Дьяконов В. П., Абраменкова И. В.

MATLAB 5.0/5.3

Система символьной математики

Москва

Издательство «Нолидж»

1999 г.

УДК 004.438 MATLAB + 004.738.5
Д 93
ББК 32.973.26-018.1

Дьяконов В. П., Абраменкова И. В.
MATLAB 5.0/5.3. Система символьной математики.
М.: Нолидж .-1999 г., 640 с., ил

Книга описывает новые версии мощной, многофункциональной интегрированной системы автоматизации математических и научно-технических расчетов MATLAB 5.0/5.3 и ряд пакетов ее расширений, включая Notebook, Symbolic и Simulinc. Современную технологию визуализации математических вычислений этой системы дополняет обширный арсенал реализаций численных методов, накопленный за последние три десятилетия. Система обладает свойствами расширяемости и адаптации к задачам пользователя, характеризуется высокой скоростью выполнения вычислений и возможностью вывода данных в числовой, табличной и графической форме. Базируясь на работе с векторами и матрицами, она имеет эффективные средства для решения задач по обработке данных, анализу и фильтрации сигналов, символьным вычислениям и моделированию блочно заданных систем и устройств. Книга рекомендуется широкому кругу читателей — студентам университетов и вузов, инженерам, научным работникам, — всем, кто интересуется математикой, физикой и другими областями науки и техники.

ISBN 5-89251-069-7

Лицензия ЛР № 064480 от 04 марта 1996 г.

Об авторах

Академик международной академии педагогических наук, Соросовский профессор по математике, доктор технических наук Дьяконов Владимир Павлович не нуждается в особом представлении. Он является автором многих известных книг по электронике и информатике. Выпущена целая серия его книг по компьютерным математическим системам, языкам программирования и современным бытовым и офисным техническим устройствам. Владимир Павлович заведует кафедрой физической и информационной электроники Смоленского Государственного педагогического университета. Вам предлагается 11-я книга данного автора из серии, посвященной описанию современных компьютерных математических систем. Ранее вы могли читать его книги о математических системах Eureka, PC MatLAB, Mathcad под MS-DOS, Mathcad 6.0 и 7.0, Derive под MS-DOS и Windows, Mathematica 2 и 3, Maple V R3, R4 и R5.

Абраменкова Ирина Владимировна — молодой ученый и педагог. Это ее вторая книга по математическим системам. Первая книга, написанная также совместно с профессором В. П. Дьяконовым, "Mathcad 7 в математике, в физике и в Internet" недавно была выпущена издательством "Нолидж" и пользуется заслуженной популярностью.

Введение

Сферы применения персональных компьютеров поистине неисчерпаемы и охватывают подготовку и обработку текстов, финансово-экономические расчеты, ведение баз данных, подготовку проектных документов, работу в глобальной сети Internet и т.д. Среди этого обилия применений видное место занимают математические и научно-технические расчеты – та область, ради которой компьютеры (от слова computer – вычислитель) были изначально созданы, вначале в виде ЭВМ, затем программируемых микрокалькуляторов [1] и, наконец, в виде современных массовых персональных компьютеров (ПК) [2,3] .

Парадоксально, но широкое внедрение компьютеров до недавнего времени не сделало математические вычисления более легкими для заинтересованных пользователей. Конечно, всех поражает высокая скорость вычислений и богатство наработанного программного обеспечения для их реализации. К услугам пользователей имеются готовые (но не всегда доступные, как в прямом, так и в переносном смысле) библиотеки научных подпрограмм, справочники (например, [4]), специально приобретенные или случайно попавшиеся программы.

Однако, чтобы разработать программу для решения поставленных задач, пользователь должен был освоить работу на компьютере, изучить основы программирования, знать один, а то несколько языков программирования, начиная со знаменитого Бейсика [4-6], вечного Фортрана и экзотических расширяемых языков Лого [7] и Форт

[8], разобраться с достаточно сложными и подчас капризными численными методами. Вот почему нередко из-под рук способного физика, математика или инженера еще совсем недавно выходили далекие от совершенства программы. Приспособить для своих целей чужую программу подчас оказывалось не проще, чем разработать свою.

Сегодня положение с использованием ПК для математических расчетов начинает в корне меняться. Это связано с появлением мощных, достаточно универсальных и простых в применении интегрированных систем (пакетов программ) для автоматизации математических и научных расчетов [9-11], таких как Eureka [12], Mercury [13], Mathcad [14-20], Derive [21, 22], Mathematica 2 и 3 [23], Maple V [24-29] и др. Каждая из этих систем имеет свои достоинства и недостатки и заслуживает отдельного рассмотрения. Повышенный интерес пользователей к подобным системам подтверждают результаты выпуска в последние годы целого ряда книг, посвященных указанной теме [12-34].

В данной книге рассматривается система MATLAB, прошедшая многолетний путь развития от программного матричного модуля, используемого на больших ЭВМ, до универсальной интегрированной системы, ориентированной на массовые ПК класса IBM-PC и Macintosh. MATLAB представляет собой хорошо апробированную и надежную систему, рассчитанную на решение широкого круга математических задач с представлением данных в универсальной (но не навязчивой) матричной форме, предложенной фирмой MathWorks, Inc.

Система MATLAB с точки зрения ее разработчиков предлагается как **язык программирования высокого уровня для технических вычислений**. Она вобрала в себя не только передовой опыт развития и компьютерной реализации численных методов за последние три десятилетия, но и весь опыт развития математики за всю историю Человечества. Систему используют более 500 000 легально зарегистрированных пользователей, ей доверяют свои проекты ведущие университеты и научные центры мира. Книга призвана помочь тем, кто хочет детально изучить возможности новых версий системы MATLAB 5.0/5.3 и использовать ее в своей работе.

За основу описания взята юбилейная 10-ая реализация системы, известная также как версия 5.2.1. Она стала предметом особой гордости разработчика — фирмы MathWorks, Inc. и уже получила широкую известность в России. Последняя 11-я реализация системы (MATLAB 5.3) появилась в марте 1999 года и по своим общим возможностям мало отличается от 10-й реализации. Материал данной книги будет одинаково полезен пользователям всех реализаций 5.* - от 5.0 до 5.3. Книга не повторяет фирменные описания систем, однако тщательно сверена с ними.

MATLAB – довольно дорогая программа (ее цена упорно держится возле цифры 3000 долларов) и считается одной из самых популярных и элитных математических систем. По данным поисковой системы Amazon в Internet, с информационной базой, включающей более трех миллионов научно-технических книг, именно системе MATLAB принадлежит первенство по количеству опубликованных изданий.

К сожалению, в России неоправданно мало публикаций по системе MATLAB. Помимо обзоров [9-11,30] и первой книги по этой системе [31], в течение ряда лет серьезных изданий, посвященных MATLAB, практически не было. Наконец в 1997-1999 г.г. появились книги [32,33], содержащие перевод части фирменных справочников по системе MATLAB 4.0/5.2. При этом книга [32] описывает лишь отдельные средства упрощенной, студенческой версии системы MATLAB 5.0. Ее целесообразно использовать совместно с двухтомником [33]. К сожалению, в [32,33] не описан пакет расширения Simulink, органично вошедший в новые реализации системы MATLAB и составляющий как бы вторую сторону "медали" системы MATLAB.

Между тем, за рубежом системе MATLAB посвящены сотни книг (их список можно найти на Internet-странице фирмы MathWorks, Inc. Таким образом, интерес к системе MATLAB, особенно к ее новейшим реализациям 5.0/5.2/5.2.1/5.3, остается у нас не удовлетворенным. Настоящая книга, по замыслу авторов, должна ликвидировать этот пробел.

Данная книга является одновременно научной монографией по математической системе MATLAB и руководством для пользователя. На роль всеобъемлющего справочника по этой системе она не претендует, хотя и содержит описание не только основ системы MATLAB, но и важнейших пакетов прикладных программ, расширяющих возможности системы — в том числе уникального по своим возможностям пакета Simulink. Полное справочное описание системы и всех ее пакетов расширений состоит из многих тысяч страниц (объем такого описания только в формате PDF достигает 200 Мбайт!) и его просто невозможно представить в виде одной книги.

Авторы, отдавая дань своим традициям в подготовке своих книг и учитывая специфику новых задач, в качестве основных принципов построения книги хотели бы отметить следующие:

- изложение материала дается от простого к сложному, в общем по принципу "чем дальше в лес, тем больше дров";
- начальный навык работы с описываемой системой можно получить уже из первой главы книги;
- книга содержит достаточно полный справочный материал как по самой системе MATLAB 5.*, так и по ряду важных пакетов ее расширения Notebook, Symbolic и Simulink;
- в книге использовано свыше 250 иллюстраций и многие сотни примеров вычислений;
- исключены повторы материала — увы, от вполне справедливой поговорки "повторение — мать учения" в данном случае пришлось отойти;
- полностью описан процесс инсталляции системы;
- значительное внимание уделено пользовательскому интерфейсу системы и правилам работы с ней, а также визуализации результатов вычислений;
- в книге тщательно отобраны и описаны те средства, которые представляют интерес для большинства читателей — как начинающих пользователей системы MATLAB,

так и профессионалов в области математических матричных методов, реализации численных методов, компьютерного моделирования и графики;

- особое внимание уделяется практическим примерам применения системы, многие из которых дополняют примеры, имеющиеся в справочных данных по системе;
- одна из целей книги — приучить читателя пользоваться обширной справочной базой данных системы MATLAB и примерами ее применения;
- описание носит авторский характер и не является прямым и формальным переводом справочной базы данных или документации по системе;
- рубрикация книги сделана подробно с указанием имен наиболее важных команд и функций, так что необходимость в индексном указателе отсутствует, при этом рубрикация заметно отличается от фирменных описаний;
- ввиду обширности излагаемого материала приняты особые меры по компактности его изложения, особенно при описании определенных групп операторов и функций;
- в качестве основы для описания взята хорошо известная версия MATLAB 5.2.1 (реализация 10), ориентированная на массовые ПК класса IBM-PC с операционной системой Windows 95/98;
- все материалы книги можно использовать при работе с версиями MATLAB от 5.0 до 5.3, в том числе созданными для иных компьютерных платформ;
- впервые в отечественной литературе описаны возможности дескрипторной графики системы, техника обработки реальных изображений и уникальные возможности пакета Simulink для моделирования блочно заданных динамических систем и устройств самого различного назначения.

Предупреждения

Книги по компьютерной тематике пишутся довольно быстро. Иначе они неизбежно устаревают уже к моменту своего выхода в свет. Разумеется, в таких условиях трудно гарантировать, что в книге будут напрочь отсутствовать недочеты. Авторы считают нужным предупредить читателей об этом, хотя сами они и издательство сделали все возможное, чтобы свести этот ущерб в ее подготовке к минимуму.

Авторы приносят извинения читателям, все же не нашедшим в книге (несмотря на большой объем) подробное описание именно тех сведений, которые им кажутся самыми важными и интересными. Мы уже отмечали, что объем документации по системам MATLAB 5.0/5.3 настолько велик, что физически невозможно дать полное описание этих систем в одной книге. Мы надеемся, что смогли собрать и выделить именно те сведения, которые представляют интерес для массового читателя. При этом подразумеваются не только начинающие, но и опытные специалисты.

Работа с такой мощной математической системой, как MATLAB, несомненно требует от читателя знания основ математики. Без этого невозможно гарантировать правильное применение используемых в системе методов и корректность получаемых результатов. В связи с этим следует отметить, что данная книга не является справоч-

ником по математике и численным методам вычислений. Она лишь описывает конкретную математическую систему. Поэтому многие математические методы в ней описаны кратко, а некоторые (в основном узкоспециализированные) лишь упоминаются по названию. Недостающие сведения можно найти в литературе, например [34-49].

Мы вынуждены предупредить читателя, что авторы и издательство не несут никакой ответственности за ваши неудачи в освоении системы и за моральный или даже экономический ущерб, который может иметь место вследствие ошибок читателя и неудачного выбора математической системы.

Сказанное ни в коей мере не означает, что в данной книге или в системе MATLAB имеются серьезные ошибки и недочеты. Просто такое предупреждение отвечает нормам современного юридического права в отношении к сложным программным продуктам и сопровождающей их документации.

Благодарности и адреса для связи

Эта книга написана в инициативном порядке на кафедре физической и информационной электроники Смоленского Государственного педагогического университета, а также в рамках работ "Internet Академии" центра информационных технологий "Телепорт" и ЗАО "Смоленский телепорт" (**www.keytown.com**). В ней частично отражены материалы работ авторов по гранту Министерства общего и профессионального образования РФ, выделенному СГПУ в 1997 году в области применения современных компьютерных математических систем для решения фундаментальных задач естествознания (руководитель работ — проф. Дьяконов В.П.).

Авторы благодарят Генерального директора ЗАО "Смоленский Телепорт" Григория Рухамина за предоставление услуг спутникового Internet 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. Мы также благодарны Генеральному директору корпорации SoftLine (Россия) Игорю Боровикову за неоднократные предложения написать данную книгу и за роль этой корпорации в распространении математических программ в России. Мы рады, что эти планы, наконец-то, нами реализованы. Авторы признательны кандидату физико-математических наук, доценту Кристаллинскому Р. Е. за критический просмотр рукописи и сделанные при этом (и учтенные) замечания. Поддержка Международной Соросовской Программы Образования в Области Точных Наук (ISSEP) скрасила завершение трудной работы над данной книгой.

С авторами можно связаться по электронной почте (**dyak@keytown.com** для Дьяконова В. П. и **abr@keytown.com** для Абраменковой И. В.). Вы можете также посетить домашнюю страницу одного из авторов данной книги (**www.keytown.com/users/dyak**, рис. 1). Авторы заранее выражают признательность всем читателям, кото-

рые готовы сообщить свое мнение о данной книге. Кроме электронной почты, замечания можно направлять по адресу: 214000, Смоленск, ул. Пржевальского 4, СГПУ. Вы можете отправить свои письма и по адресу издательства, выпустившего книгу.

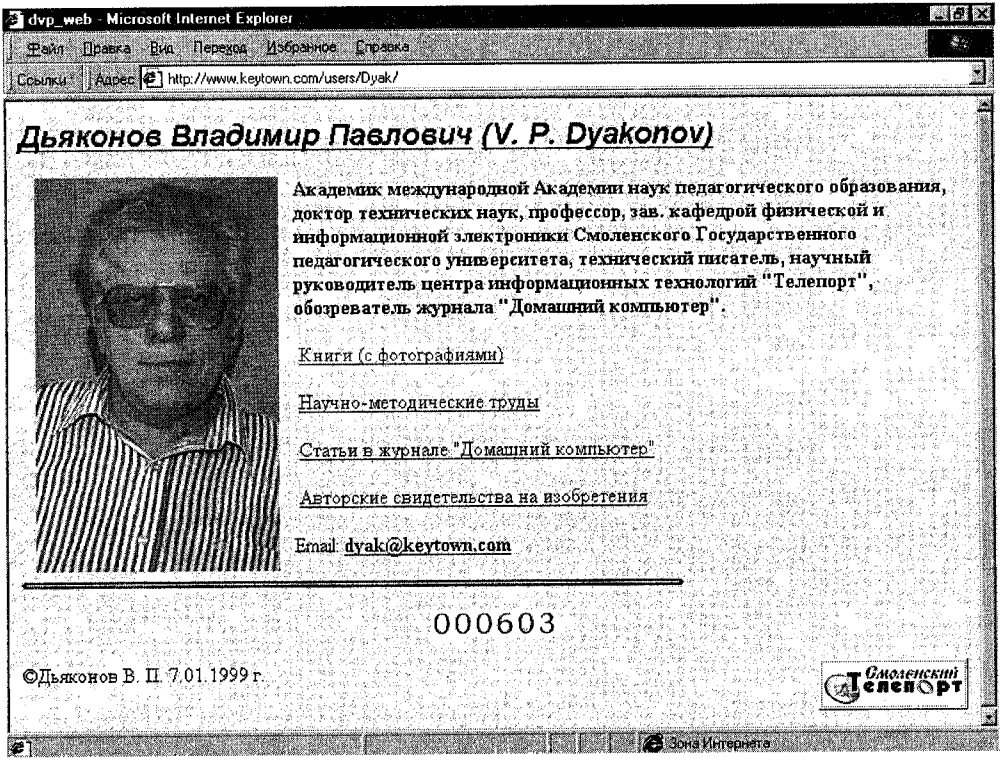


Рис. 1. Домашняя страница одного из авторов книги — проф. Дьяконова В. П.

Связаться с фирмой MathWorks вы можете, посетив сайт **www.mathworks.com**. С официальным дилером фирмы в России корпорацией SoftLine можно связаться через **www.softline.ru**.

Глава 1.
Начальное знакомство с системой MATLAB

1.1. Основные сведения о системе MATLAB

1.1.1. История появления системы MATLAB

MATLAB – одна из старейших, тщательно проработанных и апробированных систем автоматизации математических расчетов, построенная на расширенном представлении матричных операций. Этот факт отражает название системы — (**MAT**rix **LAB**oratory — **матричная лаборатория**). Однако синтаксис языка программирования системы продуман настолько тщательно, что эта ориентация совсем не ощущается пользователями, которых не интересуют конкретно матричные вычисления. И поэтому те, кто ошибочно именуют систему как "**мат**ематическая **лаб**оратория", в сущности правы.

В настоящее время MATLAB далеко вышла за пределы специализированной матричной системы и стала одной из наиболее мощных универсальных математических систем. В новую версию вошли такие мощные типы данных, как многомерные массивы, массивы ячеек и разреженные матрицы, что открывает возможности применения системы при создании и отладке новых алгоритмов матричных и основанных на них параллельных вычислений.

В целом MATLAB — это уникальная коллекция реализаций современных численных методов, созданных за последние три десятка лет. Она вобрала в себя также опыт, правила и методы математических вычислений, накопленные за тысячи лет развития математики. Систему с прилагаемой к ней обширной документацией вполне можно рассматривать как фундаментальный многотомный **электронный справочник** по математическому обеспечению ЭВМ — от массовых PC до супер-ЭВМ. Увы, пока преподанный лишь на английском и японском языках.

Система MATLAB была разработана Молером (C. B. Moler) и с конца 70-х годов широко использовалась на больших ЭВМ [30, 31]. В начале 80-х годов Дж. Литл (John Little) из фирмы MathWork Inc. разработал версию системы PC MATLAB для ПК класса IBM PC, VAX и Macintosh. В дальнейшем были созданы версии системы для рабочих станций Sun, компьютеров с операционной системой UNIX и многих других типов больших и малых ЭВМ.

Одной из основных задач системы было предоставление пользователям мощного языка программирования, ориентированного на 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.

Возможности MATLAB весьма обширны, а по скорости выполнения задач эта система превосходит другие подобные системы. Она применима практически к любой области науки и техники и широко используется при математическом моделировании физических устройств и систем, относящихся к механике, в частности, к динамике, гидро- и аэродинамике, акустике и так далее. Этому способствует не только расширенный набор матричных и иных операций и функций, но и наличие расширения Simulink, предназначенного для решения задач блочного моделирования динамических систем и устройств.

В обширном и постоянно пополняемом комплексе команд и функций Toolbox система MATLAB имеет специальные средства для электро- и радиотехнических расчетов (операции с комплексными числами, матрицами, векторами и полиномами, обработка данных, анализ сигналов и цифровая фильтрация), обработки изображений, реализации нейронных сетей и по другим новым направлениям науки и техники.

Важным достоинством системы является ее **открытость** и **расширяемость**. Большинство команд и функций системы реализованы в виде текстовых М-файлов и файлов на языке С, причем все файлы доступны для модификации. Пользователю предоставлена возможность создавать не только отдельные файлы, но и библиотеки файлов для реализации специфических задач.

Поразительная легкость **модификации** системы и возможность ее **адаптации** к решению специфических задач науки и техники привели к созданию десятков пакетов прикладных программ, намного расширивших сферы применения системы. Некоторые из них, например, Notebook (интеграция с текстовым процессором Word и подготовка "живых" электронных книг), Symbolic (символьные вычисления с применением ядра системы Maple V R4) и Simulink (симуляция заданных блочно динамических систем и устройств), органично интегрировались с системой MATLAB и стали ее составной частью. Аннотационное описание этих пакетов дано в Приложении.

1.1.2. Место MATLAB среди современных математических систем

Современная **компьютерная математика** предлагает целый набор интегрированных программных систем и пакетов программ для автоматизации математических расчетов: Eureka, Asystant, Gauss, TK Solver, Derive, Mathcad, Mathematica, Maple V и др. Возникает вопрос – какое место занимает среди них система MATLAB?

Прежде всего надо отметить, что эти системы основаны на принципиально разных подходах и базовых алгоритмах, хотя порой и решают казалось бы довольно схожие задачи – расчеты по формулам, решение систем линейных и нелинейных уравнений, минимизация функций и т. д. Система Eureka, например, построена на основе

алгоритма минимизации среднеквадратической погрешности решений систем нелинейных уравнений, пакет Derive ориентирован на символьные операции и использует язык программирования задач искусственного интеллекта MuLISP, Maple V базируется на мощном ядре символьных операций, системы Mathcad содержат уникальный формульный интерпретатор и так далее.

Среди систем автоматизации математических расчетов особое место занимает система Mathcad [14-20]. Отличительная черта этой системы – входной язык, максимально приближенный к обычному математическому языку или языку научных статей и книг. Это можно отнести к достоинствам системы. Однако на практике обнаруживаются серьезные недостатки. Так, многие специализированные знаки (квадратного корня, суммы и произведения ряда, интеграла и т.д.) на обычной клавиатуре ПК отсутствуют, и для их ввода приходится запоминать соответствующие символы или использовать многочисленные палитры спецзнаков, загромождающие окно для создания документов программы. Применение панелей с математическими спецзнаками можно считать выходом из положения, если смириться с необходимостью манипулирования множеством панелей.

Первые версии MATLAB для ПК были созданы на базе системы, ориентированной на большие ЭВМ. В них используется командный режим работы, реализованный с помощью строчного редактора. Несмотря на отсутствие современного сервиса, меню управления и встроенная база данных помощи тщательно продуманы и на них лежит отпечаток известной операционной системы UNIX (кстати, система PC MATLAB, также как и UNIX, написана на языке C).

С появлением новых версий системы MATLAB 5.0/5.3 под Windows тенденция главенства командной строки была сохранена. И сейчас после запуска системы пользователь оказывается в привычном командном режиме управления, однако в его распоряжении теперь имеются и средства интерфейса Windows. Для начала это главное меню и довольно скромная панель инструментов. Однако по мере освоения система MATLAB демонстрирует все достоинства приложений Windows, основанные на использовании графических средств интерфейса GUI.

В целом можно сказать, что система MATLAB рассчитана на самое серьезное применение в практической работе и открывает перед пользователем огромные возможности по реализации сложных математических расчетов. С равным успехом она применима к задачам математики, физики, биологии, электротехники, радиотехники и т.д. Все это не исключает достоинств прозаического применения MATLAB в повседневных расчетах.

Систему сопровождает большое количество M-файлов (с расширением .m), содержащих демонстрационные примеры и определения новых операторов и функций. Эта библиотека, все файлы которой содержат подробные комментарии, является прекрасным учебником по программированию на языке системы.

Система MATLAB превратилась в одну из самых распространенных и популярных систем автоматизации математических расчетов. В данной книге описана одна из последних ее версий — MATLAB 5.2.1, имеющая множество средств программирования вычислительных задач, в которые входят и средства предшествующих версий. Однако материал книги отобран таким образом, что он в одинаковой мере относится ко всем версиям 5.* - от 5.0 до 5.3.

Система MATLAB оказала большое влияние на разработку ряда пакетов для выполнения матричных операций, например, расчета систем управления (таких как CTRL C, MATRIX и т.д.), в свою очередь вобрав в себя лучшие из их средств, разработанных за более чем 30-летнюю историю развития матричных методов вычислений на ЭВМ. Первая версия системы MATLAB была написана на языке Fortran, после чего языком реализации системы стал язык СИ.

Стоимость новых версий системы MATLAB колеблется возле 3000 $. Это одна из самых дорогих коммерческих математических систем. Высокая стоимость системы соответствует ее возможностям и серьезности применений. Вне всякого сомнения MATLAB — это элитная система — своего рода "Ролл Ройс" в семействе современных математических систем. В области реализации численных методов и методов математического моделирования MATLAB уверенно лидирует среди известных систем компьютерной математики.

1.1.3. Возможности системы MATLAB

Предшествующие версии системы MATLAB 4.* обладали следующими средствами:

В области математических вычислений

- матричные, векторные, логические операторы
- элементарные и специальные функции
- полиномиальная арифметика
- многомерные массивы
- пользовательские структуры
- массивы структур

В области реализации численных методов

- дифференциальные уравнения
- вычисление одномерных и двумерных квадратур
- поиск корней нелинейного алгебраического уравнения
- оптимизация функций ряда переменных
- одномерная и многомерная интерполяция

В области программирования

- свыше 500 встроенных математических функций
- ввод/вывод двоичных и текстовых файлов
- применение программ, написанных на C и Fortran
- автоматическая перекодировка процедур MATLAB на языках C и C++
- типовые управляющие структуры

В области визуализации и графики

- возможность создания двумерных и трехмерных графиков
- проведения визуального анализа данных

Эти средства сочетались с открытой архитектурой систем, позволяющей изменять уже существующие функции и добавлять свои собственные. Входящая в MATLAB программа Simulink дает возможность моделировать системы и устройства, заданные своими функциональными блоками-моделями. Система имеет обширную библиотеку компонентов (блоков) и простые средства задания их параметров.

В последующих версиях системы MATLAB 5.* введены новые средства, которые мы отмечаем отдельно:

Улучшенная среда программирования

- профилировщик M-файлов для оценки времени исполнения фрагментов программ
- редактор/отладчик M-файлов с удобным графическим интерфейсом
- обьектно-ориентированное программирование, включая переназначение функций и операторов
- средства просмотра содержимого рабочей области и путей доступа
- конвертирование M-файлов функций в промежуточный P-код
- просмотр справочной документации в формате HTML с помощью Internet-браузеров

Графический интерфейс пользователя (GUI)

- интерактивное средство построения GUI-Guide
- редактор свойств графического объекта — Handle Graphics Property Editor
- панели списков, включая множественный выбор
- форма диалоговых панелей и панелей сообщений
- многострочный режим редактирования текста
- запоминание последовательности графических элементов управления
- опция "недоступный снаружи" элемент управления
- свойство переносимости между платформами
- курсор, определяемый пользователем
- подготовка документов в формате HTML (версия 5.3)

Новые типы данных

- многомерные массивы
- определяемые пользователем структуры данных и массивы структур
- массивы ячеек данных разного типа
- массивы строк с упаковкой 2 бита на символ
- массивы с упаковкой 1 байт на элемент

Средства программирования

- списки аргументов переменной длины
- переназначение функций и операторов
- применение в M-файлах функций подфункций
- оператор-переключатель switch...case...end
- короткая форма условного оператора if
- оператор wait for
- функции обработки битов

Математические вычисления и анализ данных

- 5 новых решателей ОДУ
- ускоренное вычисление функций Бесселя с повышенной скоростью
- вычисление собственных значений и сингулярных чисел для матриц разреженной структуры
- двумерные квадратурные формулы
- многомерная интерполяция
- триангуляция и вывод на терминал данных, определенных на неравномерной сетке
- анализ и обработка многомерных массивов
- поддержка времени и даты

Новые возможности обычной графики

- Z-буферизация для быстрой и точной трехмерной визуализации
- 24-битовая RGB-поддержка
- множественная подсветка поверхностей и полигонов
- перспективные изображения из произвольной точки
- новые модели подсветки
- векторизованные полигоны для больших трехмерных моделей
- поддержка данных, определенных на неравномерной сетке, включая триангуляционные и сеточные двух- и трехмерные поверхности
- дескрипторная графика для множественных объектов
- вывод на терминал, хранение и импорт 8-битовых образов
- дополнительные форматы графических объектов

Презентационная графика и звук

- двойные x и y оси
- легенда (показ разметки разными стилями линий) графика
- управление шрифтом текстовых объектов
- надстрочные, подстрочные и греческие символы
- трехмерные диаграммы, поля направлений, ленточные и стержневые графики
- увеличенное количество стилей для маркировки линий
- 16-битовый стереозвук

Интерактивная документация

- возможность просмотра на основе Internet-браузеров Netscape Navigator или Microsoft Internet Explorer
- полная справочная документация в форматах HTML и PDF
- возможность создания "живых" книг с помощью специального приложения Notebook

Новейшая версия MATLAB 5.3 (реализация 11) интегрирует в своем составе 42 программных продукта, среди которых основу составляют базовая система MATLAB и новая реализация пакета расширения Simulink 3.0. Из указанных продуктов 4 новые подсистемы:

Data Analysis, Visualization and Application Development — анализ данных, их визуализация и применение;

Control Design — контроль проектирования;

DSP and Communications System Design — проектирование коммуникационных систем и DSP;

Financial Engineering — финансовая инженерия.

Из новых возможностей версии MATLAB 5.3 существенными являются следующие:

- новые улучшенные версии Simulink 3.0 и Real-Time Workshop 3.0;
- расширенные возможности по работе с целочисленными данными;
- новый графический интерактивный редактор, облегчающий форматирование графиков;
- поддержка нового стандарта NASA HDF-EOF;
- обеспечение записи и считывания изображений PNG (Portable Network Graphics);
- улучшенная визуализация трехмерных скалярных и векторных данных объемных поверхностей;
- новые решатели дифференциальных уравнений для матриц масс и дифференциально-алгебраических уравнений;

- улучшенный редактор и профилировщик М-файлов, содержащий генератор отчетов и имеющий HTML- формат записи файлов;
- улучшенная печать, предусматривающая предварительный просмотр печатаемых страниц — команда **Print Preview**.

Из одного перечисления новых возможностей версий MATLAB 5.* вытекает, что при создании этих версий речь шла не о косметических дополнениях, а о самой серьезной доработке как интерфейса, так и математических и графических возможностей системы MATLAB, дающих новый уровень обеспеченности математических расчетов и математического моделирования средствами MATLAB. А применение пакета Symbolic превращает MATLAB 5.* в подлинно универсальную математическую систему, реализующую не только численные, но и аналитические (символьные) вычисления в области компьютерной алгебры. Некоторые качества системы MATLAB заслуживают дополнительного обсуждения (см. ниже).

1.1.4. Интеграция с другими программными системами

В последние годы разработчики математических систем уделяют огромное внимание их **интеграции** и совместному использованию. Это не только расширяет класс решаемых каждой системой задач, но и позволяет подобрать под них самые лучшие инструментальные средства. Решение сложных математических задач сразу на нескольких системах существенно повышает вероятность получения корректных результатов — увы, как и математики, так и математические системы способны ошибаться, особенно при некорректной постановке решения задач неопытными пользователями.

С системой MATLAB могут интегрироваться такие популярные математические системы, как Mathcad, Maple V и Mathematica. Есть тенденция и к объединению математических систем с современными текстовыми процессорами. Так, новое средство последних версий MATLAB Notebook позволяет готовить документы в текстовом процессоре Word 95/97 [51] со вставками в виде документов MATLAB и результатами вычислений, представленными в численном, табличном или в графическом виде. Таким образом становится возможной подготовка "живых" электронных книг, в которых приведенные примеры могут быть оперативно изменены. Так, вы можете менять условия для решения задач и тут же наблюдать изменение результатов их решения.

В систему MATLAB встроено ядро одной из самых мощных и популярных систем символьной математики (компьютерной алгебры) Maple V R4. Оно используется пакетами расширения MATLAB Symbolic Math Toolbox и Extended Symbolic Math Toolbox, благодаря которым принципиально новые возможности **символьных** и **аналитических вычислений** стали возможны в среде MATLAB.

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, система Simulink позволяет выполнять моделирование сложных устройств с высокой степенью достоверности и прекрасными средствами представления результатов. По-

мимо интеграции с пакетами расширения Symbolic и Simulink MATLAB интегрируется с десятками мощных пакетов расширения, аннотационно описанных в Приложении.

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

1.1.5. 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:

|1 2 3 4|	**| 1 2 3 4 |** **| 5 6 7 8 |**	**| a a+b a+b/c |** **| x y*x z |**
|1,2,3,4|	**| 9 8 7 6 |**	**| 1 2 3 |**
Векторы из 4-х элементов	Матрица размера 3×4	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 так далее. Так что вектор является одномерным массивом, а матрица — двумерный массив с размерностью 2. MATLAB допускает задание и использование многомерных массивов, но в этой главе пока 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×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 **жирными символами**. Элементы векторов и матриц рассматриваются как индексированные переменные. Например:

$\mathbf{V}_2$ — второй элемент вектора **V**;
$\mathbf{M}_{2,3}$ — третий элемент матрицы **M**, расположенный во второй строке.

В свое время на пользователей ПК, ориентирующихся на математические расчеты, большое впечатление произвел язык АПЛ. В нем в качестве структур данных были введены векторы и матрицы, имелось множество встроенных математических операций, отсутствующих в других языках программирования. Однако АПЛ не получил широкого распространения в основном по двум причинам: он базировался на став-

ших сейчас стандартными кодах ASCII и требовал применения у ПК нестандартной клавиатуры, содержащей множество спецсимволов.

MATLAB является прекрасной альтернативой языку АПЛ. Эта система также имеет обширный набор векторных, матричных и других математических операций, но использует ASCII-коды и может применяться на ПК с обычной клавиатурой. Система MATLAB выполняет операции над векторами и матрицами даже в режиме **прямых вычислений** без какого-либо программирования. Ею можно пользоваться как мощнейшим калькулятором, в котором, наряду с обычными арифметическими и алгебраическими действиями, могут использоваться такие сложные операции, как инвертирование матрицы, вычисление ее собственных значений и векторов, решение систем линейных уравнений, вывод графиков двумерных и трехмерных функций и многое другое.

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 каждого их элемента. Данная операция обычно называется **векторизацией** и обеспечивает упрощение записи операций над всеми элементами векторов и матриц и существенное повышение скорости их выполнения.

1.1.6. Расширяемость системы

Сколь бы мощной ни была та или иная математическая система, она не способна включить в себя все средства, которые могут потребоваться сотням тысяч и миллионам пользователей. Поэтому важно, чтобы система была достаточно гибкой и способной адаптироваться к различным задачам пользователей самых различных категорий — начинающих и опытных математиков, инженеров и научных работников, аспирантов и студентов и даже школьников.

MATLAB — расширяемая система и ее легко приспособить к решению нужных вам классов задач. Ее огромное достоинство заключается в том, что это расширение достигается естественным путем и запоминается в виде так называемых М-файлов (с расширением .m). Иными словами, расширения системы хранятся на жестком диске компьютера и вызываются в нужный момент без какого-либо **предварительного объявления** или описания, увы, присущего расширениям в большинстве универсальных языков программирования.

Благодаря текстовому формату М-файлов пользователь может ввести в систему любую новую команду, оператор или функцию и пользоваться затем ими столь же просто, как и заведомо встроенными операторами или функциями. При этом, в отличие от языков программирования, таких как Бейсик, Си или Паскаль, не требуется никакого их объявления. Это роднит MATLAB с языками Лого и Форт [7, 8], имеющими словарную организацию операторов и функций и возможности пополнения словаря новыми определениями-словами. Новые определения в системе MATLAB хранятся в

виде файлов на диске, имеющих расширение .m. 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 др. Словом, MATLAB предоставляет пользователю обширный набор готовых средств (большая часть из них это внешние расширения в виде М-файлов).

Новый уровень расширения 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 так далее. Благодаря этому MATLAB обеспечивает высочайший уровень адаптации к решению задач конечного пользователя.

1.1.7. 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. 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 операторов и функций, которые 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 так далее и тому подобное. Число таких функций с учетом пакетов расширения системы уже достигает многих тысяч и непрерывно увеличивается.

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

Система MATLAB имеет входной язык, напоминающий Бейсик (с примесью Фортрана и Паскаля). Запись программ в системе традиционна и потому привычна для большинства пользователей ПК. К тому же система дает возможность редактировать программы с помощью любого, привычного для пользователя текстового редактора. Имеет она и собственный редактор с отладчиком. Отказ от присущего системе Mathcad "шика" — задания задач в формульном виде — компенсируется заметным увеличением скорости вычислений: при прочих равных условиях она почти на порядок выше, чем у системы Mathcad. А это немаловажное достоинство!

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.

1.1.8. 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 их изучает или просто просматривает. Большую роль в визуализации решения математических задач играет графическое представление результатов решения, причем как конечных, так и промежуточных.

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

Новая версия MATLAB напрочь избавилась от примитивности графиков, которая была присуща первым версиям этой системы. Теперь новые графические средства Handle Graphics (**дескрипторная,** или **описательная графика**) могут полноценно создавать **объекты** графики высокого разрешения, как геометрического, так и по цвету. 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. Возможен учет самых различных световых эффектов — вплоть до бликов на поверхности сложных фигур при освещении их различными источниками света и с учетом свойств материалов отражающих поверхностей. Применение дескрипторной (описательной) графики позволяет создавать и типовые элементы пользовательского интерфейса — кнопки, меню, информационные и инструментальные панели и так далее.

Графики выводятся отдельно от текстов в отдельных окнах. На одном графике можно представить множество кривых, отличающихся цветом (при цветном дисплее) и символами (кружками, крестиками, прямоугольниками и т.д.). Графики можно выводить в одно или 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 так далее). 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. В отличие от системы Mathcad, построение трехмерных фигур средствами системы 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.

Введен также ряд средств на основе **графического интерфейса пользователя** GUI (Graphics User Interface), лежащего в основе графического интерфейса операционных систем Windows 95/98. Это панели инструментов, редактор и отладчик М-файлов, красочная демонстрация возможностей и так далее. Есть и возможность создавать свои средства пользовательского интерфейса.

1.1.9. Техническая документация по системе

Система MATLAB поставляется с обширной англоязычной технической документацией и с развитой справочной системой. Система помощи реализована как в стандартном для систем MATLAB варианте — в интерактивном командном режиме, так и в форме Web-страниц (HTML-файлы) с просмотром их с помощью браузеров Netscape Navigator или Microsoft Internet Explorer.

Кроме того, имеется обширный пакет электронных документов в формате PDF (объемом около 200 Мбайт), для просмотра которых используется приложение Acrobat Reader. Ниже перечислены наиболее важные из этих документов (далеко не все) для версии MATLAB 5.2.1:

- Getting Started with MATLAB — начальное знакомство с системой MATLAB.
- Using MATLAB — работа с MATLAB в командном и программируемом режимах.
- Using MATLAB Graphics — справочник по средствам графики и визуализации.
- MATLAB Application Program Interface Guide — описание интерактивного взаимодействия с языками программирования C и Fortran.
- MATLAB 5 New Features Guide — описание новых возможностей пятой версии.
- MATLAB Installation Guide — описание процедуры инсталляции.
- Building GUIs with MATLAB — описание работы с пользовательским интерфейсом GUI.
- MATLAB Notebook User Guide — справочное руководство по приложению Notebook.
- MATLAB Late-Breaking News — последние сообщения по реализации.

Состав документации может несколько меняться в зависимости от типа поставок и номеров версий (от 5.0 и выше). Помимо этого различные поставки системы могут содержать и другую техническую документацию, например, двухтомный справочник

по функциям, руководство по работе с системой Simulink, описание пакета символьных вычислений и так далее.

Документацию по системе MATLAB можно рассматривать как многотомный справочник по современным численным методам и средствам их реализации на ЭВМ, включая персональные компьютеры (ПК). Одновременно это одно из самых полных справочных пособий по математике и многочисленным сферам ее применения. К сожалению, пока указанная документация поставляется с системой только на английском и на японском языках. Поскольку эта документация доступна каждому пользователю, рискнувшему установить достаточно полную версию MATLAB 5.0/5.3 на свой ПК, ссылок на нее в списке литературы нет.

Данная книга дает относительно полное описание большинства тщательно отобранных средств системы, достаточных для ее эффективного применения пользователями различных категорий. Ядро системы и такие важные приложения, как Notebook, Symbolic и Simulink, описаны достаточно полно. Тем не менее, настоятельно рекомендуется обращаться к указанной документации всякий раз, когда требуется полное знакомство с той или иной обширной сферой применения системы MATLAB, с ее редкими функциями, операторами или иными средствами системы, которые не нашли отражения в данной книге ввиду ограниченного ее объема (в сравнении с объемом указанной документации).

1.2. Инсталляция системы MATLAB

1.2.1. Аппаратные требования для работы с системой

Новые версии системы MATLAB 5.0/5.3 представляют собой весьма громоздкие программные комплексы, которые (при полной установке) требуют дисковой памяти до 350-900 Мбайт и более (в зависимости от конкретной поставки, полноты справочной системы и числа пакетов прикладных программ). Поэтому они поставляются исключительно на CD-ROM. Благодаря легкости копирования CD-ROM появилось множество "пиратских" копий системы MATLAB, причем не все они являются работоспособными. К тому же такие копии поставляются зачастую без полной документации (о том, что она англоязычная, уже говорилось).

Для успешной инсталляции MATLAB необходимы следующие минимальные средства:

- ПК с микропроцессором не ниже 386 и математическим сопроцессором 387;
- Привод CD-ROM, мышь и монитор класса SVGA;
- Операционная система Windows не ниже версии 3.1;
- ОЗУ емкостью 4 Мбайт для минимального варианта системы без цветной графики и 8 Мбайт для цветной графики (вообще желательно иметь память не менее 16 Мбайт);

- 8 Мбайт свободного пространства на жестком диске (и не менее 10 при использовании пакета прикладных программ SIMULINK).
- устранение предшествующей версии MATLAB.

Для использования расширенных возможностей системы нужны дополнительная память не менее 8 Мбайт, видеоадаптер, поддерживающий не менее 256 цветов, оснащенный графическим ускорителем, звуковая карта, Windows-совместимый принтер, текстовый процессор Microsoft Word 95/97 для реализации Notebook и компилятор языка C для подготовки собственных файлов расширения.

Мы рассматриваем систему, ориентированную на IBM- и Intel- совместимые ПК, как наиболее распространенные. Возможна работа MATLAB на ряде других компьютерных платформ: UNIX, Sun SPARC, HP 9000, DEC Alpha, IBM RS/6000, Silicon Graphics, Macintosh и др. Отличия между ними, в основном, связаны с отдельными деталями интерфейса и почти не затрагивают базового набора возможностей ядра и пакетов прикладных программ. Поэтому пользователи MATLAB на любой платформе могут пользоваться большей частью материалов данной книги.

1.2.2. Инсталляция системы

Инсталляция системы не имеет каких-либо специфических особенностей и подобна инсталляции других программных продуктов. От вас требуется задать свое имя (фамилию), сокращенное название организации и пароль, который указывается на инсталляционном CD-ROM или в его файле. Возможна типичная инсталляция и выборочная, в ходе которой вам предлагается выбор компонентов системы. Последняя предпочтительнее, так как из-за огромного полного объема системы ее установка не всегда возможна.

В Приложении дается аннотационное описание всех пакетов прикладных программ — компонентов системы MATLAB. Рекомендуется ознакомиться с этим описанием, прежде чем начинать установку системы. Нет никакого смысла использовать все компоненты, поскольку вы всегда можете пополнить по мере необходимости набор установленных компонентов системы. Установив только нужные компоненты, вы можете уменьшить затраты памяти на жестком диске в несколько раз.

Инсталляция осуществляется с помощью ряда окон Мастеров инсталляции. Первое из них показано на рис. 1.1. Ознакомившись с информацией окна, надо нажать кнопку **Next** (Следующий). Появится окно с текстом лицензионного соглашения. Если вы с ним согласны, следует вновь нажать кнопку **Next**.

Следующее окно (рис. 1.2) требует ввода пользователем своего имени и названия организации. Их можно ввести в упрощенном виде. Кроме того, надо указать полный номер копии вашего программного продукта, который сообщается фирмой MathWorks Inc. при поставке системы MATLAB. Необходимо также отметить, какую версию MATLAB вы намерены установить — 4.0 или 5.*.

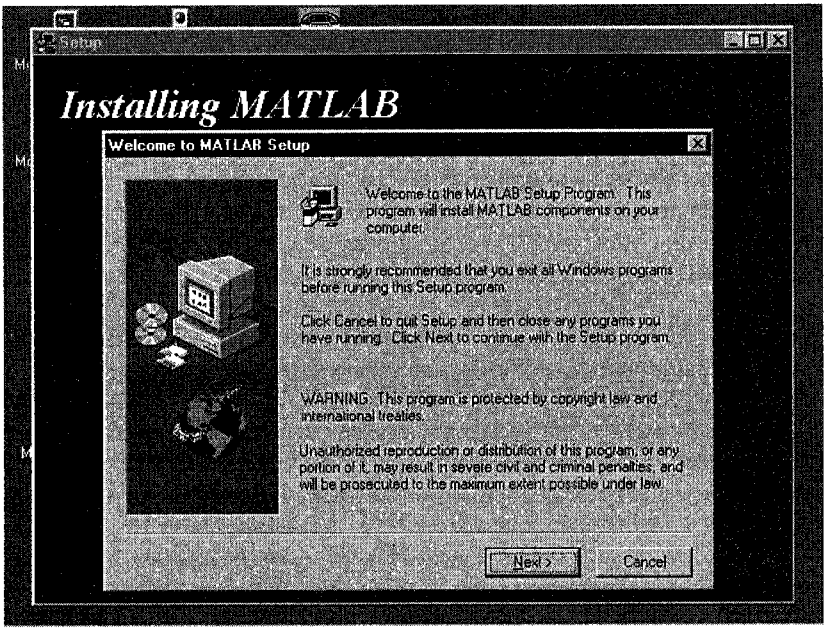


Рис. 1.1. Начало инсталляции

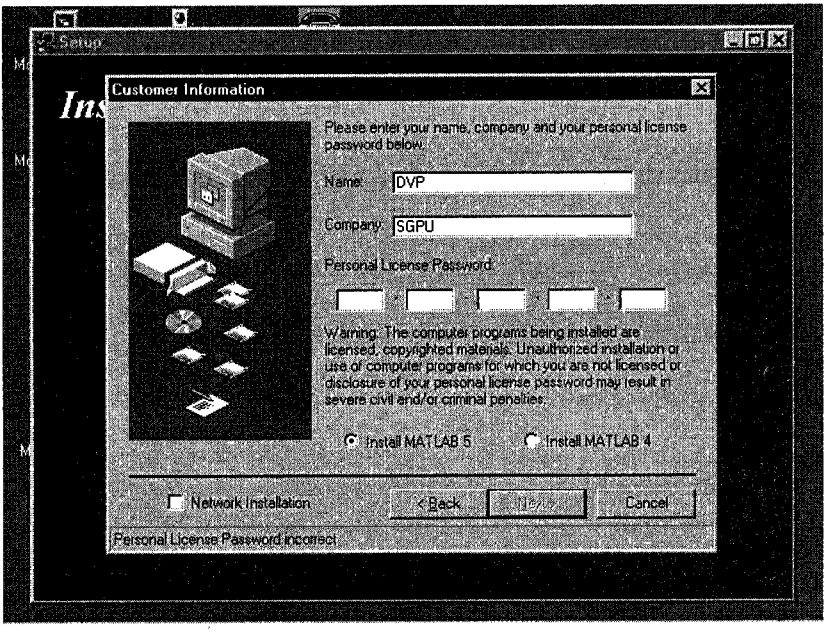


Рис. 1.2. Окно установки имени пользователя, организации, номера программного продукта и номера версии MATLAB

Выполнив эти установки и нажав кнопку **Next** (она становится доступной, если данные введены верно), можно получить окно с полным перечнем компонентов системы. Это окно представлено на рис. 1.3. В нем нужные компоненты надо отметить установкой с помощью мыши знака птички против имени соответствующих компонентов.

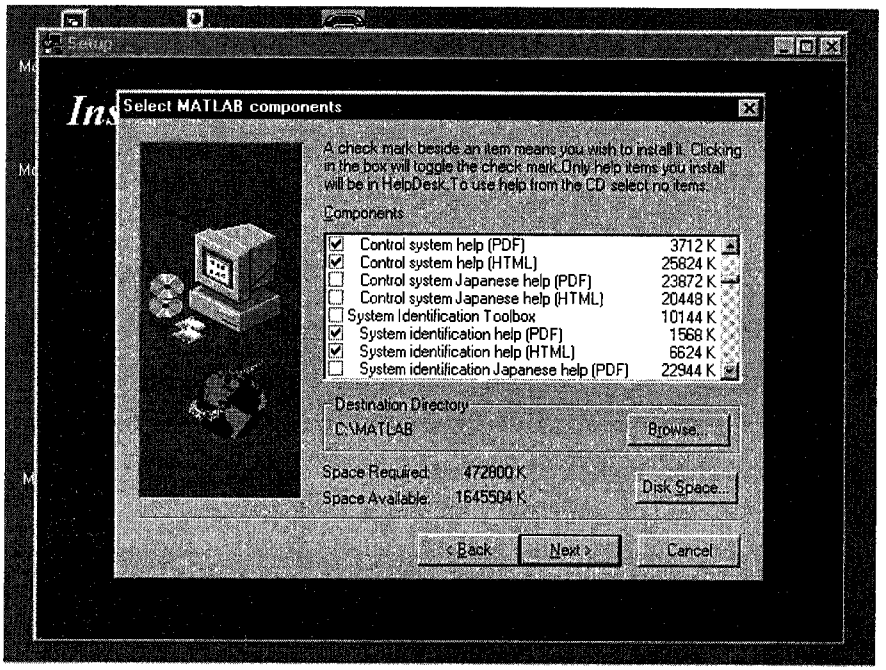


Рис. 1.3. Окно выбора компонентов системы MATLAB при инсталляций

После выбора необходимых компонентов надо нажать кнопку **Next**. После этого начинается самый длительный этап инсталляции. Все файлы системы разархивируются и переносятся в создаваемые для них директории. Этот процесс сопровождается контролем с помощью линейных индикаторов (рис. 1.4.).

Иногда возможно появление окон с сообщением о наличии на вашем ПК каких-либо файлов, которые входят и в состав системы MATLAB. В этом случае вам предоставляется возможность сохранить существующий файл или записать на его место новый файл системы MATLAB. Последнее, пожалуй, более целесообразно, так как MATLAB очень сложная и объемная система и лучше не рисковать инсталлировать ее с "чужими" файлами.

Процесс инсталляции даже на ПК Pentium II 350 МГц и с 32-скоростным приводом CD-ROM идет довольно медленно и занимает около 20 минут при установке полной версии системы. Завершается инсталляция появлением окна, показанного на рис. 1.5. В этом окне вам предлагается произвести перезагрузку компьютера или временно отложить ее.

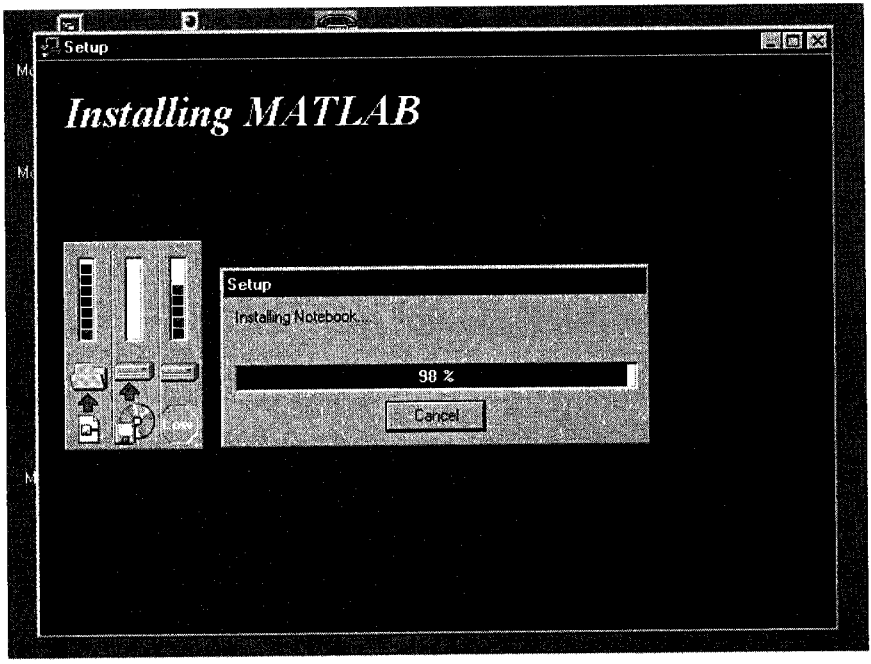


Рис. 1.4. Процесс инсталляции системы MATLAB

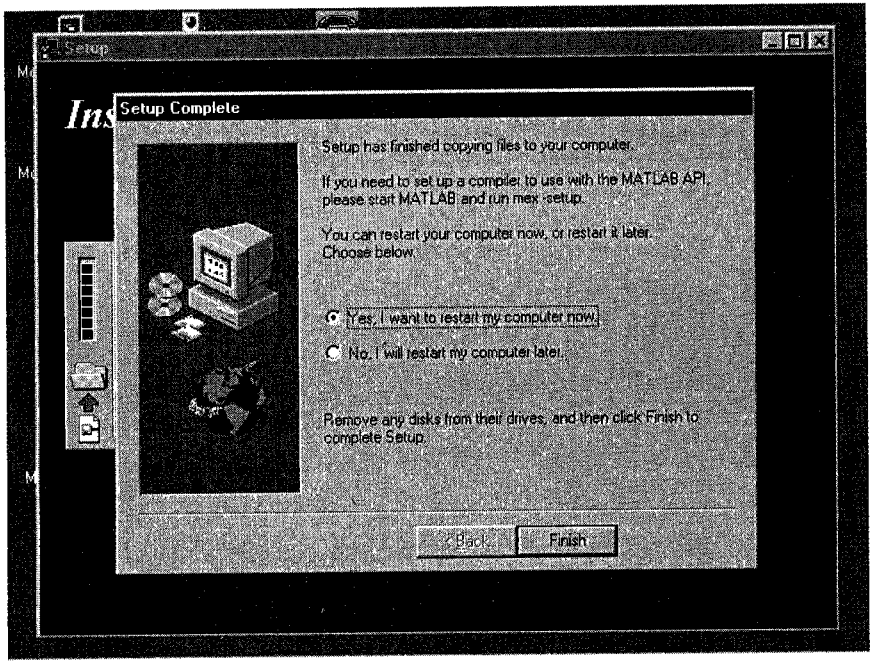


Рис. 1.5. Окно завершения процесса инсталляции

Запуск MATLAB без перезагрузки ПК не рекомендуется, поскольку может протекать не корректно. Так что если вы отложили перезагрузку ПК, не забудьте провести ее перед первым запуском системы MATLAB.

Для получения версии системы MATLAB 5.2.1 (реализация 10) возможны два пути. Можно приобрести эту версию и инсталлировать ее сразу. При этом процесс инсталляции практически полностью повторяет описанный выше. Можно приобрести предшествующую версию системы 5.2 и ее дополнение до версии 5.2.1. В этом случае инсталлируется отдельно версия 5.2, а затем ее дополнение, которое можно получить из Internet.

1.2.3. 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.

Файл это некоторая совокупность данных в широком понимании, хранящихся в памяти ПК — обычно на дисковых накопителях и объединенных под некоторым **именем файла**, после которого через точку указывается **расширение файла**. Файлы могут содержать машинные коды (исполняемые файлы), тексты (текстовые файлы), исходные данные для математических расчетов и результаты их выполнения.

В MATLAB особое значение имеют файлы двух типов — с расширениями .mat и .m. Первые являются бинарными файлами, представляющими запись сеанса (сессии) работы системы. Вторые представляют собой текстовые файлы, содержащие внешние 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 (на языке C), файлы с откомпилированными кодами с расширением .mex и другие. Исполняемые файлы имеют расширение .exe.

Особое значение имеет директория MATLAB/TOOLBOX/MATLAB. В ней содержится набор M-файлов стандартного расширения системы, называемый **Toolbox**. Просмотр этих файлов позволяет детально оценить возможности поставляемой конкретной версии системы. Ниже перечислены основные подкаталоги с этими файлами (деление условно, на самом деле все подкаталоги находятся в общем каталоге MATLAB).

Команды общего назначения

General — команды общего назначения.

Операторы, конструкции языка и системные функции

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.

Основные математические и матричные функции

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.

Команды графики

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

Полный состав файлов каждого подкаталога можно вывести на просмотр с помощью команды **help имя**, где имя — название соответствующего подкаталога.

1.3. "Пятиминутное" знакомство с MATLAB

1.3.1. Запуск MATLAB и исполнение команды matlabrc

В этой книге предполагается, что MATLAB используется в среде операционных систем Windows 95 или Windows 98. Копии сеансов работы MATLAB даны именно для этих случаев. Однако пользователи, работающие с более старыми операционными системами Windows 3.1/3.11, также могут обращаться к материалам данной книгой без каких-либо ограничений, поскольку отличия касаются лишь мелких деталей пользовательского интерфейса. В меньшей мере, но это справедливо и для пользователей систем MATLAB на иных компьютерных платформах.

Рис. 1.6 иллюстрирует подготовку к запуску системы MATLAB 5.2.1 из рабочего меню операционной системы Windows 98 со стандартным видом рабочего стола, подобным использованному в Windows 95. Для вывода рабочего меню активизируется

кнопка **Пуск** в начале панели переключателя задач, расположенной снизу рабочего стола.

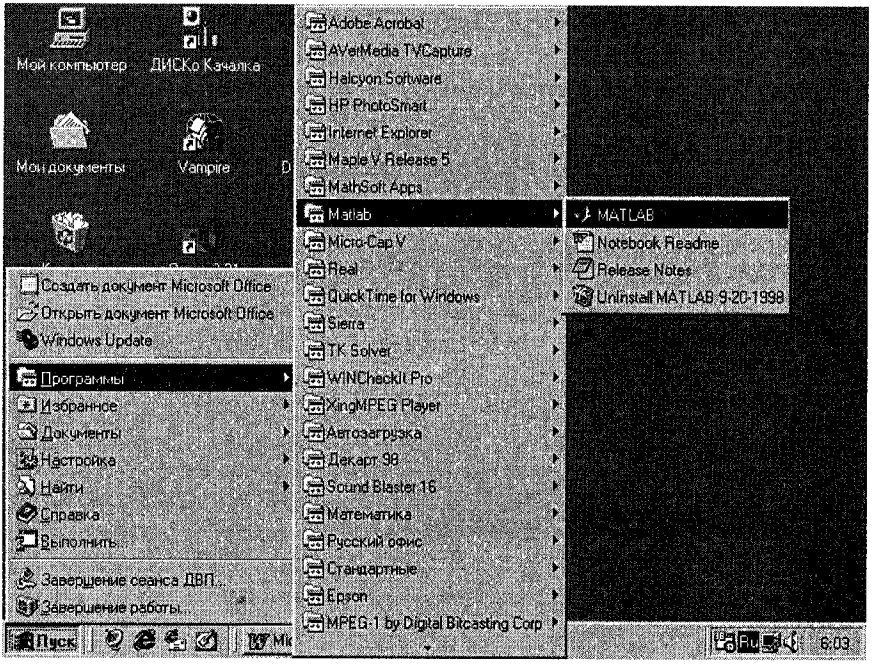


Рис. 1.6. Подготовка к запуску MATLAB

Далее мы не будем повторять полное название системы — MATLAB 5.2.1 и ограничимся сокращенным названием MATLAB.

После запуска одиночным нажатием левой клавиши мышки в Windows 98 и двойным в Windows 95 ярлыка MATLAB (см. рис. 1.6) появляется окно системы MATLAB, показанное на рис. 1.7. Обычно это окно раскрыто не полностью и занимает часть рабочего стола. Вы можете раскрыть окно полностью, нажав среднюю из трех кнопок, расположенных в конце титульной (верхней) строки окна. Левая кнопка сворачивает окно в кнопку с именем приложения, помещаемую в панель переключателя задач Windows 95/98, а правая закрывает окно и прекращает работу с MATLAB.

Система сразу же готова к проведению вычислений в командном режиме, присущем самым первым реализациям ее под MS-DOS. При этом вы можете не обращать внимания на новации пользовательского интерфейса, привнесенного операционными системами Windows 95 и 98, в виде расширяемого окна и панели инструментов. Мы обсудим их роль позже.

Полезно знать, что в начале запуска автоматически выполняется команда **matlabrc**, которая исполняет загрузочный файл matlabrc.m и файл startup.m, если 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. Таким образом опытные пользователи могут выполнить настройку системы под свои запросы. Однако в большинстве случаев в этом особой необходимости нет. Поскольку указанные файлы имеют текстовый формат, их легко просмотреть с помощью какого-либо текстового редактора или с помощью команды **type** в командном режиме работы.

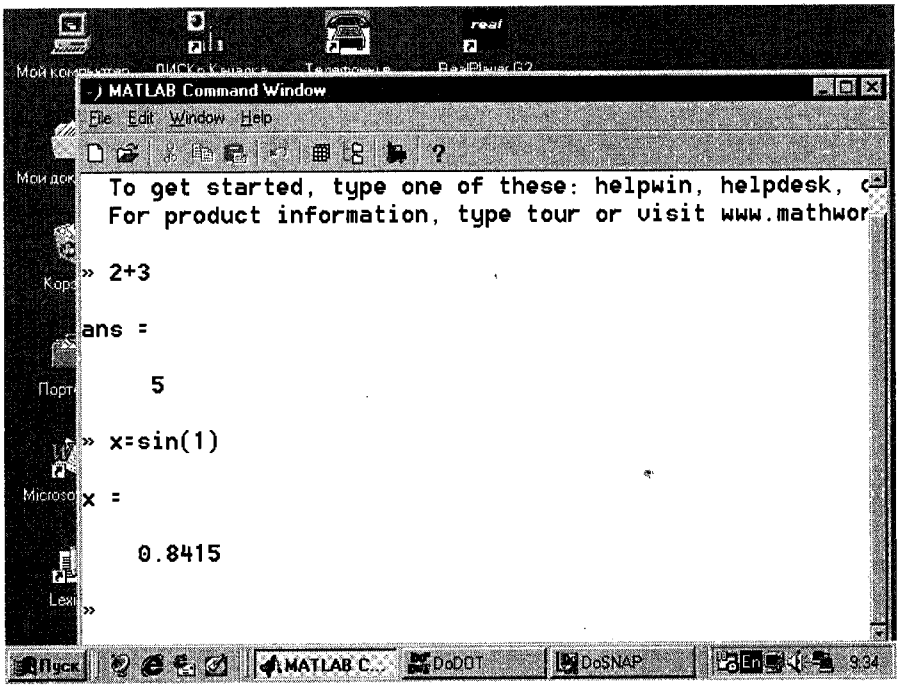


Рис. 1.7. Окно системы MATLAB после запуска и выполнения простых вычислений

Сеанс работы с MATLAB принято именовать **сессией**. 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.** Команда **load** позволяет считать с диска данные рабочей области. Фрагменты сессии можно оформить в виде дневника с помощью команды **diary**. Позже мы обсудим эти команды подробно.

1.3.2. Операции строчного редактирования

При работе с MATLAB действует простейший строчный редактор. Ниже перечислены его команды:

→ 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,n** – перелистывание строк вверх или вниз для подстановки в строку ввода,
Del или **Ctrl-d** – стирание символа, на котором установлен курсор,
←— или **Ctrl-h** – стирание символа слева от курсора,
Ctrl-k — стирание до конца строки,
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 под MS-DOS. Они обеспечивают важное свойство новых версий систем — их преемственность со старыми версиями в части навыков работы. Позже вы увидите, что в новых версиях есть вполне современный редактор со средствами отладки создаваемых документов — М-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 программного стека, хранящего строки с исполненными ранее командами.

1.3.3. Команды управления окном clc, home и echo

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 <name_file> on — включает режим вывода на экран текста Script-файла (файла-сценария).
echo <name_file> off — выключает режим вывода на экран текста Script-файла.
echo <name_file> — меняет режим вывода на противоположный.
echo on all — включает режим вывода на экран текста всех М-файлов.
echo off all — отключает режим вывода на экран текста всех М-файлов.
more on — включает режим постраничного вывода (полезен при просмотре больших М-файлов).
more off — отключает режим постраничного вывода (в этом случае для просмотра больших фаqлов надо пользоваться линейкой прокрутки).

В версии MATLAB 5.2.1 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 М-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 М-файлами сценариями и функциями мы обсудим позже.

1.3.4. MATLAB в роли суперкалькулятора

Система MATLAB создана таким образом, что любые (подчас весьма сложные вычисления) можно выполнять в режиме прямых вычислений, т.е. 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 носит диалоговый характер. Пользователь набирает на клавиатуре вычисляемое выражение, редактирует его (если нужно) в командной строке и завершает ввод нажатием клавиши ENTER. В качестве примера на рис. 1.7 уже были показаны простейшие вычисления — выражения 2+3 и значения sin(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 со значением 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 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 (рис. 1.8) иллюстрирует применение системы MATLAB для выполнения векторных операций. В этом примере задается четырехэлементный вектор **V** со значениями элементов 1, 2, 3 и 4. А далее (сосредоточьте на этом внимание!) вычисляются функции синуса и экспоненты с аргументом в виде вектора, а не скаляра.

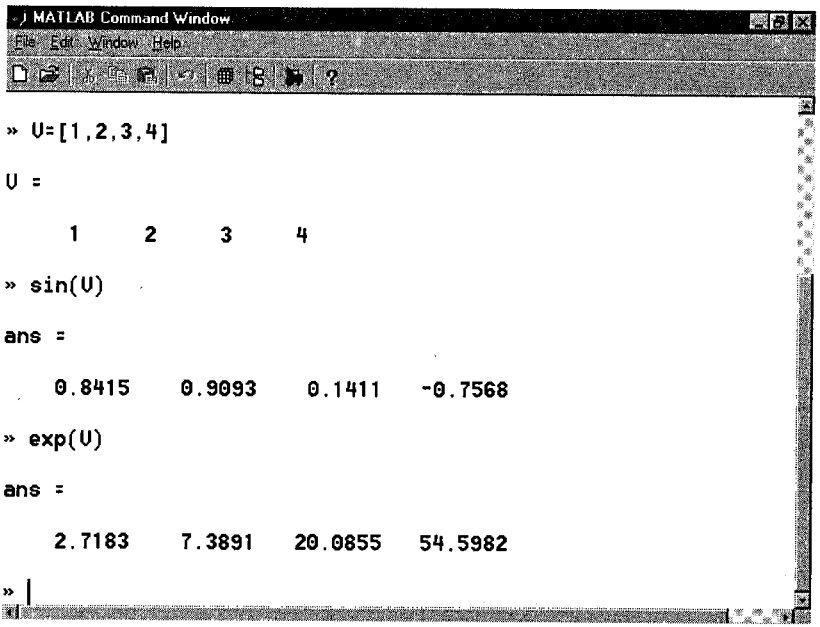


Рис. 1.8. Примеры простых векторных операций

Две записи для вектора

V=[1 2 3 4] и **V=[1,2,3,6]**

являются идентичными. Таким образом, векторы задаются списком своих элементов, разделяемым 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 размер вектора V, но элементы возвращаемого вектора будут синусами и экспонентами от элементов вектора V.

Еще один пример демонстрирует простейшие операции с матрицей — рис. 1.9. Здесь задана матрица **M** с размером 2 × 2 и вычислена матрица **MX**=sin(**M**).

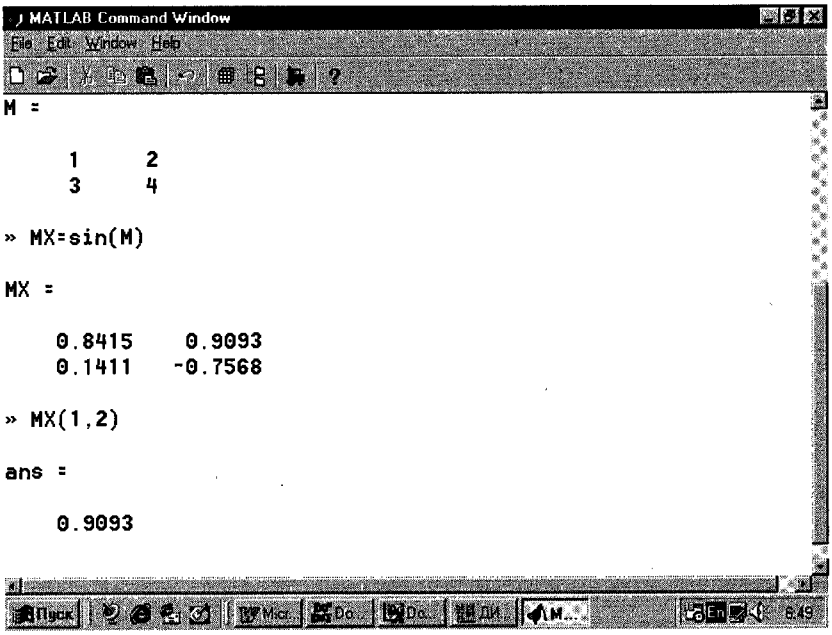


Рис. 1.9. Простейшие операции с матрицей

Матрица задается в виде ряда векторов, представляющих ее строки, 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.

1.3.5. О форме представления сессии

Как видно из рис. 1.7 — 1.9, ввод исходных выражений для вычислений в системе MATLAB осуществляется в самом обычном текстовом формате. В этом же формате выдаются результаты вычислений, за исключением графических. В этой связи в последующем материале для большинства примеров применения системы окна сессий не будут показываться, а нужные фрагменты сессий будут приводиться прямо в тексте книги. При этом строки ввода будут отмечаться маркером ввода **»** в их начале. Ради компактности записи пустые строки будут опускаться.

Приведем пример записи вычислений на рис. 1.7 и 1.8:

```
To get started, type one of these: helpwin, helpdesk, or demo.
For product information, type tour or visit www.mathworks.com.

» 2+3
 ans
   5
```

```
» x=sin(1)
x =
    0.8415
» V=[1 2 3 4]
V =
     1    2    3    4
» sin(V)
ans
    0.8415    0.9093    0.1411   -0.7568
» exp(V)
ans
    2.7183    7.3891   20.0855   54.5982
```

Сравните эти записи с записями в реальных сессиях (рис. 1.7 и 1.8). Вы, наверняка, отметите, что они практически идентичны. 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.

1.3.6. О переносе строки в сессии — символы ...

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 — 4096 (в ранних версиях оно ограничивалось 256), но со столь длинными строками работать неудобно.

1.4. Математические выражения и их компоненты

1.4.1. 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 и функций и разных спецзнаков. Ниже даются краткие пояснения о сути этих понятий.

1.4.2. Действительные и комплексные числа

Числа — простейший объект языка MATLAB, представляющий количественные данные. Числа можно считать константами, имя которых совпадает с их значениями. Числа используются в общепринятом представлении о них. Они могут быть целыми, дробными, с фиксированной точкой и плавающей точкой. Возможно представление чисел в хорошо известном научном формате с указанием мантиссы и порядка числа. Ниже приведены 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, что 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=Re(x)+Im(x)*i. Такие числа содержат действительную Re(z) и мнимую Im(z) части. Мнимая часть имеет множитель i или j, означающей корень квадратный из -1:

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

Функция **real(z)** возвращает действительную часть комплексного числа **Re(z)**, а функция **imag(z)** – мнимую Im(Z). Для получения модуля комплексного числа используется функция **abs(z)**, а для вычисления фазы – **angle(z)**. Ниже даны простейшие примеры на работу 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 не принято делить числа на целые и дробные, короткие и длинные и так далее, как это принято в большинстве языков программирования. Хотя задавать их в таких формах можно. Вообще же операции над числами выполняются в формате, который принято считать форматом чисел 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.

1.4.3. 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 называть **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 "пи" 3.1415926...;
eps — погрешность для 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 не числовой характер данных (Not-a-Number).

Вот примеры на применение 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 переменных, системные переменные не могут быть никогда.

Символьные константы — 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.

1.4.4. Текстовые комментарии

Поскольку MATLAB используется для достаточно сложных вычислений, важное значение имеет наглядность их описания. Она достигается, в частности, использованием текстовых комментариев.

Текстовые комментарии вводятся с помощью оператора — символа %, например:

```
%Ниже представлено задание функции вычисления факториала
```

Обычно в определениях М-файлов первые строки определений служат для их описания, которое выводится на экран дисплея после команды

```
» help Имя_файла
```

Считается правилом хорошего тона вводить достаточно подробные текстовые комментарии, в частности, в М-файлы. 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 содержащие их наборы шрифтов проинсталлированы. В сами системы наборы русифицированных шрифтов не входят, и их надо инсталлировать отдельно, либо использовать те наборы, которые включены в русифицированную операционную систему Windows 95/98/NT.

1.4.5. Переменные и присваивание им значений — оператор =

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 запоминаются и идентифицируются только 31 первых символа. 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 цифры и символ подчеркивания объединения _. Недопустимо включать в имена переменных пробел и специальные знаки, например: +, -, *, / и так далее, поскольку в этом случае интерпретация выражений затрудняется.

Желательно давать 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'**.

1.4.6. Уничтожение определений переменных — команда clear

В памяти переменные занимают определенное место, называемое рабочим пространством — **workspace**. Для очистки рабочего пространства используется функция clear в разных формах, например:

clear — уничтожение определений всех переменных;
clear x — уничтожение определения переменной x;
clear a, b, c — уничтожение определений переменных списка и так далее.

Уничтоженная (стертая в рабочем пространстве) переменная становится неопределенной. Использовать такие 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 остальные переменные.

1.4.7. Дефрагментация рабочей области командой pack

По мере задания одних переменных и стирания других рабочая область перестает быть непрерывной и содержит "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 пере-

писывает все определения рабочей области на жесткий диск, очищает ее и затем записывает все определения без "дыр" и "мусора" в рабочую память.

1.4.8. 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 почленное умножение и деление массивов.

Следующие примеры поясняют сказанное:

```
» 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         kron
.....
```

Функции — это имеющие уникальные имена объекты, выполняющие определенные преобразования над своими аргументами 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 явно в математических выражениях, например **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 числа и **X** в виде вектора. Многие функции допускают ряд форм записи, например, отличающихся списком своих параметров. Если функция возвращает несколько значений, то она записывается в виде:

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

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

Со списком элементарных функций можно ознакомиться, выполнив команду **help elfun**, а со списком специальных функций — **help specfun**. О всех категориях функций говорилось в разделе 1.2.3.

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 М-файлах. Задание таких функций с помощью специального редактора М-файлов мы рассмотрим в главе 2. Встроенные функции хранятся в откомпилированном ядре системы MATLAB, в силу чего они выполняются предельно быстро.

1.4.9. Применение оператора : (двоеточие)

Очень часто необходимо произвести формирование упорядоченных числовых последовательностей. Такие последовательности нужны для создания векторов или зна-

чений абсциссы при построении графиков. Для этого в MATLAB используется оператор : (двоеточие):

Начальное_значение:Шаг:Конечное_значение

Данная конструкция порождает последовательность чисел, которая начинается с начального значения, идет с заданным шагом и завершается конечным значением. При этом действуют следующие правила:

Начальное_значение < Конечное_значение Шаг > 0
Начальное_значение > Конечное_значение Шаг < 0

Если шаг не задан, то он принимает значение 1 или -1 в указанных соотношениях. 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 =
   10   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
```

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 функции sin(x)/x дает на первый взгляд неожиданный 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 массив значений sin(x)/x, надо использовать специальный оператор почленного 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. Так, в приводимом ниже примере вычислены функции Бесселя порядка от 0 до 5 со значением аргумента x=0.5:

```
x=1/2:
» 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.

1.4.10. Сообщения об ошибках и исправление последних

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 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 случаев. 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 ↓ перелистывания строк. В результате появится выражение:

```
» sqr(2)
```

Теперь с помощью клавиши → следует установить курсор после буквы r и нажать клавишу T, а затем клавишу ENTER. Выражение примет вид:

```
» sqrt(2)
ans =
      1.4142
```

Теперь 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 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 забыли об этом или ввели имя несуществующего определения, то система отреагирует на это звуковым сигналом 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 М-функция. Зато далее мы видим, что функция с именем **sinh** имеется в составе функций системы MATLAB — она задана в виде М-функции. Между тем, в последнем примере мы не давали системе никаких указаний на применение этой внешней функции! И это вычисление прошло так же просто, как вычисление встроенной функции — например **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 слово из служебных слов системы, если оно есть, то нужные вычисления выполняются сразу, если нет — система ищет М-файл с соответствующим именем на диске. Если файла нет, выдается сообщение об ошибке и вычисления останавливаются. Если же файл найден, он загружается с жесткого диска в память машины и исполняется. Этот алгоритм аналогичен применяемому в развиваемых и адаптируемых к задачам пользователя языкам программирования ЛОГО и ФОРТ [7,8].

Иногда в ходе вывода данных вычислений появляется сокращение **NaN** (от слов Not a Number – не число). Оно обозначает операцию неопределенности, 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 0). Весь диапазон представления чисел 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.

1.4.11. Форматы чисел — команда format

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 порядка), **long** – длинное представление в фиксированном формате (15 знаков), **long e** – длинное представление в экспоненциальном формате (15 знаков мантиссы и 3 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.

1.5. Основы работы с векторами и матрицами

1.5.1. Особенности задания векторов и матриц

Описанные выше простые правила вычислений распространяются на гораздо более сложные вычисления, которые (при использовании обычных языков программирования типа Бейсик или Паскаль) требуют составления специальных программ. MATLAB – система, специально предназначенная для проведения сложных вычислений с векторами, матрицами и массивами [38]. При этом она по умолчанию предполагает, что каждая заданная переменная — это вектор или матрица. Все определяется конкретным значением переменной. Например, если задано **X=1**, то это значит, что **X** есть вектор с единственным элементом, имеющим значение 1. Если надо задать вектор из трех элементов, то их значения надо перечислить в квадратных скобках, разделяя пробелами. Так, например,

```
» V=[1 2 3]
V =
   1 2 3
```

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

Задание матрицы требует указания различных строк. Для различения строк используется знак ; (точка с запятой). Этот же знак (или знак запятой) в конце ввода предотвращает вывод матрицы или вектора 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(i;j)=x
```

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

```
» M(2;2)=10
```

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*]
```

Это создает матрицу:

```
| 1+5*i    2+6*i |
|                 |
| 3+7*i    4+8*i |
```

Наряду с операциями над отдельными элементами матриц и векторов система позволяет производить операции умножения, деления и возведения в степень сразу над всеми элементами — массивами. Для этого перед операцией ставится знак точка. Например, знак * означает знак умножения для векторов или матриц, а знак .* — умножение всех элементов в виде 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 равноценными.

Имеется также ряд особых функций для задания векторов и матриц. Например, функция **magic(n)** задает магическую матрицу размера n×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 векторов и матриц из рабочего пространства служит команда **clear**.

1.5.2. 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**, т. е. замену строк столбцами. На этот раз сумма оказалась разной. Это отвергает изначально возникшее предположение, что матрица **B** 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, одинаковой является и сумма элементов по основным диагоналям.

1.5.3. 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 над 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:

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

А теперь удалим вторую строку:

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

На этом мы закончим начальную экскурсию в технику матричных вычислений системы MATLAB. Мы расширим представления о ней в дальнейших главах этой книги.

1.6. Графическая визуализация вычислений

1.6.1. Построение графиков 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 в диапазоне изменения аргумента x от 0 до 10 с шагом 0.1. Для построения графика достаточно вначале задать вектор **x=0:0.1:10**, а затем использовать функцию построения графиков **plot(sin(x))**. Это показано на рис. 1.10.

Теперь 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 представлены в виде функций пользователя, имеющих имена без указания аргумента:

Полный список операторов можно получить, используя команду

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

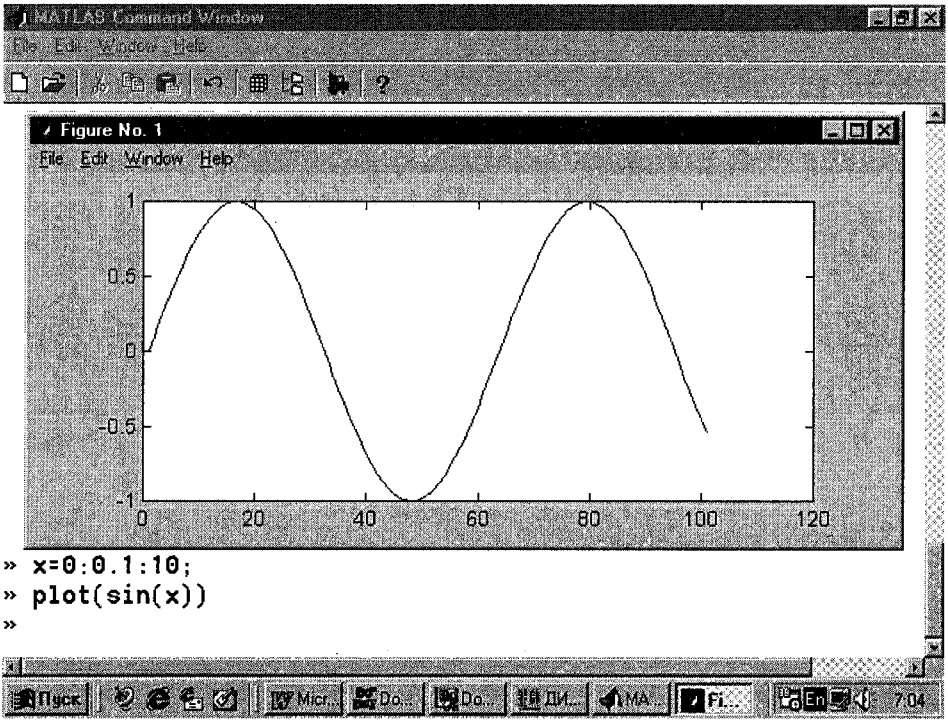


Рис. 1.10. Пример построения графика синусоиды

Теперь можно использовать одну из ряда форм оператора **plot**:

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

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

» plot(x,y1,x,y2,x,y3)

На рис. 1.11 показано построение графика по этим правилам. Нетрудно заметить, что графики функций y1 и y2 благополучно построены. А вот график функции y3 отсутствует. Причина этого казуса уже обсуждалась — при вычислении функции y3=sin(x)/x и при x в виде массива 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. 1.12.

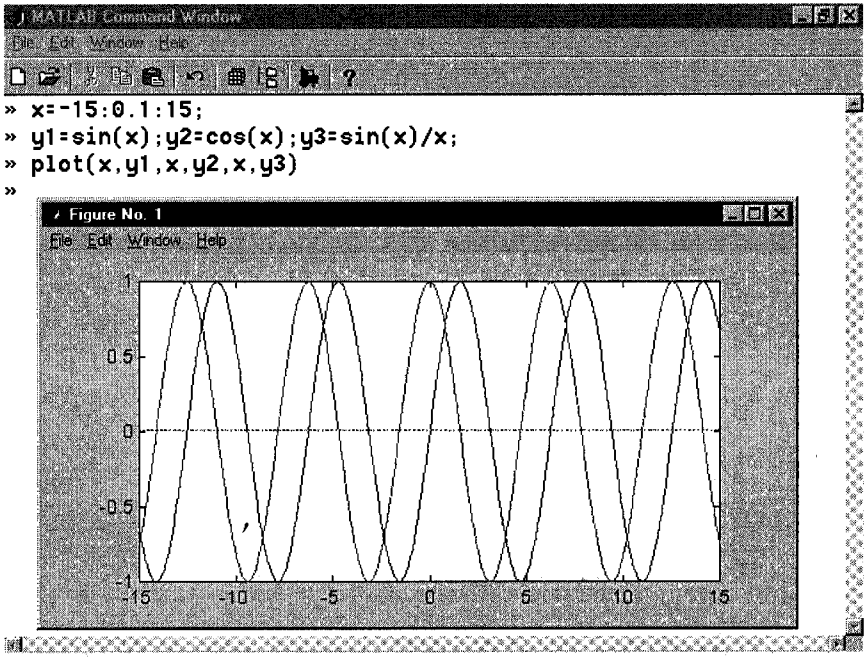


Рис. 1.11. Построение графика трех функций с явной ошибкой — один из графиков не построен

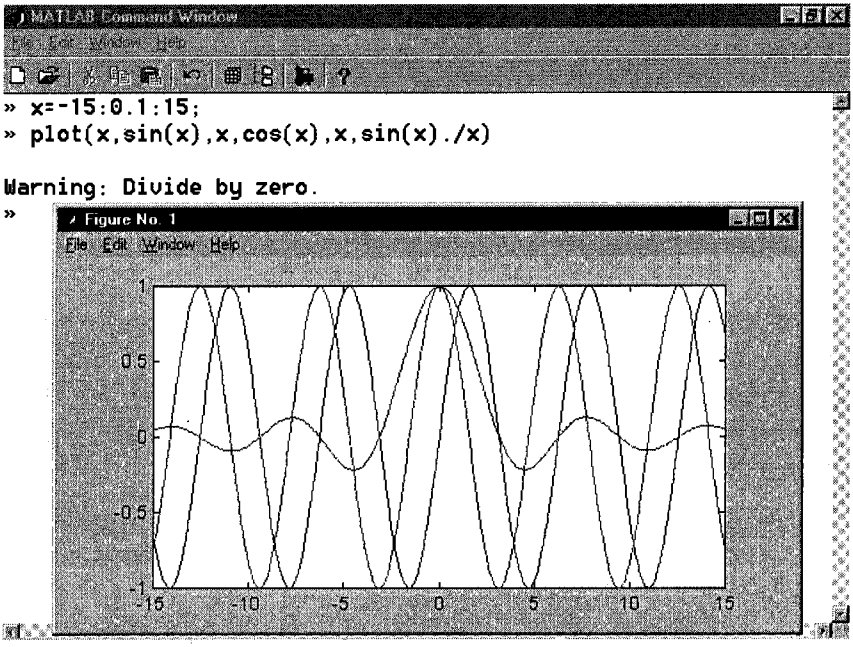


Рис. 1.12. Построение графиков трех функций

Разумеется, MATLAB имеет средства для построения графиков и таких функций. Не обсуждая их подробно (в дальнейшем такое обсуждение состоится), просто покажем, как это делается с помощью другого графического оператора

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

Он позволяет строить функцию, заданную в строковом формате в интервале изменения аргумента x от xmin до xmax. Один из вариантов его применения демонстрирует рис. 1.13. Обратите также внимание на две используемые команды: **clear** — очистка графического окна и **grid on** — включение отображения сетки, которая строится точечными линиями.

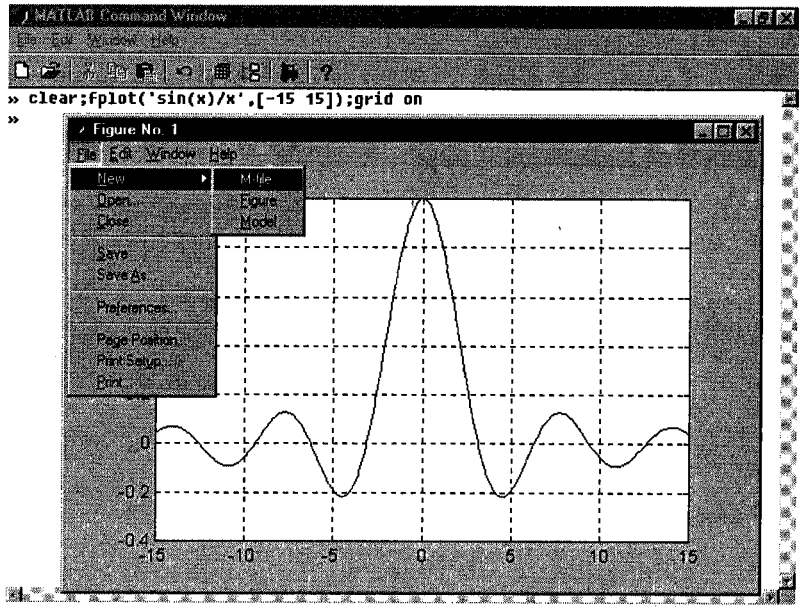


Рис. 1.13. Построение графика функции sin(x)/x

Позже мы более подробно рассмотрим возможности различных графических операторов. В частности, 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 по форматированию графиков для придания им более наглядного вида, соответствующего требованиям пользователя. Мы также обсудим множество новых форм применения графических операторов, резко расширяющих их возможности в построении графиков всех мыслимых типов.

1.6.2. Построение гистограммы

В прикладных расчетах часто встречаются графики, именуемые гистограммами. Чаще всего используются столбиковые гистограммы (или диаграммы), отражающие

содержание некоторого вектора V. При этом каждый элемент вектора представляется столбиком, высота которого пропорциональна значению элемента. Столбики нумеруются и масштабируются по отношению к максимальному значению наиболее высокого столбика. Выполняет построение команда (оператор) **bar(V)** — рис.1.14.

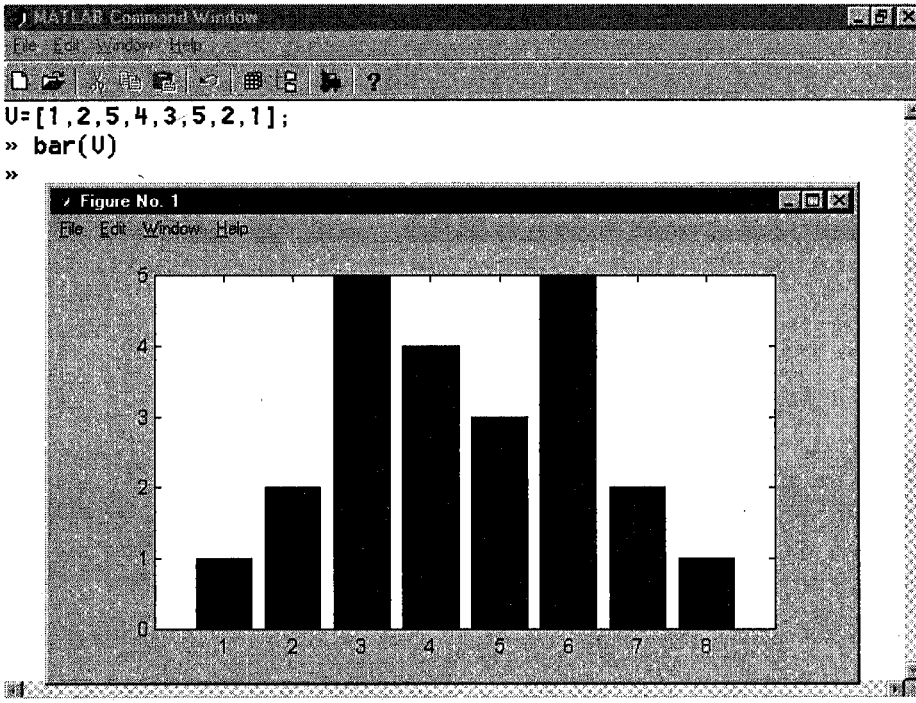


Рис. 1.14. Построение гистограммы значений элементов вектора

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

1.6.3. Построение графиков трехмерных поверхностей

Столь же просто обеспечивается построение графиков сложных трехмерных поверхностей. Надо только знать, какой командой реализуется тот или иной график. Например, для построения графика трехмерной 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. 1.15. Раньше для построения такого графика пришлось бы убить много дней на составление и отладку нужной программы. В MATLAB можно в считанные секунды изменить функцию Z(X,Y), задающую поверхность, и тут же получить новый график.

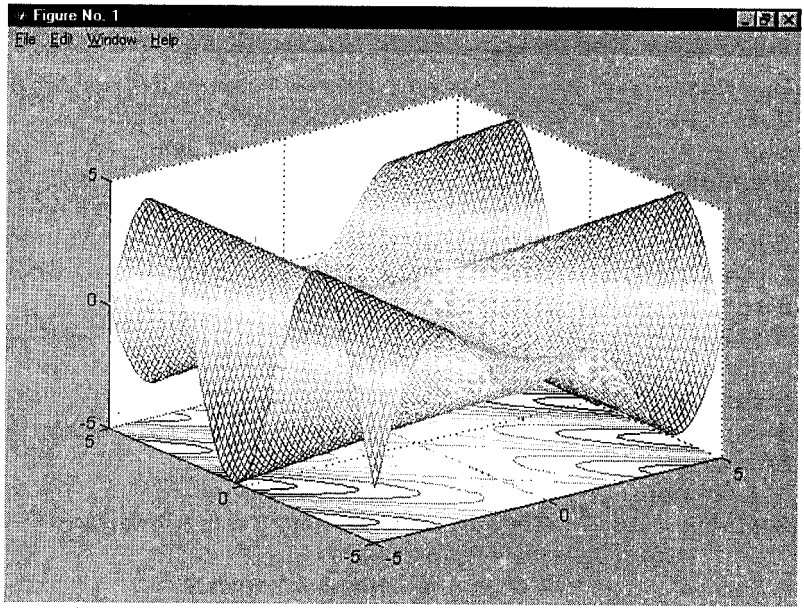


Рис. 1.15. Окно с графиком 3-D поверхности и ее проекции на плоскость под фигурой

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 поможет как данная книга, так и справочная система MATLAB.

1.7. Работа с интерактивной справочной системой

1.7.1. Вызов списка примеров интерактивной справки

MATLAB имеет интерактивную систему помощи, которая реализуется в командном режиме с помощью ряда команд. Одной из них является команда

```
» help
```

которая выводит весь список папок (каталогов), содержащих М-файлы с определениями операторов, функций и иных объектов. Ниже приводится этот список:

HELP topics:

MATLAB\general — General purpose commands.
MATLAB\ops — Operators and special characters.
MATLAB\lang — Programming language constructs.
MATLAB\elmat — Elementary matrices and matrix manipulation.
MATLAB\elfun — Elementary math functions.
MATLAB\specfun — Specialized math functions.
MATLAB\matfun — Matrix functions — numerical linear algebra.
MATLAB\datafun — Data analysis and Fourier transforms.
MATLAB\polyfun — Interpolation and polynomials.
MATLAB\funfun — Function functions and ODE solvers.
MATLAB\sparfun — Sparse matrices.
MATLAB\graph2d — Two dimensional graphs.
MATLAB\graph3d — Three dimensional graphs.
MATLAB\specgraph — Specialized graphs.
MATLAB\graphics — Handle Graphics.
MATLAB\uitools — Graphical user interface tools.
MATLAB\strfun — Character strings.
MATLAB\iofun — File input/output.
MATLAB\timefun — Time and dates.
MATLAB\datatypes — Data types and structures.
MATLAB\winfun — Windows Operating System Interface Files (DDE/ActiveX)

MATLAB\demos — Examples and demonstrations.
database\database — Database Toolbox.
database\dbdemos — Database Toolbox Demonstration Functions.
powersys\powerdemo — Power System Blockset Demos.
powersys\powersys — Power System Blockset
toolbox\compiler — MATLAB Compiler
comm\comm — Communications Toolbox
comm\commsfun — Communications Toolbox SIMULINK S-functions.
comm\commsim — Communications Toolbox SIMULINK files.
toolbox\symbolic — Symbolic Math Toolbox.
nag\nag — NAG Foundation Toolbox — Numerical & Statistical Library

nag\examples — NAG Foundation Toolbox — Numerical & Statistical Library

map\map — Mapping Toolbox
map\mapdisp — Mapping Toolbox Map Definition and Display.
map\mapproj — Mapping Toolbox Projections.
wavelet\wavelet — Wavelet Toolbox.
wavelet\wavedemo — Wavelet Toolbox Demos.
toolbox\pde — Partial Differential Equation Toolbox.
finance\finance — Financial Toolbox.
finance\calendar — Financial Toolbox calendar functions.
finance\findemos — Financial Toolbox demonstration functions.

```
lmi\lmictrl              — LMI Control Toolbox: Control Applications
lmi\lmilab               — LMI Control Toolbox
qft\qft                  — QFT Control Design Toolbox.
qft\qftdemos             — QFT Control Design Toolbox Demos
toolbox\fixpoint         — Fixed-Point Blockset
dspblks\dspblks          — DSP Blockset.
dspblks\dspmex           — (No table of contents file)
dspblks\dspdemos         — (No table of contents file)
fuzzy\fuzzy              — Fuzzy Logic Toolbox.
fuzzy\fuzdemos           — Fuzzy Logic Toolbox Demos.
mpc\mpccmds              — Model Predictive Control Toolbox.
mpc\mpcdemos             — Model Predictive Control Toolbox
fdident\fdident          — Frequency Domain Identification Toolbox.
fdident\fddemos          — Demonstrations for the FDIDENT Toolbox
hosa\hosa                — Higher-Order Spectral Analysis Toolbox.
hosa\hosademo            — Higher-Order Spectral Analysis Toolbox — Demo suite
toolbox\stats            — Statistics Toolbox.
toolbox\ncd              — Nonlinear Control Design Blockset
images\images            — Image Processing Toolbox.
images\imdemos           — Image Processing Toolbox -- demos and sample
                           images
nnet\                    — Neural Network Toolbox.
nnet\nndemos             — Neural Network Demonstrations.
nnet\nnutils             — (No table of contents file)
nnet\nnobsolete          — (No table of contents file)
mutools\commands         — Mu-Analysis and Synthesis Toolbox.
mutools\subs             — Mu-Analysis and Synthesis Toolbox.
toolbox\signal           — Signal Processing Toolbox.
toolbox\splines          — Spline Toolbox.
toolbox\optim            — Optimization Toolbox.
toolbox\robust           — Robust Control Toolbox.
toolbox\ident            — System Identification Toolbox.
toolbox\control          — Control System Toolbox.
control\obsolete         — (No table of contents file)
toolbox\rtw              — Real-Time Workshop
bin\nt                   — (No table of contents file)
stateflow\stateflow      — Stateflow
stateflow\sfdemos        — (No table of contents file)
simulink\simulink        — Simulink
simulink\blocks          — Simulink block library.
simulink\simdemos        — Simulink demonstrations and samples.
simulink\dee             — Differential Equation Editor
toolbox\tour             — MATLAB Tour
toolbox\local            — Preferences.
For more help on directory/topic, type "help topic".
```

Этот внушительный список дает наглядное представление о пакетах прикладных программ, расширяющих возможности системы MATLAB и содержащих массу серьезных примеров применения системы.

1.7.2. Справка по конкретной функции или команде — команда help

Для получения справки по какому-либо конкретному объекту используется команда:

```
» help имя
```

где имя — имя объекта, для которого требуется вывод справочной информации. Мы уже приводили пример помощи по разделу операторов **ops**. Ниже дается пример для функции вычисления гиперболического синуса, намеренно введенного с неверным указанием имени:

```
» help hsin
hsin.m not found.
```

Нетрудно заметить, что система помощи сообщает, что для функции с именем hsin соответствующий М-файл отсутствует. Если ввести имя верно

```
» help sinh
  SINH   Hyperbolic sine.
    SINH(X) is the hyperbolic sine of the elements of X.
  Overloaded methods
    help sym/sinh.m
```

то полученное сообщение будет содержать информацию о функции **sinh**. Хотя имя функции в MATLAB задается малыми (строчными) буквами, в сообщениях системы помощи имена функций и команд выделяются большими (прописными) буквами. Этот не слишком удачный прием использован для выделения заголовка текста справки в виде имени функции. В данной книге мы отказались от такого приема, вводящего начинающих пользователей в заблуждение.

Аналогичным образом можно получить справку по константам и другим объектам языка MATLAB. Ниже дан пример обращения к справке о числе "пи":

```
» help pi
  PI    3.1415926535897....
    PI = 4*atan(1) = imag(log(-1)) = 3.1415926535897....
```

При всей примитивности такой справки надо отметить ее высокую эффективность. Особенно популярна интерактивная справка у пользователей, привыкших к работе с

языками программирования, которые используются в среде операционной системы MS-DOS.

1.7.3. Справка по определенной группе объектов

Пользователя системы MATLAB часто интересует набор функций, команд или иных объектов, относящихся к определенной группе объектов. В разделе 1.2.3 указаны имена основных групп объектов системы MATLAB. Ниже дан пример вызова справки по группе объектов **timefun**:

```
» help timefun
 Time and dates.
 Current date and time.
   now      — Current date and time as date number.
   date     — Current date as date string.
   clock    — Current date and time as date vector.
 Basic functions.
   datenum  — Serial date number.
   datestr  — String representation of date.
   datevec  — Date components.
 Date functions.
   calendar — Calendar.
   weekday  — Day of week.
   eomday   — End of month.
   datetick — Date formatted tick labels.
 Timing functions.
   cputime  — CPU time in seconds.
   tic, toc — Stopwatch timer.
   etime    — Elapsed time.
   pause    — Wait in seconds.
```

После уточнения состава определенной группы объектов можно получить детальную справку по любому выбранному объекту. Как это делается, было описано выше.

1.7.4. Справка по ключевому слову — команда lookfor

Ввиду обилия М-функций в системе MATLAB, число которых около 800, важное значение имеет поиск М-функций по ключевым словам. Для этого служит команда

lookfor Ключевое слово

или

lookfor 'Ключевые слова'

В первом случае ищутся все М-файлы, в заголовках которых встречается заданное ключевое слово, а заголовки обнаруженных файлов выводятся на экран. Следует отметить, что широкий поиск по одному ключевому слову может привести к выводу подчас многих десятков определений и длится довольно долго.

Для уточнения и сокращения поиска следует использовать вторую форму команды **lookfor**. Вот пример ее применения:

```
» lookfor 'inverse sin'
ASIN  Inverse sine.
ASIN  Symbolic inverse sine.
```

В данном случае для поиска использованы слова 'inverse sin', то есть задан поиск арксинуса. Система поиска нашла только два вида арксинуса ASIN — обычного и в символьной форме. Число найденных определений зависит от того, с каким числом пакетов прикладных программ (расширений) поставляется версия системы MATLAB.

В следующей главе мы рассмотрим гораздо более эффектные средства справочной системы, ориентированные на работу в стиле приложений операционных систем Windows 95/98 с графическим пользовательским интерфейсом.

1.7.5. Некоторые дополнительные справочные команды

В командном режиме можно получить справочные данные с помощью ряда команд:

computer — выводит сообщение о типе компьютера, на котором установлена текущая версия MATLAB.

script — выводит сообщение о назначении М-файлов сценариев (Script-файлов).

function — выводит сообщение о назначении и структуре М-файлов — функций.

info — выводит информацию о применяемых вычислительных системах и наборе компонентов системы MATLAB.

subscribe — позволяет создать файл с бланком регистрации.

ver — выводит информацию об установленной версии системы MATLAB и ее компонентах.

version — выводит краткую информацию об установленной версии MATLAB.

what — выводит имена файлов текущего каталога.

what name — выводит имена файлов заданного именем name каталога.

whatsnew name — выводит на экран содержимое справки заданного именем name класса для знакомства с последними изменениями в системе и в пакетах прикладных программ.

which name — выводит данные о функции с данным именем и пути доступа к ней.

Как правило, эти команды выводят довольно обширные сообщения. Ниже показаны примеры применения отдельных команд этой группы:

```
» computer
ans =
PCWIN
» version
ans =
5.2.1.1420
» ver
---------------------
MATLAB Version 5.2.1.1420 on PCWIN
MATLAB License Identification Number: 0
---------------------
MATLAB Toolbox                              Version 5.2    18-Dec-1997
Database Toolbox                            Version 1.0    21-Oct-1997
 Database Toolbox Demonstration Functions   Version 1.0    21-Oct-1997
Power System Blockset                       Version 1.0    21-Nov-1997
MATLAB Compiler                             Version 1.2    05-Dec-1997
Communications Toolbox                      Version 1.3    01-Dec-1997
Symbolic Math Toolbox                       Version 2.0.1  21-Nov-1997
.....
» which sin
sin is a built-in function.
» which hsin
hsin not found.
» which sinh
C:\MATLAB\toolbox\matlab\elfun\sinh.m
```

В дальнейшем мы рассмотрим и другие команды, которые могут быть отнесены к этой группе.

1.8. Сохранение и считывание данных сессии

1.8.1. Сохранение рабочей области сессии — команда save

Переменные и определения новых функций в системе MATLAB хранятся в особой области памяти, именуемой **рабочей областью**. MATLAB позволяет сохранить значения всех переменных и определений в сессии, т.е. рабочей области, 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 и расширением .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 стандарте версии MATLAB.
-append — добавление в существующий MAT-файл.

Возможно задание записи в формате функции, например:

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

причем в этом случае имена файлов и переменных задаются строковыми константами.

Следует отметить, что возможности сохранения всего текста сессии, формируемой в командном режиме, с помощью команды **save** нет. 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.

Тем не менее это не значит, что вы не имеете возможности записать то рациональное зерно, что родилось в ходе реализации ваших алгоритмов и методов решения задач. Надо просто воспользоваться редактором и отладчиком, который позволяет (после отладки текста документа) получить документ в корректной форме без синтаксических и иных ошибок. Такой документ сохраняется в текстовом формате в виде файла с расширением .m.

1.8.2. Ведение дневника — команда diary

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 в виде текстового файла с указанным именем всех команд в строках ввода и полученных результатов.
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 функ-

ции **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, т.е. 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 записи в качестве программы или ее модуля и содержащая ряд ошибочных с позиций синтаксиса языка программирования MATLAB записей, 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, иным чем .m, например, .txt. Это позволит встраивать подобные текстовые файлы дневника сессии в документы, содержащие ее описание.

1.8.3. Загрузка рабочей области сессии — команда load

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 в форме функции.

Если команда (или функция) **load** используется в ходе проведения сессии, то произойдет замена значений текущих 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, save demo*.mat означает загрузку всех файлов с началом имени demo, например: demo1, demo2, demoa, demob и так далее. Имена загружаемых файлов можно формировать с помощью операций над строковыми выражениями.

1.9. Работа с демонстрационными примерами

1.9.1. Вызов списка демонстрационных примеров — demos

Одним из самых эффективных методов знакомства со сложными математическими системами является ознакомление со встроенными примерами их применения. Система MATLAB содержит многие сотни таких примеров — практически на каждый оператор или функцию. Наиболее поучительные примеры можно найти в разделе demos, исполнив команду

```
» help demos
```

Ниже представлен выводимый этой командой список примеров:

```
Examples and demonstrations.
  Type 'demo' at the command line to browse more demos of
  MATLAB, the Toolboxes, and SIMULINK.

MATLAB/Introduction.
  demo       — Browse demos for MATLAB, Toolboxes, and SIMULINK

MATLAB/Matrices.
  intro      — Introduction to basic matrix operations in MATLAB.
  inverter   — Demonstrate the inversion of a matrix.
  Buckydem   — Connectivity graph of the Buckminster Fuller geodesic dome.
  sparsity   — Demonstrate effect of sparsity orderings.
  matmanip   — Introduction to matrix manipulation.
  eigmovie   — Symmetric eigenvalue movie.
  rrefmovie  — Computation of Reduced Row Echelon Form.
  delsqdemo  — Finite difference Laplacian on various domains.
  sepdemo    — Separators for a finite element mesh.
  airfoil    — Display sparse matrix from NASA airfoil.
  eigshow    — Graphical demonstration of matrix eigenvalues.
  svdshow    — Graphical demonstration of matrix singular values.

MATLAB/Numerics.
  Funfuns    — Demonstrate functions that operate on other functions.
  fitdemo    — Nonlinear curve fit with simplex algorithm.
  sunspots   — FFT: the answer is 11.08, what is the question?
  e2pi       — 2D visual solutions: Which is greater, e^pi or pi^e?
  bench      — MATLAB Benchmark.
  fftdemo    — Use of the fast finite Fourier transform.
  census     — Try to predict the US population in the year 2000.
  spline2d   — Demonstrate GINPUT and SPLINE in two dimensions.
  lotkademo  — An example of ordinary differential equation solution.
  quaddemo   — Adaptive quadrature.
```

zerodemo — Zerofinding with fzero.
fplotdemo — Plot a function.
quake — Loma Prieta Earthquake.

MATLAB/Visualization.
graf2d — 2D Plots: Demonstrate XY plots in MATLAB.
graf2d2 — 3D Plots: Demonstrate XYZ plots in MATLAB.
grafcplx — Demonstrate complex function plots in MATLAB.
lorenz — Plot the orbit around the Lorenz chaotic attractor.
imageext — Image colormaps: changing and rotating colormaps.
xpklein — Klein bottle demo.
vibes — Vibration movie: Vibrating L-shaped membrane.
Xpsound — Visualizing sound: Demonstrate MATLAB's sound capability.
Imagedemo — Demonstrate MATLAB's image capability.
penny — Several views of the penny data.
earthmap — View Earth's topography.
xfourier — Graphic demo of Fourier series expansion.
colormenu — Select color map.
cplxdemo — Maps of functions of a complex variable.

MATLAB/Language.
xplang — Introduction to the MATLAB language.
hndlgraf — Demonstrate Handle Graphics for line plots.
graf3d — Demonstrate Handle Graphics for surface plots.
hndlaxis — Demonstrate Handle Graphics for axes.

MATLAB/ODE Suite.
odedemo — Demo for the ODE suite integrators.
a2ode — Stiff problem, linear with real eigenvalues (A2 of EHL).
a3ode — Stiff problem, linear with real eigenvalues (A3 of EHL).
b5ode — Stiff problem, linear with complex eigenvalues (B5 of EHL).
ballode — Equations of motion for a bouncing ball used by BALLDEMO.
besslode — Bessel's equation of order 0 used by BESSLDEMO.
brussode — Stiff problem modelling a chemical reaction (Brusselator).
buiode — Stiff problem with analytical solution due to Bui.
chm6ode — Stiff problem CHM6 from Enright and Hull.
chm7ode — Stiff problem CHM7 from Enright and Hull.
chm9ode — Stiff problem CHM9 from Enright and Hull.
d1ode — Stiff problem, nonlinear with real eigenvalues (D1 of EHL).
fem1ode — Stiff problem with a time-dependent mass matrix.
fem2ode — Stiff problem with a time-independent mass matrix.
gearode — Stiff problem due to Gear as quoted by van der Houwen.
hb1ode — Stiff problem 1 of Hindmarsh and Byrne.
hb2ode — Stiff problem 2 of Hindmarsh and Byrne.
hb3ode — Stiff problem 3 of Hindmarsh and Byrne.
orbitode — Restricted 3 body problem used by ORBITDEMO.

orbt2ode — Non-stiff problem D5 of Hull et al.
rigidode — Euler equations of a rigid body without external forces.
sticode — A spring-driven mass stuck to surface, used by STICDEMO.
vdpode — Parameterizable van der Pol equation (stiff for large mu).

Extras/Gallery.
knot — Tube surrounding a three-dimensional knot.
quivdemo — Demonstrate the quiver function.
klein1 — Construct a Klein bottle.
cruller — Construct cruller.
tori4 — Hoops: Construct four linked tori.
spharm2 — Construct spherical surface harmonic.
modes — Plot 12 modes of the L-shaped membrane.
logo — Display the MATLAB L-shaped membrane logo.

Extras/Games.
xpbombs — Minesweeper game.
life — Conway's Game of Life.
bblwrap — Bubblewrap.
soma — Soma cube

Extras/Miscellaneous.
truss — Animation of a bending bridge truss.
travel — Traveling salesman problem.
spinner — Colorful lines spinning through space.
xpquad — Superquadrics plotting demonstration.
codec — Alphabet transposition coder/decoder.
xphide — Visual perception of objects in motion.
makevase — Generate and plot a surface of revolution.
wrldtrv — Great circle flight routes around the globe.
logospin — Movie of The MathWorks' logo spinning.
crulspin — Spinning cruller movie.
quatdemo — Quaternion rotation.

General Demo/Helper functions.
cmdlnwin — An Demo gateway routine for playing command line demos.
cmdlnbgn — Set up for command line demos.
cmdlnend — clean up after command line demos.
finddemo — Finds demos available for individual toolboxes.
helpfun — Utility function for displaying help text conveniently.
pltmat — Display a matrix in a figure window.

MATLAB/Helper functions.
bucky — The graph of the Buckminster Fuller geodesic dome.
peaks — A sample function of two variables.
membrane — Generate MathWorks's logo.

See also SIMDEMOS

demos is both a directory and a function.

DEMOS Demo list for the Power System Blockset

Мы настоятельно рекомендуем пользователям системой MATLAB просмотреть с десяток примеров из интересующих их областей. Это займет от силы полчаса и даже меньше, но зато позволит оценить поистине неисчерпаемые возможности системы в решении сложных математических и физических задач, а также превосходные средства графической визуализации решений.

1.9.2. Тест на быстродействие ПК — bench

Большинство пользователей ПК, включая авторов данной книги, очень ревниво относится к вычислительной мощности своего компьютера. Поэтому в качестве первого демонстрационного примера возьмем тест на сравнительную оценку ПК пользователя — **bench**. Исполнив команду

» bench

можно наблюдать исполнение комплексного теста по оценке быстродействия ПК. Его итоги представляются в виде гистограммы и таблицы, которые показаны на рис. 1.16.

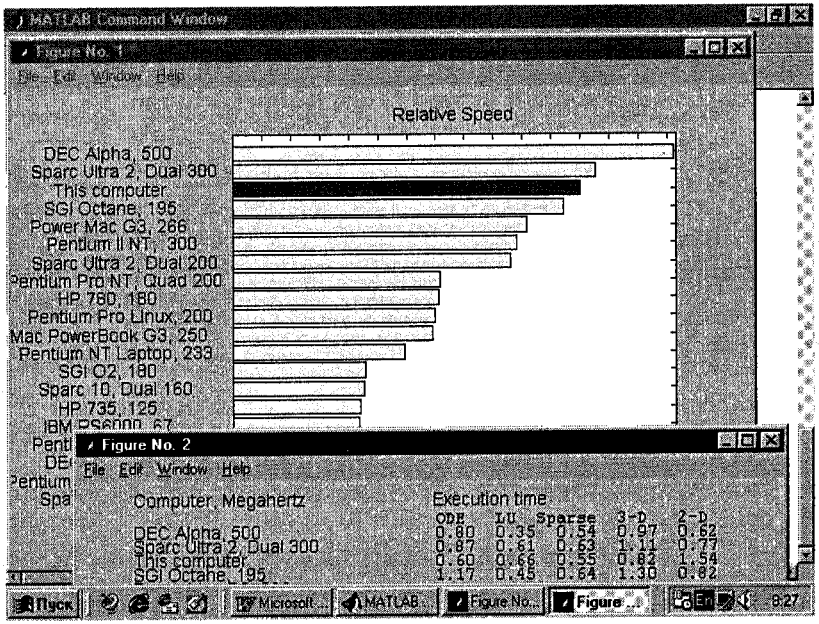


Рис. 1.16. Результаты тестирования ПК на быстродействие

Не знаем как вы, читатель, но один из авторов данной книги остался весьма довольным своим ПК с процессором Pentium II 350 МГц и 100-Мегагерцовой шиной — его ПК оказался на почетном третьем месте, уступив лишь таким монстрам, как компьютеру с процессором Alpha и двухпроцессорному серверу Sparc Ultra 2, Dual 300. Это говорит о том, что вычислительная мощность современного ПК с системой MATLAB уже приближается к таковой для суперкомпьютеров недавнего прошлого. Новые суперкомпьютеры, разумеется, куда мощнее даже ПК с процессором Pentium III, но и для них MATLAB — одна из основных систем для выполнения математических расчетов.

1.9.3. Что больше: e^pi или pi^e?

Рассмотрим еще один простой пример — ответ на вопрос, какое из значениий e^pi или pi^e больше? Для запуска этого примера надо исполнить команду

```
» e2pi
```

и наблюдать красочное шоу — графики степенных функций x^y и y^x с построением на них линий заданных функций и оценкой их значений. Заключительный слайд этого примера показан на рис. 1.17 .

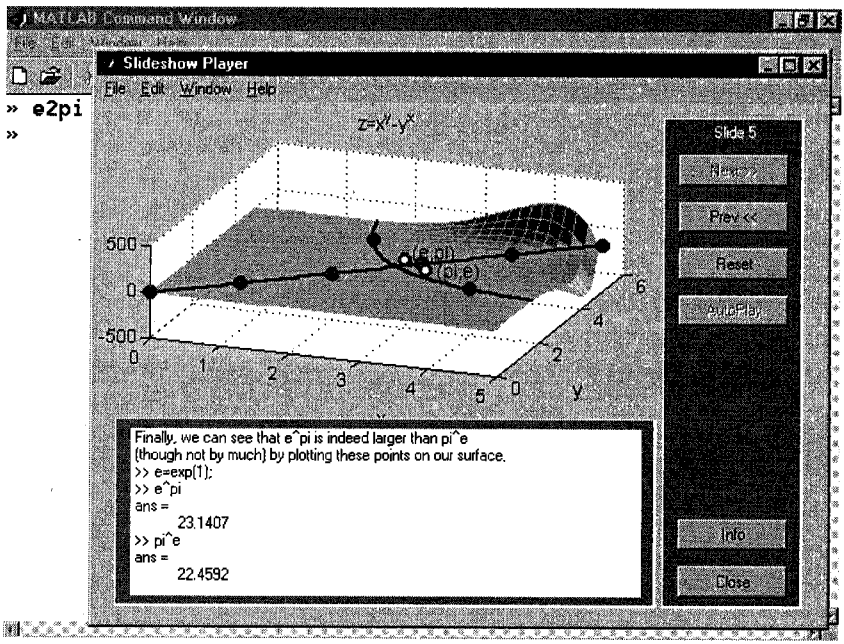


Рис. 1.17 . Заключительный слайд примера e2pi

Этот пример — наглядная демонстрация перехода от узких понятий к более широким. Разумеется, вы могли бы вместо приятного обозрения слайд-шоу просто вычислить:

```
» e=exp(1)
e =
   2.7183
» e^pi
ans =
  23.1407
» pi^e
ans =
  22.4592
```

И убедиться в том, что все же e^pi больше, чем pi^e. Но тогда это означало бы, что вы просто технарь или физик-экспериментатор, а не истинный математик. Впрочем, у каждого есть свои взгляды на применение математики. И чьи лучше — вопрос весьма спорный.

1.9.4. Великолепие трехмерной анимации

Современная трехмерная графика — одна из причин большой популярности системы MATLAB. В этом разделе мы не будем рассматривать конкретные реализации тех или иных видов трехмерной графики — ниже будет показано, как это сделать с помощью команды **type**, выводящей на экран дисплея текст (листинг) указанного файла. Ограничимся лишь тремя примерами визуализации сложных математических задач, когда используется оживление изображений — **анимация**.

На рис. 1.18 показан пример визуализации динамического процесса в так называемом аттракторе Лоренца (пример **lorenz**) — колебательной системе, создающей хаотические и довольно замысловатые колебания. Наиболее наглядна их визуализация с помощью трехмерного фазового портрета колебаний, который приведен на рис. 1.18. К сожалению, показана лишь завершающая стадия — на практике можно реально видеть движение образующей точки во времени и убедиться в своеобразной хаотичности колебаний. Для запуска анимации надо нажать кнопку **Start** окна графики.

Эмблемой системы MATLAB стало изображение цветной поверхности мембраны. В демонстрационном примере с именем **membrane** не только строится эта поверхность, но и задается эффект изменяющейся во времени окраски, которая распространяется от вершины фигуры вниз — рис. 1.19.

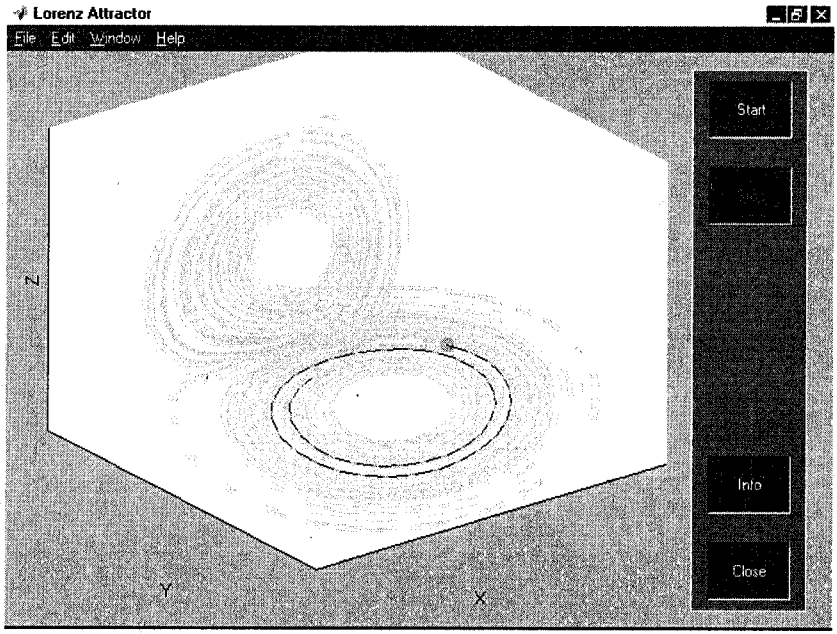


Рис. 1.18. График, иллюстрирующий работу аттрактора Лоренца

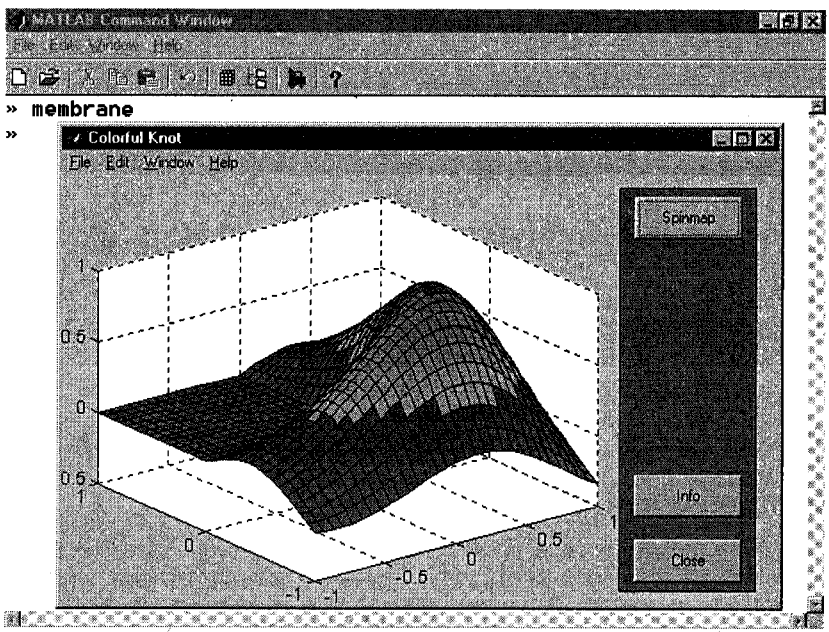


Рис. 1.19. Эмблема системы MATLAB — мембрана

Еще один пример **knot** показывает построение сложной пространственной фигуры с функциональной окраской — рис. 1.20. При пуске этого примера нажатием кнопки Spinmap можно наблюдать изменение окраски, имитирующее как бы движение жидкости внутри замкнутой трубки, образующей данную фигуру.

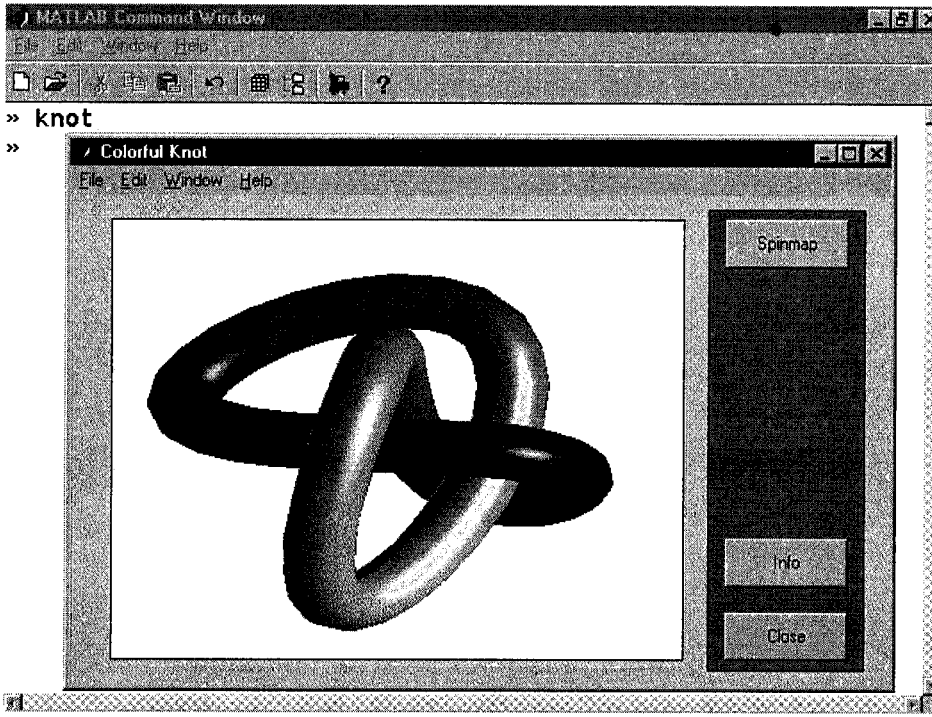


Рис. 1.20. Построение фигуры **knot**

Таким образом, можно сделать вывод о том, что система MATLAB предоставляет возможности графики, используемые для имитации и моделирования математических и физических задач — от простейших графиков функций в Декартовой системе координат до сложных анимационных графиков с динамической цветной функциональной окраской. Среди множества примеров такой графики всегда можно подобрать наиболее подходящие для решения конкретных задач пользователя в какой бы области науки, техники или образования он ни работал.

1.9.5. В паутине нейронных сетей

Даже десятка таких книг, как эта, едва ли хватит для исчерпывающего описания системы MATLAB со всеми ее пакетами прикладных программ. Многие из таких пакетов, например, по нейронным сетям, сплайнам, обработке сигналов, проектированию систем управления и так далее, относятся к самым современным и актуальным направлениям науки и техники. Нередко создание таких пакетов для системы

MATLAB возглавляют создатели указанных научных направлений, и по каждому такому направлению опубликованы десятки научных монографий.

Примером одного из таких направлений является пакет Neural Networks — **нейронные сети**. Эти сети основаны на аналогии с ячейкой нашего мозга — **нейроном**. Важное свойство нейрона — его возможность к самообучению и распознаванию различных образных представлений и сигналов.

В разделе справочной системы **Examples and Demos** имеется множество конкретных примеров применения нейронных сетей. На рис. 1.21 показан вид демонстрационной панели одного из этих примеров — идентификация линейных систем, представленных в виде нейронных сетей.

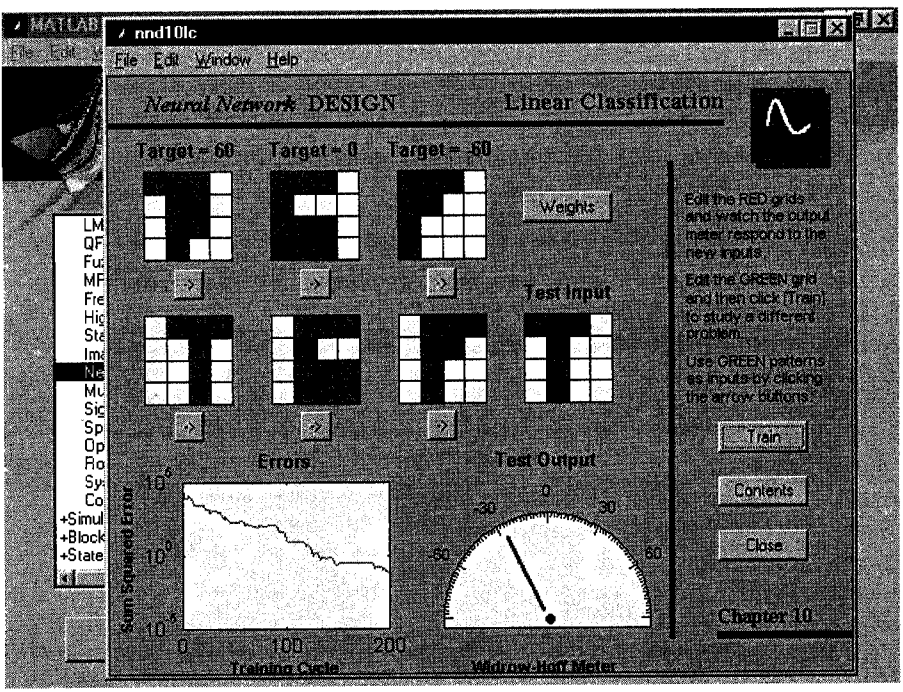


Рис. 1.21. Демонстрационная панель примера на идентификацию линейных систем, представленных в виде нейронной сети

Вы имеете возможность задавать различные параметры нейронной сети, представляющей некоторую линейную систему, и оценивать последнюю на предмет ее соответствия некоторому образцу. Демонстрационная панель построена в виде виртуальной физико-биологической лаборатории.

К сожалению, даже мало-мальски полное описание нейронных сетей не входит в задачи этой книги, ибо относительно системы MATLAB такие сети лишь одно, причем, далеко не самое массовое ее применение. Этот пример говорит о том, что весьма

желателен выпуск книг не только по самой системе MATLAB, но и по отдельным направлениям ее применения и по отдельным пакетам прикладных программ.

1.9.6. Вывод на экран текстов-примеров и М-файлов — команда type

Хотя само по себе наблюдение за тем, как MATLAB расправляется со сложными примерами и задачами довольно поучительно, жаждущие применить систему на практике пользователи, безусловно захотят узнать, а как же конкретно реализовано решение той или иной задачи ? Для этого вам достаточно просмотреть соответствующий демонстрационный (или любой другой) М-файл. Это можно сделать с помощью любого текстового редактора, редактора и отладчика М-файлов, встроенного в систему (он описан в главе 2) или с помощью команды

type Имя_М-файла

Ниже представлена часть файла демонстрационного примера **e2pi**:

```
» type e2pi
function slide=e2pi
% This is a slideshow file for use with playshow.m and makeshow.m
% To see it run, type 'playshow e2pi',
%   Copyright (c) 1984-98 by The MathWorks, Inc.
%   $Revision: 5.10 $

if nargout<1,
  playshow e2pi
else
  %========== Slide 1 ==========
  slide(1).code={
   'x=0:0.16:5;',
   'y=0:0.16:5;',
   '[xx,yy]=meshgrid(x,y);',
   'zz=xx.^yy-yy.^xx;',
   'h=surf(x,y,zz);',
   '   set(h,"EdgeColor",[0.7 0.7 0.7]);',
   '   view(20,50);',
   '   colormap(hsv);' };
  slide(1).text={
   ' Press the "Start" button to see an example of visualization',
   ' in MATLAB applied to the question:',
   '',
   ' "Which is greater, e^pi or pi^e?"'};
.....
end
```

Используя команду **help**, можно получить справку по любой конкретной функции или команде. Ввиду того, что текст примера имеет довольно большой объем, мы ограничились приведением только первого слайда этого примера. Остальные слайды просто опущены, и на их месте стоит многоточие.

1.10. Завершение работы с системой

Для завершения работы с системой можно использовать команды **quit**, **exit** или комбинацию клавиш Ctrl и Z. 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.

Глава 2.
Пользовательский интерфейс MATLAB

2.1. Общая характеристика пользовательского интерфейса

Как видно из материалов первой главы, в новых версиях MATLAB в полной мере сохранен командный интерактивный режим работы. Это — старый фасад дворца с именем MATLAB. Командный режим остается одним из наиболее удобных и апробированных методов работы с системой [2,3,50] и сохранен вплоть до последней версии MATLAB 5.3.

Имеются и типовые средства Windows 95/98 — главное меню с выпадающими подменю и панель инструментов. Но они выглядят намного скромнее, чем у большинства современных приложений Windows. Видимо, так и должно быть — чем серьезнее математическая система, тем меньше она нуждается в использовании всевозможных кнопок в панели инструментов, и тем скромнее может быть ее главное меню.

Для редактирования и отладки M-файлов MATLAB 5.* имеет встроенный современный редактор, интерфейс которого выполнен в лучших традициях Windows-приложений.

В том же стиле выполнено окно просмотра ресурсов памяти, окно просмотра путей файловой системы, справочник по возможностям системы и демонстрационные программы.

2.2. Панель инструментов

2.2.1. Назначение кнопок панели инструментов

На рис. 2.1 приведена часть окна системы MATLAB, содержащая главное меню и панель инструментов основного окна системы.

Рис. 2.1. Часть окна системы MATLAB с главным меню и панелью инструментов

Прежде всего перечислим назначение всех кнопок панели инструментов:

1. **New file** — выводит пустое окно редактора М-файлов;
2. **Open file** — открывает окно для загрузки М-файла;
3. **Cut** — вырезает выделенный фрагмент и помещает его в буфер;
4. **Copy** — копирует выделенный фрагмент в буфер;
5. **Paste** — переносит фрагмент из буфера в текущую строку ввода;
6. **Undo** — отменяет предшествующую операцию;
7. **Workspace Browser** — выводит окно просмотра ресурсов рабочего места;
8. **Path Browser** — выводит окно просмотра файловой структуры;
9. **New Simulink Model** — открывает окно симулятора;
10. **Help Windows** — открывает Windows-окно справки по системе.

Набор кнопок панели инструментов обеспечивает выполнение наиболее часто встречаемых команд и вполне достаточен для повседневной работы с системой. О назначении кнопок говорят и всплывающие подсказки, появляющиеся, когда маркер мыши устанавливается на соответствующую кнопку. Они имеют вид желтого прямоугольника с текстом.

2.2.2. Окно открытия нового файла

Кнопка 1 (**New file**) открывает окно редактора и отладчика М-файлов. Это окно показано на рис. 2.2.

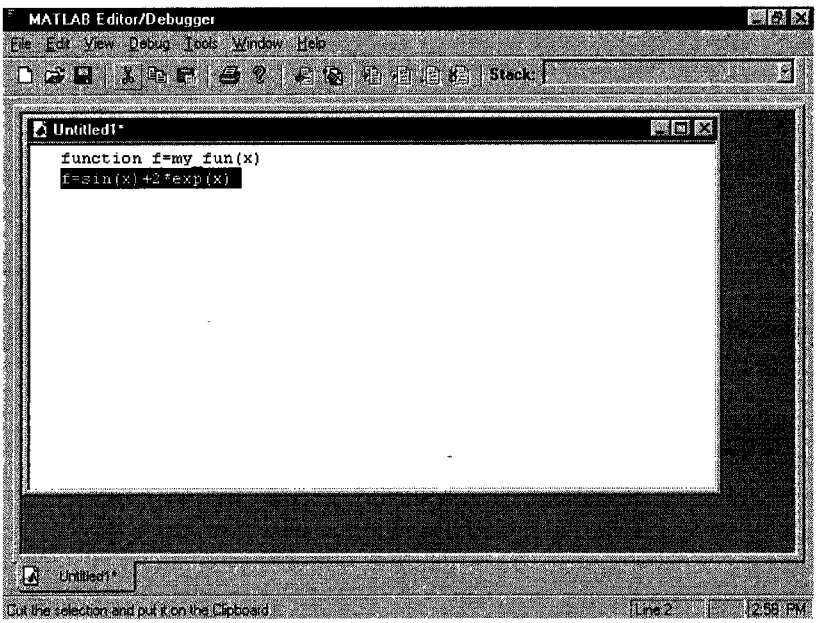


Рис. 2.2. Окно редактора и отладчика М- айлов

По умолчанию файлу дается имя Untitled (безымянный), которое впоследствии (при записи файла) можно изменить на другое, отражающее тему задачи.

На рис. 2.2 в окне редактирования нового файла приведен пример простого М-файла. Детали подготовки и отладки М-файлов мы обсудим позже. Пока же отметим, что по завершении ввода текста файла он сохраняется на диске под своим текущим именем с помощью кнопки **Save** панели инструментов редактора и отладчика М-файлов.

2.2.3. Окно загрузки имеющегося файла

Кнопка 2 (**Open file**) служит для загрузки в редактор и отладчик ранее созданных М-файлов, например, входящих в Toolbox-системы, или разработанных пользователем. Она открывает окно, которое является типичным элементом интерфейса Windows (показано на рис. 2.3.).

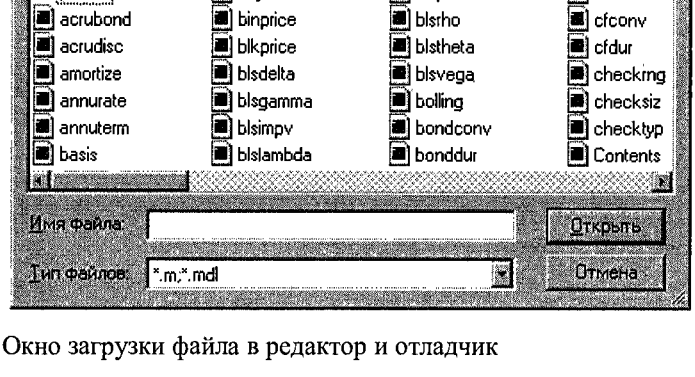


Рис. 2.3. Окно загрузки файла в редактор и отладчик

В этом окне можно просмотреть файловую систему ПК и выбрать нужный файл для загрузки.

2.2.4. Операции с буфером промежуточного хранения

Кнопки 3 (**Cut**), 4 (**Copy**) и 5 (**Paste**) реализуют наиболее характерные операции с буфером промежуточного хранения (Clipboard). Первые две операции относятся к выделенным фрагментам сессии или текста М-файлов (если они выполняются в окне редактора и отладчика М-файлов). Для выделения объектов можно использовать

мышь, перемещая курсор по тексту при нажатой левой кнопке, или клавиши стрелок в комбинации с клавишей Shift.

На рис. 2.4 показан пример выделения содержимого матрицы **M**. Эта матрица формируется функцией **magic(n)** и называется магической, поскольку сумма элементов любого столбца, любой строки и даже любой диагонали равна одному и тому же числу — 34 для матрицы при n=4. Пример выделения в окне редактора М-файлов показан на рис. 2.2.

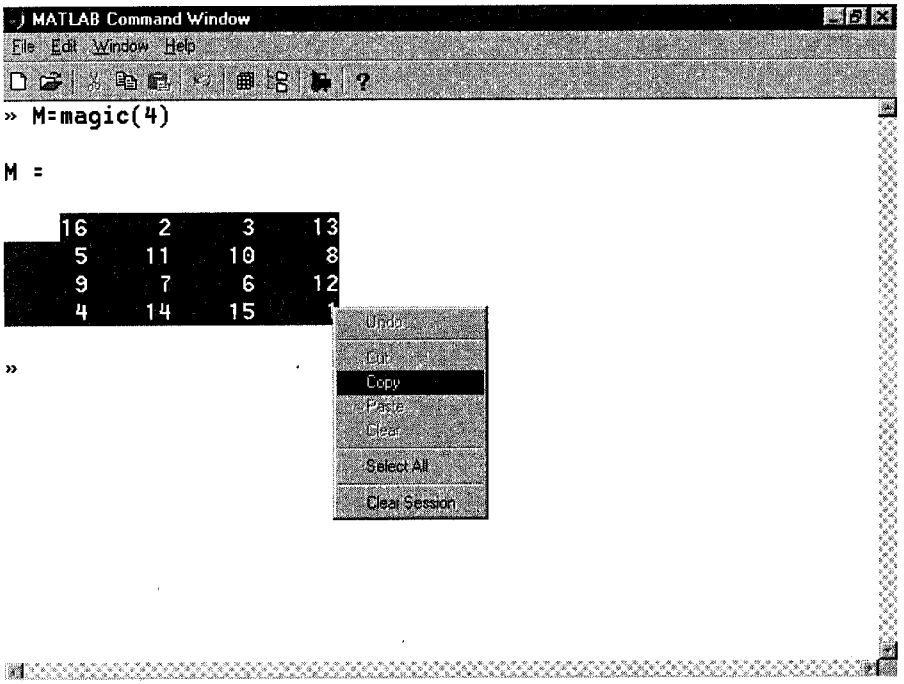


Рис. 2.4. Окно системы с выделенным содержимым матрицы **M**

Операция **Cut** осуществляет вырезку выделенного фрагмента и размещение его в буфере. При этом вырезанный фрагмент удаляется из текста документа. Операция **Copy** просто копирует выделенный фрагмент в буфер, сохраняя его в тексте.

В MATLAB можно использовать контекстно-зависимое меню, появляющееся при нажатии правой клавиши мыши. Например, установив курсор мыши на выделенный фрагмент матрицы **M** и нажав правую клавишу мыши, можно увидеть меню, показанное на рис. 2.4. В нем, кстати, дублируется позиция с командой **Copy**. Есть и ряд других, доступных в данный момент команд.

Содержимое буфера можно перенести в другое место сессии, на другую страницу или даже в другое приложение. Допустим, мы хотим создать матрицу **M1** с содержимым, которое размещено в буфере. Для этого достаточно набрать **M1=[**, затем испол-

нить команду **Paste** и ввести закрывающую квадратную скобку]. На рис. 2.5 показано, как при этом создается матрица **M1**, по содержимому аналогичная матрице **M**.

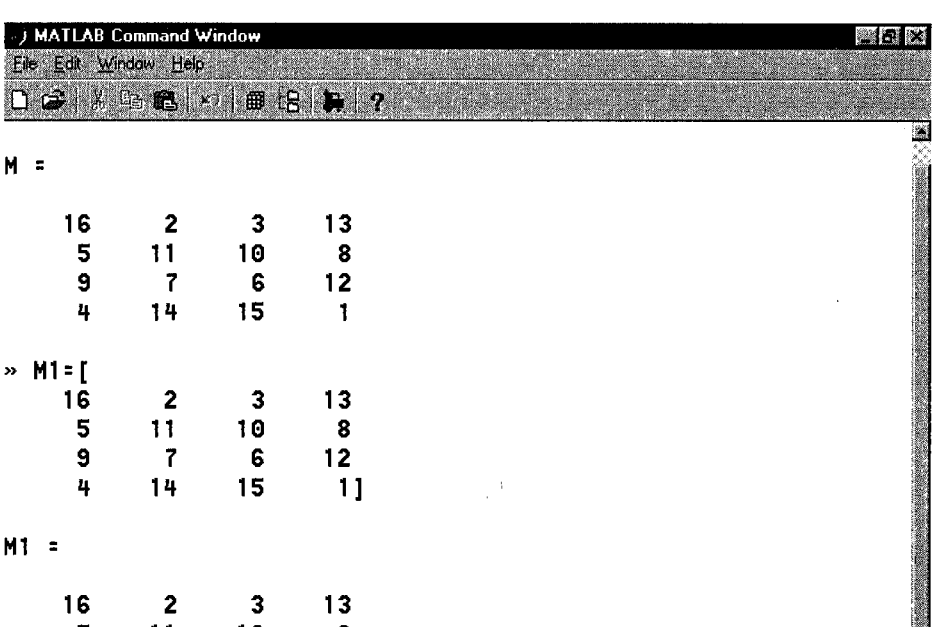


Рис. 2.5. Пример создания матрицы **M1** с содержимым, взятым из буфера

Разумеется, этот пример является чисто учебным. Не обращаясь к помощи буфера, можно было бы просто записать **M1=M**. Однако зачастую операции с буфером весьма полезны. Так, все примеры в тексте этой книги получены переносом выделенных фрагментов соответствующей сессии в окно текстового редактора Word.

Обратите внимание на команду **Select All** в контекстно-зависимом меню правой клавиши мыши. Эта команда позволяет выделить текст всей текущей сессии, а команда **Clear Session** — очистить окно от содержимого данной сессии.

2.2.5. Отмена результата предшествующей операции — команда Undo

Часто, выполнив какую-то операцию, мы отмечаем, что она оказалась ошибочной. При работе в MATLAB такой ситуации пугаться не стоит — нажатие кнопки 6 (**Undo**) приведет к отмене последнего действия, выполненного в текущей строке. Операции в завершенных строках документа этой командой не отменяются.

2.2.6. Браузер просмотра рабочего пространства

Векторы и матрицы могут занимать большой объем памяти. Конечно, речь не идет о векторах или матрицах, содержащих несколько элементов или несколько десятков элементов. Хотя и в этом случае оценка их размеров полезна при разработке алгоритмов матричных вычислений и оценке их эффективности в части использования памяти.

Кнопка 7 (**Workspace Browser**) выводит окно специального браузера для просмотра ресурсов рабочего пространства памяти. Рис. 2.6 иллюстрирует применение окна просмотра ресурсов при задании нескольких переменных различного типа. Браузер дает наглядную **визуализацию** содержимого рабочего пространства.

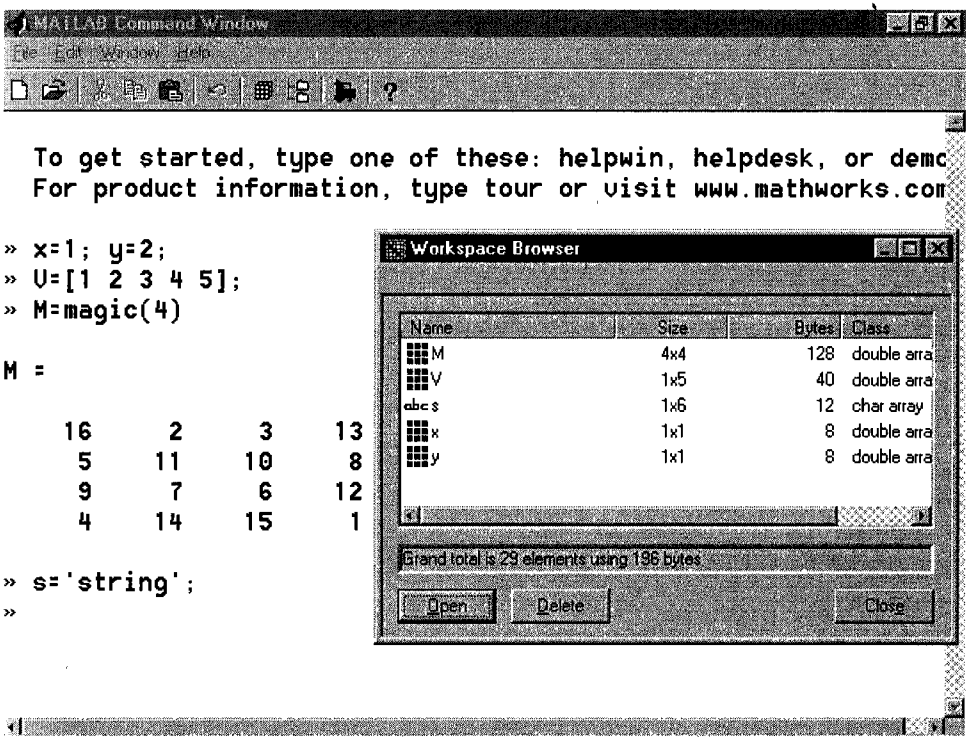


Рис. 2.6. Просмотр ресурсов памяти с помощью браузера рабочего пространства

Матрицы, несущие данные о сложных геометрических фигурах или представляющие цветные изображения с высоким разрешением, могут занимать в памяти объем, исчисляемый многими мегабайтами, и в этом случае оценка их размера становится необходимой.

Окно просмотра ресурсов выполняет и другие важные функции — позволяет просматривать существующие в памяти объекты, редактировать их содержимое и удалять объекты из памяти. Для вывода содержимого объекта достаточно выделить его имя с

помощью мыши и нажать клавишу окна **Open**. Объект можно открыть двойным щелчком левой клавишей мыши по кнопке объекта. Откроется окно, подобное показанному на рис. 2.7 применительно к матрице **M**.

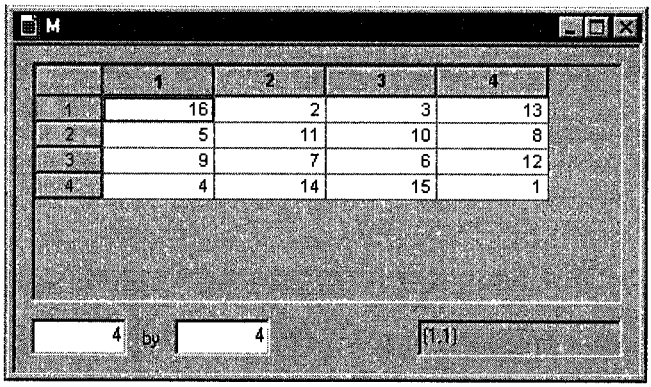


Рис. 2.7. Окно просмотра содержимого матрицы

Кнопка **Delete** уничтожает объект с заданным именем и устраняет его из памяти, а кнопка **Close** закрывает окно просмотра.

2.2.7. Команды просмотра рабочей области who и whos

Следует отметить, что просмотр рабочего пространства возможен и в командном режиме без обращения к браузеру Workspace Browser. Команда **who** выводит список определенных переменных, а команда **whos** — список переменных с указанием их размера и объема занимаемой памяти. Следующие примеры иллюстрируют действие этих команд:

```
» x=1.234;
» V=[1 2 3 4 5];
» M=magic(4);
» who

Your variables are:
M      V      x

» whos
  Name      Size          Bytes  Class
  M         4x4             128  double array
  V         1x5              40  double array
  x         1x1               8  double array

Grand total is 22 elements using 176 bytes
```

Если вы хотите просмотреть данные одной переменной, например **M**, следует использовать команду **whos M**. Естественно, что просмотр рабочего пространства с помощью браузера **Workspace Browser** более удобен и нагляден.

2.2.8. Браузер просмотра файловой структуры

Для просмотра файловой структуры MATLAB служит специальный браузер, который запускается с помощью кнопки **Path Browser**, при этом на экран выводится окно просмотра файловой структуры, показанное на рис. 2.8.

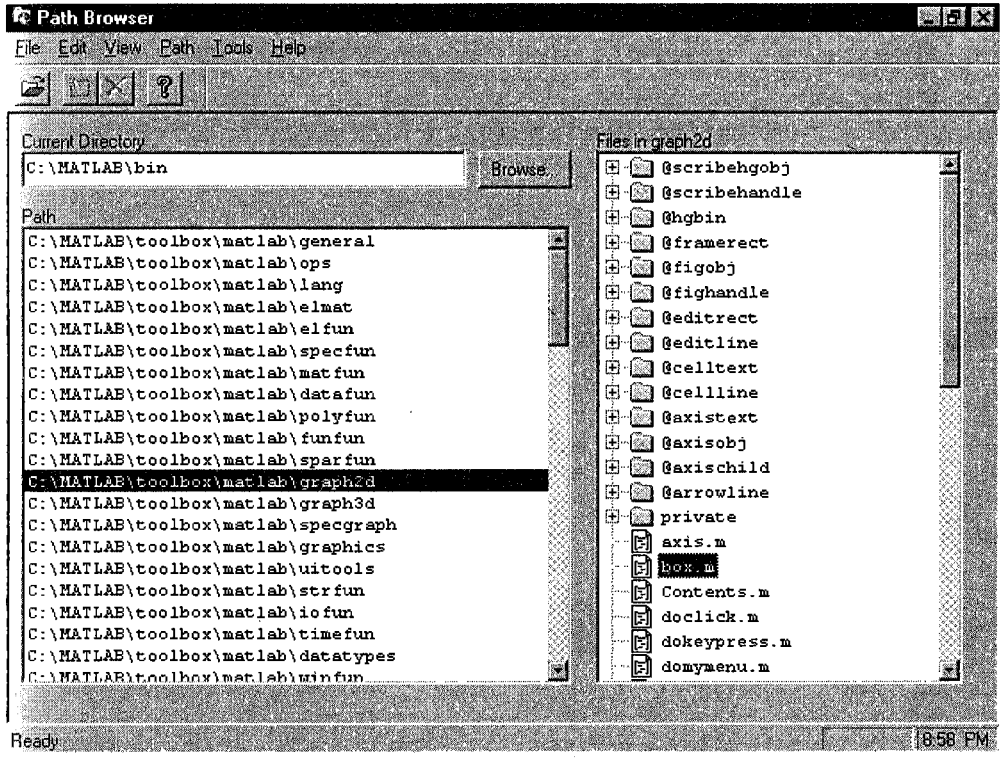


Рис. 2.8. Окно браузера просмотра файловой структуры

Окно разделено на две части. В левой расположен список папок и путей к ним, а в правой — список папок и файлов выделенного в левой части каталога. Например, на рис. 2.8 выделен каталог graph2d с папками и M-файлами двумерной графики. Двойной щелчок мыши на папке в правой части открывает ее, а двойной щелчок на файле запускает редактор и отладчик M-файлов с загруженным в него выделенным файлом — рис. 2.9 (файл box.m).

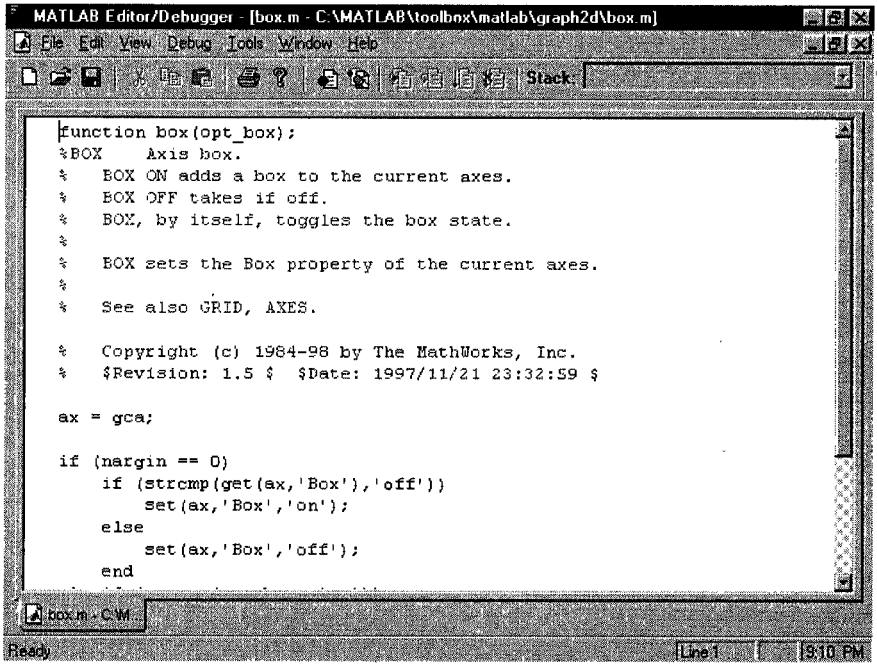


Рис. 2.9. Окно редактора и отладчика М-файлов с загруженным из браузера файлом

Кнопка **Browse...** позволяет вывести отдельное окно с содержимым выделенной папки. Такое окно показано на рис. 2.10. Окно представляет данные о файловой структуре в виде дерева папок и файлов. Каждую папку можно открыть, установив на ней курсор мышки и щелкнув ее левой клавишей. Открытая папка отображается со знаком минус, а закрытая со знаком плюс.

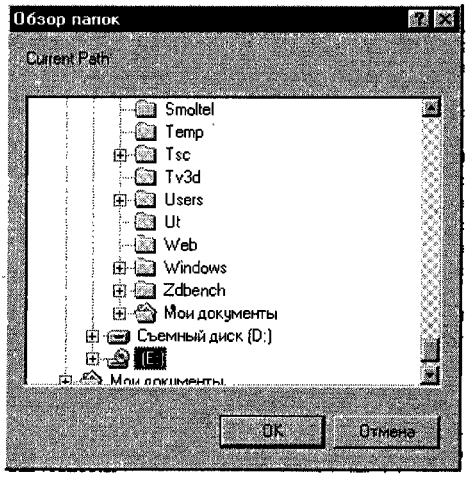


Рис. 2.10. Окно просмотра содержимого выделенной папки

Таким образом браузер просмотра файловой структуры позволяет детально ознакомиться с файловой системой MATLAB.

2.2.9. Запуск приложения Simulink

Кнопка **New Simulink Model** запускает одно из самых мощных приложений системы MATLAB — программу моделирования (симуляции) систем, построенных из типовых блоков. Эту систему (приложение Simulink) мы рассмотрим более подробно в дальнейшем (глава 10), а пока отметим лишь, что нажатие указанной кнопки выводит окно программы и окно выбора типов блоков. С его помощью можно вывести и дополнительные окна с различными типами блоков — рис. 2.11.

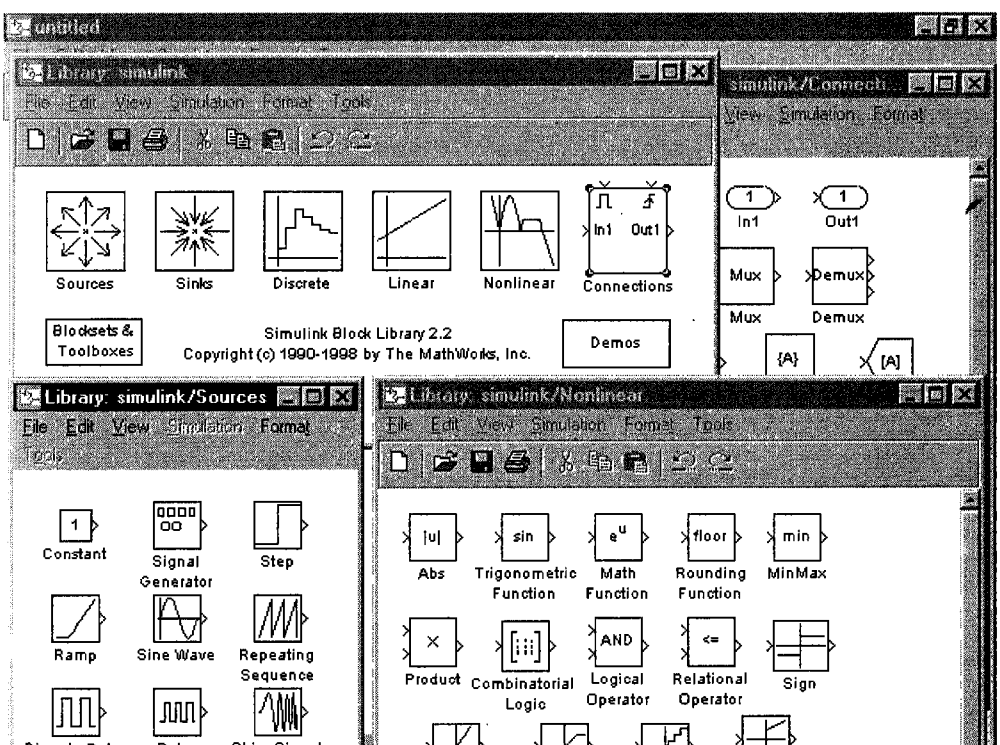


Рис. 2.11. Окно программы Simulink с окнами блоков

В MATLAB 5.2.1 применена новая модель Simulink 2 с библиотекой блоков Block Library 2.2. Она появилась в конце 1997 года и дает удобные и весьма разнообразные средства моделирования сложных систем. В марте 1999 года объявлено о выпуске на рынок новой версии Simulink 3.0, которой комплектуется новейшая версия MATLAB 5.3.

2.2.10. Справочная система MATLAB под Windows

Последняя кнопка 10 (**Help Windows**) панели инструментов открывает окно с перечнем разделов справочной системы. Это окно показано на рис. 2.12.

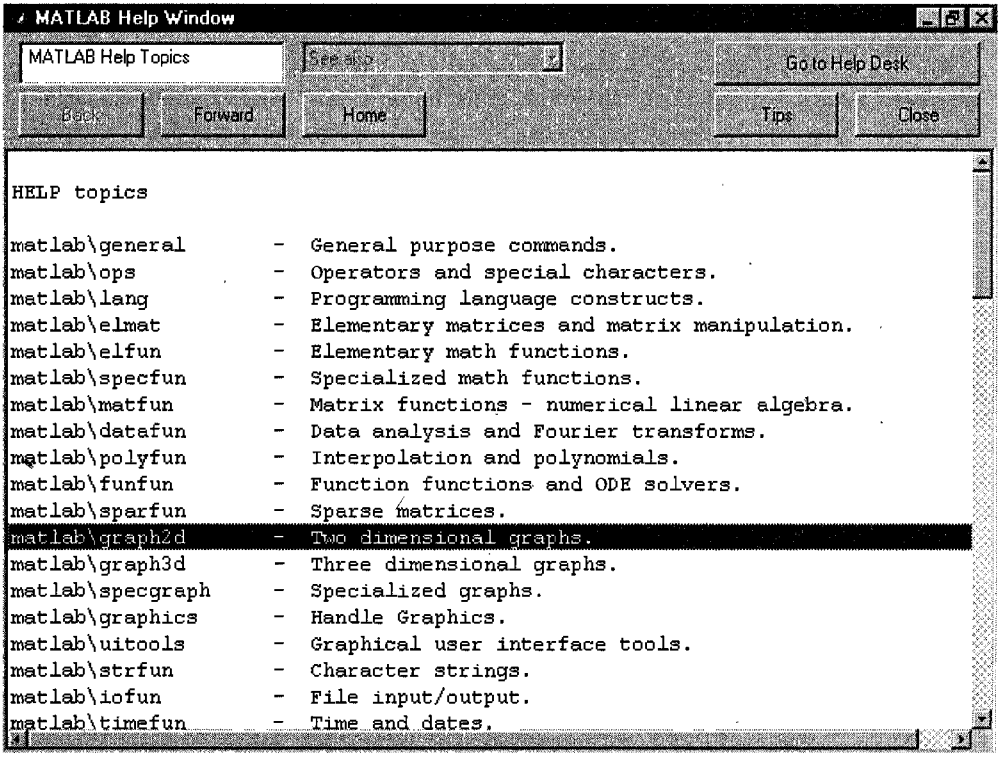


Рис. 2.12. Окно с перечнем разделов справочной системы

С помощью мыши вы можете выделить интересующий вас раздел справки (на рис. 2.12 — это раздел по двумерной графике). Двойной щелчок левой клавишей мыши открывает окно справки соответствующего раздела (см. рис. 2.13). Аналогичный результат дает нажатие кнопки **Tips** в левом верхнем углу окна справки. Кнопка **Close** закрывает окно справочной системы.

Справочный материал может состоять из нескольких разделов. Так, на рис. 2.14 показано окно справки команды **plot**, выделенной в списке команд двумерной графики, представленном на данном рисунке. С помощью клавиш **Back** и **Forward** в левом верхнем углу справочного окна можно перейти к предшествующему или следующему разделу. Клавиша **Home** возвращает к начальной странице.

При просмотре справки по отдельным командам обычно активизируется переключатель **See also** (смотри также). С помощью этого переключателя можно наблюдать список тем, близких к рассматриваемой теме.

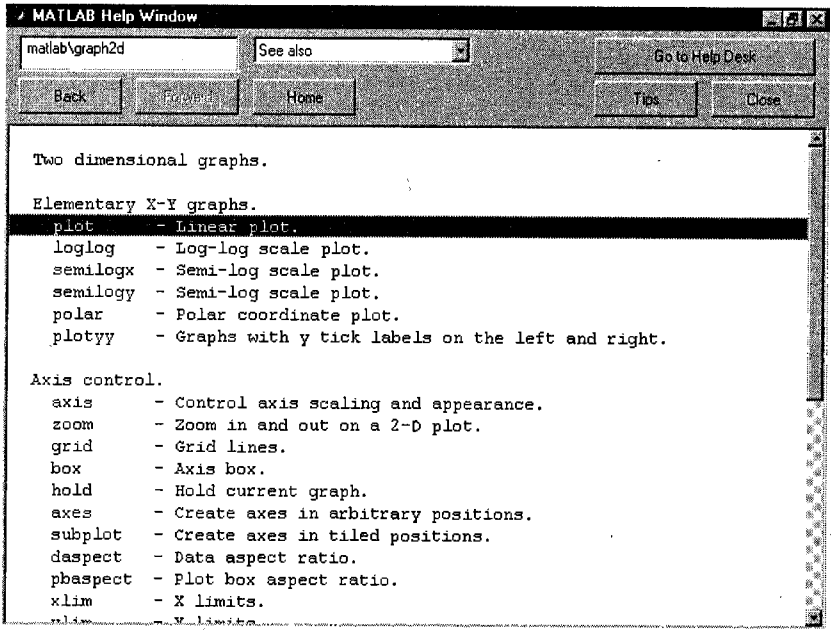


Рис. 2.13. Просмотр окна со справкой по двумерной графике

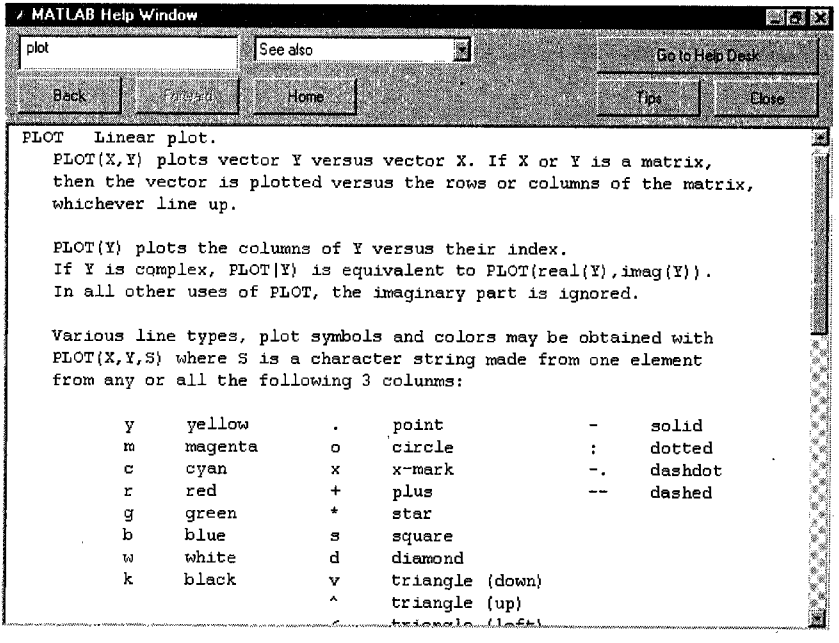


Рис. 2.14. Просмотр окна со справкой по команде **plot**

Клавиша **Go to Help Desk** открывает окно другого вида справки, страницы которой представлены в формате HTML. При этом для просмотра используется браузер (Обозреватель) Microsoft Internet Explorer. Этот, пожалуй самый мощный источник справочной информации будет рассмотрен отдельно. Пока отметим лишь, что такая справочная система может оперативно дополняться и обновляться через Internet.

2.3. Главное меню системы

2.3.1. Общее описание главного меню

Главное меню MATLAB (рис. 1.7) выглядит довольно скромно и содержит всего четыре позиции:

File — работа с файлами;
Edit — редактирование сессии;
Windows — установка Windows-свойств окна;
Help — доступ к справочным подсистемам.

Для открытия какой-либо позиции главного меню можно при нажатой клавише Alt нажать клавишу символа, подчеркнутого в названии позиции. Например, чтобы открыть позицию **Windows**, надо нажать клавишу W. Стандартный способ работы с мышью — это выбор пункта меню и его открытие левой кнопкой мыши.

2.3.2. Подменю, команды, операции и опции

Открытая позиция главного меню выглядит как **подменю**, содержащее различные операции и команды. Выделенная команда или операция исполняется при нажатии клавиши Enter. Выполнение команды можно также осуществить двойным щелчком мыши или нажатием на клавиатуре клавиши с выделенным символом в названии команды.

Между командами и операциями нет особых отличий и в литературе по информатике их часто путают. Мы будем считать **командой** действие, которое исполняется немедленно. А **операцией** — действие, которое требует соответствующей подготовки, например, открытия окна для установки определенных параметров или опций.

Опция — это значение определенного параметра, действующее во время сеанса текущей сессии. Опциями обычно являются указания на применяемые наборы шрифтов, размеры окна, цвет фона и так далее.

2.3.3. Работа с файлами — подменю File

Подменю **File** содержит внушительный список операций и команд. Оно показано на рис. 2.15.

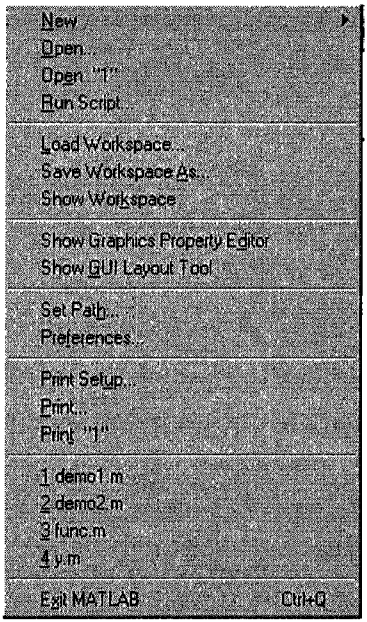


Рис. 2.15. Подменю файловых операций **File**

Эти операции и команды разбиты на несколько характерных групп.

Операции задания и загрузки файлов

Операция **New** (Новый) открывает подменю для создания трех типов документов:

M-file — M-файла;
Figure — фигуры (рисунка);
Model — модели системы Simulink.

Операция **Open** (Открыть файл) открывает окно загрузки файла. Оно уже описывалось выше — (см. рис. 2.3).

Своеобразно работает команда **Open Selection**. В нормальном состоянии системы она пассивна, поскольку это контекстно-зависимая команда. Она начинает действовать, если в сессии выделена какая-либо информация, например, оператор или функция. Так, если выделено слово type, то команда преобразуется в **Open type,** становится активной и ее выбор приводит к открытию окна с информацией о выделенном объекте.

Операция **Run Script...** открывает окно загрузки внешних (Script) файлов, показанное на рис. 2.16. Имя файла можно задать в явном виде или же, нажав кнопку **Browse**, открыть окно поиска файла.

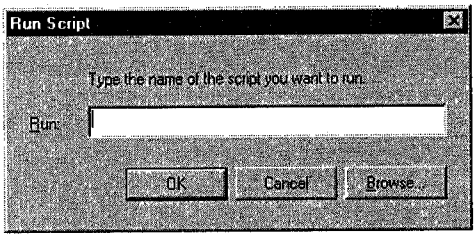


Рис. 2.16. Окно загрузки Script-файлов

Кнопка **OK** служит для подтверждения загрузки файла, а кнопка **Cancel** позволяет от нее отказаться.

Операции загрузки, записи и просмотра файлов рабочего пространства

Операция **Load Workspace...** открывает окно (подобное показанному на рис. 2.3) для загрузки файлов с расширением .mat. Это бинарные файлы (в виде машинных кодов), которые загружают в память ПК определения переменных, векторов, матриц и так далее.

Операция **Save Workspace As...** открывает окно для записи рабочего пространства на диск. Оно показано на рис. 2.17.

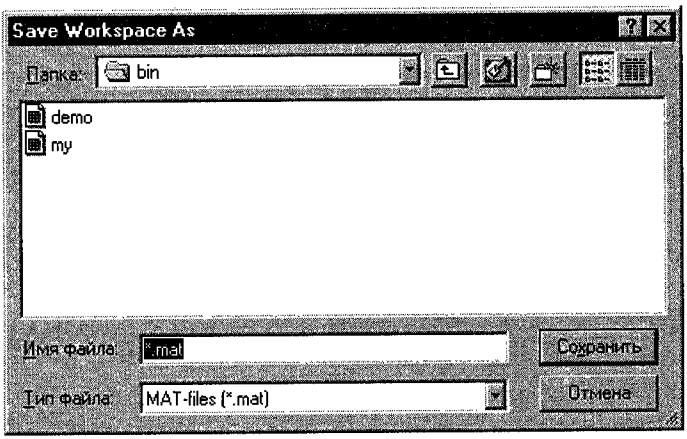


Рис. 2.17. Окно записи файлов на диск

Окно содержит панель инструментов, в которой имеются переключатель для просмотра файловой системы и пять управляющих кнопок. Перечислим их назначение (слева — направо):

- переход на верхний уровень дерева файловой системы;
- обзор рабочего стола;
- создание новой папки в текущей папке;

- вывод списка файлов в кратком виде;
- вывод списка файлов в полном виде.

Команда **Show Workspace** выводит описанное выше окно просмотра рабочего пространства (см. рис. 2.6).

2.3.4. Настройка MATLAB и функция path

Следующие две операции относятся к настройке параметров системы. Команда **Set Path...** выводит окно настройки путей файловой системы. Это окно показано на рис. 2.8. В командном режиме пути файловой системы выводятся с использованием функции **path**.

Команда **Preferences...** открывает окно настройки параметров системы, показанное на рис. 2.18. Это окно имеет ряд вкладок. В качестве примера на рис. 2.18 приведено данное окно с открытой вкладкой General, которая предназначена для установки ряда опций общего характера.

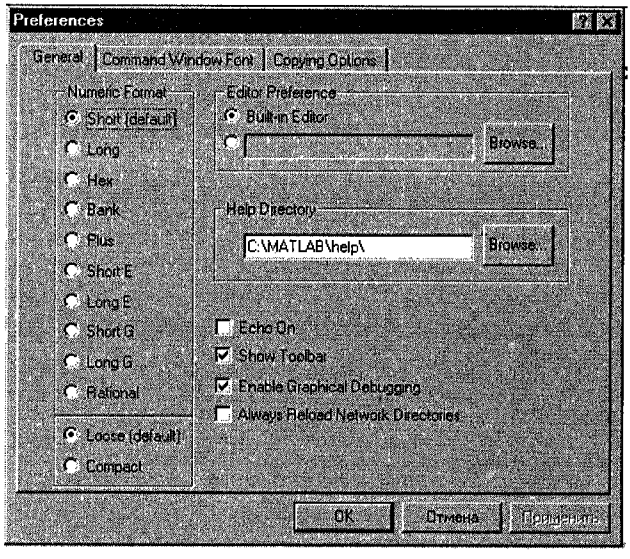


Рис. 2.18. Окно Preference с открытой вкладкой General

Прежде всего следует отметить опции Numeric format, которые определяют формат представления чисел. Перечень форматов чисел дан в левой части окна.

Следующая вкладка Command Window Font (рис. 2.19) служит для установки наборов шрифтов, которые используются для работы в командном режиме. Несмотря на широкие возможности выбора шрифтов, не рекомендуется менять установленные по умолчанию наборы, поскольку они оптимизированы для большинства пользователей.

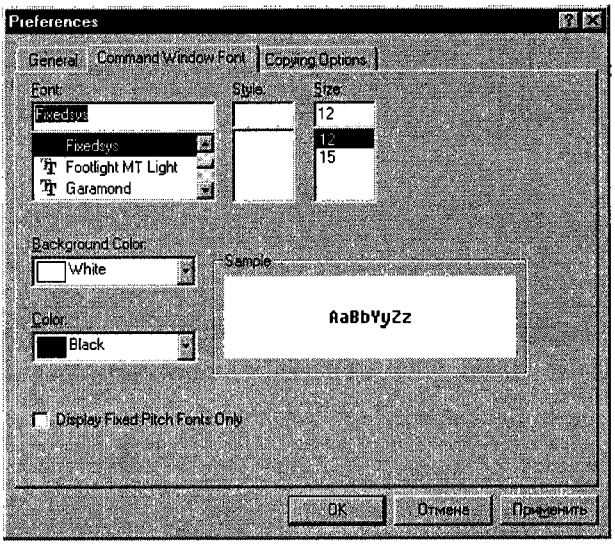


Рис. 2.19. Окно Preference с открытой вкладкой Command Window Font

Вкладка Copying Options служит для установки опций для работы с буфером промежуточного хранения Clipboard (см. рис. 2.20). Имеется возможность задания типа сохраняемых данных в виде Meta-файлов или Bitmap-файлов, а также установки цвета фона при копировании графиков в буфер.

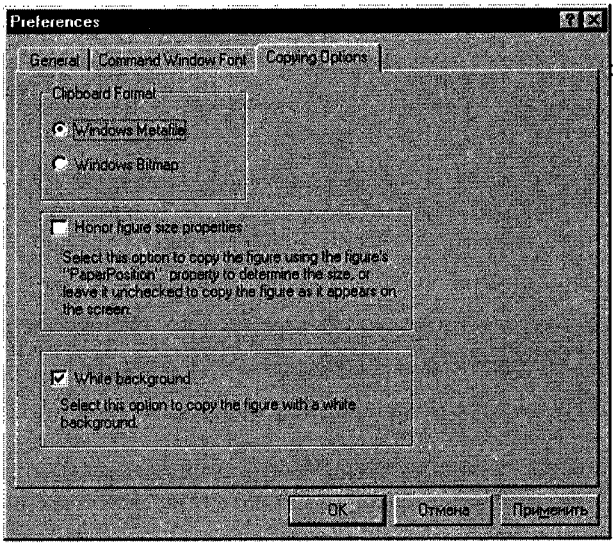


Рис. 2.20. Окно Preference с открытой вкладкой Copying Options

Таким образом, окно Preference дает обширные возможности по настройке системы.

2.3.5. Обеспечение печати — команды Print Setup, Print и Print Selection

В MATLAB для печати используются стандартные средства Windows. Пункт **File** главного меню содержит три позиции, первая из которых **Print Setup...** открывает окно установки начальных опций печати — рис. 2.21. Здесь можно выбрать тип принтера (если их несколько), формат бумаги и расположение печати.

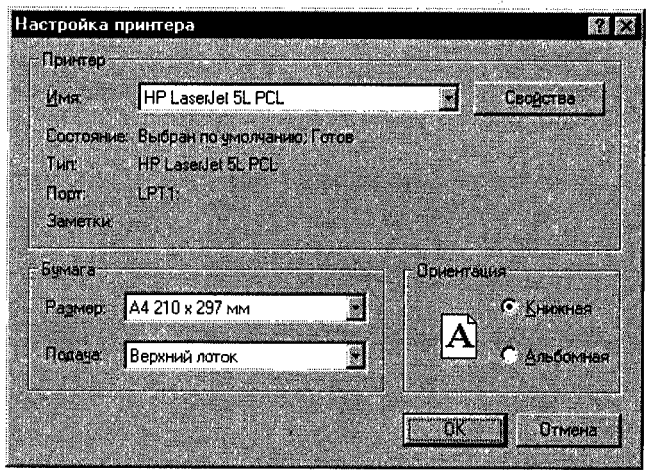


Рис. 2.21. Окно установки начальных опций печати

Другая позиция — **Print** — служит для вывода окна печати, показанного на рис. 2.22. В этом окне также имеется возможность выбора типа принтера и вывода окна со свойствами печати. В нем можно определить, с какой страницы начинается печать и объем печати.

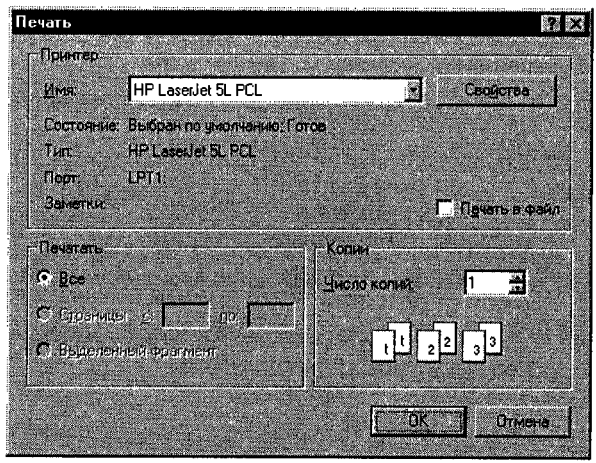


Рис. 2.22. Окно печати

Третья операция **Print Selection...** становится доступной только если в сессии выделен какой-либо фрагмент. На печать выводится только выделенный фрагмент.

Вообще говоря, MATLAB имеет специальные команды для печати, которые вводятся в командной строке, однако возможности Windows намного шире, и в данном случае нет смысла пользоваться командными средствами MATLAB.

2.3.6. Позиция Edit главного меню

Подменю позиции **Edit** (рис. 2.23) содержит операции редактирования, типичные для большинства приложений Windows.

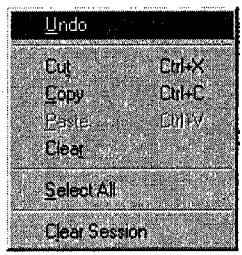


Рис. 2.23. Подменю позиции **Edit** главного меню

Как видно из рис. 2.23, это подменю имеет следующие операции и команды:

Undo — отмена результата предшествующей операции;
Cut — вырезание выделенного фрагмента и перенос его в буфер;
Copy — копирование выделенного фрагмента в буфер;
Paste — вставка фрагмента из буфера по месту расположения курсора мышки;
Clear — зарезервированная операция очистки выделенных областей;
Select All — выделение всей сессии;
Clear Session — очистка сессии (с сохранением определенных объектов).

Назначение указанных команд и операций уже обсуждалось. Отметим лишь, что операция **Clear Session** очищает окно командного режима работы и помещает маркер ввода в верхний левый угол окна. Однако все определения стертых таким образом сессий сохраняются в памяти компьютера. Для их очистки используется также команда **clc**, вводимая в командном режиме.

2.3.7. Позиция Windows главного меню

Позиция Windows главного меню выглядит как дань моде. Все, что она делает – представляет окно командного режима MATLAB в стандартном (не полностью открытом) виде. Три кнопки в конце титульной строки окна позволяют свернуть его в кноп-

ку панели задач, раскрыть на весь экран (или вернуть в частично открытое состояние) и закрыть окно, что означает завершение работы.

2.3.8. Позиция Help главного меню

Подменю **Help** главного меню открывает доступ к справочным системам MATLAB и к информации о производителе MATLAB. Подменю Help показано на рис. 2.24.

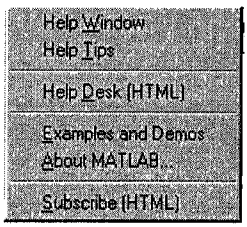


Рис. 2.24. Подменю **Help**

Основные возможности справочной системы обсуждаются в следующих разделах. Поэтому здесь мы остановимся только на позиции **About MATLAB**. Она выводит окно, содержащее информацию о системе. На рис. 2.25 это окно показано на фоне рабочего окна системы.

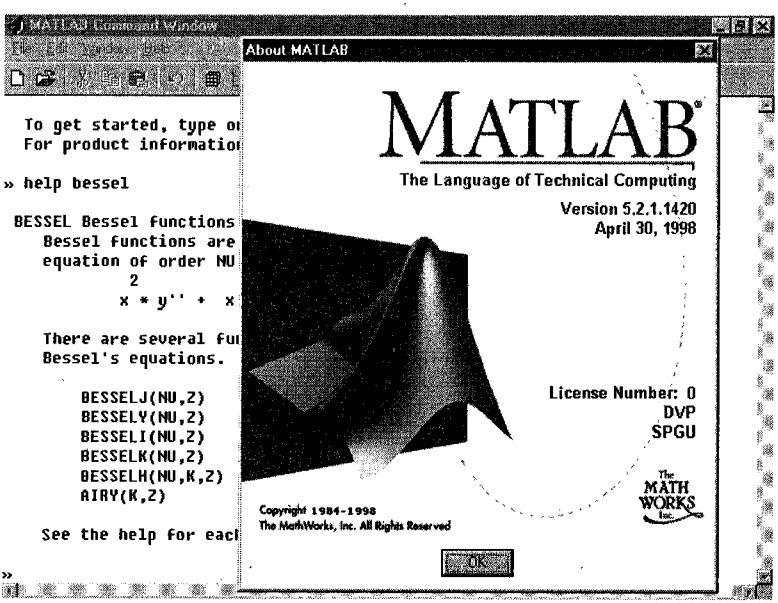


Рис. 2.25. Окно со сведениями о системе

В окне **About MATLAB** можно найти информацию о применяемой версии системы (в нашем случае это 5.2.1) и о дате ее создания (30 апреля 1998 года).

2.4. MATLAB в Internet

2.4.1. Немного о роли Internet

Представление математических систем в глобальной сети Internet [53] сегодня становится нормой. Свои сайты (страницы) в Internet имеют все фирмы-разработчики математических систем. MathWorks Inc — создатель системы MATLAB — также имеет свой, регулярно обновляемый сайт.

Необходимо отметить, что Internet дает возможность пользоваться самой последней версией справочной системы Help Desk, а также следить за модернизацией системы MATLAB фирмой-разработчиком и осуществлять (с помощью получаемых из Internet сервисных средств) модернизацию установленной на ПК пользователя системы.

2.4.2. Связь с разработчиком MATLAB через Internet

Позиция **Subscribe (HTML)** подменю **Help** выводит окно (рис. 2.26) с контактными данными фирмы MathWorks. Из этого окна можно непосредственно обратиться к главной странице фирмы (**www.mathworks.com**).

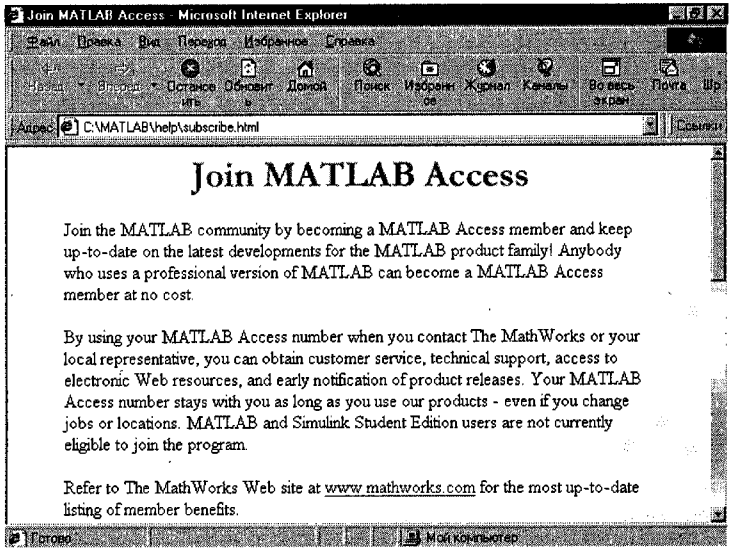


Рис. 2.26. Окно для связи через Internet с разработчиком системы

Следует отметить, что данная позиция подменю **Help** будет активна только в случае, если на ПК установлен какой-либо из браузеров для работы в сети Internet. Непосредственное подключение к Internet изначально не требуется.

2.4.3. Регистрация через Internet

Не откажите себе в удовольствии посетить страницу фирмы MathWorks. Активизируйте для этого гиперссылку **www.mathworks.com** в нижней части окна, показанного на рис. 2.27. После этого автоматически будет запущена система соединения с вашим провайдером Internet и начнется загрузка страницы с указанным адресом. На рис. 2.27 показано, как выглядит страница при использовании браузера Microsoft Internet Explorer 4.0.

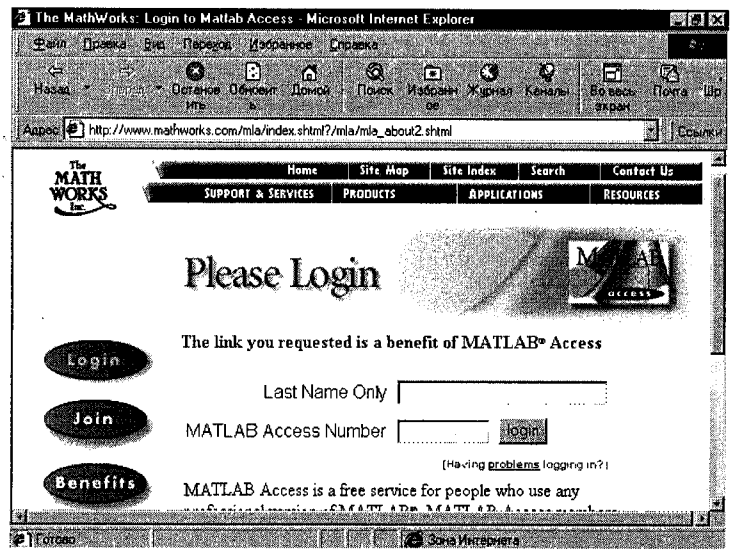


Рис. 2.27. Регистрационная страница фирмы MathWorks в Internet

Страница содержит регистрационную форму, которую следует заполнить, что бы стать официально зарегистрированным пользователем системы MATLAB.

2.4.4. Основная страница фирмы MathWorks

Активизируя гиперссылку Home на регистрационной странице, можно перейти к просмотру главной страницы сайта фирмы. Она представлена на рис. 2.28

Заметим, что основная страница фирмы MathWorks регулярно обновляется и за время подготовки этой книги уже сменилась новой — на ней представлена очередная 11-я реализация системы MATLAB 5.3 с интегрированным в нее пакетом Simulink 3.0.

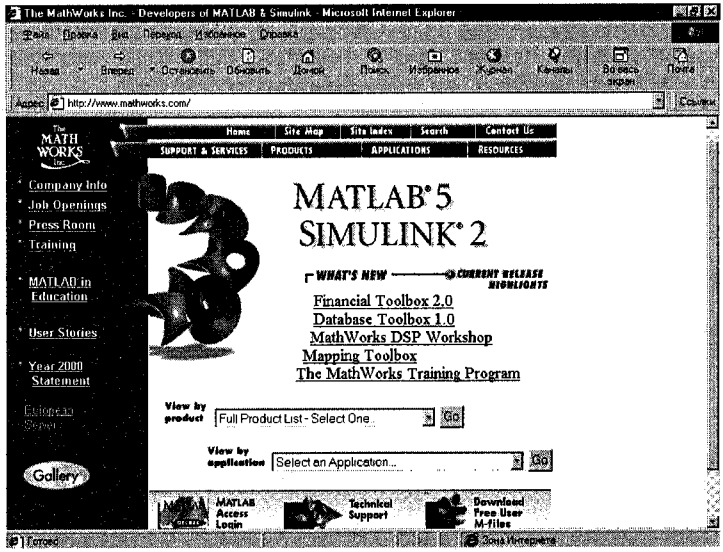


Рис. 2.28. Основная страница фирмы MathWorks

2.4.5. MATLAB в образовании

Как уже отмечалось, MATLAB широко используется в технике, в науке и в сфере образования. В качестве примера на рис. 2.29 показана Web-страница, иллюстрирующая применение MATLAB в образовании.

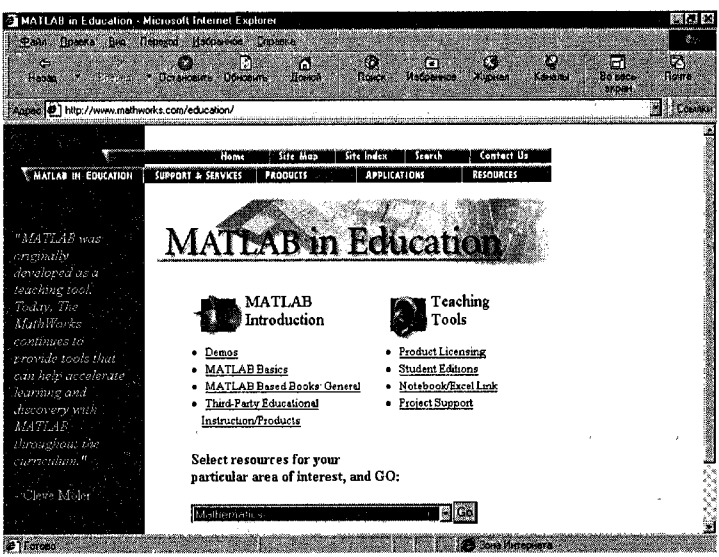


Рис. 2.29. Web-страница об использовании MATLAB в образовании

На сайте фирмы можно найти информацию как о программных продуктах фирмы, так и о пакетах расширения, примеры решения различных задач и так далее. На рис. 2.30 приведена часть страницы фирмы MathWorks с перечнем документации, которую можно получить через Internet.

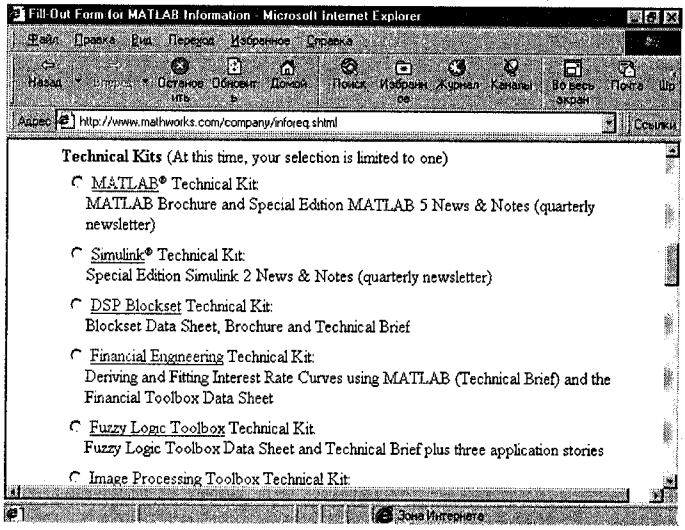


Рис. 2.30. Фрагмент Web-страницы со списком документации, которую можно получить через Internet

Университеты и другие учебные заведения наверняка заинтересует информация о дешевых (и несколько урезанных в возможностях) студенческих версиях системы MATLAB. Доступ к их получению виден на странице рис. 2.29.

2.4.6. Обновление системы MATLAB через Internet

Internet позволяет производить обновление программных продуктов. Так, систему MATLAB 5.2 можно обновить до следующей 10-й реализации (MATLAB 5.2.1), обратившись к странице Download фирмы MathWorks, которая показана на рис. 2.31.

Чтобы модернизировать систему, необходимо указать желаемую версию, а также платформу, на которой используется система. Для этого используются переключатели, расположенные внизу страницы: один из этих переключателей показан на рис. 2.31 снизу и слева. Он выводит список реализаций MATLAB. Другой переключатель (он открыт) выводит список компьютерных платформ. После этого надо нажать кнопку Go!, которая вызывает появление инструкции по загрузке файлов.

Инструкция по загрузке файлов представлена на рис. 2.32. Здесь необходимо указать, какие файлы вы намерены получить через Internet. Следует отметить, что специальной регистрации с применением паролей в данном случае не требуется.

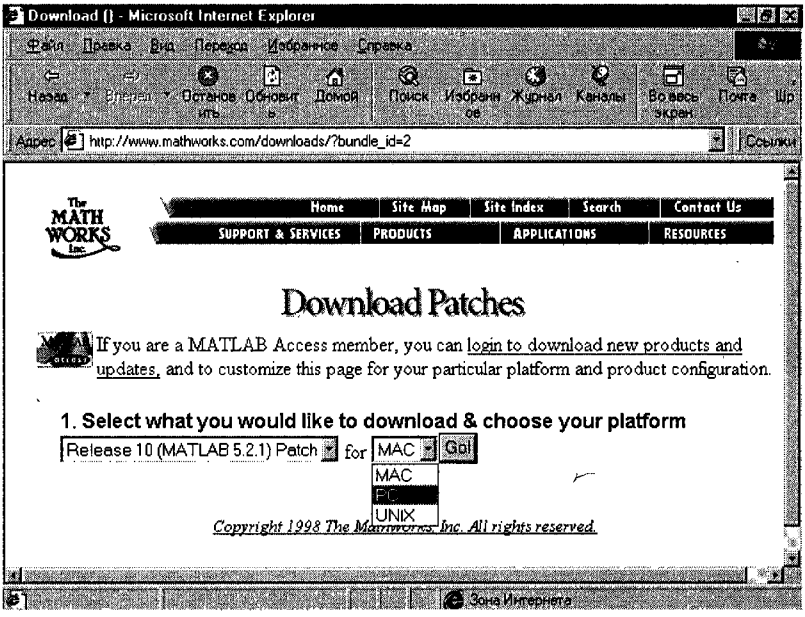


Рис. 2.31. Web-страница, позволяющая списать файлы системы MATLAB

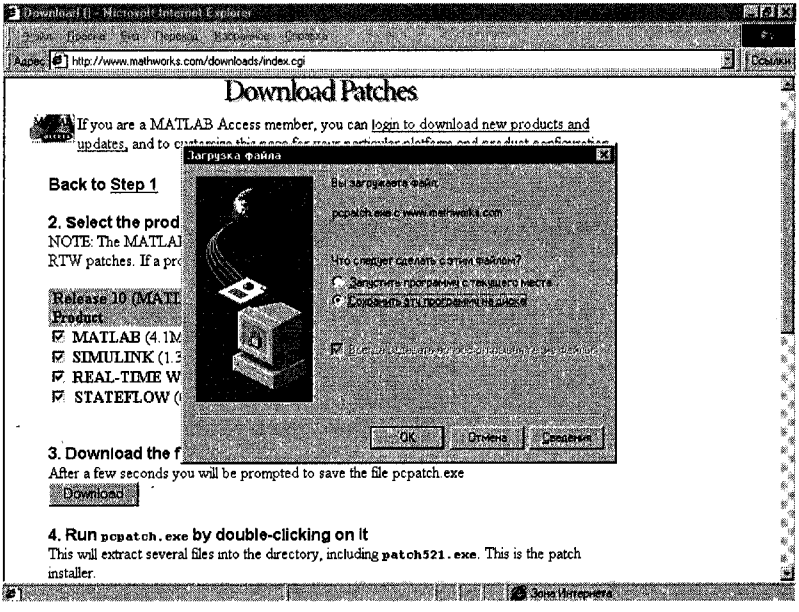


Рис. 2.32. Страница с перечнем файлов для загрузки и окном записи файлов на диск

Процесс загрузки можно контролировать с помощью окна с линейным индикатором — рис. 2.33.

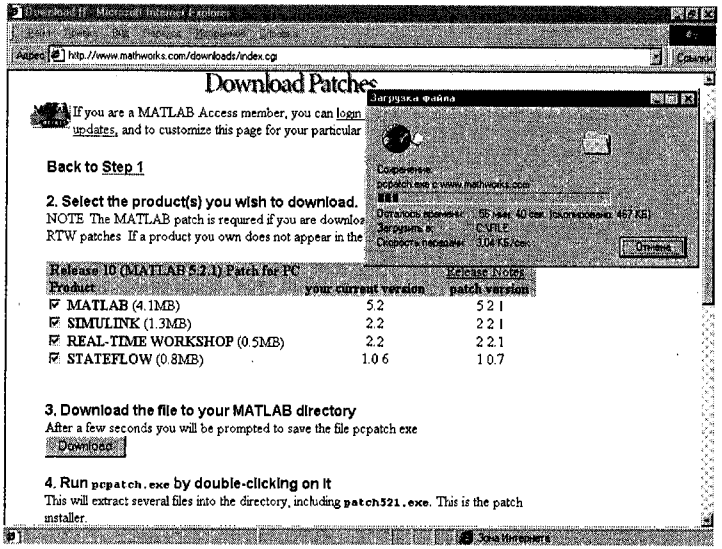


Рис. 2.33. Процесс перекачки файла с узла фирмы MathWorks на ПК пользователя

По завершении этого процесса необходимо поместить полученные файлы в основной каталог системы MATLAB и запустить файл path521.exe. Будет запущена представленная этим файлом программа, которая произведет необходимые изменения в более ранней версии системы и обновит ее до уровня 10-й реализации, получившей название MATLAB 5.2.1. Дальнейшее обновление до 11-й реализации, увы, уже невозможно — последнюю версию системы MATLAB 5.3 надо покупать в обычном порядке.

2.5. Справочная система Help Desk

2.5.1. Запуск справочной системы Help Desk

В MATLAB имеется мощная справочная система **Help Desk**, которая представляет собой набор электронных статей, оформленных в виде HTML-файлов. Такая организация справочной системы имеет два очевидных преимущества: для просмотра файлов может использоваться браузер Internet и при этом имеется возможность обновления документации.

Для запуска справочной системы Help Desk следует использовать одноименную команду в позиции Help главного меню. При этом запустится браузер и откроется окно подсистемы Help Desk, показанное на рис. 2.34.

Каждый раздел справочной системы представлен в виде гиперссылки, активизация которой приводит к переходу на соответствующую HTML-страницу. На рис. 2.35 показан один из документов справочной системы Help Desk.

Рис. 2.34. Основное окно справочной системы Help Desk

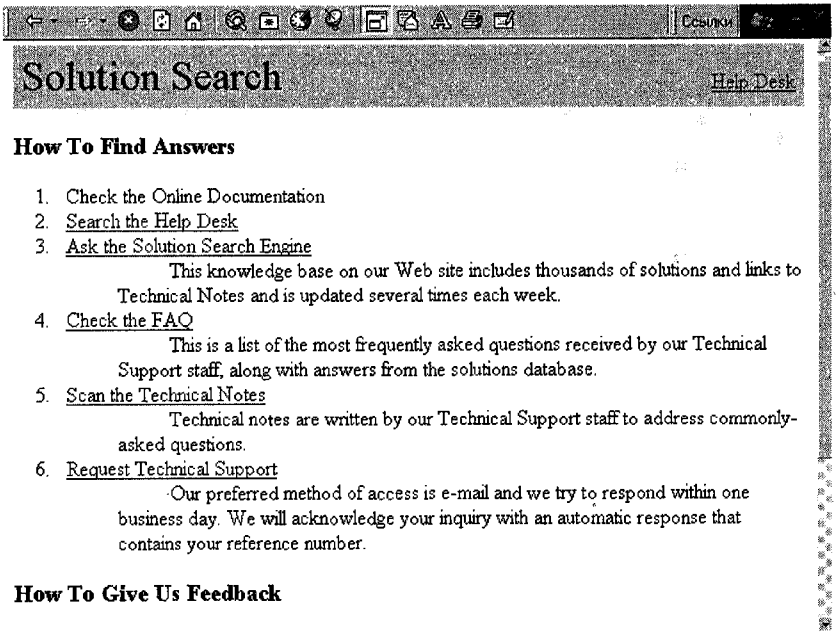


Рис. 2.35. Один из документов справочной системы Help Desk

Несмотря на удобства справочной системы Help Desk, надо отметить, что ее содержимое во многом дублирует другие справочные подсистемы MATLAB. Кроме того, многие варианты поставки системы не содержат файлов этой системы.

2.5.2. Документация системы в PDF- формате

Описанные выше средства помощи рассчитаны на то, что пользователь уже знаком с системой и желает быстро получить справку по ее конкретным возможностям. Но для общего знакомства с системой они не подходят.

Поэтому хотелось бы обратить внимание читателя на подробные электронные книги, обычно представленные в виде файлов формата PDF, для работы с которыми применяются такие программы, как Acrobat Reader или Adobe Acrobat. Первая позволяет только просматривать материалы книг, а вторая — редактировать.

Справочная система Help Desk содержит гиперссылку Online Manuals (in PDF), предоставляющую доступ к электронной справочной документации, которая поставляется в виде файлов PDF. На рис. 2.36 показано окно этой гиперссылки с перечнем документов в формате PDF. Среди документов крупные (сотни страниц) руководства по функциям системы, языку ее программирования, графическим функциям и др.

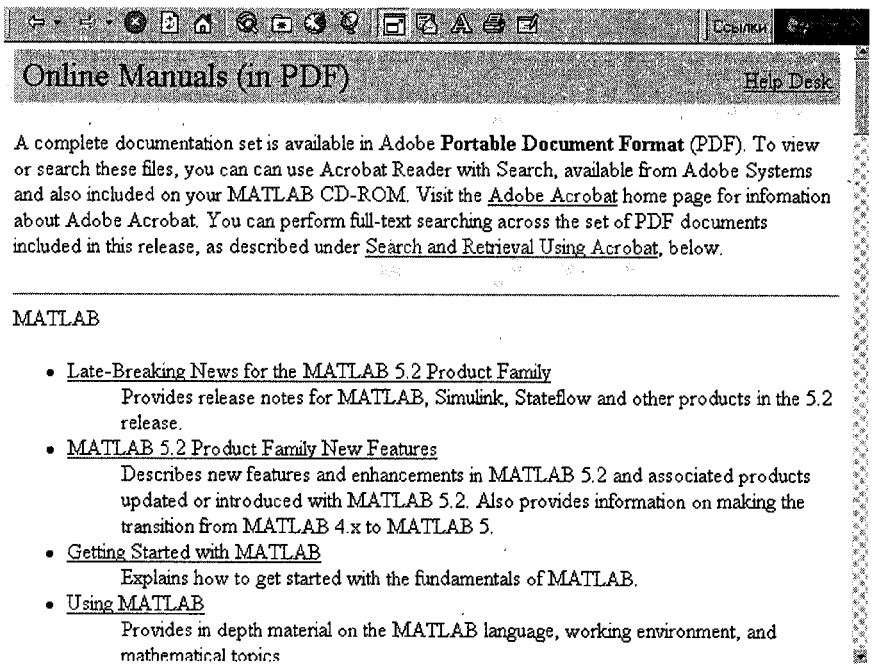


Рис. 2.36. Окно гиперссылки Online Manuals (in PDF)

Возможность просмотра документов в этом формате требует наличия программы Acrobat Reader и файлов самих документов. Не все поставки системы MATLAB имеют эти средства. Полный объем документации по системе MATLAB и пакетам прикладных программ достигает сейчас примерно 200 Мбайт, что делает документацию труднообозримой.

2.5.3. Пример просмотра документов в формате PDF

В качестве примера работы с электронными документами формата PDF рассмотрим книгу-справочник по работе с системой MATLAB (Reference Manual). Его титульная страница показана на рис. 2.37. Заметим, что она появляется после загрузки программы Acrobat Reader (временное окно ее не показано).

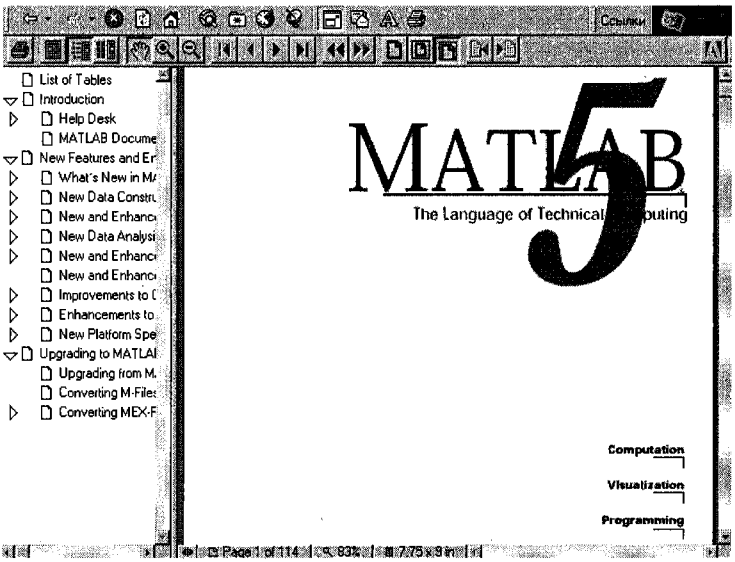


Рис. 2.37. Начальная страница электронного справочного руководства по MATLAB

Как можно видеть, страница содержит древообразный перечень разделов в левой части и большое окно просмотра в правой части. С помощью левого окна можно указать явно раздел справочника, который желательно просмотреть. Можно также перелистывать материал справочника с помощью линейки прокрутки, расположенной справа в окне просмотра. Таким образом справочные материалы представлены в типовой форме **электронных книг**.

Acrobat Reader сама по себе серьезная программа и ее описание в задачи данной книги не входит. Однако надо отметить, что работа с программой интуитивно понятна и не вызывает особых трудностей даже у не очень опытного пользователя.

На рис. 2.38 дано представление одной из страниц, данное с некоторым увеличением и в полностью открытом окне просмотра.

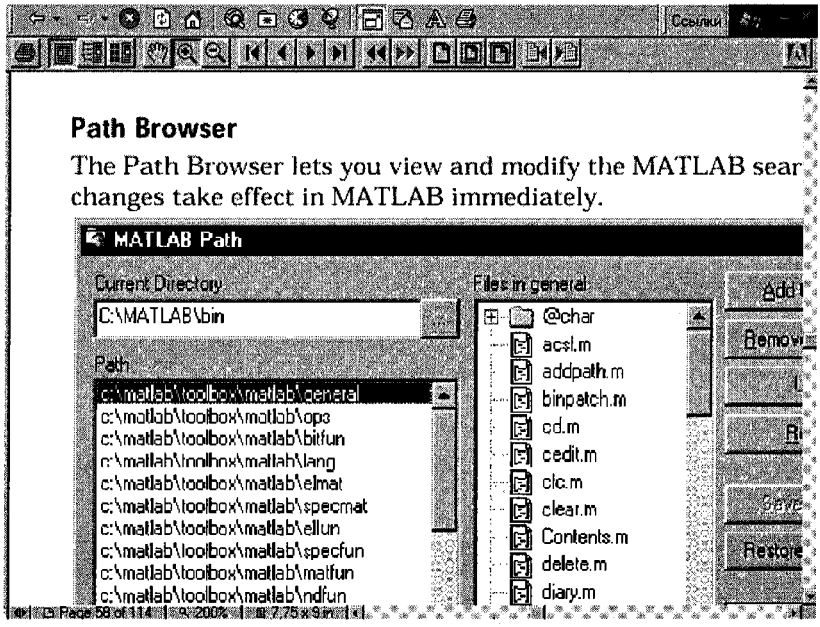


Рис. 2.38. Одна из страниц электронного справочника в окне просмотра программы Acrobat Reader

Из рис. 2.38 видно, что даже при увеличении изображения качество представления рисунков не очень высокое. Тем не менее оно достаточно для просмотра рисунков в обычном масштабе. Как и все другие виды справочной документации электронные книги по MATLAB подготовлены на английском языке.

2.5.4. Демонстрационные примеры — команда demo

В позиции **Help** главного меню имеется команда **Examples and Demos**, предоставляющая доступ к галерее примеров применения системы MATLAB, оформленной в стиле **Help Desk**. При запуске этой команды появляется окно демонстрационных примеров **MATLAB Demos**, показанное на рис. 2.39. Это окно можно вызвать выполнением команды **demo** в командном режиме.

В окне имеется три панели:

- левая панель с перечнем разделов, по которым предлагаются примеры;
- панель с описанием раздела примеров;
- панель с перечнем примеров по выбранному разделу.

Выбрав раздел примеров (указателем мыши и щелчком левой клавиши), затем следует указать соответствующий пример. На рис. 2.39 показан выбор раздела **Gallery** и примера **Hoops**. После этого нажатием кнопки Run (с именем примера) можно запустить М-файл с выбранным примером и наблюдать результат его работы (рис. 2.40).

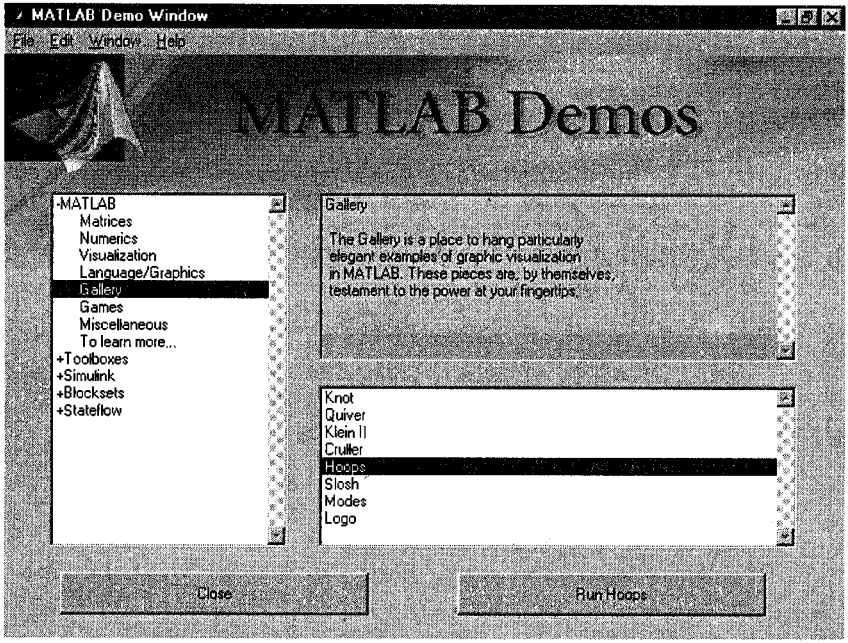


Рис. 2.39. Окно демонстрационных примеров

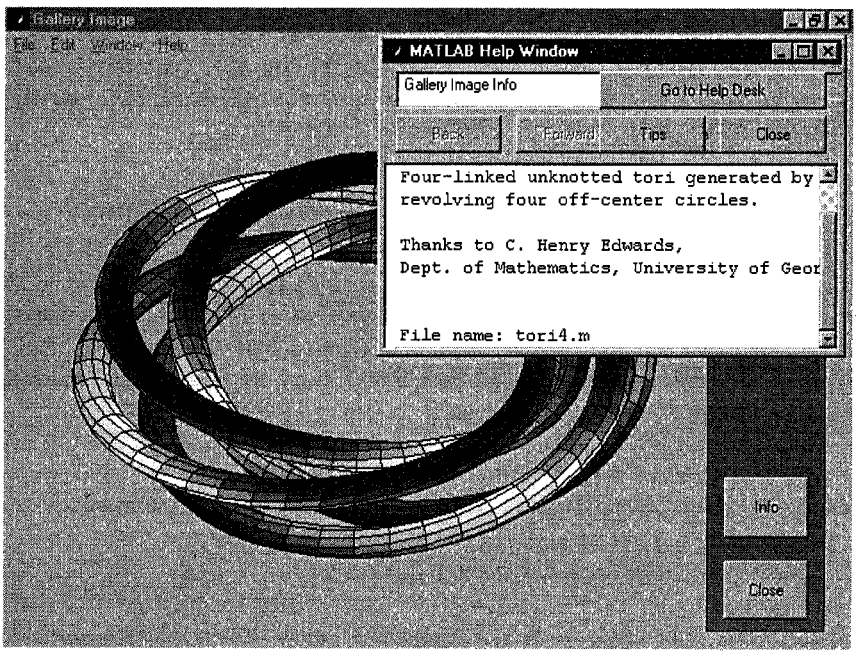


Рис. 2.40. Результат выполнения примера **Hoops** и информационное окно о его файле

Этот пример демонстрирует построение сплетающихся в пространстве нескольких объемных обручей с функциональной цветной окраской. Следующий пример, показанный на рис. 2.41, иллюстрирует работу программного симулятора Simulink — одного из самых мощных пакетов прикладных программ, расширяющих возможности системы MATLAB. Сверху показана блок-схема моделируемого объекта, а внизу результат симуляции его работы.

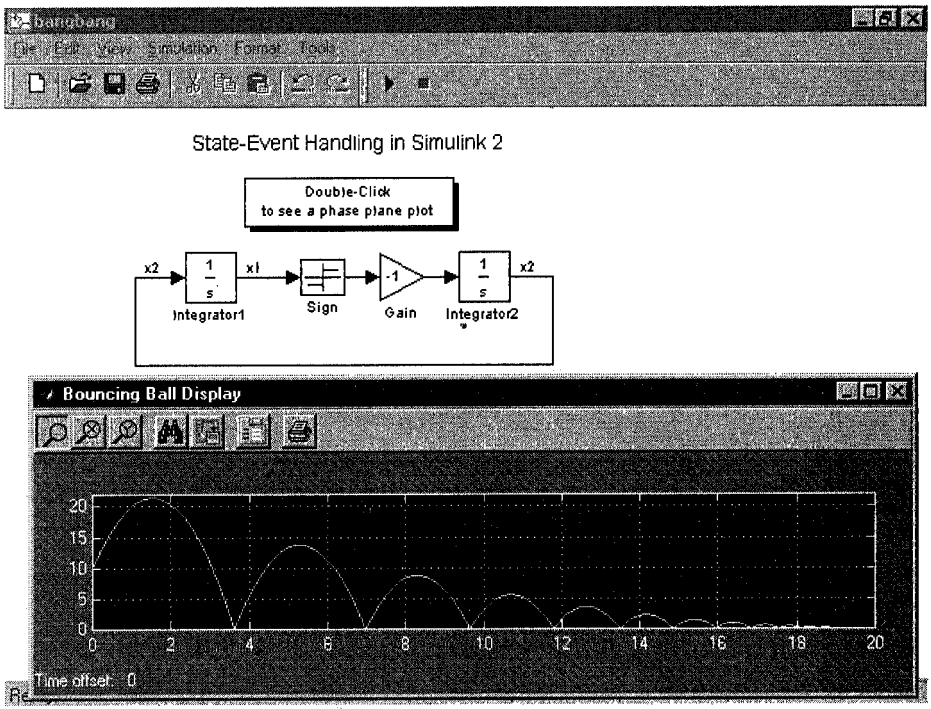


Рис. 2.41. Пример работы пакета прикладных программ Simulink

Окно MATLAB Demos дает возможность ознакомиться со многими десятками самых серьезных примеров применения системы MATLAB и позволяет убедиться в высокой степени визуализации их решений. При необходимости всегда можно ознакомиться с файлом того или иного примера и использовать его для решения схожих задач.

2.5.5. Копирование демонстрационных примеров

Вполне возможно, что вы захотите воспользоваться каким-либо примером для решения собственных задач. Для этого можно использовать М-файл примера или перенести его текст в командное окно MATLAB, используя буфер промежуточного хранения. Покажем, как это делается.

На рис. 2.42 показан один из примеров создания графика уровней равной высоты для сложной трехмерной поверхности. Высота при этом задается цветом точек поверхности, то есть используется функциональная окраска.

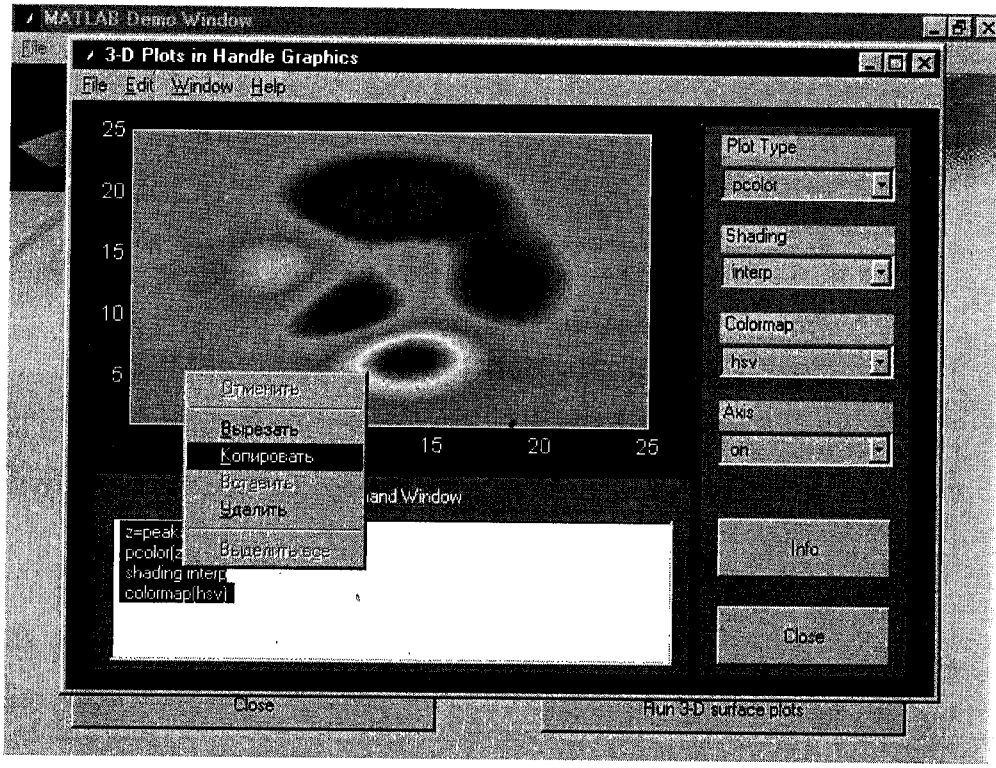


Рис. 2.42. Подготовка к копированию примера

В нижней части окна примера показано, каким образом осуществляется копирование примера: текст примера выделяется мышью и затем используется контекстно-зависимое меню правой клавиши мыши. Меню расположено рядом с копируемым текстом. В меню следует использовать команду **Копировать**, в результате чего текст примера попадет в буфер промежуточного хранения.

После этого надо вернуться в командное окно MATLAB и, используя команду **Paste** (Вставить) в главном меню, перенести текст примера из буфера в текущую ячейку ввода. Запустив ее (как обычно — клавишами Ctrl и Enter), можно наблюдать исполнение примера, как это показано на рис. 2.43.

Остается еще раз отметить, что обилие примеров в системе MATLAB облегчает знакомство с различными аспектами применения системы. Большинство примеров содержат решения отнюдь не тривиальных математических задач и прекрасно иллюстрируют обширные возможности системы.

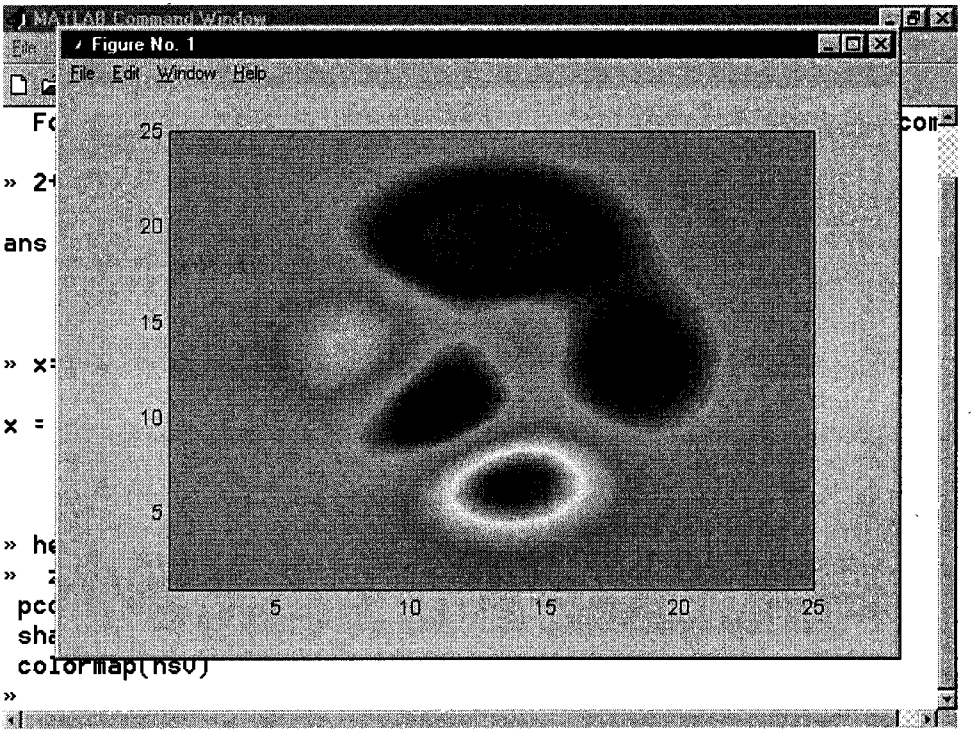


Рис. 2.43. Исполнение скопированного примера из командного окна MATLAB

2.6. Редактирование и отладка M-файлов

2.6.1. Назначение редактора и отладчика M-файлов

Для подготовки, редактирования и отладки M-файлов служит специальный многооконный редактор. Он выполнен как типичное приложение Windows. Редактор можно вызвать командой **edit** из командной строки или командой **New — M-file** из позиции **File** главного меню. 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 позиции **File** главного меню редактора.

На рис. 2.44 показано окно редактора и отладчика с текстом простого файла. После записи его на диск можно заметить, что команда **Run** в позиции **Tools** главного меню редактора становится активной (до записи файла на диск она пассивна) и позволяет произвести запуск файла.

Запустив команду **Run,** можно наблюдать исполнение M-файла — в нашем случае это построение рисунка в своем окне и вывод надписи о делении на 0 в ходе вычисления функции sin(x)/x в окне системы (рис. 2.45).

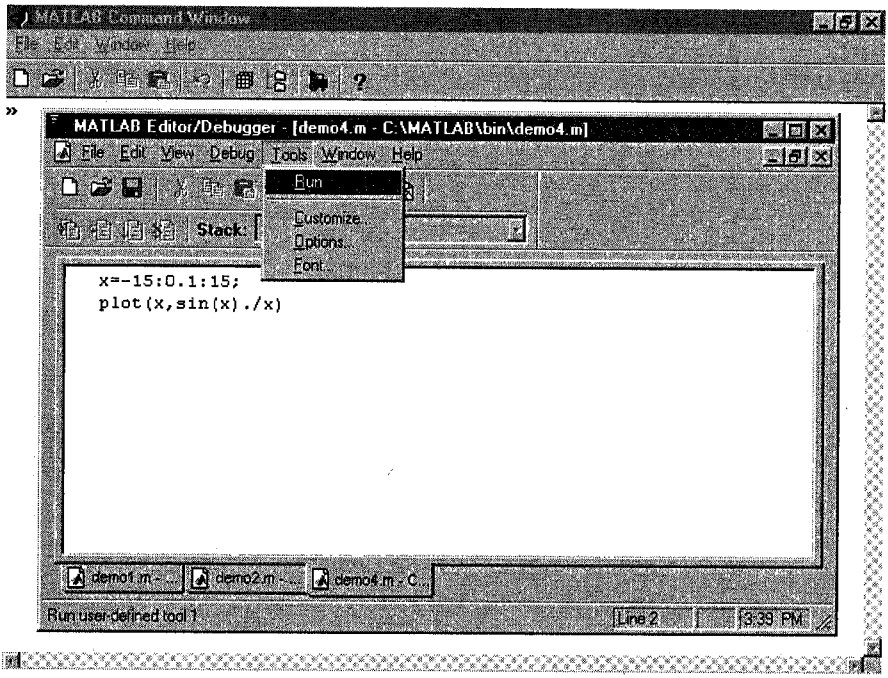


Рис. 2.44. Редактор-отладчик файлов перед запуском простого файла

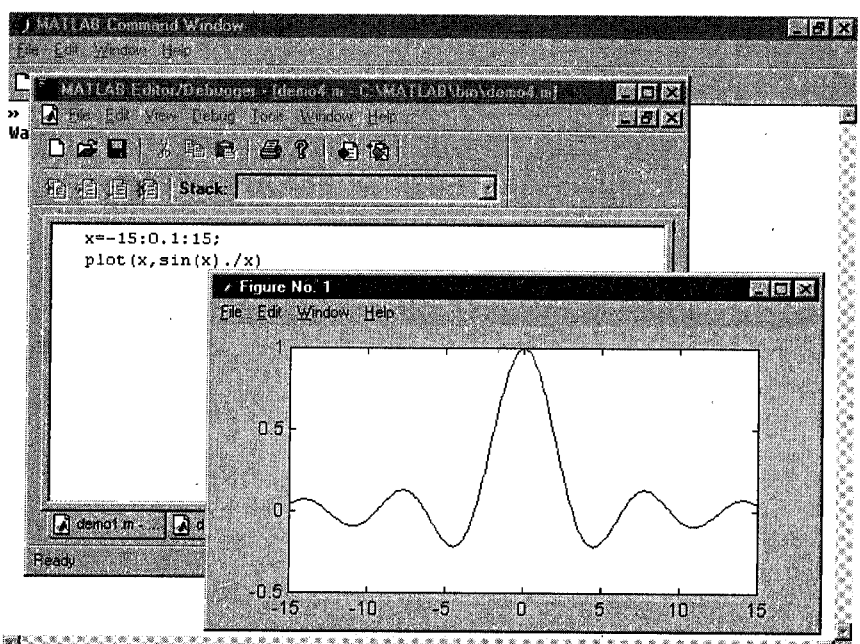


Рис. 2.45. Исполнение файла, показанного в окне редактора на рис. 2.44

С первого взгляда может показаться, что редактор и отладчик — просто лишнее звено в цепочке пользователь — MATLAB. И в самом деле, текст файла можно было бы ввести в окно системы и получить тот же результат.

Однако на деле редактор-отладчик выполняет важную роль. Он позволяет создать М-файл (программу) без той многочисленной шелухи, которая сопровождает работу в командном режиме. Далее мы убедимся, что текст такого файла подвергается тщательной синтаксической проверке, в ходе которой выявляются и отсеиваются многие ошибки пользователя. Таким образом, редактор обеспечивает синтаксический контроль файла.

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 так далее. Наконец, редактор позволяет записать файл в текстовом формате и увековечить труды в файловой системе MATLAB.

2.6.2. Цветовые выделения и синтаксический контроль

Редактор и отладчик М-файлов выполняет синтаксический контроль текстов (листингов) файлов по мере ввода текстов. 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 вероятность синтаксических ошибок резко снижается.

Однако далеко не все ошибки диагностируются. Ошибки, связанные с неверным применением операторов или функций (например, применение оператора - вместо + или функции cos(x) вместо sin(x) и так далее), не способна обнаружить ни одна система программирования. Устранение такого рода ошибок — дело пользователя, отлаживающего свои алгоритмы и программы.

2.6.3. Понятие о файлах-сценариях и файлах-функциях

Здесь полезно отметить, что М-файлы, создаваемые отладчиком, делятся на два класса:

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

Видимый в окне редактора на рис. 2.44 файл является файлом-сценарием, или Script-файлом. Этот файл вообще не получает данных извне с помощью списка входных параметров (ибо его просто нет) и является примером простой процедуры без параметров. Он использует глобальные переменные.

Файл-функция отличается от файла-сценария прежде всего тем, что созданная им функция имеет входные параметры, список которых указывается в круглых скобках. Входящие в файл-функцию переменные являются локальными. В дальнейшем мы вернемся к более подробному обсуждению этих файлов.

2.6.4. Локальные и глобальные переменные

Переменные, указанные в списке параметров функции, являются локальными переменными и служат для переноса значений, которые подставляются на их место при вызовах функций.

Эта особенность переменных-параметров хорошо видна при разборе примера, показанного на рис. 2.46. Здесь (признаемся, что неточно) задана некоторая функция двух переменных fun(x,y).

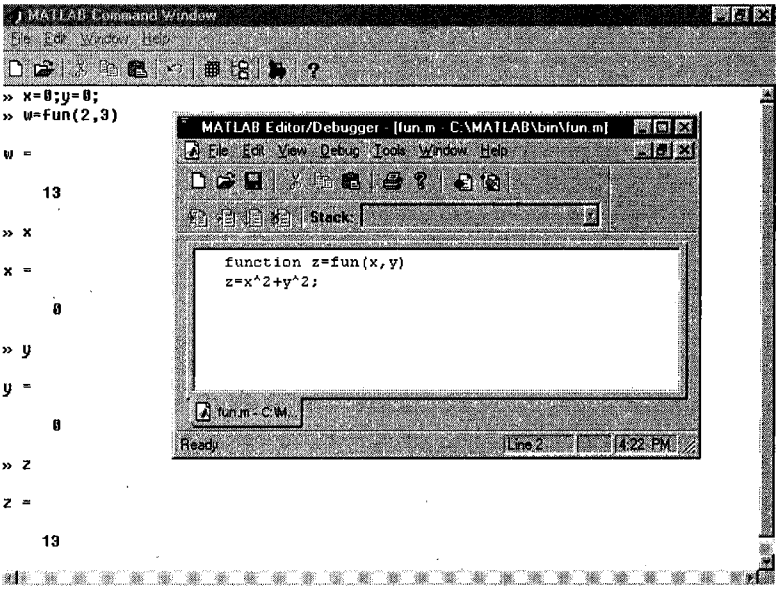


Рис. 2.46. Пример, поясняющий действие локальных и глобальных переменных при задании файла-функции

В этом примере в окне редактора создана функция fun двух переменных x и y, вычисляющая $z=x^2+y^2$. Поскольку переменные x и y указаны как параметры функции fun(x,y), то они являются локальными. В примере вне тела функции им заданы нуле-

вые значения. Очевидно, что при вычислении значения fun(2,3) в теле функции задастся x=2 и y=3. Поэтому результат — z=13. Однако после выхода из тела функции значения переменных x и y принимают свое исходное значение, равное нулю. Так что эти переменные меняют свои значения на значения параметров функции только локально — в пределах тела функции.

2.6.5. Побочные эффекты при работе с функциями

А каков статус переменной z в нашем примере? Нетрудно убедиться в том, что она будет глобальной. Изначально ее значение не определено. Но в теле функции эта переменная принимает значение z=13. И по выходе из тела функции она несет значение "чертовой дюжины". Этот пример — наглядное проявление **эффекта побочного действия** функции.

Причина побочного эффекта в данном случае — в неточном задании функции. Вы можете сами убедиться в том, что данная "функция" имеет еще один серьезный порок — она не возвращает свое значение, так что вычислить **2*fun(2,3)** вам не удастся. Для правильного задания функции нужно в конце поставить слово **return** (возврат). Тогда функция будет иметь следующий листинг:

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

Записав скорректированную функцию, можно убедиться в том, что на этот раз она будет работать верно:

```
» x=0; y=0; z=0;
» x
x =
    0
» y
y =
    0
» z
z =
    0
» 2*fun(2,3)
z =
   13
ans =

   26
» x
x =
```

```
     0
» y
y =
     0
» z
z =
     0
```

Внимательный читатель отметит, однако, что и здесь есть побочный эффект — временное значение переменной z=13 в теле цикла выводится на индикацию. Чтобы убрать этот эффект, достаточно установить знак ; после математического выражения, определяющего z, то есть задать функцию в виде:

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

После этого все вычисления пройдут гладко. Этот пример наглядно показывает, что пропуск любого слова или даже простого оператора (вроде знака ;) может привести к не сразу понятным побочным эффектам и даже неверной работе функции. Программирование требует особой точности и педантичности, именно поэтому далеко не все могут быть хорошими программистами.

2.6.6. Панель инструментов редактора и отладчика

Редактор имеет свое главное меню и свою инструментальную панель, которая может перемещаться мышью в любое подходящее место. Внешний вид инструментальной панели показан на рис. 2.47.

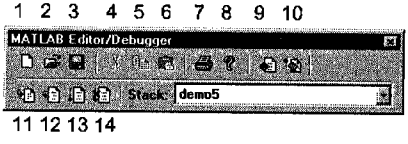


Рис. 2.47. Панель инструментов редактора и отладчика М-файлов

Назначение кнопок панели инструмента приводится ниже:

1. **New** — создание нового М-файла;
2. **Open** — вывод окна загрузки файла;
3. **Save** — запись файла на диск;
4. **Cut** — вырезка выделенного фрагмента и перенос его в буфер;
5. **Copy** — копирование выделенного объекта в буфер;
6. **Paste** — размещение фрагмента из буфера на место расположения курсора мышки;

7. **Print** — печать содержимого текущего окна редактора;
8. **About** — вывод данных о версии редактора;
9. **Set/Clear Breakpoint** — установка/сброс контрольной точки;
10. **Clear All Breakpoints** — сброс всех контрольных точек;
11. **Step In** — установка входа в М-файл;
12. **Single Step** — установка одиночного шага;
13. **Continue** — продолжение работы;
14. **Quit Debugging** — завершение отладки.

С назначением кнопок 1-8 вы уже знакомы, поскольку оно аналогично описанному для основного окна MATLAB. А вот о назначении других кнопок надо поговорить.

Основным приемом отладки М-файлов является установка в их тексте контрольных точек остановки (Breakpoints). Они устанавливаются (и сбрасываются) с помощью кнопки 9. Сброс всех контрольных точек обеспечивается кнопкой 10.

Рассмотрим рис. 2.48, на котором в окне редактора и отладчика видна программная конструкция цикла. Как будет меняться переменная s, значение которой должно давать сумму натуральных чисел? Прежде всего для отладки надо записать программу на диск, а затем установить против выражения s=s+i контрольную точку — она отчетливо видна на рис. 2.48. Для установки контрольной точки необходимо поместить курсор мыши в нужное место (против указанного выражения) и нажать кнопку 9.

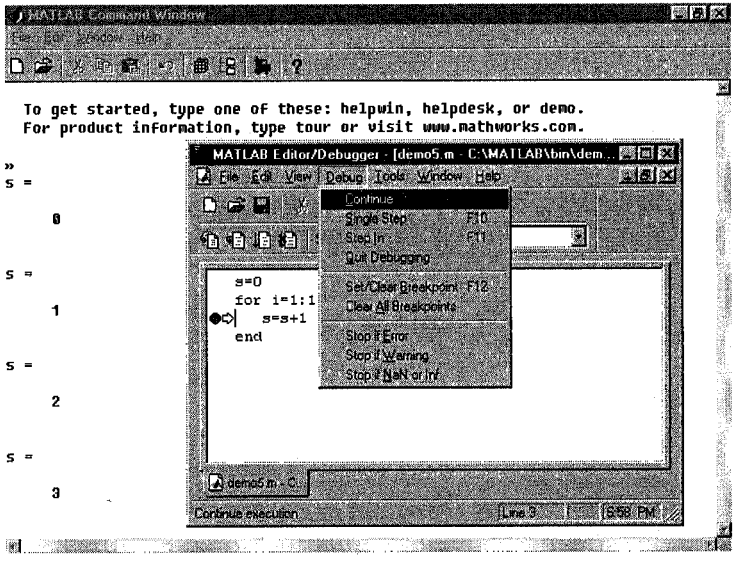


Рис. 2.48. Простейший пример на применение контрольной точки в программе

Теперь при пуске программы командой Run она будет исполнена до контрольной точки, после чего текущее значение s будет выведено в окне MATLAB. С помощью кнопки 13 Continue можно выполнить очередной шаг вычислений и так далее. Если

отпала необходимость останова в контрольных точках, достаточно кнопкой 10 удалить все контрольные точки. Желтая стрелка указывает, на какой контрольной точке произошел останов. В нашем случае такая точка одна, так что желтая стрелка будет всегда останавливаться на ней.

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 с одиночным шагом (кнопка 12) и с входом в М-файл (кнопка 11). Кнопка 13 продолжает исполнение программы после останова, а кнопка 14 прекращает операции отладки.

2.7. Ознакомительная система MATLAB Tour

2.7.1. Запуск ознакомительной системы — команда tour

MATLAB имеет еще одну ознакомительную систему — MATLAB Tour, которая интегрирована с системой демонстрационных примеров и со справочной системой Windows. Она позволяет совершить путешествие по системе MATLAB, откуда и происходит название Tour. Для вызова ознакомительной системы надо ввести в командной строке основного окна MATLAB команду

```
» tour
```

При исполнении этой команды появится окно ознакомительной системы, показанное на рис. 2.49. В этом окне имеется перечень разделов обучающей системы.

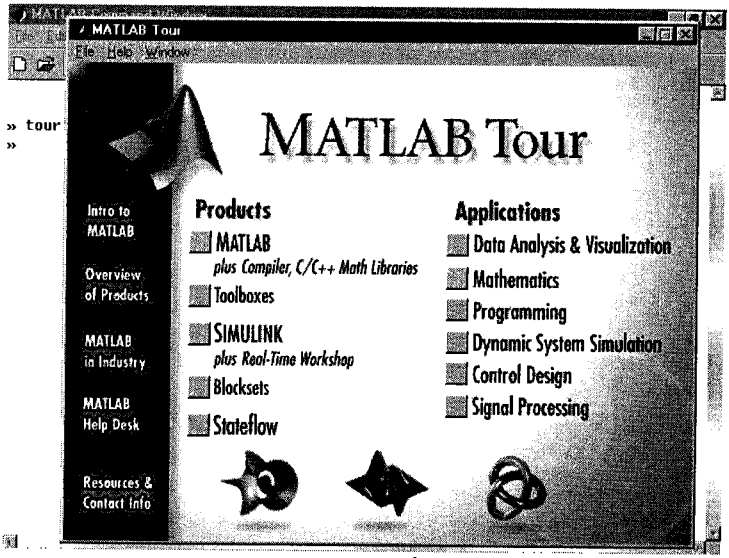


Рис. 2.49. Основное окно обучающей системы Tour

Для работы с обучающей системой достаточно выбрать нужный раздел, нажав соответствующую кнопку.

2.7.2. Пример работы с ознакомительной системой Tour

В качестве примера рассмотрим знакомство с разделом **Data Analysis&Visualization**, нажав соответствующую кнопку основного окна обучающей системы. При этом появится окно с перечнем примеров ознакомительной системы, показанное на рис. 2.50.

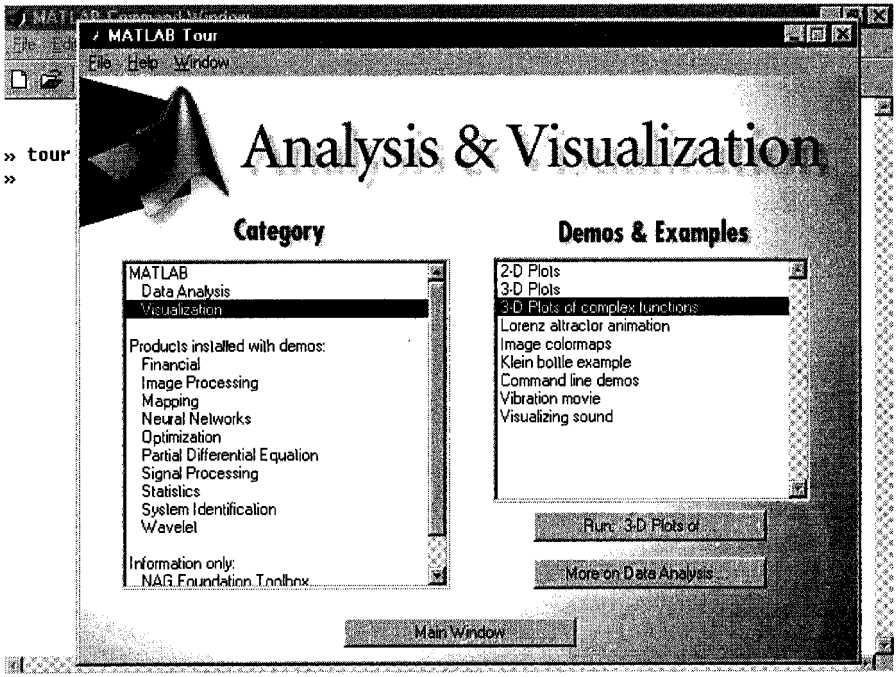


Рис. 2.50. Окно с перечнем примеров по разделу **Data analysis&Visualization**

В окне рис. 2.50 показан выбор примера на построение трехмерного (3D) графика для комплексной функции. Это один из многочисленных примеров специальной графики, которую имеет система MATLAB. После выбора примера следует нажать кнопку Run: с именем примера. В результате появится демонстрационное окно, в котором можно выбрать одну из пяти функций комплексной переменной Z. В итоге можно наблюдать построение графика функции (рис. 2.51).

Система Tour представляет собой одну из ряда возможностей знакомства с системой MATLAB. Хотя многие примеры в различных средствах ознакомления с системой повторяются, в целом Tour — одна из наиболее полных справочных систем на основе рассмотрения реальных примеров.

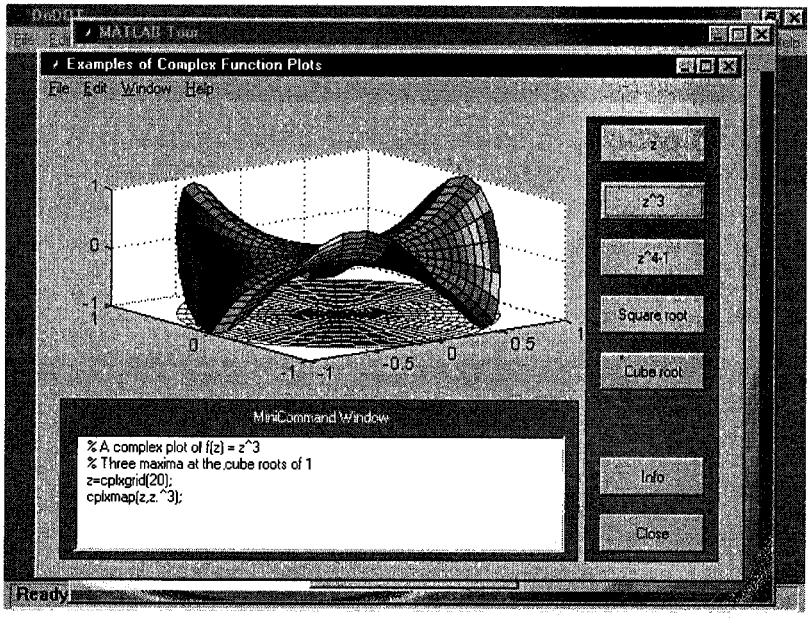


Рис. 2.51. Окно примера на построение графика функции комплексной переменной

2.8. Общение MATLAB с операционной системой и сетью Internet

Общение системы MATLAB с операционной системой MS-DOS многим покажется рудиментарной возможностью. Так, во время написания данной книги такое общение не потребовалось. Но, как говорится, "из песни слов не выкинешь" — MATLAB позволяет из командой строки пользоваться основными услугами старушки MS-DOS. Есть возможность общения и с другими операционными системами и даже с глобальной сетью Internet.

2.8.1. Работа с каталогами — команды cd, dir и pwd

Для перехода в новый каталог служит команда:

cd wd — переход в указанную директорию wd.
wd=cd — возвращает строку с полным именем текущей директории.
wd=cd.. — возвращает строку с полным именем директории, предшествующей текущей.
Примеры:

```
» cd
C:\MATLAB\bin
» cd C:\MATLAB\TOOLS
```

```
??? Name is nonexistent or not a directory
» cd C:\MATLAB\TOOLBOX\
» cd
C:\MATLAB\TOOLBOX
```

Для указания пути к текущему каталогу может использоваться функция **pwd**:

```
» pwd
ans =
C:\MATLAB\TOOLBOX
```

Для получения информации о содержимом текущего каталога используется команда **dir**:

```
» dir
.           dspblks   ident     MATLAB   optim     signal    tour
..          fdident   images    mpc      pde       simulink  wavelet
comm        finance   java      mutools  powersys  splines
compiler    fixpoint  lmi       nag      qft       stateflow
control     fuzzy     local     ncd      robust    stats
database    hosa      map       nnet     rtw       symbolic
```

Обратите внимание, что в последнем примере выведено содержимое одного из самых важных каталогов системы MATLAB — TOOLBOX. В нем содержится 36 подкаталогов с хранящимися в них пакетами расширения системы MATLAB, например, comm — каталог пакета проектирования средств телекоммуникаций, compiler — компилятор программ в коды языка C, symbolic — символьные (аналитические) вычисления и так далее.

2.8.2. Выполнение команд !, dos, unix и vms

Возможно выполнение команд из командной строки MATLAB для наиболее распространенных операционных систем:

! Команда или **dos Команда** — выполнение заданной команды из MS-DOS.
unix Команда — выполнение заданной команды из операционной системы Unix.
wms Команда — выполнение заданной команды из операционной системы VMS.

Ограничимся примером — запуском программы Norton Commander из окна MATLAB:

```
» dos C:\NC\NC
The Norton Commander ,ґабЁп 5.0,
Copyright (C) 1986 — 1995 by Symantec Corporation.
```

На экране помимо окна MATLAB появится окно запущенной программы (рис. 2.52).

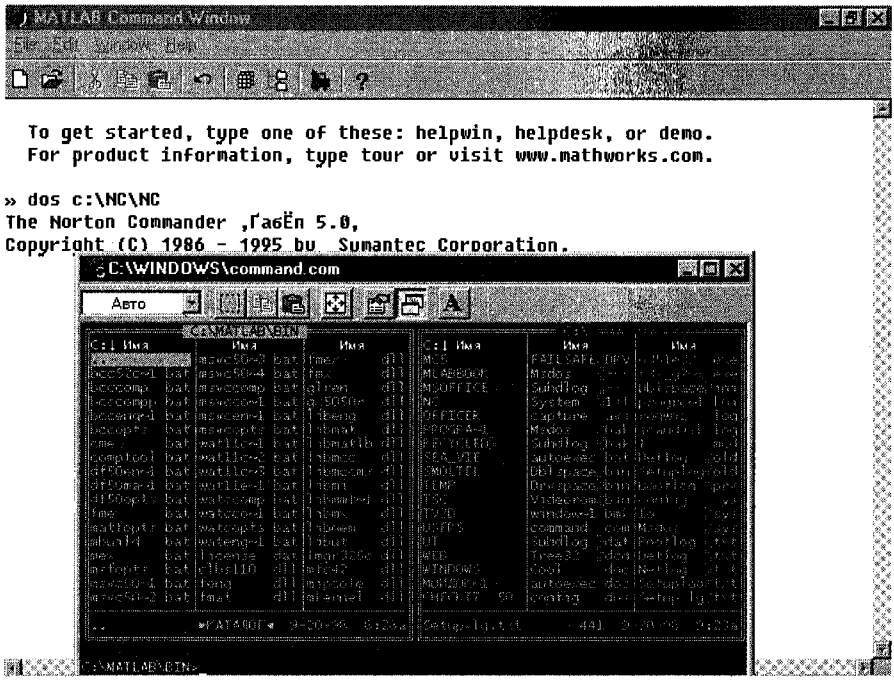


Рис. 2.52. Запуск из окна MATLAB программы Norton Commander

Необходимость в применении данной команды для ПК, ориентированных под операционные системы Windows 95/98, довольно сомнительна. Но тем не менее она есть.

2.8.3. Общение с Internet из командной строки — команда web

Для общения с Internet служит команда:

web Спецификация — дает связь с WEB-сервером.

Примеры на применение команды **web**:

web file:///disk/dir1/dir2/foo.html — открывает файл foo.html на браузере.

web(['file:///' which('foo.html')]) — открывает файл foo.html if it is on the MATLAB path.

web http://www.mathworks.com — загружает WEB- страницу MathWorks Web в браузер.

web mailto:email_address — использует браузер для получения электронной почты.

На рис. 2.53 показан пример на применение команды web для прямого выхода на домашнюю страницу одного из авторов данной книги. Как нетрудно заметить, на экране кроме окна MATLAB появляется вначале пустое окно браузера Internet Explorer 4.0 и окно установки связи с провайдером Internet — ЗАО "Смоленский телепорт".

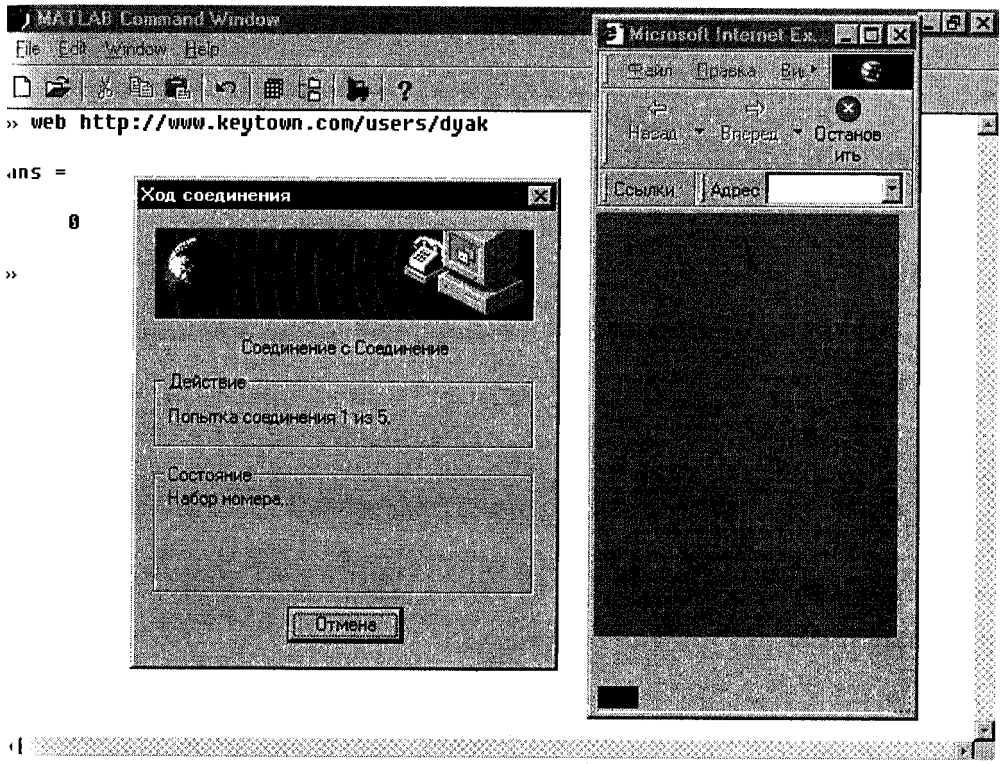


Рис. 2.53. Начало установки связи для загрузки заданной страницы

Спустя некоторое время связь с провайдером устанавливается, проверяются логическое имя и пароль пользователя (стандартная процедура перед входом в Internet), после чего указанная в команде web страница появляется в окне браузера. Это окно можно перетащить в удобное место и изменить в размерах. Завершающая картина установки связи с Internet из командной строки MATLAB представлена на рис. 2.54.

Такой выход в Internet, конечно, иначе чем экзотикой назвать трудно, благо в Windows 95/98 есть куда более простые способы выхода в Internet без необходимости вводить полный адрес загружаемой в браузер страницы в командной строке. Отнесем эту возможность к разряду приятных мелочей, которых в MATLAB очень много.

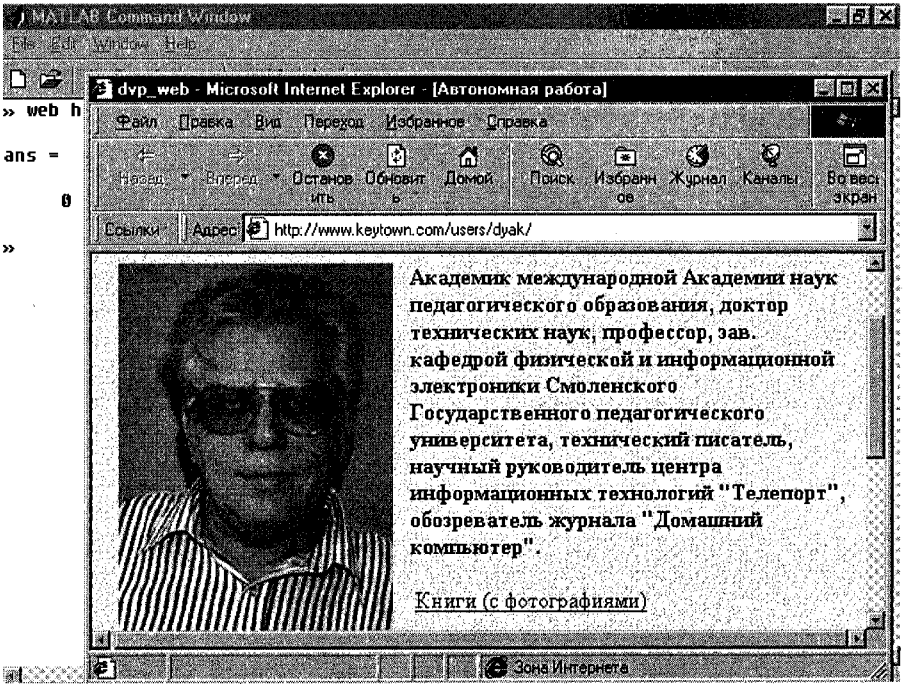


Рис. 2.54. Сеанс работы в Internet из командной строки MATLAB

2.8.4. Некоторые другие команды

Есть еще пара команд для общения с операционными системами:

delete name — стирание файла с заданным именем name (записывается по правилам операционной системы).

getenv('name') — возвращает значение переменной 'name' среды окружения. Пример:

```
» getenv('temp')
ans =
C:\TEMP
```

Команда **tempdir** дает информацию о директории для хранения временных файлов:

```
» tempdir
ans =
C:\TEMP\
```

Еще одна команда **computer** используется в двух формах:

```
» computer
ans =
PCWIN
```

и

```
» [C S]=computer
C =
PCWIN
S =
  268435455
```

Во втором случае помимо сообщения о типе компьютера выводится максимально возможное число элементов в массивах. Оно зависит от объема памяти ПК и свойственных операционной системе ограничений.

Для установки типа терминала может использоваться еще одна команда — **terminal**. Возможные типы терминалов вы найдете в справке по этой команде, выводимой командой **help terminal**. На этом рассмотрение команд прямого общения с операционными системами можно считать законченным.

Глава 3. Работа с приложением Notebook

3.1. Что такое Notebook?

Notebook (Блокнот) — это специальное приложение системы MATLAB, позволяющее готовить с помощью текстового процессора (редактора) Microsoft Word 6.0/7.0/8.0 [51] электронные книги с текстовым описанием и "живыми" примерами. Таким образом, это средство — очередное достижение разработчиков MATLAB в визуализации всех этапов работы с системой.

Как вы могли убедиться из первых глав, основой MATLAB является решатель математических задач с довольно скромным интерфейсом и скромными возможностями стилизации текстов. В части последнего неоспоримым преимуществом обладали текстовые процессоры класса Word, которые позволяли 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 могли видоизменяться при изменении исходных данных описываемых задач. Можно сказать, что текстовые процессоры позволяли готовить обычные "мертвые" книги по математическим расчетам.

Notebook обеспечивает объединение возможностей текстовых процессоров класса Word с возможностями системы MATLAB путем включения в произвольные тексты 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 так далее.

В основе создания 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 выполнять роль сервера и клиента.

3.2. Как создается Notebook?

Создание Notebook 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 в папке NOTEBOOK системы MATLAB.

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 в подменю **Файл** Word новую команду **New M-file**;
- создает новую позицию **Notebook** в главном меню Word;
- поддерживает стили ячеек Notebook и текста Word.

На рис. 3.1 показана подготовка к созданию Notebook. Необходимо запустить текстовый процессор Word (в нашем случае это Word 7.0 или Word 95). В позиции **Файл** главного меню следует исполнить операцию **Открыть...**. Появится окно для поиска нужного файла в файловой системе ПК — это окно на рис. 3.1, оно показано в окне текстового процессора Word. В окне поиска файла показана настройка на загрузку файла readme.doc в папке NOTEBOOK системы MATLAB.

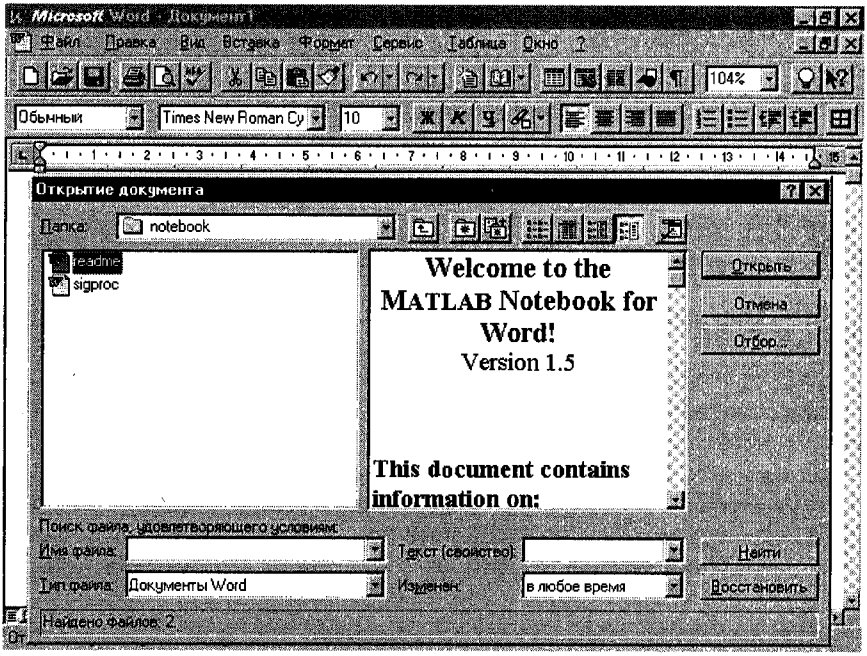


Рис. 3.1. Окна текстового процессора Word и открытия файла

После этого надо нажать клавишу **Открыть** окна открытия файла. Файл readme загрузится в окно редактирования текстового процессора Word и произведет необходимые настройки для работы с документами класса Notebooks (блокноты). Блокноты, подобно обычным блокнотам ученых и инженеров, содержат одновременно текстовые комментарии и формулы описания операций задания исходных данных для MATLAB (ячейки ввода) и результаты их вычислений.

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. 3.2.

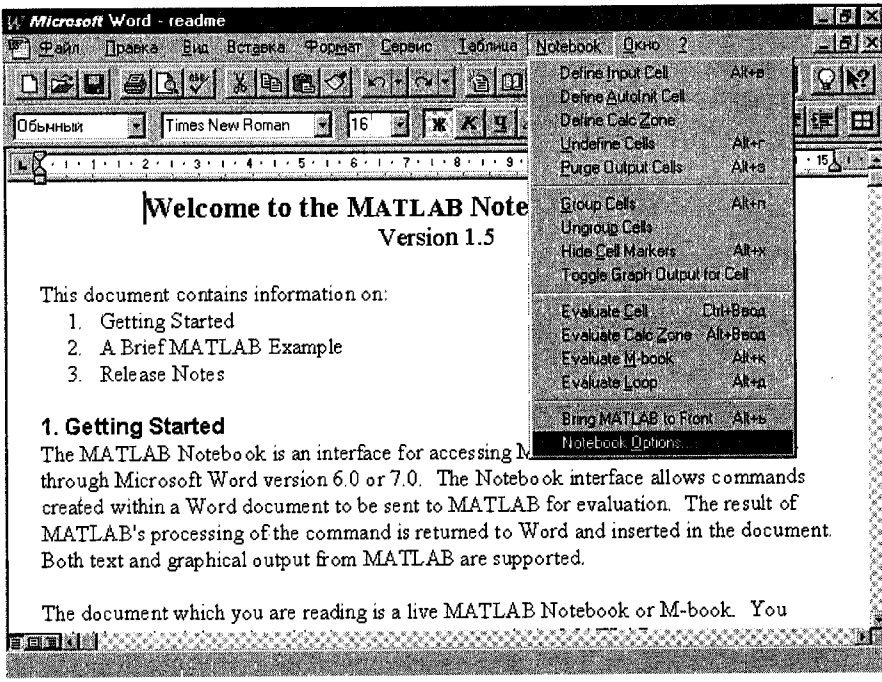


Рис. 3.2. Окно текстового процессора Word с загруженным файлом readme системы MATLAB

Внимательный читатель тут же отметит некоторые необычные свойства окна текстового процессора Word. Первое, что бросается в глаза при сравнении рис. 3.1 и 3.2, это появление в главном меню новой позиции **Notebook**, которая на рис. 3.1 отсутствует. Будучи активной (см. рис. 3.1) эта позиция порождает подменю с обширным списком команд, относящихся к приложению Notebook, созданному на основе текстового процессора Word. Этот список показан на рис. 3.2.

3.3. Демонстрация возможностей Notebook

3.3.1. Эволюция магической матрицы

Файл readme.doc содержит несколько наглядных примеров демонстрации возможностей Notebook. Для оценки этих возможностей достаточно просмотреть файл и остановиться на разделе "A Brief MATLAB Example". Часть документа (см. рис. 3.3), заключенная в жирные квадратные скобки, представляет собой ячейки ввода и вывода, динамически связанные с решателем системы MATLAB.

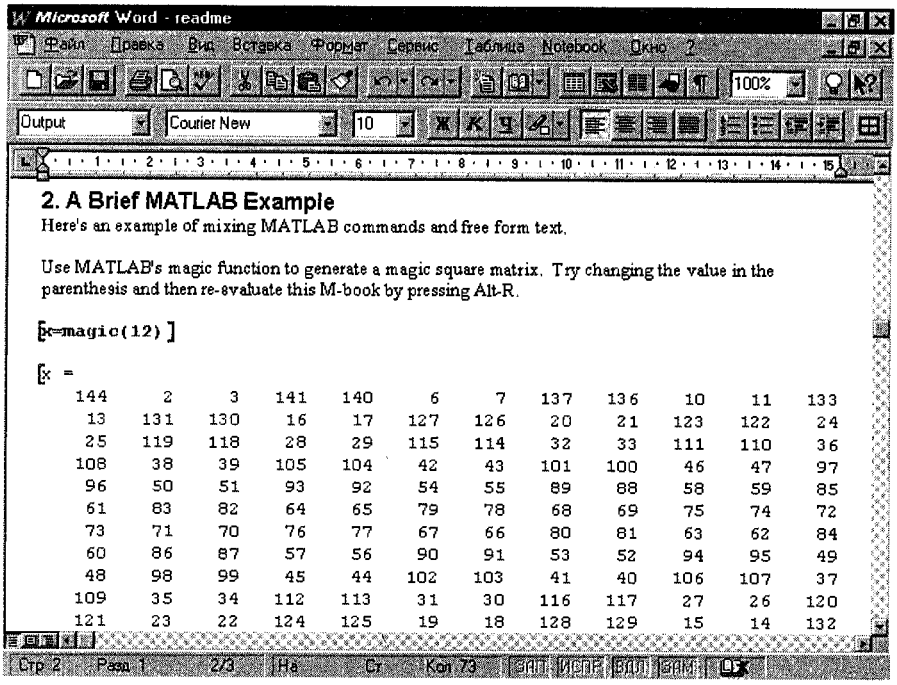


Рис. 3.3. Пример с формированием магической матрицы с размером 12×12

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. 3.2 и 3.3 виден также обычный англоязычный текст, набранный в редакторе Word разным стилем. Напоминаем, что под стилем понимается совокупность параметров текста: используемые наборы шрифтов, их размеры, 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 **эволюции**, то есть изменению. Для этого обычным путем введем маркер ввода в ячейку ввода и заменим параметр функции magic 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 эволюция ячеек Notebook (см. рис. 3.4).

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 по документу в целом.

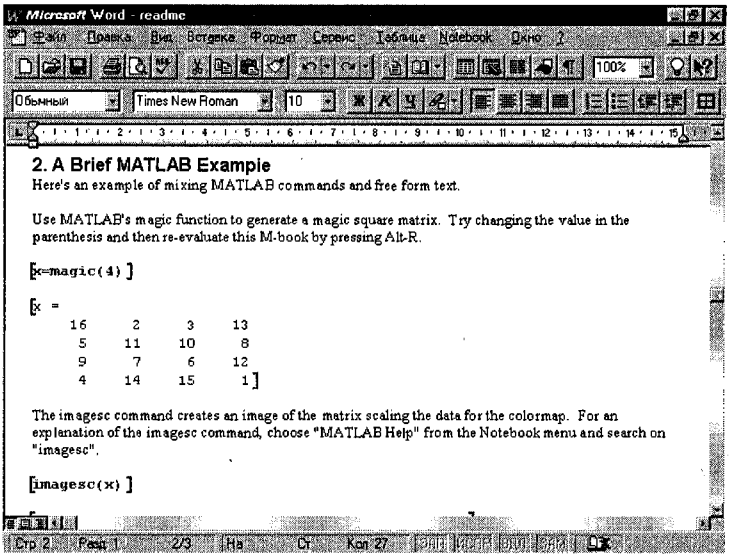


Рис. 3.4. Пример эволюции ячеек — изменение размера магической матрицы

3.3.2. 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. 3.5.

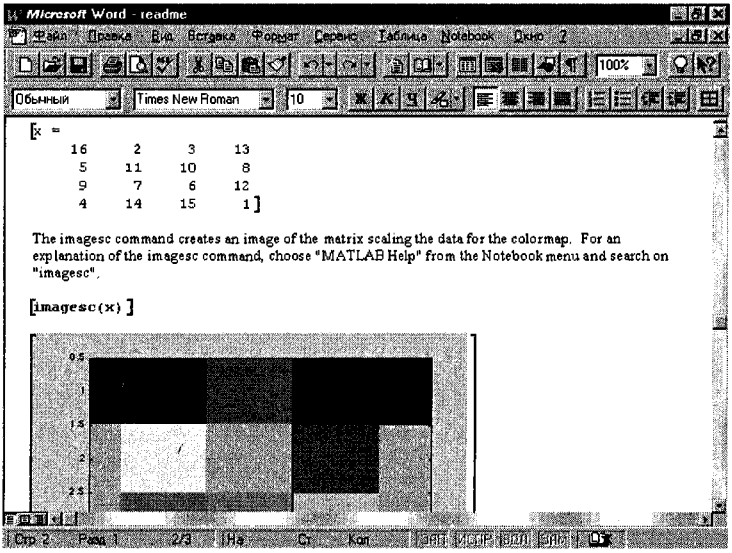


Рис. 3.5. Матрица размера 4×4 и ее графическое представление

Теперь изменим размер матрицы, например на 6×6, и создадим новую матрицу так, как это было описано выше. Далее введем в ячейку с командой images маркер ввода и нажмем клавиши **Ctrl** и **Enter**. На рис. 3.6 видно, как изменилась картинка — число квадратиков на ней возросло. Это и есть пример эволюции рисунка.

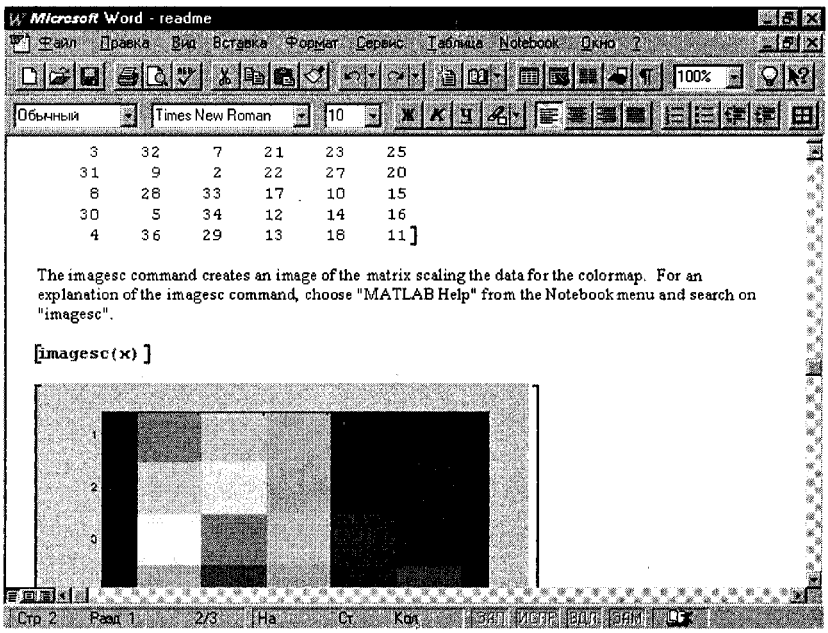


Рис. 3.6. Матрица размера 6×6 и ее графическое представление

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

3.4. Создание новых документов класса Notebooks

3.4.1. Открытие нового документа класса Notebook

Для создания своего Notebook откройте подменю **Файл** текстового процессора при загруженном в него файле readme. Вы обнаружите в подменю новую команду **New M-book** — создание новой М-книги. 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.

3.4.2. Пример создания документа класса Notebook

На рис. 3.7 показан созданный Notebook, который поясняет, как выполнить построение графика трех функций. Вначале в нем введен первый абзац текста с описанием создания ячейки ввода.

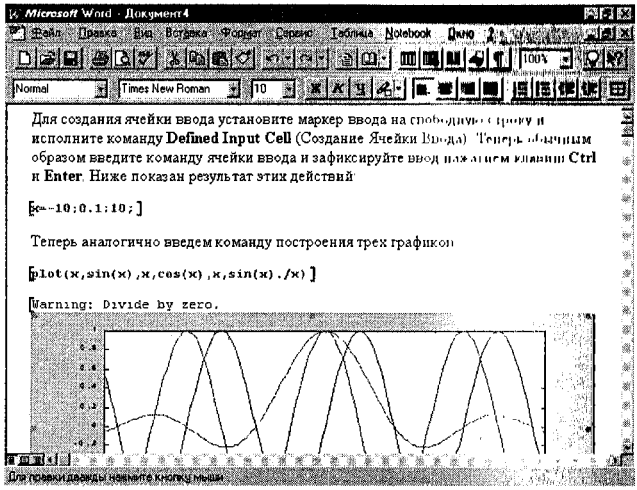


Рис. 3.7. Созданный новый Notebook

Для создания ячейки ввода надо установить маркер ввода на свободную строку и исполнить команду **Defined Input Cell** в позиции **Notebook** главного меню текстового процессора Word. После этого вводится текст команды MATLAB и фиксируется одновременным нажатием клавиш **Ctrl** и **Enter** (или исполнением команды **Evaluate Cell** в позиции **Notebook** главного меню). 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. 3.7). Теперь, после нажатия клавиш **Ctrl** и **Enter**, тут же появится ячейка вывода с графиком трех функций. Установив в нее курсор мыши и щелкнув дважды левой клавишей, можно наблюдать выделение графика и появление в его рамке прямоугольников, за которые можно уцепиться курсором мыши и изменить размеры графика.

3.4.3. Ячейки ввода MATLAB в тексте Word

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

3.4.4. Преобразование текстов Word в ячейки ввода

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

3.4.5. Сохранение документов класса Notebook

Созданный Notebook записывается так же, как любой другой документ Word. Для редактирования уже созданного Notebook достаточно загрузить его с помощью команды **Открыть** позиции **Файл** главного меню.

3.5. Операции и команды позиции Notebook главного меню Word

Возможности создания М-книг или документов класса Notebook намного шире описанных выше. Далее дается полное описание операций и команд позиции **Notebook** главного 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.

3.5.1. Создание ячейки ввода Define Input Cell

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 автовызова (см. ниже) также преобразуется в ячейку ввода, если в ней размещен маркер ввода. Пустая строка с маркером ввода становится ячейкой ввода после введения в нее соответствующего выражения и фиксации его ввода нажатием клавиш **Ctrl** и **Enter**.

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

3.5.2. Создание ячейки автостарта Define AutoInit Cell

Команда **Define AutoInit Cell** создает ячейку автостарта. Это ячейка, которая будет исполняться сразу после загрузки М-книги в текстовый процессор

Word. Здесь 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 кегля 10).

3.5.3. Создание зоны вычислений Define Calc Zone

Команда **Define Calc Zone** оформляет 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. Иллюстрацией того, где такие зоны полезны, являются сборники различных задач с «живыми» примерами.

3.5.4. Преобразование MATLAB-ячеек в обычный текст Undefine Cells

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

3.5.5. Удаление ячеек вывода Purge Output Cell

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

3.5.6. Создание многострочной ячейки ввода Group Cells

Команда **Group Gells** объединяет все ячейки ввода 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**.

3.5.7. Преобразование группы ячеек в ячейки ввода Ungroup Cells

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

3.5.8. Управление показом маркеров Hide/Show Cell Markers

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 или нет на экране дисплея.

3.5.9. Пуск эволюции ячеек Evaluate Cell

Команда **Evaluate Cell** направляет текущую ячейку ввода или группу ячеек в решатель MATLAB для проведения необходимых вычислений или обработки данных. Результат, в том числе и в виде сообщений об ошибках (они выдаются 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.

3.5.10. Пуск эволюции зоны Evaluate Calc Zone

Команда **Evaluate Calc Zone** задает эволюцию текущей зоны вычислений. Зона считается текущей, если в ней размещен маркер ввода. Для каждой ячейки ввода в зоне создается ячейка вывода.

3.5.11. Пуск эволюции всей М-книги — Evaluate M-book

Команда **Evaluate M-book** задает эволюцию всех ячеек ввода или групп ячеек для текущей М-книги. Текущей является та книга, текст которой (или его часть) видна 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 до ее конца. Результаты вычислений поступают в ячейки вывода, а если каких-то из них 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.

3.5.12. Циклическая эволюция Evaluate Loop

Выделенную ячейку ввода или группу ячеек можно выполнить циклически с помощью команды **Evaluate Loop**. Рис. 3.8 показывает подготовку к такой эволюции. В первой строке ввода задано значение переменной **i=0**. Во второй строке задано вычисление **i=i+1** и **magic(i)**. Первоначально, таким образом, выдастся магическая матрица размера 1×1.

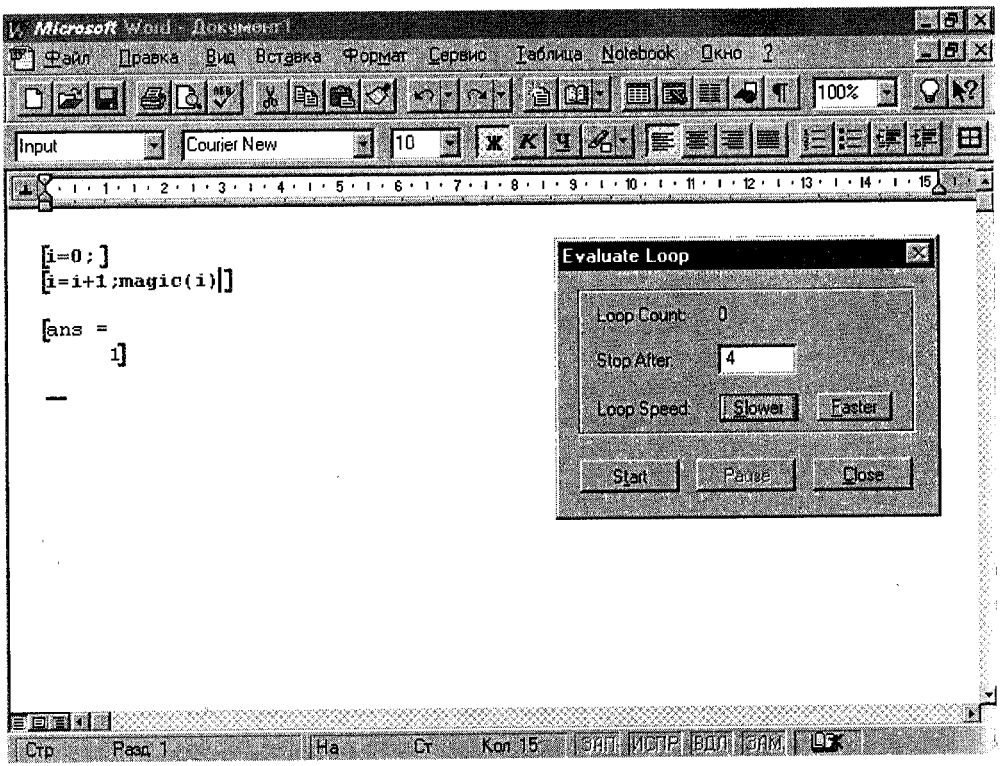


Рис. 3.8. Подготовка к циклической эволюции заданной ячейки ввода

В правой части окна рис. 3.8 видно окошко Evaluate Loop, которое появляется при исполнении команды **Evaluate Loop**. В нем имеется поле Stop After (Остановить После), задающее число циклов эволюции. Кнопки **Slower** и **Faster** задают медленную (условно) и быструю эволюцию текущей ячейки ввода — в нашем случае это вторая ячейка из двух, показанных на рис. 3.8.

Запустив циклический просмотр нажатием кнопки **Start,** можно наблюдать построение матриц постоянно увеличивающегося размера i×i. По завершении циклов можно увидеть результаты, показанные на рис. 3.9.

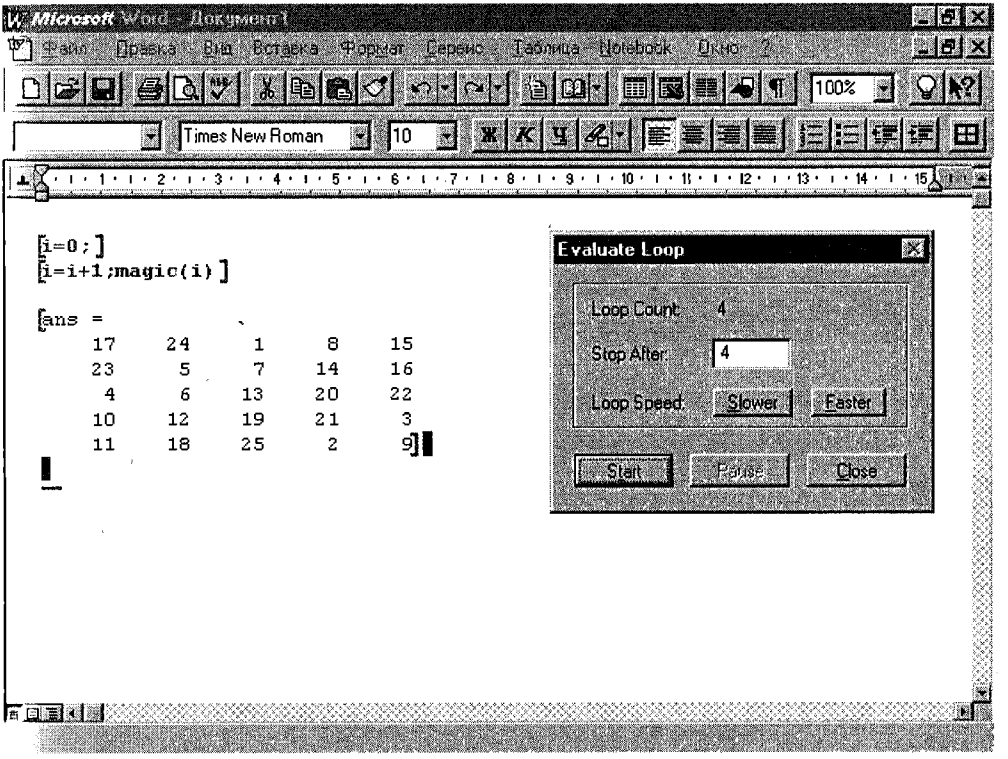


Рис. 3.9. Конец циклической эволюции

Можно приостановить циклическую эволюцию (если успеете!) нажатием кнопки **Pause**. А чтобы убрать окно контроля циклической эволюции, следует нажать его кнопку **Close**.

3.5.13. Вывод окна MATLAB на передний план Bring MATLAB to Front

Команда **Bring MATLAB to Front** выводит на передний план командное окно MATLAB (рис.3.10).

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

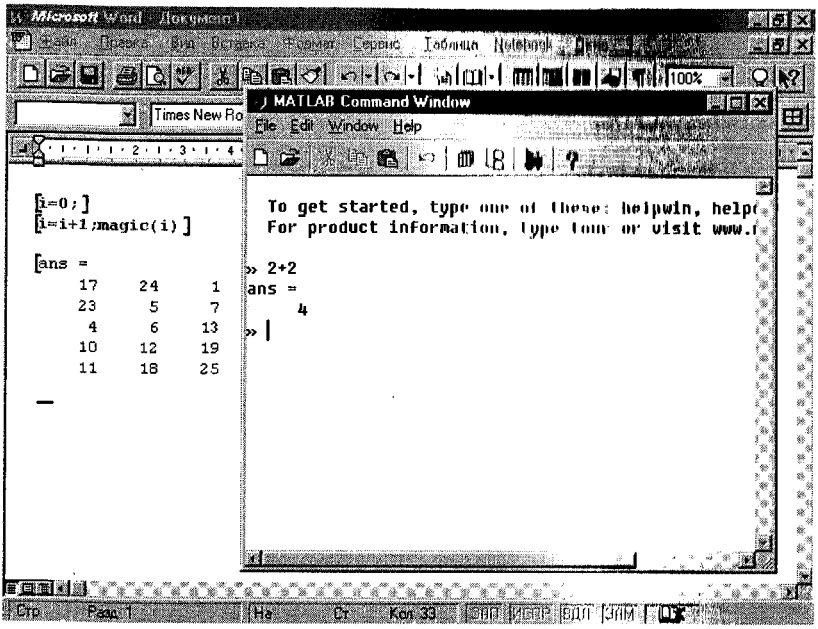


Рис. 3.10. Пример вывода окна системы MATLAB на передний план

3.5.14. Установка опций Notebook Options

Последняя позиция подменю **Notebook — Notebook Options** выводит окно установки опций, в основном связанных с параметрами вывода результатов вычислений. Это окно показано на рис. 3.11.

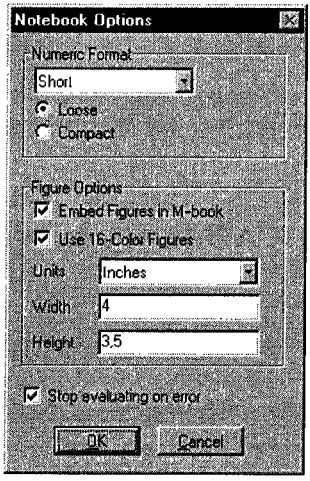


Рис. 3.11. Окно установки опций Notebook

Окно содержит две панели. Первая панель "Numetic Format" позволяет задать вывод чисел в различных форматах. Для изменения формата служит переключатель, выводящий список возможных форматов. Форматы представления чисел мы обсуждали в разделе 1.4.9. 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 галочки у опции Embed Figures in M-book обеспечивает размещение рисунков прямо в тексте книги — как разновидность ячеек вывода. Отсутствие галочки означает, что рисунки будут выведены в отдельных окнах. 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 опции Width и Height.

Дополнительная опция Stop Evaluating on error останавливает эволюцию при возникновении ошибки. Если эта опция отключена, то эволюция продолжается даже при наличии ошибки.

Глава 4.
ГРАФИЧЕСКИЕ СРЕДСТВА СИСТЕМЫ MATLAB

Одно из достоинств системы MATLAB - обилие средств графики, начиная от команд построения простых графиков функций одной переменной в Декартовой системе координат и кончая комбинированными и презентационными графиками с элементами анимации, а также средствами проектирования графического пользовательского интерфейса GUI. Особое внимание в системе уделено трехмерной графике с функциональной окраской отображаемых фигур и имитацией различных световых эффектов.

Описанию графических функций и команд посвящена обширная электронная книга в формате PDF. Поэтому в данной главе приводятся лишь наиболее важные и часто используемые графические функции и команды. Описание дескрипторной графики, ориентированной на внутренние средства системы, и применение ее разработчиками MATLAB и разработчиками других систем на базе MATLAB дано в аспекте ознакомления.

Данная глава намеренно предшествует систематизированному описанию большинства функций системы MATLAB, поскольку графическая визуализация вычислений довольно широко используется в последующих материалах книги. При этом графические средства системы доступны как в командном режиме вычислений, так и в программах.

4.1. Интерфейс графических окон

4.1.1. Общие возможности графики

Начиная с версии MATLAB 4.0, впервые ориентированной на Windows, графические средства системы были существенно улучшены. Основные отличительные черты новой графики:

- возможность создания графики в отдельных окнах;
- возможность вывода многих окон;
- возможность перемещения окон по экрану и изменения их размеров;
- задание различных координатных систем и осей;
- высокое качество графики;
- широкие возможности использования цвета;
- легкость установки графических признаков — атрибутов;
- снятие ограничений на число цветов;
- обилие опций у команд графики;
- возможность получения естественно выглядящих трехмерных фигур и их сочетаний;

- простота построения 3D-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;
- обширный набор команд графики, как говорится, "на все случаи жизни";
- 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 вы не работаете с объектно-ориентированным программированием задач графики. Связано это с тем, что большинство команд графики, ориентированной на конечного пользователя, автоматически устанавливает свойства графических объектов и обеспечивает воспроизведение графики в нужной системе координат, палитре цветов, масштабе и так далее.

На более низком уровне решения задач используется **дескрипторная графика** (Handle Graphics), в 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 так далее.

Эти обширные возможности делают графику MATLAB одной из лучших среди математических систем. Несмотря на обилие графических команд их синтаксис достаточно прост и легко усваивается даже начинающими пользователями. Руководствуясь правилом описания "от простого к сложному", мы рассмотрим сначала графику функций одной переменной, а затем уже трехмерную графику, специальную, анимационную и, наконец, дескрипторную.

Хотя данная книга не предусматривает исчерпывающе полное описание всех команд системы MATLAB, большинство команд графики будет рассмотрено с примерами, которые можно считать дополнительными к тем, что приведены в документации по системе.

4.1.2. Описание графического окна

Графическое окно MATLAB представлено на рис. 4.1. Это обычное масштабируемое и перемещаемое окно Windows- приложений.

Главное меню этого окна аналогично главному меню окна командного режима работы системы MATLAB, за исключением того, что в позиции **Edit** главного меню окна графики есть новая команда **Copy Figure**, которая служит для копирования рисунка в буфер Clipboard.

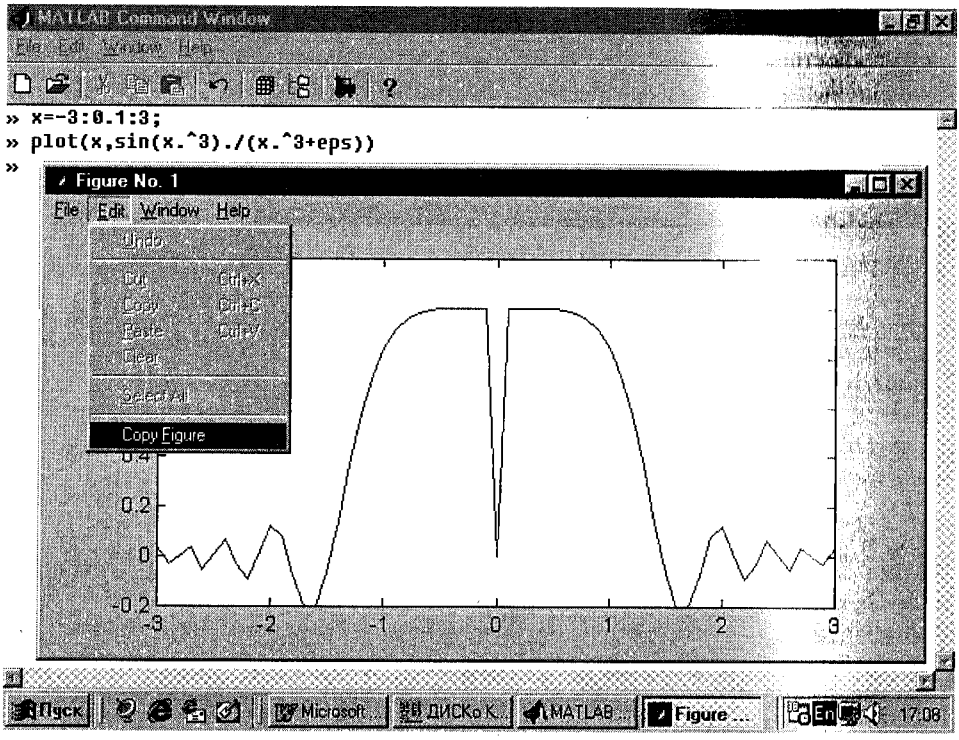


Рис. 4.1. Графическое окно MATLAB

4.2. Графики двумерных функций

4.2.1. Построение графиков отрезками прямых — команда plot

Функции одной переменной y(x) находят широкое применение в практике математических и других расчетов, а также в технике компьютерного математического моделирования. Для отображения таких функций используются графики в Декартовой (прямоугольной) системе координат. При этом обычно строятся две оси — горизонтальная X и вертикальная Y — и задаются координаты x и y, определяющие узловые точки y(x). Эти точки соединяются друг с другом отрезками прямых. Поскольку MATLAB представляет собой матричную систему, совокупность точек y(x) задается векторами **X** и **Y** одинакового размера.

Команда **plot** служит для построения графиков функций в Декартовой системе координат. Эта команда имеет ряд параметров, рассматриваемых ниже.

plot(X,Y) — строит график функции y(x), координаты точек (x,y) которой берутся из векторов одинакового размера **Y** и **X**. Если **X** или **Y** — матрица, то строятся гра-

фики по данным в матрице. Приведенный ниже пример иллюстрирует построение графиков двух функций — sin(x) и cos(x), данные которых содержатся в матрице **Y**, а значения x хранятся в малоразмерном векторе **X**:

```
» % Построение графиков двух функций с вектором значений X
» x=[0 1 2 3 4 5];
» Y=[sin(x);cos(x)];
» plot(x,Y)
```

На рис. 4.2 показан график функций из этого примера. В данном случае отчетливо видно, что график состоит из отрезков прямых, и если вам нужно, чтобы отображаемая функция имела вид гладкой кривой, необходимо увеличить количество узловых точек.

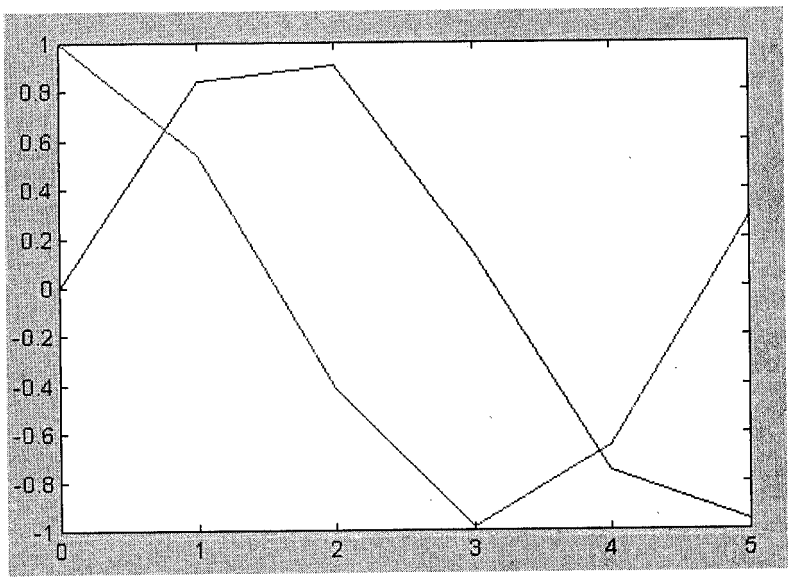


Рис. 4.2. Графики двух функций в Декартовой системе координат

plot(Y) — строит график y(i), где значения y берутся из вектора **Y**, а i представляет собой индекс соответствующего элемента. Если **Y** содержит комплексные элементы, то выполняется команда **plot(real(Y), imag(Y))**. Во всех других случаях мнимая часть данных игнорируется.

Вот пример использования команды **plot(Y)**:

```
» %График plot(y) при векторе y с комплексными элементами
» x=-2*pi:0.02*pi:2*pi;
» y=sin(x)+i*cos(3*x);
» plot(y)
```

Соответствующий график показан на рис. 4.3.

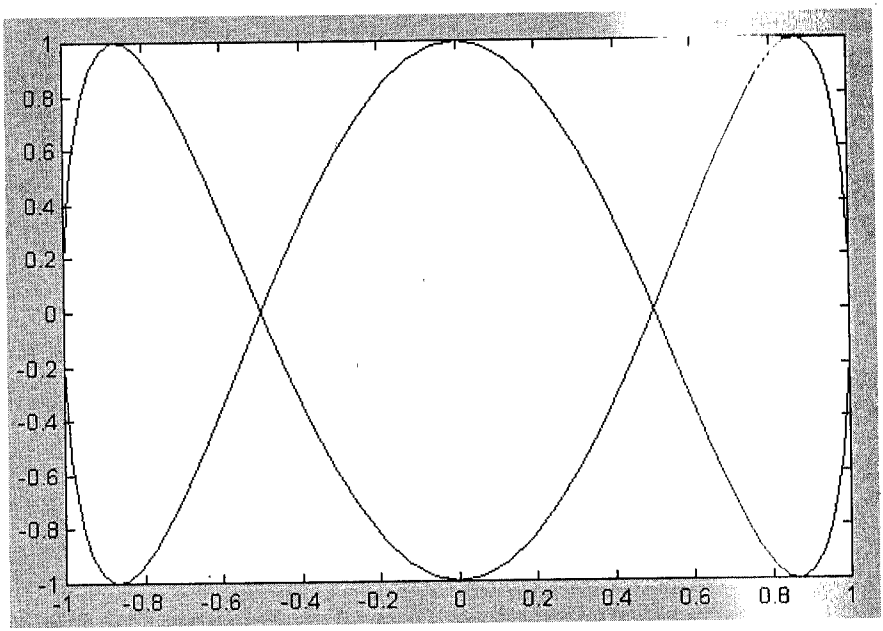


Рис. 4.3. График функции, представляющей вектор **Y** с комплексными элементами

plot(X,Y,S) — аналогична команде **plot(X,Y)**, но тип линии графика можно задавать с помощью строковой константы S. При этом значениями константы могут быть следующие символы:

Цвет линии		Тип точки		Тип линии	
y	желтый	**.**	точка	**-**	сплошная
m	фиолетовый	**o**	окружность	**:**	двойной пунктир
c	голубой	**x**	крест	**-.**	штрих-пунктир
r	красный	**+**	плюс	**--**	штриховая
g	зеленый	*****	звездочка		
b	синий	**s**	квадрат		
w	белый	**d**	ромб		
k	черный	**v**	треугольник (вниз)		
		^	треугольник (вверх)		
		<	треугольник (влево)		
		>	треугольник (вправо)		
		p	пятиугольник		
		h	шестиугольник		

Таким образом, с помощью строковой константы S можно изменять цвет линии, представлять узловые точки различными отметками (точка, окружность, крест, треугольник с разной ориентацией вершины и так далее) и менять тип линии графика.

plot(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...) — эта команда строит на одном графике ряд линий, представленных данными вида **(X',Y',S')**, где **X'** и **Y'** векторы или матрицы, а **S'** — строки. С помощью такой конструкции возможно построение, например, графика функции линией, цвет которой отличается от цвета узловых точек. Так, если надо построить график функции линией синего цвета с красными точками, то вначале надо задать построение графика с точками красного цвета, а затем графика только линии синего цвета.

При отсутствии указания на цвет линий и точек он выбирается автоматически одним из шести цветов из таблицы цветов (белый исключается). Если линий больше шести, то выбор цветов повторяется. Для монохромных систем линии выделяются стилем.

Рассмотрим пример построения графиков трех функций с различным стилем представления каждой из них:

```
» %Графики трех функций со спецификацией линий каждого графика
» x=-2*pi:0.1*pi:2*pi;
» y1=sin(x);
» y2=sin(x).^2;
» y3=sin(x).^3;
» plot(x,y1,'-m',x,y2,'-.+r',x,y3,'--ok')
```

Графики функций для этого примера показаны на рис. 4.4.

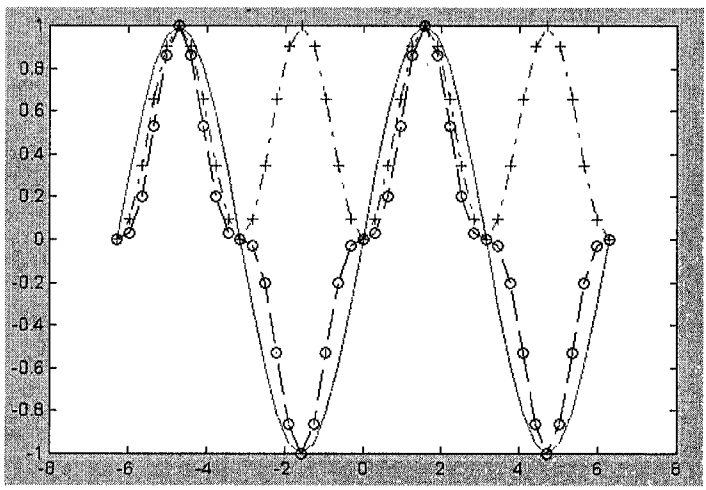


Рис. 4.4. Построение графиков трех функций на одном рисунке с разным стилем каждого графика

Здесь график функции y1 строится сплошной фиолетовой линией, график y2 строится штрих-пунктирной линией с точками в виде знака + красного цвета, а график y3 строится штриховой линией с кружками черного цвета.

4.2.2. Графики функции в логарифмическом масштабе — команда loglog

Для построения графиков функций со значениями x и y, изменяющимися в широких пределах, нередко используются логарифмические масштабы. Рассмотрим команду, которая используются в таких случаях.

loglog(...) — синтаксис команды аналогичен ранее рассмотренному для функции **plot(...)**. Логарифмический масштаб используется для координатных осей X иY. Ниже приведен пример применения данной команды:

```
» % График экспоненциальной функции в логарифмическом масштабе
» x=logspace(-1,3);
» loglog(x,exp(x)./x)
» grid on
```

На рис. 4.5 представлен график функции exp(x)/x в логарифмическом масштабе. Обратите внимание на то, что по умолчанию используется координатная мерная сетка.

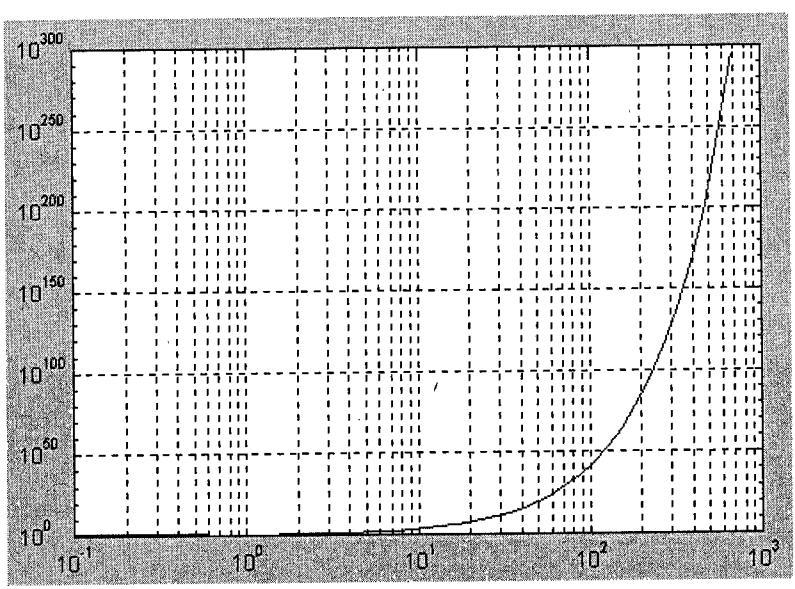


Рис. 4.5. График функции exp(x)/x в логарифмическом масштабе

Неравномерное расположение линий координатной мерной сетки указывает на логарифмический масштаб осей.

4.2.3. График функции в полулогарифмическом масштабе — команды semilogx и semilogy

В некоторых случаях предпочтителен полулогарифмический масштаб графиков, когда по одной оси задается логарифмический масштаб, а по другой — линейный. Для построения графиков функций в полулогарифмическом масштабе используются следующие команды:

semilogx(...) — строит график функции в логарифмическом масштабе (основание 10) по оси X и линейном по оси Y.

semilogy(...) — строит график функции в логарифмическом масштабе по оси Y и линейном по оси X.

Запись параметров (...) выполняется по аналогии с функцией plot(...). Ниже приводится пример построения графика экспоненциальной функции:

```
» %График экспоненциальной функции в полулогарифмическом масштабе
» x=0:0.5:10;
» semilogy(x,exp(x))
```

График функции при логарифмическом масштабе по оси Y представлен на рис. 4.6.

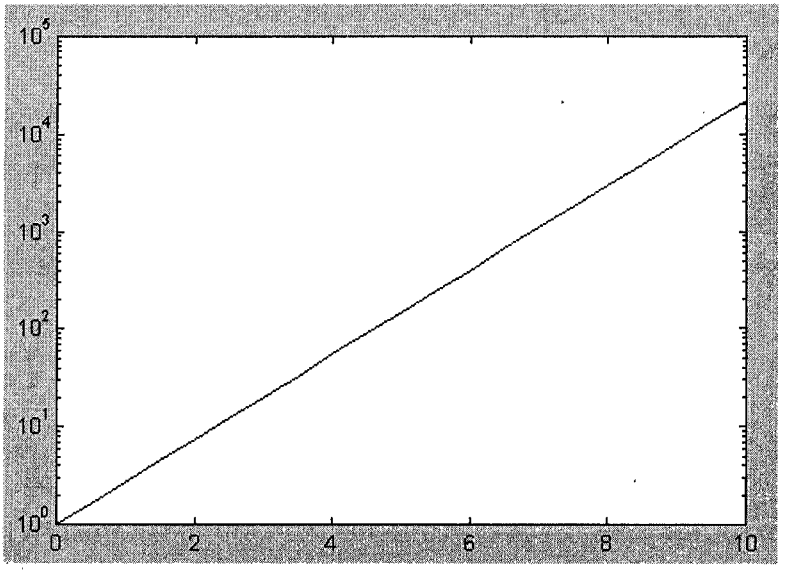


Рис. 4.6. График экспоненты в полулогарифмическом масштабе

Нетрудно заметить, что при таком масштабе график экспоненциальной функции выродился в прямую линию.

4.2.4. Столбиковые диаграммы — команда bar

Столбиковые диаграммы широко используются в литературе, посвященной финансам и экономике, а также в математической литературе. Ниже представлены варианты команды для построения столбиковых диаграмм.

bar(X,Y) — строит столбиковый график элементов массива **Y** в позициях, определяемых вектором X с упорядоченными в порядке возрастания значений элементами. Если **X** и **Y** двумерные массивы одинакового размера, то столбцы строятся попарно с надстройкой друг на друге.

bar(Y) — строит график значений элементов одномерного массива **Y**. Фактически это предшествующая команда для **X=1:M**.

bar(X,Y,WIDTH) или **BAR(Y,WIDTH)** — команда аналогична ранее рассмотренным, но со спецификацией ширины столбцов (при WIDTH > 1 столбцы перекрываются). По умолчанию задано WIDTH = 0.8

Возможно применение этих команд и в следующем виде:

bar(...,'Спецификация')

для задания спецификации графиков, например, типа линий, цвета и так далее, по аналогии с командой plot.

Пример построения столбиковой диаграммы приводится ниже:

```
» % Столбиковая диаграмма с вертикальными столбцами
» subplot(2,1,1), bar(rand(12,3),'stacked'), colormap(cool)
```

На рис. 4.7 представлен полученный график.

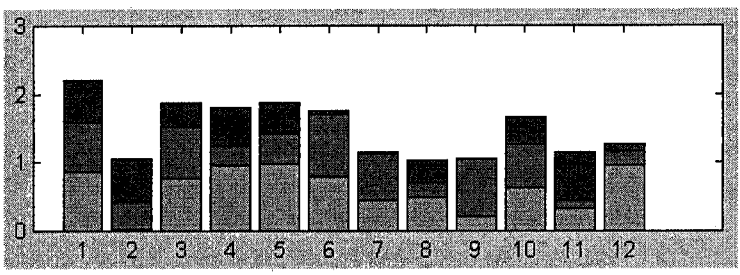


Рис. 4.7. Пример построения диаграммы с вертикальными столбцами

Помимо команды **bar(...)** существует аналогичная ей по синтаксису команда

barh(...)

которая строит столбиковые диаграммы с горизонтальным расположением столбцов.

Пример, приведенный ниже

```
» % Столбиковая диаграмма с горизонтальными столбцами
» subplot(2,1,1), barh(rand(5,3),'stacked'), colormap(cool)
```

дает построения, показанные на рис. 4.8.

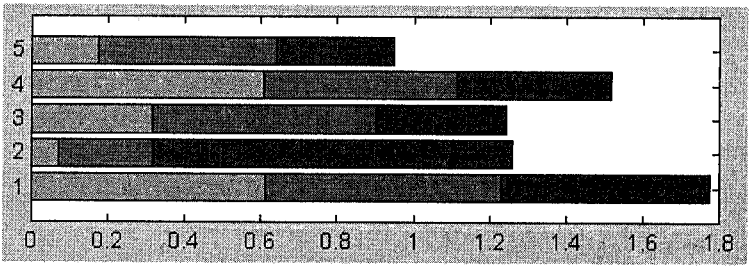


Рис. 4.8. Пример построения столбиковой диаграммы с горизонтальными столбцами

Какое именно расположение столбцов выбрать, зависит от пользователя, применяющего эти команды для представления своих данных.

4.2.5. График типа гистограммы — команда hist

Классическая гистограмма характеризует числа попаданий значений элементов вектора **Y** в **M** интервалов с представлением этих чисел в виде столбиковой диаграммы. Для получения данных для гистограммы служит функция **hist**, записываемая в виде:

N=hist(Y) -возвращает вектор числа попаданий для 10 интервалов. Если **Y** — матрица, то выдаются данные по числу попаданий в ее столбцы.

N = hist(Y,M) — аналогична выше рассмотренной, но для числа попаданий M (скаляр).

N=hist(Y,X) — возвращает число попаданий для элементов вектора Y с центральными отсчетами, заданными элементами вектора X.

[N,X] = HIST(...) — возвращает числа попаданий в интервалы и данные о центрах интервалов.

Команда

hist(...)

с синтаксисом (...), аналогичным приведенному выше, строит график гистограммы. В следующем примере строится гистограмма для 1000 случайных чисел и выводится вектор с данными о числах их попаданий в интервалы, заданные вектором **x**:.

```
» %Построение гистограммы для 1000 случайных чисел
» x=-3:0.2:3;
» y=randn(1000,1);
» hist(y,x)
» h=hist(y,x)
h =
 Columns 1 through 12
   0    0    3    7    8    9   11   23   33   43   57   55
 Columns 13 through 24
  70   62   83   87   93   68   70   65   41   35   27   21
 Columns 25 through 31
  12    5    6    3    2    1    0
```

Построенная гистограмма показана на рис. 4.9.

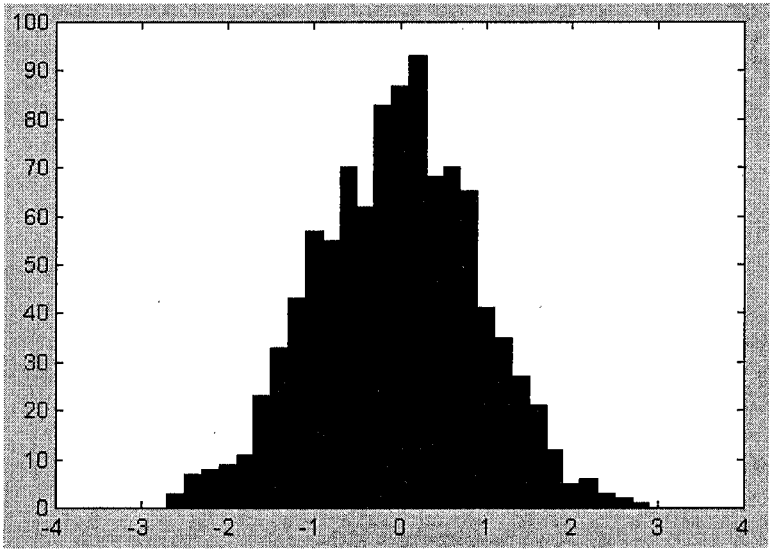


Рис. 4.9. Пример построения графика гистограммы

Как можно заметить, распределение случайных чисел близко к нормальному закону. Увеличив их количество, можно наблюдать еще большее соответствие этому закону.

4.2.6. Лестничные графики — команда stairs

Лестничные графики визуально представляют собой ступеньки с огибающей, представленной функцией y(x). Такие графики используются, например, для отображения процессов квантования функции y(x), представленной рядом своих отсчетов. При этом в промежутках между отсчетами значения функции считаются постоянными и равными величине последнего отсчета.

Для построения лестничных графиков в системе MATLAB используются варианты команды группы **stairs**:

stairs(Y) — строит лестничный график по данным вектора **Y**.
stairs(X,Y) — строит лестничный график по данным вектора **Y** с координатами x переходов от ступеньки к ступеньке заданными значениями элементов вектора **X** (они должны идти в возрастающем порядке).
stairs(...,S) — аналогична по действию вышеописанным командам, но строит график линиями, стиль которых задается строками S.

Следующий пример иллюстрирует построение лестничного графика:

```
» %Лестничный график функции x^2
» x=0:0.25:10;
» stairs(x,x.^2);
```

Результат построения представлен на рис. 4.10.

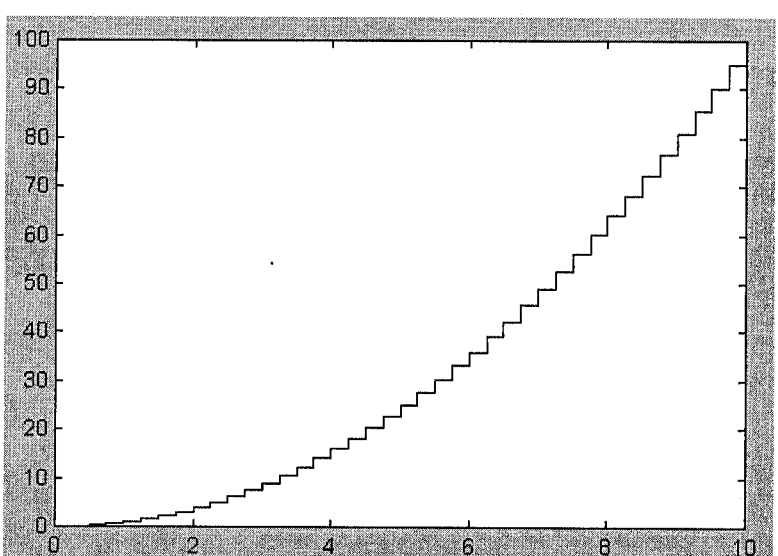


Рис. 4.10. График функции x^2 типа stairs

Обратите внимание на то, что отсчеты берутся через равные промежутки по горизонтальной оси. Если, к примеру, отображается функция времени, то **stairs** дает вид квантованной по времени функции.

Функция

H=stairs(X,Y)

возвращает вектор координат, а функция

[XX,YY]=stair(X,Y)

сама по себе график не строит, но возвращает векторы **XX** и **YY**, которые позволяют построить график с помощью команды **plot(XX,YY)**.

4.2.7. Графики функций типа errorbar

Если данные для построения функции определены с заметной погрешностью, то используют графики функций типа errorbar с оценкой погрешности каждой точки путем ее представления в виде буквы I, высота которой соответствует погрешности представления точки. Команда **errorbar** используется в виде:

errorbar(X,Y,L,U) — строит график значений элементов вектора **Y** в зависимости от данных в векторе **X** с указанием нижней и верхней границ значений, заданных в векторах **L** и **U**.

errorbar(X,Y,E) и **errorbar(Y,E)** — строит графики функции **Y(X)** с указанием этих границ в виде **[Y-E Y+E]**, где E — погрешность.

errorbar(...,'LineSpec') — аналогична описанным выше командам, но позволяет строить линии со спецификацией **'LineSpec'**, аналогичной спецификации, примененной в команде **plot**.

Следующий пример иллюстрирует применение команды **errorbar**:

```
» График функции с указанием погрешности каждой точки
» x=-2:0.1:2;
» y=erf(x);
» e = rand(size(x))/10;
» errorbar(x,y,e);
```

Построенный график показан на рис. 4.11.

Функция, записываемая в виде

H = ERRORBAR(...)

возвращает вектор координат, используемых командой **errorbar**.

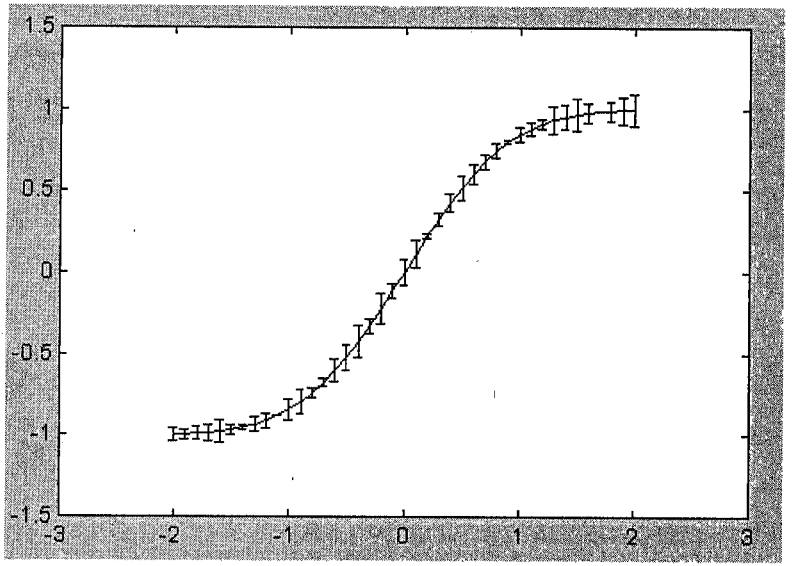


Рис. 4.11. График функции erf(x) типа errorbar

4.2.8. График дискретных отсчетов функции — команда stem

Еще один вид графика функции y(x) — ее представление дискретными отсчетами. Этот вид графика применяется, например, при описании квантования сигналов. Каждый отсчет представляется вертикальной чертой, увенчанной кружком, причем высота черты соответствует координате y точки.

Для построения графика подобного вида используются варианты команды **stem(...)**:

stem(Y) — строит график функции с ординатами в векторе **Y** в виде отсчетов.
stem(X,Y) — строит график отсчетов с ординатами в векторе **Y** и абсциссами в векторе **X**.
stem(...,'filled') — строит график функции с закрашенными маркерами.
stem(...,'LINESPEC') — дает построения, аналогичные ранее приведенным командам, но со спецификацией линий, подобной спецификации, приведенной для функции plot.

Следующий пример иллюстрирует применение команды **stem**.

```
» %График дискретных отсчетов функции
» x = 0:0.1:4;
» y = sin(x.^2).*exp(-x);
» stem(x,y)
```

Полученный для данного примера график показан на рис. 4.12 .

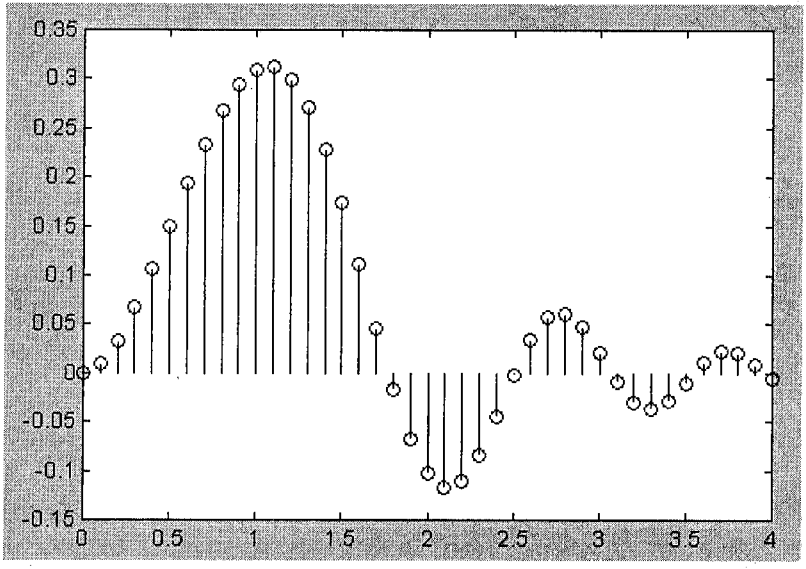


Рис. 4.12 . График функции типа **stem**

Функция

H = STEM(...)

возвращает вектор высот отсчетов.

4.3. Графики в полярной системе координат

4.3.1. Графики функций в полярной системе координат — команды polar

В полярной системе координат любая точка представляется как конец радиус-вектора, исходящего из начала системы координат, имеющего длину RHO и угол THETA. Для построения графика функции RHO(THETA) используются приведенные ниже команды. Угол THETA обычно меняется от 0 до 2*pi.

Для построения графиков функций в полярной системе координат используются команды типа **polar(...)**:

polar(THETA, RHO) строит график в полярной системе координат, представляющий собой положение конца радиус-вектора с длиной **RHO** и углом **THETA**.

polar(THETA,RHO,S) аналогична предыдущей команде, но позволяет задавать стиль построения с помощью строковой константы S по аналогии с командой plot.

Рис. 4.13 демонстрирует результат выполнения команд:

```
» %График функции sin(5*t) в полярной системе координат
» t=0:pi/50:2*pi;
» polar(t,sin(5*t))
```

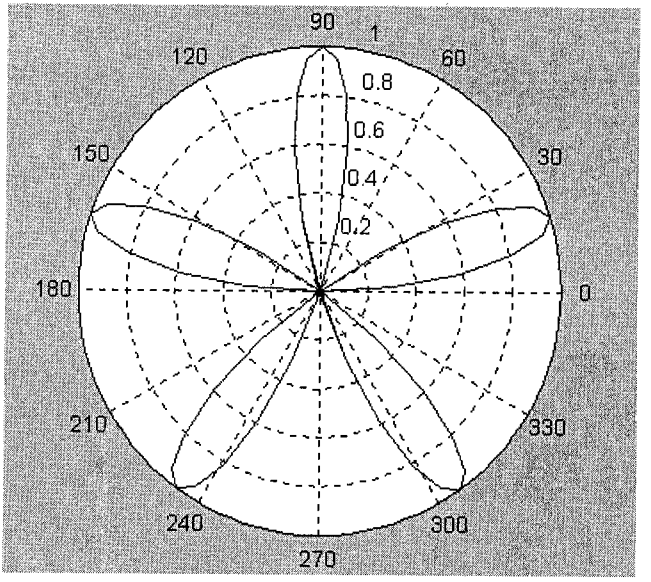


Рис. 4.13. График функции в полярной системе координат

Графики функций в полярных координатах могут иметь весьма разнообразный вид, порой напоминая такие объекты природы, как снежинки или кристаллики льда на стекле. Вы можете сами попробовать построить несколько таких графиков — многие получают от этого удовольствие.

4.3.2. Угловые гистограммы — команда rose

Угловые гистограммы находят применение в индикаторах радиолокационных станций, для отображения "розы ветров" и при построении других специальных графиков. Для этого используется команда типа **rose(...)**:

rose(THETA) — строит угловую гистограмму для 20 интервалов и данных вектора **THETA**.

rose(THETA,N) — строит угловую гистограмму для N интервалов в пределах угла от **0** до **2*pi** по данным вектора **THETA**.

rose(THETA,X) — строит угловую гистограмму по данным вектора **THETA** со спецификацией в векторе **X**.

Следующий пример иллюстрирует применение команды **rose**:

```
»% Пример построения угловой гистограммы
» rose(1:100,12)
```

На рис. 4.14 показан пример построения графика командой **rose**:

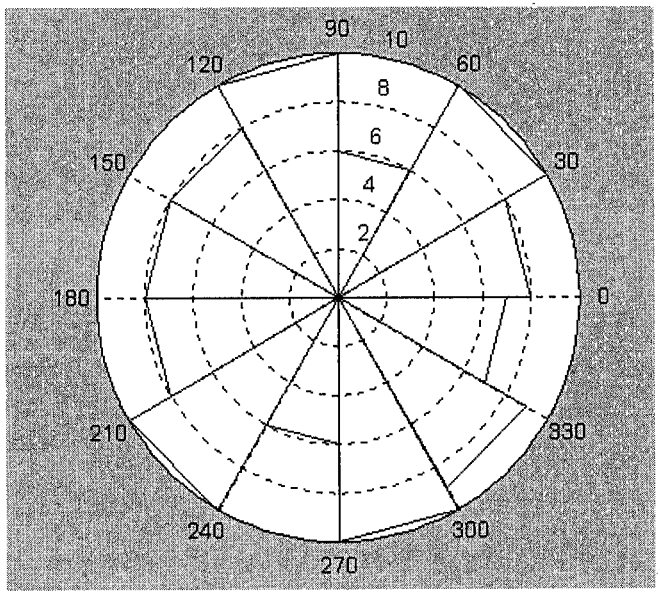


Рис. 4.14. График вида "угловая гистограмма"

Функция

H=rose(...)

возвращает вектор длин элементов гистограммы, а функция

[T,R]=rose(...)

сама по себе график не строит, но возвращает векторы **T** и **R**, которые нужны команде **polar(T,R)** для построения подобной гистограммы.

4.3.3. Графики векторов — команда compass

Иногда желательно представление ряда радиус-векторов в их обычном виде, то есть в виде стрелок, исходящих из начала координат и имеющих угол и длину, кото-

рые определяются действительной и мнимой частями комплексных чисел, представляющих данные векторы. Для этого служит группа команд **compass**:

compass(U,V) — строит графики радиус-векторов с компонентами (**U**,**V**), представляющими действительную и мнимую части каждого из радиус-векторов.

compass(Z) — эквивалентно **compass(real(Z), imag(Z))**.

compass(U,V,LINESPEC) и **compass(Z,LINESPEC)** — аналогичны представленным выше командам, но позволяют задавать спецификацию линий построения **LINESPEC**, подобную описанной для команды **plot**.

На следующем примере показано использование команды **compass**:

```
» % Построение радиус-векторов для вектора Z с комплексными
» % элементами
» Z=[-1+2i,-2-3i,2+3i,5+2i];
» compass(Z)
```

Построенный по этому примеру график представлен на рис. 4.15.

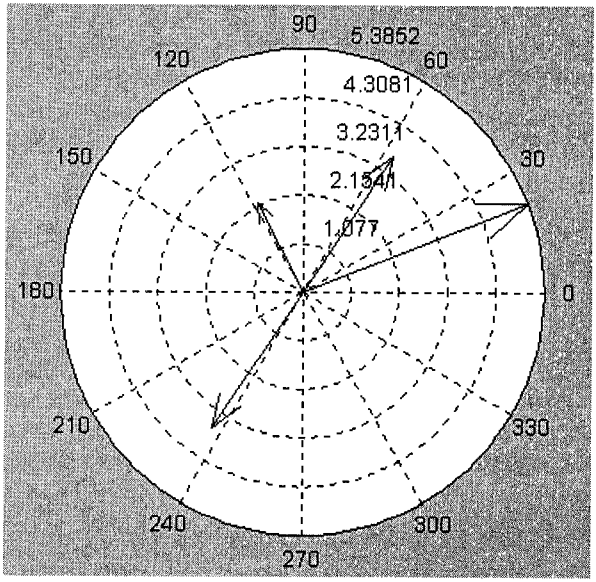


Рис. 4.15. Построение радиус-векторов

Функция

H=compass(...)

возвращает значения длин радиус-вектора.

4.3.4. График проекций векторов на плоскость — команда feather

Иногда полезно отображать комплексные величины вида z=x+yi в виде проекции радиус-вектора на плоскость. Для этого используется графическая команда **feather**:

feather(U,V) — строит график проекции векторов, заданных компонентами **U** и **V**, на плоскость.
feather(Z) — для вектора **Z** с комплексными элементами дает построения, аналогичные **feather(REAL(Z),IMAG(Z))**.
feather(..., S) — дает построения, описанные выше, но со спецификацией линий, заданной строковой константой **S** по аналогии с командой **plot**.

Пример применения команды **feather**:

```
» %Пример применения команды feather
» x=0:0.1*pi:3*pi;
» z=exp(x*y);
» feather(z)
```

График, построенный по последнему примеру, показан на рис. 4.16.

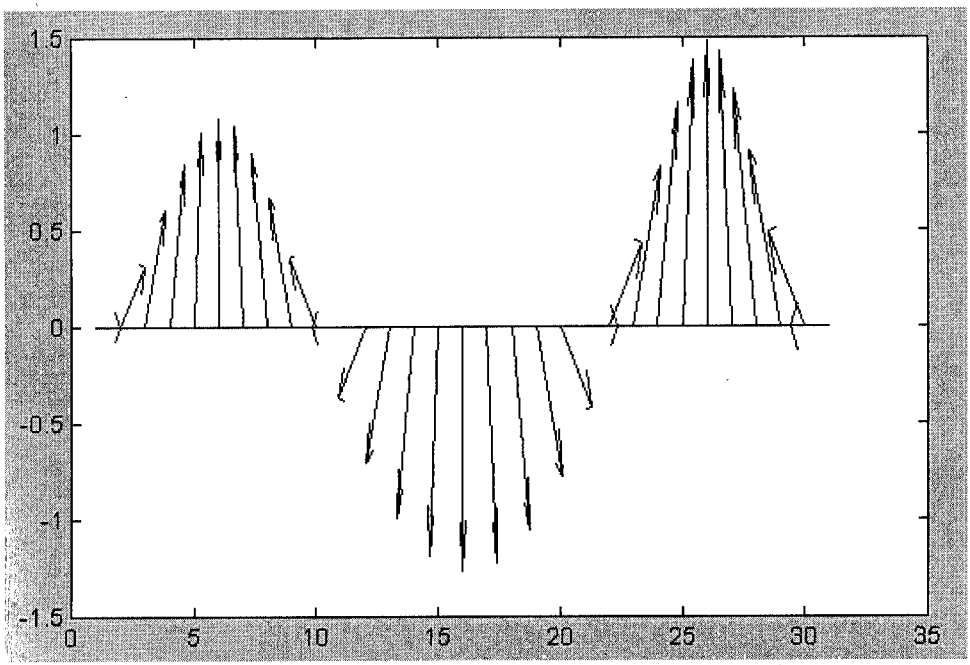


Рис. 4.16. График, построенный командой **feather**

Функция

H = FEATHER(...)

возвращает вектор, использованный командой **feather**.

4.4. Построение контурных графиков

4.4.1. Контурные графики типа contour

Контурные графики являются попыткой отобразить на плоскости функции двух переменных вида z(x,y). Их можно представить в виде совокупности линий равного уровня, которые получаются, если трехмерная поверхность пересекается рядом плоскостей, расположенных параллельно друг другу. При этом контурный график представляет совокупность спроектированных на плоскость y(x) линий пересечения поверхности z(x,y) плоскостями.

Для построения контурных графиков используются варианты команды **contour**:

contour(Z) — строит контурный график по данным матрицы **Z** с автоматическим заданием диапазонов изменения x и y.

contour(X,Y,Z) — строит контурный график по данным матрицы **Z** с указанием спецификаций для **X** и **Y**.

contour(Z,N) и **contour(X,Y,Z,N)** — дает построения, аналогичные ранее описанным, с заданием N линий равного уровня (по умолчанию N=10).

contour(Z,V) и **CONTOUR(X,Y,Z,V)** — строят линии равного уровня для высот, указанных значениями элементов вектора V.

contour[v v]) или **contour(X,Y,Z,[v v])** — вычисляет одиночный контур для уровня **v**.

[C,H] = contour(...) — возвращает дескриптор — матрицу **C** и вектор столбцов **H**. Они используются как входные параметры для команды **clabel**.

contour(...,'LINESPEC') — позволяет использовать перечисленные выше команды с указанием спецификации линий, которыми идет построение.

Пример построения контурного графика поверхности, заданной функций **peaks**:

```
» %Построение контурного графика объекта peaks
» z=peaks(27);
» contour(z,15)
```

Построенный на этом примере график показан на рис. 4.17. Заметим, что объект — функция **peaks** задана в системе в готовом виде.

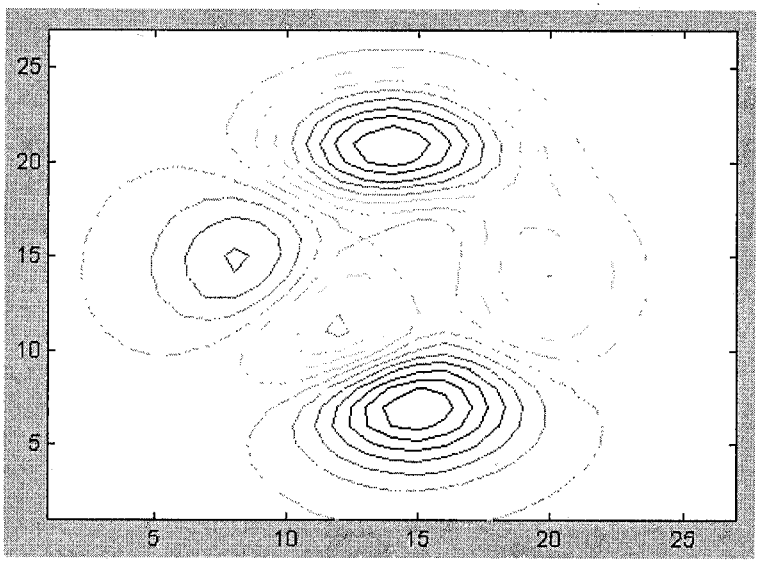


Рис. 4.17. Контурный график, построенный с помощью команды **contour**

Графики этого типа часто используются в топографии для представления на листе бумаги (как говорят математики — на плоскости) объемного рельефа местности.

4.4.2. Функции meshgrid и ndgrid для создания массивов трехмерной графики

Трехмерные поверхности обычно описываются функцией двух переменных z(x,y). Специфика построения трехмерных графиков требует не просто задания ряда значений x и y, то есть векторов **x** и **y** — она требует определения двумерных массивов для **X**- и **Y**-матриц. Для создания таких массивов служит функция:

meshgrid(x,y) meshgrid(x) meshgrid(x,y,z)

Последняя запись, по существу, аналогична **meshgrid(X,X)**. Третья форма функции возвращает три массива. В основном приведенные варианты функции используются совместно с функциями построения графиков трехмерных поверхностей. Обычно они записываются в форме:

[X,Y] = meshgrid(x,y) — преобразует область, заданную векторами **x** и **y** в массивы **X** и **Y**, которые могут быть использованы для вычисления функции двух переменных и построения трехмерных графиков. Строки выходного массива **X** являются копиями вектора **x**, а столбцы **Y** — копиями вектора **y**.

Функция **[X,Y] = meshgrid(x)** — аналогична **[X,Y] = meshgrid(x,x).**

[X,Y,Z] = meshgrid(x,y,z) — возвращает трехмерные массивы, используемые для вычисления функций трех переменных и построения трехмерных 3D-графиков.

Пример:

```
» [X,Y] = meshgrid(1:4,13:17)
X =
   1   2   3   4
   1   2   3   4
   1   2   3   4
   1   2   3   4
   1   2   3   4
Y =
  13  13  13  13
  14  14  14  14
  15  15  15  15
  16  16  16  16
  17  17  17  17
```

[X1,X2,X3,...] = ndgrid(x1,x2,x3,...) — преобразовывает область, заданную векторами **x1,x2,x3...** в массивы **X1,X2,X3...**, которые могут быть использованы для вычисления функций нескольких переменных и многомерной интерполяции. **I** — ая размерность выходного массива **Xi** является копией вектора **xi**.

[X1,X2,...] = ndgrid(x) аналогична **[X1,X2,...] = ndgrid(x,x,...).**

Типичный пример применения функции **meshgrid**

```
» [X,Y] = meshgrid(-2:.2:2, -2:.2:2)
```

позволяет задать опорную плоскость для построения 3D-поверхности при изменении x и y от -2 до 2 с шагом 0,2. Дополнительные примеры применения функции **meshgrid** будут приведены далее при описании соответствующих команд. Рекомендуется ознакомиться с командами **surf** и **slice**.

4.4.3. Графики полей градиентов quiver

Для построения графиков полей градиентов служат команды **quiver**:

quiver(X,Y,U,V) — строит график поля градиентов в виде стрелок для каждой пары элементов массивов **X** и **Y**, причем элементы массивов **U** и **V** указывают на направление и размер стрелок.

quiver(U,V) — строит векторы скорости в равнорасположенных точках на x-y-плоскости.

quiver(U,V,S) и **quiver(X,Y,U,V,S)** — при построении автоматически масштабируют стрелки по сетке и затем вытягивают их по значению S. Используйте S=0, чтобы построить стрелки без автоматического масштабирования.

quiver(...,LINESPEC) — использует тип линии, точно определенный для векторов. Любой маркер в linespec заменяет типовую стрелку. Используйте '.' для отмены любого вида маркера (см. PLOT для получения информации о других возможностях).

quiver(...,'filled') — дает заполнение любыми, точно установленными маркерами.
H=quiver(...) — возвращает вектор дескрипторов линии.

Ниже представлен пример применения команды **quiver**:

```
» % Построение поля градиентов для трехмерной поверхности
» x = -2:.2:2; y = -1:.2:1;
» [xx,yy] = meshgrid(x,y);
» zz = xx.*exp(-xx.^2-yy.^2);
» [px,py] = gradient(zz,.2,.2);
» quiver(x,y,px,py,2);
```

Построенный в этом примере график показан на рис. 4.18.

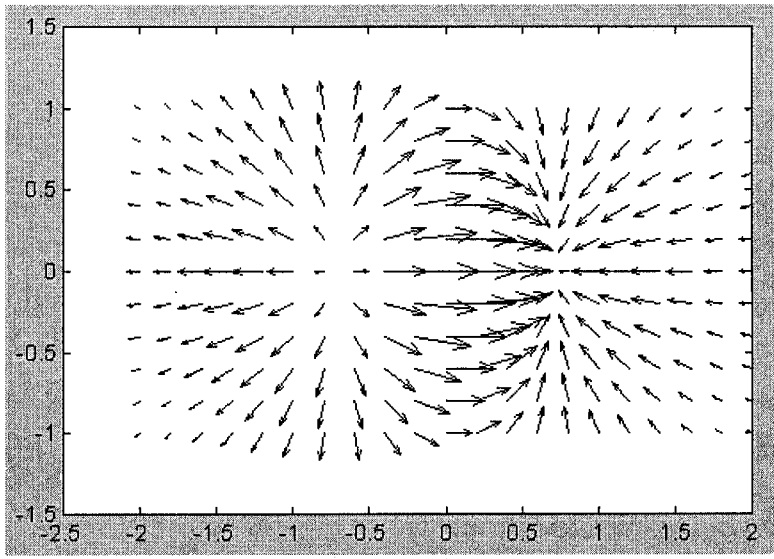


Рис. 4.18. Пример построения графика поля градиентов

Нетрудно заметить, что представление поля градиентов стрелками дает весьма наглядное представление о линиях поля, указывая области, куда эти линии впадают и откуда они исходят.

4.5. Построение графиков 3D-поверхностей

4.5.1. Команда plot3

Команда **plot3(...)** является аналогом команды **plot(...)**, но относится к функции двух переменных z(x,y). Она строит аксонометрическое изображение трехмерных (3D) поверхностей и представлена следующими формами:

plot3(x,y,z) — строит массив точек, представленных векторами **x**, **y** и **z**, соединяя их отрезками прямых. Эта команда имеет ограниченное применение.

plot3(X,Y,Z) — строит точки с координатами **X(i,:)**, **Y(i,:)** и **Z(i,:)** (где **X**, **Y** и **Z** — три матрицы одинакового размера) и соединяет их отрезками прямой.

Ниже дан пример построения трехмерной поверхности, описываемой функцией z(x,y)=x^2+y^2:

```
» % Построение графика 3D-поверхности линиями
» [X,Y]=meshgrid([-3:0.15:3]);
» Z=X.^2+Y.^2;
» plot3(X,Y,Z)
```

График этой поверхности показан на рис. 4.19.

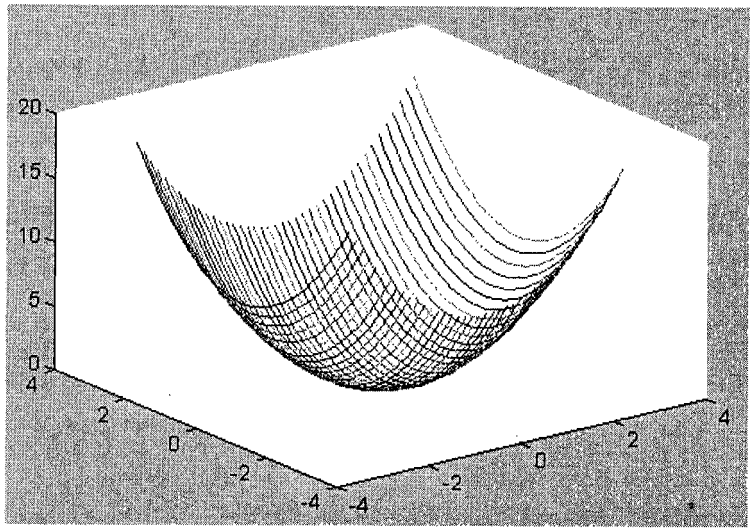


Рис. 4.19. График трехмерной поверхности

plot3(X,Y,Z,S) — обеспечивает построения, аналогичные предшествующей команде, но со спецификацией стиля линий и точек, соответствующей спецификации команды **plot**. Ниже дан пример применения этой команды для построения трехмерной поверхности кружками:

```
» % Построение графика 3D-поверхности цветными кружками
» [X,Y]=meshgrid([-3:0.15:3]);
» Z=X.^2+Y.^2;
» plot3(X,Y,Z,'o')
```

График поверхности, построенный кружками, показан на рис. 4.20.

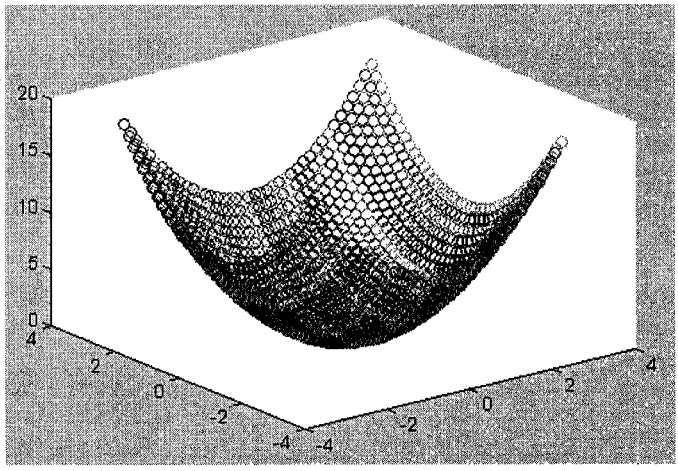


Рис. 4.20. График трехмерной поверхности, построенной разноцветными кружками

plot3(x1,y1,z1,s1,x2,y2,z2,s2,x3,y3,z3,s3,...) — строит на одном рисунке графики нескольких функций **z1(x1,y1), z2(x2,y2)** и так далее со спецификацией каждой из них.

Пример применения последней команды дан ниже:

```
» %Построение графика 3D-поверхности со спецификацией линий
» [X,Y]=meshgrid([-3:0.15:3]);
» Z=X.^2+Y.^2;
» plot3(X,Y,Z,'-k',Y,X,Z,'-k')
```

График функции, соответствующей последнему примеру, представлен на рис. 4.21.

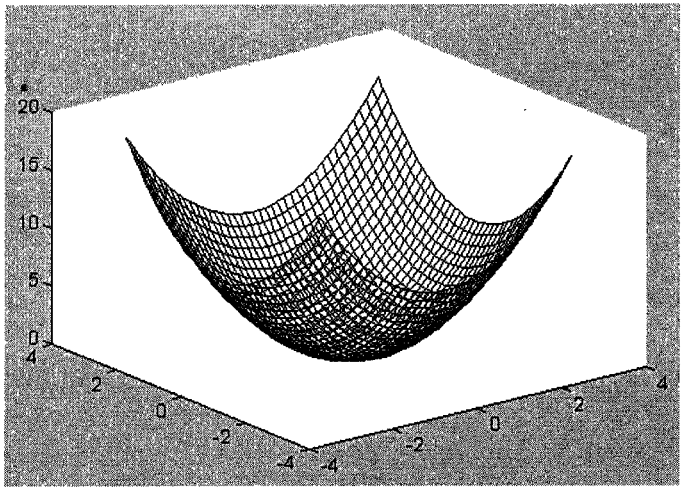


Рис. 4.21. График функции в сеточном представлении

В данном случае строится два графика одной и той же функции с взаимно перпендикулярными образующими линиями. Поэтому график имеет вид сетки без окраски ее ячеек (напоминает проволочный каркас фигуры).

4.5.2. Сеточные 3D-графики с функциональной окраской — команды класса mesh

Наиболее представительными и наглядными являются сеточные графики трехмерных поверхностей с заданной или функциональной окраской. В названии их команд присутствует слово **mesh**. Имеется три группы таких команд. Ниже приведены данные о наиболее полных формах этих команд. Наличие более простых форм можно уточнить, используя команду **help Имя**, где **Имя** — имя соответствующей команды.

mesh(X,Y,Z,C) — выводит в графическое окно сетчатую поверхность **Z(X,Y)** с цветами узлов поверхности, заданных массивом **C**.

mesh(X,Y,Z) – аналог предшествующей команды при **C = Z**, при этом используется функциональная окраска, при которой цвет задается высотой поверхности.

Возможны также формы команды **mesh(x,y,Z)**, **mesh(x,y,Z,C)**, **mesh(Z)** и **mesh(Z,C)**. Функция **mesh** возвращает дескриптор для объекта **surface**.

Ниже приводится пример применения команды **mesh**:

```
» % Пример построения 3D-поверхности с функциональной закраской
» % ячеек
» [X,Y]=meshgrid([-3:0.15:3]);
» Z=X.^2+Y.^2;
» mesh(X,Y,Z)
```

На рис. 4.22 показан график трехмерной поверхности, созданной командой **mesh(X,Y,Z)**. Нетрудно заметить, что функциональная окраска поверхности заметно усиливает наглядность ее представления.

MATLAB имеет несколько графических функций, возвращающих матричный образ трехмерных поверхностей. Например, функция **peaks(N)** возвращает матричный образ поверхности с рядом пиков и впадин. Такие функции удобно использовать для проверки работы графических команд трехмерной графики. Для упомянутой функции **peaks** можно привести такой пример:

```
» %Пример построения поверхности, описанной функцией peaks
» z=peaks(25);
» mesh(z);
```

График 3D-поверхности, описываемой функцией **peaks**, представлен на рис. 4.23.

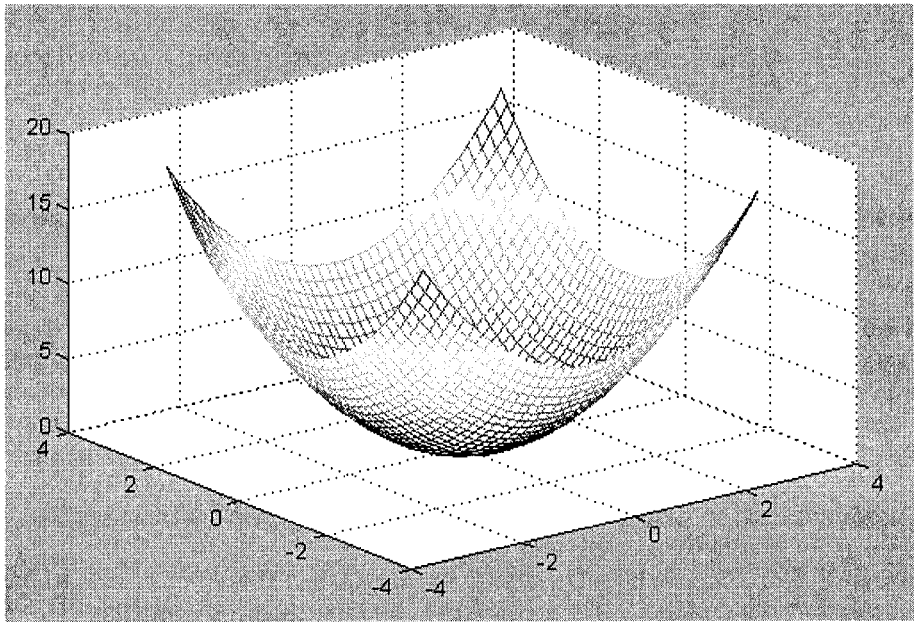


Рис. 4.22. График 3D-поверхности, созданный командой **mesh(X,Y,Z)**

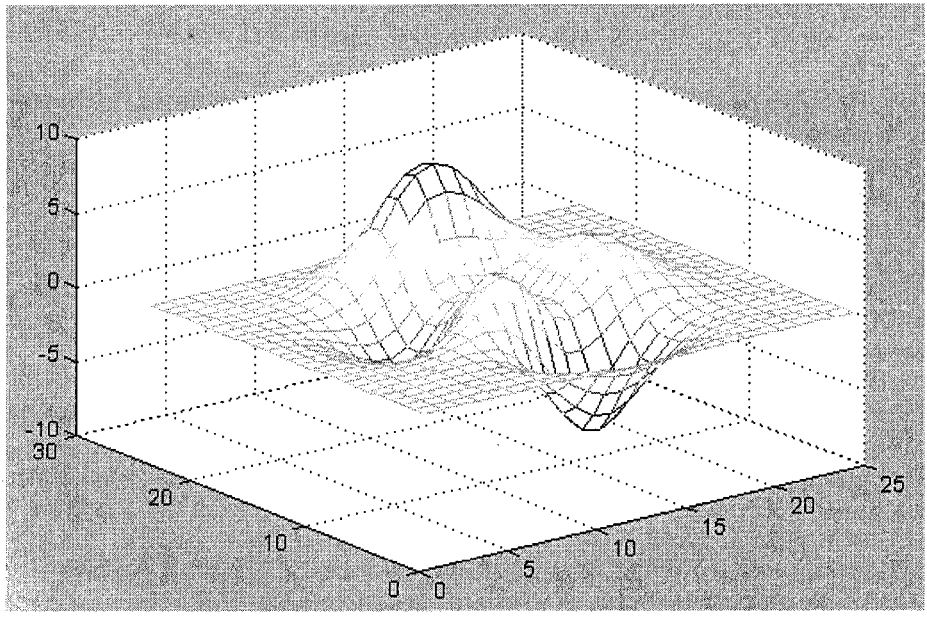


Рис. 4.23. График 3D-поверхности, описываемой функцией **peaks**

Рекомендуется ознакомиться с командами и функциями, используемыми совместно с описанными командами: **axis**, **caxis**, **colormap**, **hold**, **shading** и **view**.

4.5.3. Построение 3D-поверхности с проекцией — команда meshc

Иногда график 3D-поверхности полезно объединить с контурным графиком ее проекции на плоскость, расположенную под поверхностью. Для этого используется команда **meshc(...)**, которая аналогична **mesh(...)**, но помимо графика трехмерной поверхности дает изображение ее проекции в виде линий равного уровня (графика типа **contour**).

Ниже дан пример применения этой команды:

```
» %Построение графика 3D-поверхности и ее проекции
» [X,Y]=meshgrid([-3:0.15:3]);
» Z=X.^2+Y.^2;
» meshc(X,Y,Z)
```

Построенный график показан на рис. 4.24.

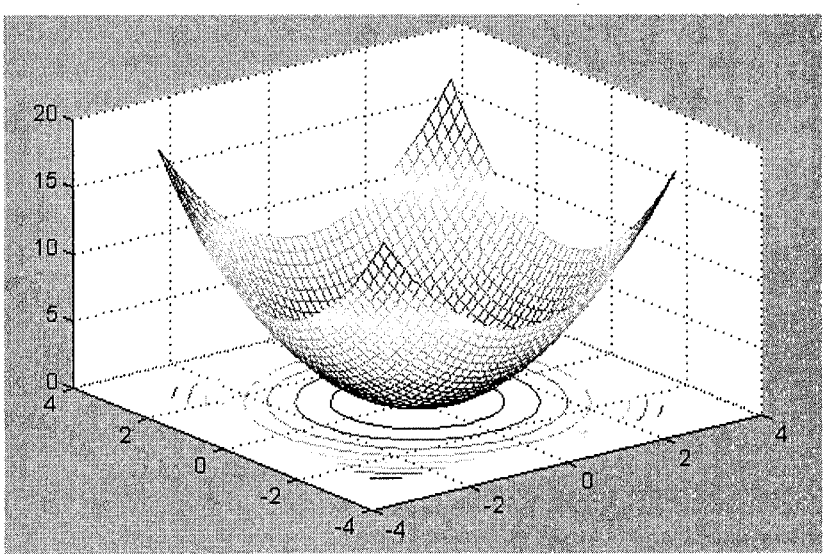


Рис. 4.24. График трехмерной поверхности и ее проекции на расположенную ниже плоскость

Нетрудно заметить, что график такого типа дает наилучшее представление об особенностях поверхности.

4.5.4. Построение 3D-поверхности столбцами — команда meshz

Еще один тип представления 3D-поверхности, когда она строится из многочисленных столбцов, дает команда **meshz**

meshz(...) — аналогична **mesh(...)**, но строит поверхность как бы в виде столбиков.

Следующий пример иллюстрирует применение команды **mesh**:

```
» %Построение 3D-поверхности столбцами
» [X,Y]=meshgrid([-3:0.15:3]);
» Z=X.^2+Y.^2;
» mesz(X,Y,Z)
```

Столбиковый график трехмерной поверхности показан на рис. 4.25.

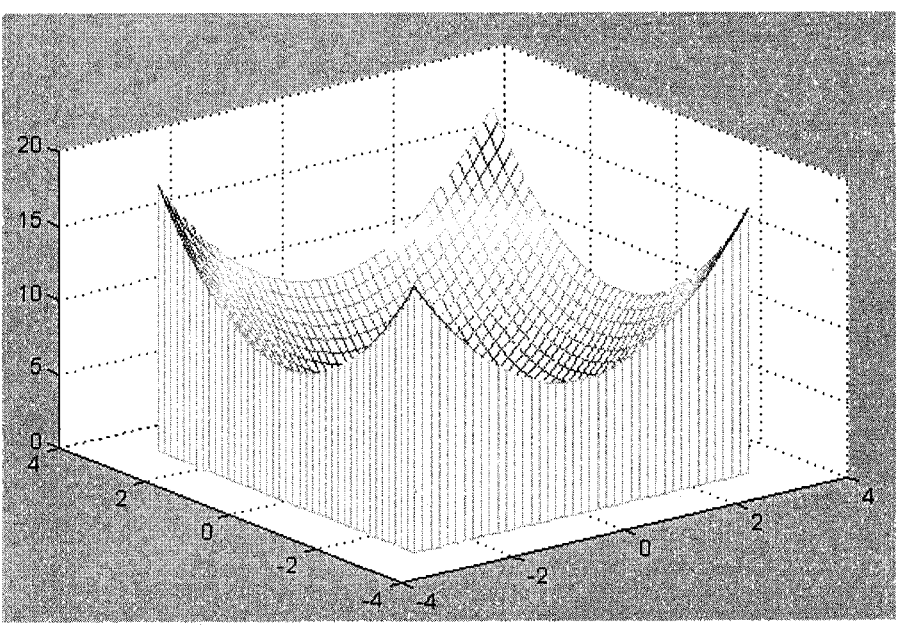


Рис. 4.25. Построение поверхности столбцами

График такого типа используется довольно редко. Возможно, он полезен архитекторам или скульпторам, поскольку дает неплохое объемное представление о поверхностях.

4.5.5. Построение 3D-поверхности с функциональной окраской — команда surf

Особенно наглядное представление о трехмерных поверхностях дают графики, использующие функциональную закраску ячеек сеток. Например, цвет окраски поверхности z(x,y) может быть поставлен в соответствие с высотой z поверхности, с выбором для малых высот темных тонов, а для больших — светлых.

Для построения таких поверхностей используются варианты команды класса **surf(...)**:

surf(X,Y,Z,C) — строит цветную параметрическую 3D-поверхность по данным матриц **X**, **Y** и **Z** и цветом, задаваемым массивом **C**.

surf(X,Y,Z) — аналогична предшествующей команде, где **C= Z**, так что цвет задается высотой той или иной ячейки 3D-поверхности.

surf(x,y,Z) и **surfSURF(x,y,Z,C)** — с двумя векторными аргументами заменяют первых два матричных аргумента и должны иметь длину **length(x) = n** и **length(y) = m**, где **[m,n] = size(Z)**. В этом случае вершины областей поверхности представлены тройками (x(j), y(i), Z(i,j)). Заметим, что x соответствует столбцам Z, а y соответствует строкам.

surf(Z) и **surf(Z,C)** — используют x = 1:n и y = 1:m. В этом случае высота Z — однозначно определенная функция, заданная геометрически прямоугольной сеткой.

surf — возвращает свойства объекта **surface**.

Команды **axis**, **caxis**, **colormap**, **hold**, **shading** и **view** задают координатные оси и свойства поверхности, которые могут использоваться для большей эффектности показа поверхности или фигуры.

Ниже приведен простой пример построения 3D-поверхности — объемной параболы:

```
» % Построение объемной параболы с функциональной окраской
» % ячеек
» [X,Y]=meshgrid([-3:0.15:3]);
» Z=X.^2+Y.^2;
» surf(X,Y,Z)
```

Соответствующий этому примеру график показан на рис. 4.26.

Можно заметить, что, благодаря функциональной окраске, график поверхности гораздо более выразителен, чем при построениях без такой окраски, представленных ранее (причем, даже в том случае, когда цветной график печатается в черно-белом виде).

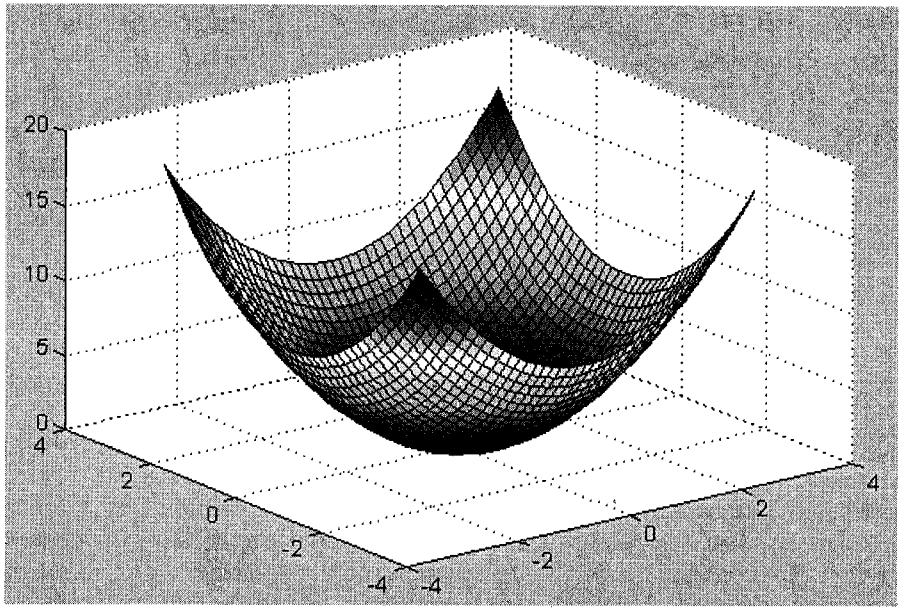


Рис. 4.26. График объемной параболы с функциональной окраской ячеек

В следующем примере используется функциональная окраска оттенками серого цвета с выводом шкалы цветовых оттенков:

```
» % Построение 3D-поверхности с функциональной окраской ячеек
» % типа
» % grayscale и выводом шкалы цветовых оттенков с плавными
» % переходами
» [X,Y]=meshgrid([-3:0.1:3]);
» Z=sin(X)./(X.^2+Y.^2+0.3);
» surf(X,Y,Z)
» colormap(gray)
» shading interp
» colorbar
```

В этом примере команда **colormap(gray)** задает окраску тонами серого цвета, а команда **shading interp** обеспечивает устранение изображения сетки и задает интерполяцию для оттенков цвета объемной поверхности. На рис. 4.27 показан вид графика, построенного по этому примеру.

Обычно применение интерполяции для окраски придает трехмерным поверхностям и фигурам более реалистичный вид, но фигуры каркасного вида дают более точные количественные данные о каждой точке.

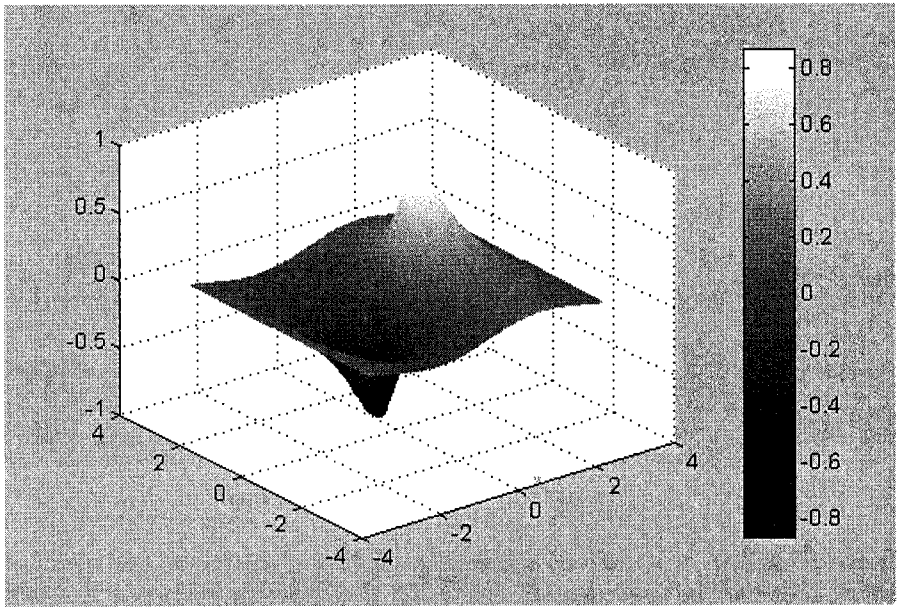


Рис. 4.27. График трехмерной поверхности с функциональной окраской серым цветом

4.5.6. Построение 3D-поверхности и ее проекции — команда surfc

Для повышения представительности 3D-поверхностей можно использовать дополнительный график линий равного уровня, получаемый путем проекции поверхности на опорную плоскость графика (под поверхностью). Для этого используется команда:

surfc(...) — аналогична команде **surf**, но обеспечивает дополнительное построение контурного графика проекции фигуры на опорную плоскость.

Пример применения команды **surfc** приводится ниже:

```
» %Построение графика 3D-поверхности и ее проекции на опорную
» % плоскость
» [X,Y]=meshgrid([-3:0.1:3]);
» Z=sin(X)./(X.^2+Y.^2+0.3);
» surfc(X,Y,Z)
```

На рис. 4.28 показаны графики, построенные по данному примеру.

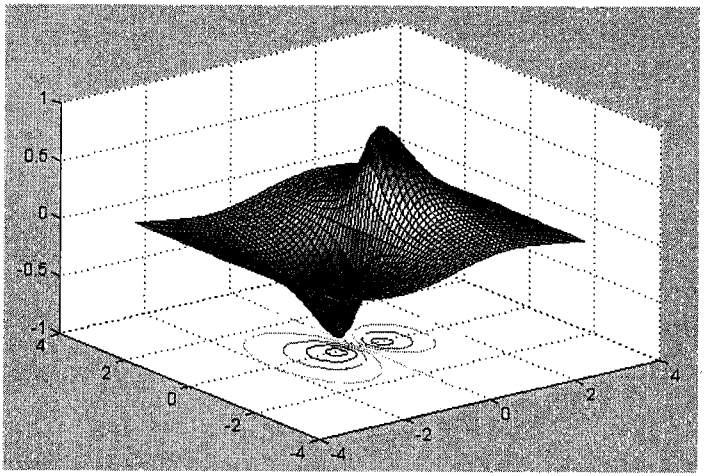


Рис. 4.28. График 3D-поверхности и ее проекции на опорную плоскость

Рассмотрим еще один пример применения команды **surfc**, на этот раз для построения поверхности, описываемой функцией **peaks** с применением для окраски интерполяции цветов и построением шкалы цветов:

```
» %Построение фигуры peaks с интерполяцией цветов и их шкалой
» [X,Y]=meshgrid([-3:0.1:3]);
» Z=peaks(X,Y);
» surfc(X,Y,Z)
» shading interp
» colorbar
```

Рис. 4.29 показывает график, построенный по этому примеру.

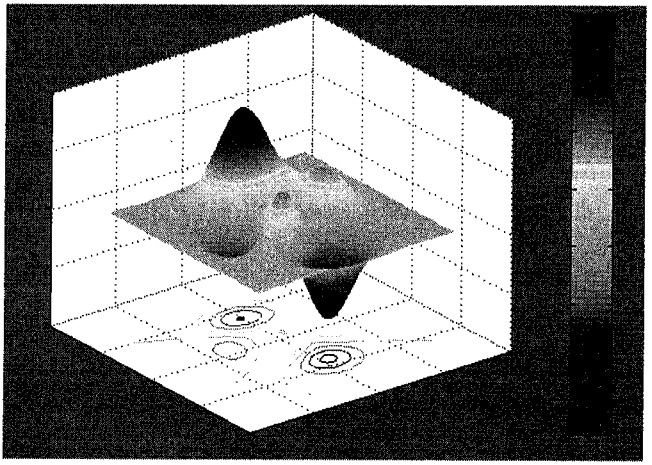


Рис. 4.29. График функции **peaks** с проекцией и шкалой цветов

И здесь нетрудно заметить, что график сложных поверхностей с интерполяцией цветовых оттенков выглядит более реалистичным, чем график сеточного вида и график без интерполяции цветов.

4.5.7. Построение освещенной 3D-поверхности — команда surfl

Пожалуй, наиболее реалистичный вид имеют графики трехмерных поверхностей, в которых имитируется освещение от точечного источника света, расположенного в заданном месте координатной системы. Подсветка учитывает рассеивание, отражение и зеркальный эффект. Для получения таких графиков используется команда **surfl**:

surfl(...) — аналогична команде **surf(...)**, но строит график 3D-поверхности с подсветкой от источника света.

surfl(Z), **surfl(X,Y,Z)**, **surfl(Z,S)** и **surfl(X,Y,Z,S)** — строят графики 3D-поверхности с подсветкой от источника света, положение которого в системе Декартовых координат задается вектором **S = [Sx,Sy,Sz]**, а в системе сферических координат вектором **S = [AZ,EL]**.

surfl(...,'light') — при построении позволяет задать цвет подсветки с помощью объекта LIGHT.

surfl(...,'cdata') — при построении имитирует эффект отражения.

surfl(X,Y,Z,S,K) — задает построение 3D-поверхности с параметрами, заданными вектором **K=[ka,kd,ks,spread]**, где ka — коэффициент отражения, kd — коэффициент диффузного отражения, ks — коэффициент зеркального отражения и spread — коэффициент зеркального распространения.

H=surfl(...) — возвращает данные о подсветке.

По умолчанию вектор **S** задает углы азимута и возвышения в 45 градусов. Используя команды **cla**, **hold on**, **view(AZ,EL)**, **surfl(...)**, **hold off**, можно получить дополнительные возможности управления освещением. Надо полагаться на упорядочение точек в **X-**, **Y-**, и **Z**-матрицах, чтобы определить внутреннюю и внешнюю стороны параметрических поверхностей. Попробуйте использовать **surfl(X',Y',Z')**, если вам не нравится результат этой функции. В зависимости от способа вывода нормальных векторов поверхности, **surfl** требует матрицу, по крайней мере, размера 3×3.

Ниже представлен пример применения команды **surfl**:

```
» % Построение графика освещенной точечным источником
» % 3D-поверхности
» [X,Y]=meshgrid([-3:0.1:3]);
» Z=sin(X)./(X.^2+Y.^2+0.3);
» surfl(X,Y,Z)
» colormap(gray)
» shading interp
» colorbar
```

Построенная по этому примеру поверхность представлена на рис. 4.30.

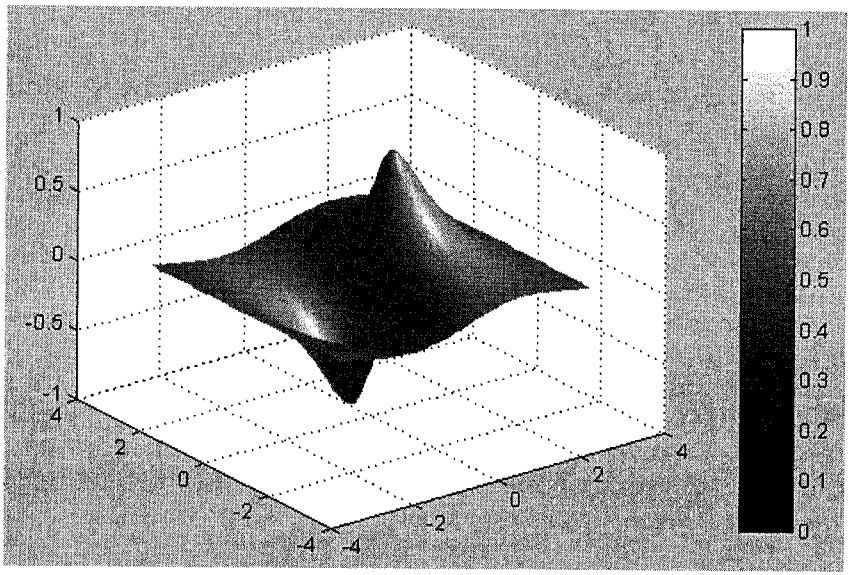


Рис. 4.30. График поверхности с имитацией ее освещения точечным источником

Сравните этот рисунок с рис. 4.27, на котором та же поверхность построена без имитации ее освещения точечным источником света.

4.5.8. Средства управления подсветкой и обзора фигур

Рекомендуется с помощью команды **help** ознакомиться с командами, задающими управление подсветкой:

diffuse — задание эффекта диффузионного рассеяния;
lighting — управление подсветкой;
material — имитация свойств рассеивания света различными материалами;
specular — задание эффекта зеркального отражения.

Следующие три команды позволяют управлять углами просмотра, под которыми рассматривается видимая в графическом окне фигура:

view — задание положения точки просмотра;
viewmtx — задание и вычисление матрицы вращения;
rotate3d — задание поворота 3D-фигуры.

В ряде случаев применением этих команд можно добиться большей выразительности 3D-объектов.

4.5.9. Построение графика сечения для 3D-поверхности — команда slice

Графики сечения трехмерных поверхностей строит команда **slice** (в переводе — ломтик). Она используется в следующих формах:

slice(X,Y,Z,V,Sx,Sy,Sz) — строит плоские сечения 3D-поверхности в направлении осей x, y, z с позициями, задаваемыми векторами **Sx**, **Sy**, **Sz**. Массивы **X**,**Y**, **Z** задают координаты для **V** и должны быть монотонными и трехмерными (как возвращаемые функцией **meshgrid**). Цвет каждой точки определяется трехмерной интерполяцией в объеме **V**. Размер массива **V** M×N×P, где M, N и P — длины векторов **X**, **Y**, **Z**.

slice(X,Y,Z,V,Xl,Yl,Zl) — строит секторы через объем V по поверхности, определенной массивами **Xl**, **Yl**, **Zl**.

slice(V,Sx,Sy,Sz) или **slice(V,Xl,Yl,Zl)** с **X**=1:N, **Y**=1:M, **Z**=1:P.

slice(...,'method') — при построении задается метод интерполяции, который может быть следующим: 'linear', 'cubic' или 'nearest'. По умолчанию используется линейная интерполяция — 'linear'.

H=slice(...) — возвращает дескриптор объекта **surface**.

График примера, показанного ниже

```
» % Построение сечений 3D-поверхности
» [x,y,z] = meshgrid(-2:.2:2, -2:.25:2, -2:.16:2);
» v = sin(x) .* exp(-x.^2 - y.^2 - z.^2);
» slice(x,y,z,v,[-1.2 .8 2],2,[-2 -.2])
```

представлен на рис. 4.31.

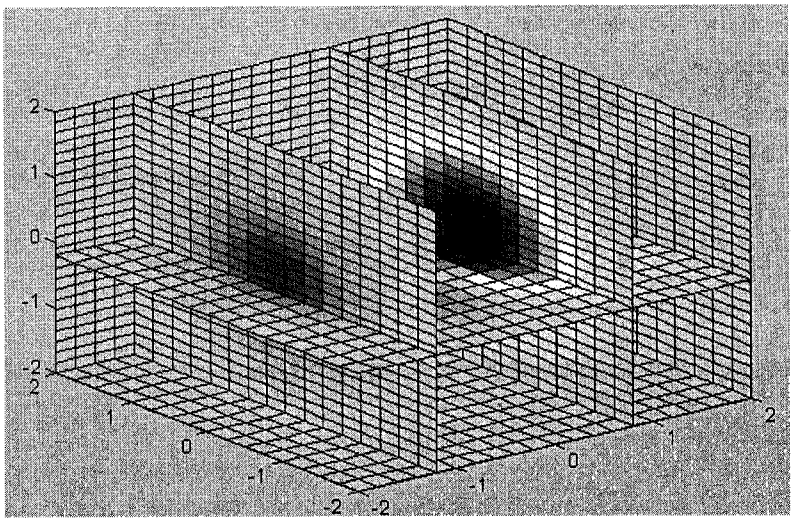


Рис. 4.31. График, показывающий сечения трехмерной поверхности

4.5.10. График трехмерной слоеной поверхности — функция waterfall

Иногда бывают полезны графики трехмерных слоеных поверхностей, как бы состоящие из тонких пластинок — слоев. Такие поверхности строит функция **waterfall**:

waterfall(...) — строит поверхность как команда **mesh(...)**, но без показа ребер сетки.. При ориентации графика относительно столбцов следует использовать запись **waterfall(Z')** или **waterfall(X',Y',Z')**.

Рассмотрим пример применения команды **waterfall**:

```
» % Построение графика слоеной поверхности
» [X,Y]=meshgrid([-3:0.1:3]);
» Z=sin(X)./(X.^2+Y.^2+0.3);
» waterfall(X,Y,Z)
» colormap(gray)
» shading interp
```

Соответствующий график показан на рис. 4.32.

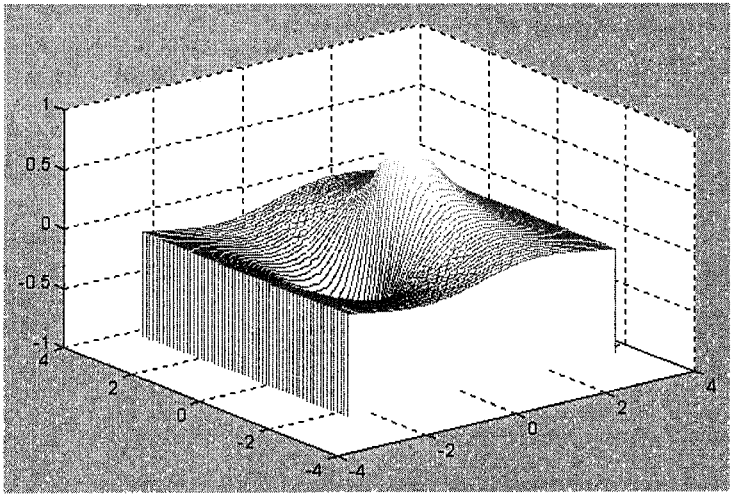


Рис. 4.32. Трехмерная слоеная поверхность

4.5.11. Трехмерные контурные графики — команда contour3

Трехмерный контурный график представляет собой расположенные в пространстве линии равного уровня, полученные от расслоения трехмерной фигуры рядом секущих плоскостей, расположенных параллельно опорной плоскости фигуры. При этом, в отличие от двумерного контурного графика, линии равного уровня отображаются в

аксонометрии. Для получения трехмерных контурных графиков используется команда **contour3**:

contour3(...) — имеет синтаксис команды **contour(...)**, но строит линии равного уровня в аксонометрии с использованием функциональной окраски (уровни оцениваются по оси **Z**).

Полезные частные формы записи этой команды:

contour3(Z) — строит контурные линии для поверхности, заданной массивом **Z** без учета диапазона изменения x и y.

contour3(Z,n) — строит то же, что предыдущая команда, но с установкой n линий (по умолчанию n=10).

contour3(X,Y,Z) — строит контурные линии для поверхности, заданной массивом **Z** с учетом изменения x и y. Двумерные массивы X и Y создаются с помощью функции **meshgrid**.

contour3(X,Y,Z,n) — строит то же, что предыдущая команда, но с установкой n линий.

Пример применения команды **contour3**:

```
» % Построение трехмерного контурного графика
» contour3(peaks,20)
» colormap(gray)
```

Соответствующий данному примеру график представлен на рис. 4.33. В данном случае задано построение двадцати линий уровня.

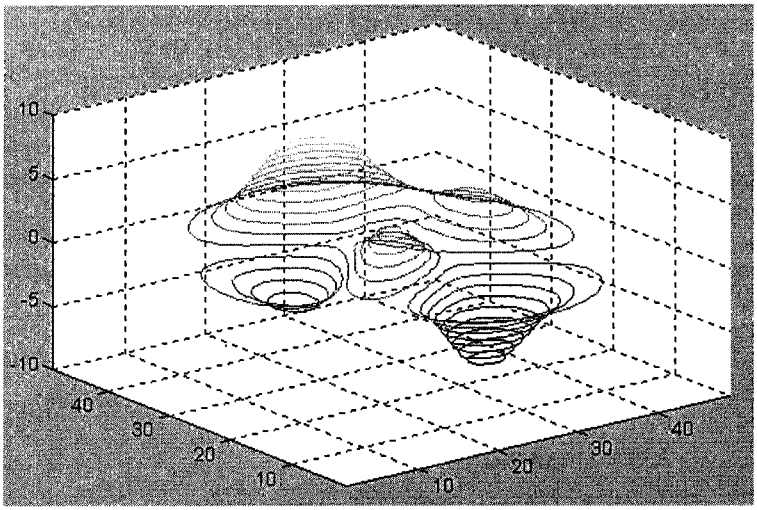


Рис. 4.33. Трехмерный контурный график для функции **peaks**

С командой **contour3** связаны следующие формы одноименной функции:

C=contour3(...) — возвращает матрицу описания контурных линий C для использования командой **clabel**.

[C,H]=contour3(...) — возвращает массив **C** и вектор-столбец **H** дескрипторов объектов **path** для каждой линии уровня. Свойство User Data каждого объекта содержит значение высоты для каждого контура.

4.6. Команды оформления графики

После того, как график уже построен, MATLAB позволяет выполнить его форматирование или оформление в нужном виде. Соответствующие этому средства описаны ниже.

4.6.1. Установка титульной надписи — команда title

Для установки над графиком титульной надписи используется следующая команда:

title('string') — установка титульной надписи на 2D- и 3D-графиках, заданной строковой константой **'string'**.

Пример применения этой команды будет дан в следующем разделе.

4.6.2. Установка осевых надписей — команды xlabel, ylabel, zlabel

Для установки надписей возле осей x, y и z используются команды:

xlabel('String') **ylabel('String')** **zlabel('String')**

Соответствующая надпись задается символьной константой переменной 'String'. Пример установки титульной надписи и надписей по осям графиков приводится ниже:

```
» % Построение 3D-фигуры с титульной надписью и надписью у осей
» [X,Y]=meshgrid([-3:0.1:3]);
» Z=sin(X)./(X.^2+Y.^2+0.3);
» surfl(X,Y,Z)
» colormap(gray)
» shading interp
» colorbar
» title('График трехмерной поверхности')
» xlabel('Ось X')
» ylabel('Ось Y')
» zlabel('Ось высот Z')
```

Построенный по этому примеру график трехмерной поверхности показан на рис. 4.34.

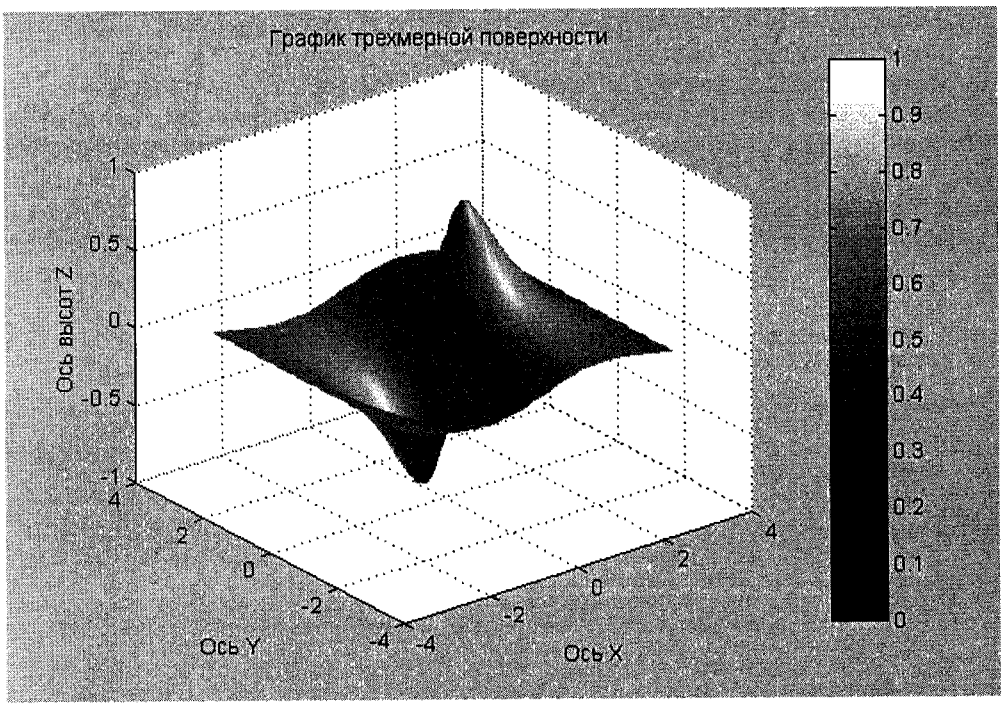


Рис. 4.34. График 3D-поверхности с титульной надписью и надписями по координатным осям

Сравните его с графиком, показанным на рис. 4. 30. Надписи делает рисунок более наглядным.

4.6.3. Ввод текста в любое место графика — команда text

Часто возникает необходимость добавления текста в определенное место графика, например, для обозначения той или иной кривой графика. Для этого используется команда **text**:

text(X,Y,'string') — добавляет в двумерный график текст, заданный строковой константой 'string', в позицию начала текста с координатами (X,Y). Если **X** и **Y** заданы как одномерные массивы, то надпись помещается во все позиции [x(i),y(i)].

text(X,Y,Z,'string') — добавляет в трехмерный график текст, заданный строковой константой 'string', в позицию, заданную координатами X, Y и Z.

В приведенном ниже примере надпись "График функции sin(x)/x" размещается под кривой графика в позиции (-5, -0.3):

```
» % Построение графика с текстовой надписью внутри окна
» x=-15:0.1:15;
» plot(x, sin(x)./(x+eps))
» text(-5,-0.3,'График функции sin(x)/x')
```

График функции с надписью под ее кривой показан на рис. 4.35.

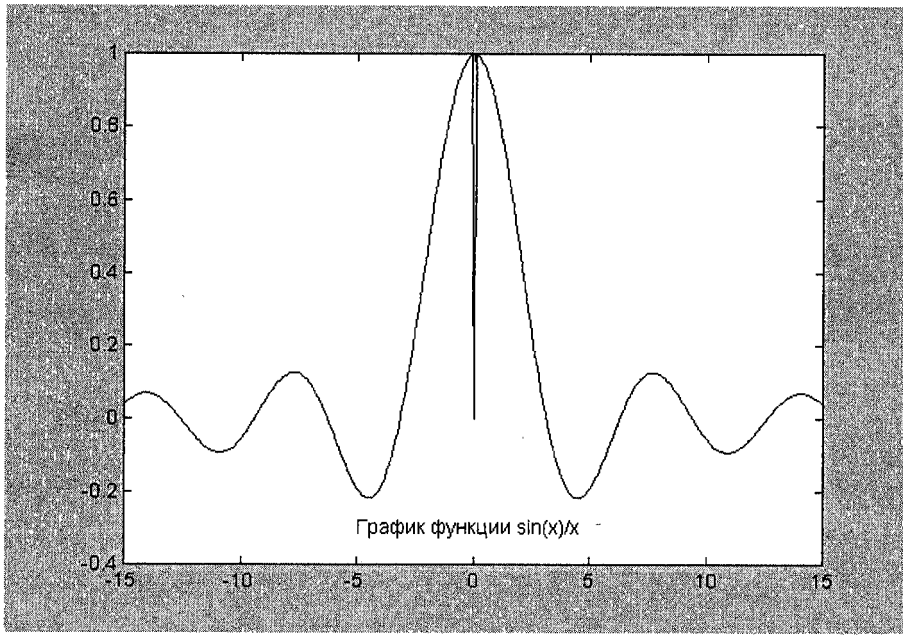


Рис. 4.35. Пример ввода надписи в поле графика функции

Функция **text** возвращает вектор-столбец свойств объекта **text**, порождаемого объектом axes.

Пары **X,Y** или **X,Y,Z**-тройки для 3D должны следовать за параметрами/значениями пар чтобы точно определить дополнительные свойства текста. Пары **X,Y** или **X,Y,Z**-тройки для 3D могут быть опущены полностью, а все свойства точно определяются при использовании пар параметров/значений.

Используйте **get(H)**, где **H** — текстовой указатель, чтобы просмотреть список свойств текстовых объектов и их текущие значения. Используйте **set(H)**, чтобы просмотреть список свойств текстовых объектов и допустимые значения свойств.

4.6.4. Ввод текста на график с помощью мыши — команда gtext

Очень удобный способ ввода текста представляет команда **gtext**:

gtext('string') — задает выводимый на график текст в виде строковой константы 'string' и выводит на график перемещаемый мышью маркер в виде крестика. Установив маркер в нужное место, для вывода текста достаточно нажать любую клавишу мышки.

gtext(C) — позволяет аналогичным образом устанавливать несколько надписей из массива строковых переменных C.

Пример применения команды **gtext**:

```
» %3D-график с надписью внутри окна
» x=-15:0.1:15;
» plot(x, sin(x)./(x+eps))
» gtext('Функция sin(x)/x')
```

При исполнении этого примера вначале можно увидеть построение графика функции с большим крестом, перемещаемым мышкой (рис. 4.36).

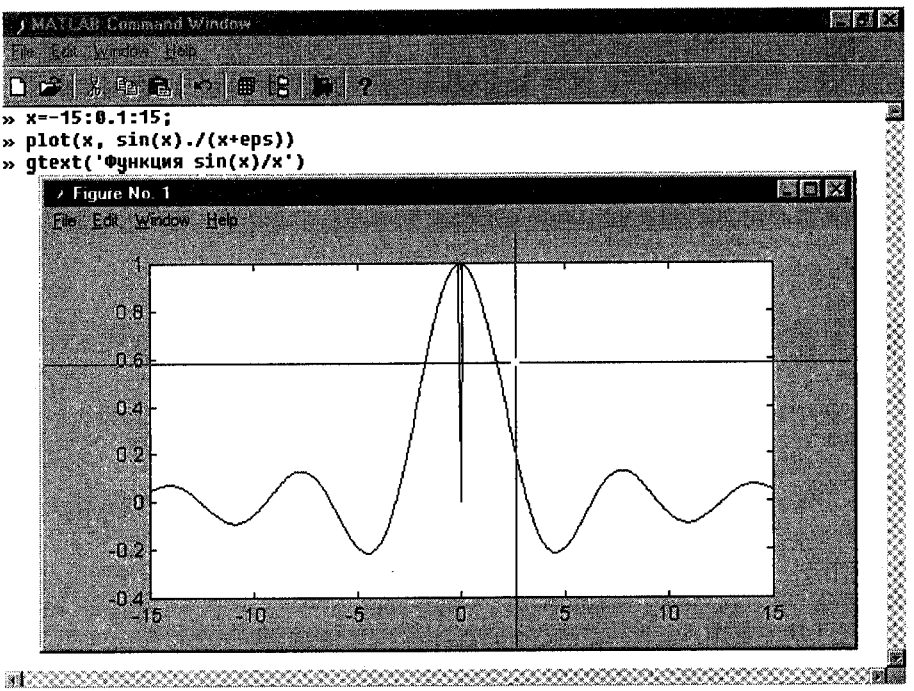


Рис. 4.36. График функции с крестом, перемещаемым мышкой

После того, как перекрестие установлено в определенное место графика, достаточно нажать любую клавишу, и на этом месте появится надпись (рис. 4.37).

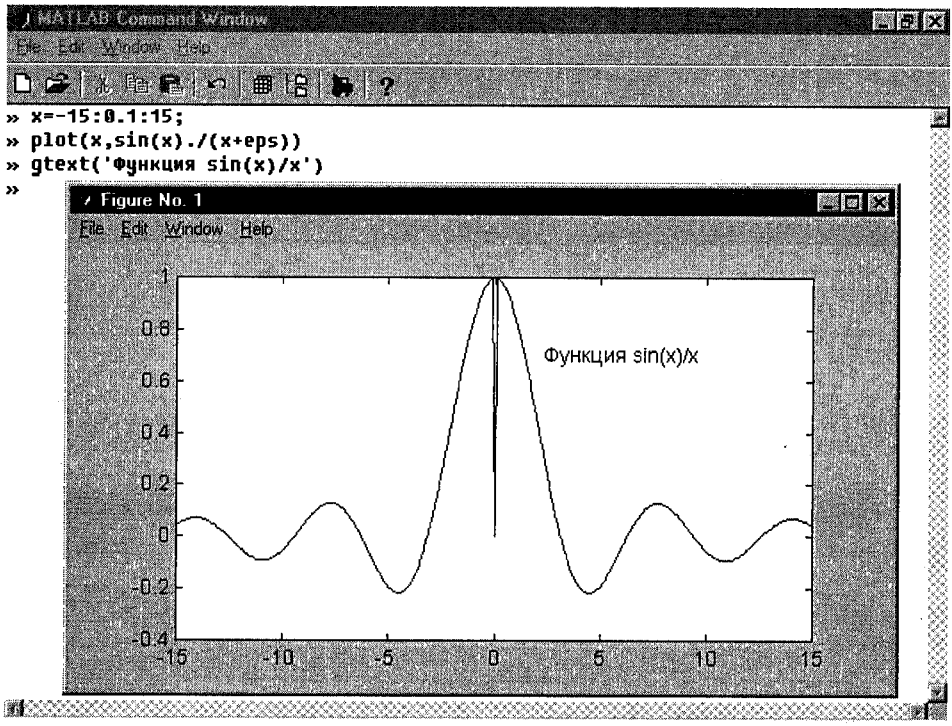


Рис. 4.37. График функции с установленной с помощью мыши надписью

Высокая точность позиционирования надписи и быстрота процесса делает данный способ нанесения надписей на графики одним из наиболее удобных.

4.6.5. Вывод пояснения — команда legend

Пояснение в виде отрезков линий со справочными надписями, размещаемое внутри графика или около него, называется **легендой**. Для создания легенды используются различные варианты команды **legeng**:

legend(string1,string2,string3, ...) — добавляет к текущему графику легенду в виде указанных в списке параметров строк.

legend(H,string1,string2,string3, ...) — помещает легенду на график, используя определенные строки как метки для соответствующих дескрипторов.

legend(AX,...) — помещает легенду на оси с дескриптором AX.

legend(M) — помещает легенду по осям, используя данные из матрицы M со строковыми данными.

legend OFF — устраняет ранее выведенную легенду.

legend — возобновляет текущую легенду, если присутствуют многочисленные легенды.

legend(legendhandle) — обновляет точно указанную легенду.

legend(...,Pos) — помещает легенду в точно определенное место, заданное позицией:

0 = лучшее место, выбираемое автоматически
1 = верхний правый угол
2 = верхний левый угол
3 = нижний левый угол
4 = нижний правый угол
-1 = справа от графика

Чтобы перенести легенду, установите на нее курсор, нажмите левую клавишу мыши и перетащите легенду в необходимую позицию. Функция

[legh,objh]=legend(...) — возвращает дескриптор к легенде и матрицу **objh** размера n×2, содержащую соответствующий текстовой дескриптор на осях легенды.

Команда **legend** может использоваться с двумерной и трехмерной графикой и со специальной графикой — столбиковыми и круговыми диаграммами и так далее. Двойным щелчком левой клавиши мыши можно вывести легенду на редактирование.

Пример, приведенный ниже

```
» % Пример построения графика с легендой в его окне
» x=-2*pi:0.1*pi:2*pi;
» y1=sin(x);
» y2=sin(x).^2;
» y3=sin(x).^3;
» plot(x,y1,'-m',x,y2,'-.+r',x,y3,'--ok')
» legend('Функция 1','Функция 2','Функция 3');
```

строит график трех функций с легендой, размещенной в поле графика. Этот график представлен на рис. 4.38.

Незначительная модификация команды **legend** (применение дополнительного параметра -1) позволяет строить график трех функций с легендой вне поля графика:

```
x=-2*pi:0.1*pi:2*pi;
y1=sin(x);
y2=sin(x).^2;
y3=sin(x).^3;
plot(x,y1,'-m',x,y2,'-.+r',x,y3,'--ok')
legend('Функция 1','Функция 2','Функция 3',-1);
```

Соответствующий график показан на рис. 4.39.

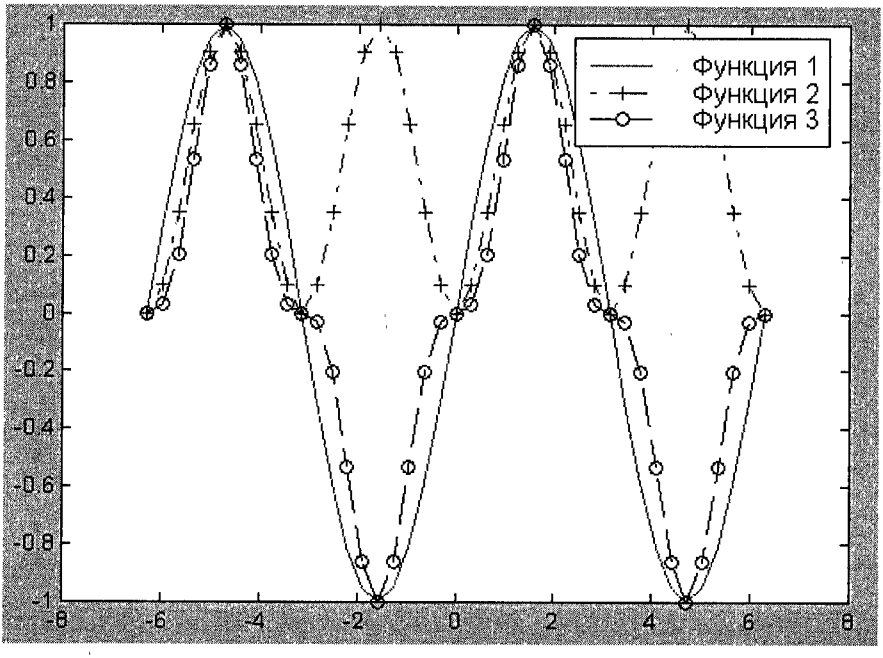


Рис. 4.38. График трех функций с легендой в поле графика

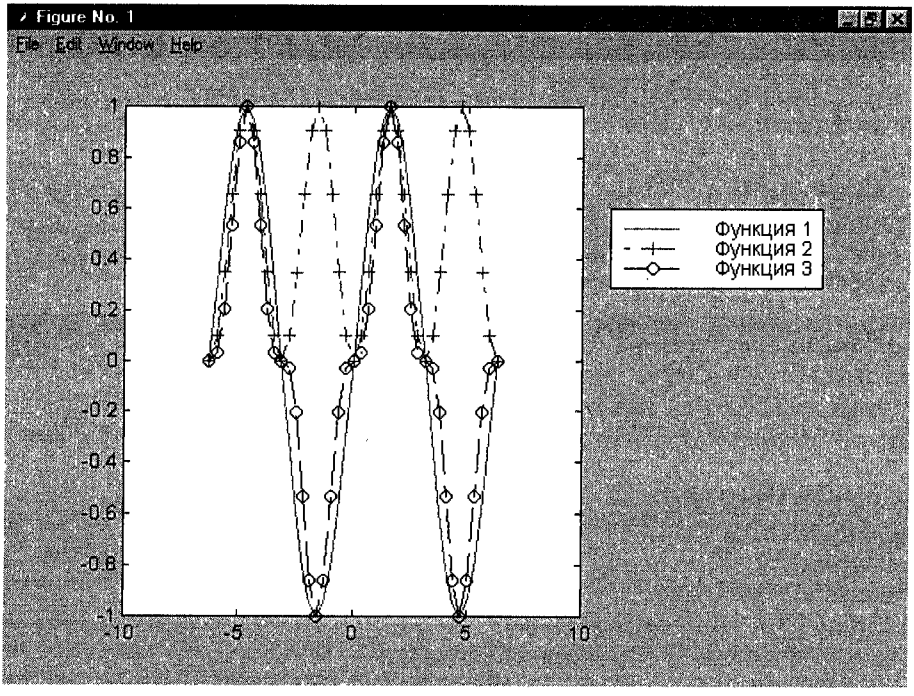


Рис. 4.39. График трех функций с легендой, расположенной вне поля графика

В данной случае недостатком можно считать сокращение рабочей площади самого графика. Остальные варианты расположения легенды пользователю предлагается изучить самостоятельно. Следует отметить, что применение легенды придает графикам более осмысленный и профессиональный вид.

4.6.6. Маркировка линий равного уровня — команда clabel

К сожалению, контурные графики плохо приспособлены для количественных оценок, если их линии не маркированы. В качестве маркеров используются крестики, рядом с которыми располагаются значения высот. Для маркирования контурных графиков используются команды группы **clabel**:

clabel(CS,H) — маркирует контурный график с данными в контурной матрице **CS** высотами, заданными в массиве **H**. Метки вращаемы и вставлены внутри контурных линий.

clabel(CS,H,V) — маркируется только тот контурный уровень, который указан в векторе **V**. По умолчанию маркируются все известные контуры. Позиции меток располагаются случайным образом.

clabel(CS,H,'manual') — маркирует контурные графики с установкой положения маркеров с помощью мыши. Нажатие клавиши Enter или клавиши мыши завершает установку маркера. При отсутствии мыши для перехода с одной линии уровня к другой используется клавиша пробела, а для перемещения надписи используются клавиши перемещения курсора.

clabel(CS), **clabel(CS,V)** и **clabel(CS,'manual')** — дополнительные возможности маркировки контурных графиков.

Пример применения команды **clabel** приводится ниже:

```
» % Построение контурного графика с маркированными линиями
[X,Y]=meshgrid([-3:0.1:3]);
Z=sin(X)./(X.^2+Y.^2+0.3);
C=contour(X,Y,Z,10);
colormap(gray)
clabel(C)
```

Рис. 4.40 показывает построение контурного графика с маркированными линиями уровня.

Функция

H=clabel(...)

возвращает дескриптор объекта **TEXT** (и, возможно, **line**).

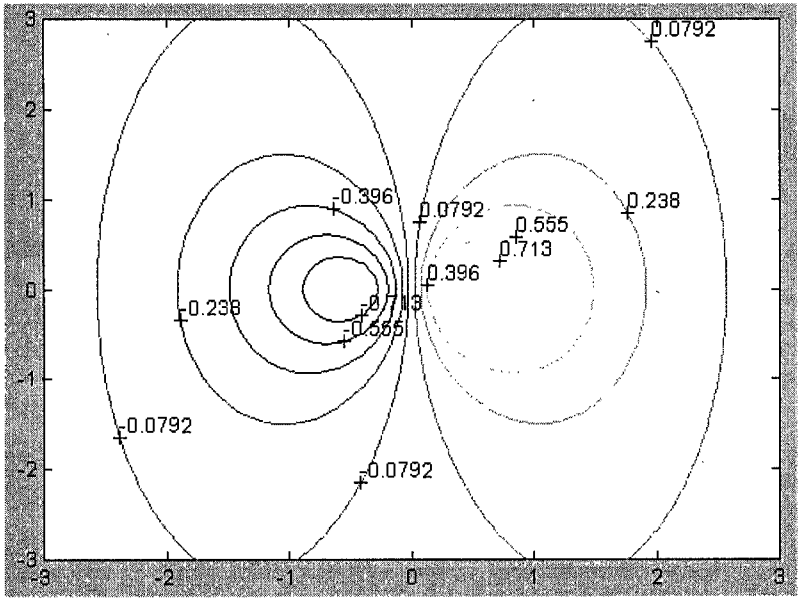


Рис. 4.40. Контурный график с маркированными линиями уровня

4.7. Дополнительные возможности изменения графиков

4.7.1. Маркировка осей графиков — команда axis

Обычно графики выводятся без указания наименований осей в режиме автоматического масштабирования. Следующие команды класса **axis** меняют эту ситуацию:

axis([XMIN, XMAX, YMIN, YMAX]) — установка масштабов по осям x и y для текущего двумерного графика.

axis([XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX]) — установка масштабов по осям x, y и z текущего трехмерного графика.

axis auto — установка параметров осей по умолчанию.

axis manual — замораживает масштабирование в текущем состоянии так, чтобы при **hold on** следующие строящиеся графики использовали то же состояние.

axis tight — устанавливает соответствие длины осей и максимального значения данных по осям.

axis ij — задает "матричную" систему координат с началом координат в левом верхнем углу.

axis xy — устанавливает Декартовую систему координат с горизонтальной осью x, размечаемой слева направо и вертикальной осью y, размечаемой снизу вверх. Начало координат размещается в нижнем левом углу.

axis equal — включает масштаб с одинаковым расстоянием между метками по осям x, y и z.

axis image — устанавливает масштаб, при котором пиксели изображения становятся квадратами.

axis square — устанавливает текущие оси в виде прямоугольника.

axis normal — восстанавливает масштаб, отменяя установки **axis equal** и **axis square**.

axis vis3d — задает свойства вращения для 3D-объектов.

axis off — убирает с осей их обозначения и маркеры.

axis on — восстанавливает ранее введенные обозначения осей и маркеры.

V=axis — возвращает вектор-строку, содержащую масштабирование для текущего графика. Если текущий график двумерный, то вектор имеет 4 компонента, если трехмерный — 6.

Следующий пример иллюстрирует применение команды **axis** при построении двумерного графика функции одной переменной:

```
» %Задание масштаба по осям x и y с помощью команды axis
» x=-5:0.1:5;
» plot(x,sin(x));
» axis([-10 10 -1.5 1.5])
```

На рис. 4.41 показан рисунок, который строится в этом примере.

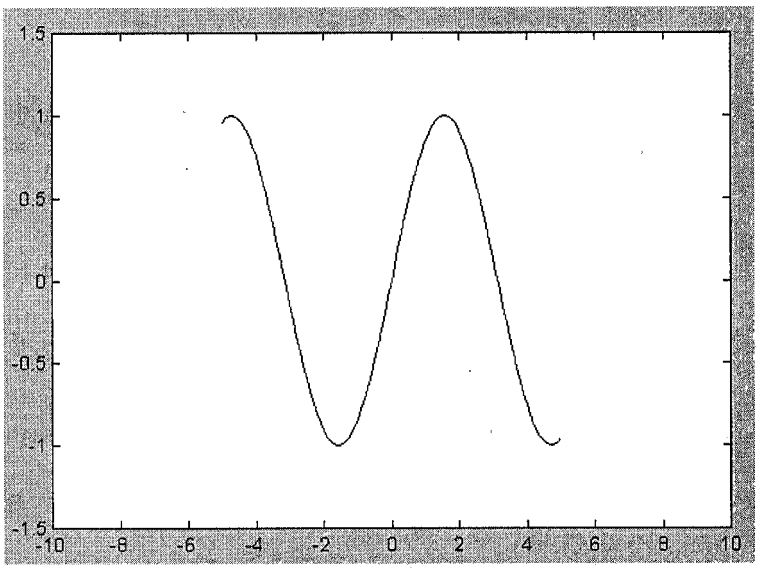


Рис. 4.41. Пример задания масштаба осей двумерного графика

Обратите внимание, что теперь масштабы осей заданы командой **axis**, а не диапазоном изменения значений x и y.

4.7.2. Включение и выключение сетки — команда grid

В математической, физической и иной литературе при построении графиков часто используют масштабную сетку в дополнение к разметке осей. Команда **grid** позволяют задавать построение сетки или отменять это построение:

grid on — добавляет сетку к текущему графику.
grid off — отключает сетку.
grid – последовательно производит включение и отключение сетки.

Команды **grid** устанавливают свойства объектов **Xgrid**, **Ygrid** и **Zgrid** для текущих осей. Ниже приведен пример из предшествующего раздела с добавлением в него команды **grid**:

```
» % График синусоиды с сеткой разметки
» x=-5:0.1:5;
» plot(x,sin(x));
» axis([-10 10 -1.5 1.5])
» grid on
```

Построенный график показан на рис. 4.42.

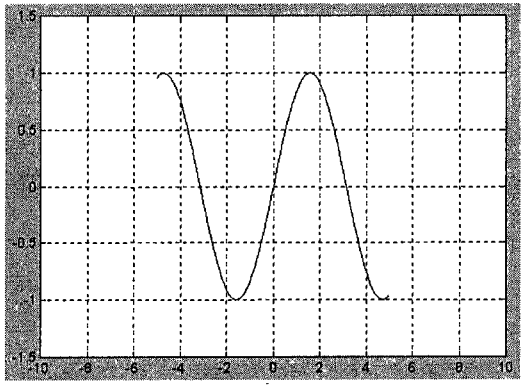


Рис. 4.42. График синусоиды с сеткой разметки

Сравните этот график с графиком на рис. 4.41, на котором сетка разметки отсутствует. Нетрудно заметить, что наличие сетки разметки облегчает количественную оценку координат каждой точки графика.

4.7.3. Продолжение построений графиков — команда hold

Во многих случаях желательно построение ряда наложенных друг на друга графиков, в одном и том же окне. Для этого используется команда продолжения графических построений **hold**. Она применима в следующих формах:

hold on — обеспечивает продолжение вывода графиков в текущее окно графики, что позволяет добавлять последующие графики к уже существующему.

hold off — отменяет режим продолжения графических построений.

hold — работает как переключатель, последовательно включая режим продолжения графических построений или отменяя его.

Команда **hold on** устанавливает значение **add** для свойства **NextPlot** объектов **figure** и **axes**, а **hold off** устанавливает значение **replace**. Рекомендуется ознакомиться с командами **ishold**, **newplot**, **figure** и **axes**.

Приведенный ниже пример показывает, как с помощью команды **hold on** на график синусоиды накладывается еще три графика параметрически заданных функций:

```
» %Построение трех графиков в одном окне с применением команды hold
» x=-5:0.1:5;
» plot(x,sin(x))
» hold on
» plot(sin(x),cos(x))
» plot(2*sin(x),cos(x))
» plot(4*sin(x),cos(x))
» hold off
```

Графики построенных функций показаны на рис. 4.43.

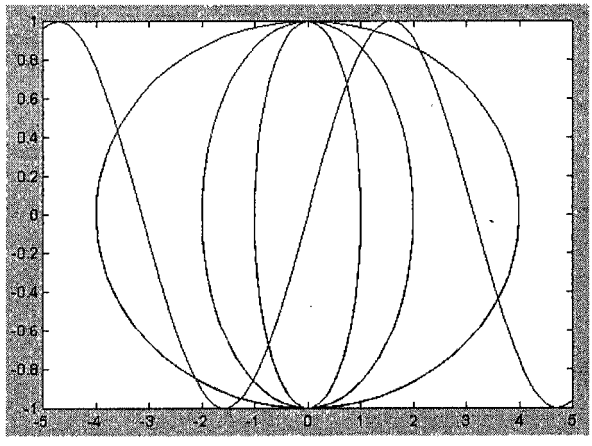


Рис. 4.43. Графики синусоиды и трех параметрических функций в одном окне

В конце приведенного фрагмента программы команда **hold off** отключает режим добавления графиков к ранее построенному графику.

4.7.4. Разбиение графического окна команда — subplot

Бывает, что в одном окне надо расположить несколько окон с различными графиками без наложения их друг на друга. Для этого используются варианты команды **subplot**, применяемые перед построением графика:

subplot — создает новые объекты **axes** (подокна).
subplot(m,n,p) или **subplot(mnp)** — разбивают графическое окно на m×n подокон, при этом m — число подокон по горизонтали, n — число подокон по вертикали и p — номер окна, в которое будет выводиться текущий график.
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 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**:

```
» % Построение в одном окне четырех графиков в разных подокнах
» 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. 4.44).

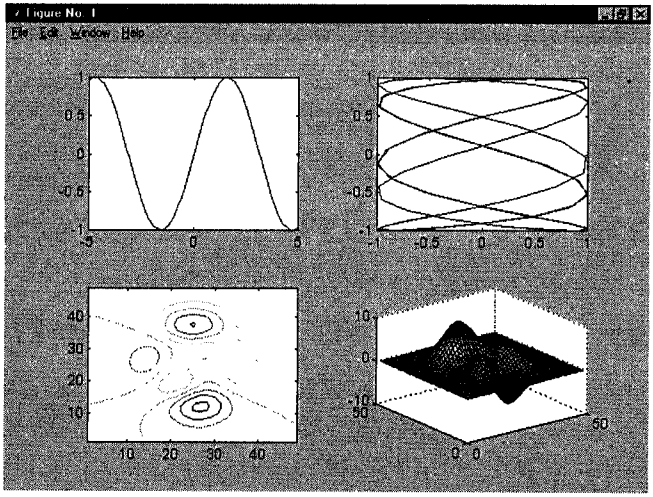


Рис. 4.44. Четыре графика различного типа, размещенных в подокнах одного окна

Следует отметить, что для всех графиков возможна индивидуальная установка дополнительных объектов, например, титульных надписей, надписей по осям и так далее.

4.7.5. Изменение масштаба графика — команда zoom

Для изменения масштаба двумерных графиков используются команды класса **zoom**:

zoom — устанавливает стандартный масштаб.
zoom(FACTOR) — устанавливает масштаб в соответствии с фактором FACTOR.
zoom on — включает режим изменения масштаба для текущего графика.
zoom off — выключает режим изменения масштаба для текущего графика.
zoom out — обеспечивает полный просмотр.
zoom xon или **zoom yon** — включают режим изменения масштаба только по оси x или по оси y.
zoom reset — отключает режим просмотра.
zoom(FIG,OPTION) — применяется к фигуре, точно определенной как FIG, при этом OPTION может быть любой из определенных выше аргументов.

Команда **zoom** позволяет управлять масштабированием графика с помощью мыши. Для этого надо подвести курсор мыши к интересующей вас области рисунка. Если команда **zoom** включена (**on**), то нажатие левой клавиши увеличивает масштаб вдвое, а правой — уменьшает вдвое. При нажатой левой клавише мыши можно выделить пунктирным черным прямоугольником нужный участок графика — при отпускании клавиши он появится в увеличенном виде и в том масштабе, который соответствует выделяющему прямоугольнику.

Рассмотрим работу команды **zoom** на следующем примере:

```
» % Применение команды zoom
» x=-5:0.01:5;
» plot(x,sin(x.^5)./(x.^5+eps))
» zoom on
```

Рис. 4.45 показывает график функции данного примера в режиме выделения его участка с помощью мыши.

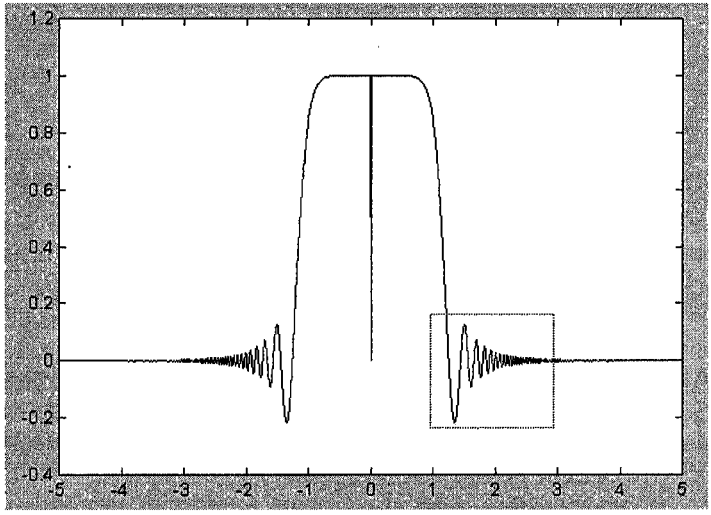


Рис. 4.45. Выделение части графика мышкой при использовании команды **zoom**

После опускания левой клавиши мыши график примет вид, показанный на рис. 4.46. Теперь в полный размер графического окна будет развернуто изображение, попавшее в выделяющий прямоугольник.

Для удобства отсчета на каждой линии устанавливается маркер в виде креста. К нему и относится указанная цифрами высота линии.

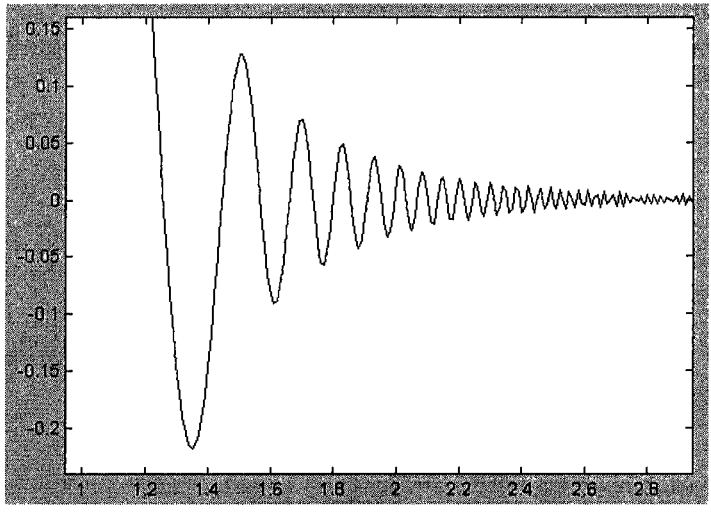


Рис. 4.46. График выделенного участка

Команда **zoom**, таким образом, выполняет функции "лупы", позволяющей наблюдать в увеличенном виде отдельные фрагменты сложных графиков. Однако следует

учитывать, что для наблюдения при большом увеличении фрагментов графиков, они должны быть заданы большим количеством точек. Иначе вид отдельных фрагментов и тем более особых точек (в нашем случае это точка при x вблизи нуля) будет существенно отличаться от истинного.

4.8. Управление цветом и световыми эффектами

4.8.1. Установка палитры цветов — команда colormap

Поскольку графика MATLAB обеспечивает получение цветных изображений, в ней есть ряд команд по управлению цветом и различными световыми эффектами. Среди них важное место занимает установка палитры цветов. Она задается однострочной матрицей MAP из трех элементов, определяющих значения интенсивности красного (red), зеленого (green) и синего (blue) цветов. Их интенсивность задается в относительных единицах от 0.0 до 1.0. Например:

[0 0 0] задает черный цвет, [1 1 1] — белый цвет, [0 0 1] — синий цвет. При изменении интенсивности цветов в указанных пределах возможно задание любого цвета. Таким образом, цвет соответствует общепринятому формату RGB.

Для установки палитры цветов служит команда **colormap**, записываемая в следующих формах:

colormap('default') — устанавливает палитру по умолчанию, при которой распределение цветов соответствует радуге (красный, желтый, зеленый, голубой, синий, фиолетовый).
colormap(MAP) — устанавливает заданную MAP-палитру.
C=colormap — создает матрицу палитры цветов C.

M-файл с именем colormap устанавливает свойства цветов для текущей фигуры.
Команда **help graph3d** наряду с прочим выводит полный список характерных палитр, используемых графической системой MATLAB:

hsv — цвета радуги,
hot — чередование черного, красного, желтого и белого цветов,
gray — линейная палитра в оттенках серого цвета,
bone — серые цвета с оттенком синего,
copper — линейная палитра с оттенками меди,
pink — розовые цвета с оттенками пастели,
white — палитра белого цвета,
flag — чередование красного, белого, синего и черного цветов,
lines — палитра с чередованием цветов линий,
colorcube — расширенная палитра RGB,
jet — разновидность палитры HSV,
prism — призматическая палитра цветов,

cool — оттенки голубого и фиолетового цветов,
autumn — оттенки красного и желтого цветов,
spring — оттенки желтого и фиолетового цветов,
winter — оттенки синего и зеленого цветов,
summer — оттенки зеленого и желтого цветов.

Все эти палитры могут служить параметром команды **colormap**, например, **colormap(hsv)** фактически устанавливает то же, что и команда **colormap('default')**. Примеры на применение команды **colormap** будут приведены в следующих разделах.

4.8.2. Установка соответствия между палитрой цветов и масштабом осей — команда caxis

При использовании функциональной окраски важное значение имеет установка соответствия между палитрой цветов и масштабом координатных осей. Так, выбор ограниченного диапазона интенсивностей цветов может привести к тому, что цветовая гамма будет блеклой и функциональная закраска не будет достигать своих целей. С помощью команды **caxis** можно обеспечить соответствие между палитрой цветов и масштабом осей.

caxis(V) — с помощью двухэлементного вектора **V** со списком элементов [cmin cmax] устанавливает диапазон используемой палитры цветов для объектов **surface** и **patch** и команд **mesh**, **pcolor** и **surf**. Пиксели, цвета которых выходят за пределы [cmin cmax], не высвечиваются.
caxis('manual') — устанавливает шкалу цветов по текущему интервалу параметра, задающего цвет.
caxis('auto') — устанавливает типовое масштабирование шкалы цветов, при котором цвета, соответствующие **Inf** и **NaN**, отсекаются.

Функция **caxis** возвращает двухэлементный вектор с элементами [cmin cmax] для текущего светового эффекта. М-файл с именем **caxis** задает свойства CLim и CLimMode объекта **axes** (см. команду **help axes**).

4.8.3. Окраска трехмерных поверхностей — команда shading

Для окраски трехмерных поверхностей используется команда shading, которая управляет объектами **surface** и **path**, а также используется с командами и функциями **surf**, **mesh**, **pcolor**, **fill** и **fill3**. Команда **shading** работает с опциями и имеет следующий вид:.
shading flat — задает окраску ячеек или граней в зависимости от текущих данных.
shading interp — задает окраску с билинейной интерполяцией цветов.

shading faceted — равномерная раскраска ячеек поверхности (принята по умолчанию).

Эти команды устанавливают свойства **EdgeColor** и **FaceColor** для графических объектов **surface** и **patch** в зависимости от того, какая из команд **mesh** (сеточная поверхность) или **surf** (затененная поверхность) используется. Примеры применения команд **shading** уже приводились.

4.8.4. Установка палитры псевдоцветов — команда pcolor

Довольно часто возникает необходимость представления в цветах той или иной матрицы. Для этого используют псевдоцвета, зависящие от содержания ячеек. Такое представление реализует команда класса **pcolor**:

pcolor(C) — задает представление матрицы **C** в псевдоцвете.
pcolor(X,Y,C) — задает представление матрицы **C** в сетке, формируемой векторами или матрицами **X** и **Y**.

Функцию **pcolor** возвращает дескриптор объекта **surface**. Пример на применение команды **pcolor** приводится ниже:

```
» %Применение команды pcolor
» z=peaks(40);
» colormap(hsv)
» pcolor(z)
```

График, построенный в этом примере, показан на рис. 4.47.

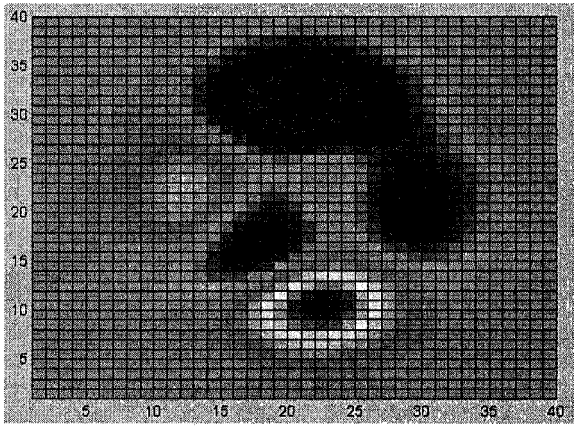


Рис. 4.47. Пример применения команды **pcolor**

Характер расцветки поверхности командой **pcolor** существенно зависит от выбора палитры цветов. В приведенном примере она задается командой **colormap**.

4.8.5. Создание закрашенного многоугольника — команда patch

Для создания закрашенного пятна в виде многоугольника может использоваться команда **patch**:

patch(X,Y,C) — создает закрашенный многоугольник, вершины которого заданы координатами в векторах **X** и **Y** в текущей системе координат со спецификацией окраски, заданной вектором цветовой палитры **C**. Можно также задавать цвет символьной переменной 'color' вида 'r','g','b','c','m','y', 'w', or 'k'. **X** и **Y** могут быть матрицами.

patch(X,Y,Z,C) — создает пятно в трехмерной системе координат, при этом матрица Z должна иметь тот же размер, что и **X** и **Y**.

Следующий пример поясняет применение команды **patch**:

```
» X=[1 2 3 2 1];
» Y=[1 2 0 5 1];
» patch(X,Y,[1 0 0])
```

Построенный многоугольник показан на рис. 4.48.

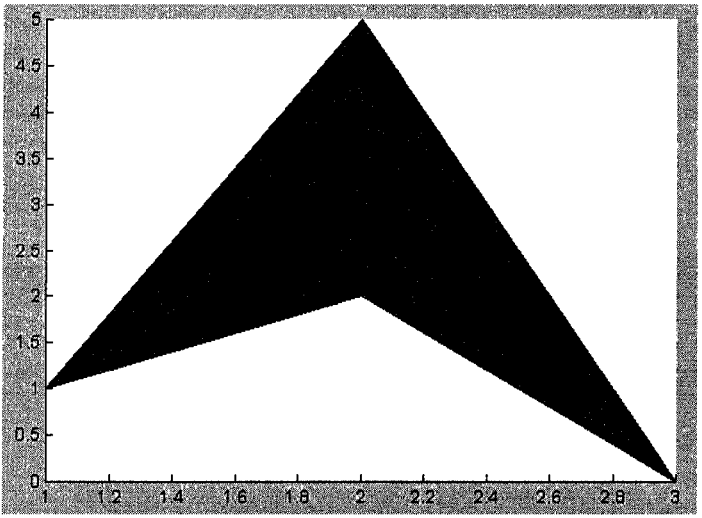


Рис. 4.48. Многоугольник, построенный командой **patch**

В данном случае многоугольник окрашен красным цветом, поскольку вектор цветов [1 0 0] указывает только на наличие красной составляющей цвета (другие составляющие представлены относительным уровнем 0).

4.8.6. Окраска плоских многоугольников — команда fill

Для построения окрашенных в заданный цвет плоских многоугольников может использоваться команда **fill**:

fill(X,Y,C) — строит закрашенный плоский многоугольник, вершины которого задаются векторами **X** и **Y** с цветом, заданным **C**. Многоугольник должен быть замкнутым. Для построения множества прямоугольников параметры команды должны быть матрицами.

fill(X1,Y1,C1,X2,Y2,C2,...) — представляет собой другой способ построения многих закрашенных прямоугольников.

Следующий пример показывает построение четырехугольника, закрашенного синим цветом:

```
» X=[1 2 3 2 1];
» Y=[5 0.5 0 4 5];
» fill(X,Y,[0 0 1])
```

Построения, реализованные этим примером, показаны на рис. 4.49.

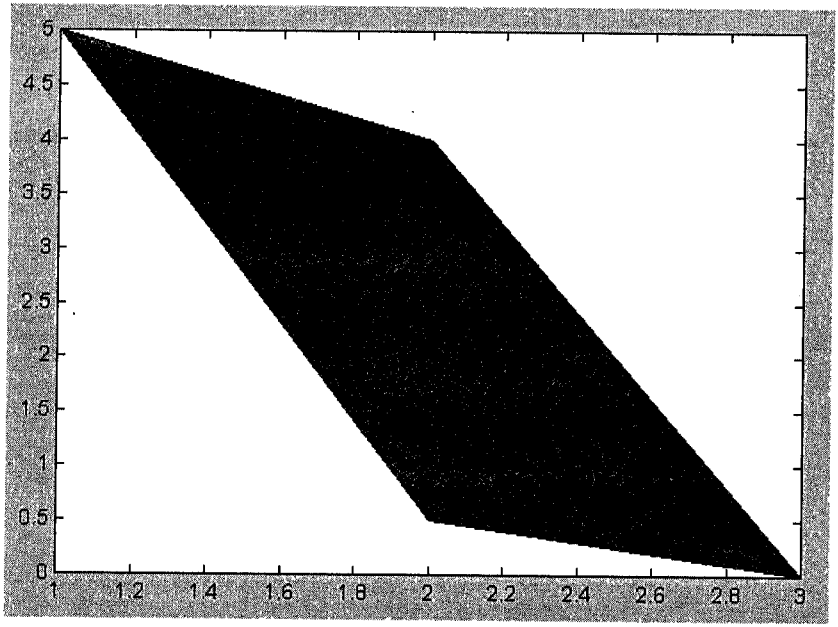


Рис. 4.49. Построение закрашенного четырехугольника на плоскости

Функция **H= fill(...)** возвращает вектор-столбец дескрипторов для объектов **path**, по одному дескриптору на каждый объект.

4.8.7. Вывод шкалы цветов — команда colorbar

При использовании функциональной окраски весьма полезным является вывод шкалы цветов командой **colorbar**. Ее варианты перечислены ниже:

colorbar('vert') — устанавливает вертикальную шкалу цветов на текущий график.
colorbar('horiz') — устанавливает горизонтальную шкалу цветов на текущий график.
colorbar(H) — устанавливает шкалу цветов на график с дескриптором H с автоматическим размещением шкалы по вертикали или по горизонтали в зависимости от соотношения размеров графика.
colorbar — устанавливает в текущий график новую вертикальную шкалу цветов или заменяет на новую уже имеющуюся.

Следующий пример показывает применение команды **colorbar** совместно с командой **fill3**:

```
» fill3(rand(5,4),rand(5,4),rand(5,4),rand(5,4))
» colorbar('vert')
```

Более подробно функция **fill3** будет рассмотрена ниже. На рис. 4.50 показана полученная при запуске этого примера картина. Следует отметить, что в случае, когда многоугольники строятся со случайными значениями координат вершин, при каждом пуске будет наблюдаться новая картина.

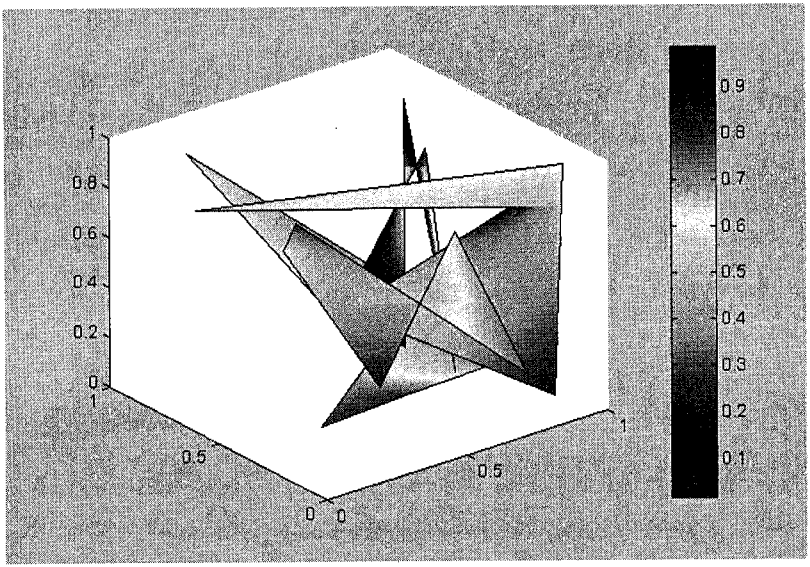


Рис. 4.50. Случайные многоугольники с функциональной окраской и вертикальной шкалой цветов

Функция **H=colorbar(...)** возвращает дескриптор для оси со шкалой цветов.

4.8.8. Цветные плоские круговые диаграммы — команда pie

Закрашенные секторы часто используются для построения круговых диаграмм. Для этого в MATLAB служит команда **pie**:

pie(X) — строит круговую диаграмму по данным нормализованного по площади вектора **X/SUM(X)**. Если **SUM(X)** <= 1.0, то значения в **X** непосредственно определяют площадь сектора.

pie(X,EXPLODE) — строит круговую диаграмму, у которой отрыв секторов задается вектором **EXPLODE**, который должен иметь ту же длину, что и вектор данных **X**.

В следующем примере строится цветная круговая диаграмма с пятью секторами, причем последний сектор отделен от остальных.

```
» % Построение плоской круговой диаграммы
» X=[1 2 3 4 5];
» pie(X,[0 0 0 0 2])
```

Построенная диаграмма показана на рис. 4.51.

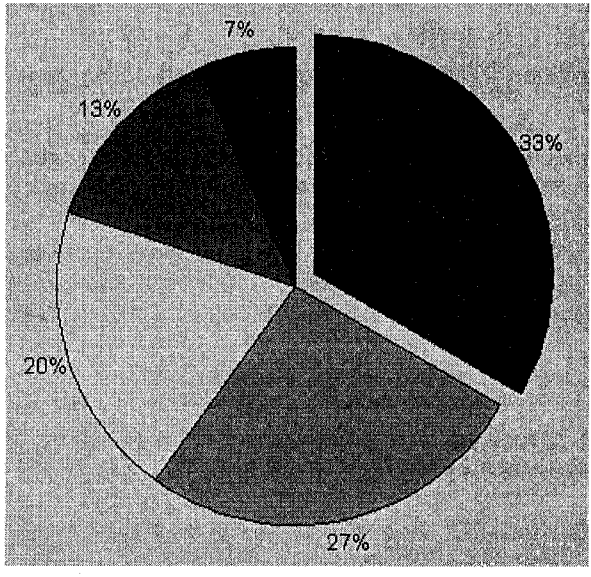


Рис. 4.51. Плоская круговая диаграмма

Функция **H=pie(...)** возвращает вектор дескрипторов объектов **patch** и **text**.

4.8.9. Другие команды управления световыми эффектами

Здесь мы только отметим некоторые дополнительные команды, связанные с управлением цветовыми палитрами:

colstyle — выделение цвета и стиля графика из заданного массива.
rgbplot — изображение палитры.
hsv3rgb — преобразование палитры **hsv** в палитру **rgb**.
rgb2hsv — преобразование палитры **rgb** в палитру **hsv**.
brighten — управление яркостью.
contrast — управление контрастом.
hidden — управление показом невидимых линий.
whiteng — управление цветом фона.

Рекомендуется ознакомиться с информацией об этих командах с помощью команды **help**.

4.9. Построение некоторых объемных фигур

4.9.1. Окрашенные многоугольники в пространстве — команда fill3

Для закраски многоугольников, определенных в пространстве, служит команда **fill3**. Ниже представлены основные ее формы:

fill3(X,Y,Z,C) — строит закрашенный многоугольник в пространстве с данными вершин, хранящимися в векторах **X**, **Y** и **Z** с цветом, заданным палитрой **C**. При построении ряда закрашенных многоугольников параметры команды должны быть матрицами.
fill3(X1,Y1,Z1,C1,X2,Y2,Z2,C2,...) — другой вариант построения множества закрашенных многоугольников в пространстве.
fill3 — функция, возвращающая вектор-столбец дескрипторов объекта patch.

Следующий пример показывает действие команды **fill3**:

```
» %Пример построения закрашенных многоугольников в пространстве
» fill3(rand(5,4),rand(5,4),rand(5,4),rand(5,4))
```

На рис. 4.52 представлены построенные по этому примеру закрашенные многоугольники. Поскольку многоугольники формируются с применением генератора случайных чисел для координат вершин, то наблюдаемая картина оказывается случайной и не будет повторяться при последующих пусках данного примера.

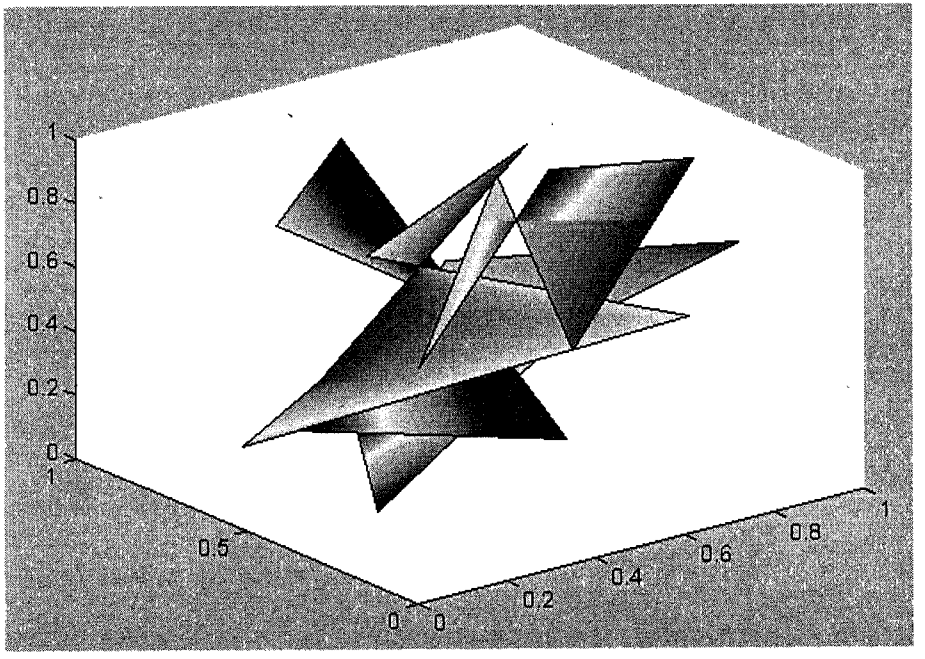


Рис. 4.52. Закрашенные многоугольники в пространстве

Следует обратить внимание на то, что команда **fill3** дает функциональную закраску построенных фигур.

4.9.2. Цветные объемные круговые диаграммы — команда pie3

Иногда используются объемные круговые диаграммы. Для их построения служит команда **pie3**:

pie3(...) — аналогична команде **pie(...),** но дает построение объемных секторов.

В приведенном ниже примере

```
» %Построение объемной круговой диаграммы
» X=[1 2 3 4 5];
» pie3(X,[0 0 1 0 1])
```

строится объемная диаграмма с отделением двух секторов, показанная на рис. 4.53.

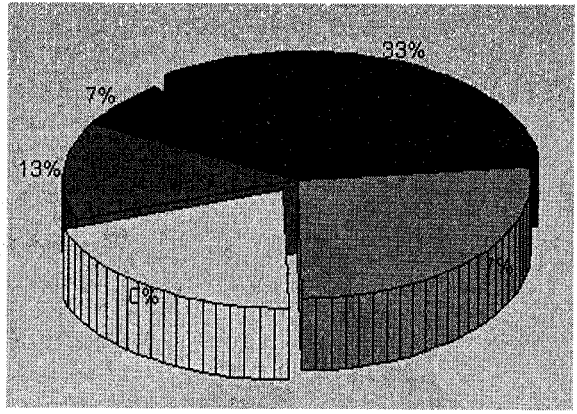


Рис. 4.53. Объемная круговая диаграмма

Функция **H=pie3(...)** возвращает вектор, содержащий **patch**, **surface** и **text**-дескрипторы.

4.9.3. Построение объемного цилиндра — функция cylinder

Для построения объемного цилиндра применяется функция:

[X,Y,Z]=cylinder(R,N) — создает массивы **X**, **Y** и **Z**, описывающие цилиндрическую поверхность с радиусом **R** и числом узловых точек N для построения с помощью функции **surf(X,Y,Z)**.

[X,Y,Z]=cylinder(R) и **[X,Y,Z]=cylinder** — подобны предшествующей записи функции для N=20 и **R=[1 1]**.

Пример построения объемного цилиндра:

```
» %Построение объемного цилиндра
» [X,Y,Z]=cylinder(10,30);
» surf(X,Y,Z,X)
```

На рис. 4.54 показано построение цилиндра для данного примера:

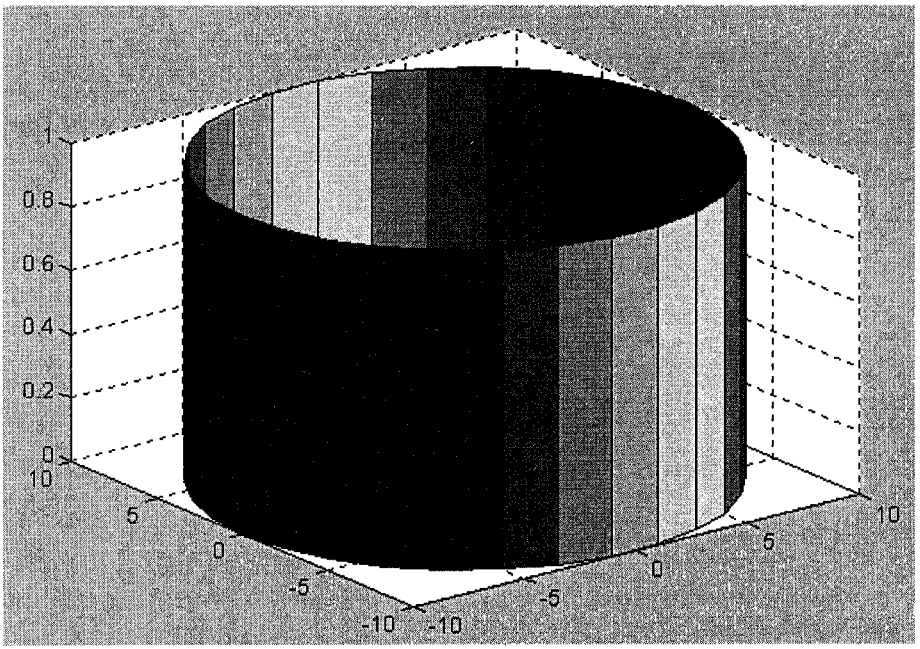


Рис. 4.54 Построение цилиндра

Естественность воспроизведения цилиндра существенно зависит от графической команды, используемой для его построения. Команда **surf** дает возможность задать функциональную окраску с цветом, определяемым вектором **X**, что делает представление цилиндра достаточно наглядным.

4.9.4. Построение объемной сферы — функция sphere

Для расчета массивов **X**, **Y** и **Z** объемной сферы используется функция:

[X,Y,Z]=sphere(N) — генерирует матрицы **X**, **Y** и **Z** размера (n+1)×(n+1) для построения сферы с помощью команд **surf(X,Y,Z)** и **surfl(X,Y,Z)**.
[X,Y,Z]=sphere — аналогична предшествующей функции при N = 20.

Пример применения этой функции:

```
» %Построение объемной сферы
» [X,Y,Z]=sphere(30);
» surfl(X,Y,Z)
```

На рис. 4.55 показана построенная по этому примеру сфера

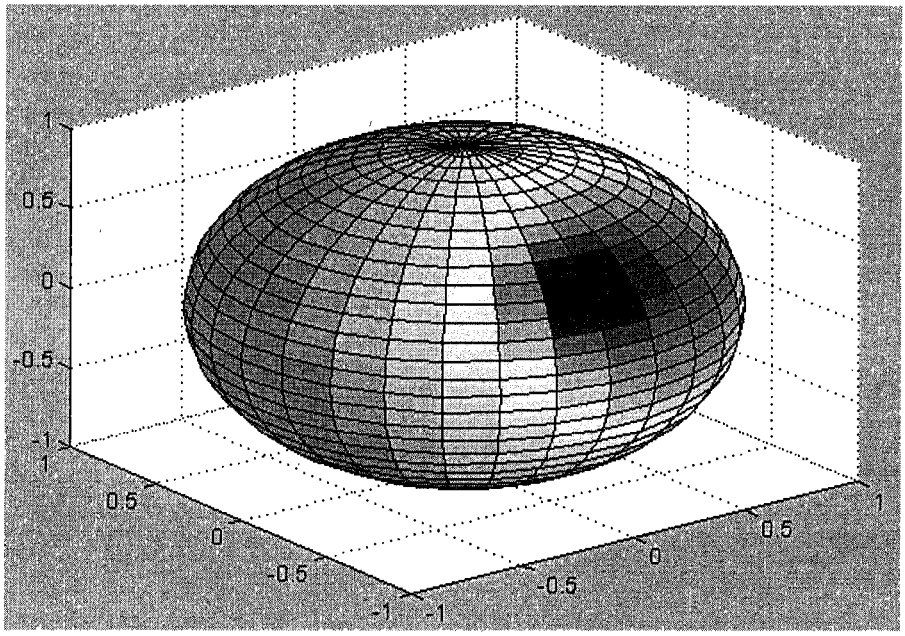


Рис. 4.55. Построение сферы

Обратите внимание на то, что именно функциональная окраска сферы придает ей довольно реалистический вид. В данном случае цвет задается вектором **Z**.

4.9.5. Трехмерная графика с треугольными плоскостями — команды trimesh и trisurf

К числу специальных видов графики относится построение объемных фигур с помощью плоских треугольников. Для построения таких фигур в виде каркаса (без окраски и отображения плоскостей) используется команда **trimesh**:

trimesh(TRI,X,Y,Z,C) — построение объемной каркасной фигуры с треугольниками, специфицированными матрицей поверхности **TRI**, строки которой индексируются векторами **X**, **Y**, **Z**, а цвета ребер вектором **C**.

trimesh(TRI,X,Y,Z) — построение, аналогичное предшествующему при **C=Z**, то есть с цветом ребер, зависящим от значений высоты.

H=trimesh(...) — возвращает дескрипторы графического объекта.

trimesh(...,'param','value','param','value'...) — добавляет значения 'value' для параметров 'param'.

Следующий пример иллюстрирует применение команды **trimesh** для построения случайной объемной фигуры, параметры которой задаются с помощью генератора случайных чисел:

```
» x = rand(1,40);
» y = rand(1,40);
» z = sin(x.*y);
» tri = delaunay(x,y);
» trimesh(tri,x,y,z)
```

Одна из построенных фигур показана на рис. 4.56.

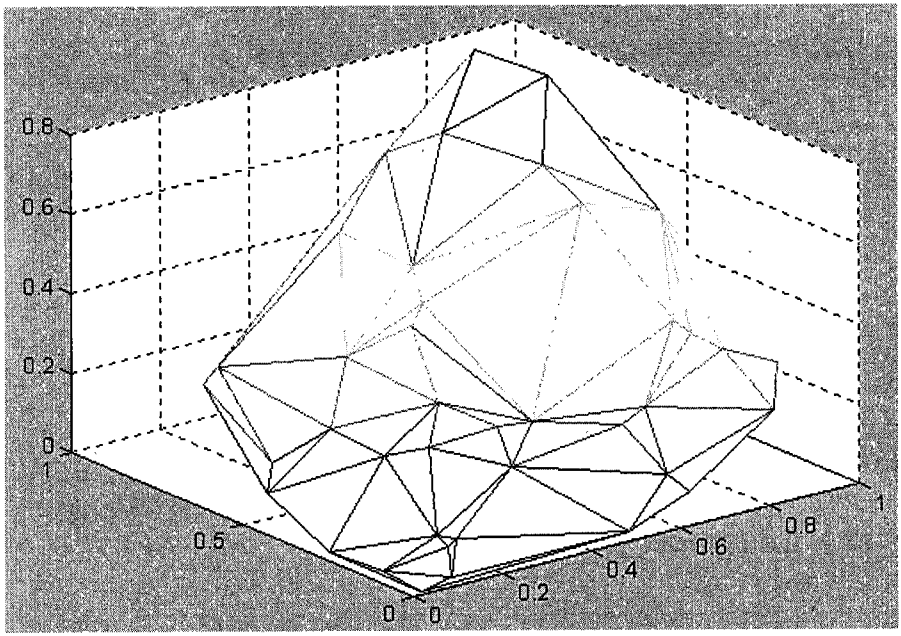


Рис. 4.56. Одна из объемных фигур, построенных командой **trimesh**

Другая, абсолютно аналогичная по заданию входных параметров команда **trisurf(...)**, отличается только закраской треугольных областей, задающих трехмерную фигуру. Если в приведенном примере заменить функцию **trimesh** на **trisurf**, то можно получить рисунок, подобный приведенному ниже:

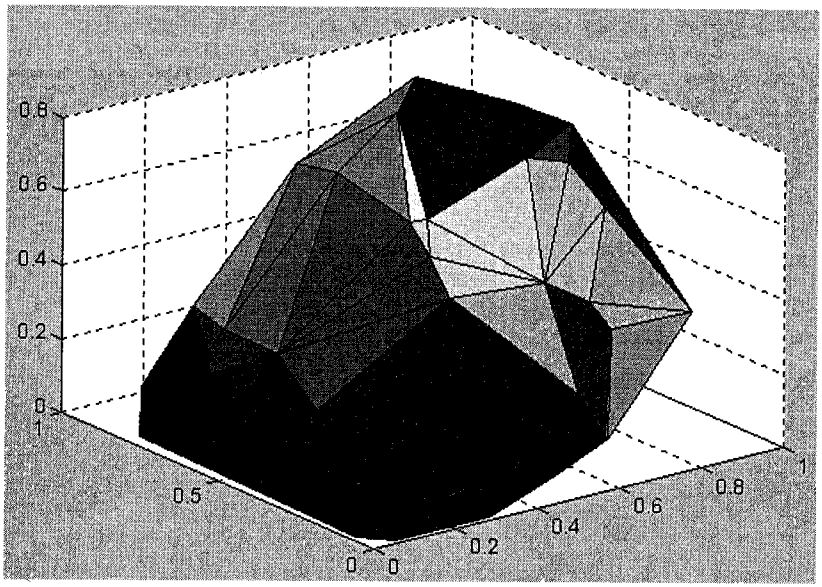


Рис. 4.57. Один из рисунков, построенных командой **trisurf**

Обратите внимание на то, что изображение на рис. 4.57 также принадлежит к множеству случайных графических построений. Поэтому возможность его буквального повторения отсутствует.

4.10. Анимационная графика

4.10.1. Движение точки по плоскости — команда comet

Для отображения движения точки по траектории используется команда **comet**. При этом движущаяся точка напоминает ядро кометы с хвостом. Используются следующие формы представления этой команды:

comet(Y) — отображает движение "кометы" по траектории, заданной вектором **Y**.
comet(X,Y) — отображает движение "кометы" по траектории, заданной парой векторов **Y** и **X**.
comet(X,Y,p) — аналогична предшествующей команде, но позволяет задавать длину хвоста кометы как p*length(Y). По умолчанию is p = 0.1.

Следующий пример иллюстрирует применение команды **comet**:

```
» X=0:0.01:15;
» comet(X,sin(X),0.15)
```

Стоп-кадр изображения показан на рис. 4.58.

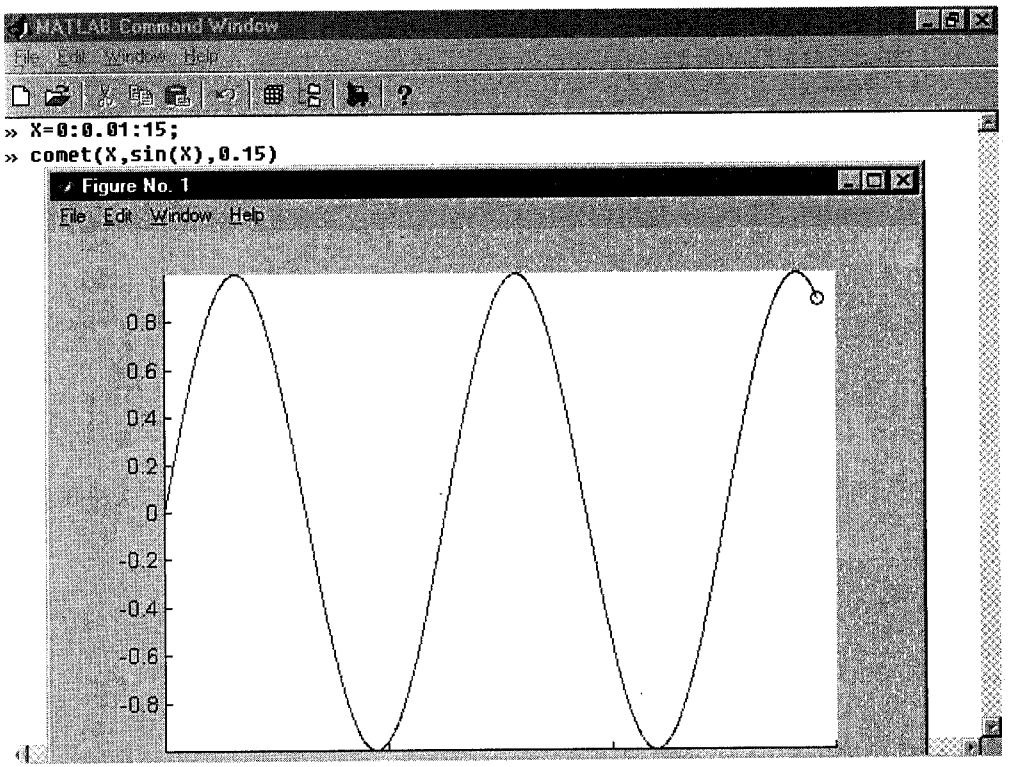


Рис. 4.58. Стоп-кадр изображения, полученный из примера использования команды **comet**.

"Хвост кометы" на черно-белом рисунке заметить трудно, поскольку он представляет собой отрезок линии с цветом, отличающимся от цвета линии основной части графика.

4.10.2. Движение точки в пространстве — команда comet3

Еще одна команда позволяет наблюдать движение точки, но уже в трехмерном пространстве. Это команда **comet3**:

comet3(Z) — отображает движение точки с цветным "хвостом" по трехмерной кривой, определенной массивом.

comet3(X,Y,Z) — отображает движение точки — "кометы" — по кривой в пространстве, заданной точками **[X(i),Y(i),Z(i)]**.

comet3(X,Y,Z,p) — аналогична предшествующей команде с заданием длины "хвоста кометы" **p*length(Z)**. По умолчанию параметр p = 0.1.

Ниже представлен пример применения команды **comet3**:

```
» W=0:pi/500:10*pi;
» comet3(cos(W),sin(W)+W/10,W)
```

А на рис. 4.59 показан стоп-кадр изображения, которое строится с помощью указанного приведенного примера.

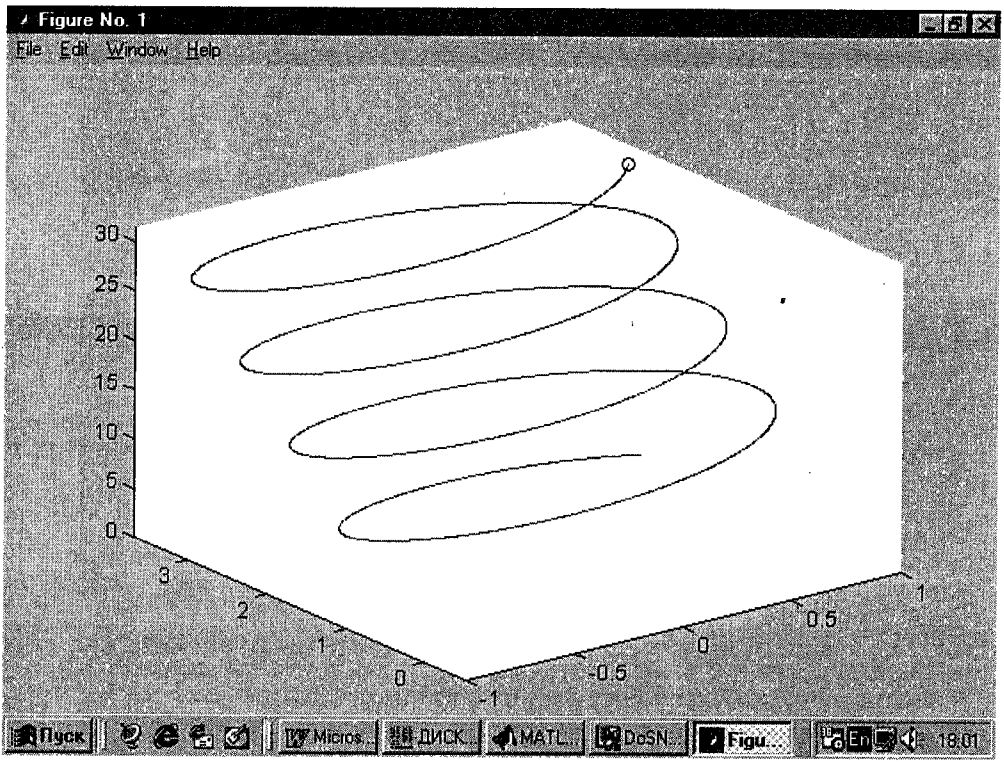


Рис. 4.59. Стоп-кадр изображения, созданного командой **comet3**

Разумеется, движение точки по заданной траектории как в двумерном, так и трехмерном пространстве дает представление о самых простейших примерах анимации. Тем не менее они существенно расширяют графическую визуализацию при решении ряда динамических задач.

4.10.3. Основные средства анимации

Для более сложных случаев анимации возможно применение техники мультипликации. Она сводится к построению ряда кадров изображения, причем каждый кадр появляется на некоторое время, затем стирается и заменяется на новый кадр, несколь-

ко отличающийся от предшествующего. Если это отличие незначительно, то создается иллюзия плавного перемещения объекта.

Отметим кратко основные команды, реализующие анимацию в системе MATLAB:

capture — захват видеоизображения.
getframe — создание фрейма для анимации.
moviein — выполнение анимации.
rotate — вращение фигуры.
frame2im — преобразование фрейма в графический образ.
im2frame — преобразование графического образа во фрейм.

Применение некоторых из этих команд мы рассмотрим далее на конкретных примерах. К сожалению, серьезные задачи по анимации обычно требуют применения программных средств — главным образом циклов. Мы рассмотрим их далее, но представляется, что читатели знакомы с понятием циклов, так что приведенные примеры не будут слишком сложны. В крайнем случае оставьте их разбор до знакомства с основами программирования системы MATLAB.

4.10.4. Вращение фигуры — логотипа MATLAB

Рассмотрим вначале не очень сложный пример вращения сложной трехмерной поверхности — логотипа системы MATLAB, которая представлена файлом logo.m. Ниже представлен фрагмент программы, обеспечивающий вращение этой поверхности (фигуры) относительно осей системы координат:

```
if ~exist('MovieGUIFlag'), figNumber=0; end;
load logo
h=surfl(L,source);
colormap(M);
ax=[0 60 0 60 -0.5 1];
ax=[7 52 7 52 -.5 .8];
axis(ax);
axis on;
shading interp;
m=moviein(25);

for n=1:25,
  rotate(h,[0 90],15,[21 21 0]);
  h=surfl(get(h,'XData'),get(h,'YData'),get(h,'ZData'),source);
  axis(ax);
  axis on;
  shading interp;
  m(:,n)=mvframe(figNumber,24);
end;
mvstore(figNumber,m);
```

Эта программа имеет два блока: в первом задаются исходная функция и ее образ, а во втором (с циклом **for**) выполняется создание фреймов и их последовательное воспроизведение, создающее эффект анимации. На рис. 4.60 показан стоп-кадр этой программы.

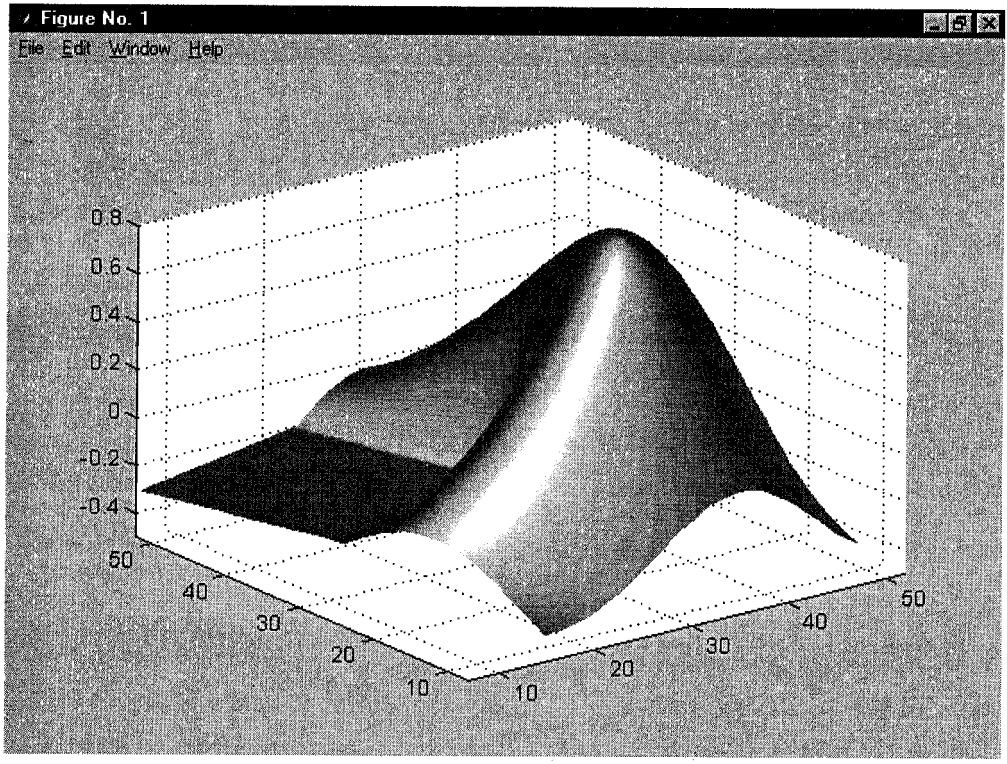


Рис. 4.60. Стоп-кадр программы, обеспечивающей вращение фигуры — логотипа MATLAB

4.10.5. Волновые колебания мембраны

Принцип мультипликации легко распространить на существенно более сложные задачи анимации. В качестве примера можно рассмотреть системный пример vibes, демонстрирующий волнообразные колебания тонкой пластины — мембраны. Ниже представлен переработанный файл данного примера, в котором сокращены подробные комментарии на английском языке и введены краткие комментарии на русском:

```
%Волновые колебания мембраны
%Переработка файла VIBES фирмы MathWorks, Inc.
if ~exist('MovieGUIFlag'), figNumber=0; end;
```

```
hlpStr= ...
      [' Это пример анимации - наблюдение колебаний '
       ' трехмерной поверхности - мембраны.        '];
mvinit(figNumber,hlpStr);

% Загрузка данных функции
load vibesdat
[n,n] = size(L1);
nh = fix(n/2);
x = (-nh:nh)/nh;

% Вычисление коэффициентов
clear c
for k = 1:12
   eval(['c(k) = L' num2str(k) '(24,13)/3;'])
end

% Установка графических параметров
axis([-1 1 -1 1 -1 1]);
caxis(26.9*[-1.5 1]);
colormap(hot);
hold on

% Генерация кадров мультипликации
delt = 0.1;
nframes = 12;
M = moviein(nframes);
for k = 1:nframes,
   % Коэффициенты
   t = k*delt;
   s = c.*sin(sqrt(lambda)*t);

   % Амплитуды
   L = s(1)*L1 + s(2)*L2 + s(3)*L3 + s(4)*L4 + s(5)*L5 + s(6)*L6 + ...
      s(7)*L7 + s(8)*L8 + s(9)*L9 + s(10)*L10 + s(11)*L11 + s(12)*L12;

   % Скорость
   s = s .* lambda;
   V = s(1)*L1 + s(2)*L2 + s(3)*L3 + s(4)*L4 + s(5)*L5 + s(6)*L6 + ...
      s(7)*L7 + s(8)*L8 + s(9)*L9 + s(10)*L10 + s(11)*L11 + s(12)*L12;

   % График поверхности, цвет задается скоростью
   V(1:nh,1:nh) = NaN*ones(nh,nh);
   cla
   surf(x,x,L,V);
```

```
    axis off

    % Создание кадров мультипликации
    M(:,k) = mvframe(figNumber,nframes);
end;
hold off
%====================================
% Запись кадров мультипликации
mvstore(figNumber,M);
```

Этот пример является типичной программой на языке программирования системы MATLAB. Поскольку он дан с целью иллюстрации, подробно эту программу мы описывать не будем. Ограничимся приведением заключительного кадра анимации (рис. 4.61), показывающего колебания мембраны на фоне координатных осей.

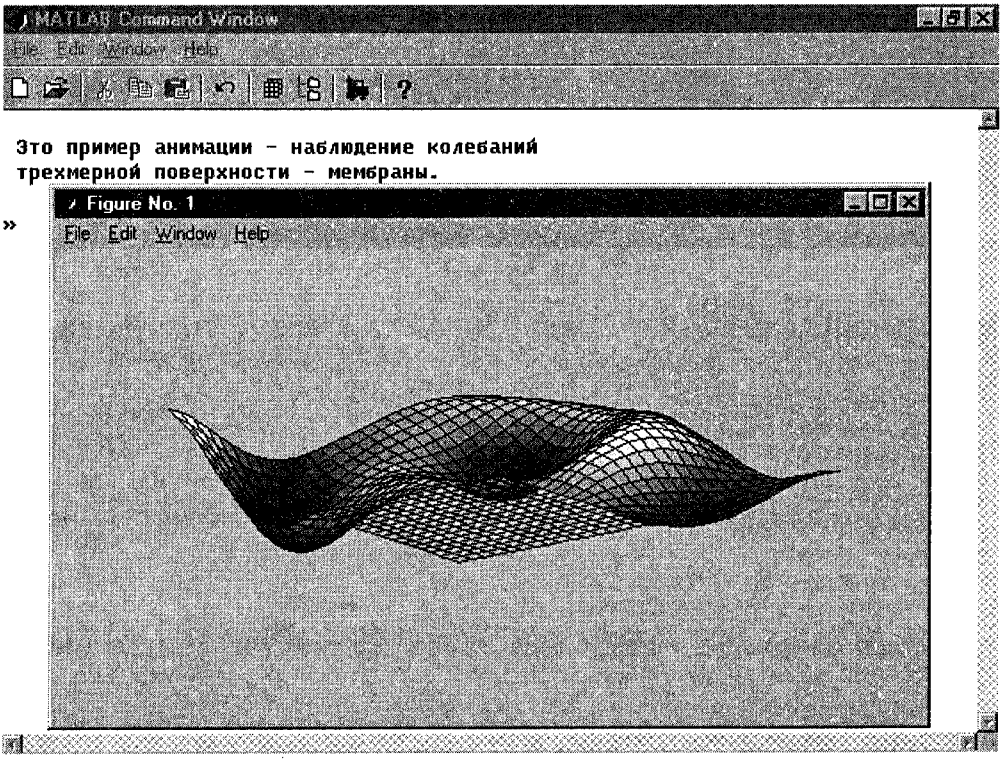


Рис. 4.61. Заключительный кадр анимации, демонстрирующей колебания мембраны

Итак, мы рассмотрели большую часть графических возможностей системы MATLAB 5.2.1. За пределами рассмотрения остались отдельные, редко используемые команды и дескрипторная графика, в основном ориентированная на поддержку графического интерфейса системы. Опытные пользователи ПК могут воспользоваться гра-

фическими средствами системы для создания собственных элементов интерфейса — кнопок, панелей, окон и так далее.

4.11. Введение в низкоуровневую дескрипторную графику

4.11.1. Объекты дескрипторной графики

Как уже отмечалось, графические средства MATLAB базируются на низкоуровневой графике, которая называется дескрипторной (описательной). По существу эта графика обеспечивает объектно-ориентированное программирование как всех рассмотренных выше графических команд, так и пользовательского интерфейса. Хотя обычный пользователь может забыть о существовании этого вида графики, все же надо учитывать, что она дает подчас новые и уникальные возможности в создании пользовательских графических программ, не говоря уже о том, что она помогает понять, каким образом реализованы пользовательские средства графики системы.

Центральным понятием дескрипторной графики является **графический объект**. Имеются следующие типы таких объектов:

Root — первичный объект, соответствующий экрану компьютера.
Figure — объект создания фигуры.
Uicontrol — объект создания пользовательского интерфейса.
Axes — объект создания области фигуры в окне.
Uimenu — объект создания меню.
Image — объект создания BitMap-графики.
Line — объект создания линии.
Patch — объект создания закрашенных фигур.
Surface — объект создания поверхности.
Text — объект создания текстовых надписей.
Light — объект создания эффектов освещенности.

Объекты подчас взаимосвязаны и могут обращаться друг к другу для получения того или иного графического эффекта.

4.11.2. Создание графического окна и управление им

Прежде чем мы рассмотрим применение дескрипторной графики на реальных примерах, отметим команды и функции, которые предназначены для создания графических окон и управления ими:

figure — открыть чистое графическое окно для построения фигуры.
gcf — получить дескриптор графического окна figure.

clf — очистить графическое окно.
shg — показать ранее свернутое графическое окно.
close — закрыть графическое окно.
refresh — обновить графическое окно.

Эти команды и функции достаточно очевидны, и мы не будем обсуждать их подробно. Заметим, что команда **help name** позволяет уточнить назначение той или иной команды или функции с обобщенным именем name.

4.11.3. Создание координатных осей и управление ими

Еще одна группа простых команд служит для создания координатных осей и управления ими:

axes — создать оси координат.
box — построить прямоугольник вокруг рисунка.
cla — убрать построения **axes**.
gca — получить дескриптор графического объекта **axes**.
hold — сохранить оси координат.
ishold — проверить, есть ли оси (дает 1 в этом случае).

Эти команды также достаточно очевидны. Их применение в обычной графике задается командами высокого уровня, но при желании данные команды можно использовать и в обычной графике, например, для устранения осей из уже созданного графика.

4.11.4. Пример применения объекта дескрипторной графики

Объем и направленность данной книги не позволяют подробно описать все многообразие возможностей дескрипторной графики. Ограничимся пока одним примером. Надо построить линию, проходящую через три точки с координатами (0,1), (2,4) и (5,-1). Для этого воспользуемся объектом **line**, который порождает графическую функцию:

```
» line([0 2 5],[1 4 -1],'Color','blue')
```

На рис. 4.62 построена заданная линия с помощью дескрипторной команды **line**, которая явно не входит в высокоуровневую графику. Однако нетрудно понять, что именно эта команда составляет основу высокоуровневой команды **plot**, описанной ранее.

Особенность команды **line** заключается в явном задании всех условий построения графика: координат конкретных точек, опции цвета 'Color' и самого цвета 'blue' (синий). В итоге строятся два отрезка прямой, проходящие через заданные точки и имеющие синий цвет.

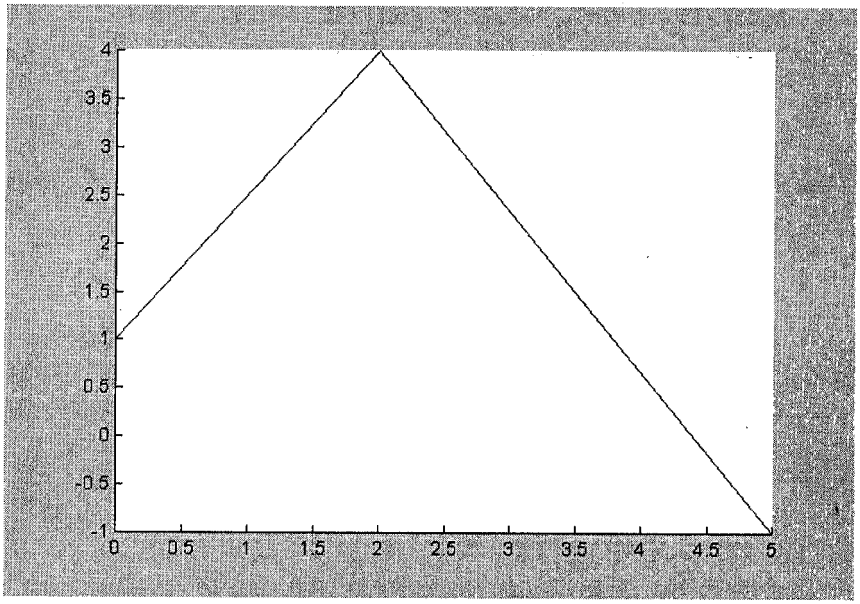


Рис. 4.62. Построение отрезков прямой объектом **line**

4.11.5. Дескрипторы объектов

С понятием объектов дескрипторной графики связана их особая характеристика — **дескриптор**. Его можно понимать, как некое число — своеобразный распознаватель объектов. Дескриптор объектов **Root** всегда равен 0, а дескриптор объектов **Figure** — это целое число, указывающее на номер графического окна. Дескрипторы других объектов — это числа с плавающей точкой. По их значениям MATLAB идентифицирует объекты. Дескриптор одного такого объекта представляет собой одно число, а если объектов несколько — несколько чисел. Например, следующие команды строят пять графиков, представляющих значения элементов магической матрицы, в одном окне:

```
» A=magic(5);
» h=plot(A)
h =
   6.0020
   1.0107
   3.0093
   4.0040
   5.0029
```

В данном случае вектор **h** содержит элементы-дескрипторы графика, показанного на рис. 4.63.

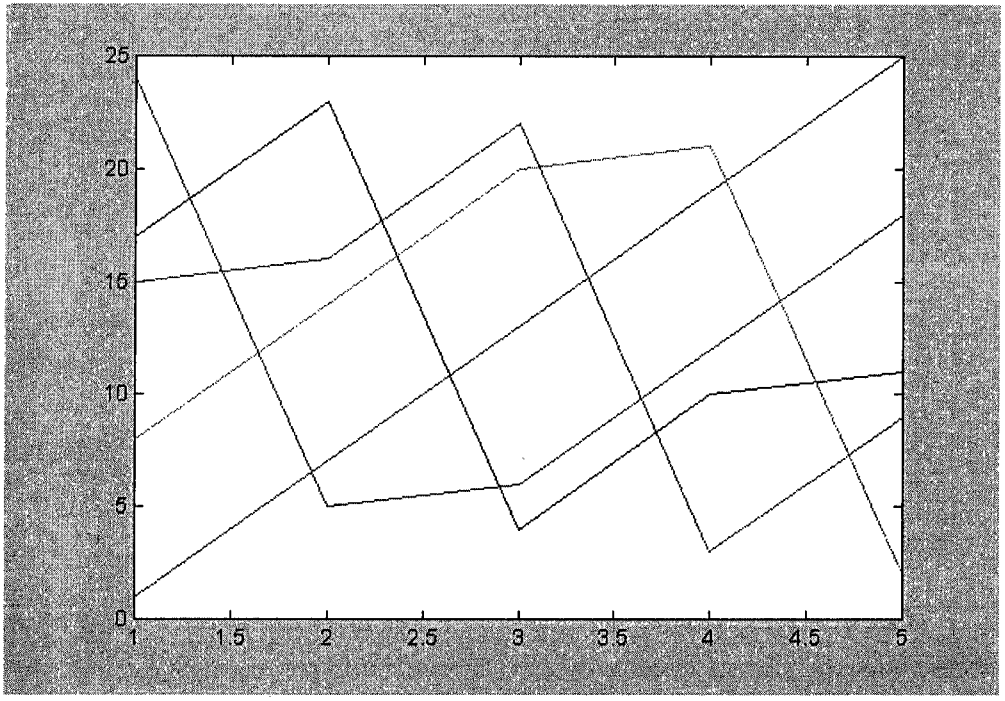


Рис. 4.63. График пяти функций, представляющих значения элементов магической матрицы **magic(5)**

Мы еще раз обращаем ваше внимание на то, что дескрипторы дают лишь внутреннее описание того или иного объекта, и ассоциировать его с привычными параметрами, например, координатами или цветом объекта, не следует.

4.11.6. Операции над графическими объектами

К графическим объектам применяется ряд операций:

set — установить свойства (опции) графического объекта.
get — вывести свойства графического объекта.
reset — восстановить свойства графического объекта по умолчанию.
delete — удалить созданный графический объект.
gco — вывести дескриптор текущего графического объекта.
gcbo — вывести дескриптор повторно вызванного объекта.
gcbf — вывести дескриптор повторно вызванного окна.
drawnow — выполнить очередь задержанных графических команд.

findobj — найти объекты с заданными свойствами.
copyobj — скопировать объект и порожденные им объекты.

Кроме того, имеются три утилиты, связанные с операциями над объектами:

closereq — закрыть окно по запросу.
ishandle — проверить дескриптор на истинность.
newplot — восстановить измененные свойства объекта NextPlot.

Действие большинства этих операций достаточно очевидно. Мы остановимся на двух, наиболее важных операциях, связанных с контролем и установкой свойств объектов.

4.11.7. Свойства объектов — команда get

Каждый объект дескрипторной графики имеет множество опций, определяющих его свойства. Вернемся к нашему примеру с построением графика из двух отрезков линии и повторим этот пример в следующем виде:

```
» h=line([0 2 5],[1 4 -1],'Color','blue')
h =
   1.0017
```

Теперь объект имеет дескриптор h, и его значение выведено наряду с построением графика. Команда **get(name)** выводит свойства объекта с заданным именем. Для нашего объекта это выглядит следующим образом:

```
» get(h)
    Color = [0 0 1]
    EraseMode = normal
    LineStyle = -
    LineWidth = [0.5]
    Marker = none
    MarkerSize = [6]
    MarkerEdgeColor = auto
    MarkerFaceColor = none
    XData = [0 2 5]
    YData = [1 4 -1]
    ZData = [ ]

    ButtonDownFcn =
    Children = [ ]
    Clipping = on
    CreateFcn =
    DeleteFcn =
    BusyAction = queue
```

```
HandleVisibility = on
HitTest = on
Interruptible = on
Parent = [2.00122]
Selected = off
SelectionHighlight = on
Tag =
Type = line
UIContextMenu = [ ]
UserData = [ ]
Visible = on
```

4.11.8. Изменение свойств объекта — команда set

С помощью команды **set** можно изменить отдельные свойства объекта дескрипторной графики. Эта команда имеет множество опций, и с ними можно ознакомиться с помощью команды **help set**. Ограничимся примером — допустим, нам надо сменить цвет линии с голубого на красный. Для этого достаточно выполнить команду:

```
» set(h,'Color','red')
```

Обратите внимание, что при этом цвет сменится на последнем, ранее построенном рисунке.

4.11.9. Примеры, иллюстрирующие возможности дескрипторной графики

Теперь рассмотрим более сложные примеры, наглядно демонстрирующие возможности дескрипторной графики. Создадим файл ms1:

```
[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** — класса **figure**, оси с метками **ah** — класса **axes** и трехмерная поверхность **sh** — класса **surface**. Теперь создадим второй файл ms2:

```
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.

При первом запуске такого файла ms1 появится плоская сетка, показанная на рис. 4.64. Она является результатом наложения объектов **fh** и **ah** друг на друга. При этом объект **ah** класса axes явно наследует свойства объекта **fh** класса **figure**.

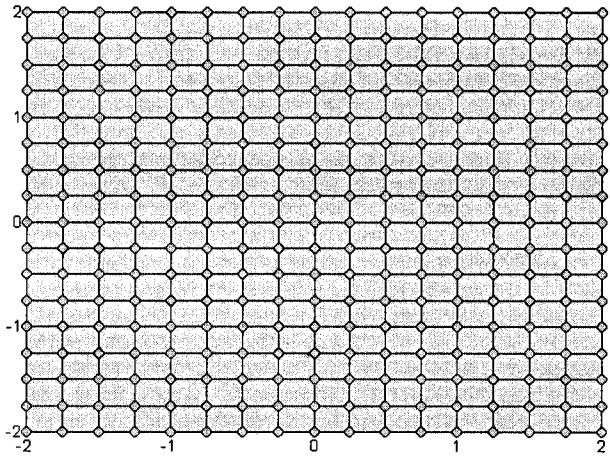


Рис. 4.64. Результат наложения объектов **fh** и **ah** друг на друга

Пока ничего неожиданного в полученной двумерной фигуре нет. Но стоит исполнить команду

```
» view(3)
```

как будет получен весьма любопытный рисунок, представленный на рис. 4.65. Нетрудно заметить, что полученная трехмерная поверхность наследует узловые точки

сетки, показанной на рис. 4.64. Таким образом здесь в явной форме проявляется такое качество объектов, как наследование свойств, производных от родительских объектов.

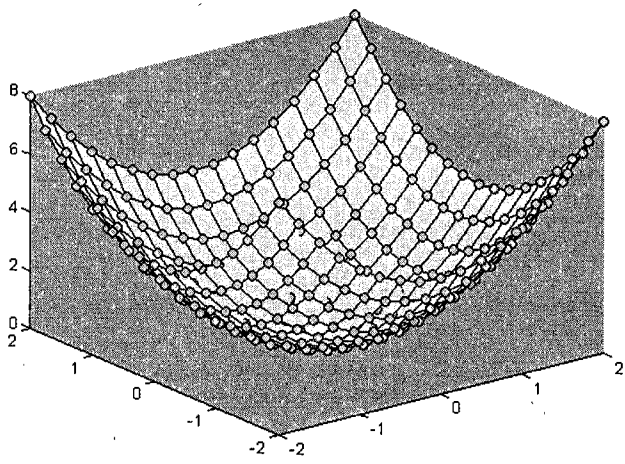


Рис. 4.65. Трехмерная поверхность, унаследовавшая узловые точки плоской фигуры

Еще более интересную картину мы получим, запустив файл ms2. Полученный в том же окне комбинированный рисунок показан на рис. 4.66. Он является результатом наложения новых построений трехмерных фигур на ранее построенный рисунок (рис. 4.63).

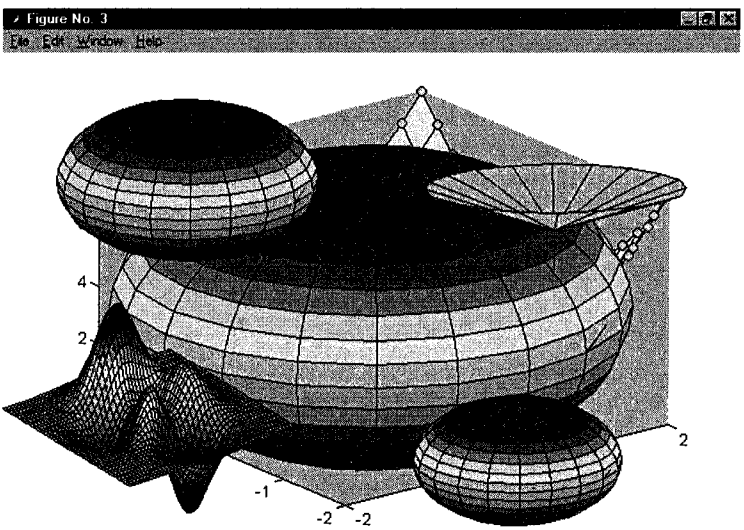


Рис. 4.66. Комбинированный рисунок, полученный при запуске файлов ms1 и ms2

Чтобы понять, какие из объектов наследуют свойства других объектов, следует рассмотреть диаграмму иерархии объектов дескрипторной графики MATLAB, представленную на рис. 4.67. Например, из нее видно, что объекты **Surface** расположены ниже объектов **Axes**, а те, в свою очередь, расположены ниже объектов класса **Figure**. Поэтому ясно, что в случае пуска файла ms1 свойства сетки, построенной с применением объекта axes, будут унаследованы объектом sh, построенным командой **Surface**.

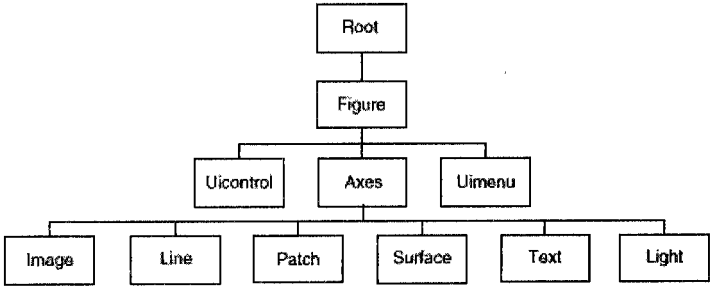


Рис. 4.67. Иерархия объектов дескрипторной графики системы MATLAB

Все объекты второго файла относятся к классу **Axes**. Именно поэтому они строятся поверх объектов, показанных на рис. 4.64. Координаты всех пяти трехмерных фигур (рис. 4.66) заданы жестко в соответствующих командах **Axes**. Любопытен вид цилиндра — похоже, что произошедшее с ним преобразование связано с изменением системы координат с Декартовой на сферическую.

В заключении этого раздела следует еще раз отметить, что дескрипторная графика рассчитана не столько на конкретных пользователей, использующих MATLAB как прикладную программу, сколько на опытных разработчиков программного обеспечения для этой системы. Разумеется, это не исключает полезную возможность изменения параметров графиков изменением свойств их объектов, что особенно наглядно видно из последних приведенных примеров (рис. 4.66). Многие тайны дескрипторной графики познаются только в ходе практических экспериментов с ней.

4.12. Создание элементов пользовательского интерфейса

4.12.1. Основные команды для создания пользовательского интерфейса

Опытные пользователи нередко использовали систему MATLAB для создания своих собственных систем. Этому во многом способствует идеология системы — хранение большей части команд и функций в виде М-файлов. Простота коррекции файлов и отсутствие необходимости особо объявлять о создании новых команд и функций

привели к созданию множества программных систем на базе MATLAB, особенно в таких областях, как решение задач линейной алгебры, моделирование различных систем и структур и так далее.

В новой версии MATLAB дескрипторная графика позволяет конструировать детали пользовательского интерфейса. Полный список команд и функций для проектирования пользовательского интерфейса можно получить, выполнив команду **help uitools**.

4.12.2. Перечень команд и функций пользовательского интерфейса

Ниже перечислены все команды и функции данного назначения.

Функции пользовательского интерфейса GUI

uicontrol — создание управляющего элемента.
uimenu — создание пользовательского меню.
ginput — графический ввод с помощью мышки.
dragrect — создание выделяющего прямоугольника с помощью мышки.
rbbox — растяжка прямоугольника мышкой.
selectmoveresize — интерактивная селекция, перемещение и копирование объектов с помощью мышки.
waitforbuttonpress — ожидание нажатия клавиши клавиатуры или мышки в окне.
waitfor — прекращение выполнения в ожидании события.
uiwait — прекращение выполнения в ожидании события.
uiresume — возобновить выполнение после блокировки.
uisuspend — прекращение интерактивного состояния фигуры.
uirestore — возобновление интерактивного состояния фигуры.

Средства проектирования пользовательского интерфейса

guide — создание GUI.
align — выровнять положение объектов интерфейса.
cbedit — изменение повторного вызова объектов.
menuedit — изменение меню.
propedit — изменение свойств объектов.

Средства создания диалоговых панелей

dialog — создание диалоговой панели.
axlimdlg — ограничение размеров диалоговой панели.
errordlg — создание панели ошибок.
helpdlg — создание панели справки.
inputdlg — ввод диалоговой панели.
listdlg — создание панели селекции.

menu — создание меню диалогового ввода.
msgbox — создание панели сообщений.
questdlg — создание диалоговой панели вопросов.
warndlg — создание диалоговой панели предупреждений.
uigetfile — создание стандартной панели открытия файлов.
uiputfile — создание стандартной панели записи файлов.
uisetcolor — создание панели выбора цвета.
uisetfont — создание панели выбора набора шрифтов.
pagedlg — создание диалоговой панели положения страницы.
printdlg — создание диалоговой панели печати.
waitbar — создание панели ожидания.

Создание меню

makemenu — создание структуры меню.
menubar — устанавка типовых свойств для MenuBar.
umtoggle — изменение статуса "checked" для объекта uimenu.
winmenu — создание подменю для позиции Window-меню.

Создание кнопок панели инструментов и управление ими

btngroup — создать кнопку панели инструментов.
btnstate — запросить статус кнопки.
btnpress — управлять кнопкой.
btndown — нажать кнопку.
btnup — отпустить кнопку.

Утилиты задания свойств объектов figure/axes

cIruprop — удалить свойство объекта.
getuprop — запросить свойство объекта.
setuprop — установить свойство объекта.

Вспомогательные утилиты

allchild — запросить все порожденные объекты.
findall — найти все объекты.
hidegui — скрыть/открыть объекты GUI.
edtext — интерактивное редактирование объектов **text**.
getstatus — запросить свойства строки объекта **figure**.
setstatus — установить свойства строки объекта **figure**.
popupstr — запросить свойства строки выпадающего меню.
remapfig — изменить положение объекта **figure**.
setptr — установить указатель на объект **figure**.
getptr — запросить наличие указателя на объекте **pointer**.
overobj — запросить дескриптор объекта с указателем.

Таким образом MATLAB содержит обширный набор команд и функций для создания типовых элементов пользовательского интерфейса. Объем данной книги не позволяет останавливаться на детальном описании этих функций, тем более что оно имеется в справочной системе. Поэтому мы ограничимся единственным примером на создание кнопки, которую можно нажимать.

4.12.3. Пример создания объекта пользовательского интерфейса — кнопки

Ниже представлена программа (распечатка М-файла), которая при запуске создает объект в виде типичной кнопки:

```
b=uicontrol('Style','pushbutton',...
  'Units','normalized','Position',[.5 .5 .2 .1],...
  'String','click here');
s='set(b,"Position",[.8*rand .9*rand .2 .1])';
```

Для создания кнопки используется команда **uicontrol** с соответствующими параметрами, задающими стиль (вид) кнопки, место ее размещения и надпись на кнопке. Кнопка размещается случайным образом вблизи от середины окна и снабжена надписью "click here" (нажми здесь). На рис. 4.68 построены две кнопки — одна в отжатом, а другая в нажатом состоянии.

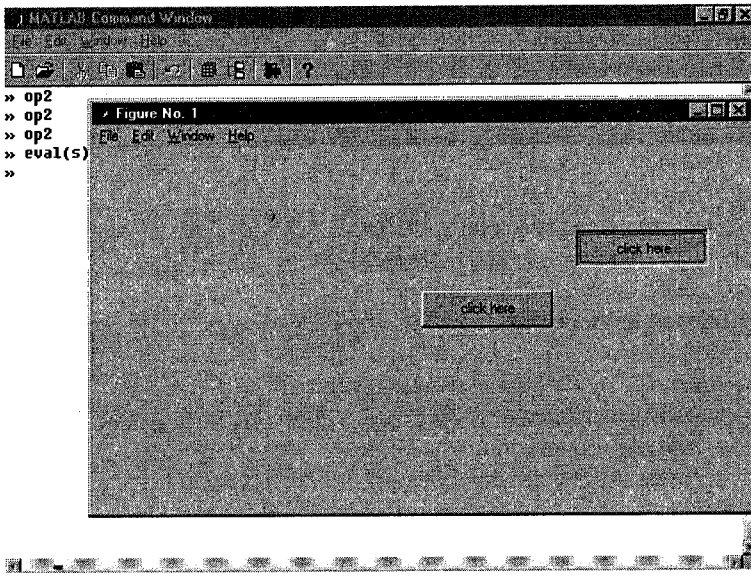


Рис. 4.68. Пример построения кнопок пользовательского интерфейса

Дескрипторная графика MATLAB позволяет создавать любые детали современного пользовательского интерфейса. Однако надо отметить, что пока она не поддержи-

вает визуально-ориентированное программирование, при котором генерация нужных кодов осуществляется автоматически при визуальном выборе нужного объекта интерфейса и размещении его в необходимом месте. Такой вид программирования поддерживает пакет Simulink, который будет описан далее.

4.13. Введение в технику обработки изображений

4.13.1. Команды image, imagesc и imfinfo

Одна из отличительных черт системы MATLAB — мощные возможности в реализации обработки изображений (images) класса BitMap (так называемая битовая графика). Весьма небольшое число команд такой графики включено в ядро системы. Часть из них была рассмотрена выше. Остановимся на некоторых наиболее важных командах.

Команды

image(A) и **imagesc(A)**

служат для представления в виде Image — рисунка содержимого матрицы A. Так, исполнив команду

```
» image(25+5*peaks)
```

можно наблюдать представление матрицы 3D-поверхности **peaks** в наглядном "цветовом" масштабе (рис. 4.69). При этом цвет каждой точки поверхности задается ее высотой.

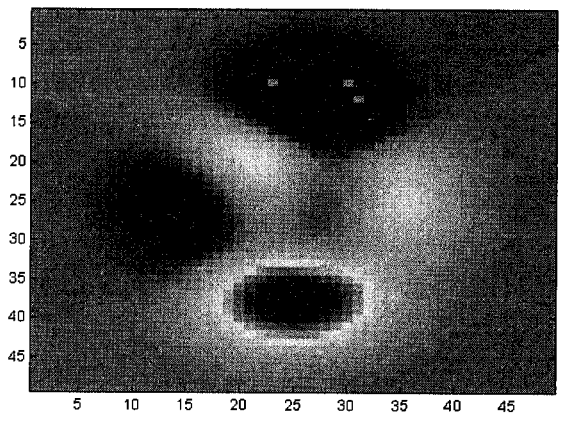


Рис. 4.69. Представление матрицы **peaks** в виде BitMap-рисунка

Для достаточно представительного отображения матрицы **peaks** в данном случае пришлось ввести нормирующие множитель 5 и слагаемое 25. Другая команда **imagesc(A)** этого уже не требует. Результат исполнения команды

```
» imagesp(peaks)
```

показан на рис. 4.70.

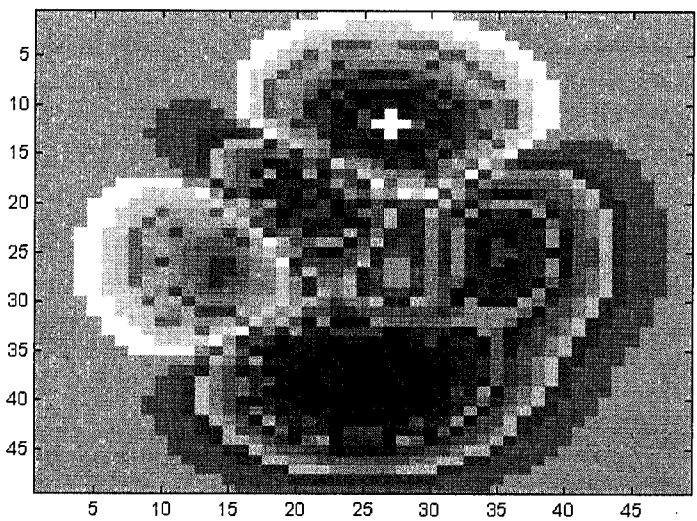


Рис. 4.70. Представление матрицы **peaks** с помощью команды **imagesp**.

На уровне ядра графических операций поддерживаются довольно очевидные функции преобразования:

RGB=hsv2rgb(HSV) — преобразует матрицу изображения HSV в матрицу изображения RGB.
HSV=rgb2hsv(RGB) — преобразует матрицу изображения RGB в матрицу изображения HSV.

Работа этих функций наглядна лишь при цветной графике. Поскольку иллюстрации в книге черно-белые, мы ограничимся лишь упоминанием о данных функциях преобразования.

Для получения детальной информации о графических файлах используется команда:

```
imfinfo('name')
```

где name — имя файла с расширением. Пример получения информации о файле saturn.tif (снимок планеты Сатурн) приводится ниже:

```
» imfinfo('saturn.tif')
ans =
                Filename: 'C:\MATLAB\toolbox\images\imdemos\saturn.tif'
             FileModDate: '25-Oct-1996 21:12:02'
                FileSize: 144184
                  Format: 'tif'
           FormatVersion: [ ]
                   Width: 438
                  Height: 328
                BitDepth: 8
               ColorType: 'grayscale'
         FormatSignature: [73 73 42 0]
               ByteOrder: 'little-endian'
          NewSubfileType: 0
           BitsPerSample: 8
             Compression: 'Uncompressed'
  PhotometricInterpretation: 'BlackIsZero'
            StripOffsets: [19x1  double]
         SamplesPerPixel: 1
            RowsPerStrip: 18
         StripByteCounts: [19x1  double]
             XResolution: 72
             YResolution: 72
          ResolutionUnit: 'Inch'
                Colormap: [ ]
      PlanarConfiguration: 'Chunky'
               TileWidth: [ ]
              TileLength: [ ]
             TileOffsets: [ ]
          TileByteCounts: [ ]
             Orientation: 1
               FillOrder: 1
        GrayResponseUnit: 0.0100
          MaxSampleValue: 255
          MinSampleValue: 0
           Thresholding: 1
        ImageDescription: [ 1x168 char ]
```

Более интересна работа MATLAB с реальными изображениями. Она положена в основу многочисленных средств создания иллюстраций в пакетах прикладных программ системы MATLAB и, прежде всего, специализированного пакета Images (полное название пакета Image Processing Toolbox).

4.13.2. Пакет прикладных программ Images

Основные средства по обработке изображений входят в пакет прикладных программ Images. С его возможностями можно детально ознакомиться выполнив команду:

```
» help images
  Image Processing Toolbox.
  Version 2.1 15-Dec-1997
  Release information.
    Readme     - Display information about versions 2.0 and 2.1.
  Image display.
    colorbar   - Display colorbar (MATLAB Toolbox).
    getimage   - Get image data from axes.
    image      - Create and display image object (MATLAB Toolbox).
    imagesc    - Scale data and display as image (MATLAB Toolbox).
.....
  Demos.
    dctdemo    - 2-D DCT image compression demo.
    edgedemo   - Edge detection demo.
    firdemo    - 2-D FIR filtering and filter design demo.
    imadjdemo  - Intensity adjustment and histogram equalization demo.
    nrfiltdemo - Noise reduction filtering demo.
    qtdemo     - Quadtree decomposition demo.
    roidemo    - Region-of-interest processing demo.
.....
```

В обширном списке (выше он дан выборочно) содержится около 120 команд для работы с BitMap-изображениями. Эти изображения могут быть получены со сканера, цифрового фотоаппарата либо от видеокамеры, подключенной к компьютеру через видеобластер или подобное ему другое устройство.

Пакет Images поддерживает следующие возможности:

- отображение рисунков различных графических форматов (в том числе с высоким разрешением) на экране дисплея;
- запись рисунков в файл, считывание их из файла и получение информации о файле;
- выполнение геометрических операций с графическими объектами, например, вращение или интерполяция данных;
- операции на уровне элементарных элементов изображений — пикселей;
- аналитические операции с рисунками;
- осуществление компрессии и декомпрессии рисунков;
- выполнение различных видов фильтрации изображений и конструирование фильтров;
- выполнение различных преобразований рисунков;
- бинарные операции с рисунками;

- операции задания и преобразования цветов;
- преобразование типов и форматов рисунков;
- демонстрация возможностей пакета;
- организация показа слайдов.

Объем данной книги не позволяет описать указанный пакет подробно, но в этом нет и особого смысла. Если вы уяснили методы работы с системой MATLAB, то не стоит большого труда ознакомиться с составом всех команд и функций пакета и опробовать их на практике. В связи с этим мы остановимся лишь на нескольких демонстрационных примерах.

4.13.3. Знакомство с демонстрационными примерами пакета Images

Есть ряд способов ознакомиться с весьма обширными и впечатляющими возможностями пакета Images: с помощью демонстрационных примеров Demos в справочной базе данных системы, путем непосредственного запуска этих примеров (список приводился выше) или апробацией отдельных команд и так далее.

Пример реконструкции изображений с контролем оригинала, создаваемого образа, и ошибки реконструкции представлен в файле dctdemo. Он выводит свое окно со средствами пользовательского интерфейса. Можно задать выбор той или иной исходной картинки для ее преобразования, задать степень компрессии изображения и визуально наблюдать за характером преобразований (рис. 4.71). Для просмотра следует нажать кнопку Apply, кнопка Close закрывает окно, а кнопка Info выводит информацию о примере.

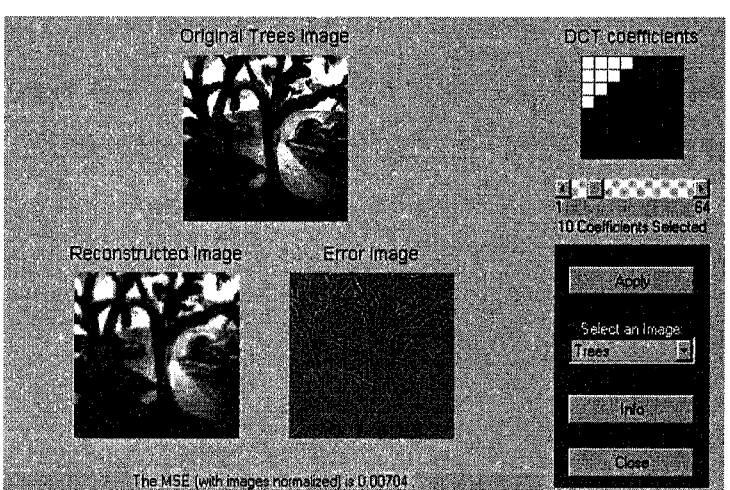


Рис. 4.71. Окно компрессии и реконструкции изображения

Другой важной сферой применения пакета Images является фильтрация изображений, например, с целью очистки изображения от шумовых помех. О том, насколько эффективна такая фильтрация, наглядно представляет рис. 4.72, полученный при запуске демонстрационного примера nrfiltdemo. В качестве исходного изображения взят фотоснимок планеты Сатурн, затем с помощью генератора случайных чисел на него нанесены помехи в виде точек. Имеется возможность оценить степень очистки от помех с применением различных алгоритмов фильтрации, представленных в пакете Images рядом функций.

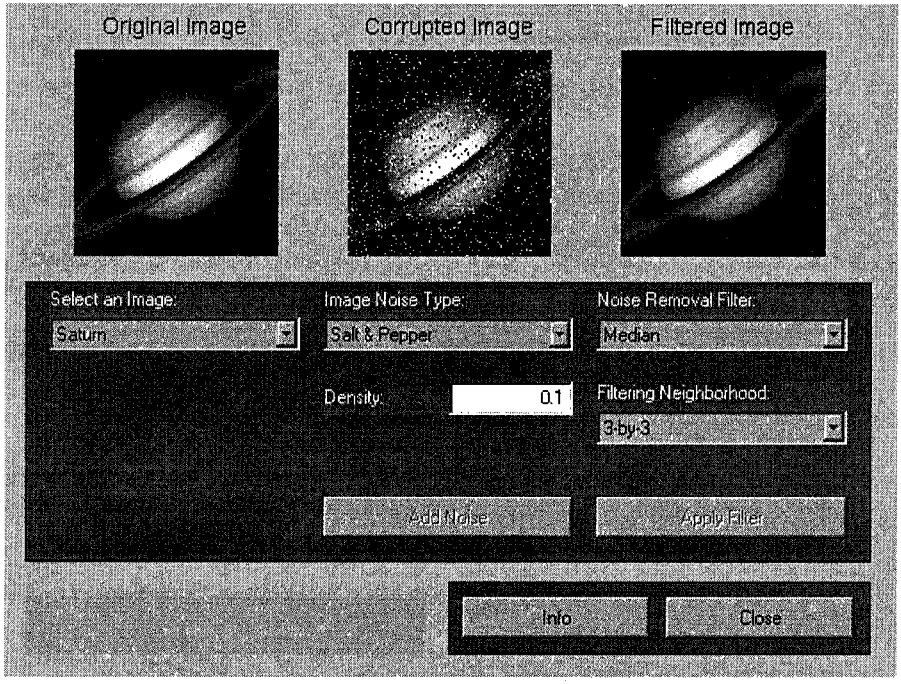


Рис. 4.72. Окно демонстрации возможностей фильтрации изображений

Следующий пример иллюстрирует возможность изменения яркости изображения (рис. 4.73). Кривая яркости может устанавливаться перемещением ее точек с помощью мыши. Можно задавать линейный или нелинейный вид этой кривой и тут же наблюдать за изменением характеристик изображения. Вид кривой существенно влияет на яркость и контрастность изображений и позволяет корректировать изображения, например, слишком темные преобразовать в светлые или наоборот.

Наконец, последний пример (остальные вы можете просмотреть самостоятельно) демонстрирует действенность алгоритма повышения четкости изображения в произвольной его области (рис. 4.74). В нашем случае эта область ограничена треугольником. В окне можно наблюдать (и выбирать) исходное изображение, область действия алгоритма и результирующее изображение.

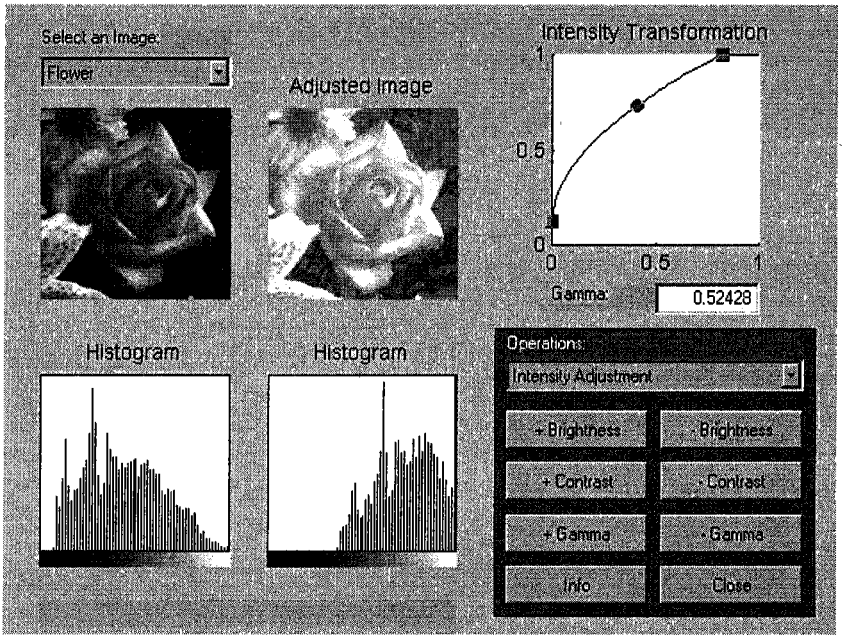


Рис. 4.73. Окно демонстрации изменения яркости изображения

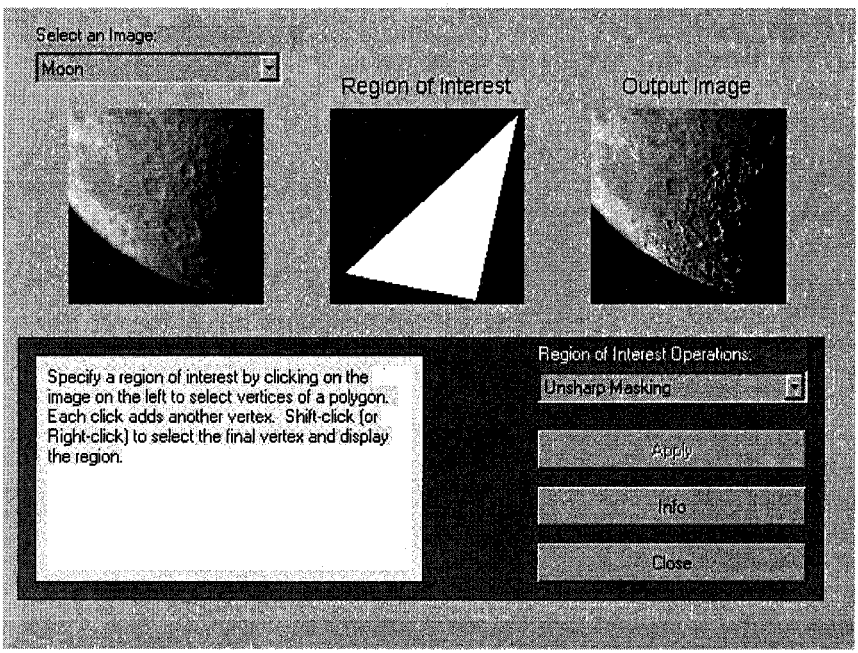


Рис. 4.74. Окно демонстрации повышения четкости изображения в заданной зоне

Даже эти примеры помогают оценить обширные возможности пакета Images в технике обработки реальных изображений.

4.12.4. Примеры программирования задач со средствами пакета Images

Вы можете ознакомиться со всеми демонстрационными примерами пакета Images, выполнив команду **type fname**, где **fname** — имя файла с демонстрационным примером. Однако следует отметить, что представленные демонстрационные программы весьма сложны, поскольку создают окна в виде стандартных панелей с современными элементами пользовательского интерфейса и переключателями выбора вариантов. Мы рекомендуем читателю воздержаться от знакомства с этими программами до ознакомления с главой 6, в которой дается систематическое описание средств программирования системы MATLAB. А пока мы ограничимся парой характерных примеров, наглядно показывающих, что при работе с системой MATLAB вполне можно руководствоваться народной поговоркой о том, что не боги горшки обжигают.

Рассмотрим, к примеру, задачу фильтрации искаженного помехами произвольного изображения, представленного некоторым файлом. Реализующая эту сложную и весьма эффективную операцию программа выглядит следующим образом:

```
% Пример фильтрации изображения из файла
I = imread('saturn.tif');
h = [1 2 1; 0 0 0; -1 -2 -1];
I2 = filter2(h,I);
imshow(I2,[ ]), colorbar
```

В результате исполнения этой простой и вполне очевидной программы можно получить отфильтрованное изображение из файла saturn.tif (рис. 4.75). Хотите попробовать обработать какой-либо снимок? Все что для этого нужно — подготовить снимок в нужном формате (например, tif) и заменить во второй строке имя демонстрационного файла на имя вашего файла.

Рассмотрим еще один достаточно простой пример – построение сферы в виде глобуса и наклейка на полушарие изображения карты погоды:

```
load earth % load image data, X, and colormap, map
sphere; h = findobj('Type','surface');
hemisphere = [ones(257,125),X,ones(257,125)];
set(h,'CData',flipud(hemisphere),'FaceColor','texturemap')
colormap(map)
axis equal
view([90 0])
set(gca,'CameraViewAngleMode','manual')
view([65 30])
```

Полученное при этом изображение показано на рис. 4.76.

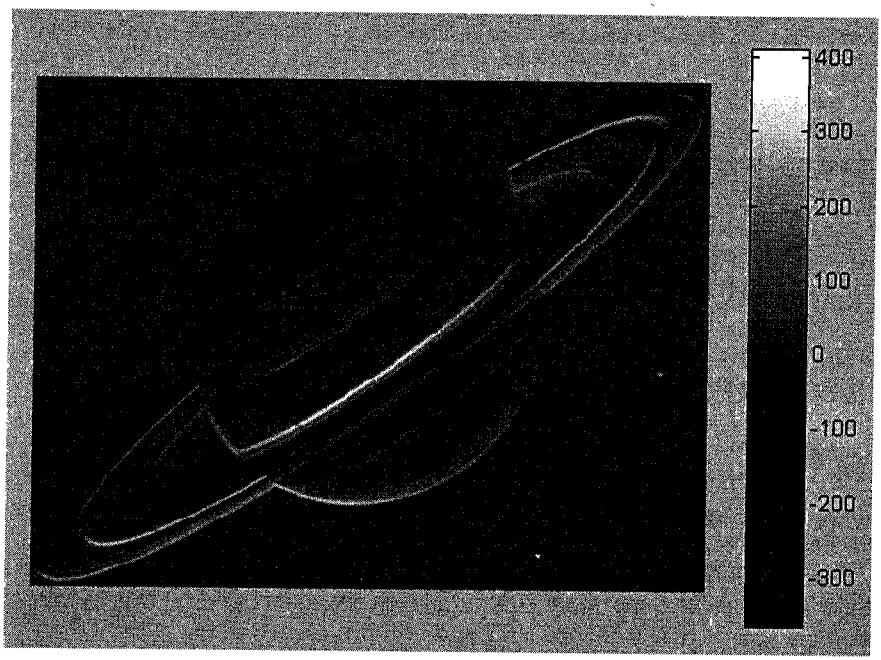


Рис. 4.75. Отфильтрованное изображение планеты Сатурн и панель градаций серого цвета

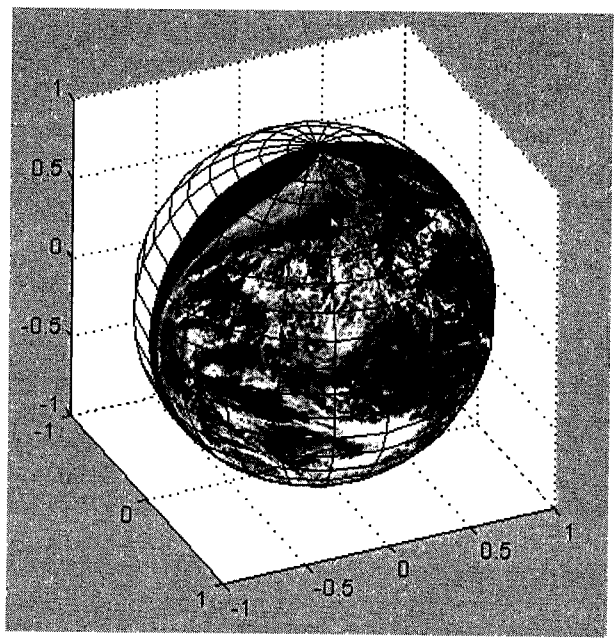


Рис. 4.76. Пример построения сложного изображения — реконструкция глобуса

Пакет Images можно рассматривать как полезный инструмент для создания новых алгоритмов и методов обработки изображений и обучения специалистов. Применение его непосредственно для обработки изображений вполне возможно, но все же едва ли целесообразно. Дело в том, что обширнейшие возможности по обработке изображений открывают профессиональные графические пакеты, например Adobe Photoshop, Ulead PhotoImpact, Corell Draw и другие, в которых реализованы самые современные методы обработки изображений и использованы последние новации пользовательского интерфейса. В этом случае достоинство средств MATLAB проявляется только в математической прозрачности реализаций алгоритмов обработки изображений.

4.14. Галерея трехмерной графики

4.14.1. Состав галереи — Galery

Для знакомства с возможностями трехмерной графики и построением пользовательского интерфейса MATLAB имеет галерею Galery в виде профессионально выполненных графических программ. Доступ к ним возможен как из режима демонстрации (позиция Examples and Demos в позиции Help главного меню окна командной работы MATLAB), так и путем запуска команды с командной строки и указанием имени соответствующего файла.

Галерея представлена следующими фигурами и файлами:

Help Galery	Файл	Наименование фигуры
knot	knot.m	Завязанный узел
quiver	quivdemo.m	Векторное объемное поле
klein II	klein1.m	Объемное кольцо
cruller	cruller.m	Объемное кольцо Мебиуса
hoops	tory4.m	Четыре объемных обруча
slosh	spharm2.m	Построение фигуры, напоминающей улитку
modes	modes.m	Демонстрация фаз анимации 3D-поверхности
logo	logo.m	Построение логотипа системы MATLAB

Обратите внимание на то, что иногда имя файла не совпадает с именем фигуры в режиме демонстрации. Некоторые из фигур галереи мы уже описывали — knot (рис. 1.15) и logo (рис. 4.60). Ниже приведено еще несколько примеров, которые дают наглядное представление о возможностях дескрипторной графики системы MATLAB.

4.14.2. Примеры графиков из галереи

Команда **quivdemo** выводит окно с демонстрацией построения пространственного векторного поля. Это окно с вырезанными элементами управления показано на рис. 4.77.

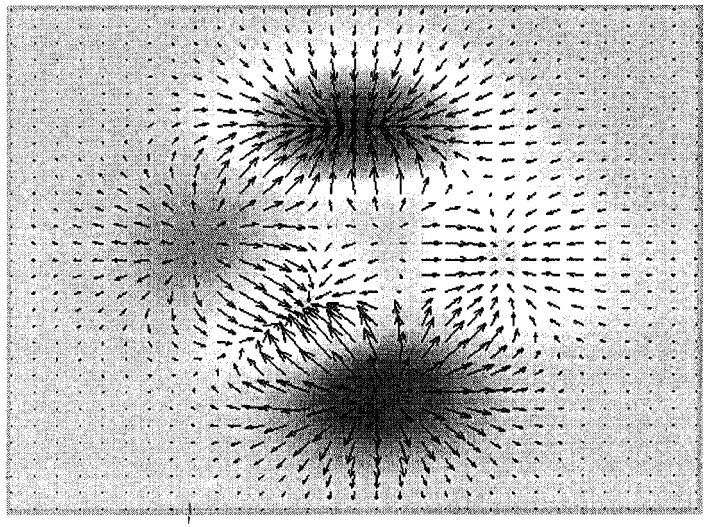


Рис. 4.77. График пространственного векторного поля

Полезно обратить внимание на то, что в этом примере сам по себе график — двумерный. Объемный вид поверхности достигается сочетанием функциональной окраски с изображением графика векторного поля с помощью стрелок.

Команда **klein1** строит график объемной ленты Мебиуса с одним перекручиванием. Вид этой фигуры показан на рис. 4.78. Этот график хорошо иллюстрирует хотя и одноцветную, но функциональную закраску фигуры с имитацией ее освещения от источника света, расположенного сверху и справа, и реализацией эффектов отражения света.

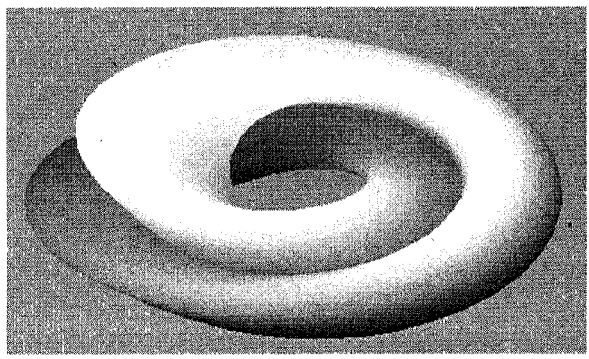


Рис. 4.78. Объемная линия Мебиуса с одним перекручиванием

Команда **cruller** строит объемное кольцо Мебиуса с двойным перекручиванием. Построенная фигура показана на рис. 4.79. В данном случае используется обычная функциональная окраска с сохранением линий каркаса фигуры.

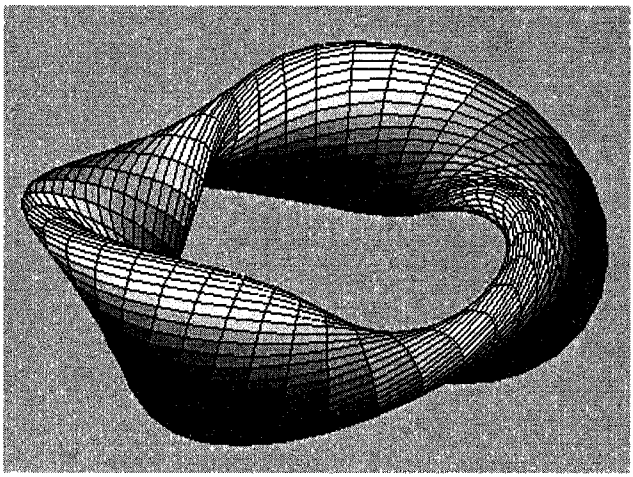


Рис. 4.79. Объемное кольцо Мебиуса

Команда **tory4** строит четыре переплетающихся друг с другом обруча (объемных кольца) в пространстве (рис. 4.80). Наглядности этой картины также способствует функциональная окраска обручей и видимые линии каркаса. Обратите внимание, что невидимые линии удалены.

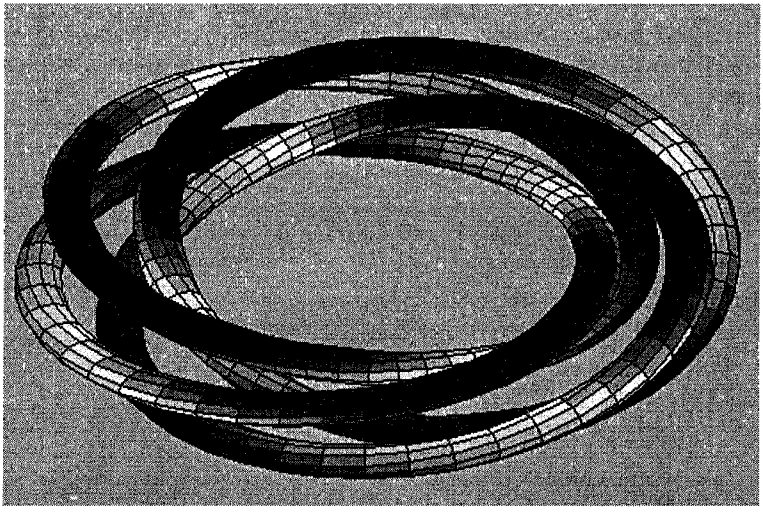


Рис. 4.80. Четыре обруча в пространстве

Любопытную фигуру, напоминающую раковину улитки, строит команда **spharm2**. Вид фигуры показан на рис. 4.81. Здесь интересно применение многоцветной функциональной окраски с использованием интерполяции по цвету, а также имитация эффектов отражения при освещении фигуры источником точечного цвета. Отчетливо видны зеркальные блики на поверхности фигуры.

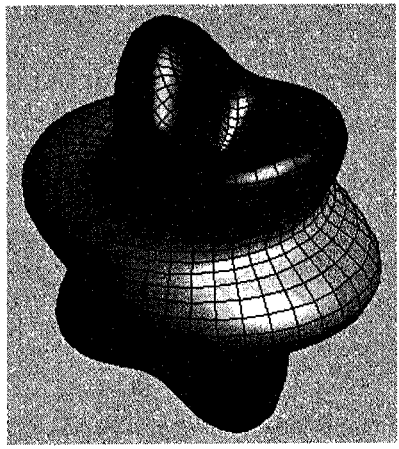


Рис. 4.81. Фигура, напоминающая улитку

Еще одна команда **modes** иллюстрирует построение фаз анимации трехмерной поверхности (рис. 4.82). Она представлена 12 фигурами, отражающими положение поверхности в пространстве в различные моменты времени.

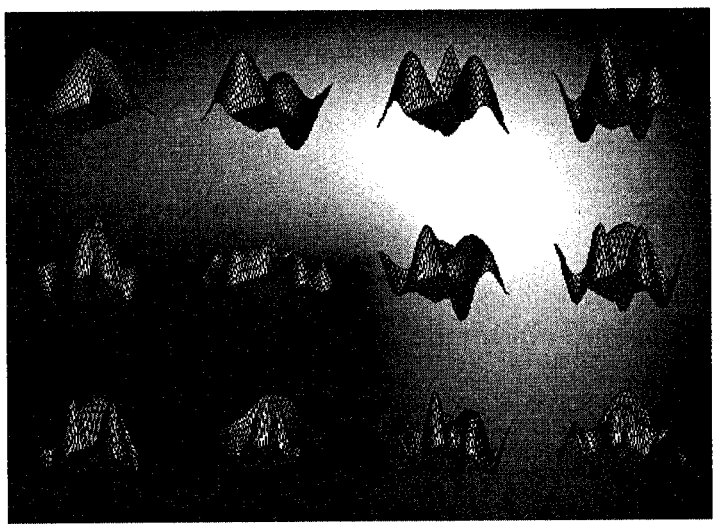


Рис. 4.82. Фазы анимации трехмерной поверхности

В целом указанный набор программ дает хорошее представление о возможностях трехмерной графики системы MATLAB. Команда **type name**, где name — имя соответствующей команды, выводит полный листинг программы, реализующей построение той или иной фигуры.

Глава 5. Операторы и функции

5.1. Операторы и специальные символы

Здесь мы рассмотрим полный набор операторов входного языка системы MATLAB и соответствующих им функций. Напомним, что полный список операторов выводится командой

```
» help ops
```

Операторы и специальные символы системы MATLAB можно разделить на ряд категорий, которые рассматриваются ниже.

5.1.1. Арифметические операторы и функции

Арифметические операторы являются самыми распространенными и известными. В отличие от большинства языков программирования в системе MATLAB практически все операторы являются матричными, т.е. предназначены для выполнения операций над матрицами. Ниже приводится список арифметических операторов и синтаксис их применения.

Функция	Название	Оператор	Синтаксис
plus	Плюс	**+**	**M1+M2**
uplus	Унарный плюс	**+**	**+M**
minus	Минус	**-**	**M1-M2**
uminus	Унарный минус	**-**	**-M**
mtimes	Матричное умножение	*****	**M1*M2**
times	Почленное умножение массивов	**.***	**A1*A2**
mpower	Возведение в степень матрицы	**^**	**M1^x**
power	Возведение в степень массива почленное	**.^**	**A1^x**
mldivide	Деление матриц обратное (справа—налево)	****	**M1\M2**
mrdivide	Деление матриц слева—направо	**/**	**M1/M2**
ldivide	Почленное деление массивов справа—налево	**.**	**A1.\A2**
rdivide	Почленное деление массивов слева—направо	**./**	**A1./A2**
kron	Тензорное умножение Кронекера	**kron**	**kron(X,Y)**

Обратите внимание на то, что каждый оператор имеет тождественную по назначению функцию. Например, оператору матричного умножения * соответствует функция

mtimes(M1,M2). Примеры применения арифметических операторов уже не раз приводились, так что ограничимся несколькими дополнительными примерами:

```
» A=[1 2 3];
» B=[4 5 6];
» B-A
ans =
   3   3   3
» minus(B,A)
ans =
   3   3   3
» A.^2
ans =
   1   4   9
» power(A,2)
ans =
   1   4   9
» A.\B
ans =
   4.0000   2.5000   2.0000
» ldivide(A,B)
ans =
   4.0000   2.5000   2.0000
» rdivide(A,B)
ans =
   0.2500   0.4000   0.5000
```

Соответствие функций операторам и командам в системе MATLAB является одним из основных положений программирования. Оно позволяет использовать одновременно как элементы операторного, так и элементы функционального программирования.

Следует отметить, что в математических выражениях операторы имеют общепринятый **приоритет** исполнения. Например, приоритет логических операторов выше арифметических, приоритет возведения в степень выше приоритета умножения и деления, приоритет умножения и деления выше приоритета сложения и вычитания. Для изменения приоритета операций в математических выражениях используются круглые скобки. Степень вложения скобок не ограничивается.

5.1.2. Операторы отношения и их функции

Операторы отношения служат для сравнения двух величин, векторов или матриц. Все операторы отношения имеют два операнда, например x и y, и записываются в виде:

Функция	Название	Оператор	Пример
eq	Равно	**==**	**x==y**
ne	Не равно	**~=**	**x~=y**
lt	Менее чем	**<**	**x<y**
gt	Более чем	**>**	**x>y**
le	Менее чем или равно	**<=**	**x<=y**
ge	Более чем или равно	**>=**	**x>=y**

Данные операторы выполняют поэлементное сравнение векторов или матриц одинакового размера и возвращают значение логической единицы 1 (**True**), если элементы идентичны, и значение логического нуля 0 (**False**) в противоположном случае. Если операнды — действительные числа, то применение операторов отношения тривиально:

```
» eq(2,2)
ans =
    1
» 2==2
ans =
    1
» 1 ne 2
» 2 ~= 2
ans =
    0
» 5 > 3
ans =
    1
» le(5,3)
ans =
    0
```

Следует отметить, что операторы **<**, **<=**, **>** и **>=** при комплексных операндах используют для сравнения только действительные части операндов — мнимые отбрасываются. В то же время операторы **==** и **~=** ведут сравнение с учетом как действительной, так и мнимой части операндов. Следующие примеры поясняют это положение:

```
» (2+3i)>=(2+i)
ans =
    1
» (2+3i)>(2+i)
ans =
    0
» abs(2+3i)>abs(2+i)
ans =
    1
» (2+3i)==(2+i)
```

```
ans =
    0
» (2+3i)~=(2+i)
ans =
    1
```

Если один из операндов — скаляр, происходит сравнение всех элементов массива со значением полученного из скаляра массива

```
M =
   -1    0
    1    2
» M>=0
ans =
    0    1
    1    1
```

В общем случае операторы отношения сравнивают два массива одного размера и выдают результат в виде массива того же размера:

```
» M>[0 1; 1 0]

ans =

    0    0
    0    1
```

Таким образом, спектр применения операторов отношения в системе MATLAB шире, чем в обычных языках программирования, поскольку он относится и к числам, и к векторам, и к матрицам. Возможно применение этих операторов и к символьным выражениям:

```
» 'b'>'a'
ans =
    1
» 'abc'=='abc'
ans =
    1    1    1
» 'cba'<'abc'
ans =
    0    0    1
```

В этом случае символы, входящие в выражения, представляются своими ASCII-кодами. Строки воспринимаются как векторы с кодами. Все это надо учитывать при использовании управляющих структур языка программирования, которые широко используют операторы отношения.

5.1.3. Логические операторы

Логические операторы и соответствующие им функции служат для реализации поэлементных логических операций над элементами одинаковых по размеру массивов:

Функция	Название	Обозначение
and	Логическое И (AND)	**&**
or	Логическое ИЛИ (OR)	**|**
not	Логическое НЕТ (NOT)	**~**
xor	Исключающее ИЛИ (EXCLUSIVE)	
any	Верно, если все элементы вектора равны нулю	
all	Верно, если все элементы вектора не равны нулю	

Работа операторов поясняется приведенными ниже примерами:

```
» A=[1 2 3];
» B=[1 0 0];
» and(A,B)
ans =
   1   0   0
» or(A,B)
ans =
   1   1   1
» A&B
ans =
   1   0   0
» A|B
ans =
   1   1   1
» not(A)
ans =
   0   0   0
» not(B)
ans =
   0   1   1
» ~B
ans =
   0   1   1
» xor(A,B)
ans =
   0   1   1
» any(A)
ans =
   1
» all([0 0 0])
```

```
ans =
    0
» all(B)
ans =
    0
» and('abc','012')
ans =
    1    1    1
```

Обратите внимание, что аргументами логических операторов могут быть числа и строки. При аргументах-числах логический 0 соответствует нулю, а любое отличное от нуля число воспринимается как логическая единица. Для строк действует уже отмеченное правило — каждый символ строки оценивается своим ASCII кодом.

5.1.4. Специальные символы

К классу операторов в системе MATLAB относятся также **специальные символы**. Они предназначены для создания самых разнообразных объектов входного языка и языка программирования системы и придания им различных форм. Ниже представлено описание полного набора специальных символов.

Тип	Название	Обозначение
colon	двоеточие	:
paren	круглые скобки	( )
paren	квадратные скобки	[ ]
paren	фигурные скобки	{ }
punct	десятичная точка	.
punct	выделение поля структуры	.
punct	родительский каталог	..
punct	продолжение строки	...
punct	разделитель	,
punct	точка с запятой	;
punct	комментарий	%
punct	вызов команды операционной системы	!
punct	присваивание	=
punct	кавычка	'
transpose	транспонирование	.'
ctranspose	транспонирование комплексно сопряженное	'
horzcat	горизонтальная конкатенация	[,]
vertcat	вертикальная конкатенация	[;]
subsasgn	присваивание подмассива	( ),{ },.
subsref	ссылка на подмассив	( ),{ },.
subsindex	индекс подмассива	

Теперь рассмотрим их более подробно.

: (двоеточие) — формирование подвекторов и подматриц из векторов и матриц. Оператор : — один из наиболее часто используемых операторов в системе MATLAB.

Оператор : использует следующие правила для создания векторов:

j:k — то же, что и **[j,j+1,...,k]**.

j:k — пустой вектор, если **j >k**.

j:i:k — то же, что и **[j,j+i,j+2i, ...,k]**.

j:i:k — пустой вектор, если **i >0** и **j >k** или если **i <0** и **j <k** , где **i, j**, и **k** — скалярные величины.

Ниже показано, как выбрать с помощью оператора : строки, столбцы и элементы из векторов, матриц и многомерных массивов:

A(:,j) — это **j-ый** столбец из **A**.

A(i,:) — это **i-ая** строка из **A**.

A(:,:) — эквивалент двумерного массива. Для матриц это аналогично **A**.

A(j:k) — это **A(j), A(j+1),...,A(k)**.

A(:,j:k) — это **A(:,j), A(:,j+1),...,A(:,k)**.

A(:,:,k) — это **k-ая** страница трехмерного массива **A**.

A(i,j,k,:) — вектор четырехмерного массива **A**. Вектор включает **A(i,j,k,1), A(i,j,k,2), A(i,j,k,3)** и т. д.

A(:) записывает все элементы массива **A** в виде столбца.

( ) (круглые скобки) — используются для задания порядка выполнения операций в арифметических выражениях, указания последовательности аргументов функции и указания индексов элемента вектора или матрицы. Если **X** и **V** — векторы, то **X(V)** можно представить как **[X(V(1)), X(V(2)), ..., X(V(n))]**. Элементы вектора **V** должны быть целыми числами, чтобы использоваться как индексы элементов. Ошибка генерируется в том случае, если индекс элемента меньше единицы или больше чем **size(X)**. Такой же принцип индексирования действителен и для матриц. Если вектор **V** имеет m компонентов, а вектор **W** — n компонентов, то **A(V,W)** будет матрицей размера m×n, сформированной из элементов матрицы **A**, индексы которой – элементы векторов **V** и **W**.

[ ] (квадратные скобки) — используются для формирования векторов и матриц. **[6.9 9.64 sqrt(–1)]** — вектор, содержащий три элемента, разделенных пробелами. **[6.9, 9.64, i]** — такой же вектор, а **[1+j 2–j 3] и [1 +j 2 –j 3]** — разные векторы: первый содержит три элемента, а второй пять. **[11 12 13; 21 22 23]** — это матрица размера 2×3. Точка с запятой разделяет первую и вторую строки.

A = [ ] — сохраняет пустую матрицу в **A**.

A(m,:) = [ ] — удаляет строку **m** из матрицы **A**.

A(:,n) = [ ] — удаляет столбец n из матрицы **A**.

{ } (фигурные скобки) — используются для формирования массивов ячеек. Например, **{magic(3) 6.9 'hello'}** — массив ячеек с тремя элементами.

. (десятичная точка) — используется для отделения дробной части чисел от целой. Например, 314/100, 3.14 and .314e1 — одно и то же число.

. (выделение поля структуры) — выделение поля структуры. Например, A.(field) и A(i).field, где A — структура, выделение поля структуры с именем «field».

.. (родительский каталог) — переход по дереву каталогов на один уровень вверх.

... (продолжение) — три или более точек в конце строки указывают на продолжение строки.

, (запятая) — используется для разделения индексов элементов матрицы и аргументов функции и для отделения операторов языка MATLAB. При отделении операторов в строке запятая должна заменяться на точку с запятой с целью запрета вывода на экран результата вычислений.

; (точка с запятой) — используется внутри круглых скобок для разделения строк матриц, а также для запрета вывода на экран результата вычислений.

% (знак процента) — используется для указания логического конца строки. Текст, находящийся после знака процента, воспринимается как комментарий и игнорируется.

! (восклицательный знак) — является указателем ввода команды операционной системы. Строка, следующая за ним, воспринимается как команда операционной системы.

= (знак присваивания) — используется для присваивания значений в арифметических выражениях.

' (одиночная кавычка) — текст в кавычках представляется как вектор символов с компонентами, являющимися ASCII-кодами символов. Кавычка внутри текста задается двумя кавычками. Например:

```
» a='привет '' друзьям'
a =
привет ' друзьям
```

' (транспонирование матрицы) — транспонирование матриц, например **A'** – транспонированная матрица **A**. Для комплексных матриц транспонирование дополняется комплексным сопряжением. Строки транспонированной матрицы соответствуют столбцам исходной матрицы.

.' (транспонирование массива) — транспонирование массива, например **A.'** – транспонированный массив **A**. Для комплексных массивов операция сопряжения не выполняется.

[,] (горизонтальная конкатенация). Так, **[A,B]** — горизонтальная конкатенация (объединение) матриц **A** и **B**. **A** и **B** должны иметь одинаковое количество строк. **[A B]** действует аналогично. Горизонтальная конкатенация может быть применена для любого числа матриц в пределах одних скобок **[A,B,C]**. Горизонтальная и вертикальная конкатенации могут быть объединены вместе **[A,B;C]**.

[;] (вертикальная конкатенация). Так, **[A;B]** — вертикальная конкатенация (объединение) матриц **A** и **B**. **A** и **B** должны иметь одинаковое число столбцов. Вертикальная конкатенация может быть применена для любого числа матриц в пределах одних скобок **[A;B;C]**. Горизонтальная и вертикальная конкатенации могут быть объединены вместе **[A;B,C]**.

( ),{ }, (присваивание подмассива). Так, **A(I)=B** присваивает значения элементов массива **B** элементам массива **A**, которые определяются вектором индексов **I**. Массив **B** должен иметь одинаковое число элементов с массивом **I**, или может быть скаляром.

A(I,J)=B — присваивает значения массива **B** элементам прямоугольной подматрицы **A**, которые определяются векторами индексов **I** и **J**. Массив **B** должен иметь **LENGTH(I)** строк и **LENGTH(J)** столбцов.

A{I}=B, где **A** — массив ячеек и **I** — скаляр, помещает копию массива **B** в определенную ячейку массива **A**. Если **I** имеет более одного элемента, то появляется сообщение об ошибке.

5.1.5. Системные переменные и константы

Как отмечалось ранее, в состав объектов MATLAB входит ряд системных переменных и констант, значения которых устанавливаются системой при ее загрузке или автоматически формируются в процессе вычислений. Описание таких объектов приводится ниже:.

ans — результат выполнения последней операции. Переменная **ans** создается автоматически, когда не определены выходные аргументы какого-либо оператора. Пример:

```
» cos([0:2*pi])
ans =
   1.0000   0.5403  -0.4161  -0.9900  -0.6536   0.2837   0.9602
```

computer — возвращает строку с информацией о типе компьютера, на котором установлена система MATLAB.

[str,maxsize] = computer — возвращает строку **str** с информацией о компьютере и целое число **maxsize**, содержащее максимально допустимое число элементов матрицы для данной версии MATLAB. Пример:

```
» [str,maxsize] = computer
str =
```

```
PCWIN
maxsize =
  268435455
```

eps — возвращает интервал между числом 1.0 и следующим наибольшим числом с плавающей запятой, которое воспринимается как отличное от 1.0. Значение **eps** определяет заданный по умолчанию порог для функций **pinv** и **rank**, а также для некоторых других функций. На машинах с IEEE-плавающей арифметикой **eps = 2^(–52)**, что приблизительно составляет **2.22e–16**. Пример:

```
» eps
ans =
  2.2204e-016
```

f = flops — возвращает общее число операций с плавающей запятой. **flops(0)** сбрасывает счетчик операций в нуль.

Пример:

```
» f = flops
f =
    8
```

i — мнимая единица (или **sqrt(–1)**), которая используется для задания мнимой части комплексных чисел. Так как i — функция, она может быть отменена и выступать в качестве переменной. Это позволяет использовать ее как индекс, например, в цикле **for**. При необходимости символ i можно использовать без знака умножения при задании комплексной константы. В качестве мнимой единицы можно также использовать символ j.

Пример:

```
» w=3+5i
w =
  3.0000 + 5.0000i
```

Inf — возвращает представление арифметики IEEE для положительной бесконечности. Бесконечность следует из операций, подобных делению на нуль и переполнения, которые ведут к результатам, слишком большим, чтобы их можно было представить в виде значений чисел с плавающей запятой. Пример:

```
» 4/0
Warning: Divide by zero.
ans =
  Inf
```

Inputname (argnum) — возвращает название переменной рабочей области окна, соответствующее номеру аргумента **argnum**. Эта команда может использоваться только внутри тела функции. Если входной аргумент не имеет никакого названия (например, если это — выражение вместо переменной), команда **inputname** возвращает пустую строку **('')**.

j — мнимая единица. Символ j можно использовать в качестве альтернативной мнимой единицы.

Как мнимая единица (или **sqrt(–1)**) j используется для задания мнимой части комплексных чисел. Все сказанное о символе i относится и к j. Пример:

```
» s=4-3j
s =
  4.0000 - 3.0000i
```

NaN — возвращает представление арифметики IEEE для нечисловых величин, например, в случае операций, которые имеют неопределенные численные результаты. Пример:

```
» s=0/0
Warning: Divide by zero.
s =
  NaN
```

Функция **nargchk** часто используется внутри М-файла для проверки соответствия количества аргументов.

msg = nargchk(low,high,number) — возвращает сообщение об ошибке, если number меньше чем low или больше чем high, в противном случае возвращается пустая строка. Пример:

```
» msg = nargchk(4,9,5)
msg =
   [ ]
» msg = nargchk(4,9,2)
msg =
Not enough input arguments
```

nargin — возвращает число входных аргументов, определенных для функции. Внутри тела М-файла функции **nargin** и **nargout** указывают, соответственно, количество входных или выходных аргументов, заданных пользователем. Вне тела М-файла функции **nargin** и **nargout** показывают, соответственно, число входных или выходных аргументов для данной функции. Отрицательное число аргументов означает, что функция имеет переменное число аргументов.

nargin('fun') — возвращает число объявленных входов для функции **fun** М-файла. Если функция содержит переменное число входных аргументов, возвращается **–1**.

nargout — возвращает число выходных аргументов, определенных для функции.

nargout('fun') — возвращает число объявленных выходов для функции **fun** М-файла.

Применение этих функций мы рассмотрим позже, при описании структуры функций.

pi — число π (отношение длины окружности к ее диаметру). **pi** возвращает число с плавающей запятой, ближайшее к значению π . Выражения **4*atan(1)** и **imag(log(–1))** выдают этот же результат. Пример:

```
» pi
ans =
   3.1416
```

realmax — возвращает самое большое число в формате с плавающей запятой, соответствующее конкретному компьютеру. Большее значение соответствует системной переменной **inf**. Пример:

```
» n = realmax
n =
 1.7977e+308
```

realmin — возвращает наименьшее нормализованное положительное число в формате с плавающей запятой, представимое на конкретном компьютере. Любое меньшее число воспринимается как ноль. Пример:

```
» n = realmin
n =
 2.2251e-308
```

varargout = foo(n) — возвращает список выходных аргументов переменной длины функции **foo**.

y = function bar(varargin) — принимает переменное число аргументов в функцию **bar**.

varargin и varargout — используются только внутри М-файла функции для задания произвольных аргументов функции. Эти переменные должны быть последними в списке входов или выходов, а для обозначения могут использоваться только строчные буквы.

Применение этих функций мы рассмотрим несколько позже.

5.1.6. Функции поразрядной обработки

Ряд функций определен для поразрядной логической обработки данных.

bitand(A,B) — возвращает поразрядное И двух неотрицательных целых аргументов **A** и **B**. Пример:

```
» f=bitand(7,14)
f =
   6
```

bitcmp(A,n) — возвращает поразрядное дополнение аргумента **A** как n-битовое целое число в формате чисел с плавающей запятой (floating-point integer или сокращенно flint). Пример:

```
» g=bitcmp(6,4)
g =
   9
```

bitor(A,B) — возвращает поразрядное ИЛИ двух неотрицательных целых аргументов **A** и **B**. Пример:

```
» v=bitor(12,21)
v =
  29
```

bitmax — возвращает максимальное целое число без знака с плавающей запятой применительно к используемому компьютеру. Это значение определяется для комбинации, когда все биты установлены. На машинах с IEEE-арифметикой это значение равно 2^{53}-1. Пример:

```
» bitmax
ans =
 9.0072e+015
```

bitset(A,bit) — устанавливает бит в позиции bit аргумента **A** в 1. Аргумент **A** должен быть неотрицательным целым. bit — это номер между 1 и числом бит в целом числе с форматом чисел с плавающей запятой, представленном **A**.

bitset(A,bit,v) — устанавливает бит в позиции bit равным значению **v**, которое должно быть или 0, или 1. Пример:

```
» d=bitset(12,2,1)
d =
  14
```

bitshift(A,n) — возвращает значение аргумента **A**, сдвинутое на n бит. Если n>0, это аналогично умножению на 2^n (левый сдвиг). Если n<0, это аналогично делению на 2^n (правый сдвиг). Пример:

```
» f=bitshift(4,3)
f =
  32
```

bitget(A,bit) — возвращает значение бита в позиции bit операнда **A**. Аргумент **A** должен быть неотрицательным целым числом. bit — это номер между 1 и числом бит в целом числе формата с плавающей запятой, представленном **A**. Пример:

```
» disp(dec2bin(23))
10111
» C = bitget(23,5:-1:1)
C =
   1   0   1   1   1
```

bitxor(A,B) — возвращает результат поразрядного исключающего ИЛИ для двух аргументов **A** и **B**. Оба аргумента должны быть целыми. Пример:

```
» x=bitxor(12,31)
```

```
x =
   19
```

Чтобы операнды этих функций гарантированно были целыми числами, при их задании рекомендуется использовать функции **ceil, fix, floor** и **round**.

5.1.7. Функции обработки множеств

Множество представляет собой первичное понятие математики, не имеющее четкого определения. Под множеством подразумевается совокупность некоторых объектов, например книг в библиотеке, людей в зале или элементов вектора. В этом разделе приводятся некоторые функции для обработки множеств, представленных векторами. Они широко используются при анализе и обработке данных.

intersect(a,b) — возвращает пересечение множеств для двух векторов **a** и **b**, то есть общие элементы векторов **b** и **b**. Результирующий вектор отсортирован по возрастанию. Если входные массивы не являются векторами, то они расцениваются как векторы-столбцы **a=a(:)** или **b=b(:)**.

intersect(a,b,'rows') — возвращает строки, общие для **a** и **b**, когда **a** и **b** представляют собой матрицы с одинаковым числом столбцов.

[c,ia,ib] = intersect(a,b) — также возвращает вектор-столбец индексов **ia** и **ib**, но так, что **c** = **a(ia)** и **c = b(ib)** (или **c = a(ia,:)** и **c = b(ib,:)**).

Пример:

```
» A = [1 7 2 6]; B = [7 2 3 4 6 1];
» [c,ia,ib] = intersect(A,B)
c =
   1   2   6   7
ia =
   1   3   4   2
ib =
   6   2   5   1
```

ismember(a,S) — возвращает вектор той же длины, что и исходный **a**, содержащий логическую единицу на месте того элемента вектора **a**, который принадлежит множеству **S**, и логический 0 на месте того элемента вектора **a**, который не принадлежит множеству **S**.

ismember(A,S,'rows') — возвращает вектор, содержащий логическую единицу, где строки матрицы **A** являются также строками матрицы **S** и логический 0 — в ином случае. **A** и **S** — это матрицы с одинаковым числом столбцов.

Пример:

```
» set = [0 1 3 5 7 9 11 15 17 19];
» a=[1 2 3 4 5 6 7 8];
» k = ismember(a,set)
k =
   1   0   1   0   1   0   1   0
```

setdiff(a,b) — возвращает разность множеств, то есть те элементы вектора **a**, которые не содержатся в векторе **b**. Результирующий вектор сортируется по возрастанию.

setduff(a,b,'rows') — возвращает те строки из матрицы **a**, которые не содержатся в матрице **b**. **a** и **b** представляют собой матрицы с одинаковым числом столбцов.

[c,i] = setdiff(...) — возвращает также вектор индексов **index** так, что **c = a(i)** или **c = a(i,:)**.

Если входной массив **a** не является вектором, то он расценивается как вектор-столбец **a(:)**.

Пример:

```
» a=[2 3 5 7 8 9 10 13 20];
» b=[1 4 5 6 8 9 4]
» c = setdiff(a,b)
c =
   2    3    7   10   13   20
```

setxor(a,b) — исключительное ИЛИ для векторов **a** и **b**. Результирующий вектор отсортирован.

setxor(a,b,'rows') — возвращает строки, которые не являются пересечениями матриц **a** и **b**. **a** и **b** — это матрицы с одинаковым числом столбцов.

[c,ia,ib] = setxor(...) — возвращает также векторы индексов **ia** и **ib** так, что **c** является отсортированной комбинацией элементов **c = a(ia)** и **c = b(ib)** или для комбинаций строк **c = a(ia,:)** и **c = b(ib,:)**.

Если входной массив **a** не является вектором, то он расценивается как вектор-столбец **a(:)**.

Пример:

```
» a = [-1 0 1 Inf -Inf NaN];
» b = [-2 pi 0 Inf];
» c = setxor(a,b)
c =
   -Inf  -2.0000  -1.0000   1.0000   3.1416      NaN
```

union(a,b) — возвращает вектор объединенных значений из **a** и **b** без повторяющихся элементов. Результирующий вектор сортируется в порядке возрастания.

union(a,b,'rows') — возвращает объединенные строки из **a** и **b**, не содержащие повторений (**a** и **b** — это матрицы с одинаковым числом столбцов).

[c,ia,ib] = union(...) — дополнительно возвращает **ia** и **ib** — векторы индексов, такие, что **c = a(ia)** и **c=b(ib)**, или для объединенных строк **c = a(ia,:)** и **c = b(ib,:)**.

Не векторный массив **a** расценивается как вектор-столбец **a(:)**.

Пример:

```
» a=[2,4,-4,9,0];b=[2,5,4];
» [c,ia,ib]=union(a,b)
c =
   -4    0    2    4    5    9
ia =
```

```
    3    5    4
ib =
    1    3    2
```

unique(a) — возвращает такие же значения элементов, как и в **a**, но не содержащие повторений. Результирующий вектор сортируется в порядке возрастания. Не векторный массив расценивается как вектор-столбец **a=a(:)**.

unique(a,'rows') — возвращает уникальные строки **a**.

[b,i,j] = unique(...) — дополнительно возвращает **i** и **j** — векторы индексов, такие что **b = a(i)** и **a = b(j)** (или **b = a(i,:)** и **a = b(j,:)**).

Примеры:

```
» b=[-2,3,5,4,1,-6,2,2,7]
b =
   -2    3    5    4    1   -6    2    2    7
» [c,i,j]=unique(b)
c =
   -6   -2    1    2    3    4    5    7
i =
    6    1    5    8    2    4    3    9
j =
    2    5    7    6    3    1    4    4    8
» a=[-2,3,5;4,1,-6;2,2,7;-2,3,5]
a =
   -2    3    5
    4    1   -6
    2    2    7
   -2    3    5
» c=unique(a,'rows')
c =
   -2    3    5
    2    2    7
    4    1   -6
```

5.1.8. Функции времени и даты

Ряд функций служит для возврата текущего времени и даты. Они перечислены ниже.

calendar — возвращает матрицу размером 6×7, содержащую календарь на текущий месяц. Календарь начинается с воскресенья (первый столбец) и завершается субботой.

calendar(d) — возвращает календарь на месяц, в который попадает день, заданный аргументом **d** от начала летоисчисления.

calendar(y,m) — возвращает календарь на месяц, заданный аргументом **m**, и год, заданный аргументом **y**.

calendar(...) — выдает календарь на экран. Примеры:

```
» calendar
```

```
          Nov 1998
  S   M   Tu   W   Th   F   S
  1   2   3    4   5    6   7
  8   9   10   11  12   13  14
 15  16   17   18  19   20  21
 22  23   24   25  26   27  28
 29  30   0    0   0    0   0
  0   0   0    0   0    0   0
» calendar(700477)
          Nov 1917
  S   M   Tu   W   Th   F   S
  0   0   0    0   1    2   3
  4   5   6    7   8    9   10
 11  12   13   14  15   16  17
 18  19   20   21  22   23  24
 25  26   27   28  29   30  0
  0   0   0    0   0    0   0
```

clock — возвращает вектор из 6 элементов, содержащий текущую дату и время в десятичной форме [год месяц день час минуты секунды]. Первые пять элементов этого вектора — целые числа. Шестой элемент имеет несколько десятичных знаков после запятой. Функция **fix(clock)** округляет число секунд до целого значения. Пример:

```
» c=clock
c =
  1.0e+003 *
    1.9980   0.0110   0.0280   0.0120   0.0180   0.0213
» fix(clock)
ans =
     1998       11       28       12       18       35
```

cputime — возвращает время работы процессора (в секундах) , использованное системой MATLAB с момента ее запуска. Это число может выйти за рамки внутреннего представления и тогда отсчет времени начинается заново. Пример:

```
» t1=cputime; w=surf(peaks(30));cputime-t1
ans =
   0.2200
```

str = date — возвращает строку , содержащую дату в формате дд-ммм-гггг (день-месяц-год). Пример:

```
» d = date
d =
```

28-Nov-1998

datenum — преобразовывает строку даты в порядковый номер даты, который отсчитывается с некоторого начального дня (по умолчанию с 01.01.00).

datenum(str) — преобразовывает дату, заданную строкой **str,** в порядковый номер даты. Строка **string** должна иметь один из следующих форматов: 0, 1, 2, 6, 13, 14, 15 или 16, как определено функцией **datestr**.

datenum(Y,M,D) — возвращает порядковый номер даты для соответствующих массивов элементов **Y, M,** и **D** (год, месяц, день) . Массивы **Y, M,** и **D** должны иметь одинаковую размерность (если один из них не скаляр).

datenum(Y,M,D,H,MI,S) — возвращает порядковый номер даты для соответствующих массивов элементов **Y, M, D, H, MI,** и **S** (год, месяц, день, часы, минуты, секунды) . Массивы **Y, M, D, H, MI,** и **S** должны иметь одинаковую размерность (если один из них не скаляр).

Пример:

```
» n1 = datenum('26-Nov-1998')
n1 =
    730085
» Y=[1998,2000];M=[1,12];D=23;N=datenum(Y,M,D)
N =
    729778    730843
```

datestr(D,dateform) — преобразовывает каждый элемент массива порядковых номеров даты **(D)** в строку. Произвольный аргумент **dateform** определяет формат результата, где **dateform** может быть номером или строкой в соответствии с приведенной ниже таблицей.

dateform (номер)	dateform (строка)	пример
0	'dd-mmM-yyyy HH:MM:SS'	01-Mar-1995 03:45
1	'dd-mmM-yyyy'	01-Mar-1995
2	'mm/dd/yy'	03/01/95
3	'mmm'	Mar
4	'm'	M
5	'mm'	3
6	'mm/dd'	03/01
7	'dd'	1
8	'ddd'	Wed
9	'd'	W
10	'yyyy'	1995
11	'yy'	95
12	'mmmyy'	Mar95
13	'HH:MM:SS'	15:45:17

datevec(A) — преобразовывает входные величины в массив размерностью n×6, каждая строка которого представляет собой вектор **[Y,M,D,H,MI,S]**. Первые пять элементов вектора — целые числа. Массив **A** может состоять или из строк, удовлетворяющих формату функции **datestr**, или из скалярных величин, созданных функциями **datenum** и **now**.

[Y, M., D, H, MI, S] = datevec (A) — возвращает компоненты вектора даты как индивидуальные переменные.

Любой компонент входного вектора, который не вписывается в нормальный диапазон дат, преобразуется в следующий диапазон (так, например, несуществующая дата June 31 преобразуется в July 1). Допускаются значения нулевого месяца и нулевого дня. Например:

```
» n = datevec('11/31/98')
n =
    1998     12      1      0      0      0
» n = datevec(710223)
n =
    1944      7     10      0      0      0
```

eomday(Y,M) — возвращает последний день месяца некоторого года, заданных соответственно элементами массивов **Y** и **M**. Пример (нахождение високосных лет этого столетия):

```
» y = 1900:1999;E = eomday(y,2*ones(length(y),1));y(find(E==29))
ans =
  Columns 1 through 6
    1904    1908    1912    1916    1920    1924
  Columns 7 through 12
    1928    1932    1936    1940    1944    1948
  Columns 13 through 18
    1952    1956    1960    1964    1968    1972
  Columns 19 through 24
    1976    1980    1984    1988    1992    1996
```

etime(t2,t1) — возвращает длительность промежутка времени (в секундах), задаваемого векторами **t1** и **t2.** Векторы должны удовлетворять формату, выдаваемому функцией **clock:**

T = [год месяц день час минуты секунды]. Функция работает некорректно, если в текущий промежуток времени попадут границы месяца или года, что, однако, случается крайне редко и исправляется при повторе операции. Пример (вычисляется время, затрачиваемое на быстрое преобразование Фурье с 2048 точками):

```
»x = rand(2048,1);t = clock; fft(x); etime(clock,t)
ans =
   0.0500
```

now — возвращает текущие время и дату в форме числа. Использование **rem(now,1)** возвращает только время, а **floor(now)** — только дату. Пример:

```
» t1 = now, t2 = rem(now,1)
t1 =
  7.3009e+005
t2 =
    0.6455
```

tic — запускает таймер, **toc** выводит время, прошедшее с момента запуска таймера.

t = toc — возвращает прошедшее время в **t**. Пример:

```
» tic,surf(peaks(50));toc
elapsed_time =
    0.7600
```

[N,S] = weekday(D) — возвращает день недели в виде числа **(N)** и в виде строки **(S)** для каждой даты массива **D**. Пример:

```
» D=[728647,735730];[N,S] = weekday(D)
N =
     2     1
S =
Mon       Sun
```

5.2. Элементарные функции

Элементарные функции, пожалуй, наиболее известный класс математических функций. Поэтому, не останавливаясь подробно на их описании, представим набор данных функций, имеющийся в составе системы MATLAB. Функции, представленные ниже, сгруппированы по функциональному назначению. В тригонометрических функциях углы измеряются в радианах. Все функции могут использоваться в конструкции вида **y=func(x)**, где **func** — имя функции. Обычно в такой форме задается информация о функции в системе MATLAB. Мы, однако, будем использовать для функций, возвращающих одиночный результат, более простую форму **func(x)**. Форма **[y,z,...]=func(x,...)** будет использоваться только в тех случаях, когда функция возвращает множественный результат.

5.2.1. Алгебраические и арифметические функции

В системе MATLAB определены следующие алгебраические и арифметические функции:

abs(X) — возвращает абсолютную величину для каждого численного элемента вектора **X**. Если **X** содержит комплексные числа, **abs (X)** вычисляет комплексный модуль каждого числа. Примеры:

```
abs(–5) = 5
abs(3+4i) = 5
» abs([1 -2 i 3i 2+3i])
ans =
   1.0000   2.0000   1.0000   3.0000   3.6056
```

exp(X) — возвращает экспоненту для каждого элемента **X**. Для комплексного числа **z = x + i*y** функция **exp(z)** вычисляет комплексную экспоненту: **e z =e x *(cos(y) + i*sin(y))**. Примеры:

```
» exp([1 2 3])
ans =
    7.3891   20.0855
» exp(2+3i)
ans =
 -7.3151 + 1.0427i
```

factor(n) — возвращает вектор-строку, содержащую простые множители числа n. Для массивов эта функция не применима. Пример:

```
f = factor(221)
f =
   13   17
```

G=gcd (A, B) — возвращает массив, содержащий наибольшие общие делители соответствующих элементов массивов целых чисел **A** и **B**. Функция **gcd (0,0)** возвращает значение 0, в остальных случаях вычисляются положительные целые числа для массива **G**.

[G, C, D] = gcd (A, B) — возвращает массив наибольших общих делителей **G** и массивов **C** и **D**, которые удовлетворяют уравнению: **A(i).*C(i) + B(i).*D(i) = G(i)**. Они полезны для выполнения элементарных эрмитовых преобразований. Примеры:

```
» A=[2 6 9];
» B=[2 3 3];
» gcd(A,B)
ans =
   2   3   3
» [G,C,D]=gcd(A,B)
G =
   2   3   3
C =
   0   0   0
D =
   1   1   1
```

lcm(A,B) — возвращает наименьшие общие кратные для соответствующих парных элементов массивов **A** и **B**. Массивы **A** и **B** должны содержать положительные целые числа и иметь одинаковую размерность (любой из них может быть скаляром). Пример:

```
» A=[1 3 5 4];
» B=[2 4 6 2];
» lcm(A,B)
ans =
   2   12   30    4
```

log(X) — возвращает натуральный логарифм элементов массива **X**. Для комплексного или отрицательного **z**, где **z = x + y*i**, вычисляется комплексный логарифм в виде: **log(z) = log(abs(z)) + i*atan2(y,x)**. Функция логарифма вычисляется для каждого элемента массива. Область функции может включать комплексные и отрицательные числа, что способно привести к непредвиденным результатам при некорректном использовании. Пример:

```
» X=[1.2 3.34 5 2.3];
» log(X)
ans =
  0.1823   1.2060   1.6094   0.8329
```

log2(X) — возвращает логарифм по основанию 2 элементов массива **X**.

[F,E] = log2(X) — возвращает массивы **F** и **E**. **F** — массив действительных значений, обычно в диапазоне **0.5 ≤ abs(F) < 1**. Для действительных **X**, **F** удовлетворяет уравнению вида: **X = F.*2.E**. **E** — массив целых чисел, который для действительных **X** удовлетворяет уравнению вида: **X = F.*2.E**. Для нулевых значений **X** вычисляются **F = 0 и E = 0**. Пример:

```
» X=[2 4.678 5;0.987 1 3];
» [F,E] = log2(X)
F =
   0.5000   0.5847   0.6250
   0.9870   0.5000   0.7500
E =
   2   3   3
   0   1   2
```

log10(X) — возвращает логарифм по основанию 10 для каждого элемента **X**. Область функции может включать комплексные числа, что способно привести к непредвиденным результатам при некорректном использовании. Пример:

```
» X=[1.4 2.23 5.8 3];
» log10(X)
ans =
  0.1461   0.3483   0.7634   0.4771
```

mod(x,y) — возвращает **x mod y**.

mod(X,Y) — возвращает остаток от выражения **X – Y.*floor(X./Y)** для ненулевого **Y** и возвращает **X** в другом случае. Поскольку операнды **X** и **Y** имеют одинаковый знак, функция **mod (X, Y)** возвращает тот же самый результат, что **rem (X, Y).** Однако для положительных **X** и **Y mod(–x,y) = rem(–x,y)+y**. Примеры:

```
» M = mod(5,2)
M =
   1
» mod(10,4)
ans =
   2
```

pow2(Y) — возвращает массив **X**, где каждый элемент есть 2^Y.

pow2(F,E) — вычисляет $X=F*2^E$ для соответствующих элементов **F** и **E**. Аргументы **F** и **E** — массивы действительных и целых чисел соответственно. Пример:

```
» d=pow2(pi/4,2)
d =
  3.1416
```

p = nextpow2(A) — возвращает такой показатель степени p, что $2^p \geq$ **abs(A).** Эта функция эффективно применяется для выполнения быстрого преобразования Фурье. Если **A** не является скалярной величиной, то **nextpow2** возвращает значение **nextpow2(length(A)).**

Пример:

```
» x=[2 6 7 8 9 3 4 5 6 7 7 8 4 3 2 4];
» length(x)
ans =
   16
» p = nextpow2(x)
p =
   4
» x=4;
» p = nextpow2(x)
p =
   2
» x=45;
» p = nextpow2(x)
p =
   6
```

p = primes(n) возвращает вектор-строку простых чисел, меньших или равных n. Пример:

```
» p = primes(25)
p =
   2   3   5   7   11   13   17   19   23
```

[N,D] = rat(X) — возвращает массивы **N** и **D** так, что **N./D** аппроксимирует **X** с точностью **1.e–6*norm(X(:),1)**. Даже при том, что все числа с плавающей запятой — рациональные числа, иногда желательно аппроксимировать их простыми рациональными числами, которые являются дробями, у которых числитель и знаменатель являются малыми целыми числами. Функция **rat** пытается это делать.

[N,D] = rat(X,tol) — возвращает массивы **N** и **D** так, что **N./D** аппроксимирует **X** с точностью **tol.**

rat(X) — без выходных аргументов просто выдает на экран цепную дробь.

rats(X,strlen) — возвращает ряд, полученный путем упрощенной рациональной аппроксимации элементов **X. strlen** — длина возвращаемой строки. Функция возвращает знак **«*»**, если полученное значение не может быть напечатано в строке, длина которой задана значением **strlen**. По умолчанию **strlen =13**.

S = rats(X) — равносильна функции **format rat**.

Пример:

```
» [g,j]=rat(pi,1e-10)
g =
    312689
j =
     99532
```

sqrt(A) — возвращает квадратный корень каждого элемента массива **X**. Для отрицательных и комплексных элементов **X** функция **sqrt(X)** вычисляет комплексный результат. Пример:

```
» A=[25 21.23 55.8 3];
» sqrt(A)
ans =
  5.0   4.6076   7.4699   1.7321
```

На рис. 5.1 представлены графики ряда распространенных алгебраических функций. Эти графики получены в результате исполнения следующего файла-сценария:

```
% Графики некоторых алгебраических функций
syms x
subplot(2,2,1),ezplot(x^2,[-5 5]),xlabel(''),grid on
subplot(2,2,2),ezplot(exp(x),[-2 2]),xlabel(''),grid on
subplot(2,2,3),ezplot(log(x),[0 5]),grid on
subplot(2,2,4),ezplot(sqrt(x),[0 10]),grid on
```

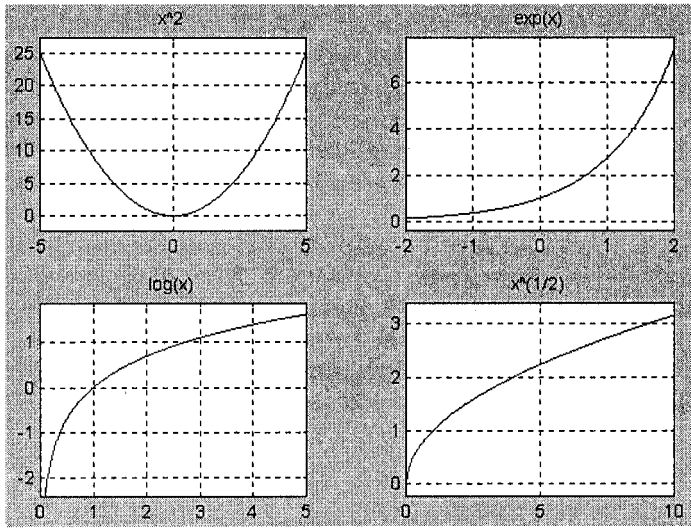


Рис. 5.1. Графики ряда алгебраических функций

Графики дают наглядный образец поведения представленных на них функций. Обратите внимание на применение графической команды **ezplot**, входящей в пакет Symbolic Math Toolboox (см. главу 10), и несколько необычное применение команды xlabel(''). Эта команда с аргументом в виде пустой строки снимает вывод обозначения горизонтальной оси в двух верхних графиках. Если этого не сделать, то символ "x" окажется наложенным на наименование функции нижних графиков, которое команда ezplot выводит автоматически поверх графиков.

5.2.2. Тригонометрические и обратные им функции

В системе MATLAB определены следующие тригонометрические и обратные тригонометрические функции. Функции вычисляются для каждого элемента массива. Входной массив допускает комплексные значения. Напоминаем, что все углы в функциях заданы в радианах.

acos (X) — возвращает арккосинус для каждого элемента **X**. Для действительных значений **X** в области [–1,1] **acos (X)** действителен в области [0,π], для действительных значений **X** вне области [–1, 1] **acos (X)** комплексен. Примеры:

```
»Y = acos (0.5)
Y =
   1.0472
» acos([0.5 1 2])
ans =
  1.0472          0          0 + 1.3170i
```

acot (X) — возвращает арккотангенс для каждого элемента **X**. Пример:

```
» Y=acot(0.1)
Y =
  1.4711
```

acsc(X) — возвращает арккосеканс для каждого элемента **X**. Пример:

```
» Y = acsc(3)
Y =
  0.3398
```

asec(X) — возвращает арксеканс для каждого элемента **X**. Пример:

```
» Y=asec(0.5)
Y =
     0 + 1.3170i
```

asin (X) — возвращает арксинус для каждого элемента **X**. Для действительных значений **X** в области [–1, 1] **asin (X)** действителен в области [-π/2,π/2], для действительных значений **X** вне области [–1, 1] **asin (X)** комплексен. Пример:

```
» Y = asin (0.278)
Y =
  0.2817
```

atan (X) — возвращает арктангенс для каждого элемента **X**. Для действительных значений **X atan (X)** находится в области [-π/2,π/2]. Пример:

```
» Y=atan(1)
Y =
  0.7854
```

atan2 (Y, X) — возвращает массив **P** той же размерности, что **X** и **Y**, содержащий поэлементно арктангенсы вещественных частей **Y** и **X**. Мнимые части игнорируются. Элементы **P** находятся в интервале [-π, π] . Специфический квадрант определен функциями **sign(Y)** и **sign(X)**. Это отличает полученный результат от результата **atan(Y/X)**, который ограничен интервалом [-π/2,π/2]. Пример:

```
» atan2(1,2)
ans =
  0.4636
```

cos(X) — возвращает косинус для каждого элемента **X**. Пример:

```
» X=[1 2 3];
» cos(X)
ans =
  0.5403  -0.4161  -0.9900
```

cot(X) — возвращает котангенс для каждого элемента **X**. Пример:

```
» Y = cot(2)
Y =
```

```
-0.4577
```

csc(X) — возвращает косеканс для каждого элемента **X**. Пример:

```
» X=[2 4.678 5;0.987 1 3];
» Y = csc(X)
Y =
   1.0998   -1.0006   -1.0428
   1.1985    1.1884    7.0862
```

sec(X) — возвращает массив той же размерности, что и **X**, состоящий из секансов элементов **X**. Пример:

```
» X=[pi/10 pi/3 pi/5];
» sec(X)
ans =
   1.0515    2.0000    1.2361
```

sin(X) — возвращает синус для каждого элемента **X**. Пример:

```
» X=[pi/2 pi/4 pi/6 pi];
» sin(X)
ans =
   1.0000    0.7071    0.5000    0.0000
```

tan(X) — возвращает тангенс для каждого элемента **X**. Пример:

```
» X=[0.08 0.06 1.09]
X =
   0.0800    0.0600    1.0900
» tan(X)
ans =
  0.802     0.0601    1.9171
```

Следующий файл-сценарий позволяет наблюдать графики четырех тригонометрических функций (рис. 5.2):

```
% Графики некоторых тригонометрических функций
syms x
subplot(2,2,1),ezplot(sin(x),[-5 5]),xlabel(''),grid on
subplot(2,2,2),ezplot(tan(x),[-5 5]),xlabel(''),grid on
subplot(2,2,3),ezplot(asin(x),[-1 1]),grid on
subplot(2,2,4),ezplot(atan(x),[-5 5]),grid on
```

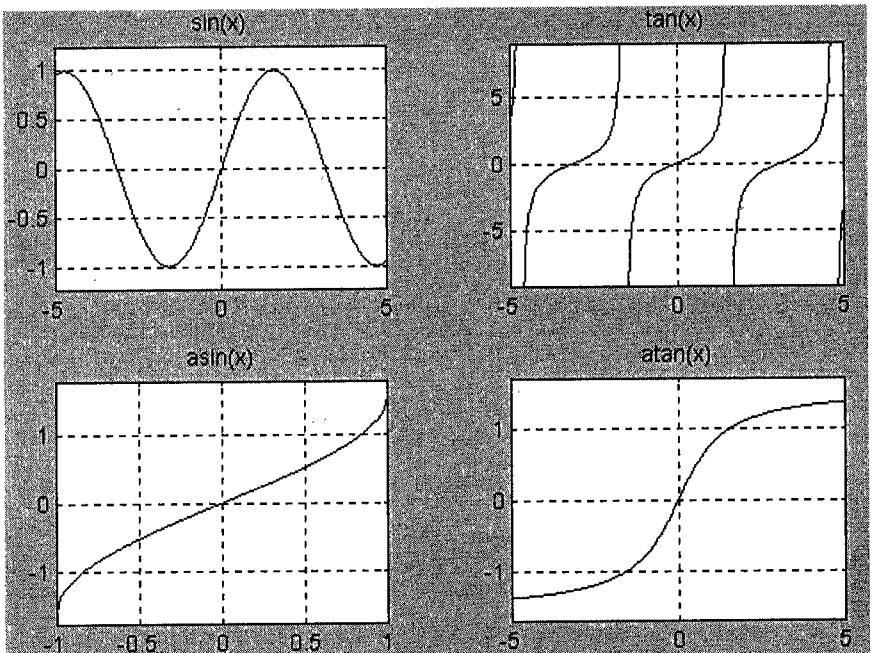


Рис. 5.2. Графики четырех тригонометрических функций

Поскольку многие тригонометрические функции периодичны, то появляется возможность формирования из них любопытных комбинаций, позволяющих создавать типовые тестовые сигналы, используемые при моделировании радиоэлектронных устройств. Следующий файл-сценарий строит графики для таких комбинаций, создающих из синусоиды три наиболее распространенных сигнала — прямоугольные, пилообразные и треугольные импульсы:

```
% Графики синусоидальной, прямоугольной, треугольной
% и пилообразной периодических функций
x=-10:0.01:10;
subplot(2,2,1),plot(x,0.8*sin(x)),xlabel('0.8*sin(x)')
subplot(2,2,2),plot(x,0.8*sign(sin(x))),xlabel('0.8*sgn(sin(x))')
subplot(2,2,3),plot(x,atan(tan(x/2))),xlabel('atan(tan(x/2))')
subplot(2,2,4),plot(x,asin(sin(x))),xlabel('asin(sin(x))')
```

Соответствующие графики представлены на рис. 5.3.

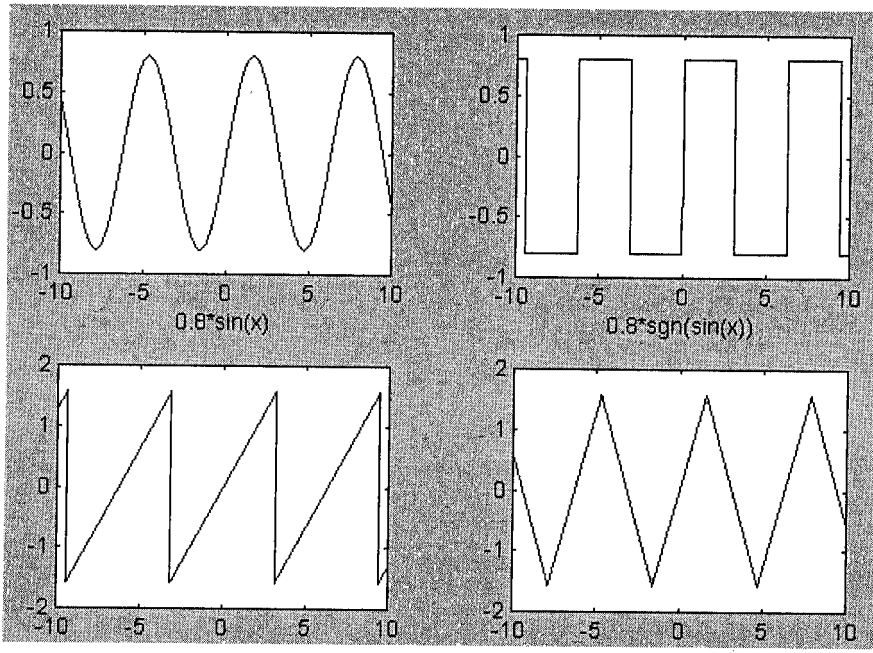


Рис. 5.3. График синусоиды, прямоугольных, пилообразных и треугольных колебаний

Дополнительный ряд графиков, полученных комбинациями элементарных функций, показан на рис. 5.4. Эти графики строятся следующим файлом-сценарием:

```
% Графики периодических сигналов без разрывов
x=-10:0.01:10;
subplot(2,2,1),plot(x,sin(x).^3),xlabel('sin(x)^3')
subplot(2,2,2),plot(x,abs(sin(x))),xlabel('abs(sin(x))'),axis([-10 10 -1 1]),
subplot(2,2,3),plot(x,tan(cos(x))),xlabel('tan(cos(x))')
subplot(2,2,4),plot(x,csch(sec(x))),xlabel('csch(sec(x))')
```

Эти графики неплохо моделируют сигналы, получаемые при выпрямлении синусоидального напряжения (или тока) и при прохождении синусоидальных сигналов через нелинейные цепи.

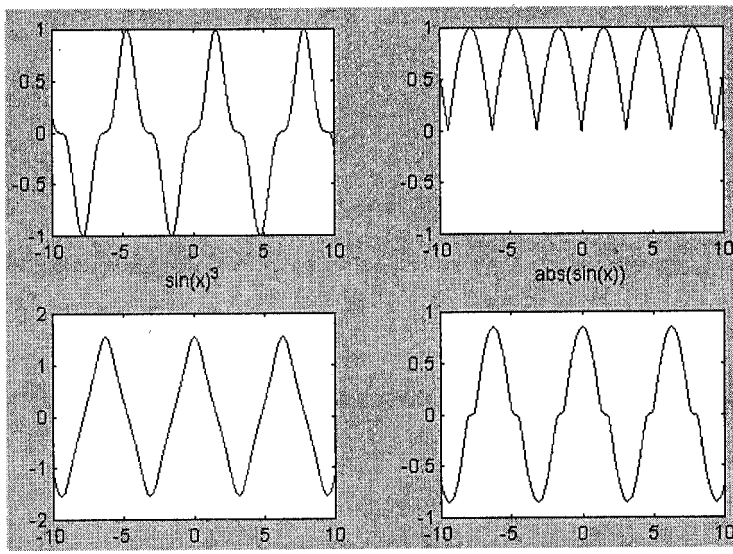


Рис. 5.4. Графики периодических сигналов без разрывов

5.2.3. Гиперболические и обратные им функции

Наряду с тригонометрическими функциями в математических расчетах часто используются и гиперболические функции. Ниже приводится список таких функций, определенных в системе MATLAB. Функции вычисляются для каждого элемента массива. Входной массив допускает комплексные значения. Все углы измеряются в радианах.

acosh (X) — возвращает гиперболический арккосинус для каждого элемента **X**. Пример:

```
»Y = acosh (0.7)
Y =
    0 + 0.7954i
```

acoth (X) — возвращает гиперболический арккотангенс для каждого элемента **X**. Пример:

```
»Y = acoth (0.1)
Y =
  0.1003 + 1.5708i
```

acsch(X) — возвращает гиперболический арккосеканс для каждого элемента **X**. Пример:

```
» Y = acsch(1)
Y =
  0.8814
```

asech(X) — возвращает гиперболический арксеканс для каждого элемента **X**. Пример:

```
» Y = asech(4)
Y =
     0 + 1.3181i
```

asinh (X) — возвращает гиперболический арксинус для каждого элемента **X**. Пример:

```
» Y = asinh (2.456)
Y =
  1.6308
```

atanh (X) — возвращает гиперболический арктангенс для каждого элемента **X**. Пример:

```
» X=[0.84 0.16 1.39];
» atanh (X)
ans =
  1.2212        0.1614        0.9065 + 1.5708i
```

cosh(X) — возвращает гиперболический косинус для каждого элемента **X**. Пример:

```
» X=[1 2 3];
» cosh(X)
ans =
  1.5431   3.7622   10.0677
```

coth(X) — возвращает гиперболический котангенс для каждого элемента **X**. Пример:

```
» Y = coth(3.987)
Y =
  1.0007
```

csch(x) — возвращает гиперболический косеканс для каждого элемента **X**. Пример:

```
» X=[2 4.678 5;0.987 1 3];
» Y = csch(X)
Y =
  0.2757   0.0186   0.0135
  0.8656   0.8509   0.0998
```

sech(X) — возвращает гиперболический секанс для каждого элемента **X**. Пример:

```
» X=[pi/2 pi/4 pi/6 pi];
» sech(X)
ans =
  0.3985   0.7549   0.8770   0.0863
```

sinh(X) — возвращает гиперболический синус для каждого элемента **X**. Пример:

```
» X=[pi/8 pi/7 pi/5 pi/10];
» sinh(X)
ans =
   0.4029   0.4640   0.6705   0.3194
```

tanh(X) — возвращает гиперболический тангенс для каждого элемента **X**. Пример:

```
» X=[pi/2 pi/4 pi/6 pi/10];
» tanh(X)
ans =
   0.9172   0.6558   0.4805   0.3042
```

Следующий М-файл-сценарий строит графики (рис. 5.5) ряда гиперболических функций:

```
% Графики гиперболических функций
syms x
subplot(2,2,1),ezplot(sinh(x),[-4 4]),xlabel(''),grid on
subplot(2,2,2),ezplot(cosh(x),[-4 4]),xlabel(''),grid on
subplot(2,2,3),ezplot(tanh(x),[-4 4]),grid on
subplot(2,2,4),ezplot(sech(x),[-4 4]),grid on
```

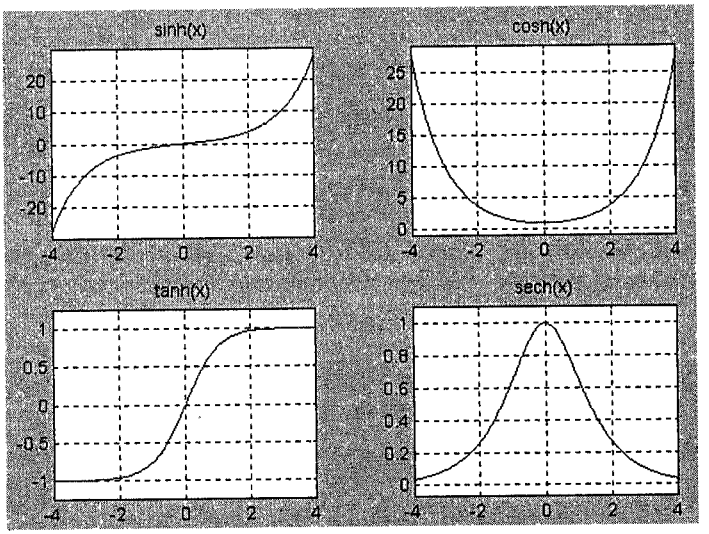


Рис. 5.5. Графики гиперболических функций

Нетрудно заметить, что гиперболические функции, в отличие от тригонометрических, не являются периодическими. Выбранные для графического представления функции дают примеры характерных нелинейностей.

В другом файле использованы команды для построения графиков (рис. 5.6) ряд обратных тригонометрических функций:

```
% Графики обратных гиперболических функций
syms x
subplot(2,2,1),ezplot(asinh(x),[-4 4]),xlabel(''),grid on
subplot(2,2,2),ezplot(acosh(x),[0 4]),xlabel(''),grid on
subplot(2,2,3),ezplot(atanh(x),[-1 1]),grid on
subplot(2,2,4),ezplot(asech(x),[0 1]),grid on
```

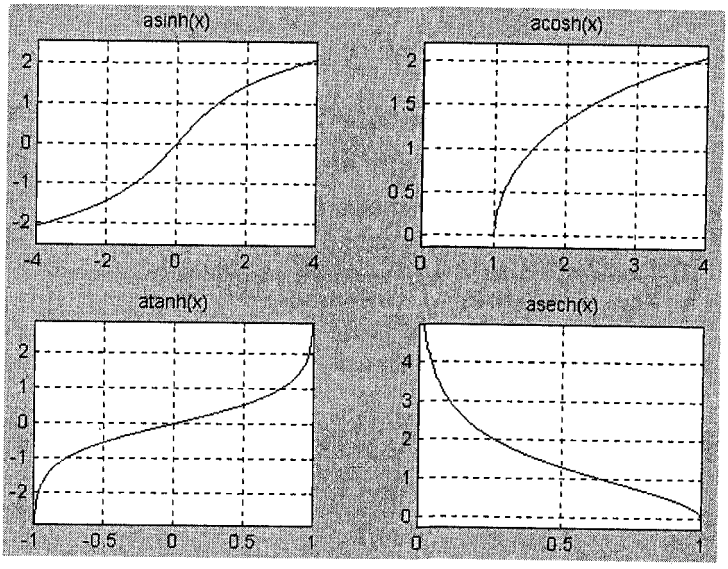


Рис. 5.6. Графики обратных гиперболических функций

На этих графиках хорошо видны особенности данного класса функций. Такие функции как обратный гиперболический синус и тангенс, "ценятся" за симметричный вид их графиков, дающий приближение к ряду типовых нелинейностей.

5.2.4. Функции округления и знака

Ряд особых функций служит для выполнения операций округления числовых данных и анализа их знака.

fix(A) — возвращает массив **A** с элементами, округленными до ближайшего к нулю целого числа. Для комплексного **A** действительные и мнимые части округляются отдельно. Примеры:

```
» A=[1/3 2/3; 4.99 5.01]
A =
   0.3333   0.6667
   4.9900   5.0100
```

```
» fix(A)
ans =
   0   0
   4   5
```

floor(A) — возвращает **A** с элементами, представляющими ближайшее меньшее или равное соответствующему элементу **A** целое число. Для комплексного **A** действительные и мнимые части преобразуются отдельно. Примеры:

```
» A=[-1/3 2/3; 4.99 5.01]
A =
  -0.3333   0.6667
   4.9900   5.0100
» floor(A)
ans =
   -1   0
    4   5
```

ceil(A) — возвращает ближайшее большее или равное **A** целое число. Для комплексного **A** действительные и мнимые части округляются отдельно. Примеры:

```
» a=-1.789;
» ceil(a)
ans =
   -1
» a=-1.789+i*3.908;
» ceil(a)
ans =
  -1.0000 + 4.0000i
```

rem(X,Y) — возвращает **X – fix(X./Y).*Y**, где **fix(X./Y)** — целая часть от частного **X/Y**.

Поскольку операнды **X** и **Y** имеют одинаковый знак, функция **rem(X,Y)** возвращает тот же самый результат, что **mod(X,Y)**. Однако для положительного **X** и **Y** — **rem(–x,y) = mod(–x,y)–y**. Функция **rem** возвращает результат, находящийся между **0** и **sign(X)*abs(Y)**. Если **Y=0**, функция **rem** возвращает **NaN**. Аргументы **X** и **Y** должны быть целые числа. Из-за неточного представления числа с плавающей запятой на компьютере вещественные (или комплексные) входные величины могут привести к непредвиденным результатам. Пример:

```
» X=[25 21 23 55 3];
» Y=[4 8 23 6 4];
» rem(X,Y)
ans =
   1   5   0   1   3
```

round(X) — возвращает округленные до ближайшего целого элементы массива **X**. Для комплексного **X** действительные и мнимые части округляются отдельно.

Пример:

```
» X=[5.675 21.6+4.897*i 2.654 55.8765];
» round(X)
ans =
  6.0000          22.0000 + 5.0000i   3.0000          56.0000
```

sign(X) — возвращает массив **Y** той же размерности, что и **X**, где каждый из элементов **Y** равен:

1 — если соответствующий элемент **X** больше **0**,
0 — если соответствующий элемент **X** равен **0**,
–1 — если соответствующий элемент **X** меньше **0**.
Для ненулевых комплексных **X** — **sign(X) = X./abs(X)**.
Пример:

```
» X=[-5 21 2 0 -3.7];
» sign(X)
ans =
  -1    1    1    0   -1
```

5.2.5. Функции комплексного аргумента

Для работы с комплексными числами и данными в MATLAB используются следующие функции:

angle(Z) — возвращает аргумент комплексного числа в радианах для каждого элемента массива комплексных чисел **Z**. Углы находятся в диапазоне [- π;+π]. Для комплексного **Z** модуль и аргумент вычисляются как: **R = abs(Z)** — модуль, **theta = angle(Z)** — аргумент . При этом формула **Z =R.*exp(i*theta)** дает переход от показательной формы представления комплексного числа к алгебраической. Примеры:

```
» Z=3+i*2
Z =
  3.0000 + 2.0000i
» theta = angle(Z)
theta =
   0.5880
» R = abs(Z)
R =
   3.6056
» Z =R.*exp(i*theta)
Z =
  3.0000 + 2.0000i
```

imag(Z) — возвращает мнимые части всех элементов массива **Z**. Пример:

```
» Z=[1+i, 3+2i, 2+3i];
» imag(Z)
ans =
   1    2    3
```

real(Z) — возвращает вещественные части всех элементов комплексного массива **Z**. Пример:

```
» Z=[1+i, 3+2i, 2+3i];
» real(Z)
ans =
   1   3   2
```

conj(Z) — возвращает комплексно-сопряженное число числу **Z**. Если **Z** комплексное, то **conj(Z) = real(Z) – i *imag(Z)**. Пример:

```
» conj(2+3i)
ans =
  2.0000 - 3.0000i
```

5.3 Специальные математические функции

Специальные математические функции являются решениями дифференциальных уравнений специального вида или обозначениями некоторых видов интегралов. Довольно полный обзор специальных функций дается в книгах [40-43], так что ниже мы ограничимся только указанием функций системы MATLAB, реализующих их вычисление. Набор специальных математических функций в системе MATLAB настолько представителен, что позволяет решать практически все задачи, связанные с применением таких функций.

5.3.1. Функции Эйри — airy

Функция Эйри формирует пару линейно независимых решений дифференциального уравнения вида:

$$\frac{d^2W}{dZ^2} - ZW = 0$$

Связь между функцией Эйри и модифицированной функцией Бесселя выражается формулой:

$$Ai(Z) = \left[\frac{1}{\pi}\sqrt{\frac{Z}{3}}\right] K_{\frac{1}{3}}(\zeta),$$

где

$$\zeta = \frac{2}{3} Z^{\frac{3}{2}}.$$

airy(Z) — возвращает функцию Эйри, **Ai(Z),** для каждого элемента комплексного массива **Z**.

airy(k,Z) — возвращает несколько значений результата в зависимости от значения k:

k=0 — тот же результат, что и **airy(Z)**;

k=1 — производную от **Ai(Z)**;

k=2 — функцию Эйри второго рода, **Bi(Z)**;

k=3 — производную от **Bi(Z)**.

Пример:

```
» D=[1,3+2i]
D =
  1.0000          3.0000 + 2.0000i
» S=airy(D)
S =
  0.1353          -0.0097 + 0.0055i
```

5.3.2. Функции Бесселя

Дифференциальное уравнение вида

$$z^2\frac{d^2y}{dz^2}+z\frac{dy}{dz}+(z^2-\nu^2)y=0,$$

где ν — неотрицательная константа, называется уравнением Бесселя, и его решения известны как функция Бесселя. **J_ν(z)** и **$J_{-\nu}$(z)** формируют фундаментальное множество решений уравнения Бесселя для неотрицательных значений ν (так называемые функции Бесселя первого рода):

$$J_\nu(z)=\frac{z^\nu}{2}\sum_{k=0}^{\infty}\frac{\left(\frac{-z^2}{4}\right)^k}{k!\Gamma(\nu+k+1)},$$

где для гамма-функции используется следующее представление:

$$\Gamma(a)=\int_0^\infty e^{-t}t^{a-1}dt$$

Второе решение уравнения Бесселя, линейно независимое от **J_ν(z),** определяется как

$$Y_\nu(z)=\frac{J_\nu(z)\cos(\nu\pi)-J_{-\nu}(z)}{\sin(\nu\pi)}$$

и задает функции Бесселя второго рода **Yν(z)**.

Функции Бесселя третьего рода (функции Ханкеля) и функции Бесселя первого и второго ряда связаны следующим выражением:

$$H^{(1)}{}_{\nu}(z) = J_{\nu}(z) + iY_{\nu}(z),$$
$$H^{(2)}{}_{\nu}(z) = J_{\nu}(z) - iY_{\nu}(z).$$

где **Jν(z)** — это **besselj** и **Yν(z)** — **bessely**.

besselj(nu,Z) — возвращает функцию Бесселя первого рода, **Jν(z)**, для каждого элемента комплексного массива **Z**. Порядок **nu** может не быть целым, однако должен быть вещественным. Аргумент **Z** может быть комплексным. Результат вещественный, если **Z** положительно. Если **nu** и **Z** массивы того же самого размера, то результат имеет тот же размер. Если любая входная величина — скаляр, результат расширяется к размеру другой входной величины. Если одна входная величина — вектор-строка и другая — вектор-столбец, результат представляет собой двумерный массив значений функции.

bessely(nu,Z) — возвращает функцию Бесселя второго рода, **Yν(z)**.
[J,ierr] = besselj(nu,Z) и **[Y,ierr] = bessely(nu,Z)** всегда возвращают массив с флагами ошибок: ierr = 1 — запрещенные аргументы; ierr = 2 — переполнение (возвращает **Inf**); ierr = 3 — некоторая потеря точности в приведении аргумента; ierr = 4 — недопустимая потеря точности, **Z** или **nu**, слишком большой; ierr = 5 — нет сходимости (возвращает **NaN**).

Примеры:

```
» S=[2-5i,4,7];T=[8,1,3];g=besselj(T,S)
g =
  -0.1114 - 0.0508i  -0.0660          -0.1676
» S=[2-5i,4,7];T=[8,1,3];[g,ierr]=bessely(T,S)
g =
   0.1871 - 0.0324i   0.3979          0.2681
ierr =
    0     0     0
```

besselh(nu,K,Z) — для **K = 1** или **2** вычисляет функцию Бесселя третьего рода (функцию Ханкеля) для каждого элемента комплексного массива **Z**. Если **nu** и **Z** массивы одинакового размера, то результат имеет тот же размер. Если одна из входных величин — скаляр, результат формируется по размеру другой входной величины. Если одна входная величина — вектор-строка и другая — вектор-столбец, результат представляет собой двумерный массив значений функции.
besselh(nu,Z) — использует по умолчанию **K = 1**.
besselh(nu,1,Z,1) — масштабирует **H1ν(z)** при **exp(–i*z)**.
besselh(nu,2,Z,1) — масштабирует **H2ν(z)** при **exp(+i*z)**.

[H,ierr] = besselh(...) — всегда возвращает массив с флагами ошибок: ierr = 1 — запрещенные аргументы; ierr = 2 — переполнение (возвращает **Inf**); ierr = 3 — некоторая потеря точности в приведении аргумента; ierr = 4 — недопустимая потеря точности, **Z** или **nu**, слишком большие; ierr = 5 — нет сходимости (возвращает **NaN**).

Пример:

```
» D=[1,3+2i];F=[3,2];besselh(F,D)
ans =
  0.0196 - 5.8215i   0.0509 - 0.0583i
```

Дифференциальное уравнение вида

$$z^2\frac{d^2y}{dz^2}+z\frac{dy}{dz}-(z^2+\nu^2)y=0,$$

где ν — неотрицательная константа — называется модифицированным уравнением Бесселя, и его решения известны как модифицированные функции Бесселя **Iν(z)** и **I-ν(z)**. **Kν(z)** — второе решение модифицированного уравнения Бесселя, линейно независимое от **Iν(z)**. **Iν(z)** и **Kν(z)** определяются как:

$$I_\nu(z)=\frac{z^\nu}{2}\sum_{k=0}^{\infty}\frac{\left(\frac{z^2}{4}\right)^k}{k!\Gamma(\nu+k+1)}$$

и

$$K_\nu(z)=\left(\frac{\pi}{2}\right)\frac{I_{-\nu}(z)-I_\nu(z)}{\sin(\nu\pi)}.$$

besseli(nu,Z) — вычисляет модифицированную функцию Бесселя первого рода.

besselk(nu,Z) — вычисляет модифицированную функцию Бесселя третьего рода.

besseli(nu,Z) — вычисляет модифицированную функцию Бесселя первого рода, **Iν(z)** для каждого элемента массива **Z**. Порядок **nu** может не быть целым, однако должен быть вещественным. Аргумент **Z** может быть комплексным. Результат вещественный, если **Z** положительно. Если **nu** и **Z** массивы того же самого размера, то результат имеет тот же размер. Если любая входная величина — скаляр, результат расширяется к размеру другой входной величины. Если одна входная величина — вектор-строка и другая — вектор-столбец, результат является двумерным массивом значений функции.

besselk(nu,Z) — вычисляет модифицированную функцию Бесселя второго рода, **Kν(z)**,для каждого элемента комплексного массива **Z**.

besseli(nu,Z,1) — вычисляет **besseli(nu,Z).*exp(–Z)**.

besselk(nu,Z,1) — вычисляет **besselk(nu,Z).*exp(–Z)**.

[I,ierr] = besseli(...) и **[K,ierr] = besselk(...)** всегда возвращают массив с флагами ошибок: ierr = 1 — запрещенные аргументы; ierr = 2 — переполнение (возвращает

Inf); ierr = 3 — некоторая потеря точности в приведении аргумента; ierr = 4 — недопустимая потеря точности, Z или nu, слишком большой; ierr = 5 — нет сходимости (возвращает NaN).

Примеры:

```
» D=[1,3+2i];F=[3,2];K=besseli(F,D)
K =
  0.0222          -1.2577 + 2.3188i
» D=[1,3+2i];F=[3,2];[K,ierr]=besselk(F,D)
K =
  7.1013          -0.0401 - 0.0285i
ierr =
   0    0
```

Естественно, что возможно построение графиков специальных функций. В качестве примера рассмотрим М-файл-сценарий, приведенный ниже:

```
x=0:0.1:10;
y0=besselj(0,x);
y1=besselj(1,x);
y2=besselj(2,x);
y3=besselj(3,x);
plot(x,y0,'-m',x,y1,'--r',x,y2,'-.k',x,y3,':b')
legend('bessel(0,x)','besselj(1,x)','besselj(2,x)','besselj(3,x)');
```

Рис. 5.7 иллюстрирует построение четырех функций Бесселя **besselj(n,x)** для n=0, 1, 2 и 3 с легендой, облегчающей идентификацию каждой кривой рисунка.

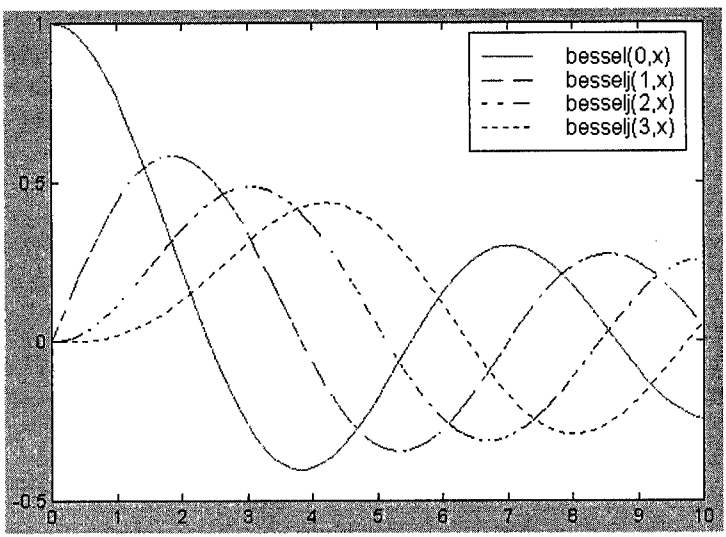


Рис. 5.7. Графики четырех функций Бесселя **besselj(n,x)**

Эти графики дают наглядное представление о поведении функций Бесселя, широко используемых при анализе поведения систем, описываемых линейными дифференциальными уравнениями второго порядка. Построение графиков функций Бесселя при комплексном аргументе можно найти в [33].

5.3.3. Бета-функция и ее варианты

Бета-функция определяется как:

$$B(z,w)=\int_0^1 t^{z-1}(1-t)^{w-1}dt=\frac{\Gamma(z)\Gamma(w)}{\Gamma(z+w)},$$

где Γ(z) — гамма-функция. Неполная бета-функция определяется по формуле:

$$I_x(z,w)=\frac{1}{B(z,v)}\int_0^x t^{z-1}(1-t)^{w-1}dt.$$

beta(Z,W) — возвращает бета-функцию для соответствующих элементов комплексных массивов **Z** и **W**. Массивы должны быть одинакового размера (или одна из величин может быть скаляром).

betainc(X,Z,W) — возвращает неполную бета-функцию. Элементы **X** должны быть в закрытом интервале [0, 1].

betaln(Z,W) — возвращает натуральный логарифм бета-функции, **log(beta(Z,W))**, без вычисления **beta(Z,W)**. Так как сама бета-функция может принимать очень большие или очень малые значения, ее логарифм иногда более полезен.

Пример:

```
» format rat;beta((1:10)',4)
ans =
   1/4
   1/20
   1/60
   1/140
   1/280
   1/504
   1/840
   1/1320
   1/1980
   1/2860
```

5.3.4. Эллиптические функции и интегралы

Эллиптические функции Якоби определяются интегралом:

$$u = \int_0^{\phi} \frac{d\theta}{(1 - m\sin^2\theta)^{\frac{1}{2}}}$$

и $sn(u) = \sin\phi$, $cn(u) = \cos\phi$, $dn(u) = (1 - \sin^2\phi)^{\frac{1}{2}}$, $am(u) = \phi$. В некоторых случаях при определении эллиптических функций используются модули **k** вместо параметра **m**. Они связаны выражением: $k^2 = m = \sin^2\alpha$.

[SN,CN,DN] = ellipj(U,M) — возвращает эллиптические функции Якоби **SN**, **CN**, и **DN**, вычисленные для соответствующих элементов — аргумента **U** и параметра **M**. Входные величины **U** и **M** должны иметь один и тот же размер (или любой из них может быть скалярным).

[SN,CN,DN] = ellipj(U,M,tol) — вычисляет эллиптическую функцию Якоби с точностью tol. Значение по умолчанию — **eps;** его можно увеличить, результат будет вычислен быстрее, но с меньшей точностью.

Пример:

```
» [SN,CN,DN]=ellipj([23,1],[0.5,0.2])
SN =
  474/719    1224/1481
CN =
 1270/1689   1457/2588
DN =
  399/451     538/579
```

Полные эллиптические интегралы первого и второго рода определяются следующим образом:

$$K(m) = \int_0^1 \left[(1 - t^2)(1 - mt^2)\right]^{-\frac{1}{2}} dt = \int_0^{\frac{\pi}{2}} \frac{d\theta}{(1 - m\sin^2\theta)^{\frac{1}{2}}},$$

$$E(m) = \int_0^1 (1 - t^2)^{-\frac{1}{2}} (1 - mt^2)^{\frac{1}{2}} dt = \int_0^{\frac{\pi}{2}} (1 - m\sin^2\theta)^{\frac{1}{2}} d\theta.$$

ellipke(M) — возвращает полный эллиптический интеграл первого рода для элементов **M.**

[K,E] = ellipke(M) — возвращает полный эллиптический интеграл первого и второго рода.

[K,E] = ellipke(M,tol) — вычисляет эллиптические функции Якоби с точностью tol. Значение по умолчанию — **eps**; его можно увеличить, результат будет вычислен быстрее, но с меньшей точностью.

Пример:

```
» [f,e]=ellipke([0.2,0.8])
f =
  707/426    1018/451
e =
  679/456    515/437
```

Для вычисления этих функций используется интерактивный метод арифметико-геометрического среднего (см. детали в Reference Book по системе MATLAB).

5.3.5. Функции ошибки

Функция ошибки определяется следующим образом:

$$\mathrm{erf}(X) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt.$$

erf(X) — возвращает значение функции ошибки для каждого элемента вещественного массива **X**.

Остаточная функция ошибки задается соотношением:

$$\mathrm{erfc}(X) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt = 1 - \mathrm{erf}(X).$$

erfc(X) — возвращает значение остаточной функции ошибки.

erfcx(X) — возвращает значение масштабированной остаточной функции ошибки. Эта функция определяется так:

$$\mathrm{erfcx}(x) = e^{x^2} \mathrm{erfc}(x).$$

erfinv(Y) — возвращает значение обратной функции ошибки для каждого элемента массива **Y**. Элементы массива **Y** должны уменьшаться в области **-1<Y< 1.**

Примеры:

```
» Y=[0.2,-0.3];a=erf(Y)
a =
   0.2227   -0.3286
» b=erfc(Y)
b =
   0.7773    1.3286
» c=erfcx(Y)
c =
   0.8090    1.4537
```

```
» d=erfinv(Y)
d =
   0.1791   -0.2725
```

При вычислении данных функций используется аппроксимация по Чебышеву (см. детали алгоритма в Reference Book по MATLAB).

5.3.6. Интегральная показательная функция

Интегральная показательная функция определяется следующим образом:

$$E_1(x) = \int_x^{\infty} \frac{e^{-t}}{t} dt$$

expint(X) — вычисляет интегральную показательную функцию для каждого элемента **X**. Пример:

```
» d=expint([2,3+7i])
d =
  0.0489          -0.0013 - 0.0060i
```

Для вычисления этой функции используется ее разложение в ряд — см. [40].

5.3.7. Гамма-функция и ее варианты

Гамма-функция определяется выражением:

$$\Gamma(a) = \int_0^x e^{-t} t^{a-1} dt.$$

Неполная гамма-функция определяется как:

$$P(x,a) = \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} dt.$$

gamma(A) — возвращает гамма-функцию от каждого элемента **A**. **A** должен быть вещественным.

gammainc(X,A) — возвращает неполную гамма-функцию соответствующих элементов **X** и **A**. Аргументы **X** и **A** должны быть вещественные и иметь одинаковый размер (или любой из них может быть скалярным).

gammaln(A) — возвращает логарифмическую гамма-функцию, **gammaln(A) = log(gamma(A))**. Команда **gammaln** избегает переполнения, которое может происходить, если вычислять логарифмическую гамма-функцию непосредственно, используя **log(gamma(A))**.

Примеры:

```
» f=[5,3];d=gamma(f)
d =
   24    2
» h=gammaln(f)
h =
   3.1781   0.6931
```

Гамма-функция имеет довольно сложный график (рис. 5.8), заслуживающий построения. Это можно осуществить с помощью следующего файла-сценария:

```
%Построение графика Гамма-функции
clear
syms x
ezplot(gamma(x)',[-4 4])
grid on
```

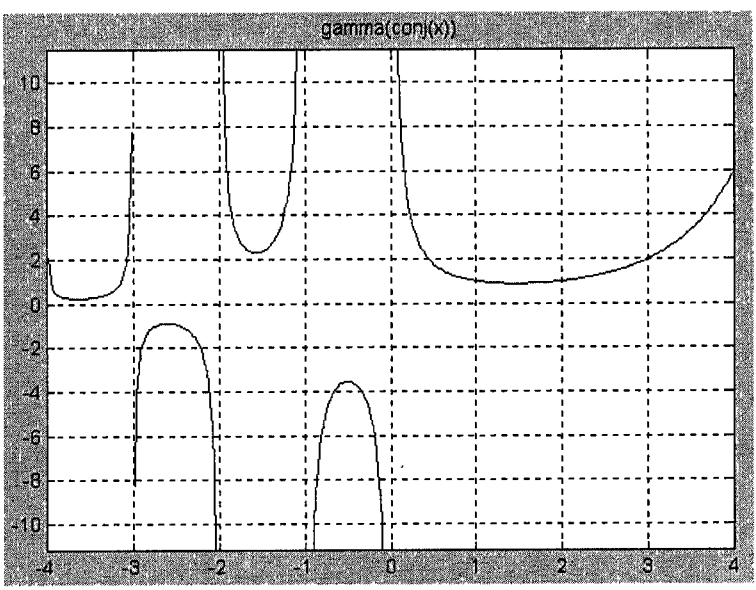


Рис. 5.8. График Гамма-функции

Гамма-функция вычисляется по известному алгоритму W. J. Kody (1989 г.). Для вычисления неполной гамма-функции используются рекурентные формулы, приведенные в [40].

5.3.8. Ортогональные полиномы Лежандра

Функция Лежандра определяется следующим образом:

$$P_n^m = (-1)^m (1-x^2)^{\frac{m}{2}} \frac{d^m P_n(x)}{dx^m},$$

где $P_n(x)$ — полином Лежандра степени n — определяется так:

$$P_n(x) = \frac{1}{2^n n!}\left[\frac{d^n (x^2-1)^n}{dx}\right].$$

legendre(n,X) — вычисляет связанные функции Лежандра степени n и порядка **m = 0,1, ..., n** для элементов вектора **X**. Аргумент n должен быть скалярным целым числом, меньше чем 256, и **X** должен содержать элементы с реальными значениями в области **-1 ≤ x≤ 1**. Возвращаемый массив **P** имеет большую размерность, чем **X,** и каждый элемент **P(m+1,d1,d2...)** содержит связанную функцию Лежандра степени n и порядка m, вычисленную в **X(d1,d2...)**.

legendre(...,'sch') — вычисляет полунормализованные связанные функции Лежандра.

Пример:

```
» g=rand(3,2);legendre(3,g)
ans(:,:,1) =
  -0.4469  -0.0277   0.1534
  -0.0558   1.4972  -2.0306
   5.4204   0.2775   4.0079
 -10.5653 -14.9923  -2.7829
ans(:,:,2) =
  -0.4472  -0.3404   0.0538
   0.0150  -1.0567  -1.9562
   5.3514   5.7350   4.4289
 -10.7782  -7.3449  -3.4148
```

Формулы для вычисления функций Лежандра приведены в [40].

5.4. Функции создания матриц и векторов

Матрицы представляют собой самый распространенный объект системы MATLAB. Ниже описываются основные операции с элементарными матрицами. По обилию матричных операторов и функций MATLAB является лидером среди массовых систем компьютерной математики.

5.4.1. Функция создания единичной матрицы eye

Для создания единичной матрицы (она обычно обозначается как **E**) служит функция **eye**:

eye(n) — возвращает единичную матрицу размера n×n.

eye(m,n) или **eye([m n])** — возвращают матрицу размера m×n с единицами по диагонали и нулями в остальных ячейках.

eye(size(A)) — возвращает единичную матрицу того же размера, что и у **A**.

Единичная матрица не определена для многомерных массивов. Функция **y = eye([2,3,4])** приведет к ошибке. Пример:

```
» t=eye(4,5)
t =
   1   0   0   0   0
   0   1   0   0   0
   0   0   1   0   0
   0   0   0   1   0
```

5.4.2. Функция создания матрицы с единичными элементами ones

Для создания матриц, все элементы которых единицы, используется функция:

ones(n) — возвращает матрицу размера m×n, состоящую из единиц. Если n не скаляр, то появится сообщение об ошибке.

ones(m,n) или **ones([m n])** — возвращают матрицу размера m×n, состоящую из единиц.

ones(d1,d2,d3...) или **ones([d1 d2 d3...])** — возвращают массив из единиц с размером **d1xd2xd3x....**

ones(size(A)) — возвращает массив единиц такой же размерности, что и у **A**.

Пример:

```
» s=ones(3,4)
 s =
   1   1   1   1
   1   1   1   1
   1   1   1   1
```

5.4.3. Функция создания матрицы с нулевыми элементами zeros

Иногда нужны матрицы, все элементы которых являются нулями. Следующая функция обеспечивает создание таких матриц:

zeros(n) — возвращает матрицу размера n×n, содержащую нули. Если n не скаляр, то появится сообщение об ошибке.

zeros(m,n) или **zeros([m n])** — возвращают матрицу размера m×n, состоящую из нулей.

zeros(d1,d2,d3...) или **zeros([d1 d2 d3...])** — возвращают массив из нулей размера **d1xd2xd3x....**

zeros(size(A)) — возвращает массив нулей того же размера, что и у **A**.

Пример:

```
» D=zeros(3,2)
D =
   0   0
   0   0
   0   0
```

5.4.4. Функция создания линейного массива равноотстоящих точек linspace

Функция **linspace** формирует линейный массив равноотстоящих узлов. Это подобно оператору :, но дает прямой контроль над числом точек. Применяется в следующих формах:

linspace(a,b) — возвращает линейный массив из 100 точек, равномерно распределенных между **a** и **b**.

linspace(a,b,n) — генерирует n точек, равномерно распределенных в интервале от a до b.

Пример:

```
» M=linspace(4,20,14)
M =
 Columns 1 through 7
  4.0000   5.2308   6.4615   7.6923   8.9231  10.1538  11.3846
 Columns 8 through 14
 12.6154  13.8462  15.0769  16.3077  17.5385  18.7692  20.0000
```

5.4.5. Функция создания вектора равноотстоящих точек в логарифмическом масштабе logspace

Функция **logspace** генерирует вектор равноотстоящих в логарифмическом масштабе точек. Она особенно эффективна при создании вектора частоты. Это логарифмический эквивалент функции **linspace** и оператора :.

logspace(a,b) — возвращает вектор-строку **y** из 50 равноотстоящих в логарифмическом масштабе точек между декадами 10^a и 10^b.

logspace(a,b,n) — возвращает n точек между декадами 10^a и 10^b.

logspace(a,pi) — возвращает точки в интервале между 10^a и **pi**, что особенно полезно для цифровых сигналов, где частоты над этим интервалом расположены вокруг единичной окружности.

Все аргументы функции **logspace** должны быть скалярными величинами. Пример:

```
» L=logspace(1,2,14)
L =
 Columns 1 through 7
  10.0000  11.9378  14.2510  17.0125  20.3092  24.2446  28.9427
 Columns 8 through 14
  34.5511  41.2463  49.2388  58.7802  70.1704  83.7678 100.0000
```

5.4.6. Функции создания массивов со случайными элементами — rand и randn

Функция **rand** генерирует массивы случайных чисел, чьи элементы равномерно распределены в промежутке (0,1):

rand(n) — возвращает матрицу размера n×n с элементами, распределенными по равномерному закону. Если n не скаляр, то появится сообщение об ошибке.

rand(m,n) или **rand([m n])** — возвращают матрицу размера m×n с элементами, распределенными по равномерному закону.

rand(m,n,p,...) или **rand([m n p...])** — возвращает массив с элементами, распределенными по равномерному закону.

rand(size(A)) — возвращает массив того же размера,что и у **A,** с элементами, распределенными по равномерному закону.

rand (без аргументов) — возвращает одно случайное число, которое изменяется при каждом последующем вызове.

rand('state') — возвращает вектор с 35 элементами, содержащий текущее состояние однородного генератора. Для изменения состояния генератора:

rand('state',s) — сбрасывает состояние к s.

rand('state',0) — сбрасывает генератор в начальное состояние.

rand('state',j) — для целых j, сбрасывает генератор в j -е состояние.

rand('state',sum(100*clock)) — каждый раз сбрасывает генератор в произвольное состояние.

Пример:

```
» Y=rand(4,3)
Y =
   0.9501   0.8913   0.8214
   0.2311   0.7621   0.4447
   0.6068   0.4565   0.6154
   0.4860   0.0185   0.7919
```

Проверить равномерность распределения случайных чисел можно, построив большое число точек на плоскости со случайными координатами. Это делается с помощью команд:

```
» X=rand(1000,1);
» Y=rand(1000,1);
» plot(X,Y,'.')
```

Полученный при этом график показан на рис. 5.9. Нетрудно заметить, что точки довольно равномерно распределены на плоскости, так что нет оснований не доверять заданному закону распределения координат точек.

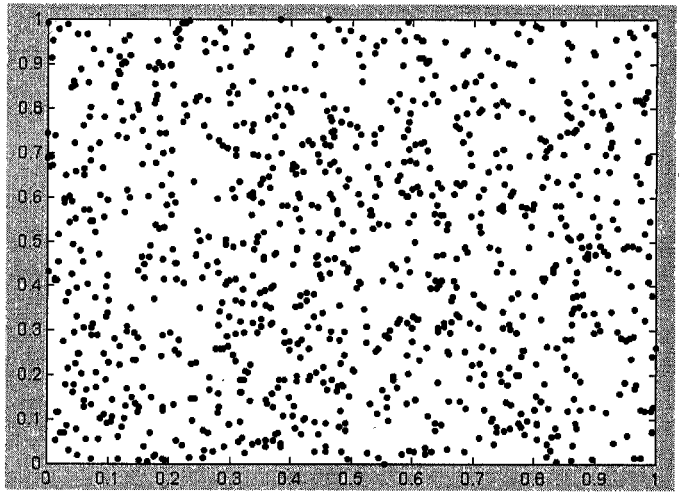


Рис. 5.9. Случайные точки с равномерным распределением координат на плоскости

Функция **randn** генерирует массив со случайными элементами , распределенными по нормальному закону с математическим ожиданием 0 и среднеквадратическим отклонением 1.

randn(n) — возвращает матрицу размера n×n с элементами, распределенными по нормальному закону. Если n не скаляр, то появится сообщение об ошибке.

randn(m,n) или **randn([m n])** — возвращают матрицу размера m×n с элементами, распределенными по нормальному закону.

randn(m,n,p,...) или **randn([m n p...])** — возвращают массив с элементами, распределенными по нормальному закону.

randn(size(A)) — возвращает массив того же размера, что и у **A** , с элементами, распределенными по нормальному закону.

randn (без аргументов) — возвращает одно случайное число, которое изменяется при каждом последующем вызове.

randn('state') — возвращает 2-х элементный вектор, включающий текущее состояние нормального генератора. Для изменения состояния генератора:

randn('state',s) — сбрасывает состояние к **s**.

randn('state',0) — сбрасывает генератор в начальное состояние.
randn('state',j) — для целых j, сбрасывает генератор в j -е состояние.
randn('state',sum(100*clock)) — каждый раз сбрасывает генератор в произвольное состояние.

Пример:

```
» Y=randn(4,3)
Y =
  -0.4326  -1.1465   0.3273
  -1.6656   1.1909   0.1746
   0.1253   1.1892  -0.1867
   0.2877  -0.0376   0.7258
```

Проверить распределение случайных чисел по нормальному закону можно, построив гистограмму распределения большого количества чисел. Например, следующие команды

```
» Y=randn(10000,1);
» hist(Y,100)
```

строят гистограмму (рис. 5.10) из 100 столбцов для 10000 случайных чисел с нормальным распределением. Из рисунка видно, что огибающая гистограммы действительно близка к нормальному закону распределения.

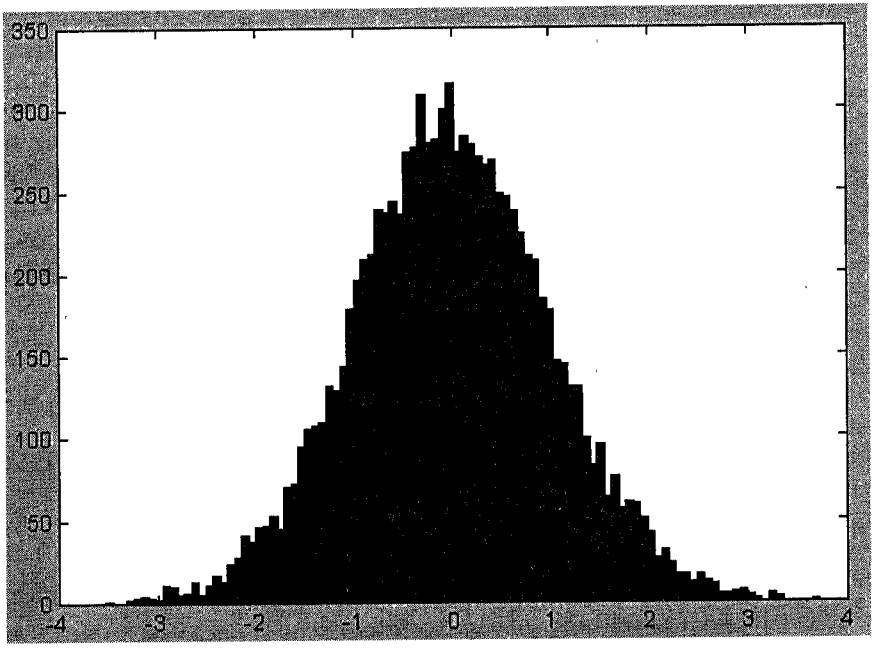


Рис. 5.10. Гистограмма для 10000 нормально распределенных чисел в 100 интервалах

В пакете расширения Statistics Toolbox можно найти множество статистических функций, в том числе для генерации случайных чисел с различными законами распределения и определения их статистических характеристик.

5.5. Основные операции над матрицами

Далее рассматриваются функции, осуществляющие операции над двумя или несколькими массивами или матрицами. Эти операции широко используются при решении задач линейной алгебры. Смысл операций поясняется заголовком соответствующего раздела. Поэтому вступительных слов в подразделах этого раздела нет. Как правило описание сути соответствующих операций не приводится, поскольку считается, что читатель должен быть с ней знаком. Желающим познакомиться с теорией матричных операций можно посоветовать специальную литературу, например, книгу [38].

5.5.1. Функция конкатенации матриц — cat

C = cat(dim,A,B) — объединяет массивы **A** и **B** в соответствии со спецификацией размерности dim и возвращает объединенный массив.

C = cat(dim,A1,A2,A3,A4,...) — объединяет все входные массивы (**A1, A2, A3, A4** и т. д.) в соответствии со спецификацией размера dim и возвращает объединенный массив.

cat(2,A,B) — это то же самое, что и **[A,B]**, а **cat(1,A,B)** это то же самое, что и **[A;B]**.

Пример:

```
» A =[2,4;3,5];B=[8,7;9,0];C=cat(1,A,B)
C =
   2   4
   3   5
   8   7
   9   0
```

5.5.2. Функция создания матриц с заданной диагональю — diag

X = diag(v,k) , где **v** — вектор, состоящий из n компонент, возвращает квадратную матрицу **X** порядка **n+abs(k)** с элементами **v** на k -ой диагонали: при k=0 — это главная диагональ, при k>0 — это верхняя диагональ, при k<0 — это нижняя диагональ. Остальные элементы — нули.

X = diag(v) — помещает вектор **v** на главную диагональ (то же, что и в предыдущем случае при k=0).

v = diag(X,k) — для матрицы **X** возвращает вектор-столбец, состоящий из элементов k-ой диагонали матрицы **X**.

v = diag(X) — возвращает главную диагональ матрицы **X** (то же, что и в предыдущем случае при k=0).

Примеры:

```
» v=[2,3];X=diag(v,2)
X =
   0   0   2   0
   0   0   0   3
   0   0   0   0
   0   0   0   0
» X=[2,5,45,6;3,5,4,9;7,9,4,8;5,66,45,2];v=diag(X,0)
v =
   2
   5
   4
   2
```

5.5.3. Функции перестановки fliplr, flipud и perms

B = fliplr(A) — переставляет столбцы матрицы **A** относительно вертикальной оси. Пример:

```
» F=[1,2,3;5,45,3]
F =
   1   2   3
   5  45   3
» fliplr(F)
ans =
   3   2   1
   3  45   5
```

B = flipud(A) — переставляет строки матрицы **A** относительно горизонтальной оси. Пример:

```
» F=[3,2,12;6,3,2]
F =
   3   2  12
   6   3   2
» flipud(F)
ans =
   6   3   2
   3   2  12
```

perms(v) — возвращает матрицу **P**, строки которой — это все возможные перестановки элементов вектора **v**. Матрица **P** содержит n! строк и n столбцов. Пример:

```
» v=[1 4 6]
```

```
v =
   1   4   6
» P=perms(v)
P =
   6   4   1
   4   6   1
   6   1   4
   1   6   4
   4   1   6
   1   4   6
```

5.5.4. Функции вычисления произведений prod, cumprod и cross

prod(A) — возвращает произведение элементов массива, если **A** — вектор, или возвращает вектор-строку, содержащую произведения элементов каждого столбца, если **A** — матрица.

prod(A,dim) — возвращает произведение элементов массива по столбцам или по строкам матрицы, в зависимости от значения скаляра dim.

Пример:

```
» A=[1 2 3 4; 2 4 5 7; 6 8 3 4]
A =
   1   2   3   4
   2   4   5   7
   6   8   3   4
» B=prod(A)
B =
  12   64   45  112
```

cumprod(A) — возвращает произведение с накоплением. Если **A** — вектор, **cumprod(A)** возвращает вектор, содержащий произведение с накоплением элементов вектора **A.** Если **A** — матрица, **cumprod(A)** возвращает матрицу того же размера, как и **A**, содержащую произведение с накоплением для каждого столбца матрицы **A**.

cumprod(A,dim) — возвращает произведение с накоплением элементов по строкам или по столбцам, в зависимости от значения скаляра dim. Например, **cumprod(A,1)** дает прирост первому индексу (строки), таким образом выполняя умножение по столбцам матрицы **A.**

Примеры:

```
» A=[1 2 3; 4 5 6; 7 8 9]
A =
   1   2   3
   4   5   6
   7   8   9
» B = cumprod(A)
```

```
B =
   1   2   3
   4  10  18
  28  80 162
B = cumprod(A,1)
B =
   1   2   3
   4  10  18
  28  80 162
```

cross(U,V) — возвращает векторное произведение векторов **U** и **V** в трехмерном пространстве. То есть, **W = U x V**. **U** и **V** — обычно векторы с тремя элементами.

cross(U,V,dim), где **U** и **V** — многомерные массивы, возвращает векторное произведение **U** и **V** по размерности, определенной скаляром dim. **U** и **V** должны иметь один и тот же размер, и **size(U,dim)**,и **size(V,dim)** должны быть равны 3. Пример:

```
» a = [6 5 3]; b = [1 7 6];c = cross(a,b)
c =
   9  -33   37
```

5.5.5. Функции суммирования sum, cumsum

Определены следующие функции суммирования элементов массивов:

sum(A) — возвращает сумму элементов массива, если **A** — вектор, или возвращает вектор-строку, содержащую сумму элементов каждого столбца, если **A** — матрица.

sum(A,dim) — возвращает произведение элементов массива по столбцам или по строкам матрицы, в зависимости от значения скаляра dim.

Пример:

```
» A=magic(4)
A =
  16   2   3  13
   5  11  10   8
   9   7   6  12
   4  14  15   1
» B = sum(A)
B =
  34  34  34  34
```

cumsum(A) — выполняет суммирование с накоплением. Если **A** — вектор, **cumsum(A)** возвращает вектор, содержащий суммирование с накоплением элементов вектора **A**. Если **A** — матрица, **cumsum(A)** возвращает матрицу того же размера, что и **A**, содержащую суммирование с накоплением для каждого столбца матрицы **A**.

cumsum(A,dim) — выполняет суммирование с накоплением элементов по размерности, точно определенной скаляром dim. Например, **cumsum(A,1)** выполняет суммирование по столбцам.

Пример:

```
» A=magic(4)
A =
   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1
» B = cumsum(A)
B =
   16    2    3   13
   21   13   13   21
   30   20   19   33
   34   34   34   34
```

5.5.6. Функции формирования матриц repmat и reshape

Для создания матриц, состоящих из других матриц, служат следующие функции:

repmat(A,m,n) — формирует матрицу **B**, состоящую из m×n копий матрицы **A**.

repmat(A,n) — формирует матрицу, состоящую из n×n копий матрицы **A**.

repmat(A,[m n]) — дает результат, подобный **repmat(A,m,n)**.

repmat(A,[m n p...]) — формирует многомерный массив (m×n×p...), состоящий из копий матрицы **A**. **A** может быть так же многомерной.

repmat(A,m,n) , где **A** — скаляр, формирует mxn матрицу со значениями **A**. Это вычисляется намного быстрее, чем **a*ones(m,n)**.

Пример:

```
F =
    3    2
   43   32
» repmat(F,2,3)
ans =
    3    2    3    2    3    2
   43   32   43   32   43   32
    3    2    3    2    3    2
   43   32   43   32   43   32
```

reshape(A,m,n) — возвращает матрицу **B** размерностью m×n, сформированную из **A** путем последовательной выборки по столбцам. Если **A** имеет не m×n элементов, то выдается сообщение об ошибке.

reshape(A,m,n,p,...) или **B = reshape(A,[m n p...])** — возвращает N-размерный D-массив с такими же элементами, как **A**, но имеющий размер **mxnxp...** . Произведение m×n×p... должно равно значению **prod(size(x))**.

reshape(A,siz) — возвращает N-размерный D-массив с такими же элементами, как **A**, но перестроенный к **siz**-вектору, определяющему размер массива.

Пример:

```
» F=[3,2,7,4;4,3,3,2;2,2,5,5]
F =
   3   2   7   4
   4   3   3   2
   2   2   5   5
» reshape(F,2,6)
ans =
   3   2   3   7   5   2
   4   2   2   3   4   5
```

5.5.7. Функция поворота матриц rot90

Следующая функция обеспечивает поворот матрицы:

rot90(A) — осуществляет поворот матрицы **A** на 90 градусов против часовой стрелки.

rot90(A,k) — осуществляет поворот матрицы **A** на величину 90*k , где k — целое число.

Пример:

```
» M=[3,2,7;3,3,2;1,1,1]
M =
   3   2   7
   3   3   2
   1   1   1
» rot90(M)
ans =
   7   2   1
   2   3   1
   3   3   1
```

5.5.8. Функции выделения треугольных частей матриц tril и triu

При выполнении ряда матричных вычислений возникает необходимость в выделении треугольной части матриц. Следующие функции обеспечивают такое выделение:

tril(X) — возвращает нижнюю треугольную часть матрицы **X**.

tril(X,k) — возвращает нижнюю треугольную часть матрицы **X**, начиная с k-ой диагонали. При k=0 — это главная диагональ, при k>0 — это верхняя диагональ, при k<0 — это нижняя диагональ.

Пример:

```
» M=[3,1,4;8,3,2;8,1,1]
M =
   3   1   4
   8   3   2
   8   1   1
» tril(M)
ans =
   3   0   0
   8   3   0
   8   1   1
```

triu(X) — возвращает верхнюю треугольную часть матрицы **X**.

triu(X,k) — возвращает верхнюю треугольную часть матрицы **X**, начиная с k-ой диагонали. При k=0 — это главная диагональ, при k>0 — это верхняя диагональ, при k<0 — это нижняя диагональ. Пример:

```
M =
   3   1   4
   8   3   2
   8   1   1
» triu(M)
ans =
   3   1   4
   0   3   2

   0   0   1
```

5.6. Специальные матрицы

В этом разделе рассматриваются функции, относящиеся к различным специальным матрицам. Они довольно широко используются при решении достаточно серьезных задач матричного исчисления. В связи с тем, что назначение соответствующих функций вытекает из их наименования, мы не будем сопровождать описание вводными комментариями. Соответствующие подробные определения вы найдете в книге [38]. Рекомендуем читателю построить графики, представляющие элементы этих матриц.

5.6.1. Вычисление сопровождающей матрицы — функция compan

compan(u) — возвращает сопровождающую матрицу, первая строка которой — **–u(2:n)/u(1)**, где **u** — вектор полиномиальных коэффициентов. Собственные значения **compan (u)** — корни многочлена. Пример: для многочлена x^3+x^2-6x-8 вектор полиномиальных коэффициентов **r**

```
» r=[1,1,-6,-8]
r =
```

```
    1    1   -6   -8
» A=compan(r) % сопровождающая матрица
A =
   -1    6    8
    1    0    0
    0    1    0
» eig(compan(r)) % корни многочлена
ans =
  -2.0000
   2.5616
  -1.5616
```

5.6.2. Вычисление тестовых матриц — функция gallery

[A,B,C,...] = gallery('tmfun',P1,P2,...) — возвращает тестовые матрицы, определенные строкой **tmfun**, где **tmfun** — это имя семейства матриц, выбранное из списка. **P1, P2,...** — входные параметры, требуемые для конкретного семейства матриц. Число используемых параметров **P1,P2,...** изменяется от матрицы к матрице. **gallery** хранит более 50 различных тестовых матричных функции, полезных для тестирующих алгоритмов и других целей. Пример:

```
» A=gallery('dramadah',5,2)
A =
    1    1    0    1    0
    0    1    1    0    1
    0    0    1    1    0
    0    0    0    1    1
    0    0    0    0    1
```

5.6.3. Вычисление матрицы Адамара — функция hadamard

H = hadamard(n) — возвращает матрицу Адамара порядка **n**. Матрица Адамара — это матрица, составленная из 1 и -1, столбцы которой ортогональны, так что справедливо соотношение **H'*H = n*I**, где **[n n] = size(H)** и **I = eye(n,n)**. Матрицы Адамара применяются в различных областях, включающих комбинаторику, численный анализ, обработку сигналов. Матрица Адамара размером n×n при n > 2 существует только если **rem(n,4) = 0**. Данный алгоритм вычисляет матрицы Адамара для тех случаев, когда величины n, n/12 или n/20 являются степенями по основанию 2. Пример:

```
» H = hadamard(4)
H =
    1    1    1    1
    1   -1    1   -1
    1    1   -1   -1
    1   -1   -1    1
```

5.6.4. Вычисление матрицы Ганкеля — функция hankel

hankel(c) — возвращает квадратную матрицу Ганкеля, первый столбец которой совпадает с вектором **c** и все элементы, лежащие ниже второй главной диагонали, равны 0.

hankel(c,r) — возвращает матрицу Ганкеля , первый столбец которой совпадает с вектором **c** и последняя строка с вектором **r**. Если последний элемент вектора **c** отличен от первого элемента вектора **r**, то предпочтение отдается последнему элементу вектора **c** .

Примеры:

```
» c=1:4
c =
   1   2   3   4
» r=6:10
r =
   6   7   8   9   10
» H = hankel(c,r)
Warning: Column wins anti-diagonal conflict.
H =
   1   2   3   4   7
   2   3   4   7   8
   3   4   7   8   9
   4   7   8   9   10
```

5.6.5. Вычисление матрицы Гильберта и обратной ей — функции hilb и invhilb

hilb(n) — возвращает матрицу Гильберта порядка n. Матрица Гильберта является примером плохо обусловленной матрицы. Элементы матрицы Гильберта определяются как **H(i,j)=1/(i+j-1)**. Пример:

```
» H = hilb(5)
H =
   1.0000   0.5000   0.3333   0.2500   0.2000
   0.5000   0.3333   0.2500   0.2000   0.1667
   0.3333   0.2500   0.2000   0.1667   0.1429
   0.2500   0.2000   0.1667   0.1429   0.1250
   0.2000   0.1667   0.1429   0.1250   0.1111
» cond(hilb(5))
ans =
  4.7661e+005
```

Значение числа обусловленности матрицы Гильберта указывает на очень плохо обусловленную матрицу.

invhilb(n) — возвращает матрицу, обратную матрице Гильберта порядка n (n<15). Для n>15 функция **invhilb(n)** возвращает приближенную матрицу. Точная обратная матрица — это матрица с большими целочисленными значениями. Эти целочисленные значения могут быть представлены как числа с плавающей точкой без погрешности округления, пока порядок матрицы n не превышает 15. Матрица, обратная матрице Гильберта, — целочисленная матрица. Пример:

```
»H = invhilb(5).
H =
     25    -300     1050    -1400     630
   -300    4800   - 18900    26880  -12600
   1050   18900    79380  -117600   56700
  -1400   26880  -117600   179200  -88200
    630  -12600    56700   -88200   44100
```

а результат обращения с плавающей точкой

```
» inv(hilb(5))
ans =
 1.0e+005 *
  0.0002  -0.0030   0.0105  -0.0140   0.0063
 -0.0030   0.0480  -0.1890   0.2688  -0.1260
  0.0105  -0.1890   0.7938  -1.1760   0.5670
 -0.0140   0.2688  -1.1760   1.7920  -0.8820
  0.0063  -0.1260   0.5670  -0.8820   0.4410
```

5.6.6. Вычисление магического квадрата — функция magic

magic(n) — возвращает матрицу размером n×n, состоящую из целых чисел от 1 до n^2, в которой суммы элементов по строкам, столбцам и главным диагоналям равны одному и тому же числу. Порядок матрицы должен быть больше или равен 3. Пример:

```
» M=magic(4)
M =
  16   2   3  13
   5  11  10   8
   9   7   6  12
   4  14  15   1
```

5.6.7. Вычисление матрицы Паскаля — функция pascal

pascal(n) — возвращает матрицу Паскаля порядка n: симметричную положительно определенную матрицу с целыми входными величинами, взятыми из треугольника Паскаля.

pascal(n,1) — возвращает нижнюю треугольную матрицу в разложении Холецкого для матрицы Паскаля.

pascal(n,2) — возвращает матрицу, полученную в результате транспонирования и перестановок матрицы **pascal(n,1)**, где **A** — кубический корень единичной матрицы.

Примеры:

```
» A=pascal(4)
A =
   1   1   1   1
   1   2   3   4
   1   3   6  10
   1   4  10  20
» A=pascal(4,2)
A =
   0   0   0  -1
   0   0  -1   3
   0   1   2  -3
   1   1   1  -1
```

5.6.8. Вычисление матрицы Теплица — функция toeplitz

toeplitz(c,r) — возвращает несимметричную матрицу Теплица **T**, где **c** — ее первый столбец, а **r** — первая строка. Если первый элемент столбца **c** и первый элемент строки **r** различны, то выдается соответствующее сообщение и отдается предпочтение элементу столбца.

toeplitz(r) — возвращает симметричную матрицу Теплица, определяемую однозначно вектором **r**.

Пример:

```
» c=1:3;
» r=1.5:4.0;
» T = toeplitz(c,r)
Warning: Column wins diagonal conflict.
T =
   1.0000   2.5000   3.5000
   2.0000   1.0000   2.5000
   3.0000   2.0000   1.0000
```

5.6.9. Вычисление матрицы Уилкинсона — функция wilkinson

wilkinson(n) — возвращает одну из тестовых матриц Уилкинсона. Это симметричная трехдиагональная матрица, собственные значения которой попарно близки, но не равны друг другу. Пример:

```
W = wilkinson(5)
W =
```

```
2  1  0  0  0
1  1  1  0  0
0  1  0  1  0
0  0  1  1  1
0  0  0  1  2
```

Данные о множестве других тестовых матриц можно найти в справочной системе MATLAB и в книге [33].

5.7. Матричные функции линейной алгебры

В этом разделе собраны наиболее часто используемые функции линейной алгебры. Их набор весьма представителен и позволяет решать самые серьезные задачи этого важного раздела математики. Относящиеся к рассмотренным понятиям определения даны в книге [38].

5.7.1. Вычисление чисел обусловленности матрицы — функции cond, condeig, rcond

Числа обусловленности матрицы определяют чувствительность решения системы линейных уравнений к погрешностям исходных данных. Следующие функции позволяют найти числа обусловленности матриц.

cond(X) — возвращает число обусловленности, основанное на второй норме, отношение самого большего сингулярного значения **X** к самому малому. Значения **cond(X)** и **cond(X,p)**, близкие к 1, указывают на хорошо обусловленную матрицу.

c = cond(X,p) — возвращает число обусловленности матрицы, основанное на p-норме: **norm(X,p) * norm(inv(X),p)**, где: **p=1** — число обусловленности матрицы, основанное на первой норме; **p=2** — число обусловленности, основанное на 2-норме; **p= «fro»** — на Фробениуса (Frobenius) норме и **p=inf** — на бесконечной норме.

Пример:

```
» d=cond(hilb(4))
d =
  1.5514e+004
```

condeig(A) — возвращает вектор чисел обусловленности для собственных значений **A**. Эти числа обусловленности — обратные величины косинусов углов между левыми и правыми собственными векторами.

[V,D,s] = condeig(A) — эквивалентно **[V,D] = eig(A); s = condeig(A);**.

Большие числа обусловленности означают, что **A** близка к матрице с кратными собственными значениями. Пример:

```
» d=condeig(rand(4))
d =
  1.0766
```

```
1.2298
1.5862
1.7540
```

rcond(A) — оценивает обратную величину обусловленности матрицы **A** первой нормы, используя оценивающий обусловленность метод LINPACK. Если **A** хорошо обусловленная, то **rcond(A)** около 1.00., если плохо обусловленная — то около 0.00. По сравнению с **cond, rcond** реализует более эффективный, но менее достоверный метод оценки обусловленности матрицы. Пример:

```
» s=rcond(hilb(4))
s =
 4.6461e-005
```

5.7.2. Определитель и ранг матрицы — функции det, rank

Для нахождения определителя (детерминанта) и ранга матриц в MATLAB имеются следующие функции:

det(X) — возвращает определитель квадратной матрицы **X**. Если **X** содержит только целые элементы, то результат тоже целое число. Использование **det(X)=0** как тест на вырожденность матрицы действителен только для матрицы малого порядка с целыми элементами. Пример:

```
» A=[2,3,6;1,8,4;3,6,7]
A =
   2   3   6
   1   8   4
   3   6   7
» det(A)
ans =
  -29
```

Детерминант матрицы вычисляется на основе треугольного разложения методом исключения Гаусса:

```
[L,U]=lu(A); s=det(L); d=s*prod(diag(U));
```

Ранг матрицы определяет число сингулярных чисел, превышающих порог **tol=max(size(A))*nprm(A)*eps**. При этом используется следующий алгорим:

```
s=svd(A);tol=max(size(A))*nprm(A)*eps;r=sum(s>tol);
```

Для вычисления ранга используется функция:

rank(A) — возвращает число сингулярных значений, которые являются большими, чем заданный по молчанию допуск.

rank(A,tol) — возвращает число сингулярных значений, которые являются большими, чем **tol.**

Пример:

```
» rank(hilb(11))
ans =
   10
```

5.7.3. Определение векторной и матричной нормы — функция norm

Норма матрицы — скаляр, дающий представление о величине элементов матрицы. Норма функции измеряет несколько различных типов норм матриц. Для вычислений этих характеристик матриц предназначена следующая функция:

norm(A) — возвращает наибольшее сингулярное значение **A, max(svd(A))**.

norm(A,p) — возвращает различные виды норм в зависимости от **p: (p=1,2,inf, «fro»)**.

Пример:

```
» A=[2,3,1;1,9,4;2,6,7]
A =
    2    3    1
    1    9    4
    2    6    7
» norm(A,1)
ans =
   18
```

5.7.4. Определение ортонормального базиса матрицы — функции orth, null

Вычисление ортонормального базиса функции обеспечивает следующие функции:

orth(A) — возвращает ортонормальный базис матрицы **A**. Столбцы **B** определяют то же пространство, что и столбцы матрицы **A**, но столбцы **B** ортогональны, т.е. **B'*B = eye(rank(A))**. Количество столбцов матрицы **B** определяет ранг матрицы **A**. Пример:

```
» A=[2 4 6;9 8 2;12 23 43]
A =
    2    4    6
    9    8    2
   12   23   43
» B=orth(A)
```

```
B =
   0.1453   -0.0414   -0.9885
   0.1522   -0.9863    0.0637
   0.9776    0.1597    0.1371
```

null(A) — возвращает ортонормальный базис для нулевого (пустого) пространства **A**. Пример:

```
» null(hilb(11))
ans =
   0.0000
  -0.0000
   0.0009
  -0.0099
   0.0593
  -0.2101
   0.4606
  -0.6318
   0.5276
  -0.2453
   0.0487
```

5.7.5. Функции приведения матрицы к треугольной форме rref, rrefmovie

В линейной алгебре часто используется приведение матриц к той или иной треугольной форме. Оно реализуется следующими функциями:

rref(A) — осуществляет приведение матрицы к треугольной форме, используя метод исключения Гаусса с частичным выбором ведущего элемента. По умолчанию принимается значение порога допустимости для незначительного элемента столбца, равное **(max(size(A))*eps *norm(A,inf)).**

[R,jb] = rref(A) — также возвращает вектор **jb**, так что **r = length(jb)** может служить оценкой ранга матрицы **A**.

x(jb) — связанные переменные в системе линейных уравнений вида **Ax = b**,

A(:,jb) — базис матрицы **A**.

R(1:r,jb) — единичная матрица размером 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.

rrefmovie(A) — показывает пошаговое исполнение процедуры приведения матрицы к треугольной.

Примеры:

```
» s=magic(3)
```

```
s =
   8   1   6
   3   5   7
   4   9   2
» rref(s)
ans =
   1   0   0
   0   1   0
   0   0   1
```

5.7.6. Определение угла между двумя подпространствами — функция subspace

Угол между двумя подпространствами вычисляет функция:

theta = subspace(A,B) — возвращает угол между двумя подпространствами, натянутыми на столбцы матриц **A** и **B**. Если **A** и **B** — векторы -столбцы единичной длины, то угол вычисляется по формуле **acos(A'*B)**. Если некоторый физический эксперимент описывается массивом **A**, а вторая реализация этого эксперимента — массивом **B**, **subspace(A,B)** измеряет количество новой информации, полученной из второго эксперимента и не связанной со случайными ошибками и флуктуациями. Пример:

```
» H = hadamard(20);A = H(:,2:4);B = H(:,5:8);
» subspace(A,B)
ans =
  1.5708
```

5.7.7. Вычисление следа матрицы — функция trace

След матрицы **A** — это сумма ее диагональных элементов. Он вычисляется функцией:

trace(A) — возвращает след матрицы. Пример:

```
» a=[2,3,4;5,6,7;8,9,1]
a =
   2   3   4
   5   6   7
   8   9   1
» trace(a)
ans =
   9
```

5.7.8. Разложение Холецкого — функции chol, cholinc

Разложение Холецкого — известный прием матричных вычислений. Функция **chol** используется только для диагональных и верхних треугольных матриц.

R = chol(X) — для положительно определенной матрицы **X** возвращает верхнюю треугольную матрицу **R**, так что **R'*R = X**, иначе выдает сообщение об ошибке. Разложение Холецкого возможно для действительных симметричных и комплексных эрмитовых матриц.

[R,p] = chol(X) — с двумя выходными аргументами никогда не генерирует сообщение об ошибке в ходе выполнения разложения Холецкого. Если **X** — положительно определенная матрица, то p=0 и **R** совпадает с вышеописанным случаем, иначе p — положительное целое число и **R** — верхняя треугольная матрица порядка q = p–1 такая, что **R'*R = X(1:q,1:q)**.

Пример:

```
» c=chol(pascal(4))
c =
     1     1     1     1
     0     1     2     3
     0     0     1     3
     0     0     0     1
```

cholinc(X,'0') — возвращает неполное разложение Холецкого для действительной симметричной положительной определенной разреженной матрицы. **cholinc(X,'0')** возвращает верхнюю треугольную матрицу.

R = cholinc(X,'0') — возвращает верхнюю треугольную матрицу, которая имеет такую же разреженную структуру, как и верхний треугольник матрицы **X**. Результат **R'*R** соответствует **X** по своей разреженной структуре. Положительной определенности матрицы **X** недостаточно, чтобы гарантировать существование неполного разложения, и в этом случае выдается сообщение об ошибке.

[R,p] = cholinc(X,'0') — никогда не выдает сообщение об ошибке в ходе разложения. Если **X** — положительно определенная матрица, то p = 0 и матрица **R** — верхняя треугольная, в противном случае p — положительное целое число, **R** — верхняя треугольная матрица размером q×n, где q = p–1. Разреженная структура матрицы **R** такая же, как и у верхнего треугольника размером q×n матрицы **X**, и **R'*R** размером n×n соответствует разреженной структуре матрицы **X** по ее первым q строкам и столбцам **X(1:q,:)** и **X(:,1:q)**.

R = cholinc(X,droptol) — возвращает неполное разложение Холецкого любой разреженной матрицы, используя **droptol. droptol** — неотрицательное число. **cholinc(X,droptol)** возвращает приближение к полному разложению Холецкого, вычисленного с помощью функции **chol(X)**. При меньших значениях **droptol** аппроксимация улучшается, пока значение **droptol** не станет равным 0. В этом случае имеет место полное преобразование Холецкого **(chol(X))**.

R = cholinc(X,options) — определяет структуру с тремя переменными, которые могут быть использованы в любой из комбинаций: **droptol, michol, rdiag**. Дополни-

тельные поля игнорируются. Если **michol** равна 1, **cholinc** возвращает модифицированное разложение Холецкого. Если **rdiag** равна 1, то все нули на диагонали верхней треугольной матрицы заменяются квадратным корнем от произведения **droptol** и нормы соответствующего столбца матрицы **X**, **sqrt(droptol*norm(X(:,j)))**. По умолчанию **rdiag** равна 0.

R = cholinc(X,droptol) и **R = cholinc(X,options)** — возвращают верхнюю треугольную матрицу в **R**. Результат **R'*R** — это аппроксимация матрицы **X**.

Пример:

```
» S = delsq(numgrid('C',4))
S =
   (1,1)      4
   (2,1)     -1
   (1,2)     -1
   (2,2)      4
   (3,2)     -1
   (2,3)     -1
   (3,3)      4
» R0 = cholinc(S,'0')
R0 =
   (1,1)    2.0000
   (1,2)   -0.5000
   (2,2)    1.9365
   (2,3)   -0.5164
   (3,3)    1.9322
```

5.7.9. Обращение матриц — функции inv, pinv

Обращение матриц — одна из наиболее распространенных операций матричного анализа. Обратной называют матрицу, получаемую в результате деления единичной матрицы **E** на исходную матрицу **X**. Таким образом, $\mathbf{X}^{-1}=\mathbf{E}/\mathbf{X}$. Следующие функции обеспечивают реализацию данной операции:

inv(X) — возвращает матрицу, обратную квадратной матрице **X**. Предупреждающее сообщение выдается, если **X** плохо масштабирована или близка к вырожденной.

Пример:

```
» inv(rand(4,4))
ans =
   2.2631   -2.3495   -0.4696   -0.6631
  -0.7620    1.2122    1.7041   -1.2146
  -2.0408    1.4228    1.5538    1.3730
   1.3075   -0.0183   -2.5483    0.6344
```

На практике вычисление явной обратной матрицы не так уж необходимо. Чаще операцию обращения применяют при решении системы линейных уравнений вида **Ax**

= b. Один из путей решения этой системы — вычисление x = inv(A)*b. Но лучшим, с точки зрения минимизации времени расчета и повышения точности вычислений, является использование оператора матричного деления x = A\b. Эта операция использует метод исключения Гаусса без явного формирования обратной матрицы:

B = pinv(A) — возвращает матрицу, псевдообратную матрице **A** (псевдообращение матрицы по Муру-Пенроузу). Результатом псевдообращения матрицы по Муру-Пенроузу является матрица **B** того же размера, что и **A'** , и удовлетворяющая условиям **A*B*A =A** и **B*A*B =B**. Вычисление основано на использовании функции **svd(A)** и приравнивании к нулю всех сингулярных чисел, меньших величины **tol**.

B = pinv(A,tol) — возвращает псевдообратную матрицу и отменяет заданный по умолчанию порог: **max(size(A))*norm(A)*eps**.

Пример:

```
» pinv(rand(4,3))
ans =
  -1.3907  -0.0485  -0.2493   1.8640
  -0.8775   1.1636   0.6605  -0.0034
   2.0906  -0.5921  -0.2749  -0.5987
```

5.7.10. LU- и QR- разложения

Так называемые LU- и QR-разложения реализуются следующими матричными функциями:

Lu -функция выражает любую квадратную матрицу **X** как произведение двух треугольных матриц, одна из них (возможно, с перестановками) — нижняя треугольная матрица, а другая — верхняя треугольная матрица. Иногда эту операцию называют **LR**-разложением. Для выполнения этой операции служат следующие функции:

[L,U] = lu(X) — возвращает верхнюю треугольную матрицу **U** и нижнюю треугольную матрицу (т.е. произведение нижней треугольной матрицы и матрицы перестановок) **L** так, что **X = L*U**.

[L,U,P] = lu(X) — возвращает верхнюю треугольную матрицу **U**, нижнюю треугольную матрицу **L** и матрицу перестановок **P** так, что **L*U = P*X**.

lu(X) — возвращает результат из LINPACK (известный пакет программ линейной алгебры).

Пример:

```
» f=[3,5,4;12,7,5;34,65,23]
f =
   3   5   4
  12   7   5
  34  65  23
» [d,h]=lu(f)
d =
   0.0882   0.0461   1.0000
   0.3529   1.0000        0
```

```
    1.0000         0         0
h =
   34.0000   65.0000   23.0000
         0   -15.9412   -3.1176
         0         0    2.1144
» d*h
ans =
    3.0000    5.0000    4.0000
   12.0000    7.0000    5.0000
   34.0000   65.0000   23.0000
```

Функция **luinc** возвращает единичную нижнюю треугольную матрицу, верхнюю треугольную матрицу и матрицу перестановок. Используется в следующих формах:

luinc(X,'0') — возвращает неполное **LU**-разложение по уровню 0 квадратной разреженной матрицы. Треугольные коэффициенты имеют такую же разреженную структуру, как и перестановка квадратной матрицы **X**, и их результат соответствует перестановке матрицы **X** над их разреженными структурами. **luinc(X,'0')** возвращает нижнюю треугольную часть коэффициентов и верхние треугольные коэффициенты, вставленные в ту же самую матрицу.

[L,U] = luinc(X,'0') — возвращает результат перестановки матрицы и единичную треугольную матрицу в **L** и верхнюю треугольную матрицу в **U**. Разреженности матриц **L**, **U** и **X** — не сравнимы, но число ненулевых элементов поддерживается равным с возможностью исключения некоторых нулей в **L** и **U** из-за сокращения: **nnz(L)+nnz(U) = nnz(X)+n**, где **X** — матрица размером n×n.

[L,U,P]=luinc(X,'0') — возвращает единичную нижнюю треугольную матрицу в **L**, верхнюю треугольную матрицу — в **U** и матрицу перестановок — в **P**. **L** имеет такую же разреженную структуру, как нижняя треугольная часть перестановленной матрицы **X — spones(L) = spones(tril(P*X))** с возможными исключениями единиц на диагонали матрицы **L**, где **P*X** может быть 0.

luinc(X,droptol) — возвращает неполное **LU**- разложение любой разреженной матрицы, используя **droptol**. **droptol** должен быть неотрицательным числом.

luinc(X,droptol) — возвращает приближение к полному **LU**- разложению, полученному с помощью функции **lu(X)**. При меньших значениях **droptol** аппроксимация улучшается, пока значение **droptol** не станет равным 0. В этом случае имеет место полное **LU**- разложение.

luinc(X,options) — определяет структуру с четырьмя переменными, которые могут быть использованы в любой из комбинаций: **droptol, milu, udiag, thresh**. Дополнительные поля игнорируются. Если **milu** равна 1, функция luinc возвращает модифицированное неполное **LU**- разложение. Если **udiag** равна1, то все нули на диагонали верхней треугольной части заменяются на локальную ошибку.

luinc(X,options) — то же самое, что и **luinc(X,droptol)**, если **options** имеет только параметр **droptol**.

[L,U] = luinc(X,options) — возвращает перестановку единичной треугольной матрицы в **L** и верхнюю треугольную матрицу в **U**. Результат **L*U** аппроксимирует **X**.

[L,U,P] = luinc(X,options) — возвращает единичную нижнюю треугольную матрицу в **L**, верхнюю треугольную матрицу в **U** и матрицу перестановок в **P**. Ненулевые входные элементы матрицы **U** удовлетворяют выражению **abs(U(i,j)) >= droptol*norm((X:,j))** с возможным исключением диагональных входов, которые были сохранены, несмотря на то, что они не удовлетворяли критерию.

[L,U,P] = luinc(X,options) — то же самое что и **[L,U,P] = luinc(X,droptol),** если **options** имеет только параметр **droptol.**

Функция **qr** выполняет **QR**-разложение матрицы. Эта операция полезна для квадратной и треугольной матриц. Она выражает матрицу как произведение действительной ортонормальной или комплексной унитарной матрицы и верхней треугольной матрицы. Функция используется в следующих формах:

[Q,R] = qr(X) — вычисляет верхнюю треугольную матрицу **R** того же размера, как и у **X**, и унитарную матрицу **Q** так, что **X =Q*R**.

[Q,R,E] = qr(X) — вычисляет матрицу перестановок, верхнюю треугольную матрицу **R** с убывающими по модулю диагональными элементами и унитарную матрицу **Q** так, что **X*E =Q*R**. Матрица **E** выбрана так, что **abs(diag(R))** уменьшается.

[Q,R] = qr(X,0) и **[Q,R,E] = qr(X,0)** — вычисляют экономное разложение, в котором **E** — вектор перестановок, так, что **Q*R = X(:,E).** Матрица **E** выбрана так, что **abs(diag(R))** уменьшается.

A = qr(X) — возвращает результат из LINPACK.

Пример:

```
» C=rand(5,4)
C =
   0.8381   0.5028   0.1934   0.6979
   0.0196   0.7095   0.6822   0.3784
   0.6813   0.4289   0.3028   0.8600
   0.3795   0.3046   0.5417   0.8537
   0.8318   0.1897   0.1509   0.5936
» [Q,R]=qr(C)
Q =
  -0.5922  -0.1114   0.5197   0.0743  -0.6011
  -0.0139  -0.9278  -0.0011  -0.3448   0.1420
  -0.4814  -0.1173   0.0699   0.5940   0.6299
  -0.2681  -0.1525  -0.8268   0.2632  -0.3898
  -0.5877   0.2997  -0.2036  -0.6734   0.2643
R =
  -1.4152  -0.7072  -0.5037  -1.4103
        0  -0.7541  -0.7274  -0.4819
        0        0  -0.3577  -0.4043
        0        0        0   0.2573
        0        0        0        0
```

5.7.11. Вычисление собственных значений и сингулярных чисел — функции balance, eig, svd

Во многих областях математики и прикладных наук важными являются средства для вычисления собственных значений матриц, принадлежащих им векторов и сингулярных чисел. Ниже дана подборка этих средств, реализованная в системе MATLAB.

[D,B] = balance(A) — возвращает диагональную матрицу **D**, элементы которой являются степенями основания 2, и масштабированную матрицу **B** таким образом, что **B = D\A*D**.

B = balance(A) — возвращает масштабированную матрицу **B**.

Несимметрические матрицы могут быть плохо обусловлены при вычислении их собственных значений. Малые изменения элементов матрицы, такие как ошибки округления, могут вызвать большие изменения в собственных значениях.

Величина, связывающая погрешность вычисления собственных значений с погрешностью исходных данных, называется числом обусловленности собственных значений матрицы и вычисляется следующим образом:

cond(V) = norm(V)*norm(inv(V)),
где **[V,D] = eig(A)**.

Масштабирование – это попытка перевести каждую плохую обусловленность матрицы собственных векторов в диагональное масштабирование. Однако масштабирование обычно не может преобразовать несимметричную матрицу в симметричную, а только пытается сделать норму каждой строки равной норме соответствующего столбца.

Функция вычисления собственных значений MATLAB **eig(A)** автоматически масштабирует матрицу **A** перед вычислением ее собственных значений. Масштабирование можно отключить, используя команду в следующем виде: **eig(A,'nobalance')**. Примеры:

```
» A=[1 1000 10000;0.0001 1 1000;0.000001 0.0001 1]
A =
 1.0e+004 *
   0.0001   0.1000   1.0000
   0.0000   0.0001   0.1000
   0.0000   0.0000   0.0001
» [F,G]=balance(A)
F =
 1.0e+004 *
   3.2768        0        0
        0   0.0032        0
        0        0   0.0000
```

```
G =
   1.0000   0.9766   0.0095
   0.1024   1.0000   0.9766
   1.0486   0.1024   1.0000
```

d = eig(A) — возвращает вектор собственных значений матрицы **A**.

[V,D] = eig(A) — вычисляет матрицу собственных значений **(D)** и собственных векторов (**V**) матрицы **A** таким образом, что **A*V =V*D**. Матрица **D** — каноническая форма матрицы **A**, которая является диагональной. Столбцы матрицы **V** — собственные векторы матрицы **A**.

Собственные векторы масштабированы, так, что норма каждого вектора около 1.0. Нужно использовать **[W,D] = eig(A'); W = W'**, чтобы вычислить левые собственные векторы, которые соответствуют **W*A =D*W**.

[V,D] = eig(A,'nobalance') — находит собственные векторы и собственные значения без предварительного масштабирования. Обычно улучшает обусловленность входной матрицы, обеспечивая большую точность вычисления собственных значений и собственных векторов.

d = eig(A,B) — возвращает вектор, содержащий обобщенные собственные значения, если **A** и **B** — квадратные матрицы.

[V,D] = eig(A,B) — возвращает диагональную матрицу собственных значений **D** и матрицу **V**, чьи столбцы являются соответствующими собственными векторами так, чтобы **A*V = B*V*D**. Собственные векторы масштабированы так, чтобы норма каждого была 1.0.

Пример:

```
» B = [3 -12 -.6 2*eps;-2 48 -1 -eps;-eps/8 eps/2 -1 10;-.5 -.5 .3 1]
B =
   3.0000  -12.0000   -0.6000    0.0000
  -2.0000   48.0000   -1.0000   -0.0000
  -0.0000    0.0000   -1.0000   10.0000
  -0.5000   -0.5000    0.3000    1.0000
» [G,H]=eig(B)
G =
  -0.2548    0.7420   -0.4842    0.1956
   0.9670    0.0193   -0.0388    0.0276
  -0.0015   -0.6181   -0.8575    0.9780
  -0.0075   -0.2588   -0.1694   -0.0676
H =
  48.5287         0         0         0
        0    3.1873         0         0
        0         0    0.9750         0
        0         0         0   -1.6909
```

s = svd(X) — возвращает вектор сингулярных чисел. Команда **svd** выполняет сингулярное разложение матрицы **X**.

[U,S,V] = svd(X) — вычисляет диагональную матрицу **S** тех же размеров, которые имеет матрица **X** с неотрицательными диагональными элементами в порядке их убывания, и унитарные матрицы **U** и **V** так, что **X = U*S*V'**.

[U,S,V] = svd(X,0) — выполняет экономичное сингулярное разложение.

Пример:

```
» F=[23 12;3 5;6 0]
F =
   23   12
    3    5
    6    0
» [k,l,m]=svd(F)
k =
   0.9628  -0.0034  -0.2702
   0.1846   0.7385   0.6485
   0.1974  -0.6743   0.7116
l =
  26.9448        0
      0         4.1202
      0         0
m =
   0.8863  -0.4630
   0.4630   0.8863
```

5.7.12. Функции приведения матриц к форме Шура и Хессенберга cdf2rdf, hess, qz, rsf2csf, schur

Ниже приводятся функции, обеспечивающие приведение матриц к специальным формам Шура и Хессенберга:

cdf2rdf — преобразование комплексной формы Шура в действительную. Если система **[V,D] = eig(X)** имеет комплексные собственные значения, объединенные в комплексно-сопряженные пары, то функция **cdf2rdf** преобразует систему таким образом, что матрица **D** становится вещественного, диагонального вида с 2 ×2 вещественными блоками, заменяющими первоначальные комплексные пары. Собственные векторы преобразованные таким образом, что **X =V*D/V**, продолжают сохраняться. Конкретные столбцы матрицы **V** больше не собственные векторы, но каждая пара векторов связана с блоком размера 2 ×2 в матрице **D**.

Пример:

```
» A=[2 3 6;-4 0 3;1 5 -2]
A =
   2   3   6
  -4   0   3
   1   5  -2
» [S,D]=eig(A)
```

```
S =
  0.7081 + 0.3296i  0.7081 - 0.3296i  -0.3355
 -0.3456 + 0.3688i -0.3456 - 0.3688i  -0.5721
  0.0837 + 0.3571i  0.0837 - 0.3571i   0.7484
D =
  3.1351 + 4.0603i        0                    0
       0                  3.1351 - 4.0603i     0
       0                  0                   -6.2702
» [S,D]=cdf2rdf(S,D)
S =
   0.7081   0.3296  -0.3355
  -0.3456   0.3688  -0.5721
   0.0837   0.3571   0.7484
D =
   3.1351   4.0603        0
  -4.0603   3.1351        0
       0        0       -6.2702
```

H = hess(A) — находит **H**, верхнюю форму Хессенберга для матрицы **A**.

[P,H] = hess(A) — возвращает матрицу Хессенберга **H** и унитарную матрицу преобразований **P** такие, что **A =P*H*P'** и **P'*P = eye(size(A))**.

Элементы матрицы Хессенберга, расположенные ниже первой поддиагонали, равны 0. Если матрица симметричная или эрмитова, то матрица Хессенберга вырождается в трехдиагональную. Эта матрица имеет те же самые собственные значения, как и оригинал, но необходимо меньшее количество операций для их вычисления.

Пример:

```
» f=magic(4)
f =
   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1
» hess(f)
ans =
  16.0000   -8.0577    8.8958    6.1595
 -11.0454   24.2131   -8.1984    2.1241
       0   -13.5058   -4.3894   -7.8918
       0         0    -3.2744   -1.8237
```

Функция **qz** обеспечивает приведение пары матриц к обобщенной форме Шура:

[AA,BB,Q,Z,V] = qz(A,B) — возвращает комплексные верхние треугольные матрицы **AA** и **BB**, соответствующие матрицам приведения **Q** и **Z** , такие что **Q*A*Z =AA** и **Q * B * Z =BB**

Функция **Qz** также возвращает матрицу обобщенных собственных векторов **V**.

Обобщенные собственные значения могут быть найдены из следующего условия:

A*V*diag(BB) = B*V*diag(AA)

Пример:

```
» A=[1 2 3;6 3 0;4 7 0];B=[1 1 1;0 7 4;9 4 1];
» [aa,bb,f,g,h]=qz(A,B)
aa =
   -2.9395    0.4775    0.8751
         0   -9.5462   -3.5985
    0.0000   -0.0000    3.2073
bb =
    5.5356    3.5345   -2.2935
         0   -8.4826   -6.7128
         0         0    0.7667
f =
    0.0367   -0.7327    0.6796
   -0.1052   -0.6791   -0.7265
    0.9938   -0.0448   -0.1020
g =
    0.7023    0.7050    0.0989
   -0.6867    0.6343    0.3552
    0.1877   -0.3174    0.9295
h =
    0.7023    0.4082    0.0476
   -0.6867    0.8375   -0.5287
    0.1877   -0.3634    0.8475
```

[U,T] = rsf2csf(U,T) — преобразование действительной формы Шура в комплексную форму Шура. Комплексная форма Шура — это верхняя треугольная матрица с собственными значениями на диагонали. Действительная форма Шура имеет действительные собственные значения на диагонали и комплексные собственные значения в 2 ×2 блоках, занимающих нижнюю поддиагональ. Матрицы **U** и **T** представляют собой унитарную матрицу и матрицу Шура исходной матрицы **A**, которая удовлетворяет условиям: **A = U * T * U'** и **U' * U = eye(size(A))**.

[U,T] = schur(A) — возвращает матрицу Шура **T** и унитарную матрицу **U** такую, что

A = U * T * U' и U' * U = eye(size(A)).

T = schur(A) — возвращает только матрицу Шура **T**.

Примеры:

```
A =
     1     1     1     1
    -3     1    -4     1
     1     0    -5     1
    -1     2     3     0
» [u,t] = schur(A)
u =
```

```
   -0.4883   -0.6416   -0.5757    0.1362
   -0.5289    0.7465   -0.3986   -0.0646
   -0.1403   -0.1528    0.0583   -0.9765
   -0.6798   -0.0884    0.7115    0.1540
t =
    1.2036   -2.7670   -0.8023   -0.0842
    1.9478    2.3183    1.5080    2.6513
         0         0   -0.6449   -2.9694
         0         0    0.0000   -5.8771
» [U,T] = rsf2csf(u,t)
U =
  -0.3226 - 0.3631i   0.4318 + 0.4771i  -0.5757          0.1362
   0.5771 - 0.3933i   0.2027 - 0.5551i  -0.3986         -0.0646
  -0.0724 - 0.1044i   0.1183 + 0.1136i   0.0583         -0.9765
   0.0682 - 0.5056i   0.4532 + 0.0657i   0.7115          0.1540
T =
   1.7610 + 2.2536i   0.5003 - 1.2897i   1.1168 + 0.5967i   1.7196 + 0.0626i
         0            1.7610 - 2.2536i   0.2383 + 1.1215i  -0.4335 + 1.9717i
         0                  0           -0.6449           -2.9694
         0                  0            0                -5.8771
```

5.7.13. Матричные функции

Весьма представителен в MATLAB набор матричных функций. Они описаны ниже.

expm(X) — возвращает e^X от матрицы **X**. Комплексный результат получается, если **X** имеет неположительные собственные значения. **expm** является встроенной функцией, но использует разложение Паде и располагается в файле **expm1.m**. Второй метод вычисления матричной экспоненты использует разложение Тейлора и находится в файле **expm2.m**. Метод Тейлора не рекомендуется применять как основной, т. к. он зачастую бывает относительно медленный и неточный. Реализация третьего способа вычисления матричной экспоненты находится в файле **expm3.m** и использует спектральное разложение матрицы **A**. Этот метод неудачен, если входная матрица не имеет полного набора линейно независимых собственных векторов. Пример:

```
» S=[1,0,3;1,3,1;4,0,0]
S =
   1   0   3
   1   3   1
   4   0   0
» a=expm(S)
a =
  31.2203        0   23.3779
  38.9659  20.0855   30.0593
  31.1705        0   23.4277
```

funm(X,'function') — позволяет вычислить любую функцию от матрицы, если она имеет имя, составленное из латинских букв. Матрица **X** должна быть квадратной. Команды **funm(X,'sqrt')** и **funm(X,'log')** эквивалентны командам **sqrtm(X)** и **logm(X).** Команды **funm(X,'exp')** и **expm(X)** вычисляют одинаковую функцию, но используя разные алгоритмы. Однако предпочтительнее использовать **expm(X)**.

[Y,esterr] = funm(X,'function') — не выдает никакого сообщения, но возвращает грубую оценку относительной погрешности результата. Если матрица **X** действительная симметричная или комплексная эрмитова, то ее форма Шура диагональна, и полученный результат может иметь высокую точность.

Примеры:

```
» S=[1,0,3;1,3,1;4,0,0]
S =
   1   0   3
   1   3   1
   4   0   0
» a=funm(S,'exp')
a =
  31.2203   0.0000  23.3779
  38.9659  20.0855  30.0593
  31.1705  -0.0000  23.4277
```

logm(X) — возвращает логарифм матрицы: функцию, обратную **expm(X).** Результат получается комплексным, если **X** имеет отрицательные собственные значения.

[Y,esterr]=logm(X) — не выдает какого-либо предупреждающего сообщения, но возвращает оценку погрешности в виде относительной невязки **norm(expm(Y)–X)/norm(X)**.

Если матрица **X** действительная симметричная или комплексная эрмитова, то теми же свойствами обладает и **logm(X)**. Пример:

```
a =
  31.2203   0.0000  23.3779
  38.9659  20.0855  30.0593
  31.1705  -0.0000  23.4277
» logm(a)
ans =
   1.0000   0.0000   3.0000
   1.0000   3.0000   1.0000
   4.0000  -0.0000  -0.0000
```

sqrtm(X) — возвращает квадратный корень **X**. Результат получается комплексным, если **X** имеет отрицательные собственные значения.

[Y,esterr]=sqrtm(X) — не выдает какого-либо предупреждающего сообщения, но возвращает оценку погрешности в виде относительной невязки **norm(Y*Y–X)/norm(X)**.

Пример:

```
» S=[2,1,0;6,7,-2;3,4,0];
```

```
» e=sqrtm(S)
e =
   1.2586   0.2334   0.0688
   1.6066   2.7006  -0.6043
   0.5969   1.1055   0.7918
```

5.7.14. Матричные функции низкого уровня

Наряду с отмеченными функциями MATLAB обладает рядом функций более низкого уровня. Рассмотрим такие функции:

[Q,R] = qrdelete(Q,R,j) — изменяет **Q** и **R** таким образом, чтобы пересчитать разложение матрицы **A** для случая, когда в ней удален j-тый столбец **A(:,j)**. Входные значения **Q** и **R** представляют **QR**-разложение матрицы **A**, как результат действия **[Q,R] = qr(A)**. Аргумент j определяет столбец, который должен быть удален из матрицы **A**. Примеры:

```
» C=rand(3,3)
C =
   0.0164   0.0576   0.7176
   0.1901   0.3676   0.6927
   0.5869   0.6315   0.0841
» [Q,R]=qr(C)
Q =
  -0.0265  -0.2416  -0.9700
  -0.3080  -0.9212   0.2378
  -0.9510   0.3051  -0.0500
R =
  -0.6171  -0.7153  -0.3123
        0  -0.1599  -0.7858
        0        0  -0.5356
» [Q1,R1]=qrdelete(Q,R,2)
Q1 =
  -0.0265   0.7459   0.6655
  -0.3080   0.6272  -0.7153
  -0.9510  -0.2239   0.2131
R1 =
  -0.6171  -0.3123
        0   0.9510
        0        0
```

[Q,R] = qrinsert(Q,R,j,x) — изменяет **Q** и **R** таким образом, чтобы пересчитать разложение матрицы **A** для случая, когда в матрице **A** вставлен столбец **x** перед **A(:,j)**. Входные значения **Q** и **R** представляют **QR**-разложение матрицы **A** как результат действия **[Q,R] = qr(A)**. Аргумент **x** — вектор-столбец, который нужно вставить в матрицу **A**. Аргумент j определяет столбец, перед которым будет вставлен вектор **x**.

Примеры:

```
» C=rand(3,3)
C =
   0.1210   0.8928   0.8656
   0.4508   0.2731   0.2324
   0.7159   0.2548   0.8049
» [Q,R]=qr(C)
Q =
  -0.1416   0.9835   0.1126
  -0.5275   0.0213  -0.8493
  -0.8377  -0.1797   0.5157
R =
  -0.8546  -0.4839  -0.9194
        0   0.8381   0.7116
        0        0   0.3152
» x=[0.5,-0.3,0.2];[Q2,R2]=qrinsert(Q,R,2,x')
Q2 =
  -0.1416   0.7995  -0.5838
  -0.5275  -0.5600  -0.6389
  -0.8377   0.2174   0.5010
R2 =
  -0.8546  -0.0801  -0.4839  -0.9194
        0   0.6112   0.6163   0.7369
        0        0  -0.5681  -0.2505
```

5.8. Функции разреженных матриц

Матрицы, содержащие большое число элементов с нулевыми значениями, принято именовать разреженными матрицами. Они широко используются при решении прикладных задач. Например, моделирование электронных и электротехнических линейных цепей часто приводит к появлению в матричном описании топологии схем сильно разреженных матриц. Для таких матриц создан ряд функций, обеспечивающих эффективную работу с ними и устраняющих тривиальные операции с нулевыми элементами матриц.

5.8.1. Элементарные разреженные матрицы — функции spdiags, speye, sprand, sprandn, sprandsym

Вначале рассмотрим элементарные разреженные матрицы и относящиеся к ним функции системы MATLAB.

Функция **spdiags** расширяет возможности встроенной функции **diag**. Допустимы четыре операции, различаемые числом входных аргументов:

[B,d] = spdiags(A) — извлекает все ненулевые диагонали из матрицы **A** размера m×n. **B** — матрица размера **min(m,n)*p**, столбцы которой **p** являются ненулевыми

диагоналями **A**. **d** — вектор длины **p**, целочисленные элементы которого точно определяют номера диагоналей матрицы **A**.

B = spdiags(A,d) — извлекает диагонали, определенные вектором **d**.

A = spdiags(B,d,A) — заменяет диагонали матрицы **A**, определенные вектором **d** со столбцами матрицы **B**.

A = spdiags(B,d,m,n) — создает разреженную матрицу размера m×n, размещая соответствующие столбцы матрицы **B** вдоль диагоналей, определяемых вектором **d**.

Пример:

```
» A=[1 3 4 6 8 0 0;7 8 0 7 0 0 5; 0 0 0 0 0 9 8 ; 7 6 54 32 0 9 6];
» d=[1 3 2 2]
» B = spdiags(A,d)
B =
   3   6   4   4
   0   0   7   7
   0   9   0   0
   0   6   9   9
```

S = speye(m,n) — формирует разреженную матрицу размера m×n с единицами а главной диагонали и нулевыми недиагональными элементами.

S = speye(n) — равносильна **speye(n,n)**.

Пример:

```
» S = speye(4)
S =
  (1,1)      1
  (2,2)      1
  (3,3)      1
  (4,4)      1
```

Матрица **R = sprand(S)** имеет ту же структуру, что и разреженная матрица **S**, но се элементы распределены по равномерному закону.

R = sprand(m,n,density) — возвращает случайную разреженную матрицу размера m×n, которая имеет приблизительно density×m×n равномерно распределенных ненулевых элементов ($0 \le density \le 1$).

R = sprand(m,n,density,rc) — в дополнение к этому имеет в числе параметров число обусловленности по отношению к операции обращения, приблизительно равное **rc**. Если вектор **rc** имеет длину lr ($lr \le min(m,n)$), то матрица **R** имеет rc в качестве своих первых lr сингулярных значений, все другие значения равны нулю. В этом случае матрица **R** генерируется с помощью матриц случайных плоских вращений, которые применяются к диагональной матрице с заданными сингулярными числами. Такие матрицы играют важную роль при анализе алгебраических и топологических структур.

Пример:

```
» d=sprand(4,3,0.6)
d =
  (1,1)      0.6614
```

```
(2,1)    0.2844
(4,1)    0.0648
(3,3)    0.4692
(4,3)    0.9883
```

R = sprandn(S) — возвращает матрицу со структурой разреженной матрицы **S**, но с элементами, распределенными по нормальному закону со средним, равным 0, и дисперсией 1.

R = sprandn(m,n,density) — возвращает случайную разреженную матрицу размера m×n , имеющую примерно density×m×n нормально распределенных ненулевых элементов ($0 \le$ density ≤ 1).

R = sprandn(m,n,density,rc) — в дополнение к этому имеет своим параметром число обусловленности по отношению к операции обращения, приблизительно равное **rc**. Если вектор **rc** имеет длину lr (lr ≤ **min(m,n)**), то матрица **R** имеет **rc** в качестве своих первых lr сингулярных значений, все другие значения равны нулю. В этом случае матрица **R** генерируется с помощью матриц случайных плоских вращений, которые применяются к диагональной матрице с заданными сингулярными числами. Такие матрицы играют важную роль при анализе алгебраических и топологических структур.

Пример:

```
» f=sprandn(3,4,0.3)
f =
   (2,1)   -0.4326
   (2,2)   -1.6656
   (2,3)    0.1253
   (2,4)    0.2877
```

sprandsym(S) — возвращает случайную симметричную матрицу, нижние поддиагонали и главная диагональ которой имеет ту же структуру, как и матрица **S**. Ее элементы распределены по нормальному закону со средним, равным 0, и дисперсией 1.

sprandsym(n,density) — возвращает симметричную случайную разреженную матрицу размера n×n, которая имеет приблизительно density×n×n ненулевых элементов; каждый элемент сформирован в виде суммы нормально распределенных случайных чисел ($0 \le$ density ≤ 1).

R = sprandsym(n,density,rc) — возвращает матрицу с числом обусловленности по отношению к операции обращения, равным **rc**. Закон распределения не является равномерным; значения случайных элементов симметричны относительно 0 и находятся в пределах [-1 1]. Если **rc** — вектор длины n, то матрица **R** имеет собственные значения, равные элементам вектора **rc**. Таким образом, если элементы вектора **rc** положительны, то матрица **R** является положительно определенной. В любом случае матрица **R** генерируется с помощью матриц случайных плоских вращений (матриц Якоби), которые применяются к диагональным матрицам с заданными собственными значениями и числом обусловленности. Такие матрицы играют важную роль при анализе алгебраических и топологических структур.

R = sprandsym(n,density,rc,kind) — возвращает положительно определенную матрицу. Аргумент kind может быть:

kind=1 — чтобы генерировать матрицу **R** из положительно определенной диагональной матрицы с помощью случайных вращений Якоби. **R** имеет точно заданное число обусловленности.

kind=2 — чтобы генерировать матрицу **R** как смещенную сумму матриц внешних произведений. Число обусловленности матрицы приблизительно, но структура более компактна (по сравнению с предыдущим случаем).

kind=3 — чтобы генерировать матрицу **R** той же структуры, что и **S**, и число обусловленности приближенно равно 1/rc. Значение density игнорируется.

Пример:

```
» a=sprandsym(4,0.3,0.8)
a =
   (1,1)      0.9818
   (3,1)      0.0468
   (2,2)     -0.9283
   (1,3)      0.0468
   (3,3)      0.8800
   (4,4)     -0.8000
```

5.8.2. Преобразование разреженных матриц — функции find, full, sparse, spconvert

Теперь рассмотрим функции преобразования разреженных матриц. Они представлены ниже:

k = find(X) — возвращает индексы массива **x** для его ненулевых элементов. Если таких элементов нет, то **find** возвращает пустую матрицу.

[i,j] = find(X) — возвращает индексы строки и столбца для ненулевого элемента матрицы **X**.

[i,j,v] = find(X) — возвращает вектор-столбец **v** ненулевых элементов матрицы **X** и индексы строки i и столбца j.

Пример:

```
» q=sprand(3,4,0.6)
q =
   (1,1)      0.7266
   (1,2)      0.4120
   (3,2)      0.2679
   (3,3)      0.4399
   (2,4)      0.7446
   (3,4)      0.9334
» [i,j]=find(q)
```

```
i =       j =
   1         1
   1         2
   3         2
   3         3
   2         4
   3
```

full(S) — преобразует разреженную матрицу **S** в полную; если исходная матрица **S** была полной, то **full(S)** возвращает **S**. Пусть **X** матрица размера m×n с **nz = nnz(X)** ненулевыми элементами. Тогда **full(X)** требует пространство, чтобы хранить m×n действительных чисел, в то время как **sparse(X)** требует пространство для хранения nz действительных чисел и (n×z+n) целых. Большинству компьютеров для хранения действительного числа требуется вдвое больше пространства , чем для целого. Для таких компьютеров **sparse(X)** требует меньше пространства, чем **full(X)**, если плотность **nnz/prod(size(X)) < 2/3**. Выполнение операций над разреженными матрицами, однако, требует больше затрат времени, чем над полными, поэтому плотность должна быть много меньше 2/3 для эффективной работы с разреженными матрицами.

Примеры:

```
» q=sprand(3,4,0.6)
q =
  (1,1)     0.0129
  (1,2)     0.3840
  (2,2)     0.6831
  (3,3)     0.0928
» d=full(q)
d =
    0.0129    0.3840         0         0
         0    0.6831         0         0
         0         0    0.0928         0
```

S=sparse(A) — преобразует полную матрицу в разреженную, удаляя нулевые элементы. Если матрица **S** разреженная, то **sparse(S)** возвращает **S**. Функция **sparse** — это встроенная функция, которая формирует матрицы в соответствии с правилами записи разреженных матриц , принятыми в системе MATLAB.

S=sparse(i,j,s,m,n,nzmax) — использует векторы **i, j,** и **s** для того, чтобы генерировать разреженную матрицу размера m×n с ненулевыми элементами, количество которых не превышает **nzmax.** Векторы **i** и **j** задают позиции элементов и являются целочисленными, а вектор **s** определяет числовое значение элемента матрицы, которое может быть действительным или комплексным. Все элементы вектора **s**, равные нулю, игнорируются вместе с соответствующими значениями **i** и **j**. Векторы **i, j** и **s** должны быть одной и той же длины.

S = sparse(i,j,s,m,n) — использует nzmax = **length(s)**.

S = sparse(i,j,s) — использует m = **max(i)** и n = **max(j)**. Максимумы вычисляются раньше, чем нулевые строки столбца **S** будут удалены.

S = sparse(m,n) равносильно **sparse([ ],[ ],[ ],m,n,0)**. Эта команда генерирует предельную разреженную матрицу, где m×n элементов нулевые.

Все встроенные в MATLAB арифметические, логические и индексные операции могут быть применены и к разреженным матрицам или к смешанным разреженным и полным матрицам. Операции над разреженными матрицами возвращают разреженные матрицы, и операции над полными матрицами возвращают полные матрицы. В большинстве случаев операции над смешанными матрицами возвращают полные матрицы. Исключение составляют случаи, когда результат смешанной операции явно сохраняет разреженный тип. Так бывает при поэлементном умножении **A*S**, где **S** — разреженный массив. Пример:

```
» i=[2,4,3];j=[1,3,8];s=[4,5+5i,9];t = sparse(i,j,s,5,8)
t =
   (2,1)     4.0000
   (4,3)     5.0000 + 5.0000i
   (3,8)     9.0000
```

Функция **spconvert** используется для создания разреженных матриц из простых разреженных форматов, легко производимых вне средств MATLAB:

S = spconvert(D) — преобразует матрицу **D** со строками, содержащими **[i,j,s]** или **[i,j,r,s]**, в соответствующую разреженную матрицу. **D** должна иметь nnz или nnz+1 строк и три или четыре столбца.Три элемента в строке генерируют действительную матрицу, четыре элемента в строке генерируют комплексную матрицу. Последняя строка массива типа **[m n 0]** или **[m n 0 0]** в **D** может быть использована для определения **size(S)**. Если **D** имеет разреженную структуру, то никаких преобразований не требуется, так как команда **spconvert** может быть использована после того, как матрица **D** загружена или из **MAT-файла**, или из **ASCII-файла**.

5.8.3. Работа с ненулевыми элементами разреженных матриц — функции nnz, nonzeros, nzmax, spalloc, spfun, spones

Поскольку разреженные матрицы содержат ненулевые элементы, то предусмотрен ряд функций для работы с ними:

nnz(X) — возвращает число ненулевых элементов матрицы **X**. Плотность разреженной матрицы определяется по формуле **nnz(X)/prod(size(X))**.

Пример:

```
h = sparse(hilb(10));
» nnz(h)
```

```
ans =
  100
```

nonzeros(A) — возвращает полный столбец-вектор ненулевых элементов матрицы **A**, выбирая их по столбцам. Эта функция дает только выход **s**, но не значения **i** и **j** из аналогичного выражения **[i,j,s] = find(A)**. Вообще, **length(s) = nnz(A) J nzmax(A) J prod(size(A))**. Пример:

```
» g=nonzeros(sparse(hankel([1,2,8])))
g =
   1
   2
   8
   2
   8
   8
```

nzmax(S) — возвращает количество ячеек памяти для ненулевых элементов. Обычно функции **nnz(S)** и **nzmax(S)** дают один и тот же результат. Но если **S** создавалась в результате операции над разреженными матрицами, такой как умножение или LU-разложение, может быть выделено больше элементов памяти, чем требуется, и **nzmax(S)** отражает это. Если **S** — разреженная матрица, то **nzmax(S)** — максимальное количество ячеек для хранения ненулевых элементов. Если **S** — полная матрица, то **nzmax(S) = prod(size(S))**.

Пример:

```
» q=nzmax(sparse(hankel([1,7,23])))
q =
   6
» q=nzmax(hankel([1,7,23]))
q =
   9
```

S=spalloc(m,n,nzmax) — создает массив для разреженной матрицы **S** размера m×n с пространством для размещения **nzmax** ненулевых элементов. Затем матрица может быть заполнена по столбцам.

spalloc(m,n,nzmax) — повторяет функцию **sparse([ ],[ ],[ ],m,n,nzmax)**.

Пример:

```
» S = spalloc(5,4,5);
```

spfun — вычисление функции для ненулевых элементов. Функция **spfun** применяется выборочно только к ненулевым элементам разреженной матрицы, сохраняя при этом разреженность исходной матрицы.

f = spfun('function',S) — вычисляет **function(S)** для ненулевых элементов матрицы **S**. Имя function — это имя M-файла, который может работать с матричным аргументом **S** и вычислить функцию для каждого элемента матрицы **S**.

Пример:

```
» S=spfun('exp',sprand(4,5,0.4))
S =
  (2,2)     1.6864
  (2,3)     2.4112
  (3,3)     2.6638
  (2,4)     1.1888
  (3,4)     1.3119
  (4,4)     2.4007
  (3,5)     1.2870
```

R = spones(S) — генерирует матрицу **R** той же разреженности, что и **S**, но заменяя на 1 все ненулевые элементы исходной матрицы. Пример:

```
» S=sprand(3,2,0.3)
S =
  (3,1)     0.2987
  (1,2)     0.1991
» spones(S)
ans =
  (3,1)      1
  (1,2)      1
```

5.8.4. Визуализация разреженных матриц — команда spy

Визуализация разреженных матриц нередко позволяет выявить не только любопытные, но и полезные и поучительные свойства тех математических закономерностей, которые порождают такие матрицы или описываются последними. MATLAB имеет специальные средства для визуализации разреженных матриц, реализованные приведенными ниже командами:

spy(S) — графически отображает структуру произвольной матрицы **S**.

spy(S,marksize) — графически отображает разреженную матрицу **S**, выводя маркеры в виде точек точно определенного размера **markersize**.

spy(S,'LineSpec') — отображает разреженную матрицу в виде графика с точно определенным типом **LineSpec** и цветом маркера.

spy(S,'LineSpec',markersize) — использует точно определенный тип, цвет, размер графического маркера. **S** — обычно разреженная матрица, но допустимо использование и полной, когда расположение элементов, отличных от нуля, составляет график.

Пример:

```
»S=sparse(sprandn(20,30,0.9));spy(S,'.r',6)
```

Построенный по этому примеру график показан на рис. 5.11.

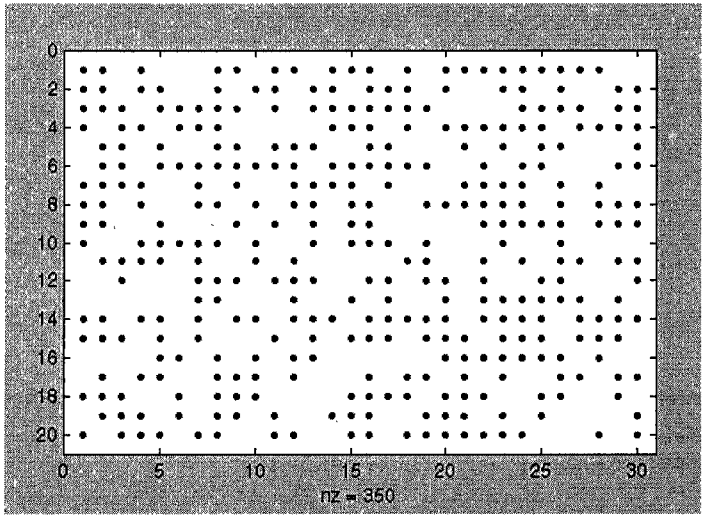


Рис. 5.11. Визуализация разреженной матрицы

5.8.5. Алгоритмы упорядочения

Упорядочение – это еще одна характерная для разреженных матриц операция. Ее алгоритм реализуется несколькими функциями:

p = colmmd(S) — возвращает вектор упорядоченности столбцов разреженной матрицы **S**. Для несимметричной матрицы **S** вектор упорядоченности столбцов **p** такой, что **S(:,p)** будет иметь более разреженное **LU**-разложение, чем **S**. Такое упорядочение автоматически применяется при выполнении операций \ и / при решении систем линейных уравнений с разреженными матрицами. Можно использовать команду **spparms,** чтобы изменить некоторые опции и параметры, связанные с эвристикой в алгоритме.

j = colperm(S) — возвращает вектор перестановок **j** такой, что столбцы матрицы **S(:,j)** будут упорядочены по возрастанию числа ненулевых элементов. Эту функцию полезно иногда применять перед выполнением **LU-** разложения. Если **S** — симметричная матрица, то **j = colperm(S)** возвращает вектор перестановок **j** такой, что и столбцы, и строки **S(j,j)** упорядочены по возрастанию ненулевых элементов. Если матрица **S** положительно определенная, то иногда полезно применять эту функцию перед выполнением разложения Холецкого. Пример:

```
» S=sparse([2,3,1,4,2],[1,3,2,3,2],[4,3,5,6,7],4,5);full(S)
ans =
     0     5     0     0     0
     4     7     0     0     0
     0     0     3     0     0
     0     0     6     0     0
```

```
» t=colperm(S)
t =
   4   5   1   2   3
» full(S(:,t))
ans =
   0   0   0   5   0
   0   0   4   7   0
   0   0   0   0   3
   0   0   0   0   6
```

p = dmperm(A) — возвращает вектор максимального соответствия **p** такой, что если исходная матрица **A** имеет полный столбцовый ранг, то **A(p,:)** — квадратная с ненулевой диагональю. Матрица **A(p,:)** называется декомпозицией Далмейджа-Мендельсона, или DM-декомпозицией.

Если **A** — приводимая матрица, линейная система **Ax = b** может быть решена приведением **A** к верхней блочной треугольной форме с неприводимым диагональным блоком. Решение может быть найдено методом обратной перестановки.

[p,q,r] = dmperm(A) — находит перестановку строк **p** и перестановку столбцов **q** квадратной матрицы **A** такую, что **A(p,q)** — матрица в блоке верхней треугольной формы.

Третий выходной аргумент **r** — целочисленный вектор, описывающий границы блоков. К-ый блок матрицы **A(p,q)** имеет индексы **r(k):r(k+1)–1**.

[p,q,r,s] = dmperm(A) — находит перестановки **p** и **q** и векторы индексов **r** и **s**, так что матрица **A(p,q)** оказывается представленной в верхней треугольной форме. Блок имеет индексы **(r(i):r(i+1)–1, s(i):s(i+1)–1)**.

В терминах теории графов диагональные блоки соответствуют компонентам Холла смежного графа матрицы **A**. Примеры:

```
» A=sparse([1,2,1,3,2],[3,2,1,1,1],[7,6,4,5,4],3,3);full(A)
ans =
   4   0   7
   4   6   0
   5   0   0
» [p,q,r]=dmperm(A)
p =
   1   2   3
q =
   3   2   1
r =
   1   2   3   4
» full(A(p,q))
ans =
   7   0   4
   0   6   4
   0   0   5
```

p = randperm(n) — возвращает случайные перестановки целых чисел **1:n**.

Пример:

```
» randperm(6)
ans =
   2   4   3   6   5   1
```

p = symmmd(S) — возвращает вектор упорядоченности для симметричной положительно определенной матрицы **S**, так что **S(p,p)** будет иметь более разреженное разложение Холецкого, чем **S**. Иногда **symmmd** хорошо работает с симметричными неопределенными матрицами. Такое упорядочение автоматически применяется при выполнении операций \ и / при решении линейных систем с разреженными матрицами.

Можно использовать команду **spparms**, чтобы изменить некоторые опции и параметры, связанные с эвристикой в алгоритме.

Алгоритм упорядочения для симметричных матриц основан на алгоритме упорядочения по разреженности столбцов. Фактически, **symmmd(S)** только формирует матрицу **K** со структурой ненулевых элементов такой, что **K'*K** имеет ту же ненулевую структуру, что и **S**, и затем вызывает алгоритм упорядочения по разреженности столбцов для **K**. Пример:

```
» B=bucky;p=symmmd(B);
» R=B(p,p);
» subplot(1,2,1),spy(B);subplot(1,2,2),spy(R)
```

На рис. 5.12. приводится пример применения функции **symmmd** к элементам разреженной матрицы.

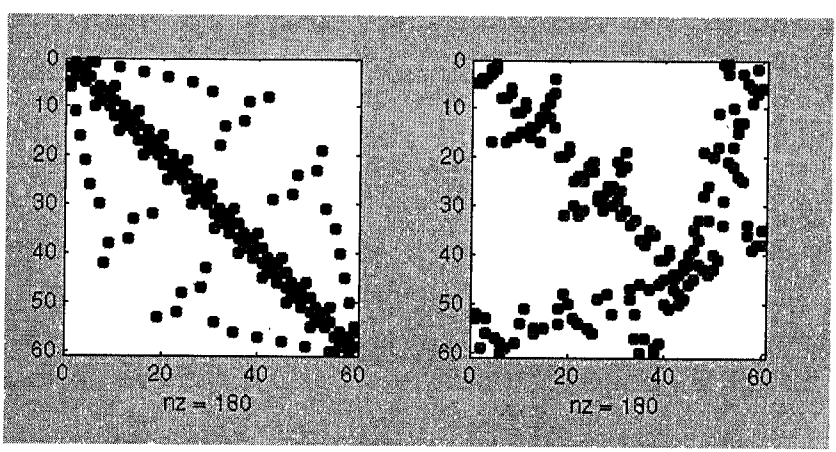


Рис. 5.12. Пример применения функции **symmmd**

r = symrcm(S) — возвращает вектор упорядоченности для симметричной матрицы **S** и называется упорядочением Катхилла-Макки. Причем, формируется такая перестановка **r**, что **S(r,r)** будет концентрировать ненулевые элементы вблизи диагонали. Это хорошее упорядочение как перед **LU**-разложением, так и перед разложением Холецкого. Упорядочение применимо как для симметричных, так и для несимметричных матриц.

Для вещественной симметричной разреженной матрицы **S** собственные значения **S(r,r)** совпадают с собственными значениями **S**, но для вычисления **eig(S(r,r))** будет затрачиваться меньше времени, чем на **eig(S)**. Пример:

```
» B=bucky;p=symrcm(B);
» R=B(p,p);
» subplot(1,2,1),spy(B);subplot(1,2,2),spy(R)
```

На рис. 5.13. приведен пример концентрации ненулевых элементов разреженной матрицы вблизи главной диагонали.

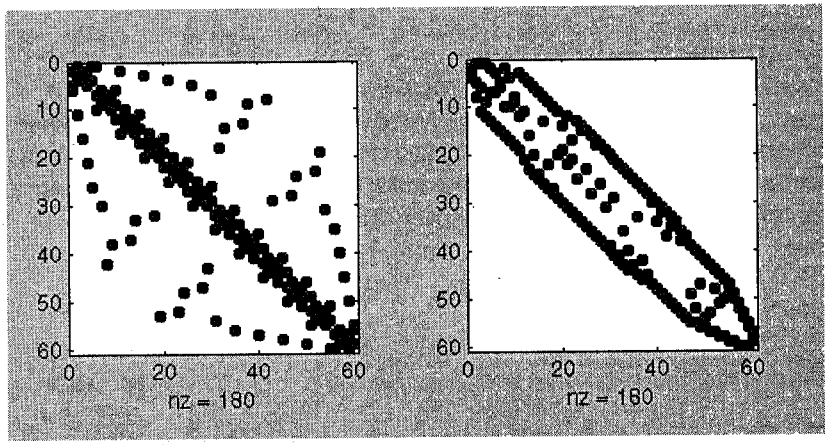


Рис. 5.13. Пример применения функции **symrcm**

5.8.6. Норма, число обусловленности и ранг разреженной матрицы

Ниже представлены функции, позволяющие вычислять числа обусловленности и ранги для разреженных матриц.

c = condest(A) — использует метод Хейджера в модификации Хаема для оценки числа обусловленности матрицы. Вычисленное значение **C** — нижняя оценка числа обусловленности матрицы **A** по первой норме.

[c,v] = condest(A) — возвращает число обусловленности и вектор **v** такой, что выполняется условие **||A*v||=||A||||v||/c**. Таким образом, для больших значений **c** вектор **v** является аппроксимацией нуль-вектора матрицы **A**

nrm = normest(S) — возвращает оценку 2-нормы матрицы **S**. Эта функция изначально предназначена для работы с разреженными матрицами, хотя она работает корректно и с большими полными матрицами.

nrm = normest(S,tol) — использует относительную погрешность **tol** вместо используемого по умолчанию значения **1.e–6**. Значение погрешности **tol** определяется, когда рассматриваемая оценка допустима.

[nrm,count] = normest(...) — возвращает оценку второй нормы и количество использованных операций.

Примеры:

```
» F=wilkinson(150);
» condest(sparse(F))
ans =
  460.2219
» normest(sparse(F))
ans =
  75.2453
```

r=sprank(S) — вычисляет структурный ранг разреженной матрицы **S**. В терминах теории графов он известен также под названиями: максимальное сечение, максимальное соответствие и максимальное совпадение. Для величины структурного ранга всегда выполняется условие **sprank(S)≥rank(S)** и более точно в точной арифметике с вероятностью 1 выполняется условие **sprank(S)==rank(sprandn(S))**.

Пример:

```
» S=[3 0 0 0 4; 5 4 0 8 0; 0 0 0 1 3];
» r=sprank(S)
r =
   3
```

5.8.7. Вычисление собственных значений и сингулярных чисел разреженных матриц — функции eigs, svds

Применение функции **eigs** решает проблему собственных значений, состоящую в нахождении нетривиальных решений системы уравнений, которая может быть интерпретирована как алгебраический эквивалент системы обыкновенных дифференциальных уравнений в явной форме Коши: **A*v = λ*v.** Вычисляются только отдельные выбранные собственные значения или собственные значения и собственные векторы.

[V,D] = eigs(A) или **[V,D] = eigs('Afun',n)** — возвращает отдельные собственные значения для первого входного аргумента. Этот аргумент может быть как квадратной матрицей (полной или разреженной, симметричной или несимметричной, вещественной или комплексной), так и строкой, содержащей имя М-файла, который применяет линейный оператор к столбцам данной матрицы. В последнем случае второй входной аргумент должен быть n-ый порядок задачи.

В случае одного выходного аргумента **D** — вектор, содержащий k собственных значений. В случае двух выходных аргументов **D** — диагональная матрица размера k× k и **V** — матрица, содержащая k -столбцов, так что **A*V = V*D** или **A*V = B*V*D**.

[U,S,V] = svds(A,k) — возвращает k наибольших сингулярных чисел и сингулярных векторов матрицы **A**. k = 5 по умолчанию. Если **A** — матрица размера m×n, то **U** — матрица размера m×k с ортонормальными столбцами, **S** — диагональная матрица размера k×k, **V** — матрицы размера nxk с ортонормальными столбцами.

[U,S,V] = svds(A,k,0) — возвращает k наименьших сингулярных чисел и сингулярных векторов.

s = svds(A,k,...) — возвращает только вектор сингулярных чисел.

Как видно из приведенного материала, система MATLAB предлагает пользователям уникальный набор матричных операторов и функций, заметно более полный, чем у других математических систем. Это открывает широчайшие возможности в решении всех видов математических задач, в которых используются современные матричные методы.

Глава 6.
Базовые средства программирования

6.1. Основные понятия программирования

6.1.1. О роли программирования системы MATLAB

До сих пор мы, в основном, использовали систему MATLAB в режиме непосредственного счета — в командном режиме. Однако при решении серьезных задач возникает необходимость сохранения используемых последовательностей вычислений, а также их последующей модификация. Иными словами, существует необходимость **программирования** решения задач.

Это может показаться отходом от важной цели, которая преследуется разработчиками большинства математических систем, — выполнения математических вычислений без использования традиционного программирования. Однако это не так. Выше было показано, что множество математических задач решается в системе MATLAB без программирования. С использованием языков высокого уровня для их решения потребовалось бы написать и отладить сотни программ.

Практически нельзя предусмотреть в одной, даже самой большой и мощной математической системе возможность решения всех специфических задач, которые могут интересовать пользователя. Программирование в системе MATLAB является эффективным средством ее расширения и адаптации к решению специфических задач пользователя. Оно реализуется с помощью **языка программирования** системы.

Большинство объектов этого языка, в частности, все команды, операторы и функции, одновременно являются объектами **входного языка** общения с системой в командном режиме работы. Так что фактически мы приступили к описанию языка программирования системы MATLAB с первых строк данной книги.

Так в чем же отличие входного языка от языка программирования? В основном - в способе фиксации создаваемых ими кодов. Сессии в командном режиме работы не сохраняются в памяти ПК (ведение дневника не в счет). Хранятся только определения введенных в ходе их выполнения переменных и функций. А вот программы на языке программирования MATLAB сохраняются в виде текстовых М-файлов. При этом могут сохраняться как целые программы в виде файлов-сценариев, так и отдельные **программные модули** — функции. Кроме того важно, что программа может менять структуру алгоритмов вычислений в зависимости от входных данных и данных, создаваемых в ходе вычислений.

С позиций программиста язык программирования системы является типичным проблемно-ориентированным языком программирования высокого уровня. Точнее

говоря, это даже язык сверхвысокого уровня, содержащий сложные операторы и функции, реализация которых на обычных языках (например, Бейсике, Паскале или Си) потребовала бы много усилий и времени. К таким функциям относятся матричные функции, функции быстрого преобразования Фурье (БПФ) и другие, а к операторам — операторы построения разнообразных графиков, генерации матриц определенного вида и т.д.

6.1.2. Основные средства программирования

Итак, программами в системе MATLAB являются М-файлы текстового формата, содержащие запись программ в виде программных кодов. Язык программирования системы MATLAB имеет следующие средства:

- данные различного типа;
- константы и переменные;
- операторы, включая операторы математических выражений;
- встроенные команды и функции;
- функции пользователя;
- управляющие структуры;
- системные операторы и функции;
- средства расширения языка.

Коды программ в системе MATLAB пишутся на языке высокого уровня, достаточно понятном для пользователей умеренной квалификации в области программирования. Язык программирования MATLAB является типичным **интерпретатором**. Это означает, что каждая инструкция программы распознается и тут же исполняется, что облегчает обеспечение диалогового режима общения с системой. Этап компиляции всех инструкций, то есть полной программы, отсутствует. Высокая скорость выполнения программ обеспечена наличием заведомо откомпилированного ядра, хранящего в себе критичные к скорости выполнения инструкции, например, такие как циклы и базовые математические функции, и тщательной отработкой системы контроля синтаксиса программ в режиме интерпретации.

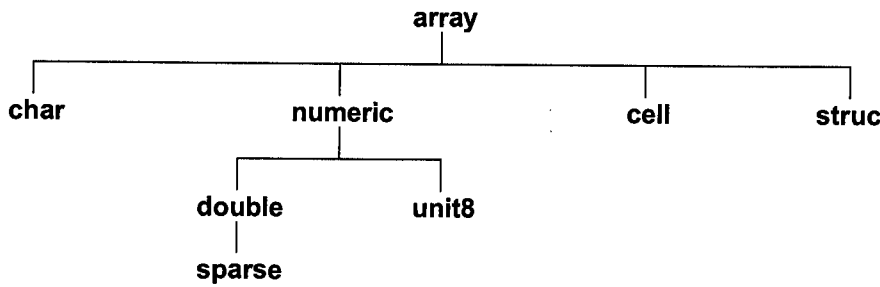
Интерпретация означает, что MATLAB не создает исполняемых конечных программ. Они существуют лишь в виде М-файлов. Для выполнения программ необходимо находиться в среде MATLAB. Однако для программ на языке MATLAB созданы компиляторы, транслирующие программы MATLAB в коды языков программирования С и С++. Это решает задачу создания исполняемых программ, изначально создаваемых в среде MATLAB. Компиляторы для системы MATLAB являются вполне самостоятельными программными средствами и в данной книге не рассматриваются.

Следует особо отметить, что не все инструкции MATLAB могут компилироваться, и перед компиляцией программы нуждаются в некоторой доработке. Зато скорость выполнения откомпилированных программ порою возрастает в 10-15 раз (правда, как правило, для простых примеров с большими циклами).

Начальное представление о переменных, встроенных константах и функциях уже было дано в предшествующих главах. В настоящей главе эти представления будут существенно расширены с позиций пользователя программиста.

6.1.3. Основные типы данных

Структура типов данных системы MATLAB представлена ниже:



Типы данных **array** и **numeric** являются **виртуальными** (кажущимися), поскольку к ним нельзя отнести какие-либо переменные. Они служат для определения и комплектования некоторых типов данных. Таким образом, в MATLAB определены следующие шесть основных типов данных, представляющих собой многомерные массивы:

double — числовые массивы с числами удвоенной точности;
char — строчные массивы с элементами-символами;
sparse — разреженные матрицы с элементами-числами удвоенной точности;
cell — массивы ячеек, которые в свою очередь могут быть массивами;
struct — массивы записей с полями, которые также могут содержать массивы;
unit8 — массивы 8-разрядных целых чисел без знаков (математические операции с ними не предусмотрены).

Кроме того, предусмотрен еще один тип данных **UserObject**, который относится к типам данных (объекты), определяемых пользователем. Типы данных **double**, **char** и **sparse** были рассмотрены ранее, так что в этой главе будут детально рассмотрены оставшиеся типы. Что касается чисел класса **unit8**, то они представляют значения от 0 до 255 и занимают в памяти 1/8 часть от размера единичного числа с двойной точностью. В основном этот тип данных применяется в служебных целях.

Каждому типу данных соответствуют некоторые характерные для него операции, называемые **методами**. Дочерние типы данных, расположенные на приведенной диаграмме ниже родительских типов, наследуют от последних их методы, что является признаком наследования объектов. Поскольку в иерархии типов данных сверху находятся данные типа **array**, это значит, что все виды данных в MATLAB являются массивами.

6.1.4. Виды программирования

На рынке программного обеспечения система MATLAB позиционируется как язык высокого уровня для научно-технических расчетов. Таким образом, возможность программирования относится к важным достоинствам данного языка, несмотря на обилие средств прямого решения задач. И действительно, именно возможность программирования сложных задач и практически неограниченного расширения системы сделала MATLAB столь почитаемой системой в университетах и в крупных научных учреждениях. MATLAB открывает широчайшие возможности по реализации новых алгоритмов вычислений, численных методов и методик расчета и проектирования различных систем и устройств.

Язык программирования системы MATLAB вобрал в себя все средства, необходимые для реализации различных видов программирования:

- процедурного;
- операторного;
- функционального;
- логического;
- структурного (модульного);
- объектно-ориентированного;
- визуально-ориентированного.

В основе процедурного, операторного и функционального типов программирования лежат процедуры, операторы и функции, используемые как основные объекты языка. Этот тип объектов присутствует в MATLAB. Логическое программирование реализуется в MATLAB с помощью логических операторов и функций. Это позволяет реализовать основные идеи логического программирования, хотя на роль выдающегося в этом классе языков программирования MATLAB не претендует.

Зато MATLAB представляет собой яркий пример плодотворности **структурного программирования**. Подавляющее большинство функций и команд языка представляют собой вполне законченные модули, обмен данными между которыми происходит через их входные параметры, хотя возможен и через глобальные переменные. Программные модули оформлены в виде текстовых М-файлов, которые хранятся на диске и подключаются к программам по мере необходимости. Важно отметить, что в отличие от многих языков программирования, применение тех или иных модулей не требует предварительного объявления, а для создания и отладки самостоятельных модулей MATLAB имеет все необходимые средства. Подавляющее большинство команд и функций системы MATLAB поставляется в виде таких модулей.

Объектно-ориентированное программирование также широко представлено в системе MATLAB. Оно особенно актуально при программировании задач графики. Что касается **визуально-ориентированного программирования**, то в MATLAB оно представлено, в основном, в пакете моделирования заданных блоками устройств и систем Simulink. Этот пакет будет рассмотрен в конце книги. В ядре системы в данный момент визуально-ориентированное программирование не используется.

6.1.5. Двойственность операторов, команд и функций

Для языка системы MATLAB понятия о командах (выполняемых при вводе с пульта) и программных операторах (выполняемых из программы) являются условными. И команды, и программные операторы могут выполняться как из программы, так и в режиме прямых вычислений. Под командами далее, в основном, понимаются средства, управляющие периферийным оборудованием, под операторами — средства, выполняющие операции с операндами (данными).

Функция преобразует одни данные в другие. Для многих функций характерен возврат значений в ответ на обращение к ним с указанием списка входных параметров — аргументов. Например говорят, что функция sin(x) в ответ на обращение к ней возвращает значение синуса x. Поэтому функцию можно использовать в арифметических выражениях, например 2*sin(x+1). Для операторов (и команд), не возвращающих значения, такое применение абсурдно. Например, "выражение" **2*plot(x,y)**, где **plot(x,y)** – оператор построения графика, будет восприниматься просто как ошибка.

В данной книге все функции, возвращающие единственное значение (или один массив), записываются малыми (строчными) буквами в виде

f_name(Список_параметров)

Тем самым мы исключаем искусственное выделение имен функций большими (заглавными) буквами, принятое в справочной системе MATLAB. Напоминаем, что как в командной строке, так и в текстах М-файлов функции записываются только малыми буквами. Для функций, возвращающих ряд значений или массивов (например **X**, **Y**, **Z**,...), запись имеет вид:

[X,Y,Z,...]=f_name(Список_параметров)

Большое значение имеет двойственность операторов и функций. Многие операторы (см. главы 4 и 5) имеют свои аналоги в виде функций. Так, например, оператор + имеет аналог в виде функции **sum**. Команды, записанные в виде

Command argument

нередко имеют форму записи и в виде функции

Command('argument')

Примеры:

```
» help sin
 SIN   Sine.
   SIN(X) is the sine of the elements of X.
 Overloaded methods
   help sym/sin.m

» help('sin')
```

```
SIN   Sine.
  SIN(X) is the sine of the elements of X.
Overloaded methods
  help sym/sin.m

» type('sin')
sin is a built-in function.

» type sin
sin is a built-in function.
```

Указанная двойственность лежит в основе выбора между процедурным и функциональным типами программирования, каждый из которых имеет своих поклонников и противников и может (в той или иной мере) подходить для решения различных классов задач. При этом переход от одного типа программирования к другому возможен в пределах одной программы и происходит настолько естественно, что большинство пользователей даже не задумывается над тем, каким же типом (или стилем) программирования они преимущественно пользуются.

6.1.6. Некоторые ограничения

Поскольку язык программирования системы MATLAB ориентирован на структурное программирование, в нем нет номеров строк (присущих до недавнего времени Бейсику) и программных операторов безусловного перехода GO TO. Имеются лишь управляющие структуры следующих типов: условных выражений **if...else...elseif...end**, циклы **for...end** и **while..end** и **while...end**. Их форма похожа на ту, которая используется в языке Pascal (т.е. область действия управляющих структур выделяется их заголовком, но без слова **begin**, и словом **end**). С позиций теории структурного программирования этих средств достаточно для решения любых задач. В версии MATLAB 5.* добавлены операторы-переключатели типа **case**.

Однако в MATLAB исключены те средства, возможности которых можно реализовать уже имеющимися средствами. Зато резко увеличен набор средств программирования для решения математических задач, прежде всего сводящихся к матричным вычислениям и реализации современных численных методов решения..

Программирование простых задач в среде MATLAB очень напоминает программирование на Бейсике [1,16]. Во многих случаях программы на Бейсике можно почти дословно перевести на язык системы, учтя небольшие отличия в синтаксисе этих языков. Это нельзя трактовать как отсутствие у языка MATLAB индивидуальных черт. Любители C, Pascal или Fortran также заметят сходство этих языков с языком программирования MATLAB. Так что правильнее считать, что этот язык имеет вполне самостоятельное значение. Он вобрал в себя лучшие средства универсальных языков программирования.

6.2. М-файлы-сценарии и функции

Итак, мы установили, что работа в командном режиме (сессия) не является программированием. Внешним атрибутом последнего в MATLAB служит задание последовательности действий по программе, записанной в виде М-файла. В главе 1 было показано, что для создания М-файлов может использоваться как встроенный редактор, так и любой текстовый редактор, поддерживающий формат ASCII. Подготовленный и записанный на диск М-файл становится частью системы и его можно вызывать как из командной строки, так и из другого М-файла. Есть два типа М-файлов: файлы-сценарии и файлы-функции. Важно, что в процессе своего создания они проходят синтаксический контроль с помощью встроенного в систему MATLAB редактора и отладчика М-файлов.

6.2.1. Структура и свойства файла-сценария

Файл-**сценарий**, именуемый также **Script-файлом**, является просто записью серии команд без входных и выходных параметров. Он имеет следующую структуру:

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

Важны следующие свойства файлов-сценариев:

- они не имеют входных и выходных аргументов;
- работают с данными из рабочей области;
- в процессе выполнения не компилируются;
- представляют собой зафиксированную в виде файла последовательность операций, полностью аналогичную той, что используется в сессии.

Основным комментарием является первая строка текстовых комментариев, а дополнительным — последующие строки. Основной комментарий выводится при исполнении команд **lookfor** и **help имя_каталога**. Полный комментарий выводится при исполнении команды **help Имя_файла**. Рассмотрим следующий файл-сценарий:

```
%Plot with color red
%Строит график синусоиды линией красного цвета
%с выведенной масштабной сеткой в интервале [xmin,xmax]
x=xmin:0.1:xmax;
plot(x,sin(x),'r')
grid on
```

Первые три строки здесь — это комментарий, остальные — тело файла. Обратите внимание на возможность задания комментария на русском языке. Знак % в комментариях должен начинаться с первой позиции строки. В противном случае команда **help**

name не будет воспринимать комментарий (иногда это может понадобиться) и возвращать сообщение вида

No help comments found in name.m.

Будем считать, что файл записан под именем pcr. Работа с ним представлена на рис. 6.1. Показана подготовка к запуску файла (задание конкретных значений для xmin и xmax), запуск файла, получение рисунка (окно внизу) и вызов комментария командой **help pcr**. Командой **type pcr** можно вывести полный листинг файла.

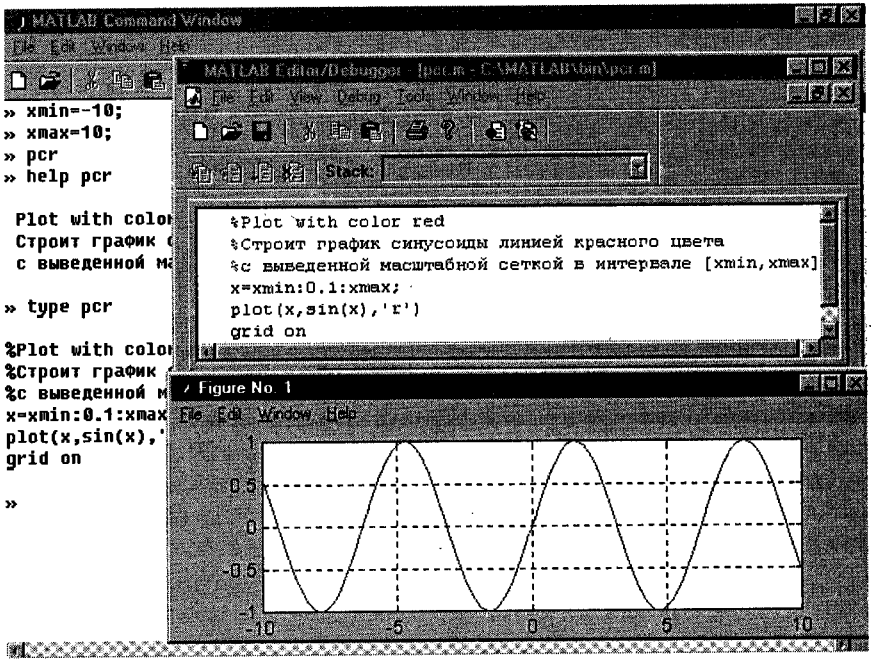


Рис.6.1. Пример работы с файлом pcr

Обратите внимание на то, что такой файл нельзя запустить без предварительной подготовки — задания значений переменным xmin и xmax, использованным в теле файла. Это следствие первого свойства файлов-сценариев — они работают с данными из рабочей области. Переменные, используемые в файлах-сценариях, являются глобальными, то есть они действуют одинаково в командах сессии и внутри программного блока, которым является файл-сценарий. Поэтому заданные в сессии значения переменных используются и в теле файла. Имена файлов-сценариев нельзя использовать в качестве параметров функций, поскольку файлы-сценарии не возвращают значений. Можно сказать, что файл-сценарий — это простейшая программа на языке программирования MATLAB.

6.2.2. Структура и свойства М-файлов-функций

М-файл-**функция** является типичным объектом языка программирования системы MATLAB. Одновременно файл является полноценным модулем с точки зрения структурного программирования, поскольку содержит входные и выходные параметры и использует аппарат локальных переменных. 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 типа **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 М-файл-функция завершается строкой с точкой с запятой (;), то для возврата значения функции используется программный оператор **return**.

Приведенная форма файла-функции характерна для функции с одним выходным параметром. Если выходных параметров больше, то они указываются в квадратных скобках после слова **function**. При этом структура модуля имеет вид:

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

Такая функция во многом напоминает полноценную процедуру. Ее нельзя использовать непосредственно в математических выражениях, поскольку она возвращает не единственный результат, а множество результатов — по числу выходных параметров. Поэтому, как отмечалось, данная функция используется как отдельный элемент программ вида:

[var1,va2,...]=f_name(Список_параметров)

После применения элемента переменные выхода var1, var2,... становятся определенными и их можно использовать в последующих математических выражениях и иных сегментах программы. Если функция используется в виде **f_name(Список_параметров)**, то возвращается значение только первого выходного параметра — переменной var1.

6.2.3. Статус переменных и команда global

Итак, из сказанного ясно, что переменные в файлах-сценариях являются глобальными, а в файлах-функциях — локальными. В главе 1 показывалось, что 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 тщательной планировки такой передачи. В жизни мы далеко не всегда едим черную икру (локальные переменные) и иногда хотим отведать черного хлебушка (глобальные переменные). Так и при создании файлов-функций порой желательно применение глобальных переменных. Ответственность за это должен брать на себя программист, создающий программные модули.

Команда

global var1 var2 ...

позволяет объявить переменные модуля-функции глобальными. Таким образом, внутри функции могут использоваться и такие переменные, если это нужно по условиям решения вашей задачи.

6.2.4. Использование подфункций

Начиная с версии 5.0, в функции системы MATLAB можно включать подфункции. Они объявляются и записываются в теле основных функций и имеют идентичную им конструкцию. Не следует путать эти функции с внутренними функциями, встроенными в ядро системы MATLAB. Ниже представлен пример функции с такой подфункцией:

```
function [mean,stdev] = statv(x)
%STATV Interesting statistics.
%Пример функции с встроенной подфункцией
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 подфункцией
```

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

При обращении к функции интерпретатор системы MATLAB прежде всего просматривает М-файл на предмет выявления подфункций. Если они обнаружены, то задаются как локальные функции. Благодаря локальному действию подфункций, их имена могут совпадать с именами основных функций системы. Если в функции и подфункциях должны использоваться общие переменные, их надо объявить глобальными как в функции, так и в ее подфункциях.

6.2.5. Частные каталоги

Для записи М-файлов используются каталоги, называемые **родительскими каталогами**. Они содержат группы файлов определенного функционального назначения, например, по статистическим расчетам, матричным операциям, вычислению определенных классов функций и так далее.

Однако, начиная с версии MATLAB 5.0, появилась возможность в родительских каталогах создавать **частные каталоги** — PRIVATE. Входящие в них М-файлы, доступны только файлам родительского каталога. Файлы частных каталогов просматриваются интерпретатором системы MATLAB в первую очередь. Применение частных каталогов позволяет изменять исходные файлы, сохраняя оригиналы в родительском каталоге в неизменном виде.

Если вы решили отказаться от применения измененного файла, достаточно стереть его в частном каталоге. Такая возможность связана с тем, что интерпретатор при обнаружении имени М-файла прежде всего просматривает частный каталог данного файла и интерпретирует найденный в нем файл. И только если файл не найден, ищется файл в родительском каталоге.

6.2.6. Вычисление строковых выражений — функции eval и feval

Строковые выражения обычно не вычисляются, так что, к примеру, вывод строки '2+3' просто повторяет строку:

```
» '2+3'
ans =
2+3
```

Однако с помощью функции **eval('Строковое_выражение')** строковое выражение, представляющее математическое выражение, может быть вычислено:

```
» eval('2+3')
ans =
    5
» eval('2*sin(1)')
ans =
   1.6829
```

Еще одна функция

```
feval('Имя_функции',x1,x2,...)
```

имеет важное достоинство — она позволяет передавать в вычисляемую функцию список ее аргументов. При этом вычисляемая функция записывается только своим именем. Это поясняют следующие примеры:

```
» feval('prod',[1 2 3])
ans =
    6
» feval('sum',[1 2 3; 4 5 6],2)
ans =
    6
   15
```

Рекомендуется применять функцию **feval** при вычислении значений функций, записанных в виде строки. М-файлы, содержащие функцию **feval,** корректно компилируются компилятором системы MATLAB.

6.2.7. Вывод сообщения об ошибке — команда error

Часто в ходе вычислений возникают ошибки. Например, мы уже сталкивались с проблемой вычисления функции sin(x)/x, когда при x=0 имеет место ошибка вида "деление на ноль". При появлении ошибки вычисления могут завершиться досрочно с выводом сообщения об ошибке. Следует, однако, отметить, что не все ошибки вызывают остановку вычислений. Некоторые сопровождаются только выдачей предупреждающей надписи.

Такие ситуации должны учитываться программистом, отмечаться как ошибочные и по возможности устраняться. Для вывода сообщения об ошибке служит команда

```
error('Сообщение об ошибке')
```

при исполнении которой вычисления прерываются и выдается сообщение об ошибке, заданное в апострофах. Ниже дан пример вычисления функции sd(x)=sin(x)/x, в котором задано сообщение об ошибке на русском языке:

```
function f=sd(x)
if x==0 error('Ошибка - деление на 0'), end
f=sin(x)/x
```

Для выявления ситуации об ошибке использован оператор условного перехода **if**, который будет описан детально несколько позднее. Результат выполнения данной функции приводится ниже:

```
» sd(1)
f =
   0.8415
ans =
```

```
   0.8415
» sd(0)
??? Error using ==> sd
Ошибка - деление на 0
```

Если остановка программы при появлении ошибки нежелательна, то может использоваться команда вывода предупреждающего сообщения:

```
warning('Предупреждающее сообщение')
```

Эта команда выводит стоящее в апострофах сообщение, но не препятствует дальнейшей работе программы. Признаком ошибки являются символы ???, а признаком предупреждения — слово Warning в соответствующих сообщениях.

6.2.8. Функция lasterr и обработка ошибок

Опытные программисты должны предусматривать ситуации с появлением ошибок. К примеру, при x=0 выражение sin(x)/x=0/0=1, и правильным решением было бы вместо его вычисления использовать значение 1.

В данном простом примере приводится функция sd0, исключающая вычисление sin(x)/x при x=0:

```
function f=sd0(x)
if x==0 f=1; else f=sin(x)/x; end
return
```

При этом вычисления пройдут корректно при любом x:

```
» sd0(1)
ans =
   0.8415
» sd0(0)
ans =
    1
```

Для вывода сообщения о последней сделанной ошибке служит функция **lasterr** (см. пример ниже):

```
» aaa
??? Undefined function or variable 'aaa'.
» 2+3
ans =
    5
» 1/0
```

```
Warning: Divide by zero.
ans =
   Inf
» lasterr
ans =
Undefined function or variable 'aaa'.
```

Как нетрудно заметить, функция **lasterr** возвращает текстовое сообщение, следующее за знаками **???** сообщения об ошибке.

В общем случае программы могут содержать обработчики ошибок, направляющие ход вычислений в нужное русло, даже если ошибка появляется. Но для этого требуются средства индикации и обработки ошибок. Основными из них являются функции **eval** и **lasterr**. О функции **lasterr** уже говорилось, а функция

```
eval('try','catch')
```

в отличии от ранее рассмотренной формы имеет два входных аргумента. Один из них — это строчное выражение, которое преобразуется в исполняемую форму и выполняется при отсутствии ошибки. Если же происходит ошибка, то строка 'catch' вызывает обращение к функции обработки ошибки.

6.2.9. Функции подсчета числа входных и выходных аргументов nargin и nargout

При создании функций со специальными свойствами весьма полезны две приведенные ниже функции:

nargin — возвращает число входных параметров данной функции.
nargout — возвращает число выходных параметров данной функции.

Пусть, к примеру, мы хотим создать функцию, вычисляющую сумму квадратов пяти аргументов x1, x2, x3, x4 и x5. Обычный путь состоит в следующем — создаем функцию с именем sum2_5:

```
function f=sum2_5(x1,x2,x3,x4,x5);
f=x1^2+x2^2+x3^2+x4^2+x5^2;
```

Теперь проверим ее в работе:

```
» sum2_5(1,2,3,4,5)
ans =
   55
» sum2_5(1,2)
??? Input argument 'x3' is undefined.
```

```
Error in ==> C:\MATLAB\bin\sum2_5.m
On line 2  ==> f=x1^2+x2^2+x3^2+x4^2+x5^2;
```

Итак, при всех пяти аргументах функция работает корректно. Но если аргументов менее пяти, она выдает сообщение об ошибке. С помощью функции **nargin** можно создать функцию sum2_5m, которая работает корректно при любом числе заданных входных аргументов в пределах от 1 до 5:

```
function f=sum2m_5(x1,x2,x3,x4,x5);
n=nargin;
if n==1 f=x1^2; end
if n==2 f=x1^2+x2^2;end
if n==3 f=x1^2+x2^2+x3^2; end
if n==4 f=x1^2+x2^2+x3^2+x4^2; end
if n==5 f=x1^2+x2^2+x3^2+x4^2+x5^2;end
```

В данной функции используется оператор условного выражения **if-end**, который будет детально описан далее. Но и без этого ясно, что благодаря применению функции **nargin** и оператора условного выражения всякий раз вычисления идут по формуле с числом слагаемых, равным числу входных аргументов — от одного до пяти. Это видно из приведенных ниже примеров:

```
» sum2_5m(1)
ans =
   1
» sum2_5m(1,2)
ans =
   5
» sum2_5m(1,2,3)
ans =
  14
» sum2_5m(1,2,3,4)
ans =
  30
» sum2_5m(1,2,3,4,5)
ans =
  55
» sum2_5m(1,2,3,4,5,6)
??? Error using ==> sum2_5m
Too many input arguments.
```

Итак, при изменении числа входных операторов от 1 до 5 вычисления проходят корректно. При большем числе операторов выводится сообщение об ошибке. Это уже действует встроенная в интерпретатор MATLAB система диагностики ошибок.

6.2.10. Особенности задания и чтения комментариев

Как отмечалось, команда **help name**, где name — имя М-файла, обеспечивает чтение первой строки с текстовым комментарием и тех строк с комментариями, которые следуют непосредственно за первой строкой. Комментарий за пределами этой области не выводится. Это позволяет использовать не выводимый программный комментарий, например:

```
Z=X+Y %Массив Z является суммой массивов A и B
```

Пустая строка прерывает вывод комментария при исполнении команды **help name**. Команда **type name** выводит текст программы со всеми комментариями, в том числе и следующими после пустых строк.

Команда **help catalog**, где catalog — имя каталога с М-файлами, позволяет вывести комментарий, общий для всего каталога. Такой комментарий содержится в файле contents.m, который пользователь может создать самостоятельно с помощью редактора М-файлов. Если такого файла нет, то будет выведен список первых строк комментариев для всех М-файлов каталога.

6.2.11. Особенности выполнения М-файлов-функций

М-файлы-функции могут использоваться как в командном режиме, так и в других М-файлах. При этом необходимо указывать все входные и выходные параметры. Исключением является случай, когда выходной параметр — единственный; в этом варианте функция возвращает единственный результат и может использоваться в математических выражениях. При использовании глобальных переменных они должны быть объявлены во всех М-файлах, используемых в решении заданной задачи и во всех входящих в них встроенных подфункциях.

Имена функций должны быть уникальными. Это связано с тем, что при обнаружении каждого нового имени MATLAB проверяет, не является ли это имя именем переменной, подфункции в данном М-файле, частной функции в каталогах PRIVATE или именем функции в пути доступа к данной функции. Если последняя встречается, то будет исполнена именно эта функция. В новой версии MATLAB возможно переопределение функции, но это не рекомендуется делать подавляющему большинству пользователей.

Если аргумент функции используется только для вычислений и его значения не меняются, то аргумент передается ссылкой, что уменьшает затраты памяти. В других случаях аргумент передается значением. Для каждой функции выделяется своя (рабочая) область памяти, не входящая в область, предоставляемую системе MATLAB. Глобальные переменные принадлежат ряду областей памяти. При их изменении меняется содержимое всех этих областей памяти.

При решении задач с большим объемом данных может ощущаться нехватка оперативной памяти. Признаком этого становится появление сообщения об ошибке "Out of memory". В этом случае может быть полезным применение следующих мер:

- стирание ставших ненужными данных — прежде всего больших массивов;
- увеличение размеров файла подкачки Windows;
- уменьшение размера используемых данных;
- снятие ограничений на размеры используемой памяти;
- увеличение объема памяти.

Чем больше емкость ОЗУ компьютера, на котором используется система MATLAB, тем меньше вероятность возникновения указанной ошибки. Опыт показывает, что даже при решении задач умеренной сложности емкость ОЗУ должна быть не менее 16-32 Мбайт.

6.2.12. Создание P-кодов — команда pcode

Когда встречается сценарий или функция в виде M-файла, то всякий раз выполняется трансляция файлов, создающая так называемые P-коды (псевдокоды). Она связана с синтаксическим контролем сценария или функции, который несколько замедляет вычисления. MATLAB позволяет создавать P-коды сценариев и функций с помощью команды **pcode**.

pcode имя_M-файла

Временные P-коды хранятся в памяти до использования команды clear или завершения сеанса работы.

Особенно полезно применение этой команды в том случае, когда используются сложная дескрипторная графика и средства создания GUI. В этом случае выигрыш по скорости выполнения вычислений может быть заметным. Переход к P-кодам полезен, если пользователь желает скрыть созданный им M-файл и реализованные в нем идеи и алгоритмы. Файл с P-кодами имеет расширение .p. Размер файла с P-кодами обычно больше, чем размер M-файла.

Рассмотрим следующий пример — создадим файл-сценарий pp.m:

```
told=cputime;
x=-15:.0001:15;
plot(x,sin(x))
t=cputime-told
```

Эта программа строит график функции sin(x) по большому числу точек. Кроме того, она вычисляет время выполнения данного сценария в секундах. При первом пуске получим:

```
» pp
t =
   0.4400
```

Теперь выполним создание P-кодов и вновь запустим программу:

```
» pcode pp
» pp
t =
   0.3900
» pp
t =
   0.3300
```

Как можно заметить, после преобразования в P-коды время построения графика несколько уменьшилось. Но гораздо важнее то, что теперь вы можете стереть файл pp.m (но оставить pp.p) и снова запустить программу. Ваши слишком любопытные коллеги едва ли разберутся с тем, что записано в машинных кодах файла pp.p, хотя с помощью специальных программ (декомпиляторов) такая возможность реализуется.

6.3. Многомерные массивы

Некоторые типы данных, например числа, константы и переменные, которые используются не только для программирования, но и расчетов в командном режиме, мы уже достаточно подробно рассмотрели в главе 1. Поэтому ниже мы коснемся вопросов, связанных с более сложными типами данных. Начнем с многомерных массивов.

6.3.1. Понятие о многомерных массивах

Многомерные массивы характеризуются размерностью более двух. Таким массивам можно дать наглядную интерпретацию. Так, матрицу (двумерный массив) можно записать на одном листе бумаги в виде строк и столбцов, состоящих из элементов матрицы.

Тогда блокнот с такими листками можно считать трехмерным массивом, полку в шкафу с блокнотами — четырехмерным массивом, шкаф со множеством полок — пятимерным массивом и так далее. В этой книге практически нигде, кроме этого раздела, мы не будем иметь дело с массивами, размерность которых выше размерности матриц, но знать о возможностях MATLAB в части задания и применения многомерных массивов все же полезно.

В нашей литературе понятия размер и размерность массивов являются почти синонимами. Однако в литературе по системе MATLAB и в данной книге они имеют явно разный смысл. Под размерностью массивов понимается число измерений в простран

ственном представлении массивов, а под размером — произведение числа элементов в каждой размерности массива.

6.3.2. Применение оператора : в многомерных массивах

При обычном задании массивов (с помощью символа точки с запятой ;) размерность массива получается на 1 больше, чем число символов ;. Оператор : (двоеточие) позволяет легко выполнять операции по увеличению размерности массивов. Приведем пример на формирование трехмерного массива путем добавления новой страницы. Пусть у нас задан исходный двумерный массив M с размером 3 ×2:

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

Для добавления новой страницы с тем же размером можно расширить M следующим образом:

```
» M(:,:,2)=[10 11 12; 13 14 15; 16 17 18]
M(:,:,1) =
   1   2   3
   4   5   6
   7   8   9
M(:,:,2) =
  10  11  12
  13  14  15
  16  17  18
```

Посмотрим, что теперь содержит массив M при явном его указании:

```
» M
M(:,:,1) =
   1   2   3
   4   5   6
   7   8   9
M(:,:,2) =
  10  11  12
  13  14  15
  16  17  18
```

Как можно заметить, числа в выражениях M(:,:,1) и M(:,:,2) означают наличие в массиве двух страниц.

6.3.3. Доступ к отдельному элементу многомерного массива

Чтобы вызвать средний элемент вначале первой, а затем второй страницы, надо записать:

```
» M(2,2,1)
ans =
   5
» M(2,2,2)
ans =
  14
```

Таким образом, в многомерных массивах используется то же правило индексации, что в одномерных и двумерных. Произвольный элемент, например трехмерного массива, задается как **M(i,j,k)**, где i — номер строки, j — номер столбца и k — номер страницы. Этот элемент можно вывести, а можно присвоить ему заданное значение x: **M(i,j,k)=x**.

6.3.4. Удаление размерности у многомерного массива

Мы уже отмечали возможность удаления отдельных столбцов присвоением им значений пустого вектора-столбца [ ]. Этот прием нетрудно распространить и на страницы и вообще размерности многомерного массива. Например, первую страницу полученного массива M можно удалить следующим образом:

```
» M(:,:,1)=[ ]
M =
  10   11   12
  13   14   15
  16   17   18
```

Нетрудно заметить, что в этом массиве осталась только вторая страница и что размерность массива уменьшилась на 1 — он стал двумерным.

6.3.5. Создание страниц, заполненных константами и случайными числами

Если после знака присваивания стоит численная константа, то соответствующая часть массива будет состоять из элементов, содержащих данную константу. Например, создадим массив из массива M (см. пример выше), у которого вторая страница содержит единицы:

```
» M(:,:,2)=1
M(:,:,1) =
   10   11   12
   13   14   15
   16   17   18
M(:,:,2) =
    1    1    1
    1    1    1
    1    1    1
```

А теперь заменим первую страницу массива на страницу с нулевыми элементами:

```
» M(:,:,1)=0
M(:,:,1) =
    0    0    0
    0    0    0
    0    0    0
M(:,:,2) =
    1    1    1
    1    1    1
    1    1    1
```

6.3.6. Использование функций ones, zeros, rand и randn

Функции **ones** (создание массивов с единичными элементами), **zeros** (создание массивов с нулевыми элементами) и **rand** или **randn** (создание массивов с элементами — случайными числами с равномерным и нормальным распределением) могут также использоваться для создания многомерных массивов. Например:

```
» E=ones(3,3,2)
E(:,:,1) =
    1    1    1
    1    1    1
    1    1    1
E(:,:,2) =
    1    1    1
    1    1    1
    1    1    1

» Z=zeros(2,2,3)
Z(:,:,1) =
    0    0
    0    0
Z(:,:,2) =
```

```
     0     0
     0     0
Z(:,:,3) =
     0     0
     0     0

» R=randn(3,2,2)
R(:,:,1) =
   -1.6656   -1.1465
    0.1253    1.1909
    0.2877    1.1892
R(:,:,2) =
   -0.0376   -0.1867
    0.3273    0.7258
    0.1746   -0.5883
```

Эти примеры достаточно очевидны и не требуют особых комментариев. Обратите, однако, внимание на легкость задания размеров массивов для каждой размерности. Кроме того, следует отметить, что если хотя бы одна размерность массива равна нулю, то массив будет пустым:

```
» A=randn(3,3,3,0)
A =
  Empty array: 3-by-3-by-3-by-0
```

Как видно из данного примера, пустой массив возвращается с соответствующим комментарием.

6.3.7. Функция объединения массивов cat

Для создания многомерных массивов служит описанная ранее для матриц специальная функция конкатенации **cat**:

cat(DIM,A,B) — возвращает результат объединения двух массивов **A** и **B** вдоль размерности DIM.
cat(2,A,B) — возвращает массив **[A,B]**, объединенный по столбцам.
cat(1,A,B) — возвращает массив **[A;B]**, объединенный по строкам.
B=cat(DIM,A1,A2,...) — объединяет множество входных массивов **A1**, **A2**,... вдоль размерности DIM.

Функции, использующие оператор запятой как разделитель **cat(DIM,C{:})** или **cat(DIM,C.FIELD)**, обеспечивают объединение массива ячеек или массива записей (см. далее), содержащего числовые матрицы, в многомерный массив. Ниже приводятся примеры применения функции **cat**:

```
» M1=[1 2;3 4]
```

```
M1 =
   1   2
   3   4
» M2=[5 6;7 8]
M2 =
   5   6
   7   8
» cat(1,M1,M2)
ans =
   1   2
   3   4
   5   6
   7   8
» cat(2,M1,M2)
ans =
   1   2   5   6
   3   4   7   8
» M=cat(3,M1,M2)
M(:,:,1) =
   1   2
   3   4
M(:,:,2) =
   5   6
   7   8
```

6.3.8. Вычисления числа размерностей массива — функция ndims

Функция **ndims(A)** возвращает размерность массива **A** (если она больше или равна двум). Следующий пример иллюстрирует применение функции **ndims**:

```
» M=rand(2:3:4:5);
» ndims(M)
ans =
   4
```

6.3.9. Перестановки размерностей массивов — функции permute и ipermute

Если представить многомерный массив в виде страниц, то их перестановка является перестановкой размерностей массива. Для двумерного массива перестановка означает транспонирование массива — замену строк столбцами и наоборот. Следующие функции обеспечивают перестановку размерностей многомерного массива:

permute(A,ORDER) — переставляет размерности массива **A** в порядке, определяемом вектором перестановок ORDER. Элементы вектора перестановок числа от 1 до N, где N — число размерностей массива.

ipermute(A,ORDER) — делает то же, но в обратном порядке.

Ниже приводятся примеры применения этих функций:

```
» A=[1 2; 3 4];
» B=[5 6; 7 8];
» C=[9 10;11 12];
» D=cat(3,A,B,C)
D(:,:,1) =
   1   2
   3   4
D(:,:,2) =
   5   6
   7   8
D(:,:,3) =
   9   10
  11   12
» size(D)
ans =
   2   2   3
» size(permute(D,[3 2 1]))
ans =
   3   2   2
» size(ipermute(D,[2 1 3]))
ans =
   2   2   3
```

6.3.10. Сдвиг размерностей массивов — функция shiftdim

Сдвиг размерностей реализуется функцией:

B=shiftdim(X,N) — сдвиг размерностей в массиве **X** на величину N. Если N>0, то сдвиг выполняется влево по кругу. Если N<0, сдвиг выполняется вправо, причем, N первых размерностей дополняются единичными.

[B,NSHIFTS]=shiftdim(X) — возвращает массив **B** с тем же числом элементов, что и у массива **X**, но с удаленными ведущими единичными размерностями и числом размерностей NSHIFTS. Если **X** — скаляр, функция ничего не возвращает.

Следующий пример иллюстрирует применение функции **shfiftdim**:

```
» A=randn(1,2,3,4);
```

```
» [B,N]=shiftdim(A)
B(:,:,1) =
  -1.5937   0.5711   0.6900
  -1.4410  -0.3999   0.8156
B(:,:,2) =
   0.7119   0.6686  -1.2025
   1.2902   1.1908  -0.0198
```

6.3.11. Удаление единичных размерностей — squeeze

Функция **squeese(A)** возвращает массив, в котором удалены все единичные размерности. Следующий пример поясняет ее работу:

```
» A=randn(1,2,1,3,1);
» B=squeeze(A)
B =
   0.6145   1.6924  -0.6436
   0.5077   0.5913   0.3803
```

Обратите внимание на то, что массив **A** превращается в массив с размерностью 3 и размером 2×3.

6.4. Массивы записей

6.4.1. Структура записей

Записи относятся к сложным типам данных. Они могут содержать разнородные данные, относящиеся к некоторому поименованному объекту. Например, объект man (человек) может характеризоваться следующими данными:

```
man(i,...)                    %Имя записи
.name       Иван              %Имя человека
.surname    Петров            %Фамилия
.date       1956              %Год рождения
.height     170.5             %Рост
.weight     70.34             %Вес
```

Первые два столбца представляют **структуру записи**. Как нетрудно заметить, каждая i-ая запись состоит из ряда полей, имеющих имена, например, man(i).name, man(i).date и так далее. Поля могут иметь данные любого типа — от пустого поля [ ] до массивов. Приведенная выше запись имеет размер 1×1. MATLAB характеризуется возможностью создания массивов записей, что позволяет создавать мощные базы данных.

Поле структуры записи может содержать другую вложенную структуру или массив структур. Это позволяет создавать вложенные структуры и даже многомерные массивы структур. Поскольку в данной книге такие записи не используются, отсылаем заинтересованного читателя к книге [33], где они описаны более подробно.

6.4.2. Создание записей и доступ к их компонентам

Для задания записей на языке MATLAB можно использовать операторы присвоения, что иллюстрирует следующий пример:

```
» man.name='Иван';
» man.surname='Петров';
» man.date=1956;
» man.height=170.5;
» man.weight=70.34;
```

Здесь построена базовая запись без индексного указателя. Теперь можно просмотреть полученную запись, просто указав ее имя:

```
» man
man =
     name: 'Иван'
  surname: 'Петров'
     date: 1956
   height: 170.5000
   weight: 70.3400
```

Нетрудно догадаться, что компоненты записи можно вызывать по имени и менять их значения. При этом имя компонента состоит из имени записи и имени поля, разделенных точкой. Это поясняют следующие примеры:

```
» man.date
ans =
     1956
» man.date=1964
man =
     name: 'Иван'
  surname: 'Петров'
     date: 1964
   height: 170.5000
   weight: 70.3400
```

Для создания массива записей вводится их индексация. Например, вектор записей можно создать, введя индекс в скобках после имени записи. Так, для создания новой, второй записи можно поступить следующим образом:

```
» man(2).name='Петр';
» man(2).surname='Сидоров';
» man(2).date=1959;
» man(2)
ans =
     name: 'Петр'
  surname: 'Сидоров'
     date: 1959
   height: [ ]
   weight: [ ]
» man(2).surname
ans =
Сидоров
» length(man)
ans =
    2
```

Обратите внимание на то, что не все поля данной записи заполнены. Поэтому значением двух последних компонентов записи 2 оказываются пустые массивы. Число записей позволяет найти функция **length** (см. последний пример).

6.4.3. Функция создания записей struct

Для создания структур записей используется следующая функция:

struct('field1',VALUES1,'field2',VALUES2,...) — возвращает созданную данной функцией запись, содержащую указанные в параметрах поля 'fieldn' с их значениями 'VALUESn'.

struct(OBJ) — конвертирует объект OBJ в эквивалентную структуру.

Пример:

```
» S=struct('student','Иванов','group',2,'estimate','good')
S =
    student: 'Иванов'
      group: 2
   estimate: 'good'
```

6.4.4. Контроль полей и записей — функции iafield и isstruct

Выполнение операций с полями и элементами полей проводится по тем же правилам, что и при работе с обычными массивами. Однако существует ряд функций, позволяющих работать с записями.

Приведенные ниже функции служат для тестирования полей и структур записей:

isfields(S,'field') — возвращает логическую 1, если 'field' является именем поля структуры S.

isstruct(S) — возвращает логическую 1, если S-структура, и 0 в ином случае.

Их использование на примере структуры man показано ниже:

```
» isfield(man,'name')
ans =
   1
» isfield(man,'family')
ans =
   0
» isstruct(man)
ans =
   1
» isstruct(many)
??? Undefined function or variable 'many'.
» isstruct('many')
ans =
   0
```

6.4.5. Функция возврата имен полей — fieldname

Следующая функция позволяет вывести имена полей заданной записи:

fieldnames(S) — возвращает имена полей записи S в виде массива ячеек. Пример:

```
» fieldnames(man)
ans =
   'name'
   'surname'
   'date'
   'height'
   'weight'
```

6.4.6. Функция возврата содержимого полей записи — getfield

В конечном счете работа с записями сводится к выводу и использованию содержимого полей. Для возврата содержимого поля записи S служит функция:

getfield(S,'field') — возвращает содержимое поля записи S, что эквивалентно **S.field**.

getfield(S,{i,j},'field',{k}) — эквивалентно **F = S(i,j).field(k)**.

Пример:

```
» getfield(man(2),'name')
ans =
Петр
```

6.4.7. Функция присвоения значений полям — setfield

Для присвоения полям заданных значений используется следующая функция:

setfield(S,'field',V) — возвращает запись S с присвоением полю 'field' значения V, что эквивалентно **S.field = V**.
setfield(S,{i,j},'field',{k},V) — эквивалентно **S(i,j).field(k) = V**.
Пример:

```
» setfield(man(2),'name','Николай')
ans =
     name: 'Николай'
   surname: 'Сидоров'
     date: 1959
   height: [ ]
   weight: [ ]
```

6.4.8. Удаление полей — функция rmfield

Для удаления полей записи можно использовать следующую функцию:

rmfield(S,'field') — возвращает структуру S с удаленным полем 'field' S.
rmfield(S,FIELDS) — возвращает структуру S с удаленными полями (более одного). FIELDS задается в виде массива символов или массива ячеек строк.
Пример:

```
» rmfield(man(2),'surname')
ans =
     name: 'Петр'
     date: 1959
   height: []
   weight: []
```

6.5. Массивы ячеек

6.5.1. Создание массивов ячеек

Массив ячеек — наиболее сложный тип данных в системе MATLAB. Это массив, элементами которого являются ячейки, содержащие любой тип массивов, включая

массивы ячеек. Отличительным атрибутом массивов ячеек является задание содержимого последних в фигурных скобках {}. Создавать массивы ячеек можно с помощью оператора присваивания.

Существует два способа присваивания данных отдельным ячейкам:

- индексацией ячеек;
- индексацией содержимого.

Рассмотрим первый способ. Для этого создадим файл-сценарий с именем **ce**:

```
A(1,1)={'Курить вредно!'};
A(1,2)={[1 2;3 4]};
A(2,1)={2+3i};
A(2,2)={0:0.1:1}
```

В этом примере задан массив ячеек с четырьмя элементами: строкой символов, матрицей, комплексным числом и одномерным массивом из 11 чисел. Теперь можно вызвать этот массив:

```
» ce
A =
   'Курить вредно!'    [2x2  double]
   [2.0000+ 3.0000i]   [1x11 double]
» A(1,1)
ans =
   'Курить вредно!'
» A(2,1)
ans =
   [2.0000+ 3.0000i]
```

Заметим, что к ячейкам такого массива можно обращаться с помощью индексирования, например A(1,1), A(2,1), и так далее.

При индексации содержимого массив ячеек задается следующим образом:

```
A{1,1}='Курить вредно!';
A{1,2}=[1 2;3 4];
A{2,1}=2+3i;
A{2,2}=0:0.1:1;
```

Теперь можно ознакомиться с созданным массивом ячеек в командном режиме:

```
» ce1
» A
ans =
```

```
    'Курить вредно!'    [2x2  double]
    [2.0000+ 3.0000i]   [1x11  double]
» A{1,1}
ans =
Курить вредно!
» A{2,1}
ans =
  2.0000 + 3.0000i
```

При серьезной работе с массивами записей и массивами ячеек полезно иметь дополнительную информацию о списках значений. Для получения такой информации следует выполнить команду **help list**.

6.5.2. Создание ячеек с помощью функции cell

Для создания массива ячеек может использоваться функция cell:

cell(N) — создает массив ячеек из N×N пустых матриц.
cell(M,N) или **cell([M,N])** — создает массив ячеек из M×N пустых матриц.
cell(M,N,P,...) или **cell([M N P ...])** — создает массив ячеек из M×N×P×... пустых матриц.
cell(size(A)) — создает массив ячеек из пустых матриц того же размера, что и массив **A**.

Следующие примеры поясняют применение данной функции:

```
» cell(2)
ans =
    []    []
    []    []
» C=cell(2,3)
C =
    []    []    []
    []    []    []
» C0=zeros(2,3)
C0 =
     0     0     0
     0     0     0
» cell(size(C0))
ans =
    []    []    []
    []    []    []
```

Созданные пустые ячейки можно заполнить, используя операции присваивания:

```
» C{1,1}=1;C{1,2}='Привет';C{2,1}='Hello';C{2,2}=[1,2; 3,4];
```

```
» C
C =
   [    1]  'Привет'          [ ]
   'Hello'  [2x2 double]      [ ]
```

6.5.3. Визуализация массивов ячеек — команды celldisp и cellplot

Для отображения массива ячеек **C** служит команда

```
celldisp(C)
```

Она дает рекурсивное отображение содержимого массива ячеек **C**. Например, для ранее созданного массива ячеек **A**:

```
» celldisp(A)
A{1,1} =
Курить вредно!

A{2,1} =
  2.0000 + 3.0000i

A{1,2} =
    1    2
    3    4

A{2,2} =
  Columns 1 through 7
       0   0.1000   0.2000   0.3000   0.4000   0.5000   0.6000
» C{1,1}=1;C{1,2}='Привет';C{2,1}='Hello';C{2,2}=[1 2; 3 4];
» C

C =

   [   1]    'Привет'         [ ]
   'Hello'   [2x2 double]     [ ]        Columns 8 through  11
   0.7000    0.8000           0.9000     1.0000
```

Для более наглядного графического представления массива ячеек может использоваться команда **cellplot**:

cellplot(C) — строит структуру массива ячеек **C**.
cellplot(C,'legend') — строит структуру массива ячеек **C** вместе с "легендой" — шкалой стилей представления данных.
H=cellplot(C) — возвращает вектор свойств.

На рис. 6.2 показан рисунок с представлением массива ячеек **A**, сформированного ранее.

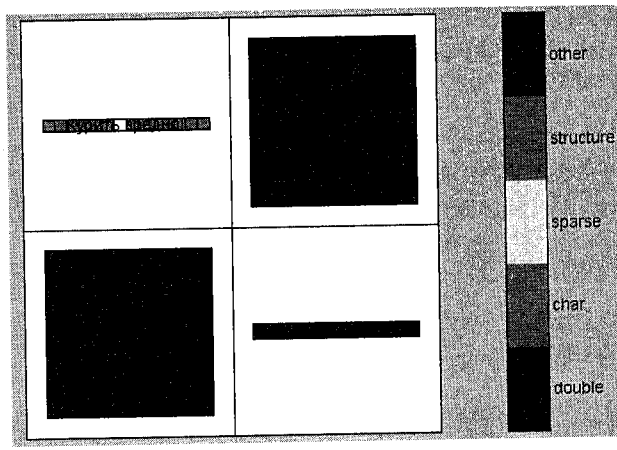


Рис. 6.2. Графическое представление массива ячеек с четырьмя ячейками

Как видно на рис. 6.2, ячейки массива представлены квадратами. Векторы и матрицы с численными данными представляются массивами красного цвета с прямоугольными ячейками, отдельные числа отображаются, выводятся и текстовые данные.

6.5.4. Создание массива символьных ячеек из массива строк — cellstr

Для создания из массива строк **S** массива символьных ячеек может использоваться функция **cellstr(S)**. Следующий пример поясняет ее применение:

```
» S={'Привет' 'дорогой' 'друг'};
» C=cellstr(S)
C =
   'Привет'   'дорогой'   'друг'
```

Это — еще один способ формирования массивов ячеек.

6.5.5. Присваивание с помощью функции deal

С помощью функции **deal** возможно множественное присваивание входных данных выходным:

[A,B,C,...]=deal(X,Y,Z,...) — обеспечивает последовательное присваивание входных данных выходным, т. е. **A=X**, **B=Y**, **C=Z**, ... и так далее.

[A,B,C,...]=deal(X) — присваивает единственный вход всем выходам, т.е. **A=X B=X**, **C=X**, ... и так далее.

Возможен ряд полезных применений функции **deal**:

[S.FIELD]=deal(X) — присваивает всем полям FIELD структуры S значения X Если S не существует, то нужно использовать конструкцию **[S(1:M).FIELD]=deal(X).**

[X{:}]=deal(A.FIELD) — копирует поля FIELD структуры A в массив ячеек X Если X не существует, следует использовать конструкцию **[X{1:M}]=deal(A.FIELD)**.

[A,B,C,...]=deal(X{:}) — копирует содержимое массива ячеек X в отдельные переменные A,B,C,...

[A,B,C,...]=deal(S.FIELD) — копирует содержимое поля FIELD в отдельные переменные A,B,C,...

Следующий пример иллюстрирует применение функции **deal**:

```
» [X,Y,Z]=deal(1,2+3i,'Привет!')
X =
     1
Y =
   2.0000 + 3.0000i
Z =
Привет!
» [X Y Z]=deal('Привет!')
X =
Привет!
Y =
Привет!
Z =
Привет!
```

6.5.6. Функция тестирования массива ячеек iscell

Ввиду обилия типов данных в системе MATLAB часто возникает необходимость в их тестировании. Для тестирования массивов ячеек может использоваться функция **iscell(C)**, которая возвращает логическое значение 1, если **C** — массив ячеек, и 0 в противном случае. Это поясняют следующие примеры:

```
» t=iscell(A)
t =
     1
» B=[1 2 3];
» iscell(B)
ans =
     0
```

6.5.7. Функции преобразования num2cell, cell2struct и struct2cell

При обработке сложных данных возникает необходимость в преобразовании типов данных. Ниже представлены такие функции, имеющие отношение к массивам ячеек:

num2cell(A) — преобразует массив чисел **A** в массив ячеек и возвращает последний. Возвращаемый массив имеет тот же размер, что и исходный **A**.

num2cell(A,DIM) — преобразует в массив ячеек только те элементы массива чисел **A**, которые соответствуют размеру, указанному DIM.

Примеры на применение данной функции:

```
» A=[1 2; 3 4; 5 6]
A =
   1   2
   3   4
   5   6
» num2cell(A,2)
ans =
   [1x2 double]
   [1x2 double]
   [1x2 double]
» num2cell(A,[1 2])
ans =
   [3x2 double]
```

cell2struct(C,FIELDS,DIM) — преобразует массив ячеек **C** в массив записей вдоль размерности DIM, сохраняя размер массива **C** по этой размерности в записи структуры. Пример преобразования:

```
» C={'Привет!',123,2+3i}
C =
   'Привет!'   [123]   [2.0000+ 3.0000i]
» f={'name','number','complex'};
» S=cell2struct(C,f,2)
S =
      name: 'Привет!'
    number: 123
   complex: 2.0000+ 3.0000i
```

struct2cell — преобразует массив записей **S** в массив ячеек с p-полями, так что возвращаемый массив будет иметь размер p×m×n. Если массив записей многомерный, то возвращаемый массив будет иметь размер, равный [p size(S)]. Пример такого преобразования приводится ниже:

```
» C=struct2cell(S)
C =
   'Привет!'
```

```
[      123]
[2.0000+ 3.0000i]
```

6.5.8. Переменные varargin и varargout

Для упрощения записи аргументов функций их можно представить списком, который определяет специальная переменная **varargin**, являющаяся массивом ячеек. Она должна записываться прописными буквами и может включать в себя как аргументы, так и опции функций. Так, в приведенных ниже примерах

```
function myplot(x,varargin)
plot(x,varargin{:}) function [s,varargout] = mysize(x)
    nout = max(nargout,1)-1;
    s = size(x);
    for i=1:nout, varargout(i) = {s(i)}; end
```

эта переменная вбирает в себя все входные параметры и опции, начиная со второго аргумента. При обращении к данной функции

```
myplot(sin(0:.1:1),'color',[.5 .7 .3],'linestyle',':')
```

varargin представляет массив ячеек размера 1×4, включающий в себя значения 'color', [.5 .7 .3], 'linestyle' и ':'.

Аналогично **varaкgin** переменная **varargout** объединяет любое число выходных аргументов в массив ячеек. Эта переменная, кстати, как и **varargin**, должна быть последней в списке аргументов. Обычно эта переменная не создается при вызове функций. Приведенный ниже пример поясняет ее создание с помощью цикла:

```
function [s,varargout] = mysize(x)
    nout = max(nargout,1)-1;
    s = size(x);
    for i=1:nout,
        varargout(i) = {s(i)};
    end
```

Более подробно циклы будут рассмотрены в дальнейшем описании. В данном случае цикл использован для объединения всех аргументов, начиная со второго, в значение переменной **varargout**..

6.5.9. Многомерные массивы ячеек

С помощью функции **cat** можно формировать многомерные массивы ячеек. Например, трехмерный массив **C** (M-файл с именем ce2.m) формируется следующим образом:

```
A{1,1}='Курить вредно!';
A{1,2}=[1 2;3 4];
A{2,1}=2+3i;A{2,2}=0:0.1:1;
B{1,1}='Пить тоже вредно!';
B{1,2}=[1 2 3 4];
B{2,1}=2;
B{2,2}=2*pi;
C=cat(3,A,B);
```

Теперь можно просмотреть данный массив, имеющий две страницы:

```
» ce2
» C
C(:,:,1) =
   'Курить вредно!'        [2x2  double]
   [2.0000+ 3.0000i]       [1x11 double]

C(:,:,2) =
   'Пить тоже вредно!'     [1x4 double]
   [                  2]   [     6.2832]
```

Этот многомерный массив можно просмотреть с помощью команды **cellplot(C)**. Полученное при этом изображение показано на рис. 6.3, где многомерный массив отображается как стопка страниц.

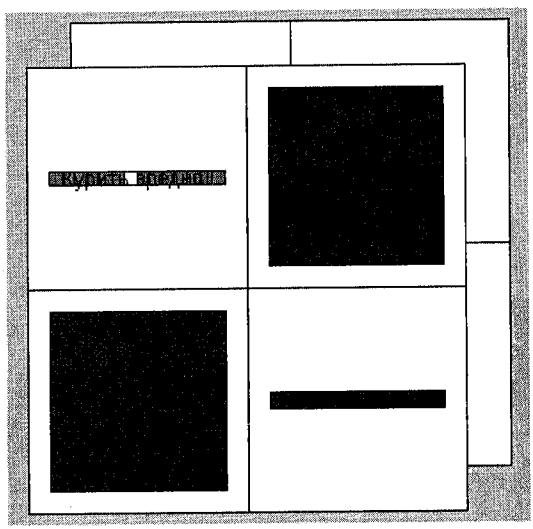


Рис. 6.3. Отображение трехмерного массива ячеек командой **cellplot**

Доступ к ячейкам многомерных массивов очевиден и поясняется следующими примерами:

```
» C(1,1,1)
ans =
   'Курить вредно!'
» C(1,1,2)
ans =
   'Пить тоже вредно!'
```

6.5.10. Вложенные массивы ячеек

Содержимым ячейки массива ячеек может быть, в свою очередь, произвольный массив ячеек. Таким образом, возможно создание вложенных массивов ячеек, пожалуй, самого сложного типа данных. В следующем примере показано формирование массива ячеек **A** с вложенным в него массивом **B** (он был создан в примере выше):

```
» clear A;
» A(1,1)={{magic(3),{'Hello!'}}};
» A(1,2)={B};
» cellplot(A)
» A
ans =
   {1x2 cell}   {2x2 cell}
» A{1}
ans =
   [3x3 double]   {1x1 cell}
» A{2}
ans =
   'Пить тоже вредно!'   [1x4  double]
   [                  2]   [      6.2832]
» cellplot(A)
```

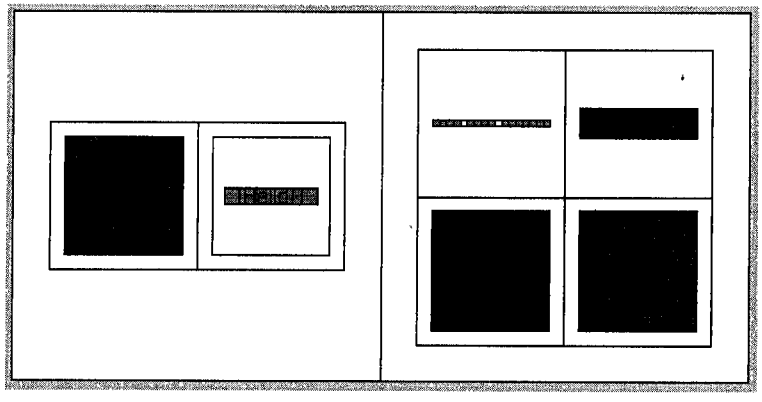


Рис. 6.4. Графическое представление массива с вложенным в него другим массивом

В данном случае вложенный массив **B** отображается полностью как часть массива **A**.

6.6. Управляющие структуры

Помимо программ с **линейной структурой**, инструкции которых исполняются строго по порядку, существует множество программ, структура которых **нелинейна**. При этом ветви программ могут выполняться в зависимости от определенных условий, иногда с конечным числом повторений — циклов, иногда при циклах, завершаемых при выполнении заданного условия. Практически любая серьезная программа имеет нелинейную структуру. Для создания таких программ необходимы специальные управляющие структуры. Они имеются в любом языке программирования и, в частности, в MATLAB.

6.6.1. Диалоговый ввод — функция input

Приведем простой пример диалоговой программы, которую легко поймут приверженцы доброго старого Бейсика:

```
% Вычисление длины окружности с диалоговым вводом радиуса
r=0;
while r>=0,
  r=input('Введите радиус окружности r=');
  if r>=0 disp(' Длина окружности l='); disp(2*pi*r), end
 end
```

Эта программа служит для многократного вычисления длины окружности по вводимому пользователем значению радиуса r. Обратите внимание на то, что здесь мы впервые показываем пример организации простейшего диалога. Он реализован с помощью команды

```
r=input('Введите радиус окружности r=');
```

При выполнении этой команды вначале выводится запрос в виде строки, затем происходит останов работы программы и ожидается ввод значения радиуса r. Ввод как обычно подтверждается нажатием клавиши Enter, после чего введенное число присваивается значению переменной r. Следующая строка

```
if r>=0 disp(' Длина окружности l='); disp(2*pi*r), end
```

с помощью команды **disp** при r>=0 выводит надпись "Длина окружности l=" и вычисленное значение длины окружности. Она представляет собой одну из наиболее простых управляющих структур типа **if-end**. В данном случае она нужна для остановки вычислений, если вводится отрицательное значение r (прием, который любят начинающие программисты).

Приведенные строки включены в управляющую структуру **while-end**. Это необходимо для циклического повторения вычислений с вводом значений r. Пока r>=0, цикл повторяется. Но стоит задать r<0, вычисление длины окружности перестает выполняться, а цикл завершается.

Если данная программа записана в виде М-файла **circ.m**, то работа с ней будет выглядеть следующим образом:

```
» circ
Введите радиус окружности R=1
 Длина окружности l=
   6.2832
Введите радиус окружности R=2
 Длина окружности l=
   12.5664
Введите радиус окружности R=-1
»
```

Итак, на примере даже простой программы мы видим пользу применения управляющих структур типа **if-end** и **while-end**, а также функций диалогового ввода **input('String')** и вывода **disp**. Обратите внимание на завершение работы программы при вводе любого отрицательного числа для радиуса окружности.

Функция **input** может использоваться и для ввода произвольных строковых выражений. При этом она задается в виде:

```
input('Комментарий','s')
```

При выполнении этой функции она останавливает вычисления и ожидает ввода строкового комментария. После ввода возвращается набранная строка. Это иллюстрирует следующий пример:

```
» S=input('Введите выражение ','s')
Введите выражение 2*sin(1)
S =
2*sin(1)
» eval(S)
ans =
   1.6829
```

Обратите внимание на то, что функция **eval** позволяет вычислить выражение, заданное функцией **input** в символьном виде. Вообще говоря, возможность ввода любого символьного выражения в сочетании с присущими языку программирования MATLAB управляющими структурами открывает путь к созданию диалоговых программ любой сложности.

6.6.2. Условный оператор if-elseif-else-end

Условный оператор if в общем виде записывается следующим образом:

```
if Условие
   Инструкции_1
    elseif Условие
      Инструкции_2
    else
      Инструкции_3
end
```

Эта конструкция допускает несколько частных вариантов. В простейшем, типа **if-end**

if Условие Инструкции **end**

пока Условие возвращает логическое значение 1 (то есть верно), выполняются Инструкции, составляющие тело структуры **if-end**. При этом оператор **end** указывает на конец перечня Инструкций. Инструкции в списке разделяются оператором , (запятая) или ; (точка с запятой). Если Условие не выполняется (дает логическое значение 0), то Инструкции также не выполняются.

Еще одна конструкция —

if Условие Инструкции_1 **else** Инструкции_2 **end**

выполняет Инструкции_1, если выполняется Условие, или Инструкции_2 в противном случае.

Условия записываются в виде:

Выражение_1 Оператор_отношения Выражение_2

причем в качестве Операторов_отношения используются следующие операторы: ==, <, >, <=, >= или ~=. Все эти операторы представляют собой двойные символы без пробела между ними.

Мы уже неоднократно показывали применение этой общеизвестной управляющей структуры в программных модулях. Читателю предлагается опробовать собственные варианты программ с условным оператором.

6.6.3. Циклы типа for-end

Циклы типа **for-end** обычно используются для организации вычислений с заданным числом повторяющихся циклов. Конструкция такого цикла имеет вид:

for var=Выражение, Инструкция, ..., Инструкция **end**

Выражение чаще всего записывается в виде s:d:e, где s — начальное значение переменной цикла variable, d — приращение этой переменной и e — конечное значение управляющей переменной, при достижении которого цикл завершается. Возможна и запись в виде s:e (в этом случае d=1). Список выполняемых в цикле инструкций завершается оператором **end**.

Следующие примеры поясняют применение цикла для получения квадратов значений переменной цикла:

```
» for i=1:5 i^2, end;
ans =
     1
ans =
     4
ans =
     9
ans =
    16
ans =
    25
» for x=0:.25:1 x^2, end;
ans =
     0
ans =
    0.0625
ans =
    0.2500
ans =
    0.5625
ans =
     1
```

Оператор **break** может использоваться для досрочного выполнения цикла. Как только он встречается в программе, цикл прерывается. Возможны циклы в цикле, например:

```
for i=1:3
  for j=1:3
    A(i,j)=i+j;
  end
end
```

В результате исполнения этого цикла (файл for2.m) формируется матрица **A**:

```
» for2
» A
A =
   2   3   4
   3   4   5
   4   5   6
»
```

Следует отметить, что формирование матриц с помощью оператора : (двоеточие) обычно занимает намного меньше времени, чем с помощью цикла. Однако применение цикла нередко оказывается более наглядным и понятным.

MATLAB допускает использование в качестве выражения переменной цикла массив **A** размера m×n. При этом цикл выполняется столько раз, сколько столбцов в массиве **A**, и на каждом цикле переменная var представляет вектор, соответствующий текущему столбцу вектора **A**:

```
» A=[1 2 3;4 5 6]
A =
   1   2   3
   4   5   6
» for var=A; var, end
var =
   1
   4
var =
   2
   5
var =
   3
   6
```

6.6.4. Циклы типа while

Цикл типа **while** выполняется до тех пор, пока выполняется Условие:

```
while Условие
   Инструкции
end
```

С целью прекращения выполнения цикла можно использовать оператор **break**. Пример применения цикла **while** уже приводился.

6.6.5. Конструкция переключателя switch-case-otherwise-end

Для осуществления множественного выбора (или ветвления) используется конструкция с переключателем типа **switch**:

```
switch switch_Выражение
      case case_Выражение
        Список_инструкций
       case {case_Выражение1, case_Выражение2, case_Выражение3,...}
        Список_инструкций
       ...
      otherwise,
       Список_инструкций
end
```

Если выражение после заголовка **switch** имеет значение логической 1, то выполняется блок операторов **case**, в противном случае — список инструкций после оператора **otherwise**. Если выполнение блока **case** разрешено, то выполняются те списки инструкций, для которых case_Выражение совпадает со switch_Выражением. Обратите внимание на то, что case_Выражение может быть числом, константой, переменной или вектором ячеек.

Поясним применение оператора **switch** на примере M-файла sw1.m:

```
switch var
case {1,2,3}
  disp('Первый квартал')
case {4,5,6}
  disp('Второй квартал')
case {7,8,9}
  disp('Третий квартал')
case {10,11,12}
  disp('Четвертый квартал')
otherwise
  disp('Ошибка в задании')
end
```

Эта программа в ответ на значения переменной var — номера месяца — вычисляет, к какому кварталу относится заданный месяц, и выводит сообщение:

```
» var=2;
» sw1
Первый квартал
» var=4;sw1
Второй квартал
» var=7;sw1
```

```
Третий квартал
» var=12;sw1
Четвертый квартал
» var=-1;sw1
Ошибка в задании
```

6.6.6. Создание паузы в вычислениях — команда pause

Для остановки программы используется оператор **pause**. Он используется в следующих формах:

pause — останавливает вычисления до нажатия любой клавиши.
pause(N) — останавливает вычисления на N секунд.
pause on — включает режим создания пауз.
pause off — выключает режим создания пауз.

Следующий пример поясняет применение команды **pause**:

```
for i=1:20;
x = rand(1,40);
y = rand(1,40);
z = sin(x.*y);
tri = delaunay(x,y);
trisurf(tri,x,y,z)
pause(1);
end
```

Команда **pause(1)** здесь обеспечивает показ 20 рисунков — построений 3D-поверхностей из треугольных окрашенных областей со случайными параметрами.

6.7. Средства объектно-ориентированного программирования

6.7.1. Понятие об объектно-ориентированном программировании

Мы уже много раз упоминали различные объекты языка программирования системы MATLAB. Это является одним из признаков объектно-ориентированного программирования системы, причем, чисто внешним. В основе объектно-ориентированного программирования лежат три основных положения:

Инкапсуляция — объединение данных программ в передачу данных через входные и выходные параметры функций. В результате появляется новый элемент программирования — **объект**.

Наследование — возможность создания родительских объектов и новых дочерних объектов, наследующих свойства родительских объектов. Возможно также множественное наследование, при котором класс наследует свойства ряда родительских объектов и своей собственной структуры. На наследовании основаны система задания типов данных, дескрипторная графика и многие другие приемы программирования. Примеры наследования мы уже неоднократно отмечали.

Полиморфизм — присвоение некоторому действию одного имени, которое в дальнейшем используется по всей цепочке создаваемых объектов сверху донизу, причем каждый объект выполняет это действие присущим ему способом.

В дополнение к этим положениям объектно-ориентированное программирование в MATLAB допускает **агрегирование** объектов, т. е. объединение частей объектов или ряда объектов в одно целое.

Объект можно определить как некоторую структуру, принадлежащую к определенному **классу**. В MATLAB определены следующие пять основных классов объектов:

double — числовые массивы с элементами-числами двойной точности;
sparse — двумерные числовые или комплексные разреженные матрицы;
char — массивы символов;
struct — массивы записей (структурные);
cell — массивы ячеек.

С объектами этих классов мы многократно встречались, особо не оговаривая их принадлежность к объектно-ориентированному программированию. Для MATLAB вообще характерно, что никакие классы объектов (в том числе заново создаваемые) не требуют объявления. Например, создавая переменную **name='Иван'**, мы автоматически получаем объект в виде переменной name класса **char**. Таким образом, для переменных принадлежность к тому или иному классу определяется их значением.

Для создания новых классов объектов служат **конструкторы классов**. По существу это М-файлы, имя которых совпадает с именем классов @Имя_класса, но без символа @. Этим символом помечаются поддиректории системы MATLAB, в которых имеются конструкторы классов. Множество таких директорий с примерами конструкторов классов вы найдете в поддиректориях MATLAB и TOOLBOX.

В качестве примера рассмотрим поддиректорию @SYM в директории TOOLBOOKS/SYMBOLIC. В этой поддиректории можно найти конструкторы для более чем сотни объектов пакета символьной математики. К примеру, конструктор функции, вычисляющей арктангенс, выглядит следующим образом:

```
» help @sym/atan.m
 ATAN   Symbolic inverse tangent.
» type @sym/atan.m
function Y = atan(X)
%ATAN   Symbolic inverse tangent.
%   Copyright (c) 1993-98 by The MathWorks, Inc.
%   $Revision: 1.10 $  $Date: 1997/11/29 01:05:16 $
Y = maple('map','atan',X);
```

В данном случае для конструирования нужного объекта используется функция maple, дающая вход в ядро системы символьной математики Maple V R4, которое поставляется в составе системы MATLAB по лицензии фирмы MapleSoft Inc. Этот пример, кстати, наглядно показывает, что пользователь системы MATLAB может существенно расширить число объектов класса sym, поскольку ядро системы Maple V содержит намного больше определений, чем пакет символьной математики системы MATLAB. Для создания новых классов объектов служит функция **class**, описанная ниже.

Итак, объектно-ориентированное программирование — это как бы кинжал, закрепленный на вашем поясе. Вы можете и не воспользоваться этим оружием, ощущая при этом его значимость и ценя красоту. Но в альтернативном варианте вы можете использовать его во время ежедневной трапезы в качестве столь необходимого столового ножа. В первом случае вы выступаете в качестве обычного пользователя, а во втором — программиста-профессионала.

Пакеты прикладных программ системы MATLAB позволяют разработчикам с большим успехом использовать возможности объектно-ориентированного программирования путем создания новых классов объектов и новых объектов. М-файлы системы представляют собой массу наглядных примеров объектно-ориентированного программирования на языке MATLAB. Это дает основание ограничиться справочным описанием основных средств такого программирования с приведением минимума простых примеров.

6.7.2. Создание класса или объекта с помощью функции class

Для создания класса объектов или объектов и их идентификации служит функция **class**. Формы ее применения представлены ниже:

class(OBJ) — возвращает класс указанного объекта OBJ. Типы классов были указаны выше. Дополнительно имеет класс объектов пользователя — <class_name>.

class(S,'class_name') — создает объект класса 'class_name' с использованием структуры S.

OBJ=class(S,'class_name',PARENT1,PARENT2,...) — создает объект класса 'class_name' на базе структуры S и родительских объектов PARENT1, PARENT2, ... При

этом создаваемый объект наследует структуру и поля родительских объектов. Объекту OBJ в данном случае присуще **множественное наследование**.

Обратите внимание на то, что эта функция обычно используется в составе М-файлов конструкторов классов объектов.

6.7.3. Контроль за отношением объекта к заданному классу – функция isa

Для контроля принадлежности заданного объекта к некоторому классу служит функция **isa**.

isa(OBJ,'Имя_класса') — возвращает логическую 1, если OBJ принадлежит классу с указанным именем. Примеры применения этой функции:

```
» X=[1 2 3];
» isa(X,'char')
ans =
   0
» isa(X,'double')
ans =
   1
```

6.7.4. Другие функции объектно-ориентированного программирования

Для получения списка методов данного класса объектов используется функция:

M=methods('class_name') — возвращает массив ячеек с указанием методов, относящихся к заданному классу объектов. Например:

```
» methods char
Methods for class char:
delete diff   int
```

Следующие две функции могут использоваться только внутри конструкторов классов:

inferiorto('CLASS1','CLASS2',...) и **superiorto('CLASS1','CLASS2',...)**

Они определяют низший и высший приоритеты классов по отношению к классу конструктора.

Любой оператор в системе MATLAB можно **переопределить** путем задания М-файла с новым именем в соответствующем каталоге классов. В частности, в главе 5 отмечалось, что все арифметические операторы имеют представления в виде соответствующих функций.

При написании книги не ставилась цель детального знакомства с техникой объектно-ориентированного программирования. Дополнительные сведения имеются в книге [33], содержащей перевод фирменного описания раздела по объектно-ориентированному программированию. Поэтому ограничимся приведенным выше справочным описанием его средств.

6.8. Отладка М-файлов в командном режиме

6.8.1. Общие замечания по отладке М-файлов

Вряд ли существует программа с длиной более десятка строк, которая после запуска сразу бы выдала верный результат. Как правило, любую программу надо отлаживать в интерактивном режиме, запуская и анализируя полученные при каждой модификации результаты. Основным средством отладки М-файлов в системе MATLAB является встроенный редактор/отладчик (M-File Editor/Debuger) с современным графическим интерфейсом. Работа с ним была описана в первой главе. Однако MATLAB предусматривает основные возможности отладки и в командном режиме. Именно они и рассматриваются в данном разделе.

Вообще говоря, отладка программ — процесс сугубо индивидуальный и творческий. Большинство пользователей средней квалификации обычно отлаживают программы, не обращаясь к специальным средствам отладки, требующим дополнительного времени и навыков при их освоении. Если алгоритм решения задачи достаточно прост, то после нескольких модернизаций программы ее удается довести до нужной "кондиции", т.е. заставить работать корректно.

Для этого часто бывает достаточным ввести в программу режим просмотра промежуточных вычислений, разблокировав их вывод снятием операторов ; (точка с запятой), или введя дополнительные переменные, значения которых отражают ход вычислений. После отладки можно вновь ввести блокирующие вывод операторы и убрать указанные переменные.

Серьезная необходимость в применении средств отладки возникает при отладке сложных программ опытными программистами, которые наверняка имеют практику работы с универсальными языками программирования с использованием средств отладки.

6.8.2. Команды отладки программ keyboard, return и dbquit

Для перехода в режим отладки в командном режиме в М-файл следует включить команду **keyboard**. Ее можно запустить и в командном режиме:

```
» keyboard
```

```
K» type sw1

switch var
case {1,2,3}
  disp('Первый квартал')
case {4,5,6}
  disp('Второй квартал')
case {7,8,9}
  disp('Третий квартал')
case {10,11,12}
  disp('Четвертый квартал')
otherwise
  disp('Ошибка в задании')
end

K» return
»
```

Признаком перехода в режим отладки становится появление комбинированного символа **K»**. Он меняется на символ **»** после возврата командой **return** в обычный командный режим работы. То же самое происходит при использовании команды **dbquit**, прекращающей режим отладки.

6.8.3. Вывод пронумерованного листинга M-файла — команда dbtype

Один из способов отладки M-файлов — размещение в них точек останова. Однако в командном режиме нельзя задать установку таких точек с помощью курсора мыши (как в отладчике Windows). Поэтому необходимо иметь листинг программы с пронумерованными строчками. Он создается с помощью команды **dbtype**, действие которой иллюстрирует следующий пример:

```
» keyboard
K» dbtype sw1

1   switch var
2   case {1,2,3}
3     disp('Первый квартал')
4   case {4,5,6}
5     disp('Второй квартал')
6   case {7,8,9}
7     disp('Третий квартал')
8   case {10,11,12}
9     disp('Четвертый квартал')
10  otherwise
```

```
11    disp('Ошибка в задании')
12  end
K»
```

6.8.4. Установка, удаление и просмотр контрольных точек — dbstop, dbclear и dbstatus

Для установки контрольных точек в тестируемый М-файл используются следующие команды:

dbstop in M-file at lineto — установить контрольную точку в заданной строке;
dbstop in M-file at subfun — установить контрольную точку в подфункции;
dbstop in M-file — установить контрольную точку в М-файле;
dbstop if error — установить контрольную точку при сообщении об ошибке;
dbstop if warning — установить контрольную точку при предупреждении;
dbstop if NaN — установить контрольную точку при результате NaN;
dbstop if infnan — установить контрольную точку при результате infnan.

Можно использовать упрощенный ввод этих команд без использования слов **in, at** и **if**. При установке контрольной точки она появляется в окне редактора/отладчика М-файлов.

Для удаления контрольных точек используется команда **dbclear** в том же синтаксисе, что и **dbstop**, например:

dbclear M-file at lineto — удалить контрольную точку в заданной строке.

Команда **dbstatus** выводит список установленных в данной сессии контрольных точек, например:

```
K» dbstatus
Breakpoint for C:\MATLAB\bin\demo1.m is on line 2.
Breakpoint for C:\MATLAB\bin\sd.m is on line 3.
```

6.8.5. Управление исполнением М-файла — команды dbstep и dbcont

После установки контрольных точек начинается собственно процесс тестирования М-файла. Он заключается в исполнении одного или нескольких шагов программы с возможностью просмотра содержимого рабочей области, т. е. значений переменных, меняющихся в ходе выполнения программы. Для пошагового выполнения программы используется команда **dbstep** в следующих формах:

dbstep — выполнение очередного шага;
dbstep nlines — выполнение заданного числа линий программы
dbstep in — эта форма позволяет пользователю перейти к первой исполняемой линии в функции, вызванной из М-файла, где следующая исполняемая линия вызывает другую функцию М-файла.

Для перехода от одной остановки программы к другой используется команда **dbcont**.

6.8.6. Работа с рабочей областью — команды dbdown и dbup

В точках останова пользователь имеет возможность просмотреть состояние рабочей области с помощью ранее описанных команд **who** и **whos**. Кроме того, для перехода из М-файла в рабочую область с перемещениями по рабочим областям вверх или вниз используются команды:

dbdown — перемещение в стеке вызываемых функций сверху вниз;
dbup — перемещение в стеке вызываемых функций снизу вверх.

Находясь в рабочей области, можно не только просматривать значения переменных, но и менять их в ходе отладки программы. С помощью команды **dbstack** удобно просматривать стек функций. По завершении отладки используется команда **dbquit**.

В заключение еще раз обращаем внимание читателя на то, что все возможности отладки реализованы в редакторе/отладчике М-файлов, который характеризуется удобным графическим интерфейсом и средствами визуализации отладки программ. К ним относятся возможность выделения различными цветами элементов М-файла (ключевых слов, переменных, комментарий и так далее), наглядное представление контрольных точек, простота их установки и так далее. В этом отношении описанные выше приемы отладки в командном режиме выглядят несколько архаично. Скорее всего они ориентированы на пользователей, привыкших к командному режиму работы с системой.

6.8.7. Профилирование М-файлов — команда profile

Вообще говоря, достижение работоспособности программы — лишь один из этапов ее отладки. Не менее важным вопросом является оптимизация программы по минимуму времени исполнения или по минимуму объема кодов. Современные ПК, в которых используется система MATLAB, имеют достаточные резервы памяти, так что размеры программы, как правило, не имеют особого значения. Намного важнее проблема оптимизации программы в части быстродействия.

Оценка времени исполнения отдельных частей программы называется ее **профилированием**. Для выполнения такой процедуры служит команда **profile**, имеющая ряд опций:

profile fun — запуск профилирования для функции fun.
profile report — вывод отчета по профилированию.
profile plot — графическое представление результатов профилирования в виде диаграммы Парето.
profile filename — профилирование файлов с заданным именем и путем.
profile report N — вывод отчета по профилированию заданных N линий.
profile report frac — выводит отчет по профилированию тех строк, относительная доля исполнения которых в общем времени исполнения составляет величину frac (от 0.0 до 1.0).
profile on — включение профилирования.
profile off — выключение профилирования.
profile reset — выключение профилирования с уничтожением всех данных.
INFO= profile — возвращает структуру со следующими полями:
file — полный путь профилируемого файла.
interval — интервалы времени в секундах.
count — вектор измерений.
state — состояние 'on' (включен) или 'off' (выключен) профилировщика.

Ниже приводится пример профилирования M-файла ellipj (эллиптическая функция Якоби):

```
» profile on
» profile ellipj
» ellipj([0:0.01:1],0.5);
» profile report
Total time in "C:\MATLAB\toolbox\matlab\specfun\ellipj.m": 0.16 seconds
100% of the total time was spent on lines:
    [96 97 86]
        85:    if ~isempty(in)
0.01s,  6%  86:      phin(i,in) = 0.5 * ...
        87:      (asin(c(i+1,in).*sin(rem(phin(i+1,in),2*pi))./a(i+1,in))
        95: m1 = find(m==1);
0.11s, 69%  96: sn(m1) = tanh(u(m1));
0.04s, 25%  97: cn(m1) = sech(u(m1));
        98: dn(m1) = sech(u(m1));
» INFO=profile
INFO =
    file: 'C:\MATLAB\toolbox\matlab\specfun\ellipj.m'
    interval: 0.0100
    count: [98x1  double]
    state: 'off'
» profile plot
```

Нетрудно заметить, что при профилировании выводятся номера строк программы, у которых время исполнения превосходит 0.01 с. С использованием этого интервала и оценивается время исполнения программного кода. Последняя команда выводит графическую диаграмму профилирования, показанную на рис. 6.5.

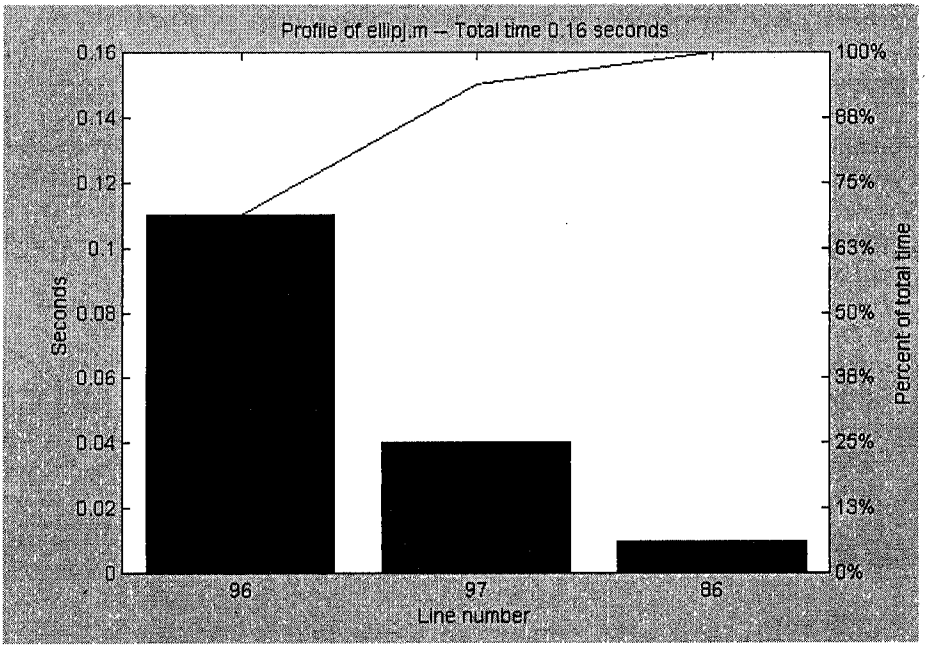


Рис. 6.5. Графическое представление результатов профилирования

При графическом представлении профилирования по горизонтальной оси указываются номера строк, а по вертикальной — время их выполнения. Сначала показываются строки с наибольшим временем выполнения. Таким образом, программист, отлаживающий работу программы, имеет возможность наглядно оценить, где именно находятся критические по быстродействию фрагменты.

6.8.8. Создание итогового отчета — команда profsumm

Для создания суммарного отчета по профилированию служит команда **profsumm**, которая используется в нескольких формах:

profsumm — вывод полного отчета по профилируемому М-файлу. Выводятся данные по времени выполнения для строк, суммарное время выполнения которых составляет 95% от общего времени, или же 10 строк программного кода..

profsumm(FRACTION) — выводит часть отчета для строк, относительное время выполнения которых составляет FRACTION (от 0.0 до 1.0) от общего времени выполнения файла.

profsumm(N) — выводится отчет по N строкам с максимальным временем исполнения.

profsumm(STR) — выводит отчет только по тем строкам, в которых встречается в строке выражение STR.

profsumm(INFO), **profsumm(INFO, FRACTION)**, **profsumm(INFO, N)** и **profsumm(INFO, STR)** — выводят итоговый отчет по строкам, заданным массивом **INFO**.

Пример применения команды **profsumm**:

```
» profile erfcore
» z = erf(0:.01:100);
» profsumm, profile done
Total time in "C:\MATLAB\toolbox\matlab\specfun\erfcore.m": 0.16 seconds

94% of the total time was spent on lines:
    [99 109 108 97 95 94 92 86 71 48]

        47: %
0.01s,  6%  48:    k = find((abs(x) > xbreak) &  (abs(x) <= 4.));
        49:    if ~isempty(k)

        70:          del = (y-z).*(y+z);
0.01s,  6%  71:              result(k) = exp(-z.*z) .* exp(-del) .* result(k)
        72:          end

        85:
0.01s,  6%  86:          y = abs(x(k));
        87:          z = 1 ./ (y .* y);

        91:          xnum = (xnum + p(i)) .* z;
0.01s,  6%  92:              xden = (xden + q(i)) .* z;
        93:          end
0.01s,  6%  94:          result(k) = z .* (xnum + p(5)) ./ (xden + q(5));
0.01s,  6%  95:          result(k) = (1/sqrt(pi) -  result(k)) ./ y;
        96:          if jint ~= 2
0.01s,  6%  97:              z = fix(y*16)/16;
        98:          del = (y-z).*(y+z);
0.06s, 38%  99:              result(k) = exp(-z.*z) .* exp(-del) .* result(k)
       100:          k = find(~isfinite(result));

       107:    if jint == 0
0.01s,  6% 108:          k = find(x > xbreak);
0.01s,  6% 109:          result(k) = (0.5 - result(k)) + 0.5;
       110:          k = find(x < -xbreak);
```

6.8.9. Построение диаграмм Парето — команда pareto

Команда **profile plot** использует для построения графическую команду **pareto.** Диаграммы Парето представляет собой столбцы, построенные в порядке убывания отображаемых значений. С другими возможностями команды pareto можно ознакомиться, запустив команду **help pareto**:

pareto(Y,NAMES) — строит диаграмму Парето с пометкой каждого столбца значений вектора **Y** соответствующим именем в векторе NAMES.

pareto(Y,X) — строит диаграмму Парето для значений вектора **Y** в зависимости от значений вектора X.

pareto(Y) — строит диаграмму Парето для значений вектора **Y** в зависимости от индексов его элементов.

[H,AX]=pareto(...) — возвращает вектор дескрипторов графических объектов диаграммы H и их осей AX.

Пример построения диаграммы Парето был рассмотрен выше — см. рис. 6.5.

Глава 7.
Численные методы и обработка данных

В этой главе описываются функции системы MATLAB, предназначенные для реализации алгоритмов типовых численных методов решения прикладных задач и обработки данных. Наряду с базовыми операциями решения систем линейных и нелинейных уравнений рассмотрены функции вычисления конечных разностей, численного дифференцирования, численного интегрирования, триангуляции, аппроксимации Лапласиана и, наконец, прямого и обратного преобразования Фурье. Отдельные разделы посвящены работе с полиномами, задачам интерполяции и аппроксимации данных и численным методам решения обыкновенных дифференциальных уравнений.

7.1. Решение систем линейных уравнений

Решение систем линейных уравнений относится к самой массовой области применения матричных методов, описанных в главе 5. В этом разделе вы найдете ответы на вопросы, каким образом применяются указанные методы и какие дополнительные функции имеет система MATLAB для решения систем линейных уравнений.

7.1.1. Элементарные средства решения систем линейных уравнений

Как известно, обычная система линейных уравнений имеет вид:

$$a_{1,1}x_1 + a_{1,2}x_2 + \ldots + a_{1,n}x_n = b_1$$
$$a_{2,1}x_1 + a_{2,2}x_2 + \ldots + a_{2,n}x_n = b_2$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$a_{n,1}x_1 + a_{n,2}x_n + \ldots + a_{n,n}x_n = b_n$$

Здесь $a_{1,1}$, $a_{1,2}$,...,$a_{n,n}$ — коэффициенты, образующие матрицу **A** и могущие иметь действительные или комплексные значения, x_1, x_2,..., x_n — неизвестные, образующие вектор **X**, и b_1, b_2,...,b_n — свободные члены (действительные или комплексные), образующие вектор **B**. Эта система может быть представлена в матричном виде как **AX=B**, где **A** — матрица коэффициентов уравнений, **X** — искомый вектор неизвестных и **B** — вектор свободных членов. В зависимости от вида матрицы **A** и ее характерных особенностей MATLAB позволяет реализовать различные методы решения.

Для реализации различных алгоритмов решения систем линейных уравнений и связанных с ними матричных операций применяются следующие операторы: **+ - * / \ ^ '**.. Как отмечалось ранее, MATLAB имеет два различных типа арифметических операций. Матричные арифметические операции определяются **правилами линейной алгебры**.

Арифметические операции сложения и вычитания над массивами выполняются поэлементно. Знак точки . отличает операции над массивом от матричных операций. Однако, поскольку операции сложения и вычитания одинаковы для матрицы и массива, знаки **.+** и **.-** не используются. Рассмотрим другие операторы и выполняемые ими операции.

***** - матричное умножение.
C = A*B - линейное алгебраическое произведение матриц **A** и **B**:

$$C(i,j) = \sum_{k=1}^{n} A(i,k) * B(k,j).$$

Для случая не скалярных **A** и **B** число столбцов матрицы **A** должно равняться числу строк матрицы **B**. Скаляр может умножаться на матрицу любого размера.

/ — слеш, или правое деление. Выражение **X=B/A** дает решение ряда систем линейных уравнений **AX=B**, где **A** — матрица размера m×n и **B** — матрица размера n×k.

**** — обратный слеш, или левое деление. Выражение **X=B\A** дает решение ряда систем линейных уравнений **XA=B**, где **A** — матрица размера m×n и **B** — матрица размера n×k. Если **A** — квадратная матрица, то **A\B** примерно то же самое, что и **inv(A)*B**, иначе возможны варианты, отмеченные ниже.

Если **A** — матрица размера n×n и **B** — вектор-столбец с n компонентами или матрица с несколько подобными столбцами, тогда **X = A\B** — решение уравнения **AX = B**, которое находится хорошо известным методом исключения Гаусса.

Если **A** — матрица размера m×n и m ~= n, а **B** представляет собой вектор-столбец с m компонентами или матрицу с несколькими такими столбцами, тогда система оказывается недоопределенной или переопределенной и решается на основе минимизации второй нормы невязок.

Ранг k матрицы **A** находится на основе **QR-** разложения с выбором ведущего элемента. Полученное решение **X** будет иметь самое большее k ненулевых компонентов на столбец. Если k<n, то решение, как правило, не будет совпадать с **pinv(A)*B**, которое имеет наименьшую норму невязок **||X||**.

^ — степень матрицы. **X^p** — это **X** в степени p, если p-скаляр. Если p — целое число, то степень матрицы вычисляется путем перемножения **X** себя на себя p раз. Если p — целое отрицательное число, то **X** сначала инвертируется. Для других значений p вычисляются собственные значения и собственные векторы, так что если **[V,D] = eig(X)**, то **X^p = V*D.^p/V**. Если X — скаляр и **P** — матрица, то **X^P** — это скаляр X, возведенный в степень матрицы **P**. Если **X** и **P** — матрицы, то **X^P** становится некорректной операцией и система выдает ошибку. Возможный вариант решения матричного уравнения с применением оператора **^** можно представить как **X=B*A**$^{-1}$.

' — транспонирование матрицы, т.е. замена строк столбцами и наоборот. Например, **A'** — транспонированная матрица **A**. Для комплексных матриц транспонирование дополняется комплексным сопряжением. Транспонирование при решении систем линейных уравнений полезно, если в матрице **A** переставлены столбцы и строки.

При записи систем линейных уравнений в матричной форме необходимо следить за правильностью записи матрицы **A** и вектора **B**. Пример (в виде M-файла):

```
A=[2  1  0  1;
   1 -3  2  4;
  -5  0 -1 -7;
   1 -6  2  6];
B=[8  9 -5  0];
X1=B/A
X2=B*A^-1
X3=B*inv(A)
```

выдает результаты решения тремя способами:

```
X1 =
   3.0000  -4.0000  -1.0000   1.0000
X2 =
   3.0000  -4.0000  -1.0000   1.0000
X3 =
   3.0000  -4.0000  -1.0000   1.0000
```

Как и следовало ожидать, результаты оказываются одинаковыми.

7.1.2. Функции для решения систем линейных уравнений lscov, nnls

Теперь рассмотрим функции, введенные для решения специальных линейных уравнений методом наименьших квадратов с ограничениями:

X =lscov(A,B,V) — возвращает вектор **X** решения системы уравнений вида **A*X =B+e** , где **e** — вектор шумов, который имеет матрицу ковариаций **V**. Реализуется метод наименьших квадратов в присутствии шумов. Матрица **A** должна быть размера m×n, где m>n. При решении минимизируется следующее выражение: **(Ax-b)'*inv(V)*(Ax-b)**. Решение имеет вид: **X=inv(A'*inv(V)*A'*inv(V)*B**. Алгоритм решения, однако, построен так, что операция инвертирования матрицы **V** не проводится.

[X,dX] = lscov(A,B,V) — возвращает стандартную погрешность (**X** в **dX**). Статистическая формула для стандартной погрешности вычисления коэффициентов имеет следующий вид:

mse = B'*(inv(V)–inv(V)*A*inv(A'*inv(V)*A)*A'*inv(V))*B./(m–n) и

dX = sqrt(diag(inv(A'*inv(V)*A)*mse))

X =nnls(A,B) — решение системы уравнений **AX=B** методом наименьших квадратов с ограничениями. Вектор **X** содержит неотрицательные элементы $X_j \geq 0$, где j=1,2,...n. Порог отбора выбран равным **max(size(A))*norm(A,1)*eps**.

X = nnls(A,B,tol) — решение системы уравнений с явно заданным порогом отбора tol.

[X,W] = nnls(A,B) — вместе с решением возвращает вектор двойственных переменных **W**, где $W_i<0$ при $i|X_i=0$ и $W_i=0$ при $i|X_i>0$.

[X,W] = nnls(A,B,tol) — решение системы уравнений с вектором двойственных переменных **W** и порогом отбора tol.

Применение ограничений позволяет избежать получения отрицательных корней, хотя и ведет к появлению несколько бóльших погрешностей решения, чем в случае решений без ограничений. Пример:

```
» C=[4 3;12 5;3 12];b=[1,45,41];D=nnls(C,b')
D =
   2.2242
   2.6954
```

7.2. Решение систем линейных уравнений с разреженными матрицами

Решение систем уравнений с разреженными матрицами, хотя и не единственная, но несомненно одна из основных сфер применения аппарата разреженных матриц, описанного в главе 5. Ниже приведены относящиеся к этой области применения разреженных матриц функции. Большинство описанных методов относится к итерационным, то есть к тем, решение которых получается в ходе ряда шагов — итераций, постепенно ведущих к получению результата с заданной погрешностью или с наибольшим правдоподобием [33].

7.2.1. Двунаправленный метод сопряженных градиентов — функция bicg

Решение системы линейных уравнений с разреженной матрицей возможно известным двунаправленным методом сопряженных градиентов. Он реализован указанной ниже функцией.

bicg (A, B) — возвращает решение **X** системы линейных уравнений **A*X=B**. Матрица коэффициентов **A** должна быть квадратной размера n×n, а вектор-столбец правых частей уравнений **B** должен иметь длину n . Функция **bicg** начинает итерации от начальной оценки, по умолчанию представляющей собой вектор длиной n, состоящий из нулей. Итерации производятся либо до сходимости решения, либо до появления ошибки, либо до достижения максимального числа итераций (по умолчанию 20). Сходимость достигается, когда относительный остаток **norm(B–A*x)/norm(B)** меньше или равен погрешности метода (по умолчанию 1e–6).

bicg(A,B,tol) — выполняет и возвращает решение с погрешностью (порогом отбора) tol.

bicg(A,b,tol,maxit) — выполняет и возвращает решение при заданном максимальном числе итераций maxit.

bicg(A,b,tol,maxit,M) и **bicg(A,b,tol,maxit,M1,M2)** — при решении используются входные условия **M** или **M = M1*M2** и эффективно решают систему **inv(M)*A*x =inv(M)*b** для **x**. Если **M1** или **M2** — пустые матрицы, то они рассматриваются как единичные матрицы, что эквивалентно отсутствию входных условий вообще.

bicg(A,B,tol,maxit,M1,M2,X0) — точно задается начальная оценка **X0**. Если **X0** — пустая матрица, то по умолчанию используется вектор, состоящий из нулей.

bicg(A,B,tol,maxit,M1,M2,X0) — возвращает решение **X**. Если метод **bicg** сходится, то выводится соответствующее сообщение. Если метод не сходится после максимального числа итераций или по другой причине, выдается на экран относительный остаток **norm(B–A*X)/norm(B)** и номер итерации, на которой метод остановлен.

[X,flag] = bicg(A,X,tol,maxit,M1,M2,X0) — возвращает решение **X** и флаг flag, описывающий сходимость метода.

[X,flag,relres] = bicg(A,X,tol,maxit,M1,M2,X0) — дополнительно возвращает относительный остаток **norm(B–A*X)/norm(B)**. Если флаг flag равен 0, то relres ≤ tol.

[X,flag,relres,iter] = bicg(A,B,tol,maxit,M1,M2,X0) — дополнительно возвращает номер итерации, на которой был вычислен **X**. Значение iter всегда удовлетворяет условию 0≤iter≤maxit.

[X,flag,relres,iter,resvec] = bicg(A,B,tol,maxit,M1,M2,X0) — дополнительно возвращает вектор остаточных норм для каждой итерации, начиная с **resvec(1) = norm(B–A*X0)**. Если флаг flag равен 0, то **resvec** имеет длину iter+1 и **resvec(end) ≤tol*norm(B)**. Возможны значения flag 0, 1, 2, 3 и 4.

Пример:

```
» A=[0 0 1 2;
      1 3 0 0;
      0 1 0 1;
      0 1 0 1];
» B=[11;
      7;
      6;
      4];
» bicg(A,B)
BICG converged at iteration 4 to a solution with relative residual 2.3e-015
ans =
   1.0000
   2.0000
   3.0000
   4.0000
```

Введенные в этом примере матрица **A** и вектор **B** будут использованы и в других примерах данного раздела.

7.2.2. Устойчивый двунаправленный метод — функция bicgstab

Еще один двунаправленный метод, называемый устойчивым, реализует функция

bicgstab(A,B), которая возвращает решение **X** системы линейных уравнений **A*X=B**. Функция **bicgstab** начинает итерации от начальной оценки, по умолчанию представляющей собой вектор длиной **n**, состоящий из нулей. Итерации производятся либо до сходимости метода, либо до появления ошибки, либо до достижения максимального числа итераций. Сходимость метода достигается, когда относительный остаток **norm(B–A*X)/norm(B)** меньше или равен погрешности метода (по умолчанию 1e–6). Функция **bicgstab(...)** имеет и ряд других форм записи, описанных для функции **bicg(...)**. Пример:

```
» bicgstab(A,B)
BICGSTAB converged at iteration 3.5 to a solution with relative residual 6.5e-012
ans =
   1.0000
   2.0000
   3.0000
   4.0000
```

7.2.3. Квадратичный метод сопряженных градиентов — функция cgs

Квадратичный метод сопряженных градиентов реализуется в системе MATLAB с помощью функции **cgs**:

cgs(A,B) — возвращает решение **X** системы линейных уравнений **A*X=B**. Функция **cgs** начинает итерации от начальной оценки, представляющей собой вектор длиной **n**, состоящий из нулей. Итерации производятся либо до сходимости метода, либо до появления ошибки, либо до достижения максимального числа итераций. Сходимость метода достигается, когда относительный остаток **norm(B–A*X)/norm(B)** меньше или равен погрешности метода (по умолчанию 1e–6). Функция **cgs(...)** имеет и ряд других форм записи, описанных для функции **bicg(...)**. Пример:

```
» cgs(A,B)
CGS converged at iteration 4 to a solution with relative residual 4e-014
ans =
   1.0000
   2.0000
   3.0000
   4.0000
```

7.2.4. Метод минимизации обобщенной невязки — функция gmres

Итерационный метод минимизации обобщенной невязки также реализован в системе MATLAB. Для этого используется функция:

gmres(A,B,restart) — возвращает решение **X** системы линейных уравнений **A*X=B**. Функция **gmres** начинает итерации от начальной оценки, представляющей собой вектор длиной n, состоящий из нулей. Итерации производятся либо до сходимости решения, либо до появления ошибки, либо до достижения максимального числа итераций. Сходимость решения достигается, когда относительный остаток **norm(B–A*X)/norm(B)** меньше или равен погрешности (по умолчанию 1e–6). Максимальное число итераций — минимум из n/restart и 10. Функция **gmres(...)** имеет и ряд других форм записи, описанных для функции **bicg(...)**.

Пример:

```
» gmres(A,B)
GMRES(4) converged at iteration 1(4) to a solution with relative residual 3e-016
ans =
   1.0000
   2.0000
   3.0000
   4.0000
```

7.2.5. Метод сопряженных градиентов — функция pcg

Итерационный метод сопряженных градиентов реализован функцией **pcg**:

pcg(A,B) — возвращает решение **X** системы линейных уравнений **A*X=B**. Функция **pcg** начинает итерации от начальной оценки, представляющей собой вектор длиной n, состоящий из нулей. Итерации производятся либо до сходимости решения, либо до появления ошибки, либо до достижения максимального числа итераций. Сходимость достигается, если относительный остаток **norm(b–A*X)/norm(B)** меньше или равен погрешности метода (по умолчанию 1e–6). Максимальное число итераций — минимум из n и 20. Функция **pcg(...)** имеет и ряд других форм записи, описанных для функции **bicg(...)**.

Пример:

```
» pcg(A,B)
Warning: PCG stopped after the maximum 4 iterations without converging to the desired tolerance 1e-006
        The iterate returned (number 4) has relative residual 0.46
> In C:\MATLAB\toolbox\matlab\sparfun\pcg.m at line 347
ans =
```

```
1.7006
1.2870
-2.0535
8.2912
```

В данном случае решение к успеху не привело.

7.2.6. Квазиминимизация невязки — функция qmr

Метод решения систем линейных уравнений с квазиминимизацией невязки реализует функция **qmr**:

qmr(A,B) — возвращает решение **X** системы линейных уравнений **A*X=b**. Функция **qmr** начинает итерации от начальной оценки, представляющей собой вектор длиной n, состоящий из нулей. Итерации производятся либо до сходимости метода, либо до появления ошибки, либо до достижения максимального числа итераций. Сходимость метода достигается, когда относительный остаток **norm(B–A*X)/norm(B)** меньше или равен погрешности метода (по умолчанию 1e–6). Максимальное число итераций — минимум из n и 20. Функция **qmr(...)** имеет и ряд других форм записи, описанных для функции **bicg(...)**. Пример:

```
» qmr(A,B)
QMR converged at iteration 4 to a solution with relative residual 8.4e-015
ans =
   1.0000
   2.0000
   3.0000
   4.0000
```

7.3. Вычисление нулей функции одной переменной

Довольно часто возникает задача решения нелинейного уравнения вида f(x)=0 или f1(x)=f2(x) [39]. Последнее, однако, можно свести к виду f(x)=f1(x)-f2(x)=0. Таким образом данная задача сводится к нахождению значений аргумента x функции f(x) одной переменной, при котором значение функции равно нулю. Соответствующая функция MATLAB, решающая данную задачу, приведена ниже:

fzero('fun',x) — возвращает уточненное значение x, при котором достигается нуль функции **fun**, представленной строкой, при начальном значении аргумента x. Возвращенное значение близко к точке, где функция меняет знак, или равно **NaN**, если такая точка не найдена.

fzero('fun',x) — возвращает значение x, при котором fun(x)=0 с заданием интервала поиска с помощью вектора **x=[x1 x2]**, такого, что **f(x(1))** отличается от знака **f(x(2))**. Если это не так, выдается сообщение об ошибке. Вызов функции **fzero** с интервалом гарантирует, что **fzero** возвратит значение, близкое к точке, где fun изменяет знак.

fzero('fun',x,tol) — возвращает результат с заданной погрешностью tol.

fzero('fun',x,tol,trace) — выдает на экран информацию по каждой итерации.

fzero('fun',x,tol,trace,P1,P2,...) — предусматривает дополнительные аргументы, помещаемые в функцию **fun(x,P1,P2,...)**. При задании пустой матрицы для **tol** или **trace** используются значения по умолчанию, например: **fzero('fun',x,[ ],[ ],P1)**

Для функции **fzero** ноль рассматривается как точка, где график функции **fun** пересекает ось x, а не касается ее. В зависимости от формы задания функции **fzero** реализуются следующие, хорошо известные численные методы поиска нуля функции: деления отрезка пополам, секущей и обратной квадратичной интерполяции.

Приведенный ниже пример показывает приближенное вычисление $\pi/2$ из решения уравнения cos(x)=0:

```
» x = fzero('cos',[1 3])
x =
  1.5708
```

В более сложных случаях настоятельно рекомендуется строить график функции f(x) для приближенного определения корней и интервалов, в пределах которых они находятся. Ниже дан пример такого рода:

```
%Функция, корни которой ищутся
function f=fun1(x)
f=0.25*x+sin(x)-1;
```

Построим график функции (рис. 7.1):

```
» x=0:0.1:10;
» plot(x,fun1(x));grid on;
```

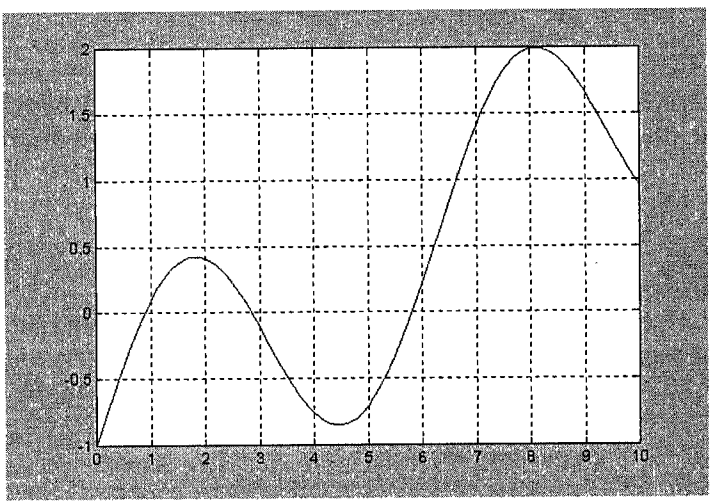


Рис. 7.1. График функции fun1(x)

Из него нетрудно заметить, что значения корней заключены примерно в интервалах [0.5 1], [2 3] и [5 6]. Найдем их, используя функцию fzero:

```
» x1=fzero('fun1',[0.5 1]).
x1 =
   0.8905
» x2=fzero('fun1',[2 3])
x2 =
   2.8500
» x3=fzero('fun1',[5,6])
x3 =
   5.8128
» x3=fzero('fun1',5,0.001)
x3 =
   5.8111
```

Обратите внимание на то, что корень x3 найден двумя способами и что его значения в третьем знаке после десятичной точки отличаются в пределах заданной погрешности tol=0.001. К сожалению, разом найти все корни функция fzero не в состоянии.

Для решения систем нелинейных уравнений следует использовать функцию **solve** из пакета Symbolic. Эта функция способна выдать результат в символьной форме, а если такого нет, то она позволяет получить решение в численном виде.

7.4. Вычисление минимума функции

7.4.1. Минимизация функции одной переменной — функция fmin

Еще одна важная задача численных методов — поиск минимума функции f(x) в некотором интервале изменения x — от x1 до x2 [39]. Если нужно найти максимум такой функции, то достаточно поставить знак минус перед функцией. Для решения этой задачи используется следующая функция:

fmin('fun',x1,x2) — возвращает значение x, которое является локальным минимумом функции fun(x) на интервале x1<x<x2.

fmin('fun',x1,x2,options) — сходна с описанной выше функцией, но использует контрольные параметры **options** (см. ниже) для управления ходом решения.

fmin('fun',x1,x2,options,P1,P2,...) — сходна с описанной выше, но передает аргументы в целевую функцию **fun(x,P1,P2,...)**. Если **options** — пустая матрица, то используются параметры по умолчанию.

[x,options] = fmin(...) — дополнительно возвращает вектор контрольных параметров **options**, в десятом столбце которого содержится число выполненных итераций.

В этих представлениях: **x1,x2** — интервал, на котором ищется минимум функции, **P1,P2...** — передаваемые в функцию аргументы, **fun** — строка, содержащая название функции, которая будет минимизирована, **options** — вектор контрольных параметров, имеющий 18 компонент. Только 3 из них используются функцией **fmin**: **options(1)** — если опция не 0, то отображаются промежуточные шаги решения (по умолчанию значение **options(1)** равно 0), **options(2)** — задается итерационная погрешность (по умолчанию она равна 1.e–4) и **options(14)** — задается максимальное число итераций, по умолчанию равное 500.

В зависимости от формы задания функции fmin вычисления минимума выполняются известными методами "золотого сечения" или параболической интерполяции [4]. Пример:

```
» [x,options]=fmin('cos',3,4,[0,1.e-10])
x =
  3.14159265402771
options =
 1.0e+002 *
 Columns 1 through 4
         0   0.00000000000100   0.00000100000000   0.00000001000000
 Columns 5 through 8
         0                  0                  0   -0.01000000000000
 Columns 9 through 12
         0   0.09000000000000                  0                  0
 Columns 13 through 16
         0   5.00000000000000                  0   0.00000000010000
 Columns 17 through 18
  0.00100000000000         0
```

7.4.2. Минимизация функции ряда переменных — функция fmins

Значительно сложнее задача минимизации функций ряда переменных f(x1,x2,...). При этом значения переменных представляются вектором x, причем начальные значения задаются вектором x0. Для минимизации функций ряда переменных MATLAB использует симплекс-метод Нелдера-Мида.

Этот метод является одним из лучших прямых методов минимизации функций ряда переменных, не требующих вычисления градиента или производных функции. Он сводится к построению симплекса в n-мерном пространстве, заданного n+1 вершиной. В двумерном пространстве симплекс является треугольником, а в трехмерном — пирамидой. На каждом шаге итераций выбирается новая точка решения внутри или вблизи симплекса. Она сравнивается с одной из вершин симплекса. Ближайшая к этой точке вершина симплекса обычно заменяется этой точкой. Таким образом симплекс

шения. Решение повторяется, пока размеры симплекса не станут меньше заданной погрешности решения по всем переменным.

Реализующая симплекс-метод Нелдера-Мида функция записывается в следующем виде:

fmins('fun',x0) — возвращает вектор **x**, который является локальным минимумом функции **fun(x)** вблизи точки **x0**.

fmins('fun',x0,options) — аналогична описанной выше функции, но использует вектор контрольных параметров **options**. Можно использовать 4 параметра из 18-ти: **options(1)** — вывод промежуточных результатов (0 — вывода нет, 1 — вывод есть), **options(2)** — задание итерационной погрешности для аргумента (по умолчанию 1e-4), **options(3)** — задание итерационной погрешности для функции (по умолчанию 1e-4) и **options(14)** — задание максимального количества итераций (по умолчанию 0, максимум 200×n, где n — число переменных).

fmins('fun',x0,options,[],P1,P2,...) — сходна с описанной выше функцией, но передает аргументы в целевую функцию **fun(x,P1,P2,...)**. Если **options** — пустая матрица, то используются параметры по умолчанию.

[x,options] = fmins(...) — дополнительно возвращает вектор контрольных параметров **options**, в десятом столбце которого содержится число выполненных итераций.

Классическим примером применения функции fmins является поиск минимума тестовой функции Розенброка, точка минимума которой находится в овраге с "плоским дном":

$$rb(x1,x2,a)=100*(x2-x1^2)^2+(a-x1)^2.$$

Минимальное значение этой функции равно нулю и достигается в точке $[a\ a^2]$. В качестве примера уточним значения x1 и x2 в точке [-1.2 1]. Зададим функцию:

```
% Тестовая функция Розенброка
function f=rb(x,a)
if nargin<2 a=1; end
f=100*(x(2)-x(1)^2)^2+(a-x(1))^2;
```

Теперь решим поставленную задачу:

```
» [xmin, opt]=fmins('rb',[-1.2 1],[0 1e-6]);
» xmin
xmin =
   1.0000   1.0000
» opt
opt =
 Columns 1 through 7
     0   0.0000   0.0001   0.0000        0        0        0
```

```
Columns 8 through 14
 0.0000    0   89.0000         0         0         0  400.0000
Columns 15 through 18
    0    0.0000    0.1000         0
```

Для лучшего понимания сути минимизации функции ряда переменных рекомендуется просмотреть пример минимизации этой функции, имеющийся в библиотеке демонстрационных примеров Demos — рис. 7.2.

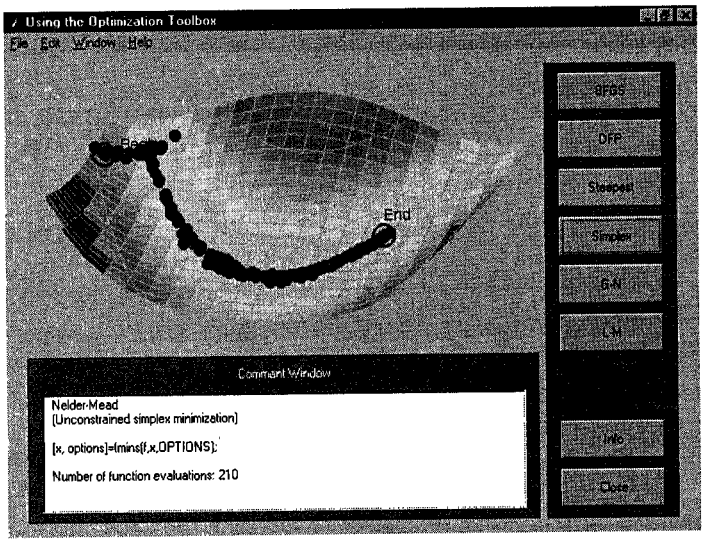


Рис. 7.2. Графическая иллюстрация минимизации функции Розенброка симплекс-методом

Для минимизации функций ряда переменных можно использовать также функции **fminu** и **leastsq**. Первая из них позволяет задать в списке параметров вектор градиентов, вторая реализует метод наименьших квадратов и, как правило, дает наименьшее число итераций при минимизации. Ограничимся приведением примеров их применения для функции Розенброка:

```
» [xmin, opt]=fminu('rb',[-1.2 1],[0 1e-6]);
» xmin
xmin =
   1.0000   1.0000

» [xmin, opt]=leastsq('rb',[-1.2 1],[0 1e-6]);
maximum number of iterations has been exceeded
» xmin
xmin =
  -0.9151   0.8412
```

Обратите внимание на то, что функция **leastsq** к успеху не привела. Выдано сообщение о превышении лимита на число итераций, а значения xmin явно далеки от верных. В библиотеке примеров Demos вы найдете наглядные примеры применения данных функций.

7.5. Вычисление конечных разностей

Вычисление конечных разностей часто возникает в задачах математики. Данный раздел посвящен описанию некоторых функций для вычисления конечных разностей, градиента и аппроксимации Лапласиана.

7.5.1. Аппроксимация Лапласиана — функция del2

Для выполнения аппроксимации Лапласиана в MATLAB используется следующая функция:

del2(U) — возвращает матрицу **L** дискретной аппроксимации дифференциального оператора Лапласа, примененного к функции **U**:

$$L = \frac{\nabla^2 u}{4} = \frac{1}{4}\left(\frac{d^2u}{dx^2} + \frac{d^2u}{dy^2}\right).$$

Матрица **L** имеет тот же размер, что у матрицы **U**, и каждый ее элемент равен разности элемента массива **U** и среднего значения четырех его соседних элементов (для узлов сетки во внутренней области). Для вычислений используется 5-точечная формула аппроксимации Лапласиана.

Другие функции возвращают так же матрицу **L**, но допускают дополнительные установки.

L = del2(U,h) — использует шаг h для установки расстояния между точками в каждом направлении (h — скалярная величина). По умолчанию h=1.

L = del2(U,hx,hy) — использует **hx** и **hy** для точного определения расстояния между точками. Если hx — скалярная величина, то задается расстояние между точками в направлении оси x, если **hx** — вектор, то он должен иметь длину, равную **size(u,2)**, и точно определять координаты точек по оси x. Аналогично, если hy — скалярная величина, то задается расстояние между точками в направлении оси y, если **hy** — вектор, то он должен иметь длину, равную **size(u,1)**, и точно определять координаты точек по оси y.

L = del2(U,hx,hy,hz,...) — если **U** — многомерный массив, то расстояния задаются с помощью **hx**, **hy**, **hz**, ...

Пример:

```
» [x,y] = meshgrid(-5:5,-4:4);
» U =x.*x+y.*y;
» U =x.*x+y.*y
U =
```

```
   41   32   25   20   17   16   17   20   25   32   41
   34   25   18   13   10    9   10   13   18   25   34
   29   20   13    8    5    4    5    8   13   20   29
   26   17   10    5    2    1    2    5   10   17   26
   25   16    9    4    1    0    1    4    9   16   25
   26   17   10    5    2    1    2    5   10   17   26
   29   20   13    8    5    4    5    8   13   20   29
   34   25   18   13   10    9   10   13   18   25   34
   41   32   25   20   17   16   17   20   25   32   41
» V=del2(U)
V =
    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    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
» surfl(U),surfl(V)
» subplot(1,2,1)
» surfl(U)
» subplot(1,2,2)
» surfl(V)
```

На рис. 7.3 представлены графики функций для поверхностей **U** и **V**.

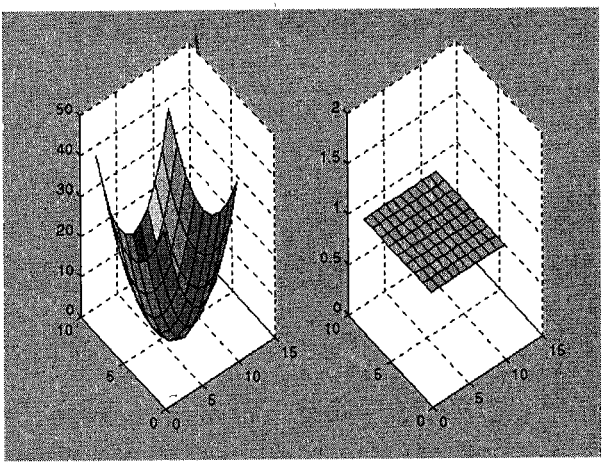


Рис. 7.3 . Графики функций для поверхностей **U** и **V**

Эти графики построены по завершающим командам данного примера.

7.5.2. Аппроксимация производных конечными разностями — функция diff

Одним из важнейших приложений конечно-разностных методов является приближенное представление производных функций — численное дифференцирование. Оно реализуется следующей функцией:

diff(X) — возвращает конечные разности смежных элементов массива **X**. Если **X** — вектор, то **diff(X)** возвращает вектор разностей соседних элементов **[X(2)–X(1) X(3)–X(2) ... X(n)–X(n–1)]**, у которого количество элементов на единицу меньше, чем у исходного вектора **X**. Если **X** — матрица, то **diff(X)** возвращает матрицу разностей столбцов:**[X(2:m,:)–X(1:m–1,:)]**.

diff(X,n) — возвращает конечные разности порядка n. Так, **diff(X,2)** — это то же самое, что и **diff(diff(X))**. При вычислениях используется рекуррентное уравнение **diff(x,n)=diff(x,n-1)**.

Y = diff(X,n,dim) — возвращает конечные разности по строкам или по столбцам для матрицы **X** в зависимости от значения величины dim. Если порядок числа n равен или превышает величину dim, то **diff** возвращает пустой массив.

Примеры:

```
» X=[1 2 4 6 7 9 3 45 6 7]
X =
   1   2   4   6   7   9   3  45   6   7
» size(X)
ans =
   1  10
» Y = diff(X)
Y =
   1   2   2   1   2  -6  42 -39   1
» size(Y)
ans =
   1   9
» Y = diff(X,2)
Y =
   1   0  -1   1  -8  48 -81  40
» X=magic(5)
X =
  17  24   1   8  15
  23   5   7  14  16
   4   6  13  20  22
  10  12  19  21   3
  11  18  25   2   9
» Y = diff(X,2)
Y =
 -25  20   0   0   5
  25   5   0  -5 -25
  -5   0   0 -20  25
```

```
» Y = diff(X,2,3)
Y =
  Empty array: 5-by-5-by-0
```

Используя функцию **diff**, можно строить графики производных заданной функции. Пример этого показан ниже:

```
» X=0:0.05:10;
» S=sin(X);
» D=diff(S);
» plot(D/0.05)
```

Для получения приближенных численных значений производной от функции sin(x) вектор конечно-разностных значений **D** поделен на шаг точек по **X**. Как и следовало ожидать, полученный график очень близок к графику функции косинуса — рис. 7.4.

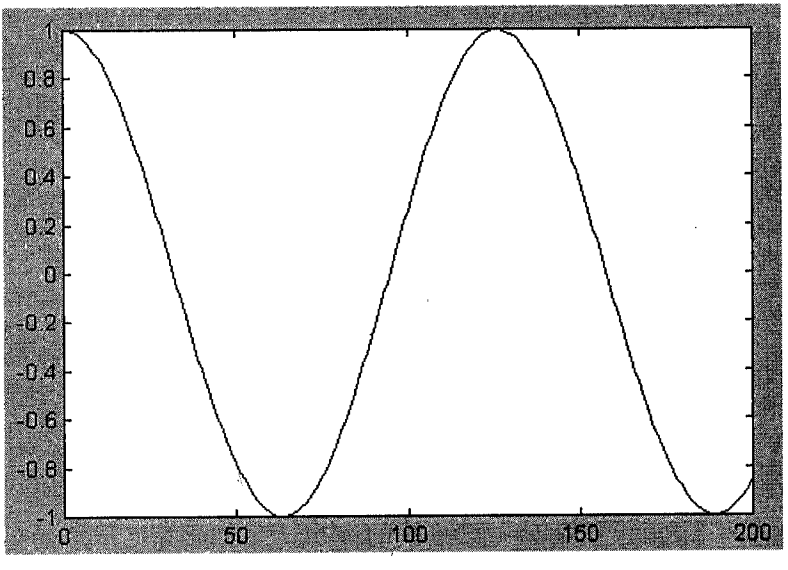


Рис. 7.4. Приближенный график производной от функции sin(x)

Обратите внимание, что по оси x отложены номера элементов вектора **X**, а не истинные значения **X**.

Пакет расширения Symbolic, описанный в главе 9, позволяет выполнять дифференцирование функций в аналитическом виде, то есть точно. Это, однако, не всегда нужно.

7.5.3. Вычисление градиента функции — gradient

Вычисление конечно-разностным методом градиента функций реализуется следующей функцией:

FX = gradient(F) — возвращает одномерный градиент от вектора **F**. **FX** соответствует конечным разностям в направлении x.

[FX,FY] = gradient(F) — возвращает двухмерный градиент от матрицы **F** в виде массивов **FX,FY**. **FX** соответствует конечным разностям в направлении x (столбцов). **FY** соответствует конечным разностям в направлении y (строк).

[FX,FY,FZ,...] = gradient(F) — возвращает N компонентов градиента от многомерного массива **F**.

[...] = gradient(F,h) — использует шаг h для установки расстояния между точками в каждом направлении (h — скалярная величина). По умолчанию h=1.

[...] = gradient(F,h1,h2,...) — если **F**- многомерный массив, то расстояния задаются с помощью h1, h2, h3, ...

Пример:

```
» F=[1 3 5 7 9 5 6 7 8]
F =
   1   3   5   7   9   5   6   7   8
» FX = gradient(F)
FX =
  Columns 1 through 7
   2.0000   2.0000   2.0000   2.0000  -1.0000  -1.5000   1.0000
  Columns 8 through 9
   1.0000   1.0000
» F=[1 2 3 6 7 8;1 4 5 7 9 3;5 9 5 2 5 7]
F =
   1   2   3   6   7   8
   1   4   5   7   9   3
   5   9   5   2   5   7
» [FX,FY] = gradient(F)
FX =
   1.0000   1.0000   2.0000   2.0000   1.0000   1.0000
   3.0000   2.0000   1.5000   2.0000  -2.0000  -6.0000
   4.0000        0  -3.5000        0   2.5000   2.0000
FY =
        0   2.0000   2.0000   1.0000   2.0000  -5.0000
   2.0000   3.5000   1.0000  -2.0000  -1.0000  -0.5000
   4.0000   5.0000        0  -5.0000  -4.0000   4.0000
```

Функция **gradient** часто используется для построения графиков поля градиентов — см. рис. 4.18.

7.6. Численное интегрирование

Численное интегрирование традиционно является одной из важнейших сфер применения математического аппарата. В данном разделе приводятся функции для численного интегрирования различными методами.

7.6.1. Численное интегрирование методом трапеций — функции cumtrapz, trapz

Численное интегрирование заключается в приближенном вычислении определенного интеграла вида

$$\int_a^b y(x)dx$$

одним из многих численных методов. Приведенные ниже функции выполняют численное интегрирование методом трапеций и методом трапеций с накоплением.

cumtrapz(Y) — возвращает численное значение интеграла для функции, заданной ординатами в векторе или матрице **Y** с шагом интегрирования, равным единице (интегрирование методом трапеций с накоплением). В случае, когда шаг отличен от единицы, но постоянен, достаточно вычисленный интеграл умножить на величину шага. Для векторов эта функция возвращает вектор, содержащий результат интегрирования с накоплением элементов вектора **Y**. Для матриц — возвращает матрицу того же размера, как и **Y**, содержащую интегрирование с накоплением для каждого столбца матрицы **Y**.

cumtrapz(X,Y) — выполняет интегрирование с накоплением от **Y** по переменной **X**, используя метод трапеций. **X** и **Y** должны быть векторами одной и той же длины, или **X** должен быть вектором-столбцом, а **Y** — матрицей.

cumtrapz(... dim) — выполняет интегрирование с накоплением элементов по размерности, точно определенной скаляром **dim**. Длина вектора X должна быть равна **size(Y,dim).**

Пример:

```
» Y=magic(4)
Y =
   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1
» Z = cumtrapz(Y,1)
Z =
         0         0         0         0
   10.5000    6.5000    6.5000   10.5000
   17.5000   15.5000   14.5000   20.5000
   24.0000   26.0000   25.0000   27.0000
```

trapz(Y) — возвращает интеграл, используя интегрирование методом трапеций. Если **Y** — вектор, то **trapz(Y)** возвращает интеграл элементов вектора **Y**, если **Y** — матрица, то **trapz(Y)** возвращает вектор-строку, содержащую интегралы каждого столбца этой матрицы.

trapz(X,Y) — возвращает интеграл от функции **Y** по переменной **X**, используя метод трапеций.

trapz(...,dim) — возвращает интеграл по строкам или по столбцам для входной матрицы в зависимости от значения переменной dim.

Пример:

```
» X=0:pi/70:pi/2;
» Y=cos(X);
» Z = trapz(Y)
Z =
  22.2780
```

7.6.2. Численное интегрирование методом квадратур — функции dblquad, quad, quad8

Приведенные ниже функции осуществляют интегрирование и двойное интегрирование, используя квадратурную формулу и метод Ньютона-Котеса. Квадратура — численный метод нахождения площади под графиком функции f(x), то есть вычисление определенного интеграла вида:

$$q = \int_a^b f(x)dx.$$

Функции **quad** и **quad8** используют два различных алгоритма квадратуры для вычисления определенного интеграла. Функция **quad** выполняет интегрирование по методу низкого порядка, используя рекурсивное правило Симпсона [4,40]. Функция **quad8** выполняет интегрирование по методу более высокого порядка, используя квадратурные формулы Ньютона-Котеса 8-го порядка [40].

dblquad('fun',inmin,inmax,outmin,outmax) — вычисляет и возвращает значение двойного интеграла для подынтегральной функции **fun(inner,outer)**, используя квадратурную функцию **quad**. **inner** — внутренняя переменная, изменяющаяся от inmin до inmax, и outer — внешняя переменная, изменяющаяся от outmin до outmax. Первый аргумент **'fun'** — строка, описывающая подынтегральную функцию. Эта функция должна быть функцией двух переменных вида **fout=fun(inner,outer)**. Функция должна брать вектор **inner** и скаляр outer и возвращать вектор **fout**, который является функцией, вычисленной в **outer** и в каждом значении **inner**.

result = dblquad('fun',inmin,inmax,outmin,outmax,tol,trace) — передает параметры **tol** и **trace** в функцию **quad**. Смотрите справку по функции **quad** для описания параметров **tol** и **trace**.

result = dblquad('fun',inmin,inmax,outmin,outmax,tol,trace,order) — передает параметры **tol** и **trace** для функции **quad** или **quad8** в зависимости от значения строки **order**. Допустимые значения для параметра **order** — **'quad'** и **'quad8'** или имя любого, определенного пользователем квадратурного метода с таким же вызовом и возвращаемыми параметрами, как у функций **quad** и **quad8**.

Пример: пусть М-файл **integ1.m** описывает функцию **2*y*sin(x)+x/2*cos(y)**, тогда:

```
» result = dblquad('integ1',pi,2*pi,0,2*pi)
result =
  -78.9574
```

quad('fun',a,b) — возвращает вычисленное значение определенного интеграла от заданной функции **'fun'** на отрезке **[a b]**. Используется адаптивный метод Симпсона.

quad('fun',a,b,tol) — возвращает вычисленное значение определенного интеграла с заданной относительной погрешностью tol. По умолчанию tol=1.e–3. Можно также использовать вектор, состоящий из двух элементов **tol =[rel_tol abs_tol]**, чтобы точно определить комбинацию относительной и абсолютной погрешности.

quad('fun',a,b,tol,trace) — возвращает вычисленное значение определенного интеграла и при значении trace, не равном нулю, строит график, показывающий ход вычислений интеграла.

quad('fun',a,b,tol,trace,P1,P2,...) — возвращает вычисленное значение определенного интеграла с подынтегральной функцией fun, использует параметры **P1, P2, ...**, которые напрямую передаются в точно определенную функцию: **G = fun(X,P1,P2,...)**.

Примеры:

```
» q = quad('exp',0,2,1e-4)
q =
   6.3891
» q = quad('sin',0,pi,1e-3)
q =
   2.0000
```

7.7. Обработка данных в массивах

Этот раздел открывает часть настоящей главы, посвященную традиционной обработке данных. В нем приведены основные функции для обработки данных, представленных массивом. Они широко используются для анализа данных физических, химических, экономических и иных экспериментов.

7.7.1. Нахождение максимального и минимального элементов массива — функции max и min

Самый простой анализ данных, содержащихся в некотором массиве, заключается в поиске его элементов с максимальным и минимальным значениями. В системе MATLAB определены быстрые функции для нахождения максимальных и минимальных элементов массива:

max(A) — возвращает наибольший элемент, если **A** — вектор; или возвращает вектор-строку, содержащую максимальные элементы каждого столбца, если **A** — матрица.

max(A,B) — возвращает массив того же размера, что **A** и **B**, каждый элемент которого есть максимальный из соответствующих элементов этих массивов.

max(A,[ ],dim) — возвращает наибольший элемент по столбцам или по строкам матрицы в зависимости от значения скаляра dim. Например, **max(A,[ ],1)** возвращает максимальные элементы каждого столбца матрицы **A**.

[C,I] = max(A) — кроме максимальных значений возвращает вектор индексов этих элементов.

Примеры:

```
» A=magic(7)
A =
   30   39   48    1   10   19   28
   38   47    7    9   18   27   29
   46    6    8   17   26   35   37
    5   14   16   25   34   36   45
   13   15   24   33   42   44    4
   21   23   32   41   43    3   12
   22   31   40   49    2   11   20
» C = max(A)
C =
   46   47   48   49   43   44   45
» C = max(A,[ ],1)
C =
   46   47   48   49   43   44   45
» C = max(A,[ ],2)
C =
   48
   47
   46
   45
   44
   43
   49
» [C,I] = max(A)
C =
   46   47   48   49   43   44   45
I =
    3    2    1    7    6    5    4
```

Для быстрого нахождения элемента массива с минимальным значением служат следующие функции:

min(A) — возвращает минимальный элемент, если **A** — вектор; или возвращает вектор-строку, содержащую минимальные элементы каждого столбца, если **A** — матрица.

min(A,B) — возвращает массив, того же размера, что **A** и **B**, каждый элемент которого есть минимальный из соответствующих элементов этих массивов.

min(A,[ ],dim) — возвращает наименьший элемент по столбцам или по строкам матрицы в зависимости от значения скаляра dim. Например, **max(A,[ ],1)** возвращает максимальные элементы каждого столбца матрицы **A**.

[C,I] = min(A) — кроме минимальных значений возвращает вектор индексов их элементов.

Пример:

```
» A=magic(4)
A =

   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1
» [C,I] = min(A)
C =
    4    2    3    1
I =
    4    1    1    4
```

Работа указанных функций базируется на сравнении численных значений элементов массива **A**, что и обеспечивает высокую скорость выполнения операций.

7.7.2. Нахождение средних, срединных значений массива и стандартных отклонений — функции mean, median и std

Элементарная статистическая обработка данных в массиве обычно сводится к нахождению их среднего значения, медианы (срединного значения) и стандартного отклонения [47,48]. Для этого в системе MATLAB определены следующие функции:

mean(A) — возвращает арифметическое среднее значение элементов массива, если **A** — вектор; или возвращает вектор-строку, содержащую средние значения элементов каждого столбца, если **A** — матрица. Арифметическое среднее значение есть сумма элементов массива, деленная на их число.

mean(A,dim) — возвращает среднее значение элементов по столбцам или по строкам матрицы в зависимости от значения скаляра dim.

Примеры:

```
» A = [1 2 6 4 8; 6 7 13 5 4; 7 9 0 8 12; 6 6 7 1 2]
A =
    1    2    6    4    8
```

```
    6    7   13    5    4
    7    9    0    8   12
    6    6    7    1    2
» mean(A)
ans =
   5.0000   6.0000   6.5000   4.5000   6.5000
» mean(A,2)
ans =
   4.2000
   7.0000
   7.2000
   4.4000
```

median(A) — возвращает медиану, если **A** — вектор; или вектор-строку медиан для каждого столбца, если **A** — матрица.

median(A,dim) — возвращает значения медиан для столбцов или строк матрицы в зависимости от значения скаляра dim.

Примеры:

```
» A=magic(6)
A =
   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
» M=median(A)
M =
  19.0000  18.5000  18.0000  19.0000  18.5000  18.0000
» M=median(A,2)
M =
  21.5000
  22.0000
  21.0000
  16.0000
  15.0000
  15.5000
```

std(X) — возвращает стандартное отклонение элементов массива, вычисляемое по формуле:

$$S=\left(\frac{1}{n-1}\sum_{i=1}^{n}\left(x_i-\bar{x}\right)^2\right)^{\frac{1}{2}},$$

где **X** — вектор. Если **X** — матрица, то **std(X)** возвращает вектор-строку, содержащую стандартное отклонение элементов каждого столбца.

std(X,flag) — возвращает то же значение, что и **std(X)**, если flag = 0; если flag = 1, **std(X,1)** возвращает стандартное отклонение, вычисляемое по формуле:

$$S = \left(\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2\right)^{\frac{1}{2}}.$$

std(X,flag,dim) — возвращает стандартное отклонение по строкам или по столбцам для матрицы **X** в зависимости от значения переменной dim.

Примеры:

```
» X = linspace(0,3*pi,10)
X =
 Columns 1 through 7
      0   1.0472   2.0944   3.1416   4.1888   5.2360   6.2832
 Columns 8 through 10
  7.3304   8.3776   9.4248
» s = std(X)
s =
  3.1705
```

7.7.3. Функции сортировки элементов массива sort, sortrows и cplxpair

Многие операции статистической обработки данных выполняются быстрее и надежнее, если данные предварительно отсортированы. Кроме того, нередко представление данных в отсортированном виде более наглядно и ценно. Ряд функций служит для выполнения сортировки элементов массива. Они представлены ниже.

sort(A) — в случае одномерного массива **A** сортирует и возвращает элементы по возрастанию их значений; в случае двумерного массива происходит сортировка и возврат элементов каждого столбца. Допустимы вещественные, комплексные и строковые элементы. Если **A** принимает комплексные значения, то сначала элементы сортируются по абсолютному значению, затем, если они равны, по аргументу. Если **A** включает **NaN** элементы, **sort** помещает их в конец.

[B,INDEX] = sort(A) — наряду с отсортированным массивом возвращает массив индексов. **INDEX** имеет размер **size(A)**, с помощью этого массива можно восстановить структуру исходного массива.

sort(A,dim), для матриц сортирует элементы по столбцам или по строкам в зависимости от значения переменной dim.

Примеры:

```
» A=magic(5)
A =
  17   24   1    8   15
```

```
   23    5    7   14   16
    4    6   13   20   22
   10   12   19   21    3
   11   18   25    2    9
» [B,INDEX] = sort(A)
B =
    4    5    1    2    3
   10    6    7    8    9
   11   12   13   14   15
   17   18   19   20   16
   23   24   25   21   22
INDEX =
    3    2    1    5    4
    4    3    2    1    5
    5    4    3    2    1
    1    5    4    3    2
    2    1    5    4    3
```

sortrows(A) — выполняет сортировку строк массива **A** по возрастанию и возвращает отсортированный массив, который может быть или матрицей, или вектором-столбцом.

sortrows(A,column) — возвращает матрицу, отсортированную по столбцам, точно указанным в векторе **column**. Например, **sortrows(A,[2 3])** сортирует строки матрицы **A** сначала по второму столбцу, и если его элементы равны, затем по третьему.

[B,index] = sortrows(A) — также возвращает вектор индексов **index**. Если **A** – вектор-столбец, то **B = A(index)**. Если **A** — матрица размера m×n, то **B = A(index,:)**.

Примеры:

```
» A=[2 3 5 6 8 9; 5 7 1 2 3 5;1 3 2 1 5 1;5 0 8 8 4 3]
A =
    2    3    5    6    8    9
    5    7    1    2    3    5
    1    3    2    1    5    1
    5    0    8    8    4    3
» B = sortrows(A)
B =
    1    3    2    1    5    1
    2    3    5    6    8    9
    5    0    8    8    4    3
    5    7    1    2    3    5
» B = sortrows(A,3)
B =
    5    7    1    2    3    5
    1    3    2    1    5    1
    2    3    5    6    8    9
    5    0    8    8    4    3
```

cplxpair(A) — сортирует элементы массива **A** по строкам или столбцам комплексного массива, группируя вместе комплексно сопряженные пары. Сопряженные пары сортируются по возрастанию действительной части. Внутри пары элемент с отрицательной мнимой частью является первым. Действительные элементы следуют за комплексными парами. Заданный по умолчанию порог 100*eps относительно **abs(A(i))** определяет, какие числа являются действительными и какие элементы являются комплексно сопряженными.

Если **A** — вектор, **cplxpair(A)** возвращает вектор **A** вместе с комплексно сопряженными парами.

Если **A** — матрица, **cplxpair(A)** возвращает матрицу **A** с комплексно сопряженными парами, сортированную по столбцам.

cplxpair(A,tol) — отменяет заданный по умолчанию порог и задает новый tol.

cplxpair(A,[],dim) — сортирует матрицу **A** по строкам или по столбцам в зависимости от значения заданной величины dim.

cplxpair(A,tol,dim) — сортирует матрицу **A** по строкам или по столбцам в зависимости от значения величины dim с заданным порогом tol.

Пример:

```
» A=[23+12i,34-3i,45;23-12i,-12,2i;-3,34+3i,-2i]
A =
 23.0000 +12.0000i  34.0000 - 3.0000i   45.0000
 23.0000 -12.0000i  -12.0000             0 + 2.0000i
 -3.0000            34.0000 + 3.0000i    0 - 2.0000i
» cplxpair(A)
ans =
 23.0000 -12.0000i  34.0000 - 3.0000i    0 - 2.0000i
 23.0000 +12.0000i  34.0000 + 3.0000i    0 + 2.0000i
 -3.0000            -12.0000             45.0000
```

7.8. Геометрический анализ данных

В этом разделе приведены функции геометрического анализа данных. Такой анализ не относится к достаточно распространенным средствам анализа данных, но для специалистов он представляет несомненный интерес.

7.8.1. Триангуляция — функции delaunay, dsearch, tsearch

Пусть есть некоторое число точек. **Триангуляция Делоне** — это множество линий, соединяющих каждую точку с ее ближайшими соседними точками. **Диаграммой Вороного** называют многоугольник, вершины которого — центры окружностей, описанных вокруг треугольников Делоне. В системе MATLAB определены функции триангуляции Делоне, триангуляции Делоне для ближайшей точки и поиска наилучшей триангуляции. Рассмотрим функции, реализующие триангуляцию Делоне.

TRI = delaunay(x,y) — возвращает матрицу размера m×3 множества треугольников (триангуляция Делоне) такую, что ни одна из точек данных, содержащихся в векторах **x** и **y**, не попадает внутрь кругов, проходящих через вершины треугольников. Каждая строка матрицы **TRI** определяет один такой треугольник и состоит из индексов векторов **x** и **y**.

TRI = delaunay(x,y,'sorted') — возвращает матрицу размера m×3 множества треугольников, принимая, что точки векторов **x** и **y** отсортированы сначала по **y**, затем по **x**, и двойные точки уже устранены.

Пример:

```
» rand('state',0);
» x=rand(1,25);
» y=rand(1,25);
» TRI = delaunay(x,y);
» trimesh(TRI,x,y,zeros(size(x)))
» axis([0 1 0 1]); hold on;
» plot(x,y,'o')
```

Построенная по этому примеру диаграмма представлена на рис. 7.5.

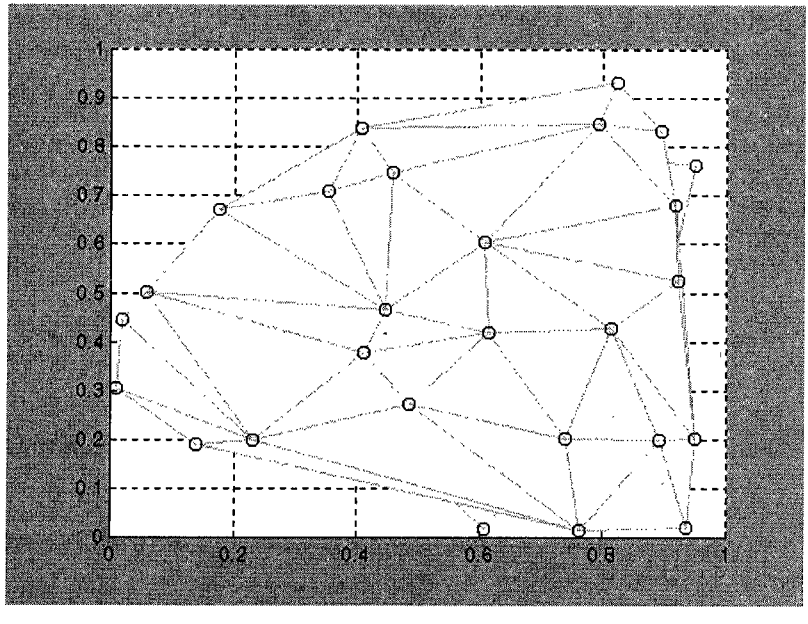


Рис 7.5. Пример применения функции **delaunay**

dsearch(x,y,TRI,xi,yi) — возвращает индекс ближайшей точки (с координатами (x,y)) к точке с координатами (xi,yi) (триангуляция Делоне для ближайшей точки).

dsearch(x,y,TRI,xi,yi,S) — делает то же, используя разреженную матрицу **S**:

S = sparse(TRI(:,[1 1 2 2 3 3]),TRI(:,[2 3 1 3 1 2]),1,nxy,nxy), где **nxy = prod(size(x))**.

tsearch(x,y,TRI,xi,yi) — выполняет поиск наилучшей триангуляции, возвращает индексы в строках **TRI** для каждой точки с координатами **xi,yi**. **tsearch** возвращает **NaN** для всех точек, находящихся вне выпуклой оболочки.

7.8.2. Функция вычисления выпуклой оболочки convhull

В системе MATLAB определена функция вычисления точек выпуклой оболочки:

convhull(x,y) — возвращает индексы точек в векторах **x** и **y** на выпуклой оболочке.

convhull(x,y,TRI) — использует триангуляцию (полученную в результате применения функции триангуляции Делоне **delaunay**) вместо сквозного вычисления — каждый раз.

Пример:

```
» xx=-0.8:0.03:0.8;
» yy = abs(sqrt(xx));
» [x,y] = pol2cart(xx,yy);
» k = convhull(x,y);
» plot(x(k),y(k),'r:',x,y,'g*')
```

Рис. 7.6 иллюстрирует применение функции **convhull** для построения выпуклой оболочки.

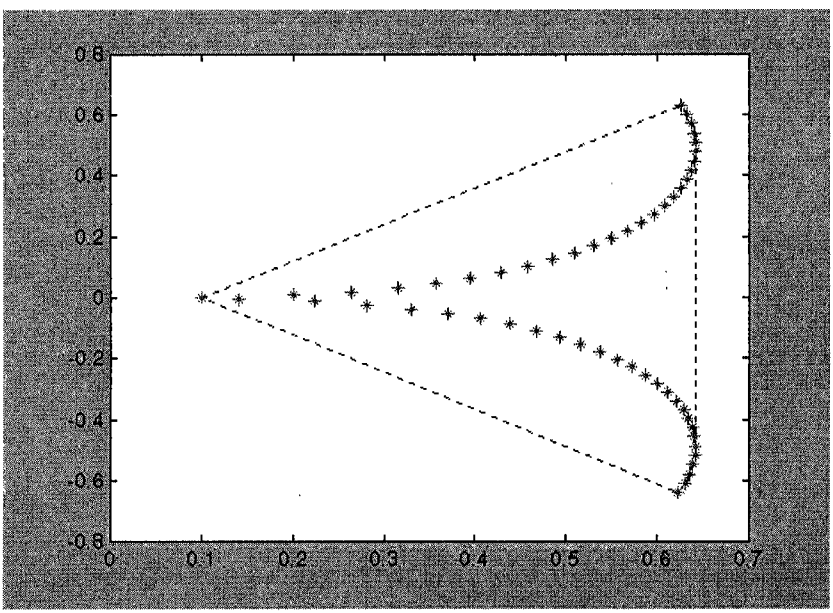


Рис 7.6. Пример использования функции **convhull**

7.8.3. Функция вычисления площади полигона — polyarea

В системе MATLAB определены функции, вычисляющие площадь полигона и анализирующие нахождение точек внутри полигона. Для вычисления площади полигона используется функция:

polyarea(X,Y) — возвращает площадь полигона, определенную вершинами, находящимися в векторах **X** и **Y**. Если **X** и **Y** — матрицы одного размера, то **polyarea** возвращает площадь полигона, определенную столбцами **X** и **Y**.

polyarea(X,Y,dim) — возвращает площадь полигона, определенную столбцами или строками в зависимости от значения переменной **dim**.

Пример:

```
»L = linspace(0,3*pi,10);
» X = sin(L)';
» Y=cos(L)';
» A = polyarea(X,Y)
A =
   3.8971
» plot(X,Y,'m')
```

Построенный по этому примеру многоугольник представлен на рис. 7.7.

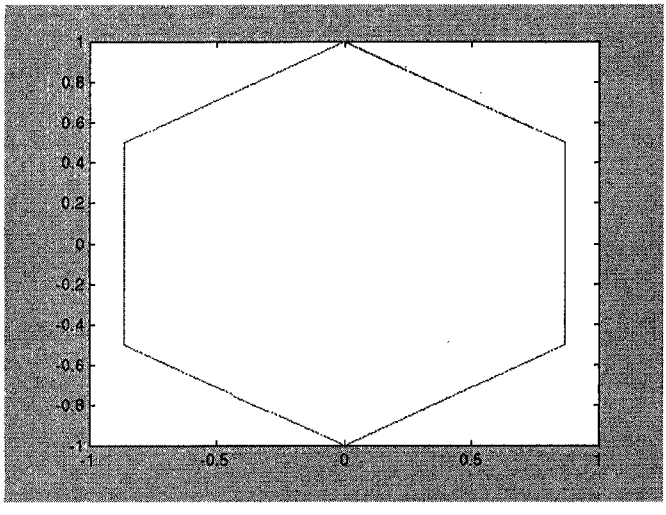


Рис. 7.7. Область многоугольника, для которого вычислена площадь

В данном примере использована функция **linspace(x1,x2,N)**, генерирующая N точек в промежутке от x1 до x2 с формированием векторов **X** и **Y** для построения многоугольника в полярной системе координат.

7.8.4. Функция анализа попадания точек внутрь полигона — inpolygon

Функция **inpolygon** используется для анализа того, попадают ли заданные точки внутрь полигона:

IN=inpolygon(X,Y,xv,yv) — возвращает матрицу **IN** того же размера, что и **X** и **Y**. Каждый элемент матрицы **IN** принимает одно из значений 1, 0.5 или 0 в зависимости от того, находится ли точка с координатами **(X(p,q),Y(p,q))** внутри полигона, вершины которого точно определяются векторами **xv** и **yv**:

IN(p,q) = 1 — если (X(p,q),Y(p,q)) внутри полигона;
IN(p,q) = 0.5 — если(X(p,q),Y(p,q)) на границе полигона;
IN(p,q) = 0 — если (X(p,q),Y(p,q)) вне полигона.

Пример:

```
» L = linspace(0,2*pi,8);
» yv = sin(L)';
» xv = cos(L)';
» x = randn(100,1); y = randn(100,1);
» IN = inpolygon(x,y,xv,yv);
» plot(xv,yv,'k',x(IN),y(IN),'r*',x(~IN),y(~IN),'bo')
```

Построенный по этому примеру массив точек и полигон представлены на рис. 7.8.

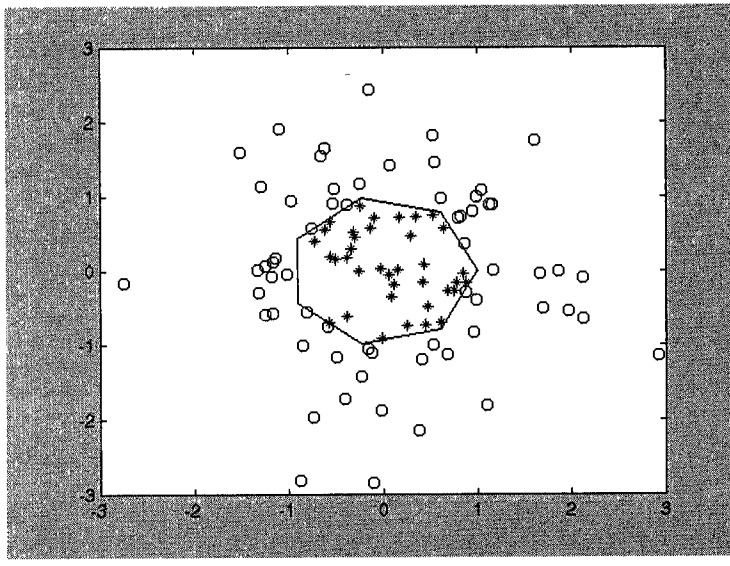


Рис. 7.8. Пример применения функции **inpolygon**

Точки, попавшие внутрь полигона, обозначены звездочками, а точки вне полигона обозначены кружками.

7.8.5. Построение диаграммы Вороного — команда voronoi

Для построения диаграммы Вороного служит следующая команда:

voronoi(x,y) — строит диаграмму Вороного для точек с координатами x,y. **voronoi(x,y,TRI)** использует триангуляцию **TRI**.

voronoi(...,'LineSpec') — строит диаграмму с заданным цветом и стилем линий.

[vx,vy] = voronoi(...) — возвращает вершины граней Вороного в векторах **vx** и **vy** так, что **plot(vx,vy,'–',x,y,'.')** создает диаграмму Вороного.

Примеры:

```
» rand('state',0);
» x = rand(1,15); y = rand(1,15);
» TRI = delaunay(x,y);
» subplot(1,2,1),...
» trimesh(TRI,x,y,zeros(size(x))); view(2),...
» axis([0 1 0 1]); hold on;
» plot(x,y,'o');
» [vx, vy] = voronoi(x,y,TRI);
» subplot(1,2,2),...
» plot(x,y,'r+',vx,vy,'b-'),...
» axis([0 1 0 1])
```

Рис. 7.9 (слева) иллюстрирует построение треугольников Делоне.

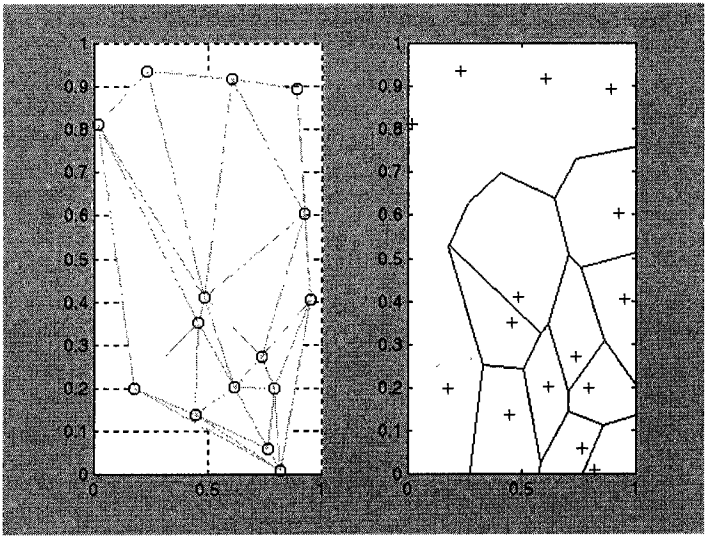


Рис. 7.9. Связь триангуляции Делоне с диаграммой Вороного

На рис. 7.9 (справа) знаками **+** изображены центры окружностей, проведенных вокруг каждого треугольника Делоне.

7.9. Корреляционный анализ

Под корреляцией понимается взаимосвязь некоторых зависимостей, представленных данными — векторами или матрицами. Общепринятой мерой этой связи является коэффициент корреляции. Его близость к 1 указывает на высокую степень идентичности зависимостей. Данный раздел посвящен описанию функции для вычисления коэффициентов корреляции и определения ковариационной матрицы элементов массива.

7.9.1. Вычисление коэффициентов корреляции — corrcoef

Приведенная ниже функция позволяет вычислить коэффициенты корреляции для входного массива данных.

corrcoef(X) — возвращает матрицу коэффициентов корреляции для входной матрицы, строки которой рассматриваются как наблюдения, а столбцы — как переменные. Матрица **S=corrcoef(X)** связана с матрицей ковариаций **C = cov(X)** следующим соотношением: **S(i,j)=C(i,j)/sqrt(C(i,i)C(j,j))**.

Функция **S = corrcoef(x,y)**, где **x** и **y** векторы-столбцы, аналогична функции **corrcoef([x y])**.

Пример:

```
» M=magic(5)
M =
   17   24    1    8   15
   23    5    7   14   16
    4    6   13   20   22
   10   12   19   21    3
   11   18   25    2    9
» S=corrcoef(M)
S =
    1.0000    0.0856   -0.5455   -0.3210   -0.0238
    0.0856    1.0000   -0.0981   -0.6731   -0.3210
   -0.5455   -0.0981    1.0000   -0.0981   -0.5455
   -0.3210   -0.6731   -0.0981    1.0000    0.0856
   -0.0238   -0.3210   -0.5455    0.0856    1.0000
```

В целом корреляция данных довольно низкая, за исключением данных, расположенных по диагонали — здесь коэффициенты корреляции равны 1.

7.9.2. Вычисление матрицы ковариации — функция cov

Приведенная ниже функция позволяет вычислить матрицу ковариации для массива данных.

cov(x) — возвращает дисперсию элементов массива **x**. Для матрицы, где каждая строка рассматривается как наблюдение, а каждый столбец — как переменная, **cov(x)** возвращает матрицу ковариаций. **diag(cov(x))** — вектор дисперсий для каждого столбца и **qrt(diag(cov(x)))** — вектор стандартных отклонений.

Функция **C = cov(x,y)**, где **x** и **y** — векторы-столбцы одинаковой длины, равносильна функции

cov([x y]).

Пример:

```
» D=[2 -3 6;3 6 -1;9 8 5];C=cov(D)
C =
  14.3333  16.3333   3.6667
  16.3333  34.3333 -10.3333
   3.6667 -10.3333  14.3333
» diag(cov(D))
ans =
  14.3333
  34.3333
  14.3333
```

7.10. Прямое и обратное преобразования Фурье

Разработка преобразований Фурье сыграла огромную роль в появлении и развитии ряда новых областей науки и техники. Достаточно отметить, что электрическая связь и радиосвязь базируются на спектральном представлении сигналов. Ряды Фурье можно также рассматривать как приближение произвольных функций (определенные ограничения в этом известны) тригонометрическими рядами бесконечной длины. При конечной длине рядов получаются наилучшие среднеквадратические приближения.

MATLAB содержит функции для выполнения одномерного и двумерного быстрых дискретных преобразований Фурье. Для одномерного массива **x** с длиной N прямое и обратное преобразования Фурье реализуются по формулам:

$$X(k) = \sum_{j=1}^{N} x(j) e^{2\pi/N(j-1)(k-1)}$$

$$x(k) = \frac{1}{N} \sum_{k=1}^{N} X(k) e^{-2\pi/N(j-1)(k-1)}$$

Прямое преобразование Фурье переводит описание сигнала (функции времени) из временной области в частотную, а обратное преобразование Фурье переводит описание сигнала из частотной области во временную. На этом основаны многочисленные методы фильтрации сигналов.

7.10.1. Функция одномерного прямого преобразования Фурье — fft

В описанных ниже функциях реализован особый метод **быстрого преобразования Фурье** — Fast Fourier Transform (**FFT** или **БПФ**), позволяющий резко уменьшить

число арифметических операций в ходе приведенных выше преобразований. Он особенно эффективен, если число обрабатываемых элементов (отсчетов) составляет 2^m, где m — целое положительное число.

Для одномерного преобразования используется следующая функция:

fft(X) — возвращает для вектора **X** дискретное преобразование Фурье, используя FFT- алгоритм быстрого преобразования Фурье. Если **X** — матрица, **fft** возвращает преобразование Фурье для каждого столбца матрицы.

fft(X,n) — возвращает n-точечное преобразование Фурье. Если длина вектора **X** меньше n, то недостающие элементы заполняются нулями. Если длина **X** больше n, то лишние элементы удаляются. Когда **X** — матрица, длина столбцов корректируется аналогично.

fft(X,[ ],dim) и **fft(X,n,dim)** — применяют преобразование Фурье к одной из размерностей массива в зависимости от величины dim.

Для иллюстрации применения преобразования Фурье создадим трехчастотный сигнал на фоне сильного шума, порожденного генератором случайных чисел:

```
t=0:0.0005:1;
x=sin(2*pi*200*t)+0.8*sin(2*pi*150*t)+0.8*sin(2*pi*250*t);
y=x+2*randn(size(t));
plot(y(1:100),'b')
```

Этот сигнал имеет среднюю частоту в 200 рад/с и два боковых сигнала с частотами 150 и 250 рад/с, что соответствует амплитудно-модулированному сигналу с частотой модуляции 50 рад/с и глубиной модуляции 0.8 (амплитуда боковых составляющих 0.8 от амплитуды центрального сигнала). На рис. 7.10 показан график этого сигнала. Нетрудно заметить, что из него никоим образом не видно, что полезный сигнал — амплитудно-модулированное колебание. Настолько оно забито шумами.

Теперь построим график спектральной плотности полученного сигнала с помощью прямого преобразования Фурье, по существу переводящего временное представление сигнала (рис. 7.10) в частотное:

```
» Y=fft(y,1024);
» Pyy=Y.*conj(Y)/1024;
» f=1000*(0:300)/1024;
» plot(f,Pyy(1:301)),grid
```

График спектральной плотности сигнала, построенный по этому примеру, представлен на рис. 7.11.

Даже беглого взгляда на рис. 7.11 достаточно, чтобы убедиться в том, что спектрограмма сигнала имеет явный пик на средней частоте амплитудно-модулированного сигнала и два боковых пика. Все эти три частотные составляющие сигнала явно выделяются на общем шумовом фоне. Таким образом данный пример наглядно иллюстрирует технику обнаружения слабых сигналов на фоне шумов, лежащую в основе работы радиоприемных устройств.

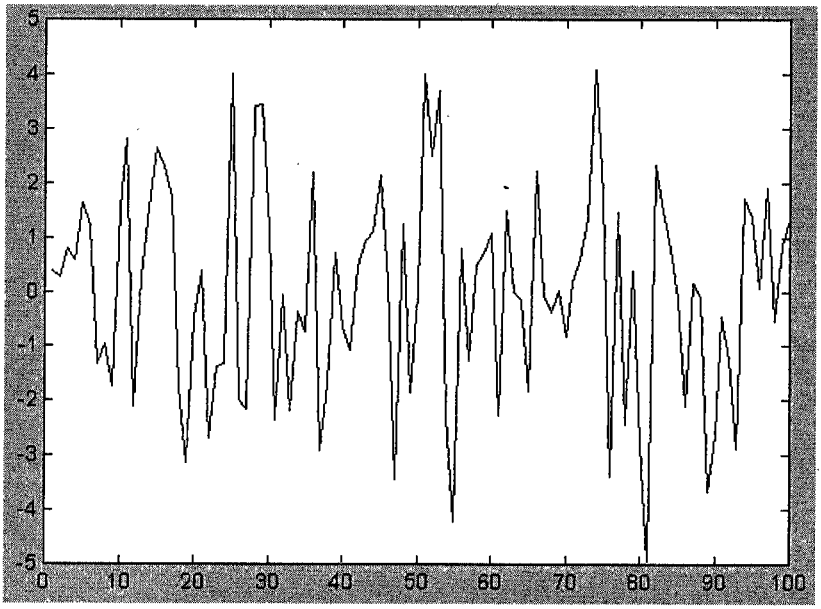


Рис. 7.10. Форма зашумленного сигнала

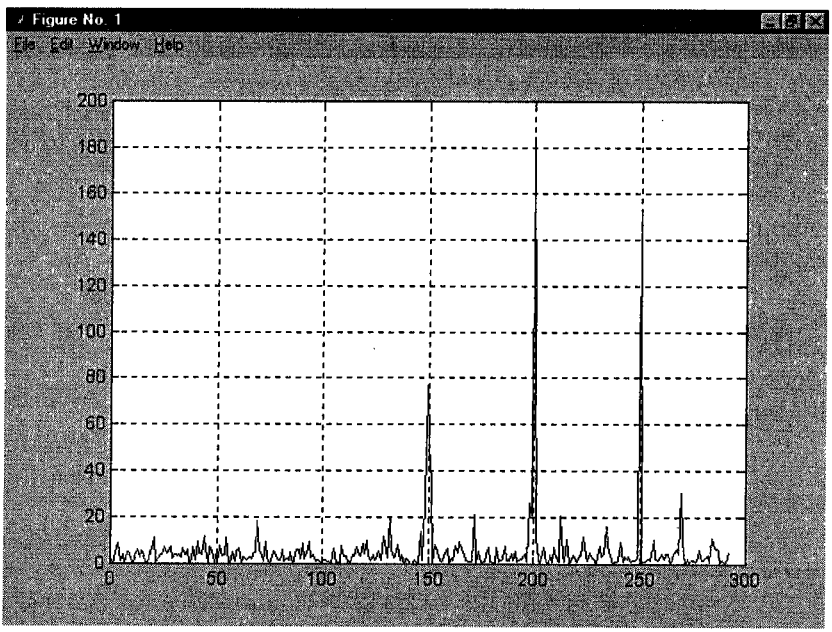


Рис. 7.11. График спектральной плотности приведенного на рис. 7.10 сигнала

7.10.2 Функции двумерного и многомерного прямого преобразования Фурье fft2 и fftn

Для двумерного прямого преобразования Фурье используется функция **fft2**:

fft2(X) — возвращает для массива данных **X** двумерное дискретное преобразование Фурье.
fft2(X,m,n) — усекает массив **X** или дополняет его нулями, чтобы создать mxn матрицу перед выполнением преобразования Фурье. Результат — матрица размера m×n.

В следующем разделе будет приведен пример применения функции **fft2**. Для многомерного прямого преобразования Фурье также существует функция:

fftn(X) — возвращает результат N-мерного дискретного преобразования для массива **X** размера N. Если **X** — вектор, то выход будет иметь ту же ориентацию.
fftn(X,SIZ) — возвращает результат дискретного преобразования для массива **X** с ограничением размера, заданным переменной SIZ.

7.10.3. Функция перегруппировки — fftshift

Функция **Y = fftshift(X)** перегруппировывает выходные массивы функции **fft** и **fft2**, размещая нулевую частоту в центре спектра, что иногда более удобно. Если **X** — вектор, то **Y** — вектор с циклической перестановкой правой и левой половины исходного вектора. Если **X** — матрица, то **Y** — матрица, у которой квадранты I и III меняются местами с квадрантами II и IV. Рассмотрим следующий пример. Вначале построим график спектральной плотности (рис. 7.12) при одномерном преобразовании Фурье:

```
» rand('state',0);
» t=0:0.001:0.6;
» x=sin(2*pi*50*t)+sin(2*pi*120*t);
» y=x+2*randn(size(t))+0.3;
» Y=fft(y);
» Pyy=Y.*conj(Y)/512;
» f=1000*(0:255)/512;
» plot(f,Pyy(1:256)),grid
```

Теперь воспользуемся функцией **fftshift**:

```
» Y=fftshift(Y);
» Pyy=Y.*conj(Y)/512;
» plot(Pyy),grid
```

Полученный при этом график представлен на рис. 7.13.

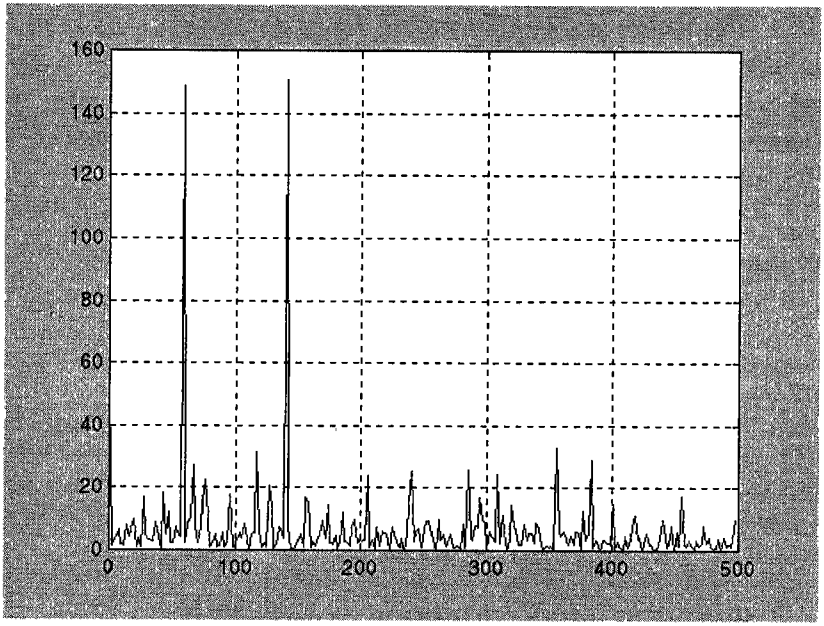


Рис. 7.12. График спектральной плотности сигнала после одномерного преобразования Фурье

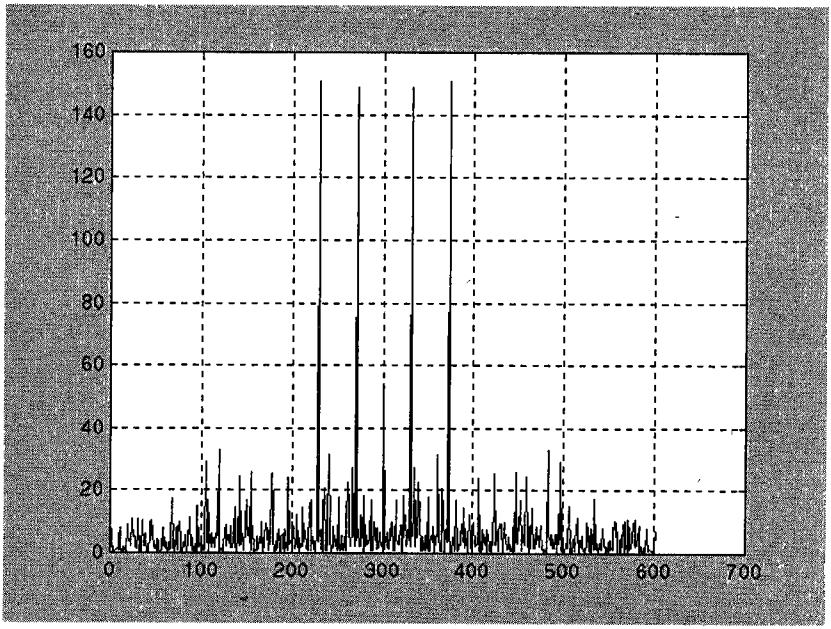


Рис. 7.13. График спектральной плотности того же сигнала после применения функции **fftshift**

7.10.4. Функции обратного преобразования Фурье ifft, ifft2 и ifftn

В случае одномерного преобразования возможно обратное преобразование Фурье, реализуемое следующей функцией:

ifft(F) — возвращает дискретное обратное быстрое преобразование Фурье вектора **F**. Если **F** — матрица, то **ifft** возвращает обратное преобразование Фурье для каждого столбца этой матрицы.

ifft(F,n) — возвращает n-точечное дискретное быстрое преобразование Фурье вектора **F**.

ifft(F,[ ],dim) и **y = ifft(X,n,dim)** — возвращают обратное дискретное преобразование Фурье массива **F** по строкам или по столбцам в зависимости от значения скаляра dim.

Для любого **X**, **ifft (fft (x))** приравнивается к **X**, с учетом погрешности округления. Если **X** — вещественный, **ifft (fft (x))** может иметь малые мнимые части. Пример:

```
» V=[1 1 1 1 0 0 0 0];
» fft(V)
ans =
 Columns 1 through 4
  4.0000          1.0000 - 2.4142i       0          1.0000 - 0.4142i
 Columns 5 through 8
     0            1.0000 + 0.4142i       0          1.0000 + 2.4142i
» ifft(fft(V))
ans =
   1   1   1   1   0   0   0   0
```

Аналогичные функции есть для двумерного и многомерного случаев:

ifft2(F) — возвращает двумерное дискретное обратное быстрое преобразование Фурье для матрицы **F**.

ifft2(F,m,n) — возвращает m×n обратное быстрое преобразование Фурье для матрицы **F**.

ifftn(F) — возвращает результат N-мерного обратного дискретного преобразования Фурье для N-мерного массива **F**.

ifftn(F,SIZ) — возвращает результат обратного дискретного преобразования для массива **F** с ограничением размера, заданным переменной SIZ. Если любой элемент SIZ меньше, чем соответствующая размерность **F**, то массив **F** будет урезан до размерности SIZ.

7.11. Операции свертки и фильтрации

В этом разделе рассмотрены базовые средства для проведения операций свертки и фильтрации сигналов на базе алгоритмов быстрого преобразования Фурье. Многие

дополнительные операции, относящиеся к этой области обработки сигналов, можно найти в пакете прикладных программ Signal Processing Toolbox.

7.11.1. Функция свертки conv и обратная ей функция deconv

Для двух векторов **x** и **y** с длиной m и n определена операция свертки:

$$z(k) = \sum_{j=\max(1,k-n+1)}^{\min(k,m)} x(j)y(k-j+1).$$

В ее результате получается вектор **z** с длиной (m+n+1), который на основании известной теоремы свертки можно считать равным **ifft(x.*y)**. Для осуществления свертки используется функция **conv(x,y)**.

Обратная свертке функция определена как **[q,r]=deconv(z,x)**. Она фактически определяет импульсную характеристику фильтра. Если **z=conv(x,y)**, то **q=y**, **r=0**. Если **x** и **y** вектора с коэффициентами полиномов, то свертка эквивалентна перемножению полиномов.

7.11.2. Функция свертки двумерных массивов conv2

Для двумерных массивов также существует функция свертки:

Z=conv2(X,Y) и **Z=conv2(X,Y,'option')**.

Для двумерных массивов **X** и **Y** с размером mx×nx и my×ny, соответственно, результат двумерной свертки порождает массив размера (mx+my-1)×(nx+ny-1). Во второй форме функции опция может иметь следующее значение: **'full'** — полноразмерная свертка (используется по умолчанию), **'same'** — центральная часть размера mx×nx, **'valid'** — центральная часть размера (mx+my-1)×(nx+ny-1), если (mx×nx)>(my×ny).

Возможность изменить решение или трактовку данных с помощью опций является свойством ряда функций системы MATLAB. Позже мы столкнемся с этой возможностью еще ни раз.

7.11.3. Дискретная одномерная фильтрация — функция filter

MATLAB может использоваться для моделирования работы цифровых фильтров. Для обеспечения дискретной одномерной фильтрации может использоваться функция **filter** в следующих формах записи:

filter(B,A,X) — фильтрует одномерный массив данных **X**, используя дискретный фильтр, описываемый конечно-разностным уравнением:

a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb) - a(2)*y(n-1) - ... - a(na+1)*y(n-na). Если **a(1)** не равно 1, то коэффициенты уравнения нормализуются относительно **a(1)**. Когда

X — матрица, **filter** оперирует со столбцами **X**. Возможна фильтрация многомерного (размерности N) массива.

[Y,Zf]=filter(B,A,X,Zi) — выполняет фильтрацию с учетом запаздывания входного Zi и выходного Zf сигналов.

filter(B,A,X,[ ],DIM) или **filter(B,A,X,Zi,DIM)** — работают в направлении размерности DIM.

Рассмотрим типовой пример фильтрации гармонического сигнала на фоне других сигналов — файл с именем filtdem.m из пакета расширения Signal Processing Toolbox. На рис. 7.14 представлен кадр примера, на котором показано формирование входной совокупности сигналов в виде трех сигналов с частотами 5, 15 и 30 Гц. Показана временная зависимость сигнала и под ней команды, которые надо выполнить для этого этапа примера.

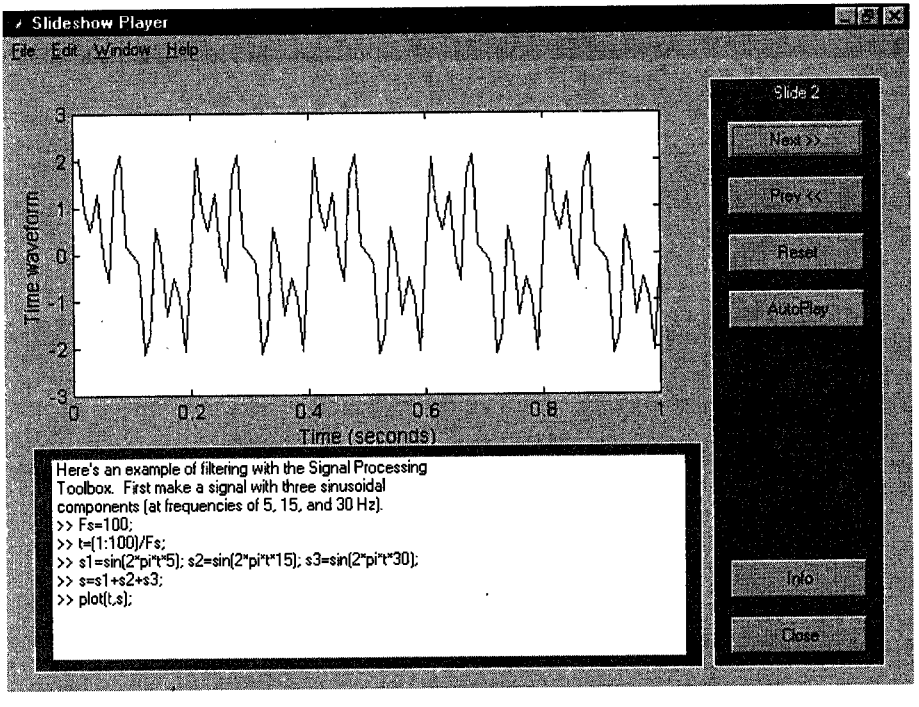


Рис. 7.14. Формирование сигнала и построение его графика

Следующий кадр (рис. 7.15) иллюстрирует конструирование фильтра с достаточно плоской вершиной амплитудно-частотной характеристики (АЧХ) и полосой частот, обеспечивающей выделение сигнала с частотой 15 Гц и подавление сигналов с часто-

тами 5 и 30 Гц. Для формирования полосы пропускания фильтра используется функция **ellip**, а для построения АЧХ функция **freqz** из пакета Signal Processing Toolbox. Это позволяет построить график АЧХ созданного фильтра.

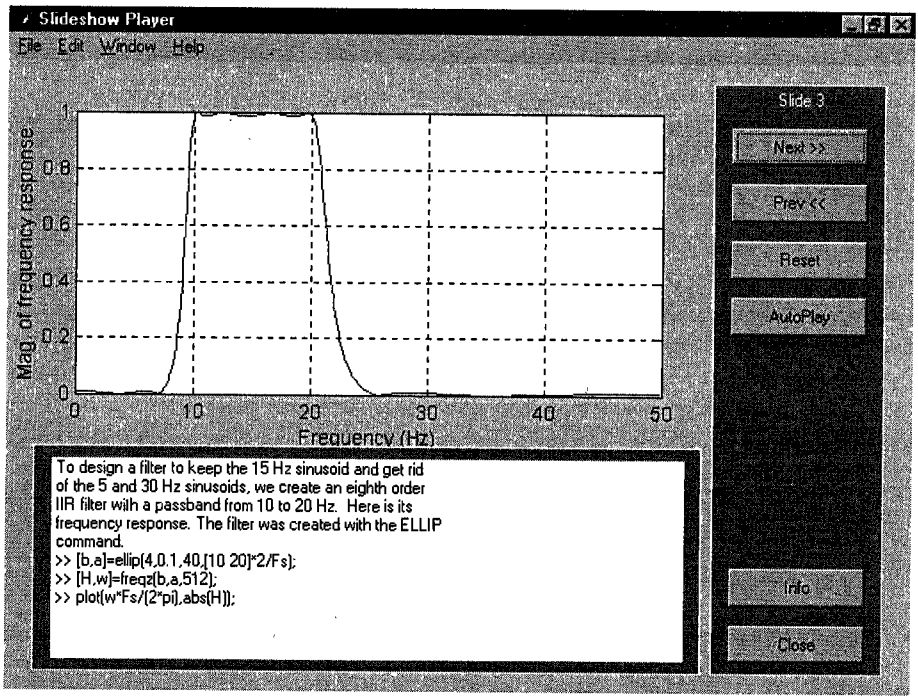


Рис. 7.15. Конструирование фильтра с заданной полосой частот и его АЧХ

Следующий кадр примера (рис. 7.16) иллюстрирует эффективность выделения сигнала заданной частоты (15 Гц) с помощью операции фильтрации — функции **filter**, описанной выше. Можно заметить два обстоятельства — полученный стационарный сигнал практически синусоидален, что свидетельствует о высокой степени фильтрации побочных сигналов. Однако нарастание сигнала во времени идет достаточно медленно и занимает несколько периодов частоты полезного сигнала. Характер нарастания сигнала во времени определяется переходной характеристикой фильтра.

Заключительный кадр (рис. 7.17) показывает спектр исходного сигнала и спектр сигнала на выходе фильтра (он показан линиями другого цвета, что, к сожалению, не видно на черно-белом рисунке). Для построения спектров используется прямое преобразование Фурье — функция **fft**.

Этот пример наглядно иллюстрирует технику фильтрации. Рекомендуется просмотреть дополнительные примеры, которые есть в разделе Demos системы по разделу Signal Processing.

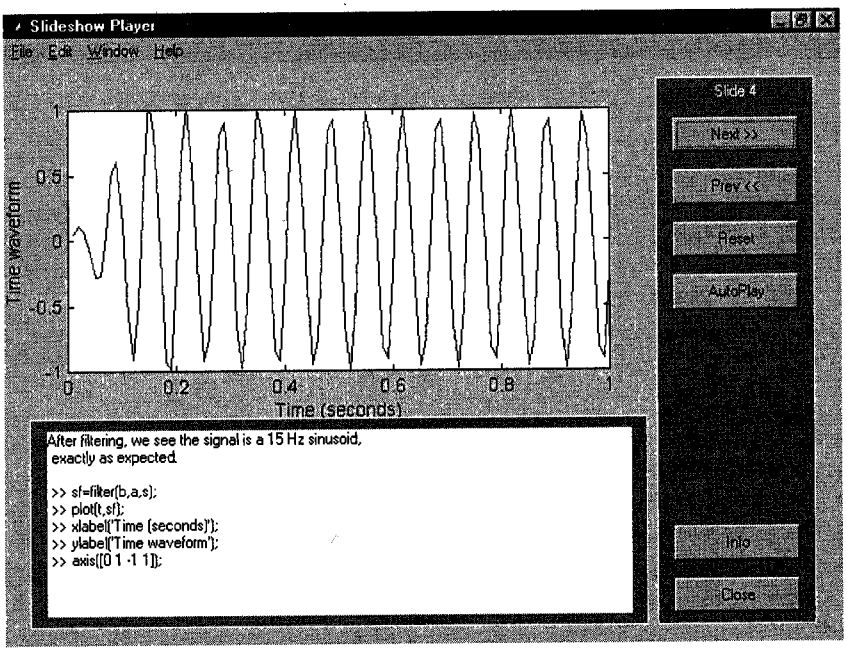


Рис. 7.16. Фильтрация и ее результат в виде временной зависимости сигнала на выходе фильтра

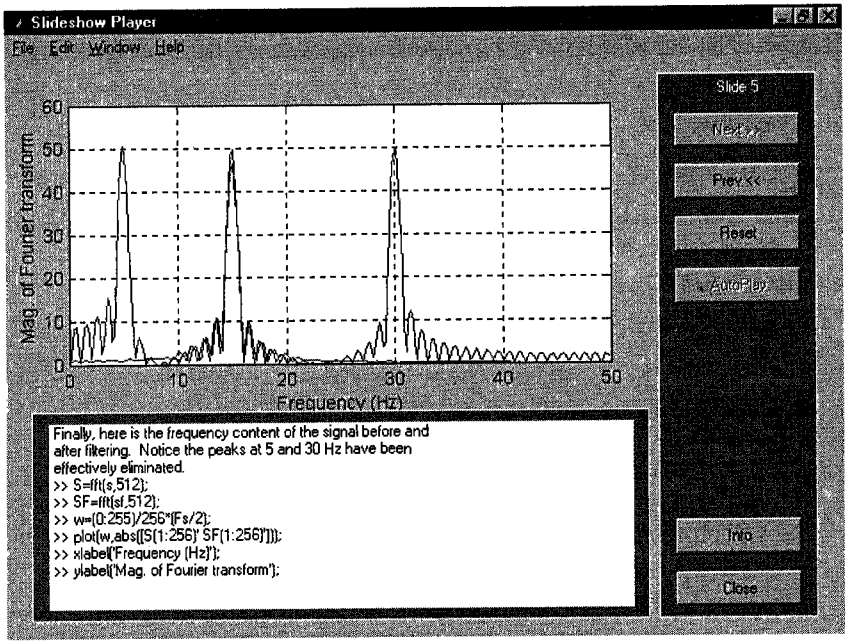


Рис. 7.17. Анализ спектров сигналов на входе и на выходе фильтра и построение их спектров

7.11.4. Двумерная фильтрация — функция filter2

Для осуществления двумерной фильтрации служит функция:

filter2(B,X) — фильтрует данные в двумерном массиве **X**, используя дискретный фильтр, описанный матрицей **B**. Результат **Y** имеет те же размеры, что и **X**.

filter2(B,X,'option') — выполняет то же, но с опцией, влияющей на размер массива **Y**: **'same'** — **size(Y)=size(X)** (действует по умолчанию), **'valid'** — **size(Y) < size(X)**, **'full'** — **size(Y) > size(X)**.

7.11.5. Функция коррекции фазовых углов unwrap

Фазовые углы одномерных массивов испытывают разрывы при переходе через значения, кратные π. Функции **unwrap(P)** и **unwrap(P,cutoff)** устраняют этот недостаток для одномерного массива P, дополняя значения углов в точках разрыва значениями $\pm 2\pi$. Если **P**-двумерный массив, то данная функция применяется к столбцам. Параметр cutoff (по умолчанию равный π) позволит назначить любой критический угол в точках разрыва. Функции используются при построении фазо-частотных характеристик (ФЧХ) фильтров. Поскольку они строятся редко, оставим за читателем изучение практического применения данной функции.

7.12. Полиномы и операции над ними

Полиномы (у нас их принято называть степенными многочленами) — широко известный объект для математических вычислений и обработки данных. Обычно полином записывается в виде

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0,$$

хотя и возможны иные формы записи, например:

$$p(x) = a_1 x^n + a_2 x^{n-1} + \ldots + a_n x + a_{n+1}.$$

Широкое применение полиномов отчасти обусловлено большими возможностями полиномов в представлении данных и их простотой и единообразием вычислений. В этой главе описаны основные функции для работы с полиномами. При этом обычно полиномы задаются вектором их коэффициентов.

7.12.1. Умножение и деление полиномов — функции conv и deconv

Ниже приведены функции, осуществляющие умножение и деление полиномов, или, что то же самое, свертку двух входных векторов, в которых находятся коэффициенты полинома, и операцию, обратную свертке.

w = conv(u,v) — возвращает свертку векторов **u** и **v**. Алгебраически свертка то же самое, что и произведение полиномов, чьи коэффициенты — элементы векторов **u** и **v**. Если длина вектора **u** равна m, а длина вектора **v** — n, то вектор **w** имеет длину m+n-1 и k -ый элемент его вычисляется по формуле:

$$w(k) = \sum_j u(j)v(k+1-j).$$

Пример:

```
» f=[2,3,5,6];d=[7,8,3];r=conv(f,d)
r =
  14  37  65  91  63  18
```

[q,r] = deconv(v,u) — возвращает результат деления полинома **v** на полином **u**. Вектор **q** представляет собой частное от деления, а **r** — остаток от деления , так что выполняется соотношение **v=conv(u,q)+r**. Пример:

```
» t=[14,37,65,91,63,18];r=[7,8,3];[w,e]=deconv(t,r)
w =
  2.0000  3.0000  5.0000  6.0000
e =
 1.0e-013
    0     0  0.1421  -0.1421  -0.2132  -0.1066
```

7.12.2. Вычисление полинома — функции poly, polyval, polyvalm

В этом разделе приведены функции вычисления коэффициентов характеристического полинома, значения полинома в точке и матричного полинома.

poly(A) — для матрицы **A** размера n×n возвращает вектор-строку размером n+1, элементы которого являются коэффициентами характеристического полинома **det(sI – A)**. Коэффициенты упорядочены по убыванию степеней. Если вектор состоит из n+1 компонент, то ему соответствует полином вида $c_1s^n+...+c_ns+c_{n+1}$.

poly(r) — для вектора **r** возвращает вектор-строку с элементами, представляющими собой коэффициенты полинома, корнями которого являются элементы вектора **r**.

Примеры:

```
A =
   2   3   6
   3   8   6
   1   7   4
» d=poly(A)
d =
  1.0000  -14.0000  -1.0000  -40.0000
```

```
» A=[3,6,8;12,23,5;11,12,32]
A =
   3    6    8
  12   23    5
  11   12   32
» poly(A)
ans =
   1.0000  -58.0000  681.0000  818.0000
```

polyval(p,x) — возвращает значения полинома **p** в точках, заданных в массиве **x**. Полином **p** — вектор, элементы которого являются коэффициентами полинома в порядке уменьшения степеней. **x** может быть матрица или вектор. В любом случае, функция **polyval** вычисляет значения полинома **p** для каждого элемента **x**.

[y,delta] = polyval(p,x,S) — использует структуру **S**, возвращенную функцией **polyfit** для оценки погрешности аппроксимации **y±delta**.

Пример:

```
» p=[3,0,4,3];d=polyval(p,[2,6])
d =
   35  675
```

polyvalm(p,X) — вычисляет значения полинома для матрицы. Это эквивалентно подстановке матрицы **X** в полином **p**. Полином **p** — вектор, чьи элементы являются коэффициентами полинома в порядке уменьшения степеней, а **X** — квадратная матрица.

Пример:

```
» D=pascal(5)
D =
   1   1   1   1   1
   1   2   3   4   5
   1   3   6  10  15
   1   4  10  20  35
   1   5  15  35  70
» f=poly(d)
f =
   1.0000  -99.0000  626.0000 -626.0000   99.0000   -1.0000
» polyvalm(f,D)
ans =
 1.0e-006 *
  -0.0003   -0.0011   -0.0038   -0.0059   -0.0162
  -0.0012   -0.0048   -0.0163   -0.0253   -0.0692
  -0.0034   -0.0131   -0.0447   -0.0696   -0.1897
  -0.0076   -0.0288   -0.0983   -0.1529   -0.4169
  -0.0145   -0.0551   -0.1883   -0.2929   -0.7984
```

7.12.3. Вычисление производной полинома — функция polyder

Ниже приведена функция, возвращающая производную полинома **p**:

polyder(p) — возвращает производную полинома **p**.
polyder(a,b) — возвращает производную от произведения полиномов **a** и **b**.
[q,d] = polyder(b,a) — возвращает числитель **q** и знаменатель **d** производной отношения полиномов **b/a.**

Примеры:

```
» a=[3,5,8];b=[5,3,8];dp=polyder(a)
dp =
   6   5
» dt=polyder(a,b)
dt =
  60  102  158   64
» [q,p]=polyder(b,a)
q =
  16   32  -16
p =
   9   30   73   80   64
```

7.12.4. Решение полиномиальных матричных уравнений — функция polyeig

Приведенная ниже функция вычисляет собственные значения матричного полинома.

[X,e] = polyeig(A0,A1,...Ap) — решает задачу собственных значений для матричного полинома степени p вида:

$$A_0 + \lambda A_1 + ... + \lambda^p A_p \; x = 0,$$

где степень полинома p — целое неотрицательное число, а $A_0, A_1, ... A_p$ — входные матрицы порядка n. Выходная матрица **X** размера n×n×p содержит собственные векторы. Вектор **e** длиной n×p содержит собственные значения. Пример:

```
» A=[1:4;5:8;9:12;13:16]
A =
   1   2   3   4
   5   6   7   8
   9  10  11  12
  13  14  15  16
» B=[4:7;2:5;10:13;23:26]
```

```
B =
    4    5    6    7
    2    3    4    5
   10   11   12   13
   23   24   25   26
» [F,a]=polyeig(A,B)
F =
   0.4373   0.0689  -0.5426  -0.7594
  -0.3372  -0.4969   0.6675  -0.1314
  -0.6375   0.7870   0.2927   0.3771
   0.5374  -0.3591  -0.4176   0.5136
a =
   4.4048
   0.4425
  -0.3229
  -1.0000
```

7.12.5. Разложение на простые дроби — функция residue

Для отношения полиномов **b** и **a** функция

[r,p,k] = residue(b,a)

возвращает вычеты, полюса и многочлен целой части отношения двух полиномов **b(s)** и **a(s)** в виде:

$$\frac{b(s)}{a(s)}=\frac{b_1+b_2s^{-1}+b_3s^{-2}+...+b_{m+1}s^{-m}}{a_1+a_2s^{-1}+a_3s^{-2}+...+a_{n+1}s^{-n}}$$

[b,a] = residue(r,p,k) — выполняет обратную свертку разложения элементарной дроби в полиномы с коэффициентами в векторах **b** и **a**. Пример:

```
» b=[4,3,1];a=[1,3,7,1];[r,p,k]=residue(b,a)
r =
  1.9484 + 0.8064i
  1.9484 - 0.8064i
  0.1033
p =
 -1.4239 + 2.1305i
 -1.4239 - 2.1305i
 -0.1523
k =
     []
» [b1,a1]=residue(r,p,k)
b1 =
```

```
   4.0000   3.0000   1.0000
a1 =
   1.0000   3.0000   7.0000   1.0000
```

7.12.6. Вычисление корней полинома — функция roots

Приведенная ниже функция вычисляет корни (в том числе комплексные) для полинома вида:

$$c_1 s^n + \ldots + c_n s + c_{n+1}.$$

roots(c) — возвращает вектор-столбец, чьи элементы являются корнями полинома **c**.

Вектор-строка **c** содержит коэффициенты полинома, упорядоченные по убыванию степеней. Если **c** имеет **n+1** компоненту, то полином, им представленный, имеет вид: $c_1 s^n + \ldots + c_n s + c_{n+1}$.

Пример:

```
» x=[7,45,12,23];d=roots(x)
d =
  -6.2382
  -0.0952 + 0.7195i
  -0.0952 - 0.7195i
```

7.13. Аппроксимация и интерполяция данных

Для обработки данных MATLAB использует различные функции интерполяции и аппроксимации данных. Набор таких функций вместе с несколькими вспомогательными функциями описан в этом разделе.

7.13.1. Полиномиальная аппроксимация — функция polyfit

Под аппроксимацией обычно подразумевается описание некоторой, порой не заданной явно зависимости или совокупности представляющих ее данных с помощью другой, обычно более простой или более единообразной зависимости. Одна из наиболее известных аппроксимаций — полиномиальная. В системе MATLAB определены функции аппроксимации данных полиномом по методу наименьших квадратов. Это выполняет функция, приведенная ниже:

polyfit(x,y,n) — возвращает вектор коэффициентов полинома p(x) степени n, который с наименьшей среднеквадратичной погрешностью аппроксимирует функцию y(x). Результатом является вектор-строка длиной **n+1** содержащий коэффициенты полинома в порядке уменьшения степеней. Если число элементов векторов **x** и **y** равно **n+1**, то

реализуется обычная полиномиальная аппроксимация, при которой график полинома точно проходит через узловые точки с координатами (x,y), хранящимися в векторах **x** и **y**. В противном случае точное совпадение графика с узловыми точками не наблюдается.

[p,s] = polyfit(x,y,n) — возвращает коэффициенты полинома **p** и структуру **S** для использования вместе с функцией **polyval** для оценки или предсказания погрешности.

Пример (полиномиальная регрессия для функции sin(x)):

```
» x=(-3:0.2:3)';y=sin(x);p=polyfit(x,y,3)
p =
  -0.0953   0.0000   0.8651  -0.0000
» x=(-4:0.2:4)';y=sin(x);
» f=polyval(p,x);plot(x,y,'o',x,f)
```

Рис. 7.18, построенный по этому примеру, дает наглядное представление о точности полиномиальной аппроксимации. Следует помнить, что она достаточна в небольших окрестностях от точки x=0, но может иметь большие погрешности за их пределами или в промежутках между узловыми точками.

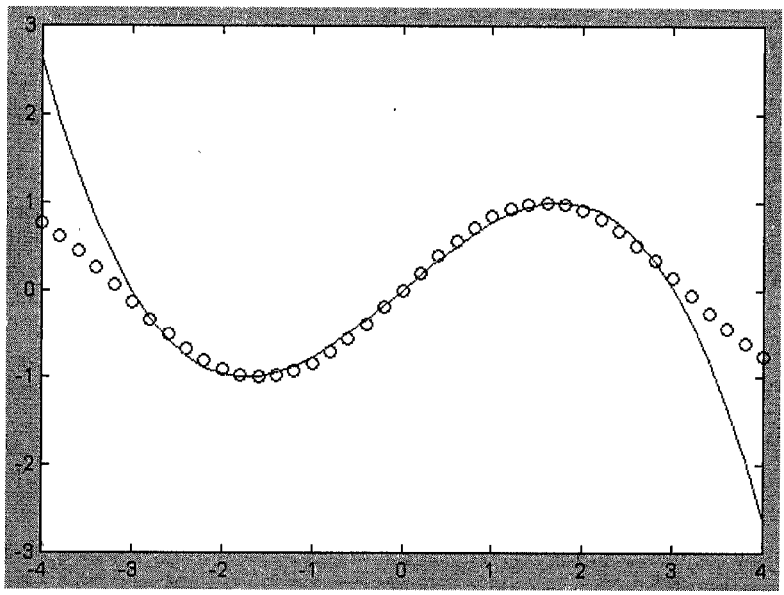


Рис. 7.18. Пример использования функции **polyfit**

График аппроксимирующего полинома третьей степени на рис. 7.18 показан сплошной линией, а точки исходной зависимости обозначены кружками. К сожалению, при степени полинома свыше 5 погрешность полиномиальной регрессии (и аппроксимации) сильно возрастает и ее применение становится рискованным. Обратите внимание на то, что при полиномиальной регрессии узловые точки не ложатся точно на график полинома, поскольку их приближение к нему является наилучшим в среднеквадратическом приближении. Об этом уже говорилось.

7.13.2. Интерполяция периодических функций рядом Фурье — функция interpft

Под интерполяцией обычно подразумевают вычисление значений функции f(x) в промежутках между узловыми точками. Линейная, квадратическая и полиномиальная интерполяции реализуются при полиномиальной аппроксимации. А вот для периодических (особенно гладких) функций хорошие результаты может дать интерполяция тригонометрическим рядом Фурье. Для этого используется следующая функция:

interpft(x,n) — возвращает вектор **y**, содержащий значения периодической функции, определенный в **n** равномерно расположенных точках. Если **length(x) = m** и **x** имеет интервал дискретности **dx**, то интервал дискретности для **y** — **dy** = **dx *m/n**, причем **n** не может быть меньше чем **m**. Если **X** — матрица, **interpft** оперирует над столбцами **X**, возвращая матрицу **Y** с таким же числом столбцов, как и у **X**, но с **n** строками. Функция **y = interpft(x,n,dim)** работает либо со строками, либо со столбцами в зависимости от величины **dim**.

Пример:

```
» x=0:10;y=sin(x).^3;
» x1=0:0.1:10;y1=interpft(y,101);
» x2=0:0.01:10;y2=sin(x2).^3;
» plot(x1,y1,'--'),hold on,plot(x,y,'o',x2,y2)
```

Рис. 7.19 иллюстрирует эффективность данного вида интерполяции на примере функции **sin(x).^3**, которая представляет собой сильно искаженную синусоиду.

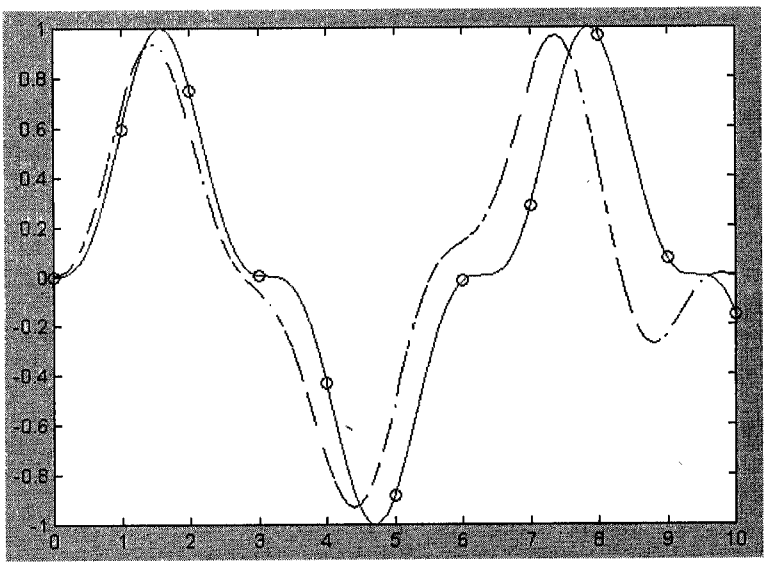


Рис. 7.19. Пример использования функции **interpft**

Исходная функция на рис. 7.19 представлена сплошной линией с кружками, а интерполирующая функция — штрих- пунктирной линией.

7.13.3. Интерполяция на неравномерной сетке — функция griddata

Для интерполяции на неравномерной сетке используется функция, приведенная ниже:

Zl = griddata(x,y,z,Xl,Yl) — преобразует поверхность вида **z = f(x,y)**, которая определяется векторами **(x,y,z)** с обычно неравномерно распределенными элементами. Функция **griddata** аппроксимирует эту поверхность в точках, определенных векторами (**Xl,Yl**) в виде значений **Zl**. Поверхность всегда проходит через заданные точки. **Xl** и **Yl** обычно формируют однородную сетку (созданную с помощью функции **meshgrid**).

Xl может быть вектор-строка, в этом случае он определяет матрицу с постоянными столбцами, Точно так же **Yl** может быть вектор-столбец, и он определяет матрицу с постоянными строками.

[Xl,Yl,Zl] = griddata(x,y,z,xi,yi) — возвращает аппроксимирующую матрицу **Zl**, как описано выше, а также возвращает матрицы **Xl** и **Yl** , сформированные из вектора-столбца **xi** и вектора-строки **yi**. Последние аналогичны матрицам, возвращаемым функцией **meshgrid**.

[...] = griddata(...,method) — использует определенный метод интерполяции:

'nearest' — ступенчатая интерполяция; **'linear'** — линейная интерполяция (принята по умолчанию); **'spline'** — кубическая сплайн-интерполяция; **'cubic'** — кубическая интерполяция.

Метод определяет тип аппроксимирующей поверхности. Методы **'cubic'** и **'invdist'** формируют гладкие поверхности, в то время как **'linear'** и **'nearest'** имеют разрывы первых и нулевых производных соответственно.

Пример:

```
» x=rand(120,1)*4-2;y=rand(120,1)*4-2;z=x.*y.*exp(-x.^2-y.^2);
» t=-2:0.1:2;[X,Y]=meshgrid(t,t);Z=griddata(x,y,z,X,Y);
» mesh(X,Y,Z),hold on,plot3(x,y,z,'ok')
```

Рис. 7.20 иллюстрирует применение функции **griddata**.

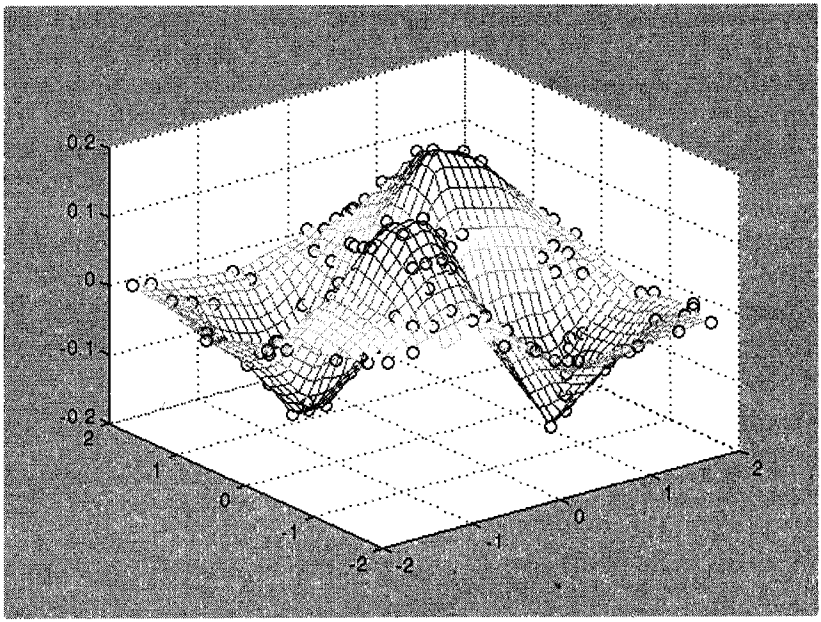


Рис 7.20. Пример использования функции **griddata**

7.13.4. Одномерная табличная интерполяция — interp1

Часто данные задают в виде отдельных узловых точек, координаты которых задаются таблицей данных. Задача интерполяции — найти данные в окрестности узловых точек. Для этого используются подходящие функции, значения которых в узловых точках совпадают с координатами этих точек. Например, при **линейной интерполяции** зависимости y(x) узловые точки соединяются друг с другом отрезками прямых, и считается, что искомые промежуточные точки расположены на этих отрезках. Для повышения точности интерполяции применяют параболы (квадратичная интерполяция) или полиномы более высокой степени (полиномиальная интерполяция).

В ряде случаев очень удобна сплайновая интерполяция и аппроксимация таблично заданных функций. При ней промежуточные точки ищутся по отрезкам полиномов третьей степени — **кубическая сплайновая интерполяция**. При этом обычно такие полиномы вычисляются так, чтобы не только значения их совпадали с координатами узловых точек, но и чтобы в смежных точках были непрерывны производные первого и второго порядков. Такое поведение характерно для гибкой линейки, закрепленной в узловых точках — откуда и название splain (сплайн) для этого вида интерполяции (аппроксимации).

Для одномерной табличной интерполяции используется функция:

yi = interp1(x,Y,xi) — возвращает вектор **yi**, содержащий элементы, соответствующие элементам **xi** и полученные интерполяцией векторов **x** и **Y**. Вектор **x** опреде-

ляет точки, в которых задано значение **Y**. Если **Y** — матрица, то интерполяция выполняется для каждого столбца **Y**, и **yi** имеет длину **length(xi)-by-size(Y,2).**

yi = interp1(x,Y,xi,method) — позволяет с помощью опции mathod задать метод интерполяции:

'nearest' — ступенчатая интерполяция; **'linear'** — линейная интерполяция (принята по умолчанию); **'spline'** — кубическая сплайн-интерполяция; **'cubic'** — кубическая интерполяция.

Все методы интерполяции требуют, чтобы значения **x** изменялись монотонно. Для более быстрой интерполяции, когда **x** — вектор равномерно распределенных точек, лучше использовать методы **'*linear', '*cubic', '*nearest'** или **'*spline'**. Обратите внимание, что в данном случае наименованию метода предшествует знак звездочки. Пример (интерполяция функции косинуса):

```
» x=0:10;y=cos(x);
» xi=0:0.1:10;
» yi=interp1(x,y,xi);
» plot(x,y,'x',xi,yl,'g'),hold on
» yi=interp1(x,y,xi,'spline');
» plot(x,y,'o',xi,yi,'m'),grid,hold off
```

Результаты интерполяции иллюстрирует рис.7.21.

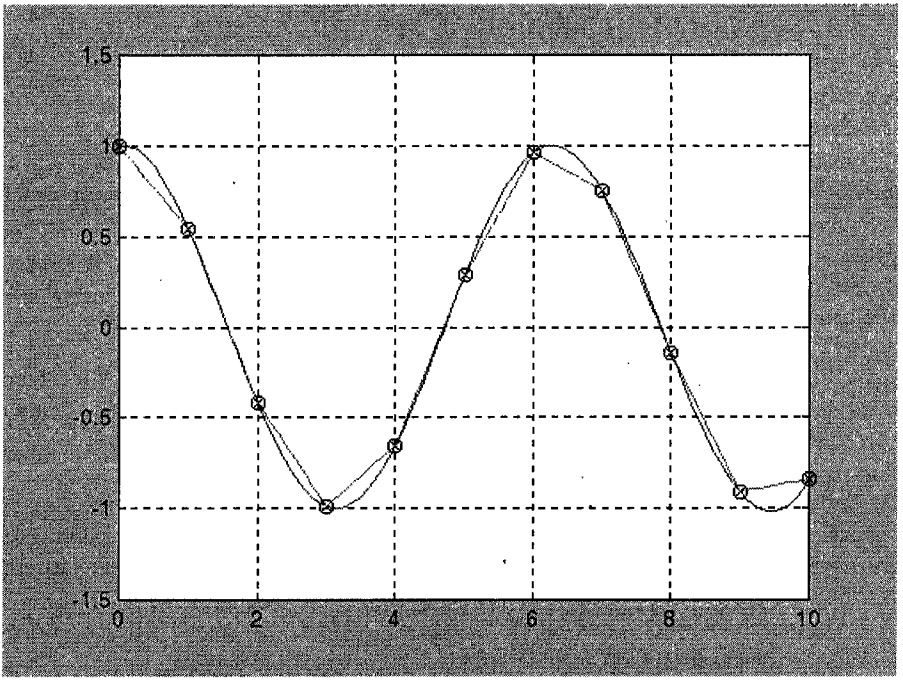


Рис. 7.21. Пример применения функции **interp1**

Узловые точки на рис. 7.21 обозначены кружками с наклонными крестиками. Одна из кривых соответствует линейной интерполяции, другая — сплайн-интерполяции. Нетрудно подметить, что сплайн-интерполяция в данном случае дает гораздо лучшие результаты, чем линейная интерполяция. При последней — точки просто соединяются друг с другом отрезками прямых, так что график интерполирующей кривой при линейной интерполяции получается не гладким.

7.13.5. Двумерная табличная интерполяция — interp2

Двумерная интерполяция существенно сложнее, чем одномерная, рассмотренная выше, хотя смысл ее тот же — найти промежуточные точки вблизи расположенных в пространстве узловых точек некоторой зависимости z(x,y). Для двумерной табличной интерполяции используется функция:

Zl = interp2(X,Y,Z,Xl,Yl) — возвращает матрицу **Zl**, содержащую элементы, соответствующие элементам **Xl** и **Yl**, и полученную интерполяцией двумерной функции, заданной матрицами **X**, **Y** и **Z**. **X** и **Y** должны быть монотонными и иметь тот же формат (**«plaid»**), как если бы они были получены с помощью функции **meshgrid**. Матрицы **X** и **Y** определяют точки, в которых задано значение **Z** .**Xl** и **Yl** могут быть матрицами, в этом случае **interp2** возвращает значения **Z**, соответствующие точкам **(Xl(i,j),Yl(i,j))**. В качестве альтернативы можно передать вектор-строку и вектор-столбец **xi** и **yi** соответственно. В этом случае **interp2** представляет эти векторы, как если бы использовалась команда **meshgrid(xi,yi).**

Zl = interp2(Z,Xl,Yl) — подразумевает, что **X** = 1:n и **Y** = 1:m, где **[m,n] =size(Z).**

Zl = interp2(Z,ntimes) — осуществляет интерполяцию рекурсивным методом с числом шагов ntimes.

Zl = interp2(X,Y,Z,Xl,Yl,method) — позволяет с помощью опции method задать метод интерполяции:

'nearest' — интерполяция по соседним точкам;

'linear' — линейная интерполяция;

'cubic' — кубическая интерполяция.

Все методы интерполяции требуют, чтобы **X** и **Y** изменялись монотонно и имели такой же формат (**«plaid»**), как если бы они были получены с помощью функции **meshgrid**. Для более быстрой интерполяции, когда **X** и **Y** — векторы равномерно распределенных точек, лучше использовать методы **'*linear', '*cubic'** или **'*nearest'**.

Пример:

```
» [X,Y]=meshgrid(-3:0.25:3);Z=peaks(X/2,Y*2);
» [X1,Y1]=meshgrid(-3:0.1:3);Z1=interp2(X,Y,Z,X1,Y1);
» mesh(X,Y,Z),hold on,mesh(X1,Y1,Z1+15),hold off
```

Рис. 7.22 иллюстрирует применение функции **interp2** для двумерной интерполяции (на примере функции **peaks**).

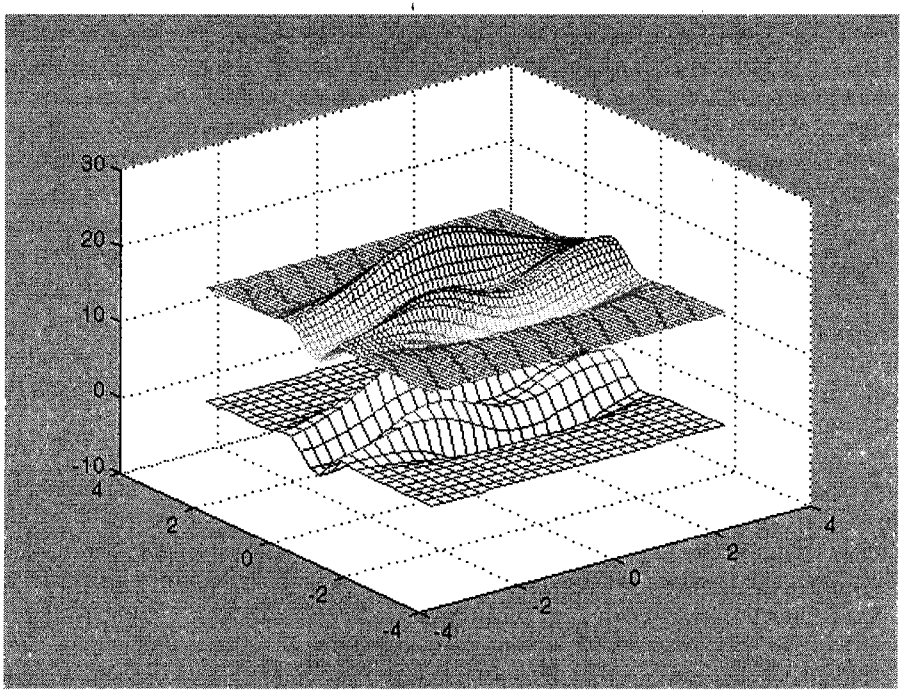


Рис 7.22. Применение функции **interp2**

В данном случае поверхность снизу — двумерная линейная интерполяция, которая реализуется по умолчанию, когда не указан параметр method.

7.13.6. Трехмерная табличная интерполяция — interp3

Для трехмерной табличной интерполяции используется функция:

Vl = interp3(X,Y,Z,V,Xl,Yl,Zl) — интерполирует, чтобы найти **Vl**, значение основной трехмерной функции **V** в точках матриц **Xl**,**Yl** и **Zl**. Матрицы **X**,**Y** и **Z** определяют точки, в которых задано значение **V**, **Xl**, **Yl**, и **Zl** могут быть матрицами, в этом случае **interp3** возвращает значения **Z**, соответствующие точкам **(Xl(i,j),Yl(i,j) ,Zl(i,j))**. В качестве альтернативы можно передать векторы **xi**, **yi** и **zi** .Векторы-аргументы, имеющие неодинаковый размер, представляются как если бы использовалась команда **meshgrid**.

Vl = interp3(V,Xl,Yl,Zl) — подразумевает **X**=1:N, **Y**=1:M, **Z**=1:P, где **[M,N,P]=size(V)**.

Vl = interp3(V,ntimes) — осуществляет интерполяцию рекурсивным методом с числом шагов ntimes.

Vl = interp3(...,method) — позволяет задать метод интерполяции: **'nearest'** ступенчатая интерполяция; **'linear'** — линейная интерполяция; **'cubic'** — кубическая интерполяция.

Все методы интерполяции требуют, чтобы **X**, **Y** и **Z** изменялись монотонно и имели такой же формат (**«plaid»**), как если бы они были получены с помощью функции **meshgrid**. Для более быстрой интерполяции, когда **X**, **Y** и **Z** — векторы равномерно распределенных в пространстве узловых точек, лучше использовать методы **'*linear'**, **'*cubic'** или **'*nearest'**.

7.13.7. N-мерная табличная интерполяция — interpn

MATLAB позволяет выполнить даже n-мерную табличную интерполяцию. Для этого используется функция:

Vl = interpn(X1,X2,X3,...,V,Y1,Y2,Y3,...) — интерполирует, чтобы найти **Vl**, значение основной многомерной функции **V** в точках массивов **Y1, Y2, Y3**,.... Для многомерной функции **V** должно быть указано **2*N+1** аргументов, где **N** — определяет размерность. Массивы **X1,X2, X3**,... определяют точки, в которых задано значение **V**. **Y1,Y2,Y3**,... могут быть матрицами, в этом случае **interpn** возвращает значения **Vl**, соответствующие точкам **(Y1(i,j),Y2(i,j),Y3(i,j),...)**. В качестве альтернативы можно передать векторы **y1, y2, y3**,... В этом случае **interpn** интерпретирует эти векторы, как если бы использовалась команда **ndgrid(y1,y2,y3,...)**.

Vl = interpn(V,Y1,Y2,Y3,...) — подразумевает **X1 =1:size(V,1), X2 = 1:size(V,2), X3 = 1:size(V,3)**.

Vl = interpn(V,ntimes) — осуществляет интерполяцию рекурсивным методом с числом шагов ntimes.

Vl = interpn(...,method) — позволяет указать метод интерполяции: **'nearest'** — ступенчатая интерполяция; **'linear'** — линейная интерполяция; **'cubic'** — кубическая интерполяция.

В связи с редким применением такого вида интерполяции, наглядная трактовка которой отсутствует, примеры ее применения не приводятся.

7.13.8. Интерполяция кубическим сплайном — spline

Сплайн-интерполяция используется для представления данных отрезками полиномов невысокой степени — чаще всего третьей. При этом кубическая интерполяция обеспечивает непрерывность в узловых точках трех первых производных — нулевой, первой и второй. Из этого вытекают следующие свойства кубической сплайн-интерполяции:

- график кусочно-полиномиальной аппроксимирующей функции проходит точно через узловые точки;
- в узловых точках нет разрывов функции и резких перегибов;
- благодаря низкой степени полиномов погрешность между узловыми точками обычно достаточно мала;
- связь между числом узловых точек и степенью полинома отсутствует;

- поскольку используется множество полиномов, то появляется возможность аппроксимации функций с множеством пиков и впадин.

Как отмечалось, в переводе splain означает "гибкая линейка". График интерполирующей функции при этом виде интерполяции можно уподобить кривой, по которой изгибается гибкая линейка, закрепленная в узловых точках. Реализуется сплайн-интерполяция следующей функцией:

yi = spline(x,y,xi) — использует векторы **x** и **y**, содержащие значения функции в точках **x** и вектор **xi**, задающий новые точки для нахождения элементов вектора **yi**, используя кубическую сплайн-интерполяцию.

pp = spline(x,y) — возвращает pp-форму сплайна, используемую в функции **ppval** и других сплайн- функциях.

Пример:

```
» x=0:10; y=3*cos(x);
» x1=0:0.1:11;
» y1=spline(x,y,x1);
» plot(x,y,'o',x1,y1,'--')
```

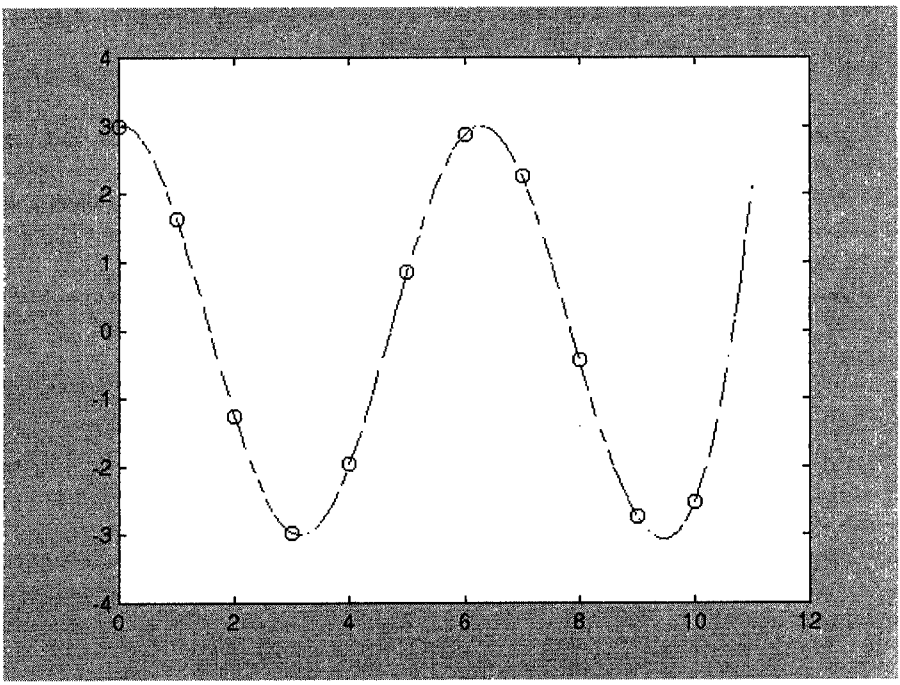


Рис. 7.23. Пример применения функции **spline**

Сплайн-интерполяция дает неплохие результаты для функций, не имеющих разрывов и резких перегибов. Особенно хорошие результаты дает сплайн-интерполяция для монотонных функций.

Ввиду важности сплайн-интерполяции и аппроксимации в обработке и представлении сложных, данных в Toolbox системы MATLAB входит пакет расширения Spline Toolbox, содержащий около 70 дополнительных функций, относящихся к реализации сплайн-интерполяции и аппроксимации, а также графического представления сплайнами их результатов. Для вызова данных об этом пакете используйте команду **help splines**.

7.14. Решение обыкновенных дифференциальных уравнений

Анализ поведения многих систем и устройств в динамике, а также решение многих задач в теории колебаний и в поведении упругих оболочек обычно базируется на решении систем дифференциальных уравнений. Их обычно представляют в виде системы из дифференциальных уравнений первого порядка в форме Коши:

$$\frac{dy}{dt} = y' = f(y,t).$$

Параметр t не обязательно означает время, хотя чаще всего решение дифференциальных уравнений ищется во временной области.

В этом разделе коротко описаны численные методы решения обыкновенных дифференциальных уравнений (ОДУ) и некоторые вспомогательные функции, полезные для решения систем ОДУ. Дается представление о пакете расширения, решающем дифференциальные уравнения с частными производными. Решатели ОДУ применяются в мощном пакете моделирования динамических систем, представленных функциональными блок-схемами Simulink. Ему посвящена глава 10 данной книги. Коротко описан другой пакет — для решения дифференциальных уравнений с частными производными.

7.14.1 Решатели ОДУ- ode45, ode23, ode113, ode15s, ode23

Для решения систем ОДУ в MATLAB реализованы различные методы. Их реализации названы **решателями** ОДУ.

Внимание! В этом разделе обобщенное название **solver (решатель)** означает один из возможных численных методов решения ОДУ: **ode45, ode23, ode113, ode15s, ode23s, ode23t, ode23tb**.

Решатели реализуют следующие методы решения систем дифференциальных уравнений:

ode45 — одношаговые явные методы Рунге-Кутта 4 и 5 порядка. Это классический метод решения, рекомендуемый для начальной пробы решения. Во многих случаях он дает хорошие результаты.

ode23 — одношаговые явные методы Рунге-Кутта 2 и 4 порядка. При умеренных требованиях к жесткости системы ОДУ и к точности решения этот метод может дать выигрыш в скорости решения.

ode113 — многошаговый метод Адамса-Башворта-Мултона переменного порядка. Это адаптивный метод, который может обеспечить высокую точность решения.

ode15s — многошаговый метод переменного порядка (от 1 до 5 по умолчанию), использующий формулы численного дифференцирования. Это адаптивный метод, его стоит применять, если решатель ode45 не обеспечивает решение.

ode23s — одношаговый метод, использующий модифицированную формулу Розенброка 2-го порядка. Может обеспечить высокую скорость вычислений при низкой точности.

ode23t — метод трапеций с интерполяцией. Этот метод дает хорошие результаты при решении задач, описывающих осцилляторы с почти гармоническим выходным сигналом.

ode23tb — неявный метод Рунге-Кутта в начале решения и метод, использующий формулы обратного дифференцирования 2-го порядка в последующем. При низкой точности этот метод может оказаться более эффективным, чем ode15s.

Все решатели могут решать системы уравнений явного вида **y'=F(t,y)**. Решатели **ode15s**, **ode23s**, **ode23t** и **ode23tb** могут решать уравнения неявного вида **My'=F(t,y)**. И, наконец, все решатели, за исключением **ode23s**, могут решить уравнения вида **M(t)y'=F(t,y)**.

Основные характеристики решателей ОДУ сведены в таблицу:

Решатели	Тип задачи	Степень точности	Когда использовать
ode45	не жесткая	средняя	в большинстве случаев
ode23	не жесткая	низкая	при допустимости грубой погрешности или при решении умеренно жестких задач
ode113	не жесткая	от низкой до высокой	при высокой точности решения или при решении сложных в вычислительном отношении задач
ode15s	жесткая	от низкой до средней	если ode45 вычисляет медленно или есть матрица масс
ode23s	жесткая	низкая	при допустимости грубой погрешности для решения жестких систем или когда есть матрица постоянных масс
ode23t	жесткая	средняя	при решении, близком к гармоническим колебаниям, и при умеренно жестких задачах
ode23tb	жесткая	низкая	при уравнениях, заданных в неявной форме Коши

В описанных далее функциях для решения систем дифференциальных уравнений приняты следующие обозначения и правила:

F — название ODE-файла, т.е. функции от **t** и **y**, которая возвращает вектор-столбец.

tspan — вектор, определяющий интервал интегрирования **[t0 tfinal]**. Для получения решений в конкретные моменты времени **t0, t1, ..., tfinal** (расположенные в порядке уменьшения или увеличения) нужно использовать **tspan = [t0 t1 ... tfinal]**.

y0 — вектор начальных условий.

options — аргумент, создаваемый функцией **odeset** (функция **odeget** позволяет вывести опции, установленные по умолчанию или с помощью функции **odeset**).

p1,p2... — произвольные параметры, передаваемые в **F**.

T,Y — матрица решений **Y**, где каждая строка соответствует времени, возвращенному в векторе-столбце **T**.

Перейдем к описанию функций для решения систем дифференциальных уравнений:

[T,Y] = solver('F',tspan,y0) с **tspan = [t0 tfinal]** — интегрирует систему дифференциальных уравнений вида **y' = F(t,y)** от времени **t0** до **tfinal** с начальными условиями **y0**. **'F'** — строка, содержащая имя ODE-файла. Функция **F(t,y)** должна возвращать вектор-столбец. Каждая строка в массиве решений **y** соответствует времени, возвращаемому в векторе-столбце **t**. Для получения решений в конкретных точках **t0, t1, ..., tfinal** (расположенных в порядке уменьшения или увеличения) нужно использовать **tspan = [t0 t1 ... tfinal]**.

[T,Y] = solver('F',tspan,y0,options) — дает решение, подобное описанному выше, но с параметрами, определяемыми значениями аргумента **options**, созданного функцией **odeset**. Обычно используемые параметры включают допустимое значение относительной погрешности **RelTol** (1e–3 по умолчанию) и вектор допустимых значений абсолютной погрешности **AbsTol** (все компоненты равны 1e–6 по умолчанию).

[T,Y] = solver('F',tspan,y0,options,p1,p2...) — дает решение, подобное описанному, помещая дополнительные параметры **p1,p2...** в М-файл **F** всякий раз, когда он вызывается. Используйте **options = []**, если никакие опции не установлены.

[T,Y,TE,YE,IE] = solver('F',tspan,y0,options) — в дополнение к описанному решению содержит свойства **Events**, установленные в структуре **options** в положение **'on'**. ODE-файл должен быть кодирован так, чтобы **F(t,y,'events')** возвращала соответствующую информацию (см. функцию **odefile** для подробного ознакомления).

Выходной аргумент **TE** — вектор-столбец времен, в которые происходят случаи (**events**), строки **YE** являются соответствующими решениями, и индексы в векторе **IE** определяют, какой случай (**event**) произошел.

Когда происходит вызов функции без выходных аргументов, по умолчанию вызывается выходная функция **odeplot** для построения вычисленного решения. В качестве альтернативы можно установить свойство **OutputFcn** в значение **'odeplot'**. Можно установить свойство **OutputFcn** в значение **'odephas2'** или **'odephas3'** для по-

строения двух- или трехмерных фазовых плоскостей (см. функцию **odefile** для подробного ознакомления).

[T,X,Y] = solver('model',tspan,y0,options,ut,p1,p2,...) — использует модель SIMULINK, вызывая соответствующий решатель из нее: **[T,X,Y] = sim(solver,'model',...)**.

Параметры интегрирования (**options**) могут быть определены и в ODE-файле, и в командной строке. Если опция определена в обоих местах, определение в командной строке имеет приоритет.

Решатели используют различные параметры в списке опций:

Параметры	ode45	ode23	ode113	ode15s	ode23s
RelTol, AbsTol	+	+	+	+	+
OutputFcn, OutputSel, Refine, Stats	+	+	+	+	+
Events	+	+	+	+	+
MaxStep, InitialStep	+	+	+	+	+
Jconstant, Jacobian, Jpattern, Vectorized	-	-	-	+	+
Mass	-	-	-	+	+
MassConstant	-	-	-	+	-
MaxOrder, BDF	-	-	-	+	-

Покажем применение решателя ОДУ на ставшем классическим примере — решении уравнения Ван-дер-Поля, записанного в виде системы из двух дифференциальных уравнений:

$$y'_1=y_2$$
$$y'_2=100*(1-y_1^2)*y_2-y_1$$
$$y_1(0)=0;\ y_2(0)=1$$

М-файл **vdp100.m**, содержащий систему ОДУ, имеет вид:

```
function dy = vdp100(t,y)
dy = zeros(2,1); % a column vector
dy(1) = y(2);
dy(2) = 2*(1 - y(1)^2)*y(2) - y(1);
```

Тогда решение решателем **ode15s** и сопровождающий его график (рис.7.24) можно получить, используя следующие команды:

```
» [T,Y] = ode15s('vdp100',[0 30],[2 0]);
» plot(T,Y)
» hold on;gtext('y1'),gtext('y2')
```

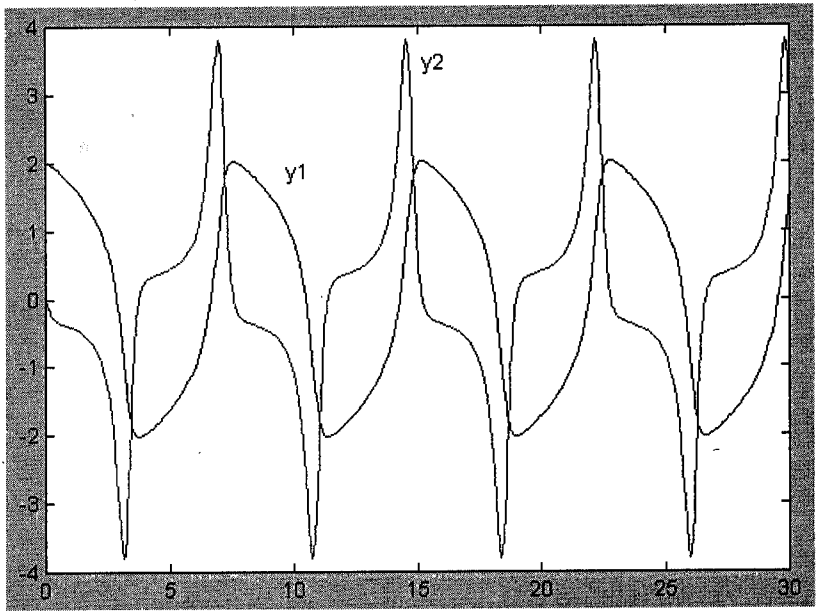


Рис. 7.24. Пример решения системы дифференциальных уравнений численным методом

Последние команды позволяют с помощью мышки нанести на графики решений y1=**y**(1) и y2=**y**(2) помечающие их надписи.

7.14.2. Описание системы ОДУ — odefile

Особый объект М-файл **odefile** представляет собой не просто функцию, а запись справки, которая описывает, как создать М-файл, определяющий правую часть системы уравнений, которая будет решена. Это определение — первый шаг в использовании любого решателя ОДУ в системе MATLAB. В описании MATLAB этот М-файл упоминается как **odefile**, хотя вместо него можно использовать любое имя. В объекте **odefile** помимо правой части системы ОДУ можно задать начальные условия и пределы интегрирования.

Можно использовать М-файл **odefile**, чтобы определить систему дифференциальных уравнений в одной из явных (первая формула) или неявных форм:

y'=F(t,y), **My'=F(t,y)** или **M(t)y'=F(t,y)**

где: **t** — независимая переменная (скаляр), которая обычно представляет время, **y** — вектор зависимых переменных, **F** — функция от **t** и **y** — возвращающая вектор-столбец такой же длины, как и **y**. **M** и **M(t)** — матрицы, которые не должны быть вырожденными. **M** может быть и константой.

Шаблон **odefile** для использования ODE-файла задается следующим образом:

1. Введите команду **help odefile** для отображения записи справки.
2. Вырежьте нужную часть файла и оформите ее в отдельный файл со своим именем
3. Отредактируйте файл, оставив в нем только нужные фрагменты
4. Вставьте соответствующие данные туда, где указано в файле.

Мы не приводим вид шаблона **odefile**, поскольку этот шаблон довольно большой и рассчитан на все "случаи жизни". Рассмотрим пример решения уравнения вида $y''_1=2*(1-y_1^2)*y_1-y'_1$. Оно сводится к системе уравнений:

$$y'_1=y_2$$
$$y'_2=2*(1-y_1^2)*y_1-y_2$$

Используя шаблон, подготовим M-файл vdp.m:

```
function [out1,out2,out3] = vdp(t,y,flag)
if nargin < 3 | isempty(flag)
out1 = [2.*y(2).*(1-y(2).^2)-y(1); y(1)];
else
switch(flag)
case 'init' % Return tspan, y0 and options
out1 = [0 20];
out2 = [2; 0];
out3 = [ ];
otherwise
error(['Unknown request ''' flag '''.']);
end
end
```

Тогда решение системы с помощью решателя **ode23** реализуется следующими командами:

```
» [T,Y] = ode23('vdp',[0 20],[2 0]);
» plot(T,Y(:,1),'-',T,Y(:,2),'-.')
```

График решения последнего примера показан на рис. 7.25.

Рекомендуется просмотреть дополнительные примеры решения систем дифференциальных уравнений, приведенные в файле odefemo и в [33]. Во многих случаях решение задач, которые сводятся к решению систем дифференциальных уравнений, удобнее осуществлять с помощью пакета расширения Simulink — он описан в главе 10.

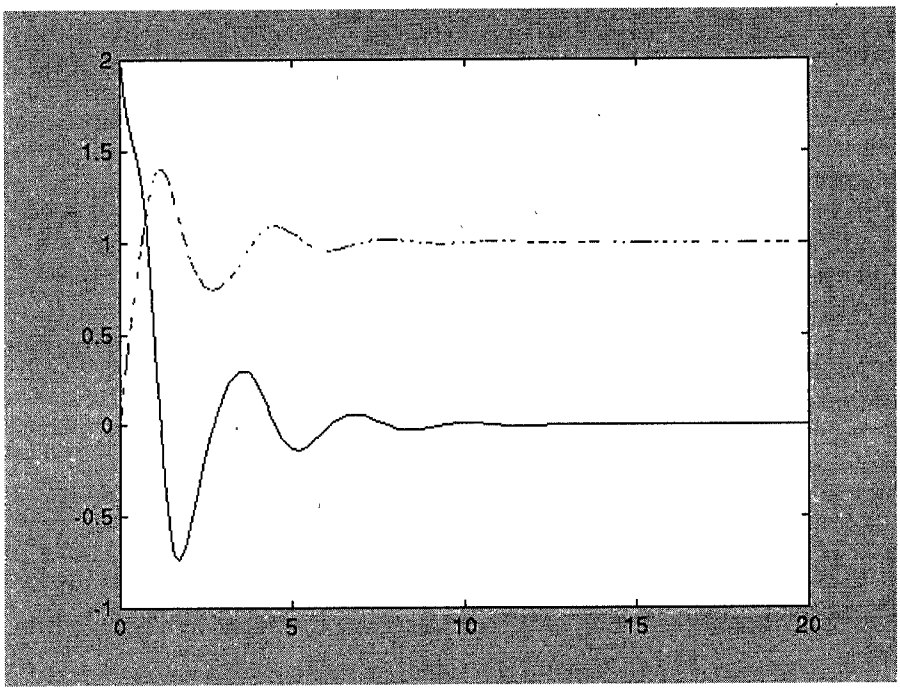


Рис. 7.25. Пример решения системы ОДУ

7.14.3 Дескрипторная поддержка опций решателя — odeget, odeset

При помощи перечисленных ниже функций поддержки опций решателя можно получить и создать или изменить опции решателя.

o=odeget(options,'name') — извлекает значение свойства, определенного строкой 'name' из структуры опций options, возвращает пустую матрицу, если значение свойства не определено в структуре options. Можно ввести только первые буквы, которые однозначно определяют имя свойства. Пустая матрица [ ] — допустимый аргумент options.

o=odeget(options,'name',default) — возвращает **o=default** , если имя свойства не определено в структуре options.

Пример:

```
» options = odeset('RelTol',[1e-6 1e-7],'AbsTol',6e-3);
» odeget(options,'Rel')
ans =
  1.0e-006 *
    1.0000   0.1000
» odeget(options,'Abs')
ans =
    0.0060
```

options=odeset('name1',value1,'name2',value2...) — создает структуру опций, в которой указанные свойства принимают следующие за ними значения. Если какое-то свойство не определено в функции **odeset**, то оно принимает значение пустой матрицы. Можно ввести только первые буквы, которые однозначно определяют имя свойства.

options=odeset(oldopts,'name1',value1,...) — изменяет в существующей структуре опций соответствующие значения.

options=odeset(oldopts,newopts) — изменяет существующую структуру опций **oldopts** путем объединения ее с новой структурой **newopts**. Все новые опции, не равные пустой матрице, заменяют соответствующие опции в структуре **oldopts**. Например:

```
oldopts
F 1 [] 4 's' 's' [] [] []
newopts
T 3 F [] '' [] [] [] []
odeset(oldopts,newopts)
T 3 F 4 '' 's' [] [] []
```

Функция **odeset** без параметров возвращает все имена свойств и их допустимые значения.

Пример:

```
» odeset
        AbsTol: [ positive scalar or vector {1e-6} ]
           BDF: [ on | {off} ]
        Events: [ on | {off} ]
    InitialStep: [ positive scalar ]
      Jacobian: [ on | {off} ]
    JConstant: [ on | {off} ]
      JPattern: [ on | {off} ]
          Mass: [ on | {off} ]
MassConstant: [ on | off ]
     MaxOrder: [ 1 | 2 | 3 | 4 | {5} ]
      MaxStep: [ positive scalar ]
  NormControl: [ on | {off} ]
    OutputFcn: [ string ]
     OutputSel: [ vector of integers ]
        Refine: [ positive integer ]
        RelTol: [ positive scalar {1e-3} ]
         Stats: [ on | {off} ]
    Vectorized: [ on | {off} ]
```

7.14.4. Информация о пакете Partial Differencial Equations Toollbox

Специалистов в области численных методов решения дифференциальных уравнений с частными производными несомненно заинтересует обширный пакет Partial Differencial Equations Toollbox (PDETB). Хотя этот пакет является самостоятельным приложением и в ядро MATLAB не входит, мы приведем краткое описание некоторых его возможностей с парой примеров.

Поскольку ряд применений пакета PDETB связан с проблемами анализа и оптимизации трехмерных поверхностей и оболочек, в пакет введены удобные функции для построения их графиков. Они могут использоваться совместно с функцией **pdeplot**, что иллюстрирует следующий пример:

```
[p,e,t]=initmesh('lshapeg');
u=assempde('lshapeb',p,e,t,1,0,1);
pdeplot(p,e,t,'xydata',u,'zdata',u,'mesh','off');
```

На рис. 7.26 представлено построение трехмерной поверхности с помощью функции **pdeplot**.

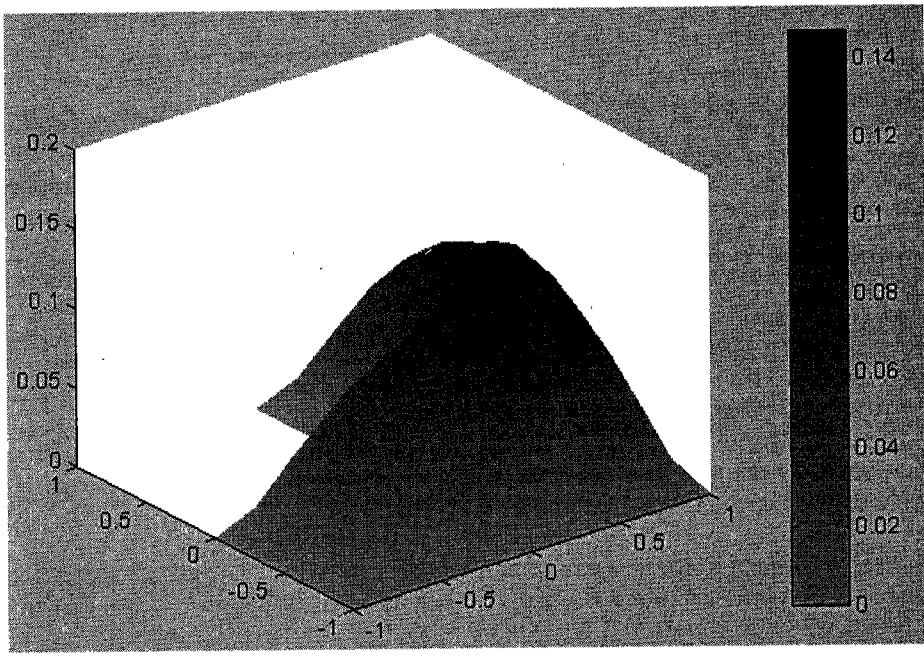


Рис. 7.26. Трехмерная поверхность, инициированная функцией **initmesh**

Как видно из рис. 7.26, средства графики пакета дают высококачественное изображение поверхностей (оболочек) с функциональной окраской (на черно-белых картинках, увы, не видимой во всей красе).

В состав пакета входит ряд полезных демонстрационных примеров с именами от **pdedemo1** до **pdedemo8**. Их можно запустить из командной строки (с указанием имени), а можно пустить и из режима демонстрации Demos. Рекомендуется, однако, сделать это из командной строки, так как примеры ориентированы на управление в командном режиме — в основном оно сводится к нажатию клавиши Enter для прерывания введенных для просмотра предварительных результатов вычислений пауз.

Рассмотрим пример **pdedemo3**, решающий проблему минимизации параболической поверхности решением дифференциального уравнения

```
-div( 1/sqrt(1+grad|u|^2) * grad(u) ) = 0
```

при граничном условии u=x^2. Ниже представлен текст файла pdedemo3.m с убранными англоязычными комментариями:

```
%PDEDEMO3 Решение проблемы минимизации параболической
%поверхности
echo on
clc
g='circleg'; % Единичная окружность
b='circleb2'; % x^2 - граничное условие
c='1./sqrt(1+ux.^2+uy.^2)';
a=0; f=0;
rtol=1e-3; % Погрешность вычислений решателем
pause % Пауза в вычислениях
clc
% Генерация поверхности
[p,e,t]=initmesh(g); [p,e,t]=refinemesh(g,p,e,t);
%  Решение нелинейной проблемы
u=pdenonlin(b,p,e,t,c,a,f,'tol',rtol);
pdesurf(p,t,u);
pause % Пауза в вычислениях
echo off
```

Запустив данный файл, можно наблюдать результаты вычислений и получить график параболической поверхности. Он представлен на рис.7.27.

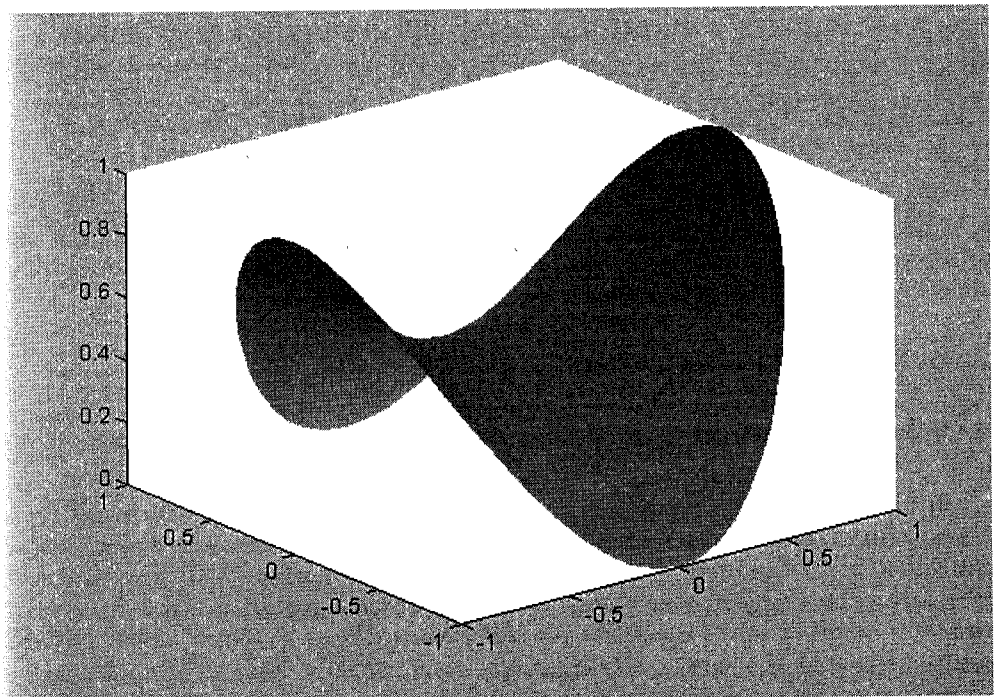


Рис. 7.27. Пример минимизации поверхности

Весьма интересны и поучительны примеры с анимацией: **pdedemo2** — появление и распространение волн, **pdedemo5** — вздутие поверхности (пузырек газа), **pdedemo6** — колебания плоскости (гиперболическая проблема) и др. К сожалению, они представлены слишком объемными файлами, что не позволяет воспроизвести их в книге. Однако их несомненно стоит просмотреть на вашем компьютере.

Глава 8. Строки, файлы и звуки

Эта глава завершает рассмотрение базовых средств языка программирования MATLAB. За пределами рассмотрения в предшествующих главах остались операции со строками и файлами, а также поддержка системой MATLAB звуковых возможностей ПК. Отсюда и вытекает содержимое этой небольшой главы.

8.1. Обработка строк

8.1.1. Основные функции

Функции **обработки строк** для математической системы могут показаться второстепенными. Однако это не так. Строковое представление данных лежит в основе **символьной математики**, арифметики произвольной точности и многочисленных программных конструкций, не говоря уже о том, что оно широко применяется в базах данных и массивах ячеек. В основе представления символов в строках лежит их кодирование с помощью сменных **таблиц кодов**. Такие таблицы ставят в однозначное соответствие каждому символу некоторый код со значением от 0 до 255.

Вектор, содержащий строку символов, в системе MATLAB задается следующим образом:

S = 'Any Characters' — вектор, компонентами которого являются числовые коды, соответствующие символам.

Первые 127 чисел — это коды ASCII, представляющие буквы латинского языка, цифры и спецзнаки. Они образуют основную таблицу кодов. Вторая таблица (коды от 128 до 255) является дополнительной и может использоваться для представления символов других языков, например, русского. Длина вектора **S** соответствует числу символов в строке, включая пробелы. Апостроф внутри строки символов должен вводиться как двойные кавычки.

К основным **строковым функциям** относятся следующие функции:

string(X) — преобразует массив **X** положительных целых чисел (числовых кодов) в массив символов системы MATLAB и возвращает его.

double(S) — преобразует символы строки **S** в числовые коды и возвращает вектор с ними.

isstr(S) — возвращает логическую единицу, если **S** является строковой переменной, и логический ноль в противном случае.

deblank(str) — возвращает строку, полученную из **str**, с удаленными с ее конца пробелами.

deblank(c) — для многомерного строкового массива **c** применяют функцию **deblank** к каждому элементу массива **c**.

Примеры:

```
» S = 'computer'
S =
computer
» X = double(S)
X =
  99  111  109  112  117  116  101  114
» isstr(S)
ans =
   1
» c{1,1}='My  ';
» c{1,2}='home   ';
» c{1,3}='computer ';
» c
c =
  'My  '  'home   '  'computer '
  » c = deblank(c)
c =
  'My'  'home'  'computer'
```

8.1.2. Операции над строками

К операциям над строками обычно относят поиск вхождений одних строк в другие, замену регистров символов, объединение строк и так далее. Следующие функции осуществляют операции над строками:

findstr(str1,str2) — обеспечивает поиск начальных индексов более короткой строки **str2** внутри более длинной **str1** и возвращает вектор этих индексов. Индексы указывают на положение первого символа строки **str2** в строке **str1**. Пример:

```
» str1='Example of the function is the findstr function';
» str2='the';
» k = findstr(str1,str2)
k =
  12  28
```

lower('str') — возвращает строку символов **str**, в которой символы верхнего регистра переводятся в нижний регистр, а все остальные символы остаются без изменений. Пример:

```
» str='Example of the function';
» t = lower(str)
t =
example of the function
```

upper('str') — возвращает строку символов **str**, в которой все символы нижнего регистра переводятся в верхний регистр, а все остальные символы остаются без изменений. Пример:

```
» str='danger!';
» t = upper(str)
t =
DANGER!
```

strcat(s1,s2,s3,...) — выполняет горизонтальное объединение соответствующих строк массивов символов **s1**, **s2**, **s3** и т. д. и возвращает объединенную строку. Все входные массивы должны иметь размер с одинаковым числом строк или должны быть представлены в виде одной строки символов. Если входной массив состоит только из символов, то выходной массив также будет являться массивом символов. Если любой из входных массивов представлен как многомерный массив из строк символов, то функция **strcat** возвращает многомерный массив строк символов, сформированный из объединенных соответствующих элементов массивов **s1**, **s2**, **s3** и т. д.

Примеры:

```
» s1{1,1}='Home';
» s1{1,2}='book';
» s1
s1 =
  'Home'   'book'
» s2{1,1}='home';
» s2{1,2}='reading';
» s2
s2 =
   'home'   'reading'
» t = strcat(s1,s2)
t =
   'Homehome'   'bookreading'
» s1=['wri']
s1 =
wri
» s2=['ter']
s2 =
ter
» t = strcat(s1,s2)
t =
writer
```

strvcat(t1,t2,t3,...) — выполняет вертикальное объединение строк **t1**, **t2**, **t3**,... в массив символов **S**. Пример:

```
» t1=['string'];
» t2=['concatenation'];
» S = strvcat(t1,t2)
```

```
S =
string
concatenation
```

strcmp('str1','str2') — возвращает логическую единицу, если две сравниваемые строки **str1** и **str2** идентичны, и логический ноль в противном случае.

TF = strcmp(S,T) — возвращает массив **TF**, содержащий единицы для идентичных элементов массивов **S** и **T** и нули для всех остальных.

Примеры:

```
» str1='computer';
» str2='computer';
» k = strcmp(str1,str2)
k =
   1
» S{1,1}='first';
» S{1,2}='second';
» S
S =
  'first'   'second'
» T{1,1}='third';
» TF = strcmp(S,T)
TF =
   0   0
» T{1,1}='second';
» TF = strcmp(S,T)
TF =
   0   1
```

strncmp('str1','str2',n) — возвращает логическую единицу, если две сравниваемые строки **str1** и **str2** содержат **n** первых идентичных символов, и логический ноль в противном случае. Аргументы **str1** и **str2** могут быть также многомерными массивами строк символов.

TF = strncmp(S,T,N) — возвращает массив **TF**, содержащий единицы для идентичных (до **n** символов) элементов массивов **S** и **T** и нули для всех остальных.

Примеры:

```
» str1='computer'
str1 =
computer
» str1='computer for me'
str1 =
computer for me
» k = strncmp(str1,str2,3)
k =
   1
```

```
» k = strncmp(str1,str2,12)
k =
   0
```

strjust(S) — возвращает выровненный массив символов.

strmatch('str',STRS) — просматривает массив символов или многомерный массив строк символов **STRS** по строкам, находит строки символов, начинающиеся с строки **str**, и возвращает соответствующие индексы строк.

strmatch('str',STRS,'exact') — возвращает только индексы строк символов массива **STRS**, точно совпадающие со строкой символов **str**.

Пример:

```
» STRS{1,1}='character';
» STRS{1,2}='array';
» STRS{2,1}='character array';
» STRS{2,2}='string';
» STRS
STRS =
   'character'         'array'
   'character array'   'string'
» i = strmatch('charac',STRS)
i =
   1
   2
» i = strmatch('character',STRS,'exact')
i =
   1
```

strrep(str1,str2,str3) — заменяет все строки символов **str2** внутри строки символов **str1** на строку **str3**.

strrep(str1,str2,str3) — возвращает многомерный массив строк символов, полученный в результате выполнения функции **strrep** над соответствующими входными массивами. **str1, str2** или **str3** — многомерные массивы строк символов.

Пример:

```
» str1='This is a good example for me.';
» str2='good';
» str3='best';
» str = strrep(str1,str2,str3)
str =
This is a best example for me.
```

strtok('str',delimiter) — возвращает часть текстовой строки **str**, ограниченную разделителем **delimiter**. Вектор **delimiter** содержит действительные символы-разделители.

strtok('str') — использует символ-разделитель по умолчанию. Символами-разделителями считаются символ табуляции (ASCII код 9), символ возврата каретки (ASCII код 13) и пробел (ASCII код 32).

[token,rem]=strtok(...) — возвращает остаток **rem** исходной строки.

Примеры:

```
» str='This is a good example for me.';
» token = strtok(str)
token =
This
» token = strtok(str,'f')
token =
This is a good example
» str='This is a good example for me.';
» [token,rem] = strtok(str)
token =
This

rem =
 is a good example for me.
```

8.1.3. Преобразование символов и строк

Рассмотрим функции системы MATLAB, обеспечивающие преобразования строк.

char(X) — возвращает массив символов системы MATLAB, преобразуя входной массив **X**, состоящий из положительных целых чисел (кодов ASCII).

char(C) — помещает каждый элемент многомерного массива строк символов **C** в строки массива символов **S**.

char(t1,t2,t3,..) — возвращает массив символов **S**, в котором строки символов **T1,T2,T3**,... размещаются по строкам.

Примеры:

```
» X=reshape(32:127,32,3);
» S = char(X')
S =
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~□
» t1='computer'
» t2='for';
» t3='home';
» t4='users';
» S = char(t1,t2,t3,t4)
S =
computer
for
```

```
home
users
```

int2str(N) — возвращает строку, содержащую символьное представление целого числа N. Аргумент **N** может быть целым числом, вектором или матрицей целых чисел. Пример:

```
» N=magic(3)
N =
   8   1   6
   3   5   7
   4   9   2
» str=int2str(N)
str =
8 1 6
3 5 7
4 9 2
```

mat2str(A) — возвращает строку, преобразуя матрицу **A** в строку.

mat2str(A,n) — преобразует матрицу **A** в строку, используя точность до **n** цифр после десятичной точки.

Пример:

```
» rand('state');
» A=rand(4,3)
A =
   0.9501   0.8913   0.8214
   0.2311   0.7621   0.4447
   0.6068   0.4565   0.6154
   0.4860   0.0185   0.7919
» str = mat2str(A,2)
str =
[0.95 0.89 0.82;0.23 0.76 0.44;0.61 0.46 0.62;0.49 0.019 0.79]
```

num2str(A) — выполняет преобразование массива чисел **A** в строку символов **str** с точностью округления до четырех разрядов и экспоненциальным представлением (если требуется).

num2str(A,precision) — выполняет преобразование массива **A** в строку символов **str** с максимальной точностью, определенной аргументом **precision**. Аргумент **precision** определяет число разрядов в выходной строке.

num2str(A,format) — выполняет преобразование массива чисел **A**, используя заданный формат **format**. По умолчанию принимается формат, имеющий четыре разряда после разделительной точки для чисел с фиксированной или плавающей точкой.

Пример:

```
» str = num2str(pi,7)
str =
3.141593
» rand('state');
```

```
» A=rand(3,5)
A =
   0.9501   0.4860   0.4565   0.4447   0.9218
   0.2311   0.8913   0.0185   0.6154   0.7382
   0.6068   0.7621   0.8214   0.7919   0.1763
» str = num2str(A,1)
str =
 1   0.5   0.5   0.4   0.9
0.2  0.9   0.02  0.6   0.7
0.6  0.8   0.8   0.8   0.2
```

str2num('str') — выполняет преобразование строки **str**, которая представлена в ASCII-символах, в арифметическое выражение системы MATLAB с его вычислением. Пример:

```
» x = str2num('5.45+2.67')
x =
   8.1200
```

Обратите особое внимание на последнюю функцию, поскольку она обеспечивает переход от символьного представления математических выражений к их вычисленным численным значениям.

8.2. Функции преобразования систем исчисления

Некоторые строковые функции служат для преобразования систем исчисления. Ниже представлен набор этих функций.

bin2dec('binarystr') — возвращает десятичное число, эквивалентное двоичной строке символов **binarystr**. Пример:

```
» bin2dec('101')
ans =
   5
```

dec2bin(d) — возвращает двоичную строку символов, эквивалентную десятичному числу d. Аргумент d должен быть неотрицательным целым числом, меньшим чем 2^{52}.

dec2bin(d,n) — возвращает двоичную строку, содержащую по меньшей мере n бит.

Пример:

```
» str = dec2bin(12)
str =
1100
```

dec2hex(d) — возвращает шестнадцатеричную строку символов, эквивалентную десятичному числу d. Аргумент d должен быть неотрицательным целым числом, меньшим чем 2^{52}.

str = dec2hex(d,n) — возвращает двоичную строку, содержащую по меньшей мере n бит.

Пример:

```
» str = dec2hex(1234)
str =
4D2
```

hex2dec('hex_value') — возвращает десятичное число d, эквивалентное шестнадцатеричному числу, находящемуся в строке символов **hex_value**. Если аргумент **hex_value** является массивом символов, то каждая строка этого массива представляется как шестнадцатеричная строка символов. Пример:

```
» d = hex2dec('4D2')
d =
     1234
```

hex2num('hex_value') — возвращает десятичное число f с удвоенной точностью, эквивалентное шестнадцатеричному числу, находящемуся в строке символов **hex_value**. Пример:

```
» f = hex2num('4831fb52a18')
f =
  6.1189e+039
```

8.3. Операции ввода- вывода файлов

Файлы — это довольно распространенный объект системы MATLAB. О некоторых типах файлов уже говорилось в предшествующих главах. Однако файлы имеют некоторые общие свойства, например, имя, заданное строкой, и расширение, также содержащее строку. В этой главе рассматриваются некоторые свойства файлов, которые не зависят от их типа и относятся к любым файлам.

8.3.1. Открытие и закрытие файлов

Файл обычно является некоторой совокупностью данных, объединенных одним именем. Тип файла обычно определяется его расширением. Мы рассматриваем файл как некое целое, хотя физически на диске он может быть представлен несколькими областями — говорят, что в этом случае файл фрагментирован.

Перед использованием любого файла он должен быть открыт, а по окончании использования — закрыт. Одновременно открытыми и доступными для чтения могут быть много файлов. Рассмотрим команды открытия и закрытия файлов. Варианты команды открытия файлов **fopen** представлены ниже.

fopen(filename, permission) — открывает файл с именем **filename** и параметром, определенным в **permission**, и возвращает идентификатор **fid** со значением: 0 — чтение с клавиатуры (**permission** установлено в 'r'); 1 — вывод на дисплей (**permission** установлено в 'a'); 2 — вывод сообщения об ошибке (**permission** установлен в 'a'); -1 — неудача в открытии файла с выводом сообщения **message,** о типе ошибки. Идентификатор **fid** часто используется в качестве аргумента другими функциями и программами ввода-вывода. Имя файла **filename** может содержать путь к файлу.

Если открываемый для чтения файл не найден в текущем каталоге, то функция **fopen** осуществляет поиск файла по пути, указанном в MATLAB.

Опция **permission** может принимать одно из следующих значений:

'r' — открытие файла для чтения (по умолчанию).
'r+' — открытие файла для чтения и записи.
'w' — удаление содержания существующего файла или создание нового и открытие его для записи.
'w+' — удаление содержания существующего файла или создание нового и открытие его для записи и чтения.
-+\'W' — запись без автоматического заполнения, используется с ЗУ на магнитной ленте.
'a' — создание и открытие нового файла или открытие существующего для записи, добавления в конец файла.
'a+' — создание и открытие нового файла или открытие существующего для записи, чтения, добавления в конец файла.
'A' — добавление без автоматического заполнения, используется с ЗУ на магнитной ленте.

Добавление **'t'** к этой строке, например, **'rt'**, в системах, которые имеют различие между текстовыми и двоичными файлами, предписывает системе открыть файл в текстовом режиме. Например, в MATLAB под MS-DOS и VMS нельзя открыть текстовый файл без параметра **'rt'**. Точно так же использование **'b'** предписывает системе открыть файл в двоичном режиме (по умолчанию).

[fid,message] = fopen(filename,permission,format) — открывает файл как описано выше, возвращая идентификатор файла и сообщение. Кроме того, значение **format** позволяет точно определить числовой формат. Определенные вызовы функций **fread** или **fwrite** могут отменить числовой формат, заданный при вызове функции **fopen**. Допустимы следующие значения:
'native' или **'n'** .

fids = fopen('all') — возвращает вектор-строку, содержащую идентификаторы всех открытых файлов, не включая 0,1, и 2. Число элементов вектора равно числу открытых файлов.

[filename,permission,format] = fopen(fid) — возвращает полное имя файла, строку **permission** и строку **format**. При использовании недопустимых значений fid возвращаются пустые строки для всех выходных аргументов.

Команда **fclose** закрывает файл. Она имеет следующие варианты.

status = fclose(fid) — закрывает файл, если он открыт. Возвращает статус файла status 0, если закрытие завершилось успешно, и 1 — в противном случае. Аргумент fid — это идентификатор, связанный с открытым файлом (см. функцию fopen для более подробного описания).

status = fclose('all') — закрывает все открытые файлы. Возвращает 0 в случае успешного завершения и 1 — в противном случае.

Пример открытия и закрытия файла:

```
» fid=fopen('c:\ex','a+')
fid =
   4
» fclose(4)
ans =
   0
```

8.3.2. Операции с двоичными файлами

Двоичными, или **бинарными**, называют файлы, данные которых представляют собой машинные коды. Основные операции с такими кодами перечислены ниже.

[A,count] = fread(fid,size,precision) — считывает двоичные данные из определенного файла и помещает их в матрицу **A**. Выходной аргумент **count** содержит число удачно считанных элементов. Значение идентификатора **fid** — это целое число, возвращенное функцией **fopen**.

size — произвольный аргумент, определяющий количество считываемых данных. Если аргумент **size** не определен, функция **fread** считывает данные до конца файла.

Используются следующие опции:

n — чтение n элементов в вектор-столбец.

inf — чтение элементов до конца файла и помещение их в вектор-столбец, содержащий такое же количество элементов, как и в файле.

[m,n] — считывает столько элементов, сколько нужно для заполнения m×n матрицы.

Заполнение происходит по столбцам. Если элементов в файле мало, то матрица дополняется нулями. Если считывание по функции **fread** достигает конца файла, не заполнив матрицу необходимой размерности, то матрица дополняется нулями. Если

происходит ошибка, чтение останавливается на последнем считанном значении. **precision** — строка, определяющая числовую точность считывания значений, она контролирует число считанных бит для каждого значения и интерпретирует эти биты как целое число, число с плавающей запятой или как символ.

[A,count] = fread(fid,size,precision,skip) — включает произвольный аргумент **skip**, который определяет число байтов, необходимое пропустить после каждого считывания. Это может быть полезно при извлечении данных в несмежных областях из записей фиксированной длины. Если **precision** имеет битовый формат, подобно 'bitN' или 'ubitN', опция **skip** определяется в битах.

count=fwrite(fid,A,precision) — записывает элементы матрицы **A** в файл, представляя их с заданной точностью. Данные записываются в файл по столбцам, выходной аргумент **count** содержит число удачно записанных элементов. Значение идентификатора **fid** — это целое число, полученное при использовании функции **fopen**.

count=fwrite(fid,A,precision,skip) — делает то же, но включает произвольный аргумент **skip**, который определяет число байтов, необходимое пропустить перед каждой записью. Это полезно при вставке данных в несмежные области в записях фиксированной длины. Если **precision** имеет битовый формат, подобно 'bitN' или 'ubitN', опция **skip** определяется в битах.

Примеры:

```
» fid = fopen('c:\prim','a+')
fid =
   3
» A=magic(7)
A =
  30  39  48   1  10  19  28
  38  47   7   9  18  27  29
  46   6   8  17  26  35  37
   5  14  16  25  34  36  45
  13  15  24  33  42  44   4
  21  23  32  41  43   3  12
  22  31  40  49   2  11  20
» count = fwrite(3,A)
count =
  49
» status = fclose(3)
status =
   0
» fid = fopen('c:\prim','r')
fid =
   3
» [B,count] = fread(3,[7,7])
B =
```

```
  30   39   48   1   10   19   28
  38   47   7    9   18   27   29
  46   6    8   17   26   35   37
   5   14   16   25   34   36   45
  13   15   24   33   42   44   4
  21   23   32   41   43   3    12
  22   31   40   49   2    11   20
count =
  49
```

8.3.3. Операции над форматированными файлами

Файлы, содержащие форматированные данные, называют **форматированными файлами**. Ниже представлены функции, которые служат для работы с такими файлами.

line = fgetl(fid) — возвращает строку из файла с идентификатором **fid** с удалением символа конца строки. Если функция **fgetl** обнаруживает конец файла, то она возвращает значение 1 (см. функцию **fopen** с более подробным описанием **fid**).

line = fgets(fid) — возвращает строку из файла с идентификатором **fid** с символом конца строки. Если функция **fgets** обнаруживает конец файла, то она возвращает значение 1.

line = fgets(fid,nchar) — возвращает большинство nchar символов строки. Никакие дополнительные символы не считываются после признака конца строки или конца файла. Смотрите примеры к функции **fscanf**

count = fprintf(fid,format,A,...) — форматирует данные в действительной части матрицы **A** под контролем строки **format** и записывает их в файл с идентификатором **fid**. Функция **fprintf** возвращает число записанных байтов. Значение идентификатора **fid** — целое число, возвращаемое функцией **fopen**.

Если опустить идентификатор fid из списка аргументов функции **fprintf**, то вывод будет осуществляться на экран так же, как при использовании стандартного вывода (fid = 1).

fprintf(format,A,...) — запись осуществляется на стандартное устройство — экран (но не в файл). Строка **format** определяет систему обозначений, выравнивание, значащие цифры, ширину поля и другие атрибуты выходного формата. Она может содержать обычные буквы алфавита наряду со спецификаторами, знаками выравнивания и т. д.

Функция **fprintf** ведет себя как и аналогичная функция **fprintf**() языка ANSI C с некоторыми исключениями и расширениями. Следующие таблицы описывают неалфавитные символы, встречающиеся в строке **format**.

Символы ESC

Символ	Описание
\n	Новая строка
\t	Горизонтальная табуляция
\b	Возврат на один символ
\r	Перевод каретки
\f	Новая страница
****	Обратный слеш
\" или **"**	Единственная кавычка
%%	Процент

Спецификаторы

Спецификатор	Описание
%c	Единственный знак
%d	Десятичная система обозначений
%e	Экспоненциальное представление чисел (использование нижнего регистра e как в 3.1415e + 00)
%E	Экспоненциальное представление чисел (использование верхнего регистра E как в 3.1415E + 00)
%f	Система обозначений с фиксированной запятой
%g	Более компактная система обозначений, чем из %e или %f. Незначащие нули не выводятся
%G	То же самое, что и %g, но используется верхний регистр E
%o	Восьмеричная система обозначений (без знака)
%s	Строка символов
%u	Десятичная система обозначений (без знака)
%x	Шестнадцатеричная система обозначений (использование символов нижнего регистра a-f)
%X	Шестнадцатеричная система обозначений (использование верхнего регистра символов A-F)

Другие знаки могут быть вставлены в спецификатор между знаком **%** и другим спецификатором.

Другие знаки

Символ	Описание	Пример
Знак минус (-)	Выравнивание преобразованных аргументов по левому краю	%–5.2d
Знак плюс (+)	Всегда печатать знак числа (+ или -)	%+5.2d
Ноль (0)	Заполнение нулями вместо пробелов	%05.2d
Количество символов (ширина поля)	Определяет минимальное число знаков, которые будут напечатаны	%6f
Разряды (точность)	Ряд цифр, включая точку ., определяет количество печатаемых символов справа от десятичной точки	%6.2f

Смотрите примеры к функции **fscanf**

A = fscanf(fid,format) — читает все данные из файла с идентификатором **fid**, преобразовывает их согласно значению **format** и возвращает в матрицу **A**. Значение идентификатора **fid** — целое число, возвращаемое функцией **fopen**. **format** представляет собой ряд, определяющий формат данных, которые необходимо прочитать.

[A,count] = fscanf(fid,format,size) — считывает количество данных, определенное параметром size, преобразовывает их в соответствии с параметром **format** и возвращает их вместе с количеством успешно считанных элементов **count**. **size** — это произвольный аргумент, определяющий количество считываемых данных. Допустимы следующие значения:

n — чтение n элементов в вектор-столбец.

inf — чтение элементов до конца файла и помещение их в вектор-столбец, содержащий такое же количество элементов, как и в файле.

[m,n] — считывает столько элементов, сколько требуется для заполнения mxn матрицы. Заполнение происходит по столбцам. **n** может принимать значение **Inf**, но не **m**.

Строка **format** состоит из обычных символов и (или) спецификаторов. Спецификаторы указывают на тип данных и включают в себя символ **%**, опцию ширины поля и символы формата.

Допустимые символы формата:

Символ	Описание
%c	Определяет ширину поля
%d	Десятичное число
%e, %f, %g	Число с плавающей запятой
%i	Целое число со знаком
%o	Восьмеричное число со знаком
%s	Последовательность символов за исключением служебных
%u	Десятичное целое число со знаком
%x	Шестнадцатеричное целое число со знаком
[...]	Последовательность символов

Между символом **%** и символом формата допустимо вставлять следующие символы:

Звездочка (*) означает пропуск над соответствующим значением, если значение не сохранено в выходной матрице.

Строка цифр — максимальная ширина поля.

Буква — размер полученного объекта: например, **h** как в **%hd** для короткого целого числа, или **l** как в **%ld** для длинного целого числа, или в **%lg** для числа с двойной точностью с плавающей запятой.

Примеры:

```
» x = 0:pi/10:pi;y=[x;sin(x)];
» fid = fopen('c:\sin.txt','w');
» fprintf(fid,'%5.3f %10.6f\n',y);fclose(fid);
0.000  0.000000
0.314  0.309017
0.628  0.587785
0.942  0.809017
1.257  0.951057
1.571  1.000000
1.885  0.951057
2.199  0.809017
2.513  0.587785
2.827  0.309017
3.142  0.000000
» fid = fopen('c:\sin.txt','r');
» q=fscanf(fid,'%g',[2,10]);
» q'
ans =
       0       0
  0.3140  0.3090
  0.6280  0.5878
  0.9420  0.8090
  1.2570  0.9511
  1.5710  1.0000
  1.8850  0.9511
  2.1990  0.8090
  2.5130  0.5878
  2.8270  0.3090
» fgetl(fid)
ans =
3.142  0.000000
» fgets(fid)
ans =
    -1
» fclose(fid)
ans =
     0
```

8.3.4. Позиционирование файла

При считывании и записи файлов они условно представляются в виде линейно расположенных данных, наподобие записи на непрерывной магнитной ленте. Место, с которого идет считывание в данный момент (или позиция, начиная с которой идет запись), определяется специальным **указателем**. Файлы последовательного доступа просматриваются указателем строго от начала до конца, а в файлах произвольного доступа указатель может быть размещен в любом месте, начиная с которого ведется запись или считывание данных файла.

Таким образом указатель обеспечивает позиционирование файлов. Имеется ряд функций позиционирования:

eofstat = feof(fid) — проверяет, установлен ли указатель конца файла с идентификатором **fid** на конец файла. Возвращает **1**, если указатель конца файла установлен, и **0** — в противном случае.

message = ferror(fid) — возвращает сведения об ошибке **message**. Аргумент **fid** — идентификатор открытого файла (смотрите функцию **fopen** с ее подробным описанием).

message = ferror(fid,'clear') — очищает индикатор ошибки для определенного файла.

[message,errnum] = ferror(...) — возвращает номер статуса ошибки **errnum** последней операции ввода-вывода для определенного файла.

Если последняя операция ввода-вывода, выполненная для определенного значением **fid** файла, была успешной, значение **message** — это пустая строка, и errnum принимает значение **0**.

Значение errnum, отличное от нуля, говорит о том, что при последней операции ввода-вывода произошла ошибка. Параметр **message** содержит строку, содержащую информацию о характере возникшей ошибки.

Пример:

```
» fid=fopen('c:\example1','a+')
fid =
    3
» t = fread(3,[4,5])
t =
  Empty matrix: 4-by-0
» ferror(3)
ans =
Is the file open for reading? . . .
```

frewind(fid) — устанавливает указатель позиции в начало файла с идентификатором **fid**.

status = fseek(fid,offset,origin) — устанавливает указатель в файле с идентификатором **fid** в заданную позицию на байт, указанный параметром **offset** относительно **origin**.

Аргументы:

fid — идентификатор файла, возвращенный функцией **fopen**.

offset — значение, которое интерпретируется следующим образом:

offset > 0 — изменяет позицию указателя на offset байт в направлении к концу файла.

offset = 0 — не меняет позицию указателя.

offset < 0 — изменяет позицию указателя на offset байт в направлении к началу файла.

origin — аргумент, принимающий следующие значения:

'bof' –1: начало файла.

'cof' 0: текущая позиция указателя в файле.

'eof' 1: конец файла.

status — выходной аргумент. Принимает значение 0, если операция **fseek** произошла успешно, и 1 — в противном случае. Если произошла ошибка, используйте функцию **ferror** для получения более подробной информации.

position=ftell(fid) — возвращает позицию указателя для файла с идентификатором **fid**, полученного с помощью функции **fopen**. Выходной аргумент **position** — неотрицательное целое число, определяющее позицию указателя в байтах относительно начала файла. Если запрос был неудачным, position принимает значение **-1**. Используйте функцию **ferror** для отображения характера ошибки. Примеры:

```
» fid=fopen('c:\example','a+')
fid =
   3
» count = fwrite(3,magic(6))
count =
  36
» ftell(3)
ans =
  36
» frewind(3);ftell(3)
ans =
   0
» fseek(3,12,0);ftell(3)
ans =
  12
» feof(3)
ans =
   0
» fclose(3)
ans =
   0
```

s=sprintf(format,A,...) — форматирует данные в матрице **A** в формате, заданном параметром **format**, и создает из них строковую переменную **s**.

[s,errmsg] = sprintf(format,A,...) — аналогична ранее описанной функции, но дополнительно возвращает строку ошибки **errmsg**, если ошибка имела место, или пустую строку в противном случае. Строка **format** определяет систему обозначений, выравнивание, значащие цифры, ширину поля и другие атрибуты выходного формата. Она может содержать обычные буквы алфавита наряду со спецификаторами, знаками выравнивания и т. д. Функция **fprintf** ведет себя как и аналогичная функция **fprintf()** языка ANSI C с некоторыми исключениями и расширениями. Примеры:

```
» sprintf('%0.5g',(1+sqrt(7))/4)
ans =
0.91144
» sprintf('%s','привет')
ans =
привет
```

A = sscanf(s,format) — считывает данные из строковой переменной **s**, преобразовывает их согласно значению **format** и создает на основе этих данных матрицу **A**. Параметр **format** определяет формат данных, которые нужно считать.

sscanf — аналогична функции **fscanf** за исключением того, что она считывает данные из строковой переменной системы MATLAB, а не из файла.

A = sscanf(s,format,size) — считывает количество данных, определенное параметром size, и преобразовывает их согласно строке **format** . Параметр **size** представляет собой аргумент, определяющий количество данных для чтения. Допустимы следующие значения:

n — чтение n элементов в вектор-столбец.

inf — чтение элементов до конца строковой переменной и помещение их в вектор-столбец, содержащий такое же количество элементов, как и в строковой переменной.

[m,n] — считывает столько элементов, сколько требуется для заполнения m×n матрицы. Заполнение происходит по столбцам. **n** может принимать значение **Inf**, но не **m**.

[A,count,errmsg,nextindex] = sscanf(...) — считывает данные из строковой переменной **s**, преобразовывает их согласно значению **format** и возвращает в матрицу **A**. Параметр count — выходной аргумент, который возвращает число успешно считанных элементов; **errmsg** — выходной аргумент, который возвращает строку ошибки, если ошибка произошла, и пустую строку в противном случае; **nextindex** — выходной аргумент, который содержит число, на единицу большее, чем количество символов в **s**.

Строка **format** состоит из обычных символов и спецификаторов. Спецификаторы указывают на тип данных и включают в себя символ %, опцию ширины поля и символы формата. Пояснения можно найти в описании функции **fscanf**.

Пример:

```
» s = '4.83 3.16 22 45';
» [A,n,err,next] = sscanf(s,'%f')
A =
  4.8300
```

```
    3.1600
   22.0000
   45.0000
n =
    4
err =
    ''
next =
   16
```

8.4. Специализированные файлы

Приведенные ниже функции относятся к некоторым **специализированным файлам**:

M = dlmread(filename,delimiter) — считывает данные из файла **filename** с ASCII-разделителем, используя разделитель delimiter, в массив **M**. Используйте '\t', чтобы определить символ табуляции.

M = dlmread(filename,delimiter,r,c) — считывает данные из файла **filename** с ASCII-разделителем, используя разделитель **delimiter**, в массив **M**, начиная со смещения r и c. Параметры r и c такие, что r=0, c=0 определяют первое значение в файле.

M = dlmread(filename,delimiter,r,c,range) — импортирует индексированный или поименованный диапазон разделенных данных в формате ASCII. Для использования диапазона ячеек нужно определить параметр **range** в виде:

range = [ВерхняяЛеваяСтрока, ВерхнийЛевыйСтолбец, НижняяПраваяСтрока, НижнийПравыйСтолбец]

Аргументы: **delimiter** — символ, отделяющий отдельные матричные элементы в электронной таблице формата ASCII; символ запятой (,) — разделитель по умолчанию; **r,c** — ячейка электронной таблицы, из которой берутся матричные элементы, соответствующие элементам в верхнем левом углу таблицы; **range** — вектор, определяющий диапазон ячеек электронной таблицы.

Команда **dlmwrite** преобразует матрицу MATLAB в файл с ASCII-разделителем, читаемый программами электронной таблицы:

dlmwrite(filename,A,delimiter) — записывает матрицу **A** в верхнюю левую ячейку электронной таблицы **filename** и использует разделитель для отделения элементов матрицы. Используйте '\t' для создания файла с элементами, разделенными табуляцией. Все элементы со значением 0 опускаются. Например, массив [1 0 2] появится в файле в виде '1,,2', если разделителем является запятая.

dlmwrite(filename,A,delimiter,r,c) — записывает матрицу **A** в файл filename, начиная с ячейки, определенной **r** и **c**, с разделителем **delimiter**.

info=imfinfo(filename,fmt) — возвращает структуру, области в которой содержат информацию об изображении в графическом файле. Аргумент **filename** — стро-

ка, определяющая имя графического файла, и **fmt** — строка, которая определяет формат файла. Файл должен находиться в текущей директории или в директории, указанной в пути MATLAB. Если **imfinfo** не может найти файл с именем **filename**, она ищет файл с именем **filename.fmt**.

Таблица показывает возможные значения для аргумента fmt:

Формат	Тип файла
'bmp'	Windows Bitmap (BMP)
'hdf'	Hierarchical Data Format (HDF)
'jpg' или **'jpeg'**	Joint Photographic Experts Group (JPEG)
'pcx'	Windows Paintbrush (PCX)
'tif' или **'tiff'**	Tagged Image File Format (TIFF)
'xwd'	X Windows Dump (XWD)

Если **filename** — TIFF или HDF-файл, содержащий более, чем одно изображение, то **info** представляет собой массив структуры с одним элементом (то есть индивидуальную структуру) для каждого изображения в файле. Например, **info(3)** содержал бы информацию относительно третьего изображения в файле. Множество областей в **info** зависит от конкретного файла и его формата. Однако первые девять областей всегда одинаковы. В приведенной ниже таблице перечислены эти области и описаны их значения.

Области и их значения

Область	Значение
Filename	Строка, содержащая название файла; если файл находится не в текущей директории, строка содержит полный путь к файлу
FileModDate	Строка, содержащая дату последнего изменения файла
FileSize	Целое число, указывающее размер файла в байтах
Format	Строка, содержащая формат файла, как определено fmt; для JPEG и TIFF-файлов возвращается значение, состоящее из трех символов
FormatVersion	Строка или число, описывающее версию формата
Width	Целое число, указывающее ширину изображения в пикселях
Height	Целое число, указывающее высоту изображения в пикселях
BitDepth	Целое число, указывающее число битов на пиксель
ColorType	Строка, описывающая тип изображения: 'truecolor' для RGB изображения, 'grayscale' для полутонового изображения, 'indexed' для индексированного изображения

info = imfinfo(filename) — пытается найти формат файла из его содержания.

Пример:

```
» info = imfinfo('C:\выставка\internet.bmp')
info =
                Filename: 'C:\выставка\internet.bmp'
            FileModDate: '04-Jan-1999 22:35:56'
                FileSize: 481078
                  Format: 'bmp'
           FormatVersion: 'Version 3 (Microsoft Windows 3.x)'
                   Width: 800
                  Height: 600
                BitDepth: 8
               ColorType: 'indexed'
         FormatSignature: 'BM'
      NumColormapEntries: 256
                Colormap: [256x3  double]
                 RedMask: [ ]
               GreenMask: [ ]
                BlueMask: [ ]
          ImageDataOffset: 1078
       BitmapHeaderSize: 40
              NumPlanes: 1
       CompressionType: 'none'
              BitmapSize: 480000
          HorzResolution: 0
           VertResolution: 0
          NumColorsUsed: 0
      NumImportantColors: 0
```

A = imread(filename,fmt) — считывает изображение из файла **filename** в **A**, чей класс — **uint8**. Если файл содержит полутоновое изображение , **A** является двумерным массивом. Если файл содержит изображение в формате RGB, **A** представляет собой трехмерный (m×n×3) массив. Аргумент **filename** — строка, определяющая имя графического файла, а **fmt** — строка, которая определяет формат файла. Файл должен находиться в текущей директории или в директории, указанной в пути MATLAB. Если **imread** не может найти файл с именем **filename**, она ищет файл с именем **filename.fmt**. По поводу аргумента **fmt** смотрите описание функции **imfinfo.**

[X,map] = imread(filename,fmt) — считывает индексированное изображение из файла **filename** в **X** и связанную с ним цветовую карту в **map**. **X** имеет класс **uint8** и **map** класса **double**. Значения colormap вычисляются в диапазоне [0, 1].

[...] = imread(filename) — пытается установить формат файла, исходя из его содержания.

[...] = imread(...,idx) — считывает в одно изображение TIFF-файл с несколькими изображениями. idx — целое число, определяющее порядок следования изображения в файле. Например, если idx равен 3, **imread** считывает третье изображение в файле. Если этот параметр опущен, **imread** считывает первое изображение в файле.

[...] = imread(...,ref) — считывает в одно изображение HDF-файл с несколькими изображениями. **ref** является целым числом, определяющим номер справки, используемой для опознавания изображения. Например, если **ref** равно 12, **imread** считывает изображение с номером справки, равным 12. Если этот аргумент опущен, **imread** считывает первое изображение в файле.

Следующая таблица содержит форматы изображений, доступных для чтения функцией **imread**.

Форматы файлов и их краткое описание

Формат	Варианты
BMP	1-битовые, 4-битовые, 8-битовые и 24-битовые несжатые изображения; 4- битовые и 8- битовые изображения (RLE)
HDF	8-разрядные растровые наборы данных изображения, содержащие или не имеющие связанного colormap; 24-разрядные растровые наборы данных изображения
JPEG	Любые JPEG-изображения; JPEG-изображения с некоторыми обычно используемыми расширениями
PCX	1-битовые, 8-битовые и 24-битовые изображения
TIFF	Любые TIFF-изображения, включая1-битовые, 8-битовые и 24-битовые несжатые изображения; 1-битовые, 8-битовые и 24-битовые изображения с packbit-сжатием; 1-битовые изображения с CCITT-сжатием
XWD	1-битовые и 8-битовые Zpixmaps; XYBitmaps; 1-битовые XYPixmaps

imwrite(A,filename,fmt) — записывает изображение из **A** в файл **filename**. **filename** — строка, определяющая имя выходного файла, а **fmt** — строка, определяющая формат файла. Если **A** содержит полутоновое изображение или **truecolor** (RGB) изображение класса **uint8**, команда **imwrite** записывает фактические значения массива в файл. Если **A** имеет класс **double**, команда **imwrite** переопределяет значения в массиве перед записью, используя **uint8(round(255*A))**. Эта операция преобразовывает числа с плавающей запятой в диапазоне [0, 1] к 8-битовым целым числам в диапазоне [0, 255].

Допустимые значения параметра **fmt** аналогичны тем, что используются в команде **imfinfo.**

imwrite(X,map,filename,fmt) — записывает индексированное изображение, находящееся в массиве **X**, и соответствующую ему colormap map в файл **filename**. Если **X** содержит изображение класса **uint8**, команда **imwrite** записывает фактические значения массива в файл. Если **X** имеет класс **double**, команда **imwrite** переопределяет значения в массиве перед записью, используя **uint8(X–1). map** должна быть класса **double**; **imwrite** переопределяет значения в **map**, используя **uint8(round(255*map)).**

imwrite(...,filename) — записывает изображение в **filename** в формате, указанном в расширении файла. Расширение может быть одним из допустимых значений параметра **fmt.**

imwrite(...,Parameter,Value,...) — определяет параметры, которые контролируют различные свойства выходного файла. Параметры используются для HDF, JPEG и TIFF-файлов.

Таблица параметров для HDF-файлов:

Параметр	Значение	Значение по умолчанию
'Compression'	Одно из значений 'none', 'rle','jpeg'	'rle'
'Quality'	Число между 0 и 100; параметр поддерживается для 'Compression'= 'jpeg'; чем больше число, тем выше качество файла (меньше искажений файла при сжатии) и тем больше его размер.	75
'WriteMode'	Одно из значений: 'overwrite', 'append'	'overwrite'

Таблица допустимых параметров для JPEG-файлов:

Параметр	Значение	Значение по умолчанию
'Quality'	Число между 0 и 100; параметр поддерживается для 'Compression'= 'jpeg'; чем больше число, тем выше качество файла (меньше искажений при сжатии файла) и тем больше его размер.	75

Таблица допустимых параметров для TIFF-файлов:

Параметр	Значение	Значение по умолчанию
'Compression'	Одно из значений: 'none', 'packbits', 'ccitt'; значение 'ccitt' допустимо только для двоичных изображений	'ccitt' для двоичных изображений; 'packbits' для всех остальных
'Description'	Любая строка; значение поля ImageDescription может быть возвращено командой **imfinfo**	пустая строка

M = wk1read(filename) — считывает электронную таблицу Lotus123 WK1 в матрицу **M**.

M = wk1read(filename,r,c) — считывает данные, начиная с ячейки, определенной значениями (r,c). r и c такие, что r=0, c=0 определяют первую ячейку в файле.

M = wk1read(filename,r,c,range) — считывает диапазон значений, определенный параметром **range**, где **range** может быть в виде:

1. Вектор с четырьмя элементами, определяющий диапазон ячеек в формате:
[верхняя_левая_строка, верхний_левый_столбец, нижняя_правая_строка, нижний_правый_столбец]
2. Диапазон ячеек, определенный строкой. Например: **'A1...C5'**.
3. Имя диапазона, определенное в виде строки. Например: **'Sales'**.

wk1write(filename,M) — записывает значения матрицы **M** в файл **filename** электронной таблицы Lotus123 WK1.
wk1write(filename,M,r,c) — записывает данные, начиная с ячейки, определенной значениями **(r,c)**. **r** и **c** со значениями **r=0**, **c=0** определяют первую ячейку в электронной таблице.

Необходимо отметить, что большинство рассмотренных выше функций редко применяются пользователями. Но они довольно широко используются в системных целях и представляют большой интерес для специалистов.

8.5. Звуковые возможности системы MATLAB

8.5.1. Основные команды поддержки звуковой системы

Начиная с версии MATLAB 5.0, в системе несколько расширены средства для работы со звуком. До этого система имела единственную звуковую команду

sound(Y,FS) — воспроизводит сигнал из вектора **Y** (с частотой FS) с помощью колонок, подключенных к звуковой карте ПК. Компоненты **Y** могут принимать значения в пределах — 1.0 <= y <= 1.0. Для воспроизведения стереозвука на допускающих это компьютерных платформах **Y** должен быть матрицей размера N×2.
sound(Y) — функционирует аналогично, принимая по умолчанию FS=8192 Hz.
sound(Y,FS,BITS) — функционирует аналогично с заданием разрядности звуковой карты **BITS**=8 или **BITS**=16.

Теперь появилась еще одна команда воспроизведения звука:

soundsc(Y,...) — масштабирует и воспроизводит сигнал из массива **Y**. По синтаксису команда аналогична **sound(Y,...)**.
soundsc(Y,...,SLIM) — аналогична предшествующей команде, но позволяет задать параметр **SLIM = [SLOW SHIGH]** для карты значений Y с целью воспроизведения полного динамического диапазона. По умолчанию **SLIM = [MIN(Y) MAX(Y)]**.

Кроме того, введены команды для считывания и записи файлов звукового формата. WAV, принятого для операционных систем класса Windows:

wavwrite(Y,WAVEFILE) — записывает файл типа **WAVE** под именем **WAVEFILE**.

wavwrite(Y,FS,WAVEFILE) — делает то же с заданием частоты **FS** (в Гц).
wavwrite(Y,FS,NBITS,WAVEFILE) — делает то же с заданием **NBITS**-битового формата, причем NBITS<=16.

Y=wavread(FILE) — считывает файл типа WAVE с именем (спецификацией) **FILE** и возвращает данные в массиве **Y**.
[Y,FS,BITS]=wavread(FILE) — считывает файл типа WAVE с именем FILE и возвращает массив данных **Y**, частоту **FS** и разрядность **BITS** кодирования звука (в битах).
[...]=wavread(FILE,N) — возвращает только первые **N** образцов с каждой панели файла.
[...]=wavread(FILE,[N1 N2]) — возвращает только образцы от **N1** до **N2** с каждого канала.
SIZ=wavread(FILE,'size') — возвращает объем аудио- данных в виде вектора **SIZ=[samples channels]**.

Для разных компьютерных платформ MATLAB имеет дополнительные команды. Так, для ПК класса Macintosh поддерживаются команды **speak**, **recordsound** и **soundcap**, а для компьютеров платформ NEXT и SUN — команды **auwrite** и **auread**. Их рассмотрение выходит за рамки данной книги, но пользователи, работающие на базе указанных платформ, могут получить необходимую информацию с помощью команды **help**.

8.5.2. Демонстрация возможностей работы со звуком

Для комплексной демонстрации возможностей работы со звуком служит файл-команда **xpsound**. Рис. 8.1 показывает результат ее выполнения.

Эта команда выводит диалоговое окно, которое позволяет выбрать несколько видов звукового сигнала, создать для них массив данных звука и воспроизвести звук (если ПК оснащен звуковой картой, совместимой с картой Sound Blaster). Кроме того, имеется возможность отобразить графически временную зависимость звукового сигнала, его частотный спектр и спектрограмму. На рис. 8.1 приведен пример временной зависимости звукового сигнала.

На рис. 8.2 представлен частотный спектр выбранного звукового сигнала. Он отражает относительный уровень звука в зависимости от частоты.

Еще один весьма наглядный способ представления массива данных звуковых сигналов — это показ их спектрограммы. Звуковой сигнал при этом делится на множество образцов, а спектрограмма дает представление о распределении частот спектра в разные моменты времени. Представление о том, насколько любопытной бывает спектрограмма сложного звукового сигнала, можно получить на примере рис. 8.3. Подобные спектрограммы могут быть использованы при разработке методов распознавания звуков.

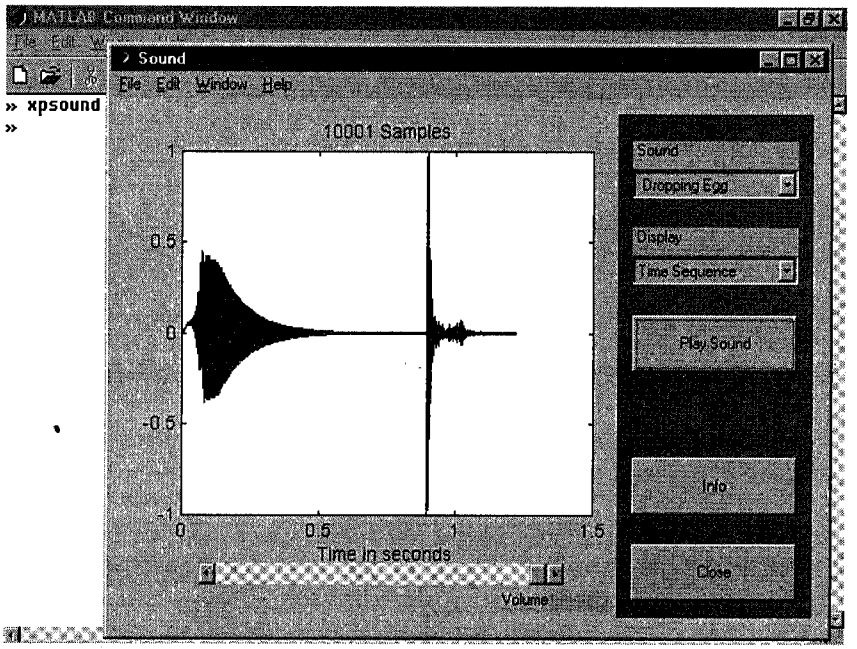


Рис. 8.1. Окно демонстрации воспроизведения звука с показом графика временной зависимости звукового сигнала

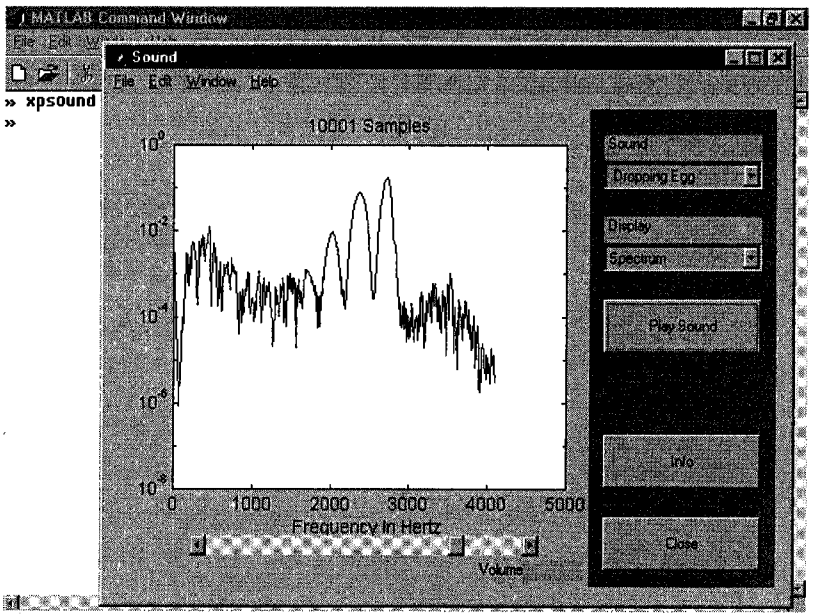


Рис. 8.2. Окно демонстрации возможности воспроизведения звука с выводом графика частотного спектра звукового сигнала

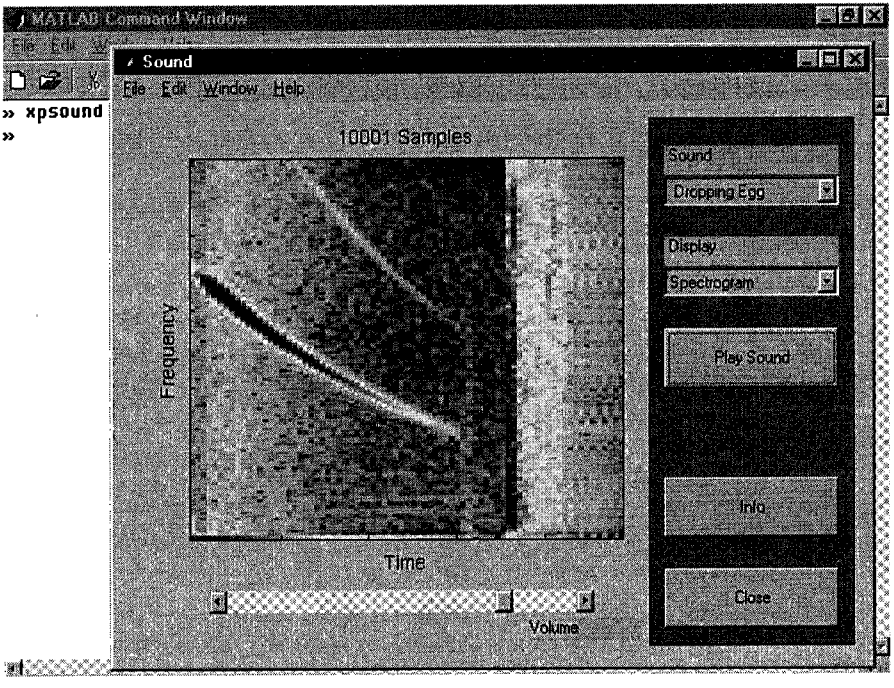


Рис. 8.3. Окно демонстрации возможности звуковоспроизведения с показом спектрограммы звукового сигнала

Демонстрационные примеры можно просмотреть с помощью команды **type xpsound**. Вы получите доступ к более подробной информации по работе со звуком в системе MATLAB.

Глава 9. Выполнение символьных операций

9.1. Назначение пакета Symbolic

9.1.1. Возможности пакета Symbolic Math Toolbox

В систему MATLAB 5.2.1 входит обновленная версия 2.0 пакета расширения Symbolic Math Toolbox (в дальнейшем сокращенно Symbolic), которая базируется на ядре символьной математической системы Maple V R4, лидирующей в области автоматизации аналитических решений. С помощью команды

```
» help symbolic
```

можно получить перечень входящих в пакет команд и функций. 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 М-файла, задающего команду или функцию.

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 системой при решении математических задач и математического моделирования в численном виде. Теперь она, с учетом новых средств, становится в полной мере **универсальной** системой.

Последняя реализация системы символьной математики Maple V R5 в своем ядре и в расширениях имеет около 2700 функций [24]. Система MATLAB с пакетом Symbolic, включающим в себя чуть больше сотни символьных команд и функций, намного уступает Maple V 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 V, что заметно расширяет круг используемых функций. Вряд ли можно считать существенным недостатком то, что в пакете Symbolic используется ядро из реализации Maple V R4 — оно более апробировано и надежно, чем ядро новейшей реализации R5 системы Maple V.

Помимо типовых **аналитических вычислений** (таких как символьное дифференцирование и интегрирование, упрощение математических выражений, подстановка и

так далее) пакет Symbolic позволяет реализовать **арифметические операции с произвольной точностью**.

О возможностях системы MATLAB в области символьной математики наглядно свидетельствуют заголовки разделов данной главы. Учитывая наличие ряда книг по системе Maple V [24-29], включая книгу одного из авторов данной книги [24], мы рассмотрим символьные возможности системы MATLAB конспективно, но достаточно полно.

9.1.2. Демонстрационные примеры работы с пакетом Symbolic

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 М-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.

9.2. Базовые символьные операции

9.2.1. Задание символьных переменных с помощью апострофов

Поскольку переменные системы MATLAB по умолчанию не определены и традиционно задаются как векторные, матричные, числовые и так далее, 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:

```
» sin(x)^2+cos(x)^2
??? Undefined function or variable 'x'.
```

```
» sin('x')^2+cos('x')^2
ans =
     1
```

Итак, в первом случае MATLAB "возмутилась" нашей небрежностью и сообщила что функция или переменная x не определена, и ни о каких вычислениях синуса и ко синуса речи быть не может. Вместе с тем она подсказала, как надо сделать это определение — заключить имя переменной в апострофы, ибо таким образом система получа ет 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.

Этот результат сразу выдает 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, а в нашем случае — это ядро системы Maple V R4.

Символьные переменные имеют много общего со строковыми, которые также задаются с использованием апострофа. 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.

9.2.2. Функция создания символьных переменных sym

Для создания символьных переменных или объектов используется 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:

```
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 виде:

'f' — число с плавающей точкой **'1.F'*2^(e) or '-1.F'*2^(e)**, где F является строкой из 13 шестнадцатеричных чисел, а e — целым числом.

Примеры применения функции **sym** представлены ниже:

```
» sym(1/50,'f')
ans =
'1.47ae147ae147b'*2^(-6)
» sym(4/6,'r')
ans =
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 оценивать его как символьный в отличие от обычных численных результатов.

9.2.3. 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 не вещественными (unreal) значениями. Последнюю функцию можно использовать для отмены задания вещественности объектов.

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

9.2.4. Функция создания списка символьных переменных findsym

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 используемую в этих функциях символьную переменную.

9.2.5. Функция вывода символьного выражения pretty

К сожалению, в отличие от современных систем символьной математики (MathCAD, Maple V 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 так далее. Этот не-

достаток, однако, часто оборачивается достоинством — запись выражений 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.

9.2.6. Функция 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.

9.2.7. Функция представления выражений в кодах языка C — ccode

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 **ccode** облегчает программистам создание библиотек программ на языке C и уменьшает вероятность ошибок в математических выражениях.

9.2.8. Функция представления 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, пишущих программы 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.

9.3. Функции математического анализа

9.3.1. Функция вычисления производных diff

Для вычисления в символьном виде производных от выражения S служит функция **diff**, записываемая в формате:

diff(S,'v') или **diff(S,sym('v'))** — 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-ую (n — целое число) производную S от символьного выражения или массива символьных выражений S по переменной v.

diff(S,'v',n) и **diff(S,n,'v')** — возвращает n-ую производную S по переменной v, т. е. 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)]
```

9.3.2. Функция вычисления интегралов int

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

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

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 переменной, 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 таких 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
```

Следующий пример относится к вычислению тройного интеграла:

$$\int_0^a\int_0^a\int_0^a (x^2+y^2)zdxdydz.$$

Здесь можно трижды использовать функцию **int**:

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

Первый пример вычисляет интеграл с бесконечным верхним пределом, а второй — тройной интеграл.

9.3.3. Функция вычисления пределов limit

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→a), значение функции неограниченно приближается к L. Это обозначается следующим образом:

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

Предел может быть конечным числом, положительной и отрицательной бесконечностью.

Есть функции (например, разрывные в точке x=a), у которых нет предела в самой точке x=a, но есть предел при x→a-0 или при x→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 (символьно) 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 предел в точке a 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]
```

9.3.4. Функция разложения выражения в ряд Тейлора — taylor

В задачах аппроксимации и приближения функций f(x) важное место занимает разложение их 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 порядке можно задавать число членов ряда n, точку, относительно которой ищется разложение a 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
```

9.3.5. Функция вычисления матрицы Якоби — jacobian

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:

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

Для ее вычисления используется функция:

jacobian(f,v) — возвращает матрицу Якоби для скаляра или вектора **f** по вектору переменных **v**. Каждый (i,j)-ый элемент матрицы представляет df_i/dv_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]
```

9.3.6. Функция вычисления сумм рядов symsum

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

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

Такие суммы принято называть конечными. При b=∞ 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 findsum.
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 числа членов от a до b.

Примеры вычисления 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]
```

9.4. Символьная линейная алгебра

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.

9.4.1. Задание или извлечение диагональных элементов матриц — diag

Для создания диагональных матриц и извлечения 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-ой диагонали размещен вектор **V**. При K = 0 вектор **V** располагается на главной диагонали, при K>0 — на K-ой 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-ую 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]
```

9.4.2. Формирование верхней треугольной матрицы — triu

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-ой диагонали (правила задания K были описаны выше).

Остальные элементы матрицы **X** заменяются нулями. Примеры на применение 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]
```

9.4.3. Формирование нижней треугольной матрицы — tril

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-той диагонали (правила задания K были описаны выше).

Остальные элементы матрицы **X** заменяются нулями. Примеры на применение 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]
```

9.4.4. Обращение матрицы — inv

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

```
» 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)]
```

9.4.5. Вычисление детерминанта матрицы — det

Функция **det(X)** вычисляет детерминант квадратной матрицы в символьном виде. Например:

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

9.4.6. Вычисление ранга матрицы — rank

Для вычисления ранга квадратной матрицы используется функция **rank**:

rank(A,tol) — число сингулярных значений матрицы **A**, определенных с погрешностью tol.

rank(A) — аналогична предшествующей функции, но при tol = max(size(**A**)) * norm(**A**) * eps.

Пример:

```
» syms a b c d;
» rank([a b;c d])
```

```
ans =
2
```

9.4.7. Приведение матрицы к верхней треугольной форме — rref

Для приведения матрицы A 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 принимается значение порога допустимости для незначительного элемента столбца, равное **(max(size(A))*eps *norm(A,inf))**.

[R,jb] = rref(A) — также возвращает вектор **jb**, так что: **r = length(jb)** может служить оценкой ранга матрицы **A**, **x(jb)** — связанные переменные в системе линейных уравнений вида **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.

9.4.8. Нуль-пространство матрицы — null

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]
```

9.4.9. Базис-пространство столбцов — colspace

Функция

B = 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]
```

9.4.10. Вычисление собственных значений и собственных векторов матриц — eig

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
```

9.4.11. Сингулярное разложение матриц — svd

Для сингулярного разложения матрицы **A** 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 ]
```

9.4.12. Вычисление канонической формы Жордана — jordan

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 подобного преобразования **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) ]
```

9.4.13. Вычисление характеристического полинома матриц — poly

Для вычисления характеристического полинома матрицы **A** используется функция:

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
```

9.4.14. Вычисление матричного экспоненциала — expm

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)]
```

9.5. Аналитические операции с выражениями

9.5.1. Функция упрощения выражений simplify

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
```

Возможности проведения упрощений с помощью команды **simplify** в Symbolic не обладают возможностями системы Maple V в полной мере в связи с отсутствием опций, определяющих путь упрощения. Дополнительные возможности упрощения обеспечивает функция **simple**.

9.5.2. Функция расширения выражений — expand

Функция **expand(S)** расширяет выражения, входящие в массив **S**. Рациональные выражения она раскладывает на простые дроби, полиномы — на полиномиальные раз

ложения и так далее. 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
```

9.5.3. Разложение выражений на простые множители — factor

Функция **factor(S)** разлагает поэлементно 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)
```

9.5.4. Комплектование по степеням — collect

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]
```

9.5.5. Упрощение выражений — simple

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

9.5.6. Приведение к рациональной форме — numden

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
```

9.5.7. Приведение к схеме Горнера — horner

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)]
```

9.5.8. Запись с подстановками — subexpr

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 так далее для **pretty(S)**. 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
```

9.5.9. Обеспечение подстановок — subs

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)]
```

9.6. Решение уравнений в символьном виде

9.6.1. Решение алгебраических уравнений — solve

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 при числе неизвестных, равных 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.

9.6.2. Решение дифференциальных уравнений — dsolve

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, включив 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 так далее. 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, а и b — константы. Если число начальных условий меньше, чем число дифференциальных уравнений, то в решении будут присутствовать произвольные постоянные C1, C2 и так далее. 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)
```

9.6.3. Обращение функции — finverse

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 **f**. Считается, что **f** — функция одной переменной, например 'x'. Тогда g(f(x)) = x.

g=finverse(f,v) — возвращает функцию, обратную f, относительно заданной переменной v, так что g(f(v)) = v. Эта форма используется, если f — функция ряда переменных.

Примеры:

```
» syms x
» finverse(sinh(x))
ans =
asinh(x)
» finverse(exp(x))
ans =
log(x)
```

9.6.4. Суперпозиция функций — compose

К числу часто встречаемых 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 x 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)
```

9.7. Арифметика произвольной точности

9.7.1. Установка количества знаков чисел — digits

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 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 =
3.14159265358979323846264338327 95
» digits(6)
» pi
ans =
  3.1416
```

9.7.2. Вычисления в арифметике произвольной точности — vpa

MATLAB обычно ведет вычисления с числами, представленными в формате плавающей точки с двойной точностью. Это довольно высокая точность, обеспечивающая потребности практических вычислений в прикладных задачах. Однако ряд задач теории чисел, численного кодирования и некоторых других требует выполнения вы-

числений вообще без какой-либо погрешности или со сколь угодно малой погрешностью. Не очень удачно такие вычисления 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 D цифр, установленным функцией **digits**. Результат R типа **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]
```

9.8. Интегральные преобразования

В этом разделе описаны наиболее важные интегральные преобразования. Преобразования Фурье составляют основу метода спектрального анализа сигналов и вопросов, связанных с анализом радиотехнических цепей. Трудно переоценить и преобразования Лапласса, 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.

9.8.1. Прямое преобразование Фурье — fourier

Прямым преобразованием Фурье является следующее преобразование:

$$F(w) = \int_{-\infty}^{\infty} f(x) e^{iwx} dx,$$

где f(x) — скалярная функция независимой переменной x. Оно реализуется функцией:

F=fourier(f) — которая 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)=>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 w 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)
```

9.8.2. Обратное преобразование Фурье — ifourier

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:

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) => f = f(x). Если F = F(x), то данная функция возвращает 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)
```

9.8.3. Прямое преобразование Лапласа — laplace

Прямое преобразование Лапласа осуществляется по основной формуле:

$$L(s) = \int_{0}^{\infty} f(t)e^{-st}dt.$$

Для осуществления этого преобразования используется функция:

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 на прямое преобразование 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  3/4*pi^(1/2)/t^(5/2)
laplace(diff(sym('F(t)'))) возвращает  laplace(F(t),t,s)*s-F(0)
```

9.8.4. Обратное преобразование Лапласа — ilaplace

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:

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)
```

9.8.5. Z-преобразование — ztrans

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 для скалярной функции f независимой переменной n (по умолчанию). В MATLAB оно реализуется функцией:

F=ztrans(f) — обеспечивает прямое Z-преобразование вида f = f(n) => F = F(z), где n — символьная переменная, определяемая функцией findsym. Если f = f(z), то ztrans(f) возвращает F=F(w).

Функция

F=ztrans(f,w) — возвращает F, заменяя аргумент по умолчанию z на w, т.е. 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 на прямое Z-преобразование:

```
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
```

9.8.6. Обратное Z-преобразование — iztrans

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,... и R — положительное число, определяющее аналитичность функции F(z) и вне круга |z|=R. Такое преобразование реализуется следующей функцией:

f=iztranse(F) — возвращает результат обратного Z-преобразования для скалярной символьной функции F независимой переменной z. Это преобразование определяется как: F = F(z) => 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 умолчению n на k. Таким образом она реализует преобразование по формуле:

$$f(k)=\frac{1}{2\pi i}\oint_{|z=R|}F(z)z^{k-1}dz,$$

где k=1,2,...

Наконец, функция

f=iztrans(F,w,k) — делает еще одну замену, заменяя аргумент исходной функции (по умолчению 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!**

9.9. Функции преобразования объектов

При практической работе с системами символьной математики зачастую необходимы взаимные преобразования различных объектов, например, численных в символьные и наоборот. Так, 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.

9.9.1. Преобразование символьной матрицы в числовую — double

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 на это преобразование 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
```

9.9.2. Преобразование вектора коэффициентов полинома в символьный полином — poly2sym

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 третьей степени оказался представленным значением sin(1) с дробно-рациональным видом.

9.9.3. Преобразование символьного полинома в вектор его коэффициентов — sym2poly

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.

9.9.4. Преобразование символьного объекта в строковый — char

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
```

9.10. Специальные функции

Несколько специальных функций удостоились чести попасть в пакет расширения Symbolic. Все они описаны в данном разделе.

9.10.1. Интегральный синус — sinint

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
```

9.10.2. Интегральный косинус — cosint

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
```

9.10.3. Дзета-функция Римана — zeta

Дзета-функция Римана определяется выражением

$$\varsigma(s) = \sum_{k=1}^{\infty} k^{-s}$$

при $Re(s)>0$. 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
```

9.10.4. W- функция Ламберта — lambertw

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-ую комплексную ветвь для многозначной функции. Примеры применеия этой функции:

```
» lambertw(2+3i)
ans =
  1.0901 + 0.5301i
» lambertw(3,2+3i)
ans =
 -1.6214 +18.1726i
```

9.11. Строчные утилиты

Трудно сказать, почему разработчики MATLAB выделили описанные ниже две функции в отдельную группу. Мы опишем их "как есть".

9.11.1. Контроль допустимости имен — isvarname

Функция

isvarname(S)

возвращает логическое значение 1, если имя S допустимо в системе MATLAB и 0, если оно не допустимо. 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
```

9.11.2. Векторизация символьных выражений — vectorize

Векторизация означает проведение почленного преобразования элементов матриц и векторов. В MATLAB функция

vectorize(S)

для символьного выражения S вставляет знак '.' после всех 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
```

9.12. Дополнительные средства

В оригинале описанные ниже средства названы "педагогическими". Они и впрямь помогут преподавателям вузов в визуализации многих понятий теории функций, в частности в получении их наглядного графического представления.

9.12.1. Суммы Римана — rsums

Функция

rsums(f) и **rsums f**

вычисляет приближение Римана к интегралу с подынтегральной функцией f(x) и строят в виде столбиковой диаграммы график функции и площадей под кривой.

Пример применения данной функции представлен на рис. 9.1.

Обратите внимание на характерный ползунковый регулятор под графиком представления площадей. Меняя положения движка, можно изменять в широких пределах число сумм и отслеживать за тем, насколько точно они представляют выбранную функцию.

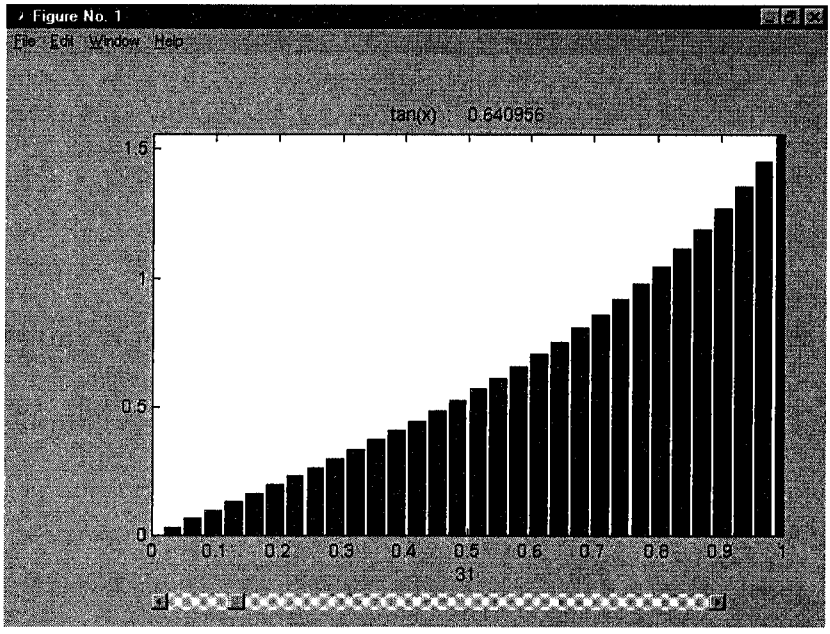


Рис. 9.1. Пример применения функции rsums

9.12.2. Графики символьных функций — ezplot

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'. Она определена 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 2. Другой пример такого рода — построение графика функции tan(x), имеющего разрывы:

```
» ezplot('tan(x)',0,20)
» grid on
```

На рис. 9.2 показан вид построенного графика. Он выглядит намного естественнее, чем аналогичный график, построенный командой **plot**.

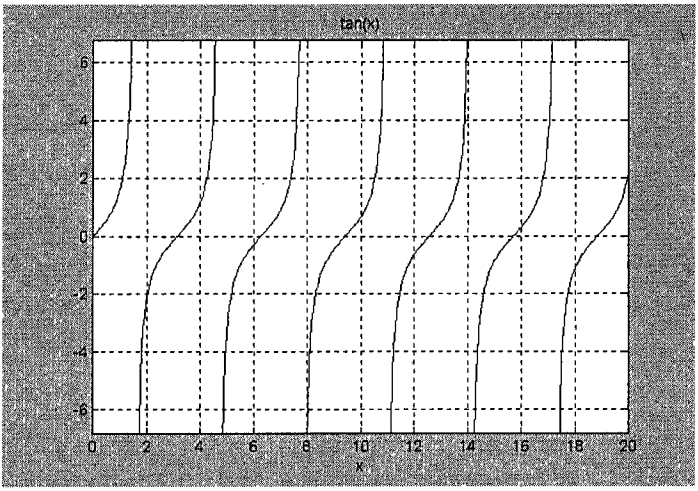


Рис. 9.2. График функции tan(x)

Эти команды лежат в основе специального приложения — вычислителя функций и графопостроителя Funtool, описанного ниже.

9.12.3. Вычислитель функций и графопостроитель — funtool

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 функции 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. 9.3.) По умолчанию заданы функции f(x)=x и g(x)=1, предел изменения x от -2*pi до 2*pi и a=1/2. Все окна масштабируемые и перемещаемые.

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.

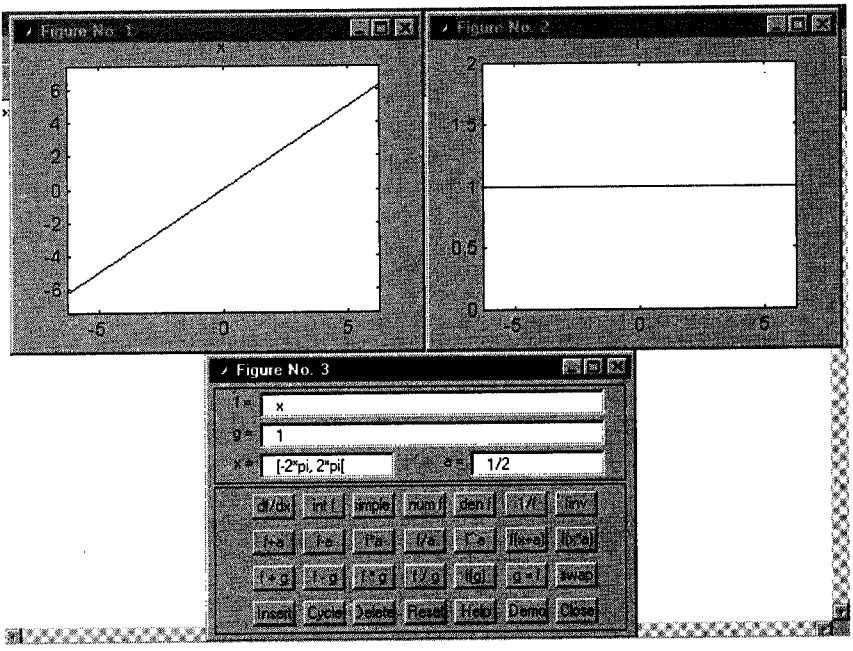


Рис. 9.3. Внешний вид вычислителя функций и окон двух графиков

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 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 работу 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. 9.4 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**.

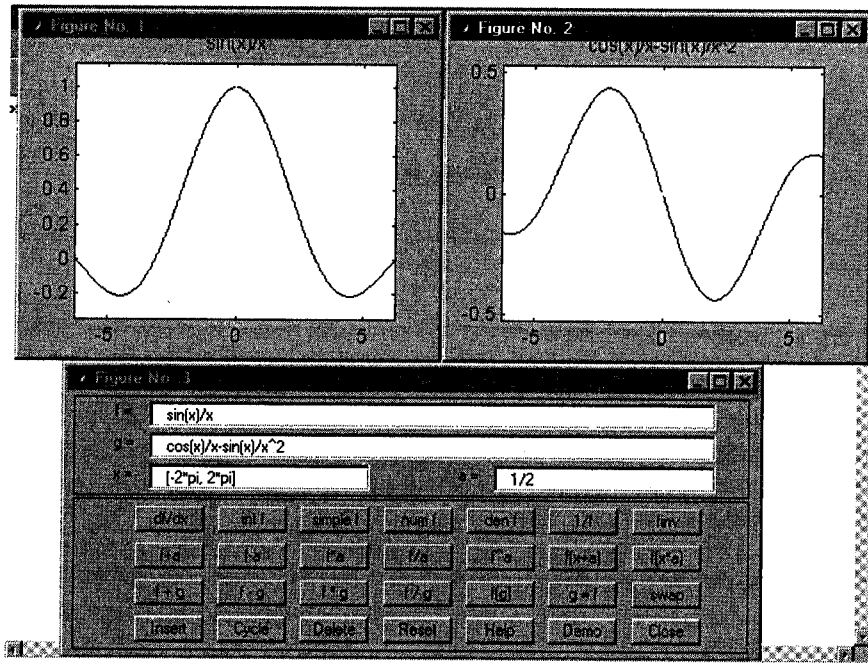


Рис. 9.4. Графики функции sin(x)/x и ее производной

Кнопка **Cycle** позволяет просмотреть графики ряда интересных функций (рис. 9.5) 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 интерес щие вас функции.

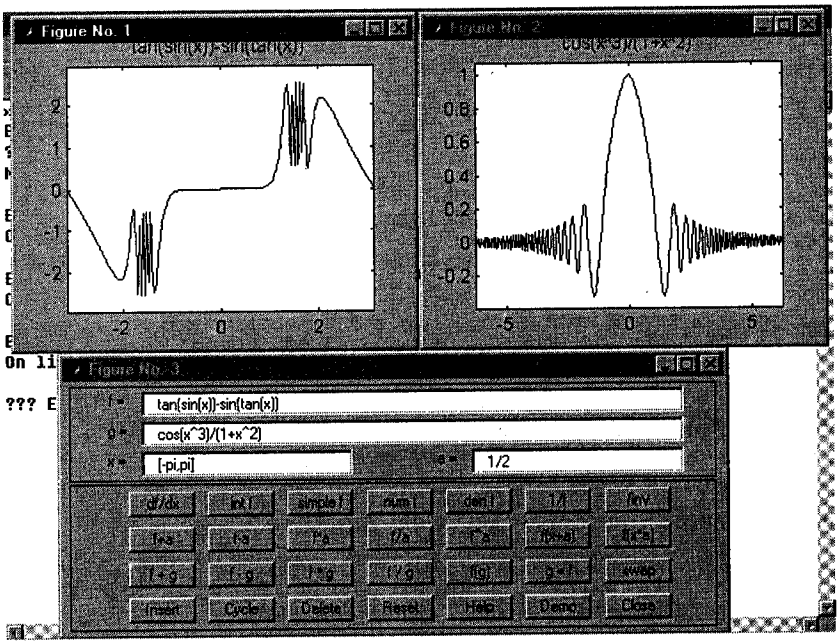


Рис. 9.5. Фрагмент демонстрации построения функций из списка fxlist

Вычислитель и графопостроитель и впрямь являются весьма удобным средством визуализации графиков самых различных функций, и в качестве утилиты заменяют множество объемных книг по графикам элементарных и специальных функций.

9.13. Доступ к ресурсам ядра системы Maple V

В этом разделе описаны функции, дающие доступ к ресурсам ядра системы символьной математики Maple V R4, включенным в систему MATLAB. В студенческой версии — последней — эти возможности отсутствуют.

Применение возможностей системы Maple V R4 совместно с возможностями системы MATLAB придает последней особую гибкость и резко расширяет возможности в решении сложных математических задач, в которых целесообразно объединять аналитические (символьные) методы с численными вычислениями.

9.13.1. Доступ к ядру системы Maple V — maple

Хотя в пакет Symbolic включено множество (около сотни) функций символьной математики, есть возможность получить доступ к многим другим функциям системы Maple V, 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 Maple-команд. 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)
```

9.13.2. Численное вычисление Maple-функций — mfun

Для вычисления числовых значений функций ядра системы Maple используется функция

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
```

9.13.3. Вызов списка функций Maple V — mfunlist

Команда **mfunlist** выводит лист с перечнем функций ядра Maple V. 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
.....
```

9.13.4. Получение справки по ядру Maple V — mhelp

Для получения справки по Maple-функциям непосредственно из среды MATLAB служат команда и функция:

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.
```

9.13.5. Инсталляция Maple-процедур — procread

С помощью специального приложения Extended Symbolic Toolbox можно готовить процедуры с синтаксисом языка системы Maple V R4. Для их инсталляции служит команда:

```
procread(FILENAME)
```

Например, подготовив процедуру с предполагаемым именем файла "check.src"

```
check := proc(A)
   #  check(A) computes A*inverse(A)
   local X;
   X := inverse(A):
   evalm(A &* X);
end;
```

можно проинсталлировать ее с помощью команды

```
procread('check.src')
```

После возможно использование этой процедуры в виде

```
maple('check',magic(3))
```
или
```
maple('check',vpa(magic(3)))
```

К сожалению, приложение Extended Symbolic Toolbox в стандартную поставку MATLAB не входит. Его необходимо приобретать отдельно.

Глава 10.
Пакет моделирования динамических систем Simulink

10.1. Назначение пакета Simulink и интегрированных пакетов

В состав расширенных версий системы MATLAB входит пакет расширения **Simulink**. В MATLAB 5.2.1 используется предпоследняя версия этого пакета — Simulink 2.0, которая на момент подготовки рукописи книги могла обновляться по Internet (или с CD-ROM) до версии 2.2.1. Она, за редкими исключениями, совместима с предшествующими версиями 1.0 и 1.3, сыгравшими большую роль в плане популярности этого уникального пакета. В реализации MATLAB 5.3 используется последняя версия пакета Simulink 3.0. С Simulink органично связан целый ряд других пакетов, краткое описание которых приводится в данном разделе.

10.1.1. Функции пакета Simulink

Пакет моделирования динамических систем Simulink является ядром интерактивного программного комплекса, предназначенного для **математического моделирования** линейных и нелинейных динамических систем и устройств, представленных своей функциональной блок-схемой, именуемой **моделью**. Simulink может поставляться самостоятельно, но входит в состав расширенной версии систем класса MATLAB. При этом возможны различные варианты моделирования: во временной области, в частотной области, с событийным управлением, на основе спектральных преобразований Фурье, с использованием метода Монте-Карло и так далее [49].

Полное описание пакета Simulink значительно превышает объем данной книги, так что приведенные ниже сведения предназначены для начального знакомства с этим мощным и замечательным расширением системы MATLAB. Они даны в той мере, которая достаточна для понимания возможностей пакета и начала работы с ним.

Для построения функциональной блок-схемы моделируемых устройств Simulink имеет обширную **библиотеку** блочных компонентов и удобный **редактор блок-схем**. Последний основан на использовании возможностей графического интерфейса пользователя и по существу является типичным средством **визуального программирования**. Используя **палитры компонентов** (наборы) блок-схем, пользователь с помощью мыши переносит нужные компоненты с палитр на рабочий стол пакета Simulink и соединяет линиями входы и выходы блоков. Таким образом создается блок-схема моделирования системы или устройства.

Simulink автоматизирует следующий, наиболее трудоемкий этап моделирования: он составляет и решает сложные системы алгебраических и дифференциальных урав-

нений, описывающих заданную функциональную схему (модель), обеспечивая удобный и наглядный визуальный контроль за поведением созданного пользователем **виртуального устройства**. Вам достаточно уточнить (если нужно) вид анализа и запустить Simulink в режиме **симуляции** (откуда и название пакета — Simulink) созданной модели системы или устройства. У нас слово "симуляция" носит несколько нарицательный оттенок, так что мы будем использовать термин "моделирование".

Средства визуализации результатов моделирования в пакете Simulink настолько наглядны, что порой создается ощущение, будто созданная вами в виде блок-схемы модель работает как "живая". Более того, Simulink практически мгновенно меняет математическое описание модели по мере ввода ее новых блоков, даже в том случае, когда этот процесс сопровождается сменой порядка системы уравнений и ведет к существенному качественному изменению поведения системы. Впрочем, это следует отнести к одной из главных целей пакета Simulink.

Ценность Simulink заключается и в обширной, открытой для изучения и модификации библиотеке компонентов. Она включает в себя источники сигналов с практически любыми временными зависимостями, масштабирующие, линейные и нелинейные преобразователи с разнообразными формами передаточных характеристик, квантующее устройство, интегрирующие и дифференцирующие блоки и так далее.

В библиотеке имеется целый набор регистрирующих устройств — от простых измерителей типа вольтметра или амперметра до универсальных осциллоскопов, позволяющих просматривать временные зависимости выходных параметров моделируемых систем — например токов и напряжений, перемещений, давления и тому подобное. Имеется даже графопостроитель для построения фигур в полярной системе координат, например, фигур Лиссажу и фазовых портретов колебаний. Simulink имеет средства анимации и звукового сопровождения. А в дополнительных библиотеках можно отыскать и такие "дорогие приборы", как анализаторы спектра сложных сигналов, многоканальные самописцы и средства анимации графиков.

Программные средства моделирования динамических систем известны давно, например, программа Tutsim или LabVIEW for Industrial Automation. Однако для эффективного применения таких средств необходимы высокоскоростные решающие устройства. Интеграция одной из самых быстрых математических систем MATLAB с пакетом Simulink открывает новые возможности использования самых современных математических методов для решения задач динамического и ситуационного моделирования сложных систем и устройств.

Средства анимации Simulink позволяют строить виртуальные физические лаборатории с наглядным представлением результатов моделирования. Возможности Simulink охватывают задачи математического моделирования сложных динамических систем в физике, электро- и радиотехнике, в биологии и химии, словом — во всех областях науки и техники. Этим объясняется популярность данного пакета как в университетах и институтах, так и научных лабораториях.

И, наконец, важным достоинством пакета является возможность задания в блоках произвольных математических выражений, что позволяет подчас решать типовые задачи, пользуясь примерами пакета Simulink или же просто задавая новые выражения, описывающие работу моделируемых пользователем систем и устройств. Важным свойством пакета является и возможность задания системных S-функций с включением их в состав библиотек Simulink. Необходимо отметить также возможность моделирования устройств и систем в реальном масштабе времени.

Как программное средство Simulink — типичный представитель визуально-ориентированного языка программирования. На всех этапах работы, особенно при подготовке моделей схем, пользователь практически не имеет дела с обычным программированием. Программа автоматически генерируется в процессе ввода выбранных блоков компонентов, их соединений и задания параметров компонентов.

Важное достоинство Simulink — это интеграция с системой MATLAB и рядом других пакетов расширения, что обеспечивает по существу неограниченные возможности в применении Simulink для решения практически любых задач имитационного моделирования. Наиболее важные пакеты кратко описаны ниже, а в дальнейшем будут рассмотрены и примеры их применения. В Приложении имеется аннотационное описание всех пакетов прикладных программ.

В новую версию пакета Simulink 3.0, интегрированную с MATLAB 5.3, добавлены следующие возможности:

- Интегрированный браузер моделей (Windows 95/98/NT)
- Возможность увеличения блок-диаграмм (zooming)
- Блок Scope может работать с многими портами и осями координат
- Интегрированные возможности линейного анализа
- Графический интерфейс для описания свойств сигнала
- Интегрированный браузер библиотек (только Windows 95/98/NT)
- Новые блоки Subsystem block, Round Sum block, Enhanced Mux block, Bus Selector block и Model Info block
- Поддержка различных типов данных и их преобразований
- Поддержка комплексных чисел при работе с базовыми блоками и комплексно-вещественные преобразования
- Оптимизация скорости и использования памяти при моделировании
- Многоцветные изображения, метки портов и маскированных блоков
- Блоки, определяемые пользователем с поддержкой множества портов и различными интервалами дискретизации

Эти новые возможности, безусловно, улучшают работу с пакетом. Однако они никак не сказываются на основах технологии его применения. Поэтому приведенные ниже избранные материалы по применению пакета Simulink в равной мере относятся ко всем его версиям, входящим в расширенные поставки систем MATLAB 5.0/5.3.

10.1.2. Пакет Communications

Пакет Communications является как бы самостоятельным средством для моделирования коммуникационных систем и устройств. Он позволяет вести разработку, анализ и тестирование моделей цифровых и аналоговых систем и устройств связи и передачи информации. В то же время он является частью пакета Simulink и включает в себя более 100 функций MATLAB и около 150 компонентов Simulink для разработки и моделирования таких систем, как устройства радиосвязи, модемы и устройства хранения информации. Пакет является прекрасным средством для использования в научных разработках в области коммуникационных устройств, а также для обучения студентов по специальностям, связанным с информационными технологиями.

Пакет Communications интегрирован с пакетом Signal Processing, предназначенным для проектирования сложных устройств обработки сигналов. Такой набор объединенных средств позволяет создавать, моделировать и макетировать проекты коммуникационных систем с необыкновенной скоростью, гибкостью и легкостью. Вы можете проверять свои идеи уже на ранней стадии разработки проекта. А для быстрого создания прототипов плат цифровой обработки сигналов можно использовать пакет Real Time Workshop, позволяющий к тому же генерировать переносимый код языка программирования C.

Пакет Communications обеспечивает возможности моделирования следующих систем и устройств:

- кодирования и оцифровки
- контроля ошибок при кодировании
- модуляции и демодуляции
- фильтрации при передаче и приеме
- систем синхронизации
- аналоговых и цифровых систем фазовой автоподстройки частоты
- коллективного доступа
- полей Галуа
- генераторов сложных сигналов
- устройств анализа и построения графиков

Пакет Communications имеет обширную библиотеку блоков для создания линейных и нелинейных, дискретных, непрерывных, гибридных и иных моделей. Для пакета характерны:

- иерархическая структура моделей с неограниченной вложенностью
- скалярные и векторные связи
- средство для создания пользовательских блоков и библиотек
- интерактивное моделирование с "живым" отображением на экране
- семь методов интегрирования с фиксированным и переменным шагом
- возможность линеаризации задач

- возможность моделирования методом Монте-Карло
- анализ устойчивости и определение точек равновесия
- различные способы вывода на экран и библиотека входных сигналов

Благодаря отмеченным свойствам и богатейшей библиотеке компонентов пакет Communications ускоряет создание и тестирование моделей сложных систем, предоставляя интегрированные средства для разработки алгоритмов и проектирования систем. Он открывает широкие возможности для апробации различных алгоритмов работы коммуникационных устройств с применением современных математических средств их моделирования, включая язык матричных вычислений, быстрого дискретного преобразования Фурье, техники фильтрации сигналов и так далее. Моделирование происходит с помощью функциональных схем непосредственно в среде пакета Simulink.

Пакет Communications содержит прекрасную документацию. Помимо полного руководства пользователя, включающего в себя описание всех блоков MATLAB и Simulink, в пакет включены примеры приложений, такие как, моделирование модема, работающего по протоколу v.34.

10.1.3. Пакет проектирования событийно-управляемых систем Stateflow

Stateflow — еще одно дополнение к среде моделирования Simulink. Это графический инструментарий для проектирования сложных систем управления и наблюдения за их поведением. Stateflow дает возможность моделировать поведение сложных **событийно-управляемых систем**, базируясь на теории конечных автоматов, что позволяет пользователям Simulink применять событийно-управляемое поведение к их моделям. Используя Simulink и Stateflow, можно проектировать, моделировать и имитировать, в том числе с помощью потоковых диаграмм, работу системы управления в единой интегрированной среде.

Потоковые диаграммы позволяют легко создавать схемы традиционных программных структур, таких как циклы **for** и операторы **if-then-else**, что превосходит возможности большинства графических сред программирования. Stateflow объединяет достоинства диаграмм состояний и структурных схем с потоковыми диаграммами и рядом других новшеств, включая эффективную генерацию С-кода. Эти свойства делают Stateflow идеальной средой для разработки систем управления в авто-, аэрокосмических и телекоммуникационных приложениях.

Пакет Stateflow обеспечивает следующие возможности:

- графическую среду программирования, включающую общеизвестные структуры программ
- поддержку иерархии, параллелизма, предыстории, а также представления в виде блок-схем

- анимацию и средства отладки
- просмотр и модификацию неграфических объектов с помощью средства Explorer
- средство Finder помогает определить положение объектов по критерию поиска
- обширные возможности справки

Пакет Stateflow тесно связан с Simulink и MATLAB. Simulink дает возможность разрабатывать модели непрерывных и дискретных динамических систем в графической среде, в то время как MATLAB обеспечивает доступ к данным, высокоуровневому программированию и инструментам визуализации.

Пакет Stateflow позволяет генерировать код с помощью специальной программы Stateflow Coder. При этом можно использовать С-код, сгенерированный Stateflow Coder, независимо или совместно с кодом Real-Time Workshop. Выразительные графические обозначения и богатая семантика кодера позволяют оптимизировать определенные аспекты сгенерированного кода. Изменяя графическое представление моделируемой системы, можно оптимизировать диаграммы Stateflow в плане скорости и размера памяти.

Комбинация Simulink и Stateflow облегчает моделирование систем с множественными режимами поведения. Применяя новые возможности исполняемых по условию подсистем в Simulink, вы можете включать управляющую логику, используя одновременно диаграммы Stateflow для выборочной активации и дезактивации подсистем. Например, в модели автопилота вертолета диаграммы Stateflow могут принимать входные данные из модели Simulink (скорость, переключатель нагрузки на колеса, переключатель для выбора способа действия вертолета: зависание, наземное управление, крейсерский полет и т.п.). Впоследствии для каждого из операционных режимов в Simulink включаются и выключаются передаточные функции с помощью диаграммы Stateflow.

10.1.4. Библиотека моделирования цифровых фильтров DSP Blockset

Одной из важных сфер применения пакета Simulink является проектирование цифровых систем обработки сигналов — цифровых сигнальных процессоров DSP (Digital Signal Processor) или, иначе говоря, цифровых фильтров. Поддержка этих возможностей осуществляется с помощью библиотеки DSP Blockset. Блоки DSP представляют собой интуитивные графические инструменты быстрого проектирования, моделирования и макетирования цифровых систем обработки сигналов. Библиотека содержит более 100 блоков для создания и анализа моделей DSP.

В сочетании с MATLAB и Simulink библиотека блоков DSP упрощает создание DSP-систем, интегрируя разработку алгоритмов и моделирование в реальном масштабе времени.

Версия 2.0 библиотеки DSP существенно обновлена по сравнению с предыдущей версией. Она включает множество новых библиотек и блоков, расширяющих возможности моделирования как дискретных, так и гибридных систем. Среди них: поддержка адаптивной и многоступенчатой фильтрации, матричная математическая библиотека, блоки обработки сигналов для моделирования во временной и частотной областях, блоки спектрального анализа, а также новые вспомогательные блоки, такие как переключатели и счетчики.

Библиотека DSP Blockset включает следующие функции обработки сигналов:

- быстрое прямое и обратное преобразование Фурье, фильтры и средства буферизации
- введение новых блоков — переключателей и счетчиков
- поддержка выборочной обработки и возможность расширения окна
- комплекс математических операций по обработке сигналов с определением величины, фазы, гармоники, сопряжением и разделением сигналов
- блоки векторной математики и статистики, обеспечивающие быстрые вычисления и более совершенную обработку сложных сигналов
- адаптивное и множественное фильтрование с дополнительным использованием FIR, IIR и других блоков для создания фильтров
- включение блоков для моделирования обработки сигналов во временной и частотной областях
- блоки спектрального анализа, реализующие метод Велша (Welch)
- новая "истинная библиотека" упрощает динамическое связывание, обновление и разделение блоков.

Новые блоки расширяют возможности применения Simulink для проектирования, моделирования и макетирования цифровой обработки сигналов (DSP) в таких устройствах и системах, как проводная и беспроводная связь, периферийные устройства компьютеров, обработка речи, управление автомобилями и медицинская диагностика.

10.1.5. Пакет моделирования в реальном времени Real Time Workshop

В расширенных вариантах поставки системы MATLAB фирма MathWorks оснащает Simulink системой моделирования в реальном времени **Simulink Real Time Workshop** (RTW). Предусмотрена возможность компиляции созданных в RTW программ и разработка на их основе законченных программных продуктов.

Пакет Real-Time Workshop является дополнением к пакетам Simulink, Stateflow, DSP Blockset и Communications Toolbox. Real-Time Workshop позволяет выполнять моделирование динамических систем на различных платформах для имитации в реальном времени. Он осуществляет методы быстрого макетирования, в том числе нату-

рального с применением плат расширения компьютера, которые позволяют корректировать ошибки проектирования и определять узкие места на ранних стадиях.

Пакет Real-Time Workshop обладает следующими возможностями:

- обеспечивает автоматическую генерацию кодов для непрерывных, дискретных, событийно-управляемых и гибридных систем
- создает оптимизированный, переносимый, краткий и четкий код. Код содержит необходимые комментарии, и его легко модернизировать
- позволяет задавать метки сигналов и названия блоков, которые переносятся из названий моделей в генерируемый код.API, который использует пользовательские файлы для автоматического создания и загрузки объектных файлов
- содержит компилятор Target Language Compiler (TLC), позволяющий настраивать сгенерированный код. С помощью TLC вы можете выполнять подключенные S-функции или создавать новые алгоритмы для встроенных блоков
- обеспечивает интерактивную загрузку параметров из блок-диаграмм во внешние устройства для оперативной настройки системы
- имеет наглядный Graphical User Interface (GUI)
- обеспечивает поддержку проектной среды Spectron Microsystems (SPOXworks).

10.1.6. Интеграция пакета Simulink с системой MATLAB

После инсталляции Simulink (отдельно от MATLAB или в ее составе) он автоматически интегрируется с системой. Внешне это отражается появлением кнопки **New Simulink Model** в панели инструментов (перед кнопкой **?**). При нажатии кнопки открывается окно редактора функциональных блок-схем и оглавления библиотеки компонентов (рис. 10.1).

Нетрудно заметить, что пользовательский интерфейс окна командного режима работы системы MATLAB и окон пакета Simulink выполнен в общем стиле, характерном для Windows-приложений. Это позволяет отказаться от детального описания его особенностей, сделанного в главе 2 для системы MATLAB.

Интеграция пакета Simulink с системой MATLAB имеет глубокий смысл. Решение большинства задач моделирования базируется на автоматическом составлении сложных конечно-разностных систем для линейных, нелинейных и дифференциальных уравнений, именуемых уравнениями состояния модели. Все это обеспечивает пакет Simulink. Наиболее эффективное решение таких систем уравнений достигается применением соответствующего аппарата матричных вычислений, который реализован в системе MATLAB. Вот почему сочетание Simulink с системой MATLAB оказалось столь плодотворным. К этому стоит добавить, что Simulink использует практически любые операторы и функции системы MATLAB (и язык программирования).

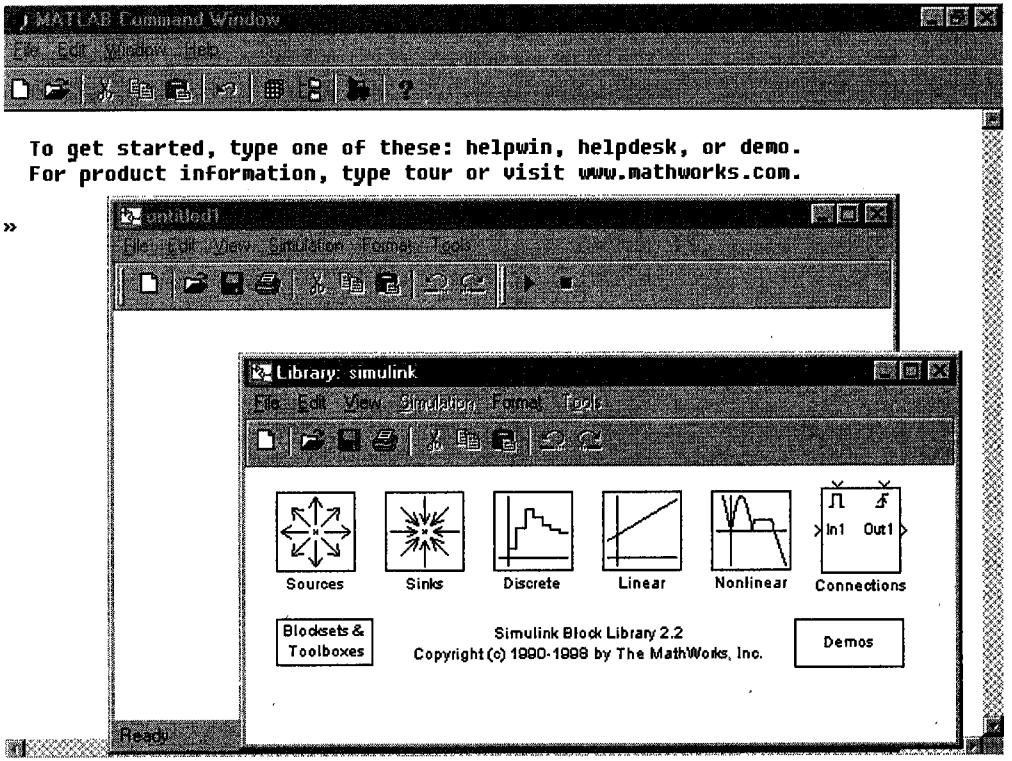


Рис. 10.1. Интеграция средств пакета Simulink с системой MATLAB

10.1.7. Решатель систем дифференциальных уравнений

Для решения автоматически составленной системы нелинейных дифференциальных уравнений первого порядка (ODE) Simulink использует решатель дифференциальных уравнений, построенный в виде программного цифрового интегратора. Решатель работает в двух основных режимах:

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.

Параметры решателя устанавливаются с помощью окна, которое появляется при исполнении команды **Parameters** в позиции **Simulation** главного меню пакета Simulink. В нем же можно установить конкретный метод решения дифференциальных уравнений: **ode45**, **ode23**, **rk45** (метод Рунге-Кутта), **ode113** (метод Адамса), **ode15s** и **ode1** (метод Эйлера). Эти методы мы уже демонстрировали при описании решения дифференциальных уравнений в среде MATLAB.

10.1.8. Понятие об S-функциях

Своими возможностями пакет Simulink во многом обязан специальному аппарату создания и применения так называемых системных **S-функций** (System Functions). Эти функции позволяют в ходе решения осуществлять сложные функциональные преобразования по различным математическим алгоритмам, например, алгоритмам решения систем дифференциальных уравнений.

Для разработки S-функций Simulink имеет специальный редактор. Создав S-функцию, пользователь фактически создает блок библиотеки, который может использоваться по всем правилам применения блоков. Блок можно переносить в окно редактирования с помощью мыши, менять заданную S-функцию, использовать необходимые связи между блоками и так далее.

Подробное описание аппарата задания и применения S-функций выходит за рамки этой небольшой обзорной главы. Этому описанию посвящена значительная часть справочника по системе Simulink, поставляемого вместе с данной системой. В PDF-формате справочник занимает около 4 Мбайт. Но это лишь небольшая часть описания библиотек Simulink. Для создания S-функций используется команда sfuntmpl MATLAB. Пример использования S-функций приводится в дальнейшем описании.

10.1.9. Особенности интерфейса Simulink

Интерфейс версии Simulink 2.0 полностью соответствует стилю интерфейса типовых приложений Windows 95/98, в том числе интерфейсу системы MATLAB. В то же время он концептуально строг, дабы не досаждать пользователю многочисленными "излишками" стандартного интерфейса Windows 95/98. Главное меню системы имеет следующие позиции:

File — работа с файлами моделей и библиотек (их создание, сохранение, считывание и печать).

Edit — операции редактирования, работа с буфером промежуточного хранения и создание субсистем.

View — вывод или удаление панелей инструментов и статуса.

Simulation — управление процессом моделирования (старт, ввод паузы и вывод окна настройки параметров симуляции).

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 позиции главного меню содержат общепринятые для Windows-приложений команды и операции, так что мы даже не будем их здесь обсуждать. Остальные позиции 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 дает доступ к множеству других подобных окон, открывающих доступ к новым пакетам компонентов (Blocksets&Toolboxes) и примеров их применения (Demos). Это дает возможность пользователю постепенно знакомиться с новыми областями применения Simulink.

10.1.10. Демонстрация возможностей Simulink

Даже незнакомый с Simulink пользователь может быстро оценить возможности этого пакета, воспользовавшись множеством интересных и поучительных примеров его применения. Они находятся в директории MATLAB/TOOLBOX/SIMULINK/SIMDEMO10. Попутно отметим, что в директории MATLAB/SIMULINK располагаются служебные файлы.

Для загрузки примеров используется (как обычно) команда **Open** в позиции **File** главного меню основного окна пакета Simulink. В уже знакомом вам окне с названием Open Model (Открыть Модель) надо войти в указанную директорию и выбрать подходящий пример (см. рис. 10.2).

Выбрав нужный пример, надо нажать кнопку **Открыть**. Появится графическая модель моделируемого устройства. На рис. 10.3 показана выбранная нами модель, позволяющая проанализировать поведение массивного кубика, который через пружину привязан к палочке (файл и пример называются **onecart**). Кубик перемещается по плоскости с трением, поэтому при возбуждении системы с помощью палочки можно наблюдать характерные затухающие колебательные движения кубика. Таким образом, в этом примере решается типичная физическая задача на колебания реального механического маятника под действием внешней силы.

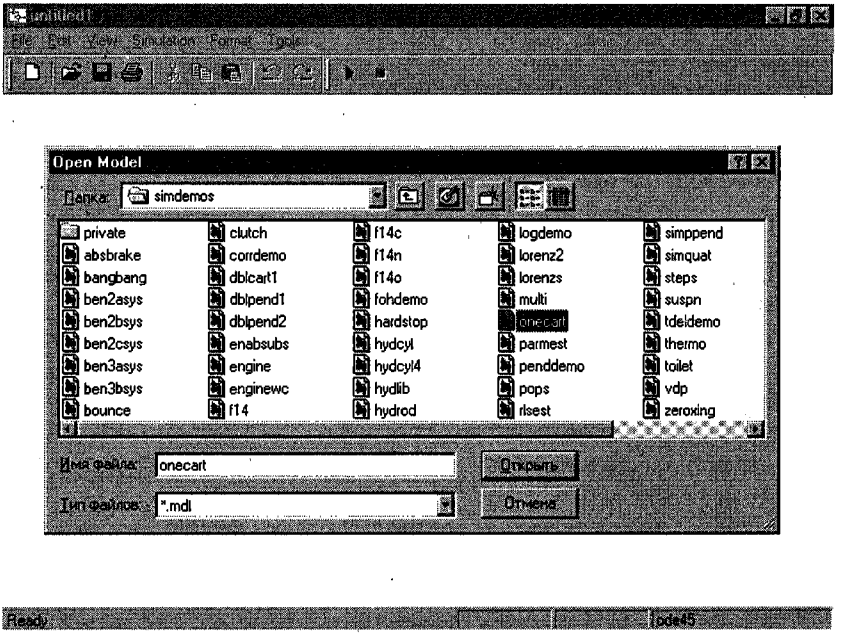


Рис. 10.2. Окно с полным перечнем файлов демонстрационных примеров пакета Simulink

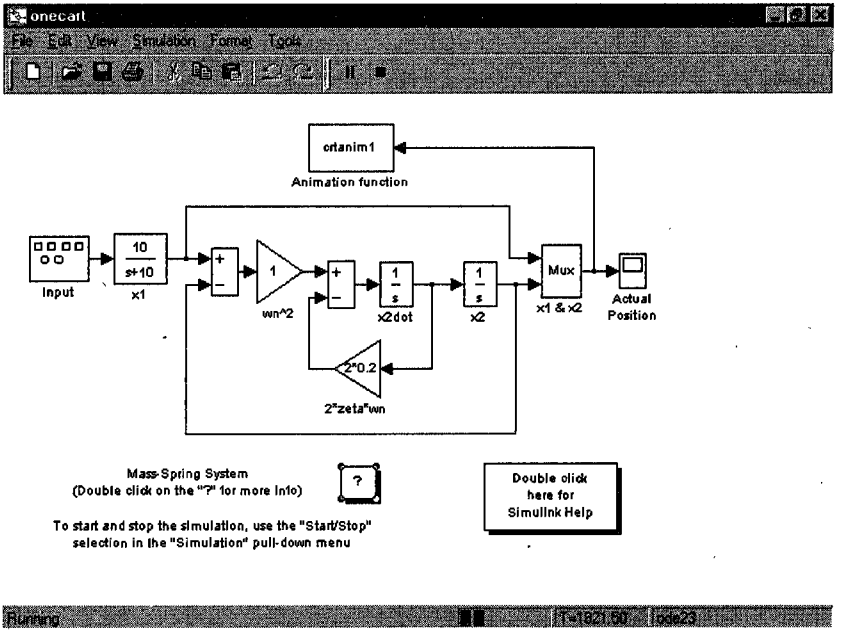


Рис. 10.3. Графическая модель колебательной системы — массивного кубика на пружине

Как можно заметить, графическая модель этого примера содержит 11 блоков. Каждый блок имеет наглядное общепринятое обозначение в виде прямоугольника, треугольника и так далее. Блоки имеют входы и выходы и описываются различными математическими зависимостями. Блоки соединяются друг с другом линиями — стрелками, причем стрелка указывает направление от выходов одних блоков ко входам других. Имеются также текстовые комментарии и средства для вывода подсказок и открытия окон системы помощи.

В конце инструментальной панели Simulink находятся две важные кнопки управления симулятором. 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 надо для симуляции выбранной модели, — это нажать кнопку с изображением треугольника. Рис. 10.4 показывает результат моделирования для выбранной модели. Вместо кнопок можно использовать команды **Start** и **Pause** в позиции **Simulation** главного меню симулятора.

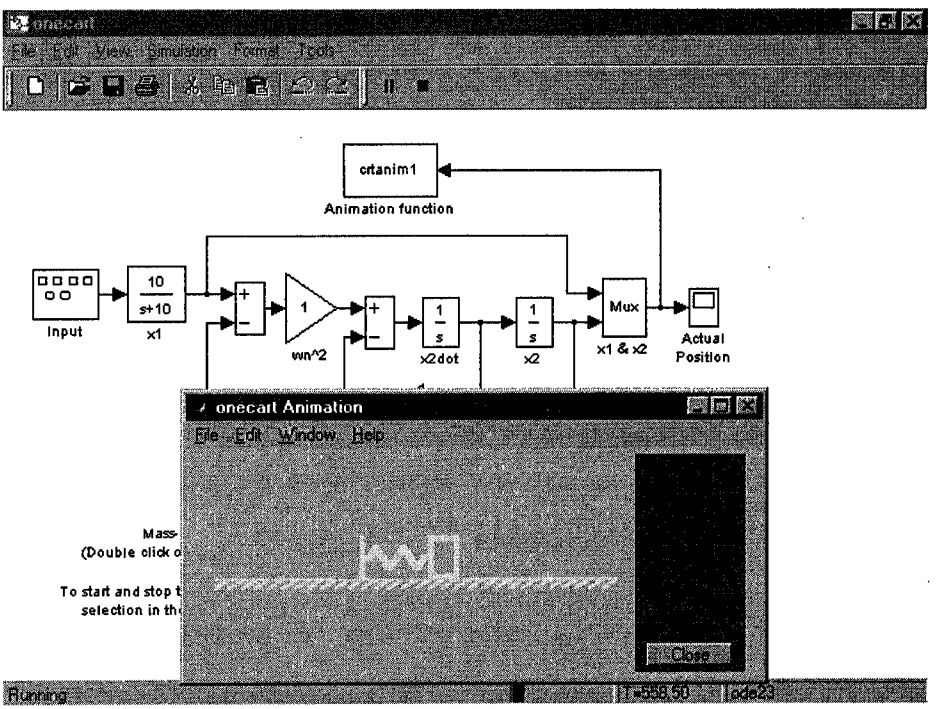


Рис. 10.4. Кадр симуляции модели поведения массивного кубика, через пружину привязанного к палочке

В данном случае результат моделирования отображается в виде анимационного видеоклипа (см. изображение физической модели кубика в окне анимации под графической моделью анализируемого физического устройства). Наглядность представления результатов поведения устройства в данном случае вполне очевидна.

10.1.11. Запуск моделей Simulink из среды MATLAB

Обычно Simulink запускается соответствующей кнопкой из панели инструментов, после чего все последующие действия выполняются в среде Windows. Для вывода полного перечня команд Simulink надо выполнить команду

```
» help simulink
```

Вывод будет дан в следующей форме (дано несколько первых команд):

```
Simulink
Version 2.2.1  24-Mar-98
Model analysis and construction function10.
Simulation.
  sim          — Simulate a Simulink model.
  sldebug      — Debug a Simulink model.
  simset       — Define options to SIM Options structure.
  simget       — Get SIM Options structure
Linearization and trimmin10.
  linmod       — Extract linear model from continuous-time system.
  linmod2      — Extract linear model, advanced method.
  dlinmod      — Extract linear model from discrete-time system.
  trim         — Find steady-state operating point.
Model Construction.
  close_system  — Close open model or block.
  new_system    — Create new empty model window.
  open_system   — Open existing model or block.
  save_system   — Save an open model.
.....
```

Дополнительную информацию можно получить, используя команды

help blocks и **help simdemos**

Мы не будем рассматривать эти многочисленные команды, поскольку удобный и наглядный интерфейс Windows для пакета Simulink — это лучший способ эффективной работы.

10.2. Библиотека компонентов пакета Simulink

10.2.1. Основная палитра компонентов

Версия симулятора Simulink 2.0 имеет существенно обновленную библиотеку компонентов. Она размещается в директории MATLAB/TOOLBOX/SIMULINK/

BLOCKS. Основная палитра компонентов представлена файлом simulink.mdl. Как основная, так и дополнительные библиотеки Simulink представлены файлами разного формата — с расширением dll, в виде mex-файлов и файлов с расширением .m. Последние могут при необходимости редактироваться и модифицироваться опытными пользователями.

Окно основной палитры компонентов пакета Simulink появляется при запуске пакета (рис. 10.1). На рис. 10.5 представлено отдельно содержимое этого окна. Каждый рисунок в данном случае носит обобщающий характер и представляет группу компонентов определенного класса. Можно считать, что эта группа рисунков представляет собой образное оглавление стандартной библиотеки графических элементов для набора функциональных схем.

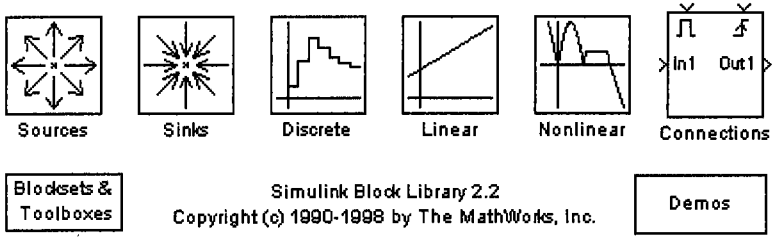


Рис. 10.5. Оглавление библиотеки компонентов (блоков) системы Simulink

Как видно из рис. 10.5, в состав библиотеки графических элементов входят следующие их наборы:

Sources — открытие окна с перечнем источников сигналов и воздействий.
Sinks — открытие окна с перечнем регистрирующих компонентов.
Discrete — открытие окна с перечнем дискретных компонентов.
Linear — открытие окна с перечнем линейных компонентов.
Nonlinear — открытие окна с перечнем нелинейных компонентов.
Connections — открытие окна с перечнем подключающих компонентов.
Blocksets&Toolboxes — открытие окна с перечнем дополнительных библиотек и примеров.
Demos — открытие окна MATLAB с демонстрационными примерами пакета Simulink.

Окно библиотек является обычным окном Simulink и имеет соответствующее главное меню, панель инструментов и панель статуса. С его помощью можно вводить и загружать из файла модели симулируемых устройств и систем. Окно можно свернуть или закрыть. Если при закрытии окна основной библиотеки оно понадобилось вновь, то из основного окна библиотеки надо загрузить файл simulink, который находится в директории MATLAB/TOOLBOX/ SIMULINK/BLOCK10.

10.2.2. Источники сигналов и воздействий

Строго говоря, окно Sources содержит графические элементы — источники воздействий. В электро- и радиотехнике их принято называть источниками сигналов, но в механике и в других областях науки и техники такое название не очень подходит – природа воздействий может быть самой разнообразной, например, в виде перепада давления или температуры, механического перемещения, звуковой волны и так далее. Окно Sources представлено на рис. 10.6.

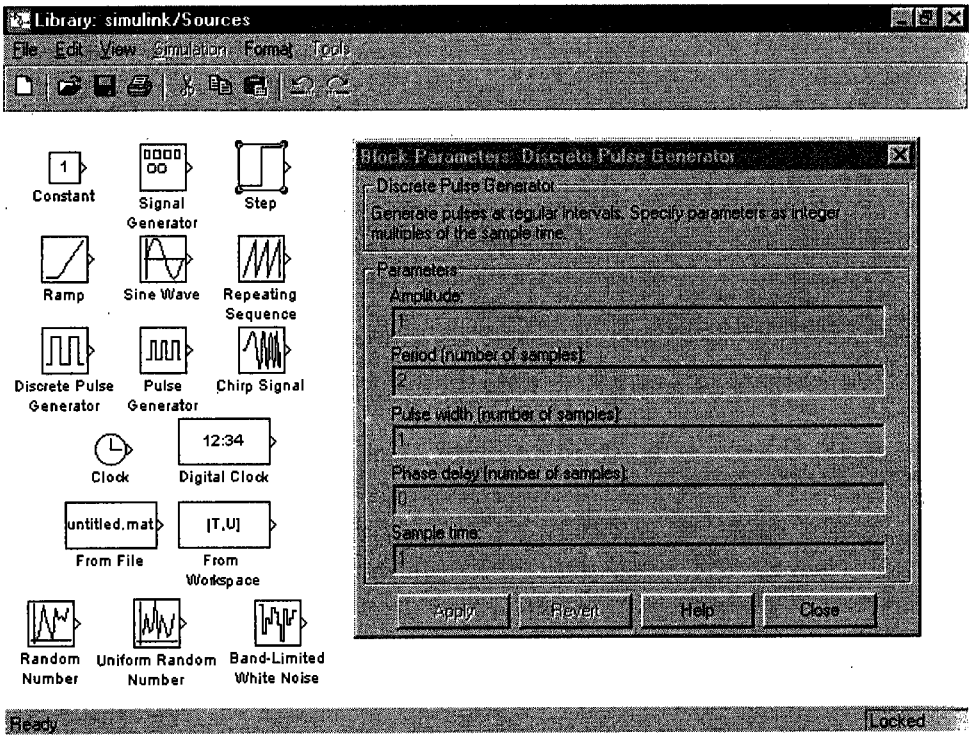


Рис. 10.6. Окно с перечнем источников сигналов и воздействий

Графические элементы источников воздействий (сигналов) имеют настолько очевидные обозначения, что не имеет смысла приводить их названия (значительная часть фирменного электронного справочника по Simulink, тем не менее, посвящена подробному описанию компонентов и их параметрам). Большинство элементов содержат рисунок, представляющий временную зависимость воздействий, например, перепад для блока Step, синусоиду для блока Sine Wave и так далее.

Набор блоков содержит практически все, часто используемые при моделировании источники воздействий с самой различной функциональной и временной зависимостью. Возможно задание произвольного воздействия из файла — элемент From File.

Имеются и случайные воздействия для моделирования систем и устройств методом Монте-Карло.

С каждым графическим элементом связана панель настроек. Так, для источника воздействия Step соответствующее окно представлено на рис. 10.6 справа от окна выбора источников. Для открытия этого окна достаточно выполнить двойной щелчок левой клавишей мышки при курсоре, установленном на нужное изображение элемента.

Естественно, что таких окон множество, как и самих графических элементов. Тем не менее для пользователей, имеющих даже начальные представления об имитационном моделировании систем, установка параметров графических элементов не вызывает трудностей, поскольку названия большинства параметров вполне очевидны. К примеру, для блока Step устанавливается амплитуда прямоугольного импульса или перепада, число периодов, длительность (ширина) импульса, время задержки и общая длительность периода сигнала. Словом, задаются типичные параметры прямоугольного импульса или перепада.

При вызове окон параметров активизацией графических элементах в окнах библиотек отображаются установки параметров по умолчанию. Как правило, они нормализованы — например, задана единичная частота, единичная амплитуда, нулевая фаза и так далее. Возможность изменения параметров в этом случае отсутствует. Она появляется после переноса графических элементов в окно подготовки и редактирования функциональных схем. Как правило, установки параметров блоков по умолчанию позволяют уверенно начать моделирование и затем уточнить эти параметры.

10.2.3. Регистрирующие элементы

Регистрирующие элементы — важный фактор качественной визуализации результатов симуляции. На рис. 10.7 показано окно виртуальных регистраторов пакета Simulink. Некоторые регистраторы выполнены в виде, весьма похожем на реальные приборы.

В состав виртуальных регистраторов входят:

Scope — осциллоскоп для наблюдения временных и иных зависимостей.
XY Graph — графопостроитель в системе полярных координат.
Display — устройство вывода на экран дисплея.
To file — устройство записи данных в файл.
To Workspace — устройство обзора рабочего пространства.
Stop Simulation — остановка симуляции.

Важно отметить, что виртуальные регистраторы фиксируют параметры любого типа, а не только электрические. Это придает некоторым виртуальным регистраторам (приборам) уникальный характер. Например, об осциллоскопе, фиксирующем не только электрические сигналы, но и перемещения механических объектов, изменения температуры или давления и вообще изменения любых физических величин, даже крупные физические лаборатории могут только мечтать.

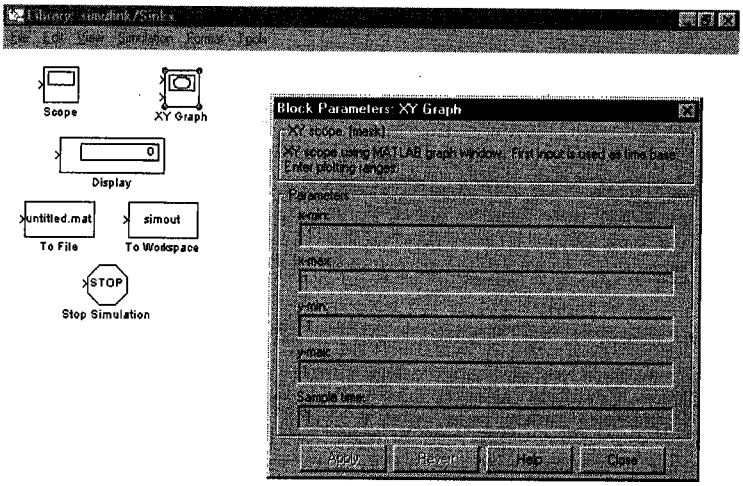


Рис. 10.7. Окно регистраторов пакета Simulink и окно настройки осциллоскопа

10.2.4. Дискретные компоненты

Дискретные компоненты включают в себя устройства задержки, дискретно-временной интегратор, дискретный фильтр и иные. На рис. 10.8 представлено окно с этими устройствами.

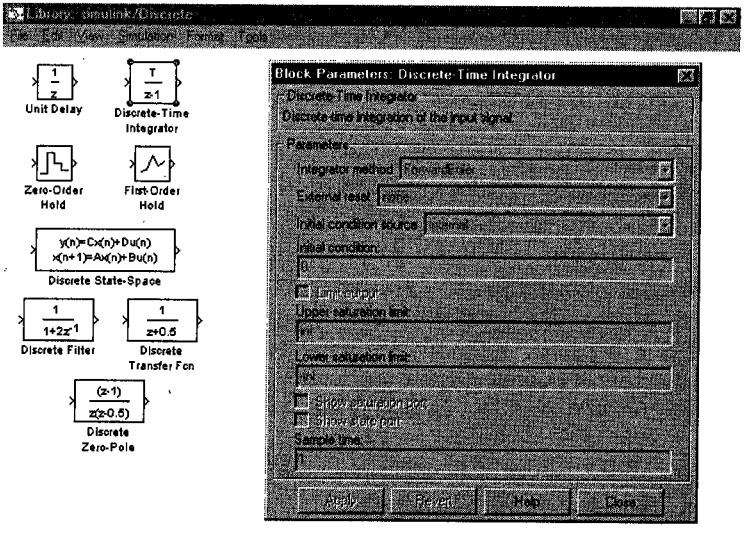


Рис. 10.8. Окно дискретных компонентов

На рис. 10.8 показано также окно установки параметров дискретно-временного интегратора. В нем отображается метод интегрирования, начальные условия интегрирования, пределы изменения выхода и иные менее важные параметры.

10.2.5. Линейные компоненты

Линейные компоненты играют важную роль в создании математических моделей многих устройств. Достаточно отметить электрические фильтры, построенные на таких компонентах (например, усилителях) и широко используемые в технике электро- и радиосвязи. На рис. 10.9 представлено окно с линейными компонентами пакета Simulink.

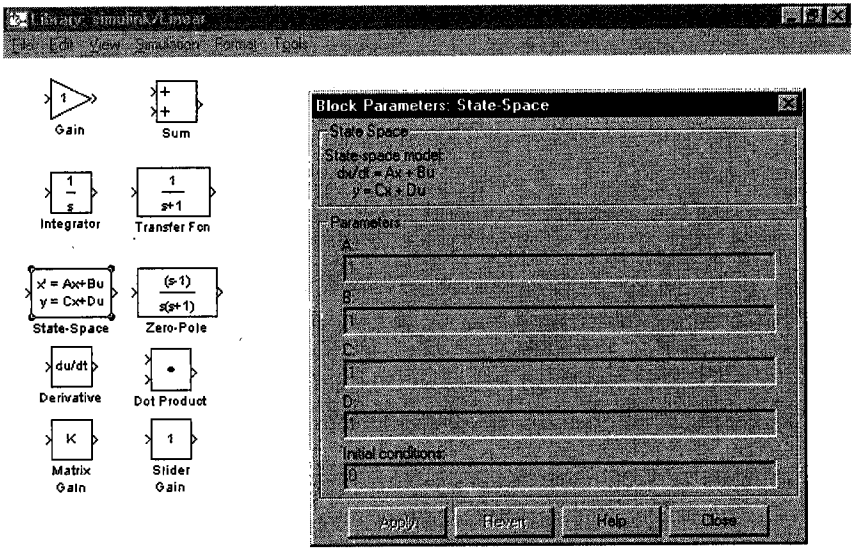


Рис. 10.9. Окно с линейными компонентами

Как видно из рис. 10.9, имеются следующие типы линейных компонентов: Gain — аналоговый усилитель (масштабирующее устройство), Sum — аналоговый сумматор, Integrator — аналоговый интегратор, Derivative — аналоговое дифференцирующее устройство и ряд других (в основном матричных) устройств. В правой части рис. 10.9 показано окно настройки блока задания компонента, работа которого описывается системой дифференциальных уравнений вида Коши.

10.2.6. Нелинейные компоненты

Поскольку Simulink предназначен главным образом для моделирования нелинейных динамических систем, раздел, посвященный нелинейным компонентам, впечатля-

ет уже их обилием (см. рис. 10.10). Окно с этими элементами настолько велико, что даже не поместилось на рисунке.

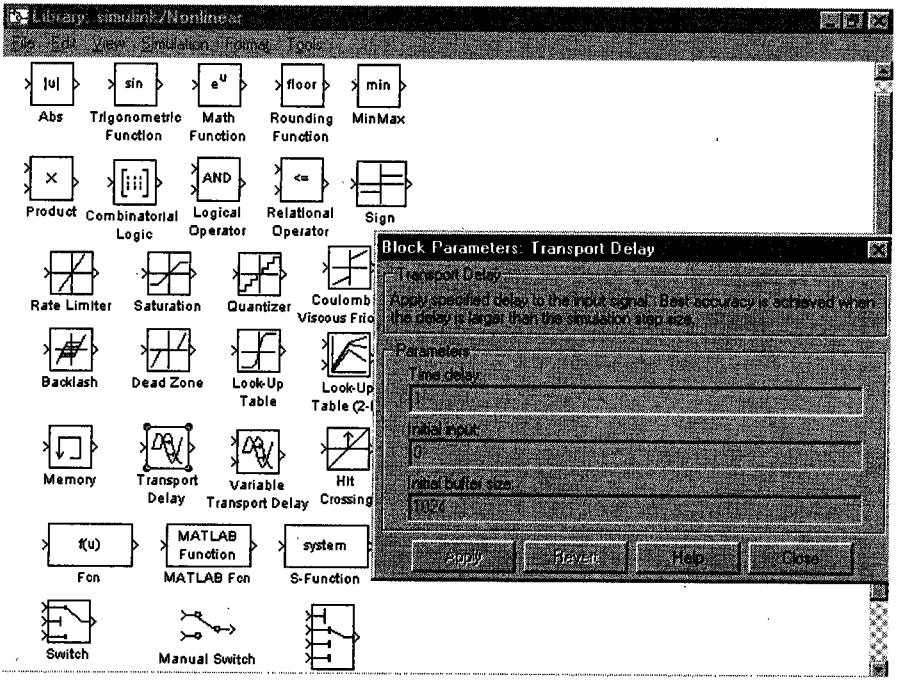


Рис. 10.10. Окно библиотеки с нелинейными компонентами

Среди нелинейных компонентов следует отметить компоненты с типичными нелинейностями, например, вида **abs(u)**, с характеристиками, описанными типовыми математическими функциями, компонентами вида идеальных и неидеальных ограничителей и так далее. Достойно представлены и такие сложные компоненты, как квантователи, блоки нелинейности, моделирующие нелинейные петли гистерезиса, временные задержки и ключи-переключатели.

Естественно, что все блоки нелинейности имеют установку своих параметров. Так, в правой части рис. 10.10 представлено окно настройки параметров устройства временной задержки.

10.2.7. Подключающие компоненты

Окно с подключающими компонентами показано на рис. 10.11. Оно также содержит впечатляющий выбор таких компонентов — от портов входа In, выхода Out и заземления Ground до компонент, имитирующих работу триггера Trig и даже задания субсистем Subsystem. Последняя компонента представляет собой пустое окно, в котором можно создать функциональную схему, рассматриваемую как подсистему (блок). Такая подсистема может многократно использоваться различными моделями.

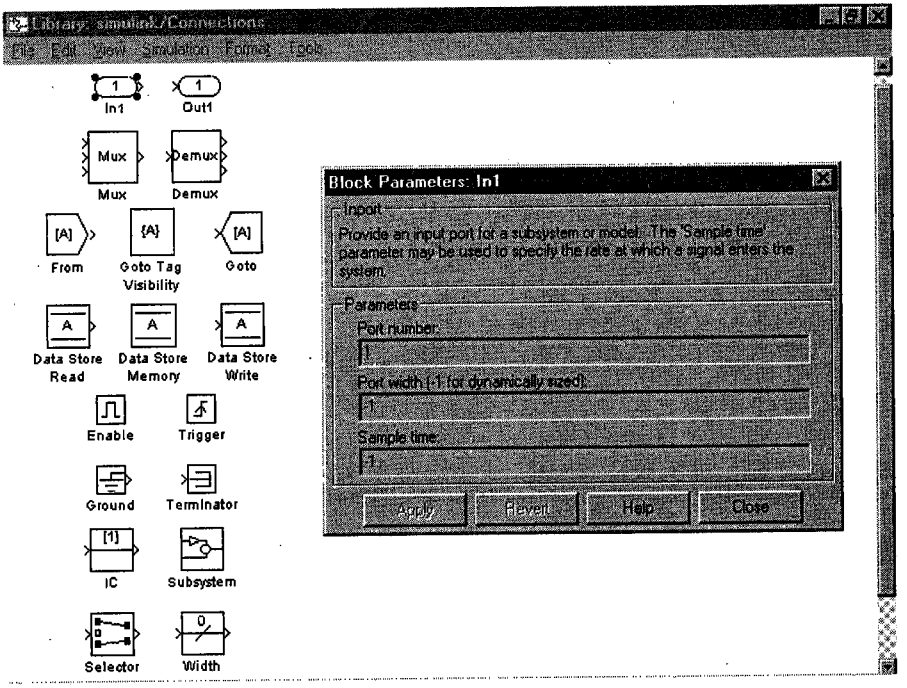


Рис. 10.11. Окно библиотеки с подключающими компонентами

Каждый компонент как и ранее имеет окно установки своих параметров. На рис. 10.11 представлено такое окно для порта ввода.

10.2.8. Внешние библиотеки и готовые решения

Если перечисленных выше компонентов вам показалось мало, то учтите, что раздел **Blocksets&Toolboxes** библиотеки открывает окно с перечнем дополнительных библиотек — рис. 10.11. Каждый ее подраздел (соответствующий рисунок) открывает дополнительное окно с новым перечнем компонентов. Для примера на рис. 10.12 открыто окно подраздела DSPLib.

Представленные в этих разделах компоненты относятся к достаточно серьезным приложениям системы MATLAB, причем подчас вполне законченным. В качестве иллюстрации на рис. 10.13 показан последовательный поиск примера решения типовой задачи спектрального анализа — выделение сигнала из его смеси с шумом. Задача решается сразу тремя методами, и можно легко провести их сравнение. Результаты решения представлены в виде спектрограмм — по крайней мере, одна из трех спектрограмм видна в правом верхнем углу рисунка. В правой части рисунка показана иерархия окон, ведущих к этому примеру.

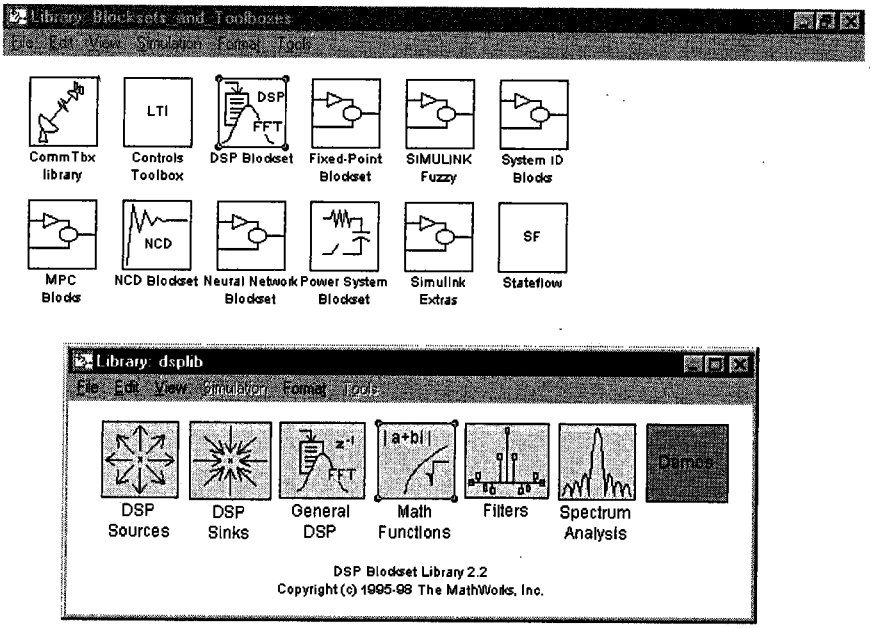


Рис. 10.12. Окно дополнительных подразделов библиотеки компонентов

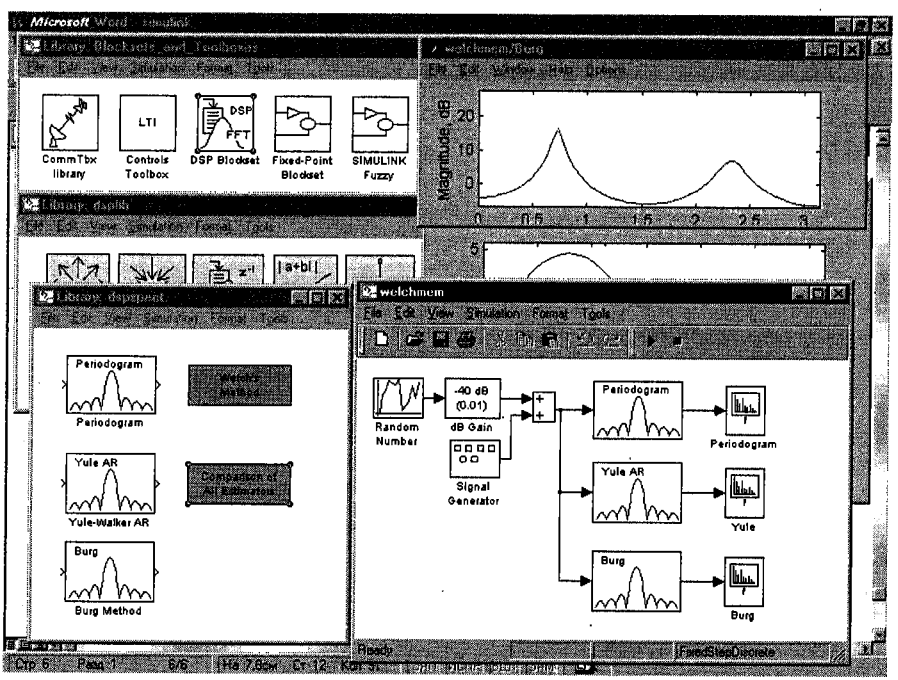


Рис. 10.13. Поиск и исполнение примера спектрального анализа

Ряд примеров на применение пакета Simulink представлен и в справочной системе MATLAB — Demo10. Выход в нее возможен из окон прочих компонентов, в которых также есть разделы с именем Demo10.

В директории MATLAB/TOOLBOX/SIMULINK/BLOCKS помимо файла основной библиотеки находится ряд файлов дополнительных библиотек. Ввиду ограниченного объема книги мы не будем обсуждать полный состав дополнительных библиотек и отметим только их названия:

mixed — преобразования вида 1/s и 1/z.
simo — решение системы уравнений x'=Ax+Bu, y=Cx+Du.
simosys — задание блока S-функции и контроль за его сигналом.
simulin — дополнительная библиотека (новые устройства регистрации, устройства дискретизации, линейные устройства, преобразователи координат и триггеры).

В разделе MATLAB/TOOLBOX/SIMULINK/DEE находится редактор и решатель систем дифференциальных уравнений с примерами его применения. Он будет описан несколько позже.

Таким образом, в поставку Simulink 2 входит множество библиотечных компонентов, с лихвой удовлетворяющих большинство пользователей, всерьез занимающихся математическим моделированием систем и устройств. При этом есть возможность корректировать описания компонент и вводить новые компоненты — в том числе в виде функциональных схем отдельных подсистем. В последней версии Simulink 3, включенной в систему MATLAB 5.3, набор компонентов существенно расширен и уменьшено время моделирования систем со сложными моделями. Однако весь приведенный ниже материал полностью справедлив и для этой новейшей версии пакета Simulimk.

10.3. Пример подготовки и решения конкретной задачи

10.3.1. Постановка задачи — моделирование ограничителя

Решение любой задачи в системе Simulink должно начинаться с постановки задачи. Чем глубже продумана постановка задачи, тем больше вероятность успешного ее решения. В ходе постановки задачи нужно оценить, насколько решение задачи отвечает возможностям пакета Simulink и какие компоненты последнего могут использоваться для решения задачи.

В качестве примера рассмотрим тривиальную задачу на моделирование работы идеального ограничителя, на вход которого подается синусоидальное напряжение с амплитудой 10 В и частотой 100 Гц. Допустим, что пороги ограничения составят +5 и -5 В.

В данном случае очевидно, что основными блоками будут генератор синусоидальных сигналов и нелинейность, моделирующая передаточную характеристику ограничителя. Кроме того, к этим блокам надо добавить регистрирующий блок — осциллоскоп. Так что функциональная схема моделируемого устройства в данном случае вполне очевидна и мы можем перейти к ее реализации.

10.3.2. Создание модели устройства (системы)

Создание модели в нашем примере начинается с выбора источника сигнала. Для этого надо удобно разместить окно с палитрой разделов основной библиотеки компонентов и активизировать рисунок с надписью Sources (Источники). Появятся окна источников воздействий, которые следует удобно расположить на экране. На рис. 10.14 эти окна показаны в правой части экрана, а левая часть освобождена для конструирования блок-схемы (модели) нашей задачи.

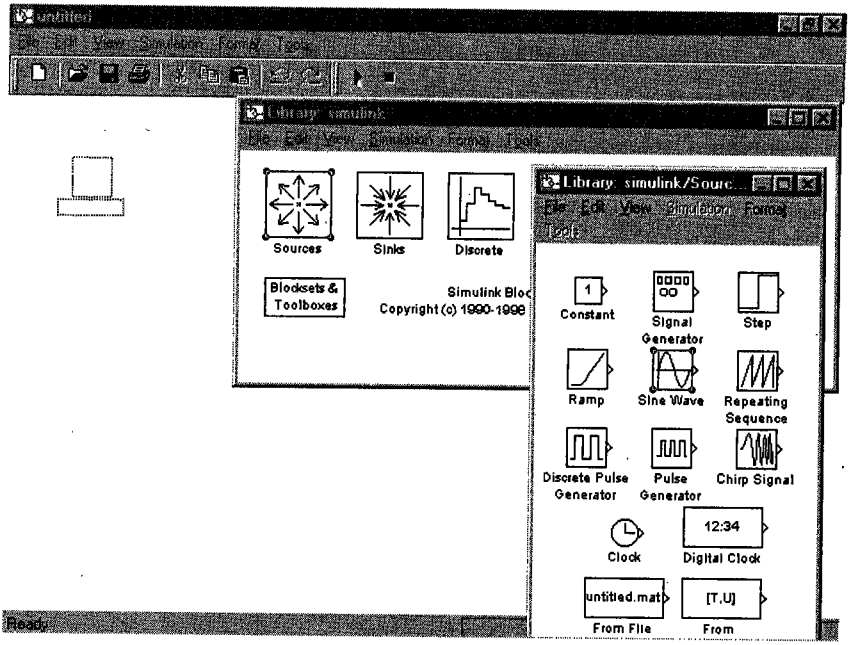


Рис. 10.14. Выбор источника и перенос его в окно редактирования

Теперь, зацепившись курсором за выбранный источник синусоидального сигнала, надо просто перетянуть его в нужное место окна редактирования. В процессе переноса правая клавиша должна быть нажата. При переносе объект отображается пунктирными линиями и сопровождает курсор. На рис. 10.14 этот объект виден в правом верхнем углу окна редактирования (курсор мыши, к сожалению, при снятии копий экрана обычно исчезает). Стоит отпустить левую клавишу мыши, как объект (источник синусоидального сигнала) появится на месте своего пунктирного собрата.

Полезно сразу установить нужные параметры источника синусоидального сигнала. Для этого сверните панели библиотек и дважды щелкните левой клавишей мыши при курсоре, наведенном на только что введенный блок. Появится окно установки параметров синусоидального сигнала — рис. 10.15. Надо установить заданную амплитуду в 10 В и частоту 100 рад./с; остальные параметры — фазу и время задержки — можно оставить нулевыми. После этого нажмите кнопки **Apply** (Применить) и **Close** (Закрыть) — заданные вами параметры будут сохранены.

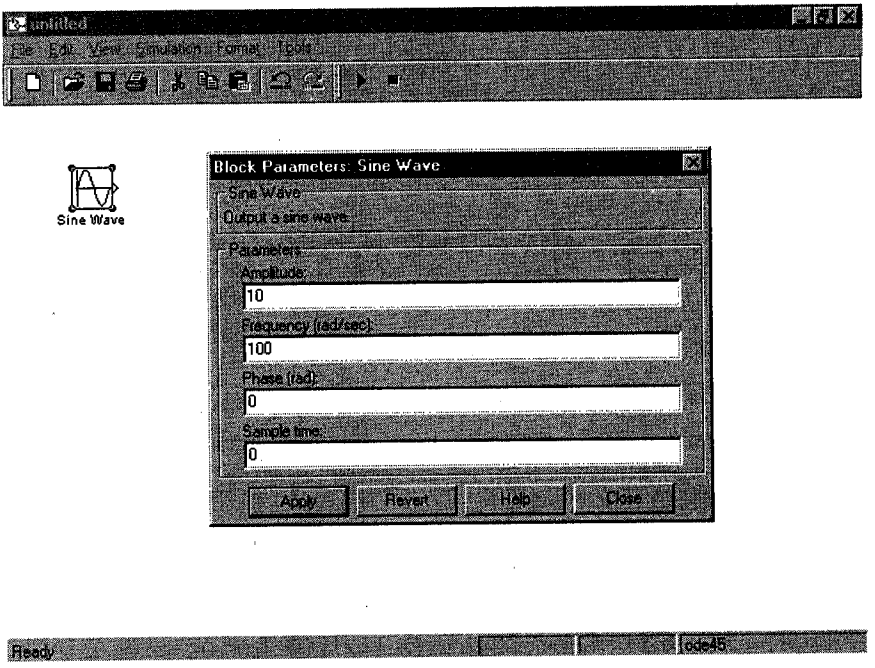


Рис. 10.15. Установка параметров входного сигнала

Теперь надо вновь вывести палитру разделов библиотеки (ее бирка находится в панели задач Windows 95/98) и выбрать раздел нелинейных элементов — Nonlinear. В появившемся окне этих элементов (см. рис. 10.16, на котором, кстати, виден введенный только что источник синусоидального сигнала) надо выбрать блок Saturation (Ограничение) и перетащить его мышкой в нужное место экрана редактирования — сзади источника синусоидального сигнала. Этот процесс наглядно показан на рис. 10.16.

Опять сверните окна библиотеки (при этом окно Nonlinear можно вообще закрыть) и соедините выход источника синусоидального сигнала со входом блока ограничения. Для этого установите курсор мыши на выход источника и, нажав и удерживая левую клавишу, добейтесь, чтобы он превратился в крестик из тонких линий. Это означает, что редактор блок-схем готов к проведению отрезка соединительной линии. Отрезок линии проводится при нажатой левой клавише мыши перемещением курсора до точки входа блока нелинейности. Отпустив левую клавишу, вы получите соединительную линию между введенными блоками (рис. 10.17).

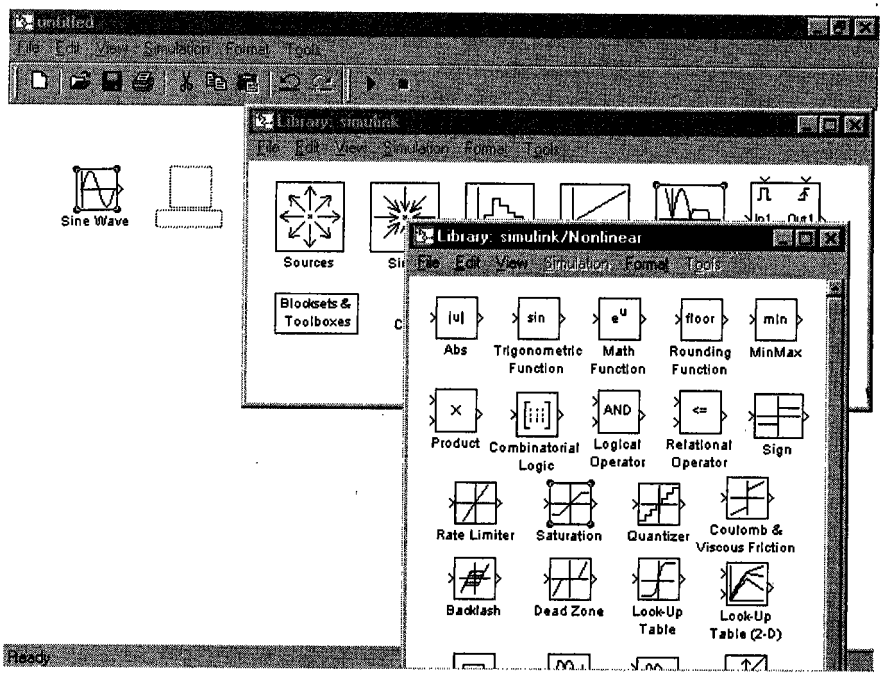


Рис. 10.16. Перенос в окно редактирования блока ограничения

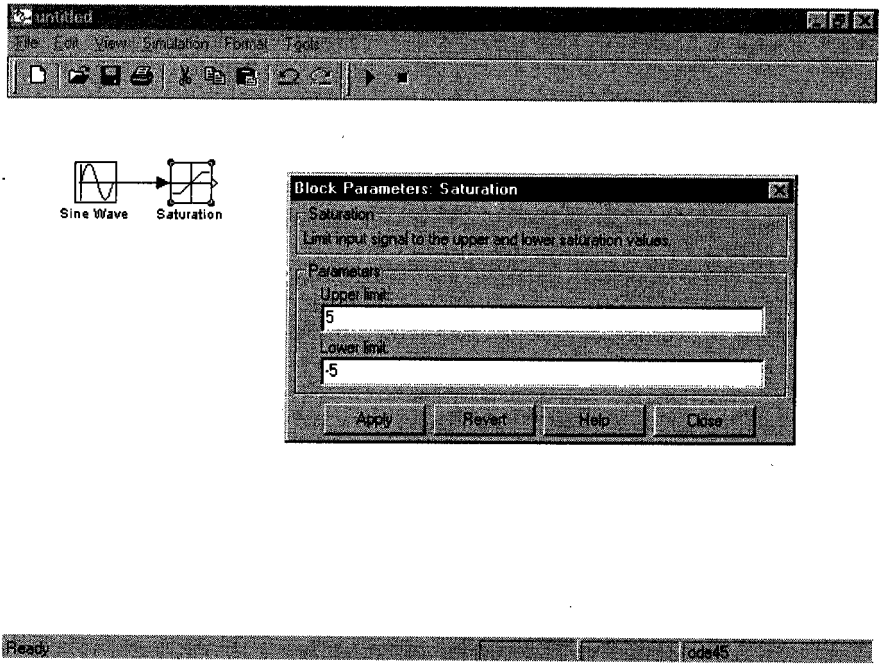


Рис. 10.17. Установка соединения между блоками и настройка блока ограничения.

Далее откройте окно установки параметров блока ограничения. Это окно также показано на рис. 10.17. В этом окне надо заметить установленные пороги ограничения на заданные условиями задачи, то есть -5 и +5 В. После чего нажмите кнопки **Apply** и **Close** для сохранения настройки и закрытия окна нелинейных элементов.

Далее подобным описанному образом введите последний блок-осциллоскоп. Процесс ввода в окно редактирования блока-осциллоскопа представлен на рис. 10.18.

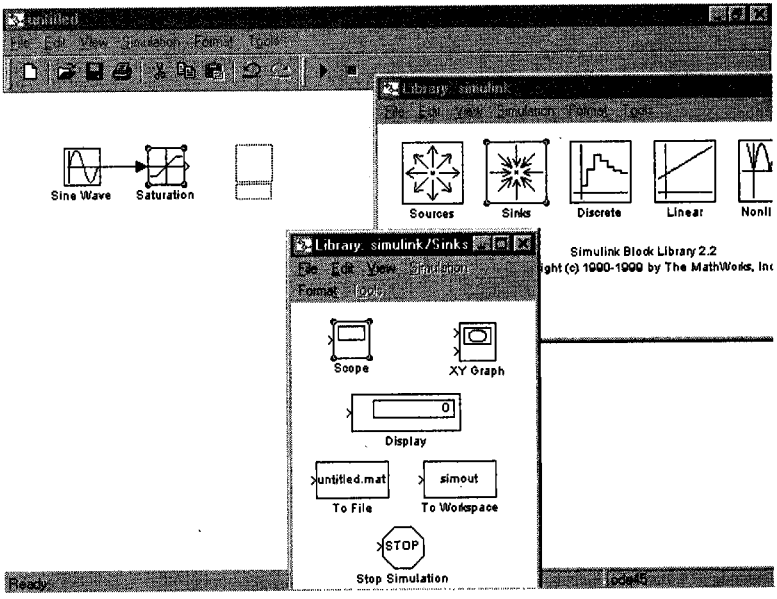


Рис. 10.18. Ввод в модель блока-осциллоскопа

После этого надо подсоединить выход блока ограничения к входу блока-осциллоскопа, как это было описано. На этом подготовка модели заданной задачи заканчивается. Впрочем, полезно выполнить еще один прием — разместить под моделью пояснительную надпись. Для этого достаточно установить маркер мыши на свободное поле редактирования и дважды щелкнуть левой клавишей. Должно появиться прямоугольное окошко с маркером ввода — мигающей вертикальной черточкой. Надпись вводится как обычно с применением средств строчного редактирования. По завершении ввода надписи надо установить маркер мыши вне поля надписи и щелкнуть левой клавишей мыши.

10.3.3. Запуск модели

Следующий этап моделирования — запуск модели. Можно выполнить его сразу, но скорее всего полученный результат окажется неудачным. Та же картина наблюдается на практике, когда к малоизвестному исследуемому устройству впервые подключается осциллограф. Нужна некоторая предварительная настройка осциллографа, в частности, выбор просматриваемого интервала времени, установка масштаба регист-

рируемой величины и так далее. По умолчанию интервал времени задан равным 1 с, что слишком много для выбранной частоты сигнала в 100 рад/с.

Для настройки запуска модели надо исполнить команду **Parametrs** в позиции главного меню **Simulation** пакета Simulink. В появившемся окне (см. рис. 10.19) в закладке Solver надо уточнить временной интервал моделирования, например, сделав его равным 0.2 с. Вы можете также выбрать метод изменения независимой переменной и метод решения дифференциальных уравнения при моделировании, а также погрешности вычислений. Как правило, однако, при этом вполне удовлетворительны установки этих параметров по умолчанию.

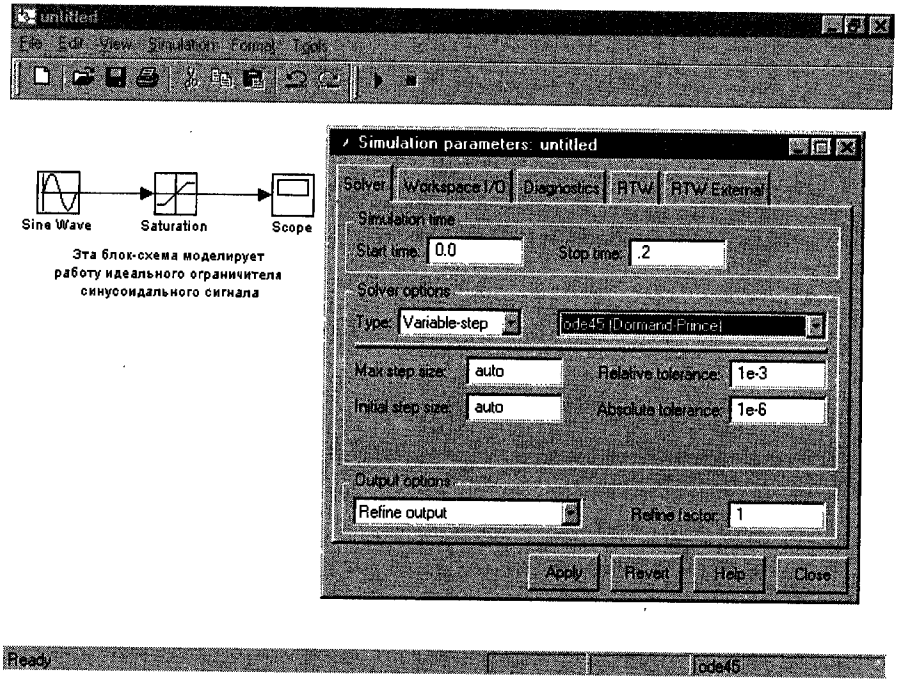


Рис. 10.19. Установка параметров запуска модели

Теперь можно запустить модель. Для этого надо нажать кнопку пуска (треугольник) на панели инструментов или исполнить команду **Start** в позиции **Simulation** главного меню. По завершении процесса моделирования (для данной схемы он занимает доли секунды) активизация объекта-осциллоскопа выводит окно, в котором виден результат симуляции. Это окно показано на рис. 10.20 и очень напоминает экран реального осциллографа.

Как и следовало ожидать, в результате симуляции получена синусоида с обрезанными на уровне 5 В вершинами. Столь быстрое получение явно верного результата достигается далеко не всегда. Чем сложнее модель, тем больше усилий придется затратить на то, чтобы добиться ее правильной "работы".

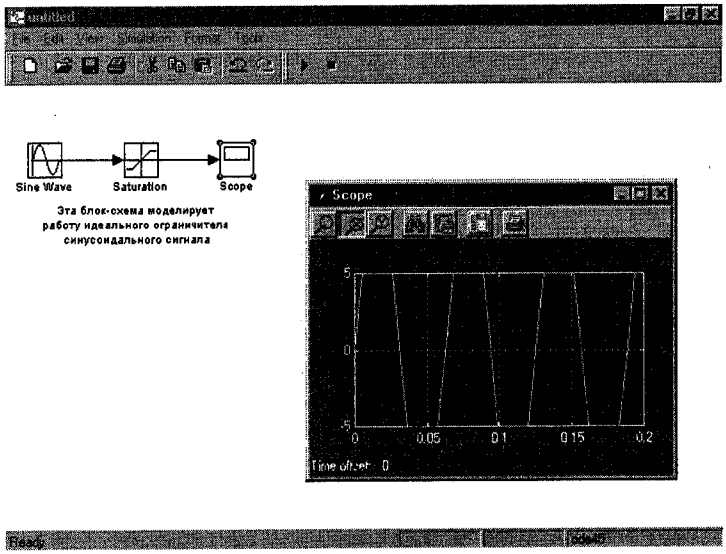


Рис. 10.20. Результат моделирования созданной модели

10.3.4. Модернизация и дополнение модели

Теперь, после создания простой модели, можно попытаться модернизировать ее и дополнить, изменить параметры модели и так далее. На рис. 10.21, к примеру, представлена модель, в которой к источнику синусоидального сигнала подключено уже три нелинейных устройства — ограничитель пиков, нелинейность типа "диодная зона" и квантующее устройство. К каждому из них подключен свой осциллоскоп.

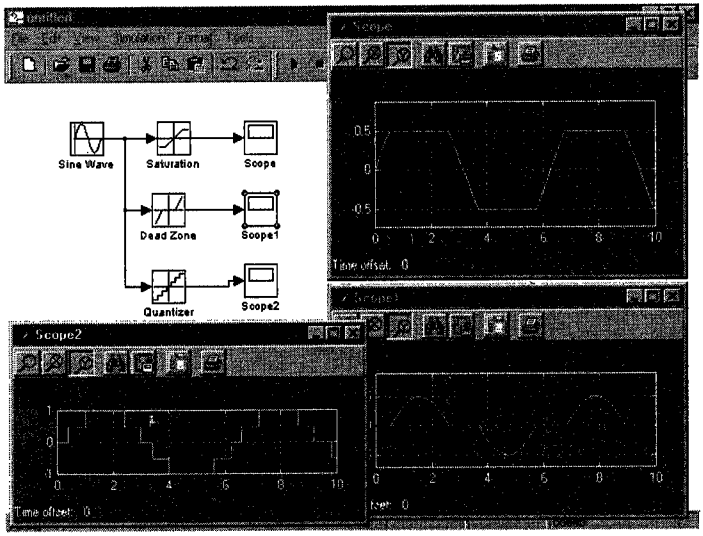


Рис. 10.21. Модель для трех нелинейных устройств

Как видно из рис. 10.21, теперь можно наблюдать сигналы с выхода всех трех нелинейных устройств. При этом выходные сигналы с каждого устройства наблюдаются с помощью своего осциллоскопа. Позже мы покажем, что возможно наблюдение и нескольких сигналов одним осциллоскопом, если перед ним установить микшер сигналов.

10.3.5. Некоторые приемы редактирования модели

В нашем последнем примере для построения новой модели нужно было убрать поясняющую надпись. Это легко сделать, активизировав надпись (кстати, как и любой другой объект) и используя команду **Clear** в позиции главного меню **Edit**. Очень удобно (особенно при стирании линий) пользоваться кнопкой в панели инструментов с изображением ножниц (**Cut**), которая помещает выделенный объект в буфер Clipboard Windows 95/98 и при этом удаляет его со своего места.

Выделение объектов удобнее всего осуществляется с помощью мыши. Достаточно установить курсор ее на нужном объекте и щелкнуть один раз ее левой клавишей. Объект будет выделен. Двойное нажатие этой клавиши обычно вызывает окно коррекции параметров объекта (блока). Мышкой также можно выделить несколько объектов. Для этого надо установить курсор вблизи от них и, нажав и удерживая левую клавишу мыши, начать перемещать ее по столу. Появится расширяющийся прямоугольник из пунктирных линий. Все попавшие в него объекты будут выделены и их можно будет перемещать в окне редактирования или стирать. Выделить разом все объекты можно используя команду **Select All** в позиции **Edit** главного меню.

Для стирания выделенного объекта можно использовать и команду **Delete** в контекстно-зависимом меню правой клавиши мыши — см. рис. 10.22. Контекстно-зависимое меню правой клавиши мыши очень удобно тем, что для любого объекта оно выводит перечень команд и операций, которые доступны для заданного контекста, иначе говоря состояния.

Как видно из рис. 10.21, для подключения новых блоков нужны новые соединения. Они также легко выполняются с помощью мыши. Вообще говоря, приемы ввода новых блоков и их соединений друг с другом выполняются очень просто и естественно, так что нет смысла останавливаться на этом подробно. Составьте две, три блок-схемы, наподобие показанной на рис. 10.1, и вы убедитесь в том, что блок-схема из десятка элементов может быть составлена в считанные минуты. При этом приемы редактирования очень напоминают работу с популярными графическими редакторами, которую особенно легко осваивают дети. Что, однако, стоит отметить, так это возможность задания наклонных линий соединений при нажатой клавише Shift.

Simulink имеет расширенные возможности редактирования блок-схем. Так, блоки в окне редактирования можно не только перемещать с помощью мыши, но и менять в размерах. Для этого блок выделяется, после чего курсор мыши надо установить на кружки по углам блока. Как только курсор превратится в двунаправленную диагональную стрелку, можно при нажатой левой клавише растягивать блоки по диагонали, увеличивая или уменьшая их размеры. Кроме того, блоки можно помещать в буфер

промежуточного хранения Clipboard операционной системы Windows 95/98 и использовать буфер для перенесения блоков из одного места в другое.

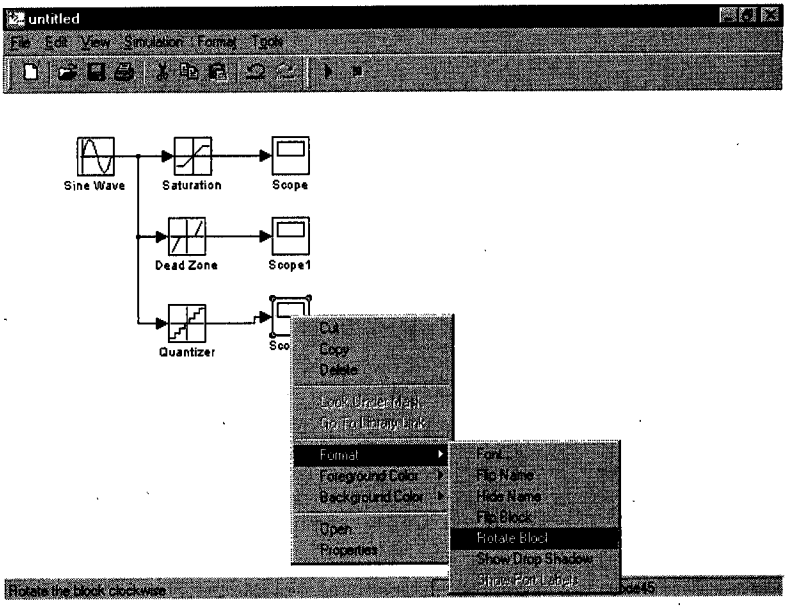


Рис. 10.22. Контекстно-зависимое меню правой клавиши мыши

В позиции **Format** главного меню (и в контекстно-зависимом меню правой клавиши мыши) можно найти ряд команд форматирования блоков: замены шрифта пояснительных надписей и их стиля (**Font**), смены надписей (**Flip name**), устранения надписей (**Show name**), поворота блоков вокруг вертикальной оси (**Flip Block**), поворота на 90 градусов (**Rotate Block**), включения и отключения тени (**Show Drop Shadow**). Там же есть очевидные опции по изменению цвета линий блока, закраске блоков и изменению цвета общего фона.

Большую помощь в редактировании оказывает команда **Undo** — отмена последней операции. Она поддерживает 101 операцию, включая операции добавления и стирания линий. Эту команду можно реализовать с помощью кнопки в панели инструментов или из позиции **Edit** главного меню.

10.4. Наглядные примеры применения пакета Simulink

10.4.1. Построение фигур Лиссажу

Вначале рассмотрим простой пример применения виртуального графопостроителя для построения фигур Лиссажу. Такие фигуры получаются, если на входы X и Y по-

дать синусоидальные сигналы строго кратных частот — при этом фигура получается неподвижной. Для построения таких фигур средствами Simulink достаточно построить модель, содержащую два генератора синусоидальных колебаний и XY-графопостроитель. На рис. 10.23 показана такая модель.

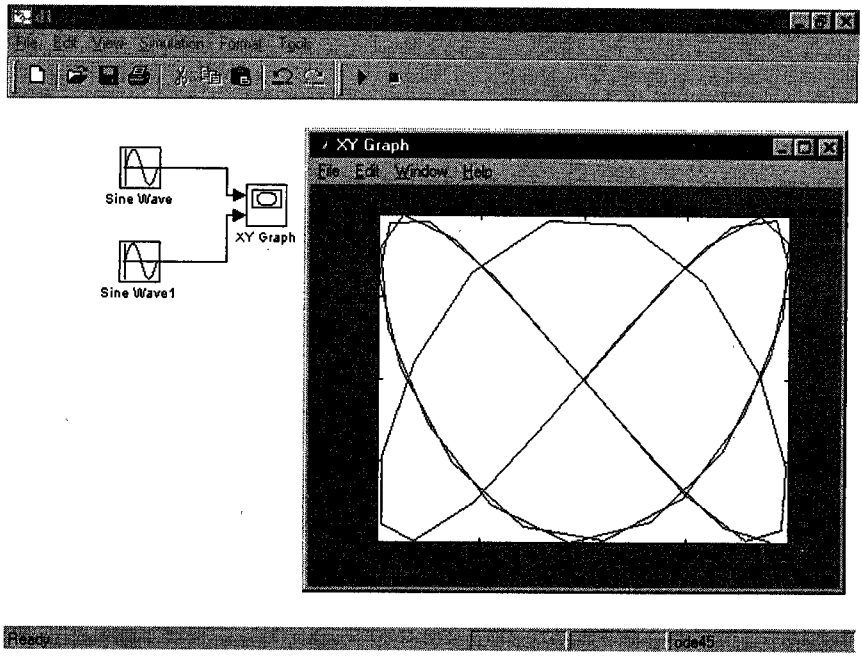


Рис. 10.23. Модель для построения фигур Лиссажу

На рис. 10.23 показан также результат симуляции для этой модели. Частоты генераторов были установлены равными 2 и 3 рад/с, то есть с кратностью 2/3. Она и определяет вид наблюдаемой замкнутой кривой. Нетрудно сообразить, что это перепевы старой песни — такие графики строго математически мы уже получали средствами системы MATLAB. Ими и пользуется при решении данной (да и многих иных задач) пакет Simulink.

10.4.2. Моделирование колебательной системы второго порядка

Системы второго порядка, описываемые дифференциальными уравнениями второго порядка, занимают важное место в таком фундаментальном разделе физики, как теория колебаний. Именно на базе таких систем созданы многочисленные генераторы периодических колебаний самого различного типа — от LC-генераторов до лазерных и иных квантово-механических генераторов.

Пример моделирования типовой системы второго порядка есть в пакете демонстрационных примеров применения пакета Simulink. Модель такой системы представле-

на на рис. 10.24. Эта система описывается хорошо известным нелинейным дифференциальным уравнением второго порядка Ван-дер-Поля, которое мы уже решали средствами системы MATLAB.

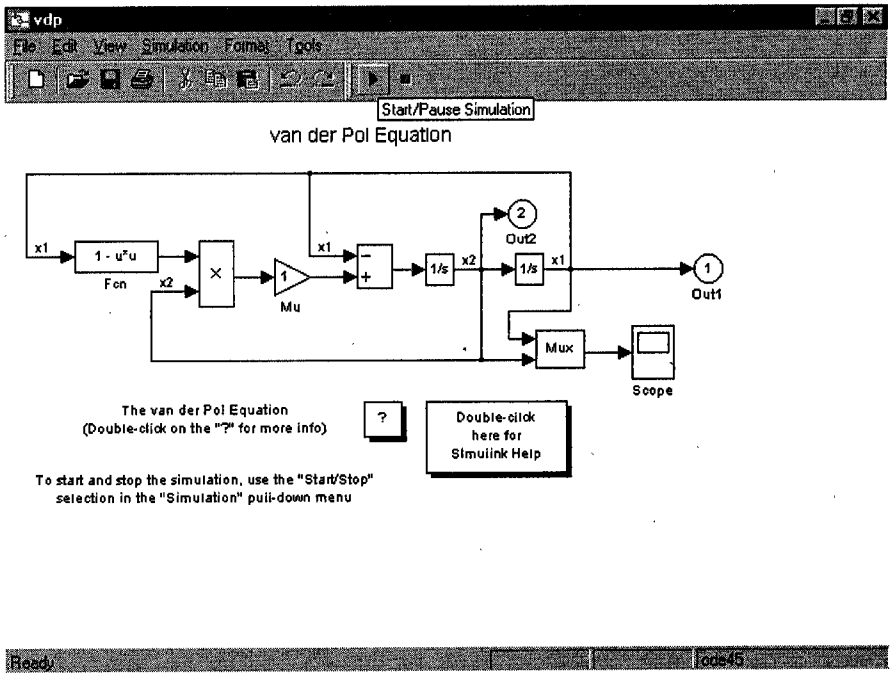


Рис. 10.24. Модель колебательной системы второго порядка

Как нетрудно заметить, данная модель представляет собой типичный усилитель с нелинейным элементом Fcn, позволяющим задать тип нелинейности с положительной обратной связью и имеющим в своем тракте блоки, ослабляющие как высокие, так и низкие частоты. Колебания в такой системе возникают на некоторой частоте, для которой фазовый сдвиг тракта равен нулю, а малосигнальный петлевой коэффициент передачи превышает 1. Характер развития колебательного процесса в решающей мере зависит от характера нелинейности, заданного в блоке Fn.

На рис. 10.25 показан результат моделирования данной модели. Нетрудно заметить, что в этом случае осциллоскоп отображает две кривые, соответствующие выходным портам Out 1 и Out 2. Для этого перед осциллоскопом размещен блок микшера Mux с двумя входами. Результат моделирования получен в виде временных зависимостей выходных сигналов.

Допустим нас интересует поведение системы для иной нелинейности, скажем, F(u)=1-exp(u). Для замены нелинейности достаточно установить курсор мыши на блок Fn и дважды быстро нажать левую клавишу. В появившемся окне параметров надо вместо функции по умолчанию ввести новую функцию, отражающую нелинейность модели, как показано на рис. 10.26.

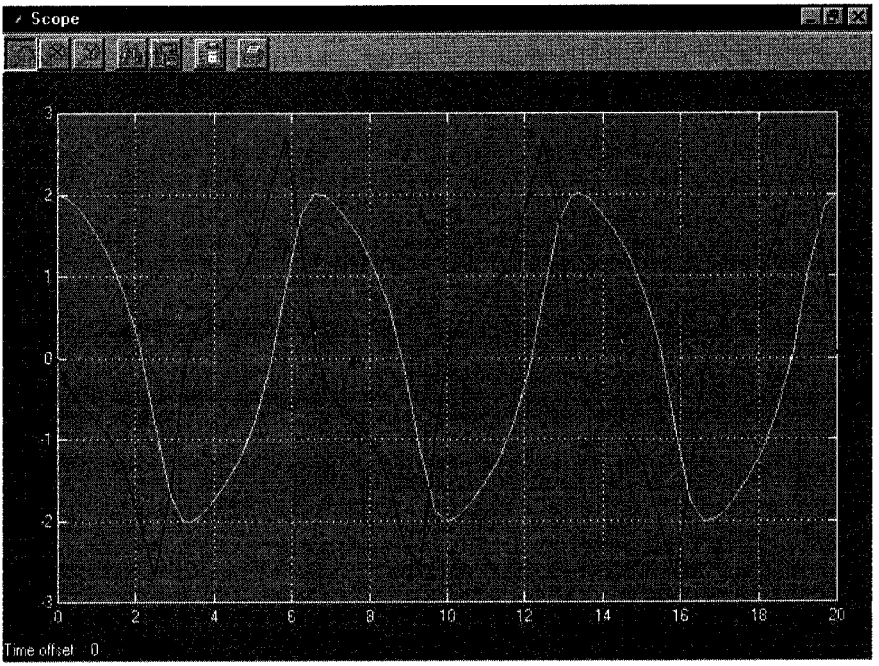


Рис. 10.25. Результат моделирования при F(u)=1-u*u'

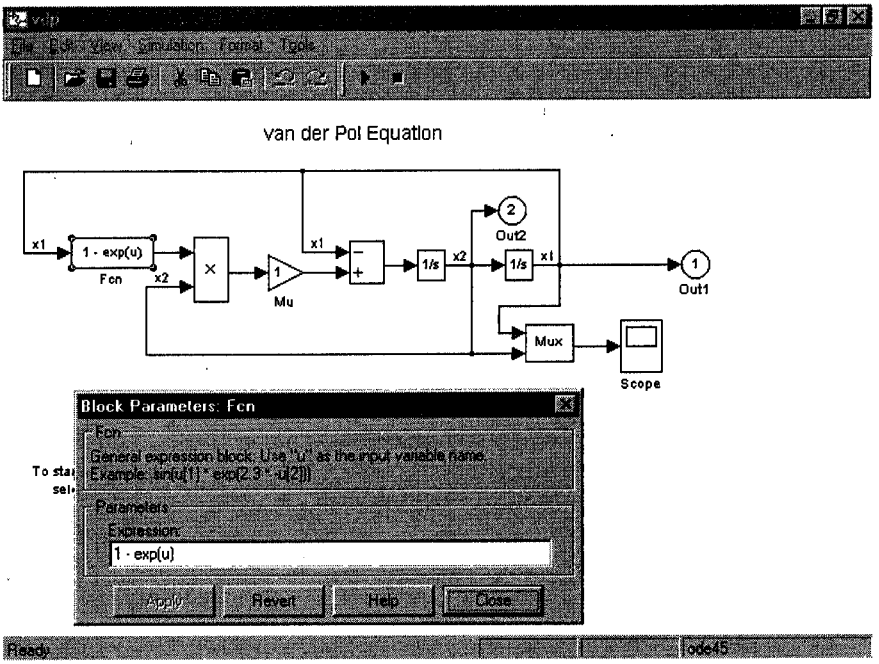


Рис. 10.26. Изменение функции нелинейности в модели системы второго порядка

После уточнения нелинейности и закрытия окна параметров можно произвести пуск измененной модели. Его результаты представлены на рис. 10.27. Сравнение временных диаграмм (осциллограмм) выходных сигналов на этом рисунке с приведенными на рис. 10.25 демонстрирует существенные изменения в характере поведения системы. Во втором варианте предварительная стадия занимает больше времени и колебания вначале имеют существенно большую амплитуду, чем в стационарном режиме.

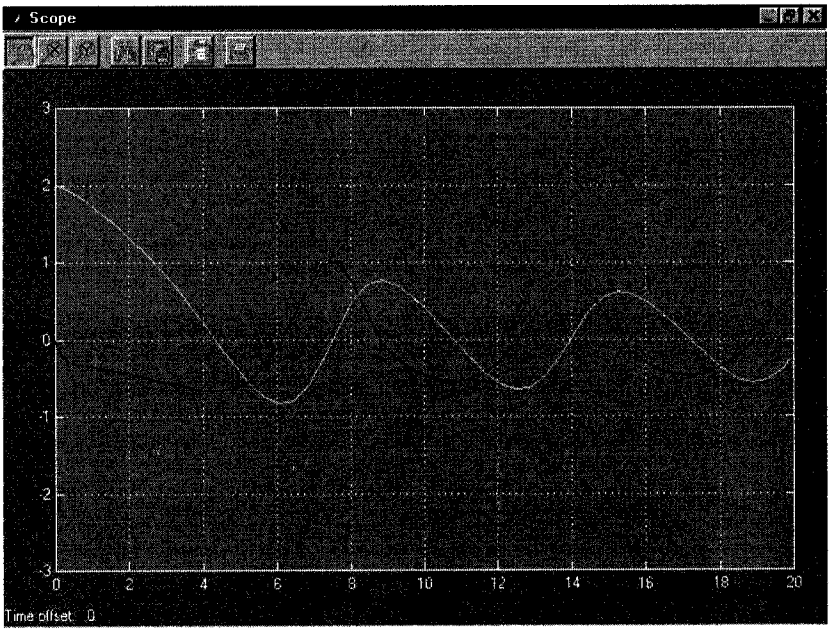


Рис. 10.27. Результат моделирования при F(u)=1-exp(u)

Иногда поведение системы второго порядка удобно представить фазовым портретом колебаний. Для этого модель достаточно дополнить графопостроителем, входы которого подключаются к выходным портам Out 1 и Out 2. На рис. 10.28 показана модель системы второго порядка, дополненная графопостроителем. Здесь Fnc=1-1.1*exp(-u).

В данном случае параметры нелинейности подобраны таким образом, что колебательный процесс возникает только в начале включения системы. Затем за несколько периодов они затухают практически до нуля. Настоящий пример дает наглядное представление о различном характере процессов в подобных системах, что хорошо известно из практики.

В этих примерах мы столкнулись с принципиально важным достоинством пакета Simulink — аналитическое описание многих моделей можно менять, причем оно выполняется по правилам, принятым в системе MATLAB. Благодаря этому математическая сущность модели оказывается достаточно понятной, а результаты моделирования наглядно и адекватно описывают работу сложных моделей при введении в их описание самых различных математических закономерностей.

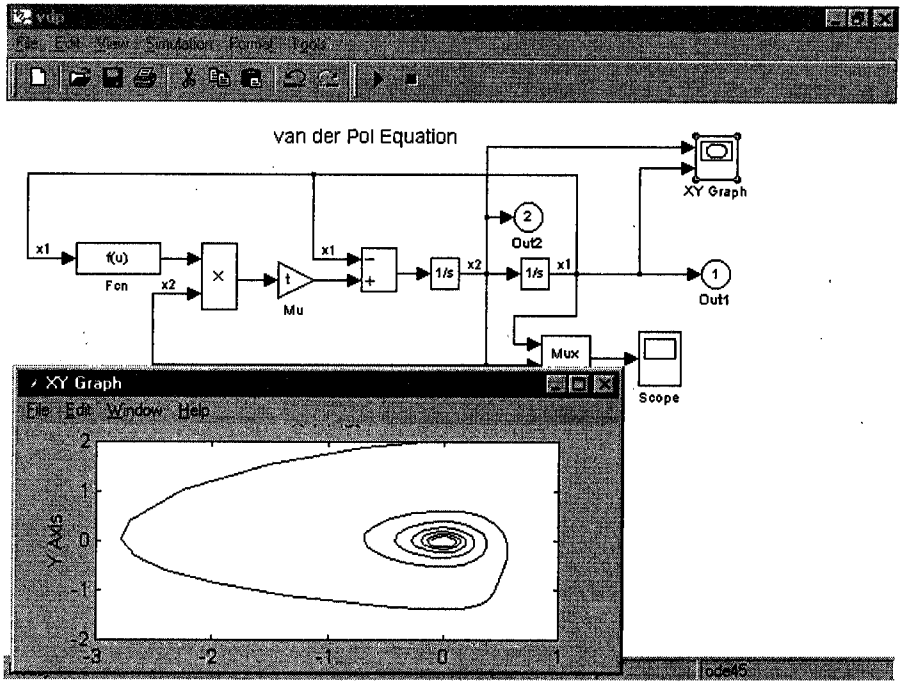


Рис. 10.28. Модель системы второго порядка с новыми данными и фазовый портрет ее колебаний

10.4.3. Работа с решателем и редактором дифференциальных уравнений

Приведенный выше пример является характерным для ситуации, когда моделируется система или устройство, поведение которого описывается дифференциальными уравнениями известного вида — в нашем случае Ван-дер-Поля. Однако Simulink имеет специальный редактор дифференциальных уравнений, с помощью которого можно задать систему дифференциальных уравнений вида Коши и тут же начать ее решение с помощью решателя. Для получения доступа к решателю надо загрузить файл dee, который находится в каталоге MATLAB/TOOLBOX/ SIMULINK/DEE. В результате будет выведена панель решателя с примерами применения, показанная на рис. 10.29 сверху.

Ограничимся примером с именем deedemo1. Он выводит окно всего с двумя блоками: блоком vdp и осциллоскопом x1. Первый блок решает заданное уравнение Ван-дер-Поля, а второй просто отображает решение в виде временной зависимости. Двойной щелчок на блоке vdp открывает редактор дифференциальных уравнений, окно которого показано в правой части рис. 10.29. В этом окне можно модифицировать решаемые уравнения или ввести новые. Для просмотра решения используется осциллоскоп — его окно также представлено на рис. 10.29.

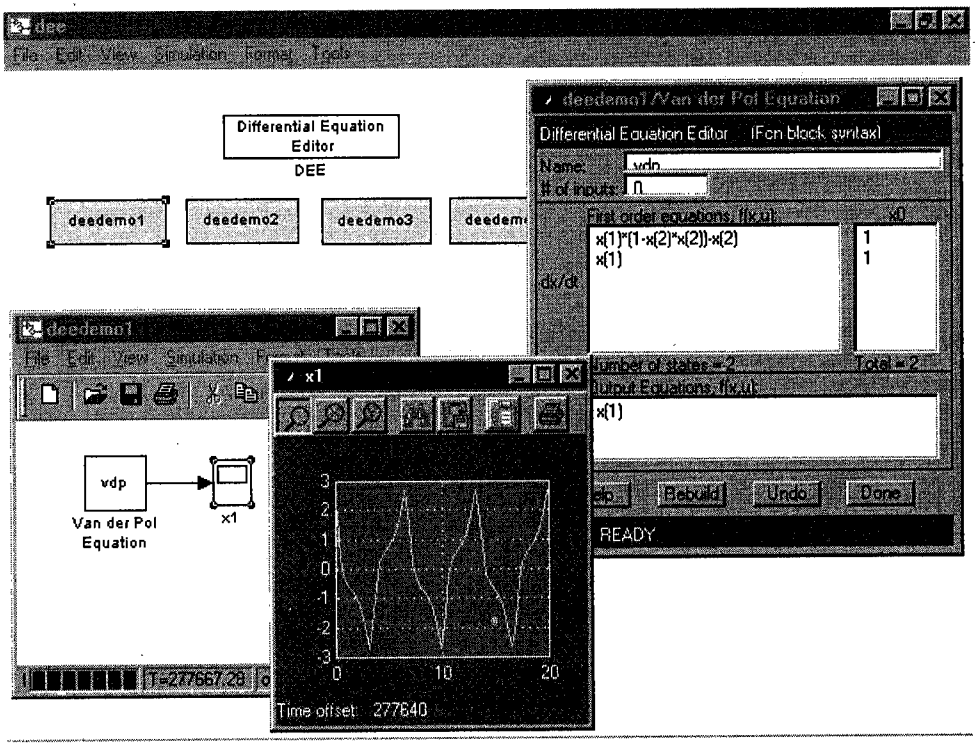


Рис. 10.29. Работа с решателем и редактором дифференциальных уравнений

10.4.4. Моделирование работы автопилота

Возьмем еще один пример из набора, входящего в директорию MATLAB/TOOLBOX/ SIMULINK/SIMDEMO10. Он представлен файлом ben2asys. Этот пример показывает работу одного из вариантов аналогового автопилота — устройства, управляющего самолетом таким образом, чтобы он летел в строго заданном направлении. Модель автопилота представлена на рис. 10.30.

На сей раз моделируется весьма сложное и вполне современное устройство или, скорее, система управления. На приведенном рисунке дана и осциллограмма, характеризующая процесс установления автопилота. Видны значительные колебания переходной характеристики автопилота, характерные для систем регулирования второго порядка.

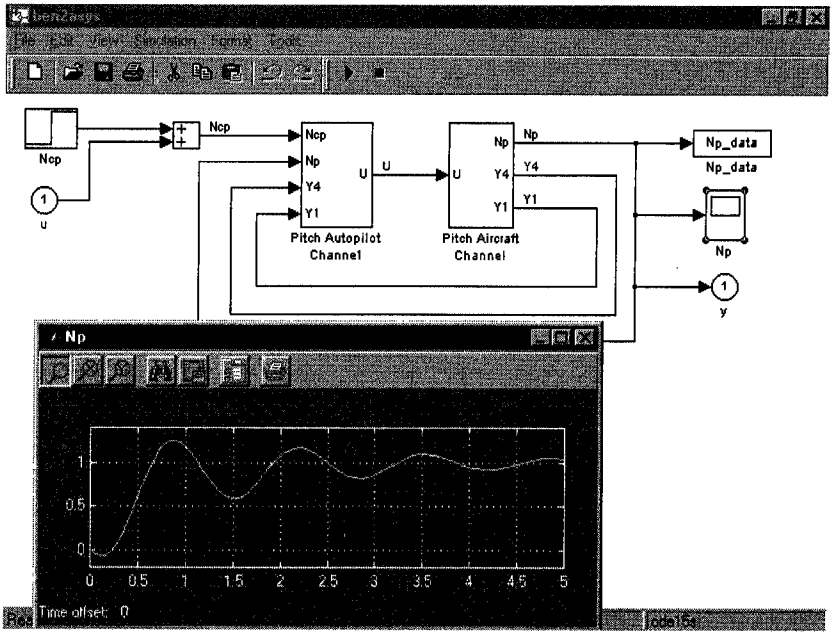


Рис. 10.30. Модель аналогового автопилота

10.4.5. Применение субмоделей

Присмотревшись внимательнее к блок-схеме автопилота, вы заметите, что входящие в нее блоки Pitch Autopilot Channel и Pitch Aircraft Cannel не являются библиотечными компонентами. Это субмодели, то есть ранее отлаженные модели, оформленные в виде блока. Практики часто называют субмодели "черными ящиками", поскольку их содержание не видно и они описываются только системой своих входных и выходных параметров.

Однако в Simulink для выявления того, что входит в такие блоки, достаточно установить на них курсор мыши и быстро дважды щелкнуть левой клавишей. Если появится окно установки параметров, значит, блок — библиотечный компонент. Если же вместо этого появится окно с блок-схемой, значит, блок является субмоделью. На рис. 10.31 показана модель одного из указанных блоков — Pitch Aircraft Cannel.

Нетрудно заметить, что субмодель отличается от обычной модели только тем, что все его входы и выходы выполнены в виде входных In и выходных Out портов. Таким образом, компонентом в модели Simulink может быть и целая система или устройство. Для создания своей библиотеки надо использовать команду **Create Subsystem** в позиции **Edit** главного меню. Можно также использовать библиотечный блок Subsystem, который открывает пустое окно для подготовки субмодели.

Использование концепции субмоделей открывает широчайшие возможности применения пакета Simulink в проектировании сложных технических систем и устройств.

Вы можете, к примеру, задать функции различных типовых микросхем, а затем строить с их помощью различных электронных устройств.

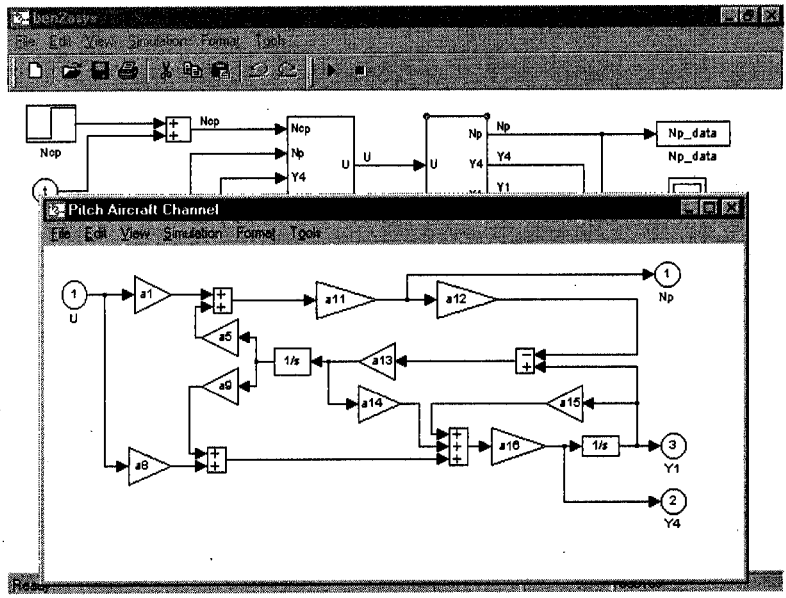


Рис. 10.31. Типовой пример применения субмоделей

10.4.6. Использование S-функции

Как отмечалось, Simulink позволяет определять и использовать системные S-функции. Показанный на рис. 10.32 пример иллюстрирует данную возможность. Здесь анализируется поведение блока с именем simom, в котором задана S-функция. Внизу справа показано окно параметров S-функции, поясняющее ее назначение и позволяющее выбрать заданную функцию и при необходимости указать ее параметры. S-функция может быть представлена файлом, записанным в формате MATLAB, C или Fortran.

Назначение S-функции simom можно определить, исполнив команду:

```
» help simom
```

что выведет сообщение:

```
SIMOM Example state-space M-file S-function with internal A,B,C,D matrices
  This S-function implements a system described by state-space equations:

      dx/dt = A*x + B*u
        y = C*x + D*u
```

where x is the state vector, u is vector of inputs, and y is the vector of output10. The matrices, A,B,C,D are embedded into the M-file.

See sfuntmpl.m for a general S-function template.

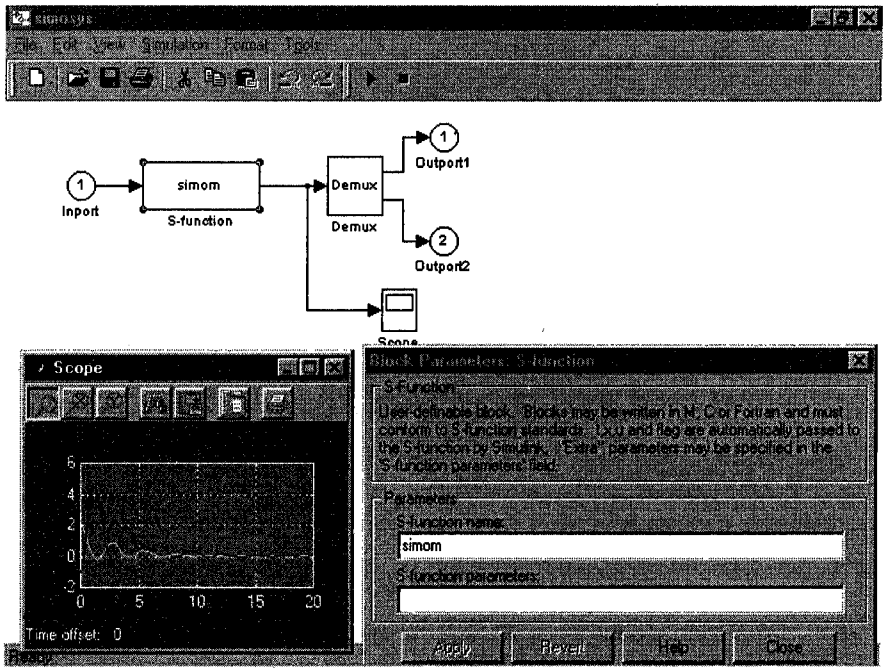


Рис. 10.32. Пример применения S-функции

Итак, в данном случае эта функция задает решение системы дифференциальных уравнений. Команда **type simom** позволяет просмотреть весь файл, что полезно тем, кто решит опробовать собственные силы в подготовке своих S-функций.

Применение аппарата S-функций способствует математической наглядности решения задач в среде пакета Simulink. Однако практическое освоение этого аппарата требует более детального знакомства с данными функциями, которое выходит за рамки настоящей книги.

10.4.7. Применение специальных источников сигналов

В состав блоков основной библиотеки Simulink входят довольно интересные блоки преобразования сигналов. Один из них запоминает значение сигнала в момент выборки и выдает его постоянный уровень до следующей выборки, осуществляя таким образом квантование. Другой источник определяет значение первой производной сигнала в момент отсчета и до следующего отсчета вырабатывает линейно-нарастающий сигнал до следующего отсчета. На рис. 10.33 показано действие таких блоков.

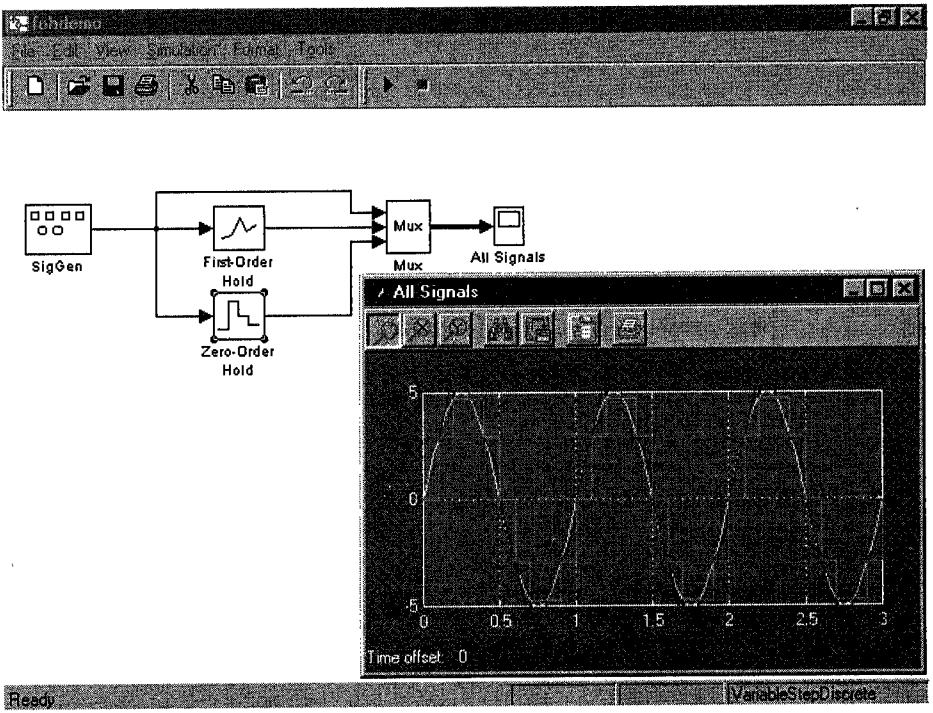


Рис. 10.33. Пример просмотра сигналов от трех блоков

Кроме того, этот пример дает наглядное представление о технике просмотра осциллоскопом сразу трех сигналов — исходного синусоидального и двух сигналов от преобразующих блоков. Для объединения сигналов используется блок Mux.

10.4.8. Как реагирует самолет F14 на действия пилота?

Вы, вероятно, знаете, что самолеты оснащены системами автоматического регулирования, позволяющими плавно менять параметры полета при резком их изменении пилотом. Рис. 10.34 показывает демонстрационную модель такой системы для самолета F14 (файл f14.m). Этот пример наглядно демонстрирует, что предметом моделирования может быть довольно сложная система, в которую, кстати, входит ряд подсистем (субмоделей).

Из приведенных осциллограмм можно сделать вывод, что система достаточно быстро и с очень небольшим перерегулированием реагирует на действие пилота. В этом можно убедиться, просматривая осциллограммы на входе системы и на ее выходе. Малое перерегулирование свидетельствует о достаточно высоком качестве работы данной системы.

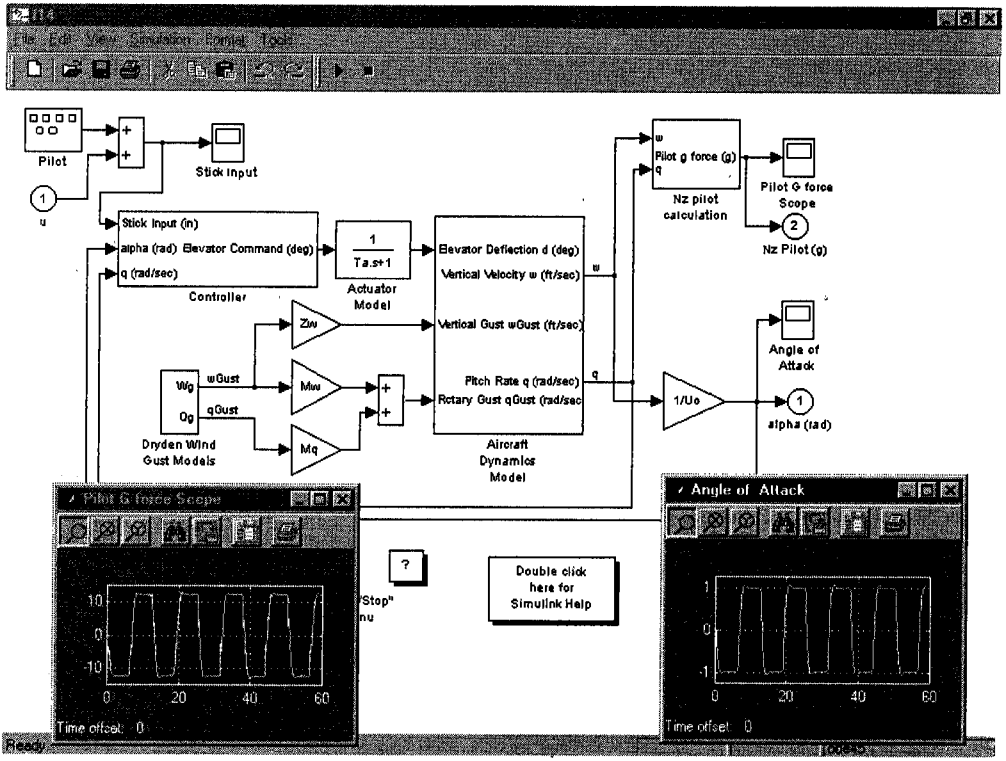


Рис. 10.34. Моделирование системы автоматического регулирования самолета F14

10.4.9. Еще один пример сложной системы

На рис. 10.35 показана модель еще одной сложной системы — Vehicle Suspension. Этот демонстрационный пример, иллюстрирующий работу химического комплекса, приведен не столько ради обсуждения конкретной системы, но как пример вставки в модель блоков нестандартного вида.

При симуляции этой модели используется оригинальное устройство регистрации, позволяющее строить графики, весьма напоминающие графики ленточных самописцев, широко применяемых в химической промышленности. Вид регистрации для данной модели представлен на рис. 10.36.

В библиотеке компонентов Simulink можно найти и множество других регистрирующих устройств, а также средства для создания анимационных видео-клипов, существенно повышающих наглядность визуализации результатов моделирования различных физических, технических и иных систем и устройств.

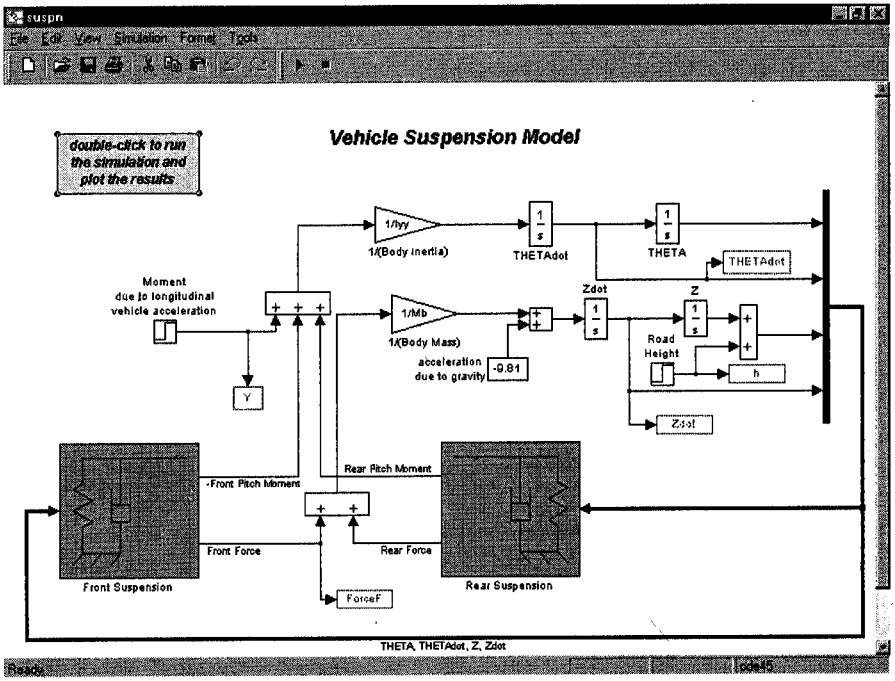


Рис. 10.35. Сложная модель Vehicle Suspension

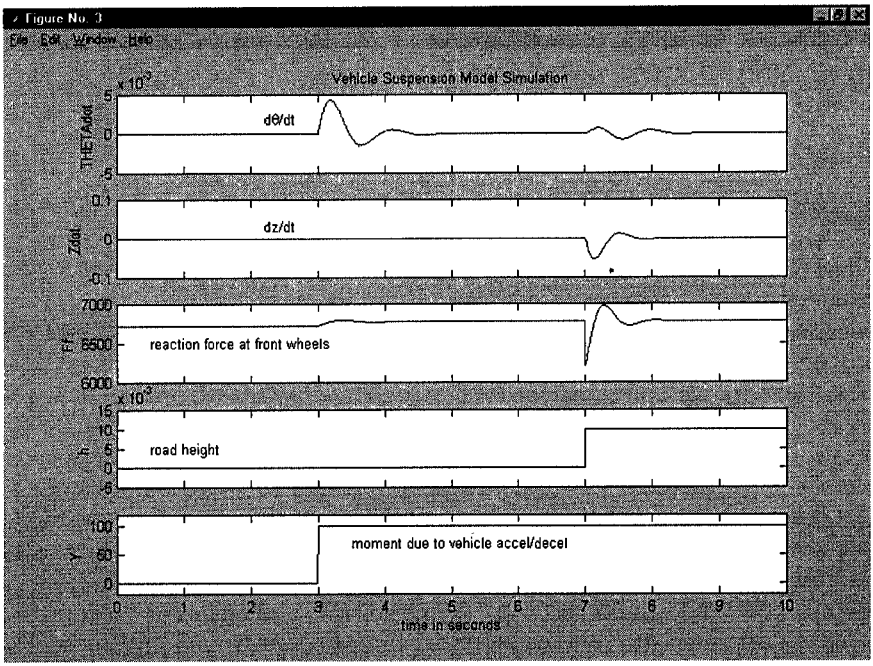


Рис. 10.36. Пример регистрации множества графиков

10.4.10. А ну их — в унитаз!

Может, вам уже надоели все эти математические штучки, модели и диаграммы? И вы про себя подумали: "А ну их — в унитаз!". Для любителей избавляться от своих проблем с помощью унитаза приведем еще один пример — моделирование столь известного устройства, как унитаз. Точнее говоря — сливной системы унитаза. Модель сливного бачка унитаза представлена на рис. 10.37.

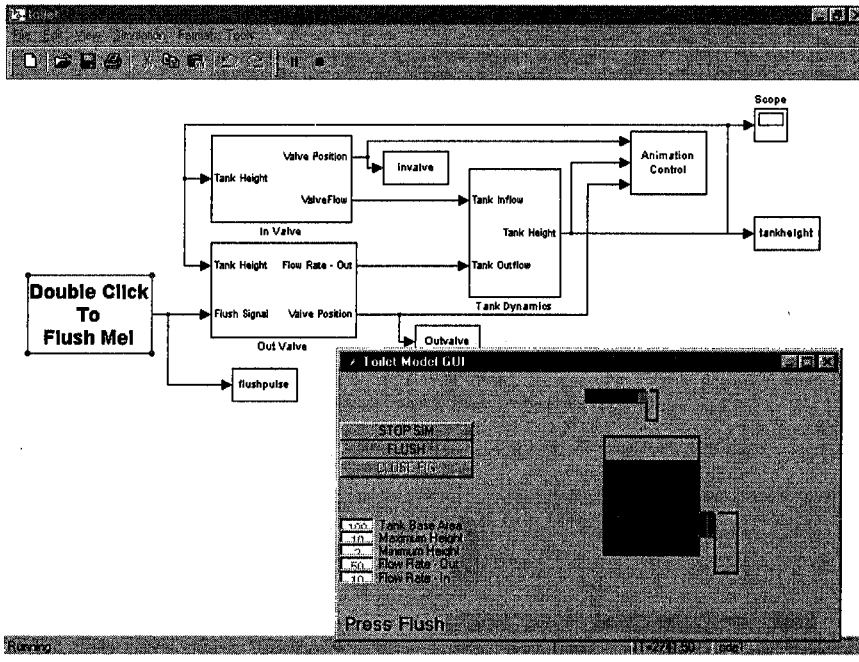


Рис. 10.37. Модель сливной системы унитаза

В этом примере моделируется процесс заполнения водой сливного бачка унитаза с ограничением уровня заполнения, а затем процесс слива воды. Дабы это моделирование и впрямь не показалось вам скучным, слив сопровождается анимационным видеоклипом и оригинальным звуковым комментарием, воспроизводящим звуки слива воды в унитазе.

На этом шуточном примере мы заканчиваем рассмотрение серии примеров применения основной библиотеки компонентов такого уникального программного комплекса, как симулятор динамических систем Simulink. Поговорка гласит: во всякой шутке есть доля правды! Так и моделирование унитаза вовсе не шуточная, а вполне серьезная задача. Она показывает, насколько широки области применения пакета Simulink — от моделирования сливного бачка унитаза или простого ограничителя до симуляции сложнейших систем автоматического управления самолетами и космическими аппаратами.

10.5. Моделирование средств связи и коммуникаций

В этом разделе мы кратко рассмотрим применение пакета Simulink для моделирования коммуникационных и связных устройств, причем не только во временной, но в частотной и спектральной областях их работы.

10.5.1. Библиотека компонентов для моделирования средств телекоммуникаций

Как отмечалось, блок Blocksets&Toolboxes открывает набор дополнительных библиотек. Нетрудно убедиться в том, что по обилию представленных в нем компонентов этот блок даже превосходит блок основной библиотеки компонентов. В качестве примера откроем окно с именем Comm Tbx Library. Пред вами предстанет типичная структурная схема канала связи, состоящего из передающего и приемного подканалов (рис. 10.38).

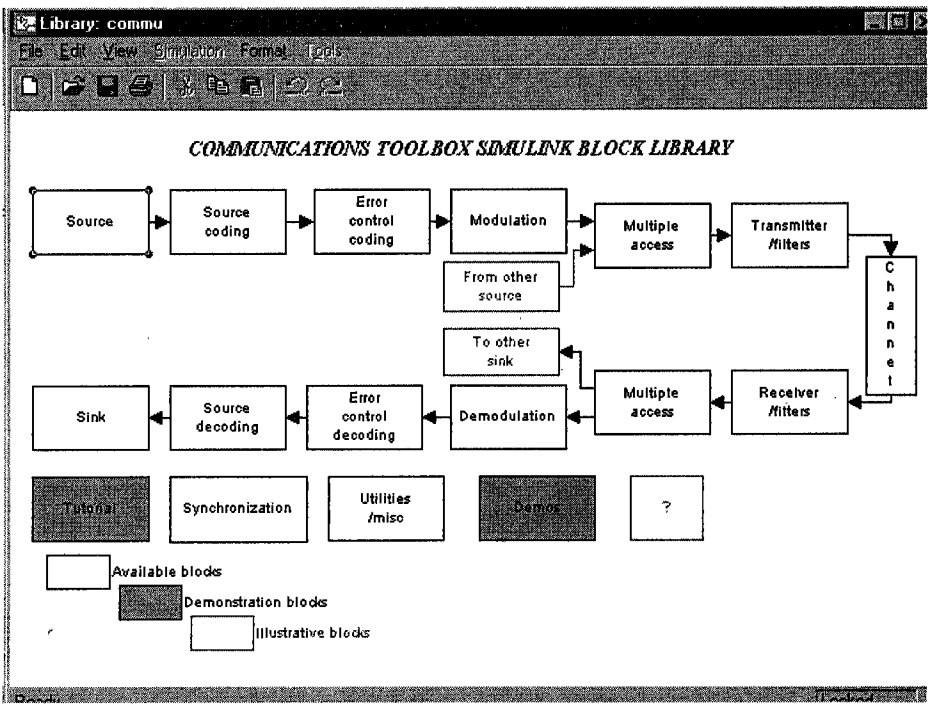


Рис. 10.38. Структура библиотеки коммуникационных средств

На самом деле это не устройство и не система связи, а структура средств библиотеки, представленных в столь оригинальном и наглядном виде. Каждый блок этой структуры открывает доступ к окнам компонентов, которые можно применить для

построения моделей коммуникационных систем и устройств. Например, активизировав блок источников сигналов, можно открыть окно с внушительным набором источников сигналов, показанное на рис. 10.39.

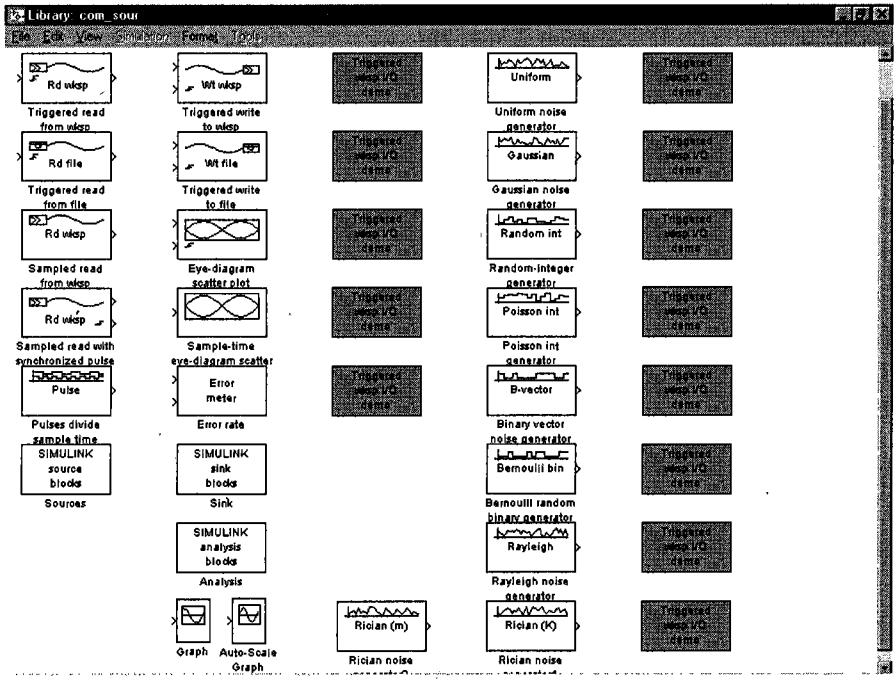


Рис. 10.39. Окно с компонентами — источниками сигналов коммуникационных средств

Выбор нужного источника сигнала (или иного компонента) и его настройка осуществляются так же, как для компонентов основной библиотеки. Далее мы рассмотрим ряд примеров применения дополнительных библиотек для моделирования разнообразных телекоммуникационных средств.

10.5.2. Моделирование высокоскоростной цифровой линии

Наглядным примером применения средств моделирования телекоммуникационных устройств является модель высокоскоростной двоичной цифровой линии, показанная на рис. 10.40.

В данном случае, как и в других примерах, мы не будем описывать детали построения модели каждого устройства и особенности их работы. Цель примеров не в этом, а в том, чтобы наглядно показать, насколько широкий класс тех или иных устройств может моделироваться и настолько наглядны результаты.

Линия, представленная моделью на рис. 10.40, содержит типовые цифровые устройства, характерные для современных средств цифровой связи. Моделирование сопровождается показом сигналов с помощью четырех осциллографов.

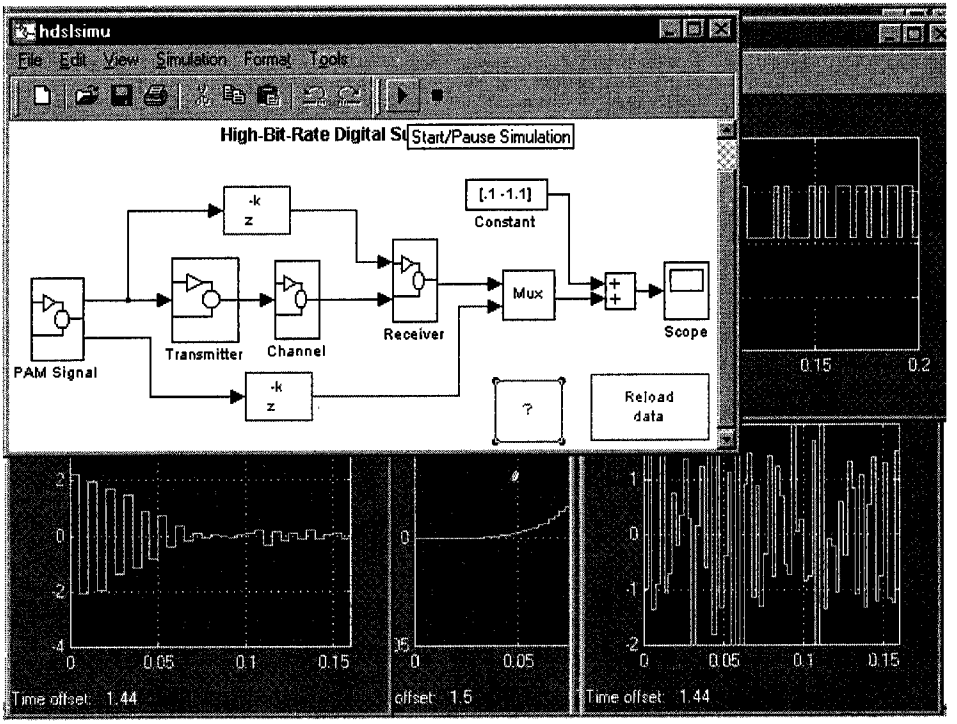


Рис. 10.40. Модель цифровой линии и иллюстрация процесса ее моделирования

10.5.3. Моделирование модема, работающего по протоколу V-34

Широко распространенное в технике телекоммуникаций устройство — модем — содержит модулятор цифровых сигналов, превращающий их в аналоговые сигналы, которые передаются по телефонной или иной линии в соответствии с определенными правилами — протоколами. Принятый с линии аналоговый сигнал с помощью демодулятора преобразуется в цифровой сигнал.

На рис. 10.41 показаны модель модема, работающего по современному протоколу V-34, и диаграммы работы модема.

Наряду с осциллографическим контролем, наглядность которого в данном случае довольно сомнительна, есть возможность контролировать поток цифровых данных на входе и выходе модема. Этот пример — наглядное свидетельство возможности применения Simulink в проектировании таких сложных устройств, как модемы.

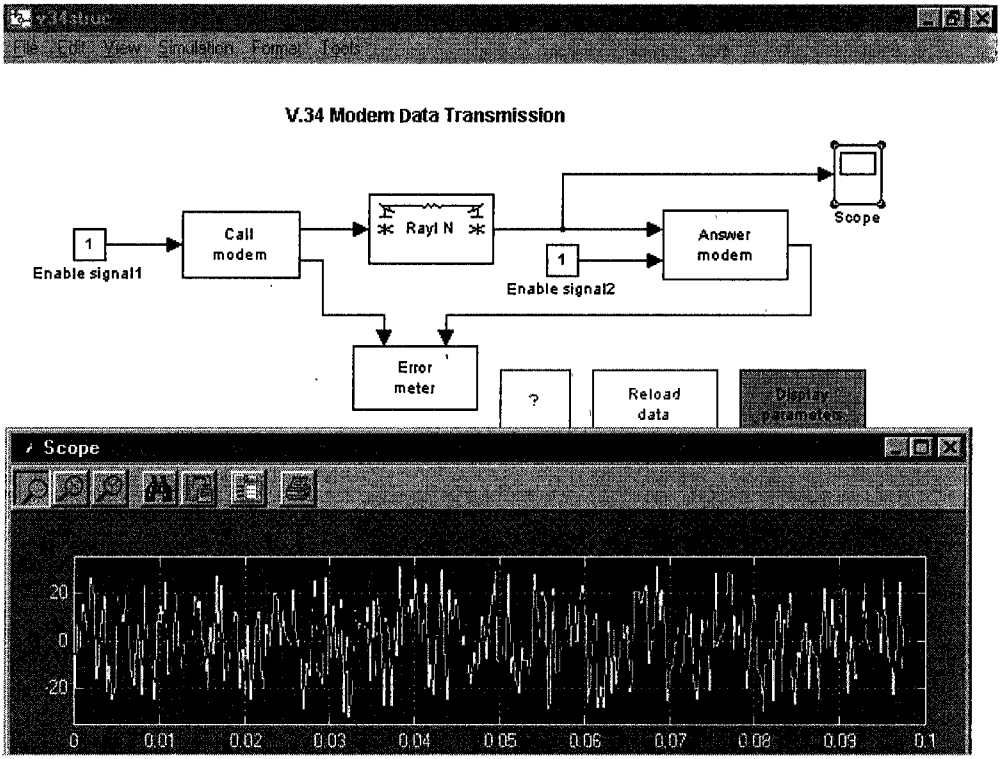


Рис. 10.41. Модель модема, работающего по протоколу V-34

10.5.4. Моделирование адаптивного цифрового фильтра

Следующий пример — это моделирование адаптивного фильтра, частотная характеристика которого непрерывно подстраивается под заданные особенности входного сигнала, что обеспечивает максимальное отношение сигнал/шум на выходе фильтра. Подобная модель фильтра представлена на рис. 10.42. Шумы, маскирующие входной сигнал, задаются с помощью шумового генератора Noise. Сам фильтр по существу является многополосным регулируемым эквалайзером.

Под моделью фильтра на рис. 10.42 показана осциллограмма входного тестового сигнала и амплитудно-частотные характеристики фильтра. Точнее говоря, они показаны в определенный момент времени, так как данная модель позволяет проследить работу фильтра во времени и наблюдать изменения указанных характеристик в течение заданного времени.

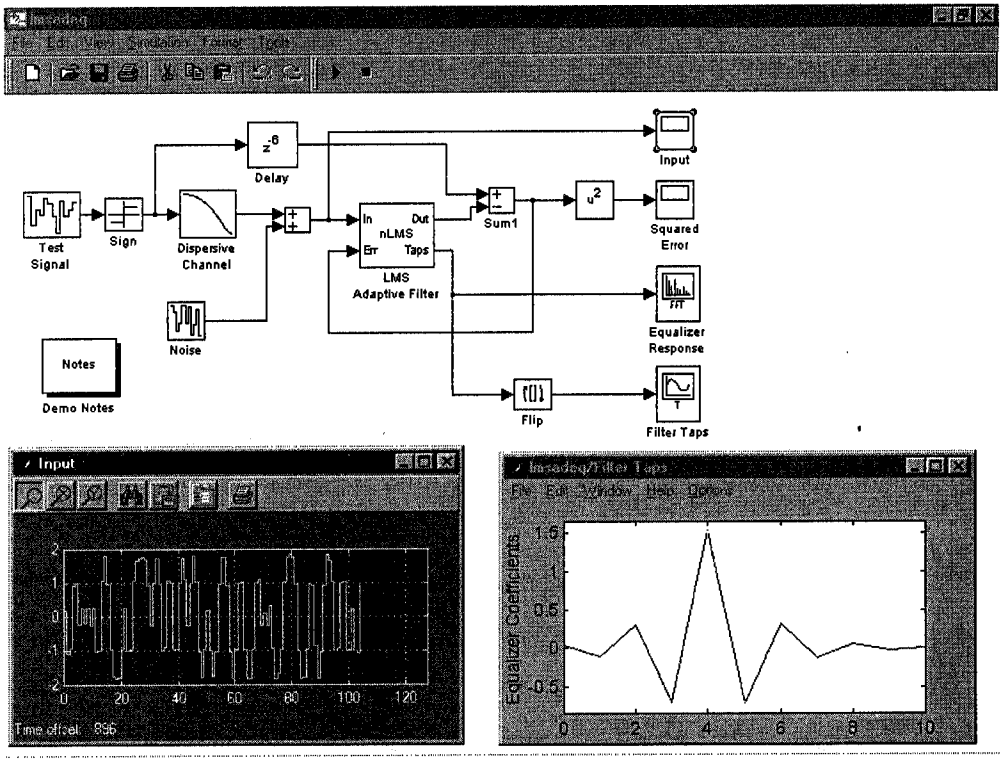


Рис. 10.42. Моделирование адаптивного цифрового фильтра

10.5.5. Моделирование акустического эха

На рис. 10.43 показана модель еще одной интересной акустической системы, создающей характерный эффект акустического эха. Для этого наряду с прямым каналом звука введен второй канал, в котором используется модулируемая цифровая задержка сигнала. Оба канала складываются. Есть возможность прослушать звук исходного источника сигнала и звук при введении акустического эха. Для этого служат кнопки (прямоугольники) с именем Audio Playback (Аудио-Проигрыватель).

Другие кнопки в группе Spectrogramma позволяют выводить спектры сигналов. О том, настолько они оригинальны, свидетельствует контурная спектрограмма, показанная на рис. 10.43 в правом нижнем углу окна. Этот пример иллюстрирует еще один полезный способ визуального представления сложных сигналов.

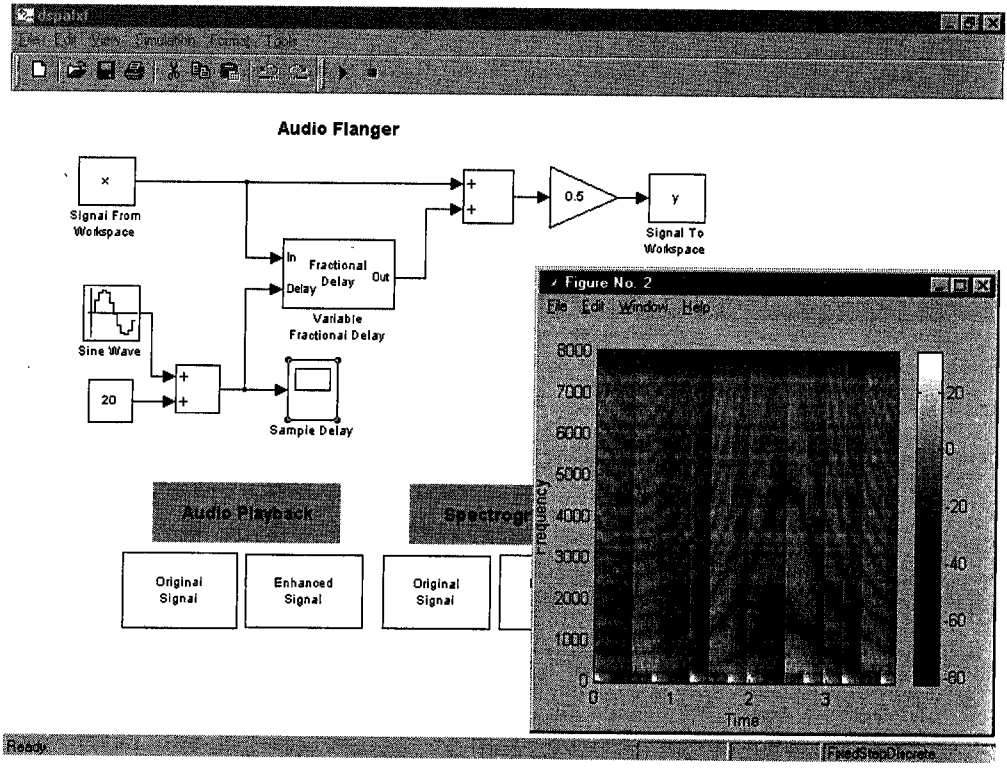


Рис. 10.43. Модель устройства для создания акустического эха

10.5.6. Моделирование однополосного модулятора

Возможность моделирования весьма тонких и сложных физических процессов демонстрирует еще один пример — модель модулятора, обеспечивающего однополосную модуляцию высокочастотного сигнала. Такой сигнал имеет вдвое меньшую полосу, чем обычный амплитудно-модулированный сигнал, и примерно в 8 раз меньшую энергию при подавлении несущей частоты сигнала. Применение однополосной модуляции позволяет резко повысить качество связи, но оно сдерживается сложностью создания модуляторов. Одна из моделей модулятора для однополосной модуляции показана на рис. 10.44.

В данном случае наряду с осциллограммами сигналов можно наблюдать эволюцию амплитудно-частотной характеристики (спектра) входного сигнала.

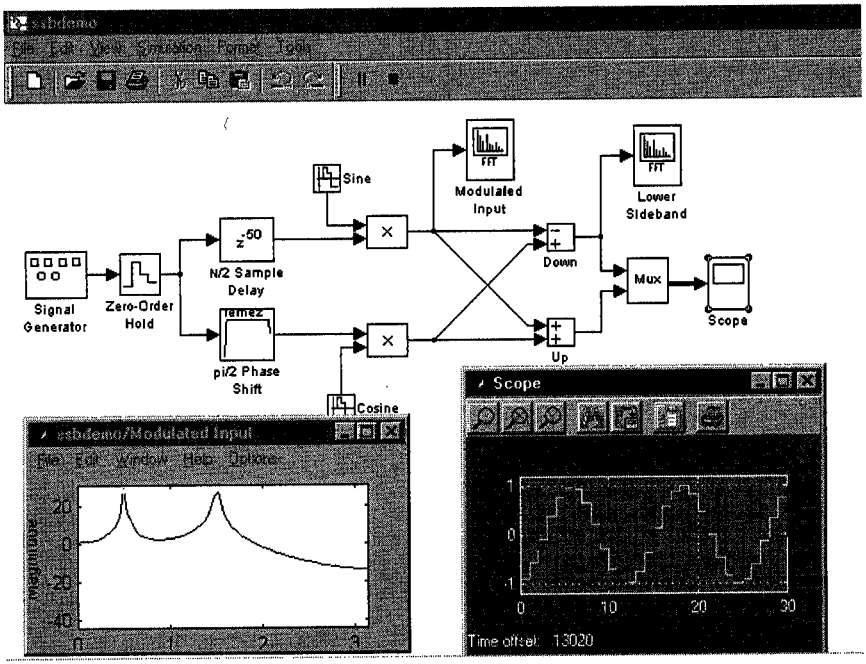


Рис. 10.44. Модель модулятора для однополосной модуляции

10.6. Моделирование преобразовательных устройств

10.6.1. Библиотека моделей преобразовательных устройств

Преобразовательные устройства — прекрасные объекты для моделирования с помощью пакета Simulink. Напомним, что эти устройства обычно предназначены для преобразования электрической энергии одного вида в другой, например, переменного тока в постоянный или наоборот. Современные преобразовательные устройства используют как пассивные компоненты (например, мощные диоды), так и активные (ключевые транзисторы).

Обычно для преобразования постоянного тока используются ключевые регуляторы, теоретический коэффициент полезного действия которых приближается к 100%. Физика процессов, особенно в современных импульсных и резонансных преобразователях, весьма сложна, и пакет Simulink способен оказать неоценимую помощь в изучении таких устройств и в их моделировании.

На рис. 10.45 представлено окно с дополнительной библиотекой компонентов для моделирования преобразовательных устройств. Можно заметить, что данная библиотека

содержит традиционный набор групп компонентов преобразовательных устройств: источники сигналов, мощные линейные компоненты (индукторы, конденсаторы, резонансные контуры и так далее), активные и пассивные компоненты мощной электроники (диоды, ключевые транзисторы — биполярные и полевые, тиристоры и др.), электрические машины, подключающие цепи, измерительные устройства и прочее.

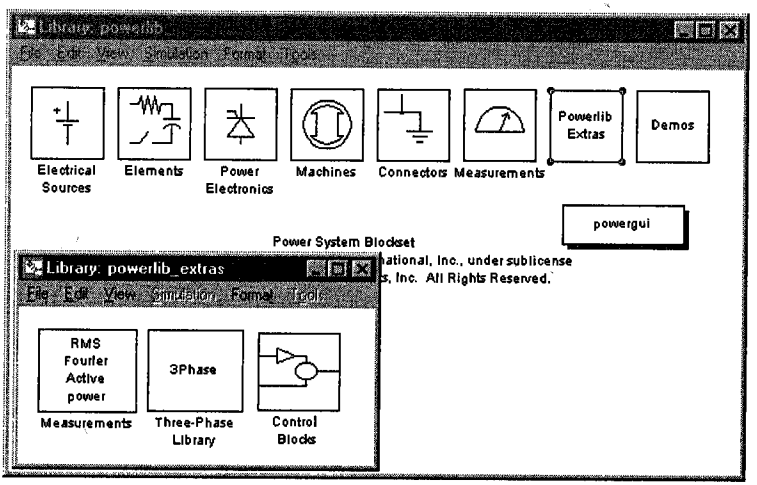


Рис. 10.45. Библиотека компонентов преобразовательных устройств

Активизация любой группы компонентов открывает окно с соответствующими компонентами. Группа Demos открывает окно демонстрационных примеров моделирования преобразовательных устройств. Приведено множество таких примеров, из которых ниже описано лишь два наиболее характерных. К сожалению, объем книги не позволяет остановиться на рассмотрении других, также весьма полезных и поучительных примерах.

10.6.2. Моделирование трехфазного выпрямителя

Трехфазные выпрямители — одно из широко распространенных устройств в промышленности. Как известно, промышленная сеть имеет три фазы, то есть энергия передается по трем проводам (возможен и четвертый нейтральный) в виде синусоидальных напряжений, фазы которых равны 0, 120 и 240 градусам. На рис. 10.46 показана модель трехфазного диодного выпрямителя.

Сам выпрямитель представлен диодами Diode 1, Diode 2 и Diode 3. Индуктивности L1, L2 и L3 имитируют индуктивность проводов, но могут представлять собой и внешние индукторы для гашения быстрых переходных процессов. Нагрузкой выпрямителя как обычно являются дроссель L и резистор Load. Модель диодов предусматривает контрольный вывод для измерения напряжения на диоде и тока, текущего через него.

На рис. 10.46 показаны также экраны осциллоскопов, контролирующих напряжение на одном из диодов, текущие через диоды токи и форму тока в нагрузке — в данном случае это слегка пульсирующий ток с постоянной составляющей около 13 А. Модель дает наглядное представление о работе данного устройства.

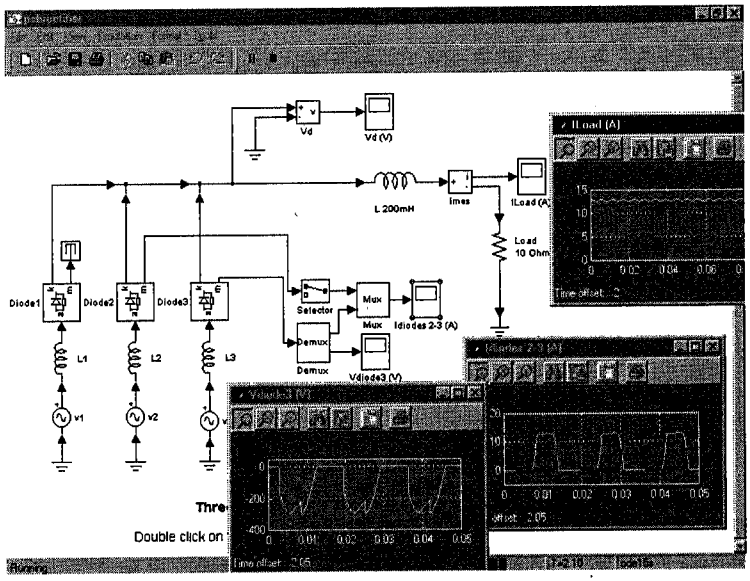


Рис. 10.46. Модель трехфазного диодного выпрямителя

10.6.3. Моделирование квазирезонансного ВЧ-преобразователя на мощном полевом транзисторе

В бытовых и промышленных преобразовательных устройствах широкое применение находят импульсные и квазирезонансные преобразовательные устройства. В них для регулирования энергии постоянного тока используются ключевые транзисторы с импульсным управлением. Если на их выходе применяется последовательный колебательный контур, то преобразователь называют резонансным, или квазирезонансным, если частота управления ключом несколько отлична от резонансной частоты контура. Управление энергией осуществляется путем изменения скважности управляющих импульсов и частоты.

В последнее время в таких преобразователях находят применение мощные полевые транзисторы, которые позволяют получить одновременно высокий коэффициент полезного действия преобразователей и высокую частоту преобразования. На рис. 10.47 представлена модель высокочастотного преобразователя на мощном полевом транзисторе Mosfet с импульсным управлением от источника Pulse Generator (Импульсный Генератор). Ключ на полевом транзисторе нагружен на последовательный резонансный контур, содержащий реальную катушку индуктивности Lr и конденсатор Cr, зашунтированный диодом.

Если резонансная частота колебательного контура близка к частоте переключающих транзистор импульсов, то имеет место режим, при котором переключение транзистора происходит в моменты, когда проходящий через него ток близок к нулю. Таким образом, потери энергии на транзисторном ключе сводятся к минимуму.

Данная модель открывает широкие возможности в изучении весьма сложных физических процессов в этой нелинейно-параметрической системе. Как видно из приведенных осциллограмм и фазового портрета колебаний, возможна регулировка выходного тока как изменением частоты с помощью ключа на мощном полевом транзисторе, так и изменением скважности импульсного управляющего сигнала. Нетрудно проанализировать работу устройства при изменении указанных параметров в широких пределах.

Варианты этой силовой цепи широко применяются во многих серийно выпускаемых источниках питания. Одной из первых такие источники стала выпускать фирма Hewlett Packard, а сейчас они становятся де-факто промышленным стандартом. Мы еще раз напоминаем, что этими примерами далеко не исчерпываются возможности применения пакета Simulink в моделировании преобразовательных и других устройств.

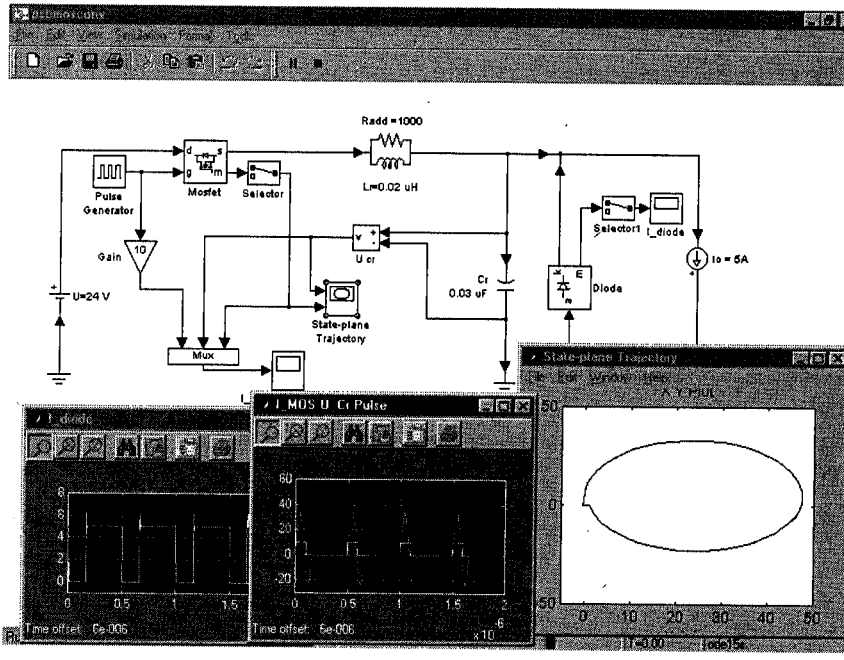


Рис. 10.47. Пример моделирования квазирезонансного преобразования с ключом на мощном полевом транзисторе

Приложение
Программные продукты класса MATLAB фирмы MathWorks Inc

Ниже дано аннотационное описание основных программных продуктов фирмы MathWorks, которыми может комплектоваться матричная математическая система MATLAB. Знакомство с этими программными продуктами полезно для оценки того, какие из них нужны конкретному пользователю при инсталляции весьма объемной по затратам дисковой памяти системы MATLAB.

Базовые средства

MATLAB for Windows 5.2.1

Основу математической матричной системы MATLAB, известной как 10-юбилейная реализация, составляют современные средства GUI-интерфейса, ядро системы и библиотеки для решения научно-технических, инженерных и математических задач. В состав MATLAB 5.2.1 входит ряд компонентов, название, номер версии и дату создания которых можно вывести командой **ver**:

MATLAB Toolbox	**Version 5.2**	**18-Dec-1997**
Database Toolbox	**Version 1.0**	**21-Oct-1997**
Database Toolbox Demonstration Functions	**Version 1.0**	**21-Oct-1997**
Power System Blockset	**Version 1.0**	**21-Nov-1997**
MATLAB Compiler	**Version 1.2**	**05-Dec-1997**
Communications Toolbox	**Version 1.3**	**01-Dec-1997**
Symbolic Math Toolbox	**Version 2.0.1**	**21-Nov-1997**
NAG Foundation Toolbox - Numerical&Statistic	**Version 1.0.2**	**21-Nov-1997**
Mapping Toolbox	**Version 1.0.1**	**21-Nov-1997**
Wavelet Toolbox	**Version 1.1**	**21-Nov-1997**
Partial Differential Equation Toolbox	**Version 1.0.3**	**21-Nov-1997**
Financial Toolbox	**Version 1.1**	**21-Nov-1997**
LMI Control Toolbox	**Version 1.0.4**	**05-Dec-1997**
QFT Control Design Toolbox	**Version 1.0.3**	**21-Nov-1997**
Fixed-Point Blockset	**Version 1.2**	**01-Jan-1998**
DSP Blockset	**Version 2.2**	**21-Nov-1997**
Fuzzy Logic Toolbox	**Version 2.0**	**15-Nov-1997**
Model Predictive Control Toolbox	**Version 1.0.3**	**21-Nov-1997**
Frequency Domain Identification Toolbox	**Version 2.0.3**	**20-Sep-1997**
Higher-Order Spectral Analysis Toolbox	**Version 2.0.2**	**21-Nov-1997**
Statistics Toolbox	**Version 2.1.1**	**21-Nov-1997**
Nonlinear Control Design Blockset	**Version 1.1.2**	**21-Nov-1997**
Image Processing Toolbox	**Version 2.1**	**15-Dec-1997**

Neural Network Toolbox	**Version 3.0**	**05-Dec-1997**
Mu-Analysis and Synthesis Toolbox	**Version 3.0.3**	**21-Nov-1997**
Signal Processing Toolbox	**Version 4.1**	**21-Nov-1997**
Spline Toolbox	**Version 2.0**	**01-Dec-1997**
Optimization Toolbox	**Version 1.5.2**	**09-Sep-1997**
Robust Control Toolbox	**Version 2.0.5**	**21-Nov-1997**
System Identification Toolbox	**Version 4.0.4**	**21-Nov-1997**
Control System Toolbox	**Version 4.1**	**05-Dec-1997**
Real-Time Workshop	**Version 2.2.1**	**24-Mar-98**
Stateflow	**Version 1.0.7**	**30-Mar-1998**
Simulink	**Version 2.2.1**	**24-Mar-98**
MATLAB Tour	**Version 1.1**	**18-Dec-1997**

The Student Edition of MATLAB

Упрощенная студенческая версия системы MATLAB, содержащая значительно урезанный состав пакетов прикладных программ, но тем не менее вполне удовлетворяющая требованиям системы высшего и специального образования, занимающая на диске намного меньше места, чем типовая система, и стоящая заметно дешевле. Широко используется в системе образования многих ведущих университетов Запада.

MATLAB Compiler

Компилятор для программ, создаваемых на языке программирования системы MATLAB. Транслирует коды этих программ в программы на языке C. Применение компилятора обеспечивает возможность создания исполняемых кодов (программ), время исполнения которых для программ с большим числом циклических операций уменьшается в 10-15 раз.

MATLAB C Math Library

Библиотека дополнительных математических функций на языке C, расширяющая возможности базовой системы MATLAB в части решения математических задач. Пользователь имеет возможность пополнять и модифицировать эту библиотеку, содержащую около 300 заранее откомпилированных и потому быстро исполняемых функций. Библиотека поддерживает следующие возможности: матричную арифметику, сравнение матриц, решение систем линейных уравнений, LU- и QR- разложения матриц, вычисление сингулярных чисел и собственных значений, нахождение обратной матрицы и вычисление детерминанта, возведение матрицы в степень, элементарную математику, вычисление бета- и гамма- функций, функции ошибки и эллиптические функции, статистику и анализ данных, нахождение корней многочленов, операции фильтрации, свертки и быстрого преобразования Фурье, интерполяцию рядом методов, операции над строками и операции ввода/вывода в файл. Таким образом, на эту библиотеку возложена реализация многих базовых функций системы MATLAB, описанных в данной книге.

MATLAB C++ Math Library

Библиотека дополнительных математических функций на языке C++, который является более современным вариантом языка C. Библиотека расширяет возможности базовой системы MATLAB в части решения математических задач. Пользователь имеет возможность пополнять и модифицировать эту библиотеку. Состав библиотеки практически аналогичен ранее отмеченной библиотеке на языке C.

Simulink for Windows

Система имитационного моделирования (симулирования) моделей, состоящих из графических блоков с заданными свойствами (параметрами). Компоненты моделей, в свою очередь, являются графическими блоками и субмоделями, которые содержатся в ряде библиотек и с помощью мыши могут переноситься в основное окно системы и соединяться друг с другом необходимыми связями. В состав моделей могут включаться источники сигналов различного вида, виртуальные регистрирующие приборы, графические средства анимации. Запуск имитации обеспечивает математическое модулирование построенной модели с наглядным визуальным представлением результатов моделирования. Пакет достаточно подробно описан в главе 10 данной книги.

The Student Edition of Simulink

Упрощенная версия системы имитационного моделирования моделей, состоящих из компонентов с графическим интерфейсом. Используется для укомплектования студенческой версии системы MATLAB.

Simulink Real Time Workshop (RTW)

Подключающаяся к Simulink мощная подсистема имитационного моделирования в реальном масштабе времени (при наличии дополнительных аппаратных средств в виде плат расширения ПК). Кроме того, система позволяет создавать исполняемые коды моделируемых моделей.

Simulink RTW Ada Extension

Расширение подсистемы Simulink Real Time Workshop (RTW), разработанное на основе языка Ada, принятого для создания программного обеспечения военных систем в США. Обеспечивает дальнейшее наращивание вычислительной мощности пакета Simulink.

Excel Link

Программное средство для создания интерфейса связи с популярными табличными процессорами класса Excel 5.0 и выше, созданными фирмой Microsoft Inc. для офисных комплексов Microsoft Office. Обеспечивается простая связь между MATLAB и Excel, двухсторонний обмен данными, просмотр, редактирование и обработка данных из среды MATLAB в Excel, подготовка Excel-приложений и др.

Пакеты математических программ

NAG Foundation Toolbox

Одна из самых мощных библиотек математических функций, созданная специальной группой The Numerical Algorithms Group Ltd. Пакет содержит более 240 функций. Названия функций и синтаксис их вызова заимствованы из известной библиотеки NAG Foundation Library. Вследствие этого опытные пользователи NAG Fortran могут без затруднений работать с пакетом NAG в MATLAB. Библиотека NAG Foundation предоставляет свои функции в виде объектных кодов и соответствующих M-файлов для их вызова. Пользователь может легко модифицировать эти MEX-файлы на уровне исходного кода. Он обеспечивает следующие вычисления:

- Корни многочленов и модифицированный метод Лагерра
- Вычисление суммы ряда: дискретное и эрмитово-дискретное преобразование Фурье
- ОДУ: методы Адамса и Рунге-Кутта
- Уравнения в частных производных
- Интерполяция
- Вычисление собственных значений и векторов, сингулярных чисел, поддержка комплексных и действительных матриц
- Аппроксимация кривых и поверхностей: полиномы, кубические сплайны, полиномы Чебышева
- Минимизация и максимизация функций: линейное и квадратичное программирование, экстремумы функций многих переменных
- Матричная факторизация
- Решение систем линейных уравнений
- Линейные уравнения (LAPACK)
- Статистические расчеты, включая дескриптивную статистику, распределения вероятностей
- Корреляционный и регрессионный анализ: линейные, многомерные и обобщенные линейные модели
- Многомерные методы: (главных компонент, ортогональные вращения и др.).
- Генерация случайных чисел: нормальное распределение, распределения Пуассона, Вейбула и Коши

- Непараметрические статистики: Фридмана, Крускала-Уоллиса, Манна-Уитни
- Временные ряды: одномерные и многомерные
- Аппроксимации специальных функций: интегральная экспонента, гамма-функция, функции Бесселя и Ганкеля

Этот пакет позволяет пользователю создавать программы на Fortran, которые динамически линкуются с MATLAB.

Neural Networks Toolbox

Пакет прикладных программ, содержащих средства для построения нейронных сетей, базирующихся на поведении нейрона. Пакет обеспечивает эффективную поддержку проектирования, обучения и моделирования множества известных сетевых парадигм, от базовых моделей персептрона до самых современных ассоциативных и самоорганизующихся сетей. Пакет может быть использован для исследования и применения нейронных сетей к таким задачам, как обработка сигналов, нелинейное управление и финансовое моделирование.

Обеспечена возможность генерации переносимого С-кода с помощью Workshop

В пакет включены более 15 известных типов сетей и обучающих правил, позволяющих пользователю выбирать наиболее подходящую для конкретного приложения или исследовательской задачи парадигму. Для каждого типа архитектуры и обучающих правил имеются функции инициализации, обучения, адаптации, создания и моделирования, демонстрации и пример приложения сети.

Для управляемых сетей вы можете выбрать прямую или рекуррентную архитектуру, используя множество обучающих правил и методов проектирования, таких как персептрон, обратное распространение, обратное распространение Левенберга, сети с радиальным базисом и рекуррентные сети. Вы можете легко изменять любые архитектуры, обучаемые правила или переходные функции или добавлять новые, не прибегая к программированию на С или Fortran.

Spline Toolbox

Пакет прикладных программ для работы со сплайнами. Поддерживает одномерную, двумерную и многомерную сплайн- интерполяцию и аппроксимацию. Обеспечивает представление и отображение сложных данных и поддержку графики. Пакет позволяет выполнять интерполяцию, аппроксимацию и преобразование сплайнов в В-форме в кусочное представление, интерполяцию кубическими сплайнами и сглаживание, осуществление операций над сплайнами: вычисление производной, интеграла и отображения.

Пакет Spline оснащен программами операций с B-сплайнами, описанными в работе "A Practical Guide to Splines" Карлом Дебуром, создателем сплайнов и автором пакета Spline. Функции пакета в сочетании с языком MATLAB и подробным руководством пользователя облегчают понимание сплайнов и их эффективное применение к решению разнообразных задач.

В пакет включены программы для работы с двумя наиболее употребительными формами представления сплайнов: B-формой и кусочными полиномами. B-форма удобна на этапе построения сплайнов, в то время как кусочно-полиномиальная форма более эффективна во время постоянной работы со сплайном. Пакет включает функции для создания, отображения, интерполяции, аппроксимации и обработки сплайнов в B-форме и в виде кусочных полиномов.

Statistics Toolbox

Пакет прикладных программ по статистике, резко расширяющий возможности системы MATLAB в реализации статистических вычислений и в статистической обработке данных. Содержит весьма представительный набор средств генерации случайных чисел, векторов, матриц и массивов с различными законами распределения, а также множество статистических функций. Следует отметить, что наиболее распространенные статистические функции входят в состав системы MATLAB (в том числе функции генерации случайных данных с равномерным и нормальным распределением). Основные возможности пакета:

- Дескриптивная статистика
- Распределения вероятностей
- Оценка параметров и аппроксимация
- Проверка гипотез
- Множественная регрессия
- Интерактивная пошаговая регрессия
- Моделирование Монте-Карло
- Аппроксимация на интервалах
- Статистическое управление процессами
- Планирование эксперимента
- Моделирование поверхности отклика
- Аппроксимация нелинейной модели
- Анализ главных компонент
- Статистические графики
- Графический интерфейс пользователя

Пакет включает 20 различных распределений вероятностей, в их числе T, F и Хи квадрат. Подбор параметров, графическое отображение распределений и способы вычисления лучших аппроксимаций предоставляются для всех типов распределений. Предусмотрено множество интерактивных инструментов для динамической визуали

зации и анализа данных. Имеются специализированные интерфейсы для моделирования поверхности отклика, визуализации распределений, генерации случайных чисел и линий уровня.

Optimization Toolbox

Пакет прикладных задач для решения задач оптимизации и решения систем нелинейных уравнений. Поддерживает основные методы оптимизации функций ряда переменных:

- Безусловная оптимизация нелинейных функций
- Метод наименьших квадратов и нелинейная интерполяция
- Решение нелинейных уравнений
- Линейное программирование
- Квадратичное программирование
- Условная минимизация нелинейных функций
- Метод минимакса
- Многокритериальная оптимизация

Разнообразные примеры демонстрируют эффективное применение функций пакета. С их помощью также можно сравнить, как одна и та же задача решается разными методами.

Fuzzy Logic Toolbox

Пакет прикладных программ, относящихся к теории размытых множеств. Обеспечивается поддержка современных методов нечеткой кластеризации и адаптивные нечеткие нейронные сети. Графические средства пакета позволяют интерактивно отслеживать особенности поведения системы. Основные возможности пакета:

- Определение переменных, нечетких правил и функций принадлежности
- Интерактивный просмотр нечеткого логического вывода
- Современные методы: адаптивный нечеткий вывод с использованием нейронных сетей, нечеткая кластеризация
- Интерактивное динамическое моделирование в SIMULINK
- Генерация переносимого С-кода с помощью Real-Time Workshop.

Partial Differencial Equations Toolbox

Весьма важный пакет прикладных программ, содержащий множество функций для решения систем дифференциальных уравнений с частными производными. Предоставляет эффективные средства для решения таких систем уравнений, в том числе

жестких. В пакете используется метод конечных элементов. Команды и графический интерфейс пакета могут быть использованы для математического моделирования PDE применительно к широкому классу инженерных и научных приложений, включая задачи по сопротивлению материалов, расчеты электромагнитных устройств, задачи тепломассопереноса и диффузии. Основные возможности пакета:

- Полноценный графический интерфейс для обработки PDE второго порядка
- Автоматический и адаптивный выбор сетки
- Задание граничных условий: Дирихле, Неймана и смешанные
- Гибкая постановка задачи с использованием синтаксиса MATLAB
- Полностью автоматическое сеточное разбиение и выбор величины конечных элементов
- Нелинейные и адаптивные расчетные схемы
- Возможность визуализации полей различных параметров и функций решения, демонстрация принятого разбиения и анимационные эффекты

Пакет интуитивно следует шести шагам решения PDE с помощью метода конечных элементов. Эти шаги и соответствующие режимы пакета таковы: определение геометрии (режим рисования), задание граничных условий (режим граничных условий), выбор коэффициентов, определяющих задачу (режим PDE), дискретизация конечных элементов (режим сетки), задание начальных условий и решение PDE (режим решения), последующая обработка решения (режим графика).

Symbolic Math Toolbox

Пакет прикладных программ, дающих системе MATLAB принципиально новые возможности — решение задач в символьном (аналитическом) виде, включая реализацию точной арифметики произвольной разрядности. Пакет базируется на применении ядра символьной математики одной из самых мощных систем компьютерной алгебры Maple V R4. Обеспечивает выполнение символьного дифференцирования и интегрирования, вычисление сумм и произведений, разложение в ряды Тейлора и Маклорена, вычисление корней полиномов, решение в аналитическом виде нелинейных уравнений, всевозможные символьные преобразования, подстановки, операции с полиномами и многое другое. Имеет команды прямого доступа в ядро системы Maple V. Пакет полностью описан в главе 9 данной книги.

Extended Symbolic Math Toolbox

Расширенный пакет прикладных программ для решения задач в символьном виде. Он дает расширенные возможности в части доступа к ядру символьных операций системы символьной математики Maple V R4. В частности, он позволяет готовить процедуры с синтаксисом языка программирования системы Maple V R4 и инсталлировать их в системе MATLAB.

Анализ и синтез систем управления

Control System Toolbox

Пакет Control System предназначен для моделирования, анализа и проектирования систем автоматического управления, как непрерывных, так и дискретных. Функции пакета реализуют традиционные методы передаточных функций и современные методы пространства состояний. Частотный и временной отклик, диаграммы расположения нулей/полюсов могут быть быстро вычислены и отображены на экране. В пакете реализованы:

- Полный набор средств для анализа MIMO и SISO линейных систем
- Временные характеристики: передаточная и переходная функции, реакция на произвольное воздействие
- Частотные характеристики: диаграммы Боде, Николса, Найквиста и др.
- Разработка обратных связей
- Проектирование LQR/LQE регуляторов
- Характеристики моделей: управляемость, наблюдаемость, понижение порядка моделей
- Поддержка систем с запаздыванием

Дополнительные функции построения моделей позволяют конструировать более сложные модели. Временной отклик может быть рассчитан для импульсного входа, единичного скачка или произвольного входного сигнала. Имеются также функции для анализа сингулярных чисел.

Интерактивная среда для сравнения временного и частотного отклика систем предоставляет пользователю графические управляющие элементы для одновременного отображения откликов и переключения между ними. Можно вычислять различные характеристики откликов, такие как время разгона и время регулирования.

Пакет Control System содержит средства для выбора параметров обратной связи. Среди традиционных методов: анализ особых точек, определение коэффициента усиления и затухания. Среди современных методов: линейно-квадратичное регулирование и др. Пакет Control System включает большое количество алгоритмов для проектирования и анализа систем управления. Кроме того, он обладает настраиваемым окружением и позволяет создавать свои собственные М-файлы.

Nonlinear Control Design Toolbox

Пакет прикладных программ для построения нелинейных систем контроля и управления Nonlinear Control Design (NCD) Blockset реализует метод динамической оптимизации для проектирования систем управления. Этот инструмент, разработанный для использования с Simulink, автоматически настраивает системные параметры, основываясь

на определенных пользователем ограничениях на временные характеристики. Пакет использует свойство "click-and-drag" для изменения временных ограничений, позволяет легко настраивать переменные и указывать неопределенные параметры, обеспечивает интерактивную оптимизацию, реализует моделирование методом Монте-Карло, поддерживает проектирование SISO и MIMO систем управления, позволяет моделировать подавления помех, слежение и другие типы откликов, поддерживает проблемы повторяющегося параметра и задачи управления системами с запаздыванием и позволяет выбирать между удовлетворенными и недостижимыми ограничениями.

Robust Control Toolbox

Пакет Robust Control включает средства для проектирования и анализа многопараметрических устойчивых систем управления. Это системы с модельными ошибками, динамика которых известна не полностью или параметры которых могут изменяться в ходе моделирования. Мощные алгоритмы пакета позволяют выполнять сложные вычисления с учетом изменения множества параметров. Возможности пакета:

- Синтез LQG регуляторов на основе минимизации равномерной и интегральной нормы
- Многопараметрический частотный отклик
- Построение модели пространства состояний
- Преобразование моделей на основе сингулярных чисел
- Понижение порядка модели
- Спектральная факторизация

Пакет Robust Control базируется на функциях пакета Control System, одновременно предоставляя усовершенствованный набор алгоритмов для проектирования систем управления. Пакет обеспечивает переход между современной теорией управления и практическими приложениями. Он имеет множество функций, реализующих современные методы проектирования и анализа многопараметрических робастных регуляторов.

Проявления неопределенностей, нарушающих устойчивость систем, многообразны — шумы и возмущения в сигналах, неточность модели передаточной функции, немоделируемая нелинейная динамика. Пакет Robust Control позволяет оценить многопараметрическую границу устойчивости при различных неопределенностях. Среди используемых методов: алгоритм Перрона, анализ особенностей передаточных функций и др.

Пакет Robust Control обеспечивает различные методы проектирования обратных связей, среди которых: LQR, LQG, LQG/LTR и др. Необходимость понижения порядка модели возникает в нескольких случаях: понижение порядка объекта, понижение порядка регулятора, моделирование больших систем. Качественная процедура понижения порядка модели должна быть численно устойчива. Процедуры, включенные в пакет Robust Control, успешно справляются с этой задачей.

Model Predictive Control Toolbox

Пакет Model Predictive Control — это полный набор средств для реализации стратегии предиктивного управления. Эта стратегия была разработана для решения практических задач управления сложными многоканальными процессами при наличии ограничений на переменные состояния и управление. Методы предиктивного управления используются в химической промышленности и для управления другими непрерывными процессами. Пакет обеспечивает:

- Моделирование, идентификацию и диагностику систем
- Поддержку MISO, MIMO, переходных характеристик, моделей пространства состояний
- Системный анализ
- Конвертирование моделей в различные формы представления (пространство состояний, передаточные функции)
- Предоставление учебников и демонстрационных примеров

Предиктивный подход к задачам управления использует явную линейную динамическую модель объекта для прогнозирования влияния будущих изменений управляющих переменных на поведение объекта. Проблема оптимизации формулируется в виде задачи квадратичного программирования с ограничениями, решаемой на каждом такте моделирования заново. Пакет позволяет создавать и тестировать регуляторы как для простых, так и для сложных объектов.

Пакет включает более 50 специализированных функций для проектирования, анализа и моделирования динамических систем с использованием предиктивного управления.

Он поддерживает следующие типы систем: импульсные, непрерывные и дискретные по времени, пространство состояний. Обрабатываются различные виды возмущений. Кроме того, в модель могут быть явно включены ограничения на входные и выходные переменные.

Средства моделирования позволяют осуществлять слежение и стабилизацию. Средства анализа включают вычисление полюсов замкнутого контура, частотного отклика, другие характеристики системы управления. Для идентификации модели в пакете имеются функции взаимодействия с пакетом System Identification. Пакет также включает две функции Simulink, позволяющие тестировать нелинейные модели.

μ-Analysis and Synthesis

Пакет μ-Analysis and Synthesis содержит функции для проектирования устойчивых систем управления. Пакет использует оптимизацию в равномерной норме и сингулярный параметр μ. В версию 3 включен графический интерфейс для упрощения операций с блоками при проектировании оптимальных регуляторов. Свойства пакета:

- Проектирование оптимальных в равномерной и интегральной норме регуляторов
- Оценка действительного и комплексного сингулярного параметра μ
- D-K итерации для приближенного μ-синтеза
- Графический интерфейс для анализа отклика замкнутого контура
- Средства понижения порядка модели
- Непосредственное связывание отдельных блоков больших систем

Модель пространства состояний может быть создана и проанализирована на основе системных матриц. Пакет поддерживает работу с непрерывными и дискретными моделями. Пакет обладает полноценным графическим интерфейсом, включающим в себя возможность установки диапазона вводимых данных, специальное окно для редактирования свойств D-K итераций и графическое представление частотных характеристик. Имеет функции для матричного сложения, умножения, различных преобразований и других операций над матрицами. Обеспечивает возможность понижения порядка моделей.

Quantiative Feedback Theory Toolbox

Пакет содержит функции для создания робастных систем с обратной связью. QFT (теория обратных связей) — это инженерный метод, использующий частотное представление моделей для удовлетворения различных требований к качеству при наличии неопределенных характеристик у объекта. В основе метода лежит наблюдение, что обратная связь необходима в тех случаях, когда некоторые характеристики объекта неопределены и/или на его вход подаются неизвестные возмущения. Свойства пакета:

- Оценка частотных границ неопределенности, присущей обратной связи
- Графический интерфейс пользователя, позволяющий оптимизировать процесс нахождения требуемых параметров обратной связи
- Функции для определения влияния введения различных блоков в модель (мультиплексоров, сумматоров, петель обратной связи) при наличии неопределенностей
- Поддержка моделирования аналоговых и цифровых контуров обратной связи, каскадов и многоконтурных схем
- Разрешение неопределенности в параметрах объекта с использованием параметрических и непараметрических моделей или комбинации этих типов моделей

Теория обратных связей является естественным продолжением классического частотного подхода к проектированию. Ее основная цель — проектирование простых регуляторов небольшого порядка с минимальной шириной полосы пропускания, удовлетворяющих качественным характеристикам при наличии неопределенностей.

Пакет позволяет вычислять различные параметры обратных связей, фильтров, проводить тестирование регуляторов как в непрерывном, так и в дискретном пространстве. Пакет имеет удобный графический интерфейс, позволяющий создавать простые регуляторы, удовлетворяющие требованиям пользователя.

QFT позволяет проектировать регуляторы, удовлетворяющие различным требованиям, несмотря на изменения параметров модели. Измеряемые данные могут быть непосредственно использованы для проектирования регуляторов без необходимости идентификации сложного отклика системы. Множество линейных объектов, описывающих ряд условий работы, могут быть непосредственно использованы для проектирования QFT.

LMI Control Toolbox

Пакет LMI (Linear Matrix Inequality) Control обеспечивает интегрированную среду для постановки и решения задач линейного программирования. Предназначенный первоначально для проектирования систем управления, пакет позволяет решать любые задачи линейного программирования практически в любой сфере деятельности, где такие задачи возникают. Основные возможности пакета:

- Решение задач линейного программирования: задачи совместности ограничений, минимизация линейных целей при наличии линейных ограничений, минимизация собственных значений
- Исследование задач линейного программирования
- Графический редактор задач линейного программирования
- Задание ограничений в символьном виде
- Многокритериальное проектирование регуляторов
- Проверка устойчивости: квадратичная устойчивость линейных систем, устойчивость по Ляпунову, проверка критерия Попова для нелинейных систем

Пакет LMI Control содержит современные симплексные алгоритмы для решения задач линейного программирования. Использует структурное представление линейных ограничений, что повышает эффективность и минимизирует требования к памяти. Пакет имеет специализированные средства для анализа и проектирования систем управления на основе линейного программирования.

С помощью решателей задачи линейного программирования можно легко выполнять проверку устойчивости динамических систем и систем с нелинейными компонентами. Ранее этот вид анализа считался слишком трудоемким для реализации. Пакет позволяет даже такое комбинирование критериев, которое ранее считалось слишком сложным и разрешимым лишь с помощью эвристических подходов.

Пакет является мощным средством для решения выпуклых задач оптимизации, возникающих в таких областях, как управление, идентификация, фильтрация, структурное проектирование, теория графов, интерполяция и линейная алгебра. Пакет LMI Control включает два вида графического интерфейса пользователя: редактор задачи линейного программирования (LMI Editor) и интерфейс Magshape. LMI Editor позволяет задавать ограничения в символьном виде.

Пакеты идентификации систем

System Identification Toolbox

Пакет System Identification содержит средства для создания математических моделей динамических систем на основе наблюдаемых входных/выходных данных. Он имеет гибкий графический интерфейс, помогающий организовать данные и создавать модели. Методы идентификации, входящие в пакет, применимы для широкого класса задач, от проектирования систем управления и обработки сигналов до анализа временных рядов и вибрации. Основные свойства пакета:

- Простой и гибкий интерфейс
- Предварительная обработка данных, включая предварительную фильтрацию, удаление трендов и смещений
- Выбор диапазона данных для анализа
- Методы авторегрессии
- Анализ отклика во временной и частотной области
- Отображение нулей и полюсов передаточной функции системы
- Анализ невязок при тестировании модели

Графический интерфейс упрощает предварительную обработку данных, а также диалоговый процесс идентификации модели. Операции загрузки и сохранения данных, выбора диапазона, удаления смещений и трендов выполняются с минимальными усилиями и доступны из главного меню.

Представление данных и идентифицированных моделей организовано графически таким образом, что в процессе интерактивной идентификации пользователь легко может вернуться к предыдущему шагу работы. Для новичков существует возможность просматривать следующие шаги. Специалисту графические средства позволяют отыскать любую из ранее полученных моделей и оценить ее качество в сравнении с другими моделями.

Начав с измерения выхода и входа, можно создать параметрическую модель системы, описывающую ее поведение в динамике. Пакет поддерживает все традиционные структуры моделей, включая авторегрессию, структуру Бокса-Дженкинса и другие. Он поддерживает линейные модели пространства состояний, которые могут быть определены как в дискретном, так и в непрерывном пространствах. Эти модели могут включать произвольное число входов и выходов.

В пакет включены функции, которые можно использовать как тестовые данные для идентифицированных моделей. Идентификация линейных моделей широко используется при проектировании систем управления, когда требуется модель объекта. В задачах обработки сигналов модели могут быть использованы для адаптивной обработки сигналов. Методы идентификации успешно применяются для финансовых приложений.

Frequency Domain System Identification Toolbox

Пакет Frequency Domain System Identification предоставляет специализированные средства для идентификации линейных динамических систем по их временному или частотному отклику. Частотные методы направлены на идентификацию непрерывных систем, что является мощным дополнением к более традиционной дискретной методике. Методы пакета могут быть применены к таким задачам, как моделирование электрических, механических и акустических систем. Свойства пакета:

- Периодические возмущения, пик-фактор, оптимальный спектр, псевдослучайные и дискретные двоичные последовательности
- Расчет доверительных интервалов амплитуды/фазы и нулей/полюсов
- Идентификация непрерывных и дискретных систем с неизвестным запаздыванием
- Диагностика модели, включая моделирование и вычисление невязок
- Преобразование моделей в формат System Identification Toolbox и обратно

Используя частотный подход, можно добиться наилучшей модели в частотной области; избежать ошибок дискретизации; легко выделить постоянную составляющую сигнала, существенно улучшить соотношение сигнал/шум. Для получения возмущающих сигналов пакет предоставляет функции генерации двоичных последовательностей, минимизации величины пика и улучшения спектральных характеристик.

Пакетом обеспечивается идентификация непрерывных и дискретных линейных статических систем, автоматическая генерация входных сигналов, а также графическое изображение нулей/полюсов передаточной функции результирующей системы. Функции для тестирования модели включают вычисление невязок, передаточных функций и нулей/полюсов, прогонку модели с использованием тестовых данных.

Дополнительные средства пакета Simulink

Communications Toolbox

Пакет прикладных программ для построения и моделирования разнообразных коммуникационных устройств: цифровых линий связи, модемов, преобразователей сигналов и др. Содержит ряд интересных примеров моделирования коммуникационных средств, например, модема, работающего по протоколу v34, модулятора для обеспечения однополосной модуляции и др. Возможности пакета описаны в главе 10.

Data Signal Processing (DSP) Blockset

Пакет прикладных программ по проектированию устройств, использующих цифровые сигнальные процессоры. Это прежде всего высокоэффективные цифровые

фильтры с заданной или адаптируемой к параметрам сигналов амплитудно-частотной характеристикой (АЧХ).

Fixed-Point Blockset

Этот специальный пакет ориентирован на моделирование цифровых систем управления и цифровых фильтров в составе пакета Simulink. Специальный набор компонент позволяет быстро переключаться между вычислениями с фиксированной и плавающей точкой. Можно указывать 8-, 16- или 32-битную длину слова на блочной основе. Пакет обладает рядом полезных свойств: применением беззнаковой или двоичной арифметики; выбором пользователем положения двоичной точки; автоматической установкой положения двоичной точки; просмотром максимального и минимального диапазона сигнала модели; переключением между вычислениями с фиксированной и плавающей точкой; коррекцией контроля переполнения; наличием ключевых компонентов для операций с фиксированной точкой, включая сложение, вычитание и умножение, запаздывание, суммирование. Имеет логические операторы, одно- и двумерные справочные таблицы.

Обработка сигналов и изображений

Signal Processing Toolbox

Мощный пакет по анализу, моделированию и проектированию устройств обработки всевозможных сигналов, обеспечению их фильтрации и множества преобразований.

Пакет Signal Processing обеспечивает чрезвычайно обширные возможности по созданию программ обработки сигналов для современных научных и технических приложений. В пакете используется разнообразная техника фильтрации и новейшие алгоритмы спектрального анализа. Пакет содержит модули для разработки новых алгоритмов обработки сигналов, разработки линейных систем и анализа временных рядов. Пакет будет полезен, в частности, в таких областях, как обработка аудио- и видеоинформации, телекоммуникации, геофизика, задачи управления в реальном режиме времени, экономика, финансы и медицина. Основные свойства пакета:

- Моделирование сигналов и линейных систем
- Проектирование, анализ и реализация цифровых и аналоговых фильтров
- Быстрое преобразование Фурье, дискретное косинусное и другие преобразования
- Оценка спектров и статистическая обработка сигналов
- Параметрическая обработка временных рядов
- Генерация сигналов различной формы
- Оконное отображение

Пакет Signal Processing — идеальная оболочка для анализа и обработки сигналов. В нем используются проверенные практикой алгоритмы, выбранные по критериям максимальной эффективности и надежности. Пакет содержит широкий спектр алгоритмов для представления сигналов и линейных моделей. Этот набор позволяет пользователю достаточно гибко подходить к созданию сценария обработки сигналов. Пакет включает алгоритмы для преобразования модели из одного представления в другое.

Пакет Signal Processing включает полный набор методов для создания цифровых фильтров с разнообразными характеристиками. Он позволяет быстро разрабатывать фильтры высоких и низких частот, полосовые пропускающие и задерживающие фильтры, многополосные фильтры, в том числе фильтры Чебышева, Юла-Уолкера, эллиптические и другие фильтры.

Графический интерфейс позволяет проектировать фильтры, задавая требования к ним в режиме "point-and-click". В пакет включены следующие новые методы проектирования фильтров:

- Обобщенный метод Чебышева для создания фильтров с нелинейной фазовой характеристикой, комплексными коэффициентами или произвольным откликом. Алгоритм разработан Макленаном и Карамом в 1995 году.
- Метод наименьших квадратов с ограничениями позволяет пользователю явно контролировать максимальную ошибку (сглаживание).
- Метод расчета минимального порядка фильтра с окном Кайзера.
- Обобщенный метод Баттерворта для проектирования низкочастотных фильтров с максимально однородными полосами пропускания и затухания.

Основанный на оптимальном алгоритме быстрого преобразования Фурье, пакет Signal Processing обладает непревзойденными характеристиками для частотного анализа и спектральных оценок. Пакет включает функции для вычисления дискретного преобразования Фурье, дискретного косинусного преобразования, преобразования Гильберта и других преобразований, часто применяемых для анализа, кодирования и фильтрации. В пакете реализованы такие методы спектрального анализа, как метод Вельха, метод максимальной энтропии и другие.

Новый графический интерфейс позволяет просматривать и визуально оценивать характеристики сигналов, проектировать и применять фильтры, применять спектральный анализ, исследуя влияние различных методов и их параметров на получаемый результат. Графический интерфейс особенно полезен для визуализации временных рядов, спектров, временных и частотных характеристик, расположения нулей и полюсов передаточных функций системы.

Пакет Signal Processing является основой для решения многих других задач. Например, комбинируя его с пакетом Image Processing, можно обрабатывать и анализировать двумерные сигналы и изображения. В паре с пакетом System Identification пакет Signal Processing позволяет выполнять параметрическое моделирование во временной

области. В сочетании с пакетами Neural Network и Fuzzy Logic может быть создано множество средств для обработки данных или выделения классификационных характеристик. Средство генерации сигналов позволяет создавать импульсные сигналы различной формы.

Higher-Order Spectral Analysis Toolbox

Пакет Higher-Order Spectral Analysis содержит специальные алгоритмы для анализа сигналов с использованием моментов высшего порядка. Пакет предоставляет широкие возможности для анализа негауссовых сигналов, так как содержит алгоритмы, пожалуй, самых передовых методов анализа и обработки сигналов. Основные возможности пакета:

- Оценка спектров высокого порядка
- Традиционный или параметрический подход
- Восстановление амплитуды и фазы
- Адаптивное линейное прогнозирование
- Восстановление гармоник
- Оценка запаздывания
- Блочная обработка сигналов

Пакет Higher-Order Spectral Analysis позволяет анализировать сигналы, поврежденные негауссовым шумом, и процессы, происходящие в нелинейных системах. Спектры высокого порядка, определяемые в терминах моментов высокого порядка сигнала, содержат дополнительную информацию, которую невозможно получить, пользуясь только анализом автокорреляции сигнала или мощности спектра. Спектры высокого порядка позволяют:

- Подавить аддитивный цветной гауссов шум
- Идентифицировать неминимально-фазовые сигналы
- Выделить информацию, обусловленную негауссовым характером шума
- Обнаружить и проанализировать нелинейные свойства сигналов

Возможные приложения спектрального анализа высокого порядка включают акустику, биомедицину, эконометрию, сейсмологию, океанографию, физику плазмы, радары и локаторы. Основные функции пакета поддерживают спектры высокого порядка, взаимную спектральную оценку, линейные модели прогноза и оценку запаздывания.

Image Processing Toolbox

Пакет Image Processing предоставляет ученым и инженерам широкий спектр средств для цифровой обработки и анализа изображений. Будучи тесно связанным со средой разработки приложений MATLAB, пакет Image Processing Toolbox освобождает вас от выполнения длительных операций кодирования и отладки алгоритмов, по-

зволяя сосредоточить усилия на решении основной научной или практической задачи. Основные свойства пакета:

- Восстановление и выделение деталей изображений
- Работа с выделенным участком изображения
- Анализ изображения
- Линейная фильтрация
- Преобразование изображений
- Геометрические преобразования
- Увеличение контрастности важных деталей
- Бинарные преобразования
- Обработка изображений и статистика
- Цветовые преобразования
- Изменение палитры
- Преобразование типов изображений

Пакет Image Processing предоставляет широкие возможности для создания и анализа графических изображений, полностью совместимых со средой MATLAB. Этот пакет обеспечивает чрезвычайно гибкий интерфейс, позволяющий манипулировать изображениями, интерактивно разрабатывать графические картины, визуализировать наборы данных и аннотировать результаты для технических описаний, докладов и публикаций. Гибкость, соединение алгоритмов пакета с такой особенностью MATLAB, как матрично-векторное описание, делает пакет очень удачно приспособленным для решения практически любых задач по разработке и представлению графики.

В MATLAB входят специально разработанные процедуры, позволяющие повысить эффективность графической оболочки. Можно отметить, в частности, такие особенности:

- Интерактивная отладка при разработке графики
- Профилировщик для оптимизации времени выполнения алгоритма
- Средство построения интерактивного графического интерфейса пользователя (GUI Builder) для ускорения разработки GUI-шаблонов, позволяющий настраивать его под задачи пользователя

Этот пакет позволяет пользователю тратить значительно меньше времени на создание стандартных графических изображений и, таким образом, позволяет сконцентрировать усилия на важных пользователю деталях и особенностях этого изображения.

MATLAB и пакет Image Processing максимально приспособлены для развития, внедрения новых идей и методов пользователя. Для этого имеется набор сопрягаемых пакетов, направленных на решение всевозможных специфических задач и задач в нетрадиционной постановке. Язык MATLAB обладает открытой архитектурой, что позволяет пользователю выбирать и сопрягать модули на C, FORTRAN, MATLAB, которые были созданы ранее. Базы данных, алгоритмы и программные коды поддерживаются в различных операционных средах, включая PC, UNIX и Macintosh.

Пакет Image Processing в настоящее время интенсивно используется в более чем 4000 компаниях и университетах по всему миру. При этом наблюдается невероятно широкий на первый взгляд круг задач, которые пользователи решают с помощью данного пакета, например: космические исследования, военные разработки, астрономия, медицина, биология, роботостроение, материаловедение, генетика и прочее.

Wavelet Toolbox

Пакет Wavelet предоставляет пользователю полный набор программ для исследования многомерных нестационарных явлений. Методы пакета Wavelet расширяют возможности пользователя в тех областях, где обычно применяется техника Фурье-разложения. Пакет может быть полезен для таких приложений, как обработка речи и аудиосигналов, телекоммуникации, геофизика, финансы и медицина. Основные свойства пакета:

- Усовершенствованный графический пользовательский интерфейс и набор команд для анализа, синтеза, фильтрации сигналов и изображений
- Преобразование многомерных непрерывных сигналов
- Дискретное преобразование сигналов
- Декомпозиция и анализ сигналов и изображений
- Широкий выбор базисных функций, включая коррекцию граничных эффектов
- Пакетная обработка сигналов и изображений
- Анализ пакетов сигналов, основанный на энтропии
- Фильтрация с возможностью установления жестких и нежестких порогов
- Оптимальное сжатие сигналов

Пользуясь пакетом, можно анализировать такие особенности, которые упускают другие методы анализа сигналов, т.е. тренды, выбросы, разрывы в производных высоких порядков. Пакет позволяет сжимать и фильтровать сигналы без явных потерь, даже в тех случаях, когда нужно сохранить и высоко-, и низкочастотные компоненты сигнала. Имеются алгоритмы сжатия и фильтрации и для пакетной обработки сигналов. Программы сжатия выделяют минимальное число коэффициентов, представляющих исходную информацию наиболее точно, что очень важно для последующих стадий работы системы сжатия.

В пакет включены следующие базисные наборы: биортогональный, Хаара, "Мексиканская шляпа", Майера и другие. Вы также можете добавить в пакет свои собственные базисы. Обширное руководство пользователя объясняет принципы работы с методами пакета, сопровождая их многочисленными примерами и полноценным разделом ссылок.

Прочие пакеты прикладных программ

Financial Toolbox

Довольно актуальный для нашего периода рыночных реформ пакет прикладных программ по финансово-экономическим расчетам. Содержит множество функций по расчету сложных процентов, операций по банковским вкладам, вычисления прибыли и многое другое. К сожалению, из-за различий в финансово-экономических формулах (в общем-то не слишком принципиальных) его применение в наших условиях не всегда разумно.

Пакет Financial является основой для решения в MATLAB множества финансовых задач, от простых вычислений до полномасштабных распределенных приложений. Пакет Financial может быть использован для расчета процентных ставок и прибыли, анализа производных доходов и депозитов, оптимизации портфеля инвестиций. Основные возможности пакета:

- Обработка данных
- Дисперсионный анализ эффективности портфеля инвестиций
- Анализ временных рядов
- Расчет доходности ценных бумаг и оценка курсов
- Статистический анализ
- Анализ чувствительности рынка
- Калькуляция ежегодного дохода и расчет денежных потоков
- Методы начисления износа и амортизационных отчислений

Учитывая важность даты выполнения той или иной финансовой операции, в пакет Financial включены несколько функций для манипулирования датой и временем в различных форматах. Пакет Financial позволяет рассчитывать цены и доходы при инвестициях в облигации. Пользователь имеет возможность задавать нестандартные, в том числе нерегулярные и несовпадающие друг с другом графики дебитных и кредитных операций и окончательного расчета при погашении векселей. Экономические функции чувствительности могут быть вычислены с учетом разновременных сроков погашения.

Алгоритмы пакета Financial для расчета показателей движения денежных средств и других данных, отражаемых в финансовых счетах, позволяют вычислять, в частности, процентные ставки по займам и кредитам, коэффициенты рентабельности, кредитные поступления и итоговые начисления, оценивать и прогнозировать стоимость инвестиционного портфеля, вычислять показатели износа и тому подобное. Функции пакета Financial могут быть использованы с учетом положительного и отрицательного кэш-флоу (превышения наличных поступлений над платежами или наличных выплат над поступлениями, соответственно).

Пакет Financial содержит алгоритмы, которые позволяют анализировать портфель инвестиций, динамику и экономические коэффициенты чувствительности. В частно-

сти, при определении эффективности инвестиций Financial позволяет сформировать портфель, удовлетворяющий классической задаче Г. Марковица. Пользователь может комбинировать алгоритмы пакета для вычисления коэффициентов Шарпе и ставок дохода. Анализ динамики и экономических коэффициентов чувствительности позволяет пользователю определить позиции для стреддл-сделок, хеджирования и сделок с фиксированными ставками.

Пакет Financial обеспечивает также обширные возможности для представления и презентации данных и результатов в виде традиционных для экономической и финансовой сфер деятельности графиков и диаграмм. Денежные средства могут по желанию пользователя отображаться в десятичном, банковском и процентном форматах.

Mapping Toolbox

Пакет Mapping предоставляет графический и командный интерфейс для анализа географических данных, отображения карт и доступа к внешним источникам данных по географии. Кроме того, пакет пригоден для работы с множеством широко известных Атласов. Все эти средства в комбинации с MATLAB предоставляют пользователям все условия для продуктивной работы с научными географическими данными. Основные возможности пакета:

- Визуализация, обработка и анализ графических и научных данных
- Более 60 проекций карт (прямые и инверсные)
- Проектирование и отображение векторных, матричных и составных карт
- Графический интерфейс для построения и обработки карт и данных
- Глобальные и региональные Атласы данных и сопряжение с правительственными данными высокого разрешения
- Функции геостатистики и навигации
- Трехмерное представление карт со встроенными средствами подсветки и затенения
- Конвертеры для популярных форматов географических данных: DCW, TIGER, ETOPO5

Пакет Mapping включает более 60 наиболее широко известных проекций, в том числе: цилиндрическую, псевдоцилиндрическую, коническую, поликоническую и псевдоконическую, азимутальную и псевдоазимутальную. Возможны прямые и обратные проекции, а также собственные виды проекции пользователя.

В пакете Mapping картой называется любая переменная или множество переменных, отражающих или назначающих численное значение географической точке или области. Пакет позволяет работать с векторными, матричными и смешанными картами данных. Мощный графический интерфейс обеспечивает интерактивную работу с картами, например, возможность подвести указатель к объекту и, щелкнув, получить информацию. Графический интерфейс MAPTOOL — полная среда разработки приложений для работы с картами.

Наиболее широко известные атласы мира, Соединенных Штатов, астрономические атласы входят в пакет. Географическая структура данных упрощает извлечение и обработку данных из атласов и карт. Географическая структура данных и функции взаимодействия с внешними географическими данными форматов Digital Chart of the World (DCW), TIGER, TBASE и ETOPO5 объединены воедино, чтобы обеспечить мощный и гибкий инструмент для доступа к уже существующим и будущим географическим базам данных.

Тщательный анализ географических данных часто требует математических методов, работающих в сферической (не декартовой) системе координат. Пакет Mapping снабжен подмножеством геостатистических и навигационных функций для анализа географических данных. Функции навигации дают широкие возможности для выполнения задач перемещения, таких как позиционирование и планирование маршрутов.

СПИСОК ЛИТЕРАТУРЫ.

1. Дьяконов В.П. Справочник по расчетам на микрокалькуляторах. Издание 3-е дополненное и переработанное. М.: Наука, Физматлит. - 1989.- 464 с.
2. Дьяконов В. П. Компьютер в быту. Смоленск: Русич. - 1996.- 640 с.
3. Дьяконов В. П. Мой Pentium. М.: АСЕ.- 1998.- 536 с.
4. Дьяконов В.П. Справочник по алгоритмам и программам на языке Бейсик для персональных ЭВМ. М.: Наука. Физматлит. 1987. - 240 с.
5. Дьяконов В. П. Применение персональных ЭВМ и программирование на языке Бейсик. М.: Радио и связь. - 1989.- 288 с.
6. Dyakonov V. P., Yemelchenkov E.P., Munerman V.I., Samoilova T.A. The Revolutionary Guide to QBASIC. UK.: Wrox Press.- 1996.- 580 s.
7. Дьяконов В. П. Язык программирования ЛОГО. М.: Радио и связь. - 1991.- 144 с.
8. Дьяконов В. П. Форт-системы программирования персональных ЭВМ. М.: Наука. Физматлит.- 1992.- 352 с.
9. В. Дьяконов. Как выбрать математическую систему? Монитор-Аспект. - № 2.-, 1993.- 22 с.
10. В. Дьяконов, А. Пеньков. Современные математические системы. PC WEEK.- No 43 (67).- 1996.- с.42.
11. Дьяконов В. П. Компьютерные математические системы в образовании. Информационные технологии. - No 4.- 1997.- с. 40.
12. Дьяконов В.П . Справочник по применению системы Eureka. М.: Наука. Физматлит. 1993.- 96 с.
13. Дьяконов В.П. Mercury - отличная система для всех. Монитор-Аспект. - 1995.- No 5.- с.86.
14. Дьяконов В.П. Система MathCAD/Справочник. М.: Радио и связь .- 1993.- 128 с.
15. Очков В.Ф. MathCAD 7 Pro для студентов и инженеров. М.: Компьютер Press. - 1968.- 384 с.
16. MathCAD 6.0 PLUS. Финансовые, инженерные и научные расчеты в среде Windows 95. Пер. с англ. - М.: Филинъ .- 1996.- 712 с.
17. Дьяконов В. П. Справочник по MathCAD PLUS 6.0 PRO. М.: СК-ПРЕСС.- 1997.- 336 с.
18. Дьяконов В. П. Справочник по MathCAD 7.0 PRO. М.: СК-ПРЕСС.- 1998.- 352 с.
19. Дьяконов В. П., Абраменкова И. В. Mathcad 7.0 в математике, в физике и в Internet. М.: Нолидж.- 1998.- 352 с.
20. Дьяконов В. П., Абраменкова И. В. Техника визуализации учебных и научных задач с применением систем класса MathCAD. Информационные технологии. - No 11.- 1998.- 39 с.
21. Дьяконов В.П. Справочник по применению системы Derive.: М. Наука. Физматлит. - 1996.-144 с.
22. Дьяконов В. П. Справочник по системе символьной математики Derive. М.: СК-ПРЕСС.- 1998.- 256 с.

23. Дьяконов В. П. Справочник по математической системе Mathematica 2 и 3. М.: СК ПРЕСС.- 1998.
24. Дьяконов В.П. Математическая система Maple V R3/R4/R5. М.: Солон. - 1998.- 400 с.
25. Говорухин В.Н., Цибулин В.Г. Введение в Maple V. Математический пакет для всех. М.: Мир.- 1997.- 208 с.
26. Прохоров Г. В., Леденев М. А., Колбеев В. В. Пакет символьных вычислений Maple V. М.: Петит. - 1997.- 200 с.
27. Манзон Б. М. Maple V Power Edition. М.: Филинъ.- 1998.- 240 с.
28. K. M. Heal, L. M. Hansen, K. M. Rickard. Maple V Realise 5. Learning Guide. Springer.- 1998.- 284 s.
29. M. B. Monagan, K. O. Geddes, K. M. Heal, G. Labahn, S. M. Vorkoetter. Maple V Realise 5. Programming Guide. Springer.-1998.- 380 с.
30. Дьяконов В.П. Расширяемые системы для численных расчетов MatLAB. Монитор-Аспект. - 1993.- № 2.- с. 26.
31. Дьяконов В.П. Справочник по применению системы PC MatLAB. М.: Наука, Физматлит.- 1993.- 112 с.
32. Потемкин В. Г. MATLAB 5 для студентов. М.: Диалог-МИФИ. - 1998.- 314 с.
33. Потемкин В. Г. Система инженерных и научных расчетов MATLAB 5.x. В 2-х т. М.: ДИАЛОГ-МИФИ.- 1999.- 366 с. (т. 1).- 304 с. (т.2).
34. Математический энциклопедический словарь. Под редакцией Ю. В. Прохорова. М.: Советская энциклопедия. - 1988.- 847 с.
35. Бабенко К. И. Основы численного анализа. М.: Наука. Физматлит. - 1986.- 744 с.
36. Марчук Г.И. Методы вычислительной математики. М.: Наука. Физматлит. - 1989. - 608 с.
37. Бахвалов Н.С., Жидков Н.П., Кобельков Г. М. Численные методы. М.: Наука. Физматлит.- 1987.- 600 с.
38. Гантмахер Ф. Теория матриц. М.: Наука. Физматлит. - 1988.- 552 с.
39. Дж. Дэннис, Р. Шнабель. Численные методы безусловной оптимизации и решения нелинейных уравнений. Пер. с англ. под ред. Ю Г. Евтушенко. М.: Мир. - 1988.- 440 с.
40. Справочник по специальным функциям с формулами, графиками и математическими таблицами. Под ред. М. Абрамовица и И. Стиган. М.: Наука. Физматлит.- 1979.- 832 с.
41. Г. Корн, Т. Корн. Справочник по математике для научных работников и инженеров. М.: Наука. - 1973.-832 с.
42. Воднев В.Т., Наумович А.Ф., Наумович Н.Ф. Основные математические формулы. Минск: Вышэйшая школа. 1988.- 270 с.
43. Spiegel, Murray R. Mathematical Handbook of Formulas and Tables. New York: McGraw Hill Book Company, 1968.
44. Дж. Дэвенпорт, И. Сирэ, Э. Турнье. Компьютерная алгебра. Системы и алгоритмы алгебраических вычислений. М.: Мир. - 1991.- 352 с.
45. Иванов В. В. Методы вычислений на ЭВМ. Справочное пособие. К.: Наукова думка. - 1986.- 584 с.
46. Толстов Г. П. Ряды Фурье. М.: Наука. Физматлит.- 1980.- 384 с.

47. Королюк В.С., Портенко Н.И., Скороход А.В., Турбин А.Ф. Справочник по теории вероятности и математической статистике. М.: Наука. Физматлит. - 1985.- 640 с.
48. Львовский Е. Н. Статистические методы построения эмпирических формул. М.: Высшая школа. - 1988.- 239 с.
49. Топчеев Ю. И. Атлас для проектирования систем автоматического регулирования. М.: Машиностроение. - 1989.- 752 с.
50. Дьяконов В. П. Windows 95 на вашем компьютере. Смоленск: Русич. - 1997. - 528 с.
51. Дьяконов В. П. Мой Word 95/97. М.: АСТ.- 1998.- 336 с.
52. Дьяконов В.П. Популярная энциклопедия мультимедиа. М.: ABF.- 1996.- 416 с.
53. Дьяконов В.П. Internet. Настольная книга пользователя. М.: Солон-Р, - 1999.- 574 с.

Содержание

Дьяконов В. П.

Абраменкова И. В.

MATLAB 5.0/5.3. Система символьной математики.

Верстка: Матусовская Е.В.
Художник: Леонова Ю.В.
Ответственный за выпуск: Павшенко Н.В.

ЛР №064480 от 4 марта 1996 г. Подписано в печать 20.10.99.
Формат 70x100 1/16. Печать офсетная. Бумага газетная. Гарнитура Таймс.
Усл. печ. л. 40. Тираж 3 000 экз. Заказ № 1248

Издательство «Нолидж»
Москва, Шлюзовая наб., д. 10
Тел.: (095) 325-29-59, 235-05-07

Отпечатано с готовых диапозитивов
в ППП «Типография «Наука»,
121099, Москва, Шубинский пер., д. 6