрешности для нее дает меньшие значения погрешности, чем в других случаях. Впрочем, одно исключение в конце интервала есть — здесь график функции явно идет вверх. Это говорит о том, что не бесполезно заняться более глубоким анализом результатов подгонки.

Нажав кнопку Analysis... основного окна подгонки, можно перейти к стадии анализа результатов подгонки. Здесь надо задать вывод нужных результатов анализа. Этот момент сессии подгонки представлен на рис. 5.14. Обратите внимание

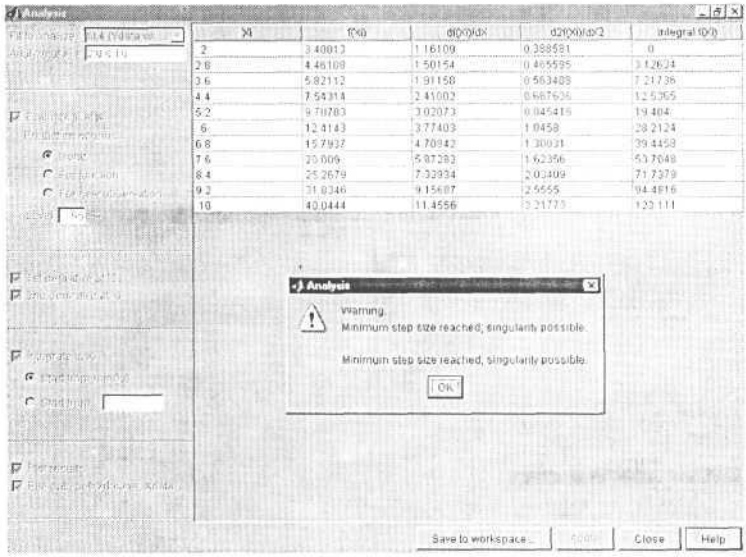

Рис. 5.14. Начало анализа результатов подгонки

на то, что пакет выдал предупреждающее сообщение о наличии сингулярной точки в начале интервала подгонки.

Нередко можно пренебречь подобным предупреждением и, нажав кнопку OK в окне предупреждения, попытаться вывести результаты анализа. Этот момент сессии представлен на рис. 5.15. Здесь показано окно результатов анализа для кривой fit4 (заданной пользователем).

Если попытаться закрыть окно результатов, то появится окошко с предложением записать результаты анализа в рабочую область. Это окошко показано в центре

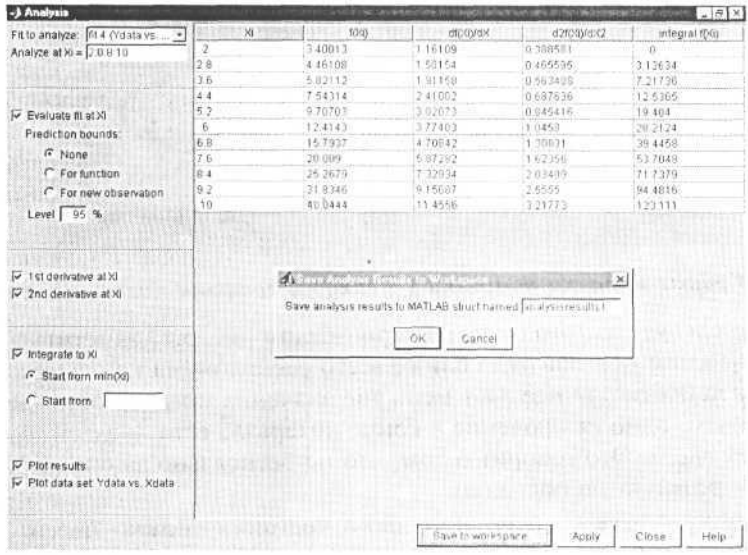

Рис. 5.15. Результаты анализа подгонки

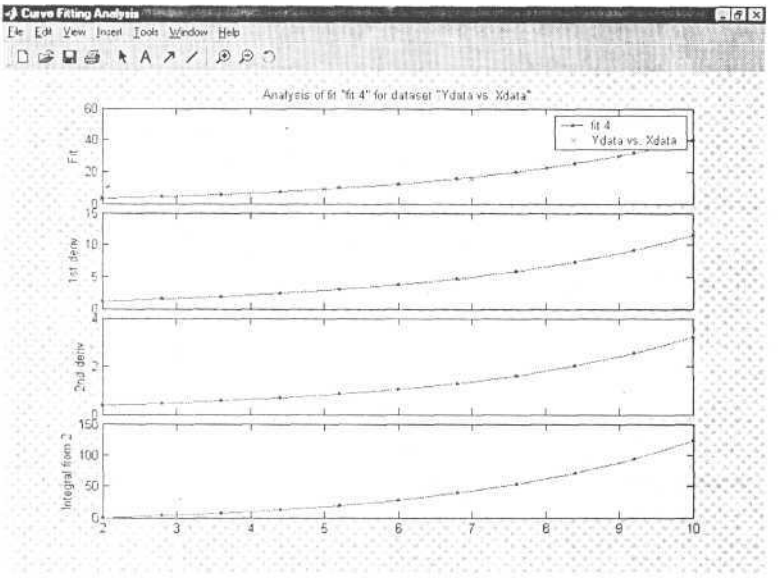

Рис. 5.16. Результаты анализа подгонки в графической форме

окна результатов на рис. 5.15. Рис. 5.16 представляет графические результаты анализа. Нетрудно заметить, что кривая функции регрессии весьма недурно представляет исходные данные (наклонные крестики), а кривые производных и интеграла от функции регрессии отличаются плавностью и отсутствием особенностей — за исключением начальной точки, где значение интеграла равно нулю. Это не мешает считать кривую fit 4 вполне приемлемой.

На этом, в принципе, сессия подгонки для данного примера может считаться законченной.

5.2.10. Установка опций подгонки

На самом деле возможности пакета Curve Fitting Toolbox гораздо выше описанных благодаря возможности задания ряда опций подгонки. Рис. 5.17 показывает окно опций подгонки, которое открывается, если нажать (с помощью мыши) кнопку Fit Options... окна подгонки Fitting. В данном случае окно опций дано для функции подгонки пользователя.

Как нетрудно заметить, в окне опций подгонки возможна установка различных алгоритмов подгонки, пределов изменения производных, максимального значения функции и начальных значений параметров регрессии. Все эти величины существенно влияют на сходимость и скорость подгонки.

5.2.11. Установка опций табличного вывода

В нижней части окна подгонки формируется таблица ее результатов. Она несет важную информацию об общих параметрах подгонки. В связи с этим полезно использовать возможность различного формирования данных для этой таблицы. Она реализуется выводом окна опций табличного вывода, которое показано на рис. 5.18 справа.

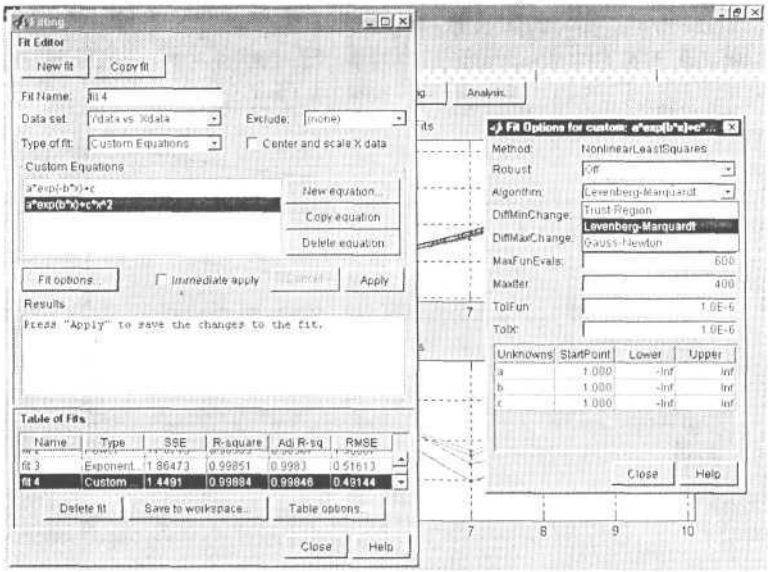

Рис. 5.17. Задание опций подгонки

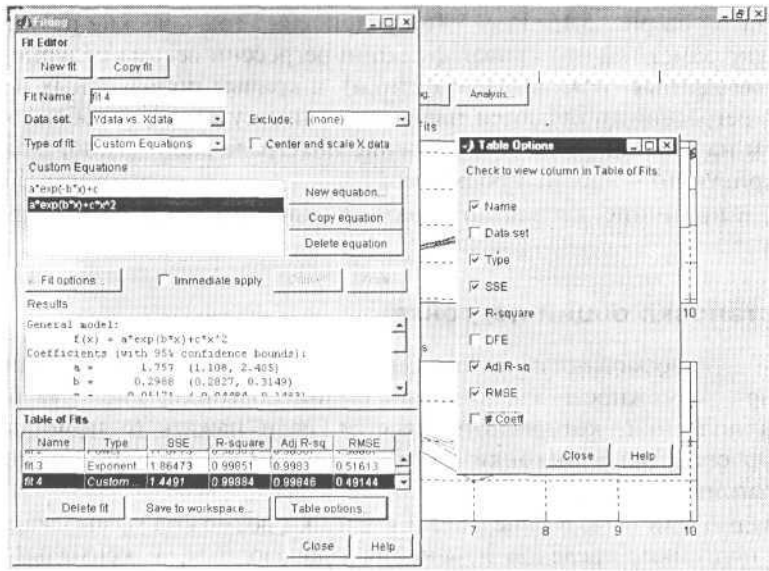

Рис. 5.18. Задание опций табличного вывода

Ряд опций табличного вывода вполне очевиден. Однако стоит упомянуть об опциях, задающих представления для погрешности с помощью различных методов ее оценки.

5.2.12. Запись и загрузка сессий подгонки

Под сессией подгонки подразумевается совокупность всех действий, приводящая к получению результатов подгонки. В частности, это установка тех или иных

опций подгонки и табличного вывода. В сложных случаях подгонки, когда число ее операций значительно, целесообразно сохранить всю сессию на диске — например для ее дополнения в последующем. Для записи секции можно воспользоваться командами `Save` или `Save As...` (рис. 5.19).

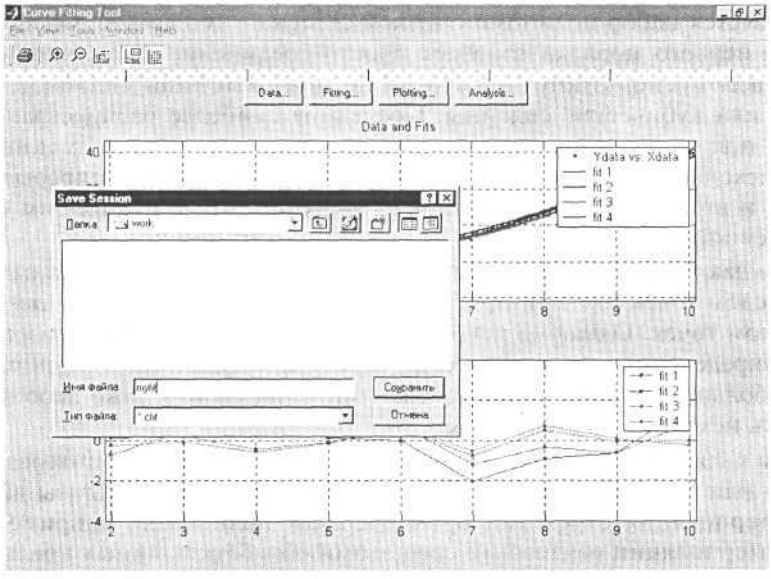

Рис. 5.19. Запись сессии на диск

Команда Save As... позволяет записать сессию под заданным именем. Сессия сохраняется в файле с расширением .cfit. Имя файла задается в окне Save Session, которое представлено на рис. 5.19 на фоне основного окна подгонки. Командой `Open` в меню `File` этого окна можно загрузить ранее сохраненную сессию.

5.2.13. Функции командного режима

Пакет Curve Fitting Toolbox предназначен в основном для работы в окнах GUI. Однако он имеет небольшой набор функций, которые не только используются при работе с пакетом в окнах GUI, но могут применяться и в командной режиме. Поскольку при наличии прекрасно сконструированного GUI-пакета самостоятельное применение этих функций мало вероятно, можно посоветовать особо любопытным пользователям ознакомиться с ними самостоятельно в разделе справки, посвященной пакету Curve Fitting Toolbox.

5.3. Приближение данных сплайнами

5.3.1. Особенности многоинтервальной интерполяции и аппроксимации

Сплайны часто представляются в виде гибкой линейки, которую закрепляют в ряде точек исходных данных и перемещают от одной группы точек к другой. Математически сплайн это степенной многочлен с малой степенью, применяемый

для интерполяции (иногда и экстраполяции) зависимости, представленной группой точек исходных данных.

Итак, главной особенностью сплайновой аппроксимации или интерполяции является то, что она является многоинтервальной. Вместо того чтобы приближать исходное облако точек полиномом высокой степени, при сплайновой интерполяции используется набор полиномов низкой степени — сплайнов. Сплайновая интерполяция первого порядка означает просто соединение точек отрезками прямых. Чаще всего используются некоторые специальные виды сплайнов, например B-сплайны или кубические сплайны. Последние наиболее распространены и используются при многоинтервальной интерполяции или аппроксимации по триадам точек исходных данных. При этом стараются обеспечить непрерывность как первой, так и второй производных сплайн-функций, что и придает им схожесть с гибкой линейкой.

Сплайны дают очень неплохие результаты при приближении плавных функций с малым числом точек или при приближении функций, представленных очень большим числом точек. Однако в некоторых случаях, для функций с разрывами или функций, определенных на неравномерной сетке, сплайновая интерполяция может давать большие выгибы между смежными точками или даже вообще неверно представлять исходные данные.

Сплайны являются далеко не единственным средством многоинтервальной интерполяции или экстраполяции. В ряде случаев неплохие результаты дает «менее гибкая» интерполяция отрезками ортогональных полиномов, например Эрмита. Этот вид интерполяции реализован средствами обработки данных прямо в графическом окне системы MATLAB новых версий.

5.3.2. Пакет Spline Toolbox

Для удобного и наглядного применения методов сплайновой аппроксимации и интерполяции служит пакет расширения Spline Toolbox. Он реализует три основных метода приближения данных сплайнами:

Spline interpolation — сплайновую интерполяцию, при которой кусочная сплайн-функция точно проходит через заданные точки и позволяет вычислять значения функции в промежутках между ними;

Spline least squre approximation — сплайновую аппроксимацию (регрессию) по методу наименьших квадратов, при которой кусочная сплайн-функция проходит в облаке точек таким образом, чтобы обеспечить наименьшую среднеквадратическую погрешность приближения;

Smoothing spline — сглаживание сплайн-функцией, при котором используется сплайновая регрессия с применением алгоритмов сглаживания данных.

Разумеется, в силу принципиально разных подходов к приближению сплайнами они дают различные результаты. Они будут проиллюстрированы в дальнейшем.

5.3.3. Работа с пакетом Spline Toolbox в командном режиме

Пакет Spline Toolbox содержит определения примерно 40 функций, которые могут использоваться в командном режиме работы. Поскольку на практике в таком применении нет особой необходимости, то пользователь может легко ознакомиться с этими функциями самостоятельно по справке на этот пакет. В свзи с этим мы ограничимся парой наглядных примеров.

Следующий m-файл использует функцию spcol для построения компонент B-сплайна в графическом виде (рис. 5.20):

```
t=[0,1,1,3,4,6,6,6]; x=linspace(-1,7,81); c=spcol(t,3,x);
[l,m]=size(c); c=c+ones(1,1)*[0:m-1]; axis([-1 7 0 m]);
hold on; for tt=t, plot([tt tt],[0 m],'-'), end
plot(x,c,'linew',2), hold off, axis off
```

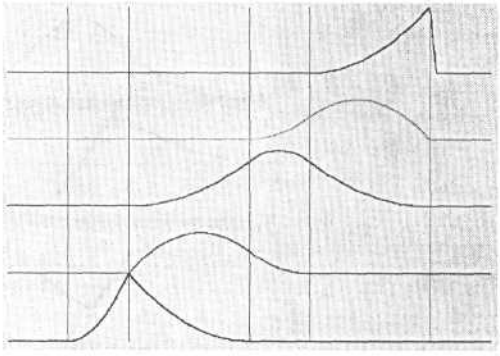

Рис. 5.20. Построение компонент B-сплайна

Другой пример иллюстрирует построение кольца Мебиусса (рис. 5.21) на основе тензорного произведения с применением двухвариантной сплайновой интерполяции:

```
x = 0:1; y = 0:4; h = 1/4; o2 = 1/sqrt(2); s = 2; ss = 4;
v(3,:,:) = h*[0,-1,-o2,0,o2,1,0;0,1,o2,0,-o2,-1,0];
v(2,:,:) = [ss,0,s-h*o2,0,-s-h*o2,0,ss;ss,0,s+h*o2,0,-s+h*o2, 0,
 ss];
v(1,:,:) = s*[0,1,0,-1+h,0,1,0;0,1,0,-1-h,0,1,0];
cs = csape({x,y},v,{'variational','clamped'});
fnplt(cs), axis([-2 2 -2.5 2.5 -.5 .5]), shading interp; axis
 off, hold on
values = squeeze(fnval(cs,{1,linspace(y(1),y(end),51)}));
plot3(values(1,:), values(2,:), values(3,:),'k','linew',2); hold
off
```

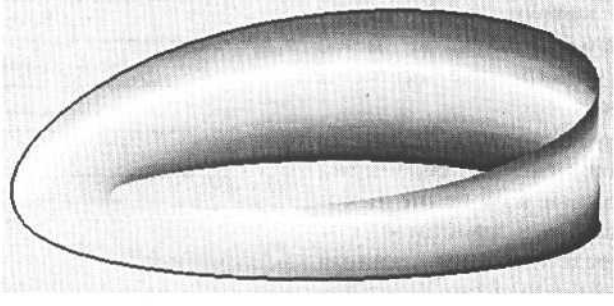

Рис. 5.21. Построение кольца Мебиусса

В основном данные функции используются для поддержки наглядных и простых в работе средств GUI пакета. Их мы и рассмотрим более детально.

5.3.4. Работа с GUI пакета Spline Toolbox

GUI пакета Spline Toolbox представляет удобные окна с деталями интерфейса, обеспечивающие эффективную работу со сплайнами. Команда bspligui открывает доступ к окну GUI, предназначенному для работы с B-сплайнами (рис. 5.22).

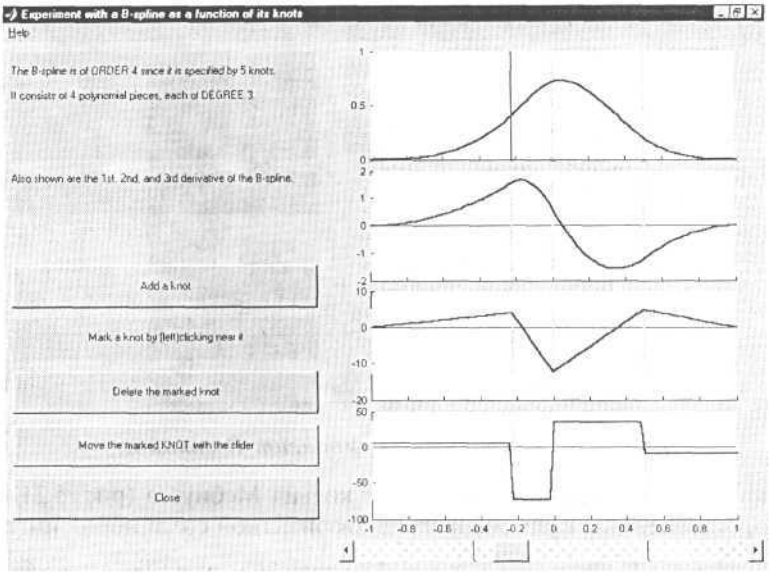

Рис. 5.22. Окно GUI для работы с B-сплайнами

С помощью кнопок в левой части окна можно выполнить ряд экспериментов с B-сплайнами. Впрочем, этот вид сплайнов нельзя отнести к широко распространенным.

Более широко используются обычные кубические сплайны. Команда splinetool открывает окно с меню примеров для работы с такими сплайнами (рис. 5.23).

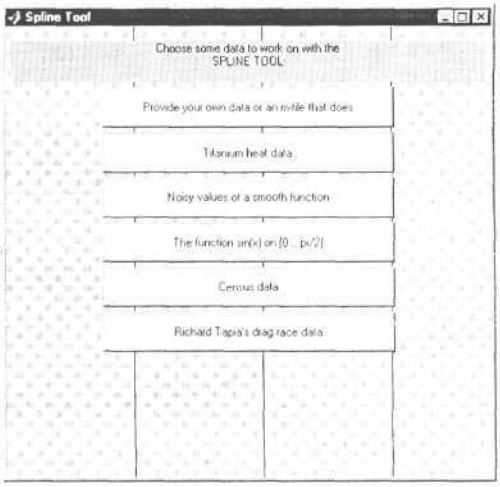

Рис. 5.23. Окно GUI с меню примеров работы с кубическими сплайнами

Первый пункт этого меню позволяет задать данные пользователя в явном виде или в виде m-файла и опробовать на этих данных методы приближения сплайнами. Остальные пункты меню дают наглядные конкретные примеры такого приближения.

На рис. 5.24 показано окно одного из примеров — приближение сплайнами кривой, описывающей изменение некоторого свойства титана в зависимости от температуры. Представленная кривая (график сверху) весьма благоприятна для приближения кубическим сплайном, о чем свидетельствует нулевая погрешность приближения — график снизу. Вы можете самостоятельно опробовать дру-

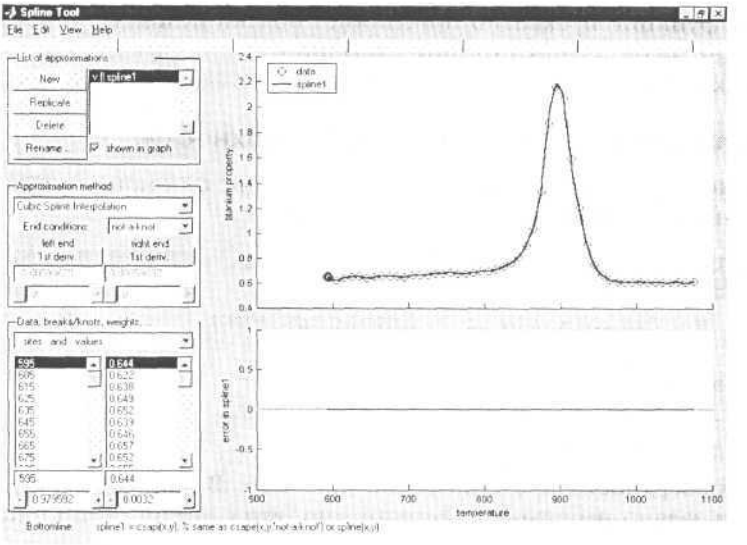

Рис. 5.24. Окно GUI с примером точного приближения данных кубическими сплайнами

гие методы приближения (аппроксимации) и убедиться в том, что в данном случае они ведут лишь к резкому нарастанию погрешности.

А теперь рассмотрим другой пример — аппроксимации синусоиды, засоренной шумами. Сама синусоида гладкая функция и прекрасно подается приближению сплайнами. Однако когда данные содержат шумы (случайные отклонения функции от синусоидального закона), то картина резко меняется и сплайновая аппроксимация по всем точкам ведет к большим погрешностям. Лучшие результаты (рис. 5.25) дает сглаживание такой функции сплайнами. Нетрудно увидеть, что

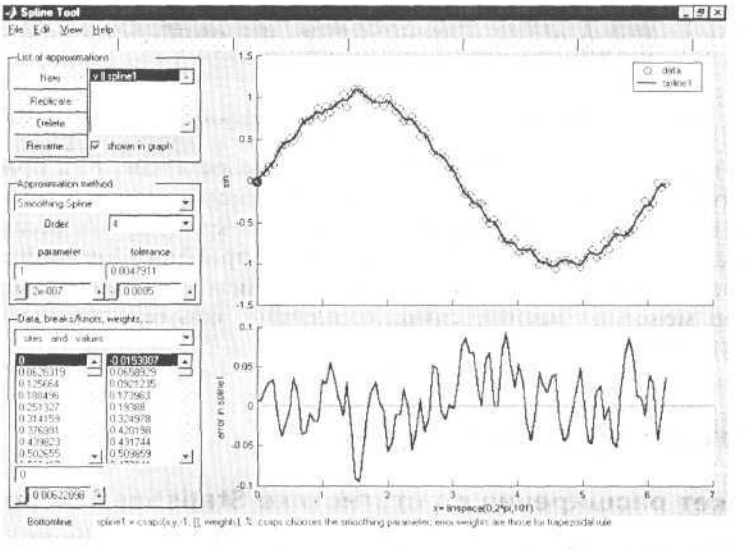

Рис. 5.25. Окно GUI с примером сглаживания данных кубическими сплайнами

сглаженная функция прекрасно приближает облако исходных точек, явно располагаясь между ними. Однако график погрешности показывает, что она значительна, поскольку сглаженная функция не проходит точно через все точки.

5.3.5. Сплайновая аппроксимация поверхности

В заключение рассмотрим пример на построение с помощью сплайнов поверхности, заданной векторами x и y и типом franke:

```
x = sort([[0:10]/10,.06 .07, .95 .85]);
y = sort([[0:6]/6,.03 .87, .74 .51]);
[xx,yy] = ndgrid(x,y); z = franke(xx,yy);
ky = 3; knotsy = augknt([0:4]/4,ky);
sp = spap2(knotsy,ky,y,z);
yy = [-.1:.05:1.1]; vals = fnval(sp,yy);
mesh(x,yy,vals.'), view(120,30)
```

График поверхности представлен на рис. 5.26. В данном случае число линий поверхности задано небольшим. Естественно, что с увеличением числа линий поверхности ее представление улучшается.

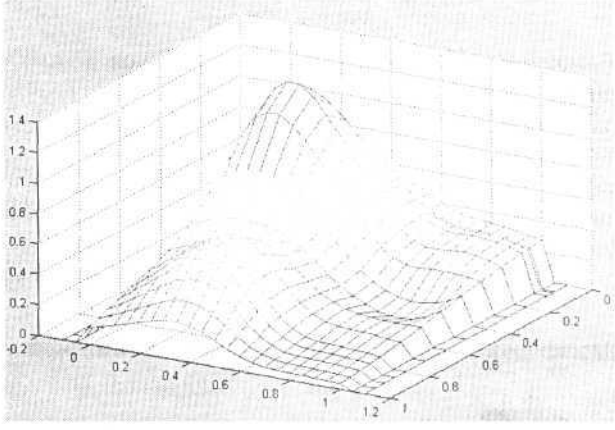

Рис. 5.26. Окно GUI с примером сглаживания данных кубическими сплайнами

Сплайн-приближение не лишено серьезных недостатков. Как правило, частичные полиномы, из которых состоит сплайн-функция, специально не выводятся. Да и толку от такого вывода мало, поскольку они громоздки и полиномов много. Таким образом, для пользователя сплайновое приближение часто оказывается «вещью в себе». Скорость такого приближения невелика вследствие его сложности. Тем не менее во многих случаях сплайны оказываются весьма удобными средствами для приближения сложных данных.

5.4. Статистические расчеты

5.4.1. Пакет расширения по статистике Statistics Toolbox

Как отмечалось в начале этой главы, средства статистических вычислений в базовой системе MATLAB ограничены. Для выполнения большинства типовых ста-

тистических вычислений служит пакет расширения Statistics Toolbox (версии 4.1 и 5 для систем MATLAB 6.5 SP1 и 7). Он ориентирован на широкий спектр задач: от генерации случайных чисел и подбора кривых под экспериментальные данные до планирования экспериментов и задач промышленного статистического контроля. Инструментальные средства пакета позволяют использовать как в режиме командной строки, так и с применением графического интерфейса пользователя GUI.

5.4.2. Вычисление распределения и плотности вероятностей

Пакет позволяет задавать около двух десятков различных типов *распределения вероятностей*, в том числе нормальный, Стьюдента, Фишера, экспоненциальный тип и т. д. С каждым *законом распределения* связаны:

- плотность распределения вероятностей или функция плотности вероятности (probability density function — pdf) — f(x);
- интегральная функция распределения (cumulative distribution function — cdf), F(x);
- функция, обратная к интегральной функции распределения, $F^{-1}(x)$;
- генерация случайных чисел;
- среднее значение и дисперсия как функции параметров распределения;
- функции оценки параметров закона распределения.

Функции данной группы в пакете Statistics Toolbox имеют стандартный формат, отличающийся только по имени используемого распределения. С «тонкостями» задания конкретных функций удобно знакомиться с помощью справки по этому пакету.

Следующий пример показывает построение кривой нормального закона распределения (рис. 5.27):

```
>> x = [-4:0.1:4]; f = normpdf(x,0,1);plot(x,f);grid on
```

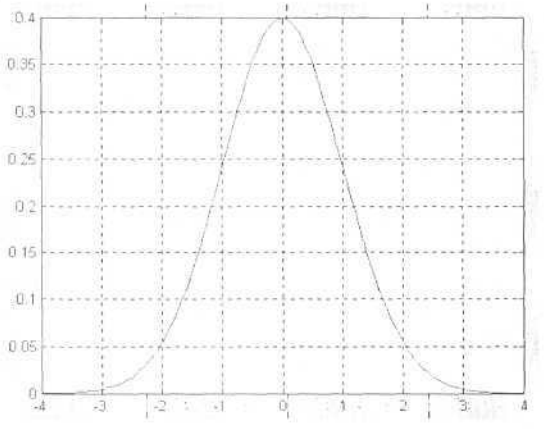

Рис. 5.27. Кривая нормального распределения вероятности

В данном случае функция `normpdf(x,0,1)` для заданного x возвращает значение функции плотности вероятности с математическим ожиданием 0 и среднеквадратическим отклонением (корнем квадратным из дисперсии) — 1.

Аналогичным образом используются функции, относящиеся к другим распределениям, — необходимо только знать (и задавать) соответствующие параметры

выбранного закона. Функции данной подгруппы представлены в табл. 5.1. Их имена образованы путем слияния первых букв названия закона (естественно, по-английски) и «окончания» pdf.

Таблица 5.1

Функции плотности распределения вероятностей

Имя функции	Закон распределения
betapdf	Бета-распределение
binopdf	Биномиальный
chi2pdf	Хи-квадрат
Exppdf	Экспоненциальный
Fpdf	Фишера
Gampdf	Гамма-распределение
Geopdf	Геометрический
hygepdf	Гипергеометрический
lognpdf	Логнормальный
nbinpdf	Отрицательный биномиальный
Ncfpdf	Нецентральное (смещенное) распределение Фишера
Nctpdf	Нецентральное (смещенное) распределение Стьюдента
ncx2pdf	Нецентральное (смещенное) Хи-квадрат распределение
Pdf	Закон распределения задается как аргумент функции
poisspdf	Пуассона
raylpdf	Рэлея
Tpdf	Стьюдента
unidpdf	Дискретное равномерное распределение
unifpdf	Равномерный (прямоугольный)
weibpdf	Вейбулла

Интегральные функции распределения вероятностей $F(x)$ имеют имена, отличаются окончанием cdf, а не pdf. Следующий пример иллюстрирует нахождение $F(x)$ для экспоненциального закона распределения с параметрами MU=1 и x=2:

```
>> expcdf(2,1)
ans =
0.8647
```

Функции, обратные к интегральным функциям распределения, возвращают значение $F^{-1}(x)$. Имя функции образуется из имени закона распределения и окончания inv. Например, для экспоненциального распределения имеем:

```
>> expinv(0.8647,1)
ans =
2.0003
```

5.4.3. Генерация случайных чисел

Функции для генерации случайных чисел возвращают такие числа с заданным законом распределения. Имя функции образовано путем слияния первых букв названия закона и окончания `rnd`. Например, команда

```
>> exprnd(1)
ans =
1.0417
```

обеспечивает генерацию одного случайного числа, подчиняющегося экспоненциальному закону распределения с параметром MU = 1. Генерация группы случайных чисел (подчиняющихся тому же закону распределения), образующих матрицу с 3 строками и 4 столбцами:

```
>> exprnd(1,[3 4])
ans =
    0.2068    1.5957    1.3013    0.2920
    4.6191    1.6158    1.6154    0.8095
    1.9741    0.5045    4.1816    0.0706
```

В другом примере

```
>> exprnd(1,3,4)
ans =
    0.7636    0.6441    0.1766    0.9690
    0.8707    1.5963    3.9302    0.1842
    0.1670    0.3973    0.3838    0.6875
```

используется несколько иной формой записи команды.

5.4.4. Вычисление среднего и дисперсии

Функции вычисления среднего и дисперсии возвращают математические ожидание и дисперсию указанного распределения в зависимости от его заданных параметров. Имя функции данной подгруппы образовано путем слияния названия закона и окончания `stat`. Следующий пример иллюстрирует нахождение математического ожидания и дисперсии экспоненциального закона распределения для четырех значений его параметра — 1, 10, 100 и 1000:

```
>> [m,v] = expstat([1 10 100 1000])
m =
1          10         100         1000
v =
1         100       10000      1000000
```

5.4.5. Оценка параметров законов распределения

Функции для оценки параметров законов распределений по экспериментальным данным имеют имена, образованные слиянием названия закона и окончания `fit`. К примеру, оценивание параметра экспоненциального закона производится следующим образом:

```
>> x=exprnd(1,100,1); % Генерация 100 элементов выборки
% Нахождение оценки параметра (точное значение - 1)
```

```
>> expfit(x)
ans =
0.9157
```

5.4.6. Дескриптивная статистика

Функции дескриптивной статистики позволяют определять набор сводных характеристик исходного множества обрабатываемых данных (выборки): оценки центра группирования значений случайной величины, степени рассеяния, коэффициента корреляции, эксцесса и т. п., иначе говоря, характеристики *описательной* или *дескриптивной статистики*. Полный перечень функций дескриптивной статистики пакета Statistics Toolbox приведен в табл. 5.2. Некоторые из них, вообще говоря, являются встроенными функциями MATLAB (например, `corrcoef`, `cov`, `mean`, `median`, `std`).

Таблица 5.2

Функции дескриптивной статистики

Имя функции	Возвращаемая величина
`corrcoef`	Оценка коэффициента корреляции
`Cov`	Оценка матрицы ковариаций
`Geomean`	Среднее геометрическое
`harmmean`	Среднее гармоническое
`Iqr`	Оценка разности между 75- и 25-процентными точками (процентилями), иначе – разность между 3-й и 1-й квартилями
`kurtosis`	Оценка коэффициента эксцесса (от обычно используемого в отечественной литературе параметра b2 отличается на величину 3, то есть kurtosis = b2 + 3)
`Mad`	Среднее абсолютное отклонение от среднего значения
`Mean`	Арифметическое среднее
`Median`	Медиана
`Moment`	Оценка центрального момента произвольного (задаваемого как аргумент) порядка
`Nanmax`	Максимальное значение, находимое с игнорированием нечисловых элементов
`Nanmean`	Арифметическое среднее, находимое с игнорированием нечисловых элементов
`nanmedian`	Медиана, находимая с игнорированием нечисловых элементов
`Nanmin`	Минимальное значение, находимое с игнорированием нечисловых элементов
`Nanstd`	Оценка среднеквадратического отклонения, находимая с игнорированием нечисловых элементов
`Nansum`	Сумма, находимая с игнорированием нечисловых элементов
`Prctile`	Выборочная процентная точка (процентиль)
`Range`	Размах выборки
`skewness`	Оценка коэффициента асимметрии

Имя функции	Возвращаемая величина
Std	Оценка среднеквадратического отклонения
trimmean	Оценка среднего, находимая с игнорированием заданного процента минимальных и максимальных элементов выборки
Var	Оценка дисперсии

В приведенном ниже примере

```
>> x = normrnd(0,1,100,100);
>> s = std(x);>> s_MAD = 1.3 * mad(x);
>> efficiency = (norm(s - 1)./norm(s_MAD - 1)).^2
efficiency =
0.7383
```

сначала создается матрица случайных чисел размера 100 × 100, подчиненных нормальному закону распределения с параметрами соответственно 0 и 1. Затем по данной выборке находятся оценки среднеквадратического отклонения и средние абсолютных отклонений от среднего значения (по 100 значений), после чего оценивается относительная эффективность (efficiency) данных величин как оценок степени рассеяния элементов выборки (для корректности подобного сравнения среднее значение абсолютных отклонений умножается на множитель 1.3). Конечный результат говорит о том, что при нормальном законе распределения более эффективной оценкой степени рассеяния является оценка среднеквадратического отклонения.

Следующий пример иллюстрирует нахождение среднего значения для массивов данных, в которых некоторые элементы не определены (пропущены):

```
>> m = magic(3);
>> m([1 6 8]) = [NaN NaN NaN]
m =
   NaN     1     6
     3     5   NaN
     4   NaN     2
>> nmean = nanmean(m)
nmean =
3.5000 3.0000 4.0000
```

Исходная матрица данных — «магический квадрат» 3 × 3, в котором затем 1-й, 6-й и 8-й элементы сделаны нечисловыми.

В еще одном примере по генерируемой выборке случайных чисел, подчиняющихся нормальному закону с нулевым средним и единичной дисперсией, определяются оценки коэффициентов эксцесса и асимметрии.

```
>> X = randn([5 4])
X =
    0.1096   -0.2009    1.0306   -3.5968
    0.1121   -0.7846    0.2014   -1.8284
    0.8969    0.8279   -1.2392   -0.6119
    0.7632   -0.2592   -0.5260   -0.7429
    0.1649   -1.5061    0.6314    0.0440
```

```
>> k = kurtosis(X)
k =
1.2783   2.1848   1.7258   2.2550
>> y = skewness(X)
y =
0.2855   0.1403   -0.3298  -0.7890
```

5.4.7. Кластерный анализ

В группу функций кластерного анализа входит 9 функций, представленных ниже. Функция pdist возвращает парные расстояния между объектами (векторами):

```
Y = pdist(X)  Y = pdist(X,'metric')
Y = pdist(X,'minkowski',p)
```

Аргументы функции: X — матрица данных, имеющая m строк и n столбцов, рассматриваемая как совокупность m векторов с n элементами каждый; 'metric' — строковая переменная, принимающая возможные значения:

- `Euclid` — евклидово расстояние;
- `SEuclid` — нормализованное евклидово расстояние;
- `Mahal` — расстояние Махаланобиса;
- `CityBlock` — расстояние по Манхэттену (расстояние Хэмминга);
- `Minkowski` — расстояние в метрике Минковского.

p — некоторое фиксированное число, показатель метрики Минковского (по умолчанию — 2).

Возвращаемая величина Y — вектор с $(m-1)m/2$ элементами, которые являются расстояниями соответственно между вектором 1 и вектором 2, вектором 1 и вектором 3, ..., вектором 1 и вектором m, ..., вектором $m-1$ и вектором m.

В следующем примере находятся 6 взаимных евклидовых расстояний между 4 векторами:

```
>> X = [1 2; 1 3; 2; 3 1]
X =
1    2
1    3
2    2
3    1
>> Y = pdist(X)
Y =
1.0000   1.0000   2.2361   1.4142   2.8284   1.4142
```

Функция Z=zscore(D) осуществляет масштабирование (нормализацию) по столбцам элементов матрицы D. Расчеты выполняются по формуле

```
Zi = (Di - mean(Di))./(std(Di)),
```

где Zi, Di — столбцы матриц Z и D. Функция используется для предварительного преобразования экспериментальных данных.

Функция squareform(Y) преобразует вектор Y, возвращаемый функцией pdist, в симметричную квадратную матрицу. Пример:

```
>> Y = [1.0000 1.0000 2.2361 1.4142 2.8284 1.4142];
>> squareform(Y)
```

```
ans =
         0    1.0000    1.0000    2.2361
    1.0000         0    1.4142    2.8284
    1.0000    1.4142         0    1.4142
    2.2361    2.8284    1.4142         0
```

Функция `Z = linkage(Y)` или `Z = linkage(Y,'method')` возвращает иерархическое дерево кластеров, используя по умолчанию алгоритм «ближайшего соседа». Она является исходной для ряда других функций, в частности, для функции построения дендрограммы dendrogram (см. ниже). Аргументами функции являются вектор Y расстояний между m объектами, возвращаемый функцией pdist, и строковая переменная 'method' (метод), которая задает метод кластеризации и может принимать одно из следующих значений:

- `` `single` `` — алгоритм «ближайшего соседа»;
- `` `complete` `` — алгоритм «дальнего соседа»;
- `` `average` `` — алгоритм «средней связи»;
- `` `centroid` `` — центроидный алгоритм, использующий расстояние по «центрам тяжести» групп;
- `` `ward` `` — пошаговый алгоритм.

Функция возвращает матрицу `Z`, имеющую $m - 1$ строку и 3 столбца и содержащую информацию об иерархическом дереве кластеров. Нижний уровень иерархии образован m исходными объектами (векторами), которые затем объединяются попарно (в зависимости от расстояния между ними), образуя новые кластеры, которые также могут объединяться попарно (в зависимости от взаимного расстояния) и т. д. Каждому новому формируемому кластеру, соответствующему i-й строке матрицы `Z`, присваивается индекс $m + i$ (m — число исходных объектов). Столбцы 1 и 2 матрицы `Z` содержат индексы объектов (кластеров), которые были объединены в пары при образовании нового кластера, а столбец 3 — расстояние между данными объектами. Всего подобным образом будет организован $m - 1$ кластер.

Пример.

```
>> X = [3 1.7; 1 1; 2 3; 2 2.5; 1.2 1; 1.1 1.5; 3 1];
>> Y = pdist(X);
>> Z = linkage(Y)
Z =
   2.0000    5.0000    0.2000
   3.0000    4.0000    0.5000
   8.0000    6.0000    0.5099
   1.0000    7.0000    0.7000
  11.0000    9.0000    1.2806
  12.0000   10.0000    1.3454
```

Результаты расчетов показывают, что кластеры образованы объектами (векторами) 2 и 5, 3 и 4, 1 и 7, новым объектом с индексом 8 и объектом 6 и т. д.

Функция `dendrogram(Z)` возвращает графическое отображение результата выполнения функции `linkage`. Пример использования функции приведен на рис. 5.28.

Функция `cophenet(Z,Y)`, где `Y` и `Z` — матрицы, возвращаемые соответственно функциями `pdist` и `linkage`, возвращает аналог коэффициента корреляции, ха-

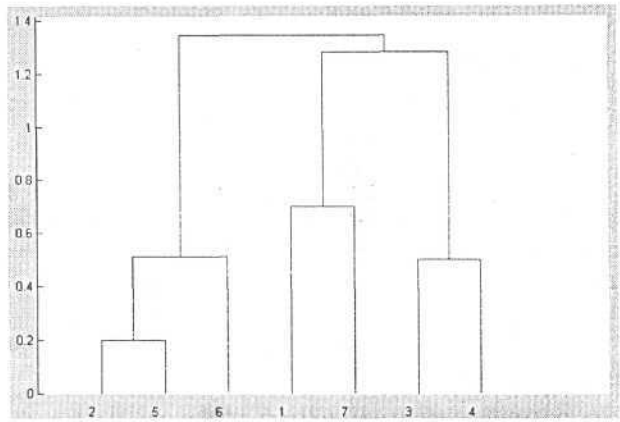

Рис. 5.28. Пример дендрограммы

рактеризующий качество (чем ближе к 1, тем лучше) разбиения исходных объектов на дерево кластеров. Применение данной функции к условиям предыдущего примера дает результат, который можно оценить как удовлетворительный:

```
>> cophenet(Z,Y)
ans =
0.7514
```

Функция `inconsistent(Z)` возвращает значения так называемых коэффициентов несовместимости для каждой связи иерархического дерева кластеров. Аргумент — матрица `Z`, возвращаемая функцией `linkage`. Может использоваться для оценки качества разбиения на кластеры исходного множества объектов.

Функция `cluster` осуществляет разбиение иерархического дерева кластеров на отдельные кластеры:

```
T = cluster(Z,cutoff)    T = cluster(Z,cutoff,depth)
```

Аргументы этой функции: `Z` — матрица, возвращаемая функцией `linkage`; `cutoff` — при значении от 0 до 1 воспринимается как пороговая величина для выделения отдельных кластеров путем исключения связей дендрограммы, если соответствующие коэффициенты несовместимости превышают данный порог, а при значении больше 1 — как задаваемое число кластеров; `depth` — аргумент, определяющий «глубину» иерархии, для которой подсчитываются коэффициенты несовместимости.

Возвращаемая величина — вектор `T`, элементы которого являются номерами кластеров, к которым отнесены исходные объекты. Для ранее рассмотренного примера имеем

```
>> z=[2.0000 5.0000 0.2000;
3.0000 4.0000 0.5000;
8.0000 6.0000 0.5099;
1.0000 7.0000 0.7000;
11.0000 9.0000 1.2806;
12.0000 10.0000 1.3454];
>> T = cluster(z,3)
T =
1
```

```
3
2
2
3
3
```

Здесь из полученного иерархического дерева выделено 3 кластера, при этом вектор 1 отнесен к первому кластеру, вектор 2 — к третьему и т. д.

Функция `T = clusterdata(X,cutoff)` осуществляет кластеризацию аналогично предыдущей функции, но по отношению к исходной матрице данных `X`.

5.4.8. Линейные модели

Функции данной группы (в табл. 5.3 приведены основные функции) реализуют процедуры одно- и двухфакторного дисперсионного анализа и линейного (по параметрам модели) регрессионного анализа.

Таблица 5.3

Функции линейных моделей

Имя функции	Реализуемая процедура
`anoval`	Однофакторный регрессионный анализ
`anova2`	Двухфакторный регрессионный анализ
`Lscov`	Линейная регрессия при заданной матрице ковариаций (встроенная функция MATLAB)
`Polyconf`	Определения доверительных интервалов для линии регрессии, возвращаемой функцией `polyfit`
`Polyfit`	Полиномиальная регрессия (встроенная функция MATLAB)
`Polyval`	Прогноз с использованием полиномиальной регрессии (встроенная функция MATLAB)
`Regress`	Множественная линейная регрессия
`Ridge`	Линейная регрессия с применением гребневых оценок (ридж-оценок)
`Rstool`	Интерактивный подбор и визуализация поверхности отклика
`Stepwise`	Пошаговая регрессия (графический интерфейс пользователя)

Большинство приведенных функций реализуют хорошо известные статистические методы обработки данных, поэтому остановимся подробнее только на двух наиболее интересных последних функциях.

Функция rstool записывается в виде:

```
rstool(x,y) rstool(x,y,'model')
rstool(x,y,'model',alpha,'xname','yname')
```

Она строит и вызывает графическое отображение линейной по параметрам (но не по факторам!) регрессионной модели. Аргументы функции:

- `x`, `y` — матрица и вектор экспериментальных данных соответственно для факторов и отклика;

- `model` — строковая переменная, задающая вид начальной модели регрессии и принимающая значения `interaction` (модель содержит константу, линейные члены и парные взаимодействия факторов), `quadratic` (модель содержит парные взаимодействия и квадратичные члены), `purequadratic` (модель содержит константу, линейные члены, парные взаимодействия и квадратичные члены);
- `alpha` — параметр, задающий доверительный коридор для прогнозируемых значений, равный 100(1 — `alpha`) %;
- `xname`, `yname` — строковые переменные, задающие наименования осей графика.

Используя меню появляющегося графического окна, можно менять вид модели, экспортировать выбранные параметры в рабочее пространство MATLAB и т. п.

Пример использования функции иллюстрируется приведенным ниже листингом m-файла и рис. 5.29 (в данном случае речь идет о восстановлении квадратичной зависимости).

```
>> X=[-1; -0.5; 0; 0.5; 1]; y=[1; 0.25; 0; 0.25; 1];
>> rstool(X,y,'purequadratic')
```

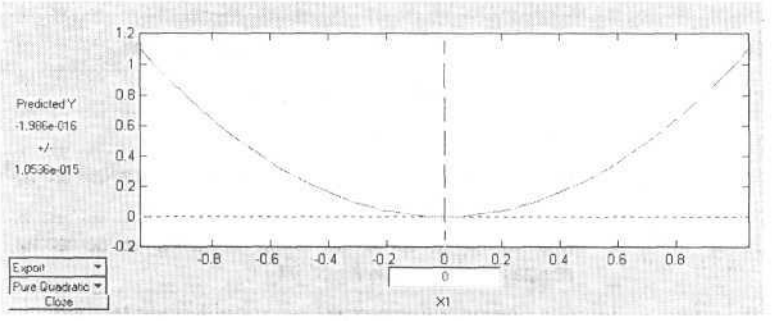

Рис. 5.29. Графический интерфейс функции rstool

5.4.9. Пошаговая регрессия

Функция `stepwise` реализует пошаговую регрессию с использованием средств графического интерфейса. Она записывается в виде:

```
stepwise(X,y) stepwise(X,y,inmodel)
stepwise(X,y,inmodel,alpha)
```

Аргументы `X`, `y` и `alpha` имеют тот же смысл, что и для предыдущей функции, `inmodel` — вектор, элементы которого указывают номера переменных (столбцов матрицы `X`), включенных в регрессию. При выполнении функции открываются три графических окна (коэффициенты регрессии, таблица значений, график истории построения модели), позволяющих контролировать качество регрессионной модели, введение и удаление отдельных факторов.

Следующий фрагмент MATLAB-программы и рис. 5.30 иллюстрируют пример применения данной функции.

```
>> load hald
>> stepwise(ingredients,heat)
```

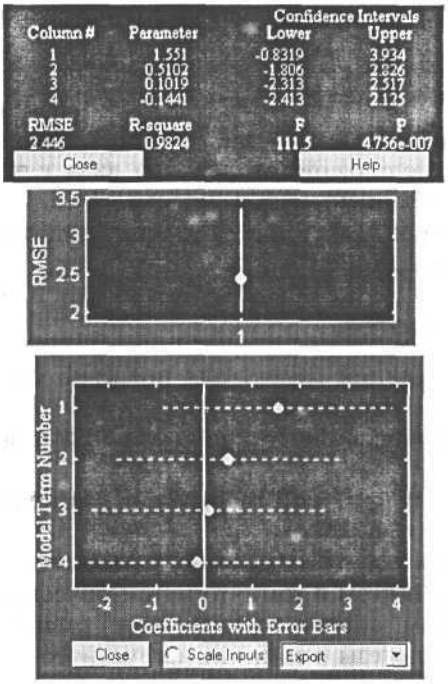

Рис. 5.30. Графический интерфейс функции stepwise

Заметим, что «включение» в модель и «выключение» из нее отдельных регрессоров производится в окне коэффициентов регрессии (нижнее окно на рис. 5.30) с использованием мыши — выделением линии, соответствующей выбранному регрессору (вначале введенными в модель считаются все факторы). В окне истории построения модели каждое изменение в наборе регрессоров (то есть каждый шаг построения наилучшей модели) отображается точкой, характеризующей величину ошибки модели; адекватные изменения происходят и в окне таблицы значений. Пошаговый интерактивный отбор регрессоров в конце концов приводит к получению наиболее экономной и точной модели (при умеренном числе исходных факторов).

5.4.10. Нелинейные регрессионные модели

Функции данной группы позволяют определять оценки параметров нелинейных моделей, постулируемых для имеющихся экспериментальных данных. Имеются следующие функции:

`nlinfit` — реализует нелинейный метод наименьших квадратов — МНК (метод Гаусса—Ньютона);
`nlintool` — возвращает график прогнозируемых значений (прямой аналог функции `rstool`);
`nlparci` — возвращает вектор доверительных интервалов для оценок параметров модели;
`nlpredci` — возвращает прогнозируемые значения нелинейной модели;
`nnls` — метод МНК, возвращающий только неотрицательные значения параметров (встроенная функция MATLAB).

В качестве примера рассмотрим одну из функций — `nlinfit`. Она записывается в виде:

```
[beta,r,J] = nlinfit(X,y,'model',beta0)
```

Аргументы функции: `X`, `y` — исходные наборы экспериментальных данных; `'model'` — задаваемая пользователем функция вида $y = f(\text{beta}, x)$; `beta0` — начальное значение вектора параметров модели `beta`. Возвращаемые значения: `beta` — вектор параметров модели; `r` — остатки модели; `J` — якобиан (для использования в функции `nlintool`).

5.4.11. Проверка гипотез

Следующие шесть функций служат для проверки статистических гипотез по критериям:

`ranksum` — Вилкоксона для проверки однородности двух генеральных совокупностей;

`signrank` — знаковому критерий Вилкоксона для проверки гипотезы о равенстве медиан двух выборок;

`signtest` — проверки гипотезы о равенстве медиан двух выборок;

`ztest` — проверки гипотезы о том, что математическое ожидание и дисперсия выборки с нормальным законом распределения равны заданным величинам;

`ttest` — проверки гипотезы о том, что математическое ожидание выборки с нормальным законом распределения равно заданному значению (дисперсия неизвестна);

`ttest2` — проверки гипотезы о равенстве математических ожиданий двух выборок при неизвестных, но одинаковых дисперсиях (предполагается нормальный закон распределения).

Следующий пример иллюстрирует применение функции `ranksum`.

```
>> x = poissrnd(5,10,1);
>> y = poissrnd(2,20,1);
>> [p,h] = ranksum(x,y,0.05)
p =
0.0010
h =
1
```

В данном случае возвращаемые величины: `p` — вероятность того, что выборки однородны, `h` — индикатор принятия гипотезы (его значение, равное 1, говорит о том, что нуль-гипотезу об однородности выборок следует отвергнуть при заданном уровне значимости 0.05).

5.4.12. Многомерные статистики

Многомерные статистики представлены двумя функциями:

```
d = mahal(Y,X) и class = classify(sample,training,group)
```

Функция `mahal` возвращает матрицу, элементами которой являются расстояния Махаланобиса от векторов (строк) матрицы `Y` до векторов (строк) матрицы `X`. Число строк данных матриц может быть различным, но число столбцов должно совпадать; число строк матрицы `X` должно превышать число столбцов. Расстояние

Махаланобиса часто используется как мера близости (сходства) объектов в задачах кластеризации и распознавания образов.

Функция class реализует процедуру линейного дискриминантного анализа. Она имеет следующие аргументы: sample — матрица анализируемых объектов (векторов-строк); training — матрица, вектор-строки которой разбиты на несколько классов; число столбцов этой матрицы совпадает с числом столбцов матрицы sample и group — вектор, элементы которого указывают на принадлежность каждого вектора матрицы training к определенному классу; число его элементов (строк) должно совпадать с числом строк матрицы training. Возвращаемая величина — вектор class с числом элементов, равным числу строк матрицы sample. Элементы этого вектора указывают на принадлежность объектов к тому или иному классу.

5.4.13. Метод главных компонент

Для реализации метода главных компонент служат три функции. Функция

```
pc = pcacov(X) [pc,latent,explained] = pcacov(X)
```

реализует метод главных компонент по заданной матрице ковариаций. Аргумент X — матрица ковариаций экспериментальных данных. Возвращаемые величины: pc — матрица, образованная векторами — главными компонентами, latent — вектор, элементами которого являются собственные числа X, соответствующие данным компонентам, и explained — вектор, элементами которого являются процентные доли общей дисперсии, объясняемые главными компонентами.

Пример применения функции pcacov:

```
>> load hald; >> covx = cov(ingredients);
>> [pc,variances,explained] = pcacov(covx)
pc =
   0.0678   -0.6460   0.5673   -0.5062
   0.6785   -0.0200   -0.5440  -0.4933
   0.0290   0.7553    0.4036   -0.5156
   0.7309   -0.1085   -0.4684  -0.4844
variances =
517.7969
67.4964
12.4054
0.2372
explained =
86.5974
11.2882
2.0747
0.0397
```

Функция residuals = pcares(X,ndim) подобна предыдущей, но реализует метод главных компонент по исходной матрице данных. Она возвращает остаток после удаления ndim главных компонент. Здесь X — исходная матрица данных, величина ndim не должна превышать числа столбцов этой матрицы.

Функция barttest реализует тест Бартлетта определения размерности (числа главных компонент) для объяснения неслучайных вариаций в исходных экспериментальных данных.

5.4.14. Статистические графики

Ряд функций пакета Statistics Toolbox позволяет графически отображать различные вероятностные характеристики. Список функций приведен в табл. 5.4.

Таблица 5.4

Функции статистических графиков

Имя функции	Реализуемая операция
boxplot	Построение графика в виде «ящика с усами» (box and whisker plot)
fsurfht	Интерактивная операция построения контурного графика заданной функции
gline	Рисование линии в текущей графической фигуре
gname	Нанесение меток на график
lsline	Нанесение линии, наиболее близкой (в смысле МНК) к точкам графика
normplot	Нормальный вероятностный график
qqplot	График типа «квантиль-квантиль» для двух выборок
refcurve	Наносит полиномиальную кривую на текущий график
refline	Наносит прямую линию на текущий график
surfht	Контурный график по матрице данных
weibplot	Вероятностный график Вейбулла

Пример использования функции boxplot:

```
>> x = normrnd(6,1,100,1); boxplot(x)
```

представлен на рис. 5.31.

На графике нижняя сторона «ящика» соответствует 25-процентной точке, верхняя — 75-процентной, горизонтальная линия внутри «ящика» — медиане, нижний «ус» — минимальному элементу выборки, верхний — ее максимальному элементу.

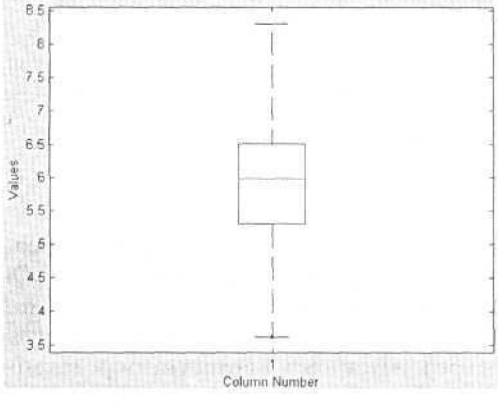

Рис. 5.31. График в виде «ящика с усами»

Пример использования функции lsline:

```
>> y = [2 3.4 5.6 8 11 12.3 13.8 16 18.8 19.9]';
>> plot(y,'+'); lsline;
```

Результат использования функции иллюстрируется рис. 5.32.

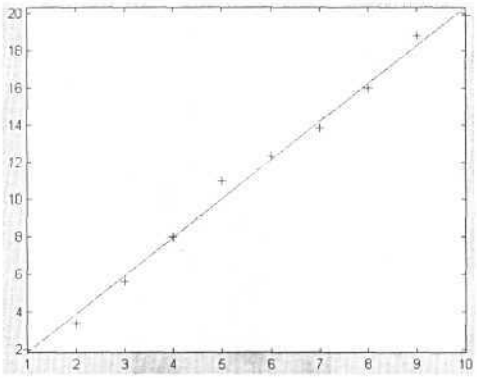

Рис. 5.32. Результат использования функции lsline

Пример на использование еще одной функции этой группы — qqplot представлен ниже:

```
>> x = poissrnd(10,50,1); y = poissrnd(5,100,1);
>> qqplot(x,y);
```

График, возвращаемый данной функцией (рис. 5.33), дает ответ на вопрос, принадлежат ли две выборки к одному и тому же (но обязательно гауссовому) распределению.

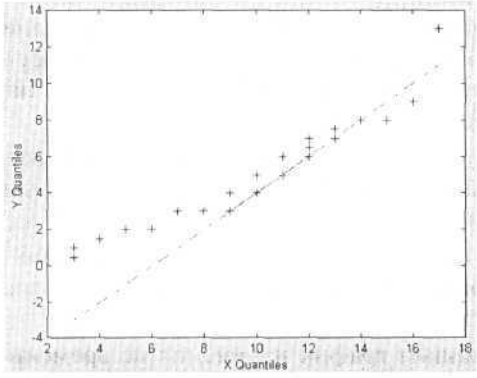

Рис. 5.33. Результат использования функции qqplot

На графике крестиками обозначены точки, соответствующие одинаковым процентилям обеих выборок. Сплошной линий соединены 25-процентные и 75-процентные процентили. Пунктирная линия является продолжением данной сплошной. Чем ближе все точки графика к данным прямым, тем с большей уверенностью можно утверждать, что выборки соответствуют одинаковому закону распределения. В рассматриваемом случае, несмотря на различия в параметрах закона и в числе элементов выборок, данное утверждение может быть принято.

5.4.15. Статистический контроль в промышленности

Функции статистического контроля качества продукции описаны в литературе (см., например, книгу [109]). Функция `histfit` строит гистограмму, построенную по элементам выборки с наложенной на нее кривой плотности вероятности нормального закона распределения. Пример применения этой функции:

```
>> r = normrnd(10,1,100,1); histfit
```

Рис. 5.34 отображает результат этого примера.

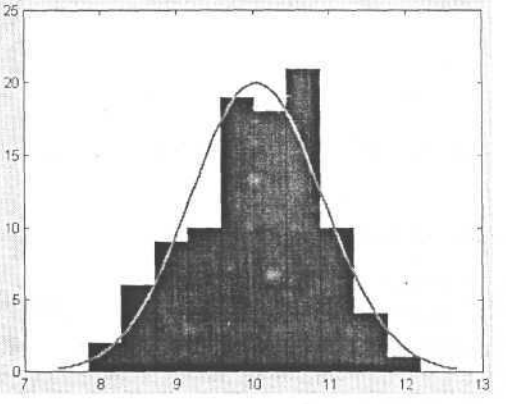

Рис. 5.34. Результат использования функции histfit

Отметим еще ряд функций этой группы:

- `capable` — возвращает оценку вероятности нахождения элементов выборки вне заданного интервала, а также оценку дисперсии, реализуя технику контрольных карт;
- `capaplot` — функция «подгоняет» имеющиеся экспериментальные данные под нормальный закон распределения, возвращает соответствующий график и вероятность того, что любое последующее наблюдение находится в заданном интервале;
- `normspec` — функция возвращает график плотности распределения нормального закона с заданными параметрами и для заданного диапазона изменения аргумента;
- `ewmaplot` — возвращает график экспоненциально сглаженных значений исходной выборки, которые рассматриваются как значения дискретной последовательности;
- `schart` — возвращает график изменения во времени среднеквадратического отклонения изучаемой последовательности, представленной выборкой данных; разновидность контрольной карты;
- `xbarplot` — функция аналогична предыдущей, но относительно среднего значения.

Ниже дан пример на применение функции `capaplot` (рис. 5.35):

```
>> data = normrnd(1,1,30,1); p = capaplot(data,[-3 3])
p =
0.9948
```

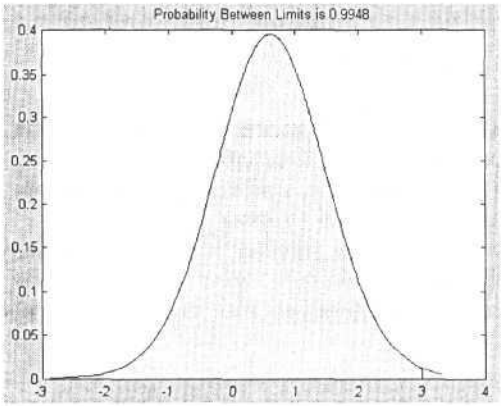

Рис. 5.35. График, возвращаемый функцией capaplot

5.4.16. Планирование эксперимента

Зачастую можно обеспечить существенную экономию временных и финансовых затрат, организуя активный изучающий (идентифицирующий) эксперимент на объекте, заранее планируя его некоторым оптимальным образом (см. книгу [110]). Для быстрого, автоматического нахождения такого наилучшего плана предназначены функции рассматриваемой группы.

`ff2n` — формирует план полного факторного эксперимента при факторах, каждый из которых задан на отрезке.

Пример использования (для трех входных факторов):

```
>> ff2n(3)
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
```

Возвращаемая величина — матрица плана эксперимента.

`fullfact` — формирует план полного факторного эксперимента при произвольном (задаваемом) числе уровней факторов.

Пример (для двух входных факторов, каждый из которых имеет по три уровня):

```
>> d = fullfact([3 3])
d =
1     1
2     1
3     1
1     2
2     2
3     2
```

```
1     3
2     3
3     3
```

Здесь возвращена матрица плана эксперимента d, каждая строка которой соответствует одному из опытов плана, а столбцы — факторам. Например, четвертая строка матрицы соответствует опыту, в котором 1-й фактор должен располагаться на своем первом уровне, а 2-й — на втором.

hadamard — данная функция возвращает матрицу Адамара, которая соответствует плану дробного факторного эксперимента в случае факторов, каждый из которых задан на отрезке [−1, 1] и построения линейной регрессионной модели.

Пример:

```
>> X = hadamard(8)
X =
  1     1     1     1     1     1     1     1
  1    -1     1    -1     1    -1     1    -1
  1     1    -1    -1     1    -1    -1    -1
  1    -1    -1     1     1    -1    -1     1
  1     1     1     1    -1    -1    -1    -1
  1    -1     1    -1    -1     1    -1     1
  1     1    -1    -1    -1    -1     1     1
  1    -1    -1     1    -1     1     1    -1
```

Следующие четыре функции связаны с построением так называемых D-оптимальных планов эксперимента:

rowexch — функция нахождения точного D-оптимального плана эксперимента.

Запись:

```
settings = rowexch(nfactors,nruns)
[settings,X] = rowexch(nfactors,nruns)
[settings,X] = rowexch(nfactors,nruns,'model')
```

Аргументы: nfactors — число факторов, nruns — число опытов (число строк матрицы плана эксперимента), 'model' — строковая переменная, задающая тип регрессионной модели и принимающая одно из следующих возможных значений: 'interaction' (неполная квадратичная модель), 'quadratic' (в модель включаются взаимодействия факторов и квадратичные члены) и 'purequadratic' (в модель включаются константа, линейные и квадратичные члены).

Возвращаемые величины: settings — матрица изменений уровней факторов, X — полная матрица эксперимента.

Пример:

```
>> [s,X]=rowexch(3,8,'interaction')
s =
    1     1    -1
    1    -1    -1
    1    -1     1
    1     1     1
    1     1     1
    1     1    -1
    1    -1     1
    1    -1    -1
```

```
X =
     1    -1     1    -1    -1     1    -1
     1     1    -1    -1    -1    -1     1
     1     1    -1     1    -1     1    -1
     1     1     1     1     1     1     1
     1    -1     1     1    -1    -1     1
     1     1     1    -1     1    -1    -1
     1    -1    -1     1     1    -1    -1
     1    -1    -1    -1     1     1     1
```

В условиях примера задано 3 фактора, 8 опытов, постулируемая регрессионная модель содержит константу, три линейных слагаемых, два парных произведения факторов и одно тройное произведение. В частности, в первом опыте 1-й фактор должен быть установлен на нижнем уровне, 2-й — на верхнем, 3-й — на нижнем и т. п.

`cordexch` — функция нахождения точного D-оптимального плана эксперимента, практически аналогична предыдущей за исключением алгоритма нахождения плана.

Пример: нахождение матрицы плана для двухфакторной квадратичной модели

```
>> settings = cordexch(2,9,'quadratic')
settings =
     1     1
     1    -1
    -1    -1
     0    -1
     1     0
     0     0
    -1     1
     0     1
    -1     0
```

`daugment` — функция возвращает матрицу плана, дополняющую матрицу заданного плана до D-оптимального.

`dcovary` — функция для построения D-оптимальных блочных планов.

5.4.17. Демонстрационные примеры

Для ознакомления с некоторыми возможностями пакета Statistics Toolbox можно использовать следующие функции (команды):

`disttool` — возвращает графический интерфейс пользователя, позволяющий выводить графики функций плотности вероятности и функций распределения вероятностей для 19 теоретических законов распределения (рис. 5.36).

`randtool` — возвращает графический интерфейс пользователя для генерации случайных чисел с заданным законом распределения и построения соответствующей гистограммы (рис. 5.37).

`polytool` — возвращает графический интерфейс пользователя, позволяющий подбирать аппроксимирующую кривую (полином) для заданных экспериментальных данных.

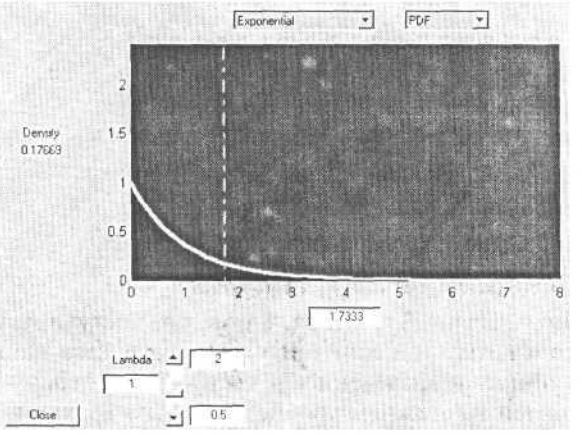

Рис. 5.36. Графический интерфейс функции disttool

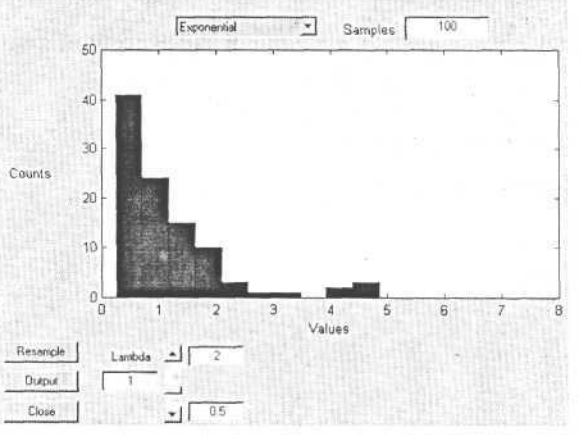

Рис. 5.37. Графический интерфейс функции randtool

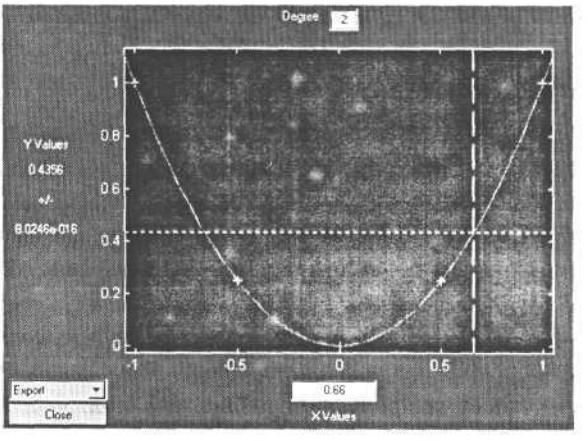

Рис. 5.38. Графический интерфейс функции polytool

Пример:

```
>> x=[-1; -0.5; 0; 0.5; 1]; y=[1; 0.25; 0; 0.25; 1];
>> polytool(x,y)
```

Рис. 5.38 иллюстрирует результат использования функции в условиях примера.

rsmdemo — функция моделирования функционирования и построения (в режиме графического интерфейса) нелинейной регрессионной модели химической реакции.

5.4.18. Функции записи/чтения файлов данных

В данную группу входит несколько функций. Функция tblread обеспечивает чтение данных в табличном формате:

```
[data,varnames,casenames] = tblread
[data,varnames,casenames] = tblread(`filename`)
[data,varnames,casenames] = tblread(`filename`,`delimiter`)
```

Аргументы функции: `filename` — имя файла данных, `delimiter` — строковая переменная, принимающая значения `tab` (разделители — табуляторы), `space` (разделители — пробелы) или `comma` (разделители — запятые). Возвращаемые величины: data — числовая матрица данных, varnames — вектор с именами переменных, casenames — вектор с именами строк (опытов или случаев).

Пример применения этой функции:

```
>> [data,varnames,casenames] = tblread(`sat.dat`)
data =
       470     530
       520     480
varnames =
Male
Female
casenames =
Verbal
Quantitative
```

В данном случае матрица данных отображает две переменных с именами Male и Female и два «опыта» или «случая» с именами Verbal и Quantitative.

Функция записи данных в файл tblwrite является «обратной» к предыдущей функции. Использование данной функции проиллюстрируем продолжением предыдущего примера:

```
>> tblwrite(data,varnames,casenames,`sattest.dat`)
>> type sattest.dat
Male Female
Verbal          470     530
Quantitative    520     480
```

Функция чтения данных из файла caseread возвращает матрицу данных вместе с соответствующими именами. Пример ее применения:

```
>> caseread(`sattest.dat`)
ans =
Male Female
```

```
Verbal          470     530
Quantitative    520     480
```

Функция записи данных в файл вместе с соответствующими именами `casewrite` является обратной к предыдущей.

5.5. Новые средства пакета Statistic Toolbox 5.0

5.5.1. Инструмент Distribution Fitting Tool

В новейшую версию пакета Statistic Toolbox введен ряд новых средств для статистической обработки данных. Прежде всего, это новый инструмент дистрибутивного приближения Distribution Fitting Tool. Для вывода окна его GUI-интерфейса достаточно исполнить команду:

```
>> dfittool
```

В результате ее исполнения появится окно интерфейса, показанное на рис. 5.39. Поначалу оно пустое, поскольку нужных для применения этого средства данных не было введено. Эти данные необходимо создать в рабочем пространстве системы MATLAB. На рис. 5.39 представлен открытый список отображаемых зависимостей. Все они были описаны выше и с помощью этого инструмента могут быть построены для выбранных данных.

Для иллюстрации работы с инструментом исполним в командном режиме MATLAB следующую команду, создающую вектор из 1000 чисел с нормальным распределением:

```
>> data = normrnd(.36, 1.4, 1000, 1);
```

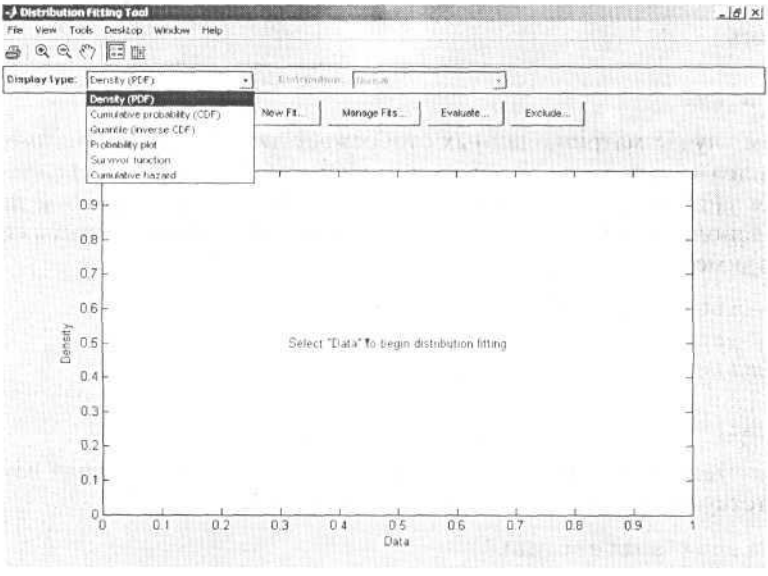

Рис. 5.39. Графический интерфейс инструмента Distribution Fitting Tool

Теперь, нажав кнопку Data... (на рис. 5.39 она прикрыта списком), можно вывести окно ввода и предварительного просмотра данных (рис. 5.40). В окне Data Preview можно просмотреть гистограмму распределения данных, а с помощью кнопки View вывести окно с более подробными данными о созданном векторе. Это окно представлено на рис. 5.40 внутри окна инструмента Distribution Fitting Tool.

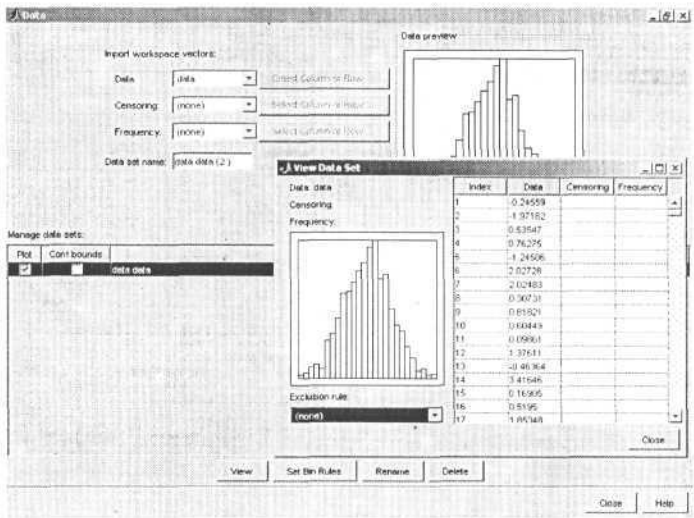

Рис. 5.40. Окно ввода и предварительного просмотра интерфейса инструмента Distribution Fitting Tool

Завершив работу с окном ввода, можно просмотреть одну из графических характеристик массива данных. Для созданного массива data на рис. 5.41 построена гистограмма распределения вероятности данных — та же, что и окне предварительного просмотра окна ввода.

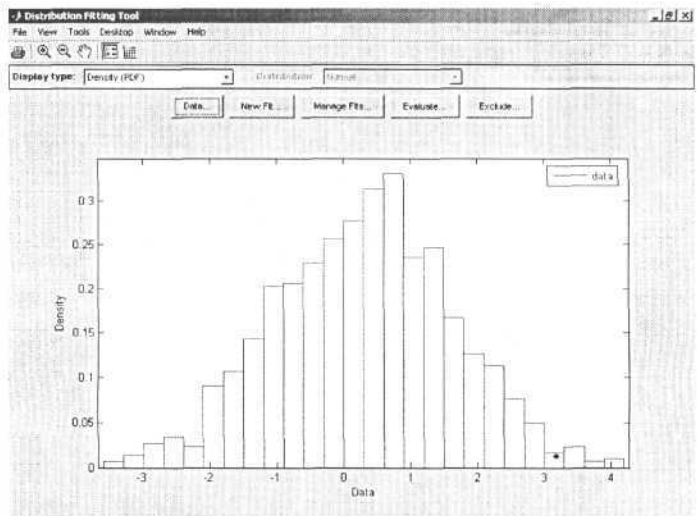

Рис. 5.41. Окно инструмента Distribution Fitting Tool с построенной гистограммой распределения

Теперь можно выполнить приближение данных гистограммы по методу наименьших квадратов. Для этого надо активизировать кнопку New Fit... Появится окно Edit Fit, в котором надо выполнить ряд очевидных установок. Работу с этим окном и построение части построенной кривой можно наблюдать на рис. 5.42.

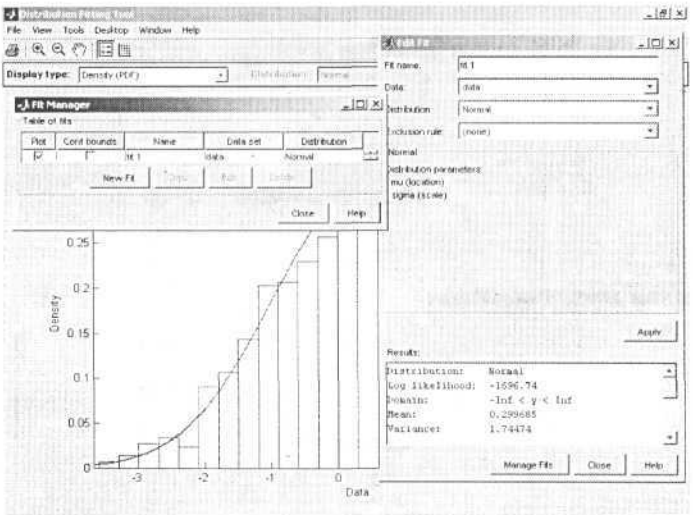

Рис. 5.42. Построение кривой приближения для распределения данных

На рис. 5.43 показано построение кривой кумулятивного распределения, кривой приближения и границ. Это иллюстрирует еще один тип представления результатов анализа вектора data. С другими типами представления и иными возможностями инструмента Distribution Fitting Tool читатель может ознакомиться самостоятельно.

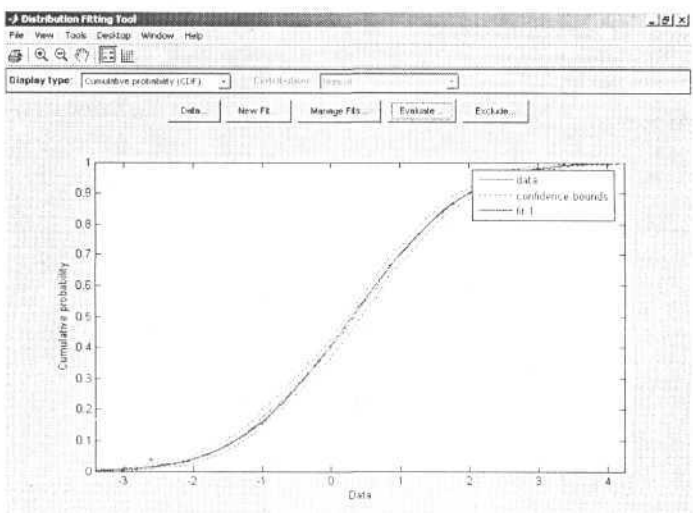

Рис. 5.43. Построение кривой приближения и границ для кумулятивного распределения данных

5.5.2. Новые функции пакета Statistic Toolbox 5.0

В пакет Statistic Toolbox 5.0 включен ряд новых функций. Их состав и описание можно найти в разделе справки по новым возможностям пакета (рис. 5.44).

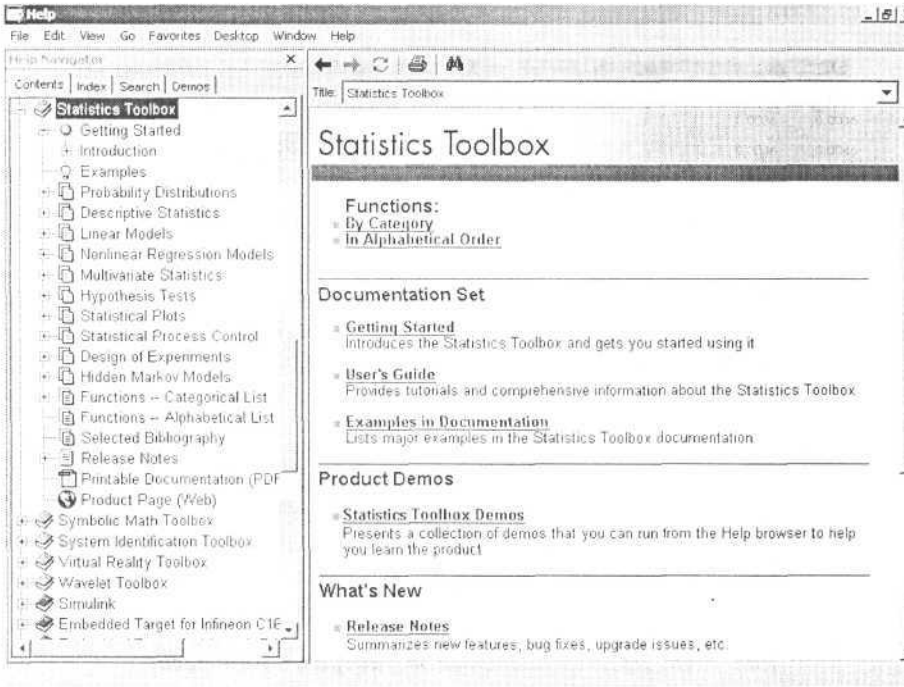

Рис. 5.44. Раздел справки с описанием новых функций пакета Statistic Toolbox 5.0

Большинство этих функций служит для построения специальных статистических графиков и диаграмм. Например, функция andrewsplot строит диаграммы Андреуса. Пример применения этой функции дан ниже:

```
% make a grouped plot of the raw data
load fisheriris
andrewsplot(meas, 'group',species);
% plot only the median and quartiles of each group
andrewsplot(meas, 'group',species, 'quantile',.25);
```

Построенные по этому примеру диаграммы представлены на рис. 5.45. Примеры на применение других новых функция этого пакета можно найти в справке по соответствующим функциям.

5.5.3. Другие новые возможности пакета Statistic Toolbox 5.0

Среди других новых возможностей пакета Statistic Toolbox 5.0 стоит отметить следующие возможности:

- поддержка у ряда функций многомерного ввода;
- 14 новых функций дистрибутивной статистики;

- 19 новых функций генерации массивов случайных чисел;
- расширенные возможности функции mle;
- расширенные возможности функций mad, pdist, ncx2rnd и princoord.

Детальную информацию об этих функциях можно получить из справки по ним.

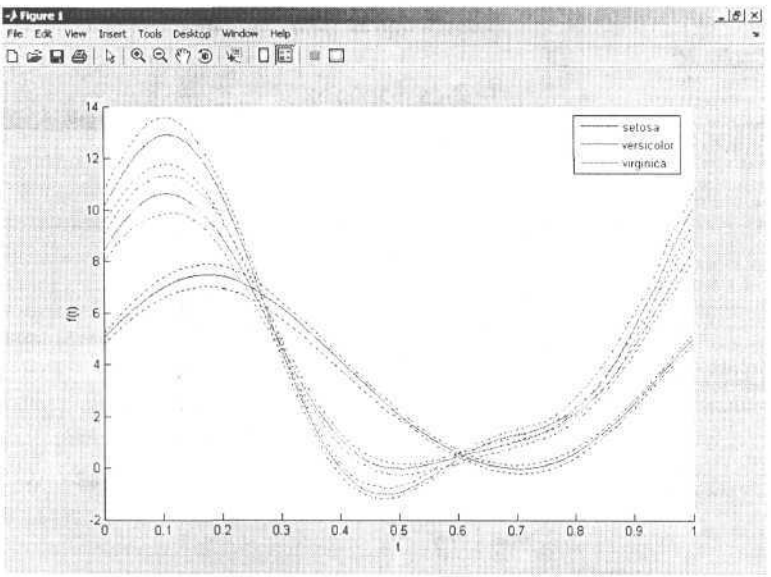

Рис. 5.45. Пример построения диаграмм Андреуса

Глава 6. Анализ и идентификация систем

6.1. Теоретическая преамбула

6.1.1. Понятие о моделировании и моделях

Начиная с этой главы мы вплотную приступаем к изучению возможностей *математического моделирования* в среде MATLAB + Simulimk с пакетом анализа и идентификации систем System Identification Toolbox. Теоретические основы идентификации и математического моделирования систем можно найти в литературе, например в [55—63]. Ниже они даны обзорно.

Моделирование это средство для изучения процессов, протекающих в различных системах и устройствах без их создания «в натуре» или «в железе». Нередко это ведет к существенному удешевлению и ускорению этапа проектирования систем и устройств, а в ряде случаев (моделирование ядерных, энергетических и космических устройств) к сохранению человеческих жизней и устранению экологического вреда.

Моделирование используется также для управления реальными системами и устройствами. При этом под *объектом управления* понимается часть окружающего нас мира, поведение которой нас интересует и на которую мы можем целенаправленно воздействовать, то есть управлять ею. В систему MATLAB + Simulink уже введены средства для управления реальными устройствами и системами в реальном масштабе времени. К сожалению, весьма дорогие и потому у нас пока мало распространенные.

Объекты моделирования (и управления) принято классифицировать как:

- статические или динамические;
- линейные или нелинейные;
- непрерывные или дискретные;
- стационарные или нестационарные;
- с сосредоточенными или распределенными параметрами.

Под *моделью* обычно понимается информация о наиболее существенных характеристиках объекта, выраженная в той или иной форме. По способу представления данной информации выделяют следующие типы моделей:

- *словесные* или *вербальные* модели;
- *физические модели* (уменьшенные копии реальных объектов или объектов иной физической природы, позволяющие имитировать процессы в исследуемом объекте);
- *математические модели* (информация об исследуемом объекте или системе представляется в виде математических терминов);
- *формальные модели* (основанные на формальной аналогии с описываемыми объектами).

В свою очередь, математические модели делятся на графические, табличные, топологические, алгоритмические и аналитические. В частности, аналитические модели представляет собой отражение взаимосвязей между переменными объекта в виде математической формулы или группы таких формул. Моделирование основано на ограниченности числа фундаментальных законов природы и принципе

подобия, означающем, что явления различной физической природы могут описываться одинаковыми математическими зависимостями.

6.1.2. Идентификация моделей и объектов моделирования

Процедуру построения модели принято называть *идентификацией*, при этом данный термин обычно относится к построению аналитических математических моделей динамических объектов.

Динамический объект — это объект, выход которого зависит не только от текущего значения входных сигналов, но и от их значений в предыдущие моменты времени. Идентифицируемый объект принято представлять в виде, показанном на рис. 6.1, где t — время; $u(t)$ — контролируемый (иногда управляемый) входной сигнал; $\tilde{y}(t)$ — теоретический выход объекта; $y(t)$ — наблюдаемый выход объекта; $e(t)$ — аддитивная случайная помеха, отражающая действие неучитываемых факторов (шум наблюдения).

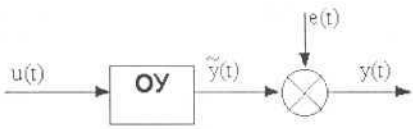

Рис. 6.1. Общее представление идентифицируемого объекта О

Обычно предполагают, что связь между входным и «теоретическим» выходным сигналами задается в виде некоторого оператора ψ *(оператор* — правило преобразования какой-либо функции в другую функцию):

$$\tilde{y}(t) = \Psi[u(t)],$$

при этом наблюдаемый выход объекта может быть описан соотношением

$$y(t) = \Psi\ [u(t)] + e(t).$$

Цель идентификации: на основании наблюдений за входным $u(t)$ и выходным $y(t)$ сигналами на каком-то интервале времени определить вид оператора, связывающего входной и теоретический выходной сигналы.

При экспериментальном определении параметров модели необходимо обеспечить подбор адекватной структуры модели и выбор такого входного сигнала, чтобы по результатам эксперимента можно было найти оценки всех параметров модели. Наиболее просто задача определения параметров решается для *линейных* объектов, для которых выполняется принцип суперпозиции. Здесь можно выделить два случая:

- объект линеен по входному воздействию:

$$\tilde{y}(t) = \Psi[\beta, u_1(t) + u_2(t)] = \Psi[\beta, u_1(t)] + \Psi[\beta, u_2(t)] = \tilde{y}_1(t) + \tilde{y}_2(t);$$

- объект линеен по параметрам:

$$\tilde{y}(t) = \Psi[\beta_1 + \beta_2,\ u(t)] = \Psi[\beta_1, u(t)] + \Psi[\beta_2, u(t)] = \tilde{y}_1(t) + \tilde{y}_2(t).$$

В задачах идентификации под линейными объектами чаще понимают объекты, линейные по входному воздействию.

В свете этого под *идентификацией динамических объектов* понимают процедуру определения структуры и параметров их математических моделей, которые при

одинаковых входном сигнале объекта и модели обеспечивают близость выхода модели к выходу объекта при наличии какого-то критерия качества.

Идентификация — многоэтапная процедура и содержит следующие этапы:

- *структурная идентификация* — определение структуры математической модели на основании теоретических соображений;
- *параметрическая идентификация* — проведение идентифицирующего эксперимента и определение оценок параметров модели по экспериментальным данным;
- *проверка адекватности* — проверка качества модели по критерию близости выходов модели и объекта.

Классификация задач идентификации показана на рис. 6.2.

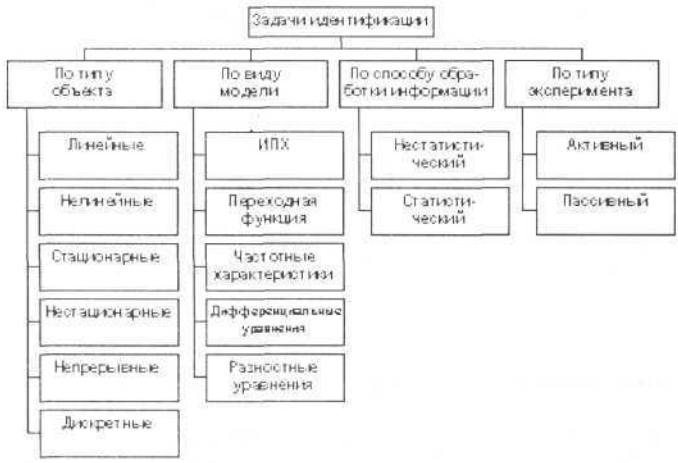

Рис. 6.2. Классификация задач идентификации

Дальнейшее изложение будет относиться к линейным стационарным динамическим объектам.

6.2. Основные характеристики (функции) систем

В целом ряде случаев реакция линейной стационарной системы на то или иное достаточно простое воздействие определяется ее теоретическими характеристиками (функциями). Рассмотрим основные из них.

6.2.1. Передаточная характеристика

Передаточная характеристика определяется с помощью преобразования Лапласа как

$$W(p) = \frac{L\{y(t)\}}{L\{u(t)\}} = \frac{Y(p)}{U(p)} = \frac{\sum_{j=0}^{nb} b_j p^j}{\sum_{i=0}^{na} a_i p^i},$$

где $L\{\bullet\}$ — символ преобразования Лапласа, p — комплексная переменная.

6.2.2. Импульсная характеристика

Импульсная характеристика (ИХ) $w(t)$ это реакция предварительно невозмущенного объекта (то есть объекта с нулевыми начальными условиями) на входной сигнал в виде δ-функции (импульс с единичной площадью с бесконечно малой длительностью).

6.2.3. Переходная функция

Переходная характеристика $h(t)$ это реакция предварительно невозмущенного объекта на входной сигнал в виде единичного скачка. Из теории управления известны следующие соотношения между этими характеристиками:

$$L\{w(t)\} = W(p),\ w(t) = h'(t),\ L\{h9t)\} = \frac{W(p)}{p}.$$

При нулевых начальных условиях связь между выходным и входным сигналами описывается интегралом свертки:

$$y(t) = \int_{-\infty}^{+\infty} w(t-\tau)u(\tau)d\tau$$

или, в операторной форме:

$$Y(p) = W(p) \cdot U(p).$$

6.2.4. Частотные характеристики

Частотные характеристики объекта определяются его комплексным коэффициентом передачи $W(j\omega) = W(p)|_{p=j\omega}$, который является фурье-преобразованием ИХ.

Модуль комплексного коэффициента передачи $|W(j\omega)| = A(\omega)$ представляет собой амплитудно-частотную характеристику (АЧХ) объекта с передаточной функцией $W(p)$, а аргумент $\arg(W(j\omega)) = \varphi(\omega)$ — фазочастотную характеристику (ФЧХ).

Графическое представление $W(j\omega)$ на комплексной плоскости при изменении частоты ω от 0 до ∞, то есть график амплитудно-фазовой характеристики (АФХ) в полярных координатах в отечественной литературе называется *годографом*, а в англоязычной — *диаграммой Найквиста*. В теории управления часто используется логарифмическая амплитудно-частотная характеристика (ЛАЧХ), равная $20 \lg |W(j\omega)|$.

6.3. Теоретические модели объектов

Рассмотрим основные виды теоретических моделей линейных непрерывных стационарных динамических объектов и их взаимосвязь (действием шума $e(t)$ пока пренебрегаем).

6.3.1. Дифференциальные уравнения

Наиболее универсальная модель основанная на дифференциальных уравнениях, описывается выражением:

$$\sum_{i=0}^{na} a_i y^{(i)}(t) = \sum_{j=0}^{nb} b_j u^{(j)}(t),$$

где na — порядок модели ($na > nb$), a_i и b_j — постоянные коэффициенты (параметры модели), $u^{(j)}(t)$ и $y^{(i)}(t)$ — производные соответственно входного и выходного сигналов.

6.3.2. Уравнения переменных состояния

При выборе n координат системы (объекта) в качестве переменных ее состояния (такими координатами, например, могут быть выходной сигнал $y(t)$ и $n - 1$ его производных) $x_i(t)$, $i = 1, 2, ..., n$, данную систему можно описать уравнениями для переменных состояния

$$\mathbf{X}'(t) = \mathbf{A}\mathbf{X}(t) + \mathbf{B}u(t),$$

$$y(t) = \mathbf{C}\mathbf{X}(t) + \mathbf{D}u(t),$$

где $\mathbf{X}(t) = [x_1(t), x_2(t), ..., x_n(t)]^{\mathrm{T}}$ — вектор-столбец переменных состояния; **A**, **B**, **C** и **D** при скалярных $u(t)$ и $y(t)$ — соответственно матрица размера $n \times n$, векторы размера $n \times 1$ и $1 \times n$ и скаляр (при векторных u(t) и y(t) — матрицы соответствующих размеров).

Применение, при нулевых начальных условиях, к последним уравнениям преобразования Лапласа позволяет получить следующее выражение для передаточной функции:

$$W(p) = \mathbf{C}(p\mathbf{I} - \mathbf{A})^{-1}\,\mathbf{B} + \mathbf{D},$$

где **I** — единичная матрица. Отметим, что все приведенные модели являются эквивалентными, то есть, зная любую из них, можно получить все остальные.

6.3.3. Разностные уравнения

Для объектов, функционирование которых представляется для дискретного времени $t_k = kT$ (в данном случае T — интервал дискретизации), то есть для дискретных объектов, наиболее общим видом описания является разностное уравнение (аналог дифференциального)

$$y_k + a_1 y_{k-1} + ... + a_{na} y_{k-na} = b_1 u_k + b_2 u_{k-1} + b_3 u_{k-2} + ... + b_{nb} u_{k-nb+1},$$

где $y_{k-i} = y[(k - i)T]$, $u_{k-j} = u[(k - j)T]$.

6.3.4. Z-преобразование

Связь между сигналами может быть отражена также через дискретную свертку

$$y_k = \sum_{i=0}^{k} w_i u_{k-i},$$

где w_i — ординаты весовой решетчатой функции объекта, или, с использованием аппарата Z-преобразования

$$Y(z) = \sum_{k=0}^{\infty} y_k z^{-k},$$

где $z = e^{pT}$, через дискретную передаточную функцию

$$W(z) = \frac{Y(z)}{U(z)} = \frac{B(z)}{A(z)},$$

которая определяется на основании разностного уравнения после применения к обеим частям этого уравнения Z-преобразования:

$$(1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_{na} z^{-na}) Y(z) = (b_1 + b_2 z^{-1} + b_3 z^{-2} + \ldots + b_{nb} z^{-nb+1}) U(z).$$

Заметим, что Z-изображением решетчатой импульсной переходной характеристики является $W(z)$, то есть $Z\{w_i\} = W(z)$.

Отметим далее, что на практике в большинстве случаев измерение непрерывных сигналов производится в дискретные моменты времени, что представляет определенное удобство при последующей обработке данных на ЭВМ. Непрерывные объекты можно, хотя бы приближенно, отображать дискретными моделями. При этом возможны различные способы перехода от непрерывных моделей к дискретным:

- с применением Z-преобразования со следующей цепочкой переходов:

$$W(p) \to L^{-1}\{W(p)\} = w(t) \to w(kT) = w_k \to W(z) = Z\{w_k\};$$

- с заменой производных в дифференциальном уравнении, описывающем непрерывный объект, разностями:

$$\frac{dy(t)}{dt} \approx \frac{y_k - y_{k-1}}{T}; \quad \text{и т. д.}$$

(данный подход дает приемлемую точность только при малых T);

- с заменой $p = \dfrac{2}{T} \cdot \dfrac{z-1}{z+1}$ (приближенный способ, предложенный А. Тастиным и называемый *билинейным преобразованием*), то есть

$$W(p)\Big|_{p=\frac{2}{T}\frac{z-1}{z+1}} \to W(z).$$

Укажем, что множитель $z^{-1} = e^{-pT}$ представляет собой *оператор задержки*, то есть $z^{-1}u_k = u_{k-1}$, $z^{-2}u_k = u_{k-2}$ и т. д. Принимая во внимание данное обстоятельство и обозначая моменты дискретного времени тем же символом t, что и непрерывное время (в данном случае $t = 0, 1, 2, \ldots$), приведем и опишем ниже несколько распространенных моделей дискретных объектов для временной области, учитывающих действие шума наблюдения.

6.3.5. Модели авторегрессии

Модель авторегрессии AR (AutoRegressive) — считается самым простым описанием:

$$A(z)\, y(t) = e(t),$$

где

$$A(z) = 1 + a_1 z^{-1} + a_2 z^{-2} + \ldots + a_{na} z^{-na}.$$

ARX-модель (AutoRegressive with eXternal input) — более сложная:

$$A(z)\, y(t) = B(z)\, u(t) + e(t),$$

или, в развернутом виде,

$$y(t) + a1\,y(t-1)+\ldots+a_{na}y(t-n) = b_1u(t) + b_2u(t-1)+\ldots+b_{nb}u(t-m) + e(t).$$

Здесь и ниже $e(t)$ — дискретный белый шум, $B(z) = b_1 + b_2z^{-1}+\ldots+b_{nb}z^{-nb+1}$.

ARMAX-модель (AutoRegressive-Moving Average with eXternal input — модель авторегрессии скользящего среднего):

$$A(z)\,y(t) = B(z)\,u(t - nk) + C(z)\,e(t),$$

где nk — величина задержки (запаздывания), $C(z) = 1 + c_1z^{-1} + c_2z^{-2}+\ldots+c_{nc}z^{-nc}$.

Модель «вход-выход» (в англоязычных источниках такая модель называется «Output-Error», то есть «выход-ошибка», сокращенно OE):

$$y(t) = \frac{B(z)}{F(z)}u(t-nk) + e(t),$$

где $F(z) = 1 + f_1z^{-1} + f_2z^{-2}+\ldots f_{nf}z^{-nf}$.

Так называемая модель Бокса—Дженкинса (BJ):

$$y(t) = \frac{B(z)}{F(z)}u(t-nk) + \frac{C(z)}{D(z)}e(t)$$

(полиномы $B(z)$, $F(z)$, $C(z)$ определены ранее, а $D(z) = 1 + d_1z^{-1} + d_2z^{-2}+\ldots+d_{nd}z^{-nd}$).

Данные модели можно рассматривать как частные случаи обобщенной параметрической линейной структуры

$$A(z)y(t) = \frac{B(z)}{F(z)}u(t-nk) + \frac{C(z)}{D(z)}e(t),$$

при этом все они допускают расширение для многомерных объектов (имеющих несколько входов и выходов).

6.3.6. Модель для переменных состояния

Модель для переменных состояния (State space):

$$x(t+1) = \mathbf{A}x(t) + \mathbf{B}u(t),$$

$$y(t) = \mathbf{C}x(t) + \mathbf{D}u(t) + v(t),$$

где **A**, **B**, **C**, **D** — матрицы соответствующих размеров, $v(t)$ — коррелированный шум наблюдений.

Возможна и другая (так называемая обновленная или каноническая) форма представления данной модели:

$$x(t+1) = \mathbf{A}x(t) + \mathbf{B}u(t) + \mathbf{K}e(t),$$

$$y(t) = \mathbf{C}x(t) + \mathbf{D}u(t) + e(t),$$

где **K** — некоторая матрица (вектор-столбец), $e(t)$ — дискретный белый шум (скаляр).

6.4. Методы оценивания

В общем случае оценивание параметров модели заданной структуры проводится, как отмечалось, путем минимизации выбранного критерия качества модели

(чаще всего — среднего квадрата рассогласования выходов объекта и его постулируемой модели). Рассмотрим несколько возможных подходов к такому оцениванию.

6.4.1. Оценивание параметрических моделей

Оценивание параметрических моделей (метод прогноза ошибки — Predictive Error Method, сокращенно PEM), заключается в следующем. Пусть модель исследуемого объекта имеет вид так называемой обобщенной линейной модели

$$y(t) = W(z)\, u(t) + v(t),$$

при этом шум $v(t)$ может быть представлен как

$$v(t) = H(z)\, e(t),$$

где $e(t)$ — дискретный белый шум, $H(z)$ — некоторый полином от z.

Из данных выражений следует, что

$$e(t) = H^{-1}(z)\, [y(t) - W(z)\, u(t)].$$

При выборе в качестве критерия (функции потерь) величины

$$V_N(W,\, H) = \sum_{t=1}^{N} e^2(t)$$

оценки коэффициентов полиномов модели могут быть найдены в результате решения следующей оптимизационной задачи (в общем случае нелинейной)

$$[W,\, H] = \arg\min \sum_{t=1}^{N} e^2(t).$$

Нахождение такого решения (различными численными методами нелинейной оптимизации), как правило, достаточно сложно и трудоемко.

Заметим, что еще более сложной является подобная процедура оценивания параметров модели для переменных состояния. Однако для ряда частных моделей существуют методы оценивания более простого вида. Рассмотрим их.

6.4.2. Оценивание импульсной характеристики

Предположим, что входной сигнал исследуемого (дискретного) объекта имеет нулевое математическое ожидание и является дискретным белым шумом, то есть имеет автокорреляционную функцию (АКФ)

$$R_u(\tau) = E\{u(t + \tau)u(t)\} = \begin{cases} \lambda, & \textit{если } \tau = 0 \\ 0, & \textit{в противоположном случае,} \end{cases}$$

где $E\{\bullet\}$ — оператор математического ожидания, $\lambda = \text{const}$ — интенсивность сигнала ($\lambda > 0$), и не коррелирован с шумом наблюдений (так что при любых t и τ справедливо $E\{e(t + \tau)u(t)\} = 0$).

Тогда для установившегося режима, исходя из дискретного аналога уравнения свертки, который запишем в форме

$$y(t) = \sum_{i=0}^{\infty} w(i)u(t - i) + e(t),$$

получим

$$R_{yu}(\tau) = E\{y(t+\tau)u(t)\} =$$

$$= E\{\sum_{i=0}^{\infty} w(i)u(t+\tau-i)u(t)\} + E\{e(t+\tau)u(t)\} = \sum_{i=0}^{\infty} w(i)E\{u(t+\tau-i)u(t)\} = \sum_{i=0}^{\infty} w(i)R_u(\tau-i).$$

Но в силу принятого предположения о виде АКФ входного сигнала в сумме в правой части от нуля отлично только слагаемое, соответствующее $\tau = i$, поэтому окончательно получаем

$$R_{yu}(\tau) = \lambda w(\tau),$$

где $R_{yu}(\tau)$ — взаимная корреляционная функция (ВКФ) выходного и входного сигналов.

Отсюда приходим к оценке ИХ по экспериментальным данным:

$$w(\tau) = \frac{1}{\lambda N}\sum_{t=1}^{N} y(t+\tau)u(t),$$

где сумма в правой части с точностью до множителя представляет собой оценку взаимной корреляционной функции сигналов $y(t)$ и $u(t)$, находимую по выборкам $\{y(t)\}$, $\{u(t)\}$, $t = 1, 2, \ldots, N$.

В случае, когда входной сигнал $u(t)$ является случайным процессом, но не белым шумом, приведенным методом оценивания $w(t)$ можно воспользоваться, если предварительно, с помощью специального формирующего фильтра $\Phi(z)$ (так называемого обеляющего фильтра) преобразовать (хотя бы приближенно) $u(t)$ в сигнал типа белого шума:

$$u_{\Phi}(t) = \Phi(z)u(t).$$

Преобразовав таким же образом $y(t)$, можно воспользоваться (с использованием выборок $\{y_{\Phi}(t)\}$, $\{u_{\Phi}(t)\}$) приведенной выше формулой для нахождения оценки $\hat{w}(t)$.

Заметим, что рассмотренная процедура относится к методам корреляционного анализа.

6.4.3. Оценивание спектров и частотных характеристик

В данном случае объект (дискретный) представляется той же моделью, что и при оценивании параметрических моделей, а входной сигнал полагается случайным процессом с нулевым математическим ожиданием, спектральной плотностью $S_u(\omega)$, некоррелированным с шумом наблюдений $v(t)$, который в данном случае имеет некоторую спектральную плотность $S_v(\omega)$.

Для приведенной модели связи между спектрами сигналов описываются известными соотношениями:

$$S_y(\omega) = |W(e^{j\omega T})|^2\, S_u(\omega) + S_v(\omega),\ S_{yu}(\omega) = W(e^{j\omega T})S_u(\omega),$$

где $S_{yu}(\omega)$ — взаимная спектральная плотность сигналов $y(t)$ и $u(t)$, которые можно использовать для нахождения оценок комплексного коэффициента передачи объекта $W(e^{j\nu T})$ и спектра шума $S_v(\omega)$.

Методика оценивания соответствует описанной ниже.

Сначала по данным эксперимента определяют оценки авто- и взаимной корреляционной функций:

$$R_u(\tau) = \frac{1}{N}\sum_{t=1}^{N} u(t+\tau)u(t),\ R_y(\tau) = \frac{1}{N}\sum_{t=1}^{N} y(t+\tau)y(t),$$

$$R_{yu}(\tau) = \frac{1}{N}\sum_{t=1}^{N} y(t+\tau)y(t);$$

Затем находят оценки спектральных характеристик:

$$S_u(\omega) = \sum_{\tau=-M}^{M} R_u(\tau)g_M(\tau)e^{-j\omega t},\ S_y(\omega) = \sum_{\tau=-M}^{M} R_y(\tau)g_M(\tau)e^{-j\omega t},$$

$$S_{yu}(\omega) = \sum_{\tau=-M}^{M} R_{yu}(\tau)g_M(\tau)e^{-j\omega t},$$

где $g_M(\tau)$ — так называемое временное окно, а M — его ширина;

Наконец, требуемые оценки определяют по соотношениям:

$$W_N(e^{j\omega T}) = \frac{S_{yu}(j\omega)}{S_u(j\omega)},\ S_v(\omega) = S_y(\omega) - \frac{|S_{yu}(j\omega)|}{S_u(j\omega)}.$$

Приведенная процедура относится к методам спектрального анализа.

Замечание. Найденная оценка $W_N(e^{j\omega T})$ комплексного коэффициента передачи является, вообще говоря, чувствительной к небольшим вариациям экспериментальных данных (вызванных, например, ошибками измерений).

6.5. Пакет расширения System Identification Toolbox

6.5.1. Назначение пакета System Identification

Пакет System Identification Toolbox содержит средства для создания математических моделей линейных динамических объектов (систем) на основе наблюдаемых входных/выходных данных. Для этого он имеет удобный графический интерфейс, помогающий организовать данные и создавать модели без использования функций командного режима работы. Точнее говоря, эти функции используются прежде всего для поддержки моделей в среде графического интерфейса GUI.

Ниже рассматриваются возможности последней версии пакета System Identification Toolbox 5.0, входящей в систему MATLAB 6.5 SP1 + Simulink 5. К сожалению, более ранние версии пакета имеют существенно отличный набор функций, так что пользователей ими мы отсылаем к книге [9], где они описаны детально. В MATLAB 7 входит пакет System Identification Toolbox 6.0, который в пределах приведенного ниже рассмотрения практически не отличается от версии 5.0. Желающие подробно ознакомиться с небольшими и непринципиальными отличиями этих версий могут обратиться к разделу справки Release Notes справки пакета System Identification Toolbox 6.0.

Методы идентификации, входящие в пакет, применимы для решения широкого класса задач — от проектирования систем управления и обработки сигналов до анализа временных рядов. Основные свойства пакета следующие:

- простой и гибкий интерфейс;
- предварительная обработка данных, включая фильтрацию, удаление трендов и смещений;

- выбор диапазона данных для анализа;
- эффективные методы авторегрессии;
- возможности анализа отклика систем во временной и частотной областях;
- отображение нулей и полюсов передаточной функции системы;
- анализ невязок при тестировании модели.

Графический интерфейс пакета упрощает как предварительную обработку данных, так и диалоговый процесс идентификации модели. Операции загрузки и сохранения данных, выбора их диапазона, исключения смещений и трендов выполняются с минимальными усилиями и доступны из главного меню.

Представление данных и моделей систем и объектов организовано таким образом, что в процессе интерактивной идентификации пользователь легко может вернуться к предыдущему этапу работы. Для начинающих пользователей существует возможность просматривать последующие этапы. Специалисту графические средства позволяют отыскать любую из ранее полученных моделей и оценить ее качество в сравнении с другими моделями.

Пакет поддерживает все традиционные виды моделей, включая модели передаточных функций, описания для переменных состояний (как для непрерывного, так и для дискретного времени) и другие, с произвольным числом входов и выходов.

В рассматриваемом пакете используются три внутренних вида матричного представления моделей: первый (для временных моделей) — так называемый тета-формат, второй (для частотных характеристик) — частотный формат и третий — формат нулей и полюсов.

Адекватность моделей устанавливается применением таких критериев, как остаточная сумма квадратов ошибки — функция потерь (Loss fcn) — и так называемый теоретический информационный критерий Акейке (Akaike's Information Theoretic Criterion — AIC) — см. ниже.

6.5.2. Графический интерфейс System Identification Toolbox

Удобный графический интерфейс пакета System Identification запускается из режима командной строки командой

```
>> ident
Opening ident ....... done.
```

В результате ее исполнения появляется диалоговое окно, показанное на рис. 6.3.

Графический интерфейс пакета имеет типовые средства, которые достаточно хорошо известны пользователям системы MATLAB + Simulink. В связи с этим детали описания этих средств в дальнейшем опускаются.

6.5.3. Пример работы в среде GUI

Работу с интерфейсом лучше всего рассмотреть на примере. В режиме командной строки введем команду

```
>> load dryer2
```

Выполнение команды приводит к загрузке в рабочую среду MATLAB массивов данных `u2` (входные данные) и `y2` (выходные данные), относящихся к такому объекту исследования, как фен (устройство для сушки волос), при этом `u2` — тепловая мощность, а `y2` — температура нагретого воздуха.

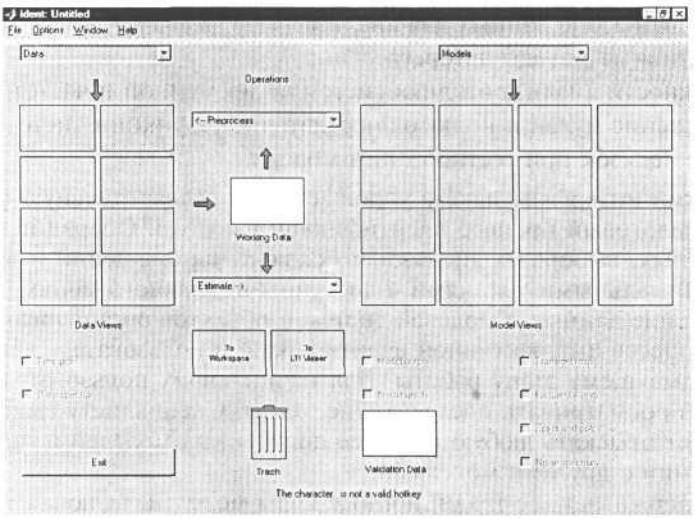

Рис. 6.3. Диалоговое окно графического интерфейса System Identification Toolbox

В верхнем левом углу окна выберем в раскрывающемся списке Data (Данные) опцию Import. Это приведет к открытию диалогового окна, показанного на рис. 6.4.

Введем в нем имя u2 в поле Input (Вход), y2 — в поле Output (Выход), в поле Starting Time (Начальное время) 0, в поле Samp. inter. (Интервал дискретизации) зададим интервал дискретизации 0.08, а в строке Data name (Имя данных) укажем произвольное название, например Фен. Затем нажмем кнопку Import. Далее нажатием кнопки Close (Закрыть) закроем окно импорта данных.

Указанные действия приведут к появлению в верхнем левом углу окна интерфейса значка в виде цветной линии, сопровождаемого надписью «Фен». Это означает, что импортируемые данные введены в среду интерфейса. Сообщения об этих данных также появляются в значках Working Data (Рабочие данные) и Validation Data (Данные для проверки модели).

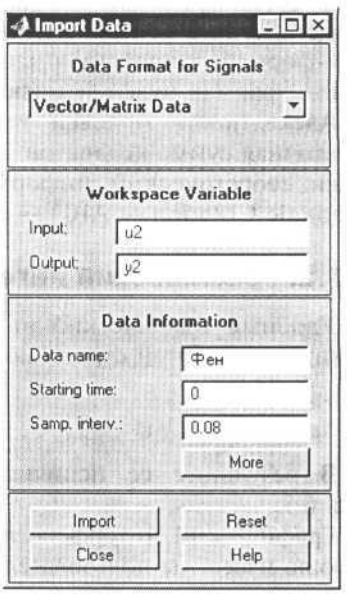

Рис. 6.4. Диалоговое окно импорта данных

6.5.4. Исследование исходных данных

Проведем исследование исходных данных, для чего установим флажок Time plot (Временной график) в левой нижней части окна интерфейса. Почти сразу появится графическое окно, содержащие графики входного и выходного сигналов $u2(t)$ и $y2(t)$ (рис. 6.5).

Заметим, что меню окна имеет ряд опций, позволяющих достаточно подробно изучать вид входных и выходных сигналов.

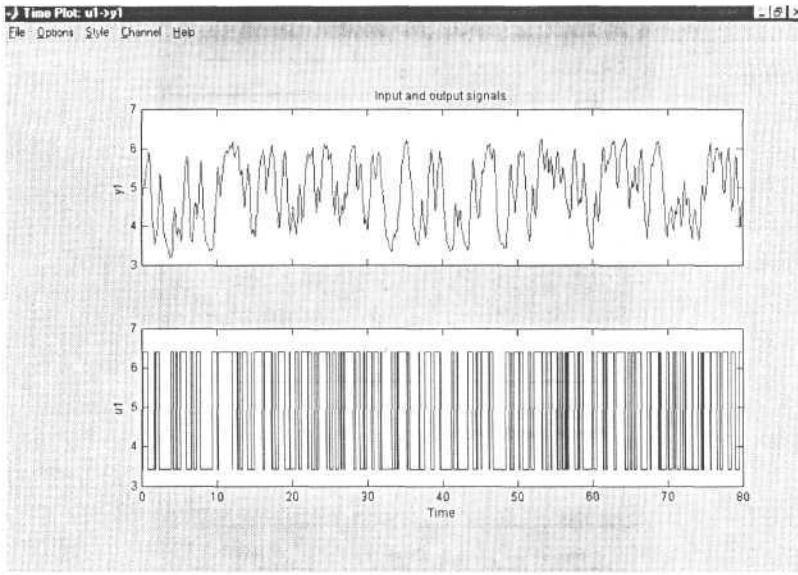

Рис. 6.5. Временные диаграммы сигналов u2(t) и y2(t)

Закроем окно Time plot и вернемся в основное окно интерфейса. Теперь проведем предварительную обработку сигналов исследуемого объекта, исключив из них постоянную составляющую. С этой целью активизируем раскрывающийся список Preprocess (Предварительная обработка) и выберем в нем опцию Remove means (Удалить среднее).

Результатом операции явится появление в одном из значков Data Views (Вид данных) в левой верхней части окна интерфейса информации о новых данных с именем Фенd. Можно опять активизировать окно Time plot и увидеть, что там появилось изображение двух новых сигналов, отличающихся от исходных отсутствием постоянной составляющей. Для их более удобного просмотра целесообразно воспользоваться командой меню окна Options | Autorange (Автоматическое масштабирование). Чтобы просмотру не мешали исходные графики, можно убрать их, просто щелкнув мышью на значке с их представлением (в левой верхней части окна интерфейса), при этом сразу изменится и масштаб изображения.

6.5.5. Построение модели

Приступим теперь к построению модели, принимая в качестве данных для ее построения данные Фенd. Перетянем их мышью в область Working Data (в центре окна интерфейса). Если мы теперь хотим получить какую-то информацию об этих данных или изменить что-либо (например, их имя), щелкнем мышью дважды на соответствующем значке в группе Data Views. Появится диалоговое окно, показанное на рис. 6.6. Изменим имя данных, например, на ФенМ и закроем данное окно.

Активизируем теперь опцию Select Range (Выбор диапазона) из списка Preprocess. Это приведет к появлению окна, показанного на рис. 6.7.

Диапазон можно задать либо в текстовом окошке Time span (Временной диапазон), либо с помощью мыши — выделяя прямоугольную область точно так же, как это проводилось для окна Time plot. Любым из этих способов укажем диапазон от 1 до 50 с и нажмем кнопку Insert (Вставить). Результат проделанной операции от-

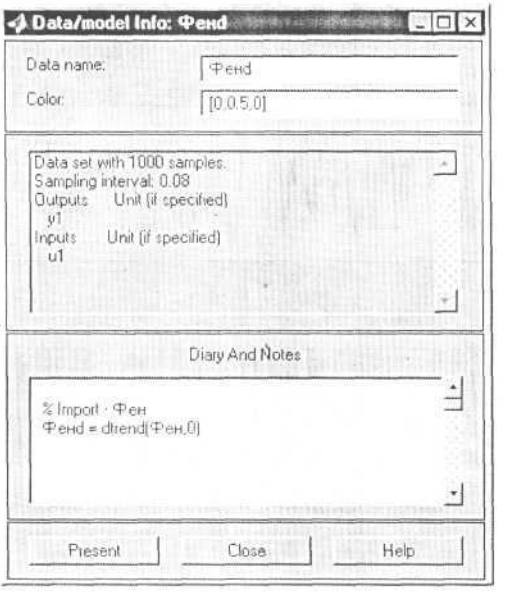

Рис. 6.6. Окно информации о выбранных данных

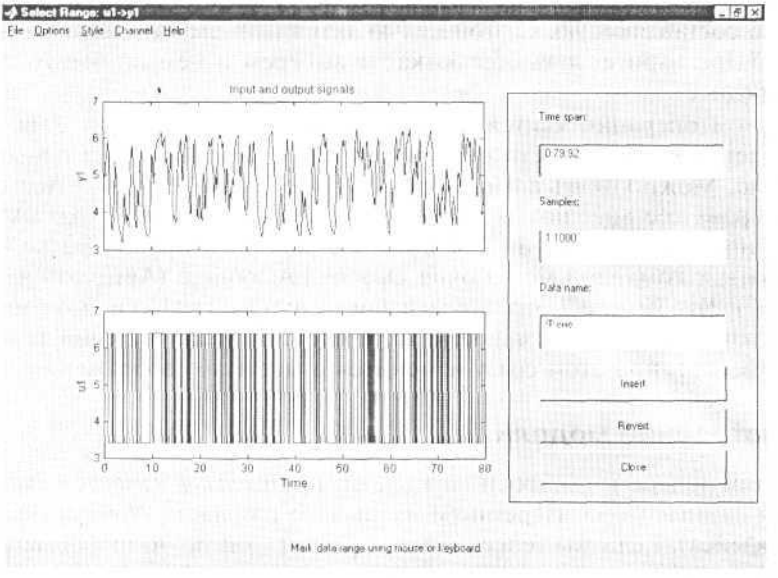

Рис. 6.7. Окно выбора диапазона

разится появлением значка, символизирующего эти (усеченные) данные (с именем Фене) в окне интерфейса.

Повторим операцию задания диапазона с целью формирования данных для проверки модели — например, из оставшейся части начального диапазона, в результате чего появится еще один значок данных с именем Фене. Закроем окно выбора диапазона.

С помощью мыши перетащим данные Фене в область Working Data, а данные ФенMv — в область Validation Data (заметим, что если на каком-то этапе работы с данными допущена ошибка и мы хотим удалить неправильные данные, необходимо просто перетащить их мышью на значок Trash (Мусор) в центральной нижней части окна интерфейса, и они исчезнут).

6.5.6. Оценка модели

Теперь можно приступать к нахождению оценки модели выбранного вида. Начнем с оценивания переходной функции объекта. В раскрывающемся списке Estimate (Оценивание) выберем опцию Correlation Model (Корреляционная модель), что приведет к появлению соответствующего диалогового окна с единственной строкой ввода. В ней можно задать данные фильтра, но в данном случае ничего не будем делать, только нажмем в нем кнопку Estimate (Оценить), после чего окно можно закрыть.

А в основном окне интерфейса в его правой части (Model Views — Вид моделей) при этом появится значок с надписью imp, означающий, что выбранная модель построена. Чтобы увидеть результат, установим флажок Transient resp (transient response — переходная функция) и увидим в появившемся окне (рис. 6.8) график переходной функции исследуемого объекта, найденной описанным выше корреляционным методом.

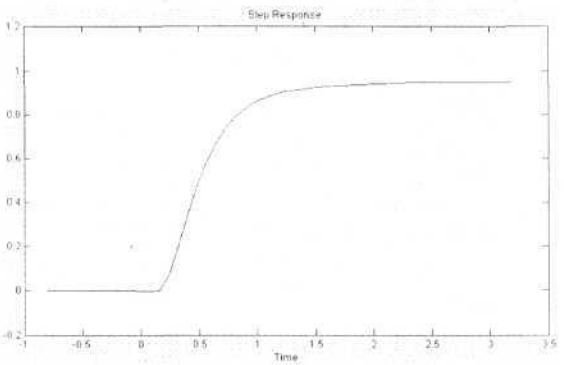

Рис. 6.8. Оценка переходной функции исследуемого объекта, найденная корреляционным методом

Используя команду Options | Impulse response меню данного окна, вместо графика переходной функции можно вывести график ИХ.

Оценим модель следующего вида — частотные характеристики объекта. Выберем в списке Estimate опцию Spectral model и повторим только что проделанные операции по нахождению оценки модели. В результате в основном окне интерфейса (в его правой части — Model Views) появится еще один значок с именем spad, символически представляющий модель в виде частотных характеристик, найденных спектральным методом. Результат отображается активизацией графического окна Frequency resp (Frequency response — частотные характеристики), показанного на рис. 6.9.

Перейдем теперь к оцениванию параметрических моделей, выбрав в списке Estimate опцию Parametric models. Данный выбор приведет к открытию диалогового окна задания структуры модели (рис. 6.10).

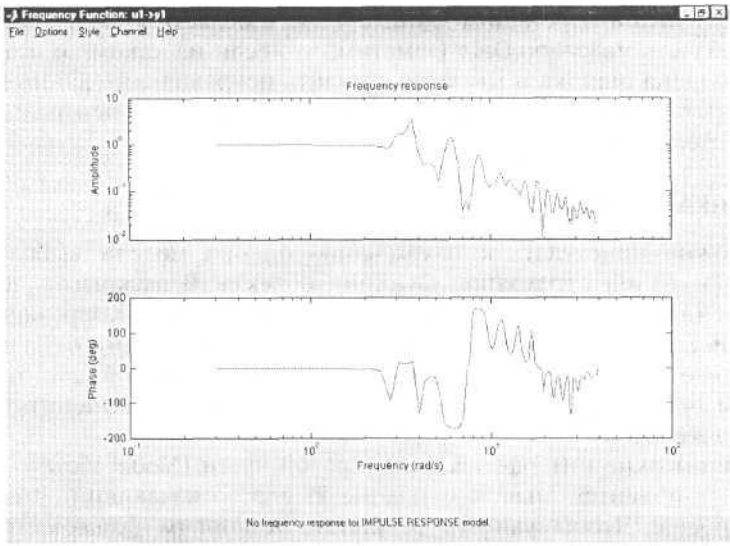

Рис. 6.9. Частотные характеристики, найденные спектральным методом

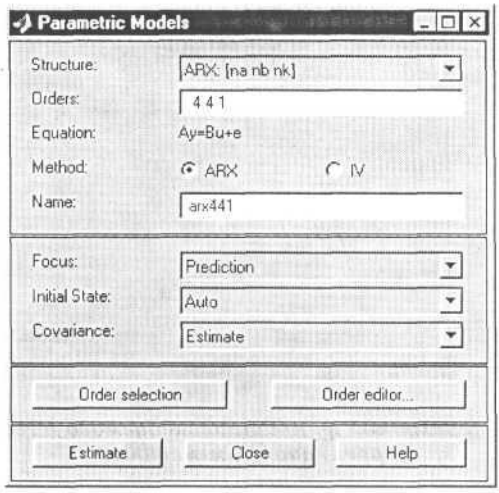

Рис. 6.10. Диалоговое окно задания структуры модели

По умолчанию пользователю предлагается модель типа ARX (см. выше) с параметрами $na = 4$, $nb = 4$, $nk = 1$. Можно согласиться или не согласиться с данными значениями. В последнем случае параметры можно изменить непосредственно в строке окна или с помощью редактора порядка модели (Order editor), вызываемого нажатием соответствующей кнопки. Можно вообще выбрать другую параметрическую модель, воспользовавшись раскрывающимся списком в верхней части окна (возможный выбор — модели типа ARX, ARMAX, OE, BJ, State Space и модель, задаваемая пользователем).

В рассматриваемом примере сохраним значения по умолчанию (ARX, 4, 4, 1) и нажмем кнопку Estimate (Оценить). Результатом действия будет появление значка модели с названием arx441.

Воспользовавшись далее возможностями редактора порядка модели (Order editor), зададим теперь ARX-модель с параметрами $na = 2$, $nb = 2$, $nk = 3$ (последнее означает наличие в объекте запаздывания с величиной $nk \times T = 3 \times 0.08 = 0.24$ с) и повторим операцию оценивания. Это приведет к появлению значка еще одной модели с именем arx223 (рис. 6.11).

Для сравнения полученных моделей активизируем окно Transient resp (рис. 6.12). В нем будут отображены графики переходных процессов, соответствующие трем построенным моделям — корреляционной (crad), arx441 и arx223.

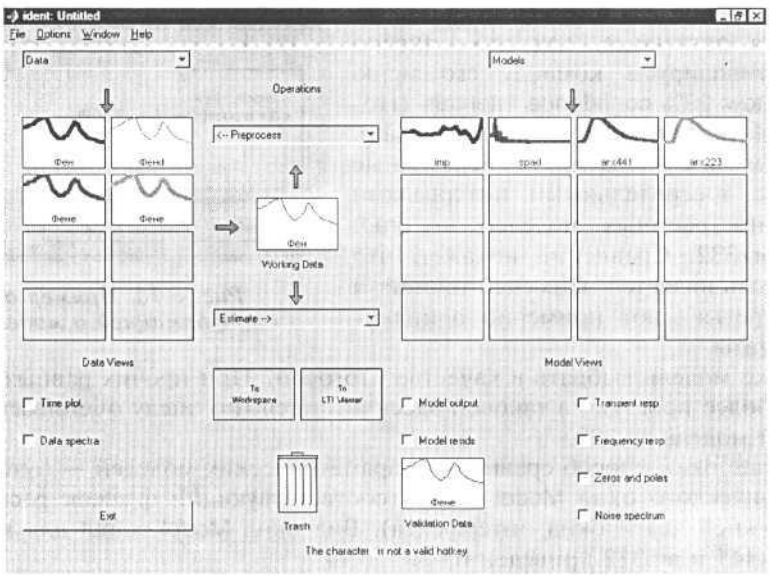

Рис. 6.11. Окно графического интерфейса с результатами построения моделей

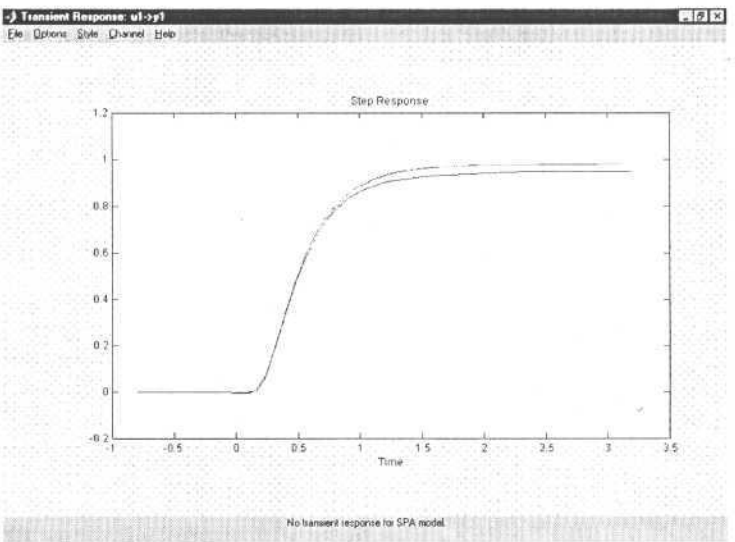

Рис. 6.12. Графики оценок переходного процесса для трех построенных моделей

Заметим, что однократный щелчок левой кнопкой мыши на графике модели в ее значке приведет к исчезновению соответствующего графика переходного процесса; повторный щелчок восстанавливает изображение. Двойной щелчок мыши на значке приводит к открытию окна с информацией о модели (рис. 6.13).

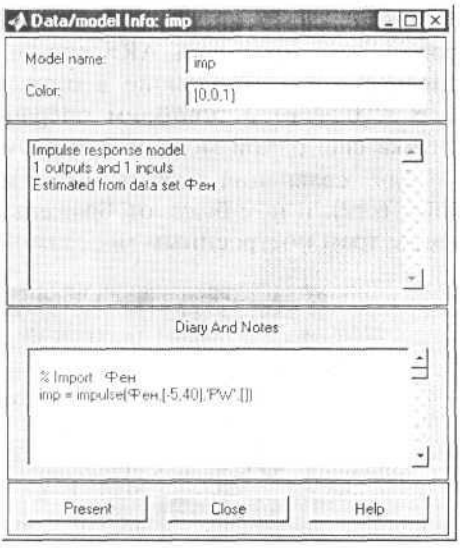

Рис. 6.13. Пример окна с информацией о модели imp

Вернемся к окну Transient resp. Оставляя в нем только по одному графику (то есть убирая остальные, как это описано выше), активизируем команду его меню Options | Show 99% confidence intervals (Показать 99%-й доверительный интервал). Просмотрим последовательно переходные процессы с доверительными интервалами для трех построенных моделей — crad, arx441 и arx332. Сравнение покажет, что первая модель является наименее точной, а вторая и третья дают примерно одинаковые результаты.

Какую же модель выбрать в качестве итоговой? При прочих равных условиях, очевидно, более простую, а таковой здесь является (по числу оцениваемых коэффициентов) модель arx332.

Есть и еще один способ сравнения параметрических моделей — путем активизации графического окна Model output (соответствующий флажок расположен в средней нижней части окна интерфейса). Вид окна Model output для выбранных моделей arx441 и arx332 приведен на рис. 6.14.

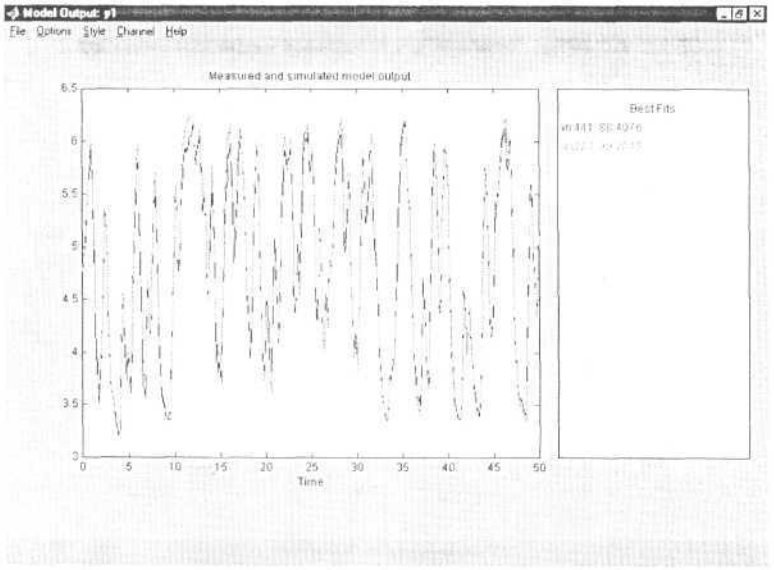

Рис. 6.14. Окно сравнения выходов моделей

В левой части данного окна приведены выходы объекта и указанных моделей, а в правой — величины, отражающие меру совпадения (среднеквадратичное рассогласование) экспериментальных и прогнозируемых данных. Как видно, несколько точнее оказывается модель arx441 (хотя и ненамного).

Возвратимся к процедуре оценивания параметрических моделей. Выберем структуру вида State Space. В появляющемся диалоговом окне задания структуры модели по умолчанию предлагается модель 4-го порядка (используется обновленная форма модели для переменных состояния — см. выше). Можно, конечно, с этим согласиться, но можно избрать поисковый метод определения наилучшей структуры (порядка) модели, указав в строке Orders (Порядок) диапазон возможных значений порядка модели, например `1:10`. Проделав это и нажав кнопку Estimate, в качестве результата выполненного действия получим открывшееся окно выбора порядка модели (рис. 6.15).

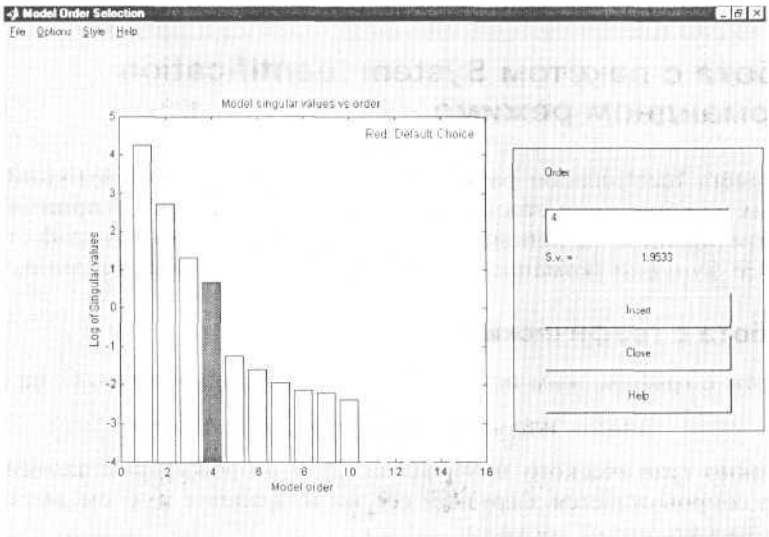

Рис. 6.15. Окно выбора порядка модели

График показывает вклад n-й составляющей вектора переменных состояния в выходной сигнал модели (значения n указаны по оси абсцисс). Целесообразно выбирать значение n таким, чтобы столбцы, соответствующие составляющим с номерами больше n, имели бы намного меньшую высоту, чем столбцы, расположенные слева от них. В рассматриваемом случае можно остановиться на $n = 4$.

Возвратимся далее в окно задания структуры модели и нажмем кнопку Estimate. Результатом явится появление значка новой модели с именем n4s4. Данную модель можно исследовать так же, как и полученные ранее модели.

Вообще-то для анализа моделей графический интерфейс пакета System Identification представляет весьма значительные возможности, среди которых отметим только возможность просмотра переходной функции модели с помощью программы LTI Viewer (см. описание ниже, посвященное пакету Control Systems). Другие возможности читатель может реализовать самостоятельно, используя приемы, описанные выше. Заметим, что в представлении окон разных версий пакета могут быть различия, порою существенные. Представленный вид окон относится к версии пакета System Identification Toolbox 5.0.

6.5.7. Сохранение модели

Сохранение построенных моделей производится в два этапа:

- сначала модель вводится в рабочее окружение системы MATLAB (перетаскиванием значка модели в область To Workspace в центре рабочего окна интерфейса), при этом модель будет фигурировать в рабочем пространстве MATLAB под тем же именем, что и в среде интерфейса;
- затем модель сохраняется командой сохранения в режиме командной строки (как любая переменная MATLAB).

Отметим, что можно сохранить все рабочее пространство интерфейса (при его закрытии даже появится соответствующая подсказка) в виде файла с расширением (по умолчанию) sid, при этом в следующем сеансе работы можно загрузить все полученные результаты. Сохраненное рабочее пространство называется сессией.

6.6. Работа с пакетом System Identification в командном режиме

Пакет System Identification располагает большим набором функций (команд), исполняемых из командной строки MATLAB и позволяющих, в принципе, решать задачи идентификации, не используя рассмотренную программу графического интерфейса. Все функции (команды) делятся на ряд групп, рассмотренных ниже.

6.6.1. Работа с графическим интерфейсом

Для работы с графическим интерфейсом служат две команды. Одна из них

```
ident или ident(session,directory)
```

открывает окно графического интерфейса. При второй форме записи открытие интерфейса сопровождается загрузкой сессии из файла с именем `session`, находящегося в директории `directory`.

Команда `midprefs` задает (изменяет) директорию для файла `midprefs.mat`, хранящего информацию о начальных параметрах графического интерфейса пользователя при его открытии.

6.6.2. Имитация и предсказание

Для имитации и предсказания служат 4 функции.

Функция `idinput` в формах

```
u = idinput(N)          u = idinput(N,type,band,levels)
u = idinput(N,'sine',band,levels,auxvar)
```

генерирует входной идентифицирующий сигнал. `N` задает количество значений сигнала, а строковая переменная `type` задает тип сигнала (по умолчанию — `'rbs'`):

- `type = 'rs'` — случайный гауссовый сигнал;
- `type = 'rbs'` — случайный бинарный (двоичный) сигнал;
- `type = 'prbs'` — псевдослучайный двоичный сигнал (ПСДС);
- `type = 'sine'` — полигармонический сигнал (сумма синусоид).

Двухэлементный вектор-строка band = [wlow, whigh] задает диапазон частот сигнала (для сигналов типа 'rs', 'rbs' и 'sine'), при этом wlow — нижняя частота, а whigh — верхняя частота (частоты задаются в долях от частоты Найквиста — см. выше). Белый шум, таким образом, характеризуется значением band = [0, 1], которое является и значением по умолчанию.

Для сигнала типа ПСДС ('prbs') используется значение band = [twologp, M], при этом 1/M — длительность интервала постоянства сигнала, а значение 2^twologp-1 определяет период ПСДС. Максимальное значение периода соответствует twologp=18 (или twologp=0); в этом случае по умолчанию band = [0, 1].

Двухэлементный вектор levels = [minu, maxu] задает минимальный и максимальный уровни для сигналов типа 'rs', 'rbs' и 'sine'. Для сигнала типа 'rs' величина minu имеет значение, равное математическому ожиданию минус среднее квадратическое (или стандартное) отклонение, а maxu — математическому ожиданию плюс среднее квадратическое отклонение.

Для сигнала типа 'sine' возможно задание четвертого аргумента — auxvar = [no_of_sinusoids, no_of_trials], где no_of_sinusoids — число гармоник, образующих сигнал, а no_of_trials определяет, сколько итераций должно быть сделано, чтобы минимизировать амплитуду сигнала путем задания различным синусоидам произвольных начальных фаз. Значение по умолчанию — auxvar = [10, 10]. Возвращаемая величина — N-элементный вектор u.

Следующие три функции имеют идентичные входные параметры:

Вычисление предсказания в модели:

```
yp = predict(m,data) [yp,mpred] = predict(m,data,k,init)
```

Вычисление ошибок предсказания:

```
e = pe(m,data) [e,x0] = pe(m,data,init)
```

Имитация линейной модели:

```
y = sim(m,ue) [y, ysd] = sim(m,ue,init)
```

Здесь data — массив исходных данных, m — число idmodel-объектов (idpoly, idss, idgrey, or idarxm), k — число шагов, используемых при предсказании (по умолчанию 1), и init — параметр инициализации.

Из этих функций особый интерес вызывает функция предсказания predict, реализующая алгоритм предсказания с применением фильтра Калмана. Для проверки применимости данной функции подготовим случайный тестовый сигнал, содержащий 400 точек, и выполним имитацию линейной модели сигнала:

```
m0 = idpoly([1 -0.99],[],[1 -1 0.2]);
e = iddata([],randn(400,1));
y = sim(m0,e);
```

Теперь создадим ARMAX модель для первых 200 точек, выполним прогноз еще на 200 точек и построим графики сигнала (все 400 точек) и прогноза (последние 200 точек):

```
m = armax(y(1:200),[1 2]);
yp = predict(m,y,4);
plot(y(1:400),yp(201:400))
```

Рис. 6.16 показывает эффективность прогноза. Здесь маленькими точками показан график исходной зависимости. Для прогноза использовано 200 первых точек, но для наглядности выведено все 400 точек. Начиная с 201 точки строится жирная линия прогноза.

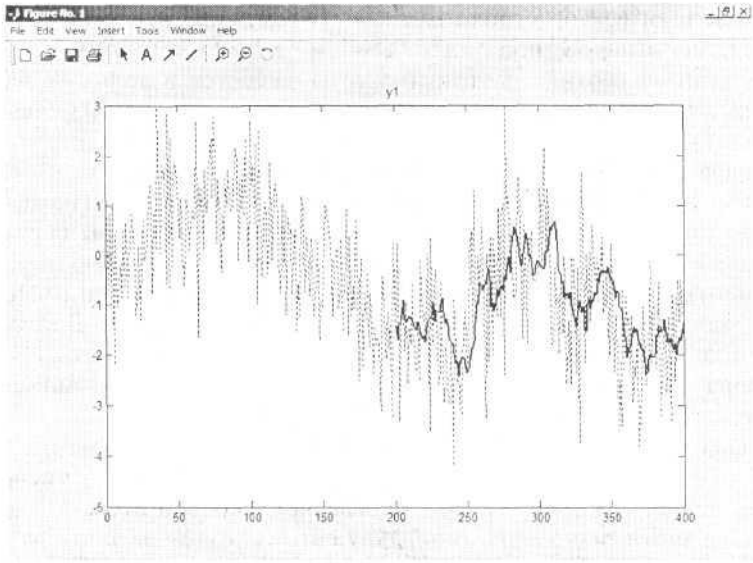

Рис. 6.16. Пример применения функции предсказания

Для сравнения результатов прогноза и оценки его погрешности воспользуемся следующей командой:

```
compare(y,m,4,201:400);
```

Она строит исходные точки, линию прогноза и вычисляет среднеквадратическую ошибку предсказания. График, построенный данной функцией, представлен на рис. 6.17. Как нетрудно заметить, погрешность предсказания составила около 8%, что является хорошим результатом для случайного сигнала.

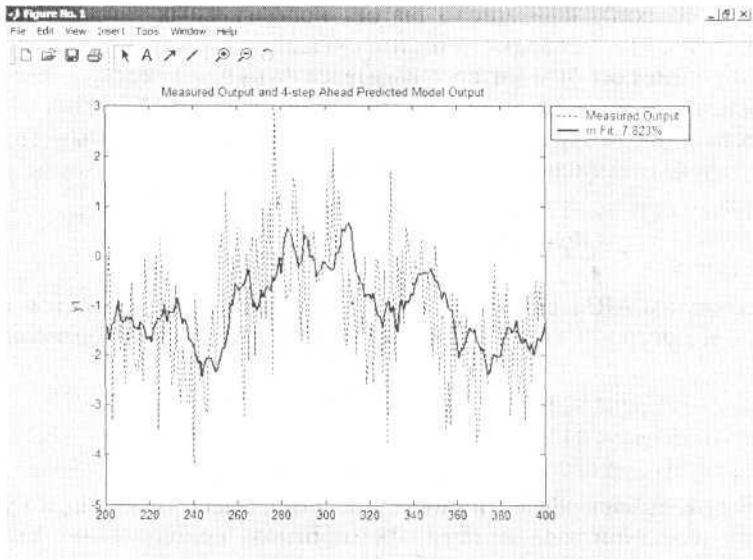

Рис. 6.17. Пример применения функции предсказания

6.6.3. Манипуляции с данными

В данную группу входят следующие восемь функций:

- `detrend` — удаляет тренды из данных;
- `get/set` — устанавливает и модифицирует свойства iddata;
- `iddata` — конструирует объект данных;
- `idfilt` — фильтрует данные;
- `merge (iddata)` — соединяет множества данных в множество экспериментальных данных;
- `misdata` — реконструирует пропущенные входные и выходные данные;
- `plot (iddata)` — строит график данных;
- `resample` — удаляет дискретизацию данных.

Рассмотрим эти функции выборочно. Функция

```
zd = detrend(z) или zd = detrend(z,o,brkp)
```

удаляет тренд из набора данных Ее аргументы: `z` — матрица исходных данных, в которой столбцы соответствуют векторам данных; `o` (по умолчанию 0) — порядок тренда (нулевой или первый); если аргумент `brkp` не определен, из вектора данных вычитается линейная функция, соответствующая тренду; если данный аргумент определен, из исходных данных вычитается «ломаная линия» с точками разрыва производной при элементах, номера которых содержатся в `brkp` (только для линейных трендов). Возвращаемая величина: `zd` — матрица преобразованных данных.

В следующем примере:

```
zd(:,1) = detrend(z(:,1),1,119);
zd(:,2) = detrend(z(:,2));
```

из матрицы z (если она задана) удаляется V-образный тренд из данных первого столбца с пиком на 119-м элементе и тренд нулевого порядка (постоянная составляющая) — для элементов второго столбца матрицы исходных данных. Приведем наглядный пример:

```
>> sig = [0 1 -2 1 0 1 -2 1 0];
>> trend = [0 1 2 3 4 3 2 1 0];
>> x = sig+trend;
>> y = detrend(x,'linear',5)
y =
  Columns 1 through 7
    -0.0000 1.0000 -2.0000 1.0000 0.0000 1.0000 -2.0000
  Columns 8 through 9
    1.0000 0.0000
```

Функция `idfilt`

```
zf = idfilt(z,filter)              zf = idfilt(z,ord,Wn)
zf = idfilt(z,ord,causality)    [zf,mf] = idfilt(z,ord,Wn,hs)
```

возвращает данные, фильтрованные с помощью фильтра Баттерворта. Эта функция создает фильтр Баттерворта порядка `ord` и осуществляет фильтрацию данных (по столбцам) матрицы данных `z = [y u]`. Если аргумент `hs` не задан и `Wn` содержит только один элемент, создается фильтр низких частот (ФНЧ) с частотой среза `Wn`.

Если `hs ='high'`, создается фильтр верхних частот (ФВЧ) с частотой среза `Wn`.

Если `Wn = [Wnl Wnh]`, то есть является двухэлементным вектором, и аргумент `hs` не определен, создается полосовой фильтр (ПФ) с полосой пропускания от `Wnl` до `Wnh`. Если же в этом случае задан аргумент `hs = 'stop'`, создается заграждающий полосовой фильтр. Порядок фильтра в этих случаях — `2*ord`.

Отметим, что частоты `Wnl` и `Wnh` задаются в относительных единицах, в долях от так называемой *частоты Найквиста*, определяемой как $\omega = \pi/T$ (или $f = 1/2T$), где T — выбранный интервал дискретизации. Возвращаемые величины: `zf` — матрица отфильтрованных данных, `thf` — описание созданного фильтра в так называемом тета-формате (см. ниже).

С другими функциями этой группы можно ознакомиться самостоятельно.

6.6.4. Непараметрическое оценивание

В данную группу входят 6 функций. Ввиду важности этих функций мы рассмотрим их все.

Функция `covf` выполняет расчет авто- и взаимных корреляционных функций совокупности экспериментальных данных:

```
R = covf(z,M) R = covf(z,M,maxsize)
```

Аргументы:

- `z` — матрица данных размера `N×nz`, каждый столбец которой соответствует входному или выходному сигналу (обычно `z=[y u]`);
- `M` — максимальная величина дискретного аргумента, для которой рассчитываются корреляционные функции, минус единица;
- `maxsize` — параметр, определяющий максимально допустимый размер матриц.

Возвращаемая величина — матрица `R` размера $nz^2 \times M$ с элементами

$$R[i + (j-1)nz,\ k+1] = \hat{R}_{ij}(k) = \frac{1}{N}\sum_{t=1}^{N} z_i(t) z_j(t+k).$$

Функция `cra` определяет оценку ИХ методом корреляционного анализа для одномерного (один вход — один выход) объекта:

```
cra(z) [ir,R,cl] = cra(z,M,na,plot) cra(R)
```

Аргументы:

- `z` — матрица экспериментальных данных вида `z = [y u]`, где `y` — вектор-столбец, соответствующий выходным данным, `u` — вектор-столбец входных данных;
- `M` — максимальное значение дискретного аргумента, для которого производится расчет оценки ИХ, по умолчанию `M=20`;
- `na` — порядок модели авторегрессии (степени многочлена $A(z)$), которая используется для расчета параметров «обеляющего» фильтра $\Phi(z)$, по умолчанию `na=10`. При `na=0` в качестве идентифицирующего используется не преобразованный входной сигнал;
- `plot` — `plot=0` означает отсутствие графика, `plot=1` (по умолчанию) — график полученной оценки ИХ вместе с 99%-м доверительным коридором, `plot=2` — выводятся графики всех корреляционных функций.

Возвращаемые величины:

- `ir` — оценка ИХ (вектор значений);
- `cl` — 99%-й доверительный коридор для оценки ИХ;
- `R` — матрица, элементы первого столбца которой — значения дискретного аргумента, элементы второго столбца — значения оценки автокорреляционной функции выходного сигнала (возможно, отфильтрованного), элементы третьего столбца — значения оценки автокорреляционной функции входного сигнала (возможно, «обеленного»), элементы четвертого столбца — значения оценки взаимной корреляционной функции.

Функции `impulse` и `step` служат для оценки импульсной и переходной характеристик. Они имеют одинаковые параметры, так что ограничимся указанием только основных форм записи функции `impulse`:

```
impulse(m)                                    impulse(data)
impulse(data,'sd',sd,'pw',na,Time)            impulse(m,'sd',sd,Time)
[y,t,ysd] = impulse(m)                        mod = impulse(data)
impulse(m1,m2,...,dat1, ...,mN,Time,'sd',sd)
```

Функция `spa` возвращает частотные характеристики одномерного объекта и оценки спектральных плотностей его сигналов для обобщенной линейной модели объекта (возвращая модель объекта в так называемом частотном формате):

```
[g,phiv] = spa(z)  [g,phiv,z_spe] = spa(z,M,w,maxsize,T)
```

Аргументы:

- `z` — матрица исходных данных — как в рассмотренных выше функциях;
- `M` — ширина временного окна (см. выше), по умолчанию `M = min(30, length(z)/10)`, где `length(z)` — число строк матрицы `z`;
- `w` — вектор частот, для которых производится расчет частотных характеристик, по умолчанию `[1:128]/128*pi/T`;
- `T` — интервал дискретизации;
- `maxsize` — параметр, определяющий максимальный размер матриц, создаваемых в процессе вычислений (оптимальный выбор этого значения позволяет добиться максимальной скорости расчетов).

Возвращаемые величины:

- `g` — оценка $W(e^{j\omega T})$ в частотном формате;
- `phiv` — оценка спектральной плотности шума $v(t)$;
- `z_spe` — матрица спектральных плотностей входного и выходного сигналов.

Рассмотрим пример. Пусть исходные данные содержатся в файле dryer2.mat. Воспользуемся функцией `spa` для нахождения оценок амплитудно- и фазочастотных характеристик объекта с выводом результата в форме графиков (диаграммы Бодэ — рис. 6.18):

```
>> load dryer2; z=[y2 u2]; g = spa(z); bode(g,3)
```

Заметим, что здесь оценка АЧХ строится в логарифмическом масштабе с помощью новой функции `bode` (вместо `bodeplot` в старых версиях пакета). В продолжение примера ниже приведены функции, обеспечивающие вывод графиков АЧХ,

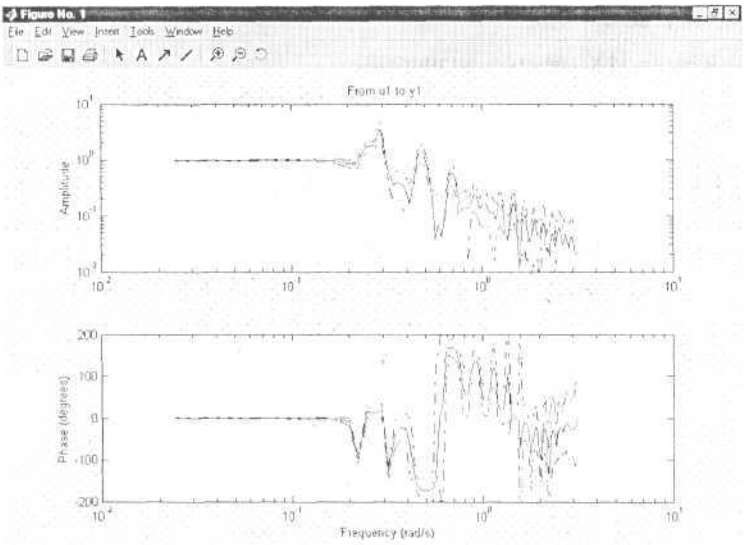

Рис. 6.18. Оценки АЧХ и ФЧХ вместе с соответствующими доверительными коридорами

ФЧХ и $S_v(\omega)$ вместе с доверительными коридорами шириной в три среднеквадратических отклонения, а также сами графики (рис. 6.19):

```
>> w = logspace(-2,pi,128); g= spa(z,[],w);
>> bode(g,3); bode(g('noise'),3)
```

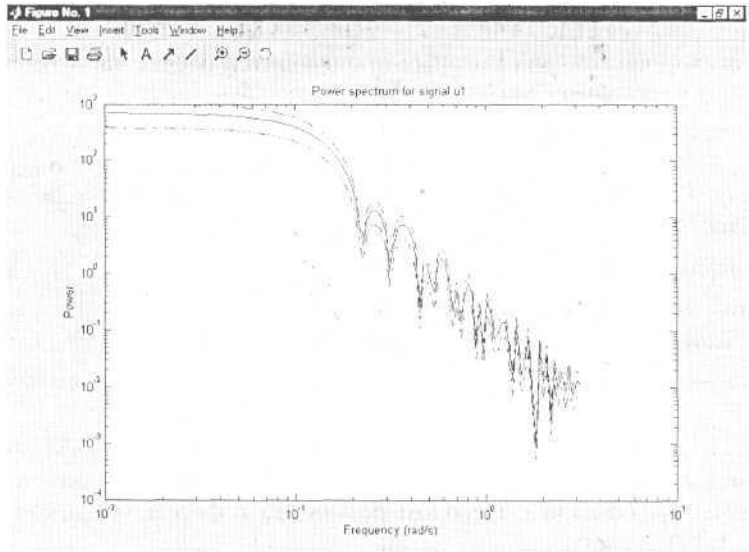

Рис. 6.19. График оценки $S_v(\omega)$ вместе с доверительным коридором

Функция

```
g = etfe(z) g = etfe(z,M,N,T)
```

так же, как предыдущая функция, возвращает оценку дискретной передаточной функции для обобщенной линейной модели (см. выше) одномерного объекта в частотной форме. Рекомендована к использованию для узкополосных объектов. Аргументы:

- `z` — как для предыдущей функции;
- `M` — как для предыдущей функции;
- `N` — определяет диапазон частот для расчета (эта величина должна быть степенью 2) значений частотных характеристик согласно формуле `w = [1:N]/N*pi/T`. По умолчанию `N=128`;
- `T` — интервал дискретизации (по умолчанию `T=1`).

Возвращаемая величина $g = W(e^{j\omega T})$.

В продолжение предыдущего примера (с его исходными данными) проведем сравнение результатов использования функций `spa` и `etfe` (рис. 6.20):

```
>> ge = etfe(z); gs = spa(z); bode(ge, gs)
```

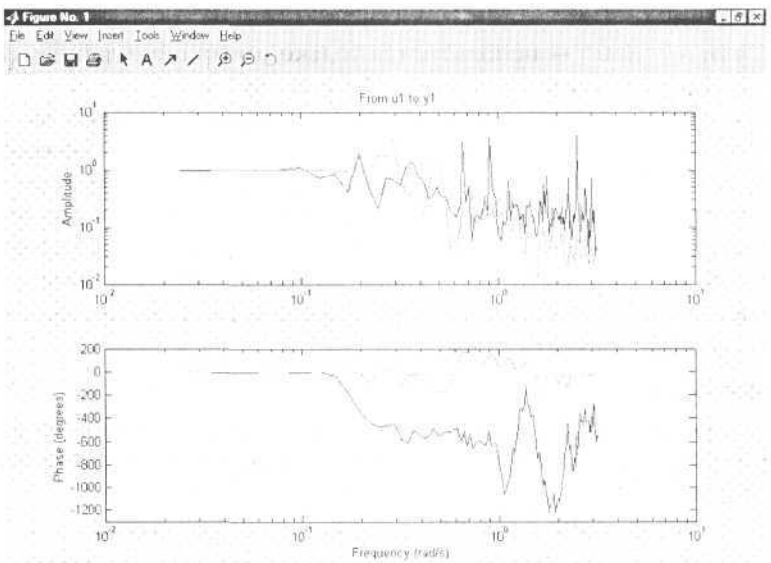

Рис. 6.20. Оценки АЧХ и ФЧХ объекта, полученные с помощью функций spa и etfe

Сравнение графиков показывает, что более гладкими (и, наверное, более реалистичными) выглядят оценки частотных характеристик, полученные с помощью функции `spa` (они представлены более светлыми кривыми).

6.6.5. Параметрическое оценивание

Для параметрического оценивания предназначены 7 функций (ранее их было 11). Это также важные функции, и ниже представлено описание их всех.

Функция `ar` оценивает параметры модели авторегрессии (AR), то есть коэффициенты полинома $A(z)$, при моделировании скалярных временных последовательностей:

```
th = ar(y,n)
[th,refl] = ar(y,n,approach,win,maxsize,T)
```

Аргументы:

- y — вектор-столбец данных, содержащий N элементов;
- n — порядок модели (число оцениваемых коэффициентов);
- approach — (строковая переменная) определяет метод оценивания:
  - 'fb' — прямой-обратный метод (разновидность метода наименьших квадратов). Используется по умолчанию;
  - 'ls' — метод наименьших квадратов (МНК);
  - 'yw' — метод Юла—Уокера;
  - 'burg' — метод Бэрга (комбинация МНК с минимизацией гармонического среднего);
  - 'gl' — метод с использованием геометрического среднего.

Если любое из данных значений заканчивается нулем (например, 'burg0'), то вычисление сопровождается оцениванием корреляционных функций.

- Аргумент win (строковая переменная) используется в случае отсутствия части данных:
  - win = 'now' — используются только имеющиеся данные (используется по умолчанию — за исключением случая approach = 'yw');
  - window = 'prw' — отсутствующие начальные данные заменяются нулями, так что суммирование начинается с нулевого момента времени;
  - window = 'pow' — последующие отсутствующие данные заменяются нулями, так что суммирование расширяется до момента времени N+n;
  - window = 'ppw' — и начальные, и последующие отсутствующие данные заменяются нулями. Используется в алгоритме Юла—Уокера;
- аргумент maxsize определяет максимальную размерность задачи;
- T — интервал дискретизации.

Возвращаемые величины:

- th — информация о модели в так называемом тета-формате (внутреннем матричном формате представления параметрических моделей пакета System Identification);
- refl — информация о коэффициентах и функции потерь.

Пример использования функции ar для оценивания модели авторегрессии 4-го порядка приведен ниже.

```
>> load dryer2; y=[y2(1:300)]; th=ar(y,4); present(th)
Discrete-time IDPOLY model: A(q)y(t) = e(t)
A(q) = 1 - 2.172 (+-0.0579) q^-1 + 1.57 (+-0.1379) q^-2 -
 0.3183 (+-0.1384) q^-3 - 0.07901 (+-0.05835) q^-4
Estimated using AR ('fb'/'now') from data set y
Loss function 0.0107822 and FPE 0.0110736
Sampling interval: 1
Created: 29-Mar-2002 00:36:10
Last modified: 29-Mar-2002 00:36:12
```

Функция armax оценивает параметры ARMAX-модели:

```
th = armax(z,nn) th = armax(z,nn,'trace')
[th, iter_info] = armax(z,nn,maxiter,tol,lim,maxsize,T,'trace')
```

Аргументы функции:

- `z = [y u]` — матрица экспериментальных данных; в случае многомерного входа `u` представляет собой матрицу с числом столбцов, равным числу входов;
- `nn = [na nb nc nk]` — задаваемые параметры ARMAX-модели (степени полиномов и величина задержки);
- `'trace'` — наличие этого аргумента задает выдачу информации о ходе итерационного процесса вычислений (трассировка);
- `maxiter` — определяет максимальное сило итераций при решении оптимизационной задачи, по умолчанию `maxiter=10`;
- `tol` — определяет заданную погрешность вычислений при поиске минимума, по умолчанию `tol=0.01`;
- `lim` — так называемый критерий робастности, по умолчанию `lim=1.6`. При `lim=0` используется обычный МНК;
- `maxsize=4096`;
- `T` — интервал дискретизации, по умолчанию равен 1.

Возвращаемые величины:

- `th` — информация о модели в тета-формате;
- `iter_info = [номер последней итерации, значение функции потерь на данной итерации]` — информация о выполнении итерационного процесса оценивания модели.

Использование функции `armax` иллюстрируется следующим примером:

```
>> load dryer2; z=[y2 u2];
>> th=armax(z,[2 2 2 1]); present(th)
Discrete-time IDPOLY model: A(q)y(t) = B(q)u(t) + C(q)e(t)
A(q) = 1 - 1.645 (+-0.01126) q^-1 + 0.6986 (+-0.01022) q^-2
B(q) = 0.006593 (+-0.002221) q^-1 + 0.04618 (+-0.002741) q^-2
C(q) = 1 - 0.2807 (+-0.02966) q^-1 - 0.4727 (+-0.02901) q^-2
Estimated using ARMAX from data set z
Loss function 0.00716988 and FPE 0.00728552
Sampling interval: 1
Created:          29-Mar-2002 00:39:10
Last modified:    29-Mar-2002 00:39:20
```

В данном случае задана структура модели, все полиномы которой имеют 2-й порядок. Для выводимой информации, по-видимому, особых разъяснений не требуется.

Функция `arx` оценивает параметры моделей AR и ARX:

```
th = arx(z,nn) th = arx(z,nn,maxsize,T)
```

Аргументы и возвращаемая величина идентичны используемым в предыдущей функции за тем исключением, что аргумент `nn` содержит только три параметра:

```
nn = [na nb nk]
```

Функция может использоваться для оценивания параметров многовходовой модели вида

$$A(z)y(t) = B_1(z)u_1(t - nk_1) + B_2(z)u_2(t - nk_2) + \ldots + B_{nu}(z)u_{nu}(t - nk_{nu}) + e(t),$$

при этом

- в представлении `z=[y u]` экспериментальных данных `u = [u1, u2, ..., unu]` есть матрица, столбцы которой соответствуют входам;
- `nb` и `nk` — векторы-столбцы, определяющие порядки полиномов $B_j(z)$ и задержки nk_j для входов объекта.

Функция `bj` оценивает параметры модели Бокса—Дженкинса:

```
th = bj(z,nn)      th = bj(z,nn,'trace')
[th, iter_info]= bj(z,nn,maxiter,tol,lim,maxsize,T,'trace')
```

Аргументы и возвращаемые величины идентичны приведенным для функции `armax` за тем исключением, что в данном случае аргумент `nn` содержит пять параметров: `nn = [nb nc nd nf nk]`, то есть степени четырех полиномов и величину задержки.

Функция `canstart` оценивает коэффициенты канонической формы модели для переменных состояния:

```
th = canstart(z,orders,nu)      th = canstart(z,orders,nu,dkx)
```

Аргументы функции следующие:

- `z` — матрица данных;
- `orders` — скаляр, задающий порядок n модели;
- `nu` — число входов;
- `dkx = [d, k, x]` — трехэлементный вектор, в котором:
  - `k=1` означает, что матрица **K** в модели полностью параметризована (то есть подлежит оцениванию); `k=0` означает, что **K** = **0**;
  - `d` имеет тот же смысл, но по отношению к матрице модели **D**;
  - `x=1` означает, что параметризован вектор начальных значений переменных состояния; при `x=0` данные значения полагаются нулевыми.

По умолчанию `dkx = [0, 1, 1]`. Возвращаемая величина `th` — модель в тета-формате.

Приведем пример использования рассматриваемой функции с целью восстановления модели для переменных состояния второго порядка по имеющимся экспериментальным данным (один вход, один выход):

```
>> load dryer2; z=[y2 u2]; th=canstart(z,2,1); present(th)
State-space model: x(t+Ts) = A x(t) + B u(t) + K e(t)
                      y(t) = C x(t) + D u(t) + e(t)
A =
                       x1             x2
            x1          0              1
            x2   -0.86694         1.8219
B =
                       u1
            x1   -0.013554
            x2    0.032831
C =
                       x1             x2
            y1          1              0
D =
```

```
                    u1
          y1         0
K =
                    y1
          x1     1.1347
          x2     1.3452
x(0) =
          x1      4.759
          x2     4.8605
Estimated using N4SID from data set z
Loss function 0.0124702 and FPE 0.0126714
Sampling interval: 1
Created:          29-Mar-2002 00:41:30
Last modified:    29-Mar-2002 00:41:35
```

Функция `ivar` оценивает параметры скалярной AR-модели

$$A(z)\, y(t) = v(t),$$

где шум $v(t)$ предполагается коррелированным случайным процессом типа процесса скользящего среднего, то есть имеющим представление $v(t) = H(z)e(t)$ (здесь $e(t)$ — белый шум), с использованием квазиоптимального метода инструментальной переменной (так называемого IV-метода). Функция записывается в виде:

```
th = ivar(y,na)
th = ivar(y,na,nc,maxsize,T)
```

где

- `y` — вектор данных;
- `na` — порядок модели авторегрессии (степень полинома $A(z)$);
- `nc` — порядок полинома $H(z)$, по умолчанию `na=nc`;
- `maxsize` и `T` имеют тот же смысл, что и для функций `armax` и `bj`.

Возвращаемая величина `th` — модель в тета-формате.

Функция `ivx` оценивает параметры ARX-модели методом инструментальной переменной:

```
th = ivx(z,nn,x)  th = ivx(z,nn,x,maxsize,T)
```

Функция аналогична функции `arx` и является ее альтернативой для случая, когда шум наблюдений является коррелированным. Аргумент `x` — матрица задаваемых инструментальных переменных. Остальные аргументы и возвращаемая величина аналогичны рассмотренным для функции `arx`.

Вместо функции `arx` рекомендуется использовать функцию `iv4`:

```
th = iv4(z,nn)          th = iv4(z,nn,maxsize,T)
```

Она оценивает параметры ARX-модели с использованием четырехступенчатого метода инструментальной переменной. Может применяться для идентификации многомерных объектов. Функция аналогична функции `arx` и является ее альтернативой для случая, когда шум наблюдений является коррелированным.

Функция `n4sid` используется для оценивания параметров моделей для переменных состояния в канонической форме при произвольном числе входов и выходов:

```
th = n4sid(z)
[th,AO] = n4sid(z,order,ny,auxord,dkx,maxsize,T,'trace')
```

где

- `z` — матрица экспериментальных данных;
- аргумент `order` задает порядок модели. Если данный аргумент вводится как вектор-строка, например, `order = [1:10]`, то предварительные расчеты выполняются по моделям всех заданных порядков (по умолчанию — от 1-го по 10-й) с выводом графика, позволяющего выбрать оптимальный порядок. Если `order = 'best'` (по умолчанию), выбирается модель «наилучшего» порядка (путем выбора наилучшей среди моделей 1-го — 10-го порядков);
- `ny` — количество выходов, по умолчанию `ny = 1`;
- `auxord` — так называемый дополнительный порядок, используемый алгоритмом оценивания. Должен быть больше, чем порядок, задаваемый параметром `order` (по умолчанию `auxord = 1.2*order+3`. Если `auxord` вводится как вектор-строка (так же, как `order`), выбирается модель наилучшего порядка;
- `dkx, maxsize, T, 'trace'` — как для функций `bj, canstart, ivar`.

Возвращаемые величины:

- `th` — имя (идентификатор) построенной модели в тета-формате;
- `AO` — наилучший порядок модели (определяемый с помощью задания `auxord`).

Функция `oe` оценивает параметры OE-модели:

```
th = oe(z,nn)       th = oe(z,nn,'trace')
[th, iter_info] = oe(z,nn,maxiter,tol,lim,maxsize,T,'trace')
```

Здесь аргумент `nn` задается в форме `nn = [nb nf nk]` и содержит информацию о степенях полиномов модели $B(z)$, $F(z)$ и о величине задержки nk.

Остальные аргументы и возвращаемые величины аналогичны рассмотренным для остальных функций данной группы.

Функция `pem` оценивает параметры обобщенной многомерной линейной модели вида

$$A(z)y(t) = \frac{B_1(z)}{F_1(z)}u_1(t-nk_1) + \frac{B_2(z)}{F_2(z)}u_2(t-nk_2)+\ldots+\frac{B_{nu}(z)}{F_{nu}(z)}u_{nu}(t-nk_{nu}) + \frac{C(z)}{D(z)}e(t).$$

Функция записывается в виде:

```
th = pem(z,nn)       th = pem(z,nn,'trace')
[th, iter_info] = pem(z,nn,index,maxiter,tol,lim,...
maxsize,T,'trace')
```

где

- аргумент `nn` имеет представление `nn = [na nb nc nd nf nk]`, где `na, nb, nc, nd, nf, nk` в многомерном случае являются векторами-столбцами;
- `index` — вектор-строка, содержащий индексы коэффициентов, которые должны быть оценены (по умолчанию оцениваются все коэффициенты).

Остальные аргументы и возвращаемые величины соответствуют рассмотренным выше.

6.6.6. Итерационное параметрическое оценивание

Семь функций позволяют проводить оценивание коэффициентов моделей итерационными (рекуррентными) методами, при которых на каждой k-й итерации учитываются экспериментальные данные, соответствующие моменту времени k.

Функция `rarmax` реализует процедуру рекуррентного оценивания коэффициентов ARMAX-модели:

```
thm = rarmax(z,nn,adm,adg)
[thm,yhat,P,phi,psi] = rarmax(z,nn,adm,adg,th0,P0,phi0,psi0)
```

Здесь:

- `z` — матрица данных;
- `nn` — аргумент, задающий параметры ARMAX-модели, `nn = [na nb nc nk]`;
- `adm` и `adg` — аргументы, задающие вид процедуры оценивания, например, значения `adm ='ff'` и `adg = lam` задают рекуррентный метод наименьших квадратов, `adm = 'ug'` и `adg = gam` — градиентный метод оценивания и т. п. (дополнительную информацию можно получить, используя команду `help rarmax`);
- `th0` — вектор-строка, содержащий начальные значения оцениваемых параметров, по умолчанию — нулевой вектор;
- `P0` — начальная матрица ковариаций оцениваемых коэффициентов, по умолчанию равна единичной матрице, умноженной на 10^4;
- `phi0` и `psi0` — соответственно начальные значения вектора данных и вектора-градиента (по умолчанию — нулевые). Их размеры достаточно сложным образом зависят от параметров модели.

Возвращаемые величины:

- `thm` — матрица оценок коэффициентов модели; k-я строка матрицы содержит оценки, соответствующие моменту времени k в следующем порядке:
- `thm(k,:) = [a1,a2,...,ana,b1,...,bnb];`
- `yhat` — вектор прогнозируемых значений выхода (соответствующий вектору `z(k,1)`);
- `P`, `phi`, `psi` — конечные значения матрицы ковариаций оценок коэффициентов, вектора данных и вектора-градиента.

Функция `rarx` реализует процедуру рекуррентного оценивания коэффициентов AR и ARX-моделей:

```
thm = rarx(z,nn,adm,adg)
[thm,yhat,P,phi] = rarx(z,nn,adm,adg,th0,P0,phi0)
```

Аргументы и возвращаемые величины аналогичны рассмотренным для предыдущей функции.

Функция `rbj` — то же, что предыдущая функция, но для модели Бокса—Дженкинса:

```
thm = rbj(z,nn,adm,adg)
[thm,yhat,P,phi,psi] = ...
rbj(z,nn,adm,adg,th0,P0,phi0,psi0)
```

Функция `roe` — то же, что предыдущая функция, но для OE-модели:

```
thm = roe(z,nn,adm,adg)
[thm,yhat,P,phi,psi] = roe(z,nn,adm,adg,th0,P0,phi0,psi0)
```

Функция `rpem` реализует рекуррентную процедуру оценивания параметров обобщенной линейной модели с несколькими входами, но с одним выходом:

```
thm = rpem(z,nn,adm,adg)
[thm,yhat,P,phi,psi] = rpem(z,nn,adm,adg,th0,P0,phi0,psi0)
```

Функция `rplr` имеет то же назначение, что и предыдущая функция. Отличие состоит в алгоритме оценивания. Функция записывается в виде:

```
thm = rplr(z,nn,adm,adg)
[thm,yhat,P,phi] = rplr(z,nn,adm,adg,th0,P0,phi0)
```

Функция `segment` осуществляет разбиение (сегментацию) данных и оценивание коэффициентов моделей для каждого сегмента в предположении, что в пределах каждого сегмента коэффициенты модели являются постоянными, но могут изменяться (скачкообразно) при переходе от сегмента к сегменту. Оцениваемые модели — AR, ARX, ARMAX. Функция записывается в виде:

```
segm = segment(z,nn)
[segm,V,thm,R2e] = segment(z,nn,R2,q,Rl,M,th0,P0,ll,mu)
```

Здесь:

- `z` — матрица данных;
- аргумент `nn` определяет порядок модели:
  - для ARMAX-модели `nn = [na nb nc nk];`
  - для ARX-модели `nn = [na nb nk];`
  - для AR-модели `nn = na;`
- `R2` — оценка дисперсии шума наблюдений;
- `q` — вероятность скачкообразного изменения модели в момент времени k, по умолчанию 0.01;
- `Rl` — предполагаемая матрица ковариаций оценок коэффициентов при их возможном скачкообразном изменении, по умолчанию — единичная матрица;
- `M` — число параллельных моделей, используемых в алгоритме, по умолчанию 5;
- `th0` и `P0` — смысл данных аргументов рассмотрен выше, при описании функции `rarmax` (здесь начальное значение `P0` — единичная матрица, умноженная на 10);
- `ll` — аргумент, определяющий существование параллельных моделей (каждая модель не исключается, по крайней мере, в течение `ll` итераций), по умолчанию `ll = 1`;
- `mu` — параметр алгоритма оценивания матрицы `R2`, по умолчанию 0.97.

Возвращаемые величины:

- `segm` — матрица, k-я строка которой содержит оценки коэффициентов, соответствующие моменту времени k (как возвращаемая матрица в функциях `rarx` и `rarmax`);
- `thm` — матрица, содержащая оценки коэффициентов, соответствующие моменту времени k без учета сегментации данных;

- V — сумма квадратов ошибок сегментированной модели;
- R2e — полученная оценка матрицы R2 для момента времени k.

Проиллюстрируем возможности функции на примере аппроксимации синусоидального сигнала кусочно-постоянным, при этом на каждом сегменте принятая модель имеет вид

$$y(t) = b_1 \cdot 1.$$

Соответствующая программа приведена ниже, результаты отражены на рис. 6.21.

```
>> y = sin([1:50]/3)';
>> thm = segment([y,ones(size(y))],[0 1 1],0.1);
>> plot([thm,y])
```

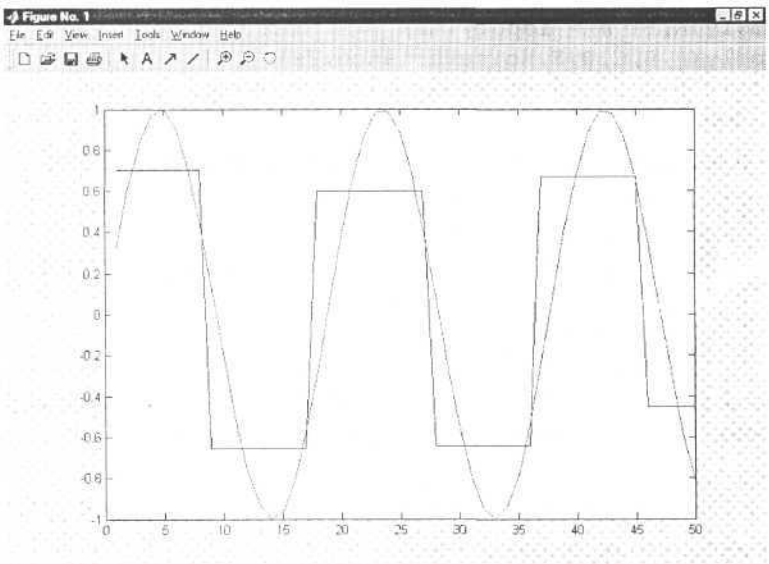

Рис. 6.21. Результаты аппроксимации синусоидального сигнала кусочно-постоянным, полученные с помощью функции segment

6.6.7. Задания структуры модели

Для задания структуры модели служат 6 функций.

Функция idarx — создает ARX модель с многими переменными:

```
m = idarx(A,B,Ts)
m = idarx(A,B,Ts,'Prop1',Value1,...,,'PropN',ValueN)
```

Во втором варианте записи функции возможно задание свойств Propi и соответствующих значений Valuei.

Функция idfrd — создает идентифицированную частотную зависимость для объекта данных:

```
h = idfrd(Response,Freqs,Ts)      h = idfrd(mod)
```

```
h = idfrd(mod,Freqs) h = idfrd(Response,Freqs,Ts,
'CovarianceData',Covariance, 'SpectrumData', Spec,
'NoiseCovariance',Speccov,'prop1', Value1, 'PropN',ValueN)
```

Функция idgrey — полутоновый «ящик» линейной модели, используя m-файл, который вы записываете:

```
m = idgrey(MfileName,ParameterVector,CDmfile)
m = idgrey(MfileName,ParameterVector,CDmfile,FileArgument,
Ts,'Prop1',Value1,...,'PropN',ValueN)
```

Структура файла следующая:

```
[A,B,C,D,K,X0] = mymfile(pars,Tsm,Auxarg)
```

Функция idpoly — создает структуру для моделей входа-выхода, определенную как отношение полиномов числителя и знаменателя:

```
m = idpoly(A,B) m = idpoly(mi)
m = idpoly(A,B,C,D,F,NoiseVariance,Ts)
m = idpoly(A,B,C,D,F,NoiseVariance,Ts,'Prop1',Value1,
'PropN',ValueN)
```

Пример применения функции idpoly наглядно поясняет ее суть:

```
>> A = [1 -1.5 0.7]; B = [0 1 0.5]; C = [1 -1 0.2];
>> m0 = idpoly(A,B,C)
Discrete-time IDPOLY model: A(q)y(t) = B(q)u(t) + C(q)e(t)
A(q) = 1 - 1.5 q^-1 + 0.7 q^-2
B(q) = q^-1 + 0.5 q^-2
C(q) = 1 - q^-1 + 0.2 q^-2
```

Функция idss — создает структуру для линейных моделей пространства состояний с определенными и неопределенными параметрами:

```
m = idss(A,B,C,D) mss = idss(m1)
m = idss(A,B,C,D,K,x0,Ts,'Prop1',Value1,...,'PropeN',ValueN)
```

Пример применения функции idss:

```
A = [-0.2, 0; 0, -0.3]; B = [2;4]; C=[1, 1]; D = 0;
m0 = idss(A,B,C,D)
State-space model: x(t+Ts) = A x(t) + B u(t) + K e(t)
                           y(t) = C x(t) + D u(t) + e(t)
A =
                      x1          x2
           x1       -0.2           0
           x2          0        -0.3
B =
                      u1
           x1          2
           x2          4
C =
                      x1          x2
           y1          1           1
D =
                      u1
```

```
            y1              0
K =
                           y1
            x1              0
            x2              0
x(0) =
            x1              0
            x2              0
This model was not estimated from data.
Sampling interval: 1
```

В более ранние версии пакета был введен ряд функций, которые поддерживаются новой реализацией System Identification 5.0, хотя и не входят в список функций этой реализации. Хотя эти функции в новых программах применять не рекомендуется, мы отметим их, поскольку число пользователей более ранними реализациями пакета System Identification значительно.

Функция `arx2th` конструирует матрицу модели так называемого тета-формата по полиномам ARX-модели многомерного объекта:

```
th = arx2th(A,B,ny,nu)  th = arx2th(A,B,ny,nu,lam,T)
```

Исходная ARX-модель многомерного объекта имеет вид:

$$y(t) + A_1 y(t-1) + A_2 y(t-2) + \ldots + A_{na} y(t-na) = B_0 u(t) + B_1 u(t-1) + \ldots B_{nb} u(t-nb) + e(t),$$

где $y(t)$ — вектор-столбец с `ny` элементами, $u(t)$ — вектор-столбец с `nu` элементами (то есть объект имеет `nu` входов и `ny` выходов), A_k и B_k — матрицы размеров `ny×ny` и `nu×nu` соответственно, $e(t)$ — вектор шума.

Аргументы функции следующие:

- `A = [I A1 A2 ... Ana]`, где `I` — единичная матрица;
- `B = [B0 B1 ... Bnb]`;
- `ny` и `nu` — количество выходов и входов модели (объекта);
- `lam` — матрица ковариаций шума $e(t)$, по умолчанию — единичная матрица;
- `T` — интервал дискретизации.

Функция `canform` конструирует каноническую форму модели для переменных состояния многомерного объекта:

```
ms = canform(orders,nu)  ms = canform(orders,nu,dkx)
```

Используется совместно с функцией `mf2th` (см. ниже). Возвращаемая величина `ms` — матрица, отражающая информацию о структуре модели.

Функция `mf2th` преобразует структуру модели для переменных состояния в модель тета-формата:

```
th = mf2th(model,cd,parval)
th = mf2th(model,cd,parval,aux,lambda,T)
```

где:

- `model` — имя M-файла, содержащего информацию о том, как матрицы модели зависят от свободных (оцениваемых) параметров;
- `cd` — строковая переменная, задающая тип модели:
- `cd='c'` — модель для непрерывного времени,
- `cd='d'` — модель для дискретного времени;

- `parval` — вектор-строка, содержащий номинальные значения параметров;
- `T` — интервал дискретизации;
- `aux` — матрица вспомогательных переменных, используемых для различных целей и задаваемых в файле `model`;
- `lambda` — интенсивность шума.

Возвращаемая величина `th` — модель в тета-формате.

Функция `modstruc` определяет структуру модели для переменных состояния с частично незаданными коэффициентами матриц модели:

```
ms = modstruc(A,B,C,D,K) ms = modstruc(A,B,C,D,K,x0)
```

Аргументы `A`, `B`, `C`, `D`, `K`, `x0` — соответственно матрицы модели и вектор начальных условий (по умолчанию — нулевой). Известные коэффициенты матриц задаются числами, неизвестные (свободные), подлежащие оцениванию, — как `NaN`. Возвращаемая матрица `ms` содержит информацию о структуре модели для дальнейшего использования с помощью функции `ms2th`.

В следующем примере системы 2-го порядка неизвестные элементы содержатся в матрицах **A** и **B**:

```
>> A=[NaN,0;0,NaN]; B=[NaN;NaN]; C=[1,1]; D=0;
>> K=[0;0]; ms=modstruc(A,B,C,D,K)
ms =
    as: [2x2 double]
    bs: [2x1 double]
    cs: [1 1]
    ds: 0
    ks: [2x1 double]
    x0s: [2x1 double]
```

Функция `ms2th` преобразует матрицу структуры в модель тета-формата:

```
th = ms2th(ms) th = ms2th(ms,cd,parval,lambda,T)
```

Для примера определим следующую структуру модели (для непрерывного времени)

$$\dot{x} = \begin{bmatrix} 0_1 & 0 \\ 0 & 0_2 \end{bmatrix} x + \begin{bmatrix} 0_3 \\ 0_4 \end{bmatrix} u,$$

$$y = [1\ 1]x + e$$

с начальными значениями коэффициентов `0 = [-0.2 -0.3 2 4]` и оценим соответствующие свободные параметры:

```
>> load dryer2; z=[y2 u2];A=[NaN,0;0,NaN];B=[NaN;NaN];
>> C=[1,1]; ms=modstruc(A,B,C,0,[0;0]);
>> th=ms2th(ms,'c',[-0.2,-0.3,2,4]);
>> th=pem(z,th); present(th)
State-space model:    dx/dt = A x(t) + B u(t) + K e(t)
                      y(t) = C x(t) + D u(t) + e(t)
A =
          x1                                  x2
   x1     -0.33353 +-2.2e+002                 0
   x2              0                  -0.35576 +-2.4e+002
```

```
B =
                    u1
        x1      6.7246 +-1.3e+005
        x2     -6.8239 +-1.3e+005
C =
                    x1                  x2
        y1           1                   1
D =
                    u1
        y1           0
K =
                    y1
        x1           0
        x2           0
x(0) =
        x1           0
        x2           0
Estimated using PEM from data set z
Loss function 0.143875 and FPE 0.145031
Created:          29-Mar-2002 00:49:45
Last modified:    29-Mar-2002 00:49:45
```

Функция poly2th конструирует модель тета-формата из исходной модели «вход-выход»:

```
th = poly2th(A,B) th = poly2th(A,B,C,D,F,lam,T)
```

Исходная модель такая же, как и для рассмотренной функции pem (см. выше). Аргументы функции следующие:

- A, B, C, D, F — векторы-строки, содержащие коэффициенты соответствующих полиномов;
- T — интервал дискретизации;
- lam — дисперсия шума модели.

Возвращаемая величина th — модель в тета-формате.

6.6.8. Манипуляции с моделями

Функции get и set соответственно получают и модифицируют структуры моделей:

```
Value = get(m,'PropertyName') get(m) Struct = get(m)
set(m,'Prop',Value) set(m,'Prop1',Value1,...'PropN',ValueN)
set(m,'Prop') set(m)
```

Функция init — выбирает или задает случайными начальные значения параметров:

```
m = init(m0) m = init(m0,R,pars,sp)
```

Здесь:

m0 — вектор номинальных параметров;
R — вектор вариаций при случайных изменениях параметров;

`pars` — функция, определяющая параметр k в выражении `pars(k) + e*sqrt(R(k))`;

`sp` — параметр, задающий тип стабилизации (значение 'b', 's' и 'p').

Функция `merge` объединяет различные множества данных в одно множество:

```
dat = merge(dat1,dat2,....,datN)
```

6.6.9. Выбор структуры модели

Для выбора структуры модели служат 6 функций.

Функция `arxstruc` вычисляет функции потерь для ряда различных конкурирующих ARX-моделей с одним выходом:

```
v = arxstruc(ze,zv,NN)  v = arxstruc(ze,zv,NN,maxsize)
```

где:

- `ze` и `zv` — соответственно, матрицы экспериментальных данных для оценивания и верификации (проверки качества) моделей;
- `NN` — матрица задания конкурирующих структур со строками вида `nn = [na nb nk];`
- `maxsize` — служебный аргумент, определяющий максимально возможную размерность задачи.

Возвращаемая величина `v` — матрица, первые (верхние) элементы каждого столбца которой (кроме последнего) являются значениями функции потерь для ARX-моделей, структура которых отображается последующими элементами столбцов (то есть каждый столбец соответствует одной модели). Первый элемент последнего столбца — число значений экспериментальных данных для верификации моделей. Последующий отбор наилучшей модели может быть произведен с помощью функции `selstruc` (см. ниже).

Функция `ivstruc` вычисляет среднее квадратичное отклонение выходов объекта и ряда различных конкурирующих ARX-моделей:

```
v = ivstruc(ze,zv,NN)  v = ivstruc(ze,zv,NN,p,maxsize)
```

Рассматриваемая функция по назначению аналогична предыдущей, но отличается используемым алгоритмом оценивания — в данном случае используется метод IV. Аргументы (кроме `p`) аналогичны аргументам функции `arxstruc`. Аргумент разрешает или запрещает (`p=0`) вывод так называемых условных чисел IV-матрицы. Возвращаемая величина `v` практически идентична матрице, возвращаемой предыдущей функцией, но отличается от последней дополнительной (нижней) строкой, в которой приводятся условные числа IV-матрицы.

Функция `selstruc` осуществляет выбор наилучшей структуры модели из ряда возможных вариантов:

```
[nn,vmod] = selstruc(v)  [nn,vmod] = selstruc(v,c)
```

Здесь:

- `v` — матрица, возвращаемая функциями `arxstruc` или `ivstruc`;
- `c` — строковая переменная, определяющая вывод графика или критерий отбора наилучшей структуры:
- при `c = 'plot'` выводится график зависимости функции потерь от числа оцениваемых коэффициентов модели;

- если c = 'log' выводится график логарифма функции потерь;
- при c = 'aic' график не выводится, но возвращается структура, минимизирующая так называемый теоретический информационный критерий Акейке (Akaike's Information Theoretic Criterion — AIC)

$$V_{mod} = V(1 + 2(d/N)),$$

где V — значение функции потерь, d — число оцениваемых коэффициентов модели, N — объем экспериментальных данных, используемых для оценивания;

- при c = 'mdl' возвращается структура, обеспечивающая минимум так называемого критерия Риссанена минимальной длины описания (Rissanen's Minimum Description Length — MDL)

$$V_{mod} = V^*(1 + \log(N)\ (d/N));$$

- при c, равном некоторому численному значению a, выбирается структура, которая минимизирует

$$V_{mod} = V(1 + a(d/N)).$$

Возвращаемые величины:

- nn — выбранная структура;
- vmod — значение соответствующего критерия.

Рассмотрим следующий пример. Используя экспериментальные данные файла dryer2.mat, сравним ARX-модели порядков 1–5 с единичной задержкой и произведем отбор лучшей структуры:

```
>> load dryer2; z=[y2 u2]; NN=struc(1:5,1:5,1);
>> V=arxstruc(z(1:200,:),z(201:400,:),NN);
>> nn=selstruc(V,0); th=arx(z,nn); present(th)
Discrete-time IDPOLY model: A(q)y(t) = B(q)u(t) + e(t)
A(q)= 1 - 0.9662 (+-0.0308) q^-1 + 0.03241 (+-0.04218) q^-2
      + 0.02569 (+-0.04173) q^-3 + 0.1168 (+-0.03187) q^-4
      - 0.05259 (+-0.01207) q^-5
B(q) = 0.0001676 (+-0.000999) q^-1 + 0.005973 (+-0.001201)
     q^-2 + 0.06419 (+-0.001212) q^-3 + 0.06169 (+-
      0.002283) q^-4 + 0.02081 (+-0.002599) q^-5
Estimated using ARX from data set z
Loss function 0.00139879 and FPE 0.00142705
Sampling interval: 1
Created:          29-Mar-2002 00:53:32
Last modified:    29-Mar-2002 00:53:32
```

Как видно, в данном случае наилучшей оказалась модель 5-го порядка.
Функция n4sid вычисляет порядок модели состояния:

```
m = n4sid(data)
m = n4sid(data,order,'Prop1',Value1,...,'PropN',ValueN)
```

Функция pem дает оценку параметров линейных моделей общего вида. Ее основные формы записи:

```
m = pem(data,mi) m = pem(data,orders)
m = pem(data,'nx',ssorder)
```

и др.

6.6.10. Преобразования модели

Для преобразования моделей имеется ряд функций.

Функция `arxdata` вычисляет параметры ARX-модели:

```
[A,B] = arxdata(m) [A,B,dA,dB] = arxdata(m)
```

Функция `idmodred` понижает порядок `ORDER` модели `M` (в тета-формате):

```
MRED = idmodred(M,ORDER,'DisturbanceModel','None')
```

Если `ORDER=[ ]` (по умолчанию), то выводится график, показывающий, как управляемость и наблюдаемость зависят от порядка модели.

Функция `c2d` преобразует непрерывное время в дискретное с методом `'zoh'` (по умолчанию) или `'foh'`:

```
md = c2d(mc,T) md = c2d(mc,T,method)
```

Обратное преобразование обеспечивает функция d2c:

```
mc = d2c(md)
mc = d2c(md,'CovarianceMatrix',cov,'InputDelay',inpd)
```

Функцмя `freqresp` вычисляет частотный отклик модели:

```
H = freqresp(m) [H,w,covH] = freqresp(m,w)
```

Функция `idfrd` — преобразует ID-модель в модель IDFRD объекта, строящего частотные и спектральные функции. Основные формы записи этой функции:

```
h = idfrd(Response,Freqs,Ts) h = idfrd(mod)
h = idfrd(mod,Freqs)
```

Функция `noisecnv` преобразует зашумленные входы в каналы измерения:

```
mod1 = noisecnv(mod) mod2 = noisecnv(mod,'norm')
```

Функция для заданной модели m вычисляет матрицы пространства состояний:

```
[A,B,C,D,K,X0] = ssdata(m)
[A,B,C,D,K,X0,dA,dB,dC,dD,dK,dX0] = ssdata(m)
```

Для вычисления передаточной характеристики служит функция `tfdata`:

```
[num,den] = tfdata(m) [num,den,sdnum,sdden] = tfdata(m)
[num,den,sdnum,sdden] = tfdata(m,'v')
```

Ряд функция `ss`, `tf`, `zpk` и `frd` служит для преобразований моделей idmodel в LTI-объект. Эти функции используются совместно с пакетом расширения Control Systems Toolbox.

Функция `zpkdata` вычисляет нули, полюса и коэффициенты передачи.

```
[z,p,k] = zpkdata(m) [z,p,k,dz,dp,dk] = zpkdata(m)
[z,p,k,dz,dp,dk] = zpkdata(m,'v')
```

Предполагается ввод в состав данных функций функции `polydata` для вычисления полиномиальных функций передачи.

6.6.11. Анализ модели

Восемь функций введено для анализа моделей. Это довольно важные функции, заслуживающие внимательного рассмотрения.

Функция bode обеспечивает построение диаграмм Боде по заданной модели m или по ряду моделей:

```
bode(m) [mag,phase,w] = bode(m)
[mag,phase,w, sdmag, sdphase] = bode(m)
bode(m1,m2,m3,...,w)
bode(m1,'PlotStyle1',m2,'PlotStyle2',...)
bode(m1,m2,m3,..'sd',sd,'mode',mode,'ap',ap)
```

Входные и выходные параметры этой функции вполне очевидны. Отметим лишь, что sdmag и sdphase означают стандартные отклонения магнитуды mag и фазы phase. Примеры применения этой функции уже приводились в разделе непараметрического оценивания моделей.

Функция compare обеспечивает сравнение выходных сигналов data для iddata модели с расчетными для модели m:

```
compare(data,m) compare(data,m,k,sampnr,init)
compare(data,m1,m2,...,mN,Yplots)
compare(data,m1,'PlotStyle1',...,mN,'PlotStyleN',k,sampnr,init)
[yh,fit] = compare(data,m1,'PlotStyle1',...,
mN,'PlotStyleN',k,sampnr,init)
```

Функция ffplot строит АЧХ и спектрограмму для заданной модели m или для ряда моделей:

```
ffplot(m) [mag,phase,w] = ffplot(m)
[mag,phase,w,sdmag,sdphase] = ffplot(m)
ffplot(m1,m2,m3,...,w)
ffplot(m1,'PlotStyle1',m2,'PlotStyle2',...)
ffplot(m1,m2,m3,..'sd',sd,'mode',mode,'ap',ap)
```

Функция impulse обеспечивает построение импульсной характеристики, а функция step — построение переходной характеристик. Эти функции уже рассматривались в разделе непараметрического оценивания.

Функция nyquist служит для построения диаграммы Найквиста:

```
nyquist(m) [fr,w] = nyquist(m)
fr,w,covfr] = nyquist(m)
nyquist(m1,m2,m3,...,w)
nyquist(m1,'PlotStyle1',m2,'PlotStyle2',...)
nyquist(m1,m2,m3,..'sd*5',sd,'mode',mode)
```

Функция present(m) дает представление модели m на экране дисплея. Применение этой функции было описано в разделе выбора структуры модели.

Функция pzmap служит для построения диаграммы нулей и полюсов модули:

```
pzmap(m) pzmap(m,'sd',sd) pzmap(m1,m2,m3,...)
pzmap(m1,'PlotStyle1',m2,'PlotStyle2',...,'sd',sd)
pzmap(m1,m2,m3,..,'sd',sd,mode,axis)
```

Следующий пример демонстрирует применение функции pzmap:

```
>> load dryer2; z=[y2 u2]; NN=struc(1:5,1:5,1);
>> V=arxstruc(z(1:200,:),z(201:400,:),NN);
>> nn=selstruc(V,0); th=arx(z,nn); pzmap (th)
```

Рис. 6.22 показывает построение нулей (кружки) и полюсов (крестики) для данного примера. Показаны в детальном представлении только нули и полюса, входящие в круг единичного радиуса.

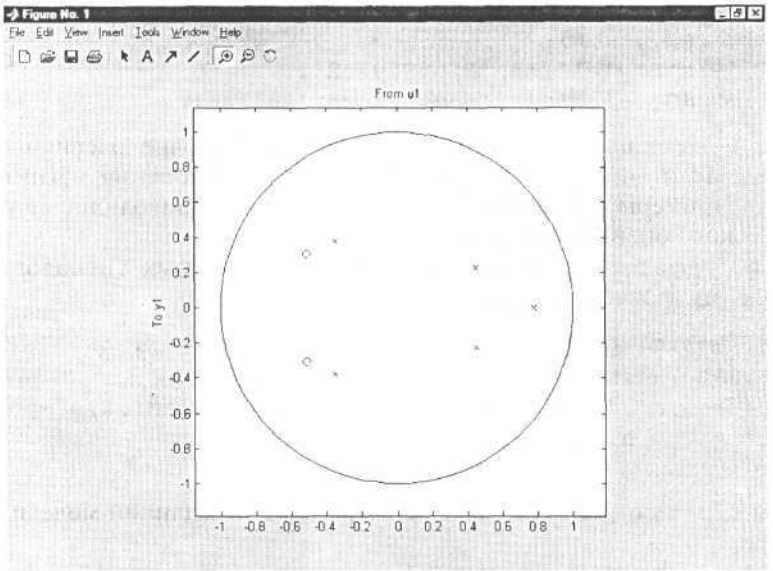

Рис. 6.22. Выходы объекта и модели, отображаемые с помощью функции compare

Функция `view` строит характеристики модели, используя LTI-вьювер расширения Control System Toolbox.

6.6.12. Извлечение информации о модели

Для извлечения информации о модели служат 6 функций:

- `getmfth` — возвращает имя M-файла, созданного функцией `mf2th`;
- `getncap` — возвращает количество данных эксперимента, использованных для оценивания модели;
- `getff` — возвращает значения АЧХ и ФЧХ для заданных частотного диапазона и модели в частотном формате;
- `gett` — возвращает значение интервала дискретизации, заданного для модели;
- `getzp` — возвращает значения нулей и полюсов модели из матрицы zepo-формата, созданной с применением функции `th2zp`;
- `th2par` — возвращает параметры модели исходя из ее тета-представления.

Для примера создадим ARX-модель 3-го порядка и выведем числовые значения ее нулей и полюсов:

```
>> load dryer2; z=[y2 u2];th=arx(z,[3 2 1]);
>> [zepo,k]=th2zp(th); [ze,po] = getzp(zepo)
```

```
>> w' % Pertransposition
Transfer function:
       -s
-----------------
  s^2 - 2 s + 10
>> inv(w)%Invertion
Transfer function:
  s^2 + 2 s + 10
-----------------
        s
>> w^2 %Power
Transfer function:
             s^2
--------------------------------
s^4 + 4 s^3 + 24 s^2 + 40 s + 100
```

Создадим модель дискретного объекта с двумя входами и двумя выходами:

```
>> nums={1 [1 0];[-1 2] 3}; Ts=0.2; W=tf(nums,[1 0.3],Ts)
Transfer function from input 1 to output...
         1
#1: ---------
     z + 0.3
      -z + 2
#2: ---------
     z + 0.3
Transfer function from input 2 to output...
        z
#1: ---------
     z + 0.3
        3
#2: ---------
     z + 0.3
Sampling time: 0.2
>> W' %Pertransposition
Transfer function from input 1 to output...
          z
#1: ------------
      0.3 z + 1
         1
#2: ------------
     0.3 z + 1
Transfer function from input 2 to output...
       2 z - 1
#1: ------------
     0.3 z + 1
        3 z
#2: ------------
     0.3 z + 1
Sampling time: 0.2
```

```
>> inv(W) %Inversion
Transfer function from input 1 to output...
        3 z + 0.9
#1: -----------------
      z^2 - 2 z + 3
     z^2 - 1.7 z - 0.6
#2: --------------------
       z^2 - 2 z + 3
Transfer function from input 2 to output...
     -z^2 - 0.3 z + 6.328e-016
#1: --------------------------
           z^2 - 2 z + 3
       z + 0.3
#2: ----------------
     z^2 - 2 z + 3
Sampling time: 0.2
```

7.3.5. Модели для переменных состояния

Теперь рассмотрим (см. ниже) функции моделей для переменных состояния. Функция `canon` возвращает так называемую каноническую форму ss-модели:

```
csys = canon(sys,'type') [csys,T] = canon(sys,'type')
```

Функцией поддерживаются две канонические формы: модальная и присоединенная.

В модальной канонической форме действительные собственные значения матрицы **A** расположены на ее главной диагонали, а паре комплексно-сопряженных собственных значений соответствует блок размера 2 × 2, также расположенный на диагонали этой матрицы. Для системы с собственными значениями (λ_1, $\sigma \pm j\omega$, λ_2) модальная матрица **A** имеет вид

$$\begin{bmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \sigma & \omega & 0 \\ 0 & -\omega & \sigma & 0 \\ 0 & 0 & 0 & \lambda_2 \end{bmatrix}.$$

В присоединенной канонической форме данная матрица имеет вид

$$\begin{bmatrix} 0 & 0 & 0 & \dots & 0 & -a_n \\ 1 & 0 & 0 & \dots & 0 & -a_{n-1} \\ 0 & 1 & 0 & \dots & 0 & -a_{n-2} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 & -a_2 \\ 0 & 0 & 0 & \dots & 1 & -a_1 \end{bmatrix},$$

где a_1–a_n — коэффициенты характеристического многочлена системы:

$$\gamma(p) = p^n + a_1 p^{n-1} + \dots + a_{n-1} p + a_n.$$

Аргументы этой функции следующие:

- sys — имя исходной ss-модели;
- 'type' — строковая переменная, задающая тип канонической формы ('modal' — модальная форма, 'companion' — присоединенная).

Возвращаемые величины: csys — преобразованная модель, T — матрица преобразования, связывающая вектор состояния в канонической форме с вектором состояния исходной модели.

Функция ctrb формирует матрицу управляемости для модели в пространстве состояний:

```
Co = ctrb(A,B) Co = ctrb(sys)
```

Здесь sys — имя ss-модели, A и B — матрицы этой модели.

Система является управляемой, если матрица управляемости имеет полный ранг. Возвращаемая величина Co — матрица управляемости, имеющая n строк и $n \times m$ столбцов (m — количество входов) и описываемая соотношением

$$\mathbf{Co} = [\mathbf{B}\ \mathbf{AB}\ \mathbf{A}^2\mathbf{B}\ \ldots\ \mathbf{A}^{n-1}\mathbf{B}].$$

Для примера проверим, является ли управляемой система 2-го порядка с матрицами

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 4 & -2 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}?$$

Проверка осуществляется с использованием следующих функций:

```
>> A=[1 1;4 -2]; B=[1 -1;1 -1];
>> Co=ctrb(A,B)
Co =
     1    -1     2    -2
     1    -1     2    -2
>> rank(Co)
ans = 1
```

В данном случае ранг матрицы управляемости равен 1, а порядок системы — 2. Следовательно, система не является полностью управляемой.

Функция ctrbf формирует так называемую каноническую форму управляемости:

```
[Abar,Bbar,Cbar,T,k] = ctrbf(A,B,C)
[Abar,Bbar,Cbar,T,k] = ctrbf(A,B,C,tol)
```

Здесь аргумент tol позволяет задать точность вычислений. Если матрица управляемости для пары {**A**, **B**} имеет ранг $r < n$, где n — порядок матрицы **A**, то существует преобразование подобия вида

$$\overline{\mathbf{A}} = \mathbf{TAT}^T, \overline{\mathbf{B}} = \mathbf{TB}, \overline{\mathbf{C}} = \mathbf{CT}^T,$$

где **T** — унитарная матрица, при котором преобразованная система имеет блочно-треугольную форму с неуправляемыми модами, расположенными в верхнем левом углу:

$$\overline{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_{uc} & 0 \\ \mathbf{A}_{21} & \mathbf{A}_c \end{bmatrix}, \quad \overline{\mathbf{B}} = \begin{bmatrix} 0 \\ \mathbf{B}_c \end{bmatrix}, \quad \overline{\mathbf{C}} = [\mathbf{C}_{uc} \quad \mathbf{C}].$$

Такая форма называется канонической формой управляемости.

Пара матриц $\{\mathbf{A}_c, \mathbf{B}_c\}$ является управляемой, так что справедливо соотношение

$$\mathbf{C}_c(p\mathbf{I} - \mathbf{A}_c)^{-1}\mathbf{B}_c = \mathbf{C}(p\mathbf{I} - \mathbf{A})^{-1}\mathbf{B},$$

где $\mathbf{I}$ — единичная матрица, то есть передаточная функция всей системы совпадает с передаточной функцией ее управляемой части, а все моды, соответствующие собственным значениям матрицы $\mathbf{A}_{uc}$, являются неуправляемыми.

Функция

```
[Abar,Bbar,Cbar,T,k] = ctrbf(A,B,C)
```

преобразует ss-модель, описываемую тройкой матриц [A, B, C], в каноническую форму управляемости [Abar, Bbar, Cbar]. Матрица T описывает преобразование подобия, а элементы вектора k указывают количество управляемых мод, выделенных на каждом шаге расчета матрицы преобразования. Число ненулевых элементов вектора k показывает, сколько итераций потребовалось для расчета матрицы T, а величина sum(k) указывает число канонических переменных состояния, соответствующих управляемой части матрицы Abar.

Пример применения функции ctrbf:

```
>> A = [[1 1]; [4 -2]]; B =[[1 -1]; [1 -1]]; C = [[1 0]; [ 0 1]];
>> [Abar,Bbar,Cbar,T,k]=ctrbf(A,B,C)
Abar =
   -3.0000    0.0000
    3.0000    2.0000
Bbar =
         0         0
   -1.4142    1.4142
Cbar =
   -0.7071   -0.7071
    0.7071   -0.7071
T =
   -0.7071    0.7071
   -0.7071   -0.7071
k =
    1     0
```

Функция gram позволяет вычислить функции Грама для оценки управляемости и наблюдаемости системы, называемые соответственно грамианами управляемости и наблюдаемости. Грамианы применяются для исследования свойств управляемости и наблюдаемости моделей систем, заданных в пространстве состояний, а также для построения их минимальных реализаций. Они более удобны для вычислений, чем матрицы управляемости и наблюдаемости (матрица $\mathbf{A}$ модели должна быть устойчивой).

Для непрерывной ss-модели грамиан управляемости определяется интегралом

$$\mathbf{G}_c = \int_0^\infty e^{\mathbf{A}\tau}\mathbf{B}\mathbf{B}^{\mathrm{T}}e^{\mathbf{A}^{\mathrm{T}}\tau}d\tau,$$

а грамиан наблюдаемости — интегралом

$$\mathbf{G}_0 = \int_0^\infty e^{\mathbf{A}^\mathrm{T}\tau}\mathbf{C}^\mathrm{T}\mathbf{C}e^{\mathbf{A}\tau}d\tau.$$

Для дискретных моделей аналогами грамианов управляемости и наблюдаемости служат выражения

$$\mathbf{G}_c = \sum_{k=0}^{\infty}\mathbf{A}^k\mathbf{B}\mathbf{B}^\mathrm{T}(\mathbf{A}^\mathrm{T})^k,\ \mathbf{G}_0 = \sum_{k=0}^{\infty}(\mathbf{A}^\mathrm{T})^k\mathbf{C}^\mathrm{T}\mathbf{C}\mathbf{A}^k.$$

Грамиан управляемости положительно определен тогда и только тогда, когда пара матриц {**A**, **B**} является управляемой. Аналогично грамиан наблюдаемости положительно определен тогда и только тогда, когда пара матриц {**A**, **C**} является наблюдаемой.

Функция `Gc = gram(sys,'c')` вычисляет грамиан управляемости для непрерывной или дискретной ss-модели.

Функция `Go = gram(sys,'o')` вычисляет грамиан наблюдаемости для непрерывной или дискретной ss-модели.

Функция `obsv` формирует матрицу наблюдаемости для модели в пространстве состояний:

```
Ob = obsv(A,C) Ob = obsv(sys)
```

Система является наблюдаемой, если матрица наблюдаемости имеет полный ранг. Возвращаемая величина `Ob` — матрица наблюдаемости, имеющая p строк (p — количество выходов) и n столбцов и описываемая соотношением

$$\mathbf{Ob} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \\ \vdots \\ \mathbf{CA}^{n-1} \end{bmatrix}.$$

Пример применения функции `obsv`:

```
>> A =[[1 1];[4 -2]]; C =[[1 0];[0 1]];
>> Ob = obsv(A,C); unob = length(A)-rank(Ob)
unob = 0
```

Функция `obsvf` формирует так называемую каноническую форму наблюдаемости:

```
[Abar,Bbar,Cbar,T,k] = obsvf(A,B,C)
[Abar,Bbar,Cbar,T,k] = obsvf(A,B,C,tol)
```

Если матрица наблюдаемости для пары {**A**, **C**} имеет ранг $r < n$, где n — порядок матрицы **A**, то существует преобразование подобия вида

$$\overline{\mathbf{A}} = \mathbf{T}\mathbf{A}\mathbf{T}^\mathrm{T},\ \overline{\mathbf{B}} = \mathbf{T}\mathbf{B},\ \overline{\mathbf{C}} = \mathbf{C}\mathbf{T}^\mathrm{T},$$

где **T** — унитарная матрица, при котором преобразованная система имеет блочно-треугольную форму с ненаблюдаемыми модами, расположенными в верхнем левом углу:

$$\overline{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_{no} & \mathbf{A}_{12} \\ 0 & \mathbf{A}_{o} \end{bmatrix}, \quad \overline{\mathbf{B}} = \begin{bmatrix} \mathbf{B}_{no} \\ \mathbf{B}_{o} \end{bmatrix}, \quad \overline{\mathbf{C}} = [0 \quad \mathbf{C}_{o}].$$

Такая форма называется канонической формой наблюдаемости.

Пара матриц $\{\mathbf{A}_o, \mathbf{C}_o\}$ является наблюдаемой, так что справедливо соотношение

$$\mathbf{C}_o(p\mathbf{I} - \mathbf{A}_o)^{-1}\mathbf{B}_o = \mathbf{C}(p\mathbf{I} - \mathbf{A})^{-1}\mathbf{B},$$

то есть передаточная функция всей системы совпадает с передаточной функцией ее наблюдаемой части, а все моды, соответствующие собственным значениям матрицы $\mathbf{A}_{no}$, являются ненаблюдаемыми.

Функция `[Abar,Bbar,Cbar,T,k] = obsvf(A,B,C)` преобразует ss-модель, описываемую тройкой матриц `[A, B, C]`, в каноническую форму наблюдаемости `[Abar, Bbar, Cbar]`. Матрица `T` описывает преобразование подобия, а элементы вектора `k` указывают количество наблюдаемых мод, выделенных на каждом шаге расчета матрицы преобразования. Число ненулевых элементов вектора `k` показывает, сколько итераций потребовалось для расчета матрицы `T`, а величина `sum(k)` указывает число канонических переменных состояния, соответствующих наблюдаемой части матрицы `Abar`.

Пример применения функции `obsvf`:

```
>> A = [[1 1];[4 -2]]; B = [[1 -1];[1 -1]]; C = [[1 0];[0 1]];
>> [Abar,Bbar,Cbar,T,k] = obsvf(A,B,C)
Abar =
     1     1
     4    -2
Bbar =
     1    -1
     1    -1
Cbar =
     1     0
     0     1
T =
     1     0
     0     1
k =
     2     0
```

Функция `ssbal` выполняет масштабирование ss-моделей:

```
[sysb,T] = ssbal(sys)  [sysb,T] = ssbal(sys,condT)
```

Рассматриваемая функция выполняет масштабирование матриц `ss`-модели, используя преобразование подобия с диагональной матрицей $\mathbf{T}$ и скаляром α такими, что матрица

$$\begin{bmatrix} \mathbf{T}\mathbf{A}\mathbf{T}^{-1} & \mathbf{T}\mathbf{B}/\alpha \\ \alpha\mathbf{C}\mathbf{T}^{-1} & 0 \end{bmatrix}$$

имеет малые числа обусловленности по отношению к задаче на собственные значения.

Функция `[sysb,T] = ssbal(sys)` возвращает масштабированную модель `sysb`, описываемую четверкой $\{\mathbf{TAT}^{-1}, \mathbf{TB}/\alpha, \alpha\mathbf{CT}^{-1}, \mathbf{D}\}$, и матрицу преобразования **T**, такую что $\psi = \mathbf{Tx}$, где ψ — новый вектор состояния модели.

Функция `[sysb,T] = ssbal(sys,condT)` задает верхнюю границу числа обусловленности `condT` для матрицы `T`. Поскольку масштабирование при плохо обусловленной матрице `T` может приводить к росту ошибок округления, задание величины `condT` дает возможность контроля данных ошибок.

Функция `ss2ss` осуществляет преобразование ss-модели при переходе к новому базису:

```
sysT = ss2ss(sys,T)
```

Для заданной `ss`-модели `sys` рассматриваемая функция выполняет преобразование вектора состояния $\psi = \mathbf{Tx}$ и создает эквивалентную модель `sysT`, описываемую системой уравнений:

$$\dot{\psi} = \mathbf{TAT}^{-1}\psi + \mathbf{TBu},$$

$$\mathbf{y} = \mathbf{CT}^{-1}\psi + \mathbf{Du}.$$

Функция `sysT = ss2ss(sys,T)` возвращает преобразованную модель `sysT`, используя исходную модель `sys` и матрицу преобразования `T` (данная матрица должна быть невырожденной). Функция применима как к непрерывным, так и к дискретным моделям.

7.3.6. Модели динамики

Модели динамики одни из наиболее применяемых. В группу функций для работы с такими моделями входит 14 функций. Рассмотрим основные из них.

Функция `bandwidth` вычисляет ширину полосы SISO модели `sys`:

```
fb = bandwidth(sys)  fb = bandwidth(sys,dbdrop)
```

Измерения выполняются на уровне −3 дБ в первом случае и `dbdrop` во втором.

Функция `covar` возвращает матрицу ковариаций сигнала на выходе устойчивой линейной модели с постоянными параметрами при действии на входах возмущений типа белого шума:

```
P = covar(sys,W)
[P,Q] = covar(sys,W)
```

Здесь `sys` — имя модели и `W` — матрица интенсивностей входного сигнала. Возвращаемые величины: `P` — матрица ковариаций выходов и `Q` — матрица ковариаций переменных состояний.

Функция `damp` возвращает собственные частоты и коэффициенты демпфирования LTI-модели:

```
[Wn,Z] = damp(sys)
[Wn,Z,P] = damp(sys)
```

Функция `damp` предназначена для расчета собственных чисел и коэффициентов демпфирования, соответствующих полюсам LTI-модели `sys`. Если функция вызывается без выходных аргументов, то на дисплей выводится таблица собственных значений `P`, соответствующих им собственных частот `Wn` и коэффициентов демпфирования `Z`.

Функция dcgain возвращает статический коэффициент передачи (матрицу таких коэффициентов в многомерном случае) LTI-модели:

```
k = dcgain(sys)
```

Рассмотрим пример. Пусть дискретная передаточная функция имеет вид

$$W(z) = \begin{bmatrix} 1 & \dfrac{p-1}{p^2+p+3} \\ \dfrac{1}{p+1} & \dfrac{p+2}{p-3} \end{bmatrix}.$$

Создадим ее tf-модель, а затем найдем матрицу статических коэффициентов усиления:

```
>> W=[1 tf([1 -1],[1 1 3]);tf(1,[1 1]) tf([1 2],[1 -3])]
Transfer function from input 1 to output...
#1: 1
      1
#2: ------
    s + 1
Transfer function from input 2 to output...
        s - 1
#1: ---------------
     s^2 + s + 3
     s + 2
#2: -------
     s - 3
>> cdcgain(W)
ans =
    1.0000    -0.3333
    1.0000    -0.6667
```

Используя свойства преобразования Лапласа, нетрудно убедиться в правильности ответа.

Функция dsort сортирует полюсы дискретной LTI-модели в порядке убывания их модулей. Неустойчивые полюсы располагаются в начале списка:

```
s = dsort(p)
[s,ndx] = dsort(p)
```

Функция eig вычисляет полюсы LTI-модели:

```
d = eig(A)     d = eig(A,B)     [V,D] = eig(A)
[V,D] = eig(A,B)        [V,D] = eig(A,B,flag)
[V,D] = eig(A,'nobalance')
```

Пример применения этой функции для сортировки матрицы B:

```
>> B = [ 3        -2       -.9       2*eps
        -2         4        1        -eps
        -eps/4     eps/2    -1       0
        -.5        -.5       .1       1 ];
>> [VB,DB] = eig(B)
```

```
>> B*VB - VB*DB
>> [VN,DN] = eig(B,'nobalance')
>> B*VN - VN*DN
VB =
  -0.6153    0.4176    0.0000   -0.1475
   0.7881    0.3261    0.0000    0.1298
   0.0000    0.0000   -0.0000   -0.9443
  -0.0189   -0.8481   -1.0000    0.2639
DB =
   5.5616         0         0         0
        0    1.4384         0         0
        0         0    1.0000         0
        0         0         0   -1.0000
ans =
  -0.0000    0.0000    0.0000    0.0000
   0.0000    0.0000    0.0000   -0.0000
   0.0000    0.0000    0.0000    0.0000
   0.0000    0.0000         0    0.4423
VN =
   0.7808   -0.4924         0   -0.1563
  -1.0000   -0.3845   -0.0000    0.1375
  -0.0000   -0.0000   -0.0000   -1.0000
   0.0240    1.0000   -1.0000    0.0453
DN =
   5.5616         0         0         0
        0    1.4384         0         0
        0         0    1.0000         0
        0         0         0   -1.0000
ans = 1.0e-014 *
  -0.2665         0   -0.0323   -0.0028
   0.4441    0.1110    0.0042   -0.0250
   0.0022    0.0002    0.0007         0
   0.0056   -0.0444    0.0444    0.0083
```

Функция `esort` сортирует полюсы непрерывной LTI-модели в порядке убывания значений их действительной части:

```
s = esort(p)  [s,ndx] = esort(p)
```

Неустойчивые полюсы располагаются в начале списка. В этой функции `p` — вектор-строка полюсов модели; `s` — вектор, содержащий отсортированные полюсы; `ndx` — вектор, содержащий индексы, использованные при сортировке.

```
>> p = [
  -0.2410+ 0.5573i
  -0.2410- 0.5573i
   0.1503
  -0.0972
  -0.2590 ]
>> esort(p)
ans =
   0.1503
```

```
-0.0972
-0.2410 + 0.5573i
-0.2410 - 0.5573i
-0.2590
```

Функция `norm` предназначена для вычисления норм типа $\|\ \|_2$ или $\|\ \|_\infty$ для непрерывной или дискретной LTI-модели (норма равна бесконечности для неустойчивых систем):

```
norm(sys)    norm(sys,2)    norm(sys,inf)
norm(sys,inf,tol)           [ninf,fpeak] = norm(sys)
```

Отметим, что норма $\|W(p)\|_2$ устойчивой непрерывной системы с передаточной матрицей $W(p)$ — это квадратный корень из среднего значения квадрата импульсной характеристики системы, а при переходе к преобразованию Лапласа данная норма, в соответствии с теоремой Парсеваля, определяется соотношением

$$\|W(p)\|_2 = \sqrt{\frac{1}{2\pi}\int_{-\infty}^{\infty} \mathrm{tr}\{W^*(j\omega)W(j\omega)\}d\omega},$$

где tr{•} — обозначение следа матрицы, $W^*(j\omega)$ — матрица, сопряженная по отношению к $W(j\omega)$.

Норма $\|\ \|_\infty$ равна максимальному значению модуля частотной характеристики:

- для одномерных моделей

$$\|W(p)\|_\infty = \max_\omega |W(j\omega)|;$$

- для многомерных моделей

$$\|W(p)\|_\infty = \max_\omega \sigma_{\max}[W(j\omega)],$$

где σ_{max} — максимальное сингулярное число матрицы $W(j\omega)$, то есть неотрицательный корень квадратный из максимального собственного числа матрицы $W^*(j\omega)W(j\omega)$;

- для дискретной модели

$$\|W(z)\|_\infty = \max_{0\in[0,\pi 0]} \sigma_{\max}[W(e^{j0})].$$

Аргументы функции `norm` следующие:

- `sys` — имя модели;
- `2` — задание нормы $\|\ \|_2$;
- `inf` — задание нормы $\|\ \|_\infty$;
- `tol` — точность при расчете нормы, по умолчанию `tol = 1e-2`.

Возвращаемые величины:

- `ninf` — норма модели (по умолчанию — $\|\ \|_2$);
- `fpeak` — частота, на которой норма достигает максимального значения.

Пример применения функции norm:

```
>> H = tf([1 -2.841 2.875 -1.004],[1 -2.417 2.003 -0.5488],0.1)
Transfer function:
```

```
z^3 - 2.841 z^2 + 2.875 z - 1.004
---------------------------------
z^3 - 2.417 z^2 + 2.003 z - 0.5488
Sampling time: 0.1
>> norm(H)
ans =
  1.2438
```

Функция pole возвращает вектор, элементами которого являются полюсы LTI-модели:

```
p = pole(sys)
```

Пример:

```
>> p=pole(H)
p =
     0.9054 + 0.2924i
     0.9054 - 0.2924i
     0.6063
```

Функция pzmap возвращает карту нулей и полюсов:

```
pzmap(sys) [p,z] = pzmap(sys)
```

В последней форме записи график не выводится, а возвращаются векторы, элементы которых являются нулями и полюсами модели. Пример:

```
>> pz=pzmap(H)
pz =
     0.9054 + 0.2924i
     0.9054 - 0.2924i
     0.6063
```

Функция rlocus выполняет расчет и построение корневого годографа по известной LTI-модели разомкнутой одномерной системы sys:

```
rlocus(sys) rlocus(sys,k) rlocus(sys1,sys2,...)
[r,k] = rlocus(sys) r = rlocus(sys,k)
```

Здесь sys — имя модели и k — начальное значение коэффициента усиления (вектор-строка таких значений для многомерного объекта) звена обратной связи. Возвращаемые значения:

- график корневого годографа;
- r — массив полюсов при изменении k от 0 до ∞;
- k — соответствующий массив значений коэффициентов усиления.

Рассмотрим следующий пример. Построим корневой годограф системы с передаточной функцией

$$W(p) = \frac{2p^2 + 5p + 1}{p^2 + 2p + 3}.$$

```
>> W=tf([2 5 1],[1 2 3]); rlocus(W)
```

Результат показан на рис. 7.18.

Функция r = roots(c) вычисляет корни полиномов, коэффициенты которых хранятся в векторе c. Например:

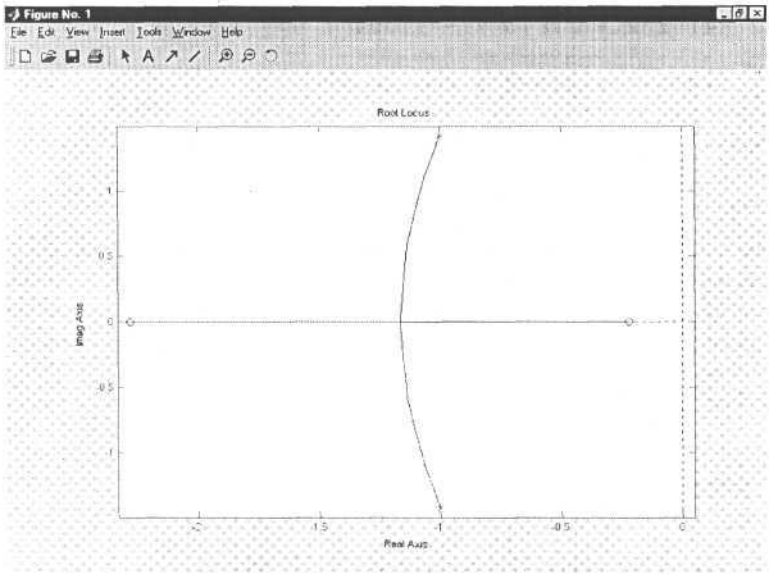

Рис. 7.18. Результат выполнения функции rlocus

```
>> p = [1 -6 -72 -27]; r = roots(p)
r =
   12.1229
   -5.7345
   -0.3884
```

Функция `zero` возвращает вектор, элементами которого являются нули LTI-модели:

```
z = zero(sys) [z,gain] = zero(sys)
```

Пример:

```
>> z=zero(H)
z =
   0.9984 + 0.4388i
   0.9984 - 0.4388i
   0.8442
```

Функция, вызванная в форме `[z,gain] = zero(sys)`, кроме нулей возвращает еще и обобщенный коэффициент передачи модели. Например:

```
>> [z,
ain] = zero(H)
z =
   0.9984 + 0.4388i
   0.9984 - 0.4388i
   0.8442
gain = 1
```

Функция

```
sgrid sgrid(z,wn)
```

обеспечивает построение линий «сетки», например:

```
>> H = tf([2 5 1],[1 2 3])
Transfer function:
2 s^2 + 5 s + 1
---------------
s^2 + 2 s + 3
>> rlocus(H); sgrid
```

Результат показан на рис. 7.19. Сравнение его с рис. 7.18 в особых комментариях не нуждается.

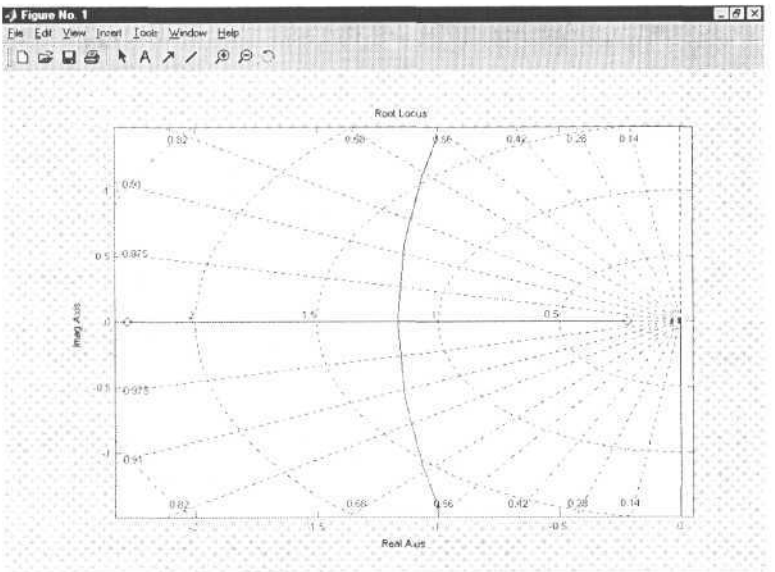

Рис. 7.19. Результат выполнения функций rlocus и sgrid

Еще одна функция обеспечивает построение «сетки» на z-плоскости:

```
zgrid zgrid(z,wn)
```

Следующий пример иллюстрирует это:

```
>> H = tf([2 -3.4 1.5],[1 -1.6 0.8],-1)
Transfer function:
2 z^2 - 3.4 z + 1.5
-------------------
z^2 - 1.6 z + 0.8
Sampling time: unspecified
>> rlocus(H);zgrid; axis('square')
```

Результат построений показан на рис. 7.20.

7.3.7. Моделирование временного отклика систем

В группу функций для расчета и построения временного отклика систем входят следующие 6 функций.

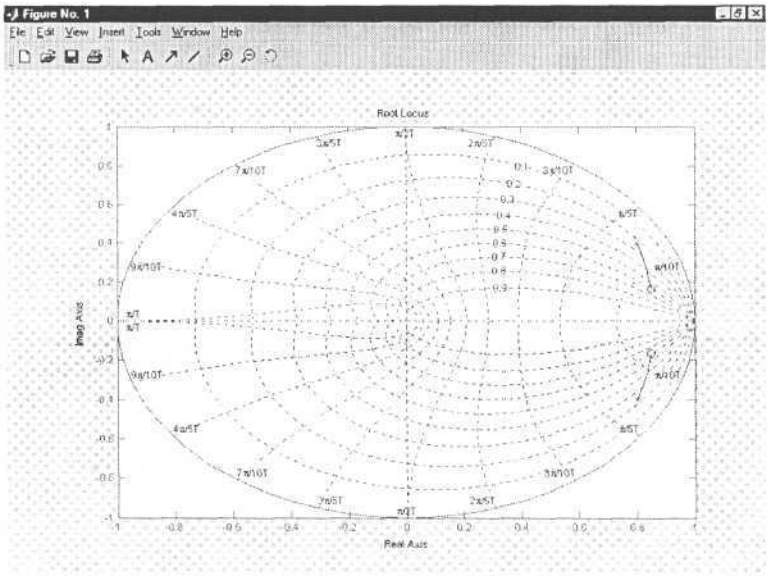

Рис. 7.20. Результат выполнения функций rlocus и zgrid

Функция `step` рассчитывает и строит реакцию модели на единичную ступенчатую функцию, то есть возвращает переходную функцию системы:

```
step(sys) step(sys,t)
step(sys1,sys2,...,sysN) step(sys1,sys2,...,sysN,t)
step(sys1,'PlotStyle1',...,sysN,'PlotStyleN')
[y,t,x] = step(sys)
```

Для моделей, заданных в пространстве состояний, начальные условия принимаются нулевыми. Аргументы функции следующие:

- `sys`, `sys1`, `sys2`, ..., `sysN` — имена моделей, для которых строятся переходные функции (все модели должны иметь одинаковое число входов и выходов);
- `t` — аргумент, задающий момент окончания моделирования — либо в форме `t = Tfinal` (в секундах), либо в форме `t = 0:dt:Tfinal`. Для дискретных моделей значение `dt` должно равняться интервалу дискретизации, для непрерывных моделей — быть достаточно малым, чтобы учесть наиболее быстрые изменения переходного процесса;
- `'PlotStyle1'`, `'PlotStyle2'`, ..., `'PlotStyleN'` — строковые переменные, задающие стили графиков (типы линий) при выводе нескольких графиков одновременно.

Возвращаемые величины:

- график (графики) переходных процессов;
- `y`, `x`, `t` — соответственно векторы, содержащие значения переходного процесса, переменных состояния и моментов времени; при возвращении данных величин график переходного процесса не отображается.

В качестве примера применения этой функции построим графики переходных процессов для следующей модели 2-го порядка:

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix}=\begin{bmatrix}-0{,}5572 & -0.7814\\ 0.7814 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}+\begin{bmatrix}1 & -1\\ 0 & 2\end{bmatrix}\begin{bmatrix}u_1\\ u_2\end{bmatrix},$$

$$y=\begin{bmatrix}1.9691 & 6.4493\end{bmatrix}\begin{bmatrix}x_1\\ x_2\end{bmatrix}.$$

```
>> a=[-0.5572 -0.7814;0.7814 0];
>> b=[1 -1;0 2]; c=[1.9691 6.4493];
>> sys=ss(a,b,c,0); step(sys)
```

Графики данных процессов приведены на рис. 7.21.

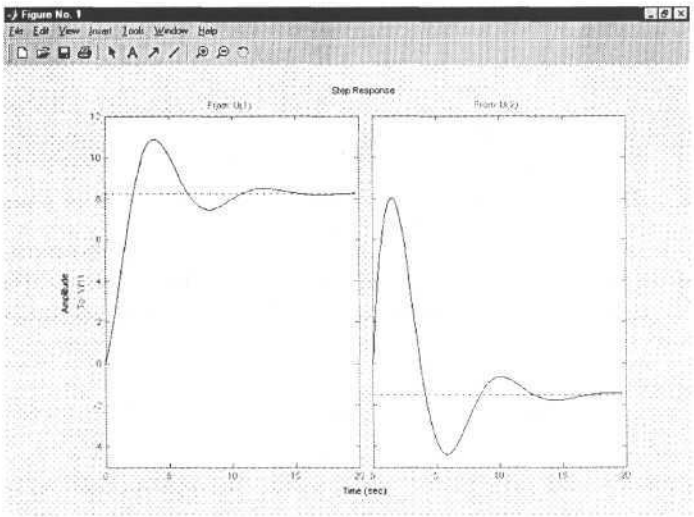

Рис. 7.21. Переходная характеристика системы 2-го порядка

Функция `impulse` возвращает импульсную характеристику модели или нескольких моделей:

```
impulse(sys) impulse(sys,t)
impulse(sys1,sys2,...,sysN) impulse(sys1,sys2,...,sysN,t)
impulse(sys1,'PlotStyle1',...,sysN,'PlotStyleN')
[y,t,x] = impulse(sys)
```

Смысл аргументов и возвращаемых величин — как у предыдущей функции, за тем исключением, что здесь все относится к ИХ.

Для ранее заданной модели имеем (рис. 7.22):

```
>> sys = ss(a,b,c,0); impulse(sys)
```

Функция `initial` возвращает графики переходных процессов для ss-модели при ненулевых начальных условиях:

```
initial(sys,x0) initial(sys,x0,t)
initial(sys1,sys2,...,sysN,x0)
initial(sys1,sys2,...,sysN,x0,t)
initial(sys1,'PlotStyle1',...,sysN,'PlotStyleN',x0)
[y,t,x] = initial(sys,x0)
```

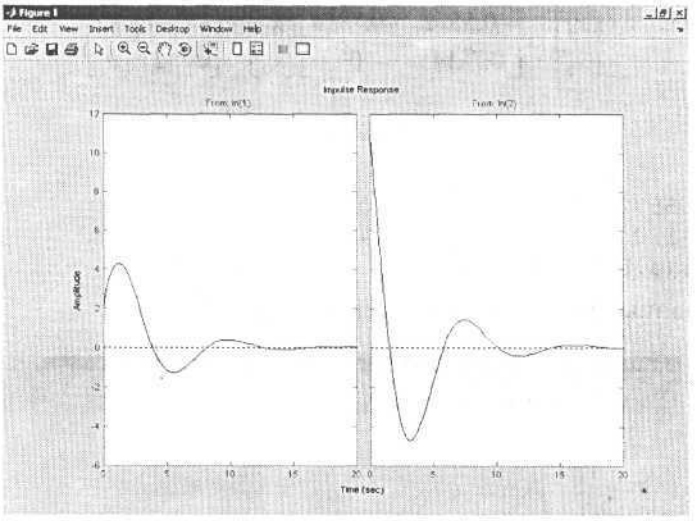

Рис. 7.22. Импульсная характеристика системы 2-го порядка

Здесь `x0` — вектор-строка начальных условий для переменных состояния. Смысл остальных аргументов и возвращаемых величин — как у функции `step`.

Функция `lsim` возвращает реакцию LTI-модели на произвольное входное воздействие:

```
lsim(sys,u,t) lsim(sys,u,t,x0)
lsim(sys1,sys2,...,sysN,u,t)
lsim(sys1,sys2,...,sysN,u,t,x0)
lsim(sys1,'PlotStyle1',...,sysN,'PlotStyleN',u,t)
[y,t,x] = lsim(sys,u,t,x0)
```

Здесь `t` — вектор-строка моментов времени, `u` — вектор-строка значений входного сигнала, остальные аргументы и возвращаемые величины — как у предыдущих функций данной группы.

Вместе с описанными функциями часто используются функции генерации тестовых временных зависимостей. Так, функция `gensig` генерирует скалярный сигнал заданного типа и с заданным периодом:

```
[u,t] = gensig(type,tau) [u,t] = gensig(type,tau,Tf,Ts)
```

Здесь:

- `type` — строковая переменная, задающая тип генерируемого сигнала и имеющая следующие возможные значения:`'sin'` — синусоида, `'square'` — периодический прямоугольный сигнал, `'pulse'` — периодические импульсы. Все генерируемые сигналы имеют единичную амплитуду;
- `tau` — период генерируемого сигнала (в секундах);
- `Tf` — продолжительность сигнала, с;
- `Ts` — период дискретности (в секундах) для генератора импульсов.

Возвращаемые величины:

- `t` — вектор значений времени;
- `u` — соответствующий вектор значений генерируемого сигнала.

Рассмотрим следующий пример. Используем рассматриваемую функцию для генерации периодического прямоугольного сигнала с периодом 5 с, продолжительностью 30 с и периодом дискретизации 0,1 с:

```
>> [u,t]=gensig('square',5,30,0.1);
```

Отобразим теперь график сигнала (рис. 7.23):

```
>> plot(t,u);axis([0 30 -1 2])
```

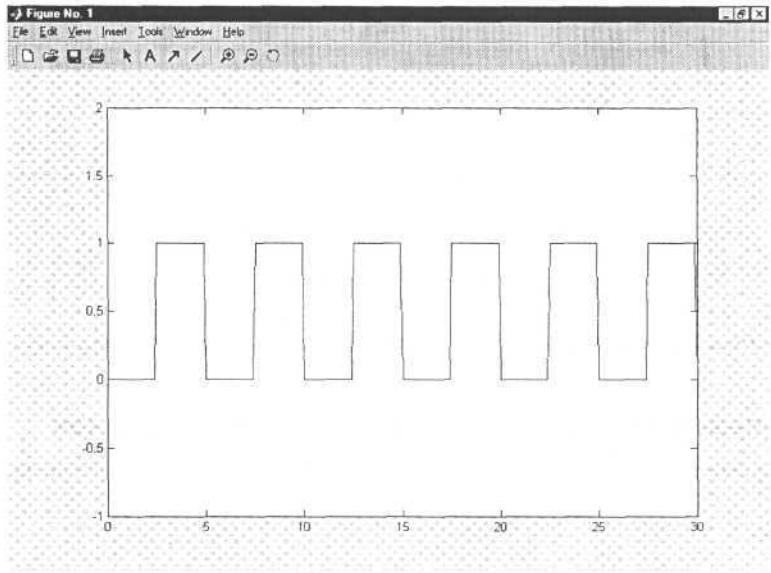

Рис. 7.23. Пример использования функции gensig

Затем для объекта, описываемого матрицей передаточных функций

$$W(p)=\begin{bmatrix}\dfrac{2p^2+5p+1}{p^2+2p+1}\\[2ex]\dfrac{p-1}{p^2+p+5}\end{bmatrix},$$

найдем реакцию на сформированный сигнал:

```
>> W=[tf([2 5 1],[1 2 3]);tf([1 -1],[1 1 5])];
>> lsim(W,u,t)
```

Результат показан на рис. 7.24.

Функция `stepfun` возвращает сигнал типа «единичный скачок»:

```
stepfun(t,t0)
```

Здесь `t` — вектор-строка моментов времени, и `t0` — некоторый момент времени. Возвращаемая величина — сигнал, соответствующий функции единичного скачка, происходящего в момент `t0`.

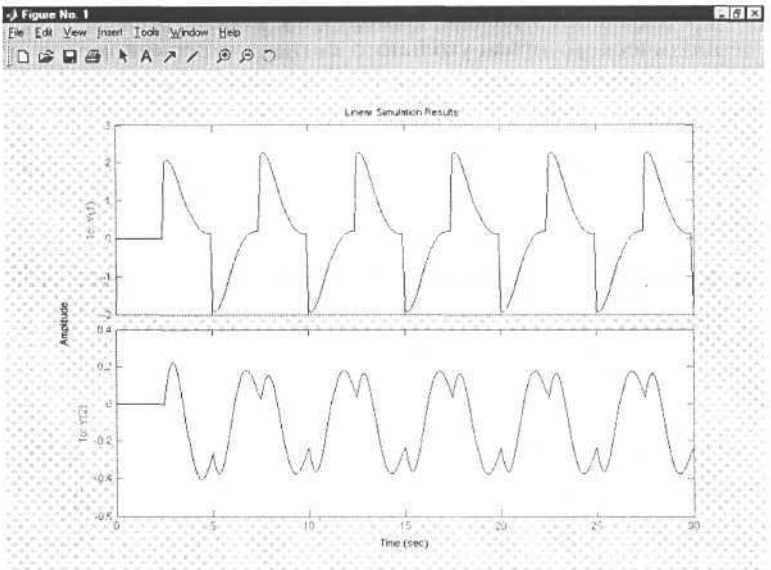

Рис. 7.24. Реакции двумерного объекта на периодический прямоугольный сигнал

7.3.8. Создание и представление временных задержек

Для создания и представления временных задержек имеются четыре простые функции.

Функция

```
sys = delay2z(sys)
```

представляет дискретную временную задержку для моделей sys типа TF, SS или ZPK с полюсами z = 0 или представляет задержки для FRD-моделей с фазовым сдвигом.

Следующий пример иллюстрирует применение функции `delay2d`:

```
>> z=tf('z',-1);
>> sys=(-.4*z -.1)/(z^2 + 1.05*z + .08)
Transfer function:
    -0.4 z - 0.1
--------------------
z^2 + 1.05 z + 0.08
Sampling time: unspecified
>> sys.InputDelay = 1;
>> sys = delay2z(sys)
Transfer function:
      -0.4 z - 0.1
-----------------------
z^3 + 1.05 z^2 + 0.08 z
Sampling time: unspecified
```

Функция

```
hasdelay(sys)
```

возвращает информацию о том, имеется ли в системе с моделью sys временная задержка.

Функция

```
td = totaldelay(sys)
```

возвращает матрицу суммарных задержек от входов к выходам модели.

```
>> sys = tf(1,[1 0]); sys.inputd = 2; sys.outputd = 1.5;
>> td = totaldelay(sys)
td = 3.5000
```

Функция pade

```
[num,den] = pade(T,N)      pade(T,N)
sysx = pade(sys,N)         sysx = pade(sys,NI,NO,Nio)
```

возвращает Паде-аппроксимацию для временной задержки. Пример применения этой функции был рассмотрен в разделе преобразований моделей, куда эта функция также входит.

7.3.9. Моделирование частотного отклика систем

Следующая группа функций служит для расчета и построения функции частотного отклика систем.

Функция

```
S = allmargin(sys)
```

вычисляет коэффициент передачи, фазу и задержку для SISO-модели.

Функция bode возвращает диаграммы Боде — графики логарифмической амплитудно-частотной характеристики (ЛАЧХ) и фазочастотной характеристики (ФЧХ):

```
bode(sys)                          bode(sys,w)
bode(sys1,sys2,...,sysN)           bode(sys1,sys2,...,sysN,w)
bode(sys1,'PlotStyle1',...,sysN,'PlotStyleN')
[mag,phase,w] = bode(sys)
```

Здесь аргумент w задает диапазон частот (w = [wmin,wmax]), остальные аргументы — такие же, как у ранее рассмотренных функций. Возвращаемые величины:

- диаграммы Боде для моделей с заданными именами;
- mag, phase, w — соответственно массивы значений амплитуд и фаз и вектор частот.

Приведем пример построения диаграммы Боде (рис. 7.25) для узкополосной системы:

```
>> g = tf([1 0.1 7.5],[1 0.12 9 0 0])
Transfer function:
s^2 + 0.1 s + 7.5
-----------------------
s^4 + 0.12 s^3 + 9 s^2
>> bode(g)
```

Функция bodemag строит диаграмму Боде только для магнитуды:

```
bodemag(sys) bodemag(sys,{wmin,wmax})
bodemag(sys,w) bodemag(sys1,sys2,...,sysN,w)
bodemag(sys1,'PlotStyle1',...,sysN,'PlotStyleN')
```

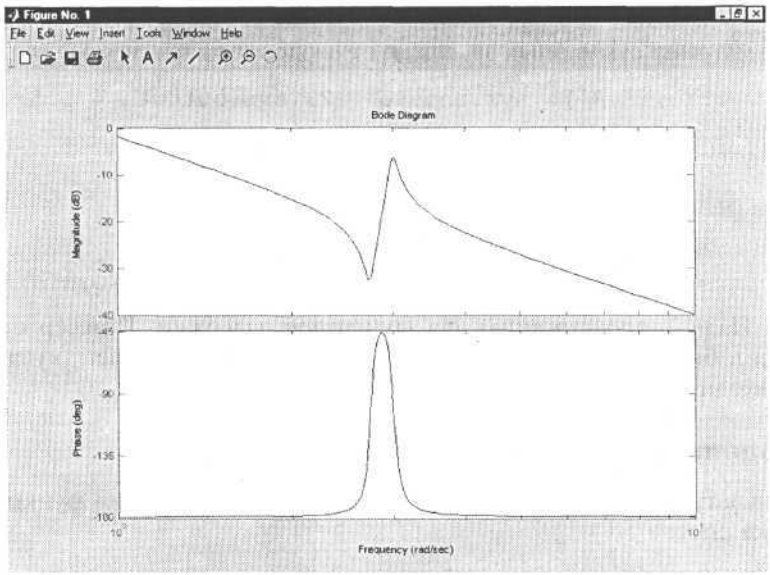

Рис. 7.25. Пример построения диаграммы Боде для узкополосной системы

Функция `evalfr` вычисляет значение комплексного коэффициента передачи для одного заданного комплексного значения частоты:

```
frsp = evalfr(sys,f)
```

Здесь `sys` — имя модели и `f` — заданное значение частоты. Возвращаемая величина `frsp` равна $W(f)$. Пример:

```
>> H = tf([1 -1],[1 1 1],-1); z = 1+j; evalfr(H,z)
ans = 0.2308 + 0.1538i
```

Функция `freqresp` вычисляет значения комплексного коэффициента передачи для заданного набора частот:

```
H = freqresp(sys,w)
```

Здесь `sys` — имя модели и `w` — вектор-строка значений частот. Возвращаемая величина `H` — массив значений $W(j\omega)$.

Функция

```
isys = interp(sys,freqs)
```

используется в качестве вспомогательной функции для интерполяции частотных зависимостей.

Функция `linspace` возвращает вектор-строку линейно возрастающих частот:

```
linspace(x1,x2) linspace(x1,x2,N)
```

Здесь `x1` и `x2` — верхняя и нижняя границы диапазона частот и `N` — количество задаваемых значений (по умолчанию 100). Возвращается вектор-строка, эле-

ментами которого являются частоты, эквидистантно расположенные на отрезке [x1, x2]. Пример, иллюстрирующий работу с этой функцией, дан ниже:

```
>> linspace(0,10,6)
ans =
    0    2    4    6    8    10
```

Функция logspace возвращает вектор-строку логарифмически возрастающих частот:

```
logspace(d1,d2) logspace(d1,d2,N)
```

Этой функцией создается вектор-строка, элементами которого являются частоты, логарифмически равномерно возрастающие на отрезке $[10^{d1}, 10^{d2}]$. Число элементов N по умолчанию равно 50. Пример:

```
>> logspace(0.1,1,6)
ans =
   1.2589    1.9055    2.8840    4.3652    6.6069    10.0000
```

Функция ltiview открывает окно просмотра LTI-моделей:

```
ltiview ltiview(sys1,sys2,...,sysn)
ltiview('plottype',sys1,sys2,...,sysn)
ltiview('plottype',sys,extras)
ltiview('clear',viewers)
ltiview('current',sys1,sys2,...,sysn,viewers)
```

Функция margin вычисляет запасы устойчивости по фазе и амплитуде:

```
margin(sys) [Gm,Pm,Wcg,Wcp] = margin(sys)
[Gm,Pm,Wcg,Wcp] = margin(mag,phase,w)
```

Функция margin(sys) строит логарифмические частотные характеристики разомкнутой системы с указанием запасов ее устойчивости.

Функция [Gm,Pm,Wcg,Wcp] = margin(sys) рассчитывает запас устойчивости по амплитуде (Gm), по фазе (Pm) и соответствующие частоты для одномерной разомкнутой системы sys. Если имеется несколько точек пересечения амплитудной характеристики на уровне 0 дБ и фазовой на уровне −180°, то возвращаются наименьшие значения запасов устойчивости.

Функция [Gm,Pm,Wcg,Wcp] = margin(mag,phase,w) возвращает значения запасов устойчивости, вычисленных по значениям массива точек (mag,phase,w), где mag — амплитуда, phase — фаза, w — частота. В этом случае для определения запасов устойчивости необходимо выполнить интерполяцию для определения соответствующих точек пересечения.

Пример:

```
>> hd = tf([0.04798 0.0464],[1 -1.81 0.9048],0.1)
Transfer function:
  0.04798 z + 0.0464
--------------------
z^2 - 1.81 z + 0.9048
Sampling time: 0.1
>> [Gm,Pm,Wcg,Wcp] = margin(hd);
[Gm,Pm,Wcg,Wcp]
ans = 2.0517  13.5711  5.4374  4.3544
```

Теперь можно построить нужные характеристики (рис. 7.26):

```
>> margin(hd)
```

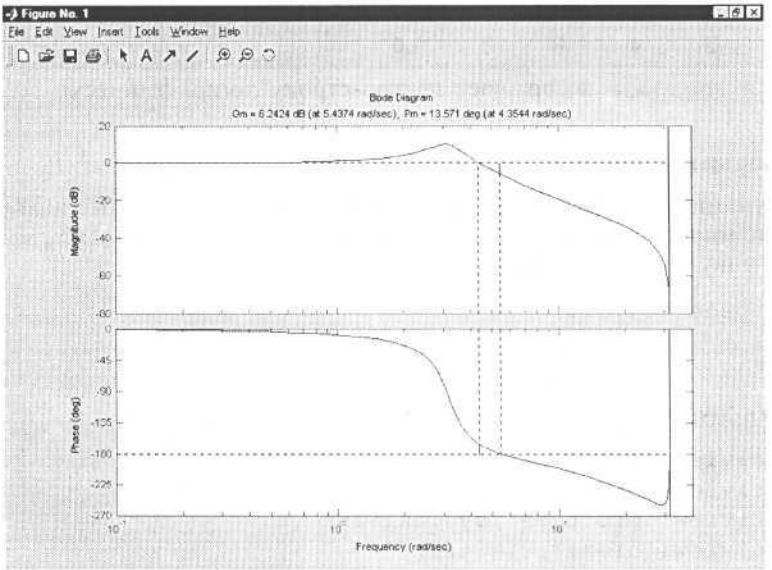

Рис. 7.26. Логарифмические частотные характеристики

Функция `nyquist` возвращает диаграмму Найквиста (годограф):

```
nyquist(sys)                        nyquist(sys,w)
nyquist(sys1,sys2,...,sysN)    nyquist(sys1,sys2,...,sysN,w)
nyquist(sys1,'PlotStyle1',...,sysN,'PlotStyleN')
[re,im,w] = nyquist(sys)  [re,im] = nyquist(sys,w)
```

Аргументы этой функции уже описывались. Возвращаемые величины:

- диаграмма Найквиста (в многомерном случае — для каждого канала «вход/выход»);
- `re` и `im` — соответственно векторы значений вещественной и мнимой частей комплексного коэффициента передачи для заданного набора частот.

Пример построения диаграммы Найквиста (рис. 7.27) представлен ниже:

```
>> H = tf([2 5 1],[1 2 3])
Transfer function:
2 s^2 + 5 s + 1
---------------
s^2 + 2 s + 3
>> nyquist(H)
```

Для построения диаграммы Найквиста с «сеткой» (рис. 7.28) достаточно исполнить команду вывода «сетки»:

```
>> grid
```

Функция `nichols` возвращает годограф Николса:

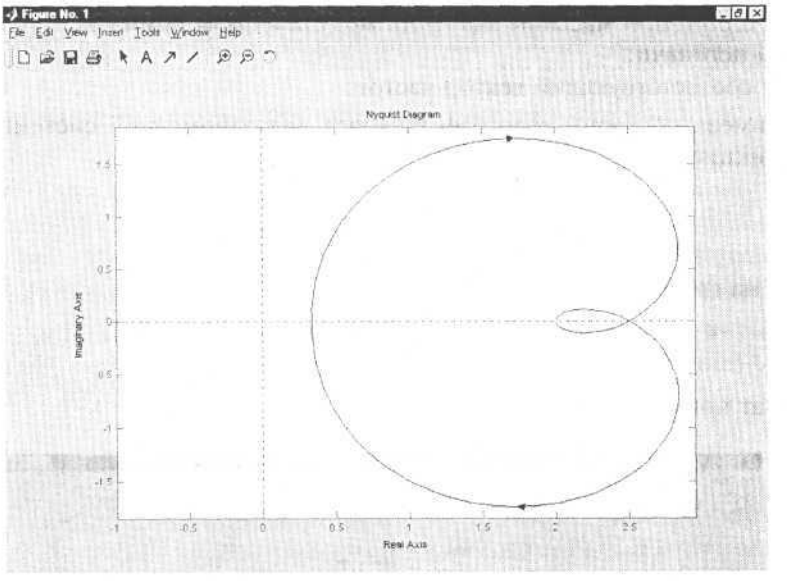

Рис. 7.27. Диаграмма Найквиста

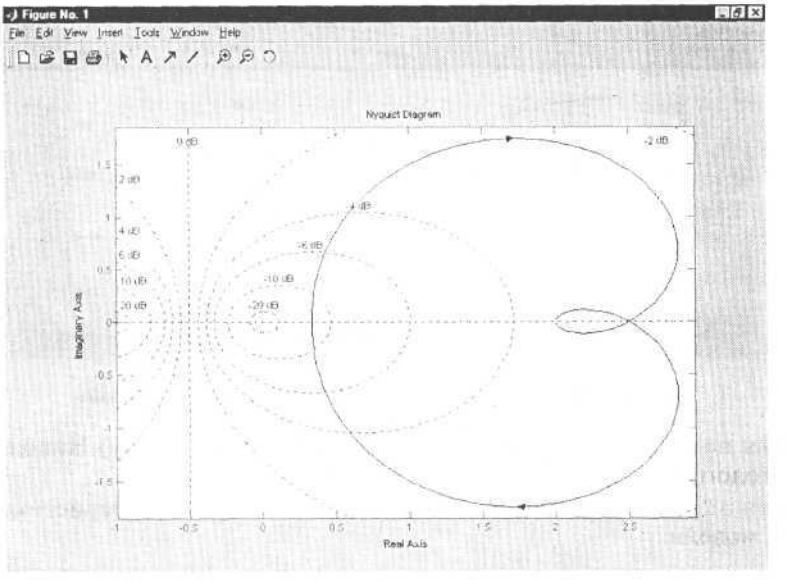

Рис. 7.28. Диаграмма Найквиста с сеткой

```
nichols(sys)                          nichols(sys,w)
nichols(sys1,sys2,...,sysN)           nichols(sys1,sys2,...,sysN,w)
nichols(sys1,'PlotStyle1',...,sysN,'PlotStyleN')
[mag,phase,w] = nichols(sys)  [mag,phase] = nichols(sys,w)
```

Аргументы — те же, что у предыдущей функции. Возвращаемые величины:

- годографы Николса;

- `mag` и `phase` — массивы значений модуля и фазы комплексного коэффициента передачи;
- `w` — соответствующий вектор частот.

Для примера построим годограф Николса для одномерной системы с передаточной функцией

$$W(p) = \frac{-4p^4 + 48p^3 - 18p^2 + 250p + 600}{p^4 + 30p^2 + 282p^2 + 525p + 60}$$

и нанесем на график «сетку»:

```
>> num=[-4 48 -18 250 600]; den=[1 30 282 525 60];
>> W=tf(num,den); nichols(W);ngrid
```

Результат показан на рис. 7.29.

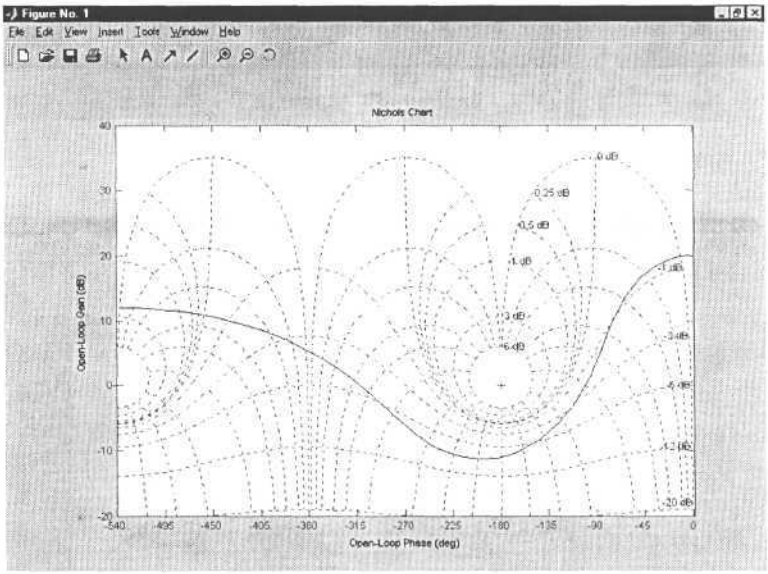

Рис. 7.29. Пример годографа Николса с «сеткой»

Функция `ngrid` (без аргументов) наносит диаграмму (сетку) Николса на одноименный годограф.

Функция `sigma` осуществляет построение частотной характеристики сингулярных чисел модели:

```
sigma(sys)                    sigma(sys,w)
sigma(sys,w,type)             sigma(sys1,sys2,...,sysN)
sigma(sys1,sys2,...,sysN,w)
sigma(sys1,sys2,...,sysN,w,type)
sigma(sys1,'PlotStyle1',...,sysN,'PlotStyleN')
[sv,w] = sigma(sys)           sv = sigma(sys,w)
```

Для непрерывных моделей рассматриваемая функция вычисляет частотную характеристику сингулярных чисел (определение сингулярных чисел см. выше, при описании функции `norm`) матрицы $W(j\omega)$. Для дискретных моделей с дискретной

матричной передаточной функцией $W(z)$ вычисляется частотная характеристика матрицы $W(e^{j\omega Ts})$ для диапазона частот от 0 до π/Ts. Частотные характеристики сингулярных чисел являются обобщением понятия логарифмических частотных характеристик и оказываются весьма полезными при анализе робастности.

Аргументы этой функции следующие:

- `sys, sys1, sys2, ..., sysN` — имена моделей;
- `w` — диапазон частот, `w = {wmin,wmax}`;
- `'PlotStyle1', ..., 'PlotStyleN'` — аргументы, определяющие стили графиков (типы линий);
- аргумент `type` задает модификацию частотной характеристики сингулярных чисел: 1 — для $W^{-1}(j\omega)$; 2 — для $\mathbf{I} + W(j\omega)$; 3 — для $\mathbf{I} + W^{-1}(j\omega)$.

Возвращаемые величины:

- графики частотных характеристик сингулярных чисел для моделей с указанными именами;
- `sv` — массив значений сингулярных чисел;
- `w` — соответствующий вектор частот.

Пример применения функции sigma:

```
>> H = [0 tf([3 0],[1 1 10]) ; tf([1 1],[1 5]) tf(2,[1 6])]
Transfer function from input 1 to output...
#1: 0
    s + 1
#2: -----
    s + 5
Transfer function from input 2 to output...
          3 s
#1: ------------
    s^2 + s + 10
      2
#2: -----
    s + 6
>> subplot(211); sigma(H); subplot(212); sigma(H,[],2);
```

Построенные по этому примеру графики представлены на рис. 7.30.

7.3.10. Композиция систем

Функции композиции систем позволяют формировать системы различной структуры, используя в качестве элементов такой структуры LTI-модели. Это нечасто применяемые функции, поэтому мы ограничимся упоминанием их имен и назначения:

- `append` — объединение LTI-систем путем объединения входов и выходов;
- `augstate` — дополняет выходы переменными состояния модели.
- `feedback` — образование замкнутой структуры из двух систем;
- `connect` — построение LTI-модели на основе структурной схемы;
- `conv` — умножение двух полиномов;
- `lft` — образование системы с перекрестными связями (соединение систем так называемой звездой Редхеффера);